Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3695

Abbildung von Datenmanagementpatterns auf ausführbare Workflowfragmente

Andreas Michael Bohrn

Studiengang:	Informatik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Dipl.-Inf. Peter Reimann
Beginn am:	29. Oktober 2014
Beendet am:	07. Mai 2015
CR-Nummer:	D.2.11, H.2.5, H.4.1, I.6.7

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Workflows haben sich im Unternehmensumfeld schon lange als wichtiges Werkzeug für die Modellierung von Prozessabläufen etabliert. Seit kurzem werden Workflows auch im wissenschaftlichen Bereich immer häufiger für die Implementierung von Datenanalysen oder Simulationen eingesetzt [TDGS14] [GSK+11]. Ein Beispiel für einen derartigen Einsatz von Workflows ist die Simulation von Strukturänderungen in Knochen unter unterschiedlichen Bewegungsabläufen [K+13]. Wissenschaftliche Workflows, die den computergestützten Ablauf einer solchen Simulation steuern, werden auch als Simulationsworkflows bezeichnet.

Um das Datenmanagement in derartigen Workflows zu erleichtern, wurde das SimTech Information Management, Processes and Languages (SIMPL) Rahmenwerk entworfen [RRS+11]. Dieses Rahmenwerk ermöglicht den einheitlichen Zugriff auf heterogene Datenquellen. Um den Entwurf von Simulationsworkflows weiter zu vereinfachen, umfasst das SIMPL Rahmenwerk zudem eine Hierarchie von Datenmanagementpatterns [RSM14b]. Diese Datenmanagementpatterns bilden eine zusätzliche Abstraktionsstufe, welche die Einzelheiten des Datenmanagements vor den Wissenschaftlern verbirgt und es Ihnen dadurch erlaubt, Datenmanagementschritte mit Begriffen und Konzepten aus dem jeweiligen Simulationsmodell zu definieren. Um aus einem Workflow mit diesen abstrakten Datenmanagementpatterns einen ausführbaren Simulationsworkflow zu machen, wurde eine Patterntransformation entworfen und implementiert [Ari12] [Pie12]. Um den Entwurf und die Abbildung von Patterns zu vereinfachen wurde eine Hierarchie auf den Datenmanagementpatterns entworfen. Anhand von Transformationsregeln werden Datenmanagementpatterns auf Workflowfragmente abgebildet. Diese Workflowfragmente können wiederum weitere Datenmanagementpatterns einer niedrigeren Hierarchiestufe enthalten. Wenn dies der Fall ist wird die Transformation rekursiv auf diese neuen Datenmanagementpatterns angewandt, bis ein ausführbares Workflowfragment entsteht. Die Patterntransformation wurde prototypisch für die Transformation während der Modellierung des Workflows umgesetzt.

Es hat sich jedoch herausgestellt, dass eine Transformation von Datenmanagementpatterns zur Laufzeit des Workflows viele Vorteile mit sich bringen würde [Pie12] [Boh14]. Bei einer Patterntransformation zur Laufzeit können Informationen, die erst zur Laufzeit feststehen, wie zum Beispiel Variablenwerte oder Systemkomponenten, die erst während der Laufzeit initialisiert werden, mit in die Patterntransformation einbezogen werden. Aus diesem Grund soll im Rahmen dieser Arbeit das SIMPL Rahmenwerk um eine Komponente für eine solche Transformation zur Laufzeit erweitert werden.

1.1 Aufgabenstellung

Im Rahmen dieser Arbeit soll zunächst untersucht werden, welchen Nutzen eine solche Patterntransformation zur Laufzeit bringt und in welchen Szenarien sie konkret eingesetzt werden sollte.

Anschließend soll das SIMPL Rahmenwerk [RRS+11] um eine neue Systemkomponente erweitert werden, welche die Patterntransformation sowohl zur Modellierungszeit als auch zur Laufzeit ermöglicht. Insbesondere werden folgende Anforderungen an diese Komponente gestellt:

- Die Patterntransformation soll möglichst effizient sein, sodass der Overhead einer regelbasierten Abbildung von Datenmanagementpatterns zur Laufzeit des Workflows gering ist.
- Das Einfügen neuer Transformationsregeln und Workflowfragmente soll möglichst einfach sein.
- Workflowfragmente, Patterns und Regeln sollen möglichst generisch einsetzbar sein, sowohl im Hinblick auf die Verwendung mit unterschiedlichen Simulationsmodellen, als auch bei der Verwendung zu den beiden Zeitpunkten Laufzeit und Modellierungszeit.

Außerdem soll untersucht werden, inwieweit bestehende Technologien für die Umsetzung verwendet werden können. Dies betrifft vor allem ein Framework für die Injektion von Prozessfragmenten zur Laufzeit [Sch11] [Hum11] [Bia14], den möglichen Einsatz von Regelengines wie die JRuleEngine [Car06], oder Drools [dro] für die Implementierung von Transformationsregeln und den Einsatz eines Triplestores, wie zum Beispiel RDF-3x [NW08], für die Verwaltung der Ontologien unterschiedlicher Simulationsmodelle.

Im Anschluss an die Implementierung soll die neue Komponente im Hinblick auf die zuvor gestellten Anforderungen evaluiert werden. Dabei steht vor allem der Overhead der Transformation zur Laufzeit und die generische Verwendbarkeit der Patterntransformer Komponente im Vordergrund. Für diese Evaluation werden bestehende Simulationsworkflows für die Simulation von Strukturänderungen in Knochen verwendet [Pie12] [Boh14].

1.2 Gliederung

Die weitere Arbeit gliedert sich in folgende Kapitel:

Kapitel 2 Grundlagen
bietet einen kurzen Überblick über die notwendigen Grundlagen dieser Arbeit. Hierzu zählen unter anderem: serviceorientierte Architekturen, Webservices, Workflows, das SIMPL Rahmenwerk und das Resource Description Framework (RDF)

Kapitel 3 Bestandsaufnahme
beschreibt die im Vorfeld dieser Arbeit prototypisch umgesetzte Patterntransformation zur Modellierungszeit eines Workflows näher. Außerdem wird der Workflow für die biomechanische Simulation von Strukturänderungen in Knochen, der im Rahmen dieser Arbeit für die Evaluation der Patterntransformation zur Laufzeit genutzt wird, beschrieben.

Kapitel 4 Motivation für eine Patterntransformation zur Laufzeit
beschäftigt sich mit dem Nutzen einer Transformation von Datenmanagementpatterns während der Laufzeit und beschreibt Szenarien, in denen sie eingesetzt werden sollte.

Kapitel 5 Auswahl und Bewertung existierende Technologien für die Umsetzung der Patterntransformation
beschäftigt sich mit der Auswahl geeigneter Frameworks und Technologien für die Umsetzung der Patterntransformation zur Laufzeit

Kapitel 6 Aufbau der Patterntransformation
beschreibt den konzeptionellen Entwurf der Patterntransformation

Kapitel 7 Implementierung
beschäftigt sich näher mit den Details der Implementierung der Patterntransformationskomponente. Außerdem wird in diesem Kapitel beschrieben, wie bestehende Tranformationsregeln und Regelsequenzen sowie zugehörige Workflowfragmente an die neue Implementierung der Patterntransformation angepasst wurden.

Kapitel 8 Bewertung
bewertet die zuvor implementierte Systemkomponente anhand ihres Overheads zur Laufzeit eines Simulationsworkflows sowie den zuvor gestellten Anforderungen

Kapitel 9 Weiterführende Arbeiten
beschreibt einige mögliche Verbesserungen und Erweiterungen der Patterntransformation. Dazu gehört zum einen die Umsetzung eines Sichtenkonzepts auf die Datenmanagementpatterns und zum anderen die Verwendung des PGM Modells zur Optimierung der Datenbereitstellung.

Kapitel 10 Zusammenfassung
fasst die vorangegangene Arbeit zusammen.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Technologien die für das Verständnis dieser Arbeit notwendig sind kurz erklärt. Als erstes werden die wichtigsten Konzepte aus dem Gebiet der Webservice kurz aufgeführt. Im Anschluss folgt eine kurze Beschreibung von Workflows und insbesondere der für diese Arbeit relevanten Workflowsprache BPEL und des Apache ODE Rahmenwerks für die Ausführung von Workflows. Danach wird das SIMPL Rahmenwerk, welches bereits in der Einleitung dieser Arbeit erwähnt wurde, näher beschrieben. Zuletzt wird das Resource Description Framework und die darauf aufbauende Ontologiesprache OWL näher erklärt.

2.1 Serviceorientierte Architektur

Der Begriff Serviceorientierte Architektur (SOA) wird in [Mel10] folgendermaßen definiert:

> ”Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendbarkeit ermöglicht.”

Die zentralen Konzepte und Prinzipien in einer SOA werden in dem sogenannten SOA- Dreieck dargestellt:

In Abbildung 2.1 werden die drei Rollen einer SOA und deren Interaktionen dargestellt. Möchte ein Anbieter einen Dienst zur Verfügung stellen, registriert er eine entsprechende Dienstbeschreibung bei einem Verzeichnisdienst(1. in 2.1). Diese Beschreibung enthält maschinenlesbare Information über die Funktionen, welche der Dienst bereitstellt, sowie eine Referenz für den Aufruf des Dienstes. Optional können auch nicht funktionale Eigenschaften des Dienstes hinterlegt werden. Ein Nutzer kann nun nach einem passenden Dienst innerhalb des Dienstverzeichnisses suchen (2. in 2.1) und erhält einen Verweis auf den passenden Dienst (3. in 2.1). Der Dienstnutzer kann nun über diesen Verweis eine Schnittstellenbeschreibung des Dienstes bei dem Dienstanbieter abfragen (4. in 2.1) um den Dienst nutzen zu können (5. in 2.1).

Durch diesen Ablauf kommt es zu einer losen Kopplung zwischen Dienstnutzer und Dienst, Dienste stehen zur Laufzeit noch nicht fest sondern werden dynamisch eingebunden. Dies erhöht die Flexibilität und die Wiederverwendbarkeit von Diensten.

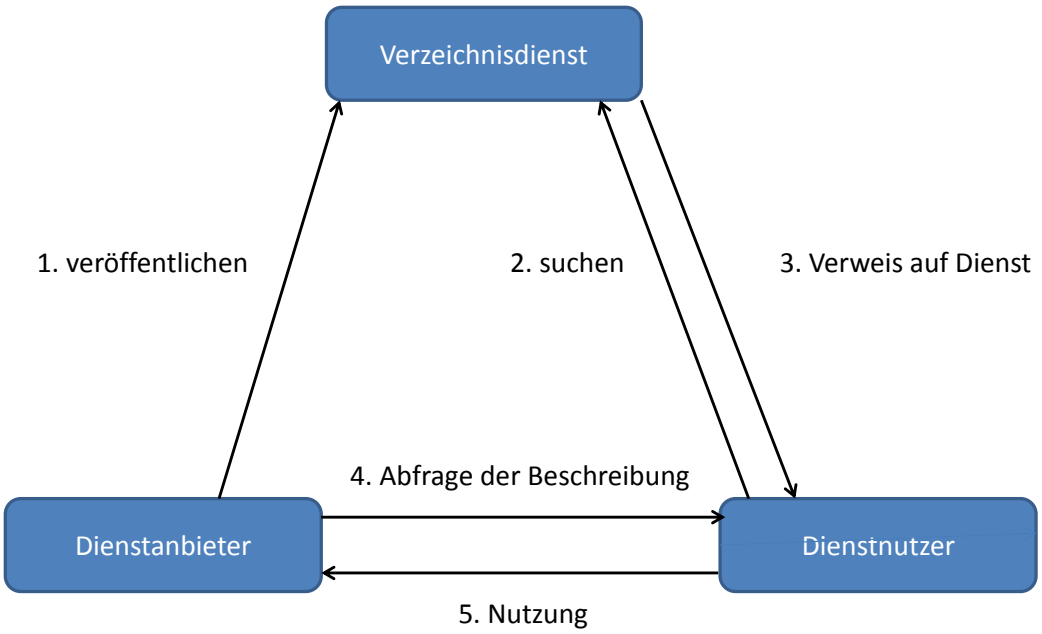

Abbildung 2.1: SOA Dreieck angelehnt an: [Mel10]

2.1.1 Webservice

Der folgende Abschnitt basiert, sofern nicht anders angegeben, auf der Quelle [Mel10]

Webservices stellen eine konkrete Umsetzung der Serviceorientierten Architektur dar. Das World Wide Web Consortium (W3C) [w3c] definiert Webservices folgendermaßen:

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

Für die Beschreibung eines Webservices wird die Web Service Definition Language (WSDL) genutzt. Die Kommunikation mit einem Webservice erfolgt über das SOAP Nachrichtenformat.

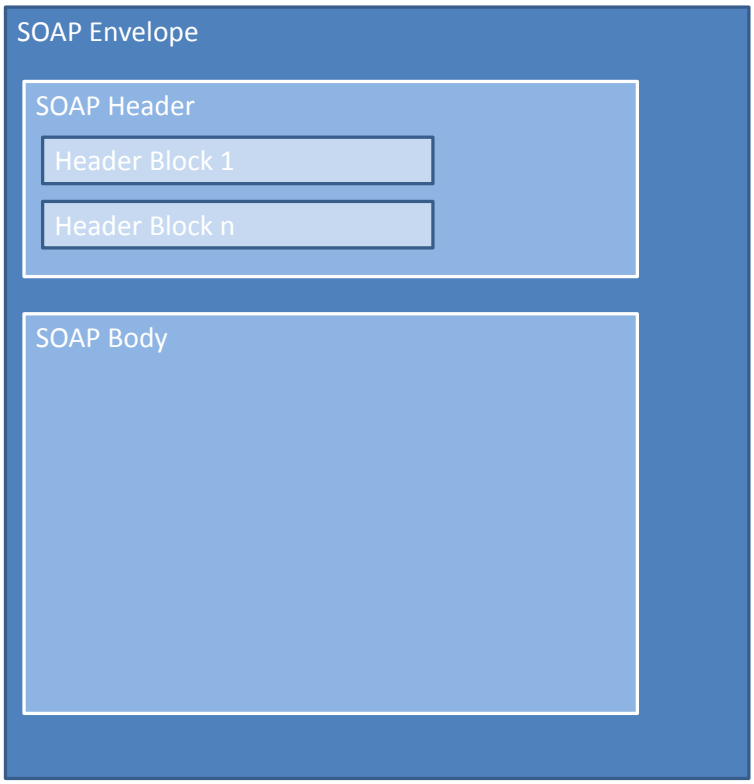

Abbildung 2.2: Aufbau einer SOAP Nachricht

SOAP

SOAP ist ein Nachrichtenprotokoll für den Austausch von Daten zwischen Systemen. Abbildung 2.2 zeigt den Aufbau einer SOAP Nachricht.

Eine SOAP Nachricht besteht aus einem Envelope Element. Dieser Envelope enthält einen oder mehrere optionale Header Elemente sowie ein nicht optionales Body Element. Die Header einer SOAP Nachricht enthalten üblicherweise Informationen für die weitere Verarbeitung der Nachricht für den Empfänger, oder für Zwischenstationen auf dem Transportweg der Nachricht. Außerdem kann der Header einer SOAP Nachricht verschiedene Attribute haben. Das wichtigste dieser Attribute ist MustUnderstand. Wenn dieses Attribut auf true gesetzt ist muss jede Station auf dem Transportweg den entsprechenden Header verstehen können, ansonsten wird die Nachricht nicht weitergeleitet Außerdem kann über Das Attribut role definiert werden, an welche Art von Zwischenstation sich der jeweilige Header richtet.

Das Body Element einer SOAP Nachricht enthält den eigentlichen Inhalt der Nachricht und richtet sich an den Empfänger (nicht an die Zwischenstationen). Der Body muss ein wohlgeformtes XML Dokument sein.

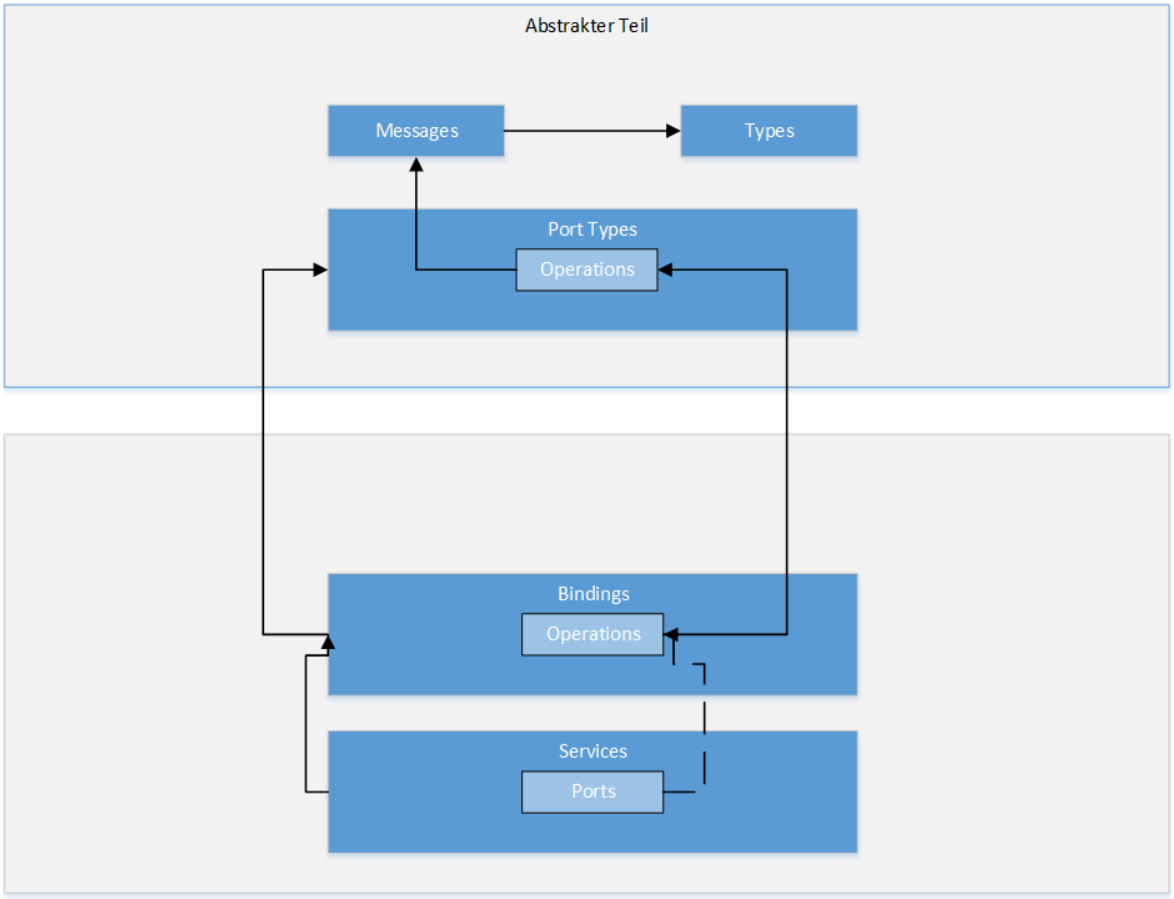

Abbildung 2.3: Aufbau einer WSDL Datei

WSDL

Die Web Service Definition Language (WSDL) ist eine Sprache zur Beschreibung von Webservices. WSDL basiert auf XML.

Abbildung 2.3 zeigt den grundlegenden Aufbau einer WSDL Datei gemäß der WSDL Version 1.0 [wsd].

Der abstrakte Teil der WSDL Datei ist eine protokollunabhängige abstrakte Interfacebeschreibung des Dienstes. Jeder Webservice definiert innerhalb des abstrakten Abschnitts der WSDL Beschreibung einen, oder mehrere Port Types. Ein Port Type fasst eine oder mehrere Operation, welche von dem Dienst bereitgestellt werden, zusammen. Dabei beschreibt eine Operation die konkrete Methode des Dienstes anhand ihrer Input- und Output Nachrichten.

Im konkreten Teil der WSDL Datei wird die Bindung der Operationen eines Webservices an einen Endpunkt und ein Transportprotokoll beschrieben. Diese Endpunkte (meist eine URI) werden dabei als ports dargestellt, welche wiederum an einen Port Type gebunden sind. Bindings definieren dabei

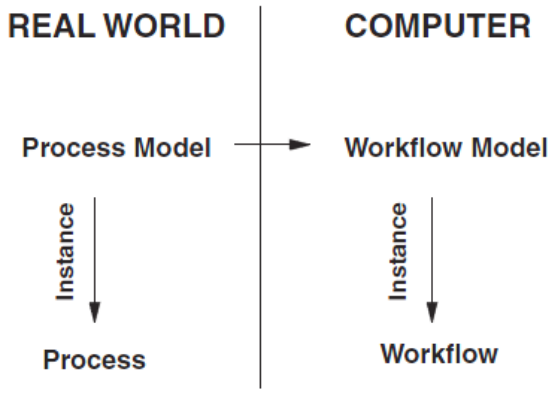

Abbildung 2.4: Prozesse und Workflows Quelle: [LR00]

die Abbildung der Operationen eines Port Types auf konkrete Datenformate und Transportprotokolle. Einer oder mehrere Ports werden zu einem Service zusammengefasst.

2.2 Workflows

Ein Workflow (deutsch: Arbeitsablauf) beschreibt die Reihenfolge, in der eine Reihe von Aktivitäten abgearbeitet werden. Dabei werden oft einzelne Aktivitäten zu größeren Komponenten zusammengefasst. Dies erhöht sowohl die Flexibilität, als auch die Wiederverwendbarkeit. Workflows haben ihren Ursprung vor allem in der Beschreibung von Geschäftsprozessen [LR00]. Sie werden aber zunehmend auch im Bereich der wissenschaftlichen Simulation eingesetzt [GSK+11].

2.2.1 Prozessmodelle und -instanzen

In [LR00]wird ein Prozessmodell definiert als eine Beschreibung eines (Geschäfts-) Prozesses in der realen Welt. Das Modell beschreibt eine Reihe von Aktivitäten und dazugehörige Logik die nötig sind um einen Prozess abzuarbeiten. Ein Prozessmodell muss dabei nicht computergestützt ausgeführt werden. Teile eines Prozessmodells können auch von Personen ausgeführt werden, während andere Aktivitäten maschinell bearbeitet werden. AUs einem Prozessmodell können einzelne Prozessinstanzen abgeleitet werden. Die tatsächliche Ausführung eines Prozessmodells nennt sich Prozessinstanz. Ein Prozessmodell, dass auf einem Computer ausgeführt wird, wird auch Workflowmodell genannt und eine Instanz eines Workflowmodells wird als Workflow bezeichnet. Die Abbildung 2.4 veranschaulicht die Zusammenhänge zwischen diesen Begriffen noch einmal.

Prozessmodelle werden in der Regel von drei Dimensionen beschrieben:

Wer
: Diese Dimension gibt an, welche Abteilung oder Person eines Unternehmens eine Aktivität bearbeiten soll.

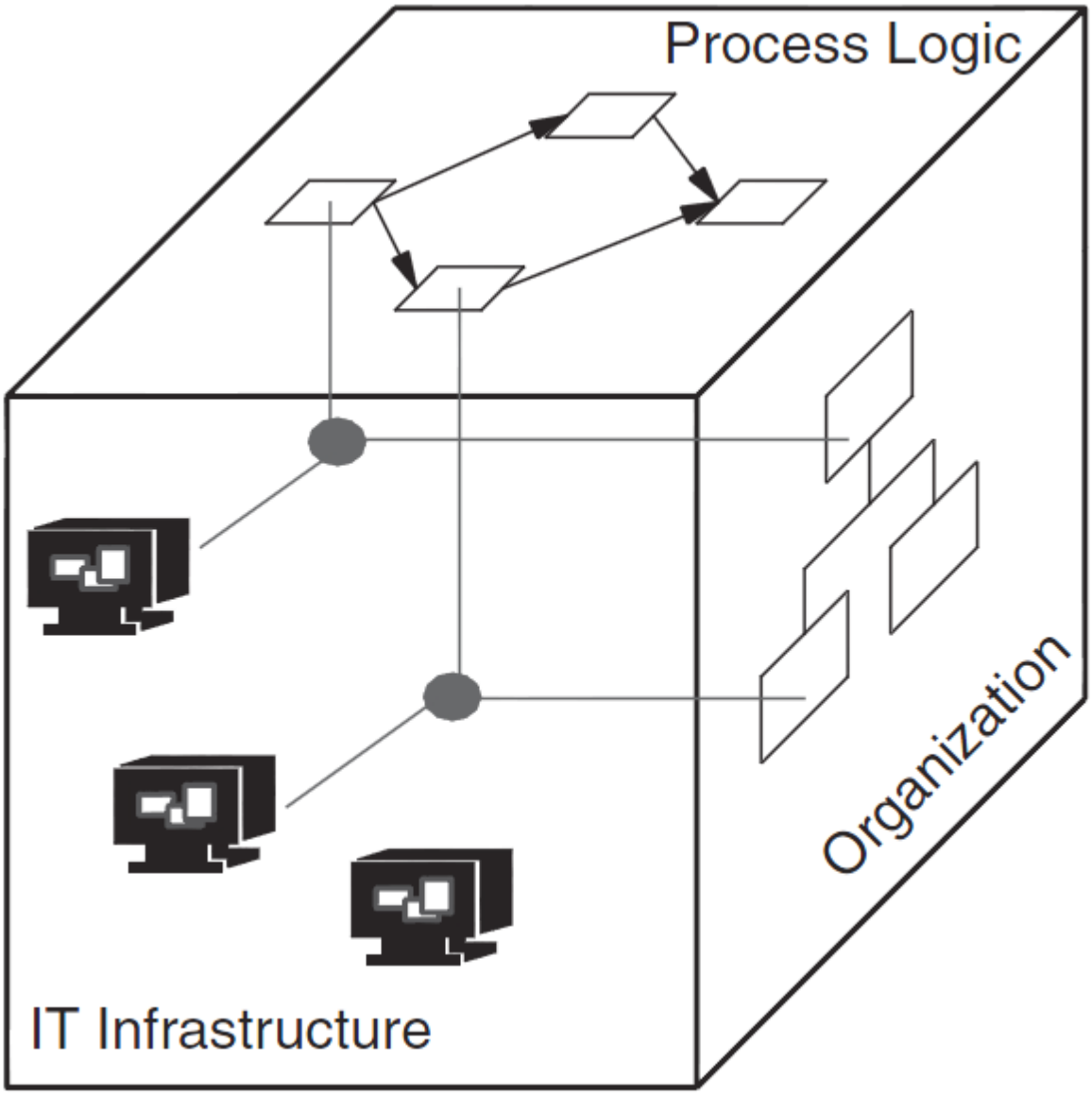

Abbildung 2.5: Dimensionen eines Workflows Quelle: [LR00]

Wie

Diese Dimension beschreibt, welche Aktivitäten in welcher Reihenfolge ausgeführt werden sollen. Dabei können Aktivitäten auch parallel ausgeführt werden, oder aber an Bedingungen geknüpft sein.

Womit

Diese Dimension listet alle Hilfsmittel, welche für die Bearbeitung einer Aktivität benötigt werden, auf.

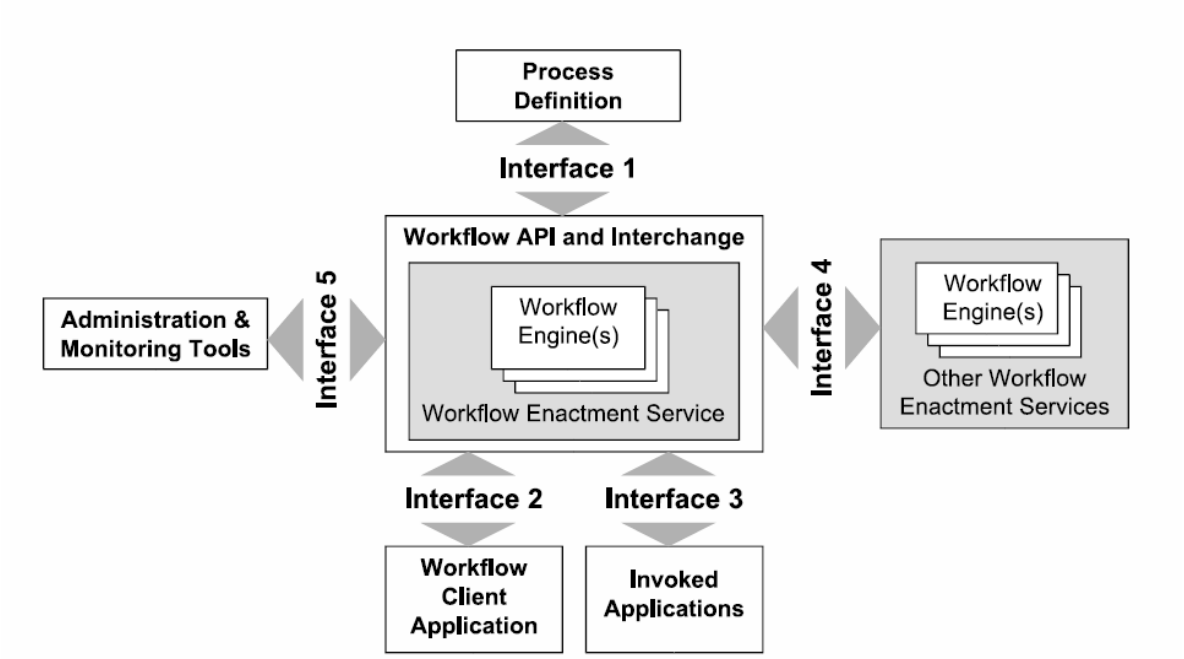

Abbildung 2.6: Referenzmodell eines WfMS Quelle: [LR00]

Abbildung 2.5 stellt die drei Dimensionen eines Workflows in dreidimensionaler Form dar. Ein Punkt in diesem Raum beschreibt, welche Person, auf welche Art und mit welchen Hilfsmitteln eine Aktivität bearbeitet.

2.2.2 Workflow Management Systeme

Der Begriff Workflow Management System beschreibt eine Anwendung zur Definition, Verwaltung und Ausführung von Workflows. Die Workflow Management Coalition (WfMC)[1] hat ein Referenzmodell für Workflow Management Systeme entwickelt (Abbildung 2.6).

Die einzelnen Komponenten dieses Referenzmodells werden in [Mel10] wie folgt näher beschrieben:

Process Definition Eine Komponente zur Modellierung der einzelnen Prozesse.

Workflow Engine Die Workflow Engine (auch Execution Engine genannt) ist für die Ausführung des zuvor definierten Prozesses verantwortlich. Ein Beispiel für eine Execution Engine ist Die Apache ODE, welche in einem späteren Abschnitt dieses Kapitels näher beschrieben wird.

Workflow Client Application Die Workflow Client Application ist eine Schnittstelle, welche dem Nutzer die Funktionen des Workflow Management Systems zur Verfügung stellt.

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

Invoked Applications Eine Zusammenfassung aller Anwendungen die in den Workflow eingebunden werden.

OtherWorflow Enactment Services Diese Komponente ermöglicht die Kommunikation mit anderen Workflowsystemen. Dadurch können bereits bestehende Prozesse in den Workflow eingebunden werden.

Administration and Monitoring Tool Eine Komponente für die Überwachung der Ausführung von Workflowinstanzen.

2.2.3 BPEL

Das Akronym BPEL steht für Business Process Execution Language [Jor07]. BPEL ist eine XML basierte Sprache zur Beschreibung von Prozessen. BPEL ist Teil der WS-*-Spezifikationen für Webservices und heißt daher eigentlich WS-BPEL. Die Aktivitäten eines BPEL Prozesses werden durch Webservices implementiert und auch ein BPEL Prozess selbst kann als Webservice bereitgestellt werden. Dies erleichtert die Komposition von Bpel Prozessen. Die Einbindung eines Webservices in einen BPEL Prozess erfolgt über dessen WSDL Beschreibung. für diese Bindung verwendet BPEL das Konzept des partnerLinks. Ein partnerlink beschreibt die Interaktion zwischen dem BPEL Workflow und einem Webservice anhand sogenannter Rollen. 2.1 zeigt ein einfaches Beispiel eines partnerLinks und dem zugehörigen partnerLinkType. Der partnerlinkType beispielPLT definiert die Rolle pltProvider anhand des Port Types, wlechen diese Rolle zur Verfügung stellen muss. der partnerLink beispielPL ist eine Instanz des Typs beispielPLT. Dieser PartnerLink weißt dem zugehörigen Webservice die Rolle pltProvider zu, das heißt der Webservice muss den entsprechenden Port Type bereitstellen. Ein Partnerlink könnte auch über das Attribut myRole auch dem Workflow einen Port Type zuweisen.

```
<partnerLinkType name="beispielPLT">
<role name="pltProvider">
<portType name="ns:beispielPT"/>
</role>
</partnerLinkType >

<partnerLink name="beispielPL" partnerLinkType ="beispielPLT"
partnerRole="pltProvider"/>
```

Listing 2.1: PartnerLink Beispiel

Bpel bietet die Möglichkeit Daten in typisierten Variablen zu speichern. Für die Definition von Variablen wird XML Schema und WSDL messages verwendet. BPEL Prozesse sind block strukturiert Ein BPEL Prozess besteht aus aus einem, oder mehreren verschachtelten Scopes. Ein Scope vereint eine Reihe von Aktivitäten zu einer transaktionalen Einheit. Zu diesem Zweck kann ein Scope Fault Handler, für die Behandlung von Fehlern, Compensation Handler für die Kompensation bereits ausgeführter Aktivitäten und Even Handler für die Reaktion auf auftretende Events definieren. Jeder Scope kann zudem eigene lokale Variablen und partnerLinks definieren.

Die Aktivitäten eines BPEL Prozesses gliedern sich in zwei Gruppen. Simple Aktivitäten sind atomar, d.h. sie sind nicht aus anderen Aktivitäten aufgebaut. Webservice Aufrufe werden in BPEL als atomar

angesehen und gehören daher zu den simple activities. Zu den wichtigsten einfachen Aktivitäten gehören:

- assign: Weist einer Variable einen Wert zu
- invoke: Synchroner Aufruf eines Webservices
- receive: Prozess wartet auf eine Nachricht eines Webservices
- reply: Prozess sendet Antwort an einen Webservice
- throw: Fehler wird geworfen
- extensionActivity: ermöglicht die Erweiterung von BPEL um zusätzliche Aktivitäten

Die zweite Gruppe bilden die komplexen Aktivitäten. Komplexe Aktivitäten bestehen aus simplen Aktivitäten und dienen hauptsächlich zur Beschreibung des Kontrollflusses eines Workflows. Zu den komplexen Aktivitäten zählen unter anderem if, while und for each, sowie die Scope Activity. Zudem existiert die Sequence Activity für die Modellierung von sequentiell ablaufenden Aktivitäten und die Flow Activity für die Modellierung von nebenläufigen Prozessabläufen.

2.2.4 Apache ODE

Die Apache Orchestration Director Engine (ODE) ist eine von der Apache Foundation entwickelte Java basierte Open Source Workflow Engine zur Ausführung von WS-BPEL Prozessen [ode].

Aufbau der ODE

Abbildung 2.7 zeigt die Architektur der Apache Orchestation Director Engine. Im folgenden werden die wichtigsten Komponenten kurz beschrieben. Für eine genauerer Erklärung sei auf die offizielle Dokumentation unter [ode] verwiesen

BPEL Compiler
: Diese Komponente ist für die Kompilierung der .bpel Dateien und der dazugehörigen WSDL und XML Dateien zu einem ausführbaren Prozess verantwortlich

ODE BPEL Runtime
: Die Runtime Komponente übernimmt die Instanziierung der Prozessinstanzen eines Prozessmodells. Außerdem übernimmt die Runtime Komponente die Zuordnung von Nachrichten an die einzelnen Prozessinstanzen.

JaCOb
: Die Java Concurrent Objects (JaCOb) Komponente ist Teil der Runtime Umgebung und erlaubt die nebenläufige Ausführung von BPEL Konstrukten. Außerdem erlaubt JaCOb die persistente Verwaltung des Ausführungsstatus von BPEL Konstrukten. JaCOb bildet damit einer Form einer persistenten virtuellen Maschine zur Ausführung von BPEL Konstrukten dar.

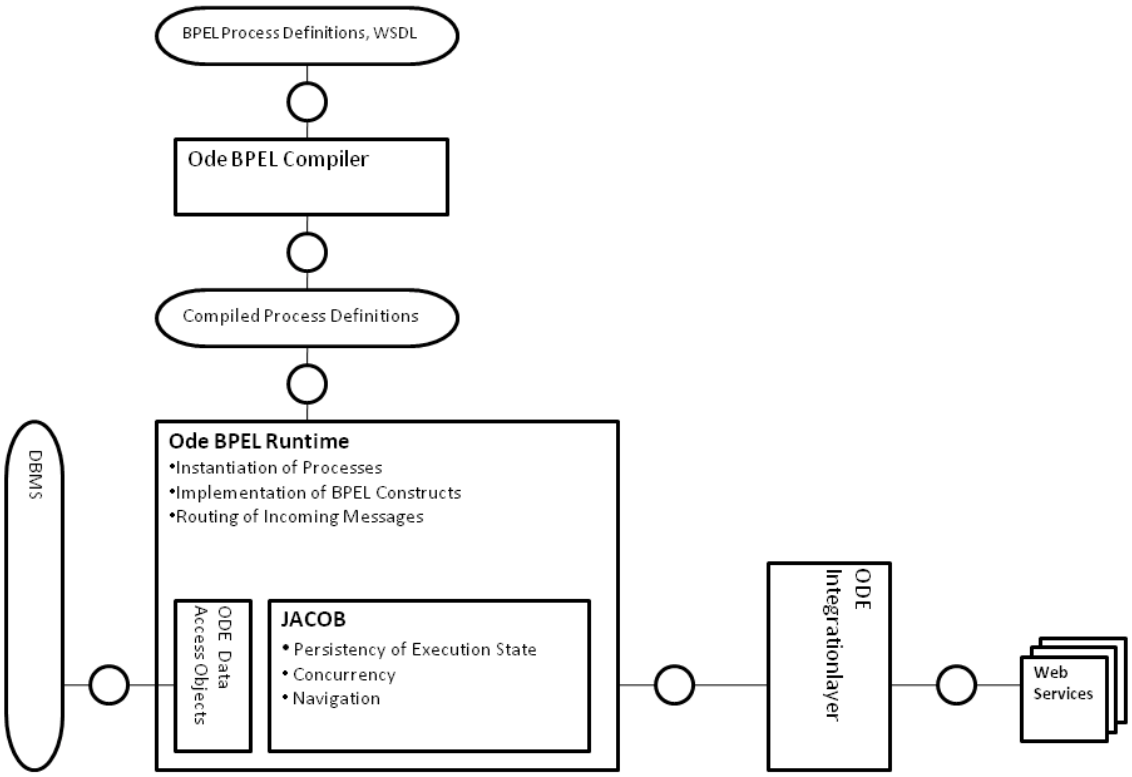

Abbildung 2.7: Apache ODE Architektur Quelle: [ode]

ODE Integration Layer

Der Integration Layer bildet eine Schnittstelle zwischen der Runtime und Webservices, mit denen die Runtime während der Ausführung kommunizieren muss. Eine mögliche Implementierung dieses Layers verwendet zum Beispiel Axis2 um die Kommunikation mit Webservices zu unterstützen.

2.3 SIMPL

Dieser Abschnitt basiert auf [RRS+11] und [RS14]

SIMPL steht für SimTech – InformationManagement, Processes, and Languages. Es ist ein Workflow Manangement System für den einheitlichen Zugriff auf verschiedene Datenquellen mit unterschiedlichen Dateiformaten. SIMPL baut auf Apache ODE als Workflowengine auf. Entwickelt wurde SIMPL in einer Zusammenarbeit des Instituts für Parallele und Verteilte Systeme (IPVS) und des Instituts für Architektur von Anwendungssystemen(IAAS).

Abbildung 2.8 zeigt den Aufbau des SIMPL Rahmenwerks. Die SIMPL Core Komponente des SIMPL Rahmenwerks bietet Zugriffsoperationen für den einheitlichen Zugriff auf externe Datenquellen.

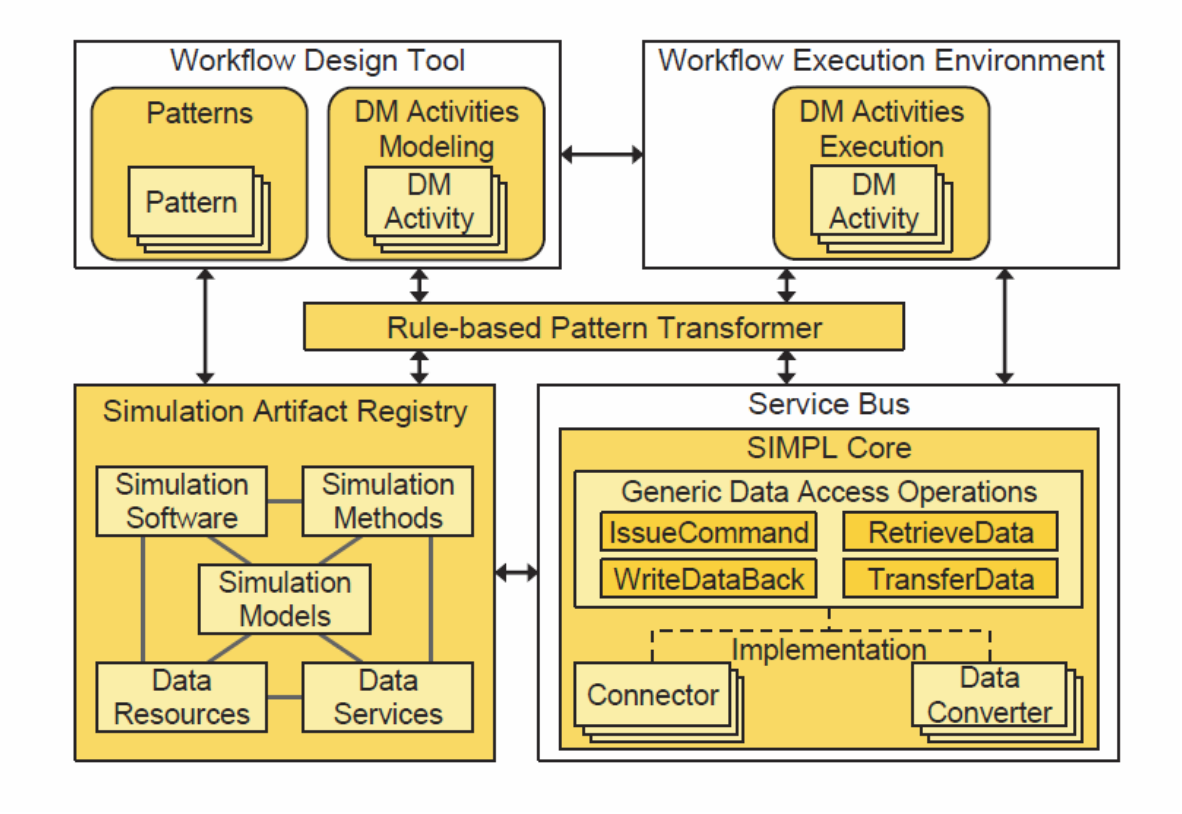

Abbildung 2.8: Das SIMPL-Rahmenwerk eingebettet in ein sWfMS Quelle: [RRS+11]

Eine Reihe neuer Datamanagement Activities ermöglichen die Nutzung dieser Funktionalität bei der Implementierung von Workflowmodellen. Folgende Activities werden implementiert:

IssueCommand Diese Aktivität erlaubt das Ausführen eines Befehls auf einer Datenquelle (zum Beispiel SQL Abfrage oder Shell Anweisung)

RetrieveData Diese Aktivität erlaubt das Laden von Daten aus einer Datenquelle

WriteDataBack erlaubt es Daten in eine Datenquelle zurückzuschreiben

TransferData erlaubt die Übertragung von Daten aus einer Datenquelle in eine andere.

Um diese Activities auch während der Laufzeit ausführen zu können erweitert SIMPL die Workflow Engine (Apache ODE) um diese Data Management Activities.

Als Workflow Design Tool verwendet SIMPL das BPEL Designer Project[2] für Eclipse. Dieses wird um die benötigten Activities erweitert. Außerdem erlaubt das SIMPL Rahmenwerk die Modellierung von Datenmanagement Schritten mit sogenannten Datenmanagementpatterns, welche im folgenden Abschnitt näher beschrieben werden. Für die Transformation dieser Patterns umfasst das SIMPL

[2] https://eclipse.org/bpel/

Rahmenwerk eine Pattern Transformer Komponente und ein Simulation Artifact Registry welches Metadaten und Workflowfragmente für die Patterntransformation beinhaltet

2.3.1 BPEL-DM

Um den einheitlichen Zugriff auf externe Datenquellen zu unterstützen erweitert das SIMPL Rahmenwerk die Workflowsprache BPEL um die sogenannte Business Process Execution Language extension for Data Management (BPEL-DM). Diese Erweiterung beinhaltet eine Reihe von zusaätzlichen Data Management (DM) Aktivitäten) für den Umgang mit externen Datenquellen. Für die Ausführung dieser zusätzlichen Aktivitäten wird bei der Ausführung der SIMPL Core aufgerufen. Die SIMPL Core Komponente übernimmt anschließend die eigentliche Ausführung der Operation auf der jeweiligen Datenquelle.

Datenquellen für eine solche DM Aktivität können Datenbanken oder auch Dateisysteme sein. DIe Operationen, die von einer DM Aktivität auf einer solchen Datenquelle ausgeführt werden, werden als DM commands bezeichnet.

Für die Umsetzung dieser DM Aktivitäten wurde das BPEL Framework zusätzlich um sogenannte Data Source Reference Variables erweitert. Diese Referenzvariablen stellen einen logischen Verweis auf die jeweilige Datenquelle dar.

Dateien, Tabellen oder Ordner innerhalb einer Datenquelle werden durch das Konzept der Datencontainer zusammengefasst. Über sogenannte Data Container Reference Variables können eben diese Datencontainer innerhalb einer Datenquelle referenziert werden. Die Abbildung dieser Referenzvariablen auf die referenzierten Datenquellen und Datencontainer übernimmt während der Ausführung das Resource Management des SIMPL Frameworks.

Im folgenden werden die einzelnen DM Aktivitäten des SIMPL Frameworks kurz erläutert:

RetrieveData Activity
: Die RetrieveData Aktivität erlaubt das Laden von Daten aus einer Datenquelle. Als Eingabeparameter benötigt diese Aktivität einen DM Command, eine Datenquelle auf der dieser Command ausgeführt werden soll und eine Data Set Variable, um das Ergebnis des Commands zu speichern. Ein solcher DM Command kann zum Beispiel ein SQL Statement, oder der Pfad der zu ladenden Datei sein. Zur Laufzeit wird der entsprechende Command an die Datenquelle geleitet und das Ergebnis wird anschließend in der zuvor angegebenen Data Set Variable gespeichert. Sollte es bei der Ausführung des DM Commands zu einem Fehler kommen erfolgt eine Rückmeldung an die Execution Engine. Die Execution Engine ermöglicht Fault Handling für DM Commands.

WriteDataBack Activity
: Die WriteDataBack Aktivität bildet das Gegenstück zu der RetrieveData Activity, da sie das Speichern von Daten in einer Datenquelle ermöglicht. Eingabeparameter dieser Aktivität sind zum einen eine Data Set Variable mit den zu speichernden Daten und zum anderen eine Container Reference Variable für den Datencontainer in dem diese Daten gespeichert werden sollen. Nach der Ausführung dieser Operation wird die Execution Engine benachrichtigt, ob die Operation erfolgreich war oder nicht.

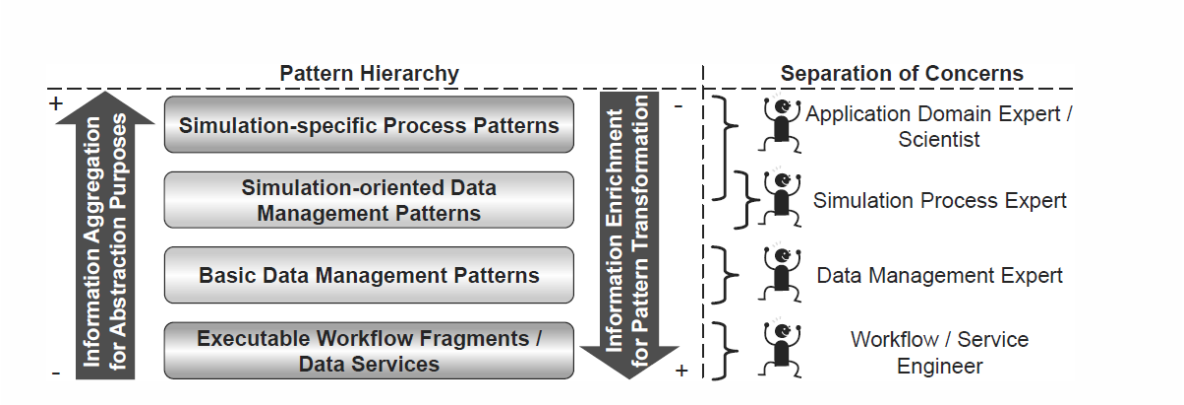

Abbildung 2.9: Hierarchie von Datenmanagementpatterns mit unterschiedlichen Abstraktionsebenen Quelle: [RSM14b]

Issue Command Activity

Die Issue Command Activity erlaubt die Ausführung eines beliebigen DM Commands auf einer Datenquelle. Die Eingabeparameter dieser Aktivität sind zum einen eine Datenquelle und zum anderen ein DM Command. Diese Aktivität dient vor allem dazu Daten auf einer Datenquelle manipulieren zu können. Nach der Ausführung dieser Aktivität wird die Execution Engine über den Erfolg, oder Misserfolg der Ausführung des DM Commands informiert.

TransferData Activity

Die TransferData Aktivität unterstützt den Datentransfer zwischen zwei Datenquellen. Eingabeparameter dieser Aktivität sind eine Datenquelle aus der Daten geladen werden sollen, eine Zieldatenquelle sowie einen Datencontainer auf dieser Zieldatenquelle und einen DM Command. Bei der Ausführung dieser Activity werden die durch den DM Command spezifizierten Daten aus der ersten Datenquelle in den Datencontainer auf der zweiten Datenquelle übertragen.

2.3.2 Datenmanagementpatterns

Um Wissenschaftlern die Modellierung von Simulationsworkflows weiter zu erleichtern wurde das SIMPL Rahmenwerk um ein Modell zur pattern-basierten Modellierung von Workflows erweitert [RSM14a]. Wissenschaftler können dabei abstrakte Patterns mit simulationsbezogenen Parameterwerten für die Workflowmodellierung verwenden, ohne sich zum Beispiel um die Details der Datenbereitstellung kümmern zu müssen.

Patternhierarchie

In [RSM14b] wird eine Hierarchie auf den Datenmanagementpatterns definiert.

Abbildung 2.9 zeigt den Aufbau dieser Hierarchie. Die Hierarchie der Datenmanagementpatterns besteht aus vier Ebenen, wobei die Abstraktion von Implementierungsdetails mit jeder Ebene zunimmt.

Bei der Abbildung von Datenmanagementpatterns innerhalb der Patternhierarchie werden diese dann durch die Abbildung auf Patterns niedrigerer Abstraktionsebenen schrittweise wieder mit Informationen angereichert, bis zuletzt ein ausführbarer Workflow entsteht.

Auf der obersten Ebene der Patternhierarchie stehen die simulationsspezifischen Prozess Patterns. Patterns dieser Abstraktionsstufe modellieren Anwendungsfälle von Simulationsmodellen. Die Parameter dieser Patterns umfassen simulationsspezifische Konzepte und Begriffe mit denen Wissenschaftler vertraut sind. Für die Beschreibung der Metadaten der verwendeten Simulationsmodelle werden oft Ontologien verwendet.

Auf der nächsten Ebene der Patternhierarchie stehen die simulationsorientierten Datenmanagementpatterns. Patterns dieser Abstraktionsebene verwenden zwar größtenteils die gleichen simulationsbezogenen Variablenwerte wie die simulationsspezifischen Prozess Patterns, beschreiben aber im Gegensatz zu diesen die Datenbereitstellung innerhalb eines Simulationsworkflows.

Auf der untersten Stufe der Patternhierarchie vor den ausführbaren Workflows stehen die grundlegenden Datenmanagementpatterns. Patterns dieser Ebene abstrahieren Implementierungsdetails der ausführbaren Workflowfragmente und Daten Service.

Die unterschiedlichen Abstraktionsebenen der Patternhierarchie ermöglicht eine Trennung von Zuständigkeiten zwischen unterschiedlichen Expertengruppen bei der Workflowmodellierung. Patterns der höchsten Abstraktionsebene können zum Beispiel aufgrund ihrer Beziehung zu Simulationsmodellen von Wissenschaftlern mit entsprechendem Domänenwissen festgelegt werden, während sich Experten für das Datenmanagement um die Implementierung von grundlegenden Datenmanagementpatterns kümmern.

Patterntransformation

Um Datenmanagementpatterns ausführen zu können müssen sie vorher auf ausführbare Workflowfragmente abgebildet werden. Zu diesem Zweck werden Patterns höherer Hierarchiestufen auf Patterns der darunterliegenden Ebene der Patternhierarchie abgebildet. Dieser Vorgang wird rekursiv solange durchgeführt, bis ein ausführbares Workflowfragment entsteht.

Die bisherige Implementierung der Patterntransformation wurde in [RSM14b] beschrieben. Abbildung 2.10 zeigt den Aufbau dieser Patterntransformation.

Die Transformation von Datenmanagementpatterns erfolgt durch sogenannte Transformationsregeln. Eine Transformationsregel besteht aus einem Condition Part und einem Action Part. Der Condition Part einer Transformationsregel wird während der Patterntransformation aufgerufen, um zu entscheiden, ob eine Regel anwendbar ist, oder nicht. Der Actionpart einer Transformationsregel enthält den Programmcode, um das zu dem Pattern gehörige Workflowfragment aus einem externen Repository zu laden und mit den benötigten Variablenwerten zu füllen. Mehrere Transformationsregeln werden zu einer Regelsequenz zusammengefasst. Die Kontrollstrategie bestimmt dabei in welcher Reihenfolge die jeweiligen Transformationsregeln auf ihre Anwendbarkeit geprüft werden. Jedem Datenmanagementpattern ist eine Kontrollstrategie zugeordnet.

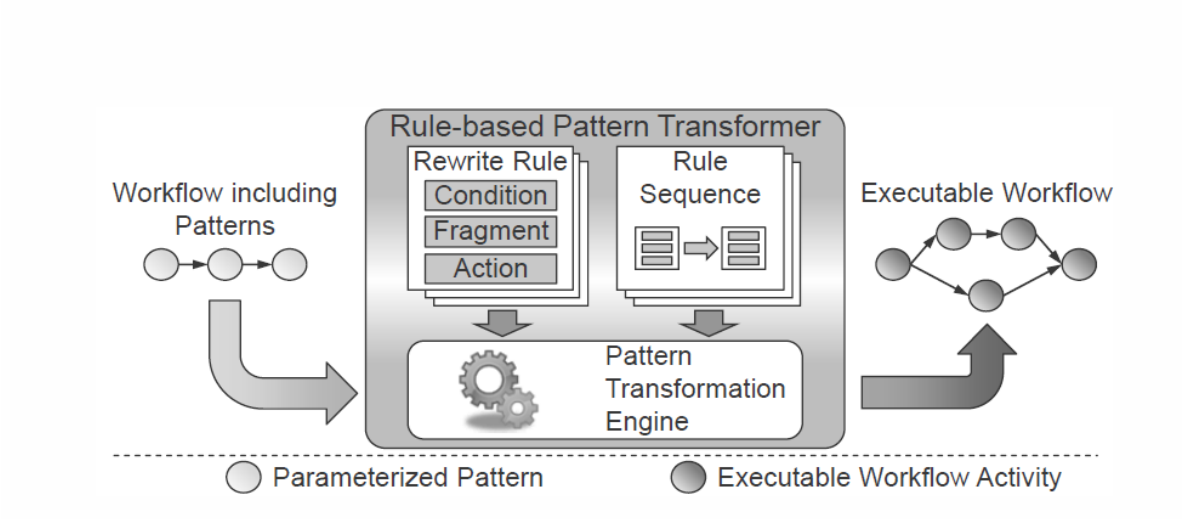

Abbildung 2.10: Modell der Transformation von Datenmanagementpatterns vgl. [RSM14b]

2.4 Ontologien

Unter dem Begriff Ontologie versteht man in der Informatik im Allgemeinen eine formale Beschreibungen eines Wissensbereich bzw. Themengebietes über eine Menge von Begriffen und den Beziehungen zwischen diesen Begriffen. Dieser Abschnitt soll einen kurzen Überblick über das Ressource Description Framework und die darauf aufbauende Ontologiesprache OWL liefern. Für eine genauere Einführung in dieses Thema sei auf die Quellen [HKRS07] und [Stu09] verwiesen.

2.4.1 RDF

Der folgende Abschnitt basiert soweit nicht anders angegeben auf der Quelle [HKRS07]: Das Ressource Description Framework (RDF) [MM04] ist eine formale Sprache, die zur Beschreibung von strukturierten Informationen dient. RDF ist Teil der W3C Recommendation [?] . RDF basiert auf der XML Syntax und verwendet in der Regel Uniform Ressource Identifier (URI) für die Bezeichnung von Ressourcen. Informationen werden in RDF als gerichteter Graph dargestellt. In der Regel wird dieser Graph durch die Menge seiner Kanten beschrieben. Die einzelnen Kanten werden dabei durch sogenannte Tripel beschrieben, wobei die einzelnen Elemente dieser Tripel häufig als Subjekt, Prädikat und Objekt bezeichnet werden. Das Subjekt eines Tripels beschreibt die Ressource über die eine Aussage gemacht wird. Das Subjekt wird durch eine URI beschrieben und kann kein Literal sein. Das Prädikat eines Tripels beschreibt die Beziehung zwischen dem Subjekt und dem Objekt. Das Objekt eines Tripels wird entweder durch eine URI oder durch ein Literal beschrieben (zum Beispiel ein String, oder eine Zahl).

Abbildung 2.11 zeigt ein einfaches Beispiel eines RDF Graphen. Die Runden Felder in der Abbildung stehen für Datenressourcen, welche über ihre jeweilige URI eindeutig gekennzeichnet sind, die eckigen Felder für Literale und Kanten für die Beziehungen zwischen verschiedenen

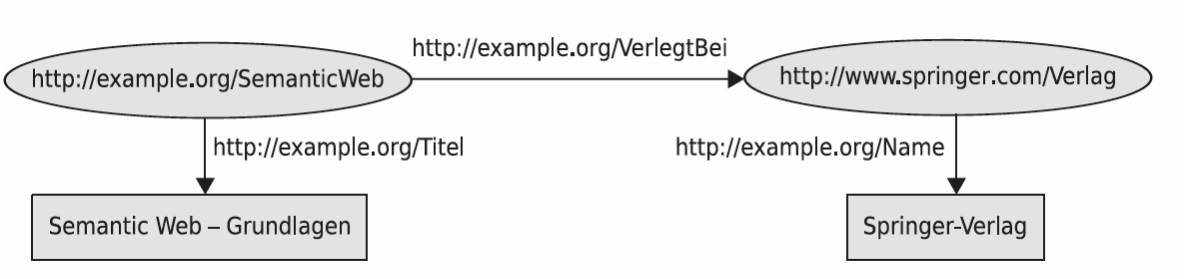

Abbildung 2.11: Einfacher RDF Graph mit Literalen. Quelle: [HKRS07]

Ressourcen. Eines der Tripel dieses Graphen wäre zum Beispiel http://example.org/SemanticWeb, http://example.org/VerleigtBei, http://www.springer.com/Verlag. Dieses Tripel bringt zum Ausdruck, dass die Ressource SemanticWeb in der Beziehung VerlegtBei mit der Ressource Springer Verlag steht. Dabei ist in diesem Tripel ist die Ressource Semantic Web das Subjekt, VerlegtBei das Prädikat und der Verlag Springer das Objekt. Solche Beziehungen werden oft auch als Eigenschaften (english Properties) bezeichnet.

2.4.2 RDFS

Wie im vorigen Abschnitt beschrieben bietet RDF die Möglichkeit Aussagen über individuelle Ressourcen zu machen. RDF bietet allerdings keine Möglichkeit Aussagen über Klassen von Ressourcen beschreiben zu können. Für diesen Zweck wurde das Ressource Description Framework Schma (RDFS) entwickelt. RDFS ist eine weitere Recommendation des W3C [w3c] und ermöglicht es ein Vokabular (bzw. eine Ontologie) auf der Basis von RDF zu definieren. Die Beziehung zwischen RDF und RDFS ähnelt also der Beziehung zwischen XML und XML Schema.

Die wichtigsten Elemente für die Definition von Klassen sind:

Class Übergeordneter Klassenbegriff. Erlaubt die Definition eigener Klassen.

Resource Die Klasse aller Ressourcen. Instanz von Class

property Basisklasse für die Beschreibung von Eigenschaften. Instanz von Resource

Literal Klasse für Literalwerte (Strings, Zahlen, etc.)

RDFS enthält auch eine Reihe von Elementen für die Beschreibung von Eigenschaften:

range Mithilfe von Range kann der Typ des Objekts einer Eigenschaft festgelegt werden.

domain Legt den Typ des Subjekts einer Eigenschaft fest

subClassOf Erlaubt die Definition von Unterklassen

subPropertyOf Erlaubt Vererbung bei Eigenschaften

2.4.3 OWL

RDFS erlaubt den Aufbau von einfachen Ontologien. Viele komplexere Themengebiete lassen sich in RDFS allerdings nicht hinreichend beschreiben. Zu diesem Zweck wurde die Web ontology Language (OWL) entwickelt. Die Web Ontology Language ist eine auf RDF aufbauende formale Beschreibungssprache für Ontologien. OWL ist genauso wie RDf und RDFS seit ein Teil der Recommendation des W3C [w3c]. Im folgenden sollen die wichtigsten Sprachkonstrukte in OWL erklärt werden.

Wie bereits RDFS besteht die Semantik von OWL im wesentlichen Aus Klassen und Eigenschaften. Im Gegensatz zu RDFS bietet OWL jedoch deutlich umfangreichere Werkzeuge für die Beschreibung von Beziehungen zwischen Klassen. Für die Beschreibung von Klassen existiert in OWL analog zu RDFS das Element owl:Class. Für Instanzen von Klassen existiert in OWL zudem die Oberklasse owl:Thing. Im Gegensatz zu RDFS bietet OWL jedoch deutlich mehr Möglichkeiten für die Bildung von Klassenhierarchien; so können Klassen zum Beispiel zusätzlich zu Unterklassen auch als Komplement, Schnittmenge, Aufzählung der Instanzen und durch Einschränkung anderer Klassen definiert werden.

Auch bei der Beschreibung von Eigenschaften unterscheidet sich OWL von der RDFS Syntax. In OWL werden Eigenschaften in zwei Klassen geteilt: ObjectProperties beschreiben Beziehungen zwischen Instanzen zweier Klassen. DatatypeProperties hingegen beschreiben Beziehungen zwischen einer Instanz einer Klasse und einem konkreten Literal, oder Datentyp.

Wie auch bereits in RDFS können über ranges und domains beschrieben werden auf welche Kalssen sich Eigenschaften beziehen. Zusätzlich können Eigenschaften beschränkt werden (owl:Restriction). Zu den wichtigsten Einschränkungen zählen:

owl:allValuesFrom Alle Werte einer Eigenschaft müssen Instanzen der angegebenen Klasse sein

owl:someValuesFrom Mindestens ein Wert einer Eigenschaft muss eine Instanz der angegebenen Klasse sein

owl:min/maxCardinality Beschreibt die mininmale/maximale Kardinalität einer Eigenschaft

owl:hasValue Eigenschaft muss einen bestimmten Wert haben

Die höhere Ausdrucksfähigkeit von OWL bringt aber auch Nachteile. SO können OWL Ontologien bei entsprechender Komplexität schlecht skalierbar und unentscheidbar sein. Aus diesem Grund wurden für den praktischen Einsatz der Sprache drei Teilsprachen definiert: OWL Lite, OWL DL und OWL Full. Im folgenden werden zu jeder dieser Teilsprachen kurz die wichtigsten Eigenschaften genannt. Für nähere Informationen sei auf die Quelle [HKRS07] verwiesen:

OWL Lite

- niedrigste Ausdrucksfähigkeit
- entscheidbar
- Teilsprache von OWL DL und OWL Full

- Einschränkung von OWL DL. Unter anderem Kardinalitäten nur 0 oder 1.

OWL DL

- mittlere Ausdrucksfähigkeit
- entscheidbar
- Teilsprache von OWL Full
- Einschränkung von OWL Full. Unter anderem dürfen Klassen keine Instanzen von Klassen sein

OWL Full

- größte Ausdrucksfähigkeit
- unentscheidbar
- keine Einschränkungen

3 Bestandsaufnahme

Dieses Kapitel beschäftigt sich näher mit der bereits bestehenden prototypischen Implementierung des SIMPL Rahmenwerks und der Patterntransformation. Das SIMPL Rahmenwerk wurde bereits in einer Reihe von Arbeiten prototypisch implementiert [sim]. Für die Implementierung wurde die BPEL Engine Apache ODE [apa] und das Modellierungstool Eclipse BPEL Designer [bpl] verwendet. In den folgenden Abschnitten wird vor allem auf die erweiterte BPEL Designer Umgebung, sowie die derzeitige Umsetzung der Patterntransformation während der Modellierungszeit näher eingegangen. Außerdem wird der Workflow für die biomechanische Simulation von Strukturänderungen in Knochen, welcher für die Evaluation dieser Arbeit verwendet wird, näher erklärt.

3.1 Erweiterung des Eclipse BPEL Designers

Um die zusätzlichen Funktionen des SIMPL Rahmenwerks (siehe Abschnitt 2.3) zu unterstützen wurde der Eclipse BPEl Designer erweitert. Dabei wurde die Designpalette der graphischen Benutzerüberfläche um zusätzliche Elemente für den Einsatz von DM Aktivitäten und Datenmanagementpatterns zur Modellierung von Workflows ergänzt.

Abbildung 3.1 zeigt einen Screenshot der Benutzeroberfläche mit Fokus auf die für die DM Aktivitäten relevanten Erweiterungen. Der rote Rahmen markiert die zusätzlichen Elemente innerhalb der Palette von Aktivitäten, die für die Workflowmodellierung genutzt werden können. Der Reiter DM Aktivitäten umfasst folgende Elemente, beziehungsweise Erweiterungsaktivitäten:

QueryData

IssueCommand

RetrieveData

WriteDataBack

TransferData

Für die Parametrisierung dieser Aktivitäten wurde jeweils eine property View implementiert (blauer Rahmen in Abbildung 3.1). In dieser Property View können die Eingabeparameter aus einer Drop Down Liste ausgewählt sowie die DM Commands spezifiziert werden. Die möglichen Variablen beschränken sich dabei auf die vom Nutzer innerhalb der Designumgebung erstellten Variablen (grüner Rahmen).

Für die Modellierung von Workflows existiert der Reiter Data Management Patterns (siehe Abbildung 3.2 roter Rahmen). Die bereits implementierten Patterns umfassen:

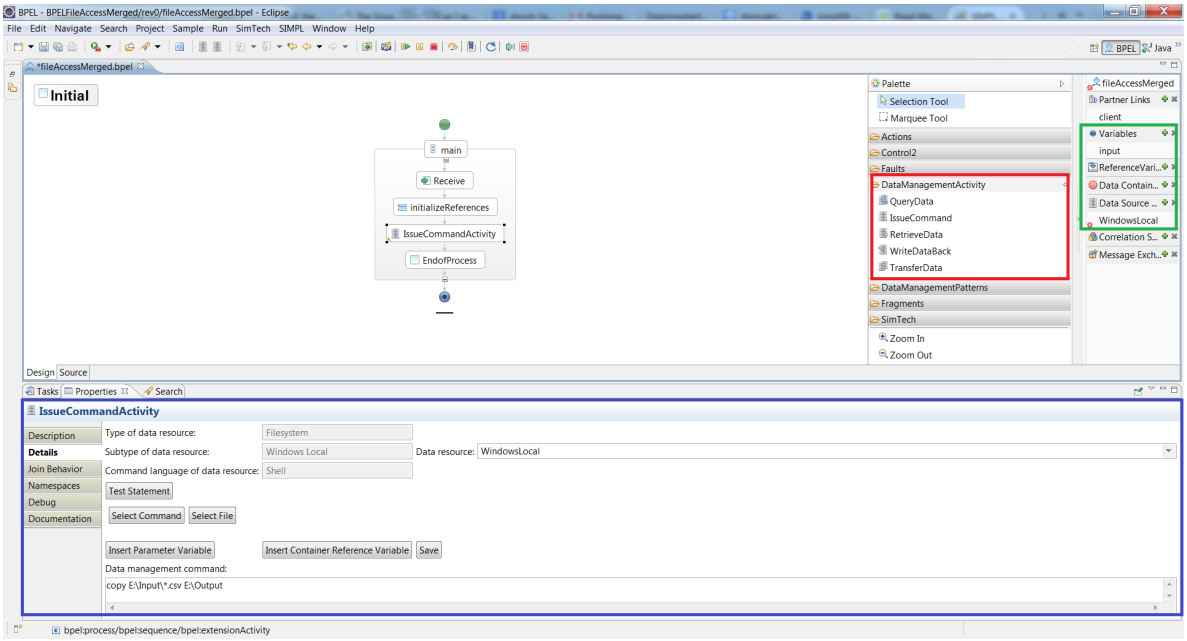

Abbildung 3.1: Data Management Aktivitäten im BPEL Designer

ContainerToContainerPattern

DataTransferAndTransformationPattern

GetTimeStepPattern

ParallelDataIterationPattern

SequentialDataIterationPattern

SimulationModelCouplingPattern

SimulationModelRealizationPattern

SimulationOrientedDataProvisioningPattern

Auch die Datenmanagementpatterns können über eine property Section parametrisiert werden. Abbildung 3.2 zeigt hier besipielsweise die Propery View des Container to Container Patterns (blauer Rahmen).

Für die Durchführung der Patterntransformation während der Modellierung eines Workflows wurde das BPEL Designertool zudem um die Schaltflächen TranformAllPatterns (grüner Rahmen in 3.2) und die Option TransformThisPattern innerhalb des Kontextmenüs bei einem Rechtsklick auf ein Datenmanagementpattern erweitert. Für beide Varianten der Patterntransformation gibt es zudem die Option diese Transformation rekursiv durchzuführen. Dabei werden die transformierten Workflowfragmente rekursiv auf weitere eingebettete Datenmanagementpatterns durchsucht, bis zuletzt ein ausführbares Workflowfragment entsteht.

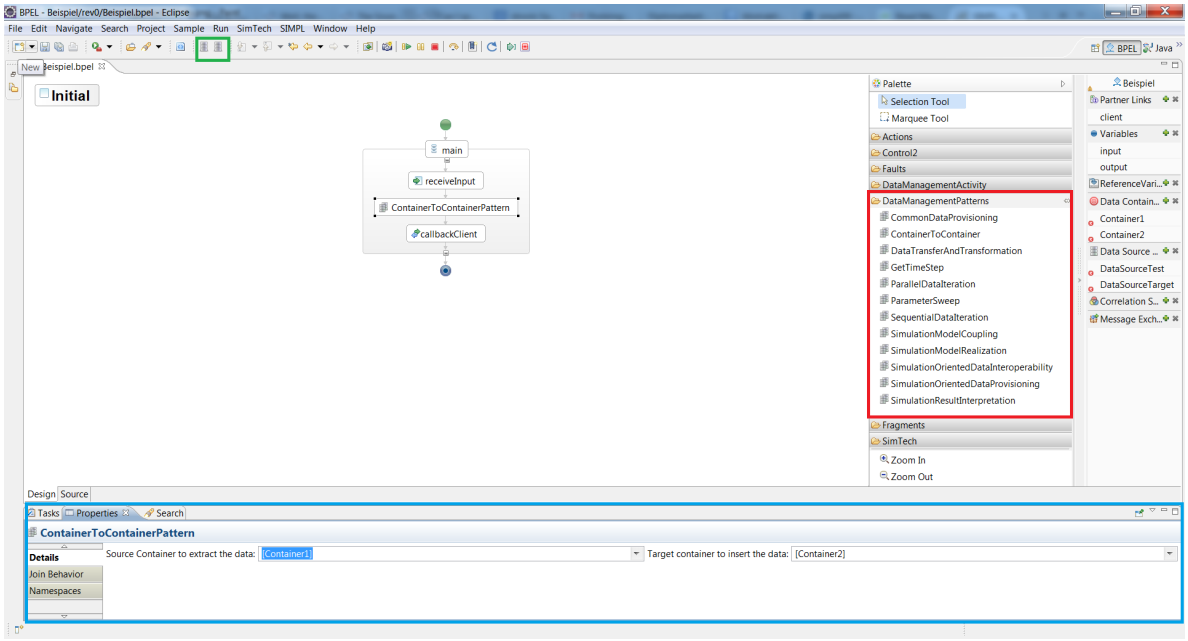

Abbildung 3.2: Datenmanagement Patterns im BPEL Designer

Da die Patterntransformation während der Modellierung nicht rückgängig gemacht werden kann, wird der Nutzer vor der Transformation gefragt, ob er eine neue Revision des Projekts in einem neuen Unterordner erstellen möchte. Wenn diese Option ausgewählt wird, erzeugt die Patterntransformation eine neue Revision des Projekts mit dem entsprechend transformierten Workflow. Andernfalls wird die derzeitige Revision des Projekts verändert.

3.2 Bisherige Umsetzung des Patterntransformation zur Modellierungszeit

In dem Abschnitt 2.3.2 des Grundlagenkapitels dieser Arbeit wurde bereits das zugrundeliegende Modell der Patterntransformation beschrieben. In diesem Abschnitt wird nun näher auf die bestehende Umsetzung dieser Transformation zur Modellierungszeit eines Workflows eingegangen

Als Teil der Diplomarbeit von Henrik Pietranek [Pie12] wurde die Patterntransformation zur Modellierungszeit prototypisch implementiert. Diese Implementierung fand in Form von Java Klassen und Java Methoden innerhalb des Eclipse Plugins *org.eclipse.simpl.ui* statt. Wie bereits zuvor erwähnt erfolgt die Transformation von Datenmanagementpatterns innerhalb des erweiterten Eclipse BPEL Designers über die Schaltflächen *Transform all patterns* bzw. *Transform all patterns (no recursion)*, oder über den Menüeintrag *Transform this pattern* bzw. *Transform this pattern (no recursion)* innerhalb des Kontextmenüs eines Datenmanagementpatterns.

Wenn ein Nutzer die Option *Transform all patterns* auswählt, dann wird die Methode *transformAllPatterns* der Klasse *PatternTransformationImplementation* aufgerufen. Anschließend wird die Methode

traverse der Klasse *WorkflowGraphTraverser* aufgerufen. In dieser Methode wird der Workflowgraph mit den zu transformierenden Datenmanagementpatterns zunächst in ein JDOM Dokument überführt. Dies geschieht, damit die Baumstruktur dieses Dokuments anschließend durch die Methode *traverse* traversiert und die einzelnen Elemente des Baums dahingehend überprüft werden, ob es sich bei ihnen um ein Datenmanagementpattern handelt.

Wenn diese Methode *traverse* ein Datenmanagementpattern findet, dann wird die Methode *transform* der Klasse *Patterntransformer* aufgerufen. Diese Methode implementiert die eigentliche Patterntransformation. Zunächst wird die Kontrollstrategie für das zu transformierende Pattern geladen. Eine Kontrollstrategie beschreibt dabei die Reihenfolge, in der die für ein Datenmanagementpattern implementierten Transformationsregeln auf ihre Anwendbarkeit überprüft werden. Die Kontrollstrategien entsprechen damit den in Abschnitt 2.3.2 beschriebenen Regelsequenzen, tragen aber noch den Namen aus einer früheren Version des Konzepts für die Patterntransformation. Die Kontrollstrategien sind bei dieser Implementierung in Form von Javaklassen umgesetzt worden. Eine Kontrollstrategie muss das in Listing 3.1 dargestellte Interface implementieren. Die Methoden dieses interfaces dienen bei der Transformation dazu herauszufinden, ob eine Kontrollstrategie noch weitere Transformationsregeln umfasst (*getCountRules() und hasNextRule()*) und diese dann gegebenenfalls aufzurufen (*hasNextRule()*).

```
public interface ControllStrategy {
      public int getCountRules();

      public boolean hasNextRule();

      public TransformationRule getNextRule();
}
```

Listing 3.1: Interface der Kontrollstrategien Quelle: [Pie12]

Auch die einzelnen Transformationregeln wurden als Java Klassen umgesetzt. Listing 3.2 zeigt das Interface, dass alle Transformationsregeln implementieren müssen.

```
public interface TransformationRule {
      public boolean evaluateConditionPart(Element dmp, Element dsVar,
            Element conVar, Element var) throws PatternTransformationException;

      public Element applyActionPart(Element dmp, Element dsVar, Element conVar,
            Element var, IFile iFile) throws PatternTransformationException;
}
```

Listing 3.2: Interface der Transformationsregeln Quelle: [Pie12]

Die beiden Methoden *evaluateConditionPart* und *applyActionPart* setzen dabei jeweils den in Abschnitt 2.3.2 beschriebenen Action und Condition Part einer Transformationsregel um. Der Fragment Part einer Regel, also das Workflowfragment auf das die Regel abbildet, wird innerhalb eines Repositories verwaltet. Die Methode *evaluateConditionPart* prüft, ob eine Transformationsregel angewendet werden kann. Bei der Patterntransformation wird also gemäß der durch die Kontrollstrategie festgelegten Reihenfolge nacheinander die Methode *evaluateConditionPart* der jeweils nächsten Transformationsregel aufgerufen, bis eine passende Regel gefunden wurde. Wenn keine geeignete Regel gefunden wurde, wirft die Patterntransformation einen entsprechenden Fehler. Sollte eine geeignete Regel

gefunden werden, wird die Methode *applyActionPart* der jeweiligen Transformationsregel ausgeführt. Innerhalb dieser Methode wird das passende Workflowfragment für das zu transformierende Datenmanagementpattern aus einem Fragmento Repository [1] geladen und die Platzhalter innerhalb dieses Workflowfragments werden mit den passenden Variablenwerten initialisiert. Wenn der Nutzer zuvor die Option *Transform all patterns (no recursion)* ausgewählt hat, ist die Transformation dieses Patterns damit beendet. Bei einer rekursiven Transformation wird nun im Anschluss noch einmal die Methode *traverse* auf dieses transformierte Workflowfragment angewendet, um rekursiv darin beeinhaltete Datenmanagementpatterns zu transformieren.

Zuletzt wird nun das zu transformierende Datenmanagementpattern Element innerhalb des ursprünglichen Workflows durch das transformierte Workflowfragment ersetzt. Dieser Vorgang wiederholt sich für jedes gefundene Datenmanagementpattern innerhalb des Workflows.

Wenn der Nutzer die Option *Transform this pattern* auswählt wird stadtdessen die Methode *transformSinglePattern* innerhalb der Klasse *PatternTransformationImplementation* aufgerufen. Auch hier wird der Workflowgraph wieder durch den *WorkflowGraphTraverser* traversiert, allerdings wird diesmal nur das ausgewählte Datenmanagementpattern transformiert.

Zum Schluss wird in beiden Fällen der Patterntransformation der transformierte Workflow gespeichert.

3.3 Knochensimulation

Für die Evaluation dieser Arbeit wird ein Workflow für die biomechanische Simulation von Strukturänderungen in Knochen aus der Studienarbeit [Boh14] verwendet. Im folgenden Abschnitt wird daher diese Simulation sowie die Implementierung dieser Simulation in Form eines Workflows mit Datenmanagementpatterns eingehender beschrieben.

Diese Simulation wurde zunächst von Krause et al. [KSR+11] beschrieben. Im Rahmen dieser Simulation werden die Auswirkungen von unterschiedlichen Belastungsmustern auf die Struktur von Knochen simuliert.

Die Simulation lässt sich in zwei Bereiche aufteilen. Der biomechanische Bereich der Simulation simuliert makroskopische Veränderungen des Knochengewebes in Form des Massenaustausches zwischen porösen Festkörpern und darin enthaltenen Flüssigkeiten. Der systembiologische Bereich simuliert Gewebebildungsprozesse auf der mikroskopischen Ebene der Zellen und deren Interaktionen. Bei der Simulation werden unterschiedliche Bewegungssequenzen verwendet, welche den typischen Bewegungsabläufen der relevanten Person entsprechen.

Abbildung 3.3 zeigt die Kopplung der beiden Simulationsmodelle. Als Simulationswerkzeuge werden bei der Implementierung dieser Simulation das Pandas Rahmenwerk [2] für die biomechanische

[1] http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm
[2] http://www.mechbau.uni-stuttgart.de/pandas/index.php

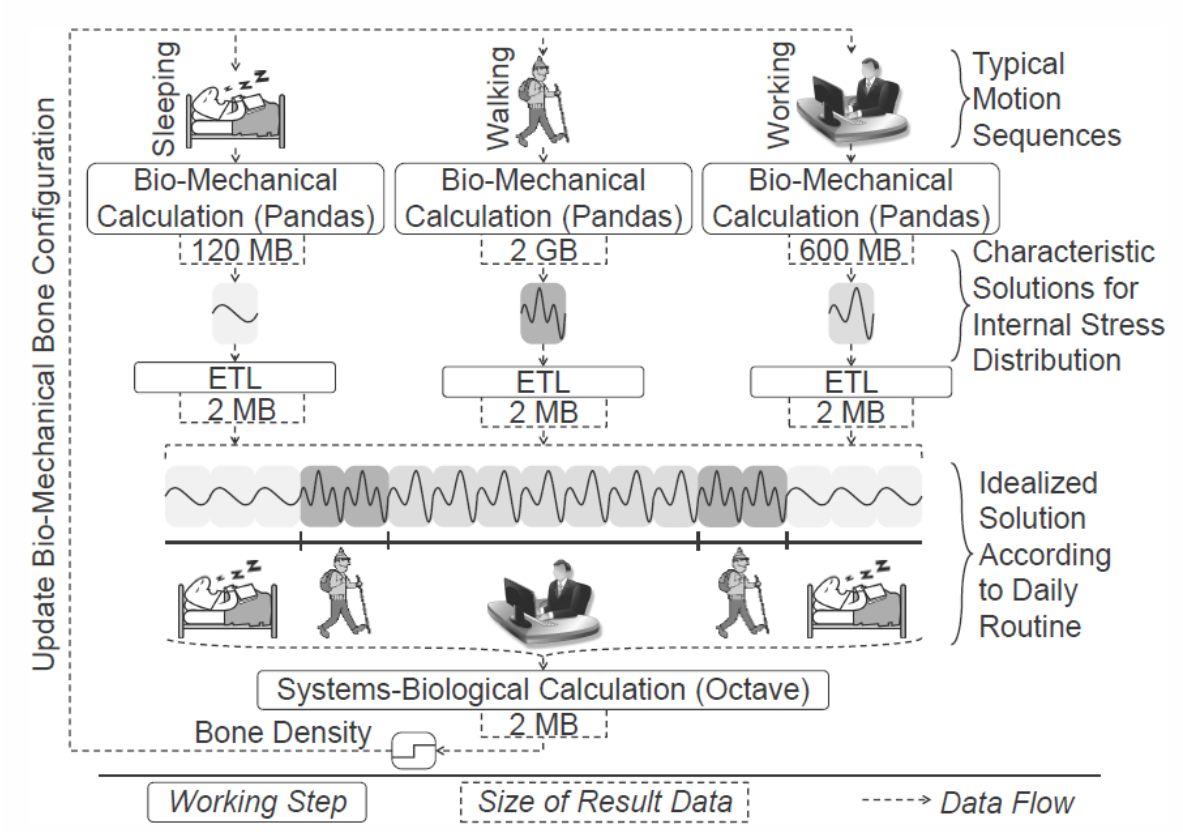

Abbildung 3.3: Gekoppelte Simulation von Strukturänderungen in Knochen Quelle: [RSM14b]

Simulation und GNU Octave [3] für die systembiologische Simulation verwendet. In der biomechanischen Simulation durch Pandas wird aus den unterschiedlichen Bewegungssequenzen, bzw. den entsprechenden Belastungsmustern, jeweils eine charakteristische Belastungsverteilung innerhalb der Knochenstruktur errechnet. Das bei dieser Simulation anfallende Datenvolumen reicht von einigen 100 Megabyte bis zu mehreren Gigabyte an Daten, die in einer SQL Datenbank gespeichert werden.

Bevor die systembiologische Simulation genutzt werden kann, um das Ergebnis der biomechanischen Simulation zu verfeinern, werden verschiedene ETL (Extract, Transform, Load) Prozesse auf die Ergebnisse der biomechanischen Simulation angewandt. Dabei wird aus den einzelnen Belastungsverteilungen der Bewegungssequenzen ein durchschnittliches tägliches Belastungsprofil einer Person erzeugt. Außerdem wird das Datenvolumen aufgeteilt, um die parallele Verarbeitung auf mehreren Octave Instanzen zu ermöglichen. Die Eingabedaten für die Octave Instanzen werden als CSV (Comma Seperated Value) Dateien bereitgestellt. Diese CSV Dateien werden dann von Octave genutzt, um die systembiologische Simulation der Veränderungen in der Dichte des Knochens unter dem täglichen Belastungsprofil durchzuführen.

[3] http://www.gnu.org/software/octave/

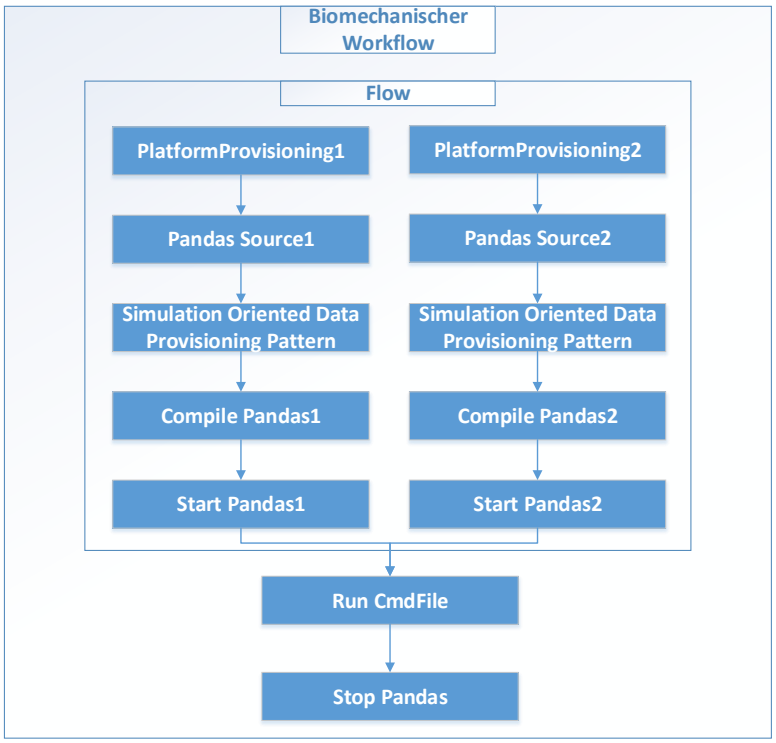

Abbildung 3.4: Aufbau des biomechanischen Workflows Quelle: [Boh14]

Zuletzt werden die Ergebnisse der systembiologischen Simulation genutzt, um die Eingabeparameter der biomechanischen Simulation für den nächsten Zeitschritt (beispielsweise den nächsten Tag) anzupassen. Anschließend wird dieser Vorgang für die Simulation weiterer Tage wiederholt.

3.3.1 Biomechanischer Workflow zur Simulation von Strukturänderungen in Knochen

Die zuvor beschriebene biomechanische Simulation unter Verwendung des Pandas Rahmenwerks wurde als Teil der Diplomarbeit von Henrik Pietranek [Pie12] als BPEL Workflow implementiert und im Rahmen der Arbeit [Boh14] an einen veränderten Simulationsablauf angepasst und zusätzliche Datenmanagementpatterns für die Datenbereitstellung innerhalb dieses Workflows implementiert. Dieser Workflow wird im Rahmen dieser Arbeit für die Auswertung der Patterntransformation zur Laufzeit als Testworkflow verwendet.

Abbildung 3.4 zeigt den Aufbau des biomechanischen Workflows. In dieser Umsetzung wird die Simulation durch zwei gekoppelte Pandas Instanzen durchgeführt. Um diese Simulationsinstanzen zu erzeugen, wird zunächst im ersten Workflowschritt, dem Platform Provisioning, ein Webservice aufgerufen, um auf dem Zielsystem eine neues Verzeichnis für eine Simulationsinstanz zu erzeugen

(PlatformProvisioning). Dabei liefert der Webservice als Rückgabe eine entsprechende SimID für diese Simulationsinstanz zurück. Im nächsten Workflowschritt wird dann der Pandas Sourcecode in das zuvor erzeugte Verzeichnis kopiert (Pandas Source). Danach erfolgt die Datenbereitstellung. Dabei wird das simulationsorientierte Data Provisioning Pattern für die Datenbereitstellung genutzt. Dieses Pattern verwendet eine Ontologie des verwendeten Simulationsmodell, um für die Pandas Simulation benötigten mathematischen Variablen jeweils die geeigneten konkreten Dateinamen und Dateipfade zu ermitteln, damit diese dann in Patterns niedriger Abstraktionsschichten auf dem Zielsystem bereitgestellt werden können. Im Anschluss wird Pandas kompiliert (Compile Pandas). Zuletzt wird dann die Simulationsinstanz gestartet (Start Pandas). Um die Performanz zu erhöhen erfolgen diese Workflowschritte für die beiden Simulationsinstanzen parallel. Nachdem die beiden Simulationsinstanzen gestartet wurden, wird die eigentliche gekoppelte Berechnung der beiden Simulationsinstanzen gestartet (Run CmdFile). Die Ergebnisse der Simulation werden dabei in einer Postgres Datenbank[4] gespeichert. Nachdem die Berechnung abgeschlossen ist, werden zum Schluss noch die beiden Simulationsinstanzen angehalten (Stop Pandas).

[4] http://www.postgresql.org/

4 Motivation für eine Patterntransformation zur Laufzeit

Dieses Kapitel beschäftigt sich mit der Motivation für eine Patterntransformation zur Laufzeit. Zu diesem Zweck werden zuerst allgemeine Vorteile einer solchen Transformation zur Laufzeit im Vergleich zu der bereits prototypisch implementierten Transformation während der Modellierung des Workflows aufgeführt. Außerdem werden mögliche Anwendungsszenarien und konkrete Beispiele aus den Arbeiten [Boh14] und [Pie12] aufgezeigt.

Bei dem bisherigen Prototyp der Patterntransformation wird der Transformationsvorgang innerhalb der Entwicklungsumgebung von dem Wissenschaftler, welcher den Workflow entwirft, ausgelöst. Da die abstrakten Patterns ohne eine Transformation nicht ausführbar sind, muss diese vor der eigentlichen Ausführung des Workflows erfolgen. Dies führt jedoch zu einer Einschränkung der für die Patterntransformation zur Verfügung stehenden Informationen, sofern diese Informationen erst zur Laufzeit des Workflows zur Verfügung stehen. Da die Transformation noch vor dem eigentlichen Beginn der Workflowausführung stattfindet, können Informationen über den Zustand der Workflowausführung vor dem zu transformierenden Pattern nicht berücksichtigt werden. Ein Beispiel für solche Informationen wären aktuelle Variablenwerte, oder auch Informationen über Simulationsinstanzen, die erst zur Laufzeit erzeugt werden.

Eine Patterntransformation zur Laufzeit würde daher große Vorteile für die Patterntransformation mit sich bringen. Bei einer Patterntransformation zur Laufzeit würde die Ausführung des Workflows mit den darin enthaltenen abstrakten Datenmanagementpatterns gestartet. Wenn die Ausführungsengine während der Laufzeit ein solches Pattern erreicht könnte dieses von einer entsprechenden Patterntransformationskomponente genau zu diesem Zeitpunkt transformiert werden. Daher kann bei der Patterntransformation zur Laufzeit der konkrete Zustand des Workflows an diesem Punkt der Ausführung miteinbezogen werden. Im folgenden werden einige mögliche Szenarien aufgelistet, wie sich diese Informationen für die Patterntransformation nutzen lassen.

4.1 Einbeziehung aktueller Variablenwerte

Bei der Transformation von Datenmanagementpatterns während der Modellierung eines Workflows kann nur der Wert einer Variablen bei der Transformation berücksichtigt werden, mit dem diese Variable bei deren Deklaration zu Beginn des Workflows initialisiert wurde. Eine Änderung eines Variablenwertes würde erst zur Laufzeit innerhalb einer Assign Aktivität erfolgen und daher bei der Patterntransformation nicht berücksichtigt werden. Aus diesem Grund wird in der Arbeit von Pietranek [Pie12] davon ausgegangen, dass Variablen bei ihrer Deklaration ein fester Wert zugeordnet wird, der sich bis zur Transformation nicht ändert. Dies führt natürlich zu Einschränkungen bei der

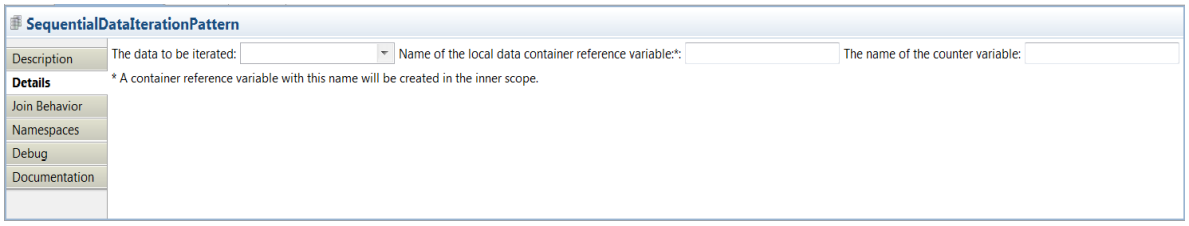

Abbildung 4.1: Property Section des Sequential Data Iteration Patterns

generischen Einsetzbarkeit einiger Patterns zur Modellierung komplexerer Workflows, bei denen Variablenwerte an verschiedenen Stellen innerhalb des Workflows geändert werden können.

Pietranek führt als eine Möglichkeit dieses Problem zu lösen eine Datenflussanalyse auf [Pie12]. Bei einer Datenflussanalyse könnten Änderungen von Variablenwerten durch Assign Aktivitäten innerhalb des Workflows bei der Patterntransformation berücksichtigt werden. Die würde die generische Einsetzbarkeit der Datenmanagementpatterns auch bei einer Patterntransformation während der Workflowmodellierung erhöhen. Allerdings bietet eine Patterntransformation zur Laufzeit des Workflows noch einen weiteren Vorteil, welchen die Datenflussanalyse nicht hat. Bei der Transformation zur Laufzeit können auch solche Variablen berücksichtigt werden, denen ein Wert durch eine externe Komponente, wie zum Beispiel einem Workflow, zugewiesen werden. Einziger Nachteil einer Patterntransformation zur Laufzeit im Vergleich zu der Durchführung einer Datenflussanalyse ist der zusätzliche Overhead bei der Workflowausführung. Auf diesen Overhead der Patterntransformation während der Laufzeit wird in Kapitel **??**näher eingegangen.

Zusammenfassend ermöglicht eine Patterntransformation zur Laufzeit es Änderungen an den Variablenwerten, sowohl durch Assign Aktivitäten als auch durch Webservices, mit in die Patterntransformation einzubeziehen und erhöht damit die universelle Einsetzbarkeit von Datenmanagementpattern.

Ein Beispiel für ein Datenmanagementpattern, dass durch aktuelle Variablenwerte verbessert werden könnte, ist das Container to Container Pattern [Pie12]. Dieses Pattern beschreibt den Transfer von Daten aus einem Datencontainer in einen anderen. Als Eingabeparameter erwartet dieses Datenmanagementpattern daher einen Sourcecontainer, aus dem Daten geladen werden sollen, und einen Targetcontainer, in den die Daten bereitgestellt werden sollen. Je nachdem, ob die Dateiformate dieser Container übereinstimmen oder nicht, wird dieses Pattern auf ein anderes Workflowfragment abgebildet. Dies wird wird während der Transformation dieses Patterns innerhalb der Transformationsregeln überprüft. Wie bereits eingangs erwähnt können bei der Transformation dieses Patterns während der Modellierung nur die Werte, mit denen diese Eingabevariablen instantiiert wurden, berücksichtigt werden. Bei einer Patterntransformation zur Laufzeit können beliebige Änderungen dieser Variablenwerte bei der Patterntransformation berücksichtigt werden. Das bedeutet, dass sich das Dateiformate der beiden Datencontainer während der Ausführung des Workflows verändern können, ohne das dies zu der Anwendung einer falschen Transformationsregel und damit zu einer fehlerhaften Abbildung des Patterns führt.

Ein weiteres Beispiel ist das Sequential Data Iteration Pattern aus [Pie12]. Bei diesem Pattern wird über eine Liste aus Container Referenz Variablen iteriert. Abbildung 4.1 zeigt die Property Seciton dieses

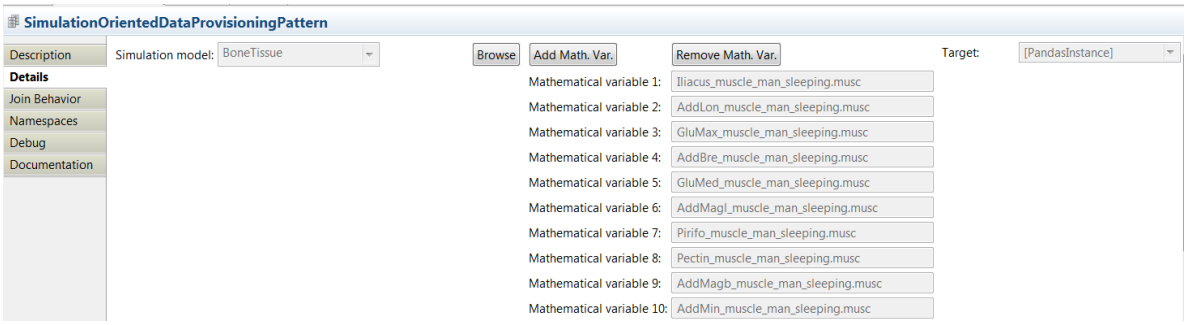

Abbildung 4.2: Property Section des simulationsorientierten Data Provisioning Patterns

Patterns innerhalb der Entwicklungsumgebung. Als Eingabeoparameter benötigt das Sequential Data Iteration Pattern eine Liste von Container Referenz Variablen und eine einzelne Container Referenz Variable.

In der derzeitigen Implementierung, auf welche dieses Pattern abgebildet wird, wird in jeder Iteration ein Eintrag aus der Liste von Container Referenzen ausgewählt und der lokalen Container Referenz Variable zugewiesen. Da das Sequential Data Iteration Pattern in den derzeit implementierten Anwendungsfällen ein eingebettetes Container to Container Pattern enthält, muss diese lokale Container Referenz Variable initialisiert werden, da das eingebettete Pattern sonst nicht auf diese Variable zugreifen kann. Aus diesem Grund ist eine Einschränkung dieses Patterns, dass alle Referenzvariablen in der Liste das gleiche Datenformat haben müssen. Würde sich das Datenformat der Referenzvariablen zwischen den Iterationen ändern, würde das eingebettete Datenmananagementpattern zur Modellierungszeit falsch abgebildet, da in diesem Fall nur das Dateiformat der ersten Referenzvariablen in der Liste bei der Transformation berücksichtigt werden würde. Zudem kann die Liste von Container Referenzvariablen zur Laufzeit nicht erweitert werden, da bei der Transformation eine lokale Kopie der Liste erstellt wird.

Eine Transformation zur Laufzeit würde laut Pietranek [Pie12] dieses Problem lösen, da das eingebettete Pattern während jeder Iteration neu transformiert werden könnte. Dadurch wäre das Sequential Data Iteration Pattern noch generischer einsetzbar, da auch über Datensätze mit unterschiedlichen Datenformaten iteriert werden kann.

4.2 Einbeziehung von Komponenten zur Laufzeit

Im vorigen Abschnitt wurde bereits darauf hingewiesen, dass bei einer Patterntransformation zur Laufzeit auch Informationen von anderen Komponenten, wie zum Beispiel Webservices, mit in die Patterntransformation miteinbezogen werden können. Ein Beispiel für ein Szenario, wo dies nützlich wäre, ist das simulationsorientierte Data Provisioning Pattern, welches in der Arbeit [Boh14] genauer beschrieben wird.

Abbildung 4.2 zeigt den Property Bereich dieses Datenmanagementpatterns innerhalb der Entwicklungsumgebung. Das Simulationsorientierte Data Provisioning Pattern zählt innerhalb des Patternhierarchie zu den simulationsorientierten Data Management Patterns und abstrahiert die Prozesse

der Datenbereitstellung. Als Parameter benötigt dieses Pattern ein Simulationsmodell in Form einer Ontologie, eine oder mehrere mathematische Variablen aus dieser Ontologie, die bereitgestellt werden sollen und eine Zielinstanz für die Datenbereitstellung. In der Praxis wird eben diese Zielinstanz, also die zugrundeliegende Instanz einer Simulationssoftware, oft erst zur Laufzeit innerhalb des Simulationsworkflows erzeugt. Während der Patterntransformation zur Modellierungszeit ist es daher unmöglich zu überprüfen, ob diese Zielinstanz tatsächlich existiert. Dies kann während der Ausführung des Workflows zu Fehlern führen, wenn die Zielinstanz nicht korrekt gestartet wurde.

Bei der Patterntransformation zur Laufzeit ließe sich dieses Problem lösen, indem bei der Patterntransformation zum Beispiel auf ein Webservice Repository aller laufenden Simulationsinstanzen zurückgegriffen wird. Durch einen solchen Webservice könnte leicht überprüft werden, ob die in den Eingabeparametern des Patterns angegebene Zielinstanz tatsächlich existiert.

4.3 Manuelle Anpassung der Patterntransformation zur Laufzeit

Wissenschaftliche Workflows beinhalten oft aufwendige und langwierige Simulationen. Oft ist es bei der Ausführung langwieriger Simulationen daher wünschenswert, das an bestimmten Punkten in der Ausführung des Simulationsworkflows für den Wissenschaftler die Möglichkeit besteht, das weitere Vorgehen der Ausführung zu verändern. Für solche Anwendungsfälle wurden bereits mehrere Frameworks für die Injektion von Workflowfragmenten in ein laufendes Prozessmodell entwickelt [Sch11] [Hum11] [Bia14].

Um einen derartigen Simulationsansatz mit Datenmanagementpatterns zu unterstützen, ist logischerweise auch eine Patterntransformation zur Laufzeit zwingend notwendig. Ansonsten müssten Wissenschaftler die Patterns zur Modellierungszeit transformieren und anschließende die ausführbaren und komplexen Workflows anpassen. Dann könnten sie aber die Abstraktion, welche ihnen die Patterns bieten nicht mehr ausnutzen. Es wäre daher sinnvoller, wenn Wissenschaftler die patternbasierten und abstrakten Workflows anpassen können und wenn die entsprechenden Änderungen durch eine Patterntransformation zur Laufzeit auf die ausführbaren Workflows übertragen werden.

4.4 Optimierung der Patterntransformation

Ein weiteres Szenario bei dem eine Patterntransformation zur Laufzeit nützlich wäre ist die Optimierung der Patterntransformation. Um für bestimmte Patterns möglichst effiziente Workflowfragmente auswählen zu können, sind Informationen aus der Laufzeit des Workflows nützlich. Beispielsweise können bei einer Patterntransformation zur Laufzeit konkrete Informationen über die Größe und Komplexität zu bearbeitender Daten für die Auswahl möglichst effizienter Patterns, beziehungsweise ausführbarer Workflowfragmente oder Services verwendet werden.

5 Auswahl und Bewertung existierender Technologien für die Umsetzung der Patterntransformation

Wie bereits in der Einleitung dieser Arbeit erwähnt sollen für die Umsetzung der Patterntransformation zur Laufzeit geeignete Technologien untersucht und ausgewählt werden. Dies umfasst zum einen ein Framework für die Injektion von Workflowfragmenten zur Laufzeit, eine Regelengine für die Trennung der Transformationsregeln sowie deren Evaluation von der eigentlichen Patterntransformationskomponente und ein Triplestore für die Verwaltung der simulationsbezogenen Ontologien.

In diesem Kapitel werden zunächst die Anforderungen an die Patterntranformationskomponente, die bereits im Kapitel Einführung erwähnt wurden, näher konkretisiert. Im Anschluss werden drei verschiedene Frameworks für eine Injektion von Workflofragmenten vorgestellt und auf ihre Eignung für diese Arbeit bewertet. Danach werden zwei mögliche Regelengines für die Auswertung von Tranformationsregeln verglichen. Zuletzt wird in diesem Kapitel der Nutzen eines Triplestores für die Verwaltung von Ontologien kurz erläutert.

5.1 Anforderungen an die Patterntransformationskomponente zur Laufzeit

In Kapitel 1 werden bereits einige Anforderungen an die im Rahmen dieser Arbeit zu implementierende Patterntransformationskomponente definiert. Im folgenden Abschnitt werden nun die Anforderungen konkretisiert sowie einige weitere Anforderungen an die Patterntransformationskomponente aufgestellt.

- Die Patterntransformation zur Laufzeit des Workflows soll möglichst effizient sein, das heißt einen möglichst geringen overhead produzieren. Um dies zu gewährleisten müssen insbesondere auch die Injektion von Workflowfragmenten, die Auswertung der Transformationsregeln durch eine Regelengine und das Abrufen simulationsbezogener Informationen aus Ontologien möglichst performant sein.
- Das Erstellen und Bearbeiten neuer Transformationsregeln in der Regelengine sollte im Idealfall einfacher, zumindest aber nicht schwerer als bisher sein. Auch der Zugriff auf die simulationsbezogenen Ontologien sollte möglichst vereinfacht werden.
- Zudem sollten Transformationsregeln hinzugefügt und bearbeitet werden können, ohne dass die Patterntransformationskomponente dabei jedes mal neu kompiliert werden muss.

- Um die Transformationsregeln und Workflowfragmente möglichst generisch zu halten sollte die Injektion zur Laufzeit der Injektion zur Modellierungszeit ähneln. Im Idealfall sollte der Ablauf der Patterntransformation zu beiden Zeitpunkten identisch sein.

5.2 Vergleich unterschiedlicher Frameworks für die Injektion von Workflowfragmenten zur Laufzeit

Im folgenden Abschnitt werden nun drei mögliche Frameworks vorgestellt und auf ihre Eignung gemäß der zuvor festgelegten Anforderungen bewertet

5.2.1 Diplomarbeit Nr. 3121 Unterstützung des „Model-as-you-go“-Ansatzes durch Modell-Versionierung und Instanzmigration

Das erste Framework wurde im Rahmen der Diplomarbeit [Sch11] entwickelt. Das wichtigste Konzept dieser Arbeit ist die sogenannte Deploy New Version Strategie. Diese Funktionalität erlaubt es dem Nutzer, ein Prozessmodell zu verändern und alle bestehenden Prozessinstanzen des alten Prozessmodells noch während ihrer Ausführung zu pausieren, auf das neue Prozessmodell zu migrieren und im Anschluss weiterlaufen zu lassen. Insbesondere ist es bei diesem Ansatz auch möglich, bereits beendete Instanzen auf ein neues Prozessmodell umzuziehen. Dafür wurde im Lebenszyklus ein zusätzlicher Zustand SUSPENDED eingeführt, in den alle Prozessinstanzen nach ihrer Ausführung übergehen. Bei diesem Ansatz wird der derzeitige Stand der Ausführung der Prozessinstanz als Wavefront bezeichnet. Alle Elemente des Workflows, die sich bereits hinter der Wavefront befinden, wurden bereits ausgeführt und können deshalb nicht mehr verändert werden. Bei der Migration auf ein neues Prozessmodell wird also nur der Bereich vor der Wavefront verändert (vgl. Abbildung 5.1).

Abbildung 5.2 zeigt den konzeptionellen Ansatz der Deploy New Version-Funktionalität. Nach dem deployen einer neuen Version eines Prozessmodells ist es möglich, Instanzen der alten Version zu migrieren. Dabei können bereits beendete Instanzen wieder aktiviert werden und führen dann die neue Prozesslogik aus. Außerdem können auch neue Instanzen der aktuellen Version des Prozessmodells erzeugt werden.

Für die Verwendung dieses Frameworks bei der Patterntransformation zur Laufzeit eines Workflows müsste das Framework um eine Methode zur automatischen regelbasierten Abbildung von Datenmanagementpatterns erweitert werden. Eine Möglichkeit dies umzusetzen wäre es die Prozessinstanz beim Erreichen eines Datenmanagementpatterns während der Ausführung automatisch in den Zustand suspended zu versetzen. Anschließend müsste eine neue Systemkomponente für die Anpassung des Prozessmodells, das heißt in diesem Fall die Transformation des Datenmanagementpatterns, aufgerufen werden. Zuletzt müsste die pausierte Prozessinstanz auf das neue Prozessmodell gezogen und die Ausführung fortgesetzt werden.

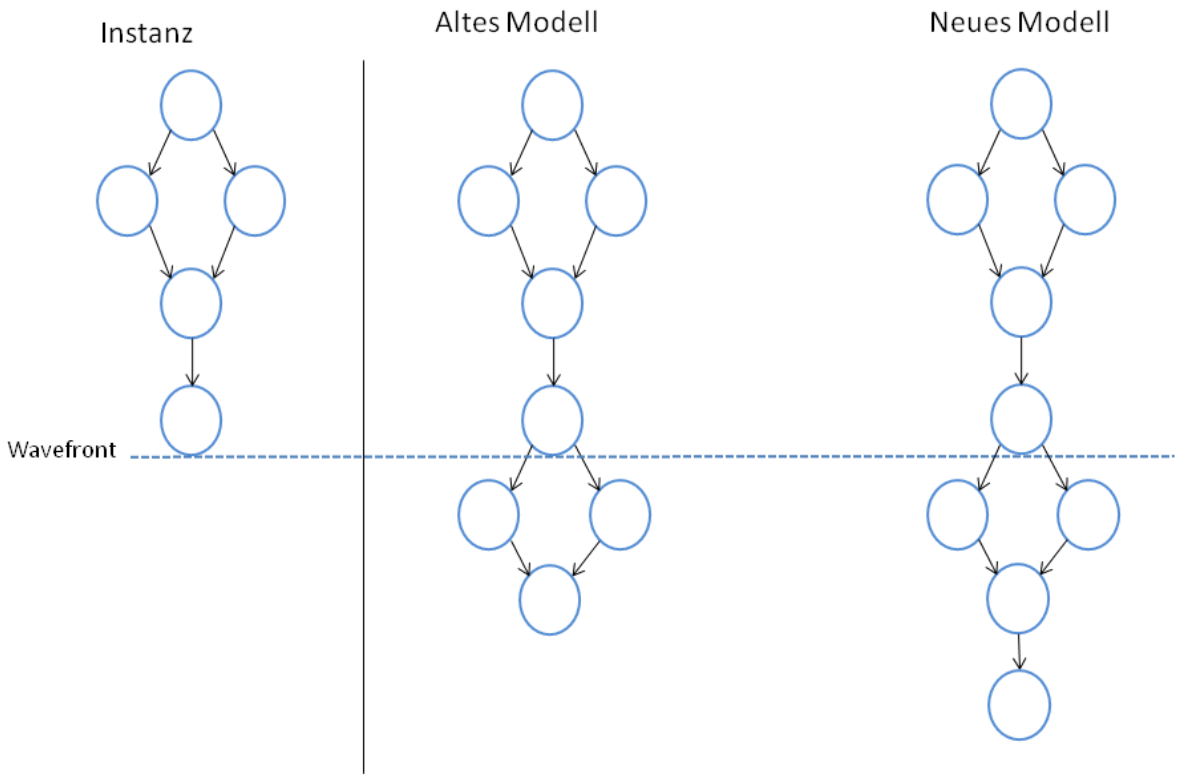

Abbildung 5.1: Wavefront einer Instanz Quelle: [Sch11]

Bewertung

Der Vergleich der Eigenschaften dieses Frameworks mit den zuvor definierten Anforderungen zeigt einige Probleme:

- Der Prozess der Erstellung einer neuen Version des Prozessmodells und der anschließenden Migration der laufenden Prozessinstanzen ist sehr aufwendig und würde daher einen hohen Overhead während der Laufzeit erzeugen.
- Der Prototyp dieses Frameworks sieht vor, dass die Anpassung des Prozessmodells manuell zur Laufzeit erfolgt. Das Framework müsste also entsprechend umfangreich erweitert werden, um die regelbasierte Auswahl von Prozessfragmenten für die Injektion in ein Prozessmodell zu unterstützen.

5.2.2 Diplomarbeit Nr. 3144 Ausführung von Workflow-Fragmenten in BPEL

Das zweite Framework wurde im Rahmen der Diplomarbeit [Hum11] entwickelt. Der Ansatz dieser Arbeit ermöglicht es, einen unvollständigen Prozess auszuführen, und die benötigten Prozessfragmente während der Laufzeit aus einem Repository zu laden und in den Prozess einzufügen. Um dies zu ermöglichen, wird die BPEL Workflowsprache um eine Reihe neuer Elemente erweitert.

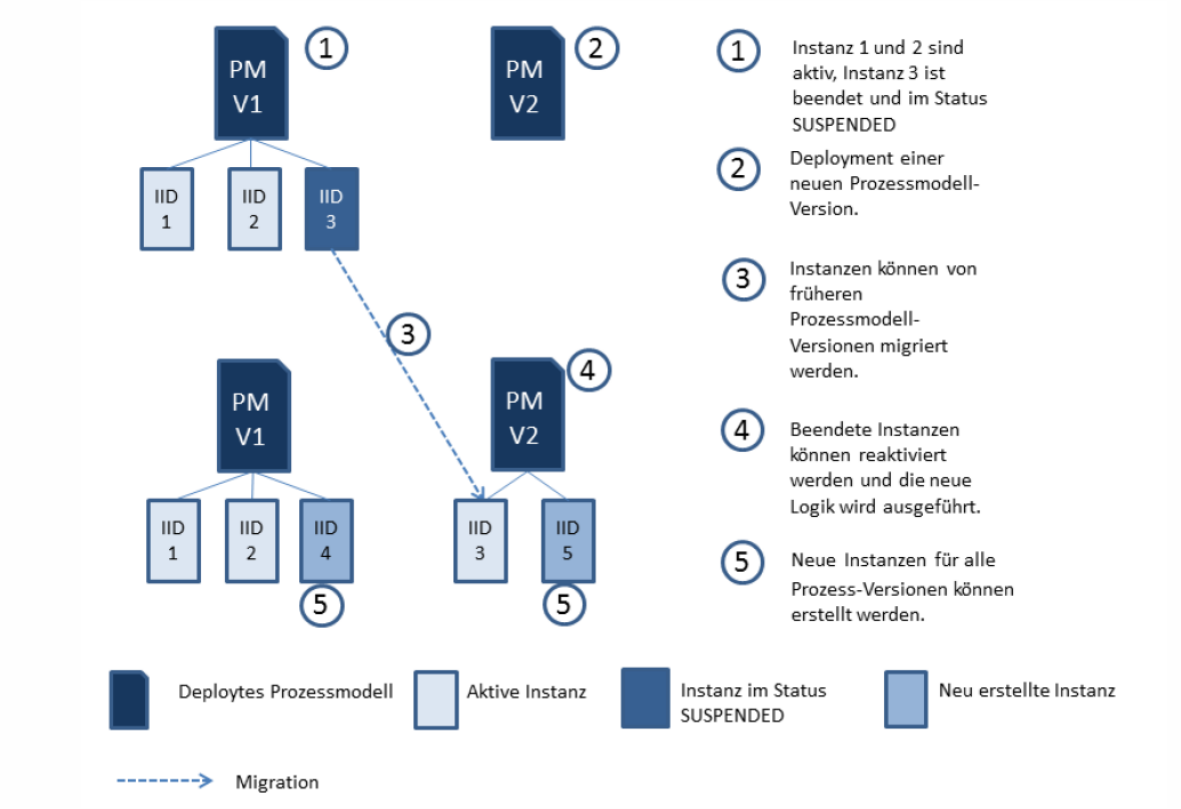

Abbildung 5.2: Hauptfunktionalitäten der Deploy New Version-Funktionalität Quelle: [Sch11]

Abbildung 5.3 zeigt die neu eingeführten Elemente, beziehungsweise Aktivitäten. Um die zur Modellierungszeit noch unbekannten Bereiche in einem Prozess zu kennzeichnen, wurden die Aktivitäten fragmentEntry und fragmentExit eingeführt. FragmentEntry wird benutzt, um die vor einer Aktivität unbekannte Prozesslogik zu markieren. Das Gegenstück zu diesem Element fragmentExit markiert die nach einer Aktivität unbekannte Prozesslogik. Diese beiden Elemente markieren somit bei der Definition von Prozessfragmenten, an welcher Stelle das bereits modellierte Fragment mit einem anderen Prozessfragment verbunden werden kann. Zur Laufzeit können Prozessfragmente damit "geklebt"werden. Weitere eingeführte Aktivitäten sind unter anderem fragmentScope, fragmentRegion, fragmentFlow und fragmentSequence. Diese Aktivitäten stellen Erweiterungen der Standard BPEL Varianten dar, welche das einfügen von Prozessfragmenten unterstützen. In einer fragmentFlow Aktivität ist es beispielsweise erlaubt ein Prozessfragment als eine parallel ablaufende Aktivität einzukleben; in einer fragmentSequence Aktivität darf ein Prozessfragment nur am Ende der Sequenz eingeklebt werden. Die Komposition von Prozessfragmenten zur Laufzeit wurde in dieser Arbeit in Form einer manuellen Komposition über eine GUI Schnittstelle implementiert.

Auch dieses Framework müsste für die Umsetzung der Patterntransformation zur Laufzeit noch um eine Komponente zur regelbasierten Auswahl von Przessfragmmenten erweitert werden. Datenmanagementpatterns innerhalb eines Workflows könnten über entsprechende fragmentEntry oder FragmentRegion Aktivitäten markiert werden. Bei der Ausführung müssten dann durch eine neue

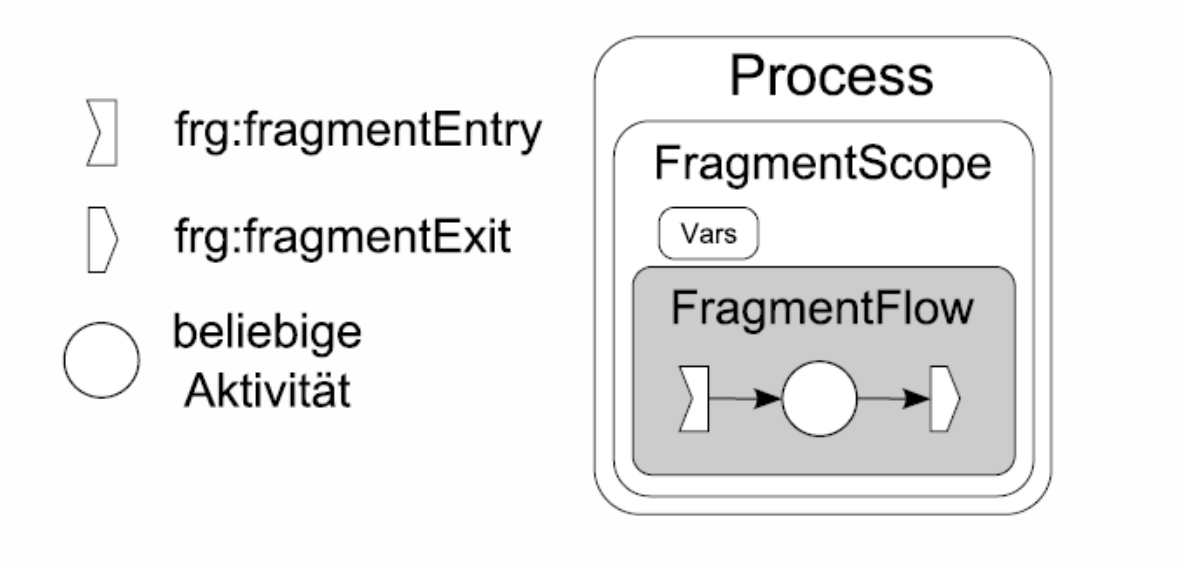

Abbildung 5.3: Neu eingeführte Elemente Quelle: [Hum11]

Systemkomponente die Abbildung der Datenmanagementpatterns umgesetzt werden. Außerdem müsste das Verbinden von zusammengehörigen fragmentEntry und fragmentExit Aktivitäten bei dieser Abbildung automatisiert werden.

Bewertung

Auch dieses Framework entspricht nicht den Anforderungen dieser Arbeit:

- Für die Modellierung von Workflowfragmente, muss vorher bereits feststehen an welchen Stellen diese angeklebt werden können (fragmentEntry, FragmentExit). Dies könnte bei der rekursiven Patterntransformation zu Problemen führen.
- Der Overhead der Patterntransformation zur Laufzeit ist bei diesem Ansatz wahrscheinlich nicht all zu hoch.
- Auch bei diesem Framework erfordert die Auswahl von Prozessfragmenten Input des Modellierers. Im besonderen ist das Mapping von Variablen und von Webservice aufrufen zwischen Workflowfragmenten derzeit nur manuell über ein graphisches Userinterface möglich.

5.2.3 Diplomarbeit Nr. 3564 Dynamic Process Fragment Injection in a Service Orchestration Engine

Das letzte Framework stammt aus der Diplomarbeit von Lukasz Bialy [Bia14]. In dieser Arbeit wurde für die Definition von unvollständigen Prozessfragmenten die Abstract Activity eingeführt. Die Abstract Activity markiert eine Stelle innerhalb eines Prozesses, an der zur Laufzeit ein Prozessfragment

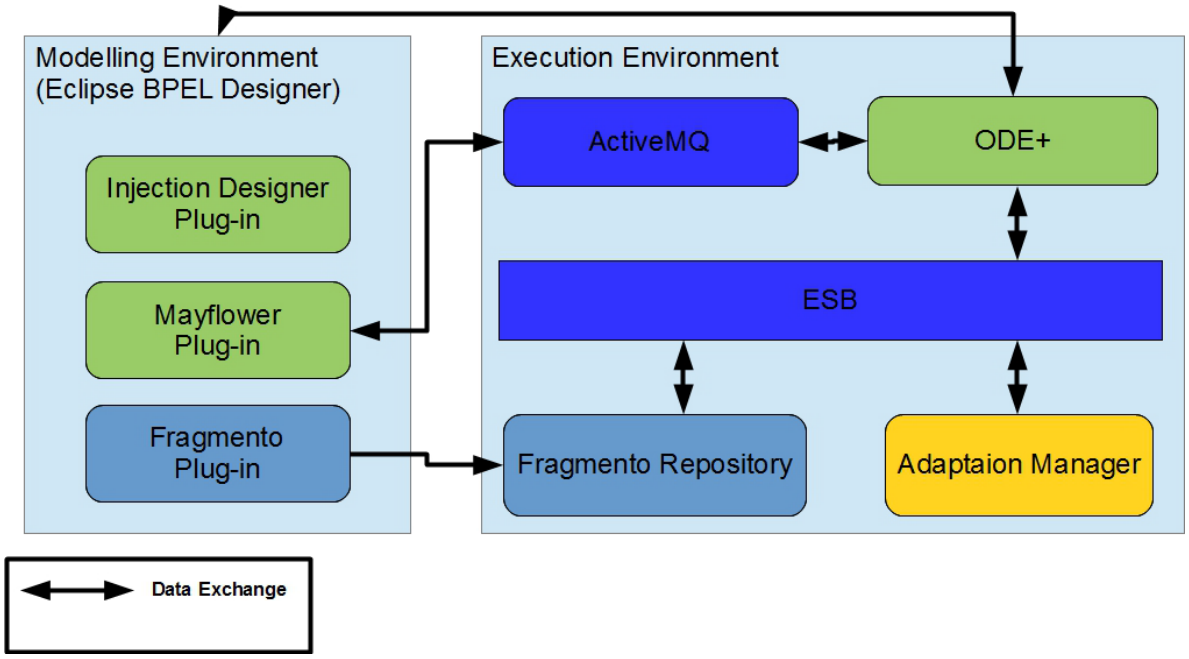

Abbildung 5.4: Architektur des Injektion Frameworks Quelle: [Bia14]

eingefügt werden soll. Innerhalb einer Abstract Activity können Informationen für die Auswahl eines Prozessfragments definiert werden.

Abbildung 5.4 zeigt die Architektur des Injektion Frameworks aus [Bia14]. Dabei wurde die Modellierungsumgebung von BPEL um ein Injection Designer Plugin erweitert. Dieses Plugin erlaubt die Verwendung von Abstract Activities bei der Definition von BPEL Workflows. Bei der Ausführungsumgebung sind drei Komponenten für die Injektion von Prozessfragmenten entscheidend. Ein Fragmento Repository wird für die Verwaltung und zum Abfragen von Prozessfragmenten genutzt. Der Adaption Manager entscheidet zur Laufzeit, welches Prozessfragment eingefügt werden soll und eine erweiterte Apache -ODE Engine erlaubt die Verwendung von Abstract Activities, sowie dieInjektion von Workflowfragmenten zur Laufzeit.

Zudem wurde dieses Framework innerhalb des Instituts für die Architektur von Anwendungssystemen der Universität Stuttgart noch um ein Pluginkonzept erweitert. Dieses Konzept erlaubt es eigene Fragment Retrieval Services zu erstellen und innerhalb des Frameworks zu registrieren. Fragment Retrieval Services erlauben es eigene Methoden für die Auswahl von Workflowfragmenten für die Injektion zur Laufzeit zu definieren.

Bei diesem Framework müsste für die Umsetzung der Patterntransformation zur Laufzeit ein geeigneter Fragment Retrieval Service implementiert werden. Dieser Service könnte zum Beispiel die Systemkomponente für die Patterntransformation aufrufen. Desweiteren müssten die bestehenden Datenmanagementpatterns entsprechend in eine Abstract Activity eingebettet werden.

Bewertung

Im Gegensatz zu den anderen betrachteten Frameworks erfüllt dieses Framework die zuvor gestellten Anforderungen am besten:

- Der Prozess der Injektion läuft performant ab und erzeugt keinen zu hohen Overhead.
- Im Gegensatz zu den anderen Frameworks bietet dieses Framework bereits ein Plugin System, welches die Implementierung einer Patterntransformation erleichtert.
- Der Injektionsprozess ähnelt sehr der Ersetzung zur Laufzeit. Bereits bestehende BPEL Elemente können in eine Abstract Activity eingefügt werden.

Aus diesen Gründen wird für die Umsetzung der Patterntransformation zur Laufzeit im Rahmen dieser Arbeit dieses Framework verwendet.

5.3 Auswahl einer Regelengine für die Implementierung und Auswertung der Patterntransformationsregeln

Bei dem bisherigen Modell der Patterntransformation wurden die Transformationsregeln in Form von Java Dateien innerhalb eines Eclipse Plugins implementiert. Diese Implementierung hatte den Nachteil, dass jede Änderung oder Erweiterung des Regelsatzes zwangsläufig ein erneutes kompilieren des Plugins erzwang. Um die Arbeit an den Transformationsregeln zu vereinfachen, ist es daher sinnvoll diese Transformationsregeln von der eigentlichen Prozesslogik zu trennen.

Um dies zu ermöglichen wird eine Regelengine benötigt, welche es erlaubt, die Transformationsregeln zur Laufzeit aus einer Datenbank zu laden und auszuführen. Im Verlauf dieser Arbeit wurden insbesondere zwei mögliche Regelengines auf ihre Eignung für diesen Zweck hin überprüft: Zum einen die eher simple JRuleEngine und zum anderen das komplexere Regel Management System Drools.

5.3.1 JRuleEngine

Die JRuleEngine ist eine einfache, in Java implementierte open source Regelengine, die auf dem Java Specification Request 94 (JRE94) basiert [Car06]. Regeln werden in Form von einfachen XML Dateien bereitgestellt.

Listing 5.1 zeigt den Aufbau einer einfachen Beispielregel. Das grundlegende Element ist das RuleExecutionSet, welches eine, oder mehrere Regeln enthalten kann. Eine Regel besteht aus einer Reihe von If Bedingungen der Form leftTerm [operator rightTerm] sowie einem oder mehreren Then Blöcken. Ein Then Teil einer Regel enthält einen Methodenaufruf an eine Java Methode sowie die entsprechenden Argumente, welche dieser Methode übergeben werden.

```
<?xml version="1.0" encoding="UTF-8"?>
 <rule-execution-set>
  <name>RuleExecutionSet1</name>
  <description>Rule Execution Set</description>

  <synonymn name="customer" class="example1.Customer" />

  <!--
  If the credit limit of the customer is greater than the amount of the
  invoice and the status of the invoice is "unpaid" then
  decrement the credit limit with the amount of the invoice and
  set the status of the invoice to "paid".
  -->
  <rule name="Rule1" description="credit limit control rule" >
   <if leftTerm="customer.getCreditLimit" op=">" rightTerm="example1.Invoice.getAmount" />
   <if leftTerm="example1.Invoice.getStatus" op="=" rightTerm="unpaid" />
   <if leftTerm="example1.getSelectedItems" op="containsatleastone"
        rightTerm="[Item1,Item2,Item3]" />
   <then method="customer.decrementCreditLimit" arg1="example1.Invoice.getAmount" />
   <then method="example1.Invoice.setStatus" arg1="paid" />
  </rule>
 </rule-execution-set>
```

Listing 5.1: Beispielregel JRuleEngine Quelle: [Car06]

Während der Regelausführung wird überprüft, ob die an die Reglengine übergebenen Objekte die If Bedingungen der Regeln erfüllen. Sollte dies der Fall sein, wird der entsprechende Then Teil der Regel ausgeführt.

Die JRuleEngine unterstützt für die Regeldefiniton folgende Operatoren: =, <>, contains, notcontains, containsatleastone, notcontainsanyone, <, >, <=, >=

Die Operatoren contains, notcontains, containsatleastone und containsanyone sind dabei nur auf String Variablen und Aufzählungen definiert. Besteht eine Bedingung nur aus einem leftTerm, wird überprüft, ob der entsprechende Term existiert. Ein Term darf entweder ein Literal (String, Zahl, etc.), das Ergebnis der Get Methode eines Objekts, oder eine Aufzählung von Werten sein.

Bewertung

Die JRuleEngine ist eine sehr einfache Form einer Regelengine. Das Format der XML basierten Regeldefinition ähnelt dem Format unserer Transformationsregeln und ist aufgrund der sehr simplen Form einfach zu implementieren. Die JRuleEngine efüllt damit das Kriterium, dass die Erstellung und Bearbeitung von Transformationsregeln möglichst einfach, bzw. zumindest nicht schwerer als zuvor sein sollte. Außerdem erzeugt die Regelauswertung einen geringen Overhead.

In der Praxis hat sich jedoch der stark beschränkte Umfang an Operatoren als Problem erwiesen. Im Rahmen dieser Diplomarbeit wurde zunächst versucht die bestehenden Transformationsregeln auf Basis der JRuleEngine umzusetzen. Viele der bereits implementierten Tranformationsregeln ließen sich mit dem beschränkten Satz an Operationen nur sehr umständlich umsetzen. Zum Beispiel enthät der

Operatorensatz keinen notequal Operator. Dieser logische Operator wird aber in den Bedingungen der bestehenden Transformationsregeln häufig verwendet. Zudem ist bei der Beschreibung der Bedingungen nur der Zugriff auf Getter Methoden eines Objektes ohne Parameter möglich. Dies hat sich bei der Umsetzung als großes Problem erwiesen. Am problematischsten hat sich die Umsetzung der Regelsequenzen erwiesen. DIe JRuleEngine bietet keine Möglichkeit die Anwendungsreihenfolge der Regeln näher zu definieren und Änderungen an der Engine wären zu umfangreich für den Rahmen dieser Arbeit gewesen.

Aus diesen Gründen wurde im Rahmen dieser Arbeit gegen den Einsatz der JRuleEngine entschieden.

5.3.2 Drools

Drools ist ein Business Regel Management System (BRMS) bestehend aus einer Regelengine, einer Anwendung zur Verwaltung von Regeln und einem Eclipse Plugin [dro] Drools ist ein Open Source Projekt und entspricht, genauso wie die JruleEngine, dem JRE94 Standard.

Drools verwendet für die Definition von Regeln ein eigenes Dateiformat mit Endung .drl.

```
package package-name

imports

globals

functions

queries

rules
```

Listing 5.2: Aufbau einer .drl Datei Quelle: [dro]

Listing 5.2 zeigt den allgemeinen Aufbau einer solchen .drl Datei. Zu Beginn einer .drl Datei kann die Datei einem package zugeordnet werden. Dies dient dazu der Regel einen Namespace zuzuordnen. Im Anschluss können in beliebiger Reihenfolge Importe, globale Variablen, Funktionen, Queries und Regeln deklariert werden.

```
rule "name"
   attributes
   when
      LHS
   then
      RHS
end
```

Listing 5.3: Aufbau einer Regel in Drools Quelle: [dro]

Listing 5.3 zeigt den groben Aufbau einer einzelnen Regel. Jede Regel besteht aus einer Left Hand Side (LHS) und einer Right Hand Side(RHS). Der LHS, oder auch when Abschnitt einer Regel bestimmt die Bedingungen, welche gelten müssen, bevor die jeweilige Regel ausgeführt werden soll. Der RHS, oder auch then Teil einer Regel enthält die Prozesslogik, die bei Anwendbarkeit der Regel ausgeführt werden soll. Außerdem können eine Reihe von Attributen für eine Regel festgelegt werden. Zu diesen Attributen zählen unter anderem das salience Attribut, welches festlegt in welcher Reihenfolge Regeln ausgeführt werden, und das Dialect Attribut, welches angibt welche Sprache Ausdrücke innerhalb der LHS und RHS haben(zum Zeitpunkt dieser Arbeit werden nur Java und MVEL unterstützt). Für eine ausführlichere Liste von Attributen sei auf die Dokumentation [dro] verwiesen.

```
rule "name"
   salience 1
   dialect "Java"
   when
      $p : Person(name == "Peter")
   then
      System.out.println($p.name)
end
```

Listing 5.4: Einfache Beispielregel in Drools

5.4 zeigt eine sehr einfache Beispielregel in Drools. Sollte eines der an die Regelengine übergebenen Objekte von der Klasse Person sein und der Name dieser Person Peter sein, so wird der then Block der Regel aufgerufen und der Name der Person wird ausgegeben. $p stellt dabei die Deklaration einer lokalen Variable dar, über die auf das jeweilige Objekt zugegriffen werden kann. Dies zeigt den grundlegenden Aufbau einer Regel. In der LHS der Regel werden über Patterns (in diesem Fall alle Objekte vom Typ Person) und Constraints auf diese Patterns (name == "Peter") die Bedingungen definiert, die für die Ausführung der Regel gelten sollen. Dabei kann der Constraint Teil eines solchen Patterns beliebige logische Ausdrücke enthalten.

Der RHS Teil einer Regel enthält die eigentlichen Aktionen, die von dieser Regel ausgeführt werden sollen. Der RHS Teil kann beliebigen Java Code, oder Methoden Aufrufe enthalten. Außerdem können über insert und modify neue Faktenobjekte für die Regelauswertung in die laufende Rulesession eingebracht, bzw bestehende Objekte verändert werden.

Bewertung

Im Vergleich zu der zuvor beschriebenen JRuleEngine wird sofort klar, dass Drools deutlich umfangreichere Ausdrucksmöglichkeiten für die Definition von Regeln zur Verfügung stellt. Der Regelaufbau einer Drools Regel ähnelt außerdem sehr stark dem Aufbau der bereits bestehenden Transformationsregeln (siehe 2.3.2). Die LHS einer Drool Regel entspricht dabei dem Condition Part der Transformationsregeln, die RHS entspricht dem Action Part. Der Fragment Part einer Regel könnte, wie in dem bestehenden Ansatz der patterntransformation, durch eine Anfrage an das Fragmento Repository der Workflowfragmente innerhalb des RHS Teils einer Regel umgesetzt werden. Die Umsetzung von Regelsequenzen ist implizit durch den salience Wert der einzelnen Regeln möglich.

Die Definition neuer Tranformationsregeln ist daher nicht komplizierter als bei der bisherigen Implementierung und entspricht somit den zuvor gestellten Anforderungen. Tatsächlich ist das Einfügen und Bearbeiten von Regeln sogar einfacher, da die Systemkomponente nicht mehr jedes mal neu kompiliert werden muss. Zudem macht dies die Portierung der bestehenden Transformationsregeln in das neue Format sehr einfach.

Die einzelnen .drl Dateien können einfach als Bytearrays in eine Datenbank gespeichert werden und dann zu Beginn der Patterntransformation abgerufen werden. Dies ermöglicht die angestrebte Trennung der Transformationsregeln von der eigentlichen Patterntransformationskomponente und erfüllt damit die zuvor definierten Anforderungen.

Zusammenfassend ist Drools daher die sinnvollere Wahl für die vorliegende Arbeit.

5.4 Gründe für den Einsatz eines RDF Triplestores

Bei der Patterntranformation von Datenmanagementpatterns der höheren Hierarchiestufen (siehe Abschnitt 2.3.2) werden unter anderem auch simulationsspezifische Ontologien genutzt [Boh14]. Bei der bisherigen Implementierung dieser Patterntransformation wurde dabei direkt innerhalb von Java Methoden auf diese OWL Dateien zugegriffen und deren Informationen dann ebenso mit Java weiterverarbeitet. Dieses Vorgehen führt zu einer Reihe von Problemen. Zum einen müssen die OWL Dateien lokal verwaltet werden, sodass die Patterntranformationskomponente Zugriff auf diese Dateien hat. Zum anderen ist diese Form des Zugriffs fehleranfällig und es fehlen die Ausdrucksmöglichkeiten, die eine Anfragesprache für RDF Datensätze, wie zum Beispiel SPARQL, bietet. Eine Solche Anfragesprache würde auch den Zugriff auf die Ontologien für den Entwickler von Transformationsregeln stark vereinfachen.

Der Einsatz eines RDF Triplestores würde die Verwaltung dieser Ontologien erheblich vereinfachen und zugleich einen einheitlichen und vereinfachten Zugriff auf die entsprechenden Ontologien mit einer entsprechenden Anfragesprache ermöglichen.

RDF-3X ist ein solcher Triplestore. Dieser Triplestore wurde im Rahmen der Arbeit [NW08] am Max-Planck-Institut für Informatik in Saarbrücken entwickelt. RDF-3X verwendet eine RISC Architektur, um die Anfragen in der Sprache SPARQL möglichst performant umzusetzen und ist damit deutlich schneller als vergleichbare Triplestores [NW08].

6 Aufbau der Patterntransformation

In diesem Kapitel wird der Entwurf der neuen Patterntransformation beschrieben. Zunächst wird der grundsätzliche Aufbau der Komponente sowie die Verbindungen zu den anderen Elementen des Frameworks beschrieben. Im Anschluss wird der Ablauf der Patterntransformation näher beschrieben.

6.1 Grundlegender Aufbau des Systemkomponente

In Abschnitt 5.1 wurden bereits die Anforderungen an die im Rahmen dieser Arbeit umzusetzende Patterntransformation näher konkretisiert. Um die Transformation von Datenmanagementpatterns sowohl während der Modellierung, als auch zur Laufzeit zu gewährleisten, ist die Umsetzung der Patterntransformation als eine separate Systemkomponente sinnvoll.

Abbildung 6.1 zeigt den grundlegenden Aufbau der neuen Pattterntransformationskomponente. Dabei sind neu implementierte und veränderte Komponenten blau unterlegt. Im Mittelpunkt steht ein Webservice, der die Patterntransformation realisiert.

Für die Transformation während der Modellierung eines Workflows wird der Webservice durch ein Plugin innerhalb der Eclipse BPEL Entwicklungsumgebung aufgerufen. Für die Transformation zur Laufzeit erfolgt der Aufruf der Patterntransformation durch ein Plugin innerhalb des modifizierten Apache ODE Framework, welches in Abschnitt 5.2.3 näher beschrieben wurde. Die Transformationsregeln für die Patterntransformation werden in einer eigenen Postgres Datenbank gespeichert und während der Transformation durch den Webservice abgerufen. Die Workflowfragmente für die Patterntransformation befinden sich weiterhin in dem bereits bestehenden Fragmento Repository. Wenn bei der Patterntransformation Informationen über Ressourcen, wie zum Beispiel Datenquellen oder Simulationsmodelle, benötigt werden, wird der bereits bestehende Resource Management Webservice aufgerufen. Dieser Webservice verwaltet seine Daten in einer PostgreSQL Datenbank. Für die Verwaltung der simulationsbezogenen Ontologien wird nun RDF-3X als Triplestore verwendet. Bei der Transformation von Datenmanagementpatterns auf der Hierarchiestufe der simulationsorientierten Datenmanagementpatterns werden die benötigten Informationen aus diesen Ontologien von den entsprechenden Regeln über SPARQL Queries abgefragt.

Abbildung zeigt den Aufbau des Patterntransformers noch einmal genauer. Für die Auswertung der Abbildungsregeln wird die Drools Regelengine verwendet. Um die rekursive Abbildung von Workflowfragmenten zu ermöglichen verfügt der Patterntransformer über einen Graphtraverser. Diese Komponente traversiert ein Workflowfragment und sucht dabei nach eingebetteten Datenmanagementpatterns. Wenn ein Pattern gefunden wurde, wird die Patterntransformation rekursiv aufgerufen.

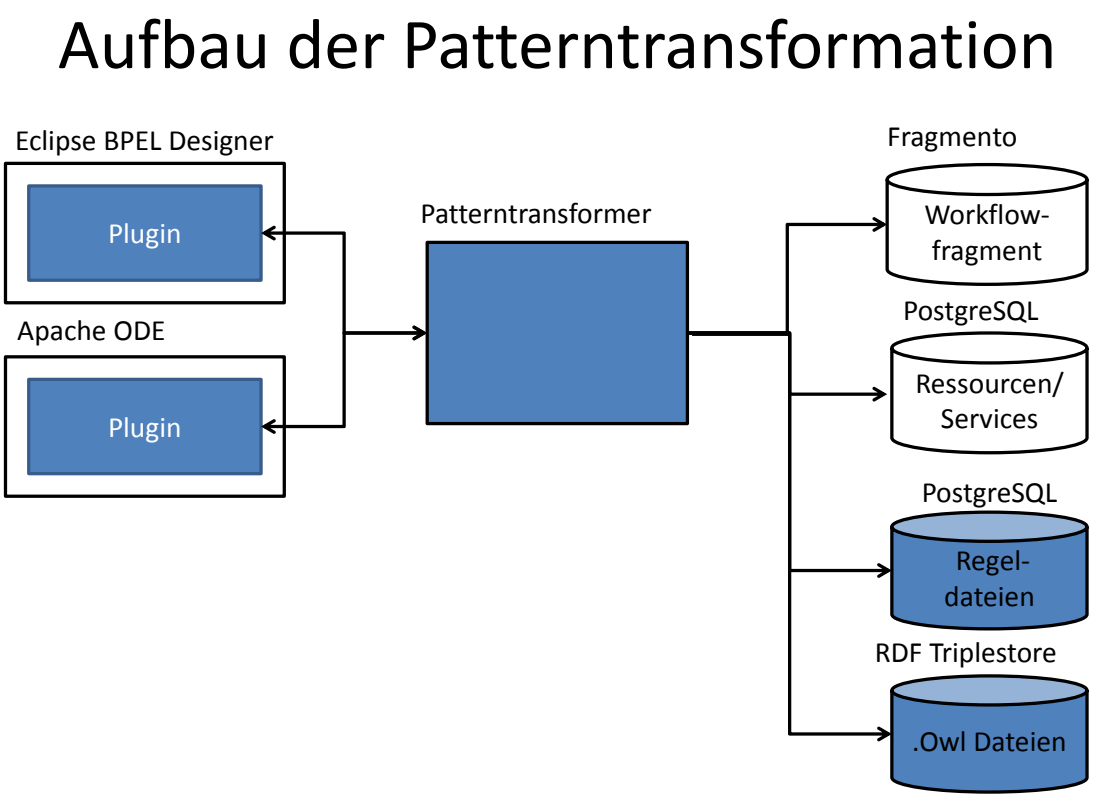

Abbildung 6.1: Entwurf der neuen Patterntransformation

6.2 Ablauf der Patterntransformation

In diesem Abschnitt wird der Ablauf der Patterntransformation näher beschrieben. Der Webservice für die Patterntransformation erwartet folgende Eingabeparameter:

Parameter Eine Map der Eingabeparameter des zu transformierenden Patterns

Recursion Ein Boolean Wert. Dieser Wert gibt an, ob ein Pattern rekursiv transformiert werden soll, oder nicht.

Abbildung 6.3 zeigt eine abstrakte Darstellung des Ablaufs einer Patterntransformation. Zunächst wird je nach Art des zu transformierenden Patterns die passende Regelsequenz aus der Regeldatenbank geladen. Mit dieser Regelsequenz wird dann eine neue Regelsession der Drools Regelengine erzeugt. Die Parameterwerte des zu transformierenden Patterns werden als Objekte an diese Regelsession übergeben. Anschließend wird die Regelsession ausgeführt. Bei der Ausführung der Regelsession werden die einzelnen Transformationsregeln des jeweiligen Datenmanagementpatterns anhand der an die Regelengine übergebenen Parameter und der Bedingungen in der LHS der Regel auf ihre Anwendbarkeit überprüft. Wenn eine passende Regel gefunden wurde, wird der entsprechende RHS Block

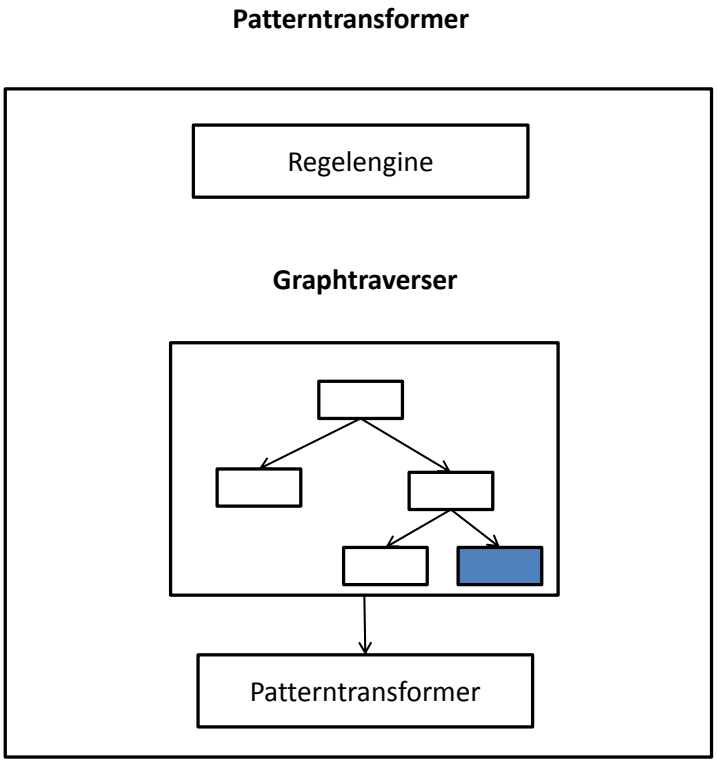

Abbildung 6.2: Aufbau des Patterntransformers

(siehe Abschnitt 5.3.2) dieser Regel ausgeführt. Innerhalb dieses Blocks wird zunächst das passende Workflowfragment aus dem Fragmento Repository geladen. Dieses Workflowfragment enthält in der Regel eine Reihe von Platzhaltern für die Variablenwerte. Diese Platzhalter werden im nächsten Schritt durch die aktuellen Variablenwerte ersetzt. Zuletzt wird das fertig initialisierte Workflowfragment aus der Regelsession wieder in den Patterntransformer geladen. Sollte beim Aufruf des Webservices die rekursive Transformation ausgewählt worden sein, wird nun das transformierte Workflowfragment traversiert und dabei nach eingebetteten Patterns gesucht. Wenn ein weiteres Datenmanagementpattern innerhalb des Workflowfragments gefunden wird, dann wird die Patterntransformation rekursiv für dieses Pattern aufgerufen. Zum Schluss wird das vollständig transformierte Workflowfragment als Ausgabe des Webservices zurückgegeben. Wenn keine geeignete Transformationsregel gefunden wurde wirft der Webservice einen entsprechenden Fehler, der dann jeweils innerhalb des Eclipse Plugins, oder des Plugins in der erweiterten Apache ODE Umgebung abgefangen und behandelt wird.

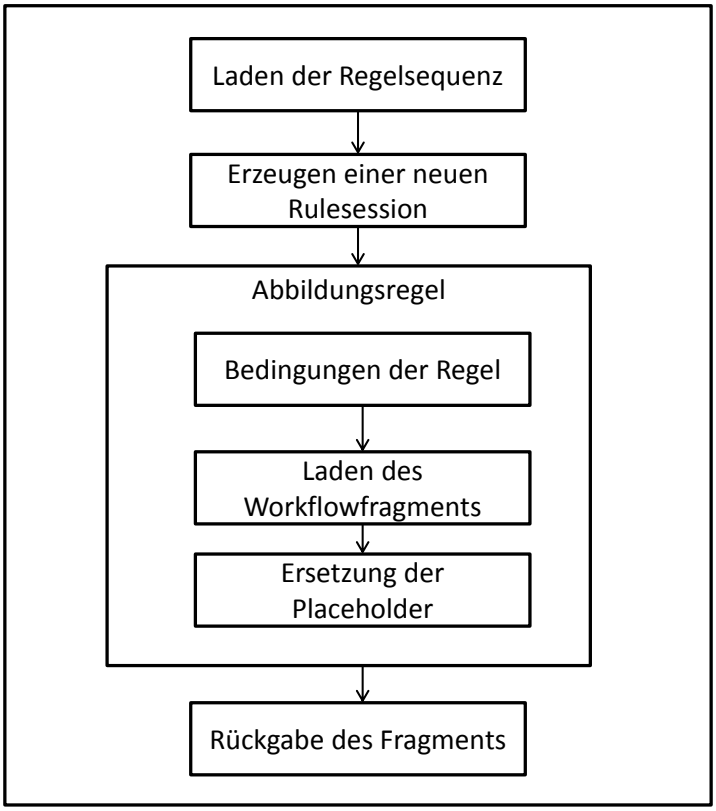

Abbildung 6.3: generischer Ablauf der Patterntransformation

7 Implementierung

Im vorherigen Kapitel wurde das grundlegende Konzept der neuen Patterntransformation genauer beschrieben. Dieses Kapitel beschäftigt sich nun mit einigen Details der Implementierung des neuen Webservices zu Transformation von Datenmanagementpatterns und der dazugehörigen Plugins in der Eclipse Entwicklungsumgebung und dem erweiterten Apache ODE Framework. Außerdem wird in diesem Kapitel die Implementierung der Transformationsregeln näher beschrieben.

7.1 BPEL Plugin

Um die Transformation von Datenmanagementpatterns innerhalb der Eclipse Entwicklungsumgebung zu ermöglichen, musste das bereits bestehende Eclipse Plugin für die Patterntransformation angepasst werden. Dabei werden die bereits bestehenden und in Abschnitt 3.2 näher beschriebenen Tranformationsoptionen weiterhin unterstützt. Über die Option *Transform all patterns* können alle patterns innerhalb des Workflows transformiert werden. Die Option *Transform this pattern* hingegen transformiert ein bestimmtes Datenmanagementpattern innerhalb des Workflows. Für beide Optionen ist nach wie vor sowohl die rekursive, als auch die nicht rekursive Ausführung der Transformation möglich. Abbildung 7.1 zeigt die entsprechenden Schaltflächen innerhalb der Entwicklungsumgebung.

Um diese beiden Transformationsoptionen an die neue Patterntransformation anzupassen, wurde der in Abschnitt 3.2 beschriebene WorkflowGraphTraverser wiederverwendet. Diese Komponente traversiert den Workflow und sucht dabei nach Datenmanagementpatterns. Anders als bei der ursprünglichen Implementierung wird nun für die Transformation der gefundenen Datenmanagementpatterns der neue Webservice für die Patterntransformation verwendet. Dazu werden zunächst die Parameter des Datenmanagementpatterns in eine Hashmap eingefügt. Anschließend erfolgt der Aufruf des Webservices über eine Stub Klasse. Der Webservice liefert als Rückgabe dann das transformierte Workflowfragment. Zum Schluss wird durch den WorkflowGraphTraversers das Datenmanagementpattern innerhalb des Workflows durch dieses transformierte Workflowfragment ersetzt. Die bestehenden Java Klassen für die Patterntransformation wurden aus dem Eclipse Plugin entfernt.

7.2 Apache ODE Plugin

Wie bereits in Abschnitt 5.2.3 erwähnt bietet das Framework für die Injektion von Workflowfragmenten zur Laufzeit aus der Arbeit [Bia14] die Möglichkeit eigene Plugins für die Auswahl von Workflowfragmenten zu implementieren. Dabei wird bei der Ausführung einer *Abstract Activity* durch die erweiterte Apache ODE Engine zunächst die Komponente *FragmentStore* aufgerufen. Der

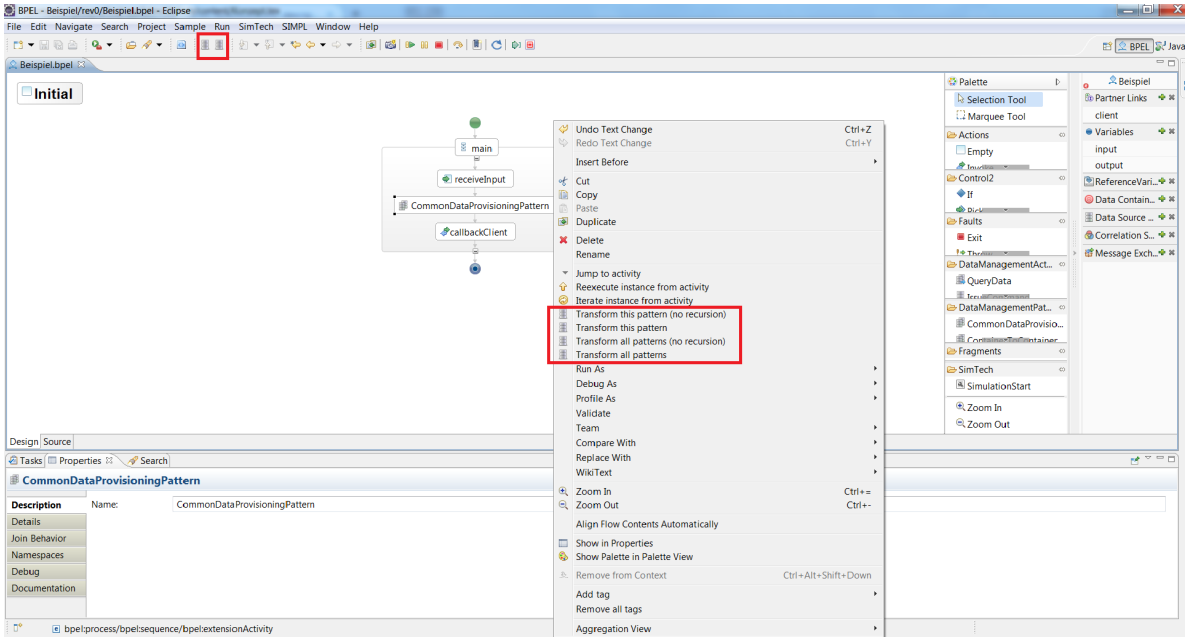

Abbildung 7.1: Optionen der Patterntransformation innerhalb der Eclipse BPEL Entwicklungsumgebung

FragmentStore überprüft den Parameter pluginType der Abstract Activity und erzeugt dann durch die Fabrikklasse *FragmentRetrievalFactory* einen entsprechenden *FragmentRetrievalService* der diesen plugintype unterstützt. Ein solcher *FragmentRetrievalService* muss das folgende Interface IFragmentRetrievalService implementieren, welches aus nur einer einzigen Methode besteht:

```
String retrieveProcessFragment(String retrievalPath, HashMap<?,?> parameters)
```

Diese Methode retrieveProcessFragment wird bei der Injektion zur Laufzeit aufgerufen, um ein geeignetes Workflowfragment für das Datenmanagementpattern zu finden. Dazu werden die Parameter des Patterns in einer Hashmap parameters an die Methode übergeben. Die Variable retrievalPath gibt das Verzeichnis an, in dem das Workflowfragment von der Ausführungsengine erwartet wird. Der *FragmentStore* erzeugt anhand der ID der zu ersetzenden *AbstractActivity* einen eindeutigen Verzeichnispfad, an dem das Workflowfragment hinterlegt werden soll und übergibt dieses als Variable retrievalpath. Außerdem fügt er alle Kindelemente der *AbstractActivity* sowie deren Attribute in eine Hashmap ein und übergibt diese als Variable parameters. Anhand dieser Parameter muss der *FragmentRetrievalService* nun ein geeignetes Workflowfragment auswählen und am Dateipfad retrievalpath hinterlegen. Zuletzt muss der *FragmentRetrievalService* noch den Dateinamen des Workflowfragments an den *FragmentStore* zurückgeben, damit diese weiß, wie diese Datei benannt ist.

Um die Abbildung von Datenmanagementpatterns zur Laufzeit zu gewährleisten, muss also ein neuer *FragmentRetrievalService* und ein zugehöriger Plugintyp implementiert werden. Der neu implementierte Service hat den Namen *DataManagementPatternRetrievalService*. Dieser Service filtert aus der übergebenen Hashmap parameters die Parameter des zu transformierenden Datenmanagementpatterns heraus. Im Anschluss wird der webservice für die Patterntransformation mit diesen Parametern

aufgerufen. In der derzeitigen Implementierung efolgt diese Transformation zur Laufzeit rekursiv, um den Overhead zur Laufzeit zu verringen, Der *DataManagementPatternRetrievalService* kann aber auch eine nicht rekursive Transformation durchführen, indem der entsprechende Parameter des Webservice Aufrufs angepasst wird. Das transformierte Workflowfragment aus der Antwort des Webservice zur Transformation von Datenmanagementpatterns wird anschließend im Dateipfad retrievalPath abgelegt und der Dateiname wird zurückgegeben.

Zusätzlich wurde innerhalb der statischen Methode *getService* der Klasse *FragmentRetrievalServiceFactory* eine neue Pluginbezeichnung *DataManagementPlugin* hinzugefügt und eine entsprechende Abfrage für den *DataManagementPatternRetrievalService* hinzugefügt.

7.3 Patterntransformation Webservice

Der Webservice für die eigentliche Patterntransformation wurde als Axis2 Webservice implementiert. Die beiden wichtigsten Klassen des Webservices sind die Klasse Patterntransformer, in der die eigentliche Patterntransformation implementiert ist, und die klasse Graphtraverser, die für die rekursive Transformation von Workflowfragmenten benötigt wird.

Der Patterntransformer implementiert die eigentliche Transformation der Workflowpatterns. Dazu wird bei einem Aufruf des Webservices innerhalb der Klasse *PatternTransformationWebservice* die Methode *transformPattern* des Patterntransformers aufgerufen. Der Patterntransformer lädt dann zunächst anhand der übergebenen Hashmap parameters die benötigte Regeldatei aus der Postgres Regeldatenbank. Um die Performanz der Patterntransformation bei der wiederholten Abbildung gleicher Datenmanagementpatterns zu erhöhen, werden die abgefragten Regeldateien in einem Cache gespeichert. Anschließend wird mit der Regeldatei eine neue *KnowledgeBase* erzeugt und aus dieser *KnowledgeBase* eine neue *KnowledgeSession* erzeugt. Die eizelnen Einträge der Hashmap parameters werden als Faktenobjekte für der Regelauswertung an diese *KnowledgeSession* übergeben. Im Anschluss wird die Regelauswertung gestartet (siehe Ablauf aus Abschnitt 6.2). Wenn keine anwendbare Transformationsregel gefunden wurde, wird ein Fehler geworfen. Wenn eine Transformationsregel anwendbar ist, wird innerhalb des Then Teils dieser Regel die Methode *retrieveFragment* der Klasse *FragmentServiceCalls* aufgerufen, um das Workflowfragment auf das diese Regel abbildet aus dem Fragmento Repository zu laden. Auch die abgerufenen Workflowfragmente werden zur Optimierung der Laufzeit der Patterntransformation in einem Cache gespeichert. Im Anschluss werden die Platzhalter innerhalb des Workflowfragments mit den korrekten Variablenwerten ersetzt. bei den simulationsorientierten Datenmanagementpatterns werden hierbau auch Informationen aus der zugehörigen Ontologie des Simulationsmodells benötigt. Um Anfragen an den Triplestore zu erleichtern wurde eine Methode *queryTriplestore* implementiert. Diese Methode benötigt als Parameter eine SPARQL Query, den Namen der Datenbank, an die diese Query gestellt werden soll und die Adresse des Systems, auf dem der Triplestore eingerichtet wurde. Als Rückgabe liefert diese Methode dann das Ergebnis der Query. Wenn keine rekursive Transformation des Workflowfragments gewählt wurde, wird das resultierende transformierte Workflofragment dann als Rückgabewert des Webservices zurückgegeben. Wenn eine rekursive Transformation gewählt wurde wird das Workflowfragment in ein JDOM Dokument geparst und anschließend die Methode *traverse* des Graphtraversers mit diesem Workflowfragment als Parameter aufgerufen.

Der Graphtraverser traversiert den transformierten Workflowgraph und sucht dabei nach weiteren eingebetteten Datenmanagementpatterns. Wird ein solches Pattern gefunden, ruft der Graphtraverser den Patterntransformer rekursiv auf, um dieses Workflowfragment abzubilden. Zum Schluss wird das vollständig transformierte Workflowfragment zurückgegeben.

7.4 Umsetzung der Transformationsregeln und Regelsequenzen

Im Rahmen dieser Arbeit mussten die bereits vorhandenen Transformationsregeln an das neue Format der Drools Regelengine angepasst werden. Da die derzeitige prototypische implementierung der Injektion von Workflowfragmenten zur Laufzeit noch nicht die Injektion von Invoke Aktivitäten unterstützt konnten nicht alle Transformationsregeln und Workflowfragmente angepasst werden. Zukünftige Versionen des Rahmenwerks werden die Verwendung von Invoke Aktivitäten bei der Injektion allerdings unterstützen, daher können die restlichen Abbildungsregeln zu einem späteren Zeitpunkt umgesetzt werden. Zu den Patterns die hiervon betroffen sind zählt unter anderem das Simulation Model Realization Pattern. Der ursprüngliche Prototyp der Patterntransformation setzte die Regelsequenzen, die die Reihenfolge in der Regeln auf ihre Anwendbarkeit überprüft werden, als eigene Java Klassen um. In der Drools Regelengine wird allerdings ein eigener Algorithmus für die Auswertung der Anwendbarkeit von Regeln verwendet (siehe Abschnitt 5.3.2). Außerdem wird in einer Regelengine jede anwendbare Regel auch ausgeführt, während das Konzept der Patterntransformation es vorsieht, dass nur eine Transformationsregel ausgeführt wird.

Das Format der Drools Regeln erlaubt es, jedoch die Reihenfolge, in der Regeln angewendet werden, durch den *salience* Wert einer Regel festzulegen (In der JRuleEngine existiert keine vergleichbare Möglichkeit dies zu tun). Dabei werden Regeln mit einem höheren Wert vor Regeln mit einem niedrigeren Wert angewandt. Um zu verhindern, dass mehr als eine Transformationsregel angewandt wird, wird der Regelengine zusätzlich ein Boolean Objekt *ruleExecuted* übergeben. Im LHS Teil der Abbildungsregeln wird als eine Bedingung für die Ausführung der Abbildungsregel überprüft, ob dieser Wert false ist. Wenn eine Regel ausgeführt wird, wird im RHS Teil dieser Regel der Wert der Variablen auf true gesetzt und somit die Anwendung weiterer Abbildungsregeln verhindert. Bei dieser Form der Implementierung existieren also keine eigenen Dateien für die Definition von Regelsequenzen. Stattdessen ist die Regelsequenz einer Regeldatei implizit durch die *salience* Werte der einzelnen Regeln festgelegt.

Im Rahmen dieser Arbeit wurden die Transformationsregeln und dazugehörigen Workflowfragmente der folgenden Datenmanagementpatterns angepasst:

simulation oriented Data Provisioning Pattern

Data Transfer and Transformation Pattern

Sequential Data Iteration Pattern

Container to Conatainer Pattern

Get Timestep Pattern

7.5 Anpassung der Datenmanagementpatterns und Workflowfragmente

Um die Transformation der Datenmanagementpatterns zur Laufzeit zu ermöglichen musste die Darstellung dieser Patterns innerhalb des Workflows verändert werden. Das Framework für die Injektion von Workflowfragmenten zur Laufzeit erwartet eine *AbstractActivity* mit den entsprechenden Parametern für die Auswahl eines geeigneten Workflowfragments.

```
<bpel:extensionActivity>
      <abcs:abstractActivity pluginType="DataManagementPattern">
            <simpl:simulationOrientedDataProvisioningPattern
            ...
            </simpl:simulationOrientedDataProvisioningPattern
      </abcs:abstractActivity>
</bpel:extensionActivity>
```

Listing 7.1: Neuer Aufbau eines Datenmanagementpatterns

Aus diesem Grund wurden die entsprechenden Klassen der Datenmanagementpatterns innerhalb des Pakets org.eclipse.bpel.simpl.model des Eclipse plugins entsprechend angepasst. Listing 7.1 zeigt den neuen Aufbau einer entsprechenden Aktivität innerhalb des Workflows. Innerhalb einer extensionActivity sind die Datenmanagementpatterns jetzt zusätzlich durch eine Abstract Activity gekapselt. Diese AbstractActivity hat als Parameter den neu implementierten pluginType für die Datenmanagementpatterns, durch den bei der Injektion zur Laufzeit der geeignete Service für die Auswahl eines Workflowfragments aufgerufen werden kann. Da sich die Form der Datenmanagementpatterns verändert hat wurden auch die bereits implementierten Workflowfragmente, die weitere eingebettete Datenmanagementpatterns enthalten, dementsprechend angepasst.

8 Bewertung

Dieses Kapitel beschäftigt sich mit der Evaluation der zuvor implementierten Patterntransformation zur Laufzeit eines Workflows. Um den Overhead dieser Patterntransformation besser beurteilen zu können, wird dazu ein bereits implementierter Workflow für die biomechanische Knochensimulation als Testworkflow verwendet [Boh14]. Dabei wird zunächst der Overhead der Patterntransformation nur an der eigentlichen Datenbereitstellung in diesem Workflow untersucht. Im Anschluss wird der Overhead der Patterntransformation im Vergleich zu der Laufzeit des gesamten Workflows aufgezeigt.

Im Anschluss daran erfolgt eine weitere Bewertung der Patterntransformation zur Laufzeit anhand der zuvor gestellten Anforderungen aus der Aufgabenstellung dieser Arbeit 1.1. Diese Bewertung umfasst zum einen, ob Transformationsregeln tatsächlich einfacher, oder zumindest nicht komplizierter, implementiert werden können. Zum anderen wird aufgezeigt inwiefern die Patterntransformation zur Laufzeit die universelle Einsetzbarkeit von Datenmanagementpatterns verbessert hat.

8.1 Laufzeit Vergleich anhand eines Testworkflows zur Knochensimulation

Wie bereits zuvor erwähnt wurde für die Auswertung des Overheads der Patterntransformation zur Laufzeit eines Workflows der biomechanische Workflow aus der Studienarbeit [Boh14] genutzt. In Abschnitt 3.3.1 wird dieser Workflow näher beschrieben. Die Ausführung fand auf einem Openstack[1] Server mit zwei Sechs-Kern-Prozessoren Intel Xeon E5-2630 @2,3GHz und insgesamt 16 x 16 GiB RAM (256GB) Hauptspeicher statt. Für die Ausführung des Workflows wurde eine Windows Instanz mit dem Betriebssystem Windows Server 2012 R2 Standard 64 Bit auf diesem Server verwendet. Für die Pandas Simulation wurde eine Linux Instanz mit dem Betriebssystem Ubuntu 14.04.2 auf diesem Server erstellt. Der RDF-3X Triplestore wurde ebenfalls auf dieser Instanz eingerichtet

8.1.1 Anteil der Patterntransformation an der Laufzeit der Datenbereitstellung

Als erstes wurde der Anteil der Patterntransformation zur Laufzeit eines Workflows an der Datenbereitstellung näher erfasst. Zu diesem Zweck wurde ein zusätzlicher Testworkflow mit einem einzelnen systemorientierten Data Provisioning Pattern genutzt.Abbildung 8.1 zeigt diesen Workflow. Die Parameter des simulationsorientierten Data Provisioning Patterns entsprechen dabei den mathematischen

[1] https://www.openstack.org/

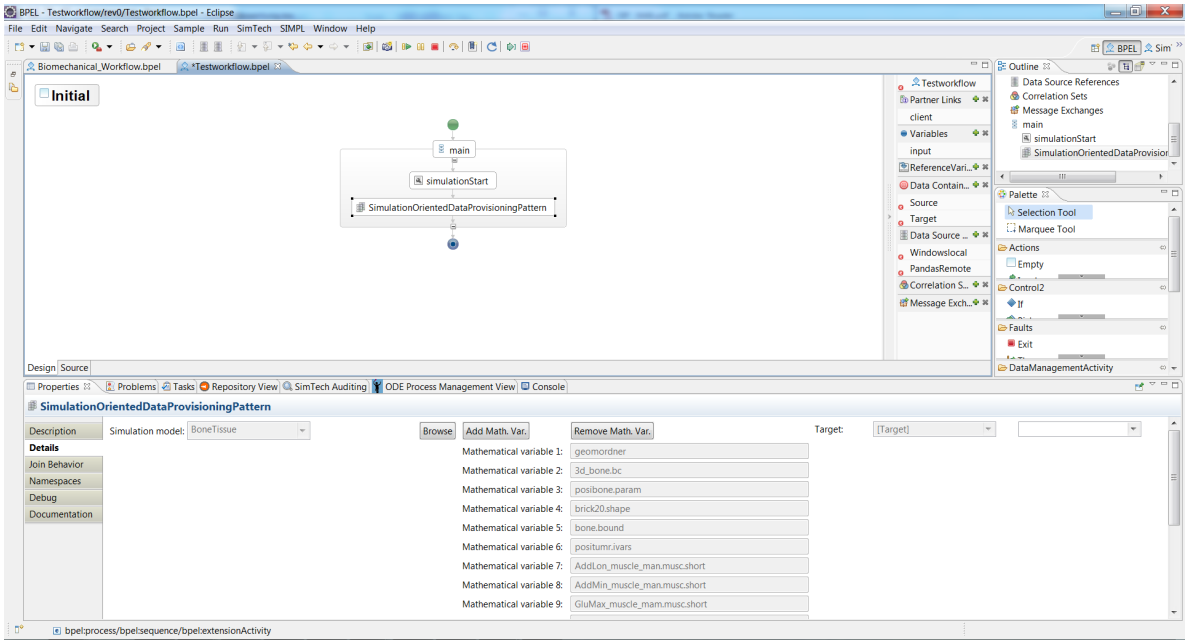

Abbildung 8.1: Testworkflow für die Messung des Anteils der Patterntransformation an der Datenbereitstellung

Variablen aus der Ontologie des biomechanischen Simulationsmodells (Bonetissue.owl) und gleichen den Variablenwerten des biomechanischen Workflows. Als Zielinstanz für die Datenbereitstellung wird ein Ordner auf der Linuxinstanz verwendet. Da bei der Patterntransformation sowohl die bei der Abfrage der Transformationsregeln aus der Postgres Datenbank, als auch bei der Abfrage der Workflowfragmente aus dem Fragmento Repository jeweils ein Cache genutzt wird, wird im folgenden jeweils die Patterntransformation mit initialisiertem Cache untersucht, das heißt der Workflow wurde zweimal hintereinander ausgeführt und der Wert der zweiten Messung der Laufzeit übernommen. Bei der Transformation des simulationsorientierten Data Provisioning Patterns kommt es zu vier rekursiven Transformationsschritten. Dabei werden vier Regeldateien mit insgesamt fünf Abbildungsregeln ausgewertet.

Abbildung 8.2 zeigt die Dauer der Patterntransformation im Vergleich zu der Dauer der Bereitstellung der Variablenwerte auf die Zielinstanz. In Abbildung 8.3 wird dieser Anteil der Patterntransformation noch einmal genauer in die Zeit, die benötigt wird um die Regelsession zu initialisieren und in die Dauer der eigentlichen Regelauswertung aufgeteilt. aufgeteilt. Wie bereits erwähnt werden Transformationsregeln und Workflowfragmente in einem Cache gespeichert. Daher ist der Anteil dieser Abfragen an der Patterntransformation bei mehrmaligem Aufruf vernachlässigbar gering und wird deshalb in der Abbildung nicht extra aufgeführt. Die Messungen zeigen deutlich, dass die Initialisierung der Regelsession mit am meisten Zeit in Anspruch nimmt. Das liegt zum einen daran, dass für die einzelnen Datenmanagementpatterns derzeit nur wenige Transformationsregeln implementiert sind und daher die Regelanwendung kaum Zeit in Anspruch nimmt. Zum anderen liegt es daran, dass bei jeder neu aufgerufenen Transformationsregel zunächst eine neue Knowledgebase erzeugt werden muss, auf der dann eine neue Rulesession gestartet wird. Hier bietet sich weiteres Optimierungspoten-

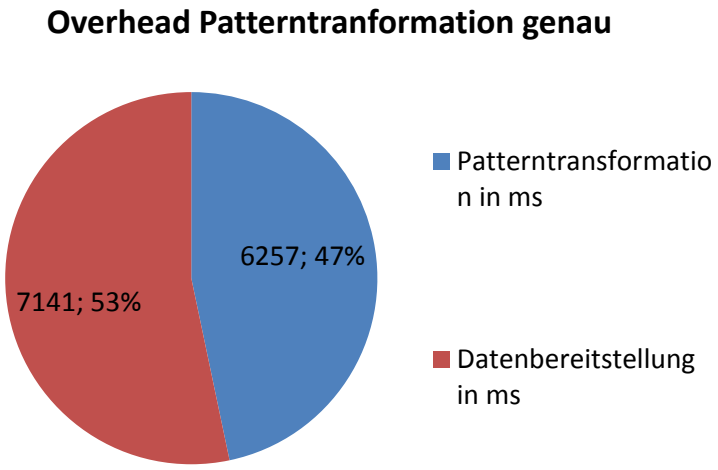

Abbildung 8.2: Anteil der Patterntransformation an der Datenbereitstellung

tial für weitere Arbeiten. Zum Beispiel könnten für jede Regeldatei eines Datenmanagementpatterns die Regeldateien der Patterns auf das dieses Patterns innerhalb der Patternhierarchie abgebildet wird in der Datenbank vermerkt werden. Dadurch könnte die KnowledgeBase der Rulesession bereits mit allen benötigten Regeln initialisiert werden und müsste nicht bei jeder Transformation neu erstellt werden.

8.1.2 Anteil der Patterntransformation an der Laufzeit des gesamten Workflows

Im Anschluss wurde nun der Overhead der Patterntransformation an dem eigentlichen biomechanischen Workflow gemessen. Die Abbildung 8.4 zeigt die Ergebnisse dieser Messung. Bei diesem Workflow erfolgt die Datenbereitstellung parallel für zwei Softwareinstanzen von Pandas, daher ist die Laufzeit der Patterntransformation auch entsprechend höher als bei der vorigen Messung. Diese Messung hat ergeben, dass die Patterntransformation mit einem Anteil von nur einem Prozent der Gesamtlaufzeit des Simulationsworkflows einen angemessen geringen Overhead produziert. Damit ist die Anforderung aus der Aufgabenstellung dieser Arbeit, dass die regelbasierte Abbildung von Da-

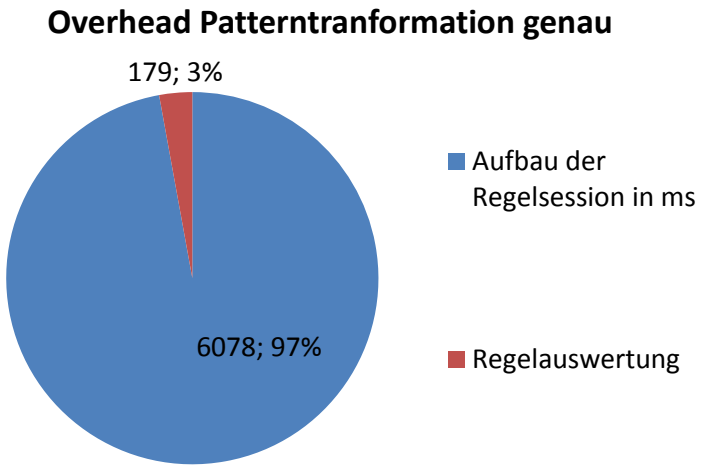

Abbildung 8.3: Genauere Aufteilung des Overheads der Patterntransformation

tenmanagementpatterns zur Laufzeit möglichst effizient sein sollte, für solche Simulationsworkflows erfüllt.

8.2 Bewertung des Aufwands zur Implementierung zusätzlicher Transformationsregeln und Workflowfragmente

Eine Anforderung aus der Aufgabenstellung dieser Diplomarbeit in Abschnitt 1.1 ist, das es möglichst einfach sein soll neue Transformationsregeln und Workflowfragmente der Patterntransformation hinzuzufügen. In der bisherigen Implementierung der Patterntransformation musste dafür zunächst eine neue Tranformationsregel als Java Klasse und ein neues Workflowfragment auf das diese Transformationsregel abbildet, implementiert werden. Anschließend musste die Kontrollstrategie des zugehörigen Datenmanagementpatterns innerhalb des Eclipse BPEL Plugins um diese Transformationsregel erweitert werden. Da die einzelnen Transformationsregeln und Kontrollstrategien als Java Klassen des Eclipse BPEL Plugins implementiert waren, musste zum Schluss dieses Plugin bei jeder Änderung einer Transformationsregel neu kompiliert werden.

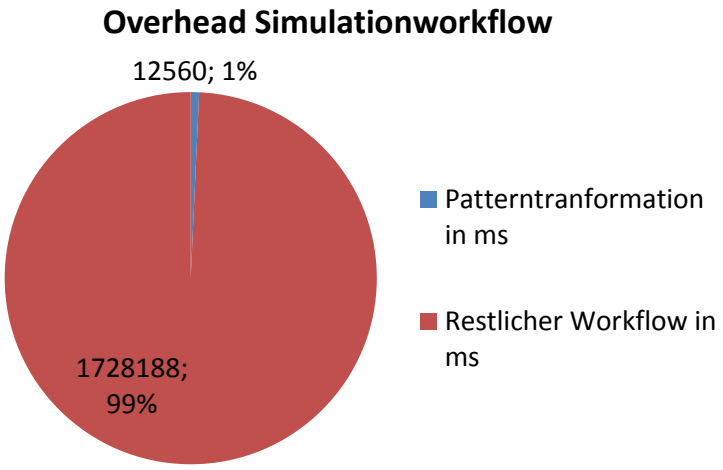

Abbildung 8.4: Overhead der Patterntransformation an gesamten Workflow

Im Gegensatz dazu ist bei der neuen Umsetzung der Patterntransformation keine Neukompilierung des Webservices nötig. Um eine neue Transformationsregel zu implementieren muss lediglich die .drl Datei mit den Regeln des jeweiligen Datenmanagementpatterns innerhalb der Postgres Datenbank der Transformationsregeln erweitert werden. Da es in der neuen Implementierung keine eigenen Dateien für die Kontrollstrategie mehr gibt, muss nach der Implementierung der Regel nur noch der salience Wert der Regel entsprechend der gewünschten Anwendungsreihenfolge der Regeln gesetzt werden. Im Vergleich zu der vorherigen Implementierung der Patterntransformation sind Änderungen und Ergänzungen an den Transformationsregeln als deutlich einfacher geworden, da nur einer Datei mit Transformationsregeln angepasst werden muss und außerdem das neukompilieren des Plugins wegfällt. Ein weiterer Vorteil der neuen Implementierung ist, dass Drools ein eigenes Eclipse Plugin hat, dass den Entwurf und die Bearbeitung von Transformationsregeln durch eine entsprechende Entwicklungsumgebung zusätzlich vereinfacht.

9 Weiterführende Arbeiten

In dem vorhergehenden Kapitel wurde der im Rahmen dieser Arbeit entwickelte Prototyp für die Patterntransformation zur Laufzeit bewertet. In diesem Kapitel sollen nun einige weitere Ideen und Konzepte für die weitere Verbesserung der Patterntransformation kurz angesprochen werden.

Eine Möglichkeit wäre die Einführung eines Sichtenkonzeptes. Dieses Konzept würde es dem Nutzer erlauben frei zwischen Sichten auf den Workflow mit Patterns, oder die transformierte Variante des Workflows zu wechseln. In diesem Kapitel wird auf ein mögliches Konzept hierfür näher eingegangen.

Zuletzt wird in diesem Kapitel noch auf das PGM Modell zur Optimierung von Datenmanagement Schritten in einem Workflow eingegangen und wie beziehungsweise zu welchem Zweck es mit der in dieser Arbeit entwickelten Transformation von Patterns kombiniert werden kann.

9.1 Umsetzung eines Sichtenkonzepts auf die unterschiedlichen Ebenen der Patternhierarchie

Eine Möglichkeit, die Nutzung von Datenmanagementpatterns für Wissenschaftler weiter zu vereinfachen, wäre die Umsetzung eines Sichtenkonzeptes auf die unterschiedlichen Ebenen der Patternhierarchie (siehe Abbildung 2.9). Bei der derzeitigen Implementierung der Patterntransformation kann ein Wissenschaftler nur den Prozessablauf des voll transformierten ausführbaren Workflows innerhalb der BPEL Designer Entwicklungsumgebung nachvollziehen. Dabei ist es für den Wissenschaftler in der Regel schwer nachzuvollziehen, welche abstrakten Datenmananagementpatterns auf welche ausführbaren Workflowfragmente abgebildet wurden. Ein weiteres Problem ist das Monitoring der Workflowausführung. Während ein Wissenschaflter innerhalb des BPEL Designers nur die abstrakten Datenmanagementpatterns sehen sollte, erzeugt die Ausführungsengine Informationen über die Ausführung des konkreten transformsformierten Patterns. Um diese Informationen zu den abstrakten Datenmanagementpatterns innerhalb der Entwicklungsumgebung in einen Zusammenhang zu bringen, wird ein Ansatz für die Aggregation dieser Informationen benötigt.

Bei der Umsetzung eines Sichtenkonzeptes könnte der Wissenschaftler frei zwischen unterschiedlichen Sichten auf die verschiedenen Transformationsebenen des Workflows wählen und damit die während der Ausführung des Workflows vollzogene Patterntransformation leichter nachvollziehen. Außerdem bilden die unterschiedlichen Sichten einen Ansatz für die Implementierung einer stufenweisen Aggregation der Informationen über die Workflowausführung. Im folgenden wird ein möglicher Ansatz für die Umsetzung eines solchen Sichtenkonzepts vorgestellt.

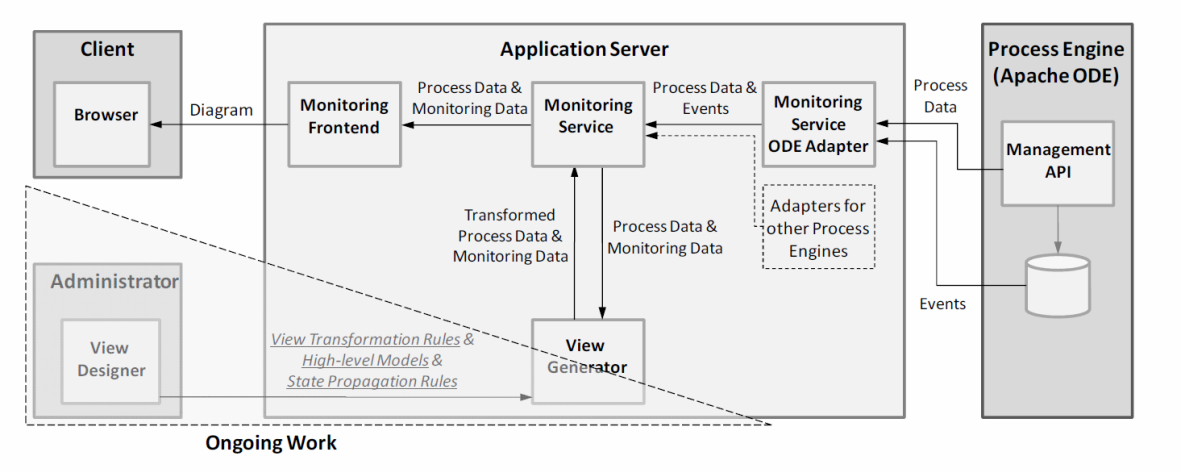

Abbildung 9.1: generische Architektur für das Monitoring von Geschäftsprozessen in mehreren Abstraktionsschichten Quelle: [SLL11a]

9.1.1 View Based Monitoring

Einen möglichen Ansatz für die Umsetzung eines Sichtenkonzepts liefert das Paper „A Prototype for View-based Monitoring of BPEL Processes “[SLL11a]. In diesem Paper wird der Prototyp eines Tools für das Monitoring von Prozessmodellen vorgestellt, welches außerdem die Transformation von Sichten auf einen Prozess unterstützt.

Abbildung 9.1 zeigt die zugrundeliegende Architektur dieses Monitoring Tools. Die wichtigsten Komponenten dieser Architektur sind ein web-basierter Client, Ein Tool für den Entwurf von Sichten genannt View Designer, der Application Server und die Ausführungsengine. Der View Designer gehört zu den Elementen der Architektur, die noch nicht vollständig umgesetzt wurden. Diese Komponente dient Administratoren dazu, Sichten auf den Ausführungsgraphen des Prozessmodels zu definieren. Die Ausführungsengine liefert zur Laufzeit Informationen über den Prozessablauf und auftretende Events an den Monitoring Service innerhalb des Application Servers. Der Monitoring Service gibt diese Informationen an den View Generator weiter. Der View Generator aggregiert aus den zuvor innerhalb des View Designers festgelegten Transformationsregeln und Modellen die übergebenen Prozessdaten und Monitoring informationen in eine der View angepasste Form. Diese transformierten Daten werden dann an den Monitoring Service zurückgegeben. Wenn ein Client in seinem Browser eine View abruft wird eine entsprechende Anfrage an das Monitoring Frontend gestellt. Dieses erhält dann von dem Monitoring Service die entsprechende Monitoring Daten und Ausführungsevents der gewünschten View und liefert diese als Diagramm an den Client zurück.

Diese Architektur unterstützt zwei unterschiedliche Formen des Monitorings von Prozessmodellen: Zum einen das normale Monitoring der ausführbaren Prozessmodelle in der Workflowsprache BPEL sowie die Abstraktion dieser Modelle mittels abstrakter Sichten. Zum anderen unterstützt diese Architektur auch das Monitoring über mehrere Prozessmodelle und Workflowsprachen über ein sogenanntes state projection Konzept. Da dieses Konzept in dem derzeitigen Prototypen noch

nicht umgesetzt wurde sei für weitere Informationen über dieses Konzept auf die Quelle [SLL+11b] verwiesen.

Dieses Monitoring Tool hat genau die Eigenschaften, die für die Umsetzung eines Sichtenkonzeptes auf die Datenmanagementpatterns nötig sind. Das Tool erlaubt es, entsprechende Sichten durch die Angabe von Transformationsregeln zu definieren. Außerdem verwendet der Prototyp ebenfalls die Apache ODE Engine für die Ausführung von Prozessen. Es wäre daher sinnvoll, in weiterführenden Arbeiten zu der Umsetzung eines Sichtenkonzepts dieses Tool näher zu untersuchen.

9.2 Optimierung der Datenverarbeitung (PGM)

ALs letztes wird in diesem Kapitel noch ein Ansatz für die Optimierung der Datenverarbeitung innerhalb eines Workflows vorgestellt. Außerdem werden mögliche Ansätze aufgezeigt, wie diese Optimierung zur Verbesserung der Patterntransformation genutzt werden kann.

In dem Paper Än Approach to Optimize Data Processing in Business Processes"[VSS+07] wurde ein Framework für die Optimierung der Datenverarbeitung in Business Prozessen entworfen. Bei diesem Ansatz wird der zu optimierende Workflowgraph von der jeweiligen Workflowsprache in ein Process Graph Model (PGM) überführt und anschließend unter Verwendung von Transformationsregeln optimiert. Ein PGM Graph wird formal durch ein Tupel der Form (A,Ec,Ed, V, P) definiert. Dabei steht A für die Menge der Aktivitäten innerhalb des Prozesses, Ec für die gerichteten Kanten des Kontrollflusses innerhalb des Prozesses, Ed entsprechend für die gerichteten Kanten des Datenflusses innerhalb des Prozesses, V für die Menge der Variablen und P für die Partner (zum Beispiel Webservices), mit denen der Prozess kommuniziert. Optimierungsregeln auf dieses Graphenmodell werden dabei Anhand von Bedingungen an die Strukur des Graphen und Änderungen an dieser Struktur definiert. Im Vergleich zu der kontrollflussorientierten Workflowsprache BPEL wird in diesem Graphenmodell sowohl der Kontrollfluss als auch der Datenfluss explizit dargestellt. Dies erleichtert die Definition der Bedingungen von Optimierungsregeln im Vergleich zu der Verwendung von BPEL mit integrierten SQL Abfragen.

Abbildung 9.2 zeigt den Ablauf dieser Optimierung. Zuerst wird der BPEL Workflow mit eingebetteten SQL Abfragen (BPEL/SQL) in einen PGM Graphen umgewandelt. Auf diesen PGM Graphen werden anschließend gemäß einer Kontrollstrategie anwendbare Transformationsregeln durch die Optimierungsengine ausgeführt, d.h. der Workflowgraph wird über die Transformationsregeln in einen optimierten Workflowgraphen restrukturiert. Diese Regeln bestehen aus einem Condition und einem Action Part, ähnlich wie bei den Abbildungsregeln bei der Transformation von Datenmanagementpatterns. Der so transformierte PGM Graph wird zuletzt wieder in einen BPEL Workflow überführt.

Abbildung 9.3 zeugt den genauen Ablauf der Optimierung. Um die ursprüngliche Semantik des Workflows durch die Anwendung von Optimierungsregeln nicht zu verändern, existiert in diesem Modell das Konzept der Optimierungssphären. Eine solche Optimierungssphäre bezeichnet einen abgegrenzten Bereich innerhalb des Workflows in dem Tranformationsregeln angewendet werden dürfen. Ein Beispiel für eine solche Optimierungssphäre ist die Scope Optimization Sphere (SOS). Wie bereits im Abschnitt 2.2.3 erwähnt existiert in der Workflowsprache BPEl das Konzept der Scopes.

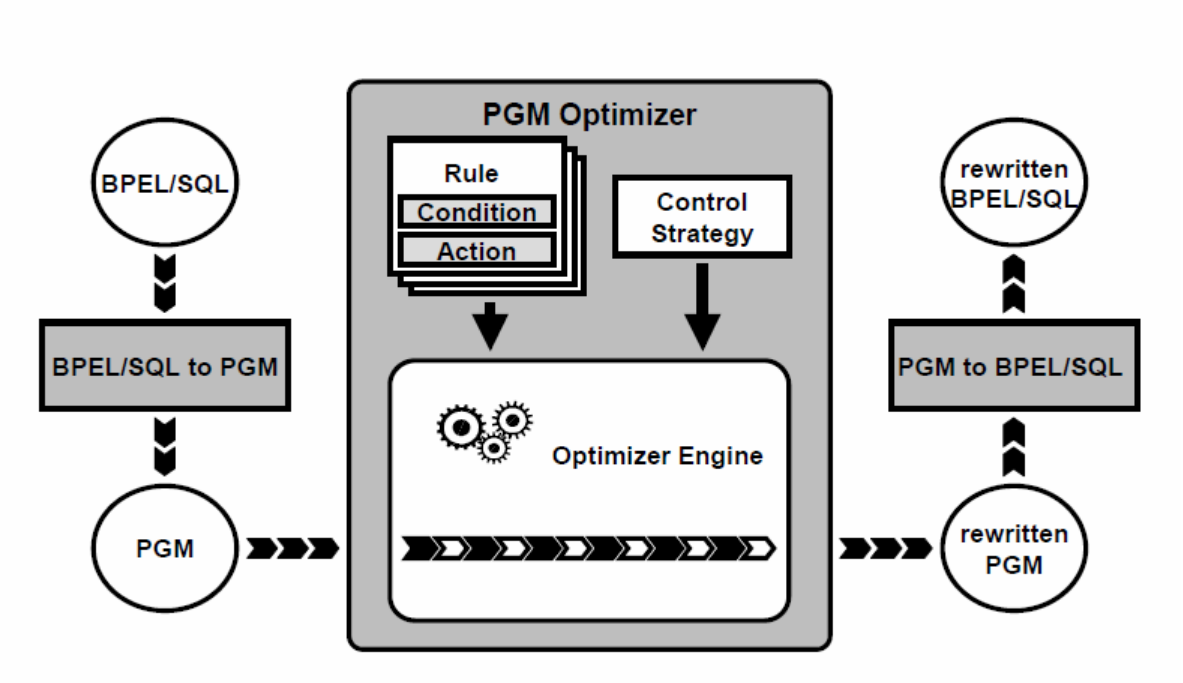

Abbildung 9.2: Ausführungsmodell Quelle: [VSS+07]

Ein Scope bezeichnet dabei einen Kontext für die darin enthaltenen Aktivitäten und definiert lokale Variablen, sowie Fault- und Compensation Handler auf die enthaltenen Aktivitäten. Wenn durch die Anwendung einer Optimierungsregel Aktivitäteten innerhalb eines Scopes entfernt, oder hinzugefügt werden, müssten die Handler des Scopes entsprechend angepasst werden. Betroffene Handler bei der Optimierung zu identifizieren und anzupassen stellt ist jedoch eine sehr komplexe Aufgabe. Aus diesem Grund werden Transformationsregeln, die zu der Optimierungssphäre des SOS gehören nur innerhalb der Grenzen eines einzelnen Scopes angewendet. Eine weitere Optimierungssphäre ist die Loop Optimization Sphere (LOS). Transformationsregeln dieser Sphäre sind jeweils auf eine Schleife (zum Beispiel for each) begrenzt.

Kontrollstrategien in welcher Reihenfolge die Transformationsregeln auf ihre Anwendbarkeit überprüft werden(siehe Abbildung 9.4). Wenn eine Gruppe von Aktivitäten gefunden wurde, die die Bedingungen innerhalb des Condition Parts einer Transformationsregel erfüllen, wird der Action Part dieser Regel auf den PGM Graphen angewandt. Dieser Vorgang wird solange wiederholt, bis das komplette Regelset überprüft wurde.

Abbildung9.4 zeigt die Kontrollstrategien für die unterschiedlichen Optimierungssphären. Dieses Modell für die regelbasierte Optimierung von Prozessen weist eine starke Ähnlichkeit zur dem Modell der Patterntransformation (vgl. Abbildung 2.10) auf. Bei der der Patterntransformation wird ein Workflowgraph mit nicht ausführbaren abstrakten Datenmanagementpatterns anhand von Transformationsregeln mit einen Condition und einem Action Part in einen ausführbaren, konkreten Workflow überführt. Die Reihenfolge, in der die Transformationsregeln dabei auf ihre Anwendbarkeit hin über-

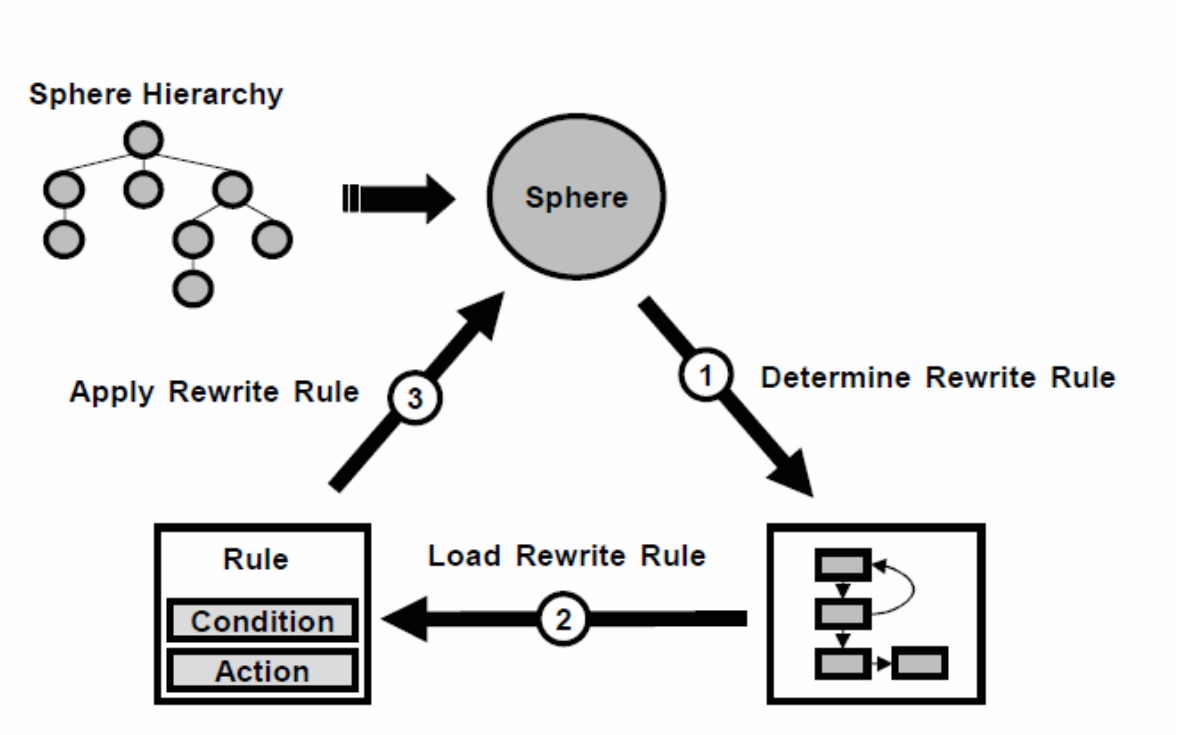

Abbildung 9.3: Ablauf der Optimierung Quelle: [VSS+07]

prüft werden ist durch eine Regelsequenz bestimmt, die Ähnlichkeiten zu einer Kontrollstrategie im PGM-Ansatz aufweist.

Bei der PGM Optimierung wird ein zuvor in das PGM Graphenmodell überführter Workflowgraph anhand von Transformationsregeln optimiert. Dabei entscheidet eine Kontrollstrategie welche Transformationsregeln in welcher Reihenfolge auf einen Bereich des Workflows angewendet werden sollen. Auch hier bestehen Transformationsregeln aus einem Condition und einem Action Part.

Diese starke Ähnlichkeit in Architektur und Ablauf der beiden Modelle legt nahe, dass es möglich sein sollte, diese Modelle miteinander zu verbinden.

Dieser Ansatz für die Optimierung der Datenvararbeitung in Prozessmodellen kann auf unterschiedliche Weise genutzt werden, um die Transformation von Datenmanagementpatterns zu verbessern. Ein möglicher Ansatz wäre es sowohl optimierte, als auch unoptimierte Workflowfragmente für die Patterntransformation bereitzustellen. Bei diesem Ansatz würde die bestehende Patterntransformation um zusätzliche Transformationsregeln auf optimierte Workflowfragmente erweitert. Bei der Patterntransformation würde dann zuerst untersucht, ob eine Abbildung auf diese optimierten Workflowfragmente möglich ist. Dieser Ansatz wurde bereits in der Diplomarbeit von Savas Kalyoncu [Kal15] aufgezeigt und für eine Reihe von Optimierungsregeln und Datenmanagementpatterns implementiert. Der Vorteil dieses Ansatzes ist es, dass an der bestehenden Architektur der Patterntransformation keine Änderungen vorgenommen werden müssen. Die Patterntransformation wird lediglich um zusätzliche Transformationsregeln und Workflowfragmente erweitert.

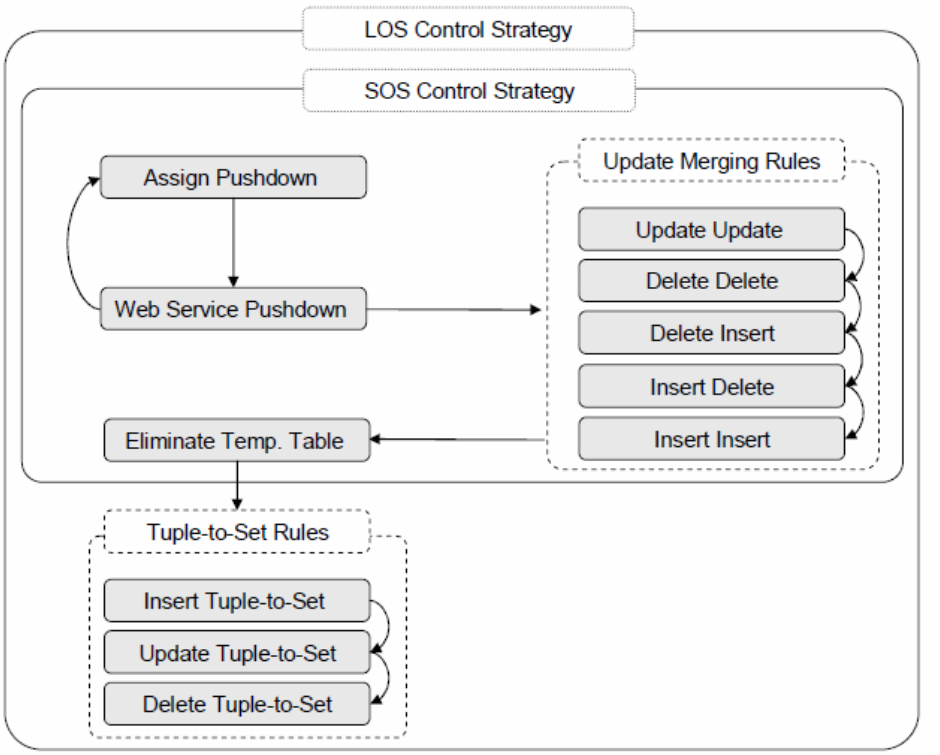

Abbildung 9.4: Kontrollstrategien der Optimierungsspheren Quelle: [VSS+07]

Eine komplexere Variante wäre es, den PGM/F Ansatz mit der bisherigen Patterntransformation zu verbinden. Dazu müsste insbesondere die Patterntransformation dahingehend angepasst werden, dass sie auf das PGM Graphenmodell anwendbar ist. Im Vergleich zu dem vorher vorgestellten Ansatz würde die Variante deutlich aufwendiger in der Umsetzung sein. Transformationsregeln, Workflowfragmente und der Webservice für die Patterntransformation müssten entsprechend auf das neue PGM Graphenmodell umgestellt werden. Außerdem müsste eine zusätzliche Komponente für die Konvertierung des Workflowgraphen aus der ursprünglichen Workflowsprache (zum Beispiel BPEL) in das PGM Modell und die entsprechende Rückführung nach der Patterntransformation implementiert werden.

Diese Variante bietet dafür allerdings auch eine Reihe von Vorteilen, die der einfachere Ansatz nicht aufweist. Durch die Konvertierung des Workflows in das PGM Modell erfolgt eine Abstraktion weg von der ursprünglichen Workflowsprache. Dadurch wäre es möglich bei der Patterntransformation unterschiedliche Workflowsprachen generisch zu unterstützen. Insbesondere erfasst das PGM Modell sowohl den Kontrollfluss eines Workflows als auch den Datenfluss. Ein BPEL Graph hingegen stellt nur den Kontrollfluss des Prozesses explizit dar. Bei der Patterntransformation könnte durch die Umstellung auf das PGM Graphenmodell also auch der Datenfluss mit in die Transformation einbezogen werden. Dadurch könnte die Patterntransformation zur Modellierungszeit verbessert werden, indem Änderungen von Variablenwerten innerhalb des Workflows durch eine Datenflussanalyse mit in die Transformation einbezogen werden könnten. Außerdem könnten auch datenflussorientierte

Workflowsprachen unterstützt werden. Zudem würde dieser Ansatz die Umsetzung der Optimierungsregeln vereinfachen. Bei dem zuvor besprochenend einfacheren Ansatz muss für jedes Pattern und für jedes Workflowfragment überprüft werden, welche Optimierungsregeln anwendbar sind. Anschließend müssen für die optimierten Workflowfragmente neue Transformationsregeln implementiert werden. Bei einer Integrierung der Optimierungsregeln in die Patterntransformation könnten zunächst die ursprünglichen Transformationsregeln für die Patterntransformation angewendet werden und dann im Anschluss darauf das transformierte Workflowfragment anhand der Optimierungsregeln optimiert werden. Ein Nachteil dieses Vorgehens könnte allerdings ein erhöhter Overhead der Patterntransformation zur Laufzeit sein.

Die Vor- und Nachteile dieser beiden Ansätze könnten in einer weiteren Arbeit verglichen und der jeweils bessere Ansatz umgesetzt werden.

10 Zusammenfassung

Im Vorfeld dieser Arbeit wurde die Abbildung von Datenmanagementpatterns auf ausführbare Workflowfragmente prototypisch für die Abbildung während der Modellierungszeit eines Workflows umgesetzt. Bei dieser Umsetzung zeigte, dass einige Datenmanagementpatterns durch eine Patterntransformation zur Laufzeit eines Workflows universeller eingesetzt werden könnten.

Im Rahmen dieser Diplomarbeit wurde der Nutzen einer Abbildung von Datenmanagementpatterns während der Laufzeit eines Workflows untersucht. Dabei wurden mögliche Anwendungsszenarien beschrieben, bei denen eine Patterntransformation während der Laufzeit die universelle Einsetzbarkeit von Datenmanagementpatterns verbessern kann.

Im Anschluss wurden die Anforderungen an eine solche Komponente für die Abbildung zur Laufzeit, welche in dieser Diplomarbeit umgesetzt werden sollte, weiter konkretisiert und anhand dieser Anforderungen unterschiedliche Technologien für die Umsetzung dieser Systemkomponente im Hinblick auf diese Anforderungen näher untersucht. Dabei wurden drei unterschiedliche Frameworks für die Injektion von Workflowfragmenten während der Laufzeit untersucht und das Framework von Lukasz Bialy [Bia14], dass die Anforderungen dieser Arbeit am besten erfüllte, ausgewählt. Um das Hinzufügen und Anpassen von Abbildungsregeln für die Patterntransformation zu erleichtern wurden zwei unterschiedliche Regelengines verglichen. Für die Umsetzung der Patterntransformation zur Laufzeit erwies sich dabei die Drools Regelengine als die besser geeignetere der beiden Regelengines. Zuletzt der Nutzen des RDF-3X Triplestores für die Verwaltung von Ontologien, die vor allem bei der Abbildung von simulationsorentierten Datenmanagementpatterns benötigt werden, näher beschrieben. Für die Umsetzung dieser Arbeit wurde der RDF-3X Triplestore gewählt.

Auf der Grundlage dieser Technologien wurde ein Konzept für die neue Systemkomponente für die Patterntransformation zur Laufzeit und zur Modellierungszeit eines Workflows vorgestellt und der allgemeine Ablauf einer Patterntransformation näher beschrieben. Danach wurde die Umsetzung dieses Konzepts genauer erläutert. Die Patterntransformationskomponente wurde in der Form eines Webservices realisiert. Für die Transformation von Datenmanagementpatterns zur Modellierungszeit wurde das bestehende Eclipse BPEL Plugin entsprechend an den neuen Webservice angepasst. Um die Patterntransformation zur Laufzeit zu unterstützen, wurde das zuvor ausgewählt Framework für die Injektion von Workflowfragmenten zur Laufzeit um einen neuen Service für die Auswahl eines geeigneten Workflowfragments erweitert. Auch dieser Service verwendet für die eigentliche Patterntransformation den neu implementierten Webservice. Dadurch unterstützt die neue Systemkomponente sowhl die Patterntransformation während der Modellierung eines Workflows, als auch zur Laufzeit.

Der Overhead der neu implementierten Patterntransformation während der Laufzeit eines Workflows wurde anhand eines Testworkflows für die biomechanische Simulation von Knochen gemessen.

Anschließend wurde bewertet, inwiefern die neue Systemkomponente den zuvor gestellten Anforderungen gerecht wird. Die Messergebnisse zeigten, dass die Patterntransformation zur Laufzeit eines Testworkflows einen angemessen geringen Overhead aufwies. Außerdem ist das Einfügen und Bearbeiten neuer Transformationsregeln für die Patterntransformation bei dieser neuen Umsetzung einfacher als beim vorherigen Prototypen.

Zum Abschluss wurden einige weitere Möglichkeiten für die Verbesserung der Patterntransformation während der Laufzeit aufgezeigt. Dabei wurden zum einen Möglichkeiten für die Umsetzung eines Sichtenkonpzepts auf die unterschiedlichen Abstraktionsebenen der Datenmanagementpatterns beschrieben [SLL11a]. Ein solches Sichtenkonzept würde Wissenschafltern das Monitoring von Workflows mit Datenmanagementpatterns deutlich erleichtern. Zum anderen wurde aufgezeigt, wie sich die regelbasierte Optimierung der Datenbereitstellung mit der Patterntransformation verbinden lassen würde [VSS+07]. Durch dieses Konzept könnte sich die Datenbereitstellung mittels Datenmanagementpatterns weiter optimieren lassen.

Literaturverzeichnis

[Alv11] M. A. Alvi. *Adaptive Prozessmodellierung mittels mehrerer Abstraktionsschichten.* Dissertation, Stuttgart, Universität Stuttgart, Diplomarbeit, 2011, 2011.

[apa] Apache ODE der Apache Software Foundation. URL http://ode.apache.org/. (Zitiert auf Seite 29)

[Ari12] S. Aristidou. Abstraktionsunterstützung für die Definition des Datenmanagements in Simulationsworkflows. 2012. (Zitiert auf Seite 7)

[Bia14] L. Bialy. Dynamic process fragment injection in a service orchestration engine. 2014. (Zitiert auf den Seiten 5, 8, 40, 45, 46, 57 und 77)

[Boh14] A. Bohrn. *Pattern-basierte Definition der Datenbereitstellung für Simulationen zu Strukturänderungen in Knochen.* Diplomarbeit, Universität Stuttgart, 2014. (Zitiert auf den Seiten 5, 7, 8, 33, 35, 37, 39, 51 und 63)

[bpl] BPEL Designer Projekt der Eclipse Foundation). URL http://www.eclipse.org/bpel/. (Zitiert auf Seite 29)

[Car06] M. Carniel. Jruleengine Website, 2006. URL http://jruleengine.sourceforge.net/. (Zitiert auf den Seiten 6, 8, 47 und 48)

[dro] Drools Documentation Version 6.2. URL http://docs.jboss.org/drools/release/6.2.0.Final/drools-docs/html_single/index.html. (Zitiert auf den Seiten 6, 8, 49 und 50)

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. Conventional workflow technology for scientific simulation. In *Guide to e-Science*, S. 323–352. Springer, 2011. (Zitiert auf den Seiten 7 und 15)

[HKRS07] P. Hitzler, M. Krötzsch, S. Rudolph, Y. Sure. *Semantic Web: Grundlagen.* Springer-Verlag, 2007. (Zitiert auf den Seiten 5, 25, 26 und 27)

[Hum11] A. Hummel. *Ausführung von Workflow-Fragmenten in BPEL.* Diplomarbeit, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2011. URL http://elib.uni-stuttgart.de/opus/volltexte/2011/6853. (Zitiert auf den Seiten 5, 8, 40, 43 und 45)

[Jor07] J. Jordan, D.; Evdemon. Web Services Business Process Execution Language Version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. (Zitiert auf Seite 18)

[K+13] R. Krause, et al. Scientific Workflows for Bone Remodelling Simulations. *Applied Mathematics and Mechanics*, 13(1), 2013. (Zitiert auf Seite 7)

[Kal15] S. Kalyoncu. Optimierung der Datenverarbeitung in Simulationsworkflows. *Universität Stuttgart*, 2015. (Zitiert auf Seite 73)

[KSR+11] R. Krause, D. Schittler, M. Reiter, S. Waldherr, F. Allgöwer, D. Karastoyanova, F. Leymann, B. Markert, W. Ehlers. Bone remodelling: A combined biomechanical and systems-biological challenge. *PAMM*, 11(1):99–100, 2011. (Zitiert auf Seite 33)

[LR00] F. Leymann, D. Roller. Production Workflow: Concepts and Techniques. 2000. (Zitiert auf den Seiten 5, 15, 16 und 17)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis*. Springer DE, 2010. (Zitiert auf den Seiten 5, 11, 12 und 17)

[MM04] F. Manola, E. Miller. Resource description framework (RDF) primer. *W3C Recommendation*, 10, 2004. URL http://www.w3.org/TR/rdf-primer/. (Zitiert auf Seite 25)

[NW08] T. Neumann, G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008. (Zitiert auf den Seiten 8 und 51)

[ode] Apache Orchestration Director Engine. URL http://ode.apache.org/. (Zitiert auf den Seiten 5, 19 und 20)

[Pie12] H. A. Pietranek. *Datenmanagementpatterns in multi-skalaren Simulationsworkflows*. Diplomarbeit, Universität Stuttgart, 2012. (Zitiert auf den Seiten 6, 7, 8, 31, 32, 35, 37, 38 und 39)

[RRS+11] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL-A Framework for Accessing External Data in Simulation Workflows. In *BTW*, S. 534–553. 2011. (Zitiert auf den Seiten 5, 7, 8, 20 und 21)

[RS14] P. Reimann, H. Schwarz. Simulation Workflow Design Tailor-Made for Scientists. *International Conference on Scientific and Statistical Database Management*, 2014. (Zitiert auf Seite 20)

[RSM14a] P. Reimann, H. Schwarz, B. Mitschang. Data Patterns to Alleviate the Design of Scientific Workflows Exemplified by a Bone Simulation. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*. ACM, Aalborg, Denmark, 2014. (Zitiert auf Seite 23)

[RSM14b] P. Reimann, H. Schwarz, B. Mitschang. A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, S. 21–38. Springer, 2014. (Zitiert auf den Seiten 5, 7, 23, 24, 25 und 34)

[Sch11] T. Schliemann. *Unterstützung des "Model-as-you-go"-Ansatzes durch Modell-Versionierung und Instanzmigration*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2011. (Zitiert auf den Seiten 5, 8, 40, 42, 43 und 44)

[sim] SIMPL Rahmenwerk (Source Code). URL https://code.google.com/p/simpl09/. (Zitiert auf Seite 29)

[SLL11a] D. Schumm, G. Latuske, F. Leymann. A prototype for view-based monitoring of BPEL processes. 2011. (Zitiert auf den Seiten 6, 70 und 78)

[SLL+11b] D. Schumm, G. Latuske, F. Leymann, R. Mietzner, T. Scheibler. State Propagation for Business Process Monitoring on Different Levels of Abstraction. In *Proceedings of the 19th European Conference on Information Systems (ECIS 2011)*. 2011. (Zitiert auf Seite 71)

[Stu09] H. Stuckenschmidt. *Ontologien: Konzepte, Technologien und Anwendungen*. Springer-Verlag, 2009. (Zitiert auf Seite 25)

[TDGS14] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields. *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014. (Zitiert auf Seite 7)

[VSS+07] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Maier, T. Kraft. An approach to optimize data processing in business processes. In *Proceedings of the 33rd international conference on Very large data bases*, S. 615–626. VLDB Endowment, 2007. (Zitiert auf den Seiten 6, 71, 72, 73, 74 und 78)

[w3c] World Wide Web Consortium. URL http://www.w3.org/. (Zitiert auf den Seiten 12, 26 und 27)

[wsd] Web Services Description Language (WSDL) 1.1. URL http://www.w3.org/TR/wsdl. (Zitiert auf Seite 14)

Alle URLs wurden zuletzt am 06. 05. 2015 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