Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 3643

Modellierung und Ausführung einer gekoppelten Festkörpersimulation mit Workflow-Choreographien

Kerstin Hintermayr

Studiengang:	Wirtschaftsinformatik
Prüfer:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer:	M.Sc. Wirt.-Inf. Andreas Weiß, Dipl.-Phys. David Molnar
Beginn am:	3. März 2014
Beendet am:	4. September 2014
CR-Nummer:	H.4.1,I.6.8

Kurzfassung

Im wissenschaftlichen Umfeld wird vermehrt die Workflow-Technologie eingesetzt, um Simulationen oder Berechnungen computergesteuert auszuführen. Die vorliegende Arbeit beschäftigt sich mit der Modellierung einer gekoppelten Festkörpersimulation als Workflow-Choreographie nach dem Top-Down-Ansatz. Auftauchende Herausforderungen werden identifiziert und mögliche Lösungsansätze beschrieben. Aufbauend auf dem Modellierungsergebnis werden die für die Ausführung der gekoppelten Festkörpersimulation benötigten Prozesse implementiert und vorgestellt. Die fertige Modellierung wird im Vergleich mit den Anforderungen beurteilt. Die Modellierung kann als Basis für zukünftige Arbeiten dienen und bietet Ansätze für aufbauende Untersuchungen. Dadurch wird eine Verfeinerung für zukünftige Workflowmodellierungen auf Basis einer Choreographie ermöglicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

Apache ODE	Apache Orchestration Director Engine
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
DD	Deployment Descriptor
ETL	Extract, Transfer and Load
HTTP	Hypertext Transfer Protocol
IMD	ITAP Molecular Dynamics
IMWF	Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre
ITAP	Institut für Theoretische und Angewandte Physik der Universität Stuttgart
JCL	Job Control Language
MAP	Multiagent Protocols
Mayflower	Model-as-you-go Workflow Developer
MC	Monte-Carlo
OPAL	Ostwald-Ripening of Precipitates on an Atomic Lattice
PBD	Participant Behavior Description
SIMPL	SimTech - Information Management, Process and Languages
SimTech	Simulationstechnologien
SOA	Service oriented Architecture
SOC	Service oriented Computing
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
VM	Virtuelle Maschine
WS-CDL	Web Service - Choreography Language
WSDL	Web Service Description Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XPath	XML Path Language

1. Einleitung

1.1. Motivation

Zur Automatisierung von (Geschäfts-) Prozessen hat sich in der Industrie die Workflow-Technologie durchgesetzt, um beispielsweise die Prozessdurchlaufzeit zu verringern oder die Qualität der Prozesse auf ein einheitlich hohes Niveau zu heben. Nach [Gad10, S.56] „erhofft man sich im Allgemeinen [mit dem Einsatz von Workflow-Management] auch Reduktionen der Kosten für die Durchführung der Geschäftsprozesse." Auch im wissenschaftlichen Bereich wird vermehrt auf die Workflow-Technologie gesetzt, um beispielsweise wissenschaftliche Experimente oder aufwändige Simulationen computergesteuert mit Scientific Workflows oder anderen Workflowarten auszuführen [RSM11].

Vor allem die Ausführung in einer verteilten und heterogenen Umgebung in paralleler und automatisierten Art und Weise bietet nach [GSK+11] Wissenschaftlern die Möglichkeit, sich intensiv auf die Lösung der wissenschaftlichen Problemstellung zu konzentrieren. Über die an der Universität Stuttgart entwickelte Software Bpel Designer werden Wissenschaftler voll umfänglich bei der Erstellung und Ausführung von Simulationen nach dem Trial-and-error-Prinzip unterstützt, sodass die modellierten Workflows beliebig oft ausgeführt werden können.

Bei der Beantwortung bestimmter wissenschaftlicher Fragestellungen ist es (allerdings) notwendig, dass verschiedene Simulationen hintereinander durch Wissenschaftler ausgeführt werden. Als Beispiel kann hier die gekoppelte Festkörpersimulation genannt werden, bei der zuerst die thermische Alterung von Stahl simuliert wird, um anschließend über eine weitere Simulation vorherzusagen, wie sich die thermische Alterung auf das Materialverhalten auswirkt.

In der Regel wird bei der Kopplung von Simulationen nach dem Bottom-Up-Ansatz vorgegangen. Dies bedeutet, dass erst nach dem Erstellen der einzelnen Simulationen geschaut wird, wie diese miteinander gekoppelt werden können. Der entgegengesetzte Ansatz, zuerst die Kopplung der Simulationen festzulegen und anschließend daraus ausführbare Workflows zu erstellen wird als Top-Down-Ansatz bezeichnet.

1.2. Ziel der Arbeit

Im Rahmen der Masterarbeit soll eine gekoppelte Festkörpersimulation als Choreographie von mehreren Simulationsworkflows konzeptionell erarbeitet und die entstandenen Workflows ausführbar gemacht werden. Hierbei wird nach dem sogenannten Top-Down-Ansatz zuerst die Choreographie zwischen zwei Simulationsworkflows modelliert, bevor mithilfe eines Choreographieeditors (Chor Designer) des Instituts für Architektur von Anwendungssystemen der Universität Stuttgart aus dem Choreographie-Modell das Code-Grundgerüst für die zwei Simulationsworkflows generiert wird. Die Arbeit soll dabei folgende Forschungsfrage beantworten:

> **Wie kann eine gekoppelte Festkörpersimulation mit Workflow-Choreographien modelliert und daraus ausführbare Prozesse erstellt werden?**

Daraus leiten sich folgende Forschungsteilfragen ab:

1. Wie ist der aktuelle Forschungsstand im Bereich Scientific- und Simulationsworkflows?
2. Wie kann die benötigte Kopplung der verschiedenen Simulationen mittels einer Workflow-Choreographie erfolgen?
3. Wie kann ein choreographieübergreifender Simulationskontext verwendet werden?

1.3. Gliederung

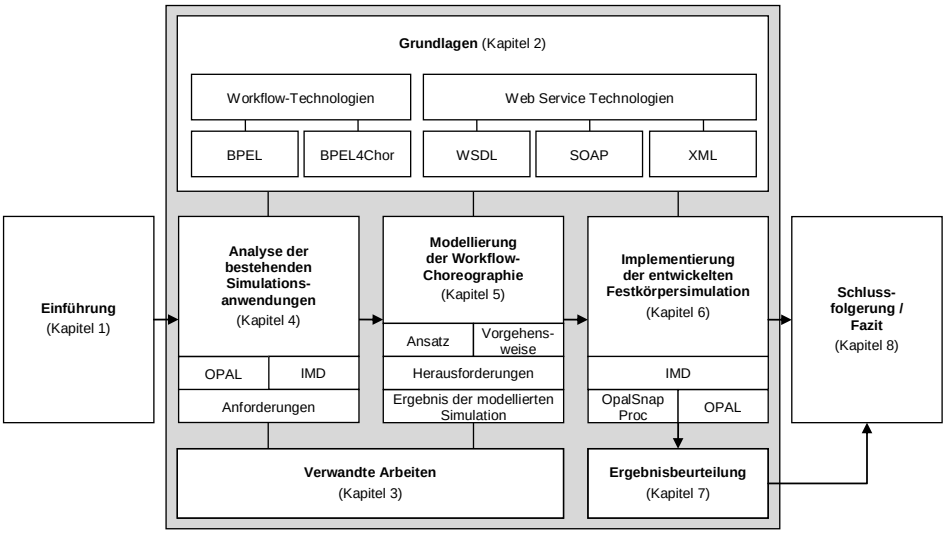


Abbildung 1.1.: Bezugsrahmen der Masterarbeit

Der in Abbildung 1.1 dargestellte Bezugsrahmen gibt einen Überblick über die einzelnen Kapitel der vorliegenden Arbeit und stellt deren Zusammenhang zueinander dar:

Kapitel 2 – Grundlagen: Zu Beginn werden in die grundlegenden Begriffe und Konzepte erörtert, welche für das Verständnis der Masterarbeit relevant sind oder der Themenabgrenzung dienen.

Kapitel 3 – Verwandte Arbeiten: Dieses Kapitel stellt weitere konzeptionelle und praktische Arbeiten vor, welche entweder als Grundlage für diese Masterarbeit relevant sind oder einen ähnlichen theoretischen Bezug haben.

Kapitel 4 – Analyse der bestehenden Simulationsanwendungen: In diesem Kapitel werden die bestehenden Simulationsanwendungen analysiert und in Anforderungen für die Modellierung der automatischen Festkörpersimulation mit Workflow-Choreographien genannt.

Kapitel 5 – Modellierung der Workflow-Choreographie: Anhand der definierten Anforderungen wird in Kapitel 5 die gekoppelte Festkörpersimulation mit einer Workflow-Choreographie modelliert, auftauchende Herausforderungen festgehalten, mögliche Lösungen beschrieben und das Ergebnis der Modellierung vorgestellt.

Kapitel 6 – Implementierung der entwickelten Festkörpersimulation: Dieses Kapitel stellt anschließend die Implementierung der einzelnen Workflows vor.

Kapitel 7 – Ergebnisbeurteilung: Aufbauend auf die Implementierung wird in Kapitel 7 die entstandene gekoppelte Festkörpersimulation hinsichtlich der Forschungsfragen beurteilt.

Kapitel 8 – Fazit und Ausblick: Die Ergebnisse der Arbeit werden in Kapitel 8 zusammengefasst und weitere Anknüpfungspunkte vorgestellt.

2. Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe und Konzepte erörtert, welche für das Verständnis der Masterarbeit relevant sind oder der Themenabgrenzung dienen.

2.1. Workflow-Technologien

Die geordnete Abfolge einer Reihe von Aktivitäten zur Erreichung eines bestimmten (Geschäfts-)Zieles ist nach [Wes12, S.17] als (Geschäfts-)Prozess definiert. Wird diese Abfolge zudem noch ganz oder teilweise automatisiert in einer Computerumgebung ausgeführt, wird nach [Wes12, S.50] von einem Workflow gesprochen. Der Hauptunterschied zwischen einem Prozess und einem Workflow liegt nach [LR00, S.7] folglich darin, dass Prozess, im Gegensatz zu einem Workflow, nicht im Ganzen oder in Teilen in einer Computerumgebung ausgeführt werden. Dennoch lassen sich sowohl Geschäftsprozesse als auch Workflows nach [LR00, S.8f.] durch die drei Dimensionen *was*, *wer* und *womit* beschreiben.

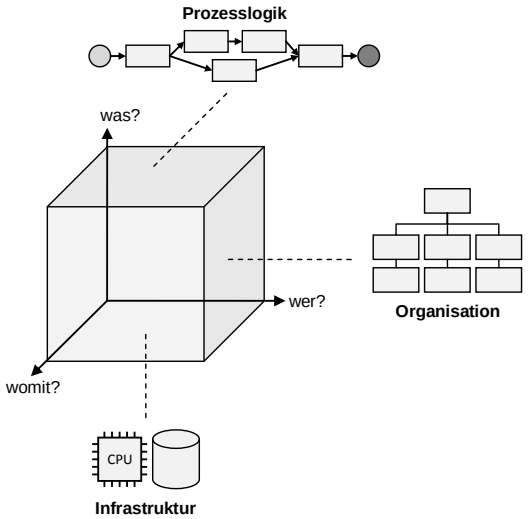


Abbildung 2.1.: Workflow-Dimensionen[1]

Wie aus Abbildung 2.1 hervorgeht, beschreibt die y-Achse als erste Dimension die Prozesslogik. Dies bedeutet, dass durch die Prozesslogik festgehalten ist, in welcher Reihenfolge und unter welchen Bedingungen Aktivitäten ausgeführt werden. Die, durch die x-Achse dargestellte, zweite Dimension umfasst die Organisation und dadurch wer bzw. welche Rolle die Aktivität ausführt. Mit der z-Achse als dritte Dimension wird die benötigte IT-Infrastruktur beschrieben, d. h. welche Systeme oder Software für die Ausführung der Aktivität benötigt werden.

Für die Darstellung von Prozessen bzw. Workflows kommen in der Praxis verschiedene Möglichkeiten in Frage. Zum einen können mithilfe der Job Control Language (JCL) die einzelnen Schritte schriftlich (chronologisch) festgehalten werden. Zum anderen gibt es grafische Modellierungssprachen wie beispielsweise Business Process Model and Notation (BPMN) zur bildlichen Darstellung einer Abfolge von einzelnen Aktivitäten. Abbildung 2.2 zeigt eine beispielhafte Darstellung eines Workflows mit BPMN am Beispiel der Molekulardynamik-Simulation.

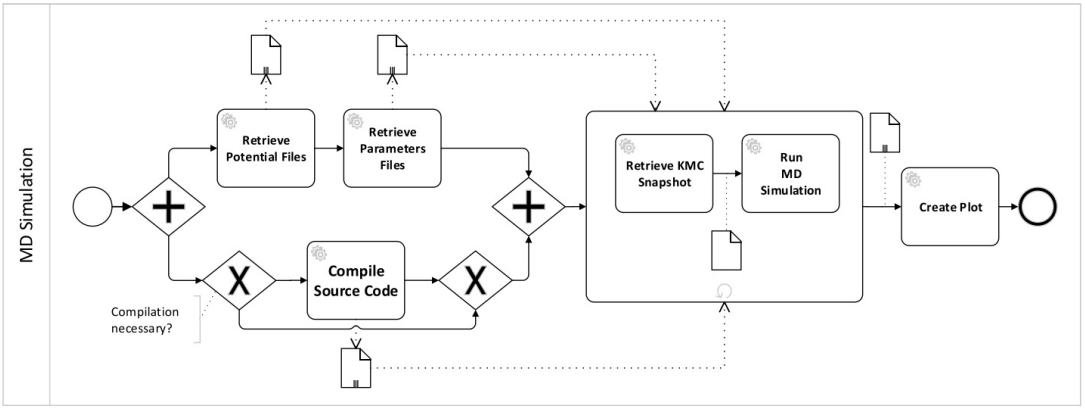


Abbildung 2.2.: Beispielhafte Darstellung eines Workflow mit BPMN[2]

2.1.1. Klassifizierung von Workflows

In der Literatur gibt es viele verschiedene Möglichkeiten zur Klassifizierung von Workflows. Nach [RSM11] können Workflows beispielsweise anhand der Datenintensität und der durch sie zu lösende Probleme klassifiziert werden. Abbildung 2.3 veranschaulicht diesen Sachverhalt grafisch. Dabei zeigen die dargestellten Rechtecke hierbei die verschiedenen Arten und Gruppen von Workflows auf,

[1]Mit Änderungen entnommen aus: [LR00, S.8]
[2]Entnommen aus: [WKMS14, S.3]

während durch die Ovalen konkrete Beispiele für die einzelnen Workflowarten angeben werden. Das folgende Teilkapitel basiert, soweit nicht anders angegeben, auf [RSM11] und [Pie12].

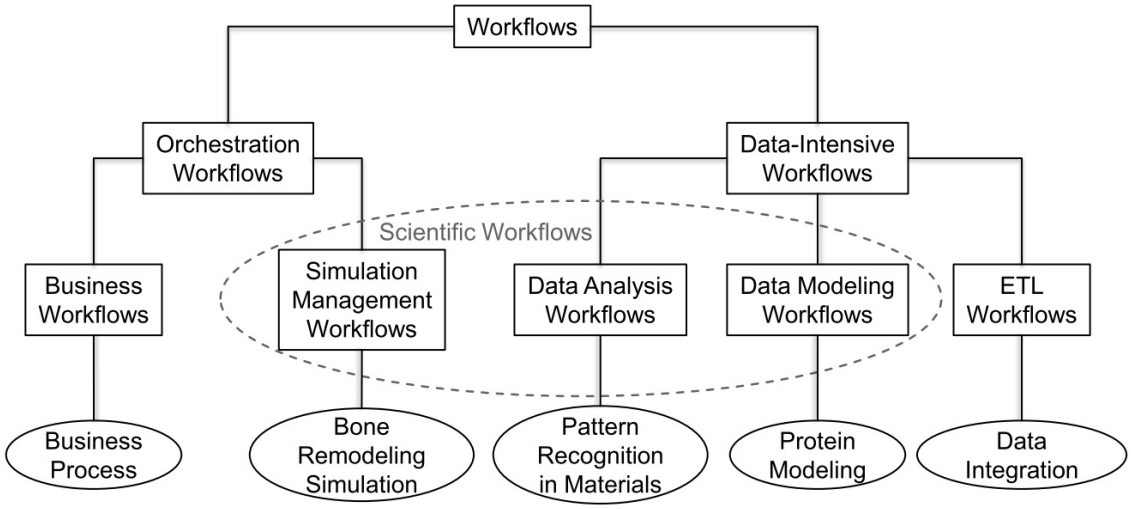


Abbildung 2.3.: Klassifizierung von Workflows nach [RSM11][3]

Geschäftsworkflows (Business Workflows) realisieren die Ausführung von wiederkehrenden Arbeitsschritten innerhalb eines Unternehmens, um eine Automatisierung von Geschäftsprozessen zu ermöglichen. Simulation Management Workflows hingegen koordinieren das Zusammenspiel von Simulationsanwendungen und Ressourcen, welche beispielsweise Berechnungen und Datenmanagement-Aufgaben für Simulationsprozesse ausführen. Gemeinsam gehören diese zwei Workflowarten zur Gruppe der Orchestration Workflows. Orchestration Workflows zeichnen sich durch die Verknüpfung von unterschiedlichen und ggf. auch heterogenen Anwendungen sowie die Verarbeitung von kleinen Datenmengen aus.

Die Gruppe der datenintensiven Workflows zeichnen sich vor allem durch die Verarbeitung großer und ggf. verteilt vorliegender Datenmengen aus. Beispielsweise können mit den ETL-Workflows[4] Daten für andere Anwendungen extrahiert und bereitgestellt werden.

Der aus dem wissenschaftlichen Umfeld geprägte Begriff Scientific Workflow soll verdeutlichen, dass es bei dieser Gruppe um die Ausführung von wissenschaftlichen Experimenten, Simulationen oder Berechnungen geht, welche in der Regel nicht durch standardisierte Software unterstützt werden. Neben den Simulation Management Workflows werden auch Data Analysis Workflows, welche

[3] Entnommen aus: [RSM11, S.355]

[4] ETL steht in diesem Kontext für Extract, Transfer and Load

beispielsweise bereits generierte Daten visualisieren, und Data Modeling Workflows, welche z. B. bestimmte Muster in Modellen suchen, zu dieser Gruppe gezählt. Durch die Nutzung von Scientific Workflows bieten sich Wissenschaftlern nach [GSK+11] beispielsweise Vorteile wie die Unterstützung des Wissensaustausch, eine Community-basierte Analyse der Ergebnisse, die Verarbeitung von riesigen Datenmengen oder eine Ausführung in verteilten und heterogenen Umgebungen in paralleler und automatisierter Art und Weise. Dadurch können sich Wissenschaftler intensiver auf die Lösung des wissenschaftlichen Problems konzentrieren.

Scientific Workflows unterscheiden sich von Business Workflows zusammenfassend darin, dass Scientific Workflows viel explorativer und nach der Trial-and-error Prinzip entwickelt werden. Nach [SK13] liegt dies vor allem daran, dass Wissenschaftler zwar das zu erreichende Ziel kennen, nicht aber den hierfür notwendigen Weg. Dies führt dazu, dass Wissenschaftler Workflows immer wieder mit veränderten Parametern ausführen, bis sie ihr Ziel erreicht haben. Dazu kommt, dass sämtliche Arbeitsschritte im Lifecycle des Workflows durch den Wissenschaftler selbst durchgeführt werden (vgl. Abbildung 2.4).

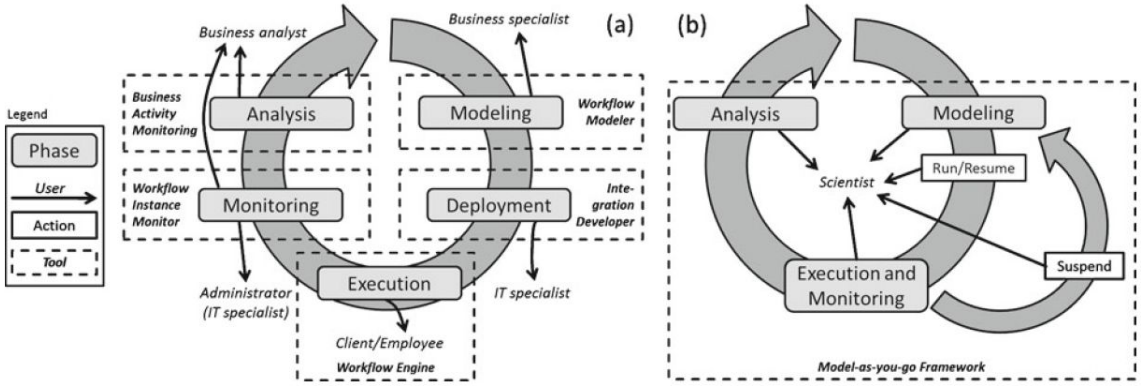


Abbildung 2.4.: Lebenszyklus von a) Business Workflows und b) Scientific Workflows[4]

Aus diesen Gründen beschreibt [SHK12] den in Abbildung 2.4 b) dargestellten Bedarf nach einer einheitlichen, integrierten und einfach zu benutzenden Software zur Unterstützung aller Phasen im Workflow-Lifecycle, ohne die Software wechseln zu müssen. Als ein beispielhaftes Konzept stellt [SHK12] das Mayflower (**M**odel-**a**s-**y**ou-go Work**flow** Develop**er**)-Framework vor.

[4] Entnommen aus: [SK13, S.557]

2.1.2. Orchestrierung und Choreographie von Workflows

Mithilfe der Orchestrierung werden nach [Wes12] verschiedene Web Services[5] zu einem Workflow verknüpft, um einen bestimmten Prozess abzubilden. Vor allem im Geschäftsbereich ist es üblich, dass übergeordnete Prozesse durch das Zusammenspiel von mehreren Prozessen aus verschiedenen Unternehmen, Unternehmensteilen oder Akteuren entstehen, um ein übergeordnetes Ziel zu erreichen. In diesem Fall wird von einer Choreographie gesprochen, da Aktivitäten (realisiert durch Web Services) verschiedener Prozessorchestrierungen ohne einen zentralen Koordinator zusammenarbeiten.

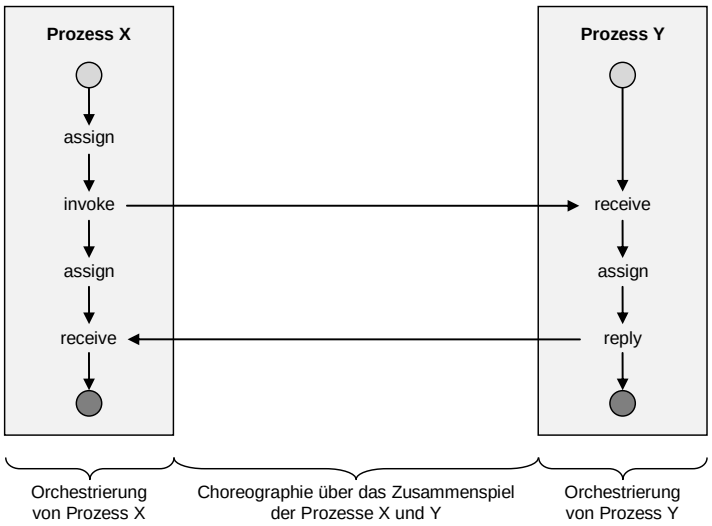


Abbildung 2.5.: Orchestrierung vs. Choreographie von Workflows[6]

Wie aus Abbildung 2.5 hervorgeht, wird bei der Choreographie vor allem aus einer globalen Sicht festgelegt bzw. modelliert, wie die einzelnen Workflows miteinander kommunizieren, nicht jedoch wie sie funktionieren. Die Funktionsweise/Arbeitsweise oder auch Abfolge eines Workflows geht wiederum aus der Orchestrierung verschiedener Web Services zu diesem Workflow hervor.

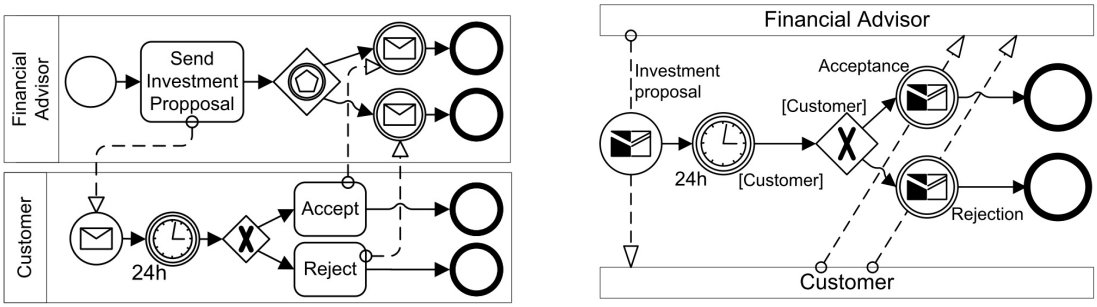
Nach [BWR09] gibt es zwei Ansätze für die Modellierung von Choreographien:

1. **Interconnection Model:** Bei diesem Ansatz werden die Teilnehmer/Akteure einer Choreographie als abstrakte Prozesse beschrieben bzw. modelliert (vgl. hierzu Abbildung 2.6 a)). Das Augenmerk der Modellierung liegt jedoch lediglich auf den Kommunikationsaktivitäten, welche für den Daten-/Nachrichtenaustausch zwischen den Akteuren benötigt werden. Da alle anderen

[5] Die Definition und Erklärung zu dem Begriff Web Service erfolgt in Kapitel 2.2
[6] Mit Änderungen entnommen aus: [Mel10, S.244f.]

Aktivitäten der Akteure ausgeblendet und somit nicht modelliert werden, sind diese abstrakten Prozesse erst nach einer manuellen Erweiterung ausführbar.

2. **Interaction Model:** Bei diesem Ansatz hingegen steht, wie aus Abbildung 2.6 b) hervorgeht, der Nachrichtenfluss über den gesamten Prozess im Vordergrund und nicht die einzelnen Aktivitäten der verschiedenen Akteure. Somit werden hierbei nur die Interaktionen der einzelnen Akteure modelliert.



(a) Interconnection Model **(b)** Interaction Model

Abbildung 2.6.: Modellierungsansätze: a) Interconnection Model, b) Interaction Model[7]

2.1.3. WS-BPEL

Nach [Wes12] und [AAA+07] steht WS-BPEL (kurz BPEL) für Business Process Execution Language und dient dazu, aus IT-Sicht das Verhalten von in Computerumgebungen ausgeführten Geschäftsprozessen (Workflows) mithilfe von BPEL-Aktivitäten zu modellieren. Zudem ist es möglich einzelne fachlichen Arbeitsschritte (z.B. das Ermitteln einer Kundenadresse) durch den Aufruf eines dafür entwickelten Web Services ausführen zu lassen.

Aktivitäten in BPEL-Prozessen können einer der zwei folgenden Gruppen zugeordnet werden: Basisaktivtäten und strukturierende Aktivitäten. Die Basisaktivitäten umfassen alle Aktivitäten, welche zur eigentlichen Arbeitsverrichtung erforderlich sind. Im Gegensatz zu den Basisaktivitäten legen die strukturierenden Aktivitäten den Prozessablauf fest. Tabelle 2.1 zeigt eine Übersicht der BPEL 2.0 Aktivitäten.

[7] Entnommen aus: [KLW10, S.3f.]

Gruppe	Aktivität	Beschreibung
Basisaktivitäten	invoke	Ermöglicht es, einen Web Service synchron oder asynchron aufzurufen.
	receive	Wird benötigt, um den Web Service nach außen für andere aufrufbar zu machen.
	reply	Wird benötigt, um eine Antwort auf den Web Service-Call an den Aufrufer zu übermitteln.
	assign	Ermöglicht die Veränderung von Variableninhalten.
	wait	Ermöglicht es, während der Prozessausführen einen bestimmten Zeitpunkt oder eine Zeitspanne abzuwarten.
	throw	Explizites Aufzeigen eines Fehlers, damit dieser speziell behandelt werden kann.
	empty	Keine Aktion ausführen, beispielsweise wenn bei einem bestimmten Fehler nichts passieren soll.
	catch	Ermöglicht das Auffangen von Fehlermeldungen und deren speziellen Fehlerbehebung.
	scope	Ermöglicht das Festlegen von Bereichen, in denen Fehler eingedämmt und behandelt werden können.
	compensate	Wird im Rahmen der Fehlerbehebung verwendet, um alle bereits fertiggestellten Aktivitäten wieder rückgängig zu machen.
	exit	Wird benötigt, um beispielsweise im Fehlerfall den Prozess zu beenden.
strukturierende Aktivitäten	sequence	Die darin definierten Aktivitäten werden der Reihe nach ausgeführt.
	flow	Erlaubt das parallele oder beliebige Verarbeiten von Aktivitäten.
	while	Ausführen von Aktivitäten solange die Schleifenbedingung erfüllt ist.
	repeatUntil	Ausführen von Aktivitäten bis eine Bedingung erfüllt ist.
	forEach	Kann eine parallele Ausführung von Aktivitäten, definiert in der forEach-Aktivität, ermöglichen. Andernfalls verhält sich die Aktivität wie eine Schleife. Die Häufigkeit der parallelen Ausführung bzw. die Anzahl der Schleifendurchläufe werden, im Gegensatz zur flow-Aktivität nicht während der Entwicklung, sondern dynamisch zur Laufzeit festgelegt.
	if	Ermöglicht eine Fallunterscheidung in der Ausführung von Aktivitäten.
	pick	Ermöglicht das Blockieren des Prozesses bis ein definiertes Ereignis von außen Eintritt (bestimmter Zeitpunkt, Zeitspanne oder Nachricht).

Tabelle 2.1.: BPEL 2.0 Aktivitäten

Wie aus Listing 2.1 hervorgeht, besteht das BPEL-Dokument zur Beschreibung eines Workflows immer aus dem **Wurzelelement <process>**. Innerhalb dieses Elements muss mindestens eine Aktivität, welche einen Arbeitsschritt im Workflow referenziert, enthalten sein. Um neue Funktionen wie beispielsweise die BPEL4People-Spezifikation verwenden zu können, kann das BPEL-Dokument mit Hilfe des **<extensions>-Elements** erweitert werden. Mittels des **<import>-Elements** ist es zudem möglich, WSDL-[8] oder XML-Dateien[9] zu referenzieren.

Listing 2.1 Aufbau eines BPEL 2.0-Dokuments

```
<process name="ProcessName">
      <extensions>...</extensions>
      <import/>
      <partnerLinks>...</partnerLinks>
      <messageExchanges>...</messageExchanges>
      <variables>...</variables>
      <correlationSets>...</correlationSets>
      <faultHandlers>...</faultHandlers>
      <compensationHandlers>...</compensationHandlers>
      <eventHandlers>...</eventHandlers>
      <!-- mindestens eine Aktivitaet -->
</process>
```

BPEL wird genutzt, um Workflows aus technischer Sicht zu modellieren. Für einzelne Arbeitsschritte innerhalb eines Workflows können Web Services eingebunden werden, welche die Funktionalitäten des Arbeitsschrittes zur Verfügung stellen. Die Verbindung zwischen den Web Services und dem BPEL-Prozess wird über das **<partnerLinks>-Element** im BPEL-Prozess dargestellt, indem für jeden Web Services der aus dem BPEL-Prozess aufgerufen wird ein <partnerLink>-Element eingefügt wird. Über dieses Element ist es zudem auch möglich anderen Nutzern den BPEL-Prozess als eigenständige Funktionalität zur Verfügung zu stellen. Vergleiche hierzu Listing 2.2.

Listing 2.2 Aufbau des <partnerLinks>-Elements im BPEL-Dokument

```
<partnerLinks>
      <partnerLink name="bestellen" partnerLinkType="GrossHandelLink"
                  myRole="Verkaeufer" partnerRole="Kunde" />
</partnerLinks>
```

Durch das **<messageExchanges>-Element** aus Listing 2.3 ist es im BPEL-Dokument möglich, den Nachrichtenaustausch zwischen dem Nutzer des BPEL-Prozesses und dem Prozess eindeutig zu

[8] Web Service Description Language

[9] Extensible Markup Language

definieren. Dies ist vor allem dann notwendig, wenn es während der Ausführung dazu kommen kann, dass mehrere eingehende (receive-Aktivitäten) und ausgehende (reply-Aktivitäten) Nachrichten-Paare auf demselben PartnerLink und derselben Operation simultan ausgeführt werden. Hierfür werden im <messageExchanges>-Element verschiedene <messageExchange>-Elemente angelegt. Bei der Verwendung der entsprechenden <receive>- und <reply>-Aktivitäten wird dann auf das definierte <messageExchange>-Element referenziert (vgl. hierzu Listing 2.3).

Listing 2.3 Verwendung eines <messageExchange>-Elements im BPEL-Dokument

```
<process>
      ...
      <messageExchanges>
            <messageExchange name="NCName" />
      </messageExchanges>
      ...
      <receive partnerLink="NCName" portType="QName" operation="NCName"
          variable="BPELVariableName" createInstance="yes|no" messageExchange="NCName"
          standard-attributes>...</receive>
      ...
</process>
```

Das **<variables>-Element** ermöglicht die Festlegung von Variablen, welche bei der späteren Ausführung des Prozesses den Prozesszustand widerspiegeln. Eine definierte BPEL-Variable enthält demnach die vom Prozess benutzten Daten. Analog zu der Programmiersprache Java müssen auch in BPEL Variablen in einem Gültigkeitsbereich eindeutig sein. Der default-Gültigkeitsbereich des BPEL-Prozesses ist mit dem Wurzelelement <process> verbunden. Über das Element <scope> können jedoch weitere Gültigkeitsbereiche definiert werden. Es ist lediglich zu beachten, dass lokale Variablen eines Gültigkeitsbereiches gleichnamige globale Variablen überschreiben. Wie aus Listing 2.4 hervorgeht, können Variablen als Datentyp sowohl einen WSDL messagetype, einen XML Schema Simple Type oder ein XML Schema Element haben.

Listing 2.4 Aufbau des <variables>-Elements im BPEL-Dokument

```
<variables>
      <variable name="vorname" element="xsd:string" />
      <variable name="adresse" type= "tns:adresse" />
      <variable name="daten" messageType="tns:bestellAnfrage" />
</variables>
```

Durch das **<correlationSets>-Element** ist es möglich Korrelationsmengen zu definieren, welche dann mit den Kommunikationsaktivitäten receive, reply, invoke und onMessage verwendet werden können, um beispielsweise eine sichere Benutzeridentifikation zu ermöglichen. Dadurch wird eine

eindeutige Zuordnung der Nachrichten zu einer bestimmten Prozessinstanz hergestellt. Dies ist vor allem dann notwendig, wenn ausgehend von einem BPEL-Prozess mehrere Instanzen gestartet werden.

Die bei der Ausführung des BPEL-Prozesses möglicherweise auftretenden Fehler (exceptions) können mit dem **<faultHandlers>-Element** behandelt werden. Dadurch ist es möglich, bei bestimmten Fehlern, eine Kompensierungsaktion auszuführen oder den Prozess ganz zu terminieren (vgl. hierzu Listing 2.5).

Listing 2.5 Aufbau des <faultHandlers>-Elements im BPEL-Dokument

```
<faultHandlers>
            <catch faultName="AbbruchDurchUser">
                  <compensate>
                        ... <invoke name="Rollback" .../>
                  </compensate>
            </catch>
            <catchAll>  <exit> </catchAll>
</faultHandlers>
```

Soll innerhalb eines BPEL-Prozesses ein asynchroner Web Service Aufruf erfolgen, ist es notwendig, dass der BPEL-Prozess nach dem Web Service-Aufruf nicht blockiert sondern mit der eigentlichen Prozessausführung fortfahren kann. Um dies zu ermöglichen, wird ein EventHandler benötigt. Durch das **<eventHandlers>-Element** wird daher definiert, wie ein BPEL-Prozess mit Events umzugehen hat, die während der laufenden Prozessausführung auftreten. Im EventHandler (vgl. Listing 2.6) können zum einen Nachrichtenevents <onMessage> und zum anderen Alarmevents <onAlarm> definiert werden.

Listing 2.6 Aufbau des <eventHandlers>-Elements im BPEL-Dokument

```
<eventHandlers>
    <onMessage partnerLink="Client" portType="buy:BuyBookPT" operation="CancelBuy"
        variable="BookPurchase">
                <terminate/>
    </onMessage>

    <onAlarm for="'PT15M'">
      <throw faultName="buy:Timeout" faultVariable="Fault" />
    </onAlarm>
</eventHandlers>
```

Nachrichtenevents werden durch ankommende Nachrichten von Web Service Operationen und Alarmevents durch eine bestimmte Zeitspanne oder das erreichen eines bestimmten Zeitpunktes ausgelöst.

Die <onAlarm> und <onMessage> Aktivitäten können auch während der eigentlichen Prozessausführung innerhalb eines <pick>-Elementes genutzt werden, um beispielsweise die Ergebnisse eines Web Service-Aufrufes direkt abzuwarten und zu verarbeiten. Allerdings wird dabei der Prozess blockiert. Durch die Definition von beliebig vielen <onMessage> und/oder <onAlarm> Aktivitäten in einem EventHandler ist es möglich, den Prozess auszuführen und gleichzeitig auf die im EventHandler festgelegten Aktivitäten zu reagieren, falls diese während der Ausführung eintreten sollten. Dadurch ist es möglich beispielsweise Timeouts (onAlarm-Aktivität) zentral an einer Stelle (EventHandler) zu definieren.

2.1.4. BPEL4Chor

Mit BPEL4Chor wird die WS-BPEL-Spezifikation um Elemente erweitert, welche die Choreographie von Web Services ermöglichen. Choreographiesprachen wie z. B. WS-CDL[10] oder BPEL4Chor unterstützen den Top-Down-Ansatz für die Entwicklung und Implementierung von Choreographien. Mit der BPEL4Chor-Spezifikation wird der BPEL-Standard um die in Abbildung 2.7 dargestellten Elemente erweitert. Anhand der in Abbildung 2.8 dargestellten Choreographie werden die drei Bestandteile der BPEL4Chor-Spezifikation näher erläutert.

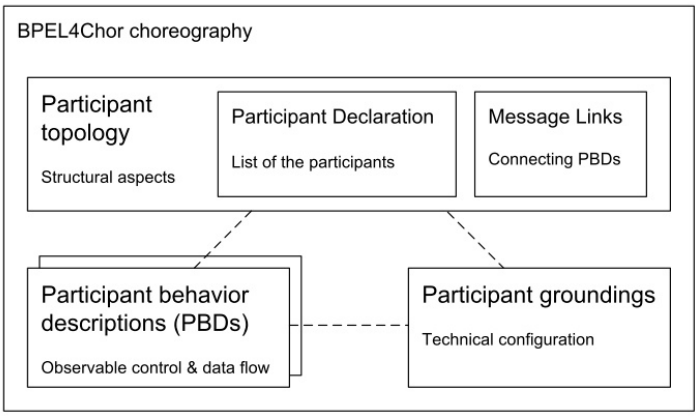


Abbildung 2.7.: BPEL4Chor Artefakte[8]

[10] Web Services Choreography Language
[11] Entnommen aus: [DKLW07, S.3]

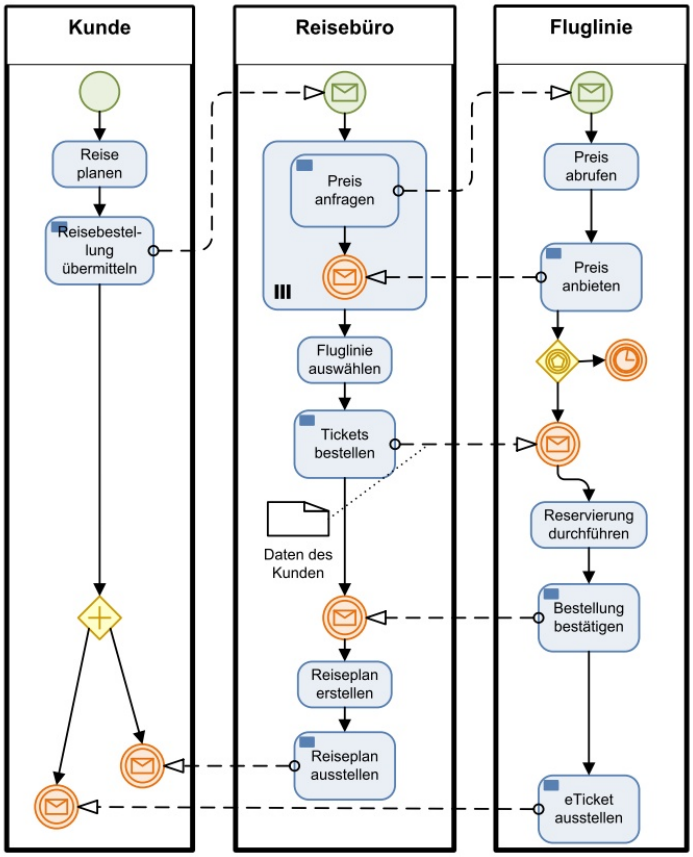


Abbildung 2.8.: Beispiel einer mit BPMN dargestellten Choreographie[9]

Die **Participant Topology** beschreibt die Kommunikation zwischen den Teilnehmern der Choreographie. Zudem wird genau spezifiziert, von welchem Typ die einzelnen Teilnehmer der Choreographie sind (participantTypes), welche Teilnehmer generell an der Choreographie beteiligt sind und deren Teilnehmerreferenzen (participants bzw. participant references). Darüber hinaus werden die Nachrichtenlinks (messageLinks) festgelegt, über welche die Teilnehmer miteinander kommunizieren. Listing 2.7 zeigt für die Beispielchoreographie aus Abbildung 2.7 eine beispielhafte Participant Topology.

[12] Entnommen aus: [Deb13, S.25]

Listing 2.7 Beispiel einer Participant Topology[10]

```
<topology name="buchungstopology" targetNamespace="urn:buchung"
  xmlns:reisebuero="urn:buchung:reisebuero">
  <participantTypes>
    <participantType name="Reisebuero"
      participantBehaviorDescription="reisebuero:reisebuero" />
    <participantType name="Kunde" ... />
    <participantType name="Fluglinie" ... />
  </participantTypes>
  <participants>
    <participant name="kunde" type="Kunde" selects="reisebuero" />
    <participant name="reisebuero" type="Reisebureo" selects="fluglinien" />
    <participantSet name="fluglinien" type="Fluglinie" forEach="reisebuero:Preis_Anfragen_FE">
      <participant name="aktuelleFluglinie" forEach="reisebuero:Preis_Anfragen_FE" />
      <participant name="gewaehlteFluglinie" />
    </participantSet>
  </participants>
  <messageLinks>
    <messageLink name="reiseBestellungUebermittelnLink"
      sender="kunde" sendActivity="Reisebestellung_Uebermitteln"
      receiver="reisebuero" receiveActivity="Erhalt_Reisebestellung"
      messageName="reiseBestellung" />
      <!-- ... -->
    <messageLink name="ticketsBestellenLink"
      sender="reisebuero" sendActivity="TicketsBestellen"
      receiver="gewaehlteFluglinie" receiveActivity="Erhalt_Bestellung"
      messageName="ticketBestellung" participantRefs="kunde" />
    <messageLink name="eTicketAusstellenLink"
      sender="gewaehlteFluglinie" sendActivity="eTicketAusstellen"
      receiver="kunde" receiveActivity="Erhalt_eTicket" messageName="eTicket" />
  </messageLinks>
</topology>
```

Da sowohl in den Participant Behavior Descriptions (PBDs) als auch im Participant Topology keinerlei technische Aspekte mehr enthalten sind, werden diese technischen Informationen in einem **Participant Grounding** spezifiziert. Wie aus Listing 2.8 hervorgeht, wird zu jedem MessageLinks aus der Participant Topology die entsprechenden technischen Informationen wie PortType und Operation (enthalten in der WSDL des aufzurufenden Web Services) im Participant Grounding angegeben.

[13] Entnommen aus: [Deb13, S.25]
[14] Entnommen aus: [Deb13, S.25]

Listing 2.8 Beispiel eines Participant Grounding[11]

```
<grounding topology="top:buchungstopology"
  xmlns:top="urn:buchung" xmlns:...>
  <messageLinks>
    <messageLink name="reiseBestellungUebermittelnLink"
      portType="ag1:reiseBuero_pt" operation="getTripRequest" />
    <messageLink name="ticketsBestellenLink"
      portType="lhx:web_pt" operation="getOrder" />
           <!-- ... -->
  </messageLinks>
  <participantRefs>
    <participantRef name="kunde" WSDLproperty="msgs:kundeProp" />
  </participantRefs>
</grounding>
```

In der **Participant Behavior Description (PBD)** wird der Kontrollfluss der einzelnen Aktivitäten definiert, die zu einem bestimmten Teilnehmer gehören und dessen Verhalten bestimmen. Dies bedeutet, dass im PBD eines Teilnehmers nur die Aktivitäten enthalten sind, welche für die Kommunikation mit den anderen Choreographieteilnehmern notwendig sind. Listing 2.9 stellt eine beispielhafte PBD dar.

Listing 2.9 Participant beavior description-Beispiel[12]

```
<process name="reisebuero" targetNamespace="urn:buchung:reisebuero"
  abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12">
  <sequence>
    <receive wsu:id="Erhalt_Reisebestellung" createInstance="yes" />
      <forEach wsu:id="Preis_Anfragen_FE" parallel="yes">
        <scope>
          <sequence>
            <invoke wsu:id="Preis_Anfragen" />
            <receive wsu:id="Preis_Erhalten" />
          </sequence>
        </scope>
      </forEach>
      <opaqueActivity name="Fluglinie_Auswaehlen" />
      <invoke wsu:id="Tickets_Bestellen" />
      <receive wsu:id="Erhalt_Bestellungsbestaetigung" />
      <opaqueActivity name="Reiseplan_Erstellen" />
      <invoke wsu:id="Reiseplan_Ausstellen" />
  </sequence>
</process>
```

2.2. Web Service Technologien

Wie bereits [Mel10] festgehalten hat, existiert derzeit keine verbindliche Definition des Begriffs Web Service. So bezeichnet beispielsweise das World Wide Web Consortium (W3C) je nach Spezifikation Web Services als

> "A software application identified by a URI[16], whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols." [ABFG04]

oder als

> "A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP[17] with an XML serialization in conjunction with other Web-related standards." [HB04]

Für diese Masterarbeit wird die Definition des W3C herangezogen, welche festhält, dass Web Services Funktionen als Services über ein Netzwerk (Intranet/Internet/Extranet) zur Verfügung stellen.

2.2.1. Web Services zur Realisierung einer serviceorientierten Architektur

Service-Oriented Computing (SOC) ist nach [PG03] ein Computerparadigma bei dem Services (bzw. Web Services) als grundlegende Elemente für die Entwicklung von Anwendungen genutzt werden. Die konkrete Umsetzung von Service oriented Computing zur Bereitstellung einer flexiblen und anpassbaren IT-Architektur wird Service-Oriented Architecture (SOA) genannt. [Sta07, S.12] definiert SOA wie folgt:

> „Eine serviceorientierte Architektur (SOA) ist eine Unternehmensarchitektur, deren zentrales Konstruktionsprinzip Services (Dienste) sind. Dienste sind klar gegeneinander abgegrenzt und aus betriebswirtschaftlicher Sicht sinnvolle Funktionen. Sie werden entweder von einer Unternehmenseinheit oder durch externe Partner erbracht."

[15] Entnommen aus: [Deb13, S.25]

[16] Uniform Resource Identifier

[17] Hypertext Transfer Protocol

Die nachfolgende Abbildung 2.9 gibt einen kurzen Einblick in den Themenbereich SOA und das Zusammenspiel der einzelnen Akteure. Eine ausführliche Erklärung zu den in der Abbildung aufgeführten Standards (z. B. UDDI[18] oder WSDL) erfolgt in den nächsten Unterkapiteln. Wie aus der Abbildung hervorgeht, veröffentlichen Dienstanbieter (Service Provider) die von ihnen angebotenen Funktionalitäten (realisiert als Web Services) in einem Dienstverzeichnis (beispielsweise UDDI) eines Dienstvermittlers (Service Broker). Dabei beschreibt der Dienstanbieter mithilfe eines WSDL-Dokuments die funktionalen Anforderungen seines Web Services (Schnittstelle). Die nicht-funktionalen Anforderungen hält er hingegen in einer Web Service Policy fest. Dadurch ist es möglich, dass ein Dienstnutzer (Service Requestor) bestimmte Funktionalitäten bei einem Dienstvermittler suchen kann. Anhand der nicht-funktionalen (Web Service Policy) und funktionalen Anforderungen (WSDL-Dokument) eines Services kann der Dienstnutzer entscheiden, ob ein Service seinen Suchkriterien entspricht. Mithilfe der Informationen des Dienstvermittlers über einen bestimmten Service ist es dem Dienstnutzer möglich den Services bei dem eigentlichen Dienstanbieter des Services zu finden und diesen beispielsweise in einen seiner bestehenden Workflows einzubinden. Für die Interaktion zwischen dem Dienstnutzer und dem Dienstanbieter bzw. Dienstvermittler wird in der Regel das Nachrichtenprotokoll SOAP verwendet.

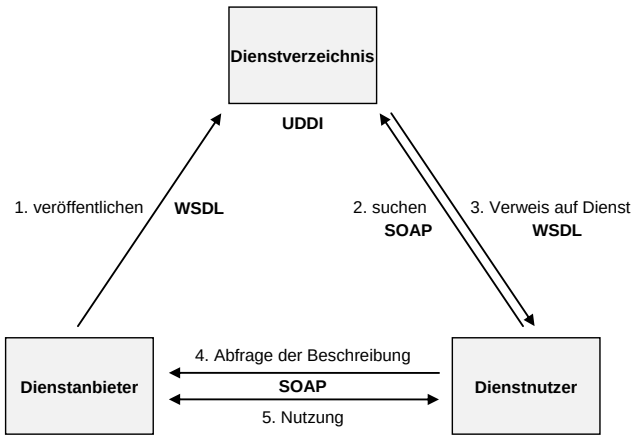


Abbildung 2.9.: SOA-Dreieck[13]

Zur Realisierung einer SOA werden, wie bereits im vorherigem Abschnitt beschrieben, einige weitere Standards verwendet (vgl. hierzu Abbildung 2.10). Funktionalitäten werden in einer SOA durch Web Services bereitgestellt. Diese Web Services werden zum einen durch WSDL-Dokumente beschrieben (welche auch vom Dienstanbieter in einem Dienstverzeichnis wie UDDI veröffentlicht sein können)

[18] Universal Description Discovery and Integration
[19] Entnommen aus: [HH10, S.64]

und zum anderen ist es möglich, über das Nachrichtenprotokoll SOAP mit dem Web Service zu kommunizieren. Die über SOAP an den Web Service verschickten Nachrichten bestehen aus einem XML-Dokument, welches durch ein XML-Schema beschrieben ist, und in der WSDL des Web Services referenziert wird.

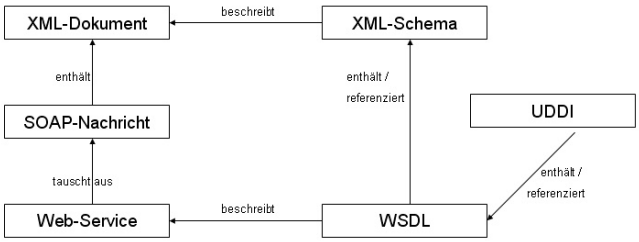


Abbildung 2.10.: Zusammenhang Web Services, WSDL, SOAP, UDDI[14]

2.2.2. Nachrichtenprotokoll SOAP

SOAP stand früher als Abkürzung für Simple Object Access Protocol, wird aber seit 2003 als Eigenname verwendet, da es nach [Bur07, Kil10] weder einfach ist noch dem reinen Zugriff auf Objekte dient. Unter SOAP wird demnach heute ein Protokoll zur Nachrichten- bzw. Datenübertragung zwischen Systemen oder Anwendungen verstanden. Es kann somit nach [Kil10, S.57] im Rahmen einer SOA-Realisierung als Kommunikationsprotokoll für die Web Services genutzt werden. Mit der SOAP-Spezifikation wird unter anderem die wesentliche Struktur einer auf XML beruhenden SOAP-Nachricht für den Austausch von Nachrichten beschrieben.

Wie aus Listing 2.10 hervorgeht, ist die SOAP-Envelope das XML-Wurzelelement, welches alle weiteren Daten enthält. Der SOAP-Header ist optional und enthält zusätzliche Informationen wie beispielsweise die Authentifizierungsdaten für einen Web Service. Im SOAP-Body ist die zu übermittelnde Nachricht enthalten und kann beispielsweise den Methodenname der aufzurufenden Methode oder auch Übergabeparameter und Rückgabewerte enthalten.

[20] Mit Änderungen entnommen aus: [Sta07, S.33]

Listing 2.10 Beispielhafte SOAP-Nachricht

```
<?xml version="1.0" encoding="utf-8"?>
      <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
                      <soap:Header>
                            <ns1:Auth>
                                  <ns1:SystemId Param="PARAM">DATA</ns1:SystemId>
                                  <ns1:UserName>USERNAME</ns1:UserName>
                                  <ns1:Password>PASSWORD</ns1:Password>
                            </ns1:Auth>
                      <soap:Header>
                      <soap:Body/>
      </soap:Envelope>
```

2.2.3. Web Service Description Language

Die Web Service Description Language (WSDL) ist ein XML-basiertes Format zur Beschreibung von Web Service Schnittstellen. Wie aus Abbildung 2.11 hervorgeht, kann mit einer WSDL-Datei zum einen der abstrakte Teil einer Service-Schnittstelle und zum anderen auch der konkrete Teil der Service-Schnittstelle beschrieben werden (vgl. hierzu [Mel10, S.116]).

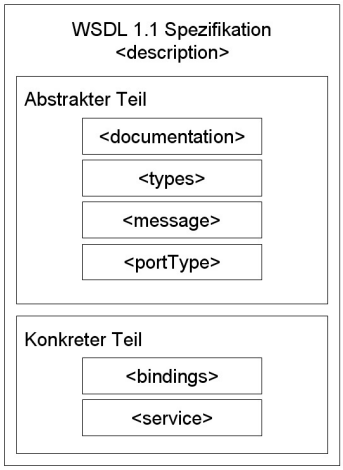


Abbildung 2.11.: WSDL 2.0 Spezifikation[15]

[21] Mit Änderungen entnommen aus: [Alo04, S.167]

Ein WSDL 1.1 Dokument besteht nach [CCMW10] aus den Kindelementen des in Abbildung 2.11 dargestellten <description>-Elements. Im **<documentation>-Element** hat der Serviceanbieter die Möglichkeit, textuelle Informationen über den von ihm angebotenen Service festzuhalten. Im **<types>-Element** hingegen werden Datentypen definiert, die in den vom Service gesendeten oder empfangenen Nachrichten enthalten sind. Durch die zentrale Definition der Datentypen können diese auch in mehreren Operationen wiederverwendet werden. Listing 2.11 zeigt eine beispielhafte Darstellung des <types>-Elements.

Listing 2.11 Definition der Datentypen im types-Element

```
<types>
      <xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.example.com/note/abfrage.xsd"
          xmlns="http://www.example.com/note/abfrage.wsdl">
            <xs:element name="matrikelnummer" type="xs:string" />
            <xs:element name="note" type="xs:double" />
            <xs:element name="eingabeUngueltig" type="xs:string" />
      </xs:schema>
</types>
```

Das **<portType>-Element** beschreibt die abstrakte Funktionalität des Services. Dies bedeutet, dass an dieser Stelle im WSDL-Dokument alle Operationen des Web Services in Form des <operation>-Elements festgehalten sind. Die <operation>-Elemente umfassen zudem alle XML-Nachrichten (messages), welche im Rahmen des Operationsaufrufes zwischen dem Web Service und dem Client ausgetauscht werden (vgl. hierzu Listing 2.12).

Listing 2.12 Definition einer Operation im portType-Element

```
<portType name="notenabfragePT">
            <operation name="notenabfrage">
                  <input message="in" />
                  <output message="out" />
                  <fault message="fault" />
      </operation>
</portType>
```

Über das **<message>-Element** werden die zu übertragenden Daten abstrakt definiert. Dabei kann das <messaage>-Element aus mehreren logischen Teilen (<part>-Elements) bestehen, welche sich jeweils auf ein zuvor definiertes Element im <types>-Element beziehen. Listing 2.13 zeigt die message-Elemente für die zuvor definierte Operation notenabfrage.

Listing 2.13 Definition der message-Elemente

```
<message name="in"> <part name="parameters" element="tns:matrikelnummer" /> </message>
<message name="out"> <part name="parameters" element="tns:note"/> </message>
<message name="fault"> <part name="parameters" element="tns:eingabeUngueltig" /> </message>
```

Im **<binding>-Element** wird (vgl. Listing 2.14) konkret festgehalten, welches Protokoll für den Nachrichtenaustausch bzw. welche Detailinformationen zur Kodierung und Transport der Nachrichten zu verwenden sind. Es referenziert daher auch auf ein vorab definiertes abstraktes <portType>-Element und ist sozusagen dessen konkrete Ausprägung.

Listing 2.14 Definition des binding-Elemente

```
<binding name="notenabfrageBinding" type="tns:notenabfragePT">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
      <operation name="notenabfrage">
            <soap:operation soapAction="http://www.example/notenabfrage" />
            <input>     <soap:body use="Literal" /> </input>
      </operation>
</binding>
```

Mit dem **<service>-Element** erfolgt die konkrete Angabe, wo der Web Service physikalisch zu erreichen ist und mithilfe des location-Atributes ist festgelegt, über welche URLs[22] der Service aufgerufen werden kann. Da ein Service über mehrere Ports verfügen kann, ist es möglich den Service auf verschiedene Weisen in diesem Fall beispielsweise mit SOAP 1.1 oder SOAP 1.2 aufzurufen (vgl. hierzu Listing 2.15).

Listing 2.15 Definition der Zugriffspunkte im service-Element

```
<service name="abfrageService">
      <port binding="tns:notenabfrageBinding" name="AbfragePortSoap11">
                  <soap:address
                       location="http://www.example.com/note/abfrageStudiengangBachelor"/>
       </port>
       <port binding="tns:notenabfrageBindingSoap12" name="AbfragePortSoap12">
                        <soap12:address
                             location="http://www.example.com/note/abfrageStudiengangBachelor"/>
       </port>
</service>
```

[22] Uniform Resource Locator

2.2.4. Universal Description Discovery and Integration

Universal Description Discovery and Integration (UDDI) war angedacht als ein standardisierter Verzeichnisdienst für Web Services, der sich bisher in der Praxis nicht durchsetzen konnte. UDDI stellte dabei lediglich eine technische und nicht eine semantische Suchfunktionalität zur Verfügung.

Die UDDI-Datenbank ist nach einem Page-Konzept (analog zu Telefonbüchern) aufgebaut. Die WhitePage enthält unternehmensspezifische Informationen. Mit den YellowPages hingegen werden die Dienste (Services) aller Anbieter nach Branchen bzw. Kategorien sortiert, sodass auch Services unbekannter Anbieter gefunden werden können. Die GreenPages enthalten schließlich die Beschreibung über den angebotenen Service. Ergänzt werden die GreenPages durch die Service Type Registration, welche die Informationen der GreenPages in einer für Maschinen lesbaren Art enthält und auf die entsprechende GreenPage verweist. Das dahinterstehende technische UDDI-Datenmodell kann aus Abbildung 2.12 entnommen werden.

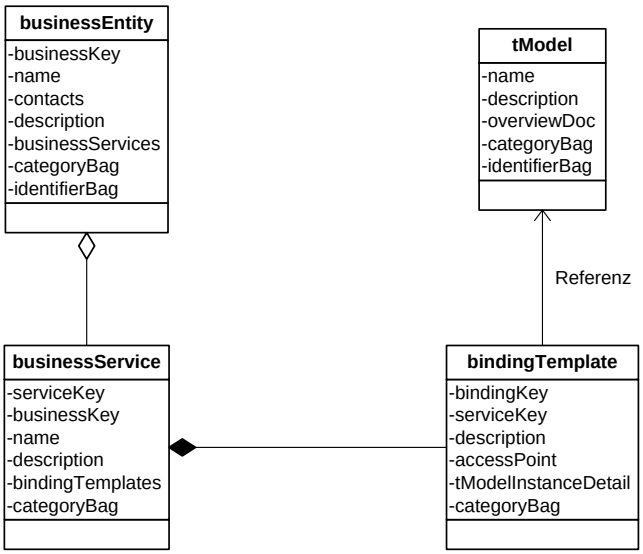


Abbildung 2.12.: UDDI-Datenmodell[16]

[23] Entnommen aus: [Mel10, S.149]

2.2.5. Entwicklungsansätze

Die Art und Weise, wie Web Services entwickelt werden, kann nach [HH10] auf zwei verschiedene Entwicklungsansätze aufgeteilt werden. Beim **Contract-First-Ansatz** wird vor der eigentlichen Web Service Entwicklung die finale Schnittstelle des zu entwickelnden Web Services in Form einer WSDL festgelegt. Wie aus Abbildung 2.13 hervorgeht, werden ausgehend von der WSDL des zukünftigen Web Services zum einen die Dienst-Stubs (Skeleton) für den Service Provider generiert und zum anderen die Dienstnutzer-Stubs (Stubs) durch den Nutzer des zukünftigen Web Service. Nach [Ham07, S.131] bezeichnen Sub bzw. Skeleton die „Gesamtheit aller Dateien, die aus der Schnittstellenbeschreibung generiert werden. Der Stub liegt auf Seiten des Clients, das Skeleton auf Seitens des Servers. Kernaufgabe ist das Marshalling und Unmarshalling [...] von Daten sowie die Vermittlung der Aufrufe." Im Anschluss an die Stubgenerierung erfolgt durch den Serviceprovider die Implementierung der Funktionalität des Web Services bzw. die Einbindung des Web Service durch den Web Service Nutzer. Bei diesem Ansatz ist somit eine nahezu zeitgleiche Entwicklung durch den eigentlichen Service Provider und die Nutzer des Web Services möglich.

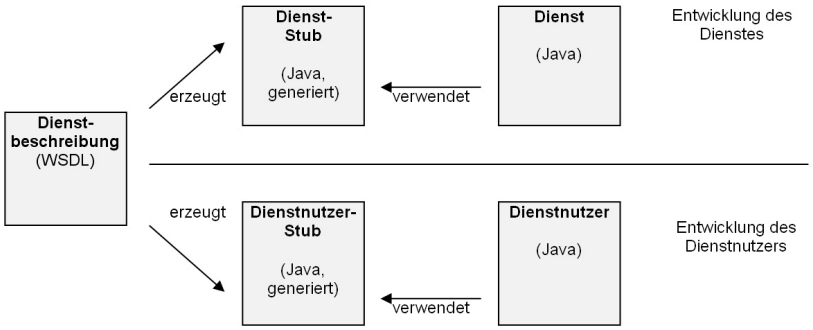


Abbildung 2.13.: Contract-First-Ansatz[17]

Im Gegensatz zum Contract-First-Ansatz wird beim **Code-First-Ansatz** zuerst der Web Service entwickelt. Anschließend wird ausgehend von den Web Service Klassen das dazugehörige WSDL-Dokument und die entsprechenden Datentypen generiert. Diese können dann wiederum von den Nutzer des Web Services genutzt werden, um die entsprechenden Stubs zu generieren und den Web Service einzubinden.

[24] Entnommen aus: [HH10, S.154]
[25] Entnommen aus: [HH10, S.164]

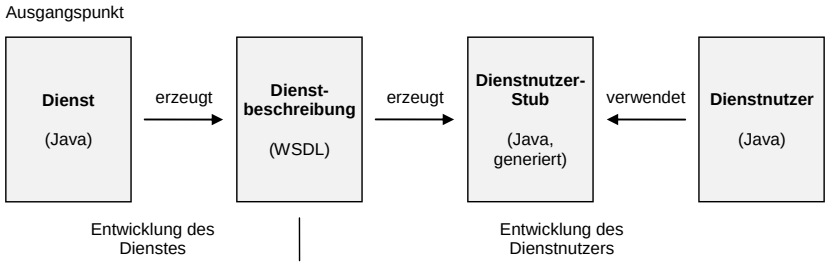


Abbildung 2.14.: Code-First-Ansatz[18]

2.2.6. XML Technologien

XML steht für eXtensible Markup Language und wird nach [SLFS06] als Dokumentenverarbeitungsstandard vom World Wide Web Consortium (W3C) empfohlen. Die nachfolgende Beschreibung beruht auf den XML-Spezifikationen des W3C [XML]. Da ein **XML-Dokument** nur die Struktur des Dokuments definiert und nicht wie mit dem Inhalt umzugehen ist, ermöglicht XML automatisch eine Trennung von Daten und Inhalten. Wie aus Listing 2.16 hervorgeht, kann ein XML-Dokument zu Beginn eine optionale XML-Deklaration, unter anderem mit einer Angabe zur verwendeten XML Version, enthalten. Anschließend folgt genau ein Wurzelelement, welches über verschiedene weitere Elemente verfügen kann. Hierbei ist zu beachten, dass die Elementnamen selbst definiert werden können, jedes Element aber immer über ein Anfangs-Tag (<Elementname>) und ein End-Tag(</Elementname>) verfügen muss und sich diese nicht überlappen dürfen.

Listing 2.16 Beispielhaftes XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Abschlussarbeiten>
            <Masterarbeit id="3643">
                        <author> Kerstin Hintermayr </author>
                        <title> Modellierung Simulation mit Workflow-Choreografien </title>
             </Masterarbeit>
</Abschlussarbeiten>
```

Durch die Verwendung von **XML-Namensräumen** wird die Eindeutigkeit von XML-Elementen sichergestellt. Mit dem Attribut xmlns wird in Listing 2.17 der Namensraum deklariert. Durch das hier exemplarische ausgewählte Präfix *ustutt* ist es möglich mehrere Elemente mit gleichem Namen aber aus unterschiedlichen Namensräumen in einem Dokument zu verwenden.

Durch die Verwendung von **XML-Schema** in einem XML-Dokument ist es möglich, die Struktur und Elemente der XML-Dokumente zu definieren und gegen eine konkrete XML-Schema Definition

Listing 2.17 Beispielhaftes XML-Dokument mit Namensraum

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<ustutt:Abschlussarbeiten xmlns:ustutt="http://universitaetStuttgart.de">
            <ustutt:Masterarbeit id="3643">
                  <ustutt:author> Kerstin Hintermayr </ustutt:author>
                  <ustutt:title> Modellierung Simulation mit Workflow-Choreografie </ustutt:title>
            </ustutt:Masterarbeit>
</ustutt:Abschlussarbeiten>
```

(*.xsd-Datei) zu validieren. Listing 2.18 zeigt eine beispielhafte XML-Schema Definition für das XML-Dokument aus Listing 2.17. Durch das Attribut maxOccurs="unbounded" im Element Masterarbeit wird definiert, dass dieses Element beliebig oft im XML-Dokument vorkommen kann. Durch das Attribut type in den verschiedenen Elementen wird festgelegt, welchem Datentyp das Element entspricht. Der type="xs:string" gehört ebenso wie xs:int und xs:double zu den vordefinierten Datentypen. Durch die Definition weiterer Datentypen ist es auch möglich, über das type-Attribut auf selbst definierte Datentypen zu verweisen.

Listing 2.18 Beispielhafte XML-Schema Definition

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xs="http://universitaetStuttgart.de"
        attributeFormDefault="unqualified"
        elementFormDefault="qualified"
        targetNamespace="http://universitaetStuttgart.de">

        <xs:element name="Abschlussarbeiten">
                  <xs:complexType>
                      <xs:sequence>
                                  <xs:element name="Masterarbeit minOccurs="0"
                                      maxOccurs="unbounded">
                                        <xs:complexType>
                                  <xs:sequence>
                                        <xs:element name="author" type="xs:string" />
                                        <xs:element name="title" type="xs:string" />
                                  <xs:sequence>
                              </xs:complexType>
                              </xs:element>
                        </xs:sequence>
                       </xs:complexType>
            </xs:element>
</xs:schema>
```

Mit der Abfragesprache **XML Path Language** (XPath) ist es möglich, Teile eines XML-Dokumentes zu adressieren und ggf. auch auszuwerten. Listing 2.19 zeigt, wie XPath in einem BPEL-Prozess eingesetzt werden kann, um beispielsweise den Inhalt der Variablen *vorname* und *nachname* mit einem Leerzeichen zu verknüpfen und an eine bestimmte Stelle in der Variable *nameRequest* zu kopieren. Die XPath Funktionen umfassen sowohl Funktionen für Knotenmengen, Strings, boolsche Werte und Zahlen. Eine ausführliche Übersicht über die verschiedenen XPath Funktionen kann der XPath Spezifikation [CD99] entnommen werden.

Listing 2.19 Beispielhafte Verbindung von BPEL und XPath

```
 <bpel:copy>
            <bpel:from>
                  <![CDATA[concat(concat($vorame," "),$nachname]]>
            </bpel:from>
            <bpel:to part="parameters" variable="nameRequest">
                  <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                <![CDATA[oh:name]]>
                  </bpel:query>
            </bpel:to>
</bpel:copy>
```

3. Verwandte Arbeiten

In diesem Kapitel werden konzeptionelle und praktische Arbeiten vorgestellt, welche entweder als Grundlage für diese Masterarbeit relevant sind oder einen ähnlichen theoretischen Bezug haben.

3.1. Ausführung von Festkörpersimulationen auf Basis der Workflow Technologie

Im Rahmen der Diplomarbeit von [Hot10] wird eine bereits vorhandene Festkörpersimulationsanwendung in einen Simulationsworkflow eingebunden. Für die Einbindung in den Simulationsworkflow wurden die in Fortran77 geschriebenen Anwendungen der Festkörpersimulationsanwendung nach einer Analyse verschiedener Lösungsvarianten fachlich gekapselt und als Web Services zur Verfügung gestellt.

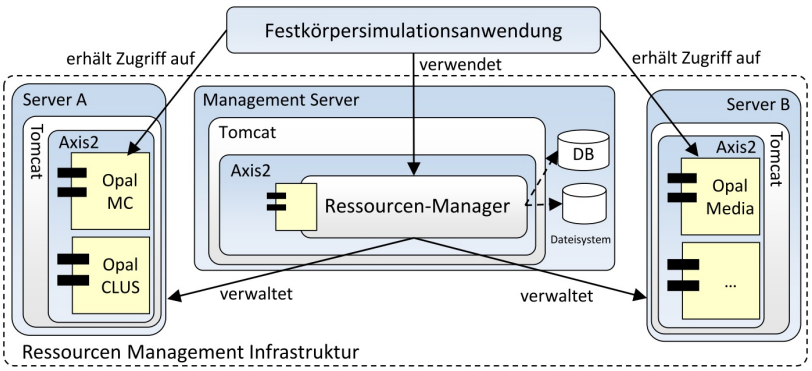


Abbildung 3.1.: Architektur der Ressourcen Management Infrastruktur[1]

Da es bei der parallelen Ausführung mehrerer Instanzen des Simulationsworkflows zu einer Überlastung der verwendeten Infrastruktur kommen kann, wurde im Rahmen der Arbeit zusätzlich ein

[26] Entnommen aus: [Hot10, S.59]

Ressourcen-Manager entwickelt. Dieser ermöglicht als Koordinierungsinstanz eine Regulierung der Web Service-Operationen, sodass es nicht zu einer Überlastung der Infrastruktur kommen kann. Wie aus Abbildung 3.1 hervorgeht, wurde der Ressourcen-Manager separat entwickelt und kann somit auch von anderen Simulationsanwendungen genutzt werden. Ein Datentransfer bzw. Datenaustausch zwischen den einzelnen Services ist durch die zentrale Datenbasis beim Ressourcen-Manager nicht notwendig.

3.2. Kapselung von bestehenden Simulationsanwendungen mithilfe von Web Services

Im Rahmen der Bachelorarbeit von [Nem14] sollte die Funktionalität des bereits existierenden ITAP[2] Molecular Dynamics Programm als Web Service bereitgestellt werden, damit eine spätere Einbindung in einen Simulationsworkflow möglich ist. Um eine flexiblere Verwendung für rechenintensive Teilfunktionalitäten zu ermöglichen, wurden die in Abbildung 3.2 dargestellten Web Services entwickelt.

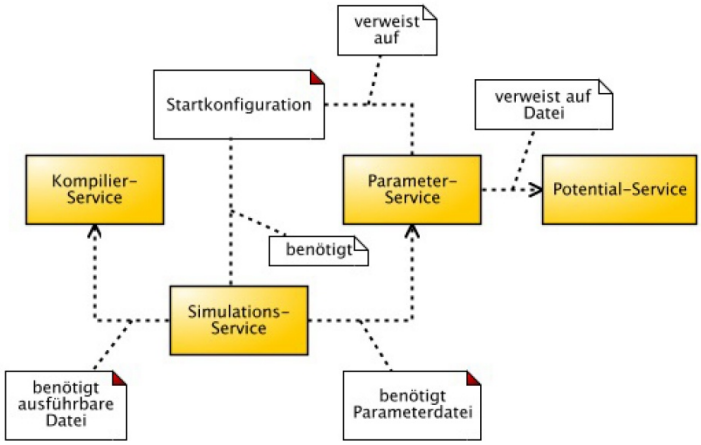


Abbildung 3.2.: Beziehungen zwischen den IMD Web Services[3]

Bei der Realisierung wurde bewusst auf einen zentralen Ressourcen-Manager verzichtet, um die Fehlerkomplexität zu verringern. Damit unter gewissen Umständen dennoch ein zentraler Ressourcen-Manager zum Einsatz kommen kann, wurde bei der Erstellung der Web Services darauf geachtet,

[2] Institut für Theoretische und Angewandte Physik der Universität Stuttgart
[27] Entnommen aus: [Nem14, S.28]

zu jedem Service Methoden zur Verfügung zu stellen, die ein Ressourcen-Manager benötigt. Dies würde beispielsweise bei einer Verlagerung des rechenintensiven Simulations-Service auf einen Hochleistungsrechner den Datenzugriff auf die Daten der anderen Services ermöglichen.

3.3. Modellierung von Scientific Workflows mit Choreographien

Das Ziel der Diplomarbeit von [Son12] war die Entwicklung eines Modellierungswerkzeugs (Chor Designer) für Choreographien sowie die Ausführung von Choreographien in einer Workflowumgebung. Dabei wurde als Basis das Datenmodell BPEL4Chor ausgewählt und darauf aufbauend das in Abbildung 3.3 dargestellte ChorModel entwickelt. Dieses enthält alle grafischen Elemente als eine eigene Entität, welche im Chor Desginer später verfügbar sind.

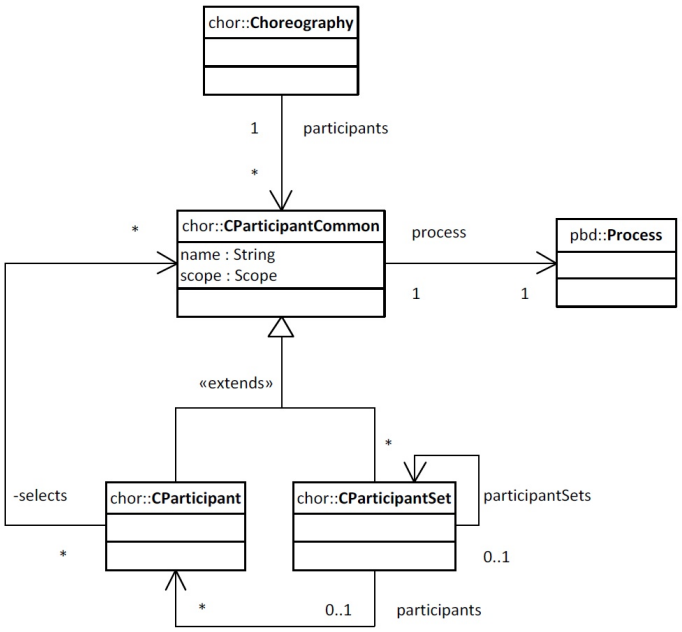


Abbildung 3.3.: Chor Model mit Participants[4]

Damit der Chor Designer später neben dem bereits existierenden Bpel Designer (Erstellung von BPEL-Prozessen) verwendet werden kann, wurde der Chor Desginer als Eclipse Plugin entwickelt. Da auch die spätere Ausführung der Choreographie bei dieser Arbeit betrachtet wurde, wurden zusätzliche Komponenten zur Transformation des Modells in ausführbare BPEL-Prozesse entwickelt.

[28] Entnommen aus: [Son12, S.40]

Diese ermöglichen nach der Modellierung der Choreographie auch einen konkreten Ausbau der entstanden BPEL-Prozesse zu ausführbaren Workflows.

3.4. Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies

In diesem technischen Report beschreiben [WAS+13] anhand von Case Studies wie der BPEL4Chor Designer dazu genutzt werden kann, Choreographien zu modellieren. Dabei wird zu Beginn auf die Vorgehensweise einer Modellierung mit BPEL4Chor, ausgehend von einer existierenden Problembeschreibung in Plaintext oder BPMN, eingegangen. Anschließend erfolgt eine Beschreibung zu BPEL4Chor und dessen Bestandteile, bevor auf den mit der Eclipse Plattform entwickelten BPEL4Chor Designer eingegangen wird. Anhand von zwei Case Studies wird erläutert, wie ausgehend von jeweiligen Beschreibungen (Plaintext oder ein BPMN-Schaubild) die verschiedenen Bestandteile im späteren Choreographiemodell genannt werden und wie diese mit der Prozessbeschreibung zusammenhängen. Anschließend wird in einer kurzen Beschreibung der aktuellen Stand des BPEL4Chor Designers erläutert, welche Aktivitäten dieser zu der Zeit umfasste und welche Punkte und Workarounds damals noch durch den Nutzer durchzuführen waren.

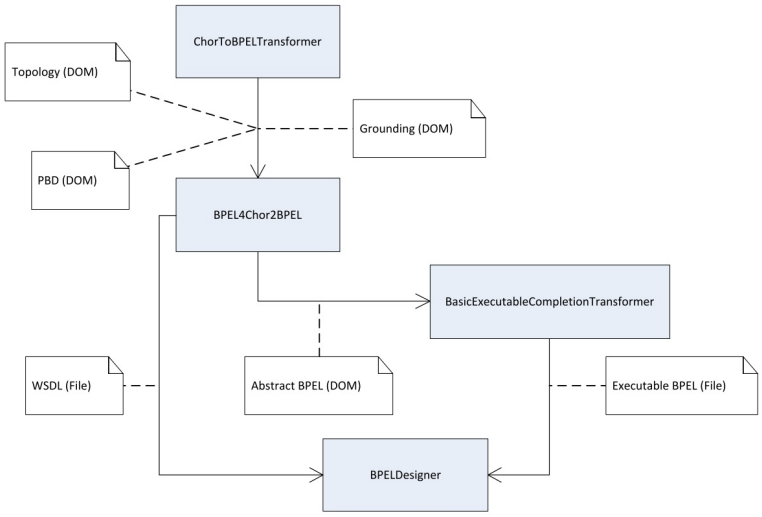


Abbildung 3.4.: Dokumentenfluss zwischen den Komponenten des BPEL4Chor Designers nach [Son12][5]

[29] Entnommen aus: [WAS+13, S.14]

Darauf aufbauend wird anhand dem in Abbildung 3.4 dargestellten Dokumentenfluss die notwendigen Transformationsschritte zum Erreichen von ausführbaren BPEL-Prozessen im Anschluss an die Modellierung erläutert. Zum Abschluss werden die Schritte der Vorgehensweise kurz zusammengefasst und aufgezeigt, dass es möglich ist, mit dem BPEL4Chor Designer Choreografien zu modellieren und diese anschließend zu ausführbaren BPEL-Prozessen zu transformieren.

3.5. Choreographing Web Services

In dem Artikel „Choreographing Web Services“ stellen [BWR09] die Multiagent Protocols (MAP) Web Service choreography language vor. Dabei beruht MAP - analog zu BPEL4Chor - auf dem Choreographie-Modellierungsansatz „interconnection model“. Dabei können mit MAP entwickelte Choreographien mittels Modelprüfung verifiziert werden und in einem verteilten Peer-to-Peer Netzwerk eingesetzt werden. Wie aus der in Abbildung 3.5 dargestellten MAP-Syntax ersichtlich wird, dient MAP lediglich zur Formulierung von Choreographien und nicht als reine Programmiersprache.

$P \in$ Protocol	::=	$n\ (r\{M\})^{+}$	(Choreography)
$M \in$ Method	::=	$\texttt{method}(\phi^{(k)})$ = op	(Method)
$op \in$ Operation	::=	α	(Action)
	|	op_1 `then` op_2	(Sequence)
	|	op_1 `or else` op_2	(Choice)
	|	op_1 `par` op_2	(Parallel Composition)
	|	`waitfor[:imax]` op_1 `timeout[:tmax]` op_2	(Iteration)
	|	$\texttt{call}(\phi^{(k)})$	(Recursion)
	|	(op)	(Precedence)
$\alpha \in$ Action	::=	ϵ	(No Action)
	|	$\phi^{(k)}$ = $\texttt{service}(\text{ws}^{+}, \phi^{(l)})$ `fault` $\phi^{(m)}$	(Service Invocation)
	|	$p(\phi^{(k)})$ => $\texttt{peer}(\phi^{(1)}, \phi^{(2)})$	(Send)
	|	$p(\phi^{(k)})$ <= $\texttt{peer}(\phi^{(1)}, \phi^{(2)})$	(Receive)
$\phi \in$ Term	::=	$\_ \mid p \mid r \mid c : \tau \mid v : \tau$	(Terms)
$\tau \in$ Type	::=	$ptype \mid rtype \mid rpctype$	(Types)
ws	::=	$\texttt{def}(\text{config}^{(k)})$	(Web Service Definition)
$config$	::=	$\langle$name, value$\rangle$	(Configuration Pair)

Abbildung 3.5.: MAP Syntax[6]

[30] Entnommen aus: [BWR09, S.155]

4. Analyse der bestehenden Simulationsanwendungen

In diesem Kapitel wird zu Beginn auf den Begriff Simulation und den damit verbundenen Kontext eingegangen. Anschließend werden die bestehenden Simulationsanwendungen analysiert und die daraus abgeleiteten Anforderungen für die Modellierung der automatischen Festkörpersimulation mit Workflow-Choreographien vorgestellt.

4.1. Simulation

[SH05, S.397f.] erklärt den den Simulationsbegriff wie folgt:

> „Mit Simulationsverfahren werden (meistens zeitliche) Verfahrensabläufe computergestützt durchgeführt. Nach einer Richtlinie des Vereins Deutscher Ingenieure (VDI) ist [die] ‚...Simulation die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.'"

Bei der in dieser Masterarbeit zu erstellenden Festkörpersimulation werden u. a. die strukturellen Veränderungen von metallischen Festkörpern über große Zeiträume simuliert. Werden dabei mehrere Simulationsmethoden über jeweils verschiedene Skalenbereiche kombiniert, wird nach [Sch12] von einer Mehrskalensimulation gesprochen. Abbildung 4.1 zeigt anhand der Größe der zu simulierenden Struktur für die einzelnen Skalenbereiche verschiedene Simulationsmethoden auf. Nach [Sch12] ermöglicht dies beispielsweise Ingenieuren bei einem Bauteil bis auf die atomare Ebene zu simulieren. Nach [Höf, S.20] ist die Monte-Carlo-Methode „[...] ein stochastisches Simulationsverfahren [bei dem] auf Basis von Zufallszahlen die Wahrscheinlichkeit zur Gewichtung des Auftretens eines bestimmten Systemzustandes ermittelt [wird]." Mit der Molekulardynamik hingegen lassen sich nach [Sch12] Simulationen in der Größenordnung von 10- bis 100 Nanometern durchführen, wobei die Lage und die Trajektorien[31] von Atomen beobachtet werden können.

[31] Nach [Sch12, S.3] bezieht sich Trajektorie in diesem Fall auf das Verhalten der Atome.

Durch eine Kombination bzw. Kopplung der Monte-Carlo-Methode mit der Molekulardynamik ist es möglich, eine gekoppelte Festkörpersimulation zu erstellen. Da das Ziel dieser Masterarbeit sich mit der Modellierung einer gekoppelten Festkörpersimulation befasst, bei der zuerst die Kommunikation der zu koppelnden Simulationsworkflows modelliert wird, wird in den nachfolgenden Unterkapiteln auf die zu koppelnden Simulationsanwendungen (bzw. Workflows) eingegangen.

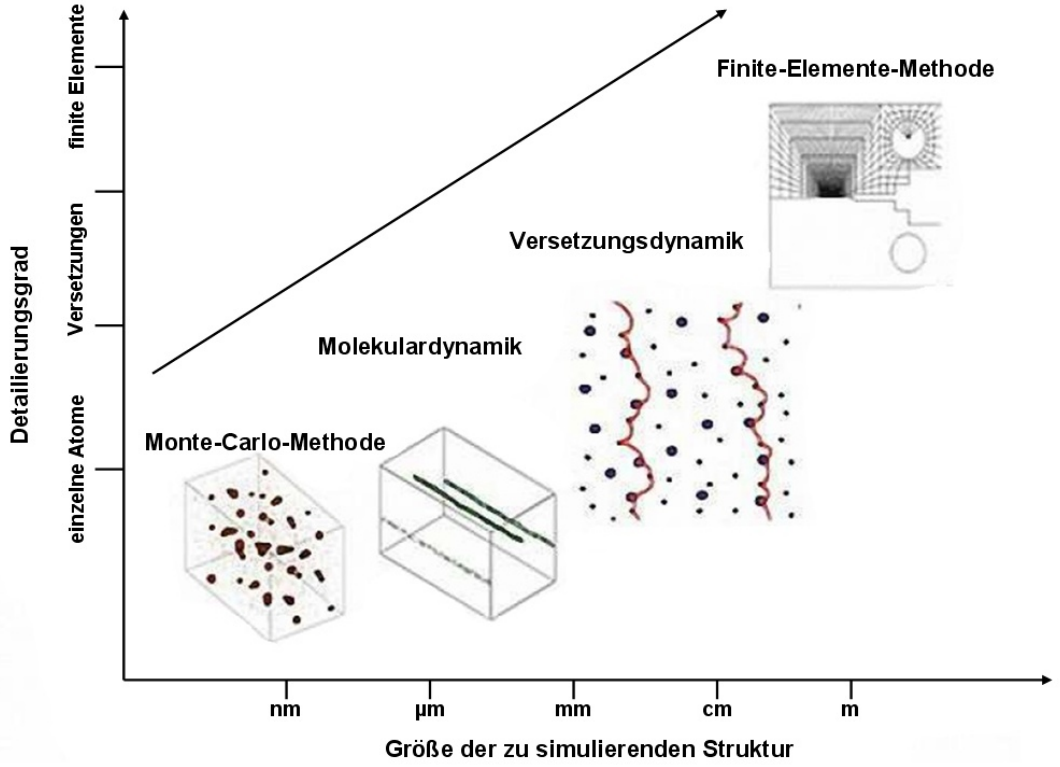


Abbildung 4.1.: Mehrskalensimulation[1]

[32] Mit Änderungen entnommen aus: [Sch12, S.3]

4.2. Simulationsanwendung OPAL

Am Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre (IMWF) der Universität Stuttgart wurden mehrere monolithische Fortran77-Anwendungen entwickelt, welche zusammen die Simulationsanwendung OPAL ergeben (vgl. hierzu Abbildung 4.2). Das nachfolgende Kapitel bezieht sich, falls nicht anders angegeben, auf [Hot10].

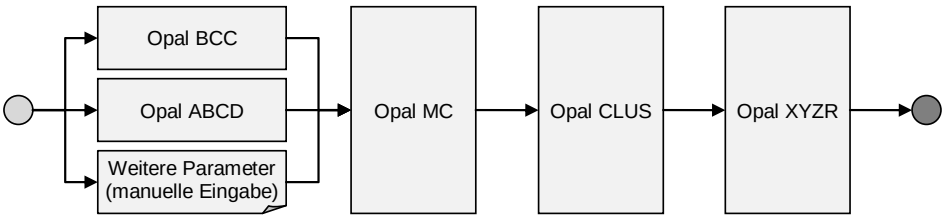


Abbildung 4.2.: Simulationsanwendung OPAL

OPAL steht hierbei für Ostwald-Ripening of Precipitates on an Atomic Lattice - übersetzt: Ostwald-Reifung von Ausscheidungen auf einem Atomgitter. Die Aufgabe der Simulationsanwendung OPAL ist dabei die thermische Alterung von Festkörpern zu simulieren. Das bedeutet, die strukturellen Veränderungen von metallischen Festkörpern über große Zeiträume hinweg zu simulieren. Dabei wird mithilfe eines Monte-Carlo-Algorithmus die Bildung von kohärenten Ausscheidungen im kubisch raumzentrischen α-Eisen auf atomarer Ebene simuliert (vgl. hierzu Abbildung 4.3). Nach [Sch12] beschleunigt die thermische Belastung die auf der Ausscheidungsbildung beruhende Festigkeitsabnahme des Materials. Anhand der Monte-Carlo-Simulation (MC-Simulation) lassen sich die Vorgänge besser verstehen und die Abschätzung der Lebensdauer verbessern.

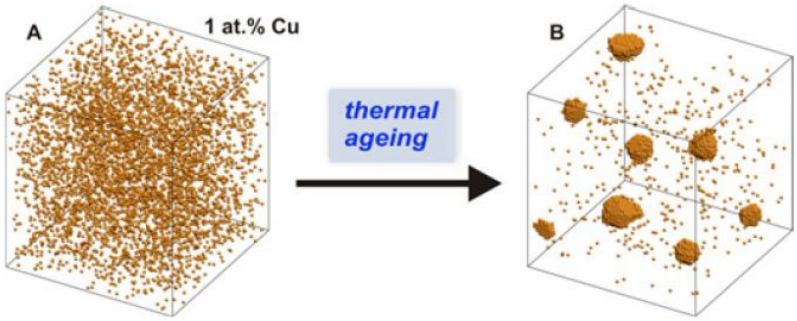


Abbildung 4.3.: Bildung von Ausscheidungen mittels der MC-Simulation[2]

[33] Mit Änderungen entnommen aus: [SHK12, S.5]

Die Ausführung der Simulationsanwendung OPAL lässt sich in die drei verschiedenen Phasen Vorbereitung, Simulation und Analyse einteilen:

1. In der **Vorbereitungsphase** der Simulationsanwendung werden durch die Anwendungen OpalBCC und OpalABCD die benötigten Konfigurationsdateien erstellt. OpalBCC steht hierbei für Body Centered Cubic und erzeugt ein anfänglich kubisch raumzentriertes Kristallgitter, welches als Grundlage für die Monte-Carlo-Schritte benötigt wird. Die Anwendung OpalABCD hingegen berechnet durch die Eingabe der einzelnen Bindungsenergien zwischen den jeweiligen Atompaaren der Atomsorten, sowie den Aktivierungsenergien in Abhängigkeit von inter-atomaren Wechselwirkungs-Energien, Leerstellen-Wechselwirkungsenergien und Nachbarschaftsbeziehungen die Energiekonfiguration für die Monte-Carlo-Simulation.

2. Während der **Simulationsphase** erhält die Anwendung OpalMC als Eingabeparameter das zuvor generierte Kristallgitter, die Energiekonfiguration sowie weitere Konfigurationsparameter wie beispielsweise die Anzahl der Snapshots, die Snapshot-Frequenz und die Checkpoint-Frequenz. Durch die Snapshot-Frequenz weiß die Anwendung OpalMC nach wie vielen Monte-Carlo-Schritten ein Snapshot, d. h. ein Schnappschuss des derzeitigen Kristallgitter-Zustands, gemacht werden soll. Die Checkpoint-Frequenz hingegen definiert nach wie vielen Snapshots ein Sicherungspunkt (Checkpointdatei) angelegt wird. Die Checkpointdatei enthält dann den genauen Zustand der Simulation zu einem bestimmten Zeitpunkt.

3. In der **Analysephase** werden die erstellten Snapshots analysiert und ausgewertet. Dabei ist OpalCLUS für das Auffinden der Ausscheidungen im Snapshot zuständig und OpalXYZR berechnet anschließend die Clusterradien im Kristallgitter. Des Weiteren wird zum Abschluss mit gnuPlot die Grafik doad.ps erstellt. Aus dieser Grafik geht der Fortschritt der Ausscheidungsbildung in Abhängigkeit von der Zeit hervor. Das heißt es wird ersichtlich, wie viel Kupfer im Eisen gelöst ist (thermisches Gleichgewicht).

[Hot10] hat im Rahmen seiner Diplomarbeit die einzelnen Komponenten der Simulationsanwendung OPAL gekapselt, in logische Einheiten gegliedert und die Fachlichkeit der Komponenten über Web Services verfügbar gemacht. Dadurch war es möglich, die Web Services in den Simulationsworkflow OPAL (vgl. hierzu Abbildung 4.4) einzubinden, sodass das Ziel der Diplomarbeit, die bestehende Simulationsanwendung OPAL zu automatisieren, erreicht wurde.

Während der Ausführung der automatisierten OPAL-Simulationsanwendung entsteht eine Vielzahl verschiedener Dateien (vgl. Abbildung 4.5), welche entweder für die weitere Ausführung von OPAL oder als Eingabeparameter für anschließende Simulationen benötigt werden.

[34] Entnommen aus: [Hot10, S.44]

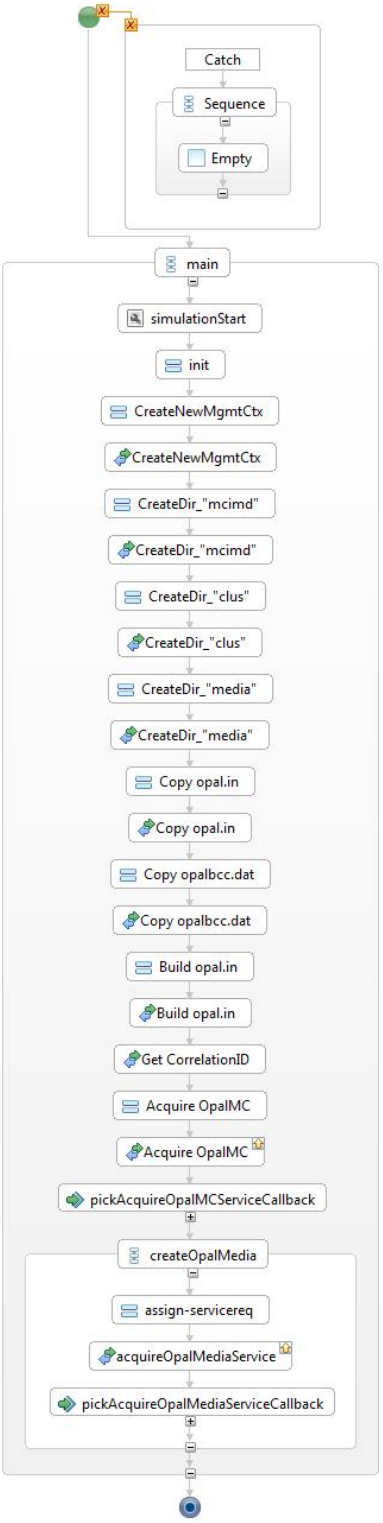


Abbildung 4.4.: Ausschnitt aus dem bisherigen Simulationsworkflow OPAL

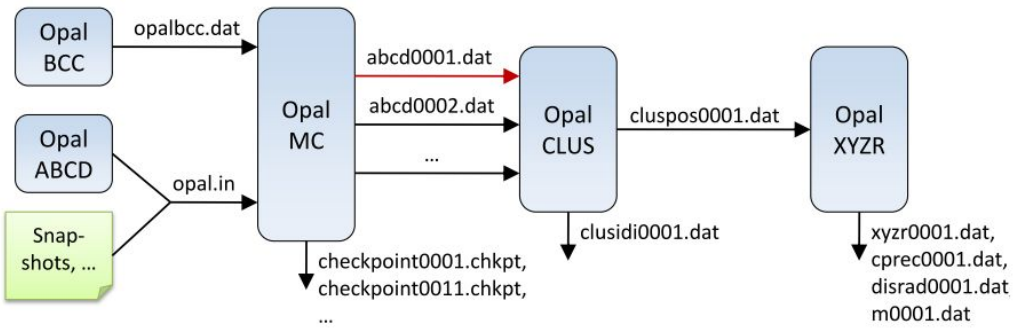


Abbildung 4.5.: Übersicht über die Input- und Output-Daten während der Ausführung des Simulationsworkflow OPAL[3]

4.3. Simulationsanwendung IMD

Die Simulationsanwendung IMD (ITAP Molecular Dynamics) des Instituts für Theoretische und Angewandte Physik der Universität Stuttgart ermöglicht, ausgehend von der OPAL Checkpointdatei, die Simulation des anisotrope Zugverhalten mittels einer Molekulardynamik-Simulation. Dabei ist ein Zugversuch nach DIN EN 10002 ein genormtes Verfahren zur Prüfung von metallischen Werkstoffen. Durch die axiale, stetig wachsende Beanspruchung kann das mechanische Verhalten analysiert werden. Dadurch lässt sich anschließend herauszufinden, wie sich die durch OPAL simulierten Ausscheidungen auf das Werkstoffverhalten auswirken.

Damit die bestehende Simulationsanwendung IMD, analog zu OPAL, automatisiert werden kann, hat [Nem14] die bestehenden Funktionalitäten von IMD mithilfe von Web Services gekapselt und für eine Einbindung in Workflows zur Verfügung gestellt. Die nachfolgende Beschreibung der IMD Web Services beruht auf [Nem14]: Der Kompilier-Service baut beispielsweise mit der Methode compileBinary() aus der IMD Bibliothek eine Anwendung, bestehend aus Algorithmen zur Berechnung der Molekulardynamik-Simulation, zusammen. Mithilfe der Methode writePotential() des Potential-Service ist es möglich, Potentialdateien anzulegen, welche für die Ausführung der Molekulardynamik-Simulation benötigt werden. Im Gegensatz dazu können mit der Methode writeParameterFile() des Parameter-Service Konfigurationsdateien (Parameterdateien) für jede einzelne Simulationsausführung erstellt werden. Die eigentliche Simulationsausführung, der Zugversuch, wird über die Web Service-Methode execute() des Simulations-Service ausgeführt. Eine Übersicht über die verschiedenen Web Service-Methoden mit jeweils einer kurzen Beschreibung ist der nachfolgenden Tabelle 4.1 zu entnehmen.

Web Service	Methode	Beschreibung
Kompilier-Service	compileBinary()	Erwartet eine IMD Modulliste und liefert nach Fertigstellung den Namen der Binärdatei.
	getBinaryFileByName()	Erwartet den Namen einer vorhandenen Binärdatei und liefert diese zurück.
	deleteBinaryByName()	Erwartet den Namen einer Binärdatei, welche anschließend gelöscht wird.
	listBinarys()	Liefert eine Liste mit allen vorhandenen Binärdateien.
	getBinaryPathByName()	Liefert den Pfad zu der entsprechenden Binärdatei.
Potential-Service	getPotentialList()	Liefert eine Liste von verfügbaren Potentialdateien.
	getPotentialFileByName()	Liefert eine bestimmte - durch Namen angegebene - Potentialdatei zurück.
	getPotentialPathByName()	Liefert den Pfad zu einer bestimmten - durch Namen angegebenen - Potentialdatei zurück.
	writePotential()	Schreibt eine Potentialdatei mit den übergebenen Parameter und speichert diese ab.
	deletePotential()	Löscht eine vorhandene Potentialdatei.
Parameter-Service	writeParameterFile()	Speichert eine Parameterdatei mit den übergebenen Werten ab.
	getParameterFileByName()	Liefert die, zum Namen gehörende, Parameterdatei zurück.
	getParameterList()	Liefert den Pfad zu einer bestimmten - durch Namen angegebenen - Parameterdatei zurück.
	deleteParameterFile()	Löscht eine vorhandene Parameterdatei.
Simulation-Service	execute()	Startet eine Molekulardynamik-Simulation.
	abort()	Bricht eine laufende Molekulardynamik-Simulation ab.
	listFilesBySimulationId()	Liefert eine Liste aller Dateien einer Molekulardynamik-Simulation.
	getFileByNameAndSimulationId()	Liefert eine Datei aus einer bestimmten Simulation.
	deleteSimulation()	Löscht eine Molekulardynamik-Simulation und alle dazugehörigen Dateien.

Tabelle 4.1.: Übersicht der Web Service Methoden der verschiedenen IMD Web Services

Um nachzuweisen, das ein Top-Down-Ansatz bei der Modellierung einer Choreographie möglich ist, ist es wichtig das bei der Verfeinerung der Prozesse der einzelnen Choreographieteilnehmer nicht komplett auf bereits existierende Prozesse zurückgegriffen wird. In dieser Masterarbeit wird lediglich der bereits bestehende OPAL-Simulationsworkflow in den generierten OPAL-Teilnehmer-Prozess

eingefügt. Für den Teilnehmer IMD ist es notwendig, lediglich auf Basis der Choreographiegenerierung, einer durch [Nem14] vorgeschlagene erste grafischen Orchestrierungsmöglichkeit der gekapselten IMD Web Services und die im nächsten Unterkapitel aufgeführten Anforderungen an den IMD-Prozess, den Prozess vollständig neu zu orchestrieren.

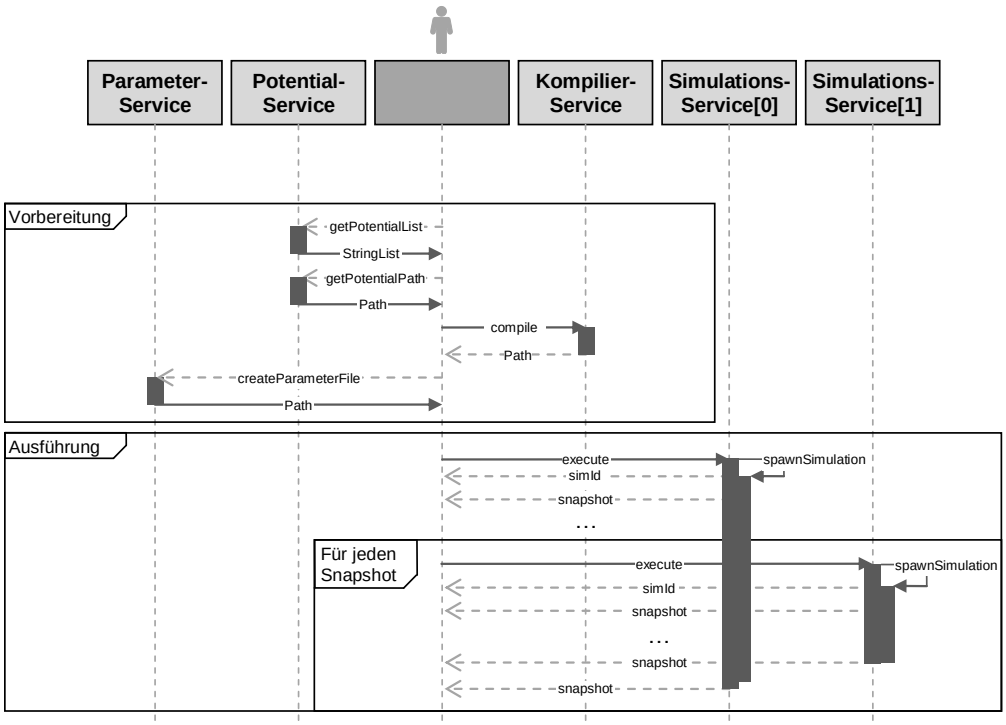


Abbildung 4.6.: Mögliche IMD-Orchestrierung[4]

Eine erste technische Analyse der Funktionsweise der IMD Web Services ergab jedoch, dass mit den IMD-Parametereinstellungen die IMD-Simulation mehrere Stunden bis Tage in Anspruch nehmen würde. Aus diesem Grund wurde entschieden, IMD nicht wie OPAL lokal, sondern in einer virtuellen Umgebung auf einem SimTech-Server der Universität Stuttgart zu installieren. SimTech steht in diesem Zusammenhang für Simulationstechnolgien. Die dadurch entstehenden Herausforderungen wie beispielsweise Zugriff auf die Virtuellen Maschine (VM) oder den Datentransfer vom lokalen Rechner zur VM während der BPEL-Prozessausführung werden in Kapitel 5.3 näher erläutert und ein möglicher Lösungsweg dafür vorgestellt.

[35] Entnommen aus [Nem14, S. 38]

4.4. Anforderungen an die Realisierung der gekoppelten Festkörpersimulation mit einer Workflow-Choreographie

Tabelle 4.2 zeigt, die Anforderungen an die Realisierung der gekoppelten Festkörpersimulation auf. Diese ergaben sich aus Gesprächen mit den zuständigen Betreuern der Arbeit.

Nr.	Anforderung	Beschreibung
A1	Modellierungsansatz	Die gekoppelte Festkörpersimulation soll mit dem Top-Down-Modellierungsansatz entwickelt werden.
A2	Eingabemaske	Benötigte Parameter sollen über eine gemeinsame Eingabemaske zu Beginn der gekoppelten Festkörpersimulation abgefragt werden.
A3	Änderungen	Änderungen an der Infrastruktur (BPEL Designer / Chor Designer) sowie der bestehenden Simulationsworkflows sind wenn notwendig vorzunehmen.
A4	Simulationsaufruf IMD	Durch die gekoppelte Festkörpersimulation soll der IMD-Prozess direkt aus der OPAL-Simulation aufgerufen werden. Der IMD Aufruf soll nur unter der Bedingungen erfolgen, dass der mittlere Radius des OPAL-Checkpoints eine bestimmte Größe erreicht hat. Checkpointdateien die nicht an IMD geschickt werden sollen, um Speicherplatz zu sparen, entsprechend gelöscht werden.
A5	Simulationsablauf IMD	Zu Beginn der IMD-Simulation soll die compileBinary() Methode nur aufgerufen werden, wenn noch keine kompilierte IMD-Anwendung zur Verfügung steht. Anschließend sollen für jede OPAL-Checkpointdatei drei sequentielle IMD-Simulationen (Aufheizen, Halten auf Temperatur und Zugversuch) durchgeführt werden.
A6	Grafik je OPAL-Checkpoint	Im Anschluss an den Zugversuch (dritte IMD-Simulation) durch den IMD-Prozess für eine OPAL Checkpointdatei, soll eine Grafik auf Basis der npt-deform.eng Datei erstellt werden. Dabei soll die Spalte 1 (Zeit in 10^{-15}) als x-Achse und Spalte 15 (pzz) als y-Achse verwendet werden. Für die korrekte Darstellung der Grafik ist es notwendig, die Werte der Spalte 15 mit -160,2 zu multiplizieren damit die Werte in der Einheit Gigapascal dargestellt werden.
A7	Grafik	Im Anschluss an die erstellten Grafiken je OPAL-Checkpoint soll eine Grafik erstellt werden, welche die Maximalwerte der bisher erstellten Grafiken zu den jeweiligen OPAL-Checkpoints aufzeigt.
A8	Parameter für die Simulationsauführung	Für die Ausführung der gekoppelten Festkörpersimulation sollen bestimmte Parameter eingehalten werden.

Tabelle 4.2.: Allgemeine Anforderungen an die gekoppelte Festkörpersimulation mit einer Workflow-Choreographie

Wie aus der Anforderung A8 hervorgeht, sollen für die Ausführung der gekoppelten Festkörpersimulation bestimmte Parameter gesetzt werden. Die Beschreibung der einzelnen Parameter können der Tabelle 4.3 entnommen werden. Dabei gibt der erste Buchstaben in der Nr. an, ob es sich um ein OPAL-Parameter (O) oder um einen IMD-Parameter (I) handelt.

Nr.	Parameter	Beschreibung
O1	Anzahl Snapshots	Die Anzahl gibt an, wie viele Snapshots (abcdXXXX.dat-Datei) während der OPAL-Simulation erstellt werden sollen. Ziel: 100 Snapshots. Zu Testzwecken kann diese Zahl während der Entwicklung flexibel verändert werden.
O2	Snapshot-Frequenz	Gibt an, nach wie vielen Monte-Carlo-Schritten ein Snapshot erstellt werden soll. Ziel: 10^{11} = 100.000.000.000. Zu Testzwecken wird während der Entwicklung die Zahl auf 1000 gesetzt.
O3	Checkpoint-Frequenz	Gibt an, nach wie vielen Snapshots eine Checkpointdatei erstellt wird. Muss auf 1 gesetzt sein, damit zu jedem Snapshot eine Checkpointdatei erstellt wird. Erst während der OPAL-Ausführung soll entschieden werden, welche Checkpointdatei wieder gelöscht werden kann.
O4	Radius	Anhand dieser Variable soll entschieden werden, wann ein Aufruf von IMD erfolgt. Mögliches default Eingabeintervall [min-max]: [0,5-1,5]. Entspricht der mittlere Radius bei der Snapshotanalyse dem Min-Wert, soll die Checkpointdatei des Snapshots an IMD zur Analyse übergeben werden und der Min-Wert um 0,1 erhöht werden. Da der Max-Wert in diesem Fall 1,5 beträgt, entstehen so - bei min = 0,5 - maximal 11 IMD-Aufrufe aus der OPAL-Simulation.
I3	compileBinary()	Für den ersten Simulationsaufruf wird die kompilierte IMD-Anwendung imd_eam_glok_homedef_stress und für den zweiten und dritten IMD-Aufruf die kompilierte IMD-Anwendung imd_eam_npt_axial_homedef_stress benötigt.
I4	executeWithPath()	Beim ersten Simulationsaufruf soll die Parameterdatei param_glok_mod, beim zweiten Simulationsaufruf die Parameterdatei param_npt und beim dritten Simulationsaufruf die Parameterdatei param_npt_def verwendet werden.
I5	Parameterdateien	Die genauen Inhalte der Parameterdatei param_glok_mod, param_npt und param_nptdef können den Listings aus Anhang A.1 entnommen werden. Zu Testzwecken während der Entwicklung werden einzelne Variablen der Parameterdateien (u.a. maxsteps und checkpt_int) gesenkt, um die Ausführung der Simulation zu beschleunigen.

Tabelle 4.3.: Parameter für die OPAL-/ IMD-Simulation

5. Modellierung der Workflow-Choreographie

Dieses Kapitel befasst sich mit dem Modellierungsansatz und der technischen Vorgehensweise zur Erstellung der Choreographie. Es werden festgestellte Herausforderungen während der Modellierung genannt und mögliche Lösungen aufgeführt.

5.1. Modellierungsansatz

Bei Modellierungen gibt es generell zwei verschiedene Ansätze. Der Bottom-Up-Ansatz beginnt, bezogen auf den Kontext der Workflow-Choreographie, bei den konkreten Workflows der einzelnen Choreographieteilnehmer. Darauf aufbauend wird die Choreographie abgeleitet. Im Gegensatz hierzu steht der Top-Down-Ansatz. Bei diesem erfolgt zuerst die Analyse des zu lösenden Problems. Anschließend folgt die Modellierung der Kommunikation zwischen den Teilnehmern als Choreographie. Dabei werden jedoch nur die Kommunikationsaktivitäten der Choreographieteilnehmer untereinander modelliert und nicht alle für den späteren Workflow der Teilnehmer notwendigen Aktivitäten. Aus dem entstandenen Choreographiemodell werden dann automatisch die abstrakten Workflows der einzelnen Teilnehmer generiert. Im letzten Schritt sind die abstrakten Workflows durch die Teilnehmer noch um weitere Aktivitäten zur Prozessverarbeitung zuergänzen, sodass ausführbare Workflows entstehen.

In der vorliegenden Masterarbeit wird die Realisierung einer gekoppelten Festkörpersimulation nach dem Top-Down-Ansatz (vgl. hierzu Abbildung 5.1) durchgeführt. Dies bedeutet, dass bei der Modellierung der Choreographie, wie in Kapitel 2.1.2 beschrieben, das interconnection-Modell verwendet wird. Für eine spätere Evaluierung der Choreographie wird der Workflow des Teilnehmers OPAL mit dem bereits bestehenden OPAL-Simulationsworkflow verglichen.

[36] Entnommen aus: [WK13]

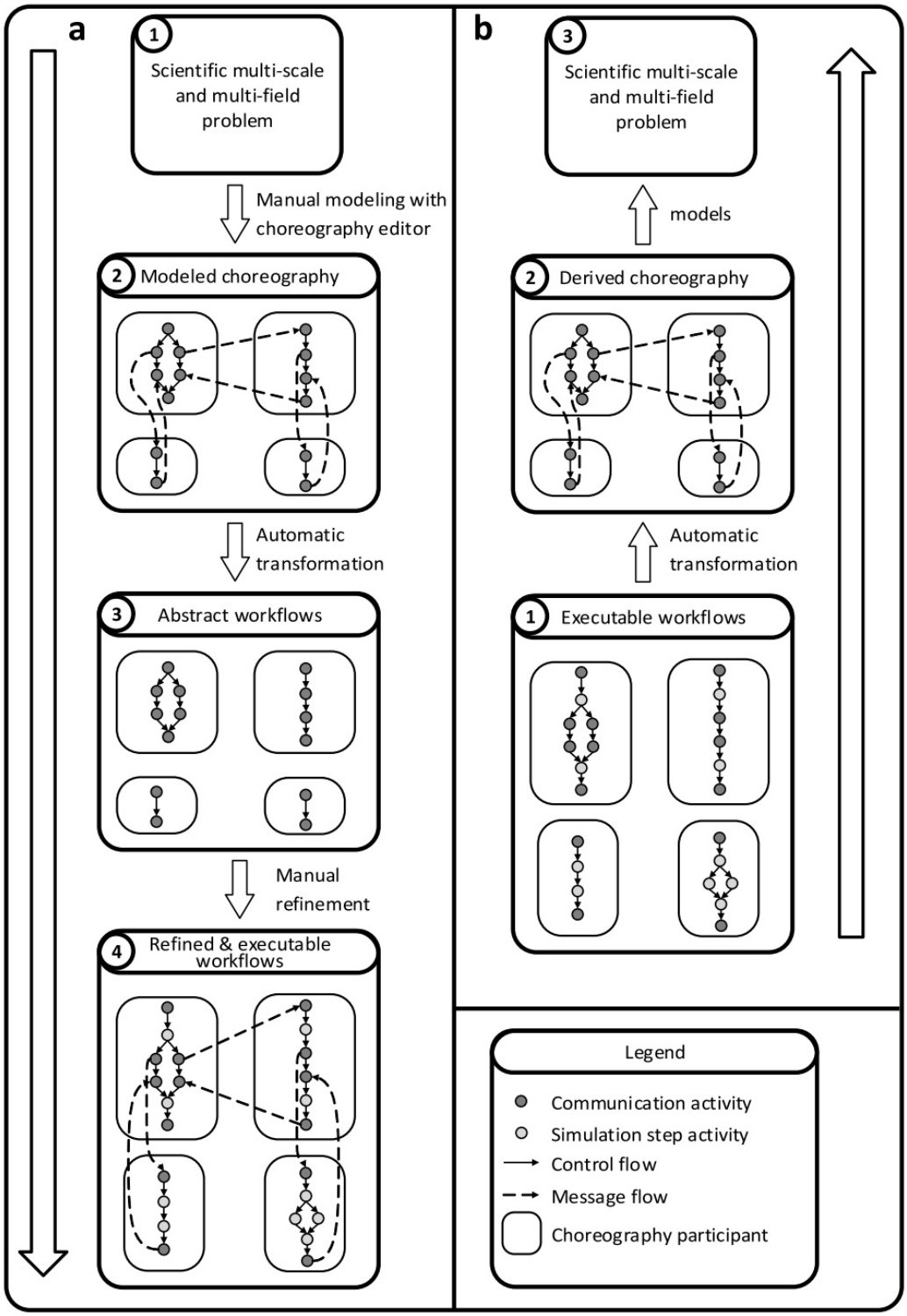


Abbildung 5.1.: Modellierungsansatz: a)Top-Down b) Bottom-Up[1]

5.2. Technische Vorgehensweise

Abbildung 5.2 zeigt die technische Vorgehensweise der Choreographie-Modellierung zur Umsetzung des im vorherigen Unterkapitel beschriebenen Modellierungsansatzes.

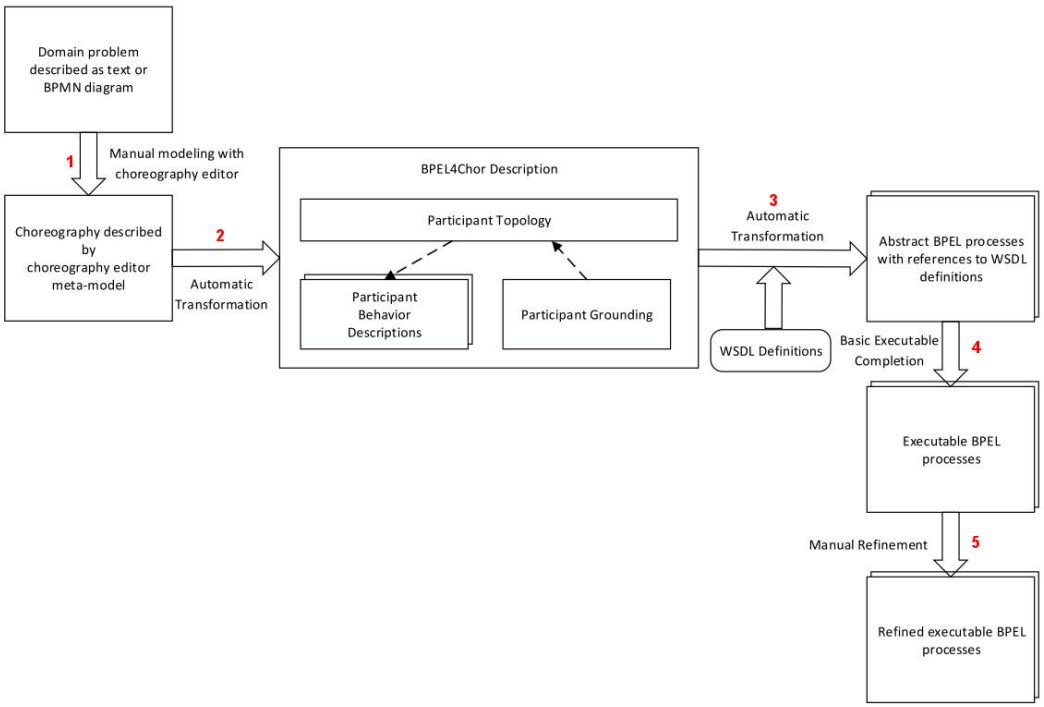


Abbildung 5.2.: Vorgehensweise zur Generierung von ausführbaren Prozessen auf Basis einer mit dem Chor Designer erstellten Choreographie[2]

Zu Beginn erfolgt die Modellierung der Choreographie mit dem Chor Designer (1), sodass am Ende in der Entwicklungsumgebung Eclipse eine *.chor-Datei entsteht. Dabei wird für alle Teilnehmer der Choreographie in der grafischen Oberfläche des Chor Designers ein Participant- oder ParticipantSet-Rechteck eingefügt. Innerhalb dieser Rechtecke sind dann anschließend alle Kommunikationsaktivitäten der einzelnen Teilnehmer zu modellieren. Die Verwendung eines ParticipantSet erfolgt, wenn zum Zeitpunkt der Modellierung noch nicht klar ist, wie viele Teilnehmer es von diesem Typ bei der Ausführung es Prozesses gibt. Für den Nachrichtenaustausch zwischen den Teilnehmern werden Verbindungspfeile (Message Links) zwischen den Kommunikationsaktivitäten der Teilnehmer verwendet. Des Weiteren können über die Nachrichtenlinks Teilnehmerreferenzen versendet werden.

[37] Entnommen aus [WAS+13, S.4]

Abbildung 5.3 zeigt eine beispielhafte, mit dem Chor Designer modellierte Choreographie für eine Taxianwendung.

Abbildung 5.3.: Ausschnitt einer Choreographie für eine Taxianwendung[3]

Nach Abschluss der Modellierung der Choreographie ist es mit der Komponente ChorToBPELTransformer des Chor Designers möglich, die optionale automatische Transformation in die Choreographiesprache BPEL4Chor anzustoßen (2). Als Ergebnis entstehen die in der BPEL4Chor-Spezifikation enthaltenen Dokumente Topology, Grounding und Participant Behavior Descriptions.

Im dritten Schritt wird mit der Komponente BPEL4Chor2BPEL die automatische Transformation zu abstrakten BPEL-Prozessen und WSDL-Dokumenten angestoßen (3). Als Grundlage (Input) dienen theoretisch die aus dem ersten Schritt erzeugten Dokumente (Topology, Grounding und Participant Behaviour Description) sowie die WSDL-Dokumente der einzelnen Teilnehmer-Prozesse. Die Transformation gelingt allerdings auch nur auf Basis des Choreographiemodells. Als Ergebnis der Transformation entstehen die abstrakten BPEL-Prozesse sowie angereicherte WSDL-Dokumente. Durch den Transformer werden zusätzlich neue <PartnerLink>-Elemente für <invoke>-, <receive>-, <reply>- und <onMessage>-Aktivitäten sowie Variablen für jede <forEach>-Iteration über ein <ParticipantSet> generiert.

[38] Entnommen aus: [WAS+13, S.9]

Wie aus Abbildung 5.2 hervorgeht, folgt im Anschluss an die Transformation nach BPEL die Basic Executable Completion (4). Hierbei werden folgende Punkte vervollständigt (in Anlehnung an [Son12]):

1. Abstrakte Namespaces werden durch ausführbare Namespaces von BPEL ersetzt.
2. Das abstractProcessProfile wird aus dem BPEL Prozess entfernt.
3. Die Platzhalter (<opaqueActivity>) werden durch leere <empty>-Aktivitäten ersetzt.
4. Für jede Platzhaltervariable (opaque variable) innerhalb einer <invoke>-, <receive>- oder <reply>-Aktivität werden neue Variablen erstellt.
5. Der Name von Platzhalter-Variablen (input und output) in <invoke>-, <receive>-, <reply>- und <onMessage>-Aktivitäten wird auf den Wert portType_operation_message gesetzt.

Als fünfter Schritt ist die manuelle Verfeinerung der generierten, abstrakten Prozesse erforderlich. Da bisher lediglich die Kommunikationsaktivitäten zwischen den verschiedenen Teilnehmern betrachtet wurden, sind in diesem letzten Schritt alle weiteren Aktivitäten und somit die Businesslogik der einzelnen Teilnehmer in jedem Teilnehmer-Prozess zu ergänzen (5).

5.3. Herausforderungen

Während der Entwicklung der Choreographie und der Analyse der bestehenden Simulationsanwendung OPAL wurden mehrere Herausforderungen (vgl. Abbildung 5.4) festgestellt. Nachgehend werden die einzelnen Herausforderungen und die gewählten Lösungswege aufgeführt und näher erläutert.

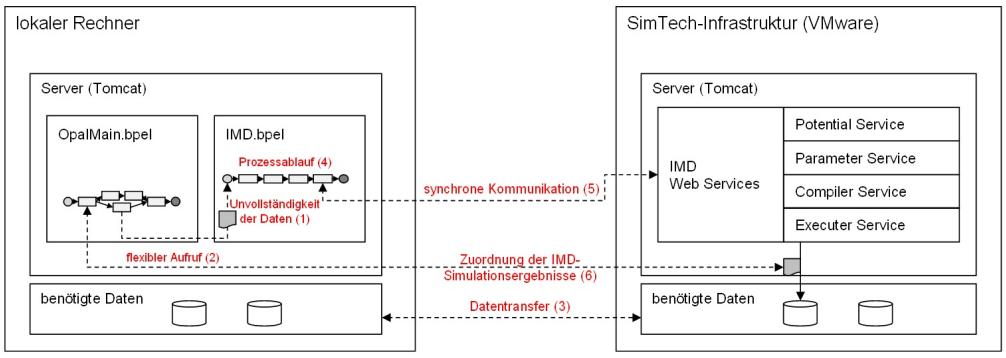


Abbildung 5.4.: Festgestellte Herausforderungen während der Choreographiemodellierung und der Ausführung der gekoppelten Festkörpersimulation

Herausforderung 1: Die Checkpointdateien aus OPAL können nicht direkt als Input für den IMD-Prozess verwendet werden, da am Anfang der Checkpointdatei Informationen für den IMD-Prozess fehlen.

Damit die von der Simulationsanwendung OPAL generierten Checkpointdateien als Input-Dateien für den IMD Workflow genutzt werden können, müssen am Anfang der Checkpointdatei sieben Zeilen (vgl. Listing 5.2) eingefügt werden. Um dies zu realisieren, wurde im ersten Schritt analysiert, an welcher Stelle im OPAL-Simulationsworkflow die Checkpointdatei erstellt wird. Wie auch aus Abbildung 4.5 hervorgeht, werden die Checkpointdateien durch die OpalMC-Anwendung erstellt. Im zweiten Schritt wurde der Quellcode der OPAL-Simulationsanwendung in Eclipse importiert und der OpalMCService (welcher die Anwendung OPAL MC kapselt) analysiert. Diese zweite Analyse ergab, dass die Java-Klasse OpalMCResultDispatcher in der Methode run() auf die Ergebnisse der OpalMC-Anwendung wartet, diese anschließend verarbeitet (z. B. komprimieren der Checkpointdateien) und zum Schluss an den Requester bzw. den Ressourcen-Manager sendet. Damit die Checkpointdatei zukünftig die für den IMD-Prozess notwendigen Informationen enthält, wurde der in Listing 5.1 und 5.3 dargestellte Java-Code in die Klasse OpalMCResultDispatcher aufgenommen.

Listing 5.1 Erweiterung der Java-Klasse OpalMCResultDispatcher zur Lösung von Herausforderung 1 - Teil 1

```
while(chkptfile != null && chkptfile.isFile() && !abort){
                File readChkptfileMax = chkptfile;
                try{
                      // copy the checkpoint file
                       FileChannel srcChannel = new FileInputStream(chkptfile).getChannel();
                       FileChannel destChannel = new
                            FileOutputStream("chkptfile-copy.txt").getChannel();
                       srcChannel.transferTo(0, srcChannel.size(), destChannel);
                       if (srcChannel != null){
                             srcChannel.close();
                       }
                       if (destChannel != null){
                             destChannel.close();
                       }
                       File readChkptfileData = new File("chkptfile-copy.txt");
```

Die zusätzlichen Zeilen am Anfang der Checkpointdatei (dargestellt in Listing 5.2) sind am Anfang der Datei einzufügen. Da jedoch nicht gleichzeitig Lese- und Schreiboperationen auf der bestehenden Checkpointdatei ausgeführt werden können, muss die Datei zuerst kopiert werden. Hierfür wird der in Listing 5.1 dargestellte Quellcode verwendet.

Listing 5.2 Notwendige Zusatzzeilen der Checkpointdatei für den IMD-Prozess

```
1          #F A 1 1 1 3 0 0
2          #C number type mass x y z
3          #X max 0 0
4          #Y 0 max 0
5          #Z 0 0 max
6          ##
7          #E
```

Der in Listing 5.3 dargestellte Quellcode berechnet zu Beginn den Maximalwert der Checkpointdatei aus den Spalten X, Y und Z der kopierten Checkpointdatei und speichert diesen in der Variable max. Als nächstes werden die für IMD notwendigen Zeilen (vgl. Listing 5.2) in die ursprüngliche Checkpointdatei geschrieben und damit die bisherigen ersten sieben Zeilen der Datei überschrieben. Dabei gibt die erste Zeile (#F A 1 1 1 3 0 0) der danach ausgeführten IMD-Simulation Auskunft darüber, ob es sich bei den Spalten der Checkpointdatei um Skalare oder Vektoren handelt. Die „1 1 1“ stehen hierbei für die drei Skalare Nummer, Typ und Masse. Die „3“ gibt an, dass ein 3er Vektor (x y z) folgt. Durch die darauf folgenden „0 0“ wird angegeben, dass keine weiteren Vektoren folgen. Die zweite Zeile (#C number type mass x y z) enthält die Spaltenbezeichnungen der Checkpointdatei. Die Zeilen drei bis fünf enthalten eine Matrix, welche für IMD die Simulationsbox beschreibt. Damit die Simulationsbox groß genug ist, um alle Daten (Atome) innerhalb der Simulationsbox darstellen zu können, muss der zuvor um eine halbe Gitterkonstante a (entspricht 0,286 Nanometer bzw. 2,86 Angstrom) erhöhte Maximalwert in die Diagonalelemente der Matrix eingesetzt werden. Die Zeilen 6 und 7 ermöglichen den IMD-Prozess zu erkennen, dass alle weiteren Zeilen den Zustand der MC-Simulation (Grad der Ausscheidungsbildung im Atomgitter) des OPAL-Prozesses, bei der Erstellung dieser Checkpointdatei widerspiegeln. Da die ersten sieben Zeilen der ursprünglichen Checkpointdatei bereits überschrieben wurden, ist es notwendig im dritten Schritt die kopierte Checkpointdatei zeilenweise einzulesen und in die ursprüngliche Checkpointdatei zu schreiben. Im Anschluss daran ist die nicht mehr benötigte Kopie der Checkpointdatei zu löschen. Für die Umsetzung wird der in Listing 5.3 dargestellte Quellcode verwendet.

Listing 5.3 Erweiterung der Java-Klasse OpalMCResultDispatcher zur Lösung von Herausforderung 1 - Teil 2

```
      // calculate the maximum of the x,y and z column and adds a/2 to the maximum
      Scanner sc = new Scanner(readChkptfileMax);
      double max = 0;
      while(sc.hasNextLine()){
                  String line = sc.nextLine();
                  String[] details = line.split(" * ");
                  double x = Double.parseDouble(details[3]);
                  double y = Double.parseDouble(details[4]);
                  double z = Double.parseDouble(details[5]);
            max = Math.max(max, Math.max(x, Math.max(y, z)));
      }
      sc.close();
      readChkptfileMax.delete();
      max = max+(0.28553/2);

      // writes the lines that are necessary for the IMD-Workflow at the beginning of the
          checkpoint file
      BufferedWriter writeChkptfile = new BufferedWriter (new FileWriter (chkptfile));
      writeChkptfile.write("#F A 1 1 1 3 0 0");
   writeChkptfile.newLine();
   writeChkptfile.write("#C number type mass x y z");
   writeChkptfile.newLine();
   writeChkptfile.write("#X "+String.valueOf(max)+" 0 0");
   writeChkptfile.newLine();
   writeChkptfile.write("#Y 0 "+String.valueOf(max)+" 0");
   writeChkptfile.newLine();
   writeChkptfile.write("#Z 0 0 "+String.valueOf(max));
   writeChkptfile.newLine();
   writeChkptfile.write("##");
   writeChkptfile.newLine();
   writeChkptfile.write("#E");

   // writes the data of the copied checkpoint file in the checkpoint-file
   FileReader fileReader = new FileReader(readChkptfileData);
   BufferedReader reader = new BufferedReader(fileReader);
   String zeile = null;
   while((zeile=reader.readLine())!=null){
      writeChkptfile.newLine();
      writeChkptfile.write(zeile);
   }
   reader.close();
   readChkptfileData.delete();
   writeChkptfile.close();
}
catch (Exception e) { e.printStackTrace(); }
```

Die Änderung des Quellcodes bedingte anschließend die Durchführung folgender Aktivitäten:

a) Installation von Apache Axis2 Version 1.54. und einfügen der benötigten Axis2-Bibliotheken in den Java Build Path des OpalMCService.

b) Installation von Apache Ant Version 1.94 zur Ausführung der build.xml als Ant Build.

c) Austausch des neu generierten Services OpalMCService.aar im Tomcat-Ordner unter \webapps\axis2\WEB-INF\services.

Herausforderung 2: Zur Umsetzung der Parameter O4 aus der Tabelle 4.3 muss während der Ausführung von OPAL flexibel bestimmt werden, ob der mittlere Radius aus den Daten der Checkpointdatei den Vorgaben entspricht. Nur wenn dies der Fall ist, soll mit dieser Checkpointdatei der IMD-Prozess gestartet und dafür die Checkpointdatei auf die VM kopiert werden. Andernfalls soll die Checkpointdatei gelöscht werden.

Für die flexible Ermittlung des mittleren Radius wurde der Web Service OpalHelperService entwickelt. Die erste Überlegung, den mittleren Radius aus der Checkpointdatei zu berechnen, wurde nach einer genaueren Analyse der bestehenden OPAL Simulationsanwendung und deren Zwischen- und Endergebnissen verworfen. Grund hierfür ist, dass während der Ausführung von OPAL durch OpalMC bereits der mittlere Radius ermittelt und in die Datei *xyzrcprec*.dat.gz* geschrieben wird. Aus diesem Grund wird lediglich ein Web Service benötigt, der aus dieser Datei den mittleren Radius ermitteln und gegen einen übergebenen Wert prüfen kann. Listing 5.5 und Listing 5.6 zeigen den Quellcode für den OpalHelperService Web Service zur Lösung dieser Herausforderung.

Für das Löschen einzelner OPAL-Checkpointdateien wurde der OpalHelperService um die Methode deleteCheckpoint() erweitert. Diese Methode, benötigt wie Listing 5.4 zeigt, den absolten Pfad zur Checkpointdatei. Nach der Prüfung, ob die Checkpointdatei eine Datei ist, wird diese gelöscht.

Damit der Web Service, analog zu den anderen OPAL Web Services, auch ohne Eclipse zur Verfügung steht, wurde aus dem Quellcode ein Axis Archiv erstellt und in den Tomcat-Ordner kopiert. Die zum OpalHelperService gehörige WSDL ist Anhang A.2 zu entnehmen.

Listing 5.4 Entwicklung eines OpalHelperService zur Lösung von Herausforderung 2 - deleteCheckpoint()

```
public void deleteCheckpoint (String filepath){
      File checkpoint = new File(filepath);
      if(checkpoint.isFile()){
            checkpoint.delete();
      }
}
```

Listing 5.5 Entwicklung eines OpalHelperService zur Lösung von Herausforderung 2 - checkRadius() Teil 1

```
package de.ustutt.simtech.opal.ws.helper;
import java.io.File;
import java.util.Scanner;
import de.ustutt.simtech.tools.GZip;

public class OpalHelperWS {
      public boolean checkRadius(double radius, String xyzrcprecgz_path, String name){

            double radius_min = radius;
            double radius_max = radius+0.1;
            double current_radius = 0.0;
            boolean check = false;

            File xyzrcprecgz = new File(xyzrcprecgz_path+name);
            File xyzrcprec = new File(xyzrcprecgz_path+"tempRadius.dat");
            try{
                  //creates a copy of the xyzrcprec-file
                  xyzrcprec.createNewFile();

                  //File decompress
                  GZip.gunzip(xyzrcprecgz, xyzrcprec);

                  //Read until the end of line 2 in the xyzrprec file.
                  Scanner sc = new Scanner(xyzrcprec);
                  sc.nextLine();
                  sc.nextLine();

                  //read the line with the data of the mean radius and split the data
                  String line3 = sc.nextLine();
                  String[] details = line3.split(" * ");
```

Listing 5.6 Entwicklung eines OpalHelperService zur Lösung von Herausforderung 2 - checkRadius() Teil 2

```
                //check if the mean radius is calculated and saved in the line
                if(details[4].equalsIgnoreCase("NaN")){
                      //System.out.println("Current Radius is unknown.");
                }else{
                      current_radius = Double.parseDouble(details[4]);
                      //check the mean radius
                      if(current_radius>=radius_min&&current_radius<radius_max){
                            check=true;
                      }
                }
                sc.close();
                xyzrcprec.delete();
          }
          catch(Exception e){
                e.printStackTrace();
          }
          return check;
     }
}
```

Herausforderung 3: Da ein Teil der eingebundenen Web Services auf dem SimTech-Server der Universität Stuttgart in einer VM laufen, ist ein Datentransfer vom lokalen Rechner notwendig.

Während die OPAL Web Services lokal ausgeführt werden können, sind die IMD Web Services aus Performancegründen auf einem SimTech-Server installiert. Aus diesem Grund müssen die Inputwerte für die IMD Web Services auf der VM des SimTech-Servers verfügbar sein. Hierfür wird das bestehende Rahmenwerk SIMPL der Universität Stuttgart eingesetzt. Nach [Pie12] ist SIMPL die Abkürzung für SimTech - Information Management, Processes and Languages und ermöglicht die Einbindung von externen, heterogenen Datenquellen in Simulationsworkflows. Nach der Installation stehen im ChorDesigner weitere Workflowaktivtäten aus den Bereichen „DataManagementActivity“ und „DataManagementPattern“ zur Verfügung. Um beispielsweise Daten in einen Ordner zu übertragen, welcher noch nicht auf dem Zielrechner vorhanden ist, muss vor der eigentlichen Datenübertragung mit der Aktivität <DataTransfer> der Ordner auf dem Zielrechner mit der Aktivität <IssueCommand> angelegt werden. Über die Aktivität <IssueCommand> ist es möglich, beliebige Kommandozeilenbefehle auszuführen.

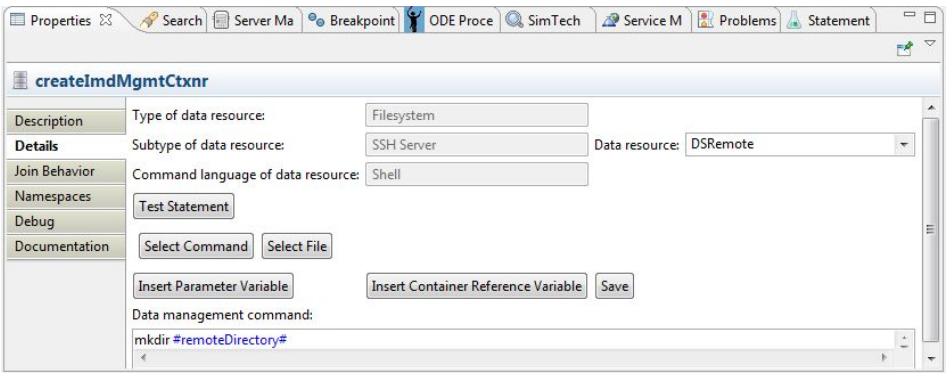


Abbildung 5.5.: Beispielhafte Einstellugen für die SIMPL Aktivität „IssueCommand“

Zu Beginn der Aktivität <IssueCommand> ist über die Datasourcereference (DSRemote) festzulegen (vgl. Abbildung 5.5), auf welcher Betriebssystemart der Befehl auszuführen ist. In dem Feld „Data management command“ ist der auszuführende Befehl (z. B. „mkdir #remoteDirectory#“) einzutragen. In dem hier vorgestellten Beispielbefehl ist #remoteDirectory# eine Variable, damit während der Prozessausführung der Wert problemlos geändert werden kann. Wird #remoteDirectory# beispielsweise auf /home/administrator/Tomcat/imdtransfer/1234 gesetzt, wird der Ordner „1234“ bei der Ausführung der „IssueCommand“-Aktivität als Unterordner von *imdtransfer* erstellt.

Alternativ könnte der in Kapitel 3 vorgestellte Ressourcen-Manager dazu genutzt werden, um Daten an einer zentralen Stelle innerhalb eines choreographieübergreifenden Simulationskontextes abzulegen. Dies wurde innerhalb dieser Arbeit bewusst nicht getan, da der von OPAL genutzte Ressourcen-Manager lokal installiert ist und damit alle Daten auf dem lokalen Rechner abspeichern würde.

Herausforderung 4: Die Servicemethode compileBinary() des IMD CompilerService soll nur ausgeführt werden, wenn noch keine kompilierte IMD-Anwendung zur Verfügung steht.

Um zu ermitteln, ob bereits ein entsprechendes Binary zur Verfügung steht, wurde der Web Service IMDHelperService entwickelt. Die in Listing 5.7 dargestellte Methode checkBinaryExisting() liefert *true* zurück, wenn das Binary bereits existiert, andernfalls *false*. Der IMDHelper Web Service wurde auf dem gleichen Tomcat-Server deployed, auf dem auch die anderen IMD Web Services installiert sind. Die WSDL zum IMDHelperService ist in Anhang A.3 zu finden. Um den IMDHelper Web Service im IMD Workflow aufzurufen, ist es erforderlich, die dazugehörige WSDL in den IMD Workflow-Ordner zu kopieren. Da der IMDHelper Web Service im Tomcat der VM läuft, ist die in der automatisch generierten WSDL enthaltene Adresse (localhost) noch auf die IP-Adresse der VM anzupassen.

Listing 5.7 Entwicklung eines IMDHelperService zur Lösung von Herausforderung 4

```
package de.ustutt.simtech.imd.helper;
import java.io.File;
public class IMDHelper {
   public boolean checkBinaryExisting(String default_binarypath, String binaryname){

      boolean check = false;
      File binary = new File(default_binarypath+binaryname);

      if (binary.exists()){
         check = true;
      }
      return check;
   }
}
```

Es ist nicht möglich, für diese Ermittlung die bereits bestehende Methode listBinarys() des IMD CompilerService zu verwenden, da in der WSDL des Web Services für diese Methode ein leeres <message>-Element hinterlegt ist. Dies bedeutet, dass die Methode ohne Parameter aufgerufen werden kann. Allerdings führt dies bei der Ausführung der IMD-Simulation dazu, dass bei der Zuweisung der Inputvariable für die <invoke>-Aktivität der BPEL-Prozess mit einem (während dieser Arbeit nicht zu lösendem) Fehler abbricht.

Herausforderung 5: Die derzeitige Schnittstellenbeschreibung des IMD Simulation-Service sieht vor, das als Outputnachricht in der Operation executeWithBinary() lediglich die SimulationsID übertragen wird.

Unmittelbar nach Aufruf der Simulationsmethode executeWithBinary() des Simulation-Service liefert der Web Service eine Antwortnachricht. Diese Nachricht enthält allerdings nur die SimulationsID, unter der die Simulationsergebnisse der gestarteten IMD-Simulation verfügbar sind. Die Vorgabe für den IMD-Prozess sieht vor, dass drei IMD-Simulationen hintereinander je OPAL-Checkpointdatei ausgeführt werden sollen. Dabei dient jeweils eine Datei der vorangegangen Simulation als Startdatei für den nächsten IMD-Simulationsaufruf. Da selbst bei einfachen Testdatensätze die Ausführung der einzelnen IMD-Simulationen zwischen Minuten und mehreren Stunden andauert, ist der Startzeitpunkt der zweiten und dritten IMD-Simulation nicht beliebig wählbar. Die Analyse der Simulationsergebnisse der executeWithBinary() Methode ergab, dass nach erfolgreicher Beendigung der IMD-Simulation in der out.log-Datei der Simulation die Statusmeldung „finished at“ enthalten ist. Fehler hingegen, welche zu einer Beendigung der IMD-Simulation führten, konnten der error.log-Datei im Simulationsverzeichnis entnommen werden.

Der in Herausforderung 4 entwickelte IMDHelperService wurde um zwei weitere Methoden ergänzt. Listing 5.8 zeigt hierbei sowohl die Methode checkSimulationEnd() als auch checkSimulationError(). Beide Methoden liefern true zurück, wenn der entsprechende Zeichensatz gefunden wird. Dadurch ist es möglich mit der Methode checkSimulationEnd() herauszufinden, ob die IMD-Simulatation erfolgreich abgeschlossen bzw. über die Methode checkSimulationError(), ob die IMD-Simulation abgebrochen wurde.

Listing 5.8 IMDHelper-Servicemethode checkSimulationEnd() und checkSimulationError()

```
public boolean checkSimulationEnd (String simulation_logfile){
      boolean check = false;
      String simulationEnd = "finished at";
      File logfile = new File(simulation_logfile);
      try{
            if(logfile.isFile()){
                  FileReader fileReader = new FileReader(logfile);
                  BufferedReader reader = new BufferedReader(fileReader);
                  String zeile = "";
                  while(check=false && (zeile=reader.readLine())!=null){
                        check = zeile.contains(simulationEnd);
                  }
                  fileReader.close();
                  reader.close();
            }
      }
      catch(Exception e){ e.printStackTrace(); }
      return check;
}
public boolean checkSimulationEnd (String simulation_errorfile){
      boolean check = false;
      String simulationError = "Error";
      File logfile = new File(simulation_errorfile);
      try{
            if(logfile.isFile()){
                  FileReader fileReader = new FileReader(logfile);
                  BufferedReader reader = new BufferedReader(fileReader);
                  String zeile = "";
                  while(check=false && (zeile=reader.readLine())!=null){
                        check = zeile.contains(simulationError);
                  }
                  fileReader.close();
                  reader.close();
            }
      }
      catch(Exception e){e.printStackTrace();}
      return check;
}
```

Herausforderung 6: Wegen der Implementierung der IMD Web Services und deren Deployment auf der VM ist eine Zuordnung der dort entstehenden IMD-Simulationsergebnisse zu den einzelnen OPAL-Checkpointdateien auf dem lokalen Rechner notwendig.

Um diese Herausforderung zu lösen, wurde die Entscheidung getroffen, dass die Antwortnachricht der IMD-Prozessinstanz an den OPAL-Prozess neben der Checkpointnummer auch die SimulationsIDs enthalten soll. Im OPAL-Prozess können durch den Aufruf einer neu entwickelten OpalHelper-Servicemethode writeIMDSimulationSummary() die Inhalte der IMD-Prozessinstanzen in die Datei *Imdsummary.txt* geschrieben werden. Diese Datei wird im Ordner der aktuellen OPAL-Prozessausführung (mgmtCtx) abgespeichert.

Listing 5.9 OpalHelper-Servicemethode writeIMDSimulationSummary()

```
public void writeIMDSimulationSummary(String path, String chkptNr, double nptdefmax, String
    Sim1Id, String Sim2Id, String Sim3Id){
        try{
            File imdsummary = new File(path);
            BufferedWriter writer = new BufferedWriter(new FileWriter(imdsummary));
            if(!imdsummary.exists()){
                imdsummary.createNewFile();
                writer.write("# OpalSnapshotNumber MaxValueNptdef ImdSimulation1id
                    ImdSimulation2id ImdSimulation3id");
                writer.newLine();
            }
            writer.write(chkptNr);writer.write(" ");
            writer.write(String.valueOf(nptdefmax));writer.write(" ");
            writer.write(Sim1Id);writer.write(" ");
            writer.write(Sim1Id);writer.write(" ");
            writer.write(Sim2Id);writer.write(" ");
            writer.write(Sim3Id);
            writer.newLine();
            writer.close();
        }
        catch(Exception e){e.printStackTrace(); }
    }
```

Wie aus Listing 5.9 hervorgeht, wird zu Beginn geprüft, ob es bereits eine Datei zu der aktuellen OPAL-Prozessausführung gibt. Existiert die Datei noch nicht, wird diese erstellt und die Kommentarzeile mit den Überschriften der darauf folgenden Werte in die Datei geschrieben. Anschließend werden die Daten aus der IMD-Prozessinstanz (chkptNr, nptdefmax, SimulationsIDs 1-3) in die nächste Zeile der Datei geschrieben.

5.4. Ergebnis der modellierten Festkörpersimulation

Abbildung 5.6 zeigt die Umsetzung der gekoppelten Festkörpersimulation mithilfe einer Workflow-Choreographie:

Abbildung 5.6.: Choreographie der gekoppelten Festkörpersimulation zur Analyse der thermischen Alterung von Festkörpern und deren Auswirkung auf das Werkstoffverhalten

Im Abbildung 5.6 ist der OPAL-Prozess durch ein Participant-Element dargestellt. Damit mehrere Instanzen des IMD-Prozesses parallel ausgeführt werden können, wurde für die Darstellung den IMD-Prozess ein Participant-Set-Element ausgewählt. Der OPAL-Unterprozess OpalSnapProc ist im Modell als weiterer eigenständiger Teilnehmer (Participant) der Choreographie modelliert. Der Grund hierfür ist, dass dieser Prozess für eine erfolgreiche Ausführung, analog zum IMD-Prozess, bereits vor dem Start der gekoppelte Festkörpersimulation auf dem Server installiert sein muss. Neben den reinen Kommunikationsaktivitäten sind im Choreographiemodell auch BPEL Steueraktivitäten (z. B. While-Aktivität) aufgenommen, wenn diese in einer konkreten Verbindung zu einer Kommunikationsaktivität stehen. Damit aus dem Choreographiemodell anschließend BPEL-Prozesse generiert werden können, ist für das Modell ein Grounding anzulegen. In diesem sind pro MessageLink (dargestellt im Model durch einen Pfeil) die in Abbildung 5.7 ersichtlichen Informationen (Namespace, PortType, Präfix und Operation des aufzurufenden Prozesses) anzugeben.

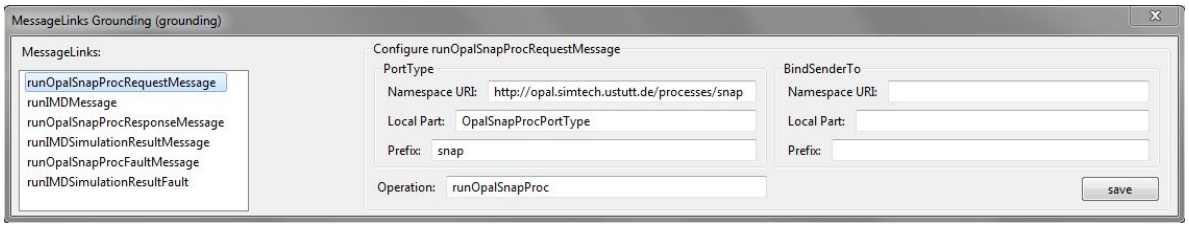


Abbildung 5.7.: Festlegung der Einstellungen für die einzelnen MessageLinks der Choreographie innerhalb der Choreographie Grounding-Definition

6. Implementierung der entwickelten Festkörpersimulation

Entsprechend der Vorgehensweise aus Abbildung 5.2 erfolgt nach der Modellierung die automatische Transformation zu BPEL und ggf. auch zu BPEL4Chor. Die automatische Transformation nach BPEL erstellte je Teilnehmer eine Prozessdatei (*.bpel) und eine dazugehörige Schnittstellenbeschreibung (*.wsdl). Beide Dateien umfassen jeweils nur die in der Choreographie modellierten Aktivitäten und die damit zusammenhängenden Informationen. Aus Abbildung 6.1 kann beispielhaft entnommen werden, wie Prozesse nach der automatische Transformation aussehen.

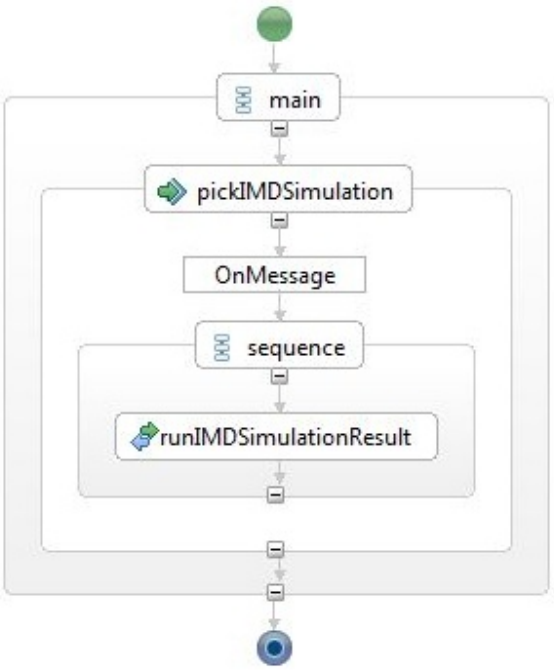


Abbildung 6.1.: Beispielhafte Darstellung wie der IMD-Prozess nach der automatischen Transformation aussehen kann

6.1. Verfeinerung des IMD-Prozesses

Für die Verfeinerung des IMD-Prozesses zu einem ausführbaren IMD-Simulationsworkflow mussten folgende Schritte vorgenommen werden:

1. Für die Kommunikation zwischen dem IMD-Prozess und den anderen Choreographieteilnehmer wurde eine entsprechende XML-Schemadatei (vgl. Listing 6.1) angelegt.

Listing 6.1 Auszug aus der XML-Schemadatei des IMD-Teilnehmers

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:rmtypes="http://resinf.simtech.ustutt.de/ws/resmgr/types"
     xmlns:tns="http://imd.simtech.ustutt.de/ws/types" attributeFormDefault="unqualified"
     elementFormDefault="qualified" targetNamespace="http://imd.simtech.ustutt.de/ws/types">
   <element name="runIMDRequest">
      <complexType>
         <sequence>
            <element name="ctxID" type="string"/>
            <element name="opalChkptName" type="string" />
            <element name="binaryGlok" type="string" />
            <element name="binaryAxial" type="string" />
            <element name="binarypath" type="string"/>
            <element name="callbackEPR" type="string"/>
            <element name="transferPath" type="string"/>
         </sequence>
      </complexType>
   </element>
   <element name="runIMDSimulationResponse">
      <complexType>
         <sequence>
            <element name="chkptNr" type="string"/>
            <element name="imdStatus" type="string"/>
            <element name="imdSimulationIdAufheizen" type="string"/>
            <element name="imdSimulationIdHalten" type="string"/>
            <element name="imdSimulationIdZugversuch" type="string"/>
            <element name="maxNptdeformPzz" type="double"/>
         </sequence>
      </complexType>
   </element>
</schema>
```

2. Einfügen aller Schnittstellenbeschreibungen (*.wsdl) und dazugehörige XML-Schemadateien (*.xsd) von Web Services, welche aus IMD aufgerufen werden, in den Teilnehmerordner von IMD. Hierbei mussten die IP-Adressen der IMD Web Services auf die IP-Adresse der VM angepasst werden.
3. Erstellung eines Deployment Descriptors (DD) in Form einer deploy.xml-Datei (vgl. Listing 6.2). Der DD enthält wichtige Grundinformationen über den Prozess und ist Voraussetzung für das Deployment auf der Apache ODE (Orchestration Director Engine). Für jeden Partnerlink, der im Prozess zum Aufruf von anderen Prozessen oder Web Services genutzt wird, muss ein <invoke>-Element inkl. dem Verweis auf den Service- und Portname aus der dazugehörigen WSDL enthalten sein.
4. Damit der IMD-Prozess von anderen Prozessen aufgerufen werden kann, ist in der WSDL des IMD-Prozesses ein Partnerlink anzulegen. Dieser muss auf das im DD anzulegende <provide>-Element verweisen.
5. Bei der Verfeinerung des IMD-Prozesses um die eigentliche Prozesslogik mussten für alle Web Services die aus dem IMD-Prozess aufgerufen werden folgende Punkte beachtet werden:
   a) Ergänzung der Namensräume.
   b) Einfügen von <import>-Anweisungen für WSDL-Dateien.
   c) Hinzufügen von Partnerlinks.

Der vollständig verfeinerte IMD-Prozess auf Basis von Anforderung A5 (Anforderungen zum Prozessablauf von IMD) aus der Tabelle 4.2 ist der Abbildung 6.2 zu entnehmen. Um je OPAL-Checkpoint drei aufeinander aufbauende IMD-Simulationen zu starten, ist in der <receive>-Aktivität des IMD-Prozesses festgehalten, dass je Aufruf eine neue Instanz des IMD-Prozesses erstellt werden soll.

Listing 6.2 Auszug aus der Deploy.xml-Datei des IMD-Teilnehmers

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
     xmlns:imd="http://imd.simtech.ustutt.de/processes/main"
     xmlns:mainp="http://opal.simtech.ustutt.de/processes/main"
     xmlns:ics="http://imd.ws.uni-stuttgart.de/ImdCompilerService/"
     xmlns:ies="http://imd.ws.uni-stuttgart.de/ImdExecuterService/"
     xmlns:ipas="http://imd.ws.uni-stuttgart.de/ImdParameterService/"
     xmlns:ipos="http://imd.ws.uni-stuttgart.de/ImdPotentialsService/"
     xmlns:ih="http://helper.imd.simtech.ustutt.de">

      <process name="imd:IMD">
            <metaData> </metaData>
            <active>true</active>
            <retired>false</retired>
            <process-events generate="all" />

            <provide partnerLink="ImdLink">
              <service name="imd:ImdService" port="ImdPort"/>
            </provide>

            <invoke partnerLink="Opal-PL">
                  <service name="mainp:OpalMainService" port="OpalMainPort" />
            </invoke>
            <invoke partnerLink="ImdCS-PL">
                  <service name="ics:ImdCompilerService" port="ImdCompilerSOAP" />
            </invoke>
            <invoke partnerLink="ImdES-PL">
                  <service name="ies:ImdExecuterService" port="ImdExecuterSOAP" />
            </invoke>
            <invoke partnerLink="ImdPaS-PL">
                  <service name="ipas:ImdParameterService" port="ImdParameterSOAP" />
            </invoke>
            <invoke partnerLink="ImdPoS-PL">
                  <service name="ipos:ImdPotentialService" port="ImdPotentialSOAP" />
            </invoke>
            <invoke partnerLink="ImdHS-PL">
                  <service name="ih:IMDHelper" port="IMDHelperHttpSoap11Endpoint"/>
            </invoke>
      </process>
</deploy>
```

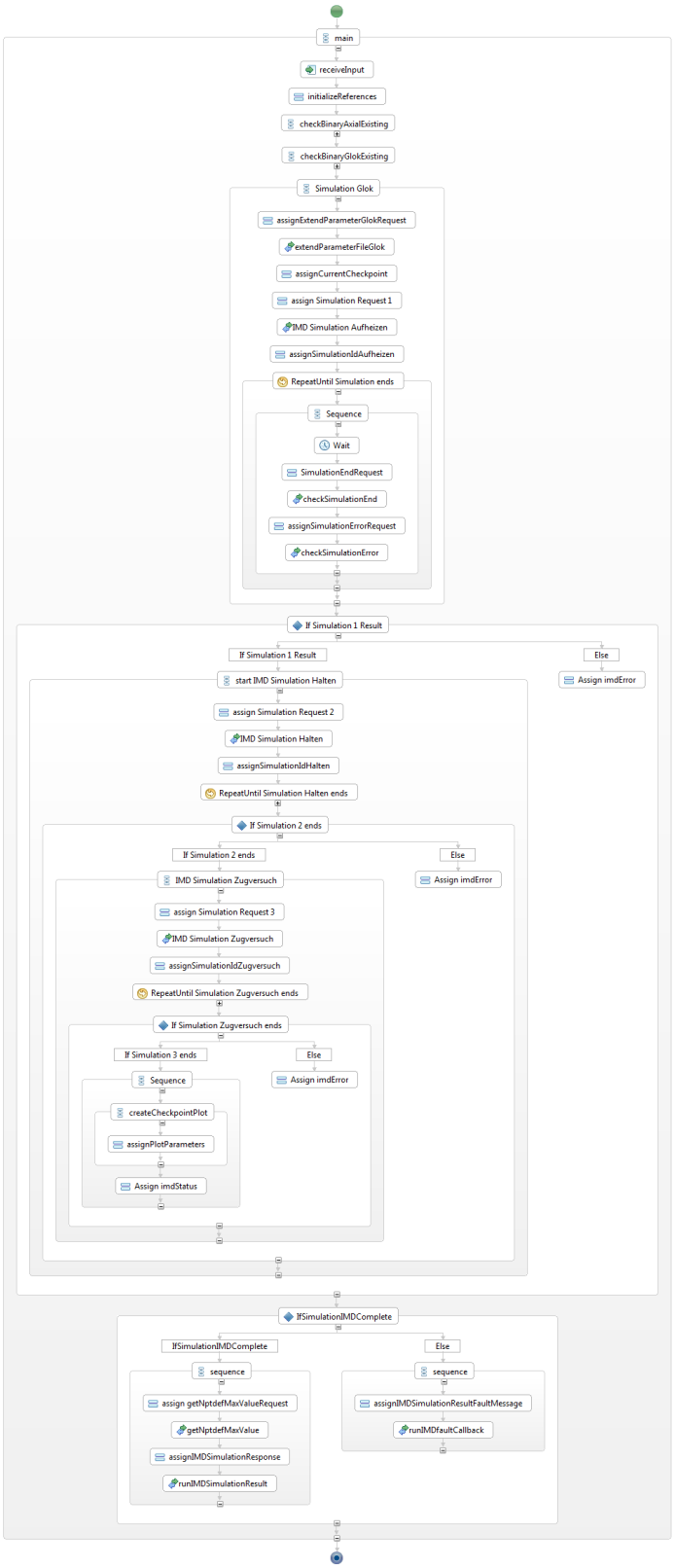


Abbildung 6.2.: Prozessausschnitt 1 des IMD-Teilnehmers nach der Verfeinerung

Zur Umsetzung der in Kapitel 5.3 beschriebenen Herausforderung 4 bezüglich des IMD-Prozessablaufes wird der in Abbildung 6.3 dargestellte Prozessausschnitt genutzt. Dabei wird zuerst mit einer <invoke>-Aktivität der IMDHelperService mit der Methode checkBinaryExisting() aufgerufen. Anschließend wird der Rückgabewert des <invoke>-Aufruf (true|false) geprüft. Lediglich wenn der Wert *false* entspricht, wird das entsprechende Binary durch den Aufruf des IMD Compiler Service kompiliert und damit dem IMD-Prozess zur Verfügung gestellt.

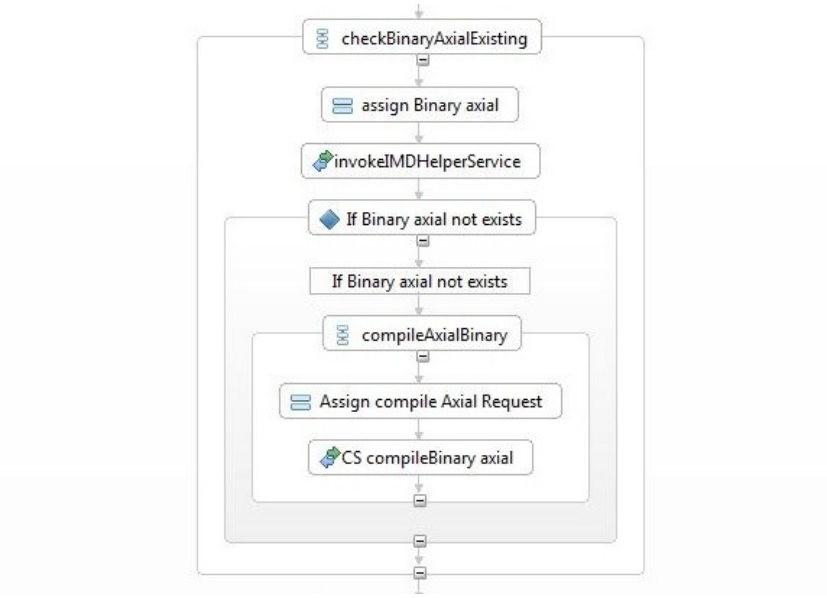


Abbildung 6.3.: Prozessausschnitt 2 des IMD-Teilnehmers: Umsetzung von Herausforderung 4

In Kapitel 5.3 wird als Herausforderung 5 beschrieben, dass der IMD Simulation-Service in seiner Schnittstelle so beschrieben ist, dass dieser im Sinne einer synchronen Kommunikation mit dem BPEL-Prozess als Antwortnachricht direkt die vergebene SimulationsID an den IMD-Prozess zurückschickt. Im Rahmen dieser Masterarbeit konnte nicht genau geklärt werden, wie der IMD Web Service eine asynchrone Kommunikation realisiert, wurden in Kapitel 5.3 zwei Methoden des IMDHelperService vorgestellt. Wie Abbildung 6.4 veranschaulicht, wird im Anschluss an jede <invoke>-Aktivität für eine IMD-Simulation mithilfe einer <repeatUntil>-Aktivität so lange überprüft bis eine der zwei Methoden checkSimulationEnd() oder checkSimulationError() true zurückliefert. Damit in diesem Schleifenkonstrukt nicht unendlich viele <invoke>-Aktivitäten ausgeführt werden, enthält die noch eine <wait>-Aktivität mit der Ausprägung 30 Sekunden.

Für die Umsetzung der Anforderung A6 (Erstellung einer Grafik über das Ergebnis der dritten IMD-Simulation zu einem OPAL-Checkpoint) aus Tabelle 4.2, war es notwendig ein zusätzliches Programm für die Erstellung von Grafiken zu installieren. Die Entscheidung fiel auf das Programm Gnuplot.

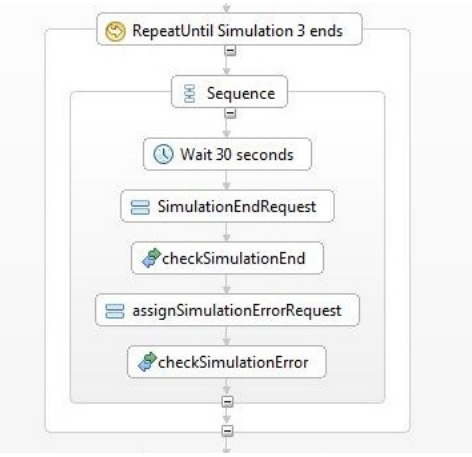


Abbildung 6.4.: Prozessausschnitt 3 des IMD-Teilnehmers: Umsetzung von Anforderung A5

Gründe hierfür waren, dass es sich sowohl auf Windows als auch Linux installieren lässt und dass es bereits in der bestehenden (alten) OPAL-Simulationsanwendung zum Einsatz kam. Realisiert wird die Umsetzung der Anforderung A6, wie Abbildung 6.6 zeigt, über eine <IssueCommand>-Aktivität, welche den eigentlichen Kommandozeilenbefehl enthält. Danach wird die erstellte Grafik mit einer <transferData>-Aktivität auf den lokalen Rechner kopiert. Wie aus Abbildung 6.5 hervorgeht, wird in dem Gnuplot-Befehl zum einen das Ausgabeformat als *png* und der Pfad zur Ausgabedatei mithilfe einer Variable festgelegt. Zum anderen wird die Datenbasis mittels einer Variable für die Erstellung der Grafik festgelegt und über den Befehl *using* die zu verwendenden Spalten der Datenbasis näher eingegrenzt. Bevor Gnuplot die Grafik erstellt, wird der Inhalt von Spalte 15 noch mit -160,2 multipliziert, um Anforderung A6 korrekt umzusetzen. Dadurch werden die Werte der y-Achse in Gigapascal umgerechnet, was eine übliche Einheit für Druck darstellt.

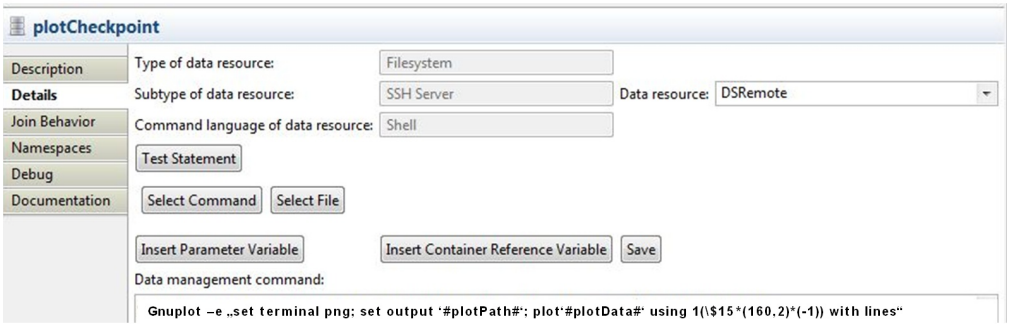


Abbildung 6.5.: Umsetzung von Anforderung A6 - Gnuplot-Befehl

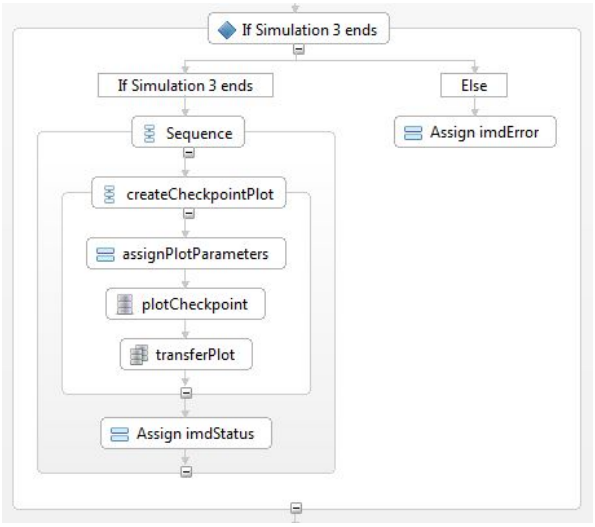


Abbildung 6.6.: Prozessausschnitt 4 des IMD-Teilnehmers: Umsetzung von Anforderung A6

Damit Herausforderung 6 (Zuordnung der IMD Simulationsergebnisse zu dem dazugehörigen OPAL-Snapshot aus einer bestimmten OPAL-Simulation) im BPEL-Prozess gelöst werden kann, ist es notwendig, die entsprechenden Daten an den OPAL-Prozess zu übergeben. Hierfür wurde der IMD-Prozess so angepasst, dass nur nach erfolgreicher Ausführung aller drei IMD-Simulationen für einen OPAL-Checkpoint ein entsprechender OPAL-Aufruf mit den zu übergebenden Werten erfolgt. Andernfalls wird, wie in Abbildung 6.7 veranschaulicht, der OPAL-Prozess mit einer anderen Methode aufgerufen. Dies ermöglicht im OPAL-Prozess eine Unterscheidung nach erfolgreichen und nicht erfolgreichen IMD-Prozessausführungen je OPAL-Checkpointdatei.

Nach Anforderung A7 aus Tabelle 4.2 muss am Ende des OPAL-Prozesses eine Grafik erstellt werden, welche die Maximalwerte der Grafiken zu den einzelnen OPAL-Checkpoints enthält. Hierfür ist es erforderlich, diese Maximalwerte bereits im IMD-Prozess zu ermitteln und im OPAL-Aufruf (runIMDSimulationResult) aus Abbildung 6.7 zu übergeben. Da die in der Grafik eingezeichneten Werte aus der Spalte 15 der Grafik-Datenbasis entnommen sind, muss der Maximalwert auch aus dieser Spalte der Datenbasis bestimmt werden. Hierfür wurde der bestehende IMDHelperService um die Methode getMaxFromNptdef() erweitert. Listing 6.3 zeigt, dass in der Methode zuerst geprüft wird, ob die Datei existiert. Ist dies der Fall, wird die Datei zeilenweise eingelesen. Dabei wird jede Zeile anhand der vorhandenen Leerzeichen gesplittet und die einzelnen Werte in einem StringArray abgespeichert. Unter Verwendung der mathematischen Funktion max der Klasse Math, wird immer der größte Wert aus den übergebenden Variablen in der Variable result gespeichert. Dies ermöglicht am Ende der Methode die Rückgabe des Maximalwertes aus Spalte 15 an den BPEL-Prozess.

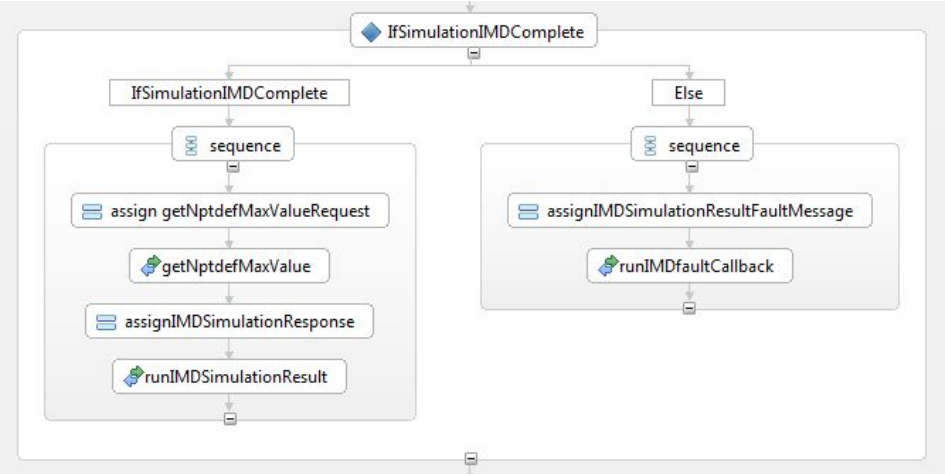


Abbildung 6.7.: Prozessausschnitt 5 des IMD-Teilnehmers: Umsetzung von Herausforderung 6

Listing 6.3 IMDHelper-Servicemethode getMaxFromNptdef

```
public double getMaxFromNptdef (String nptdefpath){
      double result = "0.0";
      File nptdef = new File(nptdefpath);
      try{
            if (nptdef.isFile()){
                  Scanner sc = new Scanner(nptdef);
                  String zeile = sc.nextLine();
                  while(sc.hasNextLine()){
                        zeile = sc.nextLine();
                        String [] details = zeile.split(" ");
                        double actValue = Double.parseDouble(details[14]);
                        result = Math.max(result,actValue);
                  }
            }
      }catch(Exception e){ e.printStackTrace();}
      return result;
}
```

6.2. Verfeinerung des OpalSnapProc-Prozesses

Für die Verfeinerung dieses Prozesses konnten die Inhalte aus dem bisherigen OpalSnapProc-Prozess kopiert werden, da sich im Zusammenspiel zwischen den Choreographieteilnehmern OPAL und OpalSnapProc durch die modellierte gekoppelte Festkörpersimulation nichts geändert hatte. Das Gleiche galt auch für die Schnittstellenbeschreibung des OpalSnapProc-Prozesses. Lediglich die Namensräumen wurden angepasst sowie die von OpalSnapProc benötigten *.wsdl- und *.xsd-Dateien von verwendeten Web Services und Prozessen mussten in den Teilnehmerordner kopiert werden. Vor dem Einfügen der Schnittstellenbeschreibung des OPAL-Teilnehmers musste diese noch an die weiteren Änderungen, welche durch die Choreographie entstanden sind, angepasst werden. Der vollständige OpalSnapProc-Prozess ist Abbildung 6.8 zu entnehmen.

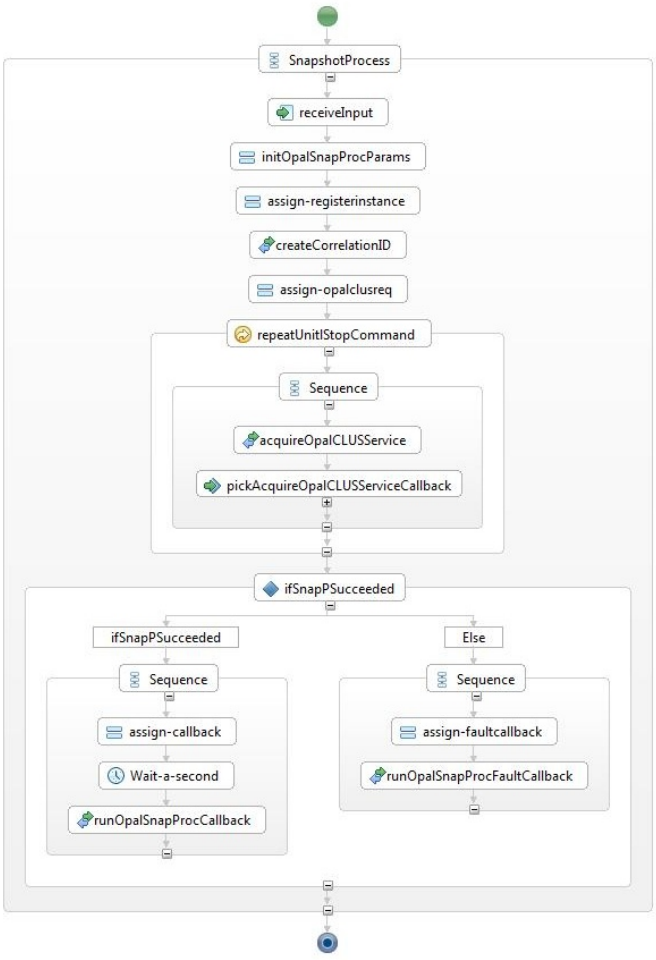


Abbildung 6.8.: OpalSnapProc-Prozessausschnitt nach der Verfeinerung

6.3. Verfeinerung des OPAL-Prozesses

Bei der Verfeinerung des OPAL-Teilnehmers wurde wie folgt vorgegangen. Als erstes wurden die generierten Dateien (Opal.bpel und OpalMainArtifacts.wsdl) mit denen des bereits funktionierenden OPAL-Simulationsworkflows verglichen. Damit später in der Eclipse-Umgebung ein Deployment des OPAL-Prozesses auf dem installierten Tomcat möglich ist, musste zudem die deploy.xml Datei (vgl. Listing 6.4) angelegt bzw. aus dem bestehenden OPAL-Simulationsworkflow kopiert und unter anderem um den IMD-Aufruf erweitert werden.

Listing 6.4 Auszug aus der Deploy.xml-Datei des OPAL-Teilnehmers

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
            xmlns:rm="http://resinf.simtech.ustutt.de/ws/resmgr"
            xmlns:omc="http://opal.simtech.ustutt.de/ws/mc"
            xmlns:om="http://opal.simtech.ustutt.de/ws/opalmgr"
            xmlns:pns="http://opal.simtech.ustutt.de/processes/main"
            xmlns:oclus="http://opal.simtech.ustutt.de/ws/clus"
            xmlns:snapp="http://opal.simtech.ustutt.de/processes/snap"
            xmlns:mainp="http://opal.simtech.ustutt.de/processes/main"
            xmlns:oh="http://helper.ws.opal.simtech.ustutt.de"
            xmlns:imd="http://imd.simtech.ustutt.de/processes/main">

      <process name="pns:OpalMain">
            <metaData> <mdProperty name="version" value="1" /> </metaData>
            <active>true</active>
            <retired>false</retired>
            <process-events generate="all" />

            <provide partnerLink="client">
              <service name="mainp:OpalMainService" port="OpalMainPort" />
            </provide>

            <invoke partnerLink="OpalSnapProcLink">
              <service name="snapp:OpalSnapProcService" port="OpalSnapProcSOAP" />
            </invoke>
            ...
            <invoke partnerLink="ImdLink">
                  <service name="imd:IMD-IMDPT-Service" port="IMD-IMDPT-Port"/>
            </invoke>
      </process>
</deploy>
```

Anschließend wurden in einem zweiten Schritt die neu generierten Dateien (BPEL-Prozess und die dazugehörige WSDL-Datei) um die Logik des bereits bestehenden OPAL-Simulationsworkflows erweitert. Da bei der Generierung aus dem Choreographiemodell der Namespace mit dem Präfix xmlns:wsdl in der WSDL des OPAL-Prozesses gesetzt wurde, konnten die Inhalte (beispielsweise <import> oder <types>) nicht einfach kopiert werden. Die entsprechenden Tags in der WSDL mussten um den Präfix wsdl (<wsdl:import> oder <wsdl:types>) ergänzt werden. Des Weiteren war zu beachten, dass durch das Hinzufügen bzw. Kopieren von <import>-Anweisungen die damit verlinkten Dateien, wie beispielsweise XML-Schema- oder WSDL-Dateien, auch in den OPAL-Teilnehmerordner kopiert wurden.

Im dritten Schritt erfolgte die Ergänzung der Logik für den Aufruf des IMD-Prozesses. Dabei mussten die bereits vorhandenen Kommunikationsaktivitäten angepasst und erweitert werden. Für die Umsetzung der festgestellten Herausforderungen mit Auswirkung auf den OPAL-Prozess mussten zusätzliche Aktivitäten eingefügt werden. Damit Daten vom lokalen Rechner auf die VM des SimTechServers übertragen werden konnten, wurde bereits in Herausforderung 3 der Einsatz von SIMPL und den dazugehörigen Aktivitäten beschrieben. Dementsprechend wurde, wie aus Abbildung 6.9 hervorgeht, der OpalMain-Prozess um jeweils zwei Sequenz-Aktivitäten erweitert.

Abbildung 6.9.: OPAL-Prozessausschnitt 1: Umsetzung Herausforderung 3 - Datentransfer

In diesen Sequenz-Aktivitäten wurde für jede zu übertragende Datei die SIMPL DatamanagementActivity „TransferData“ eingefügt. Damit die Daten übertragen werden können, sind bestimmte Einstellungen an der Aktivität vorzunehmen. Wie in Abbildung 6.10 dargestellt, muss zum einen eine Datasourcereference (DSLocal) für das Betriebssystem, auf dem die zu übertragende Datei liegt, ausgewählt werden. Zum anderen ist eine Datasource (DSRemote) auszuwählen, welche dem Betriebssystem entspricht von dem die Datei zu übertragen ist. Damit die Datasourcereferencen auswählbar sind,

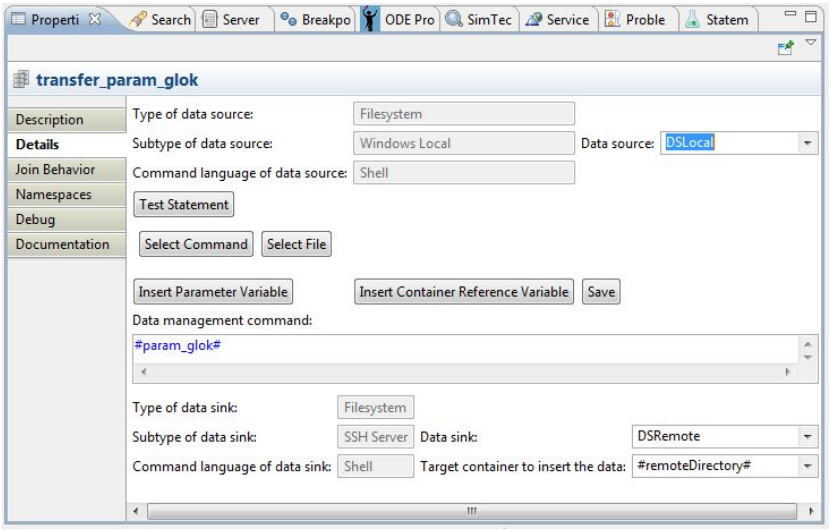


Abbildung 6.10.: Detailsicht der SIMPL DatamanagementActivity <TransferData>

mussten diese zuvor im BPEL-Prozess angelegt und mit einer Datasource verknüpft werden. Dafür wiederum musste zuvor über das SIMPL Resource Management Web Interface (localhost:8080/rmweb/) für beide Systeme im Data Source Management eine Datasource angelegt werden. Da standardmäßig bei SIMPL bereits für Windows eine Datasource namens WindowsLocal angelegt ist, musste lediglich für den Zugriff auf die VM die Datasource (vgl. hierzu Abbildung 6.11) angelegt werden. Damit bei der Übertragung der Dateien während der Prozessausführung der Zielordner für die zu übertragenden Dateien variabel bleibt, wurde in Abbildung 6.10 als TargetContainer lediglich die Variable „remoteDirectory“ ausgewählt. Diese kann dann durch eine <assign>-Aktivität im BPEL-Prozess vor der jeweiligen <transferData>-Aktivität entsprechend verändert werden. Über die Variable „param_glok“ im „Data management command“-Bereich wird die zu übertragende Datei angegeben. In diesem Fall enthält die Variable den absoluten Pfad zu der Datei. An dieser Stelle sei darauf hingewiesen, dass kein Leerzeichen vor der Variable stehen darf, ansonsten funktioniert die Aktivität nicht. Wird die Variable jedoch über den Button „Insert Parameter Variable“ eingefügt, wird standardmäßig vor den Variablennamen ein Leerzeichen gesetzt, welches entsprechend zu entfernen ist.

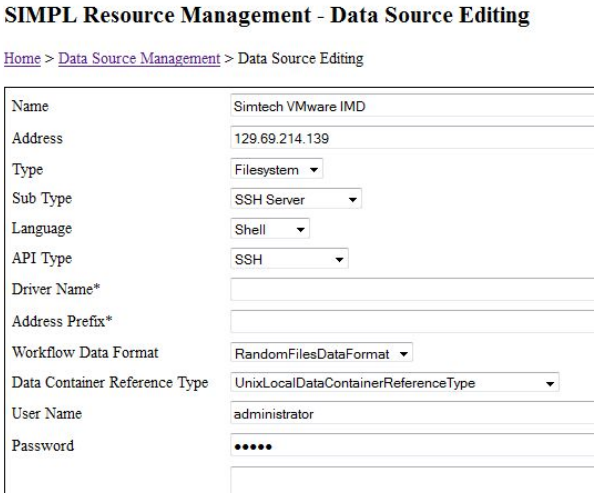


Abbildung 6.11.: Einstellungen der Datasource für den Zugriff auf die VM im SIMPL Resource Management

Damit der IMD-Prozess nur aufgerufen wird, wenn die im Snapshot enthaltenen Ausscheidungen einen entsprechend großen mittleren Radius haben (vgl. Anforderung A4 aus Tabelle 4.2), wurde zur Ermittlung des mittleren Radius in Herausforderung 2 als mögliche Lösung die Einbindung des OpalHelperService beschrieben. Der OpalHelperService setzt hierbei die in der Tabelle 4.3 vorgegebenen Einstellungen für den Parameter radius um. Abbildung 6.12 zeigt einen Ausschnitt aus dem Prozess des Teilnehmers OPAL für diesen Sachverhalt.

Der vom Benutzer eingegebene Wertebereich als Variable *radius* (z. B. 0.5-1.5) wird mittels XPath auf die zwei Variablen *radiusdouble1* (0.5) und *radiusdouble2* (1.5) aufgeteilt. Wie in der Abbildung 6.12 verdeutlicht, wird zu Beginn mit einer Schleife geprüft, ob der Wert der Variable *radiusdouble1* <= dem Wert der Variable *radiusdouble2* ist. Falls ja, wird über eine <invoke>-Aktivität die OpalHelperService-Methode checkRadius() mit der Variable radiusdouble1 und dem xyzrcprec-File des derzeitigen OPAL-Snapshots aufgerufen. Dieser liefert *true* zurück, wenn der mittlere Radius (enthalten im xyzrcprec-File) größer als die Variable *radiusdouble1* und kleiner als *radiusdouble1*+0,1 Nanometer ist. In einer zweiten <if>-Aktivität wird anschließend geprüft, ob das Ergebnis der Methode checkRadius() *true* ist. Ist dies der Fall, wird der entsprechende OPAL-Checkpoint auf die VM kopiert und die IMD-Simulation für diesen OPAL-Snapshot gestartet. Zusätzlich wird die Variable *radiusdouble1* um 0,1 erhöht, damit nicht zwei Checkpointdateien mit dem gleichen mittleren Radius jeweils eine Instanz des IMD-Prozess starten. Durch das Setzten der Variable *IMDResultCounter* ist es zum einen möglich, mitzuzählen wie viele Checkpointdateien an den IMD-Prozess übergeben werden und zum anderen ermöglicht dies, mit einer While-Schleife die Ergebnisse der IMD-Prozess-Instanzen durchzugehen.

Liefert die Methode checkRadius() *false* zurück, wird die entsprechende Checkpointdatei mit der OpalHelperService Methode deleteCheckpoint() gelöscht.

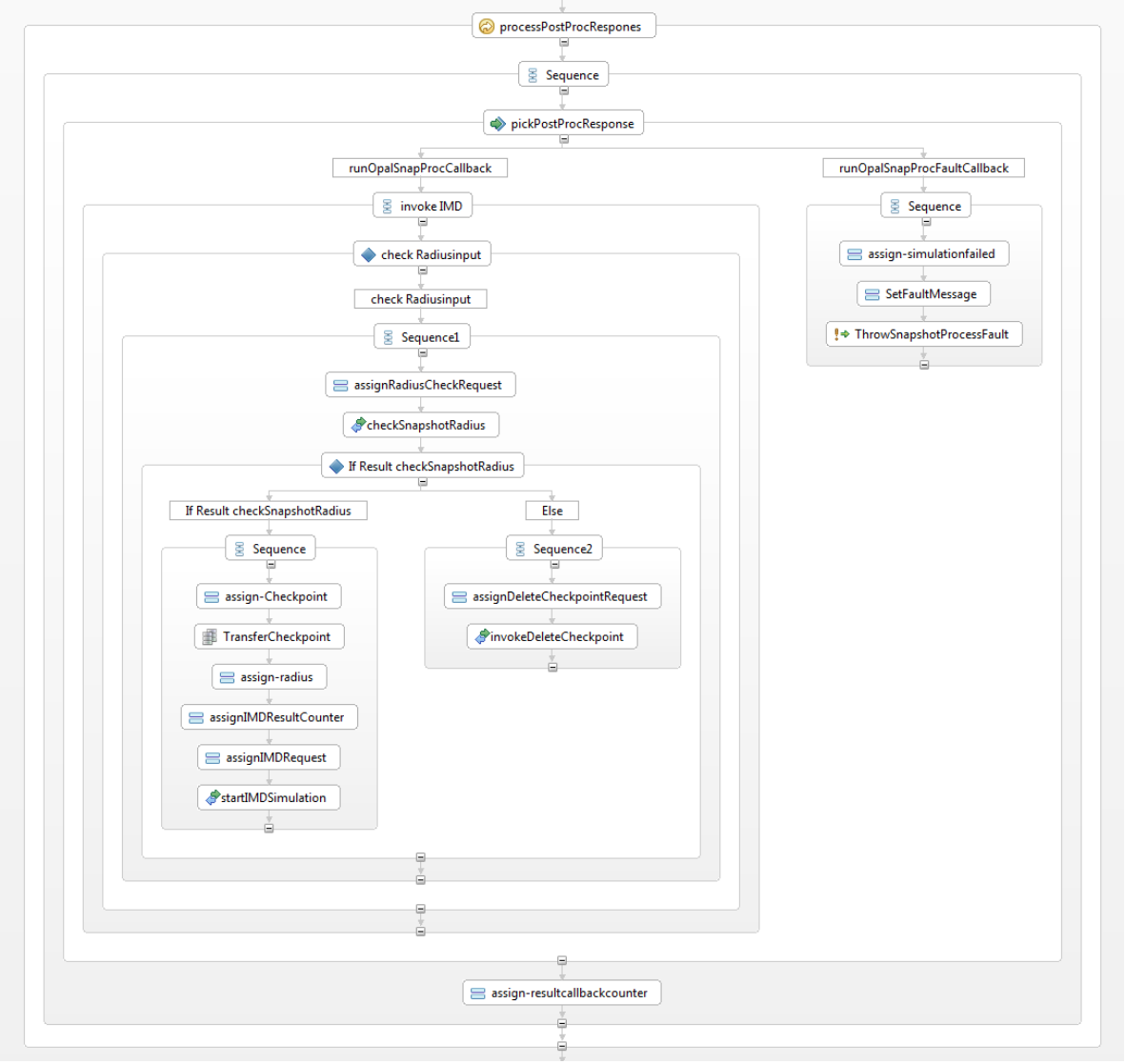


Abbildung 6.12.: OPAL-Prozessausschnitt 2: Umsetzung der Anforderung A4 aus Tabelle 4.2

Abbildung 6.13 veranschaulicht, wie der OPAL-Prozess verfeinert wurde, damit dieser die unterschiedlichen Rückgabemethoden des IMD-Prozesses unterscheiden kann und somit Herausforderung 6 umsetzt. Erhält der OPAL-Prozess eine „runIMDCallback“ Nachricht von einer IMD-Instanz, so kann der OPAL-Prozess durch die Einbindung der OpalHelper-Servicemethode writeIMDSimulationSummary() die Inhalte der Nachricht in eine Datei schreiben. Durch das Abspeichern der Datei im

Ordner des OPAL-Prozesses ist es möglich, die Antwortnachrichten aller erfolgreich abgeschlossenen IMD-Prozessinstanzen darin zu sammeln und einen Überblick zu erhalten, welche Simulationsdaten auf der VM zu welchem OPAL-Checkpoint gehören.

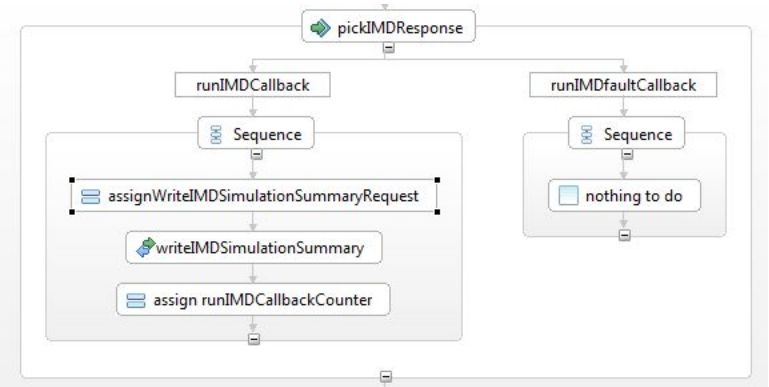


Abbildung 6.13.: OPAL-Prozessausschnitt 3: Umsetzung von Herausforderung 6

Als letzte Aktivität im OPAL-Prozess wurde eine <if>-Aktivität eingefügt. Sobald auch nur eine IMD-Instanz erfolgreich beendet wurde, erstellt diese Aktivität mithilfe einer <IssueCommand>-Aktivität eine Grafik über alle erfolgreich durch IMD analysierten OPAL-Snapshots. Dabei werden die Maximalwerte aus den zuvor erstellten Grafiken je OPAL-Checkpointdatei den Checkpointdateinamen gegenübergestellt.

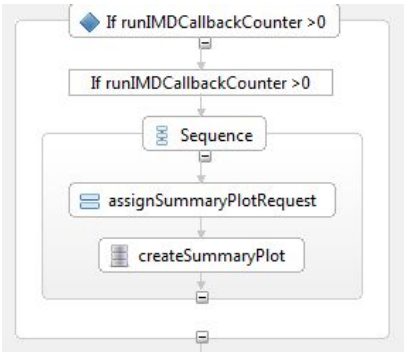


Abbildung 6.14.: OPAL-Prozessausschnitt 4: Umsetzung von Anforderung A7 aus Tabelle 4.2

7. Ergebnisbeurteilung

Für die Ergebnisbeurteilung der modellierten Festkörpersimulation erfolgt in diesem Kapitel eine Auseinandersetzung mit den in Tabelle 4.2 aufgelisteten Anforderungen. Dabei wird auf die einzelnen Anforderungen, die Art und Weise sowie den Umfang der Umsetzung eingegangen. Das Kapitel schließt mit einer Bewertung im Kontext der Forschungsfragen ab.

Anforderung A1 - Modellierungsansatz: *Die gekoppelte Festkörpersimulation soll mit dem Top-Down-Modellierungsansatz entwickelt werden.*
Aus diesem Grund wurden zuerst die Teilnehmer der gekoppelten Festkörpersimulation identifiziert und in der Choreographie als Participants bzw. ParticipantSets modelliert. Anschließend erfolgte die Modellierung der Kommunikation zwischen den Participants bevor die abstrakten Prozesse der einzelnen Teilnehmer generiert, zu ausführbaren Prozessen transformiert und verfeinert wurden. Dabei stellte sich heraus, dass die im Chor Designer enthaltene Basic Executable Completition für die Transformationen von abstrakten zu ausführbaren BPEL-Prozesse nicht zuverlässig funktioniert. Ersichtlich wurde dies lediglich über den abstrakten BPEL-Namespace in der WSDL eines Choreographieteilnehmers und nicht über eine Fehlermeldung bei der Transformation. Daher mussten entgegen der Erwartung die Schritte der Basic Executable Completition für den Prozessteilnehmer OPAL von Hand durchgeführt werden.

Anforderung A2 - Eingabemaske: *Benötigte Parameter sollen über eine gemeinsame Eingabemaske zu Beginn der gekoppelten Festkörpersimulation abgefragt werden.*
Dies wurde dahingehend realisiert, dass für alle Informationen Variablen in den OPAL-Prozess aufgenommen wurden. Die für IMD notwendigen Informationen werden beim Aufruf der IMD-Prozessinstanz mit übergeben. Die aktuelle Implementierung für die grafische Darstellung von Prozessvariablen unterstützt leider nicht alle Datentypen. Für eine Nutzung des Chor Designers durch eine breitere Anwenderschicht könnten gerade die noch nicht umgesetzten einfachen Datentypen wie z. B. double in der Implementierung ergänzt werden. Für die hier erstellte Festkörpersimulation war dies nicht notwendig. Des Weiteren wurde festgestellt, dass in bestimmten Fällen nach dem Einfügen von neuen Variablen Speicher- bzw. Darstellungsprobleme auftreten. Dies äußert sich dadurch, dass Defaultwerte von Variablen nicht mehr korrekt zugeordnet sind. Wurden vor dem Start einer Simula-

tion Variablen hinzugefügt oder gelöscht, sollten die Zuordnung der Variablen in der Eingabemaske kontrolliert werden.

Anforderung A3 - Änderungen: *Änderungen an der Infrastruktur (Bpel Designer / Chor Designer) sowie der bestehenden Simulationsanwendungen sind wenn notwendig vorzunehmen.*
Änderungen die im Rahmen der Arbeit möglich waren (z. B. Erweiterung des OpalMC Web Service zum Anpassen der OPAL-Checkpointdateien an die Anforderungen des IMD-Prozesses) wurden vorgenommen. Des Weiteren wurden neue Funktionalitäten, die für eine spätere Ausführung der gekoppelten Festkörpersimulation benötigt werden, als Web Services bereitgestellt. Beispielsweise sei hier die Methode writeImdSummary() des OpalHelperService oder die Methode getMaxFromNptdef() des IMDHelperService genannt.

Anforderung A4 - Simulationsaufruf IMD: *Durch die gekoppelte Festkörpersimulation soll der IMD-Prozess direkt aus der OPAL-Simulation aufgerufen werden, wenn eine bestimmte Bedingung erfüllt ist.*
Zur Überprüfung ob die Bedingung auf einen OPAL-Checkpoint zutrifft wurde der neu entwickelte Web Service OpalHelperService um die Methode checkRadius() erweitert. Dieser kann aus einer zum Checkpoint gehörenden Analysedatei (xyzrcprec*.dat.gz) den mittleren Radius auslesen.

Anforderung A5 - Simulationsablauf IMD: *Zu Beginn der IMD-Simulation soll die compileBinary() Methode nur aufgerufen werden, wenn noch keine kompilierte IMD-Anwendung zur Verfügung steht. Anschließend sollen für jede OPAL-Checkpointdatei drei sequentielle IMD-Simulationen (Aufheizen, Halten auf Temperatur und Zugversuch) durchgeführt werden.*
Wie aus der Abbildung 6.2 in Kapitel 6.1 hervorgeht, wurde der in Anforderung A5 grob skizzierte Prozessablauf eingehalten. Für das Feststellen des IMD-Simulationsende (abgeschlossen oder fehlerhaft) wurden für eine erste Implementierung des IMD-Prozesses zwei selbst geschriebene Methoden im IMDHelperService genutzt. In einem weiteren Schritt wäre es denkbar die Umsetzung der executeWithBinary() Methode mit Verwendung einer Endpointreference des IMD-Simulationsservice näher zu analysieren und den IMD-Prozess entsprechend anzupassen.

Anforderung A6 - Grafik je OPAL-Checkpoint: *Im Anschluss an den Zugversuch (dritte IMD-Simulation) durch den IMD-Prozess für eine OPAL Checkpointdatei soll eine Grafik auf Basis der nptdeform.eng Datei erstellt werden.*
Dies wurde innerhalb des IMD-Prozesses mittels einer SIMPL-Aktivität (<IssueCommand>) ein Gnuplot-Befehl ausgeführt. Dabei ist es erforderlich den Gnuplot-Befehl direkt in die IssueCommand einzutragen. In einer Ausbaubaustufe des IMD-Prozesses ist es denkbar, den Gnuplot-Befehl zu erweitern, um weitere Einstellungen an der Grafik vorzunehmen. Die Umsetzung dieser Anforderung innerhalb des IMD-Prozesses setzt eine Installation des Programms Gnuplot voraus.

Anforderung A7 - Grafik: *Im Anschluss an die erstellten Grafiken je OPAL-Checkpoint soll eine Grafik erstellt werden, welche die Maximalwerte der bisher erstellten Grafiken zu den jeweiligen OPAL-Checkpoints aufzeigt.*
Die Umsetzung wurde innerhalb des OPAL-Prozesses analog zur Anforderung A6 mit einem Gnuplot-Befehl in einer <issueCommand> realisiert. Die Implementierung sieht vor, dass die entsprechenden Maximalwerte aus den verschiedenen IMD-Prozessinstanzen an den OPAL-Prozess übergeben werden. Die Werte werden zusammen mit weiteren Informationen aus den einzelnen IMD-Prozessinstanzen in einer Datei abgespeichert. Dadurch kann auf Basis der Datei die Grafik erstellt werden und anhand der Datei eine Zuordnung der IMD-Simulationsergebnisse zu den verschiedenen OPAL-Snapshots bzw. OPAL-Prozessen erfolgen.

Anforderung A8 - Parameter für die Simulationsausführung: *Für die Ausführung der gekoppelten Festkörpersimulation sollen bestimmte Parameter eingehalten werden.*
Anhang A.4 enthält alle Parameter inkl. einer kurzen Beschreibung, welche für die erfolgreiche Ausführung der gekoppelten Festkörpersimulation notwendig sind. Generell gilt, dass für eine erfolgreiche Ausführung der gekoppelten Festkörpersimulation weitere Punkte zu beachten sind:

1. Installation der SimTech Umgebung.
2. Aufsetzen des SimTech BPEL Designers inkl. der Chor Designer Artefakte anhand der SimTech-Anleitung.
3. Beachtung der Installationsanweisung für den SimTech Prototyp.
4. Die Prozesse OpalSnapProc.bpel und IMD.bpel müssen bereits auf dem Tomcat-Server deployed und im Status aktiv sein, bevor ausgehend vom OpalMain.bpel-Prozess die gekoppelte Festkörpersimulation gestartet wird.
5. Im Prozess OpalMain.bpel muss zu den in Anhang A.1 enthaltenen Variablen festgelegt werden, dass diese als Parameter genutzt werden sollen. Der entsprechende Haken und der Defaultwert für diese Variable können über die Properties-View zu den einzelnen Variablen im Tab Initialization gesetzt werden.
6. Die für die Ausführung notwendigen Daten (z. B. das Initialgitter, die Energieoptionen sowie die Parameter- und Potentialdateien für den IMD-Prozess) müssen auf dem lokalen Rechner vorhanden sind.
7. Die IMD Web Services sollten aus Performancegründen nicht auf einem lokalen Rechner laufen.

Zu Beginn wurde durch die Auseinandersetzung mit den allgemeinen Grundlagen die Basis für die Modellierung einer Workflow-Choreographie geschaffen. Durch das Eingehen auf die verschiedenen Bereiche der Workflow-Technologie sowie die für die Realisierung oft eingesetzte Web Service

Technologie konnte der aktueller Forschungsstand im Bereich Scientific- und Simulationworkflows dargestellt werden.

Die benötigte Kopplung der einzelnen Simulationen kann, wie in Kapitel 6 in den einzelnen Prozessen ersichtlich wird, durch die Verwendung der BPEL-Aktivitäten <invoke> und <receive> erreicht werden. Im Modell wird die Kopplung durch die Festlegung von MessageLinks zwischen den Teilnehmern einer Choreographie dargestellt.

Wie bereits in Kapitel 4 beschrieben, kann ein gemeinsamer Simulationskontext über die Verwendung eines Ressourcen-Managers durch alle beteiligten Prozesse verwendet werden. Dies ist vor allem sinnvoll, wenn der Ressourcen-Manager nicht lokal installiert ist und alle Daten auf einem zentralen, dafür ausgelegten Speichermedium abgespeichert werden.

Zusammenfassend lässt sich die Forschungsfrage daher wie folgt beantworten: Eine gekoppelte Festkörpersimulation kann mit dem Chor Designer modelliert und in ausführbare Prozesse transformiert werden.

8. Fazit und Ausblick

Die wesentliche Aufgabe dieser Masterarbeit bestand darin, ein Choreographiemodell für eine gekoppelten Festkörpersimulation nach dem Top-Down-Ansatz zu entwickeln. Dabei sollte bei der Prozessverfeinerung die vorgegebenen Anforderungen umgesetzt und für den OPAL-Prozess auf eine bereits bestehende Implementierung des OPAL-Simulationsworkflows aufgesetzt werden.

Die Modellierung der Choreographie mit dem Chor Designer hat ergeben, dass auf Basis der daraus generierten Prozesse eine Implementierung der gekoppelten Festkörpersimulation möglich ist. Hierfür wurde die bestehende OPAL-Simulationsworkflow-Logik in den neu generierten Workflow eingefügt. In Form von BPEL-Aktivitäten wurde weitere Prozesslogik für die Kopplung der Simulationen ergänzt. Für die Anpassung des OPAL-Prozesses an die neue Prozesslogik war es notwendig, einen neuen Web Service zu entwickeln. Dieser stellt u. a. Funktionalitäten für das Auslesen des mittleren Radius aus einer Datei sowie das Löschen und Erstellen einer neuen Datei zur Verfügung. Für die Realisierung des IMD-Prozesses musste ein vollständig neuer Workflow auf Basis des generierten Grundgerüstes (aus dem Choreographiemodell) designed werden. Für die Implementierung der spezifischen Prozessfunktionalität wurde ein Web Service entwickelt.

Über die verschiedenen Modellierungsiterationen im Laufe der Masterarbeit hat sich gezeigt, dass der verwendete Chor Designer nicht fehlerfrei arbeitet. Im Bereich des nachträglichen Löschens von MessageLinks und der nicht immer für alle Teilnehmer erfolgreich ausgeführten Basic Executable Completition gibt es einen Bedarf an Funktionalitätsanpassungen. Im Rahmen dieser Arbeit konnten diese Probleme vorläufig durch manuelle Eingriffe gelöst werden. Bei der Implementierung der einzelnen BPEL-Prozesse (Verfeinerung) wurde ersichtlich, dass ausgehend von einer übergreifenden Choreographiemodellierung für sich ausführbare Prozesse erstellt werden. Im Zusammenspiel der einzelnen Prozesse innerhalb einer Apache ODE und der Verwendung von BPEL-Extensions innerhalb der Prozesse besteht allerdings noch Forschungsbedarf.

Aufbauend auf dieser Arbeit könnte die hier vorgestellte erste Implementierung des IMD-Prozesses weiter verfeinern werden. Dabei könnte das Einbinden eines zentralen Resourcen-Managers und die damit verbundene Prozessänderung, auch in Verbindung mi den eingebundenen IMD Web Services, eine zentrale Rolle spielen. Zusätzlich könnte das Verhalten der Eingabemaske geändert werden, sodass bei der Anzeige von vielen Variablen ein scrollen möglich ist.

A. Anhang

A.1. Parameterdatein für die IMD-Simulation

Listing A.1 Parameterdatei param_glok

```
outfiles glok
core_potential_file /home/administrator/Tomcat/imdtransfer/FECU2009.phi_r2
embedding_energy_file /home/administrator/Tomcat/imdtransfer/FECU2009.F_rho
atomic_e_density_file /home/administrator/Tomcat/imdtransfer/FECU2009.rho_r2
ensemble glok
maxsteps 100 #1000
timestep 0.2
checkpt_int 100 #1000
eng_int 10
starttemp 0.05
relax_rate 0.1
relax_mode axial
ntypes 3
box_from_header 1
```

Listing A.2 Parameterdatei param_npt

```
coordname glok.00001.chkpt
outfiles npt
core_potential_file /home/administrator/Tomcat/imdtransfer/FECU2009.phi_r2
embedding_energy_file /home/administrator/Tomcat/imdtransfer/FECU2009.F_rho
atomic_e_density_file /home/administrator/Tomcat/imdtransfer/FECU2009.rho_r2
ensemble npt_axial
maxsteps 200 #2000
timestep 0.2
checkpt_int 200 #2000
eng_int 5
starttemp 0.0008520
endtemp 0.0258520
# 0.0086173543 0.0258520 0.05170412 0.077556181 0.10340824 0.1292603
pressure_start 0
Pressure_end 0
relax_dirs 1 1 1
indef_interval 1
lindef_x 0 0 0
lindef_y 0 0 0
lindef_z 0 0 0
inv_tau_eta 0.66
inv_tau_xi 0.15
ntypes 3
box_from_header 1
#cna_int 500
#cna_rcut 3.0
#cna_crist 1 2 3
```

Listing A.3 Parameterdatei param_nptdef

```
coordname npt.00001.chkpt
outfiles nptdef
core_potential_file /home/administrator/Tomcat/imdtransfer/FECU2009.phi_r2
embedding_energy_file /home/administrator/Tomcat/imdtransfer/FECU2009.F_rho
atomic_e_density_file /home/administrator/Tomcat/imdtransfer/FECU2009.rho_r2
ensemble npt_axial
maxsteps 1000 #140000
timestep 0.2
checkpt_int 100 #14000
eng_int 5
starttemp 0.0258520
endtemp 0.0258520
# 0.0086173543 0.0258520 0.05170412 0.077556181 0.10340824 0.1292603
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-4 #0 0 1e-6
inv_tau_eta 0.66
inv_tau_xi 0.15
ntypes 3
box_from_header 1
#cna_int 500
#cna_rcut 3.0
#cna_crist 1 2 3
```

A.2. OpalHelperService Web Service - WSDL

Listing A.4 WSDL des OpalHelperService Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
                              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                              xmlns:ns1="http://org.apache.axis2/xsd"
                              xmlns:ns="http://helper.ws.opal.simtech.ustutt.de"
                              xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                              xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                               xmlns:xs="http://www.w3.org/2001/XMLSchema"
                               xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                               xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                               targetNamespace="http://helper.ws.opal.simtech.ustutt.de">

   <wsdl:types>
      <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
          targetNamespace="http://helper.ws.opal.simtech.ustutt.de">
         <xs:element name="writeIMDSimulationSummary">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="path" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="chkptNr" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="nptdefmax" type="xs:double"/>
                  <xs:element minOccurs="0" name="Sim1Id" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="Sim2Id" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="Sim3Id" nillable="true" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
                  <xs:element name="deleteCheckpoint">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="filepath" nillable="true" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
                  <xs:element name="checkRadius">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="radius" type="xs:double"/>
```

```
            <xs:element minOccurs="0" name="xyzrcprec_path" nillable="true"
                type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="checkRadiusResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" name="return" type="xs:boolean"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
</wsdl:types>

<wsdl:message name="checkRadiusRequestM">
   <wsdl:part name="parameters" element="ns:checkRadius"/>
</wsdl:message>
<wsdl:message name="checkRadiusResponseM">
   <wsdl:part name="parameters" element="ns:checkRadiusResponse"/>
</wsdl:message>
<wsdl:message name="deleteCheckpointRequest">
   <wsdl:part name="parameters" element="ns:deleteCheckpoint"/>
</wsdl:message>
<wsdl:message name="writeIMDSimulationSummaryRequest">
   <wsdl:part name="parameters" element="ns:writeIMDSimulationSummary"/>
</wsdl:message>

<wsdl:portType name="OpalHelperWSPortType">
   <wsdl:operation name="checkRadius">
      <wsdl:input message="ns:checkRadiusRequestM" wsaw:Action="urn:checkRadius"/>
      <wsdl:output message="ns:checkRadiusResponseM" wsaw:Action="urn:checkRadiusResponse"/>
   </wsdl:operation>
  <wsdl:operation name="deleteCheckpoint">
      <wsdl:input message="ns:deleteCheckpointRequest" wsaw:Action="urn:deleteCheckpoint"/>
   </wsdl:operation>
   <wsdl:operation name="writeIMDSimulationSummary">
      <wsdl:input message="ns:writeIMDSimulationSummaryRequest"
          wsaw:Action="urn:writeIMDSimulationSummary"/>
   </wsdl:operation>
</wsdl:portType>
```

```
<wsdl:binding name="OpalHelperWSSoap11Binding" type="ns:OpalHelperWSPortType">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
   <wsdl:operation name="checkRadius">
      <soap:operation soapAction="urn:checkRadius" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
    <wsdl:operation name="writeIMDSimulationSummary">
      <soap:operation soapAction="urn:writeIMDSimulationSummary" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
   </wsdl:operation>
    <wsdl:operation name="deleteCheckpoint">
      <soap:operation soapAction="urn:deleteCheckpoint" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
   </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="OpalHelperWSSoap12Binding" type="ns:OpalHelperWSPortType">
   <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
   <wsdl:operation name="checkRadius">
      <soap12:operation soapAction="urn:checkRadius" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
```

```
    <wsdl:operation name="writeIMDSimulationSummary">
      <soap12:operation soapAction="urn:writeIMDSimulationSummary" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
   </wsdl:operation>
   <wsdl:operation name="deleteCheckpoint">
      <soap12:operation soapAction="urn:deleteCheckpoint" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
   </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="OpalHelperWSHttpBinding" type="ns:OpalHelperWSPortType">
   <http:binding verb="POST"/>
   <wsdl:operation name="checkRadius">
      <http:operation location="OpalHelperWS/checkRadius"/>
      <wsdl:input>
         <mime:content type="text/xml" part="checkRadius"/>
      </wsdl:input>
      <wsdl:output>
         <mime:content type="text/xml" part="checkRadius"/>
      </wsdl:output>
   </wsdl:operation>
    <wsdl:operation name="writeIMDSimulationSummary">
      <http:operation location="OpalHelperWS/writeIMDSimulationSummary"/>
      <wsdl:input>
         <mime:content type="text/xml" part="writeIMDSimulationSummary"/>
      </wsdl:input>
   </wsdl:operation>
    <wsdl:operation name="deleteCheckpoint">
      <http:operation location="OpalHelperWS/deleteCheckpoint"/>
      <wsdl:input>
         <mime:content type="text/xml" part="deleteCheckpoint"/>
      </wsdl:input>
   </wsdl:operation>
</wsdl:binding>
```

```
   <wsdl:service name="OpalHelperWS">
      <wsdl:port name="OpalHelperWSHttpSoap11Endpoint" binding="ns:OpalHelperWSSoap11Binding">
         <soap:address
              location="http://localhost:8080/axis2/services/OpalHelperWS.OpalHelperWSHttpSoap11Endpoint/"/>
      </wsdl:port>
      <wsdl:port name="OpalHelperWSHttpSoap12Endpoint" binding="ns:OpalHelperWSSoap12Binding">
         <soap12:address
              location="http://localhost:8080/axis2/services/OpalHelperWS.OpalHelperWSHttpSoap12Endpoint/"/>
      </wsdl:port>
      <wsdl:port name="OpalHelperWSHttpEndpoint" binding="ns:OpalHelperWSHttpBinding">
         <http:address
              location="http://localhost:8080/axis2/services/OpalHelperWS.OpalHelperWSHttpEndpoint/"/>
      </wsdl:port>
   </wsdl:service>

     <plnk:partnerLinkType name="OpalHLT">
            <plnk:role name="OpalHProvider" portType="ns:OpalHelperWSPortType"/>
      </plnk:partnerLinkType>
</wsdl:definitions>
```

A.3. IMDHelperService Web Service - WSDL

Listing A.5 WSDL des IMDHelperService Web Service

```
<?xml version="1.0" encoding="UTF-8"?><wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:ns="http://helper.imd.simtech.ustutt.de"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
targetNamespace="http://helper.imd.simtech.ustutt.de">

   <wsdl:types>
      <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
           targetNamespace="http://helper.imd.simtech.ustutt.de">
          <xs:element name="getMaxFromNptdef">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="nptdefpath" nillable="true" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="getMaxFromNptdefResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return" type="xs:double"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="checkBinaryExisting">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="default_binarypath" nillable="true"
                       type="xs:string"/>
                  <xs:element minOccurs="0" name="binaryname" nillable="true" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
```

```
<xs:element name="checkBinaryExistingResponse">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" name="return" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="checkSimulationEnd">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" name="simulation_logfile" nillable="true"
              type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="checkSimulationEndResponse">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" name="return" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="extendParameterGlokFile">
   <xs:complexType>
         <xs:sequence>
                     <xs:element minOccurs="0" name="transferdirectory" nillable="true"
                          type="xs:string" />
                     <xs:element minOccurs="0" name="chkptNr" nillable="true"
                          type="xs:string" />
         </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="extendParameterGlokResponse">
   <xs:complexType>
         <xs:sequence>
                     <xs:element minOccurs="0" name="return" nillable="true"
                          type="xs:boolean" />
         </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="checkSimulationError">
   <xs:complexType>
      <xs:sequence>
```

```
<wsdl:message name="checkBinaryExistingRequest">
   <wsdl:part name="parameters" element="ns:checkBinaryExisting"/>
</wsdl:message>
<wsdl:message name="checkBinaryExistingResponse">
   <wsdl:part name="parameters" element="ns:checkBinaryExistingResponse"/>
</wsdl:message>

<wsdl:portType name="IMDHelperPortType">
   <wsdl:operation name="extendParameterGlokFile">
      <wsdl:input message="ns:extendParameterGlokFileRequest"
           wsaw:Action="urn:extendParameterGlokFile"/>
      <wsdl:output message="ns:extendParameterGlokFileResponse"
           wsaw:Action="urn:extendParameterGlokFileResponse"/>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationError">
      <wsdl:input message="ns:checkSimulationErrorRequest"
           wsaw:Action="urn:checkSimulationError"/>
      <wsdl:output message="ns:checkSimulationErrorResponse"
           wsaw:Action="urn:checkSimulationErrorResponse"/>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationEnd">
      <wsdl:input message="ns:checkSimulationEndRequest" wsaw:Action="urn:checkSimulationEnd"/>
      <wsdl:output message="ns:checkSimulationEndResponse"
           wsaw:Action="urn:checkSimulationEndResponse"/>
   </wsdl:operation>
    <wsdl:operation name="checkBinaryExisting">
      <wsdl:input message="ns:checkBinaryExistingRequest"
           wsaw:Action="urn:checkBinaryExisting"/>
      <wsdl:output message="ns:checkBinaryExistingResponse"
           wsaw:Action="urn:checkBinaryExistingResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getMaxFromNptdef">
      <wsdl:input message="ns:getMaxFromNptdefRequest" wsaw:Action="urn:getMaxFromNptdef"/>
      <wsdl:output message="ns:getMaxFromNptdefResponse"
           wsaw:Action="urn:getMaxFromNptdefResponse"/>
   </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="IMDHelperSoap11Binding" type="ns:IMDHelperPortType">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="getMaxFromNptdef">
      <soap:operation soapAction="urn:getMaxFromNptdef" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
```

```
<wsdl:message name="checkBinaryExistingRequest">
   <wsdl:part name="parameters" element="ns:checkBinaryExisting"/>
</wsdl:message>
<wsdl:message name="checkBinaryExistingResponse">
   <wsdl:part name="parameters" element="ns:checkBinaryExistingResponse"/>
</wsdl:message>

<wsdl:portType name="IMDHelperPortType">
   <wsdl:operation name="extendParameterGlokFile">
      <wsdl:input message="ns:extendParameterGlokFileRequest"
           wsaw:Action="urn:extendParameterGlokFile"/>
      <wsdl:output message="ns:extendParameterGlokFileResponse"
           wsaw:Action="urn:extendParameterGlokFileResponse"/>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationError">
      <wsdl:input message="ns:checkSimulationErrorRequest"
           wsaw:Action="urn:checkSimulationError"/>
      <wsdl:output message="ns:checkSimulationErrorResponse"
           wsaw:Action="urn:checkSimulationErrorResponse"/>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationEnd">
      <wsdl:input message="ns:checkSimulationEndRequest" wsaw:Action="urn:checkSimulationEnd"/>
      <wsdl:output message="ns:checkSimulationEndResponse"
           wsaw:Action="urn:checkSimulationEndResponse"/>
   </wsdl:operation>
    <wsdl:operation name="checkBinaryExisting">
      <wsdl:input message="ns:checkBinaryExistingRequest"
           wsaw:Action="urn:checkBinaryExisting"/>
      <wsdl:output message="ns:checkBinaryExistingResponse"
           wsaw:Action="urn:checkBinaryExistingResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getMaxFromNptdef">
      <wsdl:input message="ns:getMaxFromNptdefRequest" wsaw:Action="urn:getMaxFromNptdef"/>
      <wsdl:output message="ns:getMaxFromNptdefResponse"
           wsaw:Action="urn:getMaxFromNptdefResponse"/>
   </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="IMDHelperSoap11Binding" type="ns:IMDHelperPortType">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="getMaxFromNptdef">
      <soap:operation soapAction="urn:getMaxFromNptdef" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
```

```
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationError">
      <soap:operation soapAction="urn:checkSimulationError" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkBinaryExisting">
      <soap:operation soapAction="urn:checkBinaryExisting" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="extendParameterGlokFile">
      <soap:operation soapAction="urn:extendParameterGlokFile" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationEnd">
      <soap:operation soapAction="urn:checkSimulationEnd" style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

```
<wsdl:binding name="IMDHelperSoap12Binding" type="ns:IMDHelperPortType">
   <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="getMaxFromNptdef">
      <soap12:operation soapAction="urn:getMaxFromNptdef" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationError">
      <soap12:operation soapAction="urn:checkSimulationError" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="extendParameterGlokFile">
      <soap12:operation soapAction="urn:extendParameterGlokFile" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationEnd">
      <soap12:operation soapAction="urn:checkSimulationEnd" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
```

```
    <wsdl:operation name="checkBinaryExisting">
      <soap12:operation soapAction="urn:checkBinaryExisting" style="document"/>
      <wsdl:input>
         <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap12:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="IMDHelperHttpBinding" type="ns:IMDHelperPortType">
   <http:binding verb="POST"/>
  <wsdl:operation name="getMaxFromNptdef">
      <http:operation location="IMDHelper/getMaxFromNptdef"/>
      <wsdl:input>
         <mime:content type="text/xml" part="getMaxFromNptdef"/>
      </wsdl:input>
      <wsdl:output>
         <mime:content type="text/xml" part="getMaxFromNptdef"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="checkSimulationError">
      <http:operation location="IMDHelper/checkSimulationError"/>
      <wsdl:input>
         <mime:content type="text/xml" part="checkSimulationError"/>
      </wsdl:input>
      <wsdl:output>
         <mime:content type="text/xml" part="checkSimulationError"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="extendParameterGlokFile">
      <http:operation location="IMDHelper/extendParameterGlokFile"/>
      <wsdl:input>
         <mime:content type="text/xml" part="extendParameterGlokFile"/>
      </wsdl:input>
      <wsdl:output>
         <mime:content type="text/xml" part="extendParameterGlokFile"/>
      </wsdl:output>
   </wsdl:operation>
```

```
      <wsdl:operation name="checkSimulationEnd">
         <http:operation location="IMDHelper/checkSimulationEnd"/>
         <wsdl:input>
            <mime:content type="text/xml" part="checkSimulationEnd"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="checkSimulationEnd"/>
         </wsdl:output>
      </wsdl:operation>
       <wsdl:operation name="checkBinaryExisting">
         <http:operation location="IMDHelper/checkBinaryExisting"/>
         <wsdl:input>
            <mime:content type="text/xml" part="checkBinaryExisting"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="checkBinaryExisting"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="IMDHelper">
      <wsdl:port name="IMDHelperHttpSoap11Endpoint" binding="ns:IMDHelperSoap11Binding">
         <soap:address location="http://129.69.214.139:8080/IMDHelperService/services/
                                                    IMDHelper.IMDHelperHttpSoap11Endpoint/"/>
      </wsdl:port>
      <wsdl:port name="IMDHelperHttpSoap12Endpoint" binding="ns:IMDHelperSoap12Binding">
         <soap12:address location="http://129.69.214.139:8080/IMDHelperService/services/
                                                            IMDHelper.IMDHelperHttpSoap12Endpoint/"/>
      </wsdl:port>
      <wsdl:port name="IMDHelperHttpEndpoint" binding="ns:IMDHelperHttpBinding">
         <http:address location="http://129.69.214.139:8080/IMDHelperService/services/
                                                    IMDHelper.IMDHelperHttpEndpoint/"/>
      </wsdl:port>
   </wsdl:service>

    <plnk:partnerLinkType name="ImdHSLT" >
           <plnk:role name="ImdHSProvider" portType="ns:IMDHelperPortType" />
 </plnk:partnerLinkType>
</wsdl:definitions>
```

A.4. Parameter für die Ausführung der gekoppelten Festkörpersimulation

Variable	Beschreibung und mögliche Ausprägungswerte
opal.in	Gibt den relativen Pfad zur opal.in Datei an. Diese enthält die Energieoptionen für die OPAL-Simulation: /InputFiles/econf/econf.dat
opalbcc.dat	Enthält den relativen Pfad zum Initialgitter für die OPAL-Simulation. /InputFiles/bcc/ opalbcc_32_01.dat
snapshots	Enthält die Anzahl der zu erstellenden OPAL-Snapshots: 100. Zu Testzwecken 1.
snapshot-frequenz	Gibt an, nach wievielen MC-Schritten ein OPAL-Snapshot erstellt werden soll. 10^{11}. Zu Testzwecken 10.
checkpoint-frequenz	Gibt an, nach wievielen OPAL-Snapshots ein Sicherungspunkt erstellt werden soll. Defaultwert: 1
lx,ly,lz	Gibt die Größe 32 bzw. 64 des Initialgitters an.
mgmtCtx	Frei wählbarer (eindeutiger) Name für den Managementkontext.
Nb,Nc,Nd	Je nach OPAL-Initialgitter zu wählen. Z.B. 653,1,1
radius	Gibt an, ab welchem mittleren Radius eine OPAL-Checkpointdateien an den IMD-Prozess übergeben wird. Wertebereich: [0.5-1.4]
binarypath	Je nach OPAL-Initialgitter zu wählen. /home/administrator/Tomcat/ wtpwebapps/WS/binarys/
core_potential_file	Gibt den absoluten Pfad zu der ersten Potentialdatei .../*.phi_r2 an.
embedding_energy_file	Gibt den absoluten Pfad zu der zweiten Potentialdatei .../*.F_rho an.
atomic_e_density_file	Gibt den absoluten Pfad zu der dritten Potentialdatei .../*.rho_r2 an.
param_glok	Gibt den absoluten Pfad zu der Parameterdatei .../param_glok an.
param_npt	Gibt den absoluten Pfad zu der Parameterdatei .../param_npt an.
param_nptdef	Gibt den absoluten Pfad zu der Parameterdatei .../param_nptdef an.
remote_directory	Gibt den absoluten Pfad an, wohin die Dateien für den IMD-Prozess übertragen werden sollen. In diesem Fall /home/administrator/Tomcat/imdtransfer/
axialBinary	Namen der zu verwendeten IMD-Anwendung z. B. imd_eam_npt_axial_homdef_stress.
glokBinary	Namen der zu verwendeten IMD-Anwendung z.B. imd_eam_glok_homdef_stress.
localDirectory	Gibt den absoluten Pfad an, wo die OPAL-Simulation die einzelnen Simulationen-ergebnisse speichert. Beispielsweise im restorage-Ordner im Tomcat-Verzeichnis.

Tabelle A.1.: Notwendige Parameter in der OpalMain.bpel Datei, für eine erfolgreiche Ausführung der gekoppelten Festkörpersimulation

Literaturverzeichnis

[AAA+07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu. Web Service Business Process Execution Language Version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Zitiert auf Seite 10)

[ABFG04] D. Austin, A. Barbir, C. Ferris, S. Garg. Web Services Architecture Requirements, 2004. URL http://www.w3.org/TR/wsa-reqs/. (Zitiert auf Seite 19)

[Alo04] G. Alonso. *Web services: concepts, architectures and applications*. Springer, Berlin, 2004. (Zitiert auf Seite 22)

[Bur07] H. Burbiel. *SOA & Webservices in der Praxis: [service oriented architecture mit XML, SOAP, .NET, Java & Co.]*. Franzis, Poing, 2007. (Zitiert auf Seite 21)

[BWR09] A. Barker, C. D. Walton, D. Robertson. Choreographing Web Services. In *IEEE Transactions on services Computing*, Band 2, S. 152–166. 2009. (Zitiert auf den Seiten 9 und 35)

[CCMW10] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2010. URL http://www.w3.org/TR/wsdl. (Zitiert auf Seite 23)

[CD99] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0, 1999. URL http://www.w3.org/TR/xpath/. (Zitiert auf Seite 29)

[Deb13] P. Debicki. *Choreographie-basierte Konsolidierung von BPEL Prozessmodellen*. Diplomarbeit, Universität Stuttgart, 2013. (Zitiert auf den Seiten 16, 17 und 19)

[DKLW07] G. Decker, O. Kopp, F. Leymann, M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS 2007*, S. 296–303. IEEE Computer Society, 2007. (Zitiert auf Seite 15)

[Gad10] A. Gadatsch. *Grundkurs Gesch'aftsprozess-Management*, Band 6. Vieweg und Teubner, 2010. (Zitiert auf Seite 1)

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. Conventional Workflow Technology for Scientific Simulation. In X. Yang, L. Wang, W. Jie, Herausgeber, *Guide to e-Science*, Computer Communications and Networks, S. 323–352. Springer London, 2011. (Zitiert auf den Seiten 1 und 8)

[Ham07] U. Hammerschall. *Verteilte Systeme und Anwendungen: Architekturkonzepte, Standards und Middleware-Technologien.* Pearson Education, 2007. (Zitiert auf Seite 26)

[HB04] H. Haas, A. Brown. Web Services Glossary, 2004. URL http://www.w3.org/TR/ws-gloss/. (Zitiert auf Seite 19)

[HH10] O. Heuser, A. Holubek. *Java Web Services in der Praxis: Realisierung einer SOA mit WSIT, Metro und Policies.* dpunkt-Verl., Heidelberg, 1. aufl. Auflage, 2010. (Zitiert auf den Seiten 20 und 26)

[Höf] C. Höfler. *Entwicklung eines Smoothed-Particle-Hydrodynamics-(SPH)-Codes zur numerischen Vorhersage des Primäü.* Dissertation. (Zitiert auf Seite 37)

[Hot10] S. Hotta. Ausfuehrung von Festkoerpersimulationen auf Basis der Workflow Technologie, 2010. (Zitiert auf den Seiten 31, 39 und 40)

[Kil10] F. Kiltz. *Java Webservices.* mitp, Heidelberg, 1. aufl. Auflage, 2010. (Zitiert auf Seite 21)

[KLW10] O. Kopp, F. Leymann, F. Wu. Mapping interconnection choreography models to interaction choreography models. In *ZEUS*, S. 81–88. 2010. (Zitiert auf Seite 10)

[LR00] F. Leymann, D. Roller. *Production workflow: concepts and techniques.* Prentice Hall PTR, Upper Saddle River, NJ, 2000. (Zitiert auf den Seiten 5 und 6)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis.* Spektrum-Akad.-Verl., Heidelberg, 4. aufl. Auflage, 2010. (Zitiert auf den Seiten 9, 19, 22 und 25)

[Nem14] M. Nemet. Kapselung bestehender Simulationsanwendungen mit Hilfe von Web Services, 2014. (Zitiert auf den Seiten 32, 42 und 44)

[PG03] M. P. Papazoglou, D. Georgakopoulos. Service oriented Computing. *Communications of the acm*, 46(10):25–28, 2003. (Zitiert auf Seite 19)

[Pie12] H. A. Pietranek. *Datenmanagementpatterns in multi-skalaren Simulationsworkflows.* Diplomarbeit, Universität Stuttgart, 2012. (Zitiert auf den Seiten 7 und 57)

[RSM11] P. Reimann, H. Schwarz, B. Mitschang. Design, Implementation, and Evaluation of a Tight Integration of Database and Workflow Engines. *Journal of Information and Data Management*, 2(3):353–368, 2011. (Zitiert auf den Seiten v, 1, 6 und 7)

[Sch12] M. Schwarz. Simulation des anisotropen Zugverhaltens im a-Fe/Cu/Ni-System bei verschiedenen thermischen Alterungszustaenden mittels MC- und MD-Simulationen, 2012. (Zitiert auf den Seiten 37, 38 und 39)

[SH05] P. Stahlknecht, U. Hasenkamp. *Einführung in die Wirtschaftsinformatik*. Springer-Lehrbuch. Springer, 11 Auflage, 2005. (Zitiert auf Seite 37)

[SHK12] M. Sonntag, M. Hahn, D. Karastoyanova. Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In N. Lohmann, S. Moser, Herausgeber, *BPM (Demos)*, Band 940 von *CEUR Workshop Proceedings*, S. 45–50. CEUR-WS.org, 2012. (Zitiert auf den Seiten 8 und 39)

[SK13] M. Sonntag, D. Karastoyanova. Model-as-you-go: An Approach for an Advanced Infrastructure for Scientific Workflows. *Journal of Grid Computing*, 11(3):553–583, 2013. (Zitiert auf Seite 8)

[SLFS06] S. St. Laurent, M. Fritzgerald, N. Schüler. *XML kurz und gut.* O'Reilly, 2006. (Zitiert auf Seite 27)

[Son12] O. Sonnauer. Modellierung von Scientific Workflows mit Choreographien, 2012. (Zitiert auf den Seiten v, 33, 34 und 51)

[Sta07] G. Starke, Herausgeber. *SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen.* dpunkt-Verl., Heidelberg, 1 Auflage, 2007. (Zitiert auf den Seiten 19 und 21)

[WAS+13] A. Weiß, V. Andrikopoulos, S. G. Sáez, D. Karastoyanova, K. Vukojevic-Haupt. Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies. Technischer Bericht Informatik 2013/03, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, 2013. (Zitiert auf den Seiten 34, 49 und 50)

[Wes12] M. Weske. *Business process management: concepts, languages, architectures.* Springer, Berlin; Heidelberg [u.a.], 2. ed. Auflage, 2012. (Zitiert auf den Seiten 5, 9 und 10)

[WK13] A. Weiß, D. Karastoyanova. Towards Choreographies for Multi-Scale and Multi-Physics Simulations. Poster SimTech Status Seminar, 2013. (Zitiert auf Seite 47)

[WKMS14] A. Weiß, D. Karastoyanova, D. Molnar, S. Schmauder. Coupling of Existing Simulations using Bottom-up Modeling of Choreographies. In *Workshop on Simulation Technology: Systems for Data Intensive Simulations (SimTech@GI) in Conjunction with INFORMATIK 2014*, S. 1–12. Gesellschaft für Informatik e.V. (GI), 2014. (Zitiert auf Seite 6)

[XML] URL http://www.w3.org/XML/Core/#Publications. (Zitiert auf Seite 27)

Alle URLs wurden zuletzt am 01. 09. 2014 geprüft.

Erklärung

Gemäß § 14 Abs. 5 und Abs. 6 der Prüfungsordnung der Universitäten Hohenheim und Stuttgart für den Masterstudiengang Wirtschaftsinformatik.

Hiermit erkläre ich, dass ich die Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche einzeln kenntlich gemacht.

Die Masterarbeit habe ich noch nicht in einem anderen Studiengang als Prüfungsleistung verwendet.

Des Weiteren erkläre ich, dass mir weder an den Universitäten Hohenheim und Stuttgart noch an einer anderen wissenschaftlichen Hochschule bereits ein Thema zur Bearbeitung als Masterarbeit oder als vergleichbare Arbeit in einem gleichwertigen Studiengang vergeben worden ist.

Ort, Datum, Unterschrift