Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Aktualisierung und Änderungsweitergabe in Workflow-Choreographien

Markus Nemet

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	M.Sc. Wirt.-Inf. Andreas Weiß
Beginn am:	22. November 2016
Beendet am:	24. Mai 2017
CR-Nummer:	D.2.11, G.2.2, H.4.1, I.7.0

Kurzfassung

Das Forschungsfeld *e-Science* beschäftigt sich unter anderem mit Simulationen in der Wissenschaft. Eine Strategie besteht darin, die etablierten Standards, aus der Geschäftswelt, auf die Anforderungen von Wissenschaftlern, für *Scientific-Workflows*, zu übertragen.

Die angebotene Werkzeuge für Wissenschaftler sollten das Modellieren mit der *Trial and Error* Methode unterstützen, da dies eine natürliche Vorgehensweise bei der Erstellung von Experimenten darstellt. Die Experimente werden als Workflow-Choreographien beschrieben.

Diese Arbeit beschäftigt sich damit, wie Aktualisierungen von Workflow-Choreographien an die beteiligten Partner propagiert und gleichzeitig diese Aktualisierungen automatisch in das bestehende Modell des Partners übernommen werden können. Dazu wird ein Model-Integration-Konzept erarbeitet und anschließend in einem Proof of Concept die Funktionalität innerhalb eines wissenschaftlichen Prototyps bereitgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1. Einleitung

In allen Bereichen finden inzwischen computergestützte Arbeiten statt. Der Computer dient nicht nur als Unterstützung oder zur Effizienzsteigerung, sondern auch dazu Abläufe zu dokumentieren, analysieren und zu automatisieren. Eine Variante Arbeitsabläufe mit Hilfe von Computern zu modellieren sind Workflows. Diese Workflows können mit Hife eines Workflow-Management-Systeme (WfMS) ausgeführt und verwaltet werden [LR00].

In der Geschäftswelt haben sich Workflows und Workflow-Management etabliert [BG07], dadurch entwickelten sich eine Reihe von Standards und Konzepten [Wee+05][Whi04][KL08]. Es liegt nahe die bewährten Standards und Konzepte [Aal12] auch für andere Domänen, ausserhalb der Business-Workflows, einzusetzen [SHK12]. So wird versucht die gereiften Standards aufzugreiffen und für wissenschaftliche Workflows, sogennante Scientific-Workflows, zu verwenden [SKD10].

Ein Herausforderung, sowohl in Scientific-Workflows als auch in Business-Workflows, ist es existierende Workflow-Modelle bequem anzupassen. Ein wichtiger Aspekt für Unternehmen ist das reibungslose Modellieren und Ausführen von Prozessen [RWR06b]. Ein Unternehmen, welches schnell seine Prozesse anpassen kann, hat enorme wirtschaftliche Vorteile [RWR06a], da bei der Überarbeitung eines Workflow-Modells oftmals viele Experten und Personenstunden involviert sind [LR+10]. Diese Flexibilität erlaubt es Unternehmen beispielsweise sich auf neue Geschäftspartner mit anderen Vorgehensweisen einzustellen oder auf Einwirkungen von außen, wie Gesetzesänderungen, reagieren zu können.

Eine Zusammenarbeit mit Partnern kann in der Welt der Workflows mit Hilfe einer Workflow-Choreographie modelliert werden. Innerhalb dieser Choreographie ist zum Beispiel der Nachrichtenaustausch zwischen den Teilnehmern spezifiziert. Wird die Choreographie geändert, ist es gegebenenfalls erforderlich die Choreographie-Teilnehmer über die Änderungen zu informieren, damit weiterhin eine Kommunikation und damit eine Zusammenarbeit zwischen den Teilnehmern möglich ist.

Im Bereich des e-Science werden Scientific-Workflows für die Modellierung und Ausführung von Simulationen verwendet [RSM11]. An die Werkzeuge der Scientific-Workflows bestehen jedoch andere Anforderungen im Vergleich zu Business-Workflows [BG07]. Das resultiert nicht nur aus den unterschiedlichen Verwendungsweisen [SKL10]. Daher sind eine Reihe neuer Workflow-Management-Systeme entstanden, welche sich an die Bedürfnisse von Wissenschaftler richten [Lud+06][Tay+07][Hul+06]. Innerhalb dieser Systeme können Simulationen modelliert und ausgeführt werden [KR][SK10].

Die Universität Stuttgart entwickelt einen Prototyp, in dem wissenschaftliche Workflows modellierbar und ausführbar sind. Dafür werden die Standards und Konzepte der Geschäftswelt aufgriffen und zu diesem Zwecke eingesetzt.

In dieser Arbeit werden Aktualisierungen der Workflow-Choreographie betrachtet und wie diese Änderungen an die betroffenen Choreographie-Teilnehmer mitgeteilt werden können. Die Änderungen sollen automatisch in den Workflow der Choreographie-Teilnehmer integriert werden. Dazu wird ein *Model-Integration*-Konzept erarbeitet und anschließend in einem *Proof of Concept* die Funktionalität innerhalb des wissenschaftlichen Prototyps bereitgestellt.

1.1. Motivation

Mit Hilfe von Scientific-Workflow Werkzeugen können Wissenschaftler bei der Modellierung, Durchführung und Dokumentation ihrer Experimente unterstützt werden. Die Workflow-Management-Umgebungen sind dafür auf die Bedürfnisse der Wissenschaftler abgestimmt, damit sich diese voll und ganz auf ihre Kernkompetenzen konzentrieren können.

An der Universität Stuttgart wird ein Prototyp entwickelt, der einfaches Modellieren und Ausführen von Scientific-Workflows ermöglichen soll. Die Modellierung, innerhalb des Prototyps, findet mit Hilfe des Konzepts der Workflow-Choreographie statt.

Ein wichtiger Aspekt des Prototyps ist die flexible und unkomplizierte Modellierung, welche bereitgestellt werden soll. Hierzu kann per *drag and drop* ein Experiment über die grafische Oberfläche erstellt werden. Weiter sollen Simulationen mit Hilfe der *trial and error* Methode modellierbar sein, da typischerweise in der Wissenschaft das gewünschte Ergebnis bekannt, aber der Weg dorthin unerforscht ist.

Durch die *trial and error* Methode entwickelt sich ein Modell schrittweise weiter. Hierbei entstehen mehrere ähnliche Versionen eines Modells. Zwei Modelle zu vergleichen und Passagen mit Unterschieden festzustellen, um diese in ein neues Modell zu übernehmen, ist zeitaufwendig und behindert den Arbeitsfluss (Abbildung 1.1).

Deshalb ist es notwendig, den Wissenschaftlern eine praktikable Lösung bereitzustellen, die das Vergleichen und Verschmelzen zweier Modelle problemlos ermöglicht. Dies beinhaltet Aktualisierungen am Modell zu erkennen und diese gegebenenfalls an den betroffenen Stellen einzuarbeiten. Im besten Fall passiert das automatisch und ohne weiteres Zutun des Wissenschaftlers. So soll eine Verbesserung des Arbeitsablaufs bei der Modellierung entstehen, die Fehlervermeidung fördert und Arbeitszeit einspart.

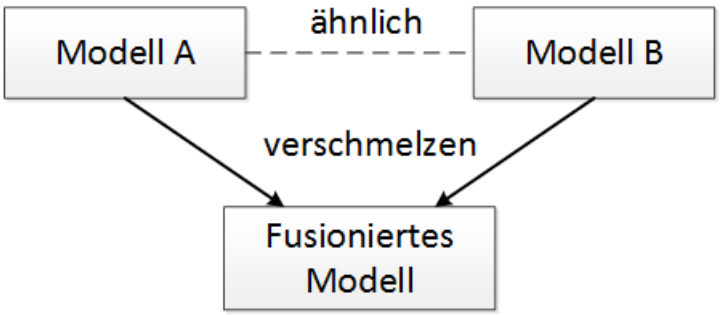


Abbildung 1.1.: Vorgehen beim Fusionieren zweier Modelle

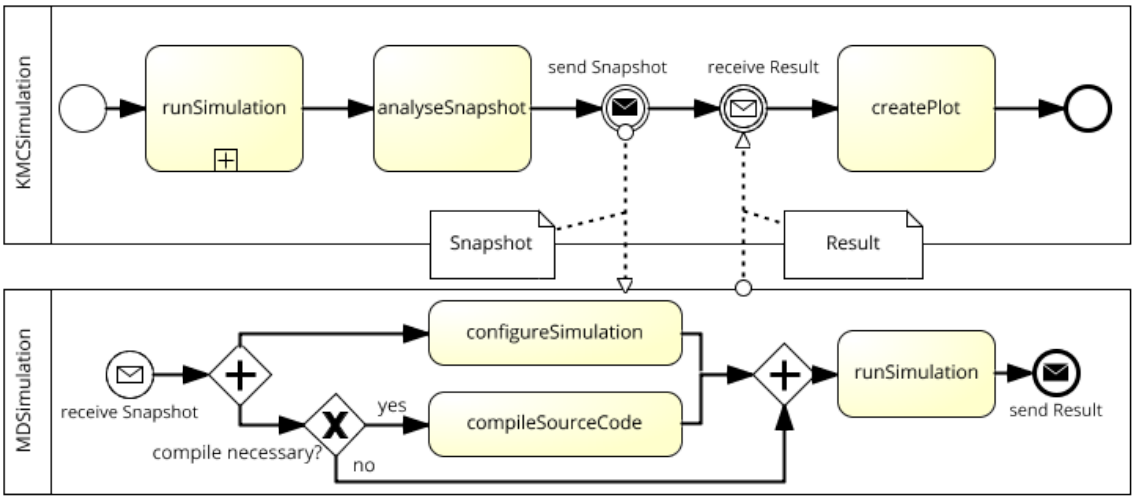
1.2. Szenario

Das aktuelle Modellieren mit dem Prototyp wird im folgenden Szenario beschrieben. Zunächst ist eine initiale Modellierung des Experiments erforderlich, welche meist aus einer textuellen Beschreibung hervorgeht. Daraus entsteht ein Modell, welches gerade so viele Informationen beinhaltet, dass dieses ausgeführt und Ergebnisse betrachtet werden können. Der Wissenschaftler modifiziert dieses Modell dann weiter, bis das gewünschte Ergebnis vorliegt. Dieser Ablauf wird beliebig oft wiederholt.

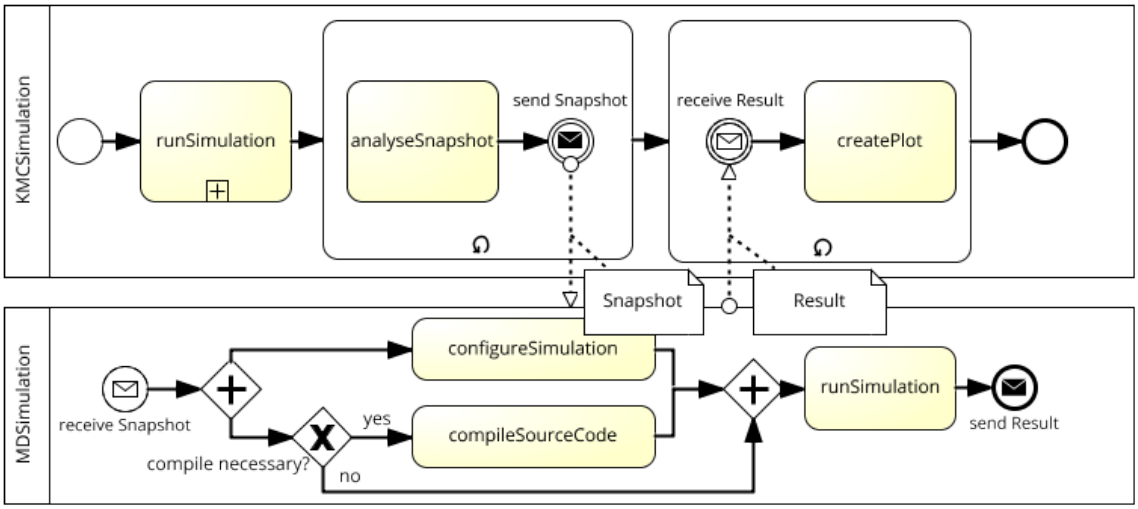
Im Folgenden wird ein Modell betrachtet, welches in vorherigen Arbeiten [WK16][Hin14][WKM] entstanden ist. Das Modell zeigt ein Experiment in Form einer Simulation. Innerhalb der Simulation werden Festkörper physikalischen Vorgängen wie Krafteinwirkung und thermischer Alterung ausgesetzt, um später das Materialverhalten beurteilen zu können. Die Festkörpersimulation besteht aus mehreren Simulationen, die zusammen das Experiment bilden. Es werden für dieses Beispiel zwei Simulationen herausgegriffen, um die Zusammenarbeit zu demonstrieren. Betrachtet wird die Monte-Carlo-Simulation (KMC) und die Molekulardynamik-Simulation (MD). Sie sind Teilnehmer der Choreographie.

Diese zwei Simulationen tauschen während des Experimentes Nachrichten miteinander aus. Ausgehend von der KMC-Simulation wird ein Zwischenstand an die MD-Simulation gesendet. Die MD-Simulation bearbeitet den empfangenen Zwischenstand und schickt das Ergebnis zurück. In Abbildung 1.2a ist der Nachrichtenaustausch innerhalb des Ablaufdiagramms dargestellt. Das Senden einer Nachricht wird durch einen ausgefüllten und das Empfangen durch einen leeren Brief symbolisiert. In diesem ersten Versuch wird ein einzelner Zwischenstand ausgetauscht.

Dem Wissenschaftler genügt dieser einzelne Zwischenstand nicht für eine Beurteilung, er möchte mehr Details. Deshalb entscheidet er im nächsten Versuch mehrere Zwischenstände analysieren zu lassen. Er erweitert das Modell so, dass die KMC-Simulation mehrere Zwischenstände von der MD-Simulation berechnen lässt. Diese Modifikation ist in Abbildung 1.2b



(a) Erster Versuch



(b) Zweiter Versuch

Abbildung 1.2.: Beschreibung einer Festkörpersimulation

dargestellt und zeigt, wie ausgehend von der KMC-Simulation mehrere Nachrichten gesendet und empfangen werden. Dies ist durch ein Wiederholungssymbol dargestellt.

Diese theoretische Betrachtung muss innerhalb des Prototyps umgesetzt werden. Die Durchführung eines Experiments kann mit Hilfe des Prototyps wie folgt umgesetzt werden: Die Modellierung der Simulation entsteht grafisch als Choreographie innerhalb des ChorDesigners. Hierbei wird lediglich ein rudimentäres Modell und die Kommunikationsaktivitäten der Teil-

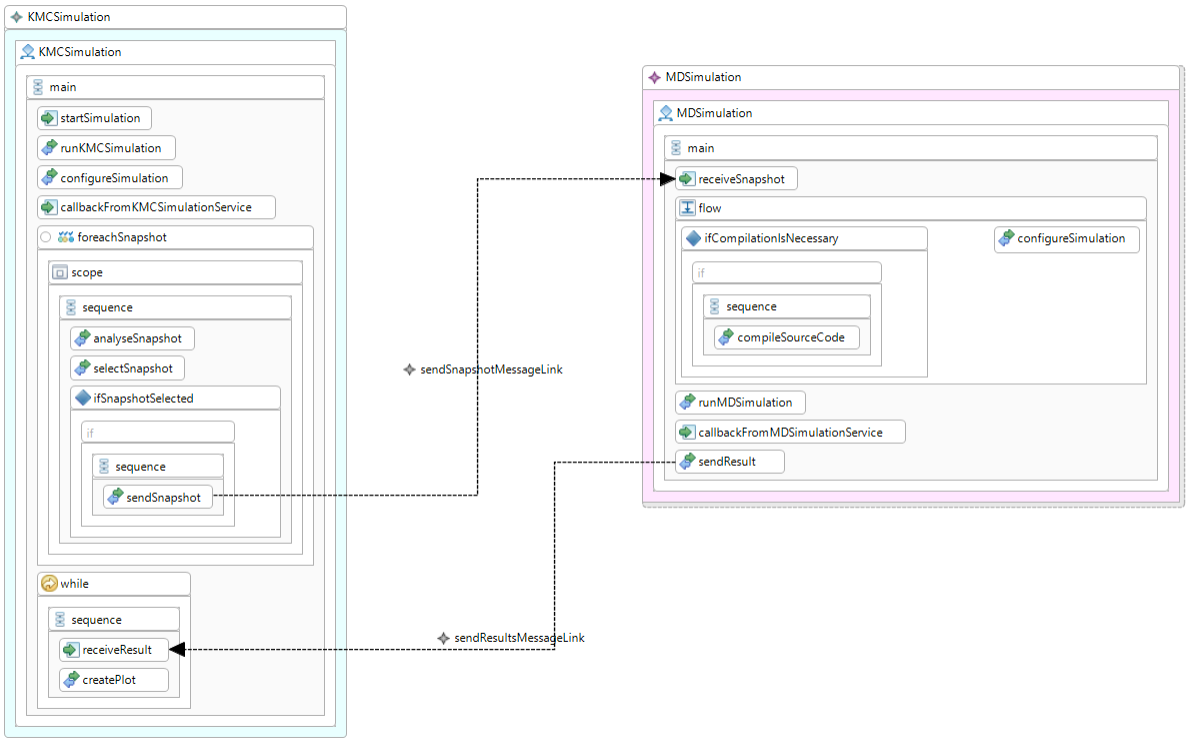


Abbildung 1.3.: Modellierte Choreographie innerhalb des Prototyps

nehmer modelliert. Das entstandene Diagramm wird anschließend so transformiert, dass jeder Teilnehmer einen Workflow in Form einer Orchestrierung erhält. Die Orchestrierung muss dann für jeden Teilnehmer mit Geschäftslogik verfeinert werden. Dadurch wird das Modell ausführbar. Das Modell kann schließlich simuliert und die Ergebnisse vom Wissenschaftler analysiert werden. Der Wissenschaftler trifft Entscheidungen aufgrund der Ergebnisse und modifiziert das Modell gegebenenfalls.

Eine detailliertere Ansicht der modellierten Festkörpersimulation, innerhalb des Prototyps, ist in Abbildung 1.3 zu sehen. Hier sind in der Choreographie zusätzlich, über die Kommunikation hinaus, weitere Aktivitäten und Ablaufstrukturen für jeden Teilnehmer ergänzt worden. Der Nachrichtenaustausch wird über die Pfeile angezeigt. Die KMC-Simulation ist auf der linken Seite zu sehen und die MD-Simulation auf der rechten Seite.

Das gezeigte Modell beschreibt die Interaktion zwischen den Teilnehmern und die Struktur des Experiments. Es ist noch nicht ausführbar und muss zunächst von jedem Teilnehmer individuell verfeinert werden. Dazu erhält jeder Teilnehmer seine Verhaltensbeschreibung aus diesem Choreographie-Modell. Diese Verhaltensbeschreibung wird aus der Choreographie heraus generiert. Die Beschreibung des eigenen Ablaufs wird Orchestrierung genannt. Jeder Teilnehmer fügt weitere Elemente in seine Orchestrierung ein, damit diese ausführbar wird.

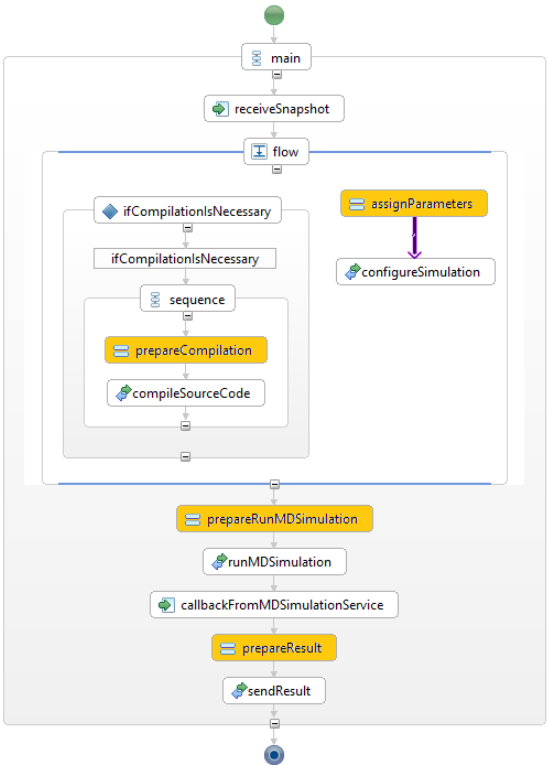


Abbildung 1.4.: Verfeinerte Orchestrierung des MD Teilnehmers

Sind alle Orchestrierungen verfeinert und ausführbar, kann die Choreographie ausgeführt werden.

Repräsentativ sind in Abbildung 1.4 die Verfeinerung der MD-Simulation Orchestrierung durch weitere Aktivitäten zu sehen. In diesem Fall wurden benötigte Vorbereitungsschritte hinzugefügt, beispielsweise die *prepareRunMDSimulation* Aktivität. Diese manuellen Verfeinerungen müssen aktuell bei jedem neuen Versuchsmodell erneut eingearbeitet werden, da sie durch eine erneute Generierung der Orchestrierung überschrieben werden.

Entwickelt sich das Choreographie-Modell weiter und erhält Modifizierungen, ist jedes Mal eine erneute Generierung der Teilnehmer-Orchestrierungen notwendig. Die Modifizierungen können dabei einfache Verschiebungen der Aktivitäten, bis hin zu komplexen Ablaufänderungen umfassen.

Die Teilnehmer erhalten die neue Version der Orchestrierung. Das hat zur Folge, dass die Verfeinerungen der Teilnehmer erneut in das neue Modell hinzugefügt oder angepasst werden müssen, obwohl diese dem alten Modell ähnlich sind. Das Anpassen der Verfeinerungen bedeutet jedes Mal einen zeitlichen Aufwand. Deshalb ist ein automatisches Integrieren des alten verfeinerten Modells in das neue noch unverfeinerte Modell eine große Erleichterung, welche sonst müßig und fehleranfällig mit *copy and paste* erledigt werden müsste.

1.3. Hintergrund

Das Institut für Architektur von Anwendungssystemen (IAAS) ist Teil des Exzellenzcluster „Simulation Technology“ (SimTech), welches im Rahmen der Exzellenzinitiative von Bund und Ländern (DFG) gefördert wird[1]. Eines der Unterprojekte beschäftigt sich mit der Modellierung von Multiskalen- und Multiphysiksimulationen[2]. In diesem Rahmen entsteht unter anderem eine Anwendung, welche Wissenschaftler bei der Modellierung von Simulationen unterstützen soll. Eine Anforderung an diese Anwendung ist, dass die Modellierung der Simulationen mit Hilfe der *trial and error* Methode durchgeführt werden kann, da dies als eine natürliche Vorgehensweise bei der Durchführung von Experimenten angesehen wird.

1.4. Ziel der Arbeit

Aktuell wird von dem Prototyp für jede gespeicherte Änderung eine neue Modell-Revision angelegt oder das bestehende alte Modell überschrieben. Anpassungen am bestehenden alten Modell werden entweder im Falle einer neuen Revision nicht übernommen oder durch Überschreiben verworfen. Es sind weitere manuelle Schritte notwendig, um die Änderungen aus beiden Modellen in ein gemeinsames Modell zu überführen. Das händische Verschmelzen beider Modelle ist ein aufwendiger, fehleranfälliger Vorgang. Zudem behindert es den *trial and error* Arbeitsablauf, da diese Schritte für jeden Versuch erneut vorgenommen werden müssen. Deshalb soll das Verschmelzen beider Modelle automatisiert werden.

Um die *trial and error* Methode besser zu unterstützen, muss die existierende Transformation, aus dem Choreographie-Modell zu den einzelnen Teilnehmer-Workflows, erweitert werden. Die Erweiterung umfasst die Weitergabe und Einarbeitung der Änderungen in bereits bestehende und verfeinerte Workflows der Teilnehmer. Dabei sollen transitive Änderungen beachtet und entsprechend an betroffene Choreographie-Teilnehmer propagiert werden. Dazu soll zunächst ein *Model-Integration*-Konzept für die Integration und Weitergabe der Aktualisierungen erarbeitet werden. Später soll die Funktionalität in einem *Proof of Concept* im Prototyp implementiert und anschließend evaluiert werden.

[1] http://www.iaas.uni-stuttgart.de/forschung/projects/simtech/
[2] http://www.simtech.uni-stuttgart.de/forschung/pn/PN6/index.html

1.5. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 1 – Einleitung: Es wird mit einer Hinführung zum Thema begonnen, welche eine Übersicht über den Themenkomplex bietet. Darauf folgt die Motivation dieser Arbeit, sowie ein Beispielszenario, in der die Ergebnisse dieser Arbeit Anwendung finden können. Des weiteren wird der wissenschaftlichen Rahmen genannt und zuletzt die Ziele der Arbeit aufgezeigt.

Kapitel 2 – Grundlagen Die für diese Arbeit erforderlichen Begriffe, Definitionen und Konzepte werden in diesem Kapitel erläutert. Zusätzlich wird ein Einblick auf die grundlegenden Technologien ermöglicht.

Kapitel 3 – Verwandte Arbeiten Verschiedene Publikationen beschäftigen sich mit dem bearbeiteten Themenkomplex. Hier werden exemplarisch einige relevante Arbeiten vorgestellt. Sie dienen dazu Unterschiede und Gemeinsamkeiten darzustellen, sowie einen weiteren Einblick in das Thema zu gewähren.

Kapitel 4 – Anforderungsanalyse Basierend auf den Grundlagen und verwandten Arbeiten wird das Thema genauer untersucht. Zudem wird die weitere Vorgehensweise erarbeitet.

Kapitel 5 – Konzept Die Anforderungen und Annahmen werden erörtert. Die getroffenen Entscheidungen werden festgehalten und Algorithmen für die Realisierung vorgeschlagen.

Kapitel 6 – Prototyp Der wissenschaftliche Prototyp wird präsentiert. Zudem wird die existierende Architektur analysiert und benötigte Anpassungen für die Umsetzung des Konzeptes vorgestellt.

Kapitel 7 – Implementierung Das vorgestellte Konzept wird im wissenschaftlichen Prototyp implementiert. Hierzu werden die entwickelten Komponenten betrachtet und Details der Implementierung erläutert.

Kapitel 8 – Evaluierung Die Algorithmen werden auf die Anforderungen geprüft. Danach wird eine kurze Laufzeitanalyse durchgeführt.

Kapitel 9 – Zusammenfassung und Ausblick Die Ergebnisse der Arbeit werden zusammengefasst und abschließend beurteilt. Danach wird ein Ausblick auf weiterführende Forschungen in Bezug auf mögliche Verbesserungen und Weiterentwicklungen gegeben.

Hinweis Diese Arbeit ist auf Deutsch verfasst, dennoch werden Fachbegriffe aus dem Englischen übernommen, um ein gemeinsames Vokabular sicherzustellen und weitere Recherchen zu vereinfachen. Oftmals ist auch keine adäquate Übersetzung vorhanden, weshalb auf die englische Bezeichnung zurückgegriffen wird.

2. Grundlagen

In der vorhergehenden Einleitung wurde die Motivation dieser Arbeit beschrieben und mittels einem Beispiel Szenario begründet. Zusätzlich wurden die Ziele dieser Arbeit vorgestellt.

In diesen Kapitel werden Grundlagen, sowie Definitionen, Konzepte und Technologien, welche für diese Arbeit von Relevanz sind, präsentiert.

Begonnen wird in Abschnitt 2.1 mit einer kurzen Einführung zu Workflows und in das Workflow-Management. Darauf folgt in Abschnitt 2.2 die Gegenüberstellung von Workflow Kompositionen. Anschließend wird in Abschnitt 2.3 ein Einblick auf die Workflow Modellierung gegeben. In Abschnitt 2.4 wird auf konkrete Technologien eingegangen und in Abschnitt 2.5 die verwendete Notationen für Workflows vorgestellt. Abschließend wird in Abschnitt 2.6 eine Verarbeitungsmethode für Workflow-Modelle beschrieben.

2.1. Workflow-Management

Aus dem Bedürfnis heraus Arbeitsabläufe computergestützt zu verwalten und zu realisieren entstanden Systeme und Hilfsmittel um Arbeitsabläufe auf dem Computer durchzuführen [Gan+07]. Aus diesem Bedürfnis heraus resultierten Workflows, welche unter anderem spezifiziert, analysiert, modelliert und optimiert werden müssen. Mit diesem Thema beschäftigt sich das Workflow-Management.

Das Feld des Workflow-Managements „[...] umfasst alle Aufgaben, die bei der Modellierung, Spezifikation, Simulation sowie bei der Ausführung und Steuerung der Workflows erfüllt werden müssen“ [LLS10].

Ein Workflow besteht aus drei Dimensionen (Abbildung 2.1). Eine Dimension beschreibt, welche Aktivitäten erledigt werden müssen (Was-Dimension), ein weiterer Aspekt, welche Personen oder Programme diese erledigen (Wer-Dimension), und zuletzt, welche IT-Infrastruktur verwendet wird (Womit-Dimension) [LR00].

Eine Möglichkeit der Repräsentation der Prozesslogik ist in der Form eines Graphen, dieser beschreibt die Abarbeitungsfolge von Aktivitäten. In Abbildung 2.1 ist eine schematische Darstellung eines Prozessgraphen(Was-Dimension) zu sehen.

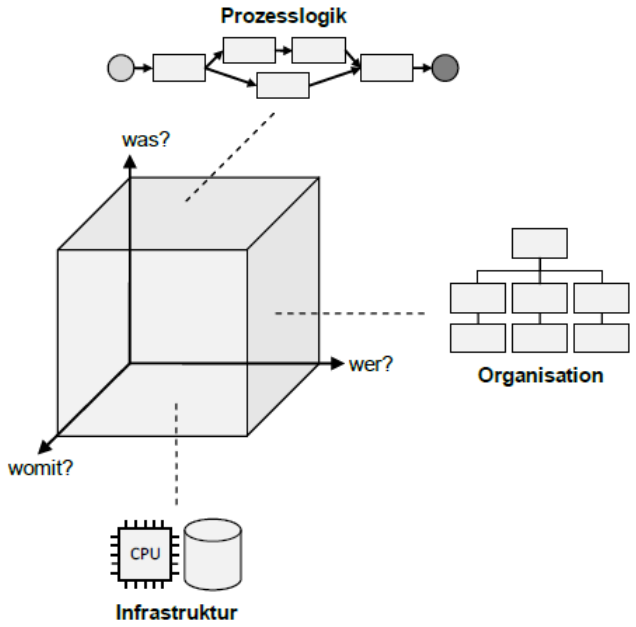


Abbildung 2.1.: Die drei Workflow-Dimensionen nach [LR00]

Business-Workflow und Scientific-Workflow Workflows werden durch das zu erreichende Ziel unterschieden [RSM11]. Für die vorliegende Arbeit sind zwei Workflow Typen relevant.

Zum einen existieren die Business Workflows mit Workflow-Umgebungen, die auf Geschäftsprozesse und deren Anforderungen spezialisiert sind. Hierzu zählt das Automatisieren von Geschäftsprozessen innerhalb eines Unternehmens. In Abbildung 2.2 (a) ist der Lebenszyklus eines Business Workflows zu sehen. Dieser besteht aus mehreren Phasen, welche von unterschiedlichen Personen ausgeführt und mit unterschiedlichen Werkzeugen durchlaufen wird.

Zum anderen gibt es die Scientific-Workflows, welche auf die Bedürfnisse von Wissenschaftlern zugeschnitten sind. In Abbildung 2.2 (b) ist der Lebenszyklus eines Scientific-Workflows abgebildet. Hier werden die Phasen von einem Wissenschaftler selbständig durchlaufen. Dieser verwendet ein Werkzeug das auf seine Anforderungen angepasst ist, um sich ganz auf seine Kernkompetenzen konzentrieren zu können. Als eine einfach zu benutzende Software, zur Unterstützung in allen Phasen, wurde das Mayflower-Framework [SHK12] entwickelt.

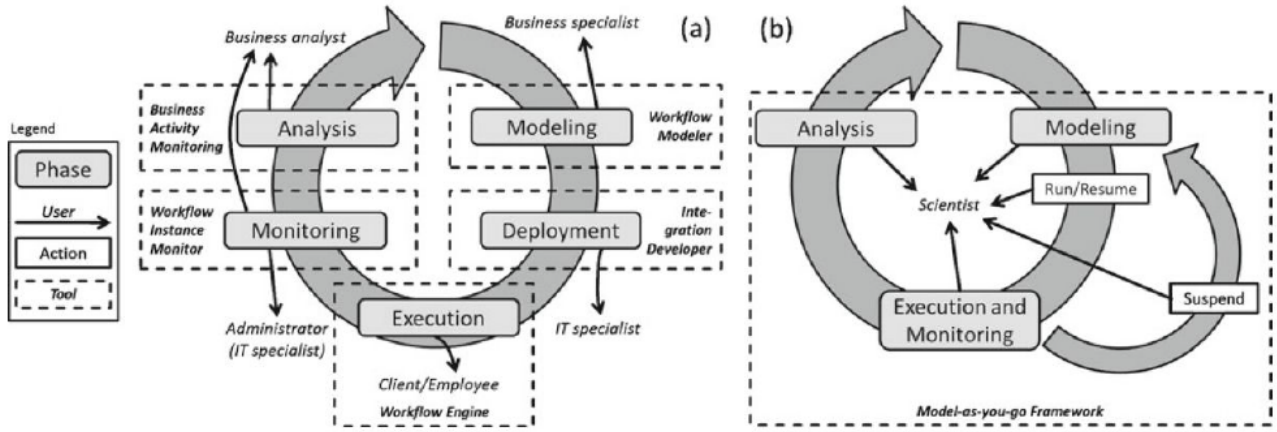


Abbildung 2.2.: Lebenszyklus von a) Business Workflows und b) Scientific Workflows [SHK12][WK14]

2.2. Kompositionen

Es existieren zwei Ansätze für die Komposition von Aktivitäten im Rahmen eines Workflows. Sind die Aktivitäten aus der Sicht eines Teilnehmers beschrieben und hat dieser alleinige Kontrolle über sämtliche Aktivitäten, dann handelt es sich um eine Orchestrierung. Innerhalb der Orchestrierung existiert ein zentraler Koordinator. Zur Veranschaulichung kann ein Orchester herangezogen werden. Der Dirigent, repräsentativ für die zentrale Steuereinheit und Koordination, leitet das Ensemble durch das Musikstück, welches stellvertretend für die Aktivitäten und den Prozess stehen.

Interagieren hingegen mehrere Teilnehmer gleichwertig miteinander, das heißt ohne zentralen Koordinator, wird von einer Choreographie gesprochen. Eine Choreographie umfasst mindestens zwei Teilnehmer. Es ist durchaus möglich, dass ein Teilnehmer in mehreren Choreographien gleichzeitig involviert ist. Um sich eine Choreographie vorzustellen greift die Analogie eines gemeinsamen Tanzes. Alle Teilnehmer wollen ein gemeinsames Ziel erreichen. Um dieses Ziel zu erreichen muss sich auf ein Folge von Interaktionen geeinigt werden.

In Abbildung 2.3 wird der Zusammenhang zwischen Orchestrierung und Choreographie verdeutlicht. Die Orchestrierung legt den Fokus auf das interne Verhalten und wird daher auch lokale Sicht genannt. Hingegen fokussiert sich die Choreographie auf die Kommunikation der Teilnehmer und bietet so eine globale Sicht auf den Prozess. In der globalen Sicht sind die Aktivitäten zur Kommunikation notwendigerweise für den Partner sichtbar. So verbinden die Paare *invoke*, *receive* und *reply*, *receive* die beiden Orchestrierungen miteinander.

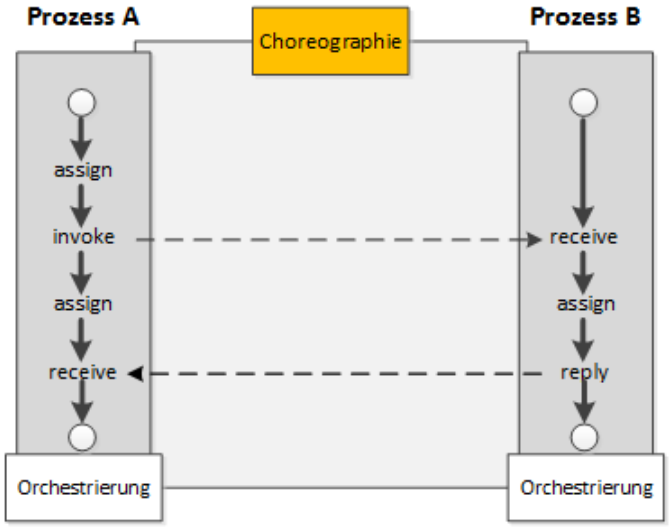


Abbildung 2.3.: Orchestrierung und Choreographie nach [Wee+05]

Eine Choreographie kann auch als Vertrag zwischen den Teilnehmern interpretiert werden, in dem sich die Teilnehmer auf den Kommunikationsablauf geeinigt haben.

2.3. Modellierung

Bei der Modellierung von Workflow Modellen wird typischerweise entweder mit der *Top-down* oder *Bottom-up* Methode vorgegangen. Wird die Top-down Methode verwendet, wird zunächst von oben mit einem allgemeinen Modell begonnen und dieses nach unten immer detaillierter spezifiziert.

Für die Modellierung einer Choreographie nach der Top-down Methode hat dies folgende Bedeutung (Abbildung 2.4): Zunächst wird mit einer Analyse des Problems begonnen. Diese umfasst unter anderem das Herausfinden beteiligter Choreographie-Teilnehmer und die notwendige Kommunikation untereinander. Aus der Analyse entsteht dann ein Choreographie Modell, welches die Teilnehmer und die Konversation, das heißt die Kommunikationsaktivitäten jedes Teilnehmers, enthält. Dieses Modell wird dann automatisch in abstrakte Workflows transformiert. Jeder Teilnehmer erhält einen abstrakten Workflow, welcher nur die eigenen Kommunikationsaktivitäten beinhaltet. Der abstrakte Workflow ist noch nicht ausführbar und muss für jeden Teilnehmer mit den eigentlichen Arbeitsschritten verfeinert werden. Sobald der Workflow vollständig verfeinert wurde besitzt jeder Teilnehmer eine ausführbare Orchestrierung. Wird das Modell hingegen Bottom-up konstruiert, werden die Schritte in die entgegengesetzte Richtung abgearbeitet. Zuerst wird mit der Modellierung der einzelnen Orchestrierungen begonnen. Aus den einzelnen Orchestrierungen wird dann eine Choreographie generiert.

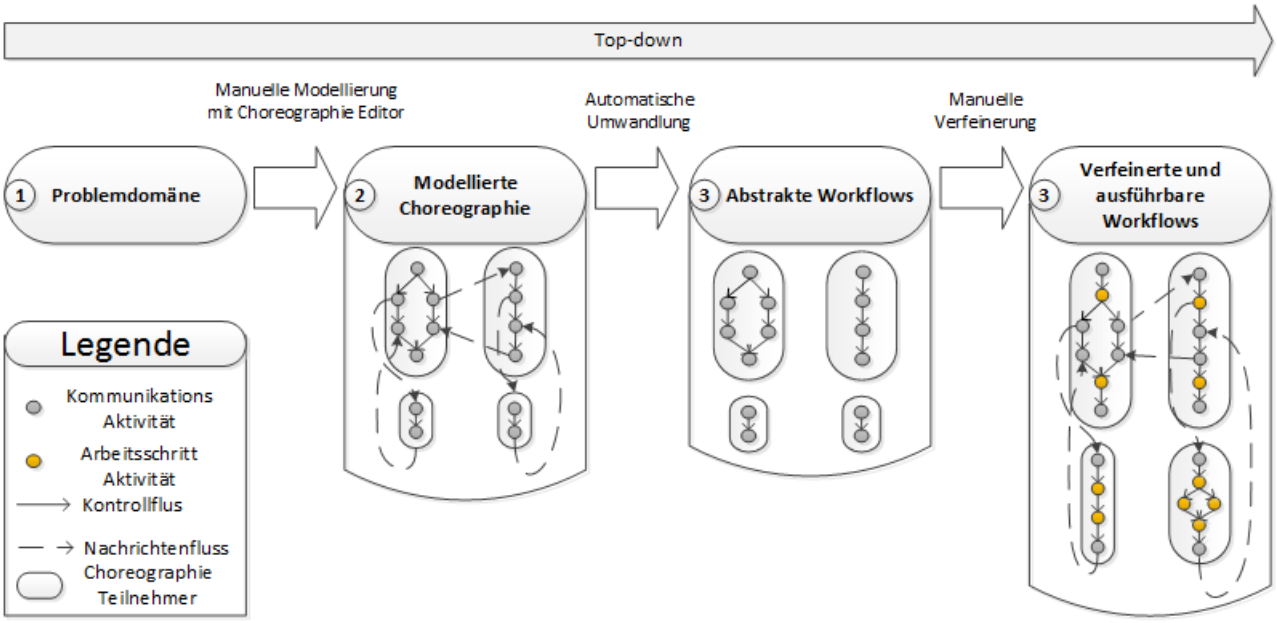


Abbildung 2.4.: Top-down Modellierungsansatz mit Beispiel, nach [WK16]

2.4. Technologien

Es hat sich etabliert, dass sich Aktivitäten eines Workflows einem Service bedienen um die Aktivität durchzuführen. Typischerweise werden die Services in Form von Webservices (WS) bereitgestellt.

Die Webservice Technologie (WST) umfasst eine große Anzahl[1] von Standards. Zwei wichtige Standards, die Grundlage für den Kern der WST sind, bilden die Web Service Description Language (WSDL) und SOAP. Alle entwickelten Standards werden unter dem Namen WS-Technologie Stack oder kurz WS-* zusammengefasst. Der Stern steht stellvertretend für die Namen der Standards, weil die Webservice Erweiterungen alle ein WS vorangestellt haben und mit dem Eigennamen enden. Die Spezifikationen aus WS-* basieren wiederum auf dem Extensible Markup Language (XML) Standard.

Extensible Markup Language Die Extensible Markup Language (XML) ist eine Auszeichnungssprache, mit der hierarchisch strukturierte Daten in Textdateien abgelegt werden können.

[1] "There are more than 150 WS-* Specs!"[Wee+05]

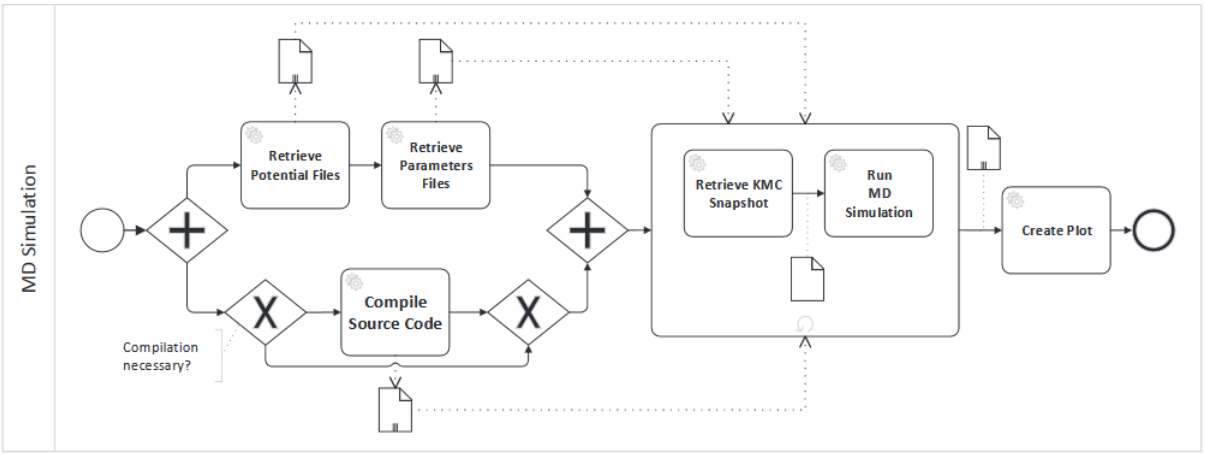


Abbildung 2.5.: Graphische Darstellung einer Orchestrierung mit BPMN [WKM]

Web Services Description Language Die bereitgestellte Funktionalität des Webservices wird mit Hilfe der Web Services Description Language (WSDL) beschrieben. Innerhalb der WSDL-Datei wird unter anderem festgehalten, welche eingehenden und ausgehenden Nachrichten vom Service erwartet werden und welche Operationen ausgeführt werden können.

WS-BPEL Die Web-Service Business Process Execution Language, oder kurz BPEL, erlaubt es Geschäftsprozesse in Form von Orchestrierungen zu spezifizieren. Hierfür werden drei Kategorien von Sprachbestandteilen angeboten. Dazu gehören die Basisaktivitäten, strukturierte Aktivitäten und Bereiche.

Business Process Model and Notation Für die graphische Darstellung der durch BPEL modellierten Prozesse gibt es keinen offiziellen Standard. Oft wird die grafische Business Process Model and Notation (BPMN) Sprache zu diesem Zweck herangezogen [Ley10]. In Abbildung 2.5 ist die grafische Darstellung einer Orchestrierung mit BPMN zu sehen. Sie zeigt den vereinfachten Workflow der Molekulardynamik-Simulation. Die Aktivitäten sind als Webservice implementiert [Nem14].

WS-BPEL4Chor Die Sprache BPEL ist für Orchestrierungen ein akzeptierter Standard, jedoch bietet sie keine Möglichkeit Choreographien zu beschreiben. Eine Lösung bevorzugt es, BPEL diesbezüglich zu erweitern. Mit der Erweiterung des Sprachumfangs von BPEL zu BPEL4Chor ist es möglich Choreographien modellieren zu können [Dec+07]. Hierfür wird BPEL so erweitert, dass an BPEL selbst keine Änderung vorgenommen werden. Dadurch wird ein nahtloses Zusammenbringen von Orchestrierungen und Choreographien erreicht.

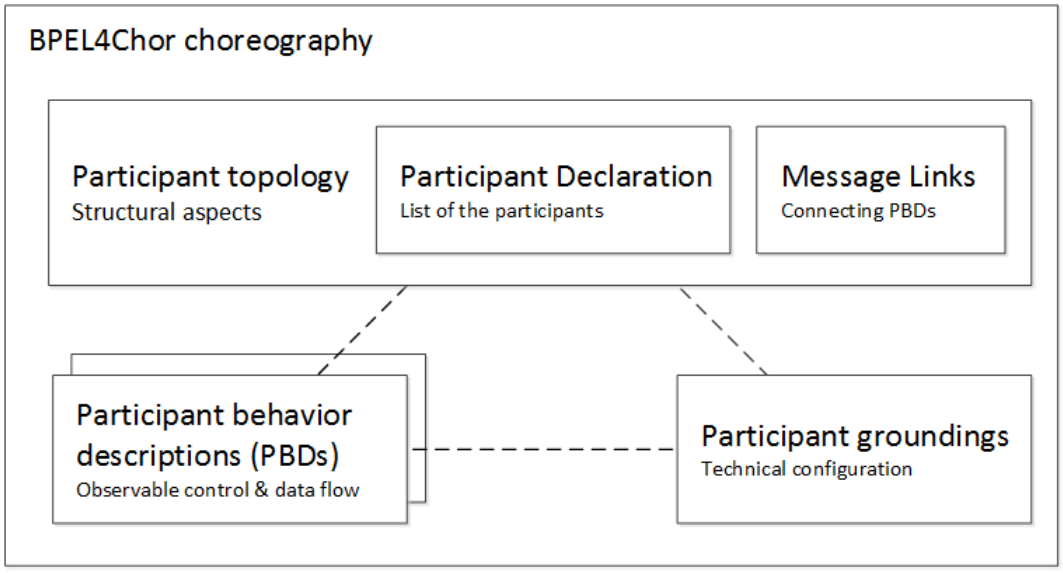


Abbildung 2.6.: BPEL4Chor Artefakte [Dec+07]

Die Erweiterung besteht aus drei Typen von Artefakten, welche zusammen BPEL4Chor bilden (Abbildung 2.6).

Für jeden Teilnehmer existiert ein Participant Behavior Description (PBD) Artefakt. Jedes Artefakt beinhaltet die Abhängigkeiten der Aktivitäten und beschreibt damit insbesondere den Kontrollfluss und jene Aktivitäten, welche für den Nachrichtenaustausch verantwortlich sind. In diesem Artefakt wird das Verhalten für jeden Teilnehmer separat festgehalten.

Die Participant Topology (PTop) beschreibt die strukturelle Ansicht der Choreographie. Hier werden die beteiligten Teilnehmer aufgeführt und die untereinander ausgetauschten Nachrichten aufgelistet. Das Artefakt verbindet so die einzelnen PBDs zu einer Gesamtheit.

In den vorherigen zwei Artefakten wurde das Verhalten und die Beziehung beschrieben. Im dritten Artefakt sind die technischen Details untergebracht. Diese Aufteilung realisiert ein entkoppeln der nicht-technischen und technischen Spezifikationen. Das erlaubt es, die Beschreibungen wiederzuverwenden und gegen unterschiedliche Implementierungen zu binden.

Das Participant Grounding (PG) Artefakt schlägt den Bogen von nicht-technischen Spezifikationen zu technischen Konfigurationen. Hier können konkrete Webservices an die vorher definierten Beschreibungen gebunden werden. Dies geschieht, indem innerhalb des Participant Groundings auf WSDL-Dateien referenziert wird.

2.5. Notation

Workflow Modelle können über Graphen-basierte Formalismen wie beispielsweise Petri-Netze, UML Aktivitätsdiagramme, BPMN und Event-driven Process Chain (EPC) beschrieben werden. Um nicht auf eine spezifische Beschreibungsart festgelegt zu sein, wird in dieser Arbeit eine allgemeine Darstellung als Graph vorgezogen. Außerdem lassen sich etablierte Graph-Algorithmen für die Verarbeitung der Modelle durch die Darstellung als Graph verwenden. Ein Graph lässt sich in die spezifischeren Beschreibungsarten transformieren, so dass dies keine Einschränkung darstellt.

Das Modell eines Workflows kann als Repräsentation eines (annotierten) Graphen verstanden werden. Der Graph enthält dabei Knoten, die stellvertretend für Aktionen stehen, und Kanten, welche die Reihenfolge der Abarbeitung festlegen. Die Pfeilspitze signalisiert die Richtung der Reihenfolge, in welcher die Knoten besucht werden.

Definition 2.5.1 (Graph, endlich und gerichtet, nach [Wei12])
Ein Graph G, bestehend aus einer endlichen Menge von Knoten V und einer endlichen Menge von Kanten E, ist ein 2-Tupel (V, E). Die Knoten und Kanten sind disjunkt, $V \cap E = \emptyset$. Eine Kante $e_i \in E$ verbindet zwei unterschiedliche Knoten $v_i, v_{i+1} \in V$. Eine Kante $E \subseteq V \times V : \{ e_i = (v_i, v_{i+1}) \mid e_i \in E, v_i, v_{i+1} \in V \}$ ist ein geordnetes Paar, wobei an erster Stelle der Startknoten und an zweiter Stelle der Endknoten notiert ist.

Wird ein Ausschnitt aus einem Workflow Modell betrachtet, entspricht dies einem Teilgraph.

Definition 2.5.2 (Teilgraph, nach [Wei12])
Gegeben sind zwei Graphen $G = (V, E)$ und $G' = (V', E')$. Falls $V' \subseteq V$ und $E' \subseteq E$ gilt, dass bedeutet G' umfasst nur Knoten und Kanten aus G, dann ist G' ein Teilgraph von G, notiert als $G' \subseteq G$.

Eine Sequenz von Aktionen, welche über Kanten verbunden sind, wird als Weg bezeichnet. Hierbei ist die Richtung der Kanten zu beachten.

Definition 2.5.3 (Weg, nach [Wei12])
Ein Weg W ist ein Teilgraph, für den ein Startknoten x_s und einen Endknoten x_e existiert. Alle Knoten und Kanten in W sind so gewählt, dass von Knoten x_s gerichtete Kanten zu Knoten x_e existieren. Alle aufeinander folgende Knoten x_i und x_{i+1} sind durch gerichtete Kanten (x_i, x_{i+1}) verbunden.

Sofern es möglich ist den Weg beliebig oft zu durchlaufen, wird der Weg auch Zyklus genannt. Das bedeutet, dass jeder Knoten des Weges mindestens eine eingehende und ausgehende Kante besitzt.

Definition 2.5.4 (Zyklus, nach [Wei12])
Ist bei einem Weg der Startknoten x_s und Endknoten x_e identisch, so heißt der Weg Zyklus.

Bei einem ungerichteten Graph kann die Kante in beide Richtungen gelesen werden. Deshalb wird oft bei der Kante auf den richtungsweisenden Pfeil verzichtet.

Definition 2.5.5 (Graph, ungerichtet, nach [Wei12])
Ein Graph ist ungerichtet, falls für jede Kante $e = (x_i, x_{i+1})$ eine Kante $e' = (x_{i+1}, x_i)$ existiert.

Ein Baum entspricht einer Spezialform des ungerichteten Graphen. Ein Knoten des Baumes kann (beliebig) bestimmt werden. Dieser Knoten wird als Wurzel bezeichnet. Ausgehend von der Wurzel verzweigt sich der Baum in die Breite. Zwei Verzweigungen dürfen sich nicht wieder verbinden.

Definition 2.5.6 (Baum, nach [Wei12])
Als Baum wird ein Graph bezeichnet, der ungerichtet ist und keine Zyklen enthält.

Die Single-Entry-Single-Exit Region [JPP94] ist eine kanonische Form eines Teilgraphen.

Definition 2.5.7 (Single-Entry-Single-Exit Region, nach [JPP94])
Eine Single-Entry-Single-Exit (SESE) Region ist ein Teilgraph mit einem Eingangsknoten x_s und Ausgangsknoten x_e. Jeder Knoten $x \in V$ ist vom Eingangsknoten x_s erreichbar. Für jeden Knoten existiert mindestens ein Weg zum Ausgangsknoten x_e. Der Eingangsknoten x_s hat eine eingehende Kante und der Ausgangsknoten x_e eine ausgehende Kante. Es darf weder eine zusätzliche Kante in die Region hinein, noch hinaus zeigen.

2.6. Verarbeitung

Für eine Transformation, der formalen Beschreibungen der Modelle in einen Baum, kann das Document Object Model (DOM) [Woo+00] herangezogen werden. Dies ist möglich, da die Beschreibungen in Form von strukturierten XML-Dokumenten vorliegen. Eine Konvertierung ist in beide Richtungen einfach möglich (Abbildung 2.7). Das DOM wurde als Schnittstelle für den Zugriff auf XML-Dokumente entwickelt. Es ermöglicht nicht nur das Navigieren durch das Dokument, sondern auch das Ändern von Inhalten.

Jeder DOM-Knotenbaum beinhaltet das Dokument als Wurzel. Der Baum lässt sich mit Beziehungen zwischen den Knoten beschreiben. Die direkt nachfolgenden Knoten werden als Kinder bezeichnet. Der vorherige Knoten entspricht dem Elternteil. Diese Verwandtschaftsbeziehungen lassen sich von jedem beliebigen Knoten aus beginnen.

Nachfolgend werden weitere Definitionen eingeführt, welche die Struktur des DOM beschreiben. Jeder Knoten erhält eine Nummer. Die Nummer wird ausgehend von der Wurzel, via einer Breitensuche, aufsteigend vergeben. Durch diese Nummerierung der Knoten lässt sich später die relative Position der Knoten bestimmen.

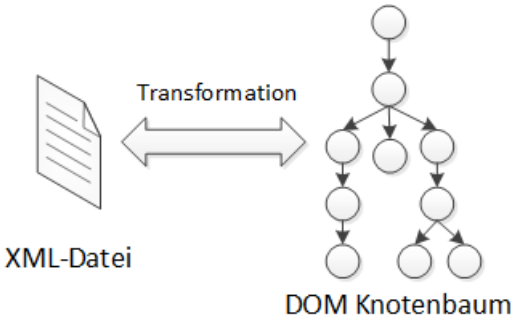


Abbildung 2.7.: XML-Datei und DOM-Knotenbaum

Definition 2.6.1 (Nummerierung der Knoten)
Gegeben sei ein Baum $B = (V, E)$ und ein Knoten $w \in V$, welcher die Wurzel bildet. Ausgehend von der Wurzel erhält jeder Knoten eine Nummer. $Nummer : Knoten \rightarrow \mathbb{N}_0$ Die Wurzel erhält die 0 zugewiesen. Die Nummern der Kinder werden via Besuchsreihenfolge der Breitensuche bestimmt.

Durch die Nummerierung der Knoten lässt sich zwischen allen Knoten eine Vorgänger- und Nachfolger-Beziehung formulieren. Ausgehend von einem Knoten x heißen alle anderen Knoten y mit $Nummer(y) < Nummer(x)$ Vorgänger- und alle Knoten mit $Nummer(x) < Nummer(y)$ Nachfolger-Knoten.

Definition 2.6.2 (Ebene eines Knoten)
Gegeben sei ein Baum $B = (V, E)$, dann ist jedem Knoten eine Ebene zugeordnet.
$Ebene : Knoten \rightarrow \mathbb{N}_0$ Die Wurzel erhält die Ebene 0 zugewiesen. Die Ebene eines Knoten wird via Expandierungsschritt der Breitensuche bestimmt.

Die Ebene eines Knoten wird später für die Verschachtelungstiefe der XML-Elemente benötigt.

An einen Eltern-Knoten kann ein Ast lokalisiert werden, um einen Teilgraphen zu erhalten. Ein Ast enthält alle Kinder des Eltern-Knotens, die sich nicht auf derselben Ebene wie der Eltern-Knoten befinden.

Definition 2.6.3 (Ast)
Gegeben sei ein Baum $B = (V, E)$ und ein Knoten $x \in V$ mit $Ebene(x) = i$. Dann sind alle von x erreichbaren Kinder, mit einer höheren Ebene, ein Ast. $Ast(x) = \{ (V_a, E_a) \mid$ Es existiert ein Weg von x nach $V_a \wedge Ebene(x) < Ebene(V_a) \}$

3. Verwandte Arbeiten

In dem vorherigen Kapitel wurden Grundlagen und Definitionen vermittelt. In diesem Kapitel werden verwandte Arbeiten vorgestellt.

Eine Reihe von ähnlichen und angrenzenden Arbeiten beschäftigen sich mit Aktualisierungen und Änderungsweitergabe innerhalb von Workflow-Choreographien. Darunter sind Arbeiten zu finden, welche sich mit dem Zusammenlegen von Prozessen oder dem identifizieren von betroffenen Teilnehmern bei einer Prozessänderung beschäftigen. Nachfolgend sind einige ausgesuchte Arbeiten beschrieben, welche Einfluss auf diese Arbeit genommen haben.

Es werden drei Themenbereiche untersucht. Zunächst wird die Domain Business Process Management (BPM) (Abschnitt 3.1) betrachtet, der diese Arbeit ebenfalls zuzuordnen ist. Hier beschäftigen sich die ausgewählten Arbeiten größtenteils mit Aktualisierungen in Prozessmodellen. Darauffolgend werden die angrenzenden Themengebiete des Data Engineerings (Abschnitt 3.2) und Software Engineerings (Abschnitt 3.3) mit einbezogen. Die Methoden des Data Engineerings sind hilfreich um identische oder ähnliche Aktivitäten innerhalb eines Prozessmodells zu identifizieren. Für das Zusammenlegen der Prozessmodelle sind die etablierten Algorithmen des Software Engineerings ein solider Ausgangspunkt.

3.1. Business Process Management

Das Optimieren von Geschäftsprozessen spielt im Business Process Management eine wichtige Rolle. Um einen Prozess zu optimieren ist es erforderlich, dass bestehende Modell zu analysieren und anzupassen. Eine Veränderung eines bestehenden Prozesses bringt jedoch Herausforderungen mit sich. Möchte beispielsweise ein Unternehmen Prozesse, an dem Partner beteiligt sind anpassen, müssen diese entsprechend informiert werden. Jedoch sollen dabei Teile von privaten Prozessen nicht veröffentlicht werden, da der Partner zu einem fremden Unternehmen gehören könnte und dies somit Geschäftsgeheimnisse verraten würde.

Change Propagation in Collaborative Processes Scenarios Die Arbeit von Fdhila et al. [FRMR12] untersucht verteilte und gemeinschaftliche Prozesse. Die beteiligten Partner handeln dabei eigenständig. Ändert ein Partner seinen privaten Prozess, ist die Frage, wie diese Änderungen an den Partner weitergeben werden können. Da der Partner meist selbst

wieder eigene Partner involviert, können diese Änderungen weitere transitive Änderungen zur Folge haben.

Zur Lösung dieses Problems wird ein generischer Ansatz für Change Propagation auf Basis eines Refined Process Structure Tree (RPST) [VVK09] vorgeschlagen. Der Ansatz besteht aus vier Schritten: (i) identifizieren der betroffenen Aktivitäten und Partner, (ii) aushandeln der Change Operationen für den betroffenen Partner, (iii) abschließen der Verhandlungen mit dem Partner und (iv) überprüfen der Konsistenz und Kompatibilität. Für transitive Änderungen muss der Vorgang entsprechend mit jedem Partner wiederholt werden.

Dieser Ansatz ist für einzelne Änderungen ausgelegt, welche einmalig an einen Partner übermittelt werden. Für eine größere Anzahl von Änderungen, welche gesamtheitlich an alle beteiligten Partner gesendet werden, ist dieses Verfahren nicht geeignet. Jedoch fließen die Überlegungen, zur Auswahl der betroffenen Partner, in die vorliegende Arbeit ein.

Change patterns and change support features In der Arbeit von Weber et al. [WRRM08] wurden typische Änderungen an Prozessen analysiert. Als Grundlage für die Analyse dienten mehrere Fallstudien. Daraus resultierten 18 Änderungsmuster (*Change-Pattern*), welche jeweils einen Änderungsvorgang an einem Prozess beschreiben. Die Änderungsmuster können als Referenz verwendet werden oder dazu Systeme, an Hand der Unterstützung der Muster, zu vergleichen.

Es lassen sich 14 der Änderungsmuster als *Adaption Pattern* identifizieren, die das direkte Modifizieren des Prozesses behandeln. Darunter sind Muster wie: Einfügen, Löschen, Bewegen oder Ersetzen von Prozessfragmenten zu finden.

Die vorliegenden Modelle dieser Arbeit unterliegen diesen Änderungsmustern. Das zu erstellende Konzept muss sich mit den Änderungsmustern auseinandersetzen und Mittel bieten diese zu unterstützen.

Merging Event-Driven Process Chains Wie zwei Prozessmodelle, welche als Event-driven Process Chains (EPC) [KSN92] modelliert wurden, zu einem einzigen Prozessmodell zusammengeführt werden können, wird in der Arbeit von Gottschalk et al. [GAJV08] vorgestellt. Die beschriebene Funktionalität wird zusätzlich im ProM (Process Mining) Framwork[1] bereitgestellt.

Die Motivation dieser Arbeit liegt in der Übernahme beziehungsweise Zusammenlegung zweier Unternehmen begründet, welche ihre Prozesse als EPC modelliert haben. Die Eingliederung des Unternehmens spiegelt sich in den EPC wieder. Deshalb müssen die EPC beider Unternehmen auf ein gemeinsames Modell gebracht werden. Für diesen Vorgang wird ein Merge-Algorithmus vorgeschlagen. Der Merge-Algorithmus besteht aus drei Phasen. Zunächst werden die EPC auf

[1] http://www.promtools.org

ein Graphen basierendes Modell reduziert. Die beiden Graphen werden dann vereinigt. Dies passiert über die jeweilige Zusammenlegung der Knoten und der Kanten. Zuletzt wird der neu entstandene Graph in ein EPC zurückgeführt.

Dieses Vorgehen dient als Ausgangssituation für weitere Schritte. Die EPC wurden zwar Zusammengelegt, jedoch wurden Abhängigkeiten und Semantik der Aktivitäten nicht berücksichtige. Das so entstandene Modell muss auf jeden Fall von einem Experten überarbeitet werden, bevor es verwendet werden kann.

Der vorgeschlagene Ansatz von [GAJV08] ist sehr stark an EPC geknüpft und wird deshalb im Kontext der vorliegenden Arbeit nicht verwendet.

Die von Experten durchgeführten Änderungen werden in der Regel dokumentiert. In sogenannten Änderungshistorien (Change-Log) werden die Änderungen festgehalten. Ein Change-Log wird häufig automatisch erstellt. Jedoch kann es vorkommen, dass alte über die Zeit gewachsene Modelle keine Change-Log aufweisen. Weitere Probleme können sein, dass der Change-Log falsch, unvollständig oder verlorengegangen ist. Doch auch ohne Change-Log müssen die Änderungen an den Modellen nachvollziehbar sein.

Business Process Merging - An Approach based on Single-Entry-Single-Exit Regions In seiner Diplomarbeit [Ger07] präsentiert Gerth einen Ansatz für Business Process Merging, der nicht auf einen initialen Change-Log angewiesen ist.

In diesem wird wie folgt vorgegangen: Zunächst werden die Unterschiede der Prozess Modelle, unter Beachtung der Abhängigkeiten, erkannt. Dazu wird das Prozess-Modell in SESE Regionen partitioniert, um eine hierarchische Zerlegung zu erhalten. Durch die Zerlegung in SESE Regionen kann das Modell einfacher auf Unterschiede untersucht werden. Aus den Unterschieden wird dann ein Change-Log generiert der minimal ist, also keine unnötigen Operationen enthält. Der Change-Log beinhaltet alle Schritte, welche durchgeführt werden müssen um die Unterschiede aufzulösen. Als letztes werden die Unterschiede, durch Anwendung des Change-Log, aufgelöst.

Die Übernahme der Änderungen erfolgt mit Hilfe einer grafischen Oberfläche. In dieser können die ermittelten Unterschiede schrittweise in das neue Modell integriert werden. Eine automatische Integration erfolgt nicht.

In der Arbeit von Gerth werden als zusätzliches Hilfsmittel „Fixpunkte“ verwendet. Diese Fixpunkte dürfen nicht von einer Änderung im Change-Log betroffen sein und dienen zur Positionierung der anderen Aktivitäten. Das Konzept der Fixpunkte wird aufgenommen und dem Kontext der vorliegenden Arbeit angepasst. In der vorliegenden Arbeit dürfen diese Punkte von Änderungen betroffen sein und dienen zum relativen positionieren der Aktivitäten. Zudem ist keine Generierung eines Change-Log erforderlich.

Aus dieser Arbeit gingen weitere Arbeiten hervor. So wird beispielsweise die Rekonstruktion des Change-Log in der folgenden Arbeit näher beschrieben.

Detecting and Resolving Process Model Differences in the Absence of a Change Log In der Arbeit von Küster et al. [Küs+08] wird das Verfahren aus [Ger07] aufgegriffen. Es wird gezeigt, wie die Unterschiede der Modelle erkannt und der Change-Log durch Berechnen der notwendigen Änderungsoperationen wiederhergestellt werden kann. Der berechnete Change-Log kann schließlich innerhalb des IBM WebSphere Business Modeler[2] verwendet werden, um die Änderungen von Hand durchzuführen.

Das Konzept der *Correspondence* [PB03] und SESE Fragmente wird, für Business Modelle, eingeführt. In der vorliegenden Arbeit findet das Prinzip der *Correspondence* auch seine Anwendung.

Eine weitere Arbeit von Gerth et al. beschäftigt sich mit der konzeptionellen Durchführung von Änderungen an Prozessmodellen. Diese Erkenntnisse sind in der folgenden Arbeit niedergeschrieben.

Towards Rich Change Management for Business Process Models Wie konzeptionell zwei oder mehrere Prozessmodelle vereinigt werden können, ist in der Arbeit [GL12] beschrieben. Dazu wird ein *Framework* für *Change-Management* vorgestellt. Das Framework erlaubt es Modelle unterschiedlicher Modellierungssprachen zu verarbeiten und darüber hinaus die Semantik dieser beim Vergleich zu berücksichtigen.

Die Umsetzung dieser Arbeit orientiert sich direkt an dem vorgeschlagene Framework. Das Framework und die notwendige Adaption an den Kontext dieser Arbeit ist in Abschnitt 4.2 beschrieben.

In der Arbeit von Pottinger und Bernstein [PB03] wird das generische Verschmelzen zweier Modelle, wie Datenbank Schemata, UML Modelle oder Ontologien untersucht. Das vorgestellte Konzept liefert eine „duplikatsfreie“ Vereinigung der Modelle. Das Vorgehen ist für Prozessmodelle ungeeignet und richtet sich primär an Modelle wie Klassendiagramme.

Für ein weitergehendes Verständnis sind außerdem Teilbereiche aus den Themengebieten des Data Engineerings und des Software Engineerings hilfreich. Im Folgenden wird der Zusammenhang dieser Gebiete erläutert.

[2] http://www.ibm.com/software/products/en/business-process-manager-family

3.2. Data Engineering

Modelle werden im Laufe der Bearbeitung selten von einer Person alleine modelliert. Das ist einer von vielen Gründen, warum für Modelle keine einheitliche Bezeichnungen der Aktivitäten verwendet werden. So könnten Aktivitäten aus verschiedenen Modellen mit der Beschriftung „erhitzen“ und „aufwärmen“ vorliegen. Liegt dann für die Aktivitäten kein eindeutiger Identifikationsschlüssel vor, ist es schwierig diese Zugehörigkeit zu erkennen.

Ein Ausschnitt des Data Engineerings beschäftigt sich mit dem Finden von semantisch identischen Entitäten. Dies ist wiederum ein Teilbereich aus dem Gebiet des *Schema Alignment*. Dieses Konzept lässt sich für das Finden von semantisch identischen Aktivitäten in Workflow Modellen übertragen. Die Methoden des Data Engineering sind hilfreich um Aktivitäten zu identifizieren, die unterschiedlich beschriftet aber logisch identisch sind, um sie weiter zu verarbeiten.

Aligning Business Process Models In der Veröffentlichung von Dijkam et al. [Dij+09] werden zwei Vorschläge mit mehreren Varianten für das Zuordnen ähnlicher Elemente in Prozessmodellen vorgestellt und evaluiert. Der erste Versuch beinhaltet einen Vergleich der Elemente auf lexikalischer Basis. Dazu werden die Beschriftungen der Elemente herangezogen und wortweise verglichen.

Weitere Versuche betrachten die Struktur der Prozesse. Dazu wird das Prozessmodell als Graph interpretiert. Hier wird zusätzlich zum Vergleich der Beschriftungen die eingehenden und ausgehenden Kanten herangezogen. Abschließend wird festgestellt, dass von allen vorgestellten Varianten der strukturelle Vergleich, mittels einem Greedy-Algorithmus, die höchste Präzision aufweist.

Die Arbeit von [Dij+09] zeigt, dass ein simples Vergleichen, der Beschriftungen in Verbindung der Graph-Struktur, meistens die besten Ergebnisse erzielt und legitimiert damit diese Vorgehensweise.

Eine weitere interessante Arbeit stammt von La Rosa et al. [LR+10]. In dieser werden Prozesse, welche als EPC vorliegen, als annotierte Graphen aufgefasst. Bevor die Graphen verschmolzen werden, müssen zusammengehörige Knoten und Kanten gefunden werden. Die Knoten und Kanten des Graphen werden mit einer *Matching Score* bewertet, welche ausdrückt wie ähnlich sich diese sind.

Die Aktivitäten der verwendeten Modelle in dieser Arbeit haben alle eine Identifikationsnummer, weshalb eine eindeutige Zuordnung zwischen mehreren Modellen einfacher möglich ist, und nicht auf die Methoden des Data Engineerings zurückgegriffen werden muss. Jedoch bieten diese Arbeiten einen Einstiegspunkt, falls bei den Elementen keine Identifikationsnummer verfügbar sein sollte.

3.3. Software Engineering

Formelle Workflow Beschreibungen liegen in der Form von textuellen Dokumenten vor. Das Problem, Unterschiede und Gemeinsamkeiten in Dokumenten zu finden, besteht auch im Software Engineering. Es ist für Programme mit Versionsverwaltung von fundamentaler Bedeutung und legt den Grundstein für ein *Merge* zweier Versionen. Es gibt bereits etablierte Algorithmen, wie beispielsweise den *Three-way Merge*, um Text-Dokumente zu verschmelzen.

Using Versioned Tree Data Structure, Change Detection and Node Identity for Three-Way XML Merging Thao und Munson [TM10] greifen die Methode des *Three-way merges* auf und präsentieren ein Konzept, um damit XML formatierte Dateien zu vereinen. Der Algorithmus verwendet dabei den herkömmlichen Three-way Merge. Der Three-way Merge wurde so erweitert, dass die XML-Dokumente als Baum verarbeitet werden. Die zu vereinigenden Knoten werden über eine *Longest Common Subsequence* (LCS) bestimmt. Für den vorgeschlagenen Ansatz sind eindeutige Identifikatoren für die Knoten notwendig, damit dieser angewendet werden kann.

In der vorliegenden Arbeit wird die Repräsentation der XML-Dokumente als Graph übernommen. Jedoch erfolgt die Verarbeitung der XML-Dokumente in [TM10] ereignisbasiert, mittels SAX (Simple API for XML) und nicht baumbasiert (DOM).

Die Arbeit von Alanen und Porres [AP03] präsentiert Algorithmen, für die Berechnung von Unterschieden und Vereinigungen, auf der Basis von Meta Object Facility (MOF) Modellen (entspricht einem Meta-Meta-Modell). Das Eclipse Modeling Framework Ecore-Metamodell basiert auf einer Teilmenge des MOF-Standards. Die Algorithmen behandelt jedoch keine Unterschiede, welche Abhängigkeiten oder verschobene Elemente umfassen.

Dies waren nur einige wenige Veröffentlichungen aus dem Gebiet *Modell-Merging*. Das Problem der Modell-Fusionierung ist allgegenwärtig und tritt in den unterschiedlichsten Themenbereichen auf. Dies hebt die Bedeutung dieser Herausforderung weiter hervor. Es ist somit nicht verwunderlich, dass bereits eine Vielzahl divergenter und kongruenter Vorschläge inklusiver Varianten existieren.

Aus diesem Grund wird in der Arbeit von Brunet et al. [Bru+06] ein Framework vorgeschlagen, mit der vorhandene Methoden verglichen werden sollen. Dazu werden algebraische Operatoren für den *merge*, sowie *match*, *diff*, *split* und *slice* eingeführt. Mit Hilfe dieser Operationen sollen die Verfahren kategorisiert und verglichen werden können.

4. Anforderungsanalyse

In dem vorherigen Kapitel wurden verwandte Arbeiten bezüglich des Themas Modell-Fusion vorgestellt. Auf dieser Basis wird im folgenden Kapitel weiter aufgebaut und Vorarbeit für das Konzept geleistet. Die Anforderungsanalyse beginnt auf einer hohen Abstraktionsebene und wird daraus konkretisiert.

Nachfolgend wird in Abschnitt 4.1 präsentiert, was unter Aktualisierung und Änderungsweitergabe in dieser Arbeit verstanden wird. Danach werden in Abschnitt 4.2 die allgemeinen Schritte vorgestellt, welche für das Erreichen des Ziels erforderlich sind. Die allgemeine Vorgehensweise wird an die Rahmenbedingung dieser Arbeit angepasst. Daraufhin werden in Abschnitt 4.3 verschiedene Änderungsmuster bei der Modelländerung betrachtet. Abschließend werden in Abschnitt 4.4 die Auswirkungen auf die BPEL4Chor Artefakte untersucht.

4.1. Aktualisierung und Änderungsweitergabe

Die Begriffe Aktualisierung und Änderungsweitergabe werden im Kontext der Modellierung mit Workflow-Choreographien verwendet. Sie beziehen sich auf die Veränderung des Choreographie-Modells und die damit verbundene Weitergabe der Änderungen an die Choreographie-Teilnehmer. Die Choreographie-Teilnehmer müssen die weitergegebenen Änderungen in ihre Orchestrierung einpflegen. Dies passiert beides in der Phase der Modellierung. Um dies zu verdeutlichen, wird der Ablauf der Modellierung im folgenden schematisch dargestellt.

Ein Wissenschaftler verfolgt eine Idee und möchte seine Idee unter zur Hilfenahme einer Simulation verifizieren. Den Versuchsaufbau erstellt er am Computer. In diesem Fall möchte er den Versuchsaufbau als Scientific-Workflow modellieren, um diesen direkt nach der Modellierung auszuführen und die Ergebnisse zu untersuchen. Innerhalb des Versuchsaufbaus werden verschiedene existierende Simulationen verwendet und zu einer neuen kombiniert. Um das Zusammenspiel der Simulationen zu beschreiben, wird der Workflow in Form einer Choreographie modelliert. Aus dieser beschriebenen Choreographie wird wiederum für jede Simulation ein separater Workflow generiert. In diesen generierten Workflows wird das individuelle Verhalten ergänzt. Für jede in der Choreographie beteiligte Simulation existiert nun eine Orchestrierung, welche die konkreten Anweisungen enthält.

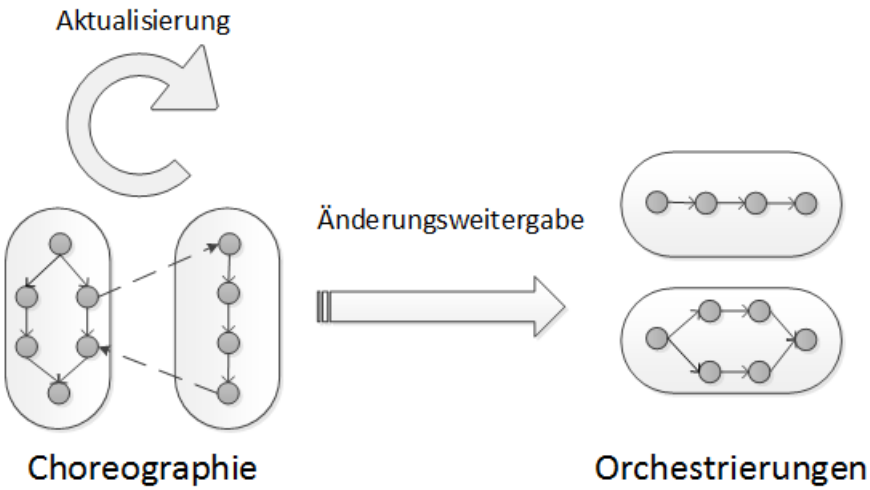


Abbildung 4.1.: Aktualisierung der Choreographie und Änderungsweitergabe an die betroffenen Orchestrierungen

Auf Grund des systematischen ausprobierens von unterschiedlichen Versuchsaufbauten ist es erforderlich, dass Versuchsmodell, und damit die Choreographie, zu verändern. Eine Veränderung des Zusammenspiels macht es wiederum erforderlich die konkreten Orchestrierungen entsprechend anzupassen. In Abbildung 4.1 ist der Zusammenhang zwischen Choreographie und Orchestrierung dargestellt. Der Versuchsaufbau wird aktualisiert und Änderungen müssen an die betroffenen Simulationen weitergeben werden.

Die betroffenen Choreographie-Teilnehmer müssen die vorgenommenen Änderungen umsetzen. Nur so ist weiterhin eine Kommunikation und somit eine Zusammenarbeit unter den Teilnehmern möglich. Im Zuge dessen erhält jeder Teilnehmer die neue Choreographie Beschreibung, um den alten Workflow anzupassen.

Mit dem Erhalt der neuen Choreographie werden die alten Beschreibungen als obsolet betrachtet. Dennoch werden alte Versionen nicht einfach gelöscht, sondern archiviert. So sind sie für den Prüfzweck (Audit) verfügbar und können darüber hinaus als Nachschlagewerk oder Dokumentation verwendet werden.

Eine neue Version entsteht für gewöhnlich aus einer Ableitung einer Version oder aus einer Kombination mehrerer Versionen. Die Versionen teilen sich deshalb mindestens eine Gemeinsamkeit. Die Gemeinsamkeiten werden aus den ursprünglichen Versionen vererbt.

Die Weitergabe der Änderungen erfolgt mit Hilfe der Teilnehmer-Verhaltens-Beschreibung (Siehe 2.4 PBD). Die Änderungshistorien (Change-Log) verbleiben beim Partner, der die Änderungen vorgenommen hat und werden meist nicht weitergegeben. Es kann nur auf eigene oder veröffentlichte Change-Log zurückgegriffen werden. Jeder Teilnehmer erhält aus der Choreographie nur seine PBD.

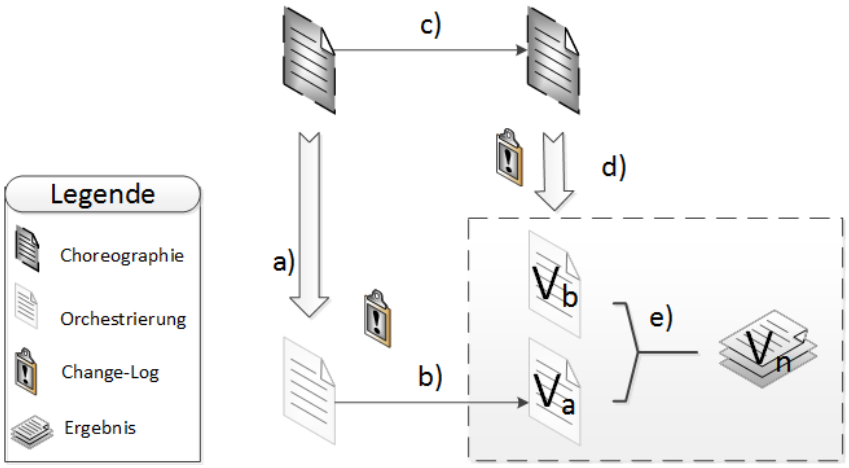


Abbildung 4.2.: Evolution einer Version durch Ableitung

Die neue Version der Choreographie wird nur an Teilnehmer weitergegeben, welche von einer Änderung betroffen sind, um diese vor unnötigen Nachrichten zu bewahren. Diese können die existierende Orchestrierung weiterhin verwenden.

Die Ereignisse und der Dokumentenfluss sind in Abbildung 4.2 zu sehen. Die Abbildung stellt den Ablauf aus der Sicht eines einzelnen Teilnehmers dar. (a) Zunächst existiert eine initiale Beschreibung der Choreographie, woraus für jeden Teilnehmer seine PBD generiert wird. (b) Die PBD wird mit weiteren Verhalten zu einer ausführbaren Orchestrierung verfeinert. Die Verfeinerungen können in einem Change-Log vermerkt werden. (c) Eine neue Version der Choreographie wird erstellt. (d) Die Änderungen werden mitgeteilt. Bei der Mitteilung kann ein Change-Log mit zusätzlichen Informationen enthalten sein. (e) Der Teilnehmer besitzt eine alte Version V_a mit Verfeinerung und eine neue Version V_b ohne Verfeinerung. (e) Aus den beiden Versionen wird die neue Version V_n generiert, welche die Neuerungen aus V_b, sowie die Verfeinerungen aus V_a enthält. Ein vorhandener Change-Log kann für eine Verbesserung des Ergebnisses herangezogen werden, sofern dieser vorliegt.

Das Vorgehen, wie zwei Modelle fusioniert werden können, wird im nächsten Abschnitt vorgestellt.

4.2. Fusion von Modellen

Das Fusionieren von Modellen bringt verschiedene Herausforderungen mit sich. In der Arbeit von Gerth und Luckey [GL12](siehe Verwandte Arbeiten) wird eine Ablaufstruktur für das Modell Änderungsmanagement vorgestellt, welches die wichtigsten Schritte zum fusionieren von Modellen festhält. Diese Struktur wird aufgegriffen und das weitere Vorgehen an Hand dieser organisiert.

Die Struktur besteht aus sieben Komponenten, welche in Abbildung 4.3 dargestellt sind. Ausgangspunkt stellen die Eingabe-Modelle dar, welche zu einem neuen Modell fusioniert werden sollen. Die erste Komponente beschreibt die Notwendigkeit die Modelle in ein einheitliches abstraktes Zwischenformat zu konvertieren. Dadurch können Modelle mit unterschiedlichen Sprachen der Modellierung weiterverarbeitet werden. Im nächsten Schritt werden die Modelle abgeglichen. Das Abgleichen der Modelle bedeutet Elemente zu erkennen und zu finden, die zusammengehören. Daraufhin lassen sich die Unterschiede zwischen den Modellen herausfinden. Es kann sein, dass Elemente Abhängigkeiten untereinander aufweisen. Abhängigkeiten können beispielsweise in der Reihenfolge der Elemente oder durch die Existenz anderer Elemente auftreten. Die vorhandenen Abhängigkeiten müssen bedacht werden. Treten Widersprüche innerhalb der Modelle auf, so müssen diese soweit möglich aufgelöst werden. Dies kann besonders dann der Fall sein, wenn mehr als zwei Modelle miteinander fusioniert werden sollen. Wurden all diese Schritte durchlaufen, findet zuletzt der Schritt des Fusionierens statt. Hierbei werden nach vordefinierten Regeln die Modelle miteinander verbunden und zurück auf ein konkretes Datenformat gebracht.

Im Folgenden werden die notwendigen Schritte im Kontext dieser Arbeit betrachtet.

Zwischenformat Die vorliegenden Choreographien sind mit Hilfe der BPEL4Chor Artefakte beschrieben. Diese werden wiederum in einem XML konformen Format bereitgestellt. Dennoch wird nicht direkt mit den XML-Daten gearbeitet, sondern es werden die Dateien eingelesen und in eine abstrahierte Graph-Repräsentation gebracht. Dies erlaubt eine leichtere Weiterverarbeitung und Formalisierung. Zudem kann auf bewährte Graph-basierte Algorithmen zurückgegriffen werden. Die Repräsentation als Graph erlaubt es außerdem, zwei Dateien mit verschiedenem XML-Standards zu vergleichen. Das Konzept lässt sich daher für alle BPEL4Chor Artefakte anwenden.

Zudem wird die Struktur der zu integrierenden Modelle durch das Abstrahieren vereinfacht. Einzelne Elemente des Modells werden zu Knoten und der Kontrollfluss zu Kanten degradiert. Dies erlaubt komplexe Strukturen wie beispielsweise Schleifen einfacher zu behandeln. Eine Schleife umschließt mehrere Elemente die öfters ausgeführt werden sollen. Im ursprünglichen Modell stellt dies einen Zyklus dar, der schwer zu verarbeiten ist. In der Repräsentation als Graph ist eine Schleife ein Knoten, an dessen Ast wiederum die umschlossenen Elemente hängen. Es ist in erster Linie nicht relevant, wie oft und ob überhaupt die Schleife durchlaufen wird. Deshalb führt eine Schleife nur eine weitere Ebene im Graph ein, ohne einen Zyklus

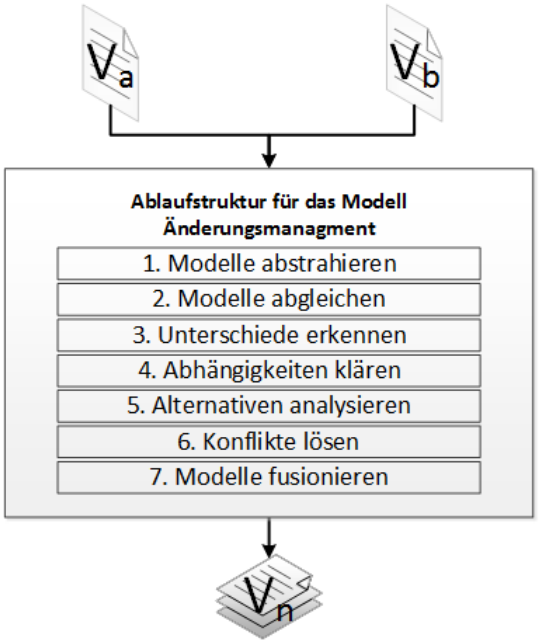


Abbildung 4.3.: Ablaufstruktur für das Modell Änderungsmanagement nach [GL12]

zu bilden. Zusätzliche Informationen werden nicht verworfen, sondern an den entsprechend Knoten oder den Kanten angehängt. Durch diese einfache Konvertierung ist es möglich, den Graph wieder in das Ursprungsformat zurückzuführen.

Das bedeutet, dass der Aufbau und das ursprüngliche Format der Modelle für die Verarbeitung nicht relevant ist, sofern sich diese in einen Graph und zurück konvertieren lassen.

Definition 4.2.1 (Modell)
Ein Modell M ist ein gerichteter endlicher Graph ohne Zyklen.

Einem Knoten $x \in V$ aus dem Modell können Attribute zugewiesen sein. Die Werte werden via den Namen des Attributs und den Knoten referenziert. Hierfür werden die Funktionen $Hat\text{-}Attribut : Knoten \times Name \mapsto Wahrheitswert$ und $Attribut : Knoten \times Name \mapsto Wert$ verwendet. Der Wahrheitswert liefert ein $\top$, falls das Attribut definiert ist, ansonsten wird ein $\bot$ zurückgegeben. Der Name des Attributs kann auch per Indexschreibweise angeben werden $Attribut_{Name}$.

Jeder Knoten erhält als Wert alle zusätzlichen Informationen, die mit ihm in Verbindung stehen, zugewiesen. Dazu zählen beispielsweise vorhandene XML-Attribute.

Identifikation Für die Modelle wird angenommen, dass diese als gerichteter endlicher Graph ohne Zyklen vorliegen. Das reduziert die Identifikation von Elementen auf das Identifizieren von Knoten.

Die vorliegenden Modelle verwenden für einen Großteil der Elemente eine eindeutige Identifikationsnummer (ID). Über diese ID lässt sich ein Element und folglich auch der Knoten eindeutig identifizieren. Für die ID kann die Kurzschreibweise $ID(x) = Attribut_{ID}(x)$ verwendet werden.

Besitzt ein Knoten keine ID, muss die Identifikation anders durchgeführt werden. Dies kann beispielsweise über den Namen oder der relativen Position des Knoten erfolgen. Für die vorliegenden Modelle ist dieses Vorgehen ausreichend. Komplizierte neue Heuristiken (Vgl. Kapitel 3.2) werden nicht benötigt.

Abgleichen Um Modelle zu einem neuen Modell fusionieren zu können, muss zunächst die Beziehung zwischen den Knoten festgestellt werden. Mit Hilfe des Zwischenformats reduziert sich das Feststellen der Beziehungen auf den Vergleich der Graphen. Es reicht aus, für jeden Knoten die An- oder Abwesenheit im anderen Modell zu analysieren. Für diese Art der Beziehung wird der Begriff der Zuordnung[PB03] verwendet.

Eine Zuordnung (vgl. [Ger07]) ist eine Abbildung von gleichwertigen Knoten oder Kanten zwischen Modellen. Es herrscht eine Gleichwertigkeit zwischen zwei Knoten, wenn zum Beispiel die gleiche Funktionalität von diesen Knoten bereitgestellt wird. Die Zuordnung lässt sich analog für Kanten durchführen. Je nachdem, ob eine Komponente des Graphen ein Gegenstück in einem anderen Modell hat, wird der Typ der Zuordnung unterschieden.

Definition 4.2.2 (Gleichwertige Knoten)
Zwei Knoten $x \in V_a$ und $y \in V_b$ sind gleich, genau dann wenn diese identisch oder übereinstimmend sind.

- ***Identisch**: Die Knoten haben beide eine ID, welche zudem identisch ist. $Hat\text{-}Attribut_{ID}(x) = \top$ und $Hat\text{-}Attribut_{ID}(y) = \top$ mit $ID(x) = ID(y)$.*
- ***Übereinstimmend**: Die Knoten haben beide keine ID und die Namen der Knoten stimmen überein. $Hat\text{-}Attribut_{ID}(x) = \bot$ und $Hat\text{-}Attribut_{ID}(y) = \bot$ mit $Name(x) = Name(y)$.*

Für diese Arbeit sind drei Zuordnungstypen relevant. Entweder ein Knoten besitzt genau einen Partnerknoten im anderen Modell oder nicht. Beim Fehlen eines Partnerknotens wird festgehalten, in welchem Modell der Knoten vorhanden ist. Deshalb existieren für die Abwesenheit eins Knoten zwei Zuordnungstypen. Es ergibt sich die adaptierte Definition der Zuordnung nach [Ger07].

Definition 4.2.3 (Zuordnung, nach [Ger07])
Sind $M_a = (V_a, E_a)$ und $M_b = (V_b, E_b)$ ein Modell und $x \in V_a$ und $y \in V_b$ Knoten. Dann werden die folgenden Typen unterschieden:

- ***1-0 Zuordnung**: Ein Knoten x hat eine 1-0 Zuordnung, genau dann wenn $\forall y \in V_b : x \neq y$. Geschrieben als $C_{1-0}(x, y) = \top$*
- ***0-1 Zuordnung**: Ein Knoten y hat eine 0-1 Zuordnung, genau dann wenn $\forall x \in V_a : y \neq x$. Geschrieben als $C_{0-1}(x, y) = \top$*
- ***1-1 Zuordnung**: Ein Knoten x hat eine 1-1 Zuordnung zu einem Knoten y, genau dann wenn $x = y$. Geschrieben als $C_{1-1}(x, y) = \top$*

Die Beziehungen werden in der Regel zwischen zwei Modellen festgelegt. Für die 1-1 Zuordnung gilt eine symmetrische $C_{1-1}(x, y) = \top \Leftrightarrow C_{1-1}(y, x) = \top$ und transitive $C_{1-1}(x, y) = \top \wedge C_{1-1}(y, z) = \top \Rightarrow C_{1-1}(x, z) = \top$ Beziehung.

Mehrwertige Beziehungen wie 1-n, n-1 oder n-m werden nicht betrachtet. Eine 1-n oder n-1 Beziehung würde bedeuten, dass einem Knoten im anderen Graph mehrere gleichwertige Knoten zugeordnet werden. Dies kann aufgrund der ID nicht passieren. Eine ID wird keinen zwei XML-Elementen gleichzeitig zugeteilt. Deshalb kann auch keine n-m Beziehung vorliegen. Dies würde sonst bedeuten, dass in beiden Modellen mehrere gleichwertige Knoten vertreten sind.

Aus den Zuordnungstypen kann direkt festgestellt werden, welche Art von Veränderungen zwischen zwei Modellen vorgenommen worden sind. Unterschiede werden hierbei von der 1-0 und 0-1 Zuordnung ausgedrückt. Diese signalisieren das Entfernen oder Hinzufügen von Komponenten. Gemeinsamkeiten lassen sich durch 1-1 Zuordnungen erkennen. In diesem Fall ist die Komponente in beiden Modellen vertreten.

Abhängigkeiten Manche Änderungen setzen sich aus mehreren kleinen Teilmodifikationen zusammen, die nur gemeinsam einen Sinn ergeben und nur gemeinsam übernommen werden dürfen. Das heißt, alle zusammengehörende Teile einer Änderung werden gruppiert und nur ganz oder gar nicht umgesetzt.

Eine semantische Abhängigkeit besteht in der Anordnung der Aktivitäten, die Wiederrum die Reihenfolge der Abarbeitung festlegt. Wird zum Beispiel eine Aktivität ersetzt, reicht es nicht aus, die alte Aktivität zu entfernen. Es muss auch die neue Aktivität eingefügt werden. Andernfalls ist die Intention der Änderung missachtet worden.

Im Rahmen dieser Arbeit sollen alle Änderungen automatisiert und auf einmal übernommen werden. Eine Schrittweise Integration einzelner Änderungen ist nicht vorgesehen. Dies verhindert zugleich semantisch inkonsistente Zustände, welche durch unvollständige Änderungen hervorgebracht werden. Jedoch müssen die Vorgänger- und Nachfolger-Beziehungen der Aktivitäten eingehalten werden. Es ist davon auszugehen, dass sich alle eingefügten Verfeinerungen auf eine vorherige Kommunikationsaktivität beziehen.

Alternativen Bei der Transformation zurück in die Artefakte muss der Graph in das XML-Format gebracht werden. Dabei können keine alternativen Varianten entstehen. Der Inhalt ist eindeutig und das Format wohlstrukturiert. Es ist demnach nicht erforderlich zwischen mehreren Repräsentationsarten zu wählen. Dieser Schritt kann daher übersprungen werden.

Konfliktmanagement Beide Modelle stehen zwar in Relation, werden jedoch unabhängig voneinander verändert. Dies kann dazu führen, dass gleichwertige Knoten unterschiedliche Eigenschaften aufweisen. Hier muss ausgewählt werden, welche Attribute für das neue Modell übernommen und welche zugunsten des Anderen verworfen werden.

Ein Konflikt tritt ein, falls in beiden Modellen ein gleichwertiger Knoten existiert und dieser für das gleiche Attribut unterschiedliche Werte aufweist.

Definition 4.2.4 (Attribut Widerspruch)
Sind $M_a = (V_a, E_a)$ und $M_b = (V_b, E_b)$ ein Modell und $x \in V_a$ und $y \in V_b$ Knoten. Dann besteht ein Konflikt, wenn sich Attribute widersprechen:

- ***Attribut Widerspruch**: Es gilt $C_{1-1}(x, y)$ mit $Hat\text{-}Attribut_z(x)$ und $Hat\text{-}Attribut_z(y)$ wobei $Attribut_z(x) \neq Attribut_z(y)$*

Der Konflikt sollte automatisch gelöst werden, vorausgesetzt es gelingt eine Herleitung der Lösung aus beiden Modellen oder letztendlich über einen Change-Log. Kann keine Lösung gefunden werden, sollte der Benutzer um eine Entscheidung gebeten werden. Auch dann, wenn es mehrere gleichwertige Lösungen gibt und keine automatische Entscheidung getroffen werden kann. Möchte oder kann der Benutzer jedoch keine Entscheidung treffen, muss eine automatische Lösung nach bestem Bestreben gefunden werden, so dass diese im Zweifel später modifiziert werden kann.

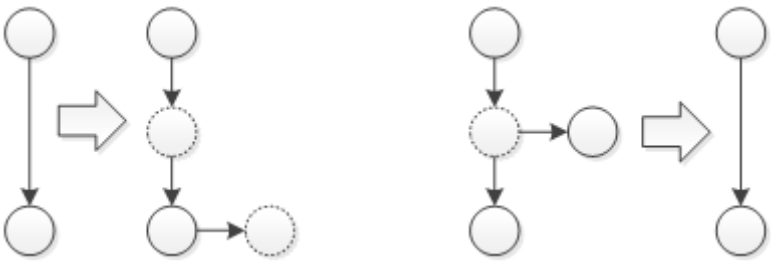
Modellfusion Die beiden Modelle müssen im letzten Schritt kombiniert werden. Dies geschieht mit den gewonnen Informationen aus den vorherigen Schritten. Aus den Informationen werden Regeln hergeleitet, die eine deterministische Fusion gewährleisten.

Definition 4.2.5 (Regel)
Eine Regel ist eine Abbildung $(V_a \cup V_b, E_a \cup E_b) \mapsto (V_n, E_n)$.

Eine Regel kombiniert zwei Graphen zu einem neuen Graphen.

Definition 4.2.6 (Modellfusion)
Sind M_a und M_b Modelle, dann beschreibt die Funktion $Fusion : Modell \times Modell \times Regeln \mapsto Modell$, wie aus den Modellen M_a und M_b und eine Menge von Regeln ein neues Modell entsteht.

Das Fusionieren der Modelle findet mit Hilfe von Graph-Operationen statt. Die Operationen werden im Abschnitt **??** genauer betrachtet. Am Ende wird das fusionierte Modell wieder in das XML-Format gebracht.



(a) hinzufügen **(b)** entfernen

Abbildung 4.4.: Knoten hinzufügen und entfernen

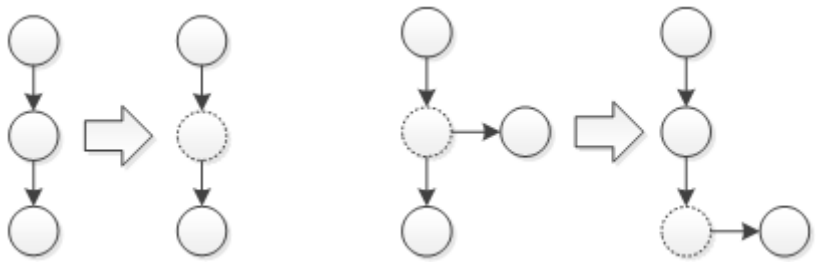
4.3. Änderungsmuster

Als Zwischenformat für die Modelle wurde im vorherigen Abschnitt eine Graph-Repräsentation gewählt. Alle Modifikationen am Modell erfordern daher die Manipulation eines Graphen. Das typische Vorgehen bei Änderungen von Modellen in Graph-Repräsentation wurde von Weber, Reichert und Rinderle-Ma[WRRM08] (siehe Verwandte Arbeiten) analysiert und niedergeschrieben. Es werden aus den so genannten *Adaptation-Patterns* fünf benötigt, um die Änderungen umsetzen zu können. Diese fünf Muster beschreiben, wie Knoten am Graph hinzugefügt, entfernt, verschoben, vertauscht und ersetzt werden können. Diese Operationen werden im Folgenden betrachtet.

Zur Verdeutlichung dienen die Abbildungen 4.4, 4.5, welche die Strukturänderungen eines Graphen zeigen. Auf der linken Seite ist jeweils die vorherige Struktur zu sehen. Auf der rechten Seite ist die resultierende Struktur abgebildet. Die modifizierten Knoten sind mit einem gestricheltem Rand ausgezeichnet. Vertikale Kanten stehen für Knoten in der selben Ebene. Horizontale Kanten signalisieren ein Absteigen in die nächst tiefere Ebene.

Das Hinzufügen eines Elements ist gleichbedeutend mit dem Hinzufügen eines Knoten zum Graphen. In Abbildung 4.4a wird das Einfügen zweier unterschiedlicher Elemente dargestellt. Ein Element soll zwischen zwei bestehenden Elementen eingefügt werden. Ein weiteres Element soll ein bestehendes Element ergänzen. Der erste Knoten wird mit Hilfe der Nachfolger-Beziehung und der zweite Knoten mit Kind-Beziehung eingefügt. Auf der linken Seite besteht der Graph zunächst aus zwei Knoten. Der Knoten mit der Nachfolger-Beziehung wird zwischen die bestehenden Knoten eingefügt. Der zweite Knoten ergänzt den letzten Knoten. Deshalb wird dieser eine Ebene tiefer an den letzten Knoten angehängt.

Das Entfernen eines Elements entspricht dem Entfernen eines Knoten aus dem Graphen. Wird ein Knoten entfernt, werden an diesem Knoten angeheftete Elemente nicht mehr benötigt. Darum müssen alle an den Knoten angehefteten Elemente mit entfernt werden. In Abbildung 4.4b wird ein Knoten der Ebene i entfernt. Mit ihm müssen alle verbundenen Knoten der nächsten Ebene $i + 1$ und größer entfernt werden. Das führt zum kompletten Abschneiden eines Astes am Graphen. Im Beispiel wird somit der angeheftete Knoten mit entfernt und



(a) ersetzen **(b)** verschieben

Abbildung 4.5.: Knoten ersetzen und verschieben

es bleibt ein Graph mit zwei Knoten übrig. Die zurückbleibenden Knoten sind jeweils die Vorgänger und Nachfolger des entfernten Knotens.

Für das Ersetzen eines Elements muss ein Knoten im Graph ersetzt werden. Dies lässt sich über die zwei vorherigen Muster Entfernen und Hinzufügen realisieren. Hierzu wird zunächst der zu ersetzende Knoten entfernt und der neue Knoten an die frei gewordene Position eingefügt.

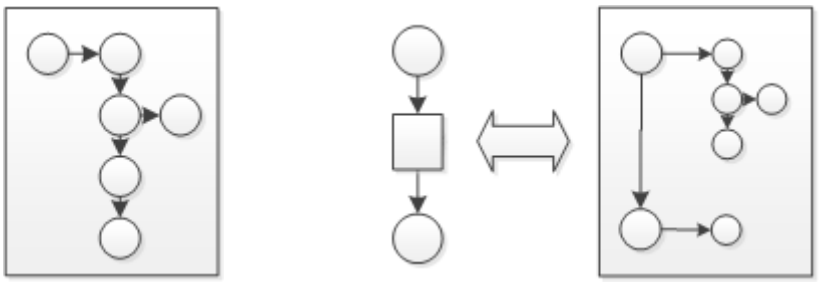
Ein verändertes Attribut (hinzugefügt, entfernt oder ersetzt) spiegelt sich über eine Ersetzung des Knoten im Graphen wieder. Hierzu wird der betroffene Knoten mit einer modifizierten Kopie, welche die veränderten Attribute enthält, ersetzt. In Abbildung 4.5a ist die Ersetzung dargestellt, die Attribute der Knoten sind dabei nicht sichtbar.

Wird ein Element verschoben, muss auch der Knoten im Graphen neu positioniert werden. In Abbildung 4.5b soll der markierte Knoten um eine Stelle verschoben werden. Es ist zu beachten, dass alle verbundenen Knoten der nächsten Ebenen angeheftet bleiben müssen. Der Knoten, welcher verschoben werden soll, wird mit samt seinen angehefteten Knoten ausgewählt und an die neue Position mittels des einfügen Musters kopiert. Danach werden die Knoten an der alten Position entfernt.

Das Tauschen von Elementen bedeutet, dass Knoten im Graphen ihre Position austauschen. Dies wird durch zweimaliges anwenden des verschieben Musters erreicht.

Unabhängig von den oben genannten Änderungsmustern wird zusätzlich die Strategie der Gruppierung eingeführt. Bei der Gruppierung wird ein Teilgraph aus dem Graphen extrahiert. Diese Gruppierung von Knoten erlaubt es einfacher eine Menge von Knoten auszuwählen. Die ausgewählte Menge kann dann eingefügt, entfernt, verschoben oder als Vorlage gespeichert und wiederverwendet werden (vgl. [Sch+11]). Eine Gruppierung lässt sich der Kategorie Fragment und Zweig zuordnen (Abbildung 4.6). Ein Fragment ist eine Gruppierung aus mehreren zusammenhängenden Knoten (vgl. [Sch+11]). Das Fragment ist in Abbildung 4.6b als Rechteck dargestellt.

Durch eine spezielle Gruppierung zu einem Zweig lassen sich alle verbundenen Knoten eines Astes zusammenfassen. Ein Zweig besteht somit aus genau einem Knoten der Ebene i und kann beliebig viele verbundene Knoten in den Ebenen größer i umfassen. Ein Zweig ist in



(a) Zweig **(b)** Fragment

Abbildung 4.6.: Gruppierungen von Knoten

Abbildung 4.6a dargestellt. Ein einzelner Zweig ist ein Fragment, jedoch ist ein Fragment nicht zwangsläufig ein Zweig.

Die Auswirkungen der Änderungen des Modells auf den Graphen wurden betrachtet. Nachfolgend wird analysiert, welche Konsequenzen dies auf die BPEL4Chor Artefakte hat.

4.4. Artefakte

Der Graph ist nicht nur ein geeignetes Zwischenformat für die Manipulation, sondern dieser abstrahiert auch die BPEL4Chor Artefakte mit den darunter liegenden Dateien. Jede Veränderung muss sich in den Dateien widerspiegeln, damit diese persistent gespeichert werden. In diesem Zug muss der Graph zurück in das XML-Format übersetzt werden. Dieser Schritt wird als Serialisieren bezeichnet.

Die BPEL4Chor Dateien haben unterschiedliche Eigenschaften bezüglich der Struktur des Inhalts und deren Darstellungsform innerhalb der grafischen Oberfläche (siehe Kapitel 6). Für die PBDs spielt die Struktur und Reihenfolge des Inhalts eine bedeutende Rolle. Durch sie wird unmittelbar die Semantik des Inhalts mitbestimmt. Für die PGs hingegen ist die Reihenfolge des Inhalts vernachlässigbar, lediglich die Struktur muss berücksichtigt werden.

Dies hat zur Folge, dass die Struktur des Graphen und die Reihenfolge der Knoten für die PBDs exakt übernommen werden müssen. In Abbildung 4.7 ist die Übersetzung von Graph zu XML schematisch dargestellt.

Jeder Knoten wird auf ein XML-Element abgebildet. Die Vorgänger und Nachfolger Beziehungen werden jeweils für Knoten auf derselben Ebene übernommen. Im XML-Dokument teilen die Elemente die gleiche Beziehung. Daraus resultiert, dass die Ebene eines Knoten der direkten Verschachtelungstiefe des XML-Elementes entspricht. Die Ebene 0 korrespondiert folglich mit der Tiefe des XML Wurzel-Elements. Knoten aus der nächsten Ebene werden eine Verschachtelungstiefe weiter, unter den zugehörigen Elternknoten, einsortiert. Hier gilt auch wieder die Vorgänger- und Nachfolger-Beziehung.

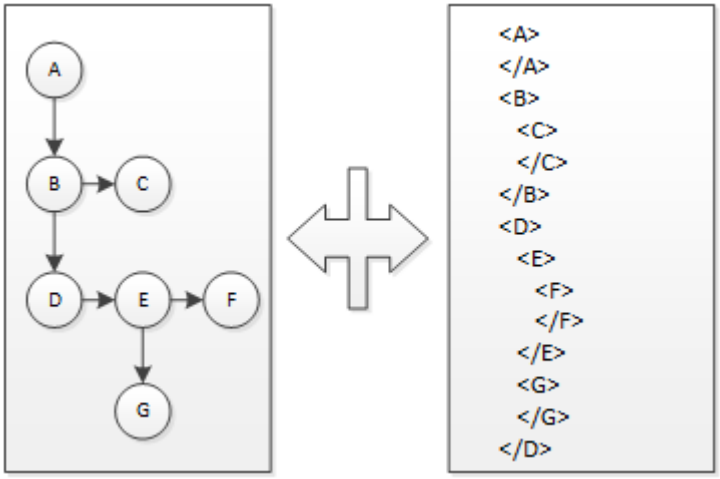


Abbildung 4.7.: Graph zu XML Übersetzung

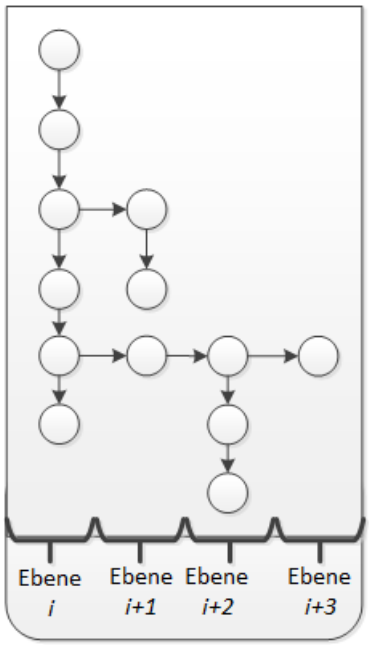


Abbildung 4.8.: Graph mit Ebenen

5. Konzept

In dem vorherigen Kapitel wurden die Anforderungen analysiert. Ferner wurde das allgemeine Vorgehen besprochen und in den Kontext dieser Arbeit einsortiert. Auf dieser Basis wird in diesem Kapitel weiter aufgebaut und ein Konzept zur Realisierung vorgeschlagen.

Es werden zunächst in Abschnitt 5.1 die Anforderungen an die Fusion der Modelle genannt und die getroffenen Annahmen in Abschnitt 5.2 besprochen. Danach werden die verwendeten Regeln für die Modell-Fusion in Abschnitt 5.3 vorgestellt. Daraufhin wird in Abschnitt 5.4 ein Beispiel geliefert, wie Modelle an Hand der Regeln fusioniert werden. Abschließend wird der entworfene Algorithmus vorgestellt.

5.1. Anforderungen

Das Fusionieren von Modellen findet immer unter verschiedensten Bedingungen statt. Die bestehenden Rahmenbedingungen werden vom Anwendungsfall bestimmt. Nachfolgend werden die Anforderungen besprochen, die an das Konzept gestellt werden und für den Anwendungsfall zugeschnitten sind. Die Anforderungen finden sich später in definierten Regeln wieder, die vom Algorithmus verwendet werden.

Änderungsübernahme Alle durchgeführten Änderungen am Choreographie-Modell müssen im fusionierten Modell des Teilnehmers enthalten sein. Bestehende unveränderte Strukturen und Aktivitäten aus der Choreographie sind ebenso zu übernehmen.

Schnittstellen-Charakter Das Choreographie-Modell wird als Schnittstelle angesehen, die eine erfolgreiche Kommunikation der Teilnehmer sicherstellt. Die Reihenfolge von Aktivitäten, welche aus der Choreographie-Beschreibung stammen, darf nicht verändert werden.

Ergänzungsübernahme Im fusionierten Modell müssen die Verfeinerungen des alten Modells enthalten sein. Das fusionierte Modell vereint die neue Choreographie mit den alten Verfeinerungen.

Syntaktisch korrekt Das Ergebnis der Fusion entspricht einem syntaktisch korrekten Modell. Das Modell ist wohlgeformt und lässt sich mit dem Prototyp verwenden.

Eigenständig Für ein erfolgreiches Fusionieren müssen nur die zu verschmelzenden Modelle vorliegen. Dem Teilnehmer kann die neue Choreographie-Beschreibung übermittelt werden. Diese reicht ihm aus um seine neue Orchestrierung zu erstellen.

Unabhängig Jede zusätzliche Information, welche nicht im Modell selbst enthalten ist, muss als optional anzusehen sein. Die Verwendung dieser Information führt zu einer Verbesserung des Resultates, ist aber nicht für eine erfolgreiche Fusion der Modelle erforderlich. Dies umfasst beispielsweise die Hinzunahme einer Änderungshistorie.

Verlustfreies verschmelzen Die Modelle sollen verlustfrei verschmolzen werden. Verlustfrei bedeutet, dass keine Information, welche zuvor in einem Modell vorhanden war, verloren gehen darf. Das umfasst alle bekannten Eigenschaften die Elemente und Attribute betreffen. Daraus folgt, dass jedes Elemente im fusionierten Modell enthalten ist. Sowie, dass jedes zugehörige Attribut eines jeden Elements sich auch im fusionierten Modell wiederfindet.

Automatische Integration Der Algorithmus, zum Fusionieren der Modelle, soll wenig Benutzerinteraktion erfordern. Soweit möglich werden Entscheidungen automatisch getroffen. Dies soll eine nahtlose Integration in den bestehenden Ablauf gewährleisten.

Darüber hinaus werden die üblichen Anforderungen an die Algorithmen gestellt. Die Algorithmen sollen nach endlicher Zeit terminieren und ein Ergebnis liefern. Des Weiteren soll das Ergebnis deterministisch berechnet werden, so dass identische Eingaben immer dasselbe Ergebnis liefert.

5.2. Annahmen

Im Folgenden werden die getroffenen Annahmen vorgestellt.

Schema Evolution Es werden Workflows betrachtet die auf Schema Ebene verändert werden. Somit betrachtet das Konzept Modifikationen am Modell und nicht an abgeleiteten Instanzen.

Träge Weitergabe Eine Änderung am Schema hat keinen Einfluss auf bereits laufende Instanzen. Außerdem erfolgt keine Weitergabe der Änderungen an laufende Prozesse. Diese werden nach Vorgabe des ursprünglichen Modells abgeschlossen. Die Änderungen treten nur für nachfolgend gestartete Instanzen in Kraft.

Lokales Wissen Jeder Choreographie-Teilnehmer verfügt nur über lokales Wissen der Aktivitäten und zusätzlicher Informationen, welche von den Partnern veröffentlicht wurden. Ein Teilnehmer kennt damit nicht zwangsläufig die gesamte Choreographie, sondern nur die Kommunikationsaktivitäten in denen er direkt beteiligt ist.

Ähnlichkeit Die neue Modellversion lässt sich aus der alten Version mit beliebig vielen Änderungsschritten ableiten. Die Versionen haben danach noch mindestens eine Gemeinsamkeit.

Aktivitäten Beziehung Ein Fragment von verfeinerten Aktivitäten hängt unmittelbar von einem eindeutigen Referenzpunkt ab. Es wird angenommen, dass der Referenzpunkt ein vorheriger Knoten ist, der in beiden Modellen existiert. Der Referenzpunkt könnte eine Kommunikationsaktivität der Choreographie sein.

Eine weitere Annahme ist, dass ein neues fusioniertes Modell immer aus zwei Modellen erstellt wird. Außerdem wird angenommen, das ein Zugriff auf die alte Choreographie möglich ist.

5.3. Vergleichen und Verschmelzen

Es werden immer zwei Modelle als Eingabe verwendet. Durch die Annahme, dass eine neue Version aus einer alten Version heraus entstanden ist, lässt sich eine Beziehung zwischen dem alten und neuen Modell definieren, welche die Evolution der Version beschreibt.

Definition 5.3.1 (Evolution der Version)
Ein Modell M' ist eine neuere Version des Modells M, wenn M' aus Änderungen von M entstanden ist und M' noch mindestens ein Knoten von M enthält.

- ***Änderungen***: $M \overset{\Rightarrow^*}{}_{\text{Änderungen}} M'$
- ***Gemeinsamkeit***: $\exists\, x \in V_{M'}, y \in V_M : C_{1-1}(x, y) = \top$

Der Operator $\overset{\Rightarrow}{}_{\text{Änderung}}$ beschreibt eine ausgeführte Änderung an dem Modell M, welche in das geänderte Modell M' resultiert. Unter dem Operator befindet sich die Bezeichnung der Änderung. Die Anwendung einzelner Änderungen können über diesen Operator dargestellt werden. Eine Änderungssequenz, in der ein Knoten einen anderen ersetzen würde, lässt sich wie folgt darstellen: $M \overset{\Rightarrow}{}_{delete(V_1)} M' \overset{\Rightarrow}{}_{insert(V_2)} M''$. Um eine Sequenz abzukürzen wird ein Stern an

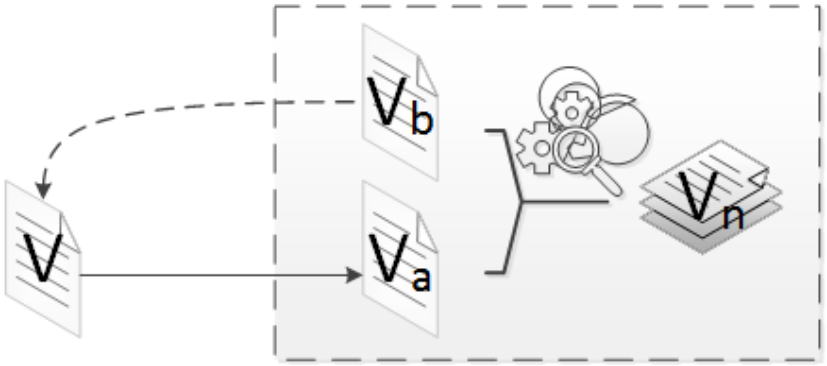


Abbildung 5.1.: Fusionieren zweier Modelle mit Hilfe von Versions-Beziehungen

den Operator angefügt. Mit dem Operator lässt sich auch die Anwendung eines Change-Log, welche eine Sequenz von einzelnen Änderungen umfasst, darstellen.

In der Abbildung 5.1 ist die Beziehung der alten und neuen Choreographie über den gestrichelten Pfeil skizziert. Die Beziehung bezieht sich dabei auf die Version in der ursprünglich noch keine Verfeinerungen durchgeführt worden sind. Aufgrund dieser Beziehung sind in beiden Versionen gemeinsame Knoten enthalten.

Würden die Versionen keine gemeinsamen Knoten teilen, ist die Frage berechtigt, in wie fern eine automatisierte Verfeinerung zweier verschiedener Modelle legitim ist. Zudem müsste auf die Entscheidung eines Menschen zurückgriffen werden oder auf einen lernenden Algorithmus, der über die Syntax hinaus auch die Semantik der Knoten versteht.

Ausgehend von V_b bedeuten alle 1-0 Zuordnungen mit V neu hinzugefügte Aktivitäten zur Choreographie. Alle 0-1 Zuordnung mit V stehen für entfernte Aktivitäten. Diese Zuordnungen werden gespeichert, um später nicht unnötige Aktivitäten, welche in V_a enthalten sind, zu übernehmen.

Jetzt werden die gemeinsamen Knoten zwischen V_a und V_b bestimmt. Die Knoten mit 1-1 Zuordnung dienen als Referenzpunkte, um die verbleibenden Knoten zu positionieren. Die verbleibenden Knoten bestehen nur noch aus Verfeinerungen (Knoten), sowie den gelöschten Aktivitäten. Für jede Verfeinerung wird ein Referenzpunkt ausgewählt. Die Auswahl wird so getroffen, dass die Verfeinerung auf einem Weg vom Referenzpunkt erreichbar und gleichzeitig der kürzeste Weg ist.

Definition 5.3.2 (Referenzpunkt)
Ein Knoten wird als Referenzpunkt bezeichnet, genau dann wenn eine 1-1 Zuordnung besteht und die Knoten Teil der zu fusionierenden Modelle sind.

Die Menge aller Referenzpunkte bilden zusammen ein Skelett. Dieses Skelett bildet das Grundgerüst für das fusionierte Modell. Um das Skelett zu vervollständigen, werden die Verfeinerungen

hinzugefügt. Jede Verfeinerung wird unter den zuvor ausgewählten Referenzpunkt gehängt. Dazu wird ein kompletter Zweig genommen oder ein ganzes Fragment, welches sich zwischen zwei Referenzpunkten befindet.

Das fusionierte Modell besitzt nach diesen Vorgang die vereinigte Struktur beider Modelle. Als nächstes müssen die Attribute der Knoten fusioniert werden. Es kann vorkommen, dass ein Knoten in den Modellen unterschiedliche Attribute besitzt. Hier muss entschieden werden, welcher Wert dem Attribut zugewiesen wird. Dazu wird zunächst für jedes Attribut festgehalten, aus welchen Modell dieses stammt.

Die Modelle erhalten ein Stimmrecht, mit der sie für ihr Attribut wählen dürfen. Bei der Wahl sind alle Modelle wahlberechtigt, welche selbst ein Attribut des betroffenen Namens besitzen. Der Wert mit den meisten Stimmen gewinnt die Wahl und wird in das fusionierte Modell aufgenommen. Bei einem Gleichstand wird ein Entscheidungsträger benötigt. Ein Entscheidungsträger bekommt ein bevorzugtes Wahlrecht und überstimmt damit die anderen Modelle. Als Entscheidungsträger wird das neuste Modell gewählt. Dieses dominiert damit die anderen Modelle.

Definition 5.3.3 (Dominanz)
Ein Modell M' dominiert ein Modell M genau dann wenn M' eine neuere Version von M ist. Die Beziehung wird als $M \prec M'$ geschrieben. Die dominierten Modelle werden als zurückhaltend bezeichnet.

Ein Konflikt wird folglich gelöst, indem das dominante Modell entscheidet. Diese Vorgehensweise ist, für zwei in der Wahl beteiligte Modelle, in der Funktion 5.1 beschrieben. Das dominante Modell stimmt für sein eigenes Attribut, sofern es eins mit dem Namen besitzt. Die Funktion erwartet zwei Knoten und den Namen des Attributs und bildet dies auf das gewählte Attribut ab: $W\ddot{a}hleAttribut : Attribut \times Knoten \times Knoten \mapsto Attribut$.

$$W\ddot{a}hleAttribut(x, V, V') := \begin{cases} Attribut(V', x) & \text{, falls } \textit{Hat-Attribut}_x(V') = \top \\ Attribut(V, x) & \text{, falls } \textit{Hat-Attribut}_x(V') = \bot \end{cases} \tag{5.1}$$

5.4. Beispiel

Bevor der formale Algorithmus vorgestellt wird, werden die einzelnen Schritte an Hand eines kurzen Beispiels erklärt. In Abbildung 5.2 sind dazu drei Graphen zu sehen. Auf der linken Seite befindet sich das neue dominante Modell. Auf der rechten Seite das alte Modell, mit den bestehenden Verfeinerungen. Das Ergebnis aus der Fusion beider Modelle ist in der Mitte abgebildet.

Zur besseren Unterscheidung der Eigenschaften sind die Knoten markiert. Die Knoten mit Verfeinerungen sind mit einem Netz bedeckt. Wohingegen die Knoten mit Kommunikationsaktivitäten, aus der Choreographie Beschreibung, schraffiert dargestellt sind. Gemeinsamkeiten in

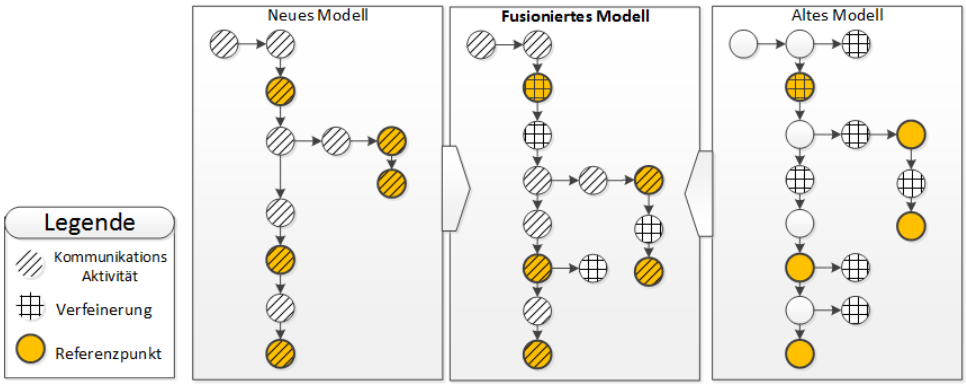


Abbildung 5.2.: Das neue Modell (Links) und altes Modell (Rechts), wird zum fusionierten Modell (Mitte) vereinigt

Form von Referenzpunkten besitzen einen ausgefüllten Knoten. Knoten die keine Markierung haben sind aus dem alten Modell und haben keine Zuordnung. Sie sind aus dem neuen Modell entfernt oder ersetzt worden.

Es wird angenommen, dass die Knoten bereits nach Typ klassifiziert sind. Deshalb wird gleich aus dem neuen Modell ein Skelett erstellt. Dazu werden alle Knoten aus den neuen Modell in den mittleren Graph kopiert. Aus dem alten Modell wird nun das Skelett stückweise ergänzt.

In den kommenden Absätzen wird die Übernahme von Verfeinerungen, in der Reihenfolge von oben nach unten und von links nach rechts, besprochen.

Die oberste Verfeinerung wird nicht in das fusionierte Modell übernommen. Das liegt daran, weil der dazugehörige Referenzpunkt der Wurzelknoten ist und dieser im neuen Modell ersetzt wurde. Die Verfeinerung wird als obsolet betrachtet und nicht ergänzt.

Die nächste Verfeinerung betrifft direkt einen Referenzpunkt. Dieser wird direkt mit den Verfeinerungen ergänzt und in das fusionierte Modell übernommen.

Die dritte Verfeinerung ist ein Kind auf einem Ast. Zwar ist diesem Knoten ein gültiger Referenzpunkt im fusionierten Modell zugeordnet, jedoch befindet sich ein gelöschter Knoten auf dem Weg zu seinem Referenzpunkt. Der gelöschte Knoten muss passiert werden um auf die gleiche Ebene wie der Referenzpunkt zu gelangen. Durch die Annahme der Vorgänger-Beziehung ist davon auszugehen, dass der Knoten vom gelöschten Knoten abhängig ist. Deshalb wird diese Verfeinerung nicht in das Modell übernommen.

Darauffolgend ist eine Verfeinerung unter einem Referenzpunkt, auf der selben Ebene. Dieser besitzt wieder den selben Referenzpunkt, wie die Verfeinerung davor. Der gelöschte Knoten liegt optisch auf dem Weg, allerdings befinden sich Verfeinerung und Referenzpunkt auf der selben Ebene. Die Verfeinerung ist dadurch nicht vom gelöschten Knoten abhängig und wird in das fusionierte Modell übernommen. Es ist dennoch nicht klar, wozu genau der Knoten

gehört. Aus diesem Grund wird er unmittelbar unter den Referenzpunkt, vor den anderen Knoten, angefügt.

Die nächsten beiden Verfeinerungen sind eindeutig. Sie befinden sich unmittelbar an ihrem jeweiligen Referenzpunkt. Diese können ohne weiteres in das Modell ergänzt werden. Für die letzte Verfeinerung trifft erneut der Fall zu, dass ein gelöschter Knoten passiert werden muss, um auf die gleiche Ebene wie der Referenzpunkt zu kommen. Demnach wird dieser nicht in das fusionierte Modell übernommen.

5.5. Algorithmus

Der Algorithmus besteht aus drei Phasen. Die erste Phase ist die Vorbereitungsphase, in der die Vorbedingungen und Annahmen überprüft werden. Darüber hinaus werden die Datenstrukturen initialisiert. Die zweite Phase generiert die neue Struktur. Abschließend wird in der dritten Phase das Modell konsolidiert.

Jeder Partner durchläuft diese drei Phasen, um seinen Workflow zu aktualisieren. Die Schritte, welche in den Phasen umgesetzt werden müssen, sind die folgenden:

1. Vorbereitung
    a) Referenzpunkte werden identifiziert
    b) Hinzugefügte und entfernte Knoten werden im aktualisiertem Modell identifiziert
    c) Verfeinerungen werden aus dem alten Modell identifiziert
2. Strukturierung
    a) Skelett wird aufgebaut
    b) Verfeinerungen werden einem Referenzpunkt zugeordnet
    c) Verfeinerungen werden dem Skelett an den Referenzpunkten angehängt
3. Konsolidierung
    a) Konflikte werden gelöst
    b) Optimierungen werden durchgeführt (optional)

Für die Umsetzung der einzelnen Schritte werden Funktionen in Pseudocode vorgeschlagen. Eine Realisierung kann unter Umständen effizienter implementiert werden. Die Algorithmen dienen nur zur Veranschaulichung des Konzeptes.

Algorithmus 5.1 Referenzpunkte werden identifiziert

```
function findeReferenzpunkte(Modell x, Modell y)
    Referenzpunkte ← ∅
    for all a ∈ gibKnoten(x) do
        for all b ∈ gibKnoten(y) do
            if true == C_{1-1}(a, b) then
                Referenzpunkte ← Referenzpunkte ∪ {(a, b)}
            end if
        end for
    end for
    return Referenzpunkte
end function
```

Referenzpunkte werden identifiziert - Pseudocode 5.1 Für das Identifizieren der Referenzpunkte wird als Eingabe das neue und alte Modell benötigt. Innerhalb der Funktion werden die Knoten auf eine 1-1 Zuordnung untersucht. Sofern eine 1-1 Zuordnung vorliegt wird das Knotenpaar in die Menge der Referenzpunkte aufgenommen. Ist die Untersuchung der Knotenpaare abgeschlossen werden alle gefundenen Referenzpunkte zurückgegeben. Referenzpunkte können selbst von Änderungen, wie Verschiebungen, betroffen sein.

Identifizierung der Aktualisierungen - Pseudocode 5.2 Um Aktualisierungen zu identifizieren wird entweder ein vorhandener Change-Log benötigt oder das ursprüngliche Modell ohne Verfeinerungen. Es wird eine Variante ohne Change-Log vorgeschlagen. Als Eingabe werden zwei Modelle erwartet. Das erstes Argument der Funktion ist das weiterentwickelte Modell. Es ist das neuste unter den Modellen und damit das dominante. Die Zuordnungen werden aus Sicht des ersten Modells erstellt. Die Knoten werden paarweise auf 1-0 und 0-1 Zuordnungen untersucht. Knoten mit 1-0 Zuordnungen werden in die Menge der neu hinzugefügten Knoten aufgenommen. Die Knoten mit 0-1 Zuordnung werden der Menge der gelöschten Knoten zugewiesen. Sind alle Vergleiche vollzogen, werden die Mengen mit den klassifizierten Knoten zurückgegeben.

Identifizierung der Verfeinerungen - Pseudocode 5.3 Aus dem alten Modell werden die Verfeinerungen extrahiert. Eine Variante benötigt dazu das alte Modell, die gefundenen Referenzpunkte und die gelöschten Knoten als Eingabe. Als Alternative kann wieder ein Change-Log verwendet werden. Der Algorithmus geht wie folgt vor: Für das Modell wird jeder Knoten betrachtet und in die Menge der Verfeinerungen aufgenommen, sofern der Knoten kein Referenzpunkt darstellt oder im neuen Modell entfernt wurde. Referenzpunkt sind bereits im Skelett enthalten und gelöschte Knoten sind nicht mehr relevant. Zum Schluss wird die Menge der Verfeinerungen zurückgegeben.

Algorithmus 5.2 Identifizierung der Aktualisierungen

```
function findeHinzugefügteUndEntfernteKnoten(Modell x, Modell y)
                                        // Modell x ist eine neue Version von Modell y
    Aktualisierungen_+ ← ∅                                  // hinzugefügte Knoten
    Aktualisierungen_− ← ∅                                  // entfernte Knoten
    for all a ∈ gibKnoten(x) do
        for all b ∈ gibKnoten(y) do
            if true == C_{1−0}(a, b) then
                Aktualisierungen_+ ← Aktualisierungen_+ ∪ {(a, b)}
            end if
            if true == C_{0−1}(a, b) then
                Aktualisierungen_− ← Aktualisierungen_− ∪ {(a, b)}
            end if
        end for
    end for
    return Aktualisierungen
end function
```

Algorithmus 5.3 Identifizierung der Verfeinerungen

```
function findeVerfeinerungen(Modell x, Referenzpunkte r, Aktualisierungen_a)   //
Finde Verfeinerungen im alten Modell und ignoriere obsolete Knoten
    Verfeinerungen ← ∅
    obsoleteKnoten ← a
    for all a ∈ gibKnoten(x) do
        if a ∉ Referenzpunkte ∧ a ∉ obsoleteKnoten then
            Verfeinerungen ← Verfeinerungen ∪ {a}
        end if
    end for
    return Verfeinerungen
end function
```

Algorithmus 5.4 Finde Referenzpunkt für Knoten

function FINDEREFERENZPUNKT(Knoten x, Modell m, Referenzpunkte r)
    return Referenzpunkte ← BREITENSUCHE(x, m, r) // Traversiere Modell und merke letzten Referenzpunkt
end function

Algorithmus 5.5 Verfeinerungen mit Skelett verbinden

function ERGÄNZEVERFEINERUNGEN(Verfeinerungen v, Modell m)
    verfeinertesModell ← m
    for all $v \in$ Verfeinerungen **do**
        Referenzpunkt ← v.GEHÖRTZUREFERENZPUNKT()
        Position ← verfeinertesModell.GEHEZUREFERENZPUNKT($Referenzpunkt$)
        Position.ERGÄNZEVERFEINERUNG(v)
    end for
    return verfeinertesModell
end function

Finde Referenzpunkt für Knoten - Pseudocode 5.4 Für jeden Wurzelknoten eines Fragments muss ein Referenzpunkt gefunden werden um diesen an das Skelett anzufügen. Die Funktion benötigt dafür den Wurzelknoten für den der Referenzpunkt gefunden werden soll, das Modell in dem der Knoten vorhanden ist und alle verfügbaren Referenzpunkte. Auf dem Modell wird eine modifizierte Breitensuche gestartet die nach dem Wurzelknoten sucht. Auf dem kürzesten Weg zum Knoten wird sich der letzte passierte Referenzpunkt gemerkt. Dieser Referenzpunkt wird dem Wurzelknoten zugeordnet und zurückgegeben.

Verfeinerungen mit Skelett verbinden - Pseudocode 5.5 Das Skelett wird mit den Verfeinerungen ergänzt. Die Funktion erhält als Argument die Menge der Verfeinerungen und das Modell in Form des Skeletts. Für jedes Fragment wird der Referenzpunkt ausgelesen um im Skelett die geeignete Position zu finden. An dieser Position wird das Fragment eingefügt. Nachdem alle Fragmente eingefügt worden sind, wird das verfeinerte Modell zurückgegeben.

Algorithmus 5.6 Fusionieren von Attributen

```
                                                    // Knoten x dominiert Knoten y
function FUSIONIEREATTRIBUTE(Knoten x, Knoten y)
    neuerKnoten ← x
    alterKnoten ← y
    Attribute ← ∅
    for all a ∈ GIBATTRIBUTE(neuerKnoten) do
        Attribute ← Attribute ∪ {a}
    end for
    for all a ∈ GIBATTRIBUTE(alterKnoten) do
        if a ∉ Attribute then
            Attribute ← Attribute ∪ {a}
        end if
    end for
    return Attribute
end function
```

Fusionieren von Attributen - Pseudocode 5.6 Um die Attribute der Knoten zu fusionieren, werden alle Knoten mit 1-1 Zuordnung als Eingabe benötigt. Das erste Argument ist ein Knoten aus dem neuen Modell und dominiert somit den Knoten aus dem zweiten Argument. Zunächst werden alle Attribute vom dominanten Modell übernommen. Danach werden die übrigen Attribute ergänzt, sofern diese nicht bereits bestimmt worden sind. Die Menge der fusionierten Attribute wird zurückgegeben und kann in das Modell übernommen werden. Es müssen nur für Knoten mit 1-1 Zuordnung Attribute fusioniert werden, da nur für diese ein Konflikt auftreten kann.

Algorithmus 5.7 Aktualisierung eines Modells

procedure AKTUALISIEREMODELL(Modell V_a, Modell V_b)
  R ← FINDEREFERENZPUNKTE(V_a, V_b)
  if R $= \emptyset$ **then**
    Abbruch: Modelle haben keine Gemeinsamkeiten
  end if
  Skelett ← V_b // Kopiere gesamtes Modell als Skelett
  A ← FINDEHINZUGEFÜGTEUNDENTFERNTEKNOTEN(V_b, V_a) // Erstes Argument ist neues Modell
  Fragmente ← FINDEVERFEINERUNGEN(V_a, R, A)
  for all $f \in$ Fragmente **do**
    $p \leftarrow$ FINDEREFERENZPUNKT($f.Wurzel, V_a, R$) // Suche ein Referenzpunkt für die Wurzel eines Fragments
    Verfeinerungen ← Verfeinerungen $\cup \{(f, p)\}$
  end for
  fusioniertesModell ← ERGÄNZEVERFEINERUNGEN($Verfeinerungen, Skelett$)
  for all $knoten \in$ GIBKNOTEN($fusioniertesModell$) **do**
    if $knoten \in$ R **then** // Konflikte können nur bei Knoten auftreten die in beiden Modellen vorhanden sind
      $attribute \leftarrow$ FUSIONIEREATTRIBUTE($R.a, R.b$)
      $knoten.attribute \leftarrow attribute$
    end if
  end for
  return fusioniertesModell
end procedure

Aktualisierung eines Modells - Pseudocode 5.7 Durch das Zusammensetzen der einzelnen Funktionen wird die Aktualisierung vorgenommen. Als Eingabe wird das zu aktualisierende alte Modell als erstes Argument und das neue Modell als zweites Argument erwartet. Optional kann ein vorhandener Change-Log zusätzlich mit übergeben werden um das Ergebnis zu verbessern. Im Beispiel wird darauf verzichtet. Zu Beginn werden die Referenzpunkte gesucht. Existieren keine so kann das Modell nicht aktualisiert werden. Dies ist eine Ausnahmesituation, den für gewöhnlich stammen die Modelle voneinander ab und teilen mindestens die Wurzelknoten. Dieser schlechteste Fall würde zu einer einfachen Vereinigung beider Modelle führen. Danach wird das Skelett aus dem neuen Modell gewonnen. Daraufhin werden die Änderungen analysiert und die Verfeinerungen in Form von Fragmenten extrahiert. Für jedes Fragment wird ein geeigneter Referenzpunkt gesucht und zugeordnet. Das fusionierte Modell wird dann aus Skelett und Verfeinerungen aufgebaut. Die Fragmente werden dazu an den Referenzpunkten des Skeletts angefügt. Zuletzt werden die Konflikte gelöst und die gewählten Attribute übernommen. Damit sind beide Modelle in ein Modell fusioniert.

6. Prototyp

In dem vorherigen Kapitel wurde ein Konzept vorgestellt, wie eine überarbeitete Choreographie an die Choreographie Teilnehmer weitergegeben werden kann und diese die Änderungen erkennen und in ihre lokale Orchestrierung übernehmen können.

In diesem Kapitel wird das erarbeitete Konzept realisiert. Hierfür wird der Algorithmus innerhalb eines wissenschaftlichen Prototyps implementiert und Details der Realisierung besprochen.

Zunächst wird der BPEL4Chor Designer in Abschnitt 6.1 vorgestellt und eine kurze Einführung in die Verwendungsweise in Abschnitt 6.2 gegeben. Darauffolgend wird die Architektur des BPEL4Chor Designers in Abschnitt 6.3 betrachtet und erforderliche Änderungen in Abschnitt 6.5 präsentiert. Die durchgeführten Anpassungen werden beschrieben und zuletzt eine Beurteilung in Abschnitt 6.6 abgegeben.

6.1. BPEL4Chor Designer

Der wissenschaftliche Prototyp mit dem Namen BPEL4Chor Designer (kurz ChorDesigner) ist ein Editor für Modellierer. Mit ihm lassen sich Choreographien grafisch modellieren. Aus den modellierten Choreographien lassen sich wiederum ausführbare BPEL Prozesse generieren. Der Editor verwendet für die Transformation das BPEL4Chor Modell, worauf auch der Name zurückzuführen ist.

Im Rahmen der Arbeit [Son13] wurde der Editor ausgearbeitet und wird seitdem in verschiedenen Arbeiten [Sch14], [Wol15], [Wei+15] weiterentwickelt. Der Editor basiert auf dem Eclipse BPEL-Designer[1] und wurde als Eclipse-Plugin konzipiert. Inzwischen umfasst der Prototyp selbst eine Reihe von Erweiterungen. Zum Stand dieser Arbeit wird Eclipse Helios in der Version 3.2.2 in Kombination mit Java 1.5 eingesetzt. Für die Umsetzung des grafischen Editors wird das Graphical Modeling Framework (GMF)[2] verwendet, was wiederum das Eclipse Modeling Framework (EMF)[3] für die Meta-Modellierung der Modelle und das Graphical Editing Framework (GEF)[4] für die Darstellung der Modelle benutzt.

[1] https://projects.eclipse.org/projects/soa.bpel
[2] http://www.eclipse.org/gmf-tooling/
[3] http://www.eclipse.org/modeling/emf/
[4] http://www.eclipse.org/gef/

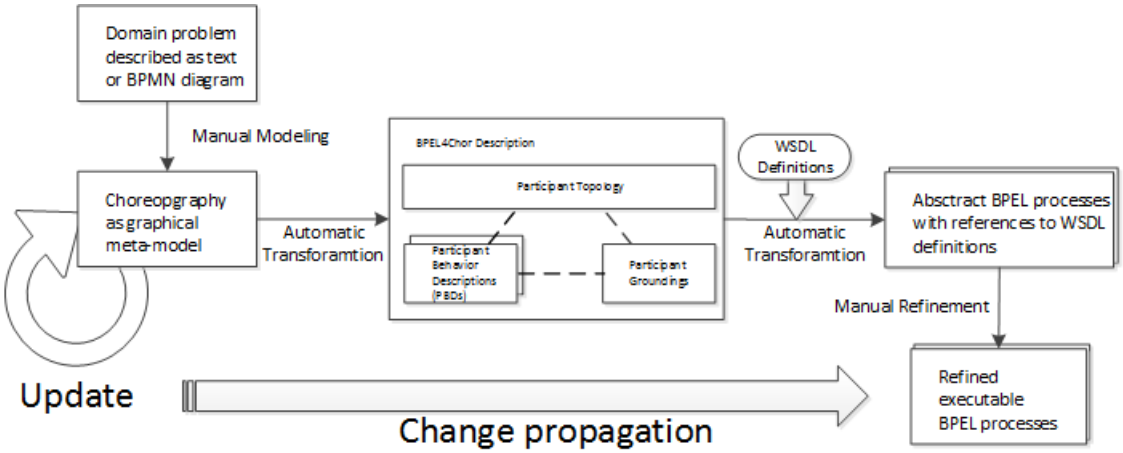


Abbildung 6.1.: Von der Problembeschreibung bis zum ausführbaren BPEL-Prozess, nach [And+13], [Dec+09]

6.2. Verwendung

Bevor das Modellieren beginnt wird angenommen, dass eine Beschreibung des zu lösenden Problems in Form von einem Text oder als BPMN-Diagramm vorliegt. Auf dieser Basis wird das Modell mit dem Choreographie Editor von Hand erstellt. Das passiert mit Hilfe von bereitgestellten Modellierungselementen, die sich durch *Drag and Drop* verbinden lassen.

Das Choreographie-Modell liegt nun in einer Choreographie-Datei (.chor) vor und kann exportiert werden. Beim Exportieren werden automatisch die BPEL4Chor Artefakte aus dem Modell generiert. Es entstehen eine Topologie-Datei (.xml) und für jeden Chorographie-Teilnehmer jeweils eine Deployment-Datei (deploy.xml) und ein abstrakter BPEL-Prozess. Aus den BPEL4Chor Artefakten werden abschließend automatisch die Webservice Beschreibungen (.wsdl) und ausführbaren BPEL-Prozesse (.bpel) erstellt. Die ausführbaren BPEL-Prozesse werden daraufhin wieder manuell mit Geschäftslogik verfeinert.

Wird das bestehende Choreographie-Modell verändert, werden die einzelnen Schritte der Transformation neu durchlaufen und es entstehen neue ausführbarer BPEL-Prozesse, die erneut verfeinert werden müssen. Durch die Implementierung der vorgestellten Algorithmen wird es ermöglicht, diesen Ablauf zu vereinfachen und Änderungen in der Choreographie automatisch in vorhandene verfeinerte Prozesse zu integrieren (Abbildung 6.1). Eine Modifikationen an der Choreographie wird so direkt an die betroffenen Prozesse weitergegeben und automatisch integriert.

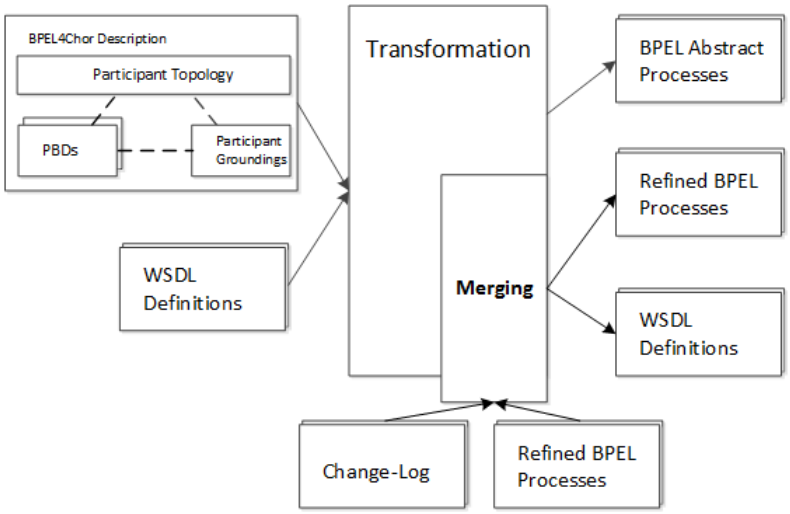


Abbildung 6.2.: Erweiterung um ein Merging-Modul

6.3. Architektur

Die bestehende Architektur des Editors wird um ein neues Modul, innerhalb der Transformation, erweitert (Abbildung 6.2). Das Modul kann eigenständig verwendet werden und übernimmt alle Aufgaben, welche für das Verschmelzen der Modelle erforderlich sind. Das Merging-Modul wird für jene Modelle ausgeführt, für die bereits ein verfeinertes Modell existiert. Auch werden nur Modelle fusioniert die von einer Änderung betroffen sind. Dies simuliert gleichzeitig die Weitergabe der BPEL4Chor Beschreibung an betroffene Teilnehmer. Modelle ohne Verfeinerung werden von der Transformation wie vorher behandelt.

Es wurde sich gegen das direkte Bearbeiten des Transformations-Moduls entschieden. Innerhalb des Transformations-Moduls wären mehr Kontext-Informationen vorhanden, da die gesamte Choreographie bekannt ist. Ein direktes bearbeiten des Tranformations-Moduls hat jedoch entscheidende Nachteile. Beim damaligen Schreiben des bestehenden Codes wurde kein Augenmerk auf Wartbarkeit gelegt und Änderungen würden sich durch die Codebasis propagieren. Darüber hinaus ist es fraglich, ob das Integrieren eines bestehenden Modells zu den Aufgaben des Transformations-Moduls zählen sollte. Wird das Prinzip der *Separation of Concerns* (SoC) berücksichtigt, dass bedeutet das strikte Trennen von Zuständigkeiten, wird klar, dass eine Modellfusion kein trivialen Bestandteil der Transformation darstellt und daher ausgelagert werden muss. Es ist zudem realistischer, dass kein globales Wissen über die Choreographie vorhanden ist und nur vom Partner mitgeteilte, sowie lokale Informationen in die Bearbeitung mit einbezogen werden können. Ein weiteres Argument für die die Bereitstellung eines neuen Moduls ist, dass sich die Funktionalität einerseits direkt in die automatische Codegenerierungsphase integrieren und andererseits als eigenständiges Plugin ansprechen und verwenden lässt.

Es wurde bei der Modulerstellung darauf geachtet, dass Komponenten leicht ersetzt werden können. Das erhöht die Wartbarkeit und ermöglicht jederzeit eine Verbesserung leicht umzusetzen. So kann beispielsweise bei der Zuordnung der Modellelemente die verwendete Strategie ausgetauscht oder der konkrete Algorithmus, der die Modelle verschmelzt, ersetzt werden.

6.4. Eigenschaften

Der Editor enthält zwei hilfreiche Funktionen, die Verschmelzungen der Modelle begünstigen. Es wird bereits eine Änderungshistorie in Form eines Change-Log bereitgestellt. Außerdem wird jede aus der Choreographie transformierte Orchestrierung in einem neuen Ordner abgelegt, der mit einer Versionsnummer gekennzeichnet ist.

Bei der Bearbeitung des Choreographie-Modells wird automatisch ein Change-Log angelegt. Der Inhalt des Change-Log enthält jedes Element, welches während des Bearbeitens angeklickt worden ist. Ein Element wird auch dann in den Log aufgenommen, wenn keine Änderung vorgenommen wurde. Dadurch wächst der Change-Log sehr schnell und enthält unnötige Informationen.

Jedem Eintrag im Change-Log wird eine Änderungskategorie zugeordnet. Aktuell existieren zwei Kategorien. Die eine Kategorie umfasst strukturelle Aspekte und gruppiert damit Änderungen wie Hinzufügen und Entfernen von Elementen. Wird ein Element verschoben, werden die Operationen in die Änderungs-Primitiven Hinzufügen und Entfernen zerlegt und jeweils separat im Change-Log aufgeführt. Die andere Kategorie umfasst Änderungen an den Eigenschaften der Elemente. Dazu zählen Attribute und Werte, welche hinzugefügt, entfernt oder geändert werden. Zusätzlich werden für jeden Eintrag immer der Zeitpunkt, der Name und die aktuelle Position des betroffenen Objekts festgehalten.

6.5. Anpassungen

Unabhängig vom neu hinzugefügten Modul mussten am bestehenden Code Änderungen vorgenommen werden. Davon betroffen war die Änderungshistorie und der Transformation-Assistent. Mit Hilfe des Transformations-Assistenten lassen sich die Einstellungen konfigurieren. Hier wird zum Beispiel der Speicherort des Resultats angegeben oder die Optionen für den Algorithmus bestimmt.

Zunächst wurden die zu speichernden Details der Änderungshistorie angepasst. Ab sofort wird für jeden Change-Log-Eintrag zusätzlich die Identifikationsnummer des Elements gespeichert. Dadurch ist es möglich, direkt nach Elementen zu suchen, ohne die komplette Historie nachvollziehen zu müssen.

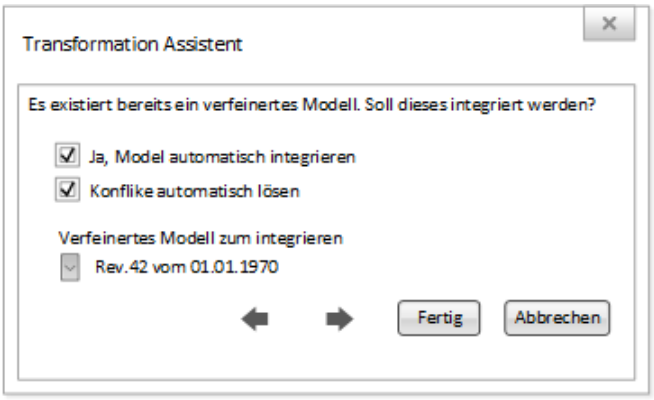


Abbildung 6.3.: Optionsseite für den Transformation Assistenten

Danach wurde der existierende Transformation-Assistent um eine weitere Optionsseite erweitert (Abbildung 6.3). Auf der zusätzlichen Seite lässt sich das automatische Integrieren ausschalten. Es kann darüber hinaus die Modell-Version bestimmt werden, welche für die Verfeinerung verwendet wird. Außerdem kann entschieden werden, ob eine manuelle oder automatische Konfliktlösung bevorzugt ist.

Der Transformationsprozess wird wie vor den Änderungen angestoßen und abgearbeitet. Sobald die Transformation abgeschlossen und die Modell-Integration im Assistenten ausgewählt ist, wird das Merging-Modul ausgeführt. Dieses holt sich die betroffenen Dateien und führt die gewünschten Aktionen aus. Das verfeinerte Modell wird in einem Ordner innerhalb der aktuellen Modell-Version abgelegt.

6.6. Beurteilung

Die Erweiterung des BPEL4Chor Designers bringt folgende Vorteile mit sich:

Die neue Funktionalität wird nahtlos im Editor zur Verfügung gestellt, ohne den bekannten Ablauf zu unterbrechen. Deshalb kann der Benutzer den Editor wie vor den Anpassungen verwenden und wird nicht seines gewohnten Ablaufes beraubt.

Es bleiben alle Modell-Versionen erhalten und werden sortiert in der Ordnerstruktur abgelegt. So können die Modelle auch später noch einfach verglichen werden. Darüber hinaus kann jede alte Version zum Verfeinern des neusten Modells herangezogen werden und nicht nur die Letzte.

Das automatische Verfeinern bereits existierender Modelle spart wertvolle Arbeitszeit, die vorher in das Kopieren und Einfügen der fehlenden Elemente von Hand investiert werden musste. Gleichzeitig werden Fehler in der Positionierung vermieden.

7. Implementierung

Im vorherigen Kapitel wurde der wissenschaftliche Prototyp präsentiert und die notwendigen Anpassungen vorgestellt. In diesem Kapitel wird das Konzept und die entworfenen Algorithmen implementiert. Als Programmiersprache wird Java verwendet.

Zunächst werden in Abschnitt 7.1 die erstellten Komponenten und ihre Beziehungen präsentiert. Anschließend werden verwendete Bibliotheken und Techniken in Abschnitt 7.2 vorgestellt. Danach wird in Abschnitt 7.3 gezeigt, wie die Komponenten in den Prototypen integriert wurden. Außerdem werden getroffene Entscheidungen des beschrittenen Implementierungswegs diskutiert.

7.1. Komponenten

In Abbildung 7.1 sind die implementierten Komponenten und ihre Beziehung untereinander dargestellt. Der Kern wird aus dem Fragment-Sucher, Fragment-Verbinder und der Modellanalyse gebildet. Die Komponenten sind eigenständig und können unabhängig voneinander verwendet werden. Der Modell-Verschmelzer kombiniert diese Bausteine, um zwei Modelle zu fusionieren. Die Modelle implementieren das *Document*-Interface, welches aus dem *org.w3c.dom* Paket stammt und zur Java API für XML Verarbeitung (JAXP) gehört.

Modellanalyse Die Modellanalyse-Komponente realisiert das Konzept der Zuordnung. Ihre Aufgabe besteht darin die Modelle zu untersuchen und 1-1 Zuordnungen, sowie 1-0 und 0-1 Zuordnungen, aus den Modellen herauszuarbeiten. Die gefundenen Zuordnungen werden dann in geeigneten Datenstrukturen bereitgestellt.

Die Komponente erwartet als Eingabe die zu untersuchenden Modelle. Die Modelle werden daraufhin analysiert. Dazu müssen die DOM-Knoten verglichen werden. Beim Vergleichen der DOM-Knoten wird auf verschiedene Vergleichsstrategien zurückgegriffen. Eine Vergleichsstrategie stellt hierfür eine Schnittstelle zur Verfügung. Die Schnittstelle erlaubt es zwei Elemente hinsichtlich ihrer Gleichwertigkeit (Listing 7.1) zu beurteilen. Die gewünschte Logik wird in konkrete Vergleichsstrategien implementiert.

Aktuell werden zwei Vergleichsstrategien angeboten. Zum einen bietet eine Strategie den Vergleich über die ID an. Hierbei werden die Knoten auf die Existenz einer ID inspiziert.

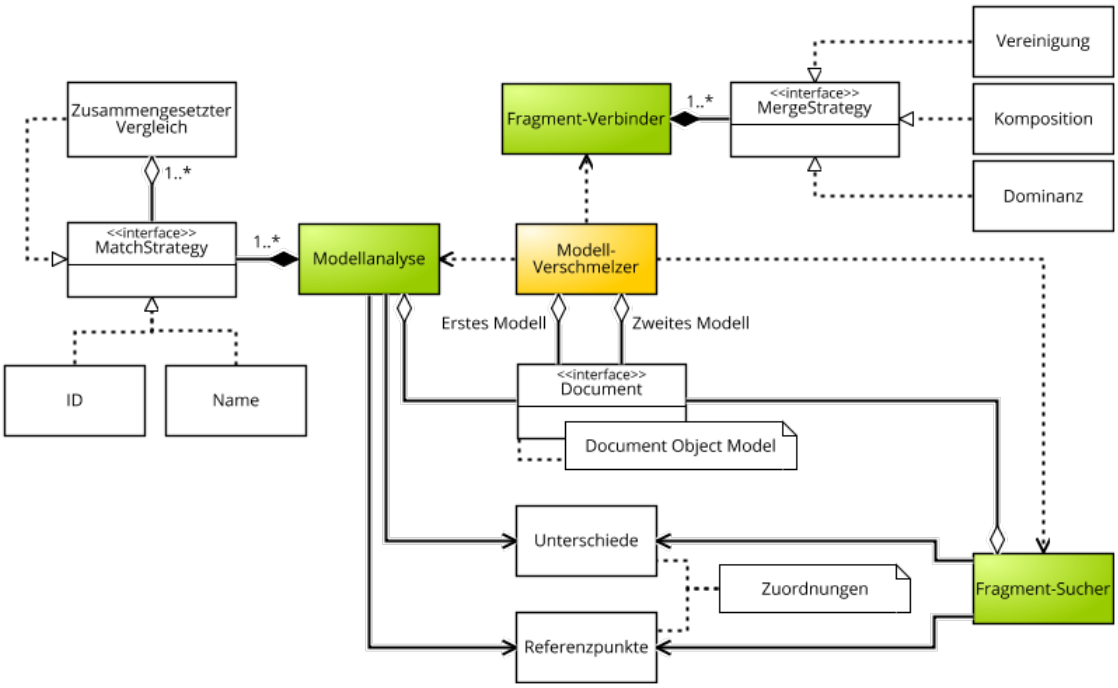


Abbildung 7.1.: Komponentendiagramm der Implementierung

Listing 7.1 Match-Strategy Interface

```
1 interface MatchStrategy {
2    public Boolean match(Element left, Element right);
3 }
```

Besitzen beide Knoten eine ID und stimmt diese ID gleichzeitig überein, dann wird die Gleichwertigkeit der Knoten mit einem *true* bestätigt. Als Alternative wird ein Vergleich über den Namen angeboten. Hierzu wird der exakte Namen beider Knoten verglichen. Im Falle einer Übereinstimmung des Namens wird mit einem *true* bestätigt, andernfalls mit einem *false* verneint.

Da dies unter Umständen nicht ausreichend ist, wird außerdem ein zusammengesetzter Vergleich angeboten. In diesem können beliebig viele Strategien kombiniert werden. Ein zusammengesetzter Vergleich kann sogar wieder einen zusammengesetzten Vergleich beinhalten. So lassen sich komplexe Vergleiche aus einfachen Strategien kombinieren.

Mit diesem Konstrukt kann zum Beispiel erst die ID überprüft werden und falls keine ID vorhanden ist, doch noch der Name zur Entscheidung hinzugezogen werden. Darüber hinaus können in der Zukunft einfach neue Vergleichsstrategien ergänzt werden.

Listing 7.2 Merge-Strategy Interface

```
interface MergeStrategy {
   public Fragment merge(Fragment left, Fragment right);
}
```

Nach abgeschlossener Analyse werden die gewonnenen Erkenntnisse in Datenstrukturen bereitgestellt. Gleichwertige Knoten werden dabei als Referenzpunkte markiert und die gescheiterten Vergleiche entsprechend als Unterschiede.

Fragment-Sucher Die Aufgabe des Fragment-Suchers ist es, ein Modell nach Fragmenten abzusuchen, die später in ein anderes Modell eingefügt werden sollen. In erster Linie wird hier gezielt nach Verfeinerungen gesucht. Die gefundenen Fragmente werden dann bereitgestellt.

Als Eingabe wird das Modell erwartet, das durchsucht werden soll. Zusätzlich werden weitere Informationen benötigt, die es erlauben die Knoten zu unterscheiden. Dazu werden die Datenstrukturen mit den Änderungen und Referenzpunkten herangezogen, welche zuvor von der Modellanalyse gefunden wurden. Diese weiteren Informationen werden als Filter verwendet, um gezielt nach den Fragmenten suchen zu können. Für die Suche werden herkömmliche Traversierungs-Algorithmen auf den Graphen angewandt, darunter vorrangig die Breitensuche.

Für jedes gefundene Fragment wird zusätzlich der nächste Referenzpunkt bestimmt. Der Referenzpunkt mit dem kürzesten Weg zum Fragment ist am nächsten. Die Position des Fragments lässt sich so relativ zum Referenzpunkt festlegen. Gehören zu einem Referenzpunkt mehrere Fragmente, werden die Fragmente zusätzlich mit einer Nummerierung versehen. Die Nummerierung legt die absolute Reihenfolge der Fragmente fest. Die Nummer wird an Hand des tatsächlichen Abstands vom Referenzpunkt zum Fragment bestimmt.

Fragment-Verbinder Mit Hilfe des Fragment-Verbinders können zwei Fragmente miteinander verbunden werden. Die Regeln, die beschreiben wie Fragmente transformiert und zusammengefügt werden, finden hier ihren Platz. Für gewöhnlich wird jeweils das Modell-Skelett mit einem Fragment zusammengeführt.

Der Fragment-Verbinder benötigt zwei Fragmente als Eingabe. Hier ist zu beachten, dass ein Fragment auch aus nur einem einzelnen Knoten bestehen darf. Es können also sowohl ganze Teilbäume, als auch einzelne Knoten verbunden werden. Die Entscheidung wie zwei Fragmente verbunden werden, wird über eine Fusionsstrategie getroffen. Eine Fusionsstrategie bietet über die Schnittstelle die Möglichkeit zwei Fragmente zusammenzuführen (Listing 7.2).

Es sind drei Fusionsstrategien implementiert. Bei der Strategie der Vereinigung wird ein Knoten aus einem Fragment ausgewählt. Der ausgewählte Knoten wird typischerweise an Hand des Referenzpunktes bestimmt, der dem Fragment zuvor zugeordnet wurde. Unter diesem

Referenzpunkt wird dann das andere Fragment, auf der gleichen Ebene, angefügt. Aus Sicht des XML-Dokuments wären die Elemente jetzt Nachbarn.

Bei der Strategie der Komposition wird das Fragment als Kind angehängt. Allerdings eine Ebene höher wie der ursprüngliche Knoten. Dies würde einer weiteren Verschachtelungstiefe im XML-Dokument entsprechen. Der Inhalt würde von einem Element umschlossen sein.

Die Strategie der Dominanz (siehe Definition 5.3.3) ergänzt das dominante Fragment mit den zusätzlichen Informationen des anderen Fragments. Das dominante Modell ist in der Regel das Skelett, welches sich durch Verfeinerungen vervollständigt.

Durch die Bereitstellung des Interfaces können auch hier sehr einfach neue Strategien entwickelt und das Verbinden weiter optimiert werden.

Modell-Verschmelzer Der Modell-Verschmelzer bringt alle Komponenten zusammen und koordiniert das Verschmelzen der Modelle. Er dient außerdem als Einstiegspunkt für das Eclipse-Plugin. Als Eingabe werden die zu fusionierenden Modelle benötigt. Des weiteren muss eine Konfiguration zur Verfügung gestellt werden. Näheres dazu wird in Abschnitt 7.3 beschrieben.

Zunächst werden die Modelle über die Methode `normalizeDocument()` normalisiert. Dies ist unbedingt erforderlich, damit der Graph keine unterschiedlichen Repräsentationen besitzt und somit eindeutig ist. Deshalb muss diese Methode vor und nach der vollständigen Modifizierung ausgeführt werden.

Die Modellanalyse und die Fragmentsuche wird gestartet. Anschließend wird aus dem neueren Modell das Skelett gebaut und schließlich mit dem Fragment-Verbinder vervollständigt. Das fertig fusionierte Modell wird zum Schluss wieder serialisiert und persistent gespeichert.

7.2. JDOM Bibliothek

Mit JDOM kann ein XML-Dokument gelesen, geschrieben, erstellt oder modifiziert werden. Für die Bearbeitung des DOM-Baums wird nicht die JAXP Programmierschnittstelle, sondern JDOM[1] in der Version 2.0.6 verwendet. Der Programmcode von JDOM steht unter einer angepassten Apache Lizenz[2]. Diese Lizenz ist mit Eclipse-Plugins kompatibel[3] und darf somit verwendet werden.

JDOM selbst bringt keine Möglichkeit mit ein Dokument als DOM einzulesen. Deshalb muss das Dokument bereits geparst vorliegen. Zum Parsen kann der DOM-Parser benutzt werden

[1] http://www.jdom.org/
[2] http://jdom.org/docs/faq.html#a0030
[3] https://eclipse.org/legal/eplfaq.php#3RDPARTY

Listing 7.3 Laden eins Modells mit dem DocumentBuilder

```
1 public org.w3c.dom.Document FileToDocument(File file) {
2    DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory
3          .newInstance();
4    DocumentBuilder documentBuilder =
        documentBuilderFactory.newDocumentBuilder();
5    return document = documentBuilder.parse(file);
6 }
```

Listing 7.4 Konvertierung eines DOM-Baums zu einem JDOM-Baum

```
1 public org.jdom2.Document convertDOM2JDOM(org.w3c.dom.Document document) {
2       DOMBuilder builder = new DOMBuilder();
3       return builder.build(document);
4 }
```

(Listing 7.3). Es ist jedoch auch möglich das Dokument mit dem SAX- oder StAX-Parser einzulesen.

Der eingelesene DOM-Baum wird dann in einen JDOM-Baum konvertiert (Listing 7.4). Eine Konvertierung zurück in einen DOM-Baum ist ebenfalls einfach möglich (Listing 7.5). In den Listings ist der vollständige Name der Klassen angegeben, um die Zugehörigkeit der Domäne zu verdeutlichen.

Der Hauptunterschied zwischen DOM und JDOM ist, dass ein XML-Knoten nicht als *DOM Node*, sondern via Java-Objekt repräsentiert wird. Wie auch bei DOM, wird der gesamte Graph im Hauptspeicher gehalten.

Ein Vorteil von JDOM gegenüber DOM ist, dass dieses extra für Java entwickelt wurde und daher komfortabel zu benutzen ist. Gleichzeitig bringt es neue Funktionen mit, die das Bearbeiten und Durchsuchen des Baums erleichtern. So erlaubt es JDOM, Filter zu erstellen, welche über die Knoten des Baums gelegt werden können. Die Filter unterstützen nach Name, Typ, Wert oder nach anderen Parametern zu selektieren. Zusätzlich können diese mit Logik-Operatoren ($\wedge$, $\vee$, $\neg$) verknüpft werden.

Listing 7.5 Konvertierung eines JDOM-Baums zu einem DOM-Baum

```
1 public org.w3c.dom.Document convertJDOM2DOM(org.jdom2.Document document)
     throws JDOMException {
2       DOMOutputter outputter = new DOMOutputter();
3       return outputter.output(document);
4 }
```

Darüber hinaus wird die XML Path Language (XPath) unterstützt, die das Abfragen des Baumes erleichtert. Sowie ein Paket für XSL Transformation (XSLT), mit der ein XML-Dokument über eine formale Beschreibung transformiert werden kann.

7.3. Integrierung in den Prototyp

Das fertige Modul wird schließlich in den Prototyp integriert. Dieses muss als weiterer Schritt nach der Transformation ausgeführt werden. Dafür wird zunächst für den geschriebenen Code ein neues Paket erstellt. Der Modell-Verschmelzer und seine Eingabe Konfiguration werden initialisiert. Anschließend wird der Modell-Verschmelzer im Transformation-Modul eingefügt.

In Listing 7.6 sind die erforderlichen Änderungen am Prototypen zu sehen. Zu erst wird eine Konfiguration erstellt, die Optionen für die Transformation und die Modell-Integrierung beinhaltet. Die Optionen können über die grafische Oberfläche gesetzt werden. Dies passiert über den in Abschnitt 6.5 gezeigten Transformations-Assistenten. Um die neuen Optionen im Assistenten einzufügen, wurde der alte Assistent (`TransformationWizardOptions`) abgeleitet und ein neuer Assistent erstellt. Der `AdvancedTransformationWizard` erweitert den Assistenten um eine weitere Optionsseite.

Nachdem die Transformation abgeschlossen ist wird überprüft, ob der Benutzer eine automatische Integration des neuen Modells wünscht. Dieses Vorgehen ist standardmäßig eingeschaltet. Dann werden die gewählten Optionen, für die Verwendung innerhalb des Merge-Moduls, aufbereitet. Das Merge-Modul erhält die Eingabe und führt die notwendigen Schritte aus. Nach Fertigstellung der Integration wird wie gehabt fortgesetzt.

Durch diese Art der Implementierung lässt sich das Modul sowohl innerhalb des Prototypen, als auch eigenständig als Anwendung verwenden. Es wäre außerdem denkbar das Modul beispielsweise als Kommandozeilenprogramm bereitzustellen. Dafür müssten lediglich die Argumente in das Konfigurationsobjekt gebracht werden.

Listing 7.6 Ergänzungen im existierenden TransformChoreographyHandler

```
1  //[...]
2  // Explizite Typumwandlung zur Verdeutlichung
3  Configuration options = new MergeConfiguration();
4  ((MergeConfiguration) options).setWorkspaceDirectory(/*...*/);
5  ((MergeConfiguration) options).setValidRevisionOptions(/*...*/);
6  ((MergeConfiguration) options).setLatestRevisionAsDefaultRevision();
7  //[...]
8  WizardDialog wizardDialog = new WizardDialog(/*...*/, new
       AdvancedTransformationWizard(options));
9  //[...]
10 Transformer handler = new Transformer((TransformationWizardOptions)
       options);
11 handler.transform(result, chorEditor);
12 //[...]
13 if (((MergeConfiguration) options).isAutoIntegrationEnabled()) {
14    MergerInput input = new MergerInput((MergeConfiguration)options);
15    input.setChoreographieName(result.getChoreographyName().getLocalPart());
16
17    ModelMerger merger = new ModelMerger(input);
18    merger.process();
19 }
20 //[...]
```

EMF Compare

Innerhalb des Prototyps wird das Eclipse Modelling Framework verwendet, um die Modelle abzubilden. Zu dem EMF gehört darüber hinaus das EMF Compare Projekt. Das EMF Compare Projekt erlaubt es, mit EMF erstellte Modelle (.ecore Dateien) zu vergleichen und zu vereinen.

Für eigene Modelle bietet EMF Compare eine Grundstruktur (Abbildung 7.2) zum Durchführen von Modell-Vergleichen und zur Modell-Integration an. Die Rollen der Komponenten müssen mit selbst erstellten Klassen besetzt werden. Leider ist die Entwickler-Dokumentation von EMF Compare lückenhaft und weist besonders Defizite im Abschnitt „Modelle verschmelzen“ auf[4]. Dies macht EMF Compare schwer zugänglich. Nach einer ausgiebigen Recherche wurde der weitere zeitliche Aufwand als zu hoch eingestuft und mit einer eigenen unabhängigen Implementierung begonnen.

Die Struktur ist der in dieser Arbeit verwendeten sehr ähnlich. Jedoch ist bei EMF Compare hauptsächlich eine Interaktion mit dem Benutzer vorgesehen. Dieser muss jeden Unterschied im Modell über eine Vergleichsansicht selbst auf Richtigkeit überprüfen und via Bestätigung in das andere Modell übernehmen. EMF Compare ist für Schrittweise Integration von Unterschieden konzipiert. Dies ist im vorliegenden Konzept nicht gewünscht. Die Modelle sollen für den Benutzer automatisch vereinigt werden, außer der Benutzer wünscht dies explizit.

Weitere Punkte sind, dass EMF-Compare sich auf strukturierte Diagramme, wie z.B. Klassendiagramme, fokussiert und nicht auf Graphen-basierte Modelle, wie es Prozessmodelle sind [GKE09]. Es wird auch kein Modell-Strukturbaum verwendet [GKE09]. Vergleiche werden Knotenweise durchgeführt ohne die Umgebung zu berücksichtigen, was Abhängigkeiten zu anderen Knoten missachtet. Das führt dazu, dass keine Änderungsoperationen, sondern Änderungsprimitiven unterstützt werden, welche ungeeignet für Prozessmodell-Änderungsmanagement sind [KGE09]. Aus diesen Gründen wurde von einer Implementierung mit Hilfe von EMF Compare abgesehen.

[4] http://www.eclipse.org/emf/compare/documentation/latest/developer/developer-guide.html, Stand: 31.03.2017

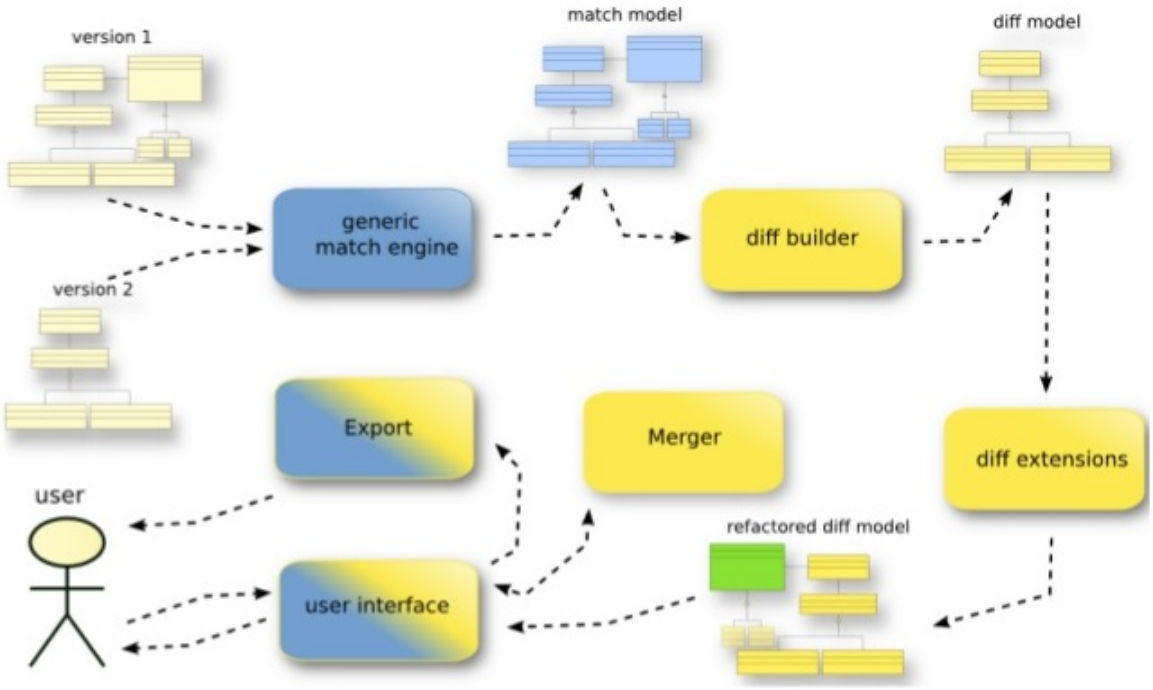


Abbildung 7.2.: EMF Compare Funktionsweisen, nach [Bru11]

8. Evaluierung

Im vorherigen Kapitel wurden Details der Implementierung vorgestellt. In diesem Kapitel wird das Konzept und die Implementierung kritisch untersucht.

Wie gut das vorgestellte Konzept in der Praxis funktioniert, wird in den kommenden Abschnitten gezeigt. Zunächst werden in Abschnitt 8.1 die Anforderungen betrachtet und gezeigt, wie diese sichergestellt wurden. Danach wird in Abschnitt 8.2 präsentiert, dass eine durchgeführte Änderung am Modell übernommen wird. Zuletzt werden in Abschnitt 8.3 die Schwächen des Konzeptes aufgezeigt und eine Laufzeitabschätzung in Abschnitt 8.4 abgegeben.

8.1. Anforderungen

Das fusionierte Modell muss die Aktualisierungen der Choreographie beinhalten. Das wird sichergestellt, indem der Teilnehmer seine neu generierte Orchestrierung aus der Choreographie erhält. Die generierte Orchestrierung beinhaltet die Änderungen der Choreographie. Aus der generierten Orchestrierung wird das Skelett, an dem später die Verfeinerungen angefügt werden, gebildet. Das Skelett ist demnach eine einfache Kopie der neuen Orchestrierung und enthält damit alle Aktualisierungen. Durch dieses Vorgehen sind außerdem Unterscheidungen der vorgenommen Änderungen, wie Einfügungen, Löschungen und Verschiebungen, nicht notwendig. Sie sind bereits im vorliegenden Skelett abgebildet.

An dem Skelett darf die Reihenfolge der initialen Elemente nicht verändert werden. Dies könnte einerseits die Aktualisierungen rückgängig machen oder andererseits die Abfolge der Kommunikationsaktivitäten zerstören. Deshalb werden nur Einfügungen am Skelett vorgenommen. Durch Einfügungen werden die Vorgänger- und Nachfolger-Beziehungen der Kommunikationsaktivitäten nicht verändert.

Die Einfügungen am Skelett bestehen ausschließlich aus Verfeinerungen des alten Modells. Es werden keine zusätzlichen Elemente eingefügt. Die alten Kommunikationsaktivitäten werden ignoriert und dienen nur zur relativen Positionierung. So wird nicht ungewollt eine alte gelöschte Kommunikationsaktivität in das fusionierte Modell eingebracht.

Durch die Manipulation des Skeletts kann kein syntaktisch inkorrektes Modell entstehen. Die verwendete Bibliothek JDOM stell zu jeder Zeit sicher, dass ein darunterliegendes XML-

Dokument wohlgeformt ist[1]. Es kann somit kein Modell erzeugt werden, welches syntaktisch inkorrekt ist. Darüber hinaus wird der generierte XML-Code automatisch leserlich formatiert.

Die präsentierten Algorithmen erwarten jeweils die zu fusionierenden Modelle als Eingabe. Aus diesen werden die Gemeinsamkeiten und Unterschiede ausgewertet. Es werden nur die Unterschiede in das fusionierte Modell eingearbeitet, welche einen Referenzpunkte in der neuen Version haben. Die Positionierung geschieht relativ zu den bestimmten Referenzpunkten, welche aus der Analyse von Gemeinsamkeiten und Unterschieden gewonnen wurden. Das Fusionieren kommt somit gänzlich ohne Change-Log aus.

Mit Hilfe eines vorhandenen Change-Log kann jedoch das Ergebnis verbessert werden, da durch diesen die genaue Abhängigkeit zu anderen Elementen rekonstruiert werden kann. Das erlaubt die Entscheidung zu optimieren, ob und wo ein Element eingefügt werden soll. Dadurch müssten weniger händische Nachbearbeitungen getätigt werden.

8.2. Funktionsweise

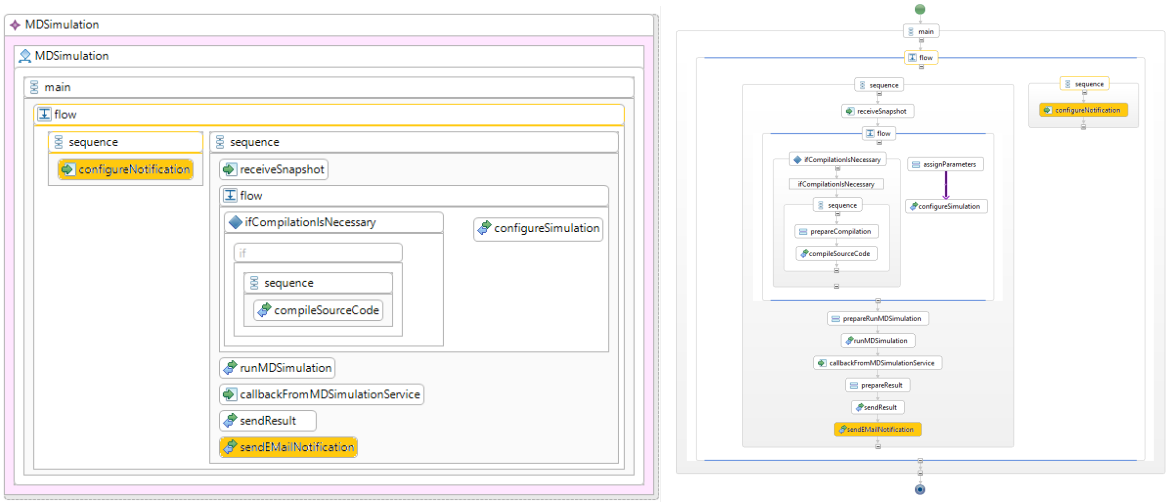
Um zu zeigen, dass die vorgestellte Methode zur automatischen Integration von Änderungen an Prozessmodellen, geeignet ist wird das Szenario der Einleitung aufgegriffen.

In der Abbildung 8.1a ist ein angepasster Choreographie-Ausschnitt der MD-Simulation zu sehen. Der Choreographie wurden zwei Aktivitäten (*configureNotfication*, *sendEMailNotification*) hinzugefügt. Mit diesen soll es dem Wissenschaftler ermöglicht werden, eine E-Mail Benachrichtigung zu konfigurieren. Möchte der Wissenschaftler eine Benachrichtigung erhalten, so kann er dies über *configureNotfication* mitteilen. Eine E-Mail wird dann an die hinterlegte E-Mail Adresse gesendet, sobald ein neuer Zwischenstand bekannt ist. Über diese Änderungen wird der Teilnehmer informiert.

Der Teilnehmer erhält die neue Choreographie-Beschreibung und muss nun seine Orchestrierung anpassen. Das neue Modell wird integriert und automatisch mit Verfeinerungen vervollständigt. In Abbildung 8.1b ist dargestellt, wie die Änderungen automatisch in die lokale Orchestrierung des Teilnehmers eingearbeitet wurden. Die Aktualisierungen müssen nicht mehr von Hand eingepflegt werden.

Das präsentierte Konzept genügt damit allen Anforderungen und kann für eine automatische Änderungsweitergabe erfolgreich verwendet werden.

[1] http://www.jdom.org/docs/faq.html#a0140



(a) Choreographie, modifizierter MD-Simulation Teilnehmer

(b) Orchestrierung, mit Aktualisierung und Verfeinerungen

Abbildung 8.1.: Aktualisierung und Änderungsweitergabe von Choreographie zur Orchestrierung

8.3. Schwächen

Das vorgestellte Konzept weißt allerdings auch Schwächen auf, die im Folgenden geschildert werden. Für die Ziele dieser Arbeit stellen diese jedoch keine Einschränkung dar.

Neu hinzugefügte Elemente werden problemlos übernommen. Für diese existieren im alten Modell noch keine Verfeinerungen. Auch verschobene Elemente stellen keine Schwierigkeit dar, da dies ein neu Positionieren und Umhängen von Teilbäumen darstellt. Werden allerdings Knoten entfernt, ist kein globales Wissen vorhanden, welche andere Knoten ebenfalls entfernt werden sollten. Damit keine Verfeinerungen verloren gehen, werden deshalb nur Knoten entfernt, die eindeutig vom entfernten Knoten abhängig waren. Dies kann dazu führen, dass im fusionierten Modell Verfeinerungen vorhanden bleiben, obwohl diese nicht mehr benötigt werden. Es wird jedoch angenommen, dass das Entfernen von überflüssigen Knoten leichter von Hand durchzuführen ist und eine deutlichere Erleichterung darstellt, als fehlende Knoten ausfindig zu machen und diese hinzuzufügen.

Eine andere Schwäche liegt in der Implementierung mittels DOM. Das komplette Modell wird mit DOM in den Arbeitsspeicher geladen. Das ermöglicht ein komfortables bearbeiten, jedoch begrenzt es die Größe von Modellen die behandelt werden können. Sie müssen in den Arbeitsspeicher passen. Heutzutage umfasst der Arbeitsspeicher in herkömmlichen Computern eine Größe von mehreren Gigabyte. Modelle, welche diese Größe überschreiten, müssen anders verarbeitet werden. Für diese bedarf es eine Methode, welche nicht das gesamte Modell in den Arbeitsspeicher lädt. Vorzugsweise werden diese Modelle partiell, strombasiert oder ereignisba-

siert verarbeitet. Die generierten und verarbeiteten Modelle, im betrachteten Anwendungsfall, sind alle deutlich kleiner, so dass dies keine Beeinträchtigung darstellt.

8.4. Laufzeitabschätzung

Im Folgenden wird eine einfache Laufzeitabschätzung für das Fusionieren der Modelle präsentiert. Dafür wird der konzipierte Algorithmus betrachtet.

Die Transformation des Modells in einen Graphen ist als Eingabe anzusehen, sowie die Transformation zurück in das ursprüngliche Format als Ausgabe. Für beides müssen alle Elemente einmal betrachtet werden und ist daher für die Abschätzung nicht relevant. Die Eingabe und Ausgabe erfolgt jeweils in linearer Laufzeit.

Um Referenzpunkte zu identifizieren werden zwei Graphen benötigt. Jeder Knoten aus einem Graphen muss paarweise mit einem Knoten aus dem anderen Graphen verglichen und dafür besucht werden. Angenommen die Graphen besitzen jeweils n und m Knoten und die Anzahl der Knoten beider Graphen unterscheiden sich um einen konstanten Faktor c, dann lässt sich die Anzahl der Vergleiche durch $\mathcal{O}(c_1 * n \times c_2 * m)$ formulieren. Für die Analyse können konstante Faktoren jedoch vernachlässigt werden, so dass sich $\mathcal{O}(n \times m)$ Besuche ergeben. Analog wird für das Erkennen der eingefügten und gelöschten Knoten vorgegangen.

Zum Finden der relevanten Verfeinerungen muss der Graph einmal mit der Liste der Referenzpunkten und Änderungen traversiert werden. Beim Durchlaufen werden die Verfeinerungen notiert. Dazu werden wieder alle Knoten einmal besucht. Für das Nachschlagen, ob ein Knoten in der Liste mit Referenzpunkten oder Änderungen steht, wird ein konstanter Zeitfaktor angenommen. Das Finden benötigt somit $\mathcal{O}(n)$ Knotenbesuche.

Für jede gefundene Verfeinerung muss ein Referenzpunkt zugeordnet werden. Dafür wird wieder der Graph traversiert und der auf dem Weg zuletzt passierte Referenzpunkt zurückgegeben. Angenommen, es wurden zuvor v Verfeinerungen gefunden, dann entspricht dies $\mathcal{O}(v \times n)$ Knotenbesuchen. Die Anzahl der Verfeinerungen pro Graphen ist immer echt kleiner als n, sonst wären die Graphen disjunkt und hätten keine Gemeinsamkeit. Außerdem können direkt innerhalb eines Durchlaufs alle Verfeinerungen zugeordnet werden. Daraus resultieren wieder $\mathcal{O}(n)$ Knotenbesuchen.

Das Modell-Skelett wird durch Kopieren eines Graphen initialisiert. Dieser Schritt des Kopierens benötigt konstante Zeit. Auch das Einfügen der Verfeinerungen, am konkreten Referenzpunkt, kann mit konstanter Zeit durchgeführt werden. Das Erstellen des fusionierten Modells fällt mit $\mathcal{O}(v)$ ins Gewicht.

Zuletzt müssen die restlichen Attribute übernommen werden. Dafür wird das fusionierte Modell, welches $n+v$ Knoten enthält, traversiert und fehlende Attribute ergänzt. Dies entspricht $\mathcal{O}(n + v)$ Knotenbesuche, wobei wie angemerkt $v < n$ zutrifft. Insgesamt ergibt sich eine Laufzeitkomplexität in $\mathcal{O}(n^2)$ für das Fusionieren zweier Graphen.

9. Zusammenfassung und Ausblick

Abschließend wird ein Überblick über das erarbeitete Konzept für die Aktualisierung und Änderungsweitergabe in Workflow-Choreographien gegeben und die wichtigsten Aspekte zusammengefasst. Anschließend erfolgt ein Ausblick auf weitere Forschungspunkte.

9.1. Zusammenfassung

Änderungen an der Workflow-Choreographie spiegeln sich in den Orchestrierungen der Teilnehmer wieder. Dafür müssen zunächst die betroffenen Teilnehmer über die Aktualisierungen informiert und zur Anpassung der eigenen Orchestrierung aufgefordert werden. Ein Teilnehmer möchte die Bestandteile der alten Orchestrierung in die neu spezifizierte Orchestrierung übernehmen. Dazu muss dieser die Änderungen erkennen und entsprechend in eine kombinierte Orchestrierung, aus alt und neu, einpflegen.

Diese Arbeit beschreibt ein Konzept, wie Aktualisierungen einer Workflow-Choreographie in eine bestehe Orchestrierung weitergegeben werden. Teil des Konzeptes ist es eine automatische Integration zu ermöglichen. Dies unterscheidet das Konzept zu bestehenden Arbeiten, welche halb-automatisch verfahren oder auf Eingabe eines Anwenders bestehen, um die Aktualisierungen einarbeiten zu können.

Das Konzept bedient sich dem Rahmenwerk zur Ablaufstruktur für Modell-Änderungs-Management[GL12], dieses definiert die notwendigen Schritte, um zwei Modelle zu fusionieren. Hierbei werden zunächst die Workflow-Modelle auf ein mathematisches Graph-Modell abstrahiert. Die Abstraktion als Graph erlaubt es, die Modelle zu vergleichen und Aktualisierungen, sowie vorhandene Bestandteile des alten Modells zu identifizieren.

Der Vergleich der Modelle erfolgt über den Graphen. Die Beziehung der Knoten beider Modelle wird mittels einer Zuordnung[Ger07] beschrieben. Eine Zuordnung kann zwischen zwei Knoten bestehen und gibt an, ob diese entweder in einem oder beiden Modellen vorhanden sind. Je nachdem welche Zuordnungen auf einen Knoten zutrifft, kann ermittelt werden, ob dieser hinzugefügt oder entfernt wurde. Zudem werden Knoten, welche in beiden Modellen vorhanden sind, als Referenzpunkte definiert.

Als nächster Schritt wird ein Skelett aus Knoten erstellt. Dazu werden die Knoten aus dem neuen Modell kopiert. Im Skelett fehlen damit noch die alten Bestandteile aus dem vorherigen

Modell. Jedoch sind im Skelett bereits alle Änderungen (Hinzufügungen, Löschungen, Verschiebungen) enthalten. Dadurch müssen die Verschiebungen, welche im Vergleich zum alten Modell erfolgten, nicht detailliert ermittelt werden. Der Vorteil dieses Vorgehens zeigt sich vor allem bei komplexeren Aktualisierungen.

Im Anschluss werden die Bestandteile des alten Modells in das Skelett eingefügt. Dazu werden die Bestandteile relativ positioniert. Die Positionierung erfolgt unter einem zuvor ermittelten Referenzpunkt. Konflikte zwischen den Modellen werden durch die Dominanz des neueren Modells gelöst, welches den Ausgang des Konflikts bestimmt.

Zudem wurde das präsentierte Konzept in den wissenschaftlichen Prototyp implementiert, welcher mit nötigen Anpassungen versehen wurde.

Das Konzept erlaubt es Aktualisierungen an die betroffenen Teilnehmer weiterzuleiten und automatisch zu integrieren. Dies bringt eine erhebliche Einsparung an Arbeitszeit, welche für die Integration der Änderungen aufgewendet werden müsste.

9.2. Ausblick

Ob sich das Konzept in der Praxis bewährt, muss für weitere Anwendungsfälle geprüft werden. Die Implementierung ist unter Beachtung möglicher Erweiterungen umgesetzt worden. Dies erlaubt das vorgeschlagene Konzept einerseits zu optimieren, und andererseits Modelle mit anderen darunter liegenden technischen Realisierungen zu unterstützen. Hier können weitere Optimierungen in Form von Vergleichs- und Vereinigungsstrategien entwickelt werden.

Das Konzept betrachtet die Weitergabe der Aktualisierungen innerhalb der Top-down Modellierung. Die automatische Weitergabe der Aktualisierungen innerhalb der Bottom-up Modellierung benötigt weitere Untersuchungen, damit Änderungen aus den einzelnen Orchestrierung hin zur Choreographie realisierbar sind.

Durch eine Versionierung von Choreographien könnten Änderungen langfristig verfolgt, sowie auf frühere Modelldefinitionen zurückgegriffen werden. Hier ist weiter Forschung nötig, wie eine solche Versionierung durchgeführt und in den Prototyp integriert werden kann.

In diesem Zusammenhang wäre auch eine Vergleichsansicht wünschenswert, in der Modelle verglichen und Änderungen übernommen werden können.

Anhang

- Klassifizierung des Match und Merge-Operators
- Typische Referenzpunkte
- Abkürzungsverzeichnis

A. Klassifizierung des Match und Merge-Operators

Die Klassifizierung des Match ($match : model \times model \mapsto relationship$) und Merge-Operators ($merge : model \times model \times relationship \mapsto model$) erfolgt nach Brunet et al. [Bru+06]. Die Beziehung der Modelle basiert auf den Zuordnungen.

Merge-Operator

- *Idempotenz* $merge(m_1, m_1) = m_1$:
  Die Idempotenz des Operators ist gegeben, das bedeutet ein Modell das mit sich selbst fusioniert wird resultiert im identischen Modell.
- *Kommutativität* $merge(m_1, m_2) = merge(m_2, m_1)$:
  Die Kommutativität ist nicht gegeben. Es wird angenommen, dass ein Modell in das andere Modell integriert wird.
- *Assoziativität* $merge(merge(m_1, m_2), m_3) = merge(m_1, merge(m_2, m_3))$:
  Die Assoziativität ist nicht gegeben. Die Reihenfolge der Anwendung des Merge-Operators ist aufgrund der Modell-Dominanz entscheidend.
- *Monotonie* $m_1 \preceq m_1' \wedge m_2 \preceq m_2' \Rightarrow merge(m_1, m_2) \preceq merge(m_1', m_2')$:
  Die Monotonie ist gegeben, falls die Relation für übernommene Änderungen steht.
- *Totalität* $\forall m_1, m_2 \in model : merge(m_1, m_2) \in model$:
  Die Totalität ist gegeben, aus zwei Modellen resultiert wieder ein gültiges Modell.

Match-Operator Der Match-Operator liefert eine binäre Relation für die Knoten V der Modelle A, B. Die Relation wird durch $A \times B \mapsto (V_a, V_b)$ beschrieben. Ein Knoten V ist maximal einmal in der Relationsmenge enthalten.

B. Typische Referenzpunkte

Innerhalb der verwendeten Modelle zeichnen sich typische Referenzpunkte ab.

B.1. BPEL-Datei

Typische Referenzpunkte der BPEL-Datei sind folgende Elemente:

- *process*
  - *extensions*
  - *partnerLinks*
  - *messageExchanges*
  - *variables*
  - *correlationSets*
  - *faultHandlers*
  - *eventHandlers*
  - *onAlarm*
  - *activity* Aktivitäten mit ID.

Für Aktivitäten ohne ID muss *Similarity-Matching* verwendet werden.

B.2. Deployment-Datei

Typische Referenzpunkte der Deployment-Datei sind folgende Elemente:

- *deploy*
  - *process*

B.3. WSDL-Datei

Typische Referenzpunkte der WSDL-Datei sind folgende Elemente:

- *definitions*
  - *types*
  - *message*
  - *portType*
  - *binding*
  - *service*

C. Abkürzungsverzeichnis

API Application Programming Interface

BPEL Business Process Execution Language

BPMN Business Process Model and Notation

DOM Document Object Model

EMF Eclipse Modeling Framework

EPC Event-driven Process Chain

GEF Graphical Editing Framework

GMF Graphical Modeling Framework

JAXP Java API for XML Processing

KMC Kinetic Monte Carlo Simulation

Mayflower Model-as-you-go Workflow Developer

MD Molekulardynamik Simulation

MOF Meta Object Facility

PBD Participant Behavior Description

PG Participant Grounding

PTop Participant Topology

RPST Refined Process Structure Tree

SAX Simple API for XML

SESE Single-Entry-Single-Exit

SimTech Simulation Technologie

SOA Service orientierte Architektur

SoC Separation of Concerns

StAX Streaming API for XML

URI	Uniform Resource Identifier
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WfMs	Workflow-Management-System
WS	Webservice
WSDL	Web Services Description Language
WST	Webservice Technologie
WS-*	Webservice Stack
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XPath	XML Path Language

Literaturverzeichnis

[Aal12] W. M. van der Aalst. „A decade of business process management conferences: personal reflections on a developing discipline“. In: *International Conference on Business Process Management*. Springer. 2012, S. 1–16 (zitiert auf S. 13).

[And+13] V. Andrikopoulos, S. Gómez Sáez, D. Karastoyanova, K. Vukojevic-Haupt et al. *Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies*. Stuttgart, Germany, Universität Stuttgart, 2013 (zitiert auf S. 62).

[AP03] M. Alanen, I. Porres. „Difference and union of models“. In: *International Conference on the Unified Modeling Language*. Springer. 2003, S. 2–17 (zitiert auf S. 36).

[BG07] R. Barga, D. Gannon. „Scientific versus business workflows“. In: *Workflows for e-Science*. Springer, 2007, S. 9–16 (zitiert auf S. 13).

[Bru+06] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, M. Sabetzadeh. „A manifesto for model merging“. In: *Proceedings of the 2006 international workshop on Global integrated model management*. ACM. 2006, S. 5–12 (zitiert auf S. 36, 85).

[Bru11] C. Brun. *What every developer should know about EMF Compare*. 2011. URL: https://www.slideshare.net/cbrun/what-every-developer-should-know-about-emf-compare (zitiert auf S. 75).

[Dec+07] G. Decker, O. Kopp, F. Leymann, M. Weske. „BPEL4Chor: Extending BPEL for modeling choreographies“. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE. 2007, S. 296–303 (zitiert auf S. 26, 27).

[Dec+09] G. Decker, O. Kopp, F. Leymann, M. Weske. „Interacting services: From specification to execution“. In: *Data & Knowledge Engineering* 68.10 (2009), S. 946–972 (zitiert auf S. 62).

[Dij+09] R. Dijkman, M. Dumas, L. Garcia-Banuelos, R. Kaarik. „Aligning business process models“. In: *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*. IEEE. 2009, S. 45–53 (zitiert auf S. 35).

[FRMR12] W. Fdhila, S. Rinderle-Ma, M. Reichert. „Change propagation in collaborative processes scenarios“. In: *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*. IEEE. 2012, S. 452–461 (zitiert auf S. 31).

[GAJV08] F. Gottschalk, W. van der Aalst, M. Jansen-Vullers. „Merging event-driven process chains“. In: *On the Move to Meaningful Internet Systems: OTM 2008* (2008), S. 418–426 (zitiert auf S. 32, 33).

[Gan+07] D. Gannon, E. Deelman, M. Shields, I. Taylor. „Introduction“. In: *Workflows for e-Science*. Hrsg. von I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields. London: Springer London, 2007, S. 1–8. ISBN: 978-1-84628-519-6 (zitiert auf S. 21).

[Ger07] C. Gerth. „Business Process Merging-An Approach based on Single-Entry-Single-Exit Regions“. Masterarbeit. 2007 (zitiert auf S. 33, 34, 42, 43, 81).

[GKE09] C. Gerth, J. M. Küster, G. Engels. „Language-independent change management of process models“. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2009, S. 152–166 (zitiert auf S. 74).

[GL12] C. Gerth, M. Luckey. „Towards Rich Change Management for Business Process Models“. In: *Softwaretechnik-Trends* 32.4 (2012), S. 32–34 (zitiert auf S. 34, 40, 41, 81).

[Hin14] K. Hintermayer. „Modellierung und Ausführung einer gekoppelten Festkörpersimulation mit Workflow-Choreographien“. Masterarbeit. 2014 (zitiert auf S. 15).

[Hul+06] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, T. Oinn. „Taverna: a tool for building and running workflows of services“. In: *Nucleic acids research* 34 (2006), W729–W732 (zitiert auf S. 13).

[JPP94] R. Johnson, D. Pearson, K. Pingali. „The program structure tree: Computing control regions in linear time“. In: *ACM SigPlan Notices*. Bd. 29. 6. ACM. 1994, S. 171–185 (zitiert auf S. 29).

[KGE09] J. M. Küster, C. Gerth, G. Engels. „Dependent and conflicting change operations of process models“. In: *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2009, S. 158–173 (zitiert auf S. 74).

[KL08] O. Kopp, F. Leymann. „Choreography Design Using WS-BPEL.“ In: *IEEE Data Eng. Bull.* 31.3 (2008), S. 31–34 (zitiert auf S. 13).

[KR] K. G.M.S. D. Karastoyanova, F. L. M. Reiter. *Conventional workflow technology for scientific simulation*, S. 22 (zitiert auf S. 13).

[KSN92] G. Keller, A.-W. Scheer, M. Nüttgens. *Semantische Prozeßmodellierung auf der Grundlage ¨Ereignisgesteuerter Prozeßketten (EPK)"*. Inst. für Wirtschaftsinformatik, 1992 (zitiert auf S. 32).

[Küs+08] J. M. Küster, C. Gerth, A. Förster, G. Engels. „Detecting and resolving process model differences in the absence of a change log“. In: *International Conference on Business Process Management*. Springer. 2008, S. 244–260 (zitiert auf S. 34).

[Ley10] F. Leymann. „BPEL vs. BPMN 2.0: Should you care?“ In: *International Workshop on Business Process Modeling Notation*. Springer. 2010, S. 8–13 (zitiert auf S. 26).

[LLS10] K. C. Laudon, J. P. Laudon, D. Schoder. *Wirtschaftsinformatik: Eine Einführung*. Pearson Deutschland GmbH, 2010 (zitiert auf S. 21).

[LR+10] M. La Rosa, M. Dumas, R. Uba, R. Dijkman. „Merging business process models“. In: *OTM Confederated International Conferences¨ On the Move to Meaningful Internet Systems"*. Springer. 2010, S. 96–113 (zitiert auf S. 13, 35).

[LR00] F. Leymann, D. Roller. „Production workflow: concepts and techniques“. In: (2000) (zitiert auf S. 13, 21, 22).

[Lud+06] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao. „Scientific workflow management and the Kepler system“. In: *Concurrency and Computation: Practice and Experience* 18.10 (2006), S. 1039–1065 (zitiert auf S. 13).

[Nem14] M. Nemet. „Kapselung von bestehenden Simulationsanwendungen mit Hilfe von Web Services“. Bachelorarbeit. 2014 (zitiert auf S. 26).

[PB03] R. A. Pottinger, P. A. Bernstein. „Merging models based on given correspondences“. In: *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment. 2003, S. 862–873 (zitiert auf S. 34, 42).

[RSM11] P. Reimann, H. Schwarz, B. Mitschang. „Design, implementation, and evaluation of a tight integration of database and workflow engines“. In: *Journal of Information and Data Management* 2.3 (2011), S. 353 (zitiert auf S. 13, 22).

[RWR06a] S. Rinderle, A. Wombacher, M. Reichert. „Evolution of process choreographies in DYCHOR“. In: *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE* (2006), S. 273–290 (zitiert auf S. 13).

[RWR06b] S. Rinderle, A. Wombacher, M. Reichert. „On the controlled evolution of process choreographies“. In: *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE. 2006, S. 124–124 (zitiert auf S. 13).

[Sch+11] D. Schumm, D. Karastoyanova, F. Leymann, S. Strauch. „Fragmento: advanced process fragment library“. In: *Information Systems Development*. Springer, 2011, S. 659–670 (zitiert auf S. 46).

[Sch14] J. Schilling. „Analyse und Erweiterung eines bestehenden Choreographiewerkzeugs“. Studienarbeit. 2014 (zitiert auf S. 61).

[SHK12] M. Sonntag, M. Hahn, D. Karastoyanova. „Mayflower-Explorative Modeling of Scientific Workflows with BPEL.“ In: *BPM (Demos)*. 2012, S. 45–50 (zitiert auf S. 13, 22, 23).

[SK10] M. Sonntag, D. Karastoyanova. „Next generation interactive scientific experimenting based on the workflow technology“. In: *Proceedings of MS'10* (2010), S. 349–356 (zitiert auf S. 13).

[SKD10] M. Sonntag, D. Karastoyanova, E. Deelman. „Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows“. In: *e-Science (e-Science), 2010 IEEE Sixth International Conference on*. IEEE. 2010, S. 206–213 (zitiert auf S. 13).

[SKL10] M. Sonntag, D. Karastoyanova, F. Leymann. „The Missing Features of Workflow Systems for Scientific Computations.“ In: *Software Engineering (Workshops)*. Citeseer. 2010, S. 209–216 (zitiert auf S. 13).

[Son13] O. Sonnauer. „Modellierung von Scientific Workflows mit Choreographien“. Diplomarbeit. 2013 (zitiert auf S. 61).

[Tay+07] I. Taylor, M. Shields, I. Wang, A. Harrison. „The triana workflow environment: Architecture and applications“. In: *Workflows for e-Science*. Springer, 2007, S. 320–339 (zitiert auf S. 13).

[TM10] C. Thao, E. V. Munson. „Using versioned tree data structure, change detection and node identity for three-way XML merging“. In: *Proceedings of the 10th ACM symposium on Document engineering*. ACM. 2010, S. 77–86 (zitiert auf S. 36).

[VVK09] J. Vanhatalo, H. Völzer, J. Koehler. „The refined process structure tree“. In: *Data & Knowledge Engineering* 68.9 (2009), S. 793–818 (zitiert auf S. 32).

[Wee+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more*. Prentice Hall PTR, 2005 (zitiert auf S. 13, 24, 25).

[Wei+15] A. Weiß, V. Andrikopoulos, M. Hahn, D. Karastoyanova. „Enabling the extraction and insertion of reusable choreography fragments“. In: *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE. 2015, S. 686–694 (zitiert auf S. 61).

[Wei12] A. Weiß. „Merging of TOSCA cloud topology templates“. Masterarbeit. 2012 (zitiert auf S. 28, 29).

[Whi04] S. A. White. „Introduction to BPMN“. In: *IBM Cooperation* 2 (2004) (zitiert auf S. 13).

[WK14] A. Weiß, D. Karastoyanova. „A life cycle for coupled multi-scale, multi-field experiments realized through choreographies“. In: *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*. IEEE. 2014, S. 234–241 (zitiert auf S. 23).

[WK16] A. Weiß, D. Karastoyanova. „Enabling coupled multi-scale, multi-field experiments through choreographies of data-driven scientific simulations“. In: *Computing* 98.4 (2016), S. 439–467 (zitiert auf S. 15, 25).

[WKM] A. Wei, D. Karastoyanova, D. Molnar. „Coupling of Existing Simulations using Bottom-up Modeling of“. In: *Workshop on Simulation Technology: Systems for Data Intensive*. Gesellschaft für Informatik eV (GI), S. 1–12 (zitiert auf S. 15, 26).

[Wol15] N. Wolter. „Konzept und Implementierung für Choreographiecontainer“. Masterarbeit. 2015 (zitiert auf S. 61).

[Woo+00] L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne. *Document object model (DOM) level 3 core specification.* 2000 (zitiert auf S. 29).

[WRRM08] B. Weber, M. Reichert, S. Rinderle-Ma. „Change patterns and change support features–enhancing flexibility in process-aware information systems“. In: *Data & knowledge engineering* 66.3 (2008), S. 438–466 (zitiert auf S. 32, 45).

Alle URLs wurden zuletzt am 22. 05. 2017 geprüft.

Erklärung

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

Ort, Datum, Unterschrift