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

Diplomarbeit Nr. 3309

Datenmanagementpatterns in multi-skalaren Simulationsworkflows

Henrik Andreas Pietranek

Studiengang:	Informatik
Prüfer:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer:	Dipl.-Inf. Peter Reimann
begonnen am:	08. März 2012
beendet am:	07. September 2012
CR-Klassifikation:	D.2.11, H.2.5, H.4.1, I.6.7

Inhaltsverzeichnis

1. Einleitung **11**

2. Grundlagen **15**

2.1. XML . . . 15
2.1.1. Tags, Elemente und Attribute . . . 15
2.1.2. Aufbau eines XML-Dokuments . . . 16
2.1.3. XML-Namensräume . . . 16
2.1.4. Document Type Definiton und XML Schema . . . 17
2.2. Serviceorientierte Architektur . . . 20
2.2.1. Merkmale einer SOA . . . 20
2.2.2. Beteiligte und deren Aktionen . . . 21
2.2.3. Web Service . . . 22
2.3. Workflow . . . 24
2.3.1. Grundlagen der Workflow-Technologie . . . 24
2.3.2. Workflow-Sprachen . . . 27
2.3.3. Arten von Workflows . . . 27
2.3.3.1. Business Workflows . . . 28
2.3.3.2. Scientific Workflows . . . 28
2.3.3.3. ETL Workflows . . . 29
2.3.4. WS-BPEL . . . 30
2.3.5. Architektur eines Scientific Workflow Management Systems . . . 32
2.4. Simulation . . . 34
2.4.1. Eigenschaften einer Simulation . . . 34
2.4.2. Multi-* Simulationen . . . 36
2.4.3. Simulationsrahmenwerke . . . 37
2.4.3.1. Pandas . . . 37
2.4.3.2. ChemShell . . . 37
2.4.4. Finite Element Methode . . . 38
2.5. Datenbank . . . 39
2.5.1. Relationale Datenbanken . . . 40
2.5.1.1. SQL . . . 41
2.5.2. XML-Datenbanken . . . 41
2.5.2.1. XPath . . . 43
2.5.2.2. XQuery . . . 45

3. SIMPL-Rahmenwerk **47**

3.1. Gründe für SIMPL . . . 47

3.2. Architektur des SIMPL-Rahmenwerks . . . 48
3.3. BPEL-DM . . . 49
3.4. SIMPL Core . . . 50
3.5. Resource Management . . . 51
3.6. Ablauf einer DM Aktivität . . . 53
3.7. Datenmanagementpatterns . . . 54

4. Anwendungsfälle **59**
4.1. Chemische Reaktion unter Verwendung eines Katalysators . . . 59
4.2. Knochenmodellierung mit Pandas . . . 60
4.3. Pandas-Matlab Kopplung . . . 61

5. Bestandsaufnahme **65**
5.1. Der erweiterte Eclipse BPEL Designer . . . 65
5.2. Resource Management . . . 67
5.3. SIMPL Core . . . 70
5.4. Datenmanagementpatterns . . . 73
5.4.1. Formalisierte Patterns . . . 73
5.4.1.1. Data Transfer and Transformation Pattern . . . 73
5.4.1.2. Data Format Conversion Pattern . . . 76
5.4.1.3. Data Iteration Pattern . . . 76
5.4.2. Bisherige Architektur des Abbildungsmechanismus . . . 77
5.4.3. Beispiele der Pandas Preprocessing-Phase . . . 81
5.4.4. Beispiele der Pandas Postprocessing-Phase . . . 84

6. Konzeptionelle Änderungen und Erweiterungen **87**
6.1. Zugriffsmechanismen für Dateisysteme . . . 87
6.1.1. Unix-basierte Dateisysteme . . . 88
6.1.2. Zugriff auf entfernte Rechner . . . 89
6.2. Datenmanagementpatterns in der Pandas Preprocessing- und Postprocessing-Phase . . . 90
6.2.1. Das einfache Container-to-Container Pattern . . . 92
6.2.2. Das komplexe Container-to-Container Pattern . . . 93
6.2.3. Das Data Format Conversion Pattern . . . 97
6.2.4. Das sequentielle Data Iteration Pattern . . . 99
6.2.5. Das Multiple Data Transfer Pattern . . . 103
6.3. Datenmanagementpatterns in der Pandas-Matlab Kopplung . . . 104
6.3.1. Das parallele Data Iteration Pattern . . . 105
6.3.2. Das Get Time Step Pattern . . . 121
6.3.3. Das Data Format Conversion Pattern . . . 123
6.4. Kontrollstrategien . . . 125

7. Umsetzung **127**
7.1. Zugriffsmechanismen für Dateisysteme . . . 127

7.2. Erweiterung der GUI . . . 128
7.2.1. DM Aktivitäten . . . 128
7.2.2. Datenmanagementpatterns . . . 130
7.3. Erweiterungen am Resource Management . . . 135
7.4. Transformation der Datenmanagementpatterns . . . 138

8. Evaluation **141**
8.1. Bewertung der geänderten Zugriffsmechanismen . . . 141
8.2. Bewertung der Workflow-Fragmente und Transformationsregeln . . . 142
8.3. Die Notwendigkeit einer Datenflussanalyse bzw. einer Transformation zur Laufzeit . . . 145
8.4. Weitere Optimierungsmöglichkeiten . . . 147
8.5. Datenmanagementpatterns in Bezug auf ChemShell . . . 148

9. Zusammenfassung und Ausblick **151**

A. Struktur von WS-BPEL **155**

B. Workflow-Fragmente **157**

Literaturverzeichnis **167**

Abbildungsverzeichnis

2.1. Beziehungen innerhalb einer SOA (in Anlehnung an [Mel10]) 21
2.2. Prozess und Workflow [LR99] 25
2.3. Die drei Workflow-Dimensionen [LR99] 25
2.4. WfMC-Workflow-Referenz-Modell [Mel10] 26
2.5. Klassifizierung von Workflows [RSM11] 28
2.6. Architektur eines sWfMS [RRS+11] und [GSK+11] 33
2.7. Multi-Domänen, Multi-Physiken, Multi-Skalen Simulation des menschlichen Körpers [KME12] 36
2.8. FEM: Ein Problem wird in mehrere Teilprobleme zerlegt.[1] 38

3.1. Das SIMPL-Rahmenwerk eingebettet in ein sWfMS [RRS+11] 48
3.2. Datenaustausch zwischen Execution Engine und SIMPL Core [RRS+11] 51
3.3. Zusammenhang zwischen den verschiedenen Metadaten [RRS+11] 52
3.4. Interaktion zwischen den verschiedenen Service Bus Komponenten [RRS+11] 53
3.5. Beispiel eines Join Patterns [RRS+11] 54
3.6. Modell für die Transformation eines Datenmanagementpatterns [Rei11] 55
3.7. Hierarchie der Datenmanagementpatterns [RM11] 56

4.1. ChemShell: Chemische Reaktion unter Verwendung eines Katalysators [Mü10] 60
4.2. Der Pandas Workflow [RRS+11] 61
4.3. Die Pandas-Matlab Kopplung [Dor11] 62
4.4. Exemplarische Verteilung der Gausspunkte (in Anlehnung an [Ari12]) 63
4.5. Dateitransfer in der Pandas-Matlab Kopplung (eine Matlab-Instanz) 64

5.1. Der erweiterte Eclipse BPEL Designer 66
5.2. Die Tabelle DataContainer_Reference_Types 69
5.3. Data Transfer and Transformation Pattern [RSRM] 73
5.4. Container-to-Container Pattern [RSRM] 74
5.5. Data Split Pattern [RSRM] 75
5.6. Data Merge Pattern [RSRM] 75
5.7. Paralleles Data Iteration Pattern [RSRM] 76
5.8. Sequentielles Data Iteration Pattern [RSRM] 77
5.9. Bisherige Architektur des Abbildungsmechanismus [Ari12] 80
5.10. Die Pandas Preprocessing-Phase in detaillierter Form 82
5.11. Die Pandas Postprocessing-Phase in detaillierter Form (nach [Ari12]) 84

6.1. Auszüge aus der RandomFileDataFormat Definition 88

6.2. Dateitransfer mithilfe des SSHConnectors (**vor** den durchgeführten Änderungen) 89
6.3. Dateitransfer mithilfe des SSHConnectors (**nach** den durchgeführten Änderungen) 90
6.4. Transformation des komplexen Container-to-Container Patterns 94
6.5. Der überarbeitete Data-Manager Workflow inklusive dem parallelen Data Iteration Pattern 106
6.6. Typen für die Referenzierung von Objekten in einem Dateisystem 108
6.7. Exemplarischer Inhalt der Tabelle *gausspunkte* in der PandasDB 109
6.8. Informationen über Elemente sowie Gausspunkte in der neu erzeugten Tabelle 110
6.9. Das parallele Data Iteration Pattern nach der Transformation 111

7.1. Die View *Palette* wurde um Datenmanagementpatterns ergänzt 131
7.2. Die *Property Section* für das Container-to-Container Pattern 132
7.3. Der Menüleisteneintrag *SIMPL* 134

8.1. Grad der universellen Einsetzbarkeit der einzelnen Workflow-Fragmente . . . 144
8.2. Datenmanagementpatterns im ChemShell Workflow (in Anlehnung an [Mü10]) 148

Tabellenverzeichnis

2.1. Relationale Datenbank: Datensätze innerhalb einer Tabelle 40
2.2. Allgemeine Form einer SQL-Anfrage [KE11] 41
2.3. Ergebnis einer SQL-Anfrage 41

7.1. Parametrisierung der IssueCommand Aktivität 129
7.2. Parametrisierung der RetrieveData Aktivität 129
7.3. Parametrisierung der QueryData Aktivität 129
7.4. Parametrisierung der WriteDataBack Aktivität 130
7.5. Parametrisierung der TransferData Aktivität 130
7.6. Parametrisierung des Container-to-Container Patterns 132
7.7. Parametrisierung des Data Format Conversion Patterns Patterns 132
7.8. Parametrisierung des sequentiellen Data Iteration Patterns 132
7.9. Parametrisierung des Multiple Data Transfer Patterns 133
7.10. Parametrisierung des parallelen Data Iteration Patterns 133
7.11. Parametrisierung des Get Time Step Patterns 133

Verzeichnis der Listings

2.1. Ein einfaches XML-Fragment . . . 15
2.2. Mehrdeutigkeiten in XML . . . 16
2.3. Namensräume in XML . . . 17
2.4. Dokument Type Definition (in Anlehnung an [Sch03]) . . . 18
2.5. Ausschnitt aus einem XML Schema . . . 19
2.6. Beispiel einer SOAP Nachricht [GHM+07] . . . 22
2.7. Komponenten einer WSDL Beschreibung [CMRW07] . . . 23
2.8. Erste Ebene eines WS-BPEL Dokuments (nach [Mel10]) . . . 30
2.9. Beispiel einer receive sowie reply Aktivität [Mel10] . . . 31
2.10. Beispiel einer synchronen invoke Aktivität [Mel10] . . . 31
2.11. Ausschnitt aus einem XML-Dokument . . . 44

5.1. Referenzierung eines im Resource Management registrierten Containers . . . 70
5.2. Referenzierung eines Containers über den lokalen Bezeichner . . . 70
5.3. Interface Beschreibung eines Konnektors [SIM] . . . 71
5.4. Interface Beschreibung eines Konverters [SIM] . . . 72
5.5. Transfer der einzelnen Dateien auf den Workflow-Rechner . . . 83

6.1. Basistyp für die Referenzierung eines Containers über dessen lokalen Bezeichner 92
6.2. Beispiel einer Data Container Reference List . . . 100
6.3. Beispiel einer Data Container Reference List mit zusätzlichen Informationen . 101
6.4. Die MatlabNodeReferenceList Definition . . . 107
6.5. Exemplarisches Ergebnis der RetrieveData Aktivität *getElements_Gausspoints* . 113
6.6. Fragment für die Definition eines *partner links* . . . 117
6.7. Fragment für die Definition eines *partner link types* . . . 117

7.1. Auszüge aus dem UnixLocalFSConnector . . . 127
7.2. Auszüge aus der Klasse *SIMPLCoreImpl* . . . 128
7.3. Interface Beschreibung einer Kontrollstrategie . . . 139
7.4. Interface Beschreibung einer Transformationsregel . . . 139

A.1. Grundlegende Struktur von WS-BPEL [JE07] . . . 155

B.1. Das Workflow-Fragment *simplC2C.xml* . . . 157
B.2. Das Workflow-Fragment *complexC2C.xml* . . . 157
B.3. Das Workflow-Fragment *dataFormatConversion_script.xml* . . . 159
B.4. Das Workflow-Fragment *sequentialDIP.xml* . . . 160
B.5. Das Workflow-Fragment *mdt.xml* . . . 162

B.6. Das Workflow-Fragment *getTimeStep_variable.xml* 163
B.7. Das Workflow-Fragment *getTimeStep_literal.xml* 163
B.8. Das Workflow-Fragment *getTimeStep_first_last.xml* 163
B.9. Das Workflow-Fragment *dataFormatConversion_importWS.xml* 164

1. Einleitung

In den vergangenen Jahren haben sich im unternehmerischen Umfeld Workflows zur Beschreibung und Ausführung von (Geschäfts-)Prozessen durchgesetzt [LR99]. Seit kurzem wird diese Technologie auch in der Wissenschaft eingesetzt. Z.B. werden Simulationsabläufe als Workflows modelliert. Charakteristisch für solche Simulationen bzw. Simulationsabläufe sind komplexe mathematische Berechnungen sowie verschiedene Aufgaben im Bereich der Datenverwaltung und Datenbereitstellung. Oftmals müssen große Datenmengen, die in proprietären Formaten vorliegen, aus verschiedenen Quellen verarbeitet werden. Damit diese Daten durch einen Simulationsworkflow und den von ihm eingebundenen Programmen und Diensten verarbeitet werden können, müssen sie in passende Eingabeformate transformiert werden. Dies hat einen erhöhten Arbeitsaufwand für den Modellierer zur Folge, der typischerweise der Wissenschaftler selbst ist. Dieser muss zum einen adäquate Quellen finden und zum anderen benötigte Transformationen implementieren oder manuell durchführen. Gerade bei umfangreichen Simulationen, die eine Vielzahl an Datenquellen benötigen, führt dies aufgrund der enormen Komplexität zu Problemen. Einerseits kann sich der Wissenschaftler nicht mehr auf sein Kernproblem (die eigentliche Simulation) konzentrieren, andererseits steigt oftmals die Fehlerrate bei der Datenverarbeitung, da selbst mit benötigtem Fachwissen solche Transformationen fehleranfällig sind [RRS+11].

Um diese Probleme zu lösen, wurde das SIMPL-Rahmenwerk (SimTech - Information Management, Processes and Languages) entwickelt. Das SIMPL-Rahmenwerk ist in ein Scientific Workflow Management System eingebettet und schafft eine Abstraktionsebene für die Definition des Datenmanagements. SIMPL bietet einheitliche Zugriffsmethoden, um, aus einem Simulationsworkflow heraus, auf beliebige Datenquellen zuzugreifen. Zur Prozessdefinition wird die Business Process Execution Language (BPEL) in einer erweiterten Form verwendet. Die Business Process Execution Language extension for Data Management (BPEL-DM) erweitert BPEL um zusätzliche Datenmanagementaktivitäten. Dadurch ist es möglich, das Datenmanagement direkt in Workflow-Modellen zu definieren [RRS+11]. Das SIMPL-Rahmenwerk wurde prototypisch umgesetzt und unterstützt bisher den Zugriff auf relationale Datenbanken, auf XML-Datenbanken, auf ein Sensornetz sowie das Windows-Dateisystem.

Ein weiterer Bestandteil des SIMPL-Rahmenwerks sind Datenmanagementpatterns. Dabei handelt es sich um vorgefertigte Datenmanagement-Operationen, die nur noch parametrisiert werden müssen. Bei der Konkretisierung eines gewählten Patterns wird der Modellierer dann in einem semi-automatischen Prozess unterstützt und muss keine oder deutlich weniger konkrete Operationen definieren [RRS+11].

In [Ari12] wurden bereits erste Datenmanagementpatterns erarbeitet. So können z.B. Daten zwischen zwei Datenressourcen ausgetauscht werden. Des Weiteren wurde ein Konzept erarbeitet, um Datenmanagementpatterns auf ausführbare Workflow-Fragmente abzubilden. Dieses Konzept nutzt Transformationsregeln sowie im Resource Management gespeicherte Metadaten über beteiligte Ressourcen als Basis.

Im Rahmen dieser Diplomarbeit wird das in [Ari12] vorgestellte Konzept erweitert und wenn nötig verändert, um auf multi-skalare Simulationen angewendet werden zu können. Dabei wird in erste Linie die Pandas-Matlab Kopplung, eine multi-skalare Simulation, betrachtet. Es wird untersucht, ob die in [Ari12] definierten Datenmanagementpatterns in den beteiligten Workflows vorkommen und, ob es noch andere bisher nicht identifizierte Datenmanagementpatterns gibt. Für alle diese Patterns werden Workflow-Fragmente und Transformationsregeln definiert bzw. deren Definition überarbeitet. Um die Ausführungsumgebung für ETL Patterns zu verbessern, werden die Zugriffsmechanismen von SIMPL überarbeitet.

Bisher fanden alle Entwicklungen bzgl. der Datenmanagementpatterns nur auf konzeptioneller Ebene statt. Im Rahmen dieser Arbeit wird die prototypische Umsetzung des SIMPL-Rahmenwerks um Datenmanagementpatterns erweitert.

Abschließend wird der im Rahmen dieser Arbeit erarbeitete Ansatz auf dessen Eignung für multi-skalare Simulationsworkflows evaluiert. Unter anderem werden die definierten Workflow-Fragmente und Transformationsregeln auf deren universelle Einsetzbarkeit bewertet, zukünftig mögliche Verbesserungen und Optimierungsmöglichkeiten aufgezeigt und die Einsatzmöglichkeiten von Datenmanagementpatterns in einem ChemShell Simulationsworkflow betrachtet.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Die Grundlagen dieser Arbeit werden hier beschrieben.

Kapitel 3 – SIMPL-Rahmenwerk Das SIMPL-Rahmenwerk wird in diesem Kapitel detailliert vorgestellt.

Kapitel 4 – Anwendungsfälle Die im Rahmen dieser Arbeit betrachteten Anwendungsfälle für Simulationsworkflows werden in diesem Kapitel erläutert.

Kapitel 5 – Bestandsaufnahme Die bestehende Implementierung von SIMPL wird in diesem Kapitel erläutert. Weiterhin wird das bisherige Konzept der Datenmanagementpatterns sowie deren Transformation auf ausführbare Workflow-Fragmente beschrieben und evaluiert.

Kapitel 6 – Konzeptionelle Änderungen und Erweiterungen Auf konzeptionelle Änderungen und Erweiterungen wird in diesem Kapitel eingegangen. Die erweiterten Zugriffsmechanismen auf Dateisysteme werden betrachtet und Workflow-Fragmente sowie zugehörige Transformationsregeln für Datenmanagementpatterns in den betrachteten Anwendungsfällen, insbesondere in der Pandas-Matlab Kopplung, werden definiert.

Kapitel 7 – Umsetzung Auf die konkrete Umsetzung der Modellierung von Datenmanagementpatterns und der Pattern Transformation in die prototypische Umsetzung des SIMPL-Rahmenwerks wird in diesem Kapitel eingegangen.

Kapitel 8 – Evaluation Dieses Kapitel evaluiert den in dieser Diplomarbeit erarbeiteten Ansatz auf dessen Eignung für multi-skalare Simulationsworkflows. Unter anderem werden die definierten Workflow-Fragmente und Transformationsregeln auf deren universelle Einsetzbarkeit bewertet, zukünftig mögliche Verbesserungen und Optimierungsmöglichkeiten aufgezeigt und die Einsatzmöglichkeiten von Datenmanagementpatterns in einem ChemShell Simulationsworkflow betrachtet.

Kapitel 9 – Zusammenfassung und Ausblick Die Ergebnisse dieser Arbeit werden hier zusammengefasst. Des Weiteren werden zukünftig zu betrachtende Fragestellungen und Aufgaben dargelegt.

2. Grundlagen

In diesem Kapitel werden die Grundlagen, die für das Verständnis der vorliegenden Arbeit nötig sind, erläutert. Es werden die Themen XML, Serviceorientierte Architektur, Workflow, Simulation und Datenbanken behandelt.

2.1. XML

XML, die Extensible Markup Language, ist eine Auszeichnungssprache. Mithilfe einer Auszeichnungssprache können Dokumentformate bzw. deren Inhalte beschrieben werden. XML wurde unter der Schirmherrschaft des World Wide Web Consortiums (W3C) entwickelt und 1998 als *Recommendation* veröffentlicht. Im Jahr 2008 wurde die fünfte Version der XML-Spezifikation [BPSM+08] veröffentlicht. Heute wird XML insbesondere als standardisiertes Datenaustauschformat, welches anwendungs- und plattformunabhängig ist, verwendet. Die folgenden Abschnitte basieren auf [KE11], [Sch03] und [Seb10].

2.1.1. Tags, Elemente und Attribute

Listing 2.1 zeigt einen Ausschnitt eines XML-Dokuments. Die Zeichenfolgen zwischen den spitzen Klammern heißen Tags. Zwei Tags bilden jeweils ein Paar. Es gibt ein Start- und ein End-Tag. Ein End-Tag besitzt immer die gleichen Zeichenfolgen wie das zugehörige Start-Tag und wird zusätzlich durch ein „/“eingeleitet. In unserem Beispiel ist u.a. <*Erster_Vorname*> ein Start- und </*Erster_Vorname*> das zugehörige End-Tag. Zwischen einem Start-Tag und seinem zugehörigen End-Tag werden die eigentlichen Information gespeichert. Die Zeichenfolge <*Erster_Vorname*>*Henrik*</*Erster_Vorname*>, also ein Start- und End-Tag sowie der Inhalt dazwischen, bildet ein Element. Das Wurzelelement ist das oberste Element der Hierachie, die vom XML-Dokument gebildet wird. Elemente können einfachen Text, andere Elemente oder eine Mischung aus beidem beinhalten. So ist es möglich, dass ein Element beliebig viele Unterelemente beinhaltet. Im gegebenen XML-Fragment ist das Element Student das Wurzelelement, da es alle anderen Elemente umschließt.

Listing 2.1 Ein einfaches XML-Fragment

```
<Student>
  <Erster_Vorname>Henrik</Erster_Vorname>
  <Zweiter_Vorname>Andreas</Zweiter_Vorname>
  <Nachname Geburtsjahr="1986">Pietranek</Nachname>
</Student>
```

Zusätzlich können für ein Element noch ein oder mehrere Attribute eines bestimmten Wertebereichs angegeben werden. Mithilfe von Attributen werden zusätzliche Elementeigenschaften beschrieben. Attribute werden immer im Start-Tag eines Elementes positioniert. Im Beispiel enthält das Element Nachname noch zusätzlich das Attribut Geburtsjahr.

2.1.2. Aufbau eines XML-Dokuments

Ein XML-Dokument besteht aus den folgenden drei Teilen:

- optionaler Prolog
- optionales Schema
- ein einziges Wurzelelement

Der optionale Prolog enthält Informationen über die zugrundeliegende XML-Version sowie den Zeichensatz zur Speicherung des Dokumentinhalts und deklariert ein Dokument als XML-Dokument. Obwohl der Prolog optional ist, kann er die weitere Verarbeitung eines Dokuments vereinfachen, da dadurch bekannt ist, um welches Format es sich handelt. Das optionale Schema legt Anforderungen fest, die das Dokument erfüllen muss, wie z.B. die Struktur des Dokumentinhalts. Es gibt zwei Möglichkeiten solche Anforderungen zu definieren: Die Erstellung einer Document Type Definition (DTD) oder die Verwendung des neueren XML Schema. Abschnitt 2.1.4 beschäftigt sich mit diesen beiden Möglichkeiten. Das Wurzelelement, das oberste Element der Hierachie, beinhaltet alle weiteren Unterelemente.

Ein XML-Dokument heißt wohlgeformt, wenn es den syntaktischen Anforderungen von XML genügt. Dazu gehört unter anderem, dass es nur ein einziges Wurzelemement gibt und dass ein Element nicht mehrere Attribute mit demselben Namen beinhaltet. Gültig oder valide heißt ein XML-Dokument, wenn es wohlgeformt ist und zusätzlich die in einer DTD oder in einem XML Schema definierten Anforderungen erfüllt.

2.1.3. XML-Namensräume

Ein einzelnes Wort kann mehrere Bedeutungen haben. Listing 2.2 zeigt ein XML-Fragment, in dem das Element *Titel* zweimal vorkommt.

Listing 2.2 Mehrdeutigkeiten in XML

```
<Vorlesungen>
  <Vorlesung VorNr="111">
    <Titel>Datenbanksysteme</Titel>
    <Dozent>
      <Name>Mustermann</Name>
      <Titel>Professor</Titel>
    </Dozent>
    <SWS>4</SWS>
  </Vorlesung>
</Vorlesungen>
```

Im ersten Fall wird der Name einer Vorlesung und im zweiten Fall der akademische Grad des Dozenten beschrieben. In *einem* XML-Dokument werden also zwei verschiedene Vokabulare verwendet.

Eine mögliche Lösung, um diese Mehrdeutigkeiten aufzulösen, ist die Verwendung von Namensräumen. Jedem Element wird ein global eindeutiger Name zugewiesen, der aus dem Namensraum und dem eigentlichen lokalen Namen des Elementes besteht. Die Identifizierung des zugehörigen Namensraums erfolgt durch einen global eindeutigen URI (Uniform Resource Identifier). Listing 2.3 zeigt den Inhalt von Listing 2.2 in modifizierter Fassung. Standardmäßig wird der Namensraum *Universität* verwendet. Bei der Beschreibung des Dozenten wird auf den Namensraum *Person* zurückgegriffen. Auf diese Weise können Mehrdeutigkeiten beseitigt werden.

Listing 2.3 Namensräume in XML

```
<Universitaet xmlns="http://www.BeispielUniversitaet.de/universitaet">
  <Vorlesungen>
    <Vorlesung VorNr="111">
      <Titel>Datenbanksysteme</Titel>
      <Dozent xmlns:person="http://www.BeispielUniversitaet.de/person">
        <person:Name>Mustermann</person:Name>
        <person:Titel>Professor</person:Titel>
      </Dozent>
      <SWS>4</SWS>
    </Vorlesung>
  </Vorlesungen>
</Universitaet>
```

2.1.4. Document Type Definiton und XML Schema

XML wird insbesondere zum Daten- bzw. Informationsaustausch eingesetzt. Deshalb ist es wichtig, dass beide Kommunikationspartner dasselbe Verständnis von den ausgetauschten Informationen haben. Das verwendete Vokabular wird durch Namensräume definiert. Eine DTD oder ein XML Schema legt weiterhin die Struktur eines Dokuments fest.

Eine DTD beginnt stets mit der Zeichenfolge „<!DOCTYPE“. Anschließend folgen für jeden Dokumenttyp Elementtyp- und Attributtypdefinitionen. Listing 2.4 zeigt eine DTD für Informationen zu Vorlesungen. Jede Vorlesung hat als Attribut eine Vorlesungsnummer, als Elemente einen Vorlesungsnamen und eine Anzahl an Semesterwochenstunden (SWS) und kann von einem oder beliebig vielen Dozenten gehalten werden. Würde man das + bei „Dozent“ durch ein * ersetzen, könnte die Vorlesung auch von keinem Dozenten gehalten werden. Das Element Vorlesung muss das Attribut *VorNr* enthalten, da es als *REQUIRRED* angegeben ist. Optionale Attribute können durch den Zusatz *IMPLIED* definiert werden. Es ist nicht genau festgelegt, welche Eigenschaften eines Elementes als Attribut, und welche als Unterelement modelliert werden sollten. Hier geben die eigenen Vorlieben den Ausschlag.

Listing 2.4 Dokument Type Definition (in Anlehnung an [Sch03])

```
<!DOCTYPE Vorlesung[
  <!ELEMENT Vorlesung (VorName, Dozent+, SWS)>
  <!ATTLIST Vorlesung VorNr CDATA #REQUIRED>
  <!ELEMENT VorName (#PCDATA)>
  <!ELEMENT Dozent (#PCDATA)>
  <!ELEMENT SWS (#CDATA)
]>

<Vorlesung VorNr="111">
  <VorName>Datenbanksysteme</VorName>
  <Dozent>Mustermann</Dozent>
  <SWS>4</SWS>
</Vorlesung>
```

Jedoch besitzt eine Document Type Definiton nur eine begrenzte Ausdruckskraft. [KE11] bemängelt die folgenden Aspekte:

> „So lassen sich keine komplexen Integritätsbedingungen angeben und auch die Datentypen sind sehr eingeschränkt (im Wesentlichen gibt es nur Strings, d.h. PCDATA).“

Aus diesem Grund wurde vom World Wide Web Consortium das XML Schema entwickelt. Nach [Seb10] gibt es vier Anwendungsgebiete, die durch XML Schema unterstützt werden:

- Validierung
- Abfrageunterstützung
- Datenbindung und Editierung
- Dokumentation

Das Anwendungsgebiet Validierung kann in die beiden Teilgebiete Strukturvalidierung und Datenvalidierung unterteilt werden. Bei der Strukturvalidierung wird überprüft, ob definierte Elemente und Attribute im Dokument vorkommen. Der Inhalt der Elemente und Attribute wird bei der Datenvalidierung bewertet. Das Element *Telefonnummer* darf z.B. nur Zahlen enthalten (Validierung). XPath 2.0 [BBC+10] und XQuery 1.0 [BCF+10] bieten Unterstützung von XML Schema Datentypen. Auf diese Weise können Dokumente genauer analysiert und Inhalte abgefragt werden (Abfrageunterstützung). Neuere XML-Editoren können für ein gegebenes XML Schema Instanzdokumente erzeugen, die dann editiert werden können (Datenbindung und Editierung). Des Weiteren kann ein XML Schema zur Dokumentation verwendet werden. Das verwendete Vokabular kann festgehalten oder ein Schema bzw. ein Instanzdokument durch einen Editor in eine für Menschen verständliche Fassung (Visualisierung) überführt werden (Dokumentation).

Listing 2.5 zeigt einen Ausschnitt aus einem XML Schema zur Beschreibung von Universitäten und deren Fakultäten. Zur einfacheren Darstellung wurden benötigte Definitionen teilweise weggelassen. Das oberste Element der Hierachie ist das Element *Universitaet*, welches vom Typ *tUniversitaet* ist. Dieser Typ wird anschließend definiert. Er besteht aus den

zwei Elementen *NameUndOrt* und *Fakultaeten*. Das Element *NameUndOrt* ist wiederum ein *complexType*, der aus den beiden Elementen *NameDerUniversitaet* und *OrtDerUniversitaet* besteht. Dem Typ für das Element *NameUndOrt* wurde jedoch kein Name zugewiesen. Stattdessen wurde eine lokale, anonyme Typdefinition erstellt.

Listing 2.5 Ausschnitt aus einem XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Universitaet" type ="tUniversitaet"/>

  <xs:complexType name="tUniversitaet">
    <xs:sequence>
      <xs:element name="NameUndOrt">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NameDerUniversitaet" type="xs:string"/>
            <xs:element name="OrtDerUniversitaet" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Fakultaeten">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Fakultaet" minOccurs="1" maxOccurs="unbounded"
                type="tFakultaet"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="tFakultaet">
    <xs:sequence>
      <xs:element name="NameDerFakultaet" type="xs:String"/>
      <xs:element name="VorlesungenDerFakultaet" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Vorlesung" type="tVorlesung"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="NummerDerFakultaet" type="xs:ID"/>
  </xs:complexType>

  <xs:complexType name="tVorlesung">...</xs:complexType>

</xs:schema>
```

Die einzelnen Fakultäten, als Unterelemente des Elementes *Fakultaeten* definiert, sind vom Typ *tFakultaet*. Da für das Element *Fakultaet minOccurs=1* angegeben wurde, muss eine

Universität mindestens eine Fakultät haben. Nach oben hingegen gibt es keine Beschränkung, da *maxOccurs=unbounded* festgelegt wurde. Werden für *minOccurs* und *maxOccurs* keine Werte angegeben, werden die beiden Werte standardmäßig als 1 interpretiert. Jede Fakultät hat eine Fakultätsnummer. Diese Nummer wird mithilfe des Attributs *NummerDerFakultaet* festgehalten. Dieses Attribut ist vom Typ *ID*. Das bedeutet, dass diese Nummer eindeutig sein muss. Es darf im gesamten XML-Dokument kein anderes Element vom Typ *tFakultaet* geben, bei dem ein Attribut vom Typ *ID* denselben Wert hat, so dass dieses Attribut einen Primärschlüssel darstellt. Mithilfe des Elementes *NameDerFakultaet* wird der Fakultätsname gespeichert. Die Vorlesungen, die eine Fakultät anbietet, werden als Kindelemente des Elementes *VorlesungenDerFakultaet* gespeichert. Für detailliertere Informationen wird auf [KE11] verwiesen.

2.2. Serviceorientierte Architektur

Der Begriff *Serviceorientierte Architektur (SOA)* steht für ein Architekturparadigma und keine konkrete Technik bzw. Implementierungsmethode. Bisher gibt es keine einheitliche Definition des Begriffs. Teilweise beziehen Definitionen Aspekte mit ein, die bei anderen Definition vollständig vernachlässigt werden. [Mel10] definiert den Begriff Serviceorientierte Architektur folgendermaßen:

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

Bei einer Serviceorientierten Architektur steht der Begriff Service im Mittelpunkt. Im weiteren Verlauf dieser Arbeit wird die deutsche Übersetzung *Dienst* verwendet.

2.2.1. Merkmale einer SOA

Nach [Mel10] zählen unter anderem die folgenden Punkte zu den grundlegenden Merkmalen einer SOA: *Lose Kopplung* bedeutet, dass Dienste erst bei Bedarf von Anwendungen oder anderen Diensten dynamisch gesucht und eingebunden werden. Bei der Übersetzung einer Anwendung steht noch nicht fest, welcher Dienst eine anfallende Aufgabe übernimmt. Dazu wird ein *Verzeichnisdienst* benötigt, in dem alle verfügbaren Dienste registriert sind. Bei Bedarf sucht eine Anwendung einen passenden Dienst in diesem Verzeichnissdienst. Dazu ist es nötig, dass *offene Standards* verwendet werden und Beschreibungen von Schnittstellen in einer maschinenlesbaren Form vorliegen. Ein weiteres Merkmal einer SOA ist die *Orchestrierung* von Diensten. Unter dem Aspekt der Wiederverwendbarkeit können die angebotenen Dienste zu größeren Diensten, z.B. den Geschäftsprozessen eines Unternehmens, zusammengesetzt (orchestriert) werden. Geschäftsprozesse werden meist von externen *Ereignissen* angetrieben. Die Komponenten einer SOA können auf solche Ereignisse reagieren und bestimmte Aktionen auslösen.

2.2.2. Beteiligte und deren Aktionen

Innerhalb einer SOA agieren die folgenden drei Beteiligten:

- Dienstnutzer
- Dienstanbieter
- Verzeichnisdienst

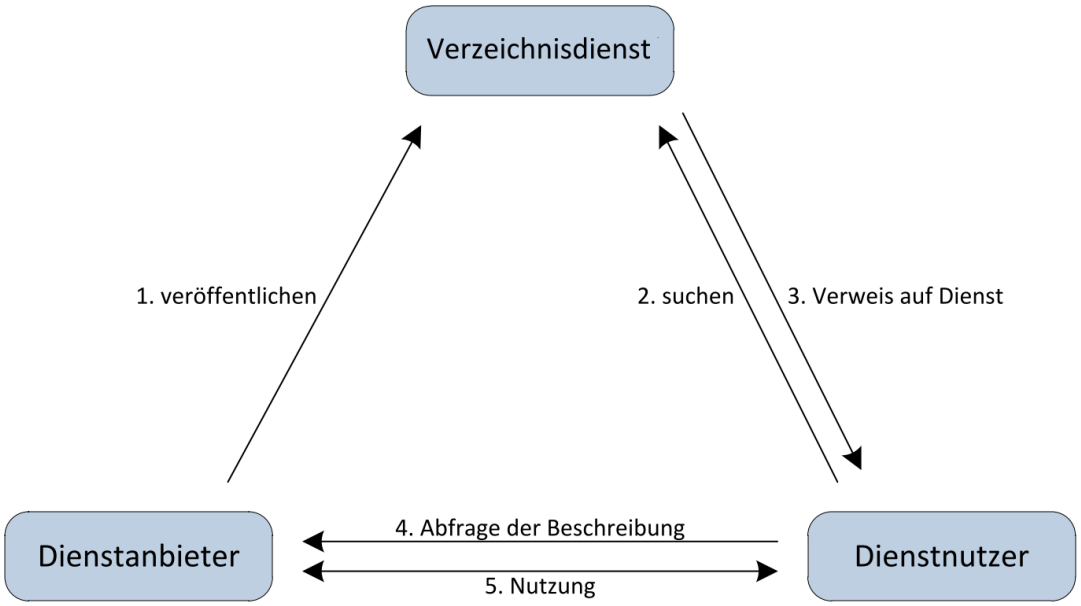

Abbildung 2.1.: Beziehungen innerhalb einer SOA (in Anlehnung an [Mel10])

Abbildung 2.1 zeigt das Zusammenspiel aller Beteiligten. Nach [Mel10] erstellt der Dienstanbieter eine *Service Description* für seine Anwendung bzw. Softwarekomponente. Diese Service Description liegt in maschinenlesbarer Form vor und beinhaltet, wie bereits erwähnt, die Beschreibung der eigentlichen Funktion sowie die Beschreibung der öffentlichen Schnittstellen. Oftmals werden auch noch nichtfunktionale Eigenschaften, wie z.B. maximale Antwortzeiten, festgehalten. Des Weiteren betreibt der Dienstanbieter die benötigte Infrastruktur, wie z.B. Rechenzentren, und übernimmt Aufgaben, die im Bereich der Quality of Service (wie z.B. Verfügbarkeit und Datenschutz) liegen. Der Dienstanbieter installiert (deployt) den Dienst innerhalb seiner Infrastruktur und registriert bzw. veröffentlicht diesen anschließend im Verzeichnisdienst. Der Dienstnutzer kann nun im Verzeichnisdienst nach einem passenden Dienst suchen. Ist dieser gefunden, wird die Schnittstellenbeschreibung des Dienstes vom Dienstanbieter abgefragt und mit diesem über Rahmenbedingungen, wie z.B. über Zertifikate, verhandelt. Anschließend kann der gewählte Dienst genutzt werden.

2.2.3. Web Service

Implementierungen einer SOA sind unter anderem *RESTful Services* oder *Web Services* [BHM+04]. Die folgende Definition für den Begriff Web Service stammt aus [BKNT10]:

> „Ein Web Service ist eine durch einen URI eindeutige identifizierte Softwareanwendung, deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Softwareagenten durch XML-basierte Nachrichten, die über Internetprotokolle ausgetauscht werden."

Grundlage bilden die drei Standards: SOAP, WSDL (Web Service Description Language) und UDDI (Universal Description Discovery & Integration). Die folgenden Abschnitte basieren auf [Mel10].

SOAP Der Begriff SOAP beschreibt ein XML-basiertes Nachrichtenformat, das insbesondere zur Kommunikation mit und zwischen Web Services eingesetzt wird. Es ist unabhängig von einem Transportprotokoll. 1999 wurde Version 1.0 veröffentlicht. Im Jahr 2003 wurde Version 1.2 als *Recommendation* veröffentlicht. Eine SOAP Nachricht ist ein XML-Dokument, das aus den folgenden drei Teilen besteht:

- SOAP Envelope
- SOAP Header
- SOAP Body

Der SOAP Envelope (Umschlag) dient als Container für die gesamte Nachricht. Er beinhaltet die beiden Teile Header und Body sowie Informationen über die verwendete SOAP Spezifikation.

Listing 2.6 Beispiel einer SOAP Nachricht [GHM+07]

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Der erste Teil einer SOAP Nachricht ist der Header. Er ist optional und der Inhalt wird durch die SOAP Spezifikation nicht genau definiert. Er kann zusätzliche Informationen für den Empfänger beinhalten, oder aber auch, da Nachrichten meist über Zwischenstationen

(Intermediäre) zum Empfänger gelangen, Informationen für Zwischenstationen beinhalten. Für SOAP Header Elemente können zusätzliche Attribute angegeben werden. Das Attribut *role* spezifiziert die Rolle, in der sich der Empfänger bzw. die Zwischenstation befinden muss, um das Header Element verarbeiten zu dürfen. Das Attribut *mustUnderstand* kann *true* oder *false* sein. Ist es *true*, muss die Station, die die Nachricht empfangen hat, das Header Element verarbeiten können. Wenn die Station die Nachricht nicht verarbeiten kann, wird sie nicht weitergeleitet/weiterverarbeitet. Listing 2.6 zeigt eine SOAP Nachricht. Der SOAP Header enthält Informationen über die Dringlichkeit sowie Gültigkeit der Nachricht. Der zweite Teil einer SOAP Nachricht ist der Body. Der Body ist verpflichtend und enthält die eigentlichen Nutzdaten (payload), die wiederum ein wohlgeformtes XML-Dokument bilden müssen. Einzig der Prolog ist an dieser Stelle nicht erlaubt.

WSDL WSDL, die *Web Service Description Language*, existiert in Version 1.1 seit 2001. 2007 veröffentlichte das W3C Version 2.0. Mithilfe von WSDL können Schnittstellen von Web Services beschrieben werden. Nach [Mel10] werden Web Services in WSDL aus zwei Blickwinkeln betrachtet:

> „abstrakt auf der Ebene der Funktionalität und konkret auf der Ebene der technischen Details. Hierzu wird die Beschreibung der Funktionalität, die ein Web Service anbietet, von den technischen Details über den Ort und die Art und Weise, wie ein Dienst angeboten wird, getrennt."

Listing 2.7 zeigt den prinzipiellen Aufbau einer WSDL Beschreibung in der Version 2.0. Eine WSDL Beschreibung beinhaltet im Wesentlichen die Komponenten *Types*, *Interface (WSDL 1.1 portType)*, *Binding* und *Service*.

Listing 2.7 Komponenten einer WSDL Beschreibung [CMRW07]

```
<description
     targetNamespace="xs:anyURI" >
  <documentation />*
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

Das Element Description bildet das Wurzelelement der WSDL Beschreibung und beinhaltet die gerade genannten Komponenten als Kindelemente. Mithilfe des Elementes Import können weitere WSDL Beschreibungen bzw. XML Schemas importiert werden. Das Element Documentation bietet die Möglichkeit beliebige Informationen festzuhalten. Die Schnittstellenbeschreibung erfolgt im Abschnitt Interface. Ein Interface bietet eine Menge an Operationen an. Für jede Operation wird festgelegt, welche Nachrichten zwischen Dienstanbieter und Dienstnutzer ausgetauscht werden müssen. Die Definition der Nachrichten erfolgt über die im Abschnitt Types festgelegten Datentypen. Die Abschnitte Types und Interface beschreiben einen Web Service auf abstrakte Weise. Konkreter wird es im Abschnitt Binding. Dort wird festgelegt, welche Protokolle für die Kommunikation verwendet werden können. Im letzten Abschnitt, dem Abschnitt Service, wird definiert, wie der Web Service physikalisch erreicht

werden kann, z.B. über eine URL. Weitere Informationen zu WSDL sind in [CMRW07] zu finden.

UDDI UDDI steht für *Universal Description Discovery & Integration*. Es handelt sich hierbei um einen Standard für den Aufbau eines Verzeichnisdiensts, der im Wesentlichen aus den White Pages, Yellow Pages und Green Pages besteht. Die White Pages enthalten Informationen über die einzelnen Dienstanbieter, die ihre Dienste im Verzeichnisdienst registriert haben und werden deshalb oftmals mit einem Telefonbuch verglichen. Eine Kategorisierung der einzelnen Dienste bilden die Yellow Pages. Aus diesem Grund spricht man auch von einem Branchenverzeichnis. Die Green Pages beinhalten die Schnittstellenbeschreibungen der einzelnen Web Services.

Die in Abbildung 2.1 gezeigte Rollenverteilung kann auf Web Services übertragen werden. Der Anbieter eines Dienstes erstellt eine entsprechende WSDL Beschreibung für seinen Dienst und registriert diesen im Verzeichnisdienst, der z.B. über UDDI aufgebaut wird. Der Dienstnutzer sucht nach einem passenden Dienst und fordert, wenn dieser gefunden wurde, die WSDL-Beschreibung an. Er erhält dazu ein URI, die auf die eigentliche WSDL-Beschreibung verweist, und kann damit den gewünschten Dienst einbinden.

2.3. Workflow

Ein Workflow, die deutsche Übersetzung lautet Arbeitsablauf, beschreibt eine Menge von Aktivitäten, die in einer bestimmten Reihenfolge ausgeführt werden müssen. Einzelne Komponenten werden unter dem Aspekt der Wiederverwendbarkeit zu größeren Komponenten zusammengesetzt. Der Vorteil des Komponenten-basierten Aufbaus ist die größere Flexibilität. Ändert sich ein definierter Ablauf, können die einzelnen Komponenten neu zusammengesetzt werden. Ziel ist es, einen definierten Workflow weitestgehend zu automatisieren. Bisher wurden Workflows meist im unternehmerischen Umfeld eingesetzt. Mittlerweile findet die Workflow-Technologie auch im wissenschaftlichen Umfeld Einsatz.

2.3.1. Grundlagen der Workflow-Technologie

Nach [LR99] beschreibt ein Prozessmodell die Struktur eines Prozesses bzw. Geschäftsprozesses in der realen Welt. Es beschreibt die einzelnen Schritte, die nötig sind, um einen Prozess abzuarbeiten. Aus diesem Modell können einzelne Instanzen gebildet werden, also konkrete Ausführungen des Prozessmodells. Nicht alle Teile eines Prozessmodells müssen computerunterstützt ausgeführt werden. Das Workflowmodell spezifiziert die Teile eines Prozessmodells, die auf einem Computer ausgeführt werden. Aus diesem Workflowmodell können wiederum Instanzen gebildet werden. In der Bankenbranche gibt es z.B. Prozessmodelle zur Überprüfung der Bonität eines Kunden. Soll die Bonität eines bestimmten Kunden überprüft werden, wird von diesem Prozessmodell eine Instanz gebildet. Werden alle Teile eines Prozessmodells auf einem Computer ausgeführt, werden die Begriffe Prozessmodell und

Workflowmodell oftmals synonym verwendet. Abbildung 2.2 stellt die Beziehung zwischen den einzelnen Begriffen grafisch dar.

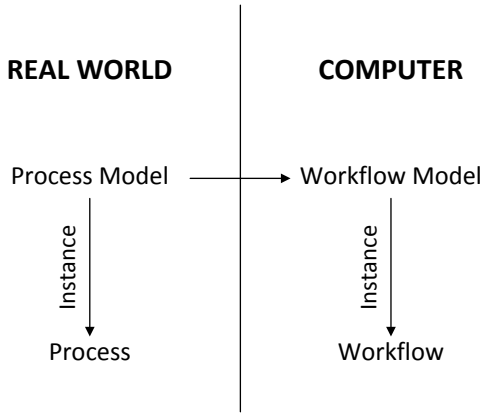

Abbildung 2.2.: Prozess und Workflow [LR99]

Prozesse werden nach [LR99] mithilfe der folgenden drei Dimensionen beschrieben: **WHAT**, **WHO** und **WITH**. Die **WHAT**-Dimension beschreibt die Aktivitäten, die ausgeführt werden müssen. Mithilfe der **WHO**-Dimension wird beschrieben, welcher Bereich einer Organisation (Abteilung, konkrete Person) die anfallende Aufgabe übernimmt. Die benötigten IT-Ressourcen, wie z.B. die benötigten Programme, werden mit der **WITH**-Dimension beschrieben.

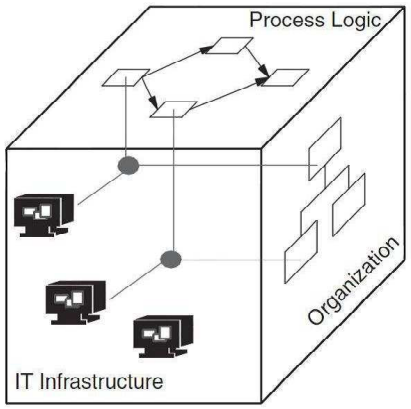

Abbildung 2.3.: Die drei Workflow-Dimensionen [LR99]

Abbildung 2.3 stellt die verschiedenen Dimensionen in einem dreidimensionalen Raum dar. Dabei wird die Ausführung eines Workflows in diesem Raum als eine Folge von Punkten intepretiert. Der Treffpunkt der drei Dimensionen besagt, welche Aktivität von welcher Person bzw. von welchem Program mit welchen Ressourcen erledigt wird [LR99].

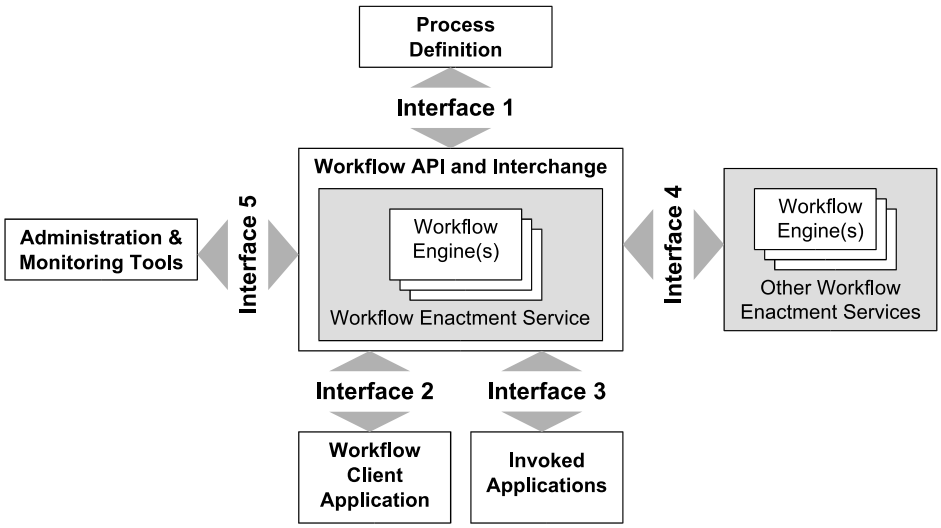

Abbildung 2.4.: WfMC-Workflow-Referenz-Modell [Mel10]

Das in Abbildung 2.4 dargestellte Worklow-Referenzmodell wurde von der Workflow Management Coalition (WfMC) entwickelt. Die WfMC wurde 1993 ins Leben gerufen und verfolgt das Ziel größtmögliche Systemunabhängigkeit und Interoperabilität zu schaffen. Die folgenden Abschnitte basieren auf [Mel10] und betrachten die einzelnen Teile des Referenzmodells genauer.

Process Definition Mit dieser Komponente werden die einzelnen Prozesse modelliert. Zur einfacheren Modellierung können grafische Anwendungsprogramme in Anspruch genommen werden.

Workflow Engine Die fertige Prozessdefinition wird von der Workflow Engine (Execution Engine) ausgeführt. Apache ODE [Foua] ist eine solche Engine und wird vom SIMPL-Rahmenwerk verwendet.

Workflow Client Application Die Workflow Client Application stellt dem Nutzer Funktionen zur Interaktion mit dem Workflow Management System bereit.

Invoked Applications Die Komponente Invoked Applications fasst alle Anwendungen bzw. deren Funktionen zusammen, die vom definierten Workflow eingebunden werden. Dadurch soll eine Unabhängigkeit vom Zielsystem realisiert werden.

Other Worflow Enactment Services Mithilfe dieser Komponente kann mit anderen Workflow Systemen kommuniziert werden. Somit können bereits vorhandene Prozesse in den eigenen Workflow integriert werden.

Administration & Monitoring Tool Diese Komponente überwacht und protokolliert die Ausführung der einzelnen Instanzen.

2.3.2. Workflow-Sprachen

Um einen Workflow zu definierien, wird eine Workflow-Sprache bzw. Modellierungssprache benötigt. Dieser Abschnitt basiert auf [RSM11]. Meist wird zwischen **datenflussorientierten** und **kontrollflussorientierten** Sprachen unterschieden.

Datenflussorientierte Sprachen beschreiben den Datenfluss also Datenabhängigkeiten zwischen verschiedenen Aktivitäten. Der Fokus liegt demzufolge auf der Datenverarbeitung. Jede Aktivität verfügt über eine *Input Queue*. Sobald die benötigten Daten bzw. Datensätze in dieser Input Queue vorliegen, wird die entsprechende Aktivität auf diesen Datensätzen ausgeführt. Die resultierenden Daten werden anschließend, gemäß des definierten Datenflusses, an eine oder mehrere andere Aktivitäten weitergeleitet. Dort werden die einzelnen Datensätze dann wiederum bearbeitet und entsprechend weitergeleitet. Demzufolge eignen sich datenflussorientierte Sprachen besonders für datenintensive Workflows (vgl. Abschnitt 2.3.3).

Bei **kontrollflussorientierten** Sprachen hingegen definiert der Kontrollfluss in der Regel einen gerichteten, azyklischen Graphen. Dabei entsprechen die Knoten den Aktivitäten und die Kanten beschreiben die kausalen Abhängigkeiten. Jede Aktivität wird höchstens einmal ausgeführt und nur dann, wenn alle vorherigen Aktivitäten erfolgreich ausgeführt wurden. Teilweise bieten kontrollflussorientierte Sprachen die Möglichkeit prozedurale Aktivitäten zu modellieren. Solch eine Aktivität definiert einen Teilgraphen, der wiederum verschiedene Aktivitäten beinhaltet und dessen Ausführung festgelegten Bedingungen unterliegt. Z.B. könnten bei einer Schleife die inneren Aktivitäten solange ausgeführt werden, wie eine festgelegte Bedingung wahr ist. In Abschnitt 2.3.4 wird auf die kontrollflussorientierte Sprache WS-BPEL eingegangen.

2.3.3. Arten von Workflows

Die folgenden Abschnitte und Teilkapitel basieren, soweit nicht anders angegeben, auf [RSM11] und [Wag11]. Abbildung 2.5 klassifiziert Workflows hinsichtlich ihrer Datenintensität sowie der Probleme, die sie lösen. *Orchestration Workflows* verknüpfen unterschiedliche und heterogene Anwendungen, um z.B. Geschäftsprozesse zu realisieren bzw. zu automatisieren [LR99]. Solche Workflows verarbeiten meist kleine Datenmengen. *Datenintensive Workflows* hingegen müssen in erster Linie große Datenmengen, die verteilt vorliegen können, verarbeiten. Die Verarbeitung erfolgt dabei entweder innerhalb der externen Ressourcen oder innerhalb des Workflow Kontexts. Erfolgt die Verarbeitung innerhalb des Workflows, müssen die Daten entsprechend geladen und innerhalb des Workflows bereitgestellt werden. Zu den Orchestration Workflows gehören *Business Workflows* sowie *Simulation Management Workflows*. Die Klasse der datenintensiven Workflows umfasst *Data Analysis Workflows*, *Data Modeling Workflows* und *ETL Workflows*.

Des Weiteren bilden *Simulation Management Workflows*, *Data Analysis Workflows* sowie *Data Modeling Workflows* eine weitere Gruppe, die Gruppe der *Scientific Workflows*. Im Folgen-

den werden nun die drei Kategorien **Business Workflows**, **Scientific Workflows** und **ETL Workflows** genauer betrachtet.

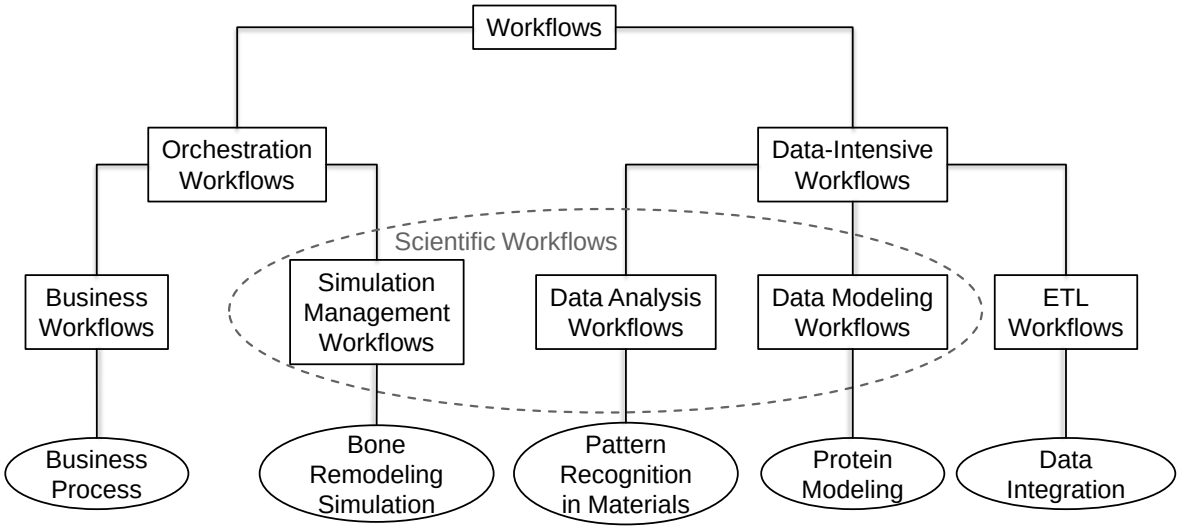

Abbildung 2.5.: Klassifizierung von Workflows [RSM11]

2.3.3.1. Business Workflows

Business Workflows modellieren in der Regel wiederkehrende Arbeitsabläufe innerhalb eines Unternehmens und automatisieren diese somit [LR99]. Das folgende Beispiel basiert auf [LR99]. Möchte z.B. ein Kunde einen Kredit bei einer Bank aufnehmen, könnte der Ablauf in etwa wie folgt aussehen. Wenn der Kreditantrag eingegangen ist, werden zuerst alle Daten (Höhe des Kredits, Kontonummer des Antragstellers, etc.) erfasst. Als nächstes wird die Bonität des Kunden geprüft. Verfügt der Kunde über eine gute Bonität, kann der Kredit gewährt werden. Sollte der Kunde über eine schlechte Bonität verfügen, muss ein Angestellter entscheiden, ob der Kredit dennoch gewährt werden soll.

Entscheidungen und die daraus resultierenden Handlungen spielen die zentrale Rolle bei solchen Workflows. In den meisten Fällen müssen nur geringe Datenmengen verarbeitet werden. Dementsprechend sind kontrollflussorientierte Sprachen für solche Workflows besser geeignet.

2.3.3.2. Scientific Workflows

Mittlerweile wird die Workflow-Technologie auch in wissenschaftlichen Bereichen eingesetzt. In den letzten Jahren hat sich dafür der Begriff **Scientific Workflow** etabliert [TDGS06]. [RSM11] unterscheidet drei Arten von Scientific Workflows. *Simulation Management Workflows* koordinieren die Kommunikation mit Simulationsprogrammen und Ressourcen, um einen

Simulationsablauf durchzuführen [GSK+11]. *Data Analysis Workflows* hingegen versuchen bereits generierte Daten zu visualisieren und/oder neue Erkenntnisse aus diesen Daten zu gewinnen. *Data Modeling Workflows* versuchen ein passendes Modell, das ein bestimmtes Problem beschreibt, zu finden bzw. versuchen bestehende Modelle zu vereinfachen oder bestimmte Muster in solchen Modellen zu finden.

Vereinfacht ausgedrückt führen Scientific Workflows wissenschaftliche Experimente oder Simulationen rechnergestützt aus bzw. werten Datensätze aus. In den meisten Fällen gibt es keine standardisierte Software, die solche Experimente oder Simulationen unterstützt. Dies führt dazu, dass für Berechnungen und Analysen verschiedene Programme eingesetzt werden müssen. Die Workflow-Technologie bietet die Möglichkeit solche Abläufe zu modellieren. Innerhalb solcher Workflows werden Daten bereitgestellt und evtl. externen Programmen übergeben. Diese Programme führen dann Berechnungen auf den bereitgestellten Datensätzen aus und werden meist als Web Service angesprochen. Das Scientific Workflow Management System ruft diese Web Services entsprechend auf. Der Fokus liegt dabei auf dem Datenaustausch zwischen den verschiedenen Aktivitäten, Programmen und Datenressourcen. Demzufolge werden in der Praxis meist datenflussorientierte Workflow-Sprachen für die Modellierung solcher Workflows eingesetzt. Nichtsdestotrotz wird zunehmend WS-BPEL (vgl. 2.3.4), eine kontrollflussorientierte Workflow-Sprache, verwendet. Dies liegt zum einen an der weiten Verbreitung dieser Sprache (de-facto Standard) und zum anderen lässt sich der Datenfluss auch innerhalb eines Kontrollflusses darstellen.

Nach [GSK+11] bietet der Einsatz von Scientific Workflows die folgenden Vorteile:

- Die Wissensweitergabe kann durch Workflows unterstützt werden, da diese als Services bereitstehen.
- Resultate können durch die Community bewertet und analysiert werden.
- Workflows können mit großen Datenmengen umgehen.
- Eine Ausführung in verteilten und heterogenen Umgebungen ist möglich (Unterstützung verschiedener Plattformen und Programmiersprachen).
- Durch die Automatisierung der Schritte zwischen der Design- und Ausführungsphase können sich Wissenschaftler auf ihre eigentliche Problemstellung konzentrieren.
- Wissenschaftliche Simulationen können parallelisiert und automatisiert ausgeführt werden.

2.3.3.3. ETL Workflows

Mithilfe von Extract Transfer and Load (ETL) Workflows werden ETL Operationen orchestriert und ausgeführt. Ziel ist es, Daten für andere Programme oder Anwendungen zu extrahieren bzw. bereitzustellen (z.B. um ein Data Warehouse mit aktuellen Daten zu versorgen). Meist bieten solche Workflows Operationen, um Daten zu laden (LOAD/RETRIEVAL) und zu filtern, sowie zwei Datenmengen zu verknüpfen (JOIN), zu vereinigen (UNION) und zusammenzuführen (MERGE).

Das SIMPL-Rahmenwerk bietet die Möglichkeit, aus einem Simulationsworkflow heraus, auf beliebige externe Datenquellen zuzugreifen und ETL Operationen auszuführen [RRS+11]. Eine genaue Betrachtung des SIMPL-Rahmenwerks erfolgt in Kapitel 3.

2.3.4. WS-BPEL

Die WS-Business Process Execution Language (WS-BPEL, ehemals BPEL4WS) [JE07] ist eine XML-basierte Sprache zur Beschreibung von Workflows. Mittlerweile existiert Version 2.0. Der Kerngedanke ist, dass Web Services in einer bestimmten Reihenfolge aufgerufen werden. WS-BPEL ist eine kontrollflussorientierte Sprache und nutzt die XML-basierten Spezifikationen WSDL 1.1, XML Schema 1.0, XPath 1.0, XSLT 1.0 und Infoset. WS-BPEL übernimmt die Grundkonzepte von WSDL. [JE07] beschreibt WS-BPEL folgendermaßen:

> „The definition of a WS-BPEL business process follows the WSDL model of separation between the abstract message contents used by the business process and deployment information (messages and port type versus binding and address information). In particular, a WS-BPEL process represents all partners and interactions with these partners in terms of abstract WSDL interfaces (port types and operations); no references are made to the actual services used by a process instance. WS-BPEL does not make any assumptions about the WSDL binding. Constraints, ambiguities, provided or missing capabilities of WSDL bindings are out of scope of this specification."

Listing 2.8 Erste Ebene eines WS-BPEL Dokuments (nach [Mel10])

```
<process name="ProcessName">
  <partnerLinks>..</partnerLinks>
  <variables>..</variables>
  <correlationSets>..</correlationSets>
  <faultHandlers>..</faultHandlers>
  <compensationHandlers>..</compensationHandlers>
  <eventHandlers>..</eventHandlers>
  <!-- genau eine activity -->
</process>
```

Listing 2.8 zeigt die erste Ebene eines WS-BPEL Dokuments. Einen genaueren Überblick über die Struktur der Sprache gibt Anhang A.1. Die folgende Beschreibung basiert auf [Mel10]. Wurzelelement ist immer das Element *Process*. Ein Prozess ruft während seiner Ausführung externe Dienste (Web Services) auf. Das Element *PartnerLinks* definiert, wie ein Prozess mit seinen Partnern kommuniziert. Mithilfe des Elementes *Variables* können Variablen definiert werden, die Daten innerhalb des Workflows halten. Es stehen die Datentypen WSDL Message Type, XML Schema Type und XML Schema Element zur Verfügung. *CorrelationSets* enthalten verschiedene Eigenschaften, die von Nachrichten einer korrelierten Gruppe geteilt werden. Mithilfe der Ausprägungen dieser Eigenschaften kann bestimmt werden, welcher Partner oder welche Prozessinstanz eine Nachricht bekommt. Somit kann sicher gestellt werden, dass eine Nachricht auch zum richtigen Empfänger kommt. *FaultHandlers* bieten die

Möglichkeit auf geworfene *Exceptions*, also Fehler während der Ausführung, zu reagieren. Bereits vollständig ausgeführte Aktivitäten können durch *CompensationHandlers* rückgängig gemacht bzw. kompensiert werden. Mithilfe von *EventHandlers* kann auf Ereignisse bzw. Alarme reagiert werden. Jeder Prozess muss genau eine Aktivität, die wiederum andere Aktivitäten beinhalten kann, beinhalten. Die zur Verfügung stehenden Aktivitäten können in zwei Kategorien unterteilt werden:

- Basisaktivitäten
- Strukturierte Aktivitäten

Mithilfe der strukturierten Aktivitäten kann die Ausführungsreihenfolge der Basisaktivitäten sowie anderer strukturierter Aktivitäten festgelegt werden. Im Folgenden werden die einzelnen Aktivitäten kurz erläutert. Grundlage bilden die Quellen [JE07] und [Mel10]. Für syntaktische Details wird auf diese beiden Quellen verwiesen.

Zu den Basisaktivitäten gehören: *receive*, *reply*, *invoke*, *assign*, *throw*, *exit*, *wait*, *empty*, *compensate*, *compensateScope*, *rethrow*, *validate* und *extensionActivity*. Eine *receive* Aktivität erlaubt es einem Prozess auf das Eintreffen einer Nachricht zu warten. Trifft eine Nachricht ein, so wird der Prozess gestartet bzw. weitergeschaltet. Durch die *reply* Aktivität kann der Prozess eine Nachricht an einen aufrufenden Partner zurückgeben (Listing 2.9).

Listing 2.9 Beispiel einer receive sowie reply Aktivität [Mel10]

```
<receive name="BerechnungReceive"
  partner="caller" portType="tns:BerechnungPT" operation="berechne"
  variable="request" createInstance="yes">
</receive>

<reply name="BerechnungReply"
  partner="caller" portType="tns:BerechnungPT" operation="berechne"
  variable="response">
</reply>
```

Listing 2.10 Beispiel einer synchronen invoke Aktivität [Mel10]

```
<invoke name="ErsterBerechnungsSchritt"
  partner="meinAddierer" portType="addiererNS:Addierer"
  operation="addiere"
  inputVariable="ersteSummanden" outputVariable="erstesErgebnis">
</invoke>
```

Durch eine *invoke* Aktivität können externe Web Services aufgerufen werden. Invoke kann sowohl für synchrone (request/response) als auch für asynchrone Aufrufe verwendet werden. Synchrone Aufrufe benötigen sowohl eine Input- als auch eine Output-Variable (Listing 2.10), asynchrone Aufrufe hingegen nur eine Input-Variable. Eine *assign* Aktivität weist einer Variablen einen Wert zu bzw. kann eine Menge solcher Variablenzuweisungen beinhalten. Ein Fehler kann durch eine *throw* Aktivität ausgelöst werden. Existiert für diesen Fehler

ein „FaultHandler“, wird eine Fehlerbehandlung gestartet. Dazu ist es notwendig einen eindeutigen Bezeichner für den Fehler anzugeben. Durch eine *exit* Aktivität kann die Ausführung eines Prozesses vollständig beendet werden. Die *wait* Aktivität hält einen Prozess für eine Weile an. Durch das Attribut „for“ wird eine entsprechende Zeitspanne festgelegt. Das Attribut „until“ ermöglicht es auf das Eintreten eines bestimmten zeitbasierten Ereignisses zu warten. Die *empty* Aktivität entspricht der leeren Anweisung und kann z.B. als Platzhalter für noch nicht definierte Aktivitäten genutzt werden. Die *compensate* Aktivität setzt fertiggestellte Aktivitäten zurück bzw. kompensiert diese. Dabei werden alle inneren *scopes* berücksichtigt. Eine *compensate* Aktivität darf nur in einem „FaultHandler“ oder „CompensationHandler“ stehen. *CompensateScope* ermöglicht es genau einen *scope* zu kompensieren. Durch *rethrow* kann eine ausgelöste und behandlelte Ausnahme erneut ausgelöst und somit weitergeleitet werden. Die Aktivität *validate* überprüft, ob Variablenwerte ihrer Definition genügen. Mithilfe des Elementes *extensionActivity* können neue Aktivitäten erzeugt werden.

Die strukturierten Aktivitäten umfassen: *sequence*, *if*, *while*, *repeatUntil*, *forEach*, *pick*, *flow* und *scope*. Aktivitäten innerhalb einer *sequence* Aktivität werden nacheinander in ihrer lexikalischen Anordnung ausgeführt. Die *switch* Aktivität wurde in WS-BPEL Version 2.0 durch eine *if* Aktivität ersetzt. Diese ermöglicht es, Bedingungen zu definieren, nach denen ein bestimmter Pfad im Workflow durchlaufen wird oder nicht. Durch eine *while* Aktivität werden innere Aktivitäten solange ausgeführt, wie eine definierte Bedingung erfüllt ist. Bei der *repeatUntil* Aktivität wird die definierte Bedingung im Gegensatz zur *while* Aktivität erst nach der Ausführung des ersten Schleifendurchlaufs geprüft. Die innere Aktivität wird somit mindestens einmal ausgeführt. Bei der *ForEach* Aktivität wird solange iteriert, bis eine bestimmte Anzahl an Schleifendurchläufen zu Ende ist. Die einzelnen Schleifendurchläufe können entweder parallel oder sequentiell ausgeführt werden. Wenn interne oder externe Ereignisse auftreten, bietet *pick* die Möglichkeit genau eine geeignete Aktivität von einer Menge von Aktivitäten zu starten. Eine *flow* Aktivität ermöglicht es innere Aktivitäten parallel oder nach einem bestimmten Ablaufgraphen auszuführen. Durch das Element *scope* können Gültigkeitsbereiche für z.B. Variablen und „PartnerLinks“ festgelegt werden.

2.3.5. Architektur eines Scientific Workflow Management Systems

In Abschnitt 2.3.1 wird bereits auf das von der Workflow Management Coalition (WfMC) entwickelte Workflow-Referenzmodell eingegangen. Für Scientific Workflows existiert eine weiterentwickelte Architektur.

Die in Abbildung 2.6 dargestellte Architektur basiert auf der in [LR99] definierten Technologie für Geschäfts- und Produktionsworkflows und unterscheidet zwischen GUI Komponenten und Laufzeitkomponenten. Im Folgenden werden nun die einzelnen Komponenten erläutert.

GUI Komponenten:

- Der **Function Catalog** enthält eine Liste der verfügbaren Dienste oder vorgefertigten Workflow-Fragmente, die innerhalb eines Workflows genutzt werden können.

- Der **sWF Modeler** (Scientific Workflow Modeler) unterstützt den Modellierer dabei Workflow-Spezifikationen sowie Deployment-Informationen zu erstellen.
- Die **Monitor** Komponente bietet die Möglichkeit die Ausführung von Workflows zu überwachen.
- Die **Result Display** Komponente stellt dem Nutzer Zwischen- und Endergebnisse der Simulationsberechnungen zur Verfügung.

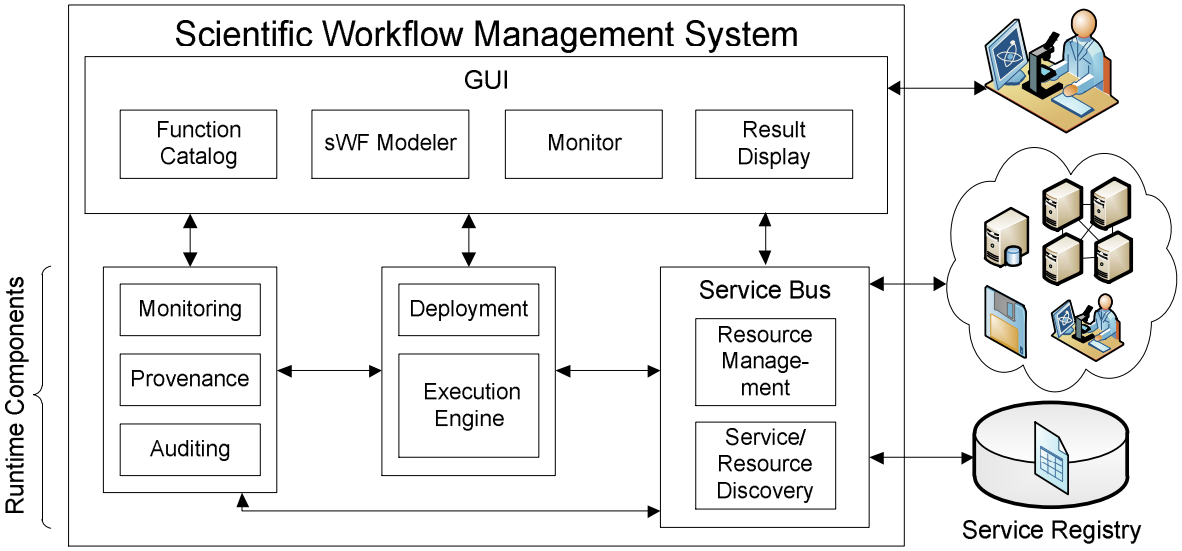

Abbildung 2.6.: Architektur eines sWfMS [RRS+11] und [GSK+11]

Laufzeitkomponenten:

- Die **Deployment** Komponente transformiert ein Workflowmodell in eine für die **Execution Engine** verständliche Form, die dann Instanzen davon ausführt.
- Die **Auditing** Komponente zeichnet Ereignisse, die zur Laufzeit auftreten, auf.
- Die **Monitoring** Komponente nutzt die Ergebnisse der Auditing Komponente, um den Zustand eines laufenden Workflows anzuzeigen.
- Die **Provenance** Komponente erfasst weitere Daten, um detailliertere Informationen über die Ausführung eines Workflows zur Verfügung zu stellen und damit die Nachvollziehbarkeit und Wiederholbarkeit solcher Ausführungen sicherzustellen.
- Die **Service Bus** Komponente sucht und wählt passende Dienste aus, die die Workflow Aktivitäten implementieren. Des Weiteren leitet sie Nachrichten weiter und transformiert Daten.
- Das **Resource Management** enthält Informationen über externe Ressourcen sowie Dienste.

- Die **Service/Resource Discovery** Komponente nutzt die im Resource Management hinterlegten Metadaten, um eine Liste mit zu den Nutzeranforderungen passenden Diensten bzw. Ressourcen zur Verfügung zu stellen.

2.4. Simulation

In Abschnitt 2.3.3.2 wird erläutert, dass mithilfe von Scientific Workflows unter anderem Simulationsabläufe koordiniert werden können. Im Folgenden wird nun der Begiff **Simulation** konkretisiert. Des Weiteren werden zwei Simulationsrahmenwerke sowie ein numerisches Verfahren zur Lösung von partiellen Differentialgleichungen vorgestellt.

2.4.1. Eigenschaften einer Simulation

Dieser Abschnitt basiert auf [Dor11]. Weitere Quellen werden explizit angegeben. Problematisch ist, dass für den Begriff **Simulation** keine einheitliche Definition existiert. Manche Definitionen bzw. Fachbereiche berücksichtigen Aspekte, die bei anderen Definitionen bzw. Fachbereichen vollständig vernachlässigt werden. Im Rahmen dieser Diplomarbeit werden nun zwei Definitionen, die den Begriff Simulation sehr abstrakt definieren, vorgestellt. Die erste Definition stammt vom Institute of Electrical and Electronic Engineers [IEE90]:

> „(1) A model that behaves or operates like a given system when provided a set of controlled inputs. See also: **emulation**
> (2) The process of developing or using a model as in (1).“

Die Definition von [Har96] ist konkreter und geht insbesondere auf den Zusammenhang zwischen *Simulation* und *Simulationsmodell* (Modell) ein:

> „Simulations are closely related to dynamic models. More concretely, a simulation results when the equations of the underlying dynamic model are solved. This model is designed to imitate the time-evolution of a real-system. To put it another way, a *simulation imitates one process by another process*. In this definition, the term „process“ refers solely to some object or system whose state changes in time. If the simulation is run on a computer, it is called a computer simulation.“

Eine Simulation versucht das Verhalten von einem System bzw. einem Prozess nachzuahmen. Bevor eine Simulation gestartet werden kann, muss ein valides Modell erzeugt werden. Dieses Modell ist ein beschränktes Abbild der Realität bzw. eines anderen Modells. Um neue Erkenntnisse zu gewinnen, wird dieses Modell parametrisiert und anschließend ein Simulationslauf gestartet. Nach [Sta73] sind die folgenden drei Merkmale Kennzeichen eines Modells:

- **Abbildung** Ein Modell ist ein Abbild bzw. eine Repräsentaion eines natürlichen oder künstlichen Originals. Dabei kann das Original selbst wiederum ein Modell sein.

- **Verkürzung** Ein Modell erfasst *nicht* alle Eigenschaften bzw. Attribute des Originals. Es erfasst vielmehr nur die Eigenschaftenn bzw. Attribute, die dem Modellierer als relevant erscheinen.
- **Pragmatismus** Ein Modell ist seinem Original nicht per se eindeutig zugeordnet. Es muss stets hinsichtlich der Fragen für *wen*, *wann* und *wozu* relativiert werden. Dies bedeutet, dass ein Modell nicht nur ein Modell von etwas ist, sondern das ein Modell auch für ein konkretes Subjekt (z.B. Person) entwickelt wird und zumindest innerhalb eines bestimmten Zeitraums sinnvoll ist.

Des Weiteren können Simulationen hinsichtlich der Aspekte, die sie berücksichtigen, klassifiziert werden. **Dynamische** Simulationen berücksichtigen den Faktor Zeit. Bei **statischen** Simulationen hingegen wird dieser Faktor vernachlässigt. Auch das Auftreten von zufälligen Ereignissen kann eine Rolle spielen. **Stochastische** Simulationen berücksichtigen den Zufall im Gegensatz zu **deterministischen** Simulationen.

Mithilfe von Simulationen können verschiedenste Aufgaben erledigt werden. [Har96] nennt die folgenden fünf Funktionsbereiche:

- **Simulationen als eine Technik** Ein wesentlicher Vorteil von Simulationen ist, dass die Dynamik eines realen Prozesses detailliert untersucht werden kann. Oftmals ist es nicht möglich, diese Daten experimentell zu extrahieren, da die relevante Zeitskala zu groß (z.B. die Ausbreitung des Universums) oder zu klein (z.B. nukleare Reaktionen) ist. In diesen Fällen bieten Simulationen die einzige Möglichkeit Datensätze zu generieren.
- **Simulationen als heuristisches Werkzeug** Auch bei der Entwicklung von neuen Hypothesen, Modellen oder Theorien spielen Simulationen eine wichtige Rolle. In der Regel werden zahlreiche Simulationsläufe mit unterschiedlichen Parameterwerten ausgeführt. Aus den Ergebnisdaten können dann unter Umständen neue und einfachere Regeln abgeleitet werden, die aus den ursprünglichen Modellannahmen nicht hätten extrahiert werden können. Diese Regeln bilden dann die Grundlage neuer und eventuell einfacherer Modelle.
- **Simulationen als Ersatz für Experimente** Oftmals ist es nicht bzw. noch nicht möglich eine gegebene Situation experimentell zu untersuchen. Es können z.B. pragmatische, theoretische oder ethische Probleme dem gegenüberstehen. So kann die Entstehung von Galaxien aus pragmatischen Gründen nicht untersucht werden. Ein theoretisch unmögliches Experiment wäre es, fundamentale Annahmen, wie z.B. die Ladung eines Elektrons, zu ändern. Aus ethischen Gründen ist es z.B. nicht möglich, die Auswirkungen einer Einkommensteuerhöhung zu untersuchen. In solchen Fällen bieten Simulationen die Möglichkeit die Situation trotzdem zu untersuchen.
- **Simulationen als Werkzeug für Experimentatoren** Des Weiteren können Simulationen die Grundlage neuer Experimente bilden oder den Wissenschaftler bei Planung und Analyse von Experimenten unterstützen. So kann z.B. eine Simulation neue Hypothesen liefern, die anschließend experimentell überprüft werden. Des Weiteren können unterschiedliche Versuchsaufbauten simuliert werden. Anschließend wird dann der erfolgversprechendste Versuchsaufbau ausgewählt und in der Realität umgesetzt. Auch

können Simulationen bei der Analyse von Ergebnissen helfen. So können z.B. einfache bzw. triviale Zusammenhänge aus den Daten ausgeblendet werden.

- **Simulationen als pädagogisches Werkzeug** Weiterhin können Simulationen das Lehren bzw. Lernen unterstützen. So können z.B. unterschiedliche Simulationsläufe gestartet und deren Ergebnisse visualisiert werden. Im besten Fall erhält der Lernende dadurch ein Verständnis für das darunterliegende reale Problem.

2.4.2. Multi-* Simulationen

Dieser Abschnitt basiert auf [RSRM]. Werden reale Objekte oder Phänomene simuliert, müssen oftmals Simulationen aus unterschiedlichen wissenschaftlichen Bereichen miteinander verknüpft werden. Daraus resultiert, dass sowohl die verschiedenen Simulationsmodelle als auch deren Annahmen kombiniert werden müssen. Zu den wichtigsten Angaben eines Simulationsmodells gehören die Physik sowie die Zeit- und/oder Raumskalen, die verwendet werden. Die Physik definiert die physikalischen Gesetze auf denen das Modell aufbaut. Oftmals werden diese durch Differentialgleichungen repräsentiert. Solch eine Physik beschreibt ein Modell in mehreren und in der Regel kontinuierlichen Dimensionen (z.B. gibt es ein bis drei räumliche Kontinua sowie eine Dimension für die Zeit). Problematisch ist, dass Computersysteme mit solchen kontinuierlichen Informationen nicht umgehen können. Aus diesem Grund muss das kontinuierliche Verhalten diskretisiert werden. So werden Differentialgleichungen meist in lineare Gleichungssysteme umgewandelt. Die kontinuierlichen Informationen werden durch mehrere Schritte in den beteiligten Dimensionen dargestellt, wobei die Schritte eine festgelegte Größe bzw. Granularität aufweisen. Die einzelnen Skalen definieren dabei die Granularität der einzelnen Dimensionen. Die Dimension der Zeit wird z.B. in mehrere Zeitschritte zerlegt, wobei eine Schrittgröße zwischen Nanosekunden und Jahren liegen kann.

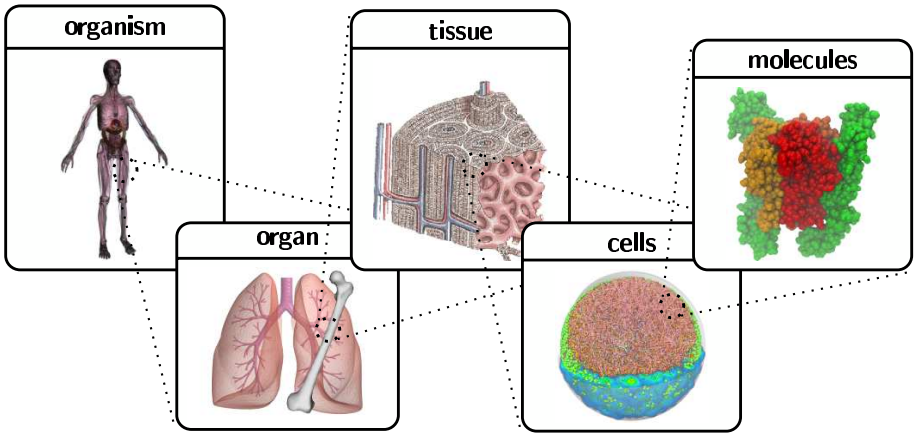

Abbildung 2.7.: Multi-Domänen, Multi-Physiken, Multi-Skalen Simulation des menschlichen Körpers [KME12]

Simulationen, die verschiedene Domänen, Physiken oder Skalen kombinieren, werden Multi-Domänen, Multi-Physiken und Multi-Skalen Simulationen genannt. Abbildung 2.7 zeigt ein umfassendes Modell des menschlichen Körpers. Dieses Modell umfasst mehrere Domänen (die medizinische Wissenschaft, die Biologie, etc.) und die Skalen reichen dabei von der Beschreibung des gesamten Organismuses bis hin zur Molekular-Ebene. In Abschnitt 4.3 wird noch auf ein Beispiel einer Multi-Skalen Simulation eingegangen.

2.4.3. Simulationsrahmenwerke

Ein Simulationsrahmenwerk ermöglicht die Durchführung von Simulationen. Im Folgenden werden die Rahmenwerk **Pandas** und **ChemShell** vorgestellt.

2.4.3.1. Pandas

Dieser Abschnitt nutzt [FS] als Basis. Pandas (Porous Media Adaptive Nonlinear finite element solver based on Differential Algebraic Systems) ist ein Simulationsrahmenwerk zur Lösung von Problemen, der porösen Medien. Solche Probleme treten in den verschiedensten Engineering-Bereichen (z.B. Boden- und Felsmechanik, Ton- und Schiefermechanik, etc.) auf. Als numerisches Verfahren wird die in Abschnitt 2.4.4 erläuterte Finite Element Methode eingesetzt. Als Beispiel umfassen solche Modelle das interaktive Verhalten eines verformten Skeletts. Des Weiteren können chemische und elektrochemische Phänomene auftreten und vom Pandas Rahmenwerk berücksichtigt werden. Dem Wissenschaftler steht sowohl ein interaktiver als auch ein Batch-Modus zur Verfügung. Auch gibt es die Möglichkeit Ergebnisse online zu visualisieren.

2.4.3.2. ChemShell

Dieser Abschnitt basiert auf [Mü10] und [Rut09]. Mithilfe von Chemshell können chemische Berechnungen im Bereich der Quantenmechanik (QM), der Molekularmechanik (MM) sowie der hybriden Quantenmechanik/Molekularmechanik (QM/MM) durchgeführt werden. Dabei liegt der Fokus auf **hybriden bzw. multi-skalaren** Simulationen, die beide Betrachtungsweisen vereinen.

Komplexe Berechnungen werden von ChemShell nicht selbst ausgeführt. Vielmehr agiert ChemShell als Koordinator und übergibt aufwendige Analysen externen Werkzeugen. ChemShell verwaltet die Kommunikation zwischen den verschiedenen externen Werkzeugen, stellt benötigte Dateien bereit und verfügt über integrierte Routinen zur Dateitransformation.

ChemShell basiert auf der Skriptsprache TCL. Dem Wissenschaftler steht eine interaktive Shell zur Verfügung. Mithilfe dieser Shell kann der Wissenschaftler Befehle eingeben, die anschließend *sofort* interpretiert und ausgeführt werden. Auf diese Weise wird eine Simulation Schritt für Schritt ausgeführt. Eine weitere Möglichkeit ist die Erstellung eines Scripts.

Dieses Script beinhaltet Befehle, die automatisch der Reihe nach abgearbeitet werden. Da bei ChemShell lediglich vordefinierte Befehle ausgeführt werden, muss ChemShell nur einmal kompiliert werden. Nur wenn neue Befehle oder externe Module hinzugefügt werden sollten, ist eine erneute Kompilation erforderlich.

2.4.4. Finite Element Methode

Dieser Abschnitt basiert auf [Ari12], [Wag11] und [ZTZ05]. Im Bereich des Engineerings basieren viele Simulationsmodelle auf Differentialgleichungen. Die **Finite Element Methode** (FEM) ist ein numerisches Verfahren, mit dessen Hilfe partielle Differentialgleichungen gelöst werden können. Dabei fasst der Begriff FEM verschiedene Ansätze für unterschiedliche Problemstellungen zusammen. So kann z.B. der Ritz-Ansatz bei Körperverformungen und der Galerkin-Ansatz bei zeitabhängingen Problemen angewendet werden.

Im Folgenden wird nun der Galerkin-Ansatz grob erläutert. Für ein tiefergehendes Verständnis wird auf [ZTZ05] verwiesen. Beim Galerkin-Ansatz wird das eigentliche Problem, also die zu simulierende Welt, in eine endliche Anzahl von Teilproblemen zerlegt. Die einzelnen Teilprobleme werden Elemente genannt und haben meist eine einfache geometrische Form (Dreiecke oder Vierecke).

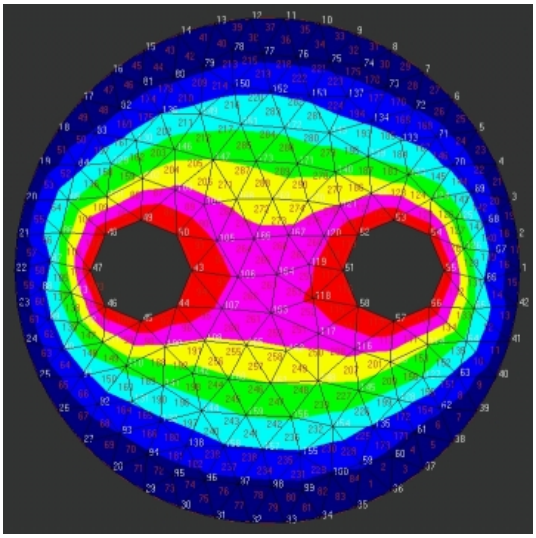

Abbildung 2.8.: FEM: Ein Problem wird in mehrere Teilprobleme zerlegt.[1]

Abbilung 2.8 zeigt exemplarisch solch eine Zerlegung. Als geometrische Form wurde das Dreieck gewählt. Um das Problem anzunähern, werden die einzelnen Elemente mithilfe der Kanten und unter Anwendung eines numerischen Verfahrens verbunden. Dadurch

[1] https://www.ite.uni-stuttgart.de/forschung/projekte/elfe/fem2d/fem2d.jpg

entsteht das sogenannte Gitter. Innerhalb dieses Gitters sind die einzelnen Elemente durchnummeriert. Diese Nummerierung legt die Reihenfolge fest, in der die einzelnen Elemente berechnet werden. Mithilfe dieses Gitters bzw. der einzelnen Elemente kann das eigentliche Problem angenähert werden, indem die Elemente in endlichen Zeitschritten $t_0 \dots t_n$ diskretisiert werden. Dazu werden an bestimmten Stellen, den sogenannten Knotenpunkten, Gleichungen aufgestellt. Diese Gleichungen nähern die gesuchte Lösungsfunktion an. Die einzelnen Gleichungen bezüglich *eines* Elements, werden in einer Matrix zusammengefasst und repräsentieren den Zustand des Elements zum Zeitpunkt t_i. Die einzelnen Matrizen werden anschließend wiederum zusammengefasst und stellen die globale Matrixgleichung auf. Diese Matrix beschreibt den Zustand der gesamten zu simulierenden Welt zum Zeitpunkt t_i. Über ein iteratives Verfahren können dann die Zeitschritte $t_0 \dots t_n$ berechnet werden.

2.5. Datenbank

Die folgenden Absätze basieren auf [KE11]. Ein Datenbanksystem (DBS) dient zur Speicherung großer Mengen an Informationen (Daten) und wird heutzutage in allen großen Organisationen zur Informationsverwaltung eingesetzt. Es besteht aus zwei Komponten: Das Datenbankverwaltungssystem (DBVS) ist ein standardisiertes Softwaresystem, mit dessen Hilfe die Daten in einer Datenbank verwaltet und ausgewertet werden können. Die eigentlichen Daten befinden sich in der Datenbank, die oftmals auch als Datenbasis bezeichnet wird.

Für den Einsatz eines Datenbanksystems sprechen nach [KE11] unter anderem die folgenden Gründe: Bei der Verwendung von Dateisystemen werden zu speichernde Informationen oftmals redundant, d.h. mehrfach, gespeichert. Datenbanken dagegen bieten einen gewissen Grad an *Redundanzfreiheit*. Des Weiteren können Dateien nur schwer miteinander verknüpft werden und es treten Probleme im Mehrbenutzerbetrieb auf. Dies kann zu ungewünschten Annomalien führen. Datenbanken hingegen bieten mächtige *Auswertungsoperationen* und einen kontrollierten *Mehrbenutzerbetrieb*. Ein weiteres Problem ist der mögliche Datenverlust. Bei der Verwendung von Dateisystemen kann im Fehlerfall nur schwer ein konsistenter Zustand wiederhergestellt werden. Datenbanken dagegen besitzen entsprechende *Recovery-Komponenten*. Zu den weiteren Vorteilen zählen unter anderem: *Daten- und Anwendungsunabhängigkeit*, die Festlegung von *Integritätsbedingungen* und die Möglichkeit einer *Zugriffskontrolle*.

Jedem Datenbanksystem liegt ein Datenmodell zu Grunde. Das Datenmodell legt die Modellierungskonstrukte fest, mit denen Datenobjekte erzeugt und verändert werden können. Im Rahmen dieser Arbeit werden relationale Datenbanken und XML-Datenbanken kurz erläutert.

2.5.1. Relationale Datenbanken

Dieser Abschnitt nutzt [KE11] als Basis. Eine relationale Datenbank besteht aus Relationen. Aus mathematischer Sicht wird eine Relation folgendermaßen definiert:

> Gegeben seien n Domänen (Wertebereiche) D_1, D_2,..., D_n. Diese müssen nicht zwangsläufig unterschiedlich sein, dürfen jedoch nur atomare Werte beinhalten. Eine Relation R ist dann eine Teilmenge des kartesischen Produkts der n Domänen:
>
> $$R \subseteq D_1 \times D_2 \times \cdots \times D_n$$

Die n Domänen beschreiben das Schema einer Relation. Die aktuelle Ausprägung des Schemas ist durch die Teilmenge des Kreuzprodukts gegeben und wird auch als Instanz bezeichnet. Ein Element der Menge R heißt Tupel.

Student

MatrikelNr	Nachname	Vorname
001	Doe	John
002	Doe	Jane

Vorlesung

VorlesungsNr	Name	SWS
111	DB1	4
222	DB2	4

hatPruefung

VorlesungsNr	MatrikelNr	Note
111	001	1.0
222	001	1.7
222	002	1.3

Tabelle 2.1.: Relationale Datenbank: Datensätze innerhalb einer Tabelle

Bildlich gesehen besteht eine relationale Datenbank aus Tabellen. Tabelle 2.1 zeigt die drei Relationen *Student*, *Vorlesung* und *hatPruefung*. Die einzelnen Spalten repräsentieren die Attribute. Die Werte, die ein Attribut annehmen kann, werden durch die zugehörige Domäne festgelegt. Eine Zeile entspricht einem Tupel. Die Relation *Student* hat die drei Attribute *MatrikelNr*, *Nachname* und *Vorname*, die z.B. vom Typ *integer*, *string* und *string* sein können. Jede Relation hat einen *Primärschlüssel*, der aus einem oder mehreren Attributen bestehen kann. Dieser muss eindeutig sein, darf keine *NULL*-Werte beinhalten und wird typischerweise durch Unterstreichen gekennzeichnet. Attribute bzw. Attributkombinationen, die auf eine andere Relation verweisen, werden Fremdschlüssel genannt. In obiger Tabelle ist das Attribut *VorlesungsNr* der Relation *hatPruefung* ein Fremdschlüssel. Es verweist auf die Relation *Vorlesung* und stellt die entsprechenden Tupel beider Relationen in Beziehung.

2.5.1.1. SQL

Dieser Abschnitt basiert auf [KE11]. Um Informationen aus einer Datenbank zu extrahieren, werden geeignete Sprachen benötigt. Für relationale Datenbanken gibt es zwei formale Abfragesprachen: die relationale Algebra sowie das Relationenkalkül. Die **Structured Query Language (SQL)** ist eine *deklarative* Abfragesprache und baut auf der relationalen Algebra und dem Relationenkalkül auf. Deklarativ bedeutet, dass der Nutzer nur definiert, **welche** Daten er haben möchte. Die eigentliche Auswertung übernimmt das Datenbanksystem. Tabelle 2.2 zeigt den Aufbau der wesentlichen Teile einer SQL-Anfrage in einer allgemeinen Form.

select $A_1, \ldots, A_n$ **from** $R_1, \ldots, R_k$ **where** P;

Tabelle 2.2.: Allgemeine Form einer SQL-Anfrage [KE11]

Dabei entspricht der **from**-Teil dem kartesischen Podukt $R_1 \times \cdots \times R_k$ der k Relationen. Der **where**-Teil definiert eine Selektion. Durch diese Selektion werden alle Tupel ausgewählt, die das Selektionssprädikat P erfüllen. Durch den **select**-Teil wird eine Projektion auf die Attribute $A_1, \ldots, A_n$ bestimmt. Sollen alle Attribute zurückgegeben werden, kann anstelle der Attribute $A_1, \ldots, A_n$ einfach ein „*“ angegeben werden. In Bezug auf Tabelle 2.1 würde der SQL-Ausdruck „***select** Nachname, Vorname, Name **from** Student, Vorlesung, hatPruefung **where** Student.MatrikelNr = hatPruefung.MatrikelNr **and** hatPruefung.VorlesungNr = Vorlesung.VorlesungsNr“ das in Tabelle 2.3 dargestellte Ergebnis liefern. Für weitere Informationen wird auf [KE11] verwiesen.

Nachname	Vorname	Name
Doe	John	DB1
Doe	Jane	DB1
Doe	Jane	DB2

Tabelle 2.3.: Ergebnis einer SQL-Anfrage

2.5.2. XML-Datenbanken

In Abschnitt 2.1 wird auf XML eingegangen. Oftmals ist es nötig XML-Dokumente, z.B. aus rechtlichen Gründen, dauerhaft zu speichern. XML-Datenbanksysteme unterstützen diese Aufgabe. [Sch03] beschreibt drei Grundmodelle: Beim **monolithischen Ansatz** repräsentiert ein einziges XML-Dokument die gesamte Datenbank. Indem dieses *eine* Dokument verändert wird, werden Informationen hinzugefügt bzw. gelöscht. Dadurch ist die Datenbank vollständig geordnet. Probleme gibt es, wenn z.B. ganze Dokumente gespeichert werden

sollen. So ist es nicht möglich einen Prolog oder eine Document Type Definition unterhalb eines Wurzelelementes zu speichern. Der **fragmentorientierte Ansatz** wird von [Sch03] folgendermaßen beschrieben:

> „Bei diesem Ansatz besteht die Datenbank aus Daten, die zunächst einmal (aus XML Sicht) nicht miteinander verknüpft sind. Vielmehr werden Daten hier erst beim Extrahieren aus der Datenbank zu XML-Fragmenten kombiniert. Analog werden XML-Fragmente (ggf. ganze XML-Dokumente) beim Abspeichern möglicherweise aufgelöst. Die Einheit der Operationen wird also im Allgemeinen dynamisch definiert."

Relationale Datenbanken, die die Speicherung von XML-Dokumenten unterstützen, verfolgen oftmals diesen Ansatz. Problematisch ist, dass der eigentliche Zusammenhang der einzelnen Teile innerhalb der Datenbank nicht sichtbar ist. Beim **dokumentorientierten Ansatz** besteht die Datenbank aus einzelnen XML-Dokumenten. Es werden immer ganze Dokumente eingefügt bzw. gelöscht. Mehrere Dokumente können in sogenannten *Collections* zusammengefasst werden. Dies kann automatisch, aufgrund festgelegter Regeln, oder manuell durch den Nutzer erfolgen.

Relationale Datenbanksysteme, wie z.B. Mircosoft SQL Server[1], die mit XML-Dokumenten umgehen können, werden als XML-enabled Datenbanksysteme bezeichnet. Native XML-Datenbanksysteme, wie z.B. MonetDB/XQuery[2] oder Exist[3], verfolgen den dokumentorientierten Ansatz.

Neben allgemeinen Anforderungen, wie z.B. Effizienz, Mehrbenutzerbetrieb, Skalierbarkeit und Verfügbarkeit, die auch für andere Datenbanksysteme gelten, müssen XML-Datenbanksysteme noch spezielle Anforderungen erfüllen. [Sch03] nennt unter anderem die Folgenden:

- Standardkonformität: Ein XML-Datenbanksystem sollte die wichtigsten XML-Spezifikationen und Empfehlungen berücksichtigen.
- Dokumentenbehandlung: Es sollte möglich sein *ganze* XML-Dokumente (inklusive Prolog und Document Type Definition) zu speichern und diese auch unverändert wiederzugeben.
- Schema-Unabhängigkeit: Es sollte möglich sein beliebige wohlgeformte XML-Dokumente zu speichern. Wenn eine schematische Beschreibung gegeben ist, muss deren Übereinstimmung (Validität) geprüft werden.
- Schemavorgabe: Es sollte möglich sein schematische Beschreibungen vorzugeben. Gegen diese werden zu speichernde XML-Dokumente dann validiert.
- Kodierung und Internationalisierung: Da XML-Dokumente insbesondere zum Datenaustausch verwendet werden, sollten möglichst viele Kodierungen verstanden werden.

[1] http://www.microsoft.com/sqlserver/en/us/default.aspx
[2] http://www.monetdb.org/XQuery/
[3] http://exist-db.org/exist/index.xml

- XML-spezifische Schnittstelle: Eine formale Abfragesprache sollte unterstützt werden und das Ergebnis wiederum wohlgeformtes XML sein.
- Originaltreue: XML-Dokumente sollten unverändert zurückgegeben werden.

2.5.2.1. XPath

Um Informationen aus einer XML-Datenbank zu extrahieren, wird eine ausdrucksstarke, deklarative Abfragesprache benötigt. XPath ist eine vom World Wide Web Consortium (W3C) standardisierte Sprache und dient zur Adressierung von Teilen von XML-Dokumenten. Dieser Abschnitt basiert auf [KE11] und [Sch03].

Ein XML-Dokument wird als Baum dargestellt. Der Wurzelknoten ist ein virtueller Knoten und kommt im eigentlichen XML-Dokument nicht vor. XPath unterstützt die folgenden Knotentypen: Elementknoten, Attributknoten, Namensraumknoten, Verarbeitungsanweisungsknoten, Kommentarknoten und Textknoten. Mithilfe von *Lokalisierungspfaden* kann durch ein XML-Dokument navigiert werden. Ein Lokalisierungspfad besteht aus aneinandergereihten *Lokalisierungsschritten*, welche durch ein „/“-Zeichen getrennt werden. Eine Knotenmenge wird ausgehend von einem Referenzknoten durch einen Lokalisierungsschritt selektiert. Im nächsten Schritt dient jeder dieser selektierten Knoten wiederum als Referenzknoten. Die auf diese Weise selektierten Knotenmengen werden vereinigt und bilden die Ergebnismenge. Ein Lokalisierungsschritt besteht aus drei Teilen:

axis::node-test[predicate]

Die *Achsen* geben die Richtung vor, in der durch die Baumstruktur navigiert wird:

- self: der Referenzknoten selbst
- attribute: die Attribute des Referenzknotens
- parent: der Vaterknoten des Referenzknotens
- child: alle direkt untergeordneten Knoten
- descendant: alle untergeordneten Knoten
- descendant-or-self: alle untergeordneten Knoten inkl. Referenzknoten
- ancester: alle übergeordneten Knoten
- ancester-or-self: alle übergeordneten Knoten inkl. Referenzknoten
- following: alle Knoten, die nach dem Referenzknoten im XML Dokument vorkommen (die Hierachie-Ebene der Knoten wird nicht beachtet) und keine direkten bzw. indirekten Kinder des Referenzknotens sind
- following-sibling: alle Knoten, die nach dem Referenzknoten im XML Dokument vorkommen und in der gleichen Hierachie-Ebene liegen

- preceding: alle Knoten, die vor dem Referenzknoten im XML Dokument vorkommen (die Hierachie-Ebene der Knoten wird nicht beachtet) und keine direkten bzw. indirekten Vorgänger des Referenzknotens sind
- preceding-sibling: alle Knoten, die vor dem Referenzknoten im XML Dokument vorkommen und in der gleichen Hierachie-Ebene liegen

Mithilfe des *Knotentests* kann die Achse eingeschränkt werden. Nur Knoten, die die definierte Bedingung erfüllen, dienen wiederum als Referenzknoten. Komplexere Bedingungen können durch *Prädikate* definiert werden.

Listing 2.11 Ausschnitt aus einem XML-Dokument

```
<Universitaet>
  <Fakultaeten>
    <Falutaet>
      <NameDerFakultaet>Informatik</NameDerFakultaet>
      <Vorlesungen>
        <Vorlesung Vorlesungsnummer=111>
          <NameDerVorlesung>DB1</NameDerVorlesung>
          <SWS>4</SWS>
        </Vorlesung>
      </Vorlesungen>
    </Falutaet>
    <Falutaet>
      <NameDerFakultaet>Mathematik</NameDerFakultaet>
      <Vorlesungen>
        <Vorlesung Vorlesungsnummer=222>
          <NameDerVorlesung>HM1</NameDerVorlesung>
          <SWS>4</SWS>
        </Vorlesung>
      </Vorlesungen>
    </Falutaet>
  </Fakultaeten>
</Universitaet>
```

Listing 2.11 zeigt ein XML-Fragment. Für dieses Fragement werden im Folgenden exemplarisch XPath Ausdrücke und deren Resultate gezeigt. So liefert der Ausdruck:

/child::Universitaet/child::Fakultaeten/child::Fakultaet/child::NameDerFakultaet

das folgende Ergebnis:

<NameDerFakultaet>Informatik</NameDerFakultaet>
<NameDerFakultaet>Mathematik</NameDerFakultaet>

Mithilfe eines Prädikats können alle Vorlesungen der Fakultät Informatik extrahiert werden:

/child::Universitaet/child::Fakultaeten/child::Fakultaet[child::NameDerFakultaet= "Informatik"]/ descendant-or-self::Vorlesung/child::NameDerVorlesung

Das Ergebnis wäre in diesem Fall:

<NameDerVorlesung>DB1</NameDerVorlesung>

2.5.2.2. XQuery

Die gerade beschriebene Sprache XPath bildet die Grundlage von XQuery. Eine Vielzahl an XML-Datenbanksystemen unterstützt XQuery als Abfragesprache. Dieser Abschnitt basiert auf [KE11] und [Sch03]. In XQuery werden Anfragen als *FLWOR*-Ausdrücke formuliert. FLWOR ist ein Apronym für:„**f**or . . . **l**et . . . **w**here . . . **o**rder by . . . **r**eturn . . . “. Mit *for* werden Variablen sukzessive, also iterativ, gebunden. Für jede durchgeführte Variablenbindung im *for*-Konstrukt, wird im *let*-Konstrukt eine einmalige Bindung durchgeführt. Das *where*-Konstrukt bietet die Möglichkeit Bedingungen zu definieren, mit denen bestimmte Knoten extrahiert werden können. Eine Sortierung kann mit dem *order-by*-Konstrukt realisiert werden. Das *return*-Konstrukt gibt die Knoten bzw. XML Fragmente zurück, die in die Ergebnismenge aufgenommen werden. Es wird nur für diejenigen Knoten ausgewertet, für die die im *where*-Konstrukt definierten Bedingungen erfüllt sind. FLWOR-Ausdrücke können beliebig tief geschachtelt werden und in einem XML-Fragment vorkommen. Auszuwertende XQuery Ausdrücke werden durch geschweifte Klammern kenntlich gemacht. Des Weiteren ist es möglich Joins sowie rekursive Anfragen zu realisieren. Der folgende XQuery Ausdruck würde in Bezug auf obiges XML-Dokument alle Vorlesungen extrahieren:

```
<Vorlesungen>
   {for $v in doc(“example.xml“)//Vorlesungen
   return $v}
</Vorlesungen>
```

3. SIMPL-Rahmenwerk

SIMPL (SimTech - Information Management, Processes and Languages) ist ein erweiterbares Rahmenwerk zur Einbindung externer, heterogener Datenquellen in Simulationsworkflows. Im Folgenden werden zunächst bestehende Probleme in Simulationsworkflows erläutert, bevor anschließend das SIMPL-Rahmenwerk detailliert betrachtet wird. Im weiteren Verlauf dieser Arbeit steht die Abkürzung *DM* für *Datenmanagement* (von engl. data management). Dieses Kapitel basiert, soweit nicht anders angegeben, auf [RRS+11].

3.1. Gründe für SIMPL

Mithilfe des Pandas-Rahmenwerks (vgl. Abschnitt 2.4.3.1) können unter anderem Knochenverformungen simuliert werden. Solche Simulationsabläufe werden oftmals als Simulationsworkflows (vgl. Abschnitt 2.3.3.2) modelliert. Innerhalb dieser Workflows werden Daten bereitgestellt und externen Programmen übergeben. Diese führen dann Berechnungen auf den Datensätzen durch. Die benötigten Programme werden als Web Services bereitgestellt und durch das Scientific Workflow Management System (sWfMS) entsprechend aufgerufen. Charakteristisch für solche Simulationen bzw. Simulationsabläufe sind komplexe mathematische Berechnungen sowie verschiedene Aufgaben im Bereich der Datenverwaltung und Datenbereitstellung. Die Vorteile der Workflow-Technologie werden in Abschnitt 2.3 erläutert. Problematisch ist jedoch, dass sehr große Datenmengen aus verschiedenen Quellen verarbeitet werden müssen, die oftmals in proprietären Formaten vorliegen. Diese müssen dann in ein für die Simulation geeignetes Format transformiert werden. Ein sWfMS bietet meist keine integrierte Möglichkeit solche heterogenen Datenquellen zu verwalten. Oftmals muss auf spezielle Dienste, die auf eine einzige Datenquelle und/oder ein einzelnes Simulationsprogramm zugeschnitten sind, zurückgegriffen werden. Dies führt zu einem erhöhten Arbeitsaufwand für den Anwender. Dieser muss unter anderem adäquate Quellen finden sowie nötige Datentransformationen implementieren oder händisch nach passenden Diensten suchen, sofern es diese überhaupt gibt. Gerade bei komplexen Simulationen, die eine Vielzahl an heterogenen Datenquellen nutzen, führt dies aufgrund der enormen Komplexität zu Problemen.

Wünschenswert wäre eine Abstraktionsebene innerhalb eines sWfMS, welche den Zugriff auf externe Datenquellen sowie deren Verwaltung vereinheitlicht und somit auch vereinfacht. An dieser Stelle setzt das SIMPL-Rahmenwerk an. SIMPL bietet einheitliche Zugriffsmethoden um, aus Simulationsworkflows heraus, auf beliebige externe Datenquellen zuzugreifen. Metadaten beschreiben die hierfür notwendigen Zuordnungen zwischen vereinheitlichten Zugriffsmethoden und konkreten Zugriffsoperationen. Dem Modellierer stehen für die

Definition des Datenmanagements zusätzliche Workflowaktivitäten zur Verfügung (vgl. Abschnitt 3.3). Zusätzlich beinhaltet das SIMPL-Rahmenwerk Datenmanagementpatterns, um z.B. ETL Operationen einfacher definieren zu können (vgl. Abschnitt 3.7).

3.2. Architektur des SIMPL-Rahmenwerks

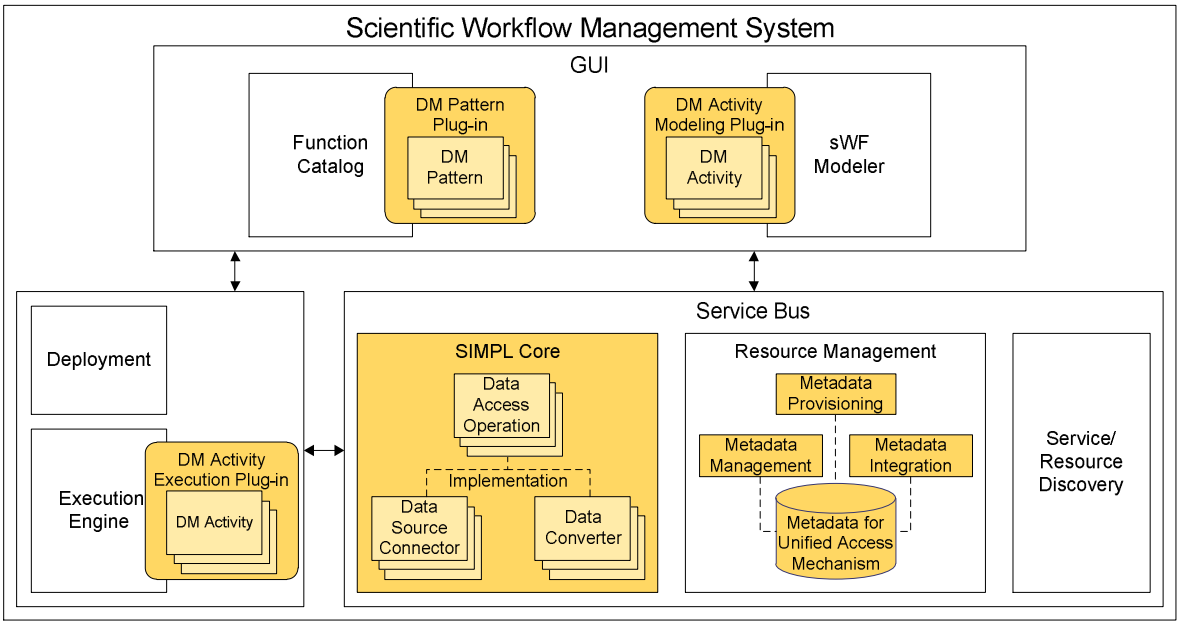

Abbildung 3.1.: Das SIMPL-Rahmenwerk eingebettet in ein sWfMS [RRS+11]

Das SIMPL-Rahmenwerk erweitert ein sWfMS (vgl. Abschnitt 2.3.5). Die erweiterte Architekur wird in Abbildung 3.1 dargestellt. Die **SIMPL Core** Komponente wurde im Service Bus eingebettet und bietet vereinheitlichte logische Schnittstellen, um auf beliebige externe Datenquellen zuzugreifen. Die **Resource Management** Komponente wurde dahingehend erweitert, dass Metadaten über die Beziehung zwischen logischen Schnittstellen und konkreten Zugriffsmechanismen gespeichert werden können. Das **DM Activity Modeling Plug-in** erweitert den sWF Modeller um zusätzliche Aktivitäten. Dem Modellierer stehen somit alle Aktivitäten aus BPEL-DM (vgl. Abschnitt 3.3) zur Verfügung. Das **DM Activity Execution Plug-in** erweitert die Execution Engine und sorgt dafür, dass die neu eingeführten Aktivitäten auch zur Laufzeit ausgeführt werden können. Das **DM Pattern Plug-in** erweitert den Function Catalog und unterstützt den Nutzer dahingehend, dass dieser vorgefertigte Patterns, welche komplexe Operationen darstellen, übernehmen kann (vgl. Abschnitt 3.7).

3.3. BPEL-DM

In Abschnitt 2.3.4 wurde bereits auf BPEL, die Business Process Execution Language, eingegangen. BPEL-DM (Business Process Execution Language extension for Data Management) erweitert BPEL um zusätzliche Aktivitäten, die sogenannten DM Aktivitäten. Diese Aktivitäten beinhalten bzw. definieren eine DM-Operation, wie z.B. eine Anweisung, die dann bei der Ausführung einer Aktivität zu einer spezifizierten Datenquelle geschickt wird. Die folgenden DM Aktivitäten stehen in BPEL-DM zur Verfügung:

- IssueCommand
- RetrieveData
- WriteDataBack

Jede dieser Workflow Aktivitäten ruft die entsprechende Operation des SIMPL Cores auf. Dieser ist wiederum für die Ausführung der eigentlichen Anweisung auf der Datenquelle verantwortlich. Im weiteren Verlauf dieser Arbeit bezeichnet der Begriff *Datenquelle* ein System, welches Daten speichern und verwalten kann. Dabei kann es sich um Datenbanken oder aber auch um Dateisysteme handeln. Die eigentlichen Anweisungen, die auf der Datenquelle ausgeführt werden, werden *DM commands* genannt. Für relationale Datenbanken wären dies z.B. SQL Konstrukte und für Dateisysteme z.B. Shell-Befehle.

Die neu definierten DM Aktivitäten erhalten als Eingabeparameter eine spezielle BPEL Variable. Diese Variable referenziert die Datenquelle, auf der die eigentliche Anweisung ausgeführt werden soll, und wird *data source reference variable* genannt. Solch eine Referenz ist ein *logical data source descriptor*, der entweder den logischen Namen der Datenquelle oder eine Anforderungsbeschreibung mit funktionalen und nicht-funktionalen Anforderungen beinhaltet. Der logische Name einer Datenquelle referenziert genau eine Datenquelle, deren Beschreibung im Resource Management hinterlegt ist. Mithilfe einer Anforderungsbeschreibung ist es möglich, eine geeignete Datenquelle erst zur Laufzeit zu suchen und einzubinden.

Jede Datenquelle verwaltet mehrere Container (*datacontainer*). In einer relationalen Datenbank wären dies z.B. die einzelnen Tabellen. Mithilfe von *data container reference variables* können einzelne Container über logische Namen referenziert werden. Das Resource Management speichert die dafür notwendigen Zuordnungen zwischen logischem Namen und konkretem Identifikator. Des Weiteren benötigt das SIMPL-Rahmenwerk noch *data set variables*. Diese Variablen dienen als Zielcontainer, um Daten in einen Workflow zu laden. Die Struktur dieser Variablen wird durch XML Schema Definitionen beschrieben. Dabei müssen die einzelnen Definitionen die Besonderheiten der unterschiedlichen Datenquellen berücksichtigen. Tabellenbasierte Daten, wie sie in relationalen Datenbanken oder CSV-basierten Dateien vorkommen, werden z.B. in einer XML RowSet Struktur gespeichert.

Im Folgenden werden nun die einzelnen DM Aktivitäten im Detail beschrieben. Die *IssueCommand* Aktivität bietet die Möglichkeit Daten zu manipulieren sowie Datenstrukturen, wie z.B. die Struktur einer Datenbanktabelle, zu erstellen, zu verwerfen oder zu ändern. Als Eingabeparameter erwartet diese Aktivität neben einer Referenz auf eine Datenquelle

noch einen *DM command*. Die spezifizierte Anweisung wird auf der gewählten Datenquelle ausgeführt. Die Execution Engine erwartet eine Bestätigung über den Erfolg bzw. Misserfolg der Operation. Bei einem Misserfolg kann eine Fehlerbehandlung bzw. Fehlerkompensation gestartet werden. Im Gegenzug wird bei einem Erfolg der Workflow gemäß seines Kontrollflusses weiter ausgeführt.

Die *RetrieveData* Aktivität erhält genau wie die IssueCommand Aktivität einen *DM command*. Diese Anweisung muss Daten generieren. Bei einer erfolgreichen Ausführung der Anweisung werden die generierten Daten an die Execution Engine zurückgeschickt. Zusätzlich wird bei dieser Aktivität eine *data set variable* angegeben. Die generierten Daten werden dann in dieser Variable gespeichert. Im Fehlerfall kann wiederum eine entsprechende Fehlerbehandlung gestartet werden.

Die *WriteDataBack* Aktivität ist das Gegenstück der RetrieveData Aktivität. Daten aus dem Workflow Kontext können in einer Datenquelle gespeichert werden. Als Eingabeparameter werden dazu eine *data set variable* sowie eine *data container reference variable* benötigt. Die durch die erste Variable spezifizierten Daten werden im Container, der durch die zweite Variable referenziert wird, gespeichert. Die Execution Engine erwartet analog zur IssueCommand Aktivität eine Benachrichtigung über den Erfolg bzw. Misserfolg der Ausführung.

Des Weiteren bietet das SIMPL-Rahmenwerk die Möglichkeit *data container reference variables* als Parameter in *DM commands* einzubauen. So können diese z.B. in der FROM Klausel einer SQL Anweisung genutzt werden. Dasselbe gilt für herkömmliche BPEL Variablen. Variablen vom Type String oder Integer können z.B. in Vergleichsoperationen eingebunden werden. Um eine Variable innerhalb einer Anweisung kenntlich zu machen, wird der Variablenname mit „#"-Zeichen umschlossen. Die Execution Engine erkennt diese Kennzeichnung und löst sie auf. Der Variablenwert wird gelesen und an entsprechender Stelle eingefügt. Bei *data container reference variables* ist das der logische Name des referenzierten Containers, der wiederum vom SIMPL Core aufgelöst wird.

3.4. SIMPL Core

Der SIMPL Core bietet generische Operationen, um auf beliebige externe Datenquellen zuzugreifen. Die Spezifikationen sind unabhängig von den darunter liegenden Datenquellen. Die Operationen, die der SIMPL Core bietet, übernehmen die Namen der BPEL-DM Aktivitäten: IssueCommand, RetrieveData und WriteDataBack. Jede BPEL-DM Aktivität ruft die zugehörige SIMPL Core Operation auf. Abbildung 3.2 zeigt die auszutauschenden Daten zwischen Execution Engine und SIMPL Core. Der SIMPL Core leitet *DM commands*, generierte Daten und Benachrichtigungen nur weiter und führt selbst keine komplexen Transformationen oder Analysen durch.

Jede SIMPL Core Operation erwartet einen *logical data source descriptor* als Eingabe, um die entsprechende Datenquelle zu finden. Die Operationen IssueCommand und RetrieveData benötigen des Weiteren jeweils einen *DM command*. Bei der RetrieveData Operation muss dieser Daten generieren. Die WriteDataBack Operation erwartet ein *data set* sowie eine

Referenz auf einen Container als Eingabe. Die IssueCommand und WriteDataBack Operationen benachrichtigen die Execution Engine über Erfolg bzw. Misserfolg. Die RetrieveData Operation sendet nach erfolgreicher Ausführung die generierten Daten an die Execution Engine. Im Fehlerfall wird wiederum eine entsprechende Benachrichtigung gesendet.

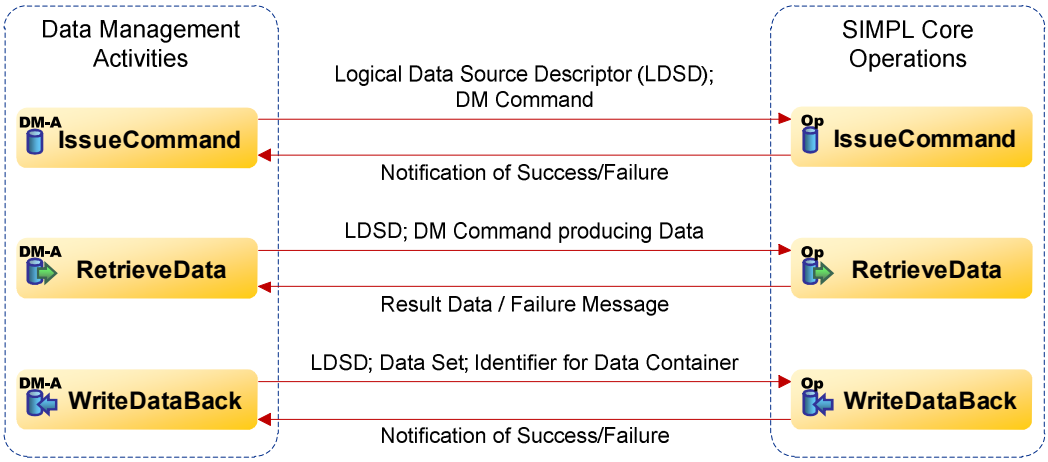

Abbildung 3.2.: Datenaustausch zwischen Execution Engine und SIMPL Core [RRS+11]

Für den Zugriff auf beliebige Datenquellen müssen die generischen Operationen des SIMPL Cores für konkrete Datenquellen oder für bestimmte Typen von Datenquellen implementiert werden. *Data source connectors* übernehmen diese Aufgabe. So könnte es z.B. einen Konnektor für relationale Datenbanken, die JDBC zur Kommunikation nutzen, geben, sowie einen Konnektor für Dateisysteme. Ein Konnektor muss jedoch nicht alle generischen Operationen unterstützten bzw. implementieren. Bei Sensornetzen z.B. macht eine WriteDataBack Operation keinen Sinn. Zusätzlich zu den *data source connectors* gibt es noch *data converter*. Diese Konverter transformieren Daten zwischen dem Ausgabeformat des Konnektors und einem XML-basierten Format, welches vom Workflow unterstützt wird. So könnte es z.B. einen Konverter geben, der ein JDBC Result Set in eine XML RowSet Darstellung überführt und umgekehrt.

3.5. Resource Management

Im Resource Management werden benötigte Metadaten gespeichert. Diese Metadaten werden zur Abbildung der generischen Operationen des SIMPL Cores auf konkrete Zugriffsmechanismen benötigt. Dazu können im Resource Management die folgenden vier Objekte hinterlegt werden:

- Data Sources
- Data Containers

- Data Source Connectors
- Data Converters

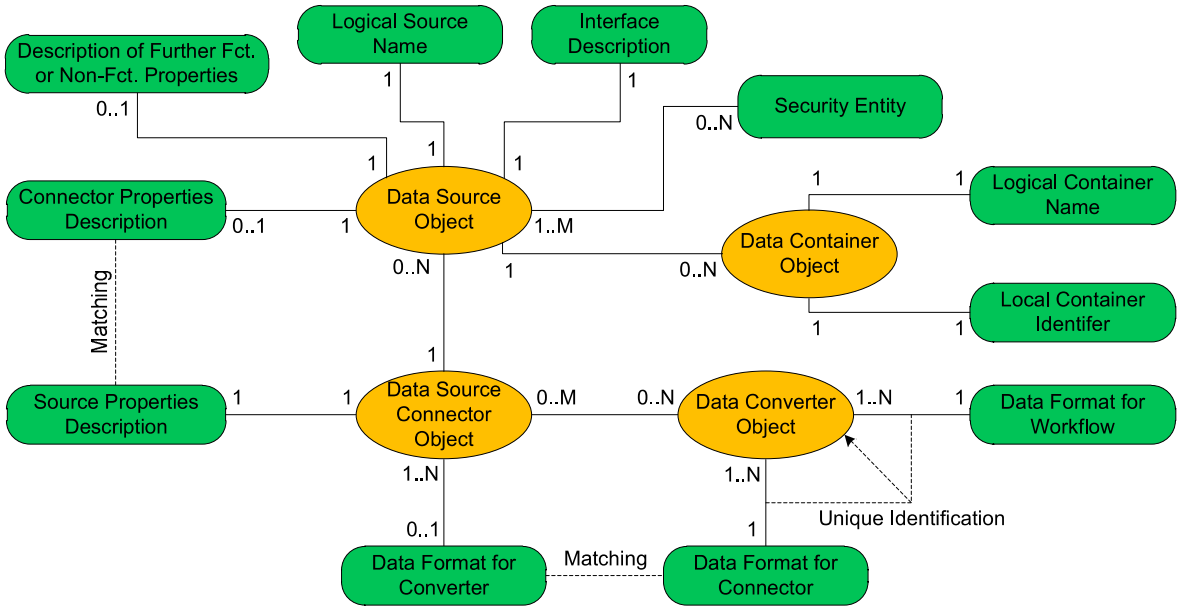

Abbildung 3.3.: Zusammenhang zwischen den verschiedenen Metadaten [RRS+11]

Abbildungg 3.3 zeigt die Beziehung der verschiedenen Objekte untereinander. Eine Datenquelle wird über Attribute beschrieben. Für jede Datenquelle gibt es einen einzigartigen *logical source name* der zur Identifizierung innerhalb des SIMPL-Rahmenwerks dient. Dieser Name kann innerhalb eines Workflows als *logical datasource descriptor*, z.B. in einer *data source reference variable*, genutzt werden. Die *Interface Description* beinhaltet Informationen über das Interface der Datenquelle, z.B. unter welcher Addresse die Datenquelle erreicht werden kann. Das Attribut *Security entitiy* enthält Authentifizierungsinformationen, wie z.B. Benutzername und Passwort. Das Attribut *description of further functional or non-functional properties* beinhaltet Information über die Eigenschaften einer Datenquelle, wie z.B. die maximale Antwortzeit. Diese Informationen werden für ein Late Binding benötigt. Ein *data container object* beschreibt die Container, die von einer Datenquelle verwaltet werden. Die einzelnen Container erhalten einen *logical container name*, der innerhalb eines Workflows als Containerreferenz genutzt werden kann. Der *logical container name* wird mit einem *local container identifier* gepaart, der eindeutig den entsprechenden Container in der Datenquelle identifiziert.

Wenn eine Datenquelle im Resource Management registriert oder verändert wird, kann der Nutzer direkt einen passenden *data source connector* angeben. Oder aber es kann eine *connector properties description* hinterlegt werden, die besagt, welche Eigenschaften ein geeigneter Konnektor haben muss. Jeder *data source connector* erhält dafür eine *source properties description*, die im Gegenzug besagt, welche Eigenschaften der Konnektor von einer Datenquelle erwartet.

Darauf aufbauend kann dann ein passender Konnektor anhand der *connector properties description* und *source properties description* ausgewählt werden.

Das Attribut *data format for connector* beinhaltet eine Formatbeschreibung. Diese legt fest, welches Ein- und Ausgabeformat der Konnektor unterstützt. Konverter teilen diese Informationen. Dadurch können passende Paare von Konnektor und Konverter gebildet werden. Des Weiteren benötigen Konverter noch eine weitere Formatbeschreibung (*data format for workflow*). Diese legt fest, in welchem Format der Konverter die Eingabedaten, z.B. als XML RowSet, aus dem Workflow erwartet, und in welchem Format er die Ausgabedaten wiederum zurückschickt.

3.6. Ablauf einer DM Aktivität

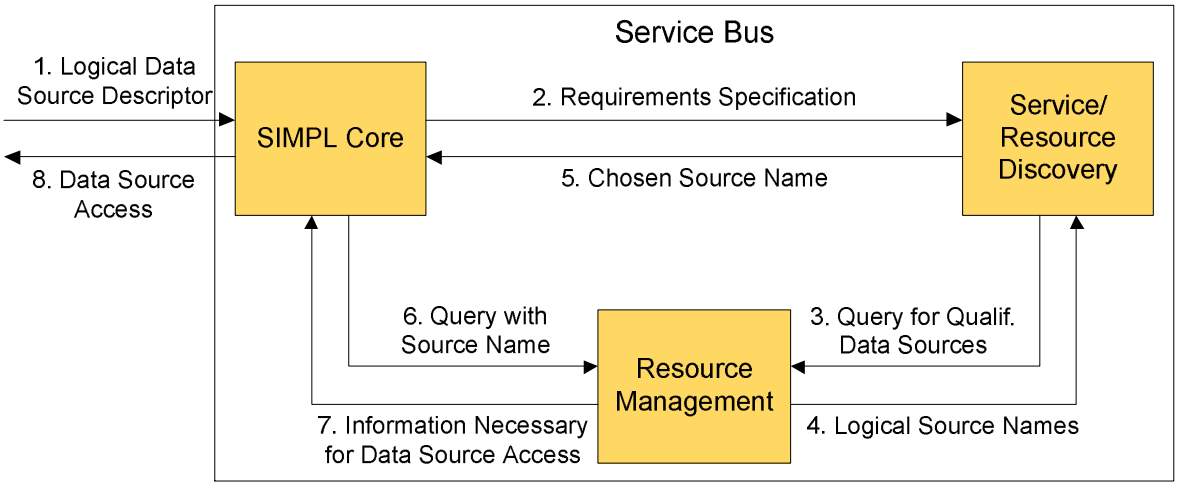

Abbildung 3.4.: Interaktion zwischen den verschiedenen Service Bus Komponenten [RRS+11]

Greift eine DM Aktivität mithilfe eines *logical data source descriptors* auf eine Datenquelle zu, muss der SIMPL Core alle benötigten Informationen über die referenzierte Datenquelle bereitstellen. Dazu zählen unter anderem die *Interface Description*, die Zugangsdaten sowie geeignete Konnektoren und Konverter. Des Weiteren müssen logische Containerreferenzen aufgelöst werden. Abbildung 3.4 zeigt exemplarisch die Interaktion der beteiligten Service Bus Komponenten. Der SIMPL Core erhält einen *logical data source descriptor* (1). Verweist die Referenz auf keine konkrete Datenquelle, sondern beinhaltet sie eine Anforderungsbeschreibung, leitet der SIMPL Core diese an die Service/Resource Discovery Komponente weiter (2). Diese sucht wiederum in der Resource Management Komponente nach geeigneten Datenquellen und wählt eine davon aus (3 und 4). Der logische Name dieser Datenquelle wird der SIMPL Core Komponente mitgeteilt (5), welche wiederum in der Resource Management Komponente die für den Zugriff auf die Datenquelle notwendigen Informationen abfragt

(6 und 7). Die entsprechende SIMPL Core Operation kann nun mithilfe der gesammelten Informationen auf die Datenquelle zugreifen (8). Beinhaltet *logical data source descriptor* bereits den logischen Namen der Datenquelle, entfallen die Schritte 2 bis 5. Bei einer RetrieveData oder WriteDataBack Aktivität sendet die Execution Engine noch zusätzlich den Datentyp der zugehörigen BPEL Variablen, in der die Daten abgelegt sind/werden, mit. Mithilfe dieser Information kann dann ein passender Konverter gewählt werden.

3.7. Datenmanagementpatterns

Wie bereits in Abschnitt 3.2 erwähnt, können *Datenmanagementpatterns* bei der Modellierung verwendet werden. Dabei handelt es sich um vorgefertigte Datenmanagement-Operationen, die nur noch parametrisiert werden müssen. Dazu wurde der Function Catalog des sWfMS um das *DM Pattern Plug-in* erweitert, welches eine Liste der verfügbaren Patterns zur Verfügung stellt. Dadurch wird eine neue Abstraktionsebene geschaffen. Der Modellierer wird bei der Konkretisierung eines gewählten Patterns in einem semi-automatischen Prozess unterstützt und muss keine konkreten Datenmanagement-Operationen definieren.

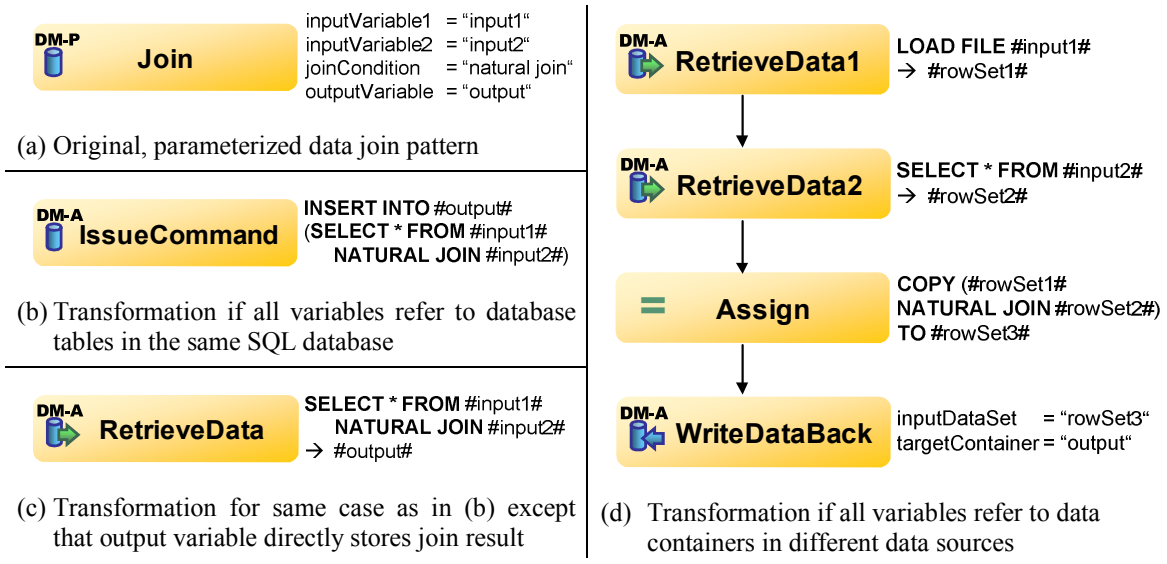

Abbildung 3.5.: Beispiel eines Join Patterns [RRS+11]

Abbildung 3.5 zeigt ein Pattern, das einen Join zwischen zwei Datensätzen realisiert. Der Modellierer muss lediglich spezielle Parameter setzen. In diesem Fall muss er zwei Input-Variablen, eine Output-Variable sowie eine Join-Condition angeben. Transformationsregeln überführen das abstrakte Pattern (vgl. Abbildung 3.5 (a)) in ausführbare Workflow-Fragmente. Welche Aktivitäten anschließend im Speziellen ausgeführt werden, bestimmen die gewählten Eingabeparameterwerte. Handelt es sich bei den benötigten Variablen um *data container reference variables*, die auf Datenbanktabellen in derselben Datenbank verweisen, reicht eine

IssueCommand Aktivität. Diese führt dann den Join innerhalb der Datenbank aus (vgl. Abbildung 3.5 (b)). Sollen die generierten Daten im Workflow bereitgestellt werden, muss eine RetrieveData Aktivität verwendet werden (vgl. Abbildung 3.5 (c)). Verweisen alle drei Variablen auf Container in unterschiedlichen Datenquellen, werden z.B. zwei RetrieveData Aktivitäten sowie ein WriteDataBack Aktivität benötigt (vgl. Abbildung 3.5 (d)).

Nachdem das Prinzip der Datenmanagementpatterns nun grob erläutert wurde, wird der Ansatz noch konkretisiert. Die nächsten Absätze basieren auf [Rei11]. Bevor ein Workflow, der Datenmanagementpatterns beinhaltet, ausgeführt werden kann, müssen die einzelnen Datenmanagementpatterns in ausführbare Workflow-Fragmente transformiert werden. Zu beachten ist, dass die Transformation erst dann geschehen kann, wenn die benötigten Datenquellen feststehen. Bei einer statischen Beschreibung der Datenquellen kann die Transformation bereits zur Modellierungszeit oder kurz vor dem Deployment geschehen. Bei einem Late Binding der Datenquellen erst zur Laufzeit. Abbildung 3.6 zeigt das benötigte Verarbeitungsmodell. Der **Pattern Transformer** beinhaltet die **Transformer Engine** Komponente. Auf der linken Seite ist ein Workflow dargestellt, der Datenmanagementpatterns beinhaltet. Der Pattern Transformer bzw. die Transformer Engine generiert daraus einen Workflow (Workflow auf der rechten Seite), der nur noch aus ausführbaren Workflow-Fragmenten besteht. Um dies zu erreichen, wird durch den Workflow-Graphen traversiert. Für jedes Datenmanagementpattern wird nach einer geeigneten Transformationsregel gesucht. Eine Transformationsregel besteht dabei aus einem *Condition Part* sowie einem *Action Part*.

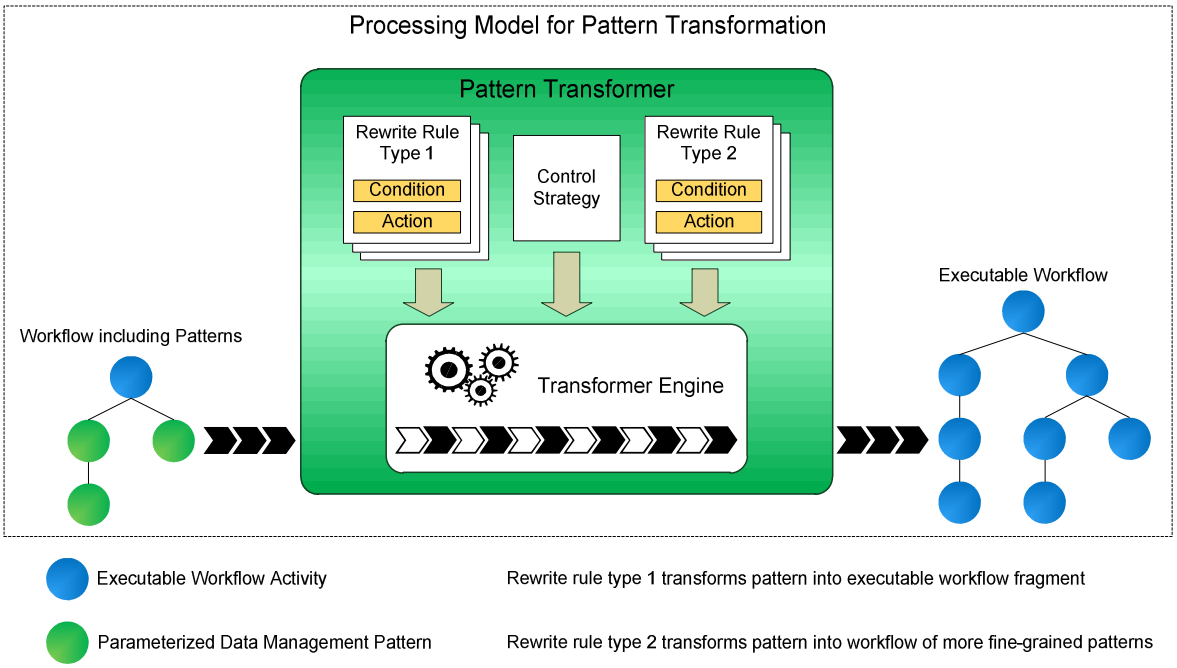

Abbildung 3.6.: Modell für die Transformation eines Datenmanagementpatterns [Rei11]

Der Condition Part legt die Bedingungen fest, wann diese Transformationsregel angewendet werden kann bzw. darf. Dabei hängt der Condition Part von den Parameterwerten des zu transformierenden Patterns sowie den Metadaten über beteiligte Ressourcen (Datenquellen, Dienste, etc.) ab. Durch die Parameterwerte werden unter anderem die beteiligten Datenquellen und Container beschrieben. Die eigentliche Operation, die auf den referenzierten Daten ausgeführt werden soll, wird durch das Datenmanagementpattern selbst beschrieben.

Der Action Part hingegen legt fest, wie das eigentliche Datenmanagementpattern transformiert wird bzw. auf ein ausführbares Workflow-Fragment abgebildet wird. Dabei gibt es zwei Typen von Regeln: Transformationsregeln vom **Regeltyp-1** überführen das abstrakte Pattern *direkt* in ein ausführbares Workflow-Fragment. Dabei geschieht die Transformation entweder über die DM Aktivitäten des SIMPL-Rahmenwerks oder über Service-Aufrufe. D.h., dass entsprechende Workflow-Fragmente bereits existieren, oder mithilfe von Metadaten generiert werden müssen. Wird hingegen eine Transformationsregel vom **Regeltyp-2** angewendet, ist das Resultat ein Workflow-Fragment, welches immer noch Datenmanagementpatterns beinhaltet. Meist sind diese Patterns dann jedoch feingranularer. In so einem Fall muss die Transformer Engine iterativ immer wieder nach geeigneten Transformationsregeln suchen und diese anwenden, bis eine Transformationsregel vom Regeltyp-1 gefunden und angewendet werden kann.

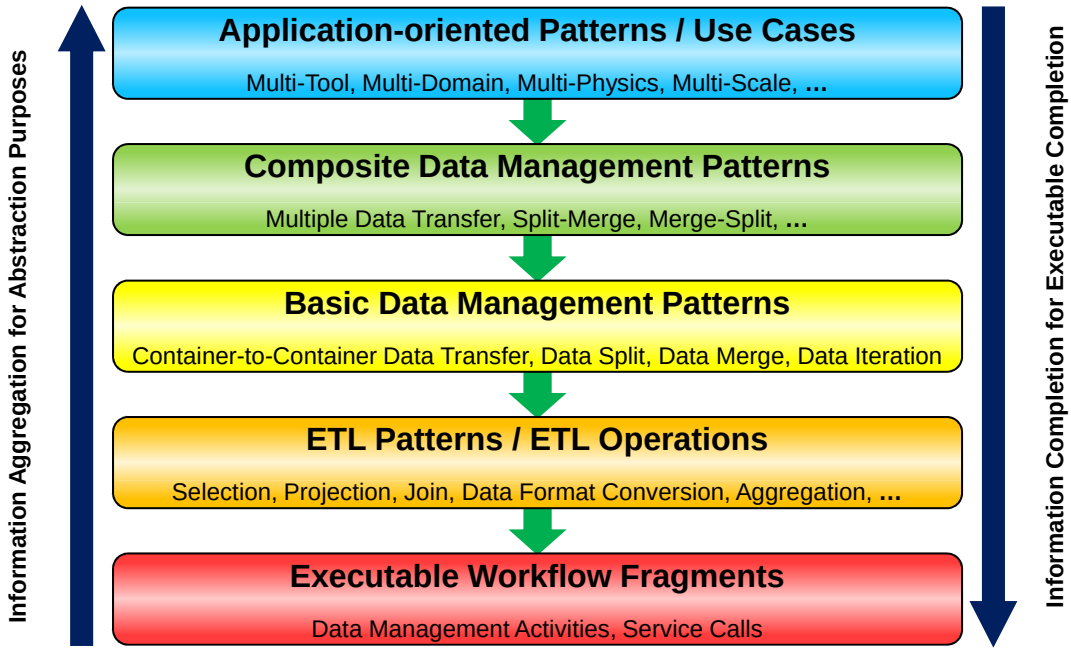

Abbildung 3.7.: Hierarchie der Datenmanagementpatterns [RM11]

Die **Kontrollstrategie** legt dabei fest, welche Transformationsregeln auf ein Datenmanagementpattern angewendet werden können und in welcher Reihenfolge die zur Verfügung stehenden Transformationsregeln auf eine mögliche Anwendbarkeit geprüft werden. Zuerst werden alle Regeln bestimmt, die auf das aktuell betrachtete Pattern angewendet werden können. Anschließend werden z.B. in der Regel zunächst Regeln vom Regeltyp-1 auf eine mögliche Anwendbarkeit geprüft. Auf diese Weise wird versucht Datenmanagementpatterns so früh wie möglich in ausführbare Workflow-Fragmente zu transformieren.

Abbildung 3.7 zeigt eine Hierarchie, die verschiedene Klassen von Datenmanagementpatterns definiert. Auf der untersten Ebene befinden sich die *ausführbaren Workflow-Fragmente*. Eine Schicht höher befinden sich die *ETL Patterns/ETL Operationen*. Diese Datenmanagementpatterns modellieren typische feingranulare DM-Operationen. Diese Operationen bieten unter anderem die Möglichkeit Daten zu extrahieren, zu kopieren sowie neu einzufügen. Die mittlere Ebene beschreibt die *Basisdatenmanagementpatterns*. Diese Patterns definieren komplexere DM-Operationen und kapseln in der Regel mehrere ETL Patterns. Sie bilden die Basis der nächten Hierarchiestufe. Die *zusammengesetzten Datenmanagementpatterns* bestehen aus mehreren Basisdatenmanagementpatterns und schaffen auf diese Weise eine weitere Abstraktionsstufe. Auf der obersten Schicht befinden sich dann die *anwendungsorientierten Patterns/Anwendungsfälle*. Dabei handelt es sich um Datenmanagementpatterns, die aus konkreten Anwendungsfällen (z.B. Simulationsworkflows) extrahiert wurden.

Je höher die Hierarchiestufe, desto mehr Information werden aggregiert und können somit weggelassen werden. Auf diese Weise wird ein gewisser Grad an Abstraktion geschaffen und der Nutzer muss bei der Modellierung über weniger Informationen verfügen. Dementsprechend müssen, je tiefer die Hierarchiestufe, Informationen hinzugefügt werden, damit das Datenmanagementpattern in ein ausführbares Workflow-Fragment transformiert werden kann. Generell soll der Wissenschaftler eher in der Sprache seiner Simulationsmodelle, und nicht in der Sprache der Workflowmodelle oder Datenmodelle reden. Dies gilt insbesondere je höher die Hierarchiestufe. In Abschnitt 5.4 wird noch detailliert auf den aktuellen Stand des Konzeptes und der Implementierung des Patternansatzes eingegangen.

4. Anwendungsfälle

In diesem Kapitel werden Anwendungsfälle, die im weiteren Verlauf der Arbeit von Bedeutung sind, erläutert. Zuerst wird auf eine Simulation, die mithilfe von ChemShell (vgl. Abschnitt 2.4.3.2) durchgeführt wird, eingegangen. Anschließend wird eine Simulation unter Verwendung des Pandas-Rahmenwerks (vgl. Abschnitt 2.4.3.1) erläutert. Zuletzt wird diese Simulation noch einmal betrachtet. Jedoch werden in diesem Szenario dann die Simulationsprogramme Pandas und Matlab gekoppelt.

4.1. Chemische Reaktion unter Verwendung eines Katalysators

Dieses Kapitel basiert auf [Mü10] und [Ari12]. Mithilfe von ChemShell können **hybride bzw. multi-skalare** quantenmechanische/molekularmechanische Simulationen durchgeführt werden. Im Gegensatz zum Pandas-Rahmenwerk basiert ChemShell nicht auf der Finiten Element Methode. Stattdessen wird versucht die Schrödinger Gleichung zu lösen, was in der Regel sehr rechen- und datenintensiv ist.

Abbildung 4.1 zeigt das Konzept eines Simulationsworkflows, der die Konversion eines Glutamats in Methyl-Aspartat simuliert. Dabei wird das Enzym Glutamat Mutase als Katalysator verwendet. Im Folgenden wird nun der Ablauf dieses Workflows abstrakt beschrieben. Dabei werden die chemischen Details vernachlässigt.

Zuerst werden die benötigte Eingabedateien, wie z.B. eine Protein-Datei, die aus einer Protein-Datenbank geladen werden kann, geladen (*load input files*). Danach wird eine molekulardynamische Berechung mithilfe dieser Daten durchgeführt (*molecular dynamic calculation*). Die Resultate werden in einer Datei gespeichert. Anschließend müssen die Eingabedaten für die eigentliche **hybride** Simulation selektiert werden. Dies kann entweder automatisch (*automatically select input for QM/MM*) oder durch den Wissenschaftler selbst (*select input for QM/MM*) geschehen. Im letzteren Fall werden die zur Verfügung stehenden Daten visualisiert (*visualization*), um dem Wissenschaftler die Entscheidung über die zu selektierenden Daten zu vereinfachen. Sowohl die selektierten als auch noch weitere Daten werden anschließend verarbeitet und bereitgestellt (*prepare QM/MM input*). Danach wird die eigentliche Simulation ausgeführt (*hybrid quantum mechanincs/molecular mechanincs simulation*). Da die Lösung der Schrödinger Gleichung in der Regel sehr rechen- und datenintensiv ist, werden Zwischenergebnisse in Dateien gespeichert. Diese Zwischenergebnisse werden parallel zu den nachfolgenden Berechnungen visualisiert, damit der Wissenschaftler sie asynchron validieren kann. Wenn die Simulation beendet wurde, werden auch die finalen Endergebnisse in Dateien gespeichert. Diese Dateien beinhalten alle Informationen, die für eine erneute

Ausführung bzw. Reproduktion erforderlich sind. Zuletzt werden auch die Endergebnisse visualisiert, damit der Wissenschaftler diese validieren kann (*visualization, verify final results*).

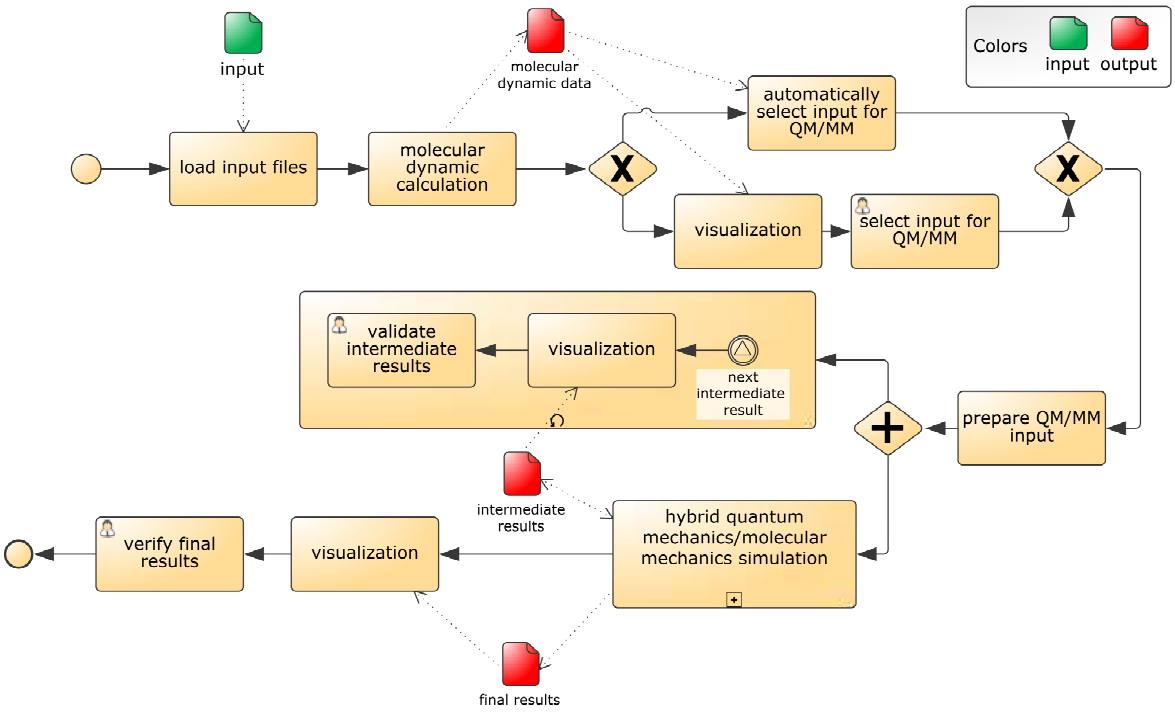

Abbildung 4.1.: ChemShell: Chemische Reaktion unter Verwendung eines Katalysators [Mü10]

4.2. Knochenmodellierung mit Pandas

Dieser Abschnitt basiert auf [RSRM] und [RRS+11]. Im Folgenden wird nun ein Simulationsworkflow erläutert, der die Verformung eines Knochens unter Belastung simuliert. Dabei wird das Pandas-Rahmenwerk genutzt, um die Knochenstrukturänderungen auf der biomechanischen Ebene zu simulieren. Abbildung 4.2 stellt diesen Workflow grafisch dar und zeigt die einzelnen Aktivitäten sowie relevante Ein- und Ausgabedaten. Dieser Workflow kann in drei Phasen unterteilt werden: die *Preprocessing Phase*, die *Solving Phase* und die *Postprocessing Phase*.

Die erste Aktivität der Preprocessing Phase (*Define Simulation Body*) lädt die benötigten Eingabedaten, die Informationen über den zu simulierenden Knochen enthalten. Dazu gehören unter anderem Informationen über die Knochenstruktur sowie die Materialeigenschaften. Je nach konkretem Anwendungsszenario können die benötigten Informationen in Datenbanken oder Dateien vorliegen oder aber sogar Bilder sein. Die zweite Aktivität

(*Create FEM Parameters*) dient dem Zweck FEM Parameter, wie z.B. Interpolationsfunktionen, aus einer proprietären Datei zu extrahieren. Die nächste Aktivität (*Adjust Initial/Boundary Conditions*) legt zum einen Anfangsbedingungen, wie z.B. die Struktur des Knochens zu Beginn der Simulation, und zum anderen Randbedinungen, wie z.B. der zeitabhängige Druck von außen auf den Knochen, fest. Diese Daten werden aus strukturierten CSV-Dateien gelesen. Wissenschaftler müssen geeignete Datensätze selektieren und in ein für das Pandas-Rahmenwerk geeignetes Format überführen. Die letzte Aktivität der Preprocessing Phase (*Create Simulation Commands*) schreibt Simulationsbefehle in eine Datei. So wird z.B. ein geeigneter Matrixlöser gewählt und die Diskretisierung der kontinuierlichen Simulationszeit in n Zeitschritte festgelegt.

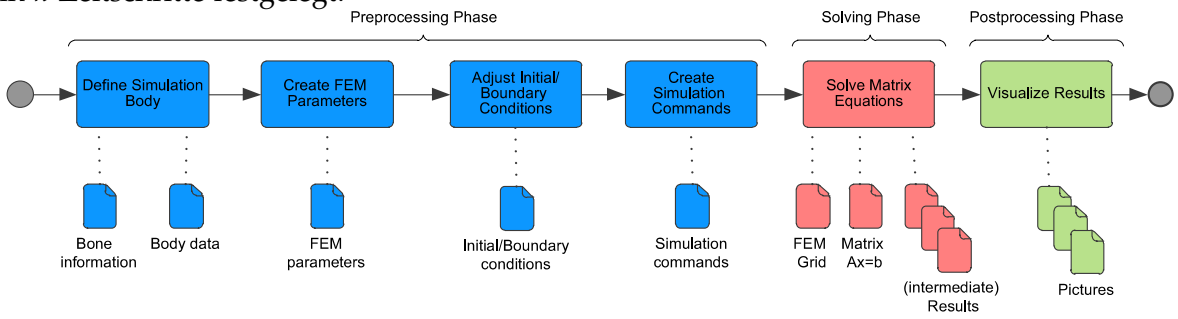

Abbildung 4.2.: Der Pandas Workflow [RRS+11]

Danach folgt die Solving Phase. In der Solving Phase werden die Eingabedaten genutzt, um Matrixgleichungen aufzustellen und zu lösen und um darauf aufbauend die Zwischenergebnisse bzw. Endergebnisse der Simulation zu berechnen. Für jeden Zeitschritt wird ein FEM Gitter (vgl. Abschnitt 2.4.4) erzeugt bzw. angepasst, welches die Basis zur Aufstellung der benötigten Matrixgleichungen $A_i x_i = b_i$ bildet. Diese Gleichungen werden anschließend gelöst. Solch ein Gitter besteht aus einer großen Anzahl an Elementen sowie Informationen, die die Beziehung zwischen den Elementen beschreiben. In der Regel werden diese Informationen sowie die Matrix A und die Vektoren x_i und b_i im Hauptspeicher zwischengespeichert. Jedoch besteht auch die Möglichkeit diese Information persistent in einer Datenbank zu speichern. Wurde der letzte Zeitschritt berechnet, werden die Endergebnisse in einer CSV-Datei gespeichert.

In der anschließenden Postprocessing Phase transformiert die die Aktivität *Visualize Results* die Endergebnisse in ein für ein Visualisierungswerkzeug geeignetes Format. Dadurch hat der Wissenschaftler die Möglichkeit die Ergebnisse der Simulation grafisch zu analysieren bzw. zu validieren.

4.3. Pandas-Matlab Kopplung

Dieser Abschnitt basiert auf [Dor11] und [Ari12]. Im Folgenden wird nun noch eine Multi-* Simulation (vgl. Abschnitt 2.4.2) vorgestellt. Auch bei diesem Szenario werden Knochenstrukturänderungen simuliert. Jedoch werden Pandas und Matlab gekoppelt, um gemeinsam

eine Simulation auszuführen. Pandas berechnet wiederum die biomechanische Belastung auf einen Knochen. Matlab hingegen führt eine Berechnung im Bereich der Systembiologie auf zellularer Ebene durch. Demzufolge werden bei dieser Simulation unterschiedliche Skalen genutzt: Zum einen rechnet Pandas auf einer größeren Raumskala als Matlab und zum anderen verwendet Matlab eine feinere Zeitskala als Pandas.

Abbildung 4.3 zeigt die beteiligten Workflows: Der **Pandas-Bone** führt die Pandas-Simulation aus und entspricht im Wesentlichen dem im Abschnitt 4.2 vorgestellten Simulationsworkflow (in dieser Grafik wurden die einzelnen Phasen jeweils in einer Aktivität zusammengefasst). Der **Matlab-Bone** führt die Matlab-Berechnung aus. Der **Data-Manager** wird benötigt, um die Kommunikation und den Datenaustausch zwischen Pandas-Bone und Matlab-Bone zu koordinieren. Im Folgenden wird nun der Ablauf der Simulation erläutert.

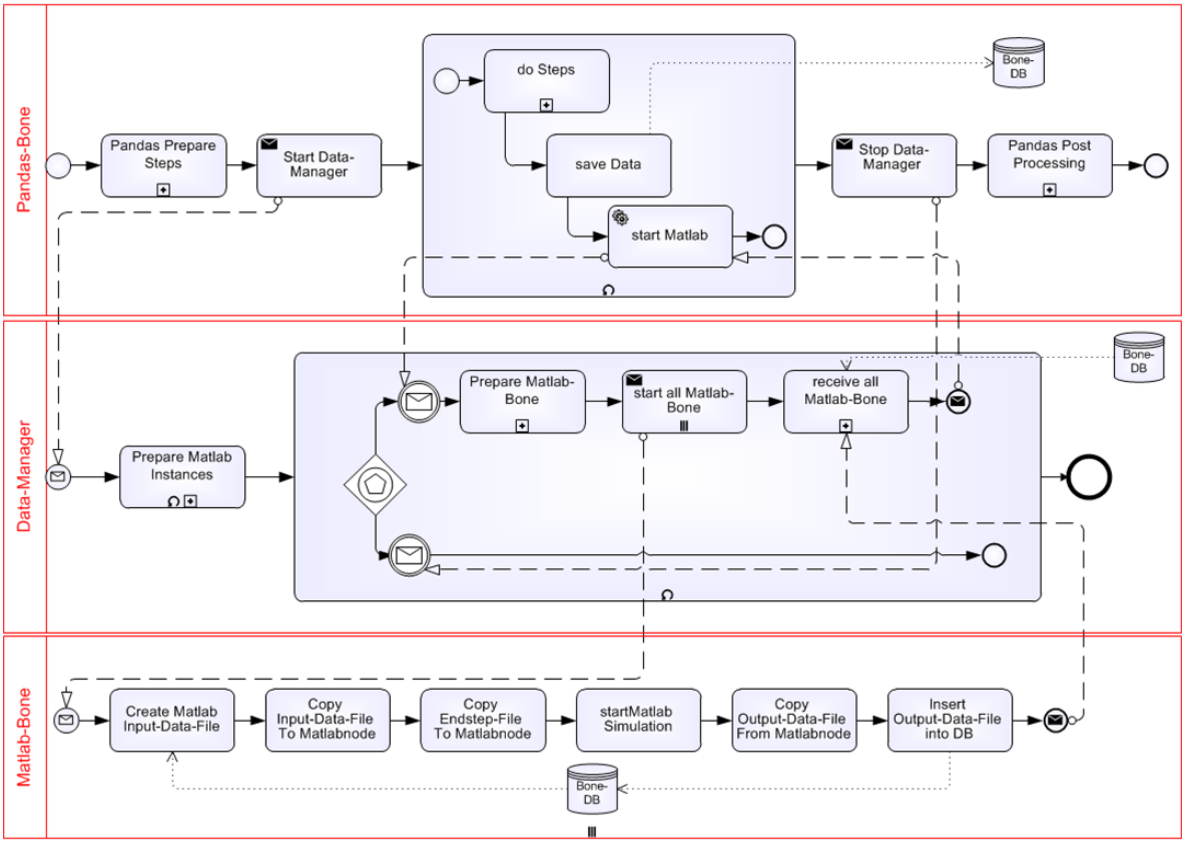

Abbildung 4.3.: Die Pandas-Matlab Kopplung [Dor11]

Die Aktivität *Pandas Prepare Steps* führt die in Abschnitt 4.2 erläuterten Aktivitäten der Preprocessing Phase aus. Nach Abschluss dieser Phase wird der Data-Manager Workflow über die Aktivität *Start Data-Manager* gestartet.

Bevor eine Matlab-Simulation ausgeführt werden kann, muss eine neue Simulationsinstanz erzeugt werden. Zu beachten ist, dass mehrere Matlab-Instanzen, die auf unterschiedlichen Rechnern laufen können, parallel eine Simulation durchführen. Aus diesem Grund startet die Aktivität *Prepare Matlab Instances* eine Schleife. Innerhalb dieser Schleife werden die einzelnen Matlab-Instanzen vorbereitet. So wird eine Simulations-ID erzeugt, das zugehörige Arbeitsverzeichniss auf dem zugehörigen Rechner wird erstellt und die Simulationsdaten werden dorthin kopiert und anschließend entpackt. Die Simulationsdaten umfassen unter anderem die M-Datei (eine Liste mit Matlab-Befehlen) sowie weitere Initialisierungsdaten. Die Schleife wird so oft ausgeführt, wie Matlab-Instanzen gebildet werden sollen. Anschließend wartet der Data Manager Workflow. Der Pandas-Bone gerät währenddessen in die Solving Phase. In dieser Phase wird eine bestimmte Anzahl an Simulationsschritten berechnet. Die Ergebnisse werden in einer Datenbank, der sogenannten PandasDB, gespeichert. Dazu zählen Informationen über die Gitterelemente sowie die darin enthaltenen Gausspunkte. Anschließend wird der Matlab-Bone Workflow über den Data-Manager angetriggert. Bevor jedoch die einzelnen Matlab-Instanzen Berechnungen durchführen können, müssen die Eingabedaten aus der PandasDB gelesen und auf die einzelnen Rechner verteilt werden. Der Simulationsraum muss also aufgeteilt werden.

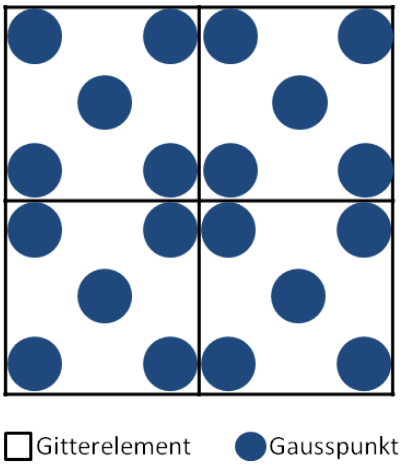

Abbildung 4.4.: Exemplarische Verteilung der Gausspunkte (in Anlehnung an [Ari12])

Dazu wird zunächst die Größe der Datenmenge bestimmt (*Prepare Matlab Bone*). Aus den zuvor in der PandasDB gespeicherten Daten wird die Anzahl der Elemente sowie die Anzahl der Gausspunkte in diesen Elementen bestimmt. Pro Gausspunkt werden mehrere Variablenwerte in der PandasDB gespeichert. Die Gausspunkte bzw. die Informationen über die einzelnen Gausspunkte werden auf die einzelnen Matlab-Instanzen aufgeteilt. Wie viele Gausspunkte *eine* Matlab-Instanz berechnen muss, hängt von der Anzahl der Elemente bzw. Gausspunkte in Bezug auf die Anzahl der Matlab-Instanzen ab. Abbildung 4.4 zeigt eine exemplarische Verteilung der Gausspunkte in den Gitterelementen. Als geometrische Form

wurde in diesem Fall Viereck gewählt. Anschließend wird der Matlab-Bone so oft gestartet, wie es Matlab-Instanzen gibt (*start all Matlab Bone*).

Der Matlab-Bone startet die eigentliche Berechnung. Zuvor müssen jedoch noch die Eingabedaten kopiert werden. Die Aktivität *Create Matlab Input-Data-File* erzeugt aus der PandasDB die benötigte Eingabedatei und speichert diese auf dem Pandas-Rechner. In dieser CSV-Datei befinden sich die Informationen über die einzelnen Gausspunkte, die die jeweilige Matlab-Instanz berechnen muss. Dabei entspricht eine Zeile genau einem Gausspunkt. Die Aktivität *Copy Input-Data-File To Matlabnode* kopiert dann diese Datei von dem Pandas-Rechner auf den entsprechenden Matlab-Rechner. Zusätzlich wird noch eine Endstep-Datei durch die Aktivität *Copy Endstep-File To Matlabnode* kopiert. Mithilfe der Aktivität *start Matlab Simulation* wird die eigentliche Berechnung gestartet. Nach deren Abschluss werden die Ergebnisse der jeweiligen Matlab-Instanz wiederum gespeichert und auf den Pandas-Rechner zurückkopiert (*Copy Output-Data-File From Matlab Node*). Die Ergebnisdaten werden gemerged und wiederum in der PandasDB gespeichert (*Insert Output-Data-File into DB*, *receive all Matlab-Bone*). Abbildung 4.5 zeigt exemplarisch den Dateitransfer zwischen den beteiligten Rechnern.

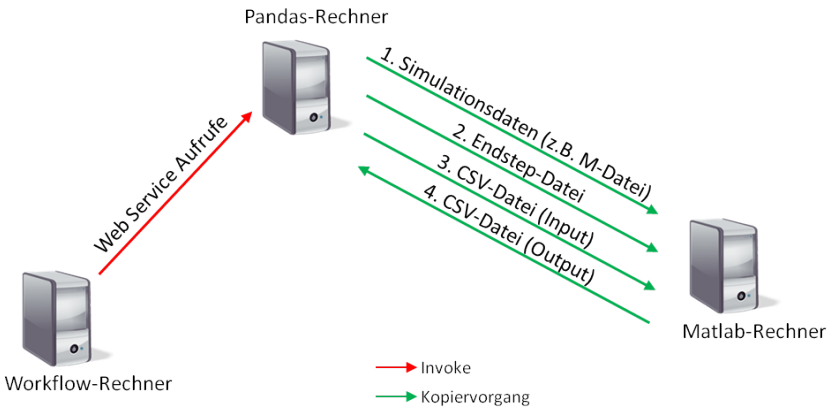

Abbildung 4.5.: Dateitransfer in der Pandas-Matlab Kopplung (eine Matlab-Instanz)

Danach wird im PandasBone Workflow überprüft, ob die Simulation beendet werden kann. Falls der n-te Zeitschritt berechnet wurde und die Ergebnisse zufriedenstellend sind, wird über die Aktivität *Stop Data Manager* der Data-Manager Workflow gestoppt. Anschließend geht der Pandas-Bone in die Postprocessing-Phase über, wie sie in Abschnitt 4.2 beschrieben ist.

5. Bestandsaufnahme

Das SIMPL-Rahmenwerk wurde bisher in weiten Teilen prototypisch umgesetzt und wird sukzessive weiterentwickelt [SIM]. Die Implementierung ist in ein Workflow Management System eingebunden. Dieses basiert auf BPEL [JE07], sowie auf der BPEL-Engine Apache ODE (Orchestration Director Engine) [Foua] und dem Modellierungstool Eclipse BPEL Designer [Foub]. Im Folgenden werden nun der erweiterte Eclipse BPEL Designer, das Resource Management sowie die Umsetzung des SIMPL Cores erläutert. In diesem Kapitel wird eine nicht veröffentlichte SIMPL-Version betrachtet. Diese basiert auf Version 1.0.8, beinhaltet aber zusätzlich noch Container Referenzen (vgl. Abschnitte 3.3 und 3.5). Die in Abschnitt 3.7 erläuterten Datenmanagementpatterns wurden bisher nur konzeptionell entwickelt und werden deshalb in Abschnitt 5.4 gesondert betrachtet.

5.1. Der erweiterte Eclipse BPEL Designer

Der Eclipse BPEL Designer wurde im Zuge der Implementierung des SIMPL-Rahmenwerks erweitert. Abbildung 5.1 zeigt die resultierende Oberfläche. Exemplarisch wurde eine RetrieveData Aktivität modelliert. Die Palette (grüne Markierung) beinhaltet die neuen DM Aktivitäten. Diese sind:

- QueryData
- IssueCommand
- RetrieveData
- WriteDataBack
- TransferData

Die Funktionen der IssueCommand, RetrieveData und WriteDataBack Aktivitäten werden in Abschnitt 3.3 erläutert. Aus diesem Grund werden nun nur noch die übrigen Aktivitäten beschrieben.

Mithilfe einer **QueryData** Aktivität können Daten extrahiert und in einem referenzierten externen Datencontainer abgelegt werden. Dieser Container muss jedoch, im Gegensatz zur TransferData Aktivität, in der selben Datenquelle vorliegen. Als Eingabeparameter werden dazu eine *data source reference variable* sowie eine *data container reference variable* benötigt. Die zu extrahierenden Daten werden über einen *DM command* (im weiteren Verlauf der Arbeit auch als Statement bezeichnet) spezifiziert.

Die **TransferData** Aktivität bietet die Möglichkeit Daten aus einer Datenquelle zu extrahieren und diese wiederum in einer anderen Datenquelle (bzw. eine Datensenke) zu speichern. Dazu werden folgende Eingabeparameter benötigt: Die beiden *data source reference variables* referenzieren die beiden Datenquellensysteme (Datenquelle und Datensenke), zwischen denen die Daten ausgetauscht werden sollen. Der Container, in dem die Daten in der Datensenke gespeichert werden sollen, wird mithilfe einer *data container reference variable* referenziert. Ein Statement spezifiziert wiederum die eigentlichen Daten, die aus der Datenquelle extrahiert und zur Datensenke übertragen werden sollen.

Die Parametrisierung der einzelnen Aktivitäten geschieht in der Property-View (rote Markierung). Dort können entsprechende Variablen ausgewählt und Statements spezifiziert werden. Die vom Nutzer erzeugten Variablen werden im Bereich Variables angezeigt (blaue Markierung). Dort können auch neue Variablen erstellt werden. Jede der fünf DM Aktivitäten nutzt die generischen Operationen des SIMPL Cores, um die eigentliche Operation auf der Datenquelle auszuführen.

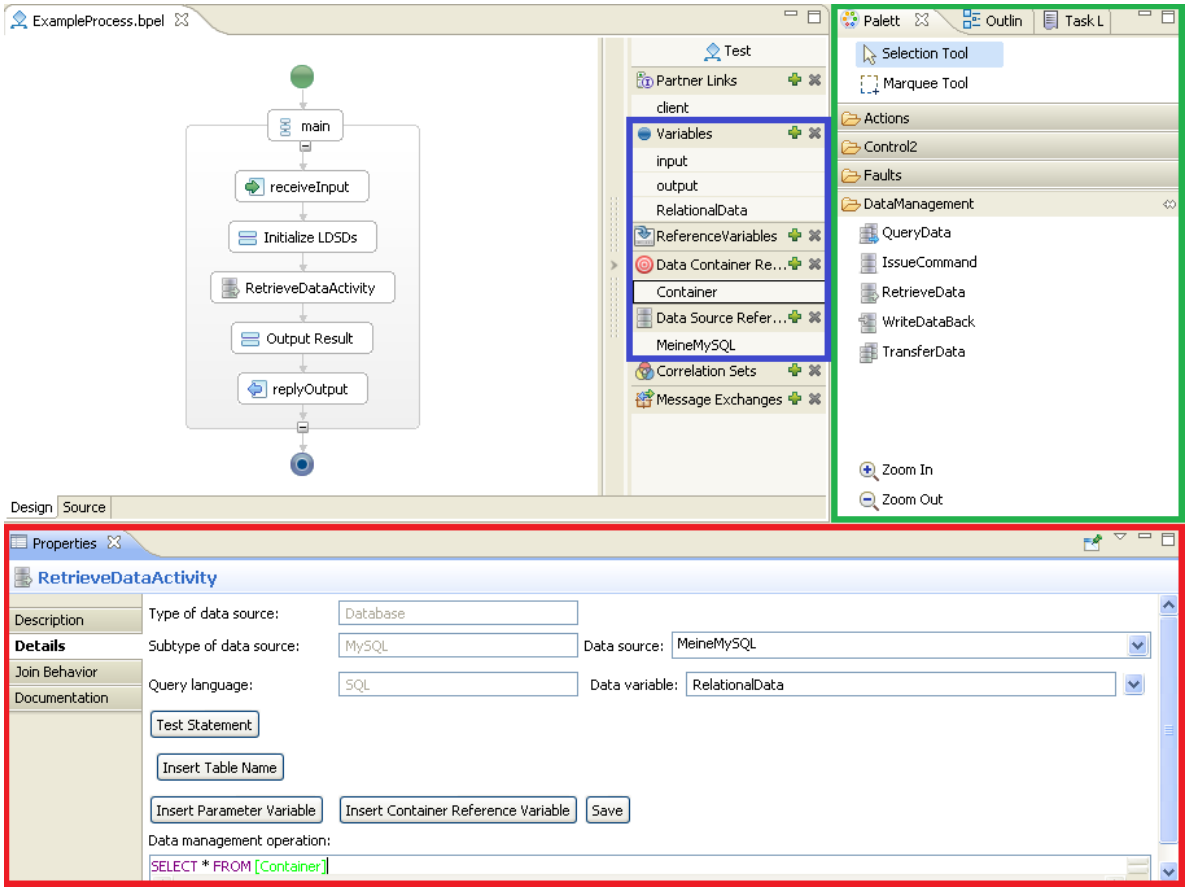

Abbildung 5.1.: Der erweiterte Eclipse BPEL Designer

5.2. Resource Management

In Abschnitt 3.5 wird auf die Aufgabe des Resource Managements eingegangen. In der bestehenden Implementierung läuft das Resource Management als Web Service (vgl. Abschnitt 2.2.3) in einer Axis2 Laufzeitumgebung. Zusätzlich gibt es die Möglichkeit direkt über das Java Interface auf das Resource Management zuzugreifen. Beim SIMPL Core kann der Nutzer einstellen, welche der beiden Optionen genommen wird. Der erweiterte Eclipse BPEL Designer nutzt immer den Web Service zur Kommunikation mit dem Resource Management. Die eigentlichen Daten, die vom Resource Management verwaltet werden, sind in einer PostgreSQL Datenbank [Gro] gespeichert. Bei der Initialisierung des Resource Managements werden zwei Schemata erzeugt: *Simpl_Resources* und *Simpl_Definitions*. In diesen Schemata werden die folgenden Tabellen erzeugt:

Simpl_Resources:

- **Connectors(**
  Id, Dataconverter_Id, Name, Input_Datatype, Output_Datatype, Implementation, Properties_Description**)**
- **Dataconverters(**
  Id, Name, Connector_Input_Datatype, Connector_Output_Datatype, Workflow_Dataformat, Direction_Output_Workflow, Direction_Workflow_Input, Implementation, XML_Schema**)**
- **Datasources(**
  Id, Connector_Id, Logical_Name, Security_Username, Security_Password, Interface_Description, Properties_Description, Connector_Properties_Description, Datacontainer_Reference_Type**)**
- **Datacontainers**
  Id, Datasource_Id, Logical_Name, Local_Identifier**)**
- **Datatransformationservices(**
  Id, Name, Input_Dataformat, Output_Dataformat, Direction_Input_Output, Direction_Output_Input, Implementation**)**
- **StrategyPlugins(**
  Id, Name, Implementation**)**

Simpl_Definitions:

- **DataContainer_Reference_Types(**
  Id, Name, XSD_Type**)**
- **DataSource_Reference_Types(**
  Id, Name, XSD_Type**)**
- **Languages(**
  Id, Name, Statement_Description**)**

- **Workflow_Dataformat_Types**
  Id, Name, XSD_Type**)**

Die ersten fünf Tabellen speichern im Wesentlichen die in Abschnitt 3.5 erläuterten Metadaten. Zum besseren Verständnis wird auf einige Attribute nun noch einmal eingegangen und der Zusammenhang zwischen den einzelnen Tabellen beschrieben.

Die Tabelle **Connectors** beinhaltet Metadaten über verfügbare Konnektoren. Die Attribute *InputDataType* und *OutputDataType* beschreiben das Ein- und Ausgabeformat des Konnektors. Ein Verweis auf die konkrete Implementierung wird unter dem Attribut *Implementation* gespeichert. Das Attribut *Properties_Description* beinhaltet Eigenschaften, die der jeweilige Konnektor aufweist. Das Attribut *Dataconverter_Id* verweist auf einen geeigneten Konverter in der Tabelle **Dataconverters**. Dort sind wiederum Metadaten über verfügbare Konverter gespeichert. Das Attribut *Workflow_Dataformat* enthält den Namen einer Formatbeschreibung. Diese legt das Format zum Austausch mit dem Workflow fest. Das zugehörige XML Schema wird unter dem Attribut *XML_Schema* gespeichert. Die Attribute *Connector_Input_Datatype* und *Connector_Output_Datatype* beschreiben auch hier wieder Datenformate. Bei einem für einen Konnektor geeigneten Konverter muss das Attribut *Connector_Input_Datatype* dem Attribut *InputDataType* in der Tabelle **Connectors** entsprechen. Dasselbe gilt für das Attribut *Connector_Output_Datatype* in Bezug auf Attribut *OutputDataType* in der Tabelle **Connectors**. Das Attribut *Direction_Output_Workflow* enthält einen Wahrheitswert. Dieser besagt, ob der Konverter Daten, die im *Connector_Output_Dataformat* vorliegen, in das Workflow Datenformat überführen kann. Ob eine Transformation in Rückrichtung, also vom Workflow Datenformat in das *Connector_Input_Dataformat* möglich ist, wird mithilfe des Attributs *Direction_Workflow_Input* festgehalten. Das Attribut *Implementation* verweist auf die konkrete Implementierung des jeweiligen Konverters.

Metadaten über verfügbare Datenquellen werden in der Tabelle **Datasources** gespeichert. Das Attribut *Connector_Id* verweist auf einen geeigneten Konnektor für die jeweilige Datenquelle. Der logische Name einer Datenquelle muss eindeutig sein. Über diesen kann dann die Datenquelle im Workflow referenziert werden. Das Attribut *Datacontainer_Reference_Type* referenziert ein XML Schema Typ in der Tabelle **Datacontainer_Reference_Types**. Dieser legt fest, wie ein konkrete Identifikator bei der jeweiligen Datenquelle aussieht. Die restlichen Attribute speichern Authentifizierungsinformationen, Eigenschaften über die Datenquelle sowie Eigenschaften, die ein geeigneter Konnektor aufweisen muss (vgl. Abschnitt 3.5).

In den Abschnitten 3.3 und 3.5 wird erläutert, dass jede Datenquelle verschiedene Container (*data container*) verwaltet. WriteDataBack, QueryData und TransferData Aktivitäten erhalten als Eingabeparameter jeweils eine *data container reference variable*. Diese Variable referenziert den Ort, an dem Daten gespeichert bzw. bereitgestellt werden sollen. Metadaten über die einzelnen Container werden in der Tabelle **Datacontainers** gespeichert. Jeder im Resource Management registrierte Container kann über seinen logischen Namen, der mithilfe des Attributs *Logical_Name* festgehalten wird, referenziert werden. Ein Container gehört immer zu *einer* Datenquelle. Die zugehörige Datenquelle wird mithilfe des Attributs *Datasource_Id* referenziert. Unter dem Attribut *Local_Identifier* wird der konkrete Identifikator, mit dem der Container innerhalb der Datenquelle referenziert werden kann, gespeichert. Listing 5.2 zeigt exemplarisch solch einen konkreten Identifikator.

Mithilfe von Data Transformation Services können Daten in andere Formate transformiert werden. So ist es möglich Daten aus einem Workflow Datenformat in ein anderes Workflow Datenformat zu transformieren. Metadaten über Data Transformation Services werden in der Tabelle **Datatransformationservices** gespeichert. Der Name des Dienstes wird mithilfe des Attributs *Name* festgehalten. Die Attribute *Input_Dataformat* und *Output_Dataformat* beschreiben wiederum zwei Datenformate. Bietet der Data Transformation Service die Möglichkeit Daten, die im *Input_Dataformat* vorliegen, in das *Output_Dataformat* zu transformieren, wird mithilfe des Attributs *Direction_Input_Output* ein positiver Wahrheitswert gespeichert. Wird solch eine Transformation hingegen nicht unterstützt, wird ein negativer Wahrheitswert gespeichert. Das Attribut *Direction_Output_Input* enthält ebenfalls einen Wahrheitswert. Dieser besagt, ob eine Transformation in die Rückrichtung, also vom *Output_Dataformat* in das *Input_Dataformat* möglich ist.

Werden Datenquellen erst zur Laufzeit gesucht und eingebunden, bestimmt die Suchstrategie, welche der in Frage kommenden Datenquellen ausgewählt wird. Die Tabelle **StrategyPlugins** enthält Verweise auf implementierte Suchstrategien. Die Tabelle **DataSource_Reference_Types** enthält XML Schema Typen. Diese Typen definieren die Struktur einer Referenz auf eine Datenquelle. Die Tabelle **DataContainer_Reference_Types** enthält ebenfalls XML Schema Typen. Diese definieren jedoch, wie die konkreten Identifikatoren bei den einzelnen Typen von Datenquellen aussehen. Abfragesprachen, wie z.B. SQL, die vom SIMPL-Rahmenwerk unterstützt werden, werden in der Tabelle **Languages** gespeichert. Das Attribut *Statement_Description* enthält ein XML-Dokument. Dieses beschreibt, wie Ausdrücke der jeweiligen Sprache aussehen dürfen. In der Tabelle **Workflow_Dataformat_Types** werden XML Schema Typen gespeichert, die die verschiedenen Workflow Datenformate definieren.

	id [PK] serial	name character varying(255)	xsd_type xml
1	1	DataContainerReferenceType	<xsd:complexType name="DataContainerReferenceType"></xsd:complexType>
2	2	LocalDataContainerReferenceType	<xsd:complexType name="LocalDataContainerReferenceType"> <xsd:complexContent> <xsd:extension base="simpl:Da
3	3	LogicalDataContainerReferenceType	<xsd:complexType name="LogicalDataContainerReferenceType"> <xsd:complexContent> <xsd:extension base="simpl:I
4	4	RelationalDatabaseDataContainerReferenceType	<xsd:complexType name="RelationalDatabaseDataContainerReferenceType"> <xsd:complexContent> <xsd:extensio
5	5	XMLDatabaseDataContainerReferenceType	<xsd:complexType name="XMLDatabaseDataContainerReferenceType"> <xsd:complexContent> <xsd:extension bas
6	6	WindowsLocalDataContainerReferenceType	<xsd:complexType name="WindowsLocalDataContainerReferenceType"> <xsd:complexContent> <xsd:extension bas

Abbildung 5.2.: Die Tabelle DataContainer_Reference_Types

Gerade wurde erläutert, dass in der Tabelle **Datacontainers** Metadaten über registrierte Container und in der Tabelle **DataContainer_Reference_Types** XML Schema Typen für Referenzen auf Container gespeichert werden. Um einen Container aus einem Workflow heraus zu referenzieren, muss dieser nicht notwendigerweise im Resource Management registriert sein. Der Modellierer kann bei der Parametrisierung einer DM Aktivität auch eine *data container reference variable* angeben, die direkt den lokalen Bezeichner beinhaltet. Dies soll nun an einem Beispiel verdeutlicht werden. Abbildung 5.2 zeigt exemplarisch den Inhalt der Tabelle **DataContainer_Reference_Types**. Ist ein Container im Resource Management registriert und soll dieser über seinen logischen Namen referenziert werden, muss der Modellierer eine Variable vom Typ *LogicalDataContainerReferenceType* erzeugen. Dieser Variablen wird dann das in Listing 5.1 dargestellte XML-Fragment zugewiesen. Dieses

XML-Fragment beinhaltet dann lediglich den logischen Namen des Containers, alle anderen wichtigen Informationen, also die lokalen Bezeichner und die Referenz auf eine Datenquelle, werden im Resource Management abgelegt. Möchte der Modellierer hingegen lieber direkt den lokalen Bezeichner des Containers angeben, muss dieser z.B. bei einer relationalen Datenbank eine Variable vom Typ *RelationalDatabaseDataContainerReferenceType* erzeugen. Dieser Variablen muss dann das in Listing 5.2 dargestellte XML-Fragment, das den lokalen Bezeichner des Containers sowie den Namen einer zugehörigen *data source reference variable* enthält, zugewiesen werden.

Listing 5.1 Referenzierung eines im Resource Management registrierten Containers

```
<LogicalDataContainerReferenceType stringPattern="logicalIdentifier"
    xmlns:simpl="http://www.example.org/simpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/simpl simpl.xsd">
  <logicalIdentifier>ExampleContainer</logicalIdentifier>
</LogicalDataContainerReferenceType>
```

Listing 5.2 Referenzierung eines Containers über den lokalen Bezeichner

```
<RelationalDatabaseDataContainerReferenceType stringPattern="schema.table"
    xmlns:simpl="http://www.example.org/simpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/simpl simpl.xsd">
  <schema>public</schema>
  <table>personen</table>
  <dataSourceReferenceVariable>PostgreSQL</dataSourceReferenceVariable>
</RelationalDatabaseDataContainerReferenceType>
```

5.3. SIMPL Core

Der SIMPL Core bildet zusammen mit dem Resource Management das Herzstück des SIMPL-Rahmenwerks. Der SIMPL Core wurde ebenfalls wie das Resource Management als Web Service (vgl. Abschnitt 2.2.3) implementiert. Zusätzlich ist ein Zugriff über das Java Interface möglich. Somit kann Apache ODE wahlweise über den Web Service oder das Java Interface auf den SIMPL Core zugreifen. Der SIMPL Core bietet die folgenden öffentlichen Operationen:

- IssueCommand
- RetrieveData
- WriteDataBack
- QueryData
- GetMetaData

Die in Abschnitt 5.1 erläuterte TransferData Aktivität baut auf diesen Operationen auf. Eine RetrieveData Operation extrahiert die spezifizierten Daten aus der Datenquelle. Eine WriteDataBack Operation speichert diese Daten dann wiederum in der Datensenke. Zusätzlich bietet der SIMPL Core noch die Operation GetMetaData. Diese Operation stellt Metadaten über Datenquellen zur Verfügung. Bei relationalen Datenbanken sind dies die Strukturen der einzelnen Tabellen, die in dieser Datenbank gespeichert sind. Bei einem Dateisystem werden alle Befehle, die auf Kommandozeilenebene ausgeführt werden können, zurückgegeben. Die Operationen wurden im generischen SIMPL Core jeweils auf zwei Arten implementiert. Der Unterschied zwischen den beiden Implementierungen soll nun anhand der RetrieveData Operation verdeutlicht werden. Wird eine Datenquelle über einen logischen Namen referenziert, können über diesen alle benötigten Informationen im Resource Management abgefragt werden. Das übergebene Statement kann dann auf der referenzierten Quelle ausgeführt werden. Für diesen Zweck bietet der SIMPL Core die Operation *retrieveDataByDataSourceName*. Die Operation *retrieveDataByDataSource* wird hingegen verwendet, wenn eine Datenquelle nicht über ihren logischen Namen referenziert wird. In diesem Fall wird zuerst überprüft, ob die für ein Late Binding benötigten Informationen vorliegen. Eine WS-Policy Beschreibung [VOH+07] spezifiziert die Eigenschaften, die eine in Frage kommende Datenquelle aufweisen muss. Welche Datenquelle schließlich ausgewählt wird, wird durch die Suchstrategie bestimmt. Die WS-Policy Beschreibung und die Suchstrategie werden der Resource Discovery Komponente übergeben. Wurde eine passende Datenquelle gefunden, kann anschließend das spezifizierte Statement auf dieser ausgeführt werden.

Listing 5.3 Interface Beschreibung eines Konnektors [SIM]

```
public interface Connector<S, T> {

        public boolean issueCommand(DataSource dataSource, String statement)
                        throws ConnectionException;

        public T retrieveData(DataSource dataSource, String statement)
                        throws ConnectionException;

        public boolean writeDataBack(DataSource dataSource, S data, String target)
                        throws ConnectionException;

        public boolean queryData(DataSource dataSource, String statement,
                        String target) throws ConnectionException;

        public DataObject getMetaData(DataSource dataSource, String filter)
                        throws ConnectionException;

        public boolean createTarget(DataSource dataSource, DataObject dataObject,
                        String target) throws ConnectionException;
}
```

Dazu wird für spezifische Datenquellen ein geeigneter Konnektor benötigt, der die generischen SIMPL Core Operationen implementiert. Metadaten über die einzelnen Konnektoren

sind im Resource Management gespeichert. Bisher unterstützt das SIMPL-Rahmenwerk die folgenden Typen von Datenquellen:

- Relationale Datenbanken
- XML-Datenbanken
- Windows-Dateisystem
- ein Sensornetz-Gateway (TinyDB)

Die einzelnen Konnektoren implementieren das in Listing 5.3 gezeigte Interface. Die generischen Variablentypen *S* und *T* legen das Ein- bzw. Ausgabeformat des Konnektors fest. Die Konnektoren für relationale Datenbanken erwarten z.B. als Eingabe eine Liste mit Strings (SQL Konstrukte), um Daten mit der WriteDataBack Operation in einer Datenbank zu speichern. Als Ausgabe erzeugt die RetrieveData Operation ein RDBResult (JDBC ResultSet + zugehörige Metdaten).

Eine DM Aktivität übergibt bzw. erwartet Daten in einem XML-basierten Format. Konverter implementieren die benötigten Transformationen von den Formaten eines Konnektors zu diesen XML-basierten Formaten und umgekehrt. Das vom Workflow gewünschte Format (im weiteren Verlauf dieser Arbeit als Workflow Datenformat bezeichnet) ist als XML Schema Definition im Resource Management hinterlegt.

Listing 5.4 Interface Beschreibung eines Konverters [SIM]

```
public interface DataConverter<S, T> {

        public DataObject toSDO(S data);

        public T fromSDO(DataObject data);

        public DataObject getSDO();

        public String getDataFormat();
}
```

Die zugehörige Interface-Beschreibung ist in Listing 5.4 dargestellt. Als Datenstruktur werden *Service Data Objects* [BBB+05] verwendet. Dadurch können Daten aus unterschiedlichen Quellen einfacher gehandhabt werden. Die generischen Variablentypen *S* und *T* legen das Ein- bzw. Ausgabeformat des Konverters fest. Zur Ausführung einer RetrieveData oder WriteDataBack Operation auf einer Datenquelle wird ein Konnektor sowie ein passender Konverter benötigt. Das Ausgabeformat des Konnektors muss dem Eingabeformat des Konverters entsprechen. Dasselbe gilt für die andere Richtung.

Die *toSDO* Funktion wird genutzt, um Daten, die im Ausgabeformat eines Konnektors vorliegen, in ein SDO zu überführen. Diese Struktur kann wiederum ohne großen Aufwand in eine XML Darstellung überführt und dem Workflow zur Verfügung gestellt werden. Mithilfe der *fromSDO* Funktion kann ein SDO in das Eingabeformat eines Konnektors überführt werden.

5.4. Datenmanagementpatterns

Die bisherige Implementierung des SIMPL-Rahmenwerks unterstützt Datenmanagementpatterns (vgl. Abschnitt 3.7) noch nicht. Eine Umsetzung fand bisher nur auf konzeptioneller Ebene statt. Dieser Abschnitt nutzt [Ari12] und [RSRM] als Basis. In diesem Abschnitt werden zuerst die bisher formalisierten Datenmanagementpatterns erläutert. Anschließend wird auf die Architektur des Abbildungsmechanismus eingegangen, bevor zuletzt konkrete Beispiele aus der Pandas Preprocessing- sowie Postprocessing-Phase erläutert werden.

5.4.1. Formalisierte Patterns

Bisher wurden die folgenden Datenmanagementpatterns formalisiert: das *Data Transfer and Transformation Pattern*, das *Data Format Conversion Pattern* sowie das *Data Iteration Pattern*. Zu beachten ist, dass das Data Transfer and Transformation Pattern in verschieden Formen vorkommen kann. Aus diesem Grund werden ebenfalls diese verschiedenen Formen, also das *Container-to-Container Pattern*, das *Data Split Pattern* und das *Data Merge Pattern* erläutert. In Bezug auf die in Abschnitt 3.7 vorgestellte Pattern-Hierarchie gehören das Data Transfer and Transformation Pattern sowie seine Formen und das Data Iteration Pattern zu den Basisdatenmanagementpatterns. Das Data Format Conversion Pattern hingegen ist ein ETL Pattern.

5.4.1.1. Data Transfer and Transformation Pattern

Das **Data Transfer and Transformation Pattern** (DTTP) beschreibt einen generischen Prozess, um Daten aus einer oder aus mehreren Daten*quellen* in eine oder in mehrere Daten*senken* zu transferieren. Abbildung 5.3 stellt dies grafisch dar.

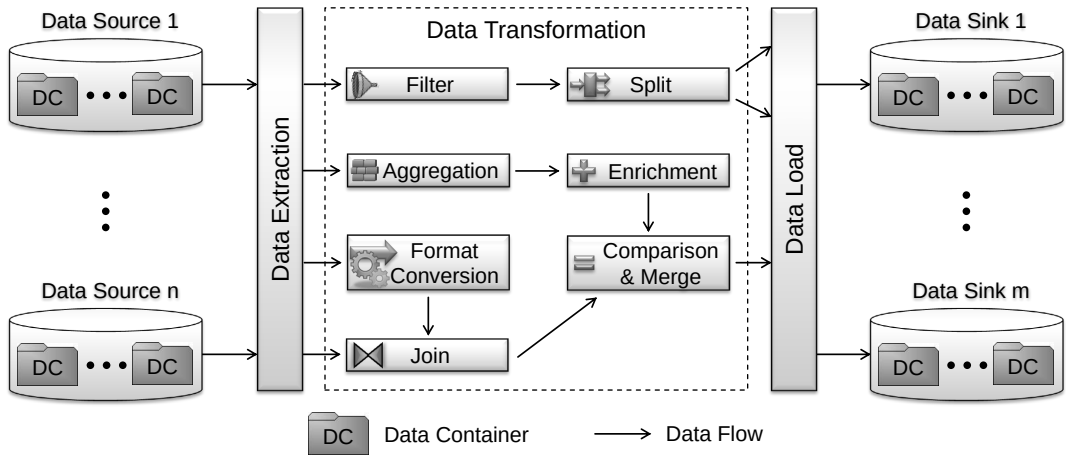

Abbildung 5.3.: Data Transfer and Transformation Pattern [RSRM]

Jede dieser n Datenquellen bzw. m Datensenken verwaltet eine bestimmte Anzahl an Containern. Das Data Transfer and Transformation Pattern extrahiert nun Datensätze aus einem oder aus mehreren Containern einer oder mehrerer Datenquellen und speichert diese dann in einem oder in mehreren Containern in einer oder in mehreren Datensenken. Dieser Transfer geschieht in der Regel durch einen ETL Prozess, der wiederum aus ETL Operationen besteht. So können unter anderem die Operationen Filterung, Aggregation, Datenformatkonvertierung (Data Format Conversion Pattern), Vereinigung oder Verbund eingesetzt werden. Zu beachten ist, dass die Schnittmenge der Datenquellen und -senken nicht zwangsläufig leer sein muss. Eine Datenquelle kann demzufolge auch wiederum eine Datensenke sein.

Container-to-Container Pattern Das Container-to-Container Pattern (C2CP) stellt die erste Form des Data Transfer and Transformation Patterns dar und wird in Abbildung 5.4 grafisch dargestellt. Bei diesem Pattern werden Datensätze aus einem Container einer Datenquelle extrahiert, transformiert und in einem Container einer Datensenke gespeichert. Als ETL Operationen kommen nur unäre Operationen, also Operationen, die nur *eine* Eingabemenge erwarten, in Frage. In Abschnitt 5.4.3 wird auf ein Beispiel eingegangen, das dieses Pattern beinhaltet.

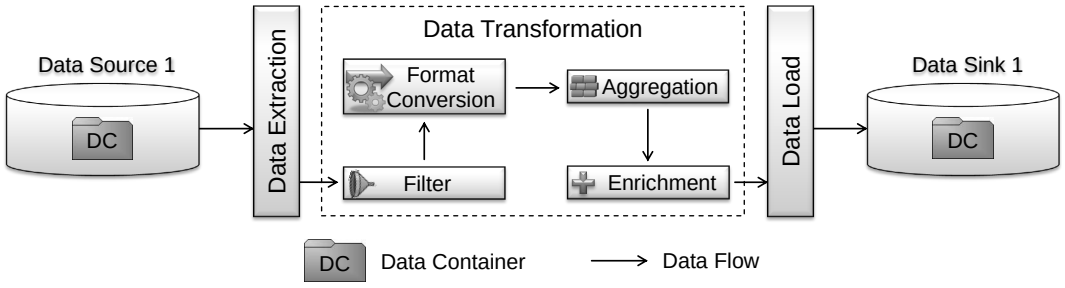

Abbildung 5.4.: Container-to-Container Pattern [RSRM]

Data Split Pattern Eine weitere Form des Data Transfer and Transformation Patterns stellt das Data Split Pattern (DSP) dar. Bei diesem Pattern wird eine Datenmenge S aus genau einem Container extrahiert und in $n > 1$ Teilmengen $T_i \subseteq S$ mit $i \in \{1 \dots n\}$ aufgeteilt. Jede Teilmenge T_i wird in einem Container einer Datensenke gespeichert. Insgesamt gibt es m ($m \geq 1$) Datensenken. Bei diesem Pattern können auch komplexe ETL Operationen eingesetzt werden. Abbildung 5.5 stellt dies grafisch dar.

Dieses Pattern wird oftmals eingesetzt, wenn komplexe Berechnungen ausgeführt werden sollen. Z.B. wird bei der Pandas-Matlab Kopplung (vgl. Abschnitt 4.3) der Simulationsraum aufgeteilt. Es werden mehrere Matlab-Instanzen erzeugt, die jeweils Berechnungen auf unterschiedlichen Datensätzen durchführen. In solch einem Szenario könnte das Data Split Pattern den Simulationsraum aufteilen und den einzelnen Matlab-Instanzen benötigte Daten zur Verfügung stellen. Wird eine vollständige Parallelisierung angestrebt, muss es genau so

viele Datensenken (bzw. dazugehörige Rechner) wie Teilmengen T_i geben. Bei jeder einzelnen Datensenke wird in so einem Fall dann nur ein Container verwendet.

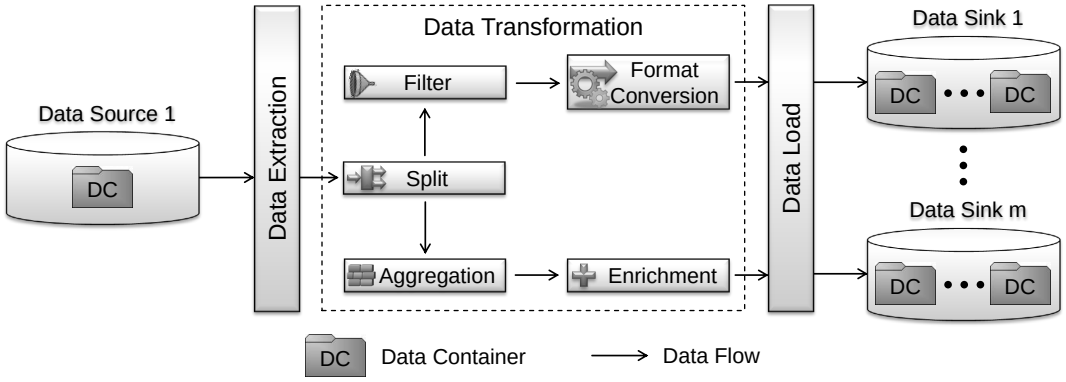

Abbildung 5.5.: Data Split Pattern [RSRM]

Data Merge Pattern Das Data Merge Patten (DMP) stellt die dritte Form des Data Tranfer and Transformation Patterns dar und ist das Gegenstück des Data Split Patterns. Bei diesem Pattern werden n Teilmengen T_i, die in jeweils einem Container in einer oder in mehren Datenquellen gespeichert sind, in einer Datenmenge zusammengefasst. Diese Datenmenge wird dann in einem Container einer Datensenke gespeichert. Auch bei diesem Pattern können komplexe ETL Operation eingesetzt werden. In Abbildung 5.6 wird dieses Pattern grafisch dargestellt.

Genau wie das Data Split Pattern, wird das Data Merge Pattern oftmals bei komplexen Berechnungen eingesetzt. Z.B. werden bei der Pandas-Matlab Kopplung (vgl. Abschnitt 4.3) die Ergebnisdaten der einzelnen Matlab-Instanzen gemerged und in einer Datenbank gespeichert.

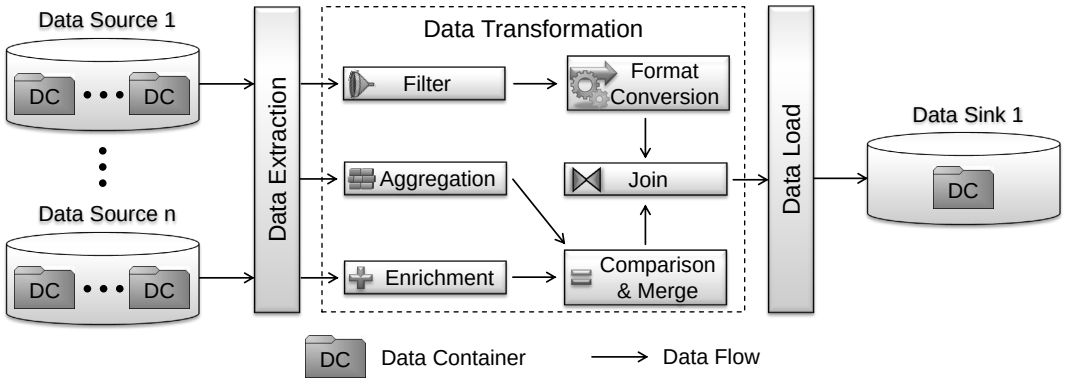

Abbildung 5.6.: Data Merge Pattern [RSRM]

5.4.1.2. Data Format Conversion Pattern

Beim Data Format Conversion Pattern (DFCP) wird ein Container in einer Datenquelle betrachtet. Die Datensätze, die in diesem Container gespeichert sind, weisen ein bestimmtes Datenformat auf. Oftmals ist es nötig, diese Daten in ein anderes Format zu überführen. Das Data Format Conversion Pattern unterstützt eine solche Konvertierung. Daten, die in einem Container vorliegen, werden in ein anderes Format konvertiert und anschließend in einem anderen bzw. in demselben Container gespeichert. Die Konvertierung kann durch die Methoden des Service Bus oder durch einen Script- bzw. Web Service-Aufruf erfolgen. In Abschnitt 5.4.4 wird auf ein Beispiel eingegangen, das dieses Pattern verwendet.

5.4.1.3. Data Iteration Pattern

Ein weiteres Pattern ist das Data Iteration Pattern (DIP). Dieses Pattern erwartet als Eingabe eine Datenmenge S. Bei jeder Iteration wird eine Teilmenge dieser Datenmenge S als Eingabe einer Operation verwendet. Für das Aufteilen bzw. Zusammenfügen der Teilmengen können Data Transfer and Transformation Patterns oder ETL Patterns verwendet werden. Es gibt zwei unterschiedliche Arten der Iteration: die parallele Iteration und die sequentielle Iteration.

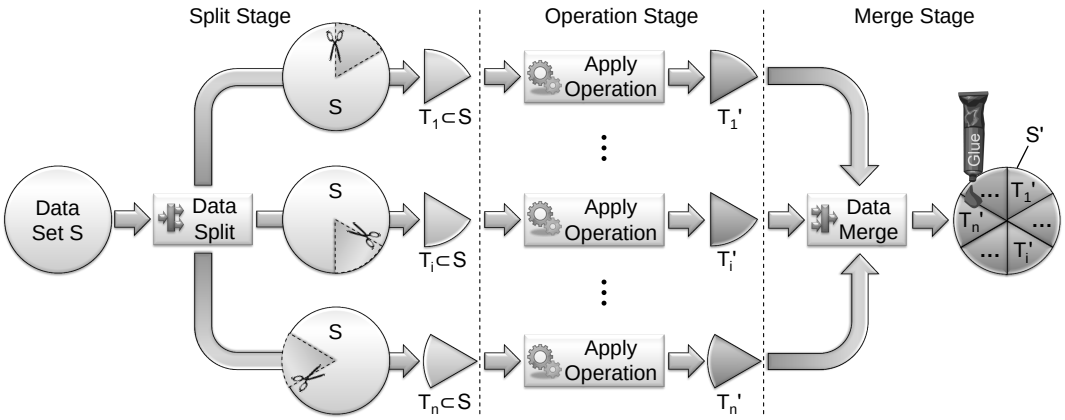

Abbildung 5.7.: Paralleles Data Iteration Pattern [RSRM]

Abbildung 5.7 zeigt das **parallele** Data Iteration Pattern. Ziel ist es eine definierte Operation parallelisiert auf n Teilmengen $T_i \subseteq S$ anzuwenden. Demzufolge kann das parallele Data Iteration Pattern in drei Phasen unterteilt werden: die *Split Stage*, die *Operation Stage* sowie die *Merge Stage*. Die *Split Stage* umfasst ein Data Split Pattern, das die Datenmenge S in n Teilmengen aufteilt. Anschließend wird in der *Oparation Stage* auf jede Teilmenge T_i die definierte Operation angewendet, die die Teilmenge T_i in die Teilmenge $T_{i'}$ überführt. Die *Merge Stage* umfasst wiederum ein Data Merge Pattern, das die geänderten Teilmengen $T_{i'}$ zusammenfasst. Das Resultat ist die Datenmenge S'.

Das parallele Data Iteration Pattern kann z.B. bei der Pandas-Matlab Kopplung eingesetzt werden, wenn der Simulationsraum aufgeteilt und die einzelnen Matlab-Instanzen gestartet werden. In Abschnitt 6.3.1 wird auf dieses Szenario noch genauer eingegangen.

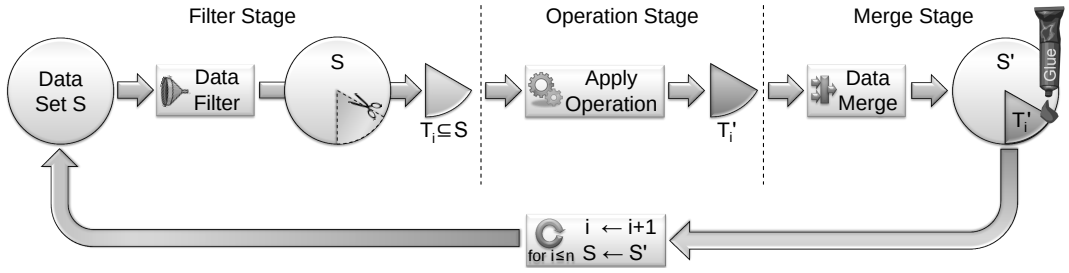

Abbildung 5.8.: Sequentielles Data Iteration Pattern [RSRM]

Das **sequentielle** Data Iteration Pattern wird in Abbildung 5.8 dargestellt. Bei diesem Pattern wird die Datenmenge S *nicht* aufgeteilt und es findet *keine* Parallelisierung statt. Stattdessen werden n Iterationen nacheinander ausgeführt. Bei jeder Iteration wird in der ersten Phase, der *Filter Stage*, über eine Filter-Operation eine Teilmenge $T_i \subseteq S$ bestimmt. Zu beachten ist, dass diese Teilmenge T_i bei jeder Iteration andere Daten umfassen kann. Anschließend wird in der *Operation Stage* die vom Nutzer definierte Operation auf die Teilmenge T_i angewendet. In der *Merge Stage* wird dann die geänderte Teilmenge T_i' in die Datenmenge S integriert. Die resultierende Datenmenge S' dient dann wiederum als Eingabe für die nächste Iteration. Die Anzahl der Iterationen kann auf verschiedene Weise festgelegt werden. So kann der Modellierer die Anzahl der Iterationen bei der Modellierung bestimmen oder aber auch zur Laufzeit berechnen lassen (z.B. indem er einen XPath Ausdruck (vgl. Abschnitt 2.5.2.1) angibt, der bei Beginn der Schleife die Anzahl der Iterationen berechnet) oder ein Abbruchkriterium angibt, das am Ende des Schleifendurchlaufs geprüft wird.

Das sequentielle Data Iteration Pattern kann z.B. in der Pandas Preprocessing-Phase genutzt werden, wenn mehrere Dateien auf den Pandas-Rechner kopiert werden müssen. In Abschnitt 5.4.3 wird auf dieses Szenario noch genauer eingegangen.

5.4.2. Bisherige Architektur des Abbildungsmechanismus

In Abschnitt 3.7 wird erläutert, dass der Pattern Transformer einen Workflow mit Datenmanagementpatterns als Eingabe erhält. Da ein gegebener Workflow als Graph dargestellt werden kann, wird durch diesen Graphen traversiert. Wenn ein Datenmanagementpattern gefunden wird, wird versucht dieses Pattern durch die Anwendung von definierten Transformationsregeln auf ein ausführbares Workflow-Fragment abzubilden. Dafür wird die Kontrollstrategie benötigt. Die Kontrollstrategie definiert, welche Transformationsregeln für ein bestimmtes Datenmanagementpattern in Frage kommen und in welcher Reihenfolge die einzelnen Transformationsregeln auf Anwendbarkeit (Evaluierung des Conditions Parts) geprüft werden.

Im Folgenden werden zwei Algorithmen vorgestellt, die dieses Vorgehen umsetzen. Die vorgestellten Algorithmen basieren auf [VSS+07] und wurden in [Ari12] an die Problematik der Datenmanagementpatterns angepasst. Algorithmus 5.1 zeigt die Traversierung durch den Workflow-Graphen. Solange der Workflow-Graph noch nicht vollständig durchlaufen wurde, wird nach dem nächsten Datenmanagementpattern gesucht (*getNextDMP(wg)*). Das aktuell betrachtete Datenmanagementpattern wird der Prozedur *TransformDMP(dmp)* übergeben.

Algorithmus 5.1 Traversierung durch den Workflow-Graphen (vgl. [VSS+07] und [Ari12])

```
procedure TraverseWG(wg)
    while wg is not fully traversed do
        dmp ←getNextDMP(wg)
        if dmp≠null then
            TransformDMP(dmp)
        end if
    end while
end procedure
```

Diese Prozedur wird in Algorithmus 5.2 dargestellt. Die Prozedur verfolgt das Ziel, das übergebene Datenmanagementpattern auf ein ausführbares Workflow-Fragment abzubilden. Mithilfe der Funktion *getControlStrategy(dmp)* wird die zugehörige Kontrollstrategie geladen und in der Variablen *cs* gespeichert. Anschließend werden die einzelnen Transformationsregeln dieser Kontrollstrategie auf Anwendbarkeit geprüft. Die Funktion *getNextRule* bekommt als Eingabeparameter die Kontrollstrategie übergeben und liefert die jeweils nächste Transformationsregel. Danach wird der Condition Part der aktuell betrachteten Transformationsregel evaluiert. Sollte die Transformationsregel *nicht* anwendbar sein, wird durch die Schleife einfach die nächste Transformationsregel geprüft. Ansonsten wird der Action Part der Transformationsregel mithilfe der Funktion *applyRule(dmp,r)* ausgeführt. Dazu wird ein durch die Transformationsregel definiertes Workflow-Fragment geladen. Dieses Fragment beinhaltet z.B. Platzhalter für bestimmte Parameterwerte. Diese Parameterwerte werden ersetzt und das Ergebnis wird in der Variablen *wf* gespeichert. Anschließend müssen zwei Sonderfälle berücksichtigt werden. Ist eine Transformationsregel vom Regeltyp-2 (vgl. Abschnitt 3.7) enthält das resultierende Workflow-Fragment weitere Datenmanagementpatterns. Auch bei einem Data Iteration Pattern kann die Operation op Datenmanagementpatterns beinhalten. In diesen beiden Fällen wird das Workflow-Fragment der Prozedur *TraverseWG(wf)* übergeben. Auf diese Weise werden rekursiv alle Datenmanagementpatterns aufgelöst. Zuletzt kann das ursprüngliche Datenmanagementpattern durch das parametrisierte Workflow-Fragment ersetzt werden (*replace(dmp, wf)*).

Die Ausführung der Algorithmen wird beendet, wenn der Workflow keine Datenmanagementpatterns mehr beinhaltet. Es kann aber passieren, dass für ein bestimmtes Pattern keine Transformationsregel anwendbar ist. In so einem Fall kommt es zur sogenannten *Escalation*. Nach [Ari12] können dann die folgenden Maßnahmen ergriffen werden, um dennoch ein ausführbares Workflow-Fragment für dieses Pattern zu erhalten:

- Der Nutzer kann ein eigenes Workflow-Fragment modellieren.
- Dem Nutzer können verschiedene Workflow-Fragmente vorgeschlagen werden. Dieser kann dann ein geeignetes Fragment auswählen.
- Das benötigte Workflow-Fragment kann mithilfe eines externen Werkzeugs automatisch generiert werden.

Algorithmus 5.2 Transformation eines Datenmanagementpatterns (vgl. [VSS+07] und [Ari12])

```
procedure TransformDMP(dmp)
    cs ←getControlStrategy(dmp)
    while cs is not finished do
        r ←getNextRule(cs)
        if r isApplicable(dmp,r) then
            wf ←applyRule(dmp, r)
            if r.type=2 ∨ dmp.type=DataIterationPattern then
                TraverseWG(wf)
            end if
            replace(dmp,wf)
            return
        end if
    end while
    Escalation("There is no applicable rule")
end procedure
```

In [Ari12] wird eine Architektur vorgestellt, um das illustrierte Vorgehen umzusetzen. Diese Architektur wird in Abbildung 5.9 dargestellt. Jedoch wird die gerade erläuterte *Escalation* in dieser Architektur nicht berücksichtigt. Im Folgenden werden nun die sieben Komponenten erläutert:

- **Workflow Graph Traverser:** Der Workflow Graph Traverser traversiert durch den Workflow-Graphen und entspricht im Wesentlichen dem Algorithmus *TraverseWG(wg)*. Wenn ein Datenmanagementpattern gefunden wurde, übergibt der Workflow Graph Traverser dieses Pattern der Pattern Transformer Engine. Die Transformation der Datenmanagementpatterns kann entweder zur Modellierungszeit oder zur Laufzeit stattfinden. Bei einer Transformation zur Modellierungszeit wird der Workflow Graph Traverser bzw. der Algorithmus *TraverseWG(wg)* in die GUI, z.B. in den erweiterten Eclipse BPEL Designer (vgl. Abschnitt 5.1) integriert. Bei einer Transformation zur Laufzeit hingegen ist diese Komponente Bestandteil der Execution Engine.
- **Ruleset and Control Strategies:** Mithilfe dieser Komponente werden die Kontrollstrategien sowie Transformationsregeln festgehalten und verwaltet. In Bezug auf obige Algorithmen implementiert diese Komponente die Funktion *getControlStrategy*.

- **Pattern Transformer Engine:** Die Pattern Transformer Engine bildet das Herzstück dieser Architektur und ist für die Transformation der Datenmanagementpatterns in ausführbare Workflow-Fragmente verantwortlich. Diese Komponente entspricht im Wesentlichen dem Algorithmus *TransformDMP(dmp)* und kann in zwei weitere Komponenten unterteilt werden: Die *Control Strategy Engine* verarbeitet die Kontrollstrategie eines bestimmten Datenmanagementpatterns und liefert die jeweils nächste Regel, die evaluiert werden soll. In Bezug auf obige Algorithmen entspricht sie der Funktion *getNextRule*. Die *Rule Engine* stellt die zweite Komponente dar und kann wiederum weiter unterteilt werden. Die *Condition Part Evaluation Engine* überprüft den Condition Part einer gegebenen Transformationsregel und implementiert die Funktion *isApplicable*. Sollte eine Transformationsregel anwendbar sein, führt die *Action Part Evaluation Engine* den zugehörigen Action Part aus. Diese Komponente implementiert in Bezug auf obige Algorithmen die Funktion *applyRule*.

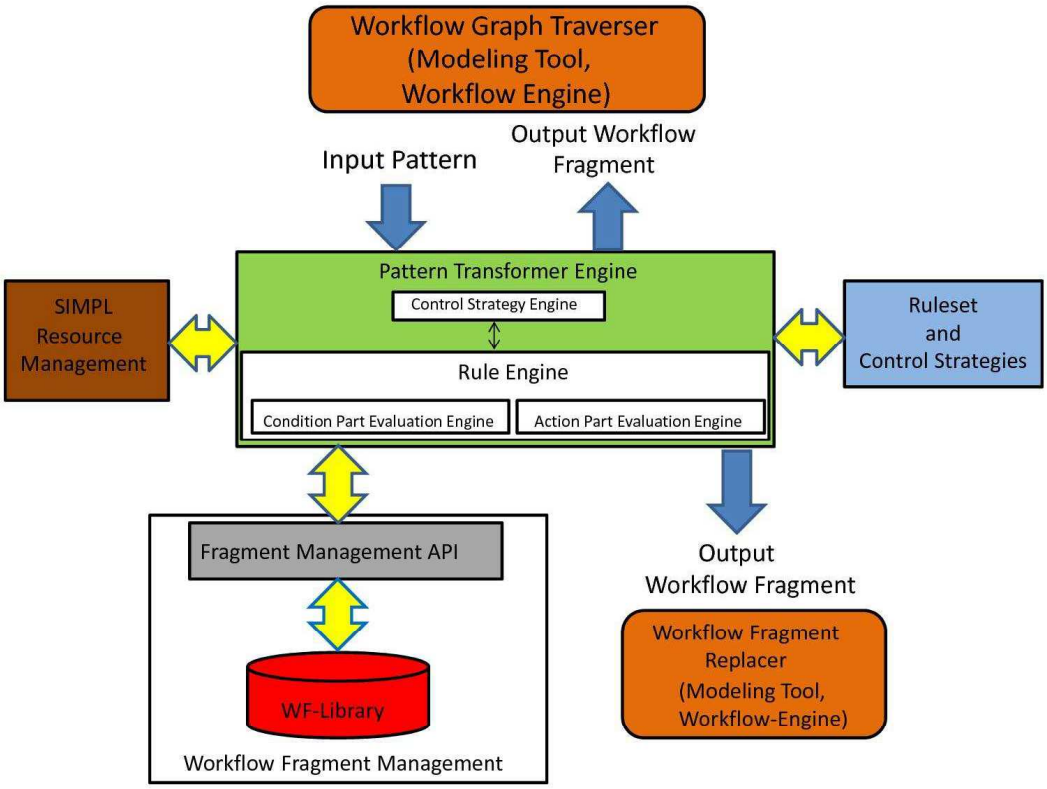

Abbildung 5.9.: Bisherige Architektur des Abbildungsmechanismus [Ari12]

- **SIMPL Resource Management:** Das Resource Management verwaltet benötigte Metadaten. So werden im Resource Management z.B. Informationen über einzelne Container, Datenressourcen sowie andere Objekte gespeichert. Diese Informationen werden dann bei Überprüfung einer Transformationsregel auf Anwendbarkeit bei Bedarf miteinbezogen.

- **Workflow Fragment Management:** Diese Komponente verwaltet die einzelnen Workflow-Fragmente, die bei der Anwendung eines Action Parts einer Transformationsregel identifiziert und anschließend parametrisiert werden. Dabei werden die konkreten Fragmente in der WF-Library gespeichert.
- **WF Management API:** Diese Komponente stellt die Schnittstelle zwischen der WF-Library und den anderen Komponenten dieser Architektur (insbesondere der Pattern Transformer Engine) dar.
- **Workflow Fragment Replacer:** Der Workflow Fragment Replacer ersetzt das ursprüngliche Datenmanagementpattern durch das erzeugte bzw. parametrisierte Workflow-Fragment und entspricht in Bezug auf obige Algorithmen der Funktion *replace*. Bei einer Transformation zur Modellierungszeit ist diese Komponente in die GUI integriert. Bei einer Transformation zur Laufzeit hingegen ist diese Komponente wiederum Bestandteil der Execution Engine.

5.4.3. Beispiele der Pandas Preprocessing-Phase

Im Folgenden wird nun die Pandas Preprocessing-Phase (vgl. Abschnitt 4.2) betrachtet. Es wird gezeigt, welche Möglichkeiten es nach [Ari12] gibt, Datenmanagementpatterns in dieser Phase zu verwenden. Innerhalb der Pandas Preprocessing-Phase werden Dateien, die für die Ausführung der Simulation benötigt werden, geladen und vom Workflow-Rechner auf den Pandas-Rechner kopiert. Auf dem Workflow-Rechner wird der Workflow ausgeführt, auf dem Pandas-Rechner hingegen die eigentliche Simulation. Abbildung 5.10 zeigt das zugehörige Workflow-Fragment. In diesem Workflow-Fragment wird eine Schleife (*ForEach*) definiert. Diese Schleife iteriert über eine Liste von Dateipfaden der Eingabedateien, die auf dem Workflow-Rechner gespeichert sind. Bei jedem Schleifendurchlauf wird ein Dateipfad selektiert und die entsprechende Datei wird durch eine TransferData Aktivität (*Transfer_Input_Data*) auf den Pandas-Rechner kopiert. Das Zielverzeichnis, also das Arbeitsverzeichnis des Pandas-Rechners, bleibt dabei in jedem Schleifendurchlauf dasselbe. Listing 5.5 zeigt den zugehörigen Quellcode. Die Variable *List* enthält die Liste der Dateipfade. Die *ForEach*-Schleife iteriert nun über diese Liste. Der aktuelle Iterationsschritt bzw. der Index wird mithilfe der Variablen *Counter* festgehalten. Zu Begin der Ausführung wird der Startwert der Schleife (*startCounterValue*) als eins gesetzt. Die Anzahl der Iterationen (*finalCounterValue*) wird mithilfe eines XPath-Ausdrucks berechnet und entspricht der Anzahl der Dateipfade in der Liste *List*. Das Element *Scope* umfasst den Schleifenrumpf. Bei jedem Iterationsschritt wird der Index (*Counter*) der Variablen *ListNumber* zugewiesen. Anschließend wird mithilfe der Variablen *ListNumber* und eines XPath-Ausdrucks der aktuelle Dateipfad ausgelesen (*Get_Path*) und der Variablen *Input_Data_Path* zugewiesen. Danach erfolgt der eigentliche Dateitransfer durch eine TransferData Aktivität (*Transfer_Input_Data*). Die Variable *workingDirectory* beinhaltet dabei die Referenz auf das Arbeitsverzeichnis auf dem Pandas-Rechner.

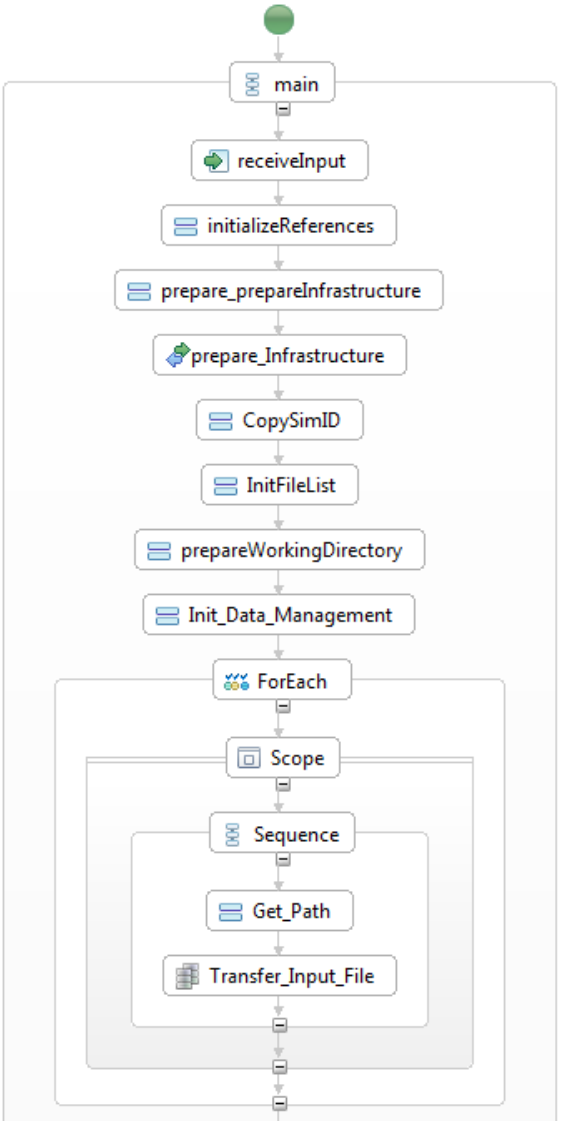

Abbildung 5.10.: Die Pandas Preprocessing-Phase in detaillierter Form

Listing 5.5 Transfer der einzelnen Dateien auf den Workflow-Rechner

```
<bpel:forEach parallel="no" counterName="Counter" name="ForEach">
  <bpel:startCounterValue>
    <![CDATA[1]]>
  </bpel:startCounterValue>
  <bpel:finalCounterValue>
    <![CDATA[round(count($/fileReference))]]>
  </bpel:finalCounterValue>
  <bpel:completionCondition/>
  <bpel:scope>
    <bpel:sequence>
      <bpel:assign validate="no" name="Get_Path">
        <bpel:copy>
          <bpel:from variable="Counter"/>
          <bpel:to variable="ListNumber"/>
        </bpel:copy>
        <bpel:copy>
          <bpel:from>
            <![CDATA[concat($List/fileReference[number($ListNumber)]/directory/text(),
                $List/fileReference[number($ListNumber)]/fileName/text())]]>
          </bpel:from>
          <bpel:to variable="Input_Data_Path"/>
        </bpel:copy>
      </bpel:assign>
      <bpel:extensionActivity>
        <simpl:transferDataActivity name="Transfer_Input_Data"
            dsStatement="#Input_Data_Path#" dsKind="Windows Local" dsType="Filesystem"
            dsIdentifier="DataSourceLocal" dsLanguage="Shell" targetDsType="Filesystem"
            targetDsKind="SSH Server" targetDsIdentifier="DataSourceRemote"
            targetDsLanguage="Shell" targetDsContainer="#workingDirectory#"/>
      </bpel:extensionActivity>
    </bpel:sequence>
  </bpel:scope>
</bpel:forEach>
```

Da die aktuelle Version des SIMPL-Rahmenwerks *data container references* unterstützt, werden im weiteren Verlauf dieser Arbeit *data container reference variables* zur Referenzierung der einzelnen Dateien genutzt und dieser Begriff ab sofort verwendet. Der Pandas-Workflow wird im Rahmen dieser Arbeit ebenfalls auf solche Variablen umgestellt.

Die gerade erläuterte Schleife kann durch ein sequentielles Data Iteration Pattern (vgl. Abschnitt 5.4) in Verbindung mit einem Container-to-Container Pattern beschrieben werden. Das Data Iteration Pattern iteriert über eine Liste, die *data container reference variables* beinhaltet. Bei jedem Iterationsschritt wird als Operation ein Container-to-Container Pattern ausgeführt. Das Data Iteration Pattern benötigt zwei Eingabeparameter: der Parameter S entspricht der Variablen *List* (also der Liste mit den *data container reference variables*). Die bei jedem Iterationsschritt selektierte Teilmenge wird mithilfe des Parameters T festgehalten. Im erläuterten Beispiel wäre dies eine *data container reference variable*, die in jedem Iterationsschritt eine andere Datei referenziert. Diese Filter-Operation (vgl. Abschnitt 5.4.1.3) realisiert die Assign Aktivität *Get_Path*. Die erläuterte *Merge Stage* entfällt in diesem Beispiel.

Das Container-to-Container Pattern benötigt ebenfalls zwei Eingabeparameter. Durch den Parameter *Source* wird der Quellcontainer referenziert. Der Parameter *Target* referenziert entsprechend den Zielcontainer. In unserem Beispiel entspricht der Parameter *Source* der selektierten Teilmenge T. Der Paramter *Target* ist in jedem Iterationsschritt gleich und entspricht der Variablen *workingDirectory*.

In den Abschnitten 3.7 und 5.4.2 wird erläutert, dass Datenmanagementpatterns mithilfe von Transformationsregeln auf ausführbare Workflow-Fragmente abgebildet werden. Dabei besteht eine Transformationsregel aus einem Condition Part sowie einem Action Part. Im erläuterten Beispiel wird das Data Iteration Pattern durch eine Transformationsregel auf die erläuterte *ForEach*-Schleife abgebildet. Dabei wird das enthaltene Container-to-Container Pattern als Black Box betrachtet und bei der Anwendung dieser Transformationsregel nicht transformiert. Das Container-to-Container Pattern wird anschließend wiederum durch eine andere Transformationsregel auf die beschriebene TransferData Aktivität abgebildet. Die konkreten Transformationsregeln sind in [Ari12] zu finden und werden in Abschnitt 6.2 weiter verfeinert.

5.4.4. Beispiele der Pandas Postprocessing-Phase

Auch die Pandas Postprocessing-Phase bietet die Möglichkeit Datenmanagementpatterns zu verwenden. Im Folgenden werden nun die in [Ari12] erläuterten Möglichkeiten vorgestellt. Abbildung 5.11 zeigt eine schematische Darstellung der Postprocessing-Phase.

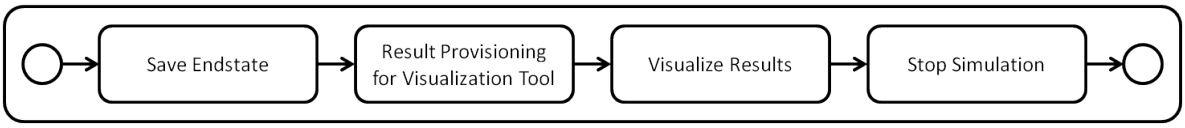

Abbildung 5.11.: Die Pandas Postprocessing-Phase in detaillierter Form (nach [Ari12])

Die Aktivität *Save Endstate* speichert den Endzustand der Simulation in einer Datei. Danach werden durch die Aktivität *Result Provisioning for Visualization Tool* diese Ergebnisdaten vom Pandas-Rechner auf dem Workflow-Rechner kopiert. Da die Ergebnisdaten im *TecPlot*-Format (tecplot_final.dat) vorliegen, das Visualisierungsswerkzeug die Daten aber im *VTK*-Format (visualizationInput.vtk) benötigt, muss eine entsprechende Datenformatkonvertierung stattfinden, bevor die Daten kopiert werden können. Durch die Aktivität *Visualize Results* wird dann die eigentliche Visualisierung angestoßen. Die Aktivität *stop Simulation* stoppt zuletzt die gesamte Pandas-Simulation.

Der erläuterte Dateitransfer kann auch hier durch ein Container-to-Container Pattern beschrieben werden. Der Parameter *Source* entspricht der *data container reference variable*, die die Datei tecplot_final.dat referenziert. Als *Target* Parameter muss die *data container reference variable* angegeben werden, die die Datei visualizationInput.vtk referenziert. Da die referenzierten Dateien unterschiedliche Datenformate aufweisen, reicht eine TransferData Aktivität

alleine nicht aus. Deshalb wird dieses Container-to-Container Pattern auf eine Sequenz, die aus einem Data Format Conversion Pattern gefolgt von einem Container-to-Container Pattern mit gleichen Dateiformaten besteht, abgebildet. Es handelt sich hierbei also um eine Transformationsregel vom Regeltyp-2, da das Pattern auf ein Workflow-Fragment abgebildet wird, das wiederum Datenmanagementpatterns enthält (vgl. Abschnitt 3.7). Das Data Format Conversion Pattern benötigt ebenfalls zwei Eingabeparameter: Der *Source* Parameter entspricht der *data container reference variable*, die die eigentliche Quelldatei (tecplot_final.dat) referenziert. Das Ergebnis der Konvertierung wird wiederum in einer Datei gespeichert und durch den *Target* Parameter referenziert. Dies ist dann ebenfalls der *Source* Parameter des Container-To-Container Patterns innerhalb der Sequenz. Der *target* Parameter des Container-to-Container Patterns referenziert die eigentliche Zieldatei auf dem Workflow-Rechner (visualizationInput.vtk).

Das Data Format Conversion Pattern wird durch eine Transformationsregel auf eine IssueCommand Aktivität abgebildet. Diese IssueCommand Aktivität startet die Datenformatkonvertierung, indem sie ein Transformationsscript aufruft, welches die eigentliche Formatumwandlung durchführt. Um dies zu ermöglichen muss das Resource Management erweitert werden. Es muss eine Möglichkeit geschaffen werden externe Datenformate (wie z.B. das *TecPlot-Format* oder das *VTK-Format*) zu registrieren. Desweiteren muss es möglich sein, Transformationsscripte bzw. Web Services zu registrieren. Bei der Transformation des Data Format Conversion Patterns würde dann im Resource Management nach einem geeigneten Transformationsservice gesucht werden. Wird ein geeigneter Transformationsservice gefunden, kann dieser durch die erläuterte IssueCommand Aktivität aufgerufen werden.

Nach dieser Datenformatkonvertierung können die Dateien durch das eingebettete Container-to-Container Pattern einfach kopiert werden, da beide Dateien dasselbe Datenformat aufweisen. Das eingebettete Container-to-Container kann also direkt durch die in Abschnitt 5.4.3 erläuterte Transformationsregel auf eine TransferData Aktivität abgebildet werden. Die definierten Transformationsregeln für das komplexe Container-to-Container Pattern sowie das Data Format Conversion Pattern sind in [Ari12] zu finden.

Im Folgenden werden die in [Ari12] definierten Transformationsregeln kurz zusammengefasst:

- **Transformationsregel für das einfache Container-to-Container Pattern:** Diese Transformationregel bildet ein Container-to-Container Pattern auf eine TransferData Aktivität ab. Voraussetzung ist, dass die Datenformate der referenzierten Objekte **identisch** sind.
- **Transformationsregel für das komplexe Container-to-Container Pattern:** Wenn die Datenformate der referenzierten Objekte **unterschiedlich** sind, bildet diese Tranformationsregel ein Container-to-Container Pattern auf eine Sequenz ab. Diese Sequenz beinhaltet wiederum ein Data Format Conversion Pattern sowie ein Container-to-Container Pattern. Durch das Data Format Conversion Pattern wird das zu kopierende Objekt konvertiert. Anschließend sind die Datenformate identisch und das eingebettete Container-to-Container Pattern kann das Objekt kopieren.

- **Transformationsregel für das Data Format Conversion Pattern:** Diese Transformationsregel bildet ein Data Format Conversion Pattern auf eine IssueCommand Aktivität ab. Voraussetzung ist, dass im Resource Management ein geeignetes Script, das die Konvertierung durchführt, gefunden wird. Durch die IssueCommand Aktivität wird dieses Script gestartet.
- **Transformationsregel für das sequentielle Data Iteration Pattern:** Ein sequentielles Data Iteration Pattern wird durch diese Transformationsregel auf eine ForEach-Schleife abgebildet. Diese Schleife iteriert über eine Liste, die *data container reference variables* beinhaltet und führt die vom Nutzer definierte Operation aus.

6. Konzeptionelle Änderungen und Erweiterungen

In Kapitel 5 werden die prototypische Umsetzung des SIMPL-Rahmenwerks erläutert und die bisher nur konzeptionell entwickelten Datenmanagementpatterns betrachtet. Dieses Kapitel beschäftigt sich nun mit konzeptionellen Änderungen und Erweiterungen, die für multi-skalare Simulationsworkflows wie die gesamte Pandas-Matlab Kopplung (vgl. Abschnitt 4.3) notwendig sind. Der Zugriff auf Dateisysteme wird in Abschnitt 6.1 analysiert. Ziel ist es die Ausführungsumgebung von SIMPL für ETL Patterns (vgl. Abschnitt 3.7) zu verbessern. Dies ist wichtig, da multi-skalare Simulationsworkflows unterschiedlichste Dateien in unterschiedlichen Dateisystemen verarbeiten können müssen. In Abschnitt 6.2 wird dann erneut die Pandas Preprocessing- und Postprocessing Phase betrachtet. Die in [Ari12] definierten Transformationsregeln werden weiter verfeinert sowie ein neues Datenmanagementpattern, das Multiple Data Transfer Pattern, wird vorgestellt. Zuletzt wird in Abschnitt 6.3 noch die Kopplungsphase der Pandas-Matlab Kopplung betrachtet. Der Verwendung des parallelen Data Iteration Patterns wird erläutert und zugehörige Transformationsregeln werden definiert.

6.1. Zugriffsmechanismen für Dateisysteme

ETL Patterns (vgl. Abschnitt 3.7) modellieren typische feingranulare DM Operationen. Wird eine DM Aktivität, die solch ein ETL Pattern oder Teile davon umsetzen kann, ausgeführt, ruft die Execution Engine die entsprechende generische Operation des SIMPL Cores auf. Konnektoren und Konverter implementieren die generischen Operationen des SIMPL Cores für bestimmte Datenquellen bzw. Typen von Datenquellen (vgl. Abschnitt 3). Die prototypische Umsetzung des SIMPL-Rahmenwerks unterstützt, in Bezug auf lokale Dateisysteme, bisher nur das Windows-Dateisystem. Der Zugriff auf Unix-basierte Dateisysteme ist bisher nicht bzw. nur eingeschränkt möglich. Ein Zugriff auf entfernte Dateisysteme ist mittels SSH und des SSHConnectors möglich. Jedoch weist der Transfer einer Datei auf einen entfernten Rechner eine gewisse Imperformanz auf. Im Folgenden wird nun auf die Integration von Unix-Dateisystemen eingegangen sowie der Zugriff auf einen entfernten Rechner mittels SSH betrachtet.

6.1.1. Unix-basierte Dateisysteme

Der Konnektor für das Windows-Dateisystem funktioniert teilweise auch unter Linux, ist jedoch an die Besonderheiten von Windows (z.B. Laufwerksbuchstaben) angepasst. Um die Ausführungsumgebung von SIMPL zu verbessern, wird im Rahmen dieser Diplomarbeit ein neuer Konnektor implementiert. Dieser implementiert alle generischen Operationen des SIMPL Cores (vgl. Abschnitt 5.3) für Unix-Dateisysteme. Auf die konkrete Umsetzung des neuen Konnektors wird in Abschnitt 7.1 eingegangen.

Eine Workflow Aktivität erwartet bzw. übergibt Daten in einem XML-basierten Format. Konverter implementieren die benötigten Transformationen von den Formaten der Konnektoren zu diesen XML-basierten Formaten und umgekehrt. Durch eine RetrieveData Operation wird beim Zugriff auf ein Dateisystem entweder eine einzelne Datei oder ein Ordner mit mehreren Dateien geladen. Der RandomFileDataConverter überführt die geladenen Daten in das XML-basierte RandomFileDataFormat. Ausschnitte aus dem zugehörigen XML Schema sind in Abbildung 6.1 dargestellt.

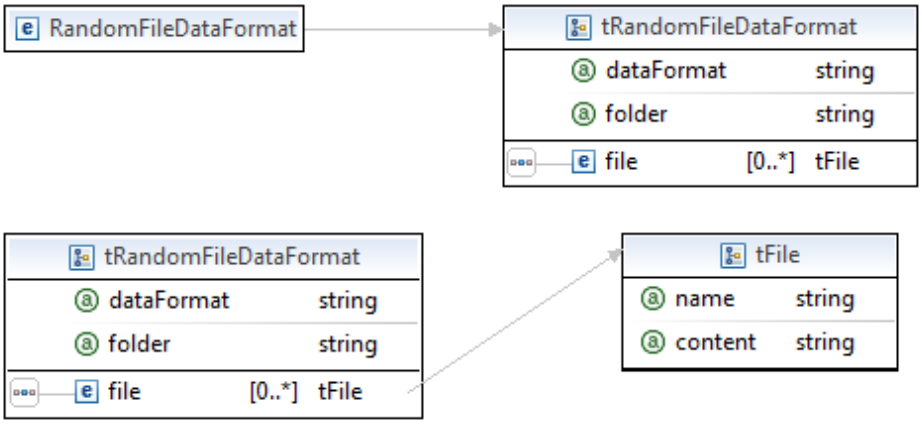

Abbildung 6.1.: Auszüge aus der RandomFileDataFormat Definition

Das Wurzelelement *RandomFileDataFormat* beinhaltet Attribute. Zum einen werden Informationen über das verwendete Datenformat gespeichert, zum anderen wird ein Pfad festgehalten. Dieser Pfad verweist entweder auf die einzelne Datei oder, wenn es sich um mehrere Dateien handelt, auf den übergeordneten Ordner. Für jede Datei wird ein Element vom Typ *tFile* erzeugt. In jeweils einem Attribut wird der Name der Datei sowie eine HTML-Kodierung des Inhalts gespeichert. Sollen die einzelnen Dateien aus dieser Struktur wieder extrahiert werden, erzeugt der RandomFileDataConverter ein temporäres Verzeichnis. In diesem Verzeichnis werden dann die einzelnen Dateien gespeichert, jeweils mit dem Namen und dem Dateiinhalt aus den Attributen *name* bzw. *content*. Das erzeugte Verzeichnis und die einzelnen Dateien darin können anschließend von der WriteDataBack Operation eines geeigneten Konnektors weiterverarbeitet werden.

Da sowohl der Konnektor für das Windows-Dateisystem sowie der neue Konnektor für das Unix-Dateisystem eine Datei bzw. einen Ordner mit mehreren Dateien lädt bzw. speichert, kann für beide Konnektoren der gerade erläuterte Konverter genutzt werden. Demzufolge ist es *nicht* nötig einen neuen Konverter zu implementieren.

6.1.2. Zugriff auf entfernte Rechner

Mithilfe von SSH (Secure Shell) kann eine verschlüsselte Verbindung mit einem entfernten Rechner hergestellt werden. Die Umsetzung des SIMPL-Rahmenwerks unterstützt den Zugriff auf entfernte Rechner durch den sogenannten SSHConnector. Dieser nutzt SSH, um Dateien zu transferieren sowie um Befehle auf einem entfernten Rechner auszuführen.

In der Pandas Preprocessing-Phase (vgl. Abschnitte 4.2 und 5.4.3) werden benötigte Dateien mithilfe einer ForEach-Schleife und einer eingebetteten TransferData Aktivität vom Workflow-Rechner in das Arbeitsverzeichniss des Pandas-Rechners kopiert. Intern in der Execution Engine wird diese TransferData Aktivität auf eine RetrieveData sowie eine WriteDataBack Aktivität abgebildet (vgl. Abschnitt 5.3). Anschließend werden die entsprechenden generischen Operationen des SIMPL Cores aufgerufen. Mithilfe der RetrieveData Operation des Konnektors für das Windows-Dateisystem wird die referenzierte Datei geladen und durch den in Abschnitt 6.1.1 erläuterten RandomFileDataConverter in das XML-basierte RandomFileDataFormat überführt. Danach wird die jeweilige Datei aus dieser XML-basierten Struktur wieder extrahiert und durch die WriteDataBack Operation des SSHConnectors auf den Pandas-Rechner kopiert.

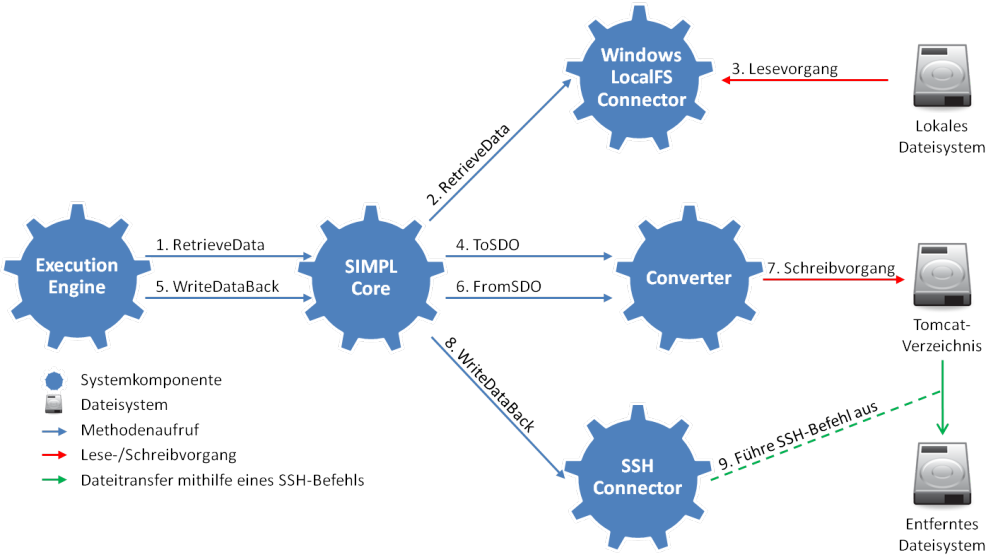

Abbildung 6.2.: Dateitransfer mithilfe des SSHConnectors (**vor** den durchgeführten Änderungen)

Problematisch ist, dass bei dieser Vorgehensweise bei jeder Iteration die aktuell zu kopierende Datei in das RandomFileDataFormat überführt wird, anschließend wieder aus diesem Format extrahiert und im Tomcat-Verzeichnis *zwischengespeichert* (vgl. Abschnitt 6.1.1) wird. Erst dann wird die temporäre Kopie auf den Pandas-Rechner kopiert (vgl. Abbildung 6.2).

In diesem Szenario ist es eigentlich nicht nötig eine Datenformatkonvertierung und eine Zwischenspeicherung der Datei im Tomcat-Verzeichnis vorzunehmen. Die jeweilige Datei könnte direkt mittels eines geeigneten SSH-Befehls auf den Pandas-Rechner kopiert werden. Deshalb wäre ein Vorgehen wünschenswert, das die Datei direkt auf den Pandas-Rechner kopiert und somit noch Performanz-Vorteile erzielt. Um dies zu realisieren, muss der SIMPL Core um die generische Operation TransferData erweitert werden. Die einzelnen Konnektoren *können* diese Operation dann implementieren.

Wenn eine TransferData Aktivität ausgeführt werden soll, impliziert dieser Ansatz ein geändertes Vorgehen: Die Execution Engine ruft die generische TransferData Operation des SIMPL Cores auf. Innerhalb des SIMPL Cores findet dann eine Fallunterscheidung statt. Handelt es sich bei der Datenquelle um ein lokales Dateisystem und bei der Datensenke um einen entfernten Rechner, der mittels SSH angesprochen wird, so wird die TransferData Operation des SSHConnectors aufgerufen (vgl. Abbildung 6.3). Auf diese Weise kann eine Datei direkt, ohne Zwischenspeicherung, auf einen entfernten Rechner übertragen werden. Ansonsten wird wie bisher zuerst die RetrieveData Operation eines geeigneten Konnektors und anschließend die WriteDataBack Operation eines geeigneten Konnektors ausgeführt.

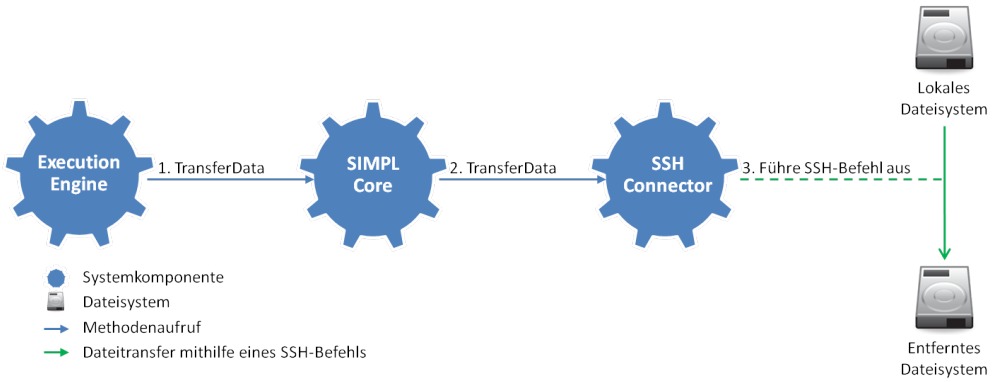

Abbildung 6.3.: Dateitransfer mithilfe des SSHConnectors (**nach** den durchgeführten Änderungen)

6.2. Datenmanagementpatterns in der Pandas Preprocessing- und Postprocessing-Phase

In den Abschnitten 5.4.3 und 5.4.4 werden die Einsatzmöglichkeiten von Datenmanagementpatterns in der Pandas Preprocessing- sowie Postprocessing-Phase erläutert. In [Ari12]

werden diesbezüglich für das einfache sowie das komplexe Container-to-Container Pattern, das Data Format Conversion Pattern und das sequentielle Data Iteration Pattern Workflow-Fragmente und zugehörige Transformationsregeln definiert. Bevor in Kapitel 7 auf die konkrete Umsetzung bzw. Implementierung der Pattern Transformation eingegangen wird, werden in diesem Abschnitt die in [Ari12] definierten Transformationsregeln analysiert und verfeinert. Des Weiteren wird ein neues Pattern, das sogenannte Multiple Data Transfer Pattern, definiert.

In Abschnitt 3.7 wird erläutert, dass bei der Transformation eines Datenmanagementpatterns in der Regel ein Template für ein Workflow-Fragment geladen wird, das bei der Transformation noch ausführbar gemacht werden muss, da es z.B. Platzhalter für bestimmte Parameterwerte beinhaltet. Durch diese Platzhalter können die einzelnen Workflow-Fragmente im Allgemeinen universell eingesetzt werden, da die konkreten Parameterwerte erst bei der Transformation eines Patterns bestimmt und eingesetzt werden. Zu beachten ist jedoch, dass es Sonderfälle gibt, bei denen die Workflow-Fragmente *nicht* universell einsetzbar sind. Solche Workflow-Fragmente berücksichtigen die Besonderheiten *eines* bestimmten Anwendungsfalls bzw. sind auf einen Anwendungsfall zugeschnitten. Aus diesem Grund gibt es drei Arten von Workflow-Fragmenten: generische Workflow-Fragmente, Workflow-Fragmente, die teilweise generisch sind, sowie Workflow-Fragmente, die überhaupt nicht generisch sondern spezifisch sind. In Kapitel 8 wird dieser Punkt noch einmal aufgegriffen und die im Rahmen dieser Diplomarbeit definierten Workflow-Fragmente und Transformationsregeln werden hinsichtlich ihrer universellen Einsetzbarkeit bewertet.

Eine weitere Verfeinerung der in [Ari12] definierten Workflow-Fragmente und zugehörigen Transformationsregeln ist nötig, da zwei Aspekte außer Acht gelassen werden: Die definierten Workflow-Fragmente enthalten Platzhalter für konkrete Parameterwerte. In den zugehörigen Transformationsregeln werden diese Platzhalter jedoch durch festgelegte Werte ersetzt. Bei der Transformation des Data Format Conversion Patterns z.B. wird im Resource Management ein geeigneter Data Transformation Service gesucht. Der Platzhalter *?python* wird jedoch immer durch den String *python tecp2vtk.py* ersetzt. Die vom Modellierer spezifizierten Parameterwerte sowie der gefundene Data Transformation Service werden nicht berücksichtigt. Des Weiteren sollen einzelne Workflow-Fragmente voneinander isoliert werden. Jedoch werden beim sequentiellen Data Iteration Pattern Variablen für das ausführbare Workflow-Fragment teilweise global im Prozess, und nicht im Scope der Schleife, deklariert. Dies führt dazu, dass die definierten Workflow-Fragmente bzw. die Transformationsregeln nicht generisch, sondern auf einen Anwendungsfall zugeschnitten sind. Im Folgenden werden nun die einzelnen Transformationsregeln überarbeitet. Ziel ist es, die Workflow-Fragmente sowie zugehörigen Transformationsregeln **so generisch wie möglich** zu definieren.

Zuvor werden jedoch noch mal *data container reference variables* betrachtet. Ein Container einer Datenquelle kann auf zwei Arten referenziert werden: über die Angabe des logischen Namens des Containers (wenn dieser im Resource Management registriert wurde) oder über die Angabe des lokalen Bezeichners (vgl. Abschnitt 5.2). Wenn ein Container im Resource Management registriert wurde, sind dort auch Informationen über das Datenformat sowie über die zugehörige Datenquelle, in der sich der Container befindet, gespeichert. Wird ein Container hingegen mithilfe eines lokalen Bezeichners referenziert, muss diese Referenz

auch das Datenformat sowie den Namen einer zugehörigen *data **source** reference variable* beinhalten. Listing 6.1 zeigt den Basistyp für die Referenzierung eines Containers über dessen lokalen Bezeichner. Dieser Basistyp bietet die Möglichkeit das Datenformat sowie die zugehörige Datenquelle zu spezifizieren. Bei den folgenden Transformationsregeln wird davon ausgegangen, dass ein Container stets über dessen lokalen Bezeichner referenziert wird und *nicht* im Resource Management registriert wurde (in Kapitel 9 wird dieser Aspekt nochmal aufgegriffen).

Listing 6.1 Basistyp für die Referenzierung eines Containers über dessen lokalen Bezeichner

```
<xsd:complexType name="LocalDataContainerReferenceType">
  <xsd:complexContent>
    <xsd:extension base="simpl:DataContainerReferenceType">
      <xsd:sequence>
        <xsd:element name="dataSourceReferenceVariable" type="xsd:string" maxOccurs="1"
            minOccurs="0"/>
        <xsd:element name="dataFormat" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

6.2.1. Das einfache Container-to-Container Pattern

In Abschnitt 5.4.3 wird erläutert, dass das einfache Container-to-Container der Pandas Preprocessing- und Postprocessing-Phase auf eine TransferData Aktivität abgebildet wird und als Eingabeparameter zwei *data container reference variables* bzw. deren Namen erwartet. Der Condition Part der zugehörigen Transformationsregel sieht folgendermaßen aus:

- Condition 1: Der Parameter *sourceContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 2: Der Parameter *targetContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 3: Wenn bei beiden *data container reference variables* das Element *dataFormat* spezifiziert wurde, muss dieses gleich sein.

Wenn diese Bedingungen erfüllt sind, kann das Workflow-Fragment simpleC2C.xml (vgl. Anhang B.1) geladen und folgender Action Part ausgeführt werden:

- Action 1: Lade das Workflow-Fragment *simpleC2C.xml*
- Action 2: Ersetze den Platzhalter *?dataSource* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *sourceContainer* spezifiziert wird. Indem der Variablenname von eckigen Klammern umschlossen wird, werden bei den DM Aktivitäten des SIMPL-Rahmenwerks *data source reference variables* und *data container reference variables* kenntlich gemacht. Dieser

Variablenname muss von eckigen Klammern umschlossen sein, damit der SIMPL Core die Referenz auflöst.

- Action 3: Ersetze den Platzhalter *?dataSourceCommand* durch den Wert des Parameters *sourceContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 4: Ersetze den Platzhalter *?dataSink* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 5: Ersetze den Platzhalter *?dataSinkContainer* durch den Wert der Parameters *targetContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.

Der definierte Action Part berücksichtigt bei der Ersetzung der Platzhalter die vom Modellierer spezifizierten Parameter. Die Werte der Platzhalter werden erst bei der Transformation bestimmt und stehen nicht schon vorher fest. Dadurch ist das Workflow-Fragment bzw. die zugehörige Transformationsregel universell einsetzbar.

Werden Platzhalter, die z.B. eine Datenquelle referenzieren, ersetzt, muss folgendes beachtet werden: bei einer Transformation der Datenmanagementpatterns zur Modellierungszeit stehen Variablenwerte nicht immer fest. Wird ein Workflow ausgeführt, können sich während der Ausführung des Workflows Variablenwerte ändern. Soll bei der Transformation eines Patterns zur Modellierungszeit dann z.B. der Wert des Elementes *data source reference variable* bestimmt werden, ist nicht immer klar, welchen Wert dieses Element hat. Theoretisch müsste in so einem Fall eine Datenflussanalyse durchgeführt werden, um den Variablenwert bzw. den Wert eines Elementes zu bestimmen. Dies ist jedoch kein Bestandteil dieser Diplomarbeit. Es wird davon ausgegangen, dass eine Variable bei ihrer Deklaration initialisiert und ihr ein *fixed value* zugewiesen wurde. Bei der Transformation eines Datenmanagementpatterns wird dann *nur* dieser Initialwert berücksichtigt. Die Werte der entsprechenden Variablen dürfen also zwischen ihrer Initialisierung und dem jeweiligen Pattern nicht mehr geändert werden. Bei einer Transformation zur Laufzeit hingegen, stehen die Variablenwerte fest und können ohne Datenflussanalyse bestimmt werden.

6.2.2. Das komplexe Container-to-Container Pattern

In Abschnitt 5.4.4 wird erläutert, dass das in [Ari12] vorgestellte komplexe Container-to-Container Pattern auf eine Sequenz abgebildet wird. Die Sequenz umfasst ein Data Format Conversion Pattern sowie ein einfaches Container-to-Container Pattern.

In Zukunft wird das komplexe Container-to-Container Pattern auf einen Scope (vgl. Abbildung 6.4) abgebildet. Dieser Scope besteht aus einer Sequenz, die wiederum eine Assign Aktivität, ein einfaches Container-to-Container Pattern sowie ein Data Format Conversion Pattern beinhaltet. Die referenzierte Datei bzw. das referenzierte Objekt wird zuerst transferiert und anschließend konvertiert. Dies geschieht aus folgendem Grund: in der Pandas Postprocessing-Phase soll eine Datei, die im TecPlot-Format vorliegt, in das VTK-Format konvertiert und an einem referenzierten Ort bereitgestellt werden. Dateien, die im TecPlot-Format

vorliegen, sind in der Regel kleiner als in das VTK-Format konvertierte Dateien. Da in erster Linie der vorgestellte Anwendungsfall durch Datenmanagementpatterns unterstützt werden soll, wird zuerst die Dateiübertragung und anschließend die Datenformatkonvertierung durchgeführt.

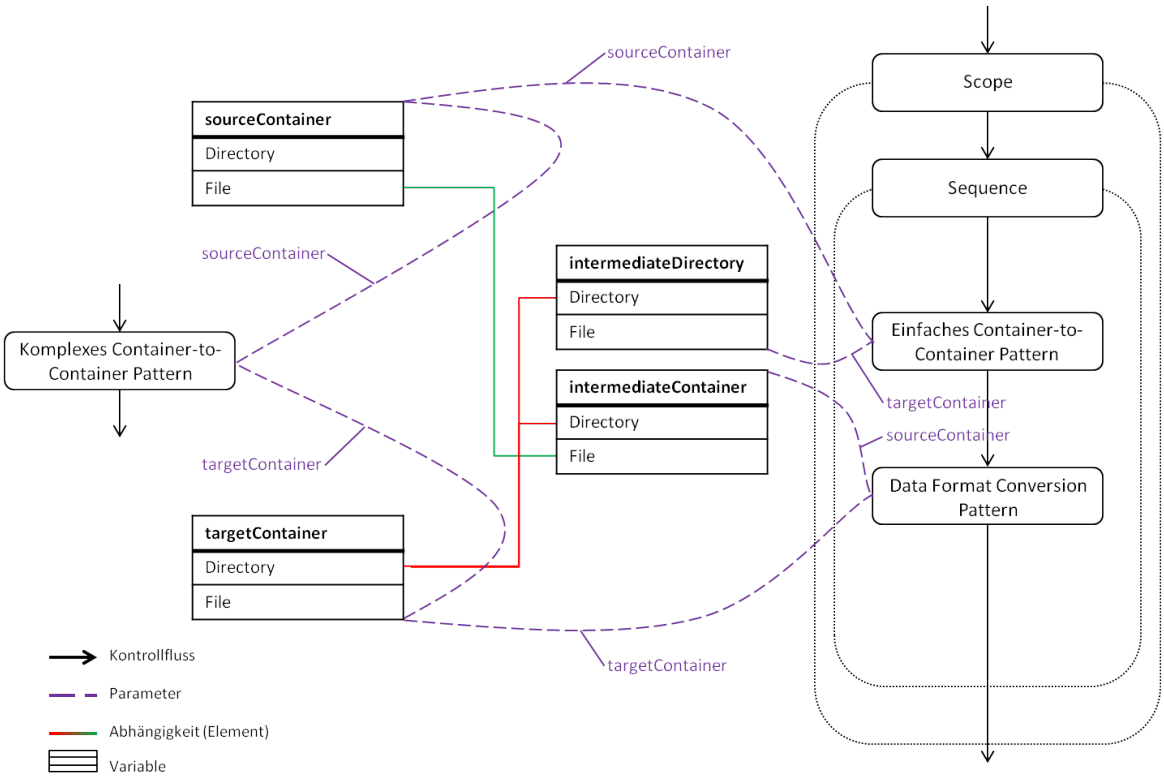

Abbildung 6.4.: Transformation des komplexen Container-to-Container Patterns

Der Scope wird benötigt, da in diesem Workflow-Fragment lokale Variablen erzeugt werden müssen. Das komplexe Container-to-Container Patterm erwartet als Eingabeparameter zwei *data container reference variables* bzw. deren Namen. Im betrachteten Anwendungsfall sind dies jeweils Referenzen innerhalb eines Dateisystems. Solch eine Referenz beinhaltet u.a. die Elemente *directory* und *file*. Zu beachten ist, dass das Element *file* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird, keinen Wert enthalten muss (es wird also nur ein Verzeichnis und keine Datei referenziert). Trotzdem hat der Nutzer in so einem Fall ein Datenformat, wie z.B. VTK angegeben. D.h., dass in diesem Verzeichnis nur Dateien im VTK-Format gespeichert werden sollen. Die durch den Parameter *sourceContainer* referenzierte Datei soll also in das, durch den Parameter *targetContainer*, referenzierte Verzeichnis kopiert und konvertiert werden. Dabei muss die konvertierte Datei keinen bestimmten (vom Modellierer spezifizierten) Namen aufweisen. In den Abschnitten 6.2.3 und 6.2.4 wird erneut auf dieses Szenario eingegangen.

Das eingebettete einfache Container-to-Container Pattern erwartet für die Eingabeparameter *sourceContainer* und *targetContainer* ebenfalls zwei *data container reference variables*. Mithilfe des Parameters *sourceContainer* wird die Datei referenziert, die kopiert werden soll. Dieser Parameter entspricht dem Parameter *sourceContainer* des zu transformierenden komplexen Container-to-Container Patterns. Mithilfe des Parameters *targetContainer* des einfachen Container-to-Container Patterns wird das Zielverzeichnis referenziert, in das die Datei kopiert werden soll. Das Zielverzeichnis wird durch das Element *directory* der *data container reference variable* spezifiziert, die für den Parameter *targetContainer* des zu transformierenden komplexen Container-to-Container Patterns angegeben wurde. Demzufolge muss also eine lokale Variable erzeugt werden, der das erläuterte Element *directory* zugewiesen wird. Das Element *file* wird nicht angegeben, da nur ein Verzeichnis referenziert wird. Löst der SIMPL Core diese Referenz auf, ist das Ergebnis das gewünschte Zielverzeichnis.

Das eingebettete Data Format Conversion Pattern erwartet für die Eingabeparameter *sourceContainer* und *targetContainer* auch zwei *data container reference variables*. Der Parameter *sourceContainer* referenziert die Datei, die konvertiert werden soll. Mithilfe des zweiten Parameters wird wiederum das Ziel spezifiziert. Dabei kann es sich um einen Verzeichnisnamen oder um einen Verzeichnisnamen in Verbindung mit einem konkreten Dateinamen handeln. Je nachdem, ob die konvertierte Datei einen bestimmten Namen aufweisen soll. Für den Parameter *sourceContainer* muss wiederum eine lokale *data container reference variable* erzeugt werden. Diese lokale Variable referenziert die Datei, die durch das einfache Container-to-Container Pattern zuvor in das Zielverzeichnis kopiert wurde. Zum einen wird dieser lokalen Variable das Element *directory* der globalen *data container reference variable*, die durch den Parameter *targetContainer* des zu transformierenden komplexen Container-to-Container Patterns referenziert wird, zugewiesen. Zum anderen wird dieser Variablen das Element *file* der globalen *data container reference variable*, die durch den Parameter *sourceContainer* des komplexen Container-to-Container Patterns referenziert wird, zugewiesen. Der Parameter *targetContainer* des Data Format Conversion Patterns entspricht der *data container reference variable*, die der Modellierer für den Parameter *targetContainer* des zu transformierenden komplexen Container-to-Container Patterns angegeben hat. Das erläuterte Vorgehen ist in Abbildung 6.4 schematisch dargestellt.

Dieses Vorgehen impliziert, dass das zugehörige Workflow Fragment nur für *data container reference variables* geeignet ist, die Dateien bzw. Verzeichnisse innerhalb eines Dateisystems referenzieren. Innerhalb der Assign Aktivitäten wird auf Elemente zugegriffen, die es nur bei solchen Referenzen gibt. Der Condition Part der zugehörigen Transformationsregel lautet deswegen:

- Condition 1: Der Parameter *sourceContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 2: Der Parameter *targetContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 3: Beide *data container references variables* referenzieren Ordner bzw. Dateien innerhalb eines Dateisystems.

- Condition 4: Bei beiden *data container reference variables* wurde das Element *dataFormat* spezifiziert und dieses ist unterschiedlich.

Wenn diese Bedingungen erfüllt sind, kann der im Folgenden beschriebene Action Part ausgeführt werden. Zu beachten ist, dass in diesem Workflow-Fragment erstmals lokale *data container reference variables* neu erzeugt werden. In Abschnitt 6.2.1 wird erläutert, dass Variablen von diesem Typ einen Initialwert benötigen. Aus diesem Grund wird auch einer lokalen *data container reference variable* ein Datenformat sowie eine *data source reference variable* zugewiesen. Auf diese Weise kann auch bei lokal definierten Variablen deren Initialwert ausgelesen werden.

- Action 1: Lade das Workflow-Fragment *complexC2C.xml* (vgl. Anhang B.2).
- Action 2: Ersetze alle Vorkommen des Platzhalters *?randomNumber* durch eine Zufallszahl. Dies ist nötig, damit die einzelnen Scopes eindeutig benannt werden.
- Action 3: Ersetze den Platzhalter *?sourceContainer* durch den Wert des Parameters *sourceContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 4: Ersetze alle Vorkommen des Platzhalters *?sourceContainerName* durch den Wert des Parameters *sourceContainer*. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 5: Ersetze den Platzhalter *?targetContainer* durch den Wert des Parameters *targetContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 6: Ersetze alle Vorkommen des Platzhalters *?targetContainerName* durch den Wert des Parameters *targetContainer*. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 7: Ersetze alle Vorkommen des Platzhalters *?dataSourceReferenceVariable* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein. Dieser Platzhalter wird benötigt, um die lokalen Variablen zu initialisieren.
- Action 8: Ersetze alle Vorkommen des Platzhalters *?dataFormat* durch den Wert des Elementes *dataFormat* der *data container reference variable*, die durch den Parameter *sourceContainer* spezifiziert wird. Dies ist nötig, damit das eingebettete einfache Container-to-Container Pattern abgebildet werden kann und die Datenformate gleich sind.

Dieses Workflow-Fragment bzw. die zugehörige Transformationsregel kann immer dann eingesetzt werden, wenn Dateien an einem referenzierten Ort bereitgestellt werden sollen und eine Datenformatkonvertierung nötig ist.

Das eingebettete einfache Container-to-Container Pattern kann, wie in Abschnitt 6.2.1 beschrieben, transformiert werden. Wie das Data Format Conversion Pattern transformiert wird, wird im nächsten Teilkapitel erläutert.

6.2.3. Das Data Format Conversion Pattern

Mithilfe des Data Format Conversion Patterns werden Datenformatkonvertierungen durchgeführt. Im Allgemeinen kann solch eine Konvertierung auf unterschiedliche Weise erfolgen: so können z.B. die Methoden des Service Bus genutzt, eine Script auf Kommandozeilenebene gestartet oder ein Web Service bzw. RESTful Service aufgerufen werden. Metadaten über solche Data Transformation Services sowie Metadaten über Datenformate werden im Resource Management gespeichert. Weiterhin werden für die Data Transformation Services entsprechende Datenformatpaare angegeben, die die Services transformieren können. Auf die konkrete Umsetzung und auf die Unterschiede bzgl. der verschiedenen Arten von Data Transformations Services wird in Kapitel 7.3 eingegangen. In der Pandas Postprocessing-Phase wird eine Datei, die im TecPlot-Format vorliegt, mithilfe des Data Format Conversion Patterns in das VTK-Format überführt. Die eigentliche Transformation geschieht über einen Script-Aufruf. Im Folgenden wird nun ein Workflow-Fragment (sowie eine zugehörige Transformationsregel) vorgestellt, das auf solche Script-Aufrufe ausgelegt ist. In Abschnitt 6.3.3 wird noch auf ein Beispiel eingegangen, das einen Web Service aufruft.

In [Ari12] wird erläutert, dass das Data Format Conversion Pattern der Pandas Postprocessing-Phase auf eine IssueCommand Aktivität abgebildet wird. In Zukunft wird dieses Pattern jedoch auf einen Scope abgebildet. Dieser Scope beinhaltet eine Sequenz, die wiederum aus einer Assign Aktivität und einer IssueCommand Aktivität besteht. Dies ist nötig, da innerhalb des Workflow-Fragments eine lokale *data container reference variable* erzeugt werden muss. Die eingebettete IssueCommand Aktivität startet das im Resource Management gefundene Script über die Kommandozeile. Zuvor muss jedoch in das Verzeichnis gewechselt werden, in dem das Script sowie die Eingabedatei liegen (*cd [intermediateDirectory]*). Dafür wird die lokale *data container reference variable* benötigt. Das Zielverzeichnis wird durch das Element *directory* der *data container reference variable* bestimmt, die durch den Parameter *sourceContainer* des Data Format Conversion Patterns spezifiziert wird. Dieses Element wird der lokalen *data container reference variable* zugewiesen. Wenn der SIMPL Core diese Referenz auflöst, ist das Resultat das gewünschte Verzeichnis, in dem das Script sowie die Eingabedatei liegen.

Die IssueCommand Aktivität führt also einen DM Command aus, der zuerst in das Verzeichnis wechselt und anschließend das gewünschte Script startet: *cd [intermediateDirectory] & ?script ?sourceContainer*. Der Platzhalter *?script* entspricht dem Script-Aufruf (z.B. *python tecp2vtk.py*). Die Datei, die konvertiert werden soll, wird durch den Platzhalter *?sourceContainer* bzw. der *data container reference variable*, die diesen Platzhalter ersetzt, referenziert.

Zu beachten ist, dass bei diesem Vorgehen der Parameter *targetContainer* beim Script-Aufruf nicht berücksichtigt wird. Dies geschieht aus folgendem Grund: Im Rahmen dieser Diplomarbeit wird ein sequentielles Data Iteration Pattern mit eingebettetem komplexen Container-to-Container Pattern in die Postprocessing-Phase von Pandas integriert (auf die Transformation des sequentiellen Data Iteration Patterns wird in Abschnitt 6.2.4 eingegangen). Als eingebettete Operation wird ein komplexes Container-to-Container Pattern ausgeführt. Da es beim sequentiellen Data Iteration Pattern nicht möglich ist, für jeden Quell- auch einen Zielcontainer anzugeben, referenziert der Parameter *targetContainer* des

eingebetteten komplexen Container-to-Container Patterns ein Verzeichnis. Es werden also **mehrere** Dateien und nicht mehr wie bisher eine Datei, die im TecPlot-Format vorliegen, in ein referenziertes Verzeichnis kopiert und ins VTK-Format konvertiert. Dieses komplexe Container-to-Container Pattern wird u.a. auf ein einfaches Container-to-Container Pattern, gefolgt von einem Data Format Conversion Pattern abgebildet (vgl. Abschnitt 6.2.2). Das einfache Container-to-Container Pattern kopiert die Datei in das referenzierte Verzeichnis. Das Data Format Conversion Pattern führt die gewünschte Datenformatkonvertierung durch. Da der Parameter *targetContainer* des komplexen Container-to-Container Patterns ein Verzeichnis referenziert, referenziert auch der Parameter *targetContainer* des Data Format Conversion Patterns ein Verzeichnis. Die Ausgabedatei muss also wiederum in dem Verzeichnis liegen, in dem auch die Eingabedatei liegt. Deshalb wird im Rahmen dieser Arbeit für das Data Format Conversion Pattern der Pandas Postprocessing-Phase ein Workflow-Fragment sowie eine zugehörige Transformationsregel definiert, die dann anwendbar ist, wenn das im Resource Management gefundene Script keine Referenz für die Ausgabe erwartet und die konvertierte Datei im selben Verzeichnis wie die Eingabedatei speichert. In Zukunft könnte es für das Data Format Conversion Pattern ein weiteres Workflow-Fragment sowie eine zugehörige Transformationsregel geben, die den Paramter *targetContainer* vollständig berücksichtigt. Aus dieser Beschreibung ergibt sich der folgende Condition Part:

- Condition 1: Der Parameter *sourceContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 2: Der Parameter *targetContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 3: Beide *data container references variables* referenzieren Ordner bzw. Dateien innerhalb eines Dateisystems.
- Condition 4: Bei beiden *data container reference variables* wurde das Element *dataFormat* spezifiziert und dieses ist unterschiedlich.
- Condition 5: Die Datenformate sind im Resource Management registriert und es gibt einen geeigneten Data Transformation Service. Des Weiteren erfolgt die Konvertierung mithilfe eines Scripts.
- Condition 6: Das Script speichert die konvertierte Datei im selben Verzeichniss. Dies ist nötig, da das Workflow-Fragment ansonsten noch eine IssueCommand oder QueryData Aktivität umfassen müsste, die das Resultat an den referenzierten Ort kopiert.
- Condition 7: Beim Script-Aufruf muss *kein* Parameter für die Ausgabe angegeben werden. Wie dies mithilfe der im Resource Management gespeicherten Metadaten geprüft werden kann, wird in Abschnitt 7.3 erläutert.

Wenn diese Bedingungen erfüllt sind, kann der folgende Action Part ausgeführt werden:

- Action 1: Lade das Workflow-Fragment *dataFormatConversion_script.xml* (vgl. Anhang B.3).
- Action 2: Ersetze den Platzhalter ?randomNumber durch ein Zufallszahl.

- Action 3: Ersetze den Platzhalter *?sourceContainer* durch den Wert des Parameters *sourceContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 4: Ersetze alle Vorkommen des Platzhalters *?sourceContainerName* durch den Wert des Parameters *sourceContainer*. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 5: Ersetze den Platzhalter *?dataResource* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird. Dieser Variablenname muss von eckigen Klammern umschlossen sein, damit der SIMPL Core diese Referenz auflöst. Auf dieser Datenquelle wird die IssueCommand Aktivität ausgeführt.
- Action 6: Ersetze alle Vorkommen des Platzhalters *?dataSourceReferenceVariable* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 7: Ersetze den Platzhalter *?script* durch den Script-Aufruf. Diese Metadaten sind im Resource Management hinterlegt.
- Action 8: Ersetze alle Vorkommen des Platzhalters *?dataFormatDir* durch den Wert des Elementes *dataFormat* der *data container reference variable*, die durch den Parameter *sourceContainer* spezifiziert wird.

6.2.4. Das sequentielle Data Iteration Pattern

In der Pandas Preprocessing-Phase wird das sequentielle Data Iteration Pattern genutzt, um mehrere Dateien in das Arbeitsverzeichnis des Pandas-Rechners zu kopieren (vgl. Abschnitt 5.4.3). Des Weiteren kann dieses Datenmangementpattern, wie in Abschnitt 6.2.3 bereits angesprochen, auch in der Pandas Postprocessing-Phase eingesetzt werden, da Pandas nicht nur eine Ergebnisdatei generiert, sondern noch zusätzlich weitere Dateien im TecPlot-Format speichert. Sollen mehrere dieser Dateien auf den Workflow-Rechner kopiert werden, könnte auch an dieser Stelle ein sequentielles Data Iteration Pattern genutzt werden.

In [Ari12] wird erläutert, dass das sequentielle Data Iteration Pattern auf eine ForEach-Schleife abgebildet wird und als Eingabeparameter zwei Parameter erwartet: eine Liste mit *data container reference variables* (die einzelnen Einträge referenzieren die zu kopierenden Dateien) sowie eine weitere *data container reference variable* (in dieser Variablen wird bei jedem Schleifendurchlauf die betrachtete Teilmenge (vgl. Abschnitt 5.4.1.3) gespeichert - im betrachteten Anwendungsfall ist dies genau eine Referenz auf eine Datei). Anschließend wird die vom Nutzer modellierte eingebettete Operation auf diese Teilmenge angewendet.

In Zukunft erwartet das sequentielle Data Iteration Pattern als Eingabeparameter eine Liste mit *data container reference variables* sowie den **Namen** einer, bei der Transformation des Patterns, lokal zu erzeugenden *data container reference variable*. Der Nutzer kann diese Variable bzw. diesen Variablennamen bereits bei der Modellierung der eingebetteten Operation nutzen.

Ebenso kann der Modellierer einen Namen spezifizieren, der bestimmt, wie der Zähler der ForEach-Schleife benannt wird. Des Weiteren wird das sequentielle Data Iteration Pattern in Zukunft auf einen Scope abgebildet. Dieser beinhaltet eine Sequenz, die wiederum eine Assign Aktivität sowie die in Abschnitt 5.4.3 erläuterte ForEach-Schleife umfasst.

Listing 6.2 Beispiel einer Data Container Reference List

```
<conRef:dataContainerReferenceList
    xmlns:conRef="http://org.simpl.core.data.container.reference.list/">
  <GenericContainerReference>
    <simpl:WindowsLocalDataContainerReference>
      <dataSourceReferenceVariable>DataSourceLocal_1</dataSourceReferenceVariable>
      <dataFormat />
      <directory>c:\Users\testdaten</directory>
      <file>2d_wanne.bc</file>
    </simpl:WindowsLocalDataContainerReference>
  </GenericContainerReference>
  <GenericContainerReference>
    <simpl:WindowsLocalDataContainerReference>
      <dataSourceReferenceVariable>DataSourceLocal_2</dataSourceReferenceVariable>
      <dataFormat />
      <directory>c:\Users\testdaten</directory>
      <file>2d_wanne.elems</file>
    </simpl:WindowsLocalDataContainerReference>
  </GenericContainerReference>
</conRef:dataContainerReferenceList>
```

Dies ist nötig, da die vom Modellierer spezifizierte *data container reference list* Referenzen auf Container enthalten kann, die in **verschiedenen** Datenressourcen liegen. Listing 6.2 stellt dies exemplarisch dar: der erste Eintrag verweist auf die *data source reference variable* **DataSourceLocal_1** - der zweite Eintrag hingegen auf die *data source reference variable* **DataSourceLocal_2**.

Bei jeder Iteration wird ein Eintrag dieser Liste selektiert und der lokalen *data container reference variable* zugewiesen. In Abschnitt 6.2.2 wird erläutert, dass lokal erzeugte *data container reference variables* ebenfalls initialisiert werden müssen, damit eingebettete Datenmanagementpatterns benötigte Informationen extrahieren können. Im beschriebenen Anwendungsfall werden die einzelnen Dateien mithilfe eines Container-to-Container Patterns in das Zielverzeichnis auf den Pandas-Rechner kopiert. Problematisch ist die Initialisierung dieser lokalen Variablen: die Frage ist, welcher Wert dem Element *dataSourceReferenceVariable* zugewiesen wird. Theoretisch wäre dies in jedem Schleifendurchlauf eine andere *data **source** reference variable*, was aber bei einer statischen Initialisierung von Variablen nicht berücksichtigt werden kann.

Um dieses Problem zu lösen, wird zusätzlich eine lokale *data **source** reference variable* erzeugt. Das Element *dataSourceReferenceVariable* der lokalen *data container reference variable* verweist dann fest auf diese Variable. In jedem Schleifendurchlauf wird der lokalen *data **source** reference variable* dann ein geeigneter Wert zugewiesen. In Bezug auf Listing 6.2 wäre dies im ersten Durchlauf der Wert der Variablen **DataSourceLocal_1** und im zweiten Schleifendurchlauf der

Wert der Variablen **DataSourceLocal_2**. Dadurch wird über eine Indirektion gewährleistet, dass bei eingebetteten Datenmanagementpatterns nur der Name für die jeweils betrachtete *data source reference variable* ausgelesen werden muss und hier auch immer der richtige, da gleiche Name ausgelesen wird.

Jedoch konnte kein geeigneter XPath Ausdruck gefunden werden, der solch eine Indirektion ermöglicht hätte. Deshalb wird ein Workaround umgesetzt. Es wird eine lokale Kopie der *data container reference list* angelegt. In dieser Liste enthalten die einzelnen Einträge noch zusätzlich die *data source reference* (u.a. den logischen Namen einer Datenquelle, also nicht den Namen der entsprechenden Variablen, sondern direkt deren Wert). Dies ist in Listing 6.3 exemplarisch dargestellt. Auf diese Weise kann bei jeder Iteration der lokalen *data source reference varibale* die zugehörige Datenquelle zugewiesen werden. Des Weiteren können so die referenzierten Container in unterschiedlichen Datenquellen liegen und eingebettete Datenmanagementpatterns trotzdem zur Modellierungszeit transformiert werden.

Listing 6.3 Beispiel einer Data Container Reference List mit zusätzlichen Informationen

```
<conRef:dataContainerReferenceList
    xmlns:conRef="http://org.simpl.core.data.container.reference.list/">
  <GenericContainerReference>
    <simpl:WindowsLocalDataContainerReference>
      <dataSourceReferenceVariable>DataSourceLocal_1</dataSourceReferenceVariable>
      <dataSourceReference>
        <name>Windows Local 1</name>
        <requirements />
        <strategy />
      </dataSourceReference>
      <dataFormat />
      <directory>c:\Users\testdaten</directory>
      <file>2d_wanne.bc</file>
    </simpl:WindowsLocalDataContainerReference>
  </GenericContainerReference>
  <GenericContainerReference>
    <simpl:WindowsLocalDataContainerReference>
      <dataSourceReferenceVariable>DataSourceLocal_2</dataSourceReferenceVariable>
      <dataSourceReference>
        <name>Windows Local 2</name>
        <requirements />
        <strategy />
      </dataSourceReference>
      <dataFormat />
      <directory>c:\Users\testdaten</directory>
      <file>2d_wanne.elems</file>
    </simpl:WindowsLocalDataContainerReference>
  </GenericContainerReference>
</conRef:dataContainerReferenceList>
```

Dieses Vorgehen führt allerdings zu einer Einschränkung. Sollte der Modellierer nach der Transformation des sequentiellen Data Iteration Patterns die ursprüngliche *data container*

reference list noch erweitern bzw. verändern, werden diese Änderungen bei der Ausführung des Workflows *nicht* berücksichtigt, da die lokale Kopie der Liste ja schon erstellt wurde.

Des Weiteren müssen alle Einträge der ursprünglichen *data container reference list* kein oder dasselbe Datenformat aufweisen. Dies ist nötig, damit die lokale *data container reference variable* initialisiert werden kann und eingebettete Datenmanagementpatterns abgebildet werden können. Das komplexe Container-to-Container Pattern z.B. sucht anhand der Datenformate des übergebenen *data container reference variables* einen geeigneten Data Transformation Service (vgl. Abschnitt 6.2.2).

Der Condition Part der zugehörigen Transformationsregel sieht folgendermaßen aus:

- Condition 1: Der Parameter *data* entspricht einer *data container reference list* bzw. deren Namen.
- Condition 2: Für den Parameter *currentContainerName* wurde vom Modellierer ein Wert angegeben (innerhalb des Workflow-Fragments wird eine lokale *data container reference variable* mit diesem Namen erzeugt).

Wenn diese Bedingungen erfüllt sind, kann der folgende Action Part ausgeführt werden.

- Action 1: Lade das Workflow-Fragment *sequentialDIP.xml* (vgl. Anhang B.4).
- Action 2: Ersetze alle Vorkommen des Platzhalters *?randomNumber* durch eine Zufallszahl.
- Action 3: Ersetze alle Vorkommen des Platzhalters *?counterName* durch den Wert des Parameters *counterName*, wenn dieser spezifiziert wurde. Ansonsten ersetze alle Vorkommen durch den Wert *Counter* (dies ist nötig, damit der Laufvariablen der ForEach-Schleife ein Name zugewiesen wird - in WS-BPEL ist dieser standardmäßig *Counter*).
- Action 4: Ersetze alle Vorkommen des Platzhalters *?currentContainerName* durch den vom Modellierer spezifizierten Namen im Parameter *currentContainerName*.
- Action 5: Ersetze alle Vorkommen des Platzhalters *?dataFormat* durch das Datenformat der Container, die mithilfe der vom Modellierer spezifizierten *data container reference list* referenziert werden (vgl. Condition 1).
- Action 6: Ersetze den Platzhalter *?listContent* durch einen Literalwert, der die **lokale** *data container reference list* initialisiert. Der Wert ergibt sich aus den Einträgen der ursprünglichen Liste, die durch den Parameter *data* spezifiziert wird (vgl. Condition 1), in Verbindung mit den zusätzlichen Informationen über die Datenquellen.
- Action 7: Ersetze den Platzhalter *?operation* durch die eingebettete Operation, die der Nutzer modelliert hat.

Der Vollständigkeit halber wird nun noch einmal kurz auf das sequentielle Data Iteration Pattern in der Pandas Postprocessing-Phase eingegangen. Problematisch ist, dass die TecPlot-Dateien im Arbeitsverzeichnis des Pandas-Rechners liegen. Der Verzeichnisname steht zur Modellierungszeit noch nicht fest. Erst zur Laufzeit wird dieses Verzeichnis durch

den Aufruf eines Web Service generiert, der als Ergebnis den Namen des Verzeichnisses zurückgibt. Das Problem ist, dass das eingebettete komplexe Container-to-Container Pattern u.a. auf ein einfaches Container-to-Container Pattern abgebildet wird, welches wiederum einer TransferData Aktivität entspricht (vgl. Abschnitt 6.2.1). Diese Aktivität benötigt für den Kopiervorgang jedoch den vollen Pfad, der zur Modellierungszeit noch nicht feststeht. Dieses Problem wird durch ein Workaround gelöst. Das sequentielle Data Iteration Pattern der Pandas Postprocessing-Phase führt als eingebettete Operation eine Assign Aktivität gefolgt von einem komplexen Container-to-Container Pattern aus. In diesem Szenario wird bei jeder Iteration eine *data container reference variable* der Liste, die durch den Parameter *data* spezifiziert wird, selektiert. Innerhalb der Assign Aktivität wird auf eine global definierte Variable (die Variable *outputDir*) zugegriffen und die selektierte *data container reference variable* vervollständigt (das Element *directory* wird spezifiziert).

6.2.5. Das Multiple Data Transfer Pattern

Um dem Modellierer die Arbeit weiter zu erleichtern, wird im Rahmen dieser Diplomarbeit noch ein weiteres Datenmanagementpattern, das sogennante Multiple Data Transfer Pattern (MDTP), vorgestellt. In Bezug auf die in Abschnitt 3.7 vorgestellte Hierarchie befindet sich dieses Pattern auf der Ebene der zusammengesetzten Datenmanagementpatterns.

Die Idee ist, dass der Nutzer bei der Modellierung des Patterns *n* Container (*n data container reference variables*) auswählt, die in einem referenzierten Container bereitgestellt werden sollen. Um dieses Ziel zu spezifizieren, wählt der Nutzer ebenfalls eine *data container reference variable* aus. So könnte der Nutzer z.B. *n* Dateien auswählen, die in einem anderen Verzeichnis (Zielcontainer) bereitgestellt werden sollen. Bei der Modellierung wird eine *data container reference list* (vgl. Abschnitt 6.2.4) erzeugt, die die *n* Quellcontainer referenziert, und als Kindelement des Multiple Data Transfer Patterns (XML-Element *dataContainerReferenceList*) gespeichert wird. Die *data container reference Variable*, die den Zielcontainer referenziert, wird mithilfe des Parameters *targetContainer* festgehalten.

Der Modellierer wählt in der Modellierungsumgebung also nacheinander *n data container reference variables* aus. Wenn der erste Quellcontainer ausgewählt wurde, wird die benötigte *data container reference list* erzeugt und als Element im Pattern gespeichert. Werden weitere Quellcontainer ausgewählt, wird diese Liste erweitert bzw., wenn ein Container durch den Modellierer entfernt wird, um den entsprechenden Eintrag verkleinert. Problematisch ist wiederum, dass keine Datenflussanalyse durchgeführt wird und nur die Initalwerte der jeweiligen *data container reference variables* bei der Erzeugung der *data container reference list* berücksichtigt werden.

Bei der Transformation wird dieses Datenmanagementpattern auf einen Scope abgebildet. Dieser Scope besteht aus einer Sequenz, die wiederum eine Assign Aktivität sowie ein sequentielles Data Iteration Pattern mit eingebettetem Container-to-Container Pattern beinhaltet. Der Scope wird benötigt, da eine lokale *data container reference list* erzeugt werden muss. Über die Assign Aktivität wird dieser lokalen Liste ein Wert (fixed value to variable) zugewiesen. Dieser Wert ergibt sich aus dem XML-Element (dataContainerRefernceList),

das als Kindelement des Multiple Data Transfer Patterns bei der Modellierung gespeichert wurde. Das sequentielle Data Iteration Pattern iteriert über diese *data container reference list* und das eingebettete Container-to-Container kopiert in jedem Iterationsschritt ein Objekt dieser Liste in den referenzierten Zielcontainer.

Aus dieser Beschreibung resultiert der folgende Condition Part:

- Condition 1: Der Wert des Parameters *containerReferences* entspricht einer *data container reference list*, die n $(n > 0)$ *data container reference variables* beinhaltet.
- Condition 2: Der Parameter *targetContainer* entspricht einer *data container reference variable* bzw. deren Namen.

Wenn diese Bedingungen erfüllt sind, kann der folgende Action Part ausgeführt werden:

- Action 1: Lade das Workflow-Fragment *MDT.xml* (vgl. Anhang B.5).
- Action 2: Ersetze den Platzhalter *?randomNumber* durch eine Zufallszahl.
- Action 3: Ersetze den Platzhalter *?listContent* durch den Wert des Kindelementes *dataContainerReferenceList* des Multiple Data Transfer Patterns. Dadurch wird die lokale *data container reference list* bzw. Variable initialisiert.
- Action 4: Ersetze den Platzhalter *?targetContainer* durch den Wert des Parameters *targetContainer*. Dieser Variablenname muss von eckigen Klammern umschlossen sein.

Zu beachten ist, dass dieses Pattern in der Pandas Postprocessing-Phase **nicht** verwendet werden kann, da dieses Pattern auf ein Workflow-Fragment abgebildet wird, das u.a. ein sequentielles Data Iteration Pattern mit eingebettetem Container-to-Container Pattern beinhaltet. In Abschnitt 6.2.4 wird jedoch erläutert, dass das sequentielle Data Iteration Pattern in Bezug auf die Pandas Postprocessing-Phase noch eine Assign Aktivität umfassen muss, um die Referenzen zu vervollständigen.

6.3. Datenmanagementpatterns in der Pandas-Matlab Kopplung

In Abschnitt 4.3 wird auf die Pandas-Matlab Kopplung, eine multiskalare Simulation, eingegangen. Es wird erläutert, dass dieses Szenario drei Workflows umfasst: der Pandas-Bone, der Data-Manager (auch Matlab-Dispatcher genannt) sowie der Matlab-Bone.

Der Data-Manager Workflow bereitet die Matlab-Instanzen auf den entfernten Rechnern vor und berechnet die Verteilung der Gitterelemente bzw. Gausspunkte auf diese Matlab-Instanzen. Anschließend wird der Matlab-Bone so oft gestartet, wie es Matlab-Instanzen gibt. Innerhalb des Matlab-Bone Workflows werden aus der Pandas-Datenbank die benötigten CSV-Dateien auf dem Pandas-Rechner erstellt und in das Arbeitsverzeichnis auf dem jeweiligen Matlab-Rechner kopiert. Nachdem die Matlab-Simulation beendet wurde, werden die Ergebnisdaten wiederum auf den Pandas-Rechner zurückkopiert und gemerged (vgl. Abschnitt 4.3). Im Folgenden wird nun erläutert, wie sich Datenmanagementpatterns in diesem Anwendungsfall einsetzen lassen.

6.3.1. Das parallele Data Iteration Pattern

Der Data-Manager Workflow bietet die Möglichkeit ein paralleles Data Iteration Pattern (vgl. Abschnitt 5.4.1.3) einzusetzen. Dieses Pattern iteriert über die Datenmenge in der Pandas-Datenbank, die auf die einzelnen Matlab-Instanzen aufgeteilt werden soll. In Abschnitt 5.4.1.3 wird erläutert, dass das parallele Data Iteration Patten die drei Phasen *Split Stage*, *Operation Stage* und *Merge Stage* umfasst. In der *Split Stage* wird der Simulationsraum über ein eingebettetes Data Split Pattern (vgl. Abschnitt 5.4.1.1) aufgeteilt. Die vom Nutzer spezifizierte Operation wird in der *Operation Stage* ausgeführt, bevor in der *Merge Stage* die einzelnen Teilmengen wiederum durch ein eingebettetes Data Merge Pattern (vgl. Abschnitt 5.4.1.1) zusammengefasst werden. Um das parallele Data Iteration Pattern zu integrieren, sind Anpassungen am Data-Manager Workflow nötig, die in Abbildung 6.5 dargestellt werden. Zu beachten ist, dass im Zuge der Integration der Datenmanagementpatterns in das SIMPL-Rahmenwerk auch die Workflows der Pandas-Matlab Kopplung auf *data source references* sowie *data container references* umgestellt werden. Problematisch ist hierbei, dass innerhalb der Workflows Web Services durch Invoke Aktivitäten (z.B. die Aktivität *setMatlabPath* in der Abbildung) aufgerufen werden. Diesen Web Services können solche Referenzen *nicht* übergeben werden. Aus diesem Grund werden *data container references*, wenn diese einem solchen Web Service übergeben werden müssten, bei der Ausführung des Workflows im Workflow über proprietäre Assign Aktivitäten aufgelöst. Ebenfalls werden, wenn Datenquellen mithilfe von *data source references* referenziert werden, alle weiteren benötigten Informationen zu diesen Datenressourcen aus dem Resource Management in den Workflow geladen.

Der Data-Manager Workflow erwartet in Zukunft als Eingabe u.a. eine Liste vom Typ *MatlabNodeReferenceListType*. Diese Liste bestimmt die Rechner, auf denen die einzelnen Matlab-Instanzen ausgeführt werden. Das zugehörige XML Schema ist in Listing 6.4 dargestellt und erlaubt es, beliebig viele Elemente vom Typ *MatlabNodeType* zu erzeugen. Dabei entspricht ein Element von diesem Typ einem Matlab-Rechner. Zusätzliche Informationen werden über drei Kindelemente festgehalten. Durch das erste Kindelement wird die eigentliche Datenressource, also das entsprechende Dateisystem, über eine *data source reference* referenziert. Das Element *SimPath* referenziert einen Datencontainer in dieser Datenressource, und zwar das Arbeitsverzeichnis auf dem Matlab-Rechner. Der Datencontainer bzw. das Verzeichnis, in dem sich Matlab befindet, wird mithilfe des Elementes *MatlabPath* festgehalten. Die beiden letzten Elemente sind vom Typ *FileReferenceType*. Dieser Typ beinhaltet wiederum das Element *FileSystemDataContainerReference*. Dieses Element wird in der *simpl.xsd* definiert, die automatisch aus dem Resource Management geladen wird, wenn eine DM Aktivität bzw. ein Datenmanagementpattern hinzugefügt wird. Dieses Element bzw. der zugehörige Typ (*FileSystemDataContainerReferenceType*) wurde im Rahmen dieser Diplomarbeit neu definiert und kann als Basistyp für alle *data container references* verwendet werden, die auf Dateien in beliebigen Dateisystemen verweisen. Der Typ erweitert den Basistyp *LocalDataContainerReferenceType*, fügt jedoch keine neuen Elemente bzw. Attribute hinzu. Werden Objekte innerhalb eines Dateisystems referenziert, sind die *data container reference variables* vom Typ *WindowsLocalDataContainerReferenceType* bzw. *UnixLocalDataContainerReferenceType*. Diese beiden Typen erweitern den in der *MatlabNodeReferenceList* verwendeten Basistyp *FileSystemDataContainerReferenceType*. Auf diese Weise kann sichergestellt werden, dass nur Objekte innerhalb eines

Abbildung 6.5.: Der überarbeitete Data-Manager Workflow inklusive dem parallelen Data Iteration Pattern

Listing 6.4 Die MatlabNodeReferenceList Definition

```
<xsd:schema targetNamespace="http://org.simpl.core.matlab.node.reference.list/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://org.simpl.core.matlab.node.reference.list/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:simpl="http://www.example.org/simpl">
  <xsd:import schemaLocation="simpl.xsd" namespace="http://www.example.org/simpl"/>
  <xsd:element name="MatlabNodeReferenceList" type="MatlabNodeReferenceListType"/>
  <xsd:complexType name="MatlabNodeReferenceListType">
    <xsd:sequence>
      <xsd:element name="MatlabNode" type="MatlabNodeType" minOccurs="1"
          maxOccurs="unbounded"/>
      <xsd:element name="SimNode" type="SimNodeType" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="MatlabNodeType">
    <xsd:sequence>
      <xsd:element ref="simpl:DataSourceReference"/>
      <xsd:element maxOccurs="1" minOccurs="1" name="SimPath" type="FileReferenceType"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="MatlabPath" type="FileReferenceType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="FileReferenceType">
    <xsd:sequence>
      <xsd:element ref="simpl:FileSystemDataContainerReference"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SimNodeType">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" ref="simpl:DataSourceReference" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Dateisystems referenziert werden können (vgl. Abbildung 6.6). Zusätzlich werden in dieser Liste noch Informationen über den Pandas-Rechner, auf dem die Simulation ausgeführt wird, gespeichert. Dies geschieht mithilfe einer *data source reference* im Element *SimNode*.

Zu Beginn des Workflows wird die dem Data-Manager Workflow übergebene *matlab node reference list* aus der Input-Variablen extrahiert und mithilfe einer Assign Aktivität einer globalen Variablen zugewiesen. Anschließend folgt das parallele Data Iteration Pattern.

Dazu erwartet das parallele Data Iteration Pattern folgende Eingabeparameter:

- **data**: Über diese Datenmenge soll iteriert werden.
- **referenceList**: Dieser Parameter bestimmt die Instanzen (im betrachteten Anwendungsfall die Matlab-Rechner), auf denen die jeweilige Teilmenge bereitgestellt werden soll. Die Anzahl der Instanzen in dieser Liste bestimmt dabei die Anzahl der Teilmengen.
- **simulationId**: Dieser Parameter bestimmt die Simulations-Id, die berücksichtigt werden soll. Dieser Parameter ist nötig, da mithilfe des parallelen Data Iteration Patterns

in der Regel ein Simulationsraum aufgeteilt werden soll (im betrachteten Anwendungsfall sollen die Daten einer Simulation, die in der Pandas-Datenbank gespeichert sind. aufgeteilt werden).

- **mode**: Dieser Parameter bestimmt, wie die Daten aufgeteilt werden sollen. Im betrachteten Anwendungsfall können z.B. entweder die Gitterelemente oder die Gausspunkte gleichmäßig verteilt werden.
- **timeStep**: Dieser Parameter bestimmt, welchen Zeitschritt das parallele Data Iteration Pattern bei der Aufteilung des Simulationsraums betrachten soll (so wären z.B. *first* oder *last* denkbar). Dieser Parameter beschreibt also eine weitere Filter-Operation, da *nur* die Daten des entsprechenden Zeitschritts berücksichtigt werden. Dieser Parameter ist optional.
- **counterName**: In der Regel wird das parallele Data Iteration Pattern auf eine Schleife abgebildet. In WS-BPEL kann z.B. bei einer ForEach-Schleife der Name des Zählers festgelegt werden. Mithilfe dieses Parameters, der optional ist, kann *der Nutzer* den Namen festlegen. Dadurch kann er ihn bereits bei der Modellierung des parallelen Data Iteration Patterns in den darin eingebetteten Operationen wieder verwenden (z.B. in den oben angesprochenen Invoke Aktivitäten).

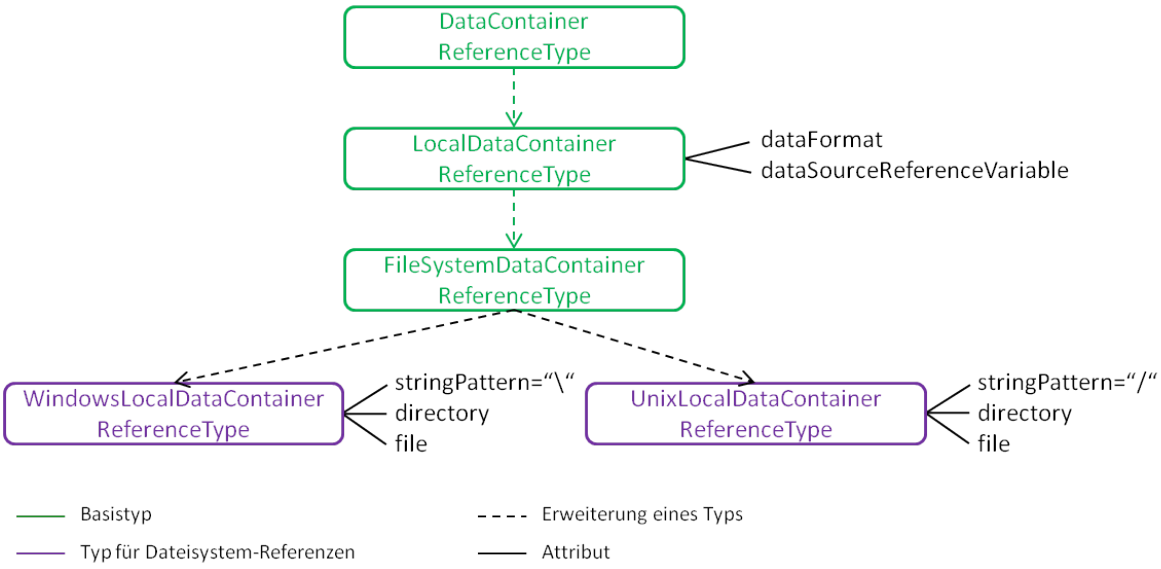

Abbildung 6.6.: Typen für die Referenzierung von Objekten in einem Dateisystem

Bevor die Transformation des parallelen Data Iteration Patterns im Detail betrachtet wird, wird der generelle Ablauf, der nach dieser Transformation ausgeführt werden soll, beschrieben. In Abschnitt 4.3 wird erläutert, dass in der Pandas Datenbank Informationen über die Gitterelemente sowie die darin enthaltenen Gausspunkte gespeichert werden. Abbildung 6.7

zeigt exemplarisch den Inhalt der Tabelle *gausspunkte*. *Ein* Gausspunkt innerhalb eines Gitterelementes wird wiederum durch mehrere Variablen beschrieben. Jeweils eine Variable eines Gausspunktes wird durch eine Zeile in der Tabelle beschrieben: in der ersten Zeile wird z.B. der Variablen *EPL_V* (*name*) des Gausspunktes mit der Nummer 0 (*gaussnr*) im Gitterelement mit der Nummer 2 (*elementnr*) ein Wert (*value*) zugewiesen - in der vierten Zeile hingegen der Variable *EPL11*. Zusätzlich werden in jeder Zeile noch die Simulations-Id (*sid*), der Simulationszeitschritt (*stepnr*), der Index der Variablendefinition in der Physik (*physidx*), der Index des Elementes (*index*), ein Flag (dieser bestimmt, ob es sich um eine historische Variable handelt - (*hist*)), ein ausführlicherer Variablenname (*nick*) sowie die Skalierung der Variablen (*scale*) beschrieben [Dor11]. Verweist eine *data container reference variable* auf eine Tabelle, die dieses Relationenschema implementiert, wird davon ausgegangen, dass für das Element *dataFormat* das Datenformat *PandasGausspointTable* spezifiziert wurde.

	sid [PK] bigint	stepnr [PK] int	elementnr [PK] intege	gaussnr integer	physidx integer	index integer	hist boolean	name [PK] text	nick text	scale doubl	useit boolean	value double precis
1	1314908303110	10	2	0	4	6	FALSE	EPL_V	Norm plastic s	1	TRUE	0
2	1314908303110	10	2	1	4	6	FALSE	EPL_V	Norm plastic s	1	TRUE	0
3	1314908303110	10	2	2	4	6	FALSE	EPL_V	Norm plastic s	1	TRUE	0
4	1314908303110	10	2	0	24	0	TRUE	EPL11	Plastic strain	1	FALSE	-0.6952997
5	1314908303110	10	2	1	24	0	TRUE	EPL11	Plastic strain	1	FALSE	-0.6723437
6	1314908303110	10	2	2	24	0	TRUE	EPL11	Plastic strain	1	FALSE	-0.7060719
7	1314908303110	10	2	0	27	3	TRUE	EPL12	Plastic strain	1	FALSE	-0.6959027
8	1314908303110	10	2	1	27	3	TRUE	EPL12	Plastic strain	1	FALSE	-0.6706153
9	1314908303110	10	2	2	27	3	TRUE	EPL12	Plastic strain	1	FALSE	-0.704594
10	1314908303110	10	2	0	25	1	TRUE	EPL22	Plastic strain	1	FALSE	-0.0004384028
11	1314908303110	10	2	1	25	1	TRUE	EPL22	Plastic strain	1	FALSE	9.844447e-005
12	1314908303110	10	2	2	25	1	TRUE	EPL22	Plastic strain	1	FALSE	-6.03546e-005
13	1314908303110	10	2	0	26	2	TRUE	EPL33	Plastic strain	1	FALSE	-1.624371
14	1314908303110	10	2	1	26	2	TRUE	EPL33	Plastic strain	1	FALSE	-1.563035
15	1314908303110	10	2	2	26	2	TRUE	EPL33	Plastic strain	1	FALSE	-1.642569
16	1314908303110	10	2	0	22	7	FALSE	ESTIM	Error indicato	1	TRUE	0.04429312
17	1314908303110	10	2	1	22	7	FALSE	ESTIM	Error indicato	1	TRUE	0.04429312
18	1314908303110	10	2	2	22	7	FALSE	ESTIM	Error indicato	1	TRUE	0.04429312
19	1314908303110	10	2	0	28	4	TRUE	LAMPL	Plastic multipli	1	FALSE	1.93291e-010
20	1314908303110	10	2	1	28	4	TRUE	LAMPL	Plastic multipli	1	FALSE	1.93291e-010
21	1314908303110	10	2	2	28	4	TRUE	LAMPL	Plastic multipli	1	FALSE	1.93291e-010

Abbildung 6.7.: Exemplarischer Inhalt der Tabelle *gausspunkte* in der PandasDB

Bei der Pandas-Matlab Kopplung müssen diese Daten (bzw. die Daten eines Zeitschrittes einer Simulation) in n CSV-Dateien exportiert und auf die n Matlab-Instanzen verteilt werden. Um dies zu realisieren, werden die in Frage kommenden Daten (also die Daten der Simulation, die durch den Parameter *simulationId* bestimmt werden, in Bezug auf einen Zeitschritt, der durch den Paramter *timeStep* festgelegt wird - *sid=x* und *stepnr=y*) über einen Web Service in ein neues Schema überführt. Die Daten werden bereits an dieser Stelle in Bezug auf die Simulations-Id und den Zeitschritt gefiltert, da dadurch weniger Datensätze konvertiert werden müssen. Abbildung 6.8 zeigt exemplarisch das Ergebnis einer solchen Konvertierung. Eine Zeile beschreibt genau *einen* Gausspunkt inklusive **aller** Variablen bzw. deren Werte. D.h., es gibt nun für jede dieser Variablen eine Spalte in der Tabelle. Der Vorteil

dieser Konvertierung ist, dass die zu exportierenden Daten bestimmt werden und diese der von Matlab gewünschten Form entsprechen (eine Zeile beinhaltet alle Variablenwerte).

	sid [PK] bigint	elementnr [PK] intege	gaussnr [PK] inte	stepid [PK] in	EPL12 double	LAMPL double	SIG12 double	WF2 double	EPL_V double	SIG33 double	TSE11 double	EPL33 double	WF1 double	SIG11 double
1	1314908303110	2	0	10	-0.69590	1.93291e	-0.00043	3.8535	0	-9.07256	-0.69529	-1.6243	1.9329	-9.07196
2	1314908303110	2	1	10	-0.67061	1.93291e	9.844447	3.8535	0	-9.10588	-0.67234	-1.5630	1.9329	-9.10761
3	1314908303110	2	2	10	-0.70459	1.93291e	-6.03546	3.8535	0	-9.06038	-0.70607	-1.6425	1.9329	-9.06186
4	1314908303110	3	0	10	-1.65414	-1.06065	-0.00065	2.7886	0	-7.80009	-1.65505	-3.8587	-1.0606	-7.801
5	1314908303110	3	1	10	-1.78043	-1.06065	0.001187	2.7886	0	-7.62294	-1.78159	-4.1531	-1.0606	-7.62410
6	1314908303110	3	2	10	-1.53441	-1.06065	-0.00149	2.7886	0	-7.95153	-1.53495	-3.5797	-1.0606	-7.95207
7	1314908303110	4	0	10	-1.22286	-1.0649e-	0.000243	2.0932	0	-8.37515	-1.2233	-2.8528	-1.0649	-8.37558
8	1314908303110	4	1	10	-1.31843	-1.0649e-	0.002635	2.0932	0	-8.23838	-1.31865	-3.0761	-1.0649	-8.23859
9	1314908303110	4	2	10	-1.13479	-1.0649e-	-0.00129	2.0932	0	-8.48510	-1.1354	-2.6472	-1.0649	-8.48572

Abbildung 6.8.: Informationen über Elemente sowie Gausspunkte in der neu erzeugten Tabelle

Beim Export in eine CSV-Datei werden nur die Variablenwerte in die Datei geschrieben. Zusätzlich werden noch zwei weitere Dateien erzeugt: in der ersten Datei werden für jede Zeile die Werte des zusammengesetzten Primärschlüssels gepeichert (*sid, elementnr, gaussnr, stepid*) - in der zweiten Datei die Namen der Attribute (z.B. *EPL12, LAMPL, ... ,WF1, SIG11*). Durch diese zusätzlichen Informationen ist es später möglich eine durch Matlab geänderte CSV-Datei wiederum in die Datenbank zu importieren und die einzelnen Variablenwerte zu aktualisieren.

Zu beachten ist, dass die Aufteilung auf n CSV-Dateien auf unterschiedliche Weise erfolgen kann: So können z.B. die Gitterelemente oder die Gausspunkte gleichmäßig verteilt werden. Im ersten Fall wird dazu die Anzahl der unterschiedlichen Elemente bestimmt - im zweiten Fall hingegen die Anzahl der unterschiedlichen Gausspunkte.

Bevor die erzeugten CSV-Dateien auf die Matlab-Rechner kopiert werden können, muss auf den beteiligten Rechnern das benötigte Arbeitsverzeichnis erzeugt und notwendige Initialisierungsdateien dorthin kopiert werden. Dies geschieht durch die in Abbildung 6.5 dargestellten Aktivitäten zur Vorbereitung einer Matlab-Instanz. Anschließend können die erzeugten CSV-Dateien aus der Pandas-Datenbank kopiert werden. Danach wird der Matlab-Bone Workflow gestartet. Dieser startet wiederum Matlab, das die gewünschte Simulation ausführt. Zuletzt werden die Ergebnisdateien wieder auf den Pandas-Rechner kopiert und in die Datenbank importiert.

Die beschriebene Konvertierung, der Export in eine CSV-Datei sowie der anschließende Import werden durch Web Services realisiert. Diese wurden bereits implementiert und werden im Rahmen dieser Diplomarbeit nur angepasst. Es werden Web Services und keine DM Aktivitäten (z.B. eine IssueCommand Aktivität, die ein SQL-Ausdruck ausführt, der Daten in eine CSV-Datei exportiert) genutzt, da u.a. zusätzliche Dateien, die die Primärschlüssel und Variablennamen beinhalten, erzeugt werden müssen, um die Daten später wieder zu importieren.

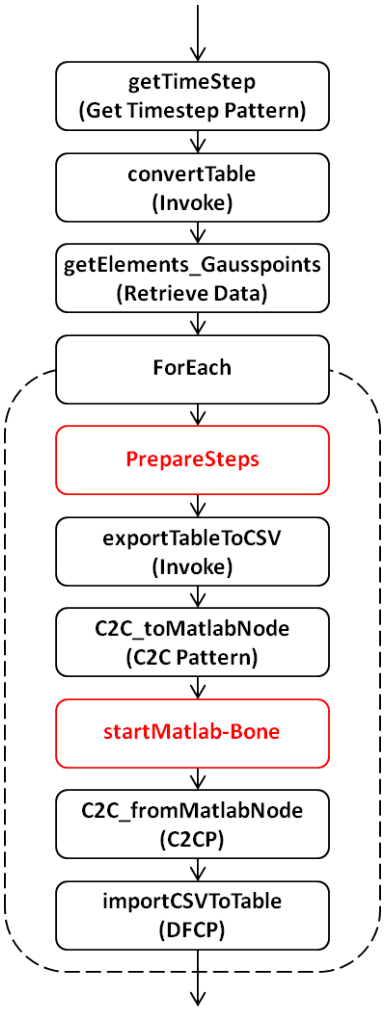

Abbildung 6.9.: Das parallele Data Iteration Pattern nach der Transformation

Nachdem das Vorgehen bisher grob beschrieben wurde, wird die Transformation des parallelen Data Iteration Patterns auf ein ausführbares Workflow-Fragment nun genauer betrachtet. Leider ist das in [Dor11] beschriebene Web Service Interface für die Pandas-Matlab Kopplung momentan nicht lauffähig. Aus diesem Grund werden die Invoke Aktivitäten, die Funktionen von diesem Interface nutzen, durch Empty Aktivitäten ersetzt. Des Weiteren startet auch der Matlab-Bone Workflow keine Matlab-Simulation. Stattdessen wird ein Web Service aufgerufen, der die kopierte CSV-Datei einliest, auf die einzelnen Werte einen konstanten Betrag addiert und diese anschließend wieder in eine neue CSV-Datei, die Datei *output.csv*, schreibt. Auf diese Weise kann die Aufteilung des Simulationsraums und der anschließende Merge simuliert werden.

Bei der Transformation des parallelen Data Iteration Patterns wird das Workflow-Fragment *parallelDIP_pandas_gausspoint_table.xml* (dieses ist auf der Abgabe-CD zu finden) geladen und

die Platzhalter werden ersetzt. Das resultierende Workflow-Fragment ist in Abbildung 6.9 dargestellt (die vom Nutzer modellierte eingebettete Operation ist in rot dargestellt). Aufgrund der Komplexität dieses Workflow-Fragments ist diese Abbildung stark vereinfacht (u.a. fehlen Scope und Assign Aktivitäten). Aus diesem Grund wird im Folgenden auch nicht auf die Definition lokaler Variablen und deren Wertezuweisung eingegangen (dies funktioniert prinzipiell wie in Abschnitt 6.2 beschrieben).

Zuerst müssen die Daten in der Pandas-Datenbank in das neue Schema (Datenformat *GausspointInstanceTable*) überführt werden. Dazu ruft die Invoke Aktivität *convertTable* die Operation *createNewGausspointInstanceTable* des Web Service *TableToCSVTransformer* auf. Diese Operation erwartet als Eingabeparameter u.a. die Simulations-Id sowie den Zeitschritt. Durch diese beiden Parameter werden die entsprechenden Zeilen der Tabelle (vgl. Abbildung 6.7) selektiert, die in das neue Relationenschema überführt werden sollen. Die Simulations-Id wird durch den Parameter *simulationId* und der Zeitschritt durch den Parameter *timeStep* des parallen Data Iteration Patterns bestimmt. Für den Zeitschritt erwartet der Web Service eine konkrete Zahl. Der Modellierer kann bei der Modellierung des parallelen Data Iteration Patterns jedoch auch Ausdrücke wie z.B. *first* oder *last* angeben. Aus diesem Grund wird zuvor ein *Get Timestep Pattern* ausgeführt. Dieses Pattern liefert für einen Ausdruck wie z.B. *first* oder *last* eine konkrete Zahl. Diese Zahl wird in einer vom Modellierer festgelegten Variablen gespeichert. Der Modellierer spezifiziert diese Variable mithilfe des Parameters *targetVariable* des Get Time Step Patterns (bzw. wird im betrachteten Anwendungsfall diese Variable bei der Transformation des parallelen Data Iteration Patterns lokal erzeugt und dem Get Time Step Pattern übergeben). In Abschnitt 6.3.2 wird dieser Parameter sowie das gesamte Pattern detailliert beschrieben.

Anschließend werden Informationen über die Gitterelemente bzw. Gausspunkte mithilfe einer RetrieveData Aktivität (*getElements_Gausspoints*) in den Workflow geladen. Diese Informationen werden benötigt, um die Anzahl der Gitterelemente bzw. Gausspunkte zu bestimmen und deren Aufteilung auf die einzelnen Matlab-Instanzen zu berechnen. Die Art und Weise, wie die Daten aufgeteilt werden, legt der Modellierer mithilfe des Parameters *mode* des parallelen Data Iteration Patterns fest (z.B. *elements equally distributed* oder *gausspoints equally distributed*). Das Statement, das die RetrieveData Aktivität ausführt ist von folgender Form: **select distinct ?keys from [intermediateTable] order by ?keys**. Die Variable *intermediateContainer* referenziert die Tabelle mit den Daten. Der Platzhalter *?keys* bestimmt die Attribute bzw. Spalten, die für den jeweiligen Fall berücksichtigt werden sollen. Sollen die Gitterelemente gleichmäßig aufgeteilt werden, müssen an dieser Stelle die Spalten *sid* und *elementnr* berücksichtigt werden. Im anderen Fall, wenn die Gausspunkte gleichmäßig aufgeteilt werden sollen, muss zusätzlich noch die Spalte *gaussnr* mit einbezogen werden. Doch woher kommen diese Informationen bzw. Metadaten?

In Zukunft können Metadaten über Datenformate von entsprechenden Datencontainern im Resource Management gespeichert werden. Dazu wird ein Datenformat (z.B. das Datenformat *GausspointInstanceTable*) mit einem Term (z.B. dem Term *gausspoints*) in Bezug gesetzt. Zusätzlich werden noch die eigentlichen Metadaten wie z.B. die Spaltennamen, die in unserem Beispiel die Gitterelemente oder Gausspunkte eindeutig identifizieren, in einer

XML Struktur gespeichert. In Abschnitt 7.3 wird die konkrete Umsetzung der Speicherung dieser Metadaten genauer betrachtet.

Listing 6.5 Exemplarisches Ergebnis der RetrieveData Aktivität *getElements_Gausspoints*

```
<sdo:dataObject xmlns:sdo="commonj.sdo"
    xmlns:simpl="http://www.example.org/tRelationalDataFormattRelationalDataFormat"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" dataFormat="RDBDataFormat"
    xsi:type="simpl:tRelationalDataFormat">
  <dataFormatMetaData>
    <dataSource>LocalPandas</dataSource>
  </dataFormatMetaData>
  <table>
    <rdbTableMetaData catalog="" name="" schema="">
      <columnType columnName="sid" isPrimaryKey="true">int8(19)</columnType>
      <columnType columnName="elementnr" isPrimaryKey="true">int4(10)</columnType>
      <columnType columnName="gaussnr" isPrimaryKey="true">int4(10)</columnType>
    </rdbTableMetaData>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">2</column>
      <column name="gaussnr">0</column>
    </row>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">2</column>
      <column name="gaussnr">1</column>
    </row>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">3</column>
      <column name="gaussnr">0</column>
    </row>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">3</column>
      <column name="gaussnr">1</column>
    </row>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">4</column>
      <column name="gaussnr">0</column>
    </row>
    <row>
      <column name="sid">1314908303110</column>
      <column name="elementnr">4</column>
      <column name="gaussnr">1</column>
    </row>
  </table>
</sdo:dataObject>
```

Wenn das parallele Data Iteration Pattern abgebildet wird, wird der vom Modellierer über den Parameter *mode* festgelegte Modus auf einen solchen Term abgebildet. In Verbindung mit

dem Datenformat (dieses wird durch die zugehörige *data container reference variable* bestimmt) können dann die benötigten Metadaten im Resource Management abgefragt werden und der Platzhalter *?keys* kann entsprechend ersetzt werden. Listing 6.5 zeigt ein exemplarisches Ergebnis der RetrieveData Aktivität *getElements_Gausspoints*. In dieser XML RowSet Struktur sind Informationen über sechs unterschiedliche Gausspunkte (die Spalten *sid*, *elementnr* und *gaussnr* wurden berücksichtigt) gespeichert.

Anschließend wird eine parallele ForEach-Schleife ausgeführt. Die Anzahl der Iterationen bestimmt die *matlab node reference list*, die in diesem Anwendungsfall für den Parameter *referenceList* angegeben werden muss. Im Folgenden wird eine Iteration dieser parallelen Schleife erläutert.

Zuerst wird die Aktivität *PrepareSteps* ausgeführt. Diese Aktivität führt die in Abbildung 6.5 dargestellten Aktivitäten zur Vorbereitung einer Matlab-Instanz aus. Danach wird durch die Invoke Aktivität *exportTableToCSV* die Operation *exportTableToCSVFile* des Web Service *TableToCSVTransformer* aufgerufen. Diese Web Service Operation generiert die benötige CSV-Datei für die jeweilige Matlab-Instanz, speichert diese auf dem Pandas-Rechner und erwartet u.a. die folgenden Eingabeparameter: elem_min, elem_max, gauss_min und gauss_max. Diese Werte bestimmen die Gitterelemente bzw. Gausspunkte, die von der jeweiligen Matlab-Instanz berechnet werden sollen. Zuvor müssen jedoch noch die konkreten Werte für die aktuelle Iteration berechnet werden. Mithilfe der Aktivität *Get Elements/Gausspoints* wurden Informationen über die verschiedenen Gitterelemente bzw. Gausspunkte geladen. Aus dieser XML RowSet Struktur wird die Gesamtanzahl der Gitterelemente bzw. Gausspunkte (die Anzahl der *row* Elemente) sowie die Anzahl der Elemente pro Matlab-Instanz (die Gesamtanzahl wird durch die Zahl der Matlab-Instanzen geteilt) mithilfe einer Assign Aktivität berechnet. Nun werden zwei Fälle unterschieden:

- **Fall 1 - die Gausspunkte sollen gleichmäßig verteilt werden:** Sollen z.B. sechs Gausspunkte (vgl. Listing 6.5) auf drei Matlab-Instanzen aufgeteilt werden, muss jede Matlab-Instanz zwei Gausspunkte berechnen.

  Um die Werte elem_min, elem_max, gauss_min und gauss_max für den Web Service Aufruf zu bestimmen, werden die Attributwerte aus der XML RowSet Struktur gelesen. Bei der ersten Iteration werden dazu die *row* Elemente an den Positionen eins und zwei, bei der zweiten Iteration an den Positionen drei und vier und bei der dritten Iteration an den Positionen fünf und sechs genutzt. Das bedeutet, dass bei der ersten Iteration das erste *row* die **unteren** Grenzen bestimmt. Der Wert elem_min entspricht dem Kindelement mir dem Namen *elemntnr*. Das Kindelement *gaussnr* bestimmt den Wert gauss_min. Zur Berechnung der oberen Grenzen wird das zweite *row* Element genutzt. Das Kindelement *elementnr* bestimmt den Wert elem_max und das Kindelement *gaussnr* den Wert gauss_max. Bei den weiteren Iterationen sind es jeweils die nächsten *row* Elemente. Die Operation *exportTableToCSV* des Web Service kann mithilfe dieser Werte die zu exportierenden Zeilen bestimmen. Wenn elem_min und elem_max unterschiedliche Werte aufweisen, wird ein Statement folgender Form ausgeführt: **SELECT * FROM** *table* **(WHERE elementnr BETWEEN** *elem_min* **+ 1 AND** *elem_max* **- 1) OR (elementnr =** *elem_min* **AND gaussnr >=** *gauss_min***) OR (elementnr =** *elem_max* **AND gaussnr <=** *gauss_max***)**. Dieses Statement würde, wenn elem_min und elem_

gleiche Werte aufweisen, in manchen Fällen eine falsche Datenmenge bestimmen. Deshalb wird in diesem Fall ein Statement folgender Form ausgeführt: **SELECT * FROM** *table* **(WHERE elementnr =** *elem_min* **AND gaussnr BETWEEN** *gauss_min* **AND** *gauss_max***)**.

- **Fall 2 - die Gitterelemente sollen gleichmäßig verteilt werden:** In diesem Fall ist das Vorgehen sehr ähnlich. Jedoch enthalten die einzelnen *row* Elemente das Kindelement mit dem Namen *gaussnr* nicht, die auch nicht benötigt werden. Deshalb werden für die Eingabeparameter *gauss_min* und *gauss_max* keine Werte spezifiziert. Die Operation des Web Service verwendet das folgende Statement, um die gewünschten Zeilen zu extrahieren: **SELECT * FROM** *table* **WHERE (elementnr BETWEEN** *elem_min* **AND** *elem_max***)**.

Um die Anzahl der Gitterelemente bzw. Gausspunkte zu berechnen, die eine Matlab-Instanz berechnen soll, wird, wie bereits erläutert, die Gesamtanzahl durch die Anzahl der Matlab-Instanzen geteilt. Das Resultat ist nicht immer eine Zahl ohne Nachkommastellen. In so einem Fall wird nach oben gerundet. Dies führt allerdings dazu, dass die letzte Matlab-Instanz weniger Gitterelemente bzw. Gausspunkte als die anderen $n - 1$ Instanzen berechnen muss.

Nachdem die CSV-Datei für die jeweilige Matlab-Instanz erzeugt wurde, wird die CSV-Datei mithilfe eines einfachen Container-to-Container Patterns (die Aktivität *C2C_toMatlabNode*) auf den jeweiligen Matlab-Rechner kopiert. Dieses Pattern wird wie in Abschnitt 6.2.1 beschrieben transformiert. Danach kann der Matlab-Bone Workflow gestartet werden (dies geschieht durch die Aktivität *StartMatlab-Bone*), der die kopierte Datei verändert. Anschließend folgt wiederum ein einfaches Container-to-Container Pattern (die Aktivität *C2C_fromMatlabNode*), das die geänderte CSV-Datei vom Matlab-Rechner zurückkopiert. Zuletzt wird mithilfe eines Data Format Conversion Patterns (*importCSVToTable*) die geänderte CSV-Datei in die Datenbank importiert und die einzelnen Variablenwerte werden aktualisiert. Auf die Transformation dieses Patterns wird in Abschnitt 6.3.3 eingegangen.

In Abschnitt 5.4.1.3 wird erläutert, dass das parallele Data Iteration Pattern drei Phasen umfasst: die *Split Stage*, die *Operation Stage* sowie die *Merge Stage*. So eine klare Aufteilung ist bei dem parallelen Data Iteration Pattern in der Pandas-Matlab Kopplung aus zwei Gründen nicht möglich. Bereits vor der parallen ForEach-Schleife werden mithilfe einer RetrieveData Aktivität Daten in den Workflow geladen, die in der Schleife bzw. in der *Split Stage* bei der Aufteilung des Simulationsraums benötigt werden. Des Weiteren muss die vom Nutzer modellierte eingebettete Operation aufgeteilt werden, da zuerst die Matlab-Instanz vorbereitet werden muss (z.B. muss das Arbeitsverzeichnis erstellt werden). Erst danach kann die CSV-Datei erstellt und auf den Matlab-Rechner kopiert werden, ehe der zweite Teil der eingebetteten Operation ausgeführt wird. Konzeptionell gesehen entsprechen die RetrieveData Aktivität vor der Schleife sowie die Erstellung und Bereitstellung der CSV-Datei in der Schleife einem Data Split Pattern (vgl. Abschnit 5.4.1.1). Des Weiteren entsprechen das zweite Container-to-Container Pattern und das Data Format Conversion Pattern konzeptionell gesehen einem Data Merge Pattern (vgl. Abschnit 5.4.1.1). Durch diese beiden Patterns bzw. das Data Merge Pattern wird in jedem Schleifendurchlauf eine Datei zurückkopiert und in die Datenbank importiert. In weiteren Arbeiten könnte evaluiert

werden, ob es beim parallelen Data Iteration Pattern neben der eingebetteten Operation noch Aktivitäten zur Vorbereitung des Data Splits bzw. zur Nachbereitung des Data Merge geben sollte bzw. ob dies sinnvoll ist. Dies wird im Rahmen dieser Diplomarbeit jedoch nicht betrachtet.

Im erläuterten Workflow-Fragment ist der Import der jeweiligen CSV-Datei in die Datenbank als Data Format Conversion Pattern modelliert. Die Überführung der referenzierten Daten in das neue Schema und die Erzeugung der CSV-Dateien werden hingegen direkt über Web Service Aufrufe (Invoke Aktivitäten) realisiert. Dies geschieht aus folgendem Grund: Diese beiden Vorgänge können *nicht* als Data Format Conversion Pattern modelliert werden, da den Web Services (die Data Format Conversion Patterns würden bei einer Transformation ja trotzdem auf diese Invoke Aktivitäten abgebildet werden) Parameter übergeben werden müssen, die mithilfe eines solchen Patterns nicht festgehalten werden können (z.B. die Simulations-Id sowie der Zeitschritt). Die Workflow-Fragmente, die diese Data Format Conversion Patterns ersetzen würden, müssten also auf Variablen bzw. Parameter zugreifen, die es innerhalb des Fragments nicht gibt und die daher auch nicht in einer entsprechenden Abbildungsregel genutzt werden könnten.

Des Weiteren stehen innerhalb des Workflow-Fragments für das parallele Data Iteration Pattern der Pandas-Matlab Kopplung die Web Services, die genutzt werden, schon fest. Sie werden nicht dynamisch im Resource Management gesucht. Dies geschieht aus folgendem Grund: Wenn ein Web Service aufgerufen wird, müssen im synchronen Fall der zugehörigen Invoke Aktivität die Parameter *inputVariable* und *outputVariable* spezifiziert werden. Dazu müssen im Prozess eine Input- sowie eine Output-Variable erzeugt werden. Der *message type* der Variablen ergibt sich aus der WSDL-Beschreibung des Web Service und umfasst je nach Web Service andere Elemente. Des Weiteren müssen solche Variablen initialisiert werden. Deshalb ist es nötig zu wissen, welche Parameter solch einer Variablen zugewiesen werden müssen und wie die Variable initialisiert wird. Da bei der Transformation eines Patterns nicht entschieden werden kann, welche Bedeutung ein Parameter eines *message types* hat und welche Variable diesem Parameter zugewiesen werden müsste, werden Web Services in die Workflow-Fragmente proprietär eingebunden. Dies führt allerdings dazu, dass das im Rahmen dieser Diplomarbeit für das parallele Data Iteration Pattern definierte Workflow-Fragment nicht universell einsetzbar ist, sondern auf diesen einen Anwendungsfall (die Pandas-Matlab Kopplung) zugeschnitten ist.

Ebenfalls muss, wenn ein Web Service genutzt wird, für den zugehörigen *partner link* ein *partner link type* in der WSDL des Prozesses erzeugt werden und die WSDL-Beschreibung des Web Service im Projektordner bereitgestellt werden. Deshalb werden die im Workflow-Fragment genutzten Web Services, obwohl diese schon feststehen, trotzdem im Resource Management registriert. Auf diese Weise kann die WSDL geladen werden. Jedem Web Service wird bei der Registrierung im Resource Management eine eindeutige Id zugewiesen. Im Rahmen dieser Arbeit wird davon ausgegangen, dass die *partner links* in den Fragmenten immer die in Listing 6.6 dargestellte Form aufweisen. Wenn der Platzhalter *?wsID* durch die Id ersetzt wird, ist der *partner link* eindeutig. Des Weiteren kann auf diese Weise auch einfach das in Listing 6.7 definierte Fragment in die WSDL des Prozesses eingefügt werden, um einen zugehörigen *partner link type* zu definieren. Auch hier wird das Fragment durch

die Ersetzung des Platzhalter *?wsID* wieder eindeutig. Auf die Umsetzung der Speicherung der Data Transformation Services im Resource Management wird in Abschnitt 7.3 genauer eingegangen.

Listing 6.6 Fragment für die Definition eines *partner links*

```
<bpel:partnerLinks>
  <bpel:partnerLink name="TransformationWS_ID?wsIDPartnerLink" partnerLinkType=
      "tns:TransformationWS_ID?wsIDPartnerLink"
      partnerRole="TransformationWS_ID?wsIDProvider"/>
</bpel:partnerLinks>
```

Listing 6.7 Fragment für die Definition eines *partner link types*

```
<plnk:partnerLinkType name="TransformationWS_ID?wsIDPartnerLink">
  <plnk:role name="TransformationWS_ID?wsIDProvider" portType="ws?wsID:?portType"/>
</plnk:partnerLinkType>
```

Nachdem die Transformation des parallelen Data Iteration Patterns der Pandas-Matlab Kopplung erläutert wurde, werden im Folgenden nun drei Transformationsregeln definiert. Die erste Transformationsregel bzw. das zugehörige Workflow-Fragment umfasst eine Konvertierung der durch den Parameter *data* referenzierten Daten in das neue Schema. Die zweite Transformationsregel bzw. das zugehörige Workflow-Fragment umfasst diese Konvertierung *nicht* und kann dann eingesetzt werden, wenn die durch den Parameter *data* referenzierten Daten schon konvertiert wurden. Die dritte Transformationsregel bzw. das zugehörige Workflow-Fragment kann in beiden Fällen verwendet werden.

Der Condition Part der ersten Transformationsregel lautet folgendermaßen:

- Condition 1: Der Parameter *data* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 2: Der Parameter *referenceList* spezifiziert eine *matlab node reference list*.
- Condition 3: Für den Parameter *mode* wurde vom Modellierer der Wert *gausspoints equally distributed* festgelegt.
- Condition 4: Der Paramter *simulationId* entspricht einer Variablen bzw. deren Namen. Diese Variable beinhaltet die eigentliche Simulations-Id.
- Condition 5: Der Parameter *timeStep* wurde angegeben. Beim parallelen Data Iteration Pattern ist dieser Parameter zwar optional, in diesem Fall ist er jedoch zwingend erforderlich.
- Condition 6: Die durch den Parameter *data* referenzierte Datenmenge bzw. der entsprechende Datencontainer weist das Datenformat *PandasGausspointTable* auf.

- Condition 7: Im Resource Management sind Metadaten über die im Workflow-Fragment genutzten Web Services (die Operationen *createNewGausspointInstanceTable* und *exportTableToCSVFile* des CSVToTableTransformer Web Service) gespeichert. Diese Metadaten werden benötigt, um z.B. die benötigte WSDL dem Projekt hinzuzufügen.
- Condition 8: Als eingebettete Operation wurden die in Abbildung 6.5 dargestellten Aktivitäten modelliert.

Wenn diese Bedingungen erfüllt sind, kann der folgende Action Part ausgeführt werden:

- Action 1: Lade das Workflow-Fragment *parallelDIP_pandas_gausspoint_table.xml* (dieses ist auf der Abgabe-CD zu finden).
- Action 2: Ersetze alle Vorkommen des Platzhalters *?randomNumber* durch ein Zufallszahl.
- Action 3: Ersetze den Platzhalter *?data* durch den Wert des Parameters *data*. Der Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 4: Ersetze alle Vorkommen des Platzhalters *?dataDataSource* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *data* spezifiziert wird. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 5: Ersetze alle Vorkommen des Platzhalters *?list* durch den Wert des Parameters *referenceList*.
- Action 6: Ersetze alle Vorkommen des Platzhalters *?simID* durch den Wert des Parameters *simulationId*.
- Action 7: Ersetze alle Vorkommen des Platzhalters *?keys* durch die zu berücksichtigenden Spaltennamen. Diese können wie oben beschrieben im Resource Management anhand eines Datenformates (in diesem Fall das Datenformat *GausspointInstanceTable*, da die durch den Parameter *data* referenzierten Daten wie bereits erläutert in dieses Datenformat konvertiert werden) sowie eines Terms (der Term wird durch den Parameter *mode* bestimmt und lautet in diesem Fall entweder *gausspoints* oder *elements*) abgefragt werden.
- Action 8: Ersetze alle Vorkommen des Parameters *?counterName* durch den Wert des Parameters *counterName*, wenn dieser spezifiziert wurde. Ansonsten ersetze alle Vorkommen durch den Wert *Counter* (dies ist nötig, damit der Laufvariablen der ForEach-Schleife ein Name zugewiesen wird - in WS-BPEL ist dieser standardmäßig *Counter*).
- Action 9: Ersetze den Platzhalter *?wsAddressSplit* durch die Adresse des Web Service, der die CSV-Dateien erstellt.
- Action 10: Ersetze alle Vorkommen des Platzhalters *?wsID2* durch die Id des Web Service, der die benötigten CSV-Dateien erzeugt.

- Action 11: Ersetze die Platzhalter *?start* und *?stop* folgendermaßen: Wenn bei der Erzeugung der CSV-Dateien die Eingabeparameter gauss_min und gauss_max berücksichtigt werden sollen, ersetze die Platzhalter durch einen leeren String. Ansonsten ersetze auch den XML-Code zwischen diesen Platzhaltern durch einen leeren String.
- Action 12: Ersetze den Platzhalter *?preprocessingOperation* durch die in Abbildung 6.5 dargestellten Aktivitäten, die eine Matlab-Instanz vorbereiten.
- Action 13: Ersetze den Platzhalter *?operation* durch die in Abbildung 6.5 dargestellten Aktivitäten, die den Matlab-Bone Workflow starten.
- Action 14: Ersetze den Platzhalter *?data_ts* durch den Wert des Parameters *data*. Der Variablenname muss von eckigen Klammern umschlossen sein.
- Action 15: Ersetze den Platzhalter *?timestep* durch den Wert des Parameters *timeStep*.
- Action 16: Ersetze den Platzhalter *?wsAddressConvert* durch die Adresse des Web Service, der die durch den Parameter *data* referenzierte Datenmenge in das neue Schema überführt.
- Action 17: Ersetze alle Vorkommen des Platzhalters *?wsID* durch die Id des Web Service, der für die Konvertierung in das neue Relationenschema zuständig ist.

Sollte der Modellierer mithilfe des Parameters *data* eine Tabelle referenzieren, die schon konvertiert wurde (Datenformat *GausspointInstanceTable*), so muss die referenzierte Tabelle nicht mehr in das neue Relationenschema überführt werden. Das Workflow-Fragment *parallelDIP_gausspoint_instance_table.xml* (dieses ist auf der Abgabe-CD zu finden) berücksichtigt dies, indem die Operation *createNewGausspointInstanceTable* nicht aufgerufen wird. Der Condition Part der zugehörigen Transformationsregel muss folgendermaßen angepasst werden:

- Condition 1: Überprüfe die Conditions 1 - 5 sowie 8 der obigen Transformationsregel.
- Condition 2: Die durch den Parameter *data* referenzierte Datenmenge weist das Datenformat *GausspointInstanceTable* auf.
- Condition 3: Im Resource Management sind Metadaten über den im Workflow-Fragment genutzten Web Service (die Operation *exportTableToCSVFile* des CSVTo-TableTransformer Web Service) gespeichert.

Auch der Action Part muss angepasst werden:

- Action 1: Lade das Workflow-Fragment *parallelDIP_gausspoint_instance_table.xml* (dieses ist auf der Abgabe-CD zu finden).
- Action 2: Führe die Actions 2 - 13 der obigen Transformationsregel aus.
- Action 3: Ersetze den Platzhalter *?dataImport* durch den Wert des Parameters *data*. Der Variablenname muss von eckigen Klammern umschlossen sein.

Im Workflow-Fragment *parallelDIP_choice.xml* sind Bereiche, die die Konvertierung der Tabelle (Datenformat *PandasGausspointTable*) betreffen, von Platzhaltern umschlossen. Die zugehörige Transformationsregel kann angewendet werden, wenn das Datenformat *PandasGausspointTable* **oder** *GausspointInstanceTable* lautet. Bei der Anwendung des Action Parts wird dann mittels If-Bedingungen geprüft, ob eine Konvertierung durchgeführt werden muss, oder eben nicht. Dementsprechend werden die einzelnen Platzhalter ersetzt. Daraus ergibt sich der folgende Condition Part:

- Condition 1: Überprüfe die Conditions 1 - 5 sowie 8 der obigen Transformationsregel.
- Condition 2: Die durch den Parameter *data* referenzierte Datenmenge weist das Datenformat *PandasGausspointTable* **oder** *GausspointInstanceTable* auf.
- Condition 3: Im Resource Management sind Metadaten über die im Workflow-Fragment genutzten Web Services gespeichert.

Auch der Action Part muss wieder geringfügig geändert werden:

- Action 1: Lade das Workflow-Fragment *parallelDIP_choice.xml* (dieses Workflow-Fragment ist auf der Abgabe-CD zu finden).
- Action 2: Führe die Actions 2 - 13 der obigen Transformationsregel aus.
- Action 3: Lautet das Datenformat *PandasGausspointTable*, dann führe auch die Actions 14 - 17 aus.
- Action 4: Ersetze die Platzhalter *?startChoiceX* und *?stopChoiceX* folgendermaßen: Lautet das Datenformat *PandasGausspointTable*, dann ersetze die Platzhalter durch einen leeren String. Ansonsten ersetzte auch den XML-Code zwischen diesen Platzhaltern durch einen leeren String.

Generell sollte die Kontrollstrategie für das parallele Data Iteration Pattern entweder die beiden obigen Regeln (erste Variante) oder die untere Regel (zweite Variante) umfassen. Im Rahmen dieser Diplomarbeit werden beide Varianten berücksichtigt, da jede Variante Vor- **und** Nachteile aufweist. Bei der ersten Variante wird, wenn der Condition Part erfüllt ist, im Action Part nur *eine* Bedingung geprüft. Diese Bedingung ergibt sich durch den Parameter *mode*. Im betrachteten Anwendungsfall kann dieser *elements equally distributed* oder *gausspoints equally distributed* lauten. Bei beiden Varianten wird über eine Bedingung geprüft, ob den Eingabeparametern gauss_min und gauss_max des Web Service für den Export ein Wert zugewiesen werden muss. Um dies zu verhindern müsste ansonsten bei beiden Varianten für alle möglichen Modi eigene Transformationsregeln mit einer entsprechenden Bedingung im Condition Part definiert werden. Ansonsten werden jedoch bei dieser Variante *keine* weiteren Bedingungen im Action Part geprüft und es werden lediglich die Platzhalter im geladenen Workflow-Fragment ersetzt. Jedoch müssen, wenn die durch den Parameter *data* referenzierte Datenmenge das Datenformat *GausspointInstanceTable* aufweist, zwei Transformationsregels auf Anwendbarkeit geprüft werden. Die erste Transformationsregel ist nicht anwendbar, da dort die referenzierte Datenmenge das Datenformat *PandasGausspointTable* aufweisen muss. Bei dieser Variante müssen also mehr Regeln auf Anwendbarkeit geprüft werden und

unter Umständen werden Bedingungen doppelt evaluiert, jedoch ist der Action Part weniger komplex.

Die zweite Variante umgeht dieses Problem. Die zugehörige Transformationsregel ist anwendbar, wenn das Datenformat der referenzierten Datenmenge *PandasGauspointTable* oder *GausspointInstanceTable* lautet. Erst im Condition Part wird über *weitere* Bedingungen geprüft, weche Teile des Workflow-Fragments tatsächlich ausgeführt werden müssen. Demzufolge kann bei dieser Variante die Transformationsregel in mehr Fällen angewendet werden, jedoch ist der Action Part deutlich komplexer. In weiteren Arbeiten könnte evaluiert werden, welche Variante im Allgemeinen besser ist (so könnte z.B. geprüft werden, welche Variante im Allgemeinen ein Pattern schneller auf ein ausführbares Workflow-Fragment transformiert).

6.3.2. Das Get Time Step Pattern

In Abschnitt 6.3.1 wird erläutert, dass das parallele Data Iteration Pattern der Pandas-Matlab Kopplung auf ein Workflow-Fragment abgebildet wird, das ein weiteres Pattern, das sogennante Get Time Step Pattern, beinhaltet. Dieses Pattern liefert für einen Ausdruck, wie z.B. *first* oder *last*, eine konkrete Zahl als Ergebnis. Diese wird in einer Variablen bereitgestellt. Das Pattern erwartet die folgenden Eingabeparameter:

- **timeStep**: Dieser Parameter spezifiziert den gewünschten Zeitschritt. Dabei kann es sich um eine Variable, die eine konkrete Zahl beinhaltet, oder aber um einen Literalwert (wie z.B. *first* oder *last* oder eine konkrete Zahl) handeln.
- **targetVariable**: Das Ergebnis, also die konkrete Zahl, die den Zeitschritt repräsentiert, wird in der durch den Parameter *targetVariable* spezifizierten Variablen gespeichert. Das Ergebnis wird immer, auch wenn der Modellierer eine Variable, die eine konkrete Zahl beinhaltet, angegeben hat, in dieser Variablen gespeichert. Dadurch kann der Modellierer diese Variable in folgenden Aktivitäten verwenden und hat des Weiteren die Möglichkeit später den Parameter *timeStep* zu verändern, ohne den restlichen Workflow anzupassen.
- **data**: Dieser Parameter entspricht einer *data container reference variable* bzw. deren Namen. Diese referenziert die Daten (z.B. die Tabelle *gausspunkte* in der Pandas Datenbank), aus denen der Time Step bzw. die konkrete Zahl extrahiert werden soll. Dieser Parameter ist optional. Er kann in der Modellierungsumgebung nur dann spezifiziert werden, wenn beim Parameter *timeStep* ein Ausdruck wie z.B. *first* oder *last* angegeben wird.
- **simulationId**: Bei der Pandas-Matlab Kopplung werden die Daten über die Gitterelemente bzw. Gausspunkte in der Tabelle *gausspunkte* gespeichert. Ziel ist es, z.B. den letzten Time Step in Bezug auf eine Simulations-Instanz zu bestimmen (in der Tabelle wäre dies die größte Zahl der Spalte *stepnr*). Um nur eine Simulations-Instanz zu berücksichtigen, muss auch die Simulations-Id festgelegt werden. Dieser Parameter ist ebenso wie der Parameter *data* optional. Er kann in der Modellierungsumgebung

nur dann spezifiziert werden, wenn beim Parameter *timeStep* ein Ausdruck wie z.B. *first* oder *last* angegeben wird.

Für das Get Time Step Pattern werden im Rahmen dieser Diplomarbeit drei Workflow-Fragmente sowie zugehörige Transformationsregeln definiert.

1. Fall - der Modellierer hat für den Parameter *timeStep* eine Variable primitiven Typs angegeben, die wiederum eine konkrete Zahl beinhaltet: In diesem Fall wird der Wert der referenzierten Variablen einfach der Zielvariablen, die durch den Paramter *targetVariable* spezifiziert wird, mithilfe einer Assign Aktivität zugewiesen. Die zugehörige Transformationsregel lautet folgendermaßen:

- Condition 1: Der Parameter *timeStep* entspricht einer Variablen primitiven Typs bzw. deren Namen.
- Condition 2: Der Parameter *targetVariable* entspricht einer Variablen primitiven Typs bzw. deren Namen.
- Action 1: Lade das Workflow-Fragment *getTimeStep_variable.xml* (vgl. Anhang B.6).
- Action 2: Ersetze den Platzhalter *?sourceVariable* durch den Wert des Paramters *timeStep*.
- Action 3: Ersetze den Platzhalter *?targetVariable* durch den Wert des Parameters *targetVariable*.

2. Fall - der Modellierer hat ein Literal angegeben und dieses Literal entspricht einer konkreten Zahl: Mithilfe einer Assign Aktivität (fixed value to variable) wird der Zielvariablen die konkrete Zahl zugewiesen. Die Transformationsregeln sieht dann folgendermaßen aus:

- Condition 1: Der Parameter *timeStep* entspricht einem Literal und dieses beinhaltet eine konkrete Zahl.
- Condition 2: Der Parameter *targetVariable* entspricht einer Variablen primitiven Typs bzw. deren Namen.
- Action 1: Lade das Workflow-Fragment *getTimeStep_literal.xml* (vgl. Anhang B.7).
- Action 2: Ersetze den Platzhalter *?literal* durch den Wert des Parameters *timeStep*.
- Action 3: Ersetze den Platzhalter *?targetVariable* durch den Wert des Parameters *targetVariable*.

3. Fall - der Modellierer hat ein Literal angegeben und dieses Literal entspricht einem Ausdruck wie z.B. *first* oder *last*: Dies ist der interessanteste Fall. Um die konkrete Zahl, die den Time Step repräsentiert, zu bestimmen, wird eine RetrieveData Aktivität ausgeführt. Soll z.B. der letzte Zeitschritt, der durch die Simulationsanwendung berechnet wurde, ermittelt werden, wird folgendes Statement ausgeführt: **SELECT MAX(***?key***) FROM** *?data*. Der Wert des Parameters *data* ersetzt den Platzhalter *?data*. Der Platzhalter *?keys* wird auch in diesem Fall mithilfe der im Resource Management gespeicherten Metadaten über Datenformate ersetzt (vgl. Abschnitt 7.3). So wird z.B. nach dem Datenformat *PandasGausspointTable* und dem Term *TimeStep* gesucht und anschließend der Platzhalter durch den Wert *stepnr* ersetzt.

Wenn durch den Modellierer zusätzlich der Parameter *simulationId* spezifiziert wurde, wird das Statement erweitert: **SELECT MAX(***?key***) FROM** *?data* **WHERE** *?key_sim* **=** **#***?simulationId***#**. Der Platzhalter *?simulationId* wird durch den Wert des Parameters *simulationId* ersetzt. Der Platzhalter *?key_sim* wird wiederum mithilfe der Metadaten über Datenformate aus dem Resource Management bestimmt (im betrachteten Anwendungsfall wird nach dem Term *SimulationId* gesucht - das Ergebnis ist die Spalte *sid*). Die zugehörige Transformationsregel lautet folgendermaßen:

- Condition 1: Der Parameter *timeStep* entspricht einem Literal und dieses lautet *first* oder *last*.
- Condition 2: Der Parameter *targetVariable* entspricht einer Variablen primitiven Typs bzw. deren Namen.
- Condition 3: Der Parameter *data* entspricht einer *data container reference variable* bzw. deren Namen. Diese Referenz verweist auf eine Tabelle innerhalb einer relationalen Datenbank. Dies ist nötig, da die Transformationregel eine SQL-Anfrage generiert, um den konkreten Wert für den Zeitschritt zu bestimmen.
- Condition 4: Der Parameter *simulationId* entspricht einer Variablen primitiven Typs bzw. deren Namen.
- Action 1: Lade das Workflow-Fragment *getTimeStep_first_last.xml* (vgl. Anhang B.8).
- Action 2: Ersetze den Platzhalter *?randomNumber* durch eine Zufallszahl.
- Action 3: Ersetze den Platzhalter *?targetVariable* durch den Wert des Parameters *targetVariable*.
- Action 4: Ersetze den Platzhalter *?dataSource* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *data* spezifiziert wird. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 5: Ersetze den Platzhalter *?dmCommand* durch ein geeignetes Statement. Dieses wird innerhalb der Transformationsregel generiert und hat die oben beschriebene Form.

6.3.3. Das Data Format Conversion Pattern

Das parallele Data Iteration Pattern der Pandas-Matlab Kopplung wird auf ein Workflow-Fragment abgebildet, das ein Data Format Conversion Pattern beinhaltet (vgl. Abschnitt 6.3.1). Durch dieses Pattern wird eine CSV-Datei (Datenformat *CSV*) in die Pandas Datenbank (Datenformat *GausspointInstanceTable*) importiert. Dies geschieht indem die Operation *importCSVFileToTable* des Web Service *TableToCSVTransformer* aufgerufen wird. Auch in diesem Workflow-Fragment bzw. bei der zugehörigen Transformationsregel steht der Web Service, der genutzt wird, schon fest. In Abschnitt 6.3.1 wird bereits erläutert, warum Web Services nicht dynamisch im Resource Management gesucht werden.

Dieser Importvorgang wurde als eingebettetes Data Format Conversion Pattern modelliert, da alle benötigten Informationen aus den Eingabeparametern bestimmt werden können und so auch in anderen Anwendungsfällen eine CSV-Datei in eine Datenbank (vorausgesetzt, die Tabelle implementiert ein geeignetes Relationenschema) importiert werden kann. Der folgende Condition Part muss erfüllt sein, damit die Transformationsregel angewendet werden kann:

- Condition 1: Der Parameter *sourceContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 2: Der Parameter *targetContainer* entspricht einer *data container reference variable* bzw. deren Namen.
- Condition 3: Das Datenformat der ersten *data container reference variable* lautet *CSV*.
- Condition 4: Das Datenformat der zweiten *data container reference variable* lautet *GauspointInstanceTable*.
- Condition 5: In dem Verzeichnis, in dem die CSV-Datei liegt, liegen noch zwei weitere Dateien: die Datei, die die Primärschlüssel beschreibt, sowie die Datei, die die Variablennamen beinhaltet. Des Weiteren müssen diese Primärschlüssel sowie Variablennamen Bestandteil der referenzierten Tabelle sein (vgl. Abschnitt 6.3.1). Diese Bedingung ist nötig, um eventuellen Inkompatibilitäten vorzubeugen.
- Condition 6: Im Resource Management sind Metadaten über den im Workflow-Fragment genutzten Web Service (die Operation importCSVFileToTable) gespeichert.

Wenn diese Bedingungen erfüllt sind, kann der folgende Action Part ausgeführt werden:

- Action 1: Lade das Workflow-Fragment *dataFormatConversion_importWS.xml* (vgl. Anhang B.9).
- Action 2: Ersetze den Platzhalter *?randomNumber* durch eine Zufallszahl.
- Action 3: Ersetze den Platzhalter *?dataResource* durch den Wert des Elementes *dataSourceReferenceVariable* der *data container reference variable*, die durch den Parameter *targetContainer* spezifiziert wird. Dieser Variablenname muss von eckigen Klammern umschlossen sein.
- Action 4: Ersetze den Platzhalter *?container* durch Wert des Parameters *sourceContainer*. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 5: Ersetze den Platzhalter *?table* durch den Wert des Parameters *targetContainer*. Dieser Variablenname darf nicht von eckigen Klammern umschlossen sein.
- Action 6: Ersetze den Platzhalter *?wsAddressImport* durch die Adresse des Web Service, der die CSV-Datei in die Datenbank importiert.
- Action 7: Ersetze alle Vorkommen des Platzhalters *?wsID* durch die Id des Web Service.

6.4. Kontrollstrategien

In Abschnitt 3.7 wird erläutert, dass die Kontrollstrategie eines Datenmanagementpatterns die Reihenfolge bestimmt, in der die Transformationsregeln, die dieser Kontrollstrategie angehören, auf Anwendbarkeit geprüft werden. Im Folgenden werden nun die einzelnen Kontrollstrategien definiert. Da es bei den bisher definierten Transformationsregeln keine offensichtlichen Abhängigkeiten zwischen den Regeln gibt, die eine bestimmte Reihenfolge notwendig machen würden, werden die Regeln weitestgehend zufällig angeordnet. Es wird lediglich versucht, dass Transformationsregeln, die weniger Bedingungen prüfen, zuerst evaluiert werden. Einzig bei der Kontrollstrategie für das Get Time Step Pattern wird bei der Anordnung der Transformationsregeln eine Eigenschaft berücksichtigt.

- **Kontrollstrategie für das Container-to-Container Pattern:** Zuerst wird die Regel für das einfache Container-to-Container Pattern geprüft (vgl. Abschnitt 6.2.1). Erst danach folgt die Regel für das komplexe Container-to-Container Pattern (vgl. Abschnitt 6.2.2).
- **Kontrollstrategie für das Data Format Conversion Pattern:** Zuerst wird die Regel, die ein Data Transformation Script startet, geprüft (vgl. Abschnitt 6.2.3). Sollte diese Regel nicht anwendbar sein, wird die in Abschnitt 6.3.3 für die Datenformate *CSV* und *GausspointInstanceTable* definierte Regel geprüft.
- **Kontrollstrategie für das sequentielle Data Iteration Pattern:** Für dieses Pattern wurde bisher nur die in Abschnitt 6.2.4 erläuterte Regel definiert.
- **Kontrollstrategie für das Multiple Data Transfer Pattern:** Auch für dieses Pattern wurde bisher nur eine Regel definiert (vgl. Abschnitt 6.2.5).
- **Kontrollstrategie für das parallele Data Iteration Pattern:** Zuerst wird die Regel geprüft, bei der die durch den Parameter *data* spezifizierte Datenmenge das Datenformat *PandasGausspointTable* aufweisen muss. Erst danach wird die Regel geprüft, bei der keine Konvertierung notwendig ist (Datenformat *GausspointInstanceTable*). Alternativ könnte auch nur die Regel überprüft werden, die über Bedingungen im Action Part prüft, ob eine Konvertierung notwendig ist (vgl. Abschnitt 6.3.1).
- **Kontrollstrategie für das Get Time Step Pattern:** Zuerst wird die Regel für Ausdrücke, wie z.B. *first* oder *last*, auf Anwendbarkeit geprüft (vgl. Abschnitt 6.3.2). Als nächstes wird die Regel für ein Literal, das eine konkrete Zahl beinhaltet, evaluiert. Zuletzt wird die Regel überprüft, die dann anwendbar ist, wenn der Paramter *timeStep* einer Variablen primitiven Typs entspricht. Dies geschieht aus folgendem Grund: der Patternansatz soll dem Workflow-Modellierer die Arbeit erleichtern und eine gewisse Abstraktion schaffen. Ausdrücke, wie z.B. *first* oder *last*, sind im Allgemeinen abstrakter als eine konkrete Zahl und werden deshalb durch den Modellierer vielleicht häufiger angegeben und die zugehörige Transformationsregel kann demzufolge öfters als die anderen Beiden angewendet werden.

7. Umsetzung

Nachdem in Kapitel 6 konzeptionelle Änderungen und Erweiterungen, die für multi-skalare Simulationsworkflows wie die gesamte Pandas-Matlab Kopplung (vgl. Abschnitt 4.3) notwendig sind, erläutert werden, wird in diesem Kapitel auf deren konkrete Umsetzung in den Prototyp des SIMPL-Rahmenwerks eingegangen. In Abschnitt 7.1 werden die erweiterten Zugriffsmechanismen für Dateisysteme erläutert. Erweiterungen der GUI werden in Abschnitt 7.2 und Erweiterungen am Resource Management in Abschnitt 7.3 betrachtet. Zuletzt wird in Abschnitt 7.4 auf die Implementierung der Pattern Transformation eingegangen.

7.1. Zugriffsmechanismen für Dateisysteme

Für Unix-basierte Dateisysteme wurde ein neuer Konnektor, der **UnixLocalFSConnector**, implementiert. Dieser implementiert alle generischen Operationen des SIMPL Cores (vgl. Abschnitt 5.3) für Unix-Dateisysteme. Einzig, die im Rahmen dieser Diplomarbeit vorgestellte TransferData Operation wird nicht unterstützt, da diese nur der SSHConnector implementiert (vgl. Abschnitt 6.1.2).

Listing 7.1 Auszüge aus dem UnixLocalFSConnector

```
Process process = null;
File directory = new File("/bin");

process = runtime.exec("/bin/bash", null, directory);
if (process != null) {
  BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
    process.getInputStream()));
  PrintWriter printWriter = new PrintWriter(new BufferedWriter(
    new OutputStreamWriter(process.getOutputStream())), true);
  // now execute the specified command
  printWriter.println(command);
  printWriter.println("exit");
  ...
  printWriter.close();
  process.destroy();
}
```

Der UnixLocalFSConnector beachtet, dass bei Ordner- bzw. Dateipfaden ein „/“als Trennzeichen verwendet wird. Sollen Befehle auf der Kommandozeile ausgeführt werden, wird eine *Bash* gestartet. Erst danach wird der eigentliche Befehl, der ausgeführt werden soll,

ausgeführt. Dieses Vorgehen ist notwendig, damit beliebige Befehle ausgeführt werden können. Listing 7.1 zeigt Auszüge der Implementierung. Der Befehl, der ausgeführt werden soll, ist in der Variablen *command* gespeichert.

In Abschnitt 6.1.2 wird erläutert, dass der SIMPL Core um die generische Operation **TransferData** erweitert wird. Die einzelnen Konnektoren **können** diese Operation dann implementieren. Bisher implementiert nur der **SSHConnector** diese Operation. Die TransferData Methode des SSHConnectors kopiert die referenzierte Datei mittels eines SSH-Befehls **direkt** auf den entfernten Rechner. Wird die generische TransferData Operation des SIMPL Cores durch die Execution Engine aufgerufen, findet eine Fallunterscheidung statt (vgl. Abschnitt 6.1.2). Diese ist in Listing 7.2 dargestellt.

Listing 7.2 Auszüge aus der Klasse *SIMPLCoreImpl*

```
if (dataSink.getAPIType().equals("SSH")) {
  // SSH server -> use TransferData operation
  // get connector instance
  connector = ConnectorProvider.getInstance(dataSink.getType(),
    dataSink.getSubType());
  ...
  success = connector.transferData(dataSource, dataSink, statement, target);
} else {
  // NO SSH server -> use RetrieveData and WriteDataBack operation
  // instead
  retrievedData = this.retrieveData(dataSource, statement);
  if (retrievedData != null) {
    success = this.writeDataBack(dataSink, retrievedData, target);
  } else {
    // no data retrieved
    success = false;
  }
}
```

7.2. Erweiterung der GUI

Im Rahmen dieser Diplomarbeit wurde auch die GUI der prototypischen Umsetzung des SIMPL-Rahmenwerks erweitert bzw. angepasst. Zum einen wurde die Parametrisierung der DM Aktivitäten überarbeitet, zum anderen wurden Datenmanagementpatterns in diese integriert.

7.2.1. DM Aktivitäten

Die Parametrisierung der DM Aktivitäten (vgl. Abschnitt 5.1) wurde aus zwei Gründen überarbeitet. Zum einen wurden in früheren Versionen Parameter im XML-Code gespeichert, die bei der Ausführung der Aktivität nicht benötigt werden bzw. nicht explizit angegeben

werden müssen, da diese anhand anderer Parameter und mithilfe des Resource Managements bestimmt werden können, zum anderen waren die Parameternamen teilweise nicht aussagekräftig. Im Rahmen dieser Arbeit wurde die Parametrisierung der Aktivitäten überarbeitet und auch die Beschreibung der Parameter in den *Property Sections* angepasst. Bei allen Aktivitäten werden der Typ, der Subtyp sowie die Abfragesprache nicht mehr im XML-Code gespeichert. Wenn die zugehörige *Property Section* erzeugt wird, werden wie bisher Metadaten über die referenzierte Datenquelle aus dem Resource Management geladen. Diese Metadaten beinhalten u.a. den Typ, den Subtyp sowie die Abfragesprache. Aus diesem Grund ist eine separate Speicherung im XML-Code nicht notwendig.

Die Tabellen 7.1 bis 7.5 zeigen die überarbeitete Parametrisierung. Zu beachten ist, dass die RetrieveData Aktivität, die QueryData Aktivität sowie die WriteDataBack Aktivität ebenfalls die in Tabelle 7.1 erläuterten Parameter benötigen. Aus Platzgründen werden diese aber in den jeweiligen Tabellen weggelassen und nur zusätzliche Parameter werden angegeben.

Parametername (alt)	**Parametername (neu)**	**Beschreibung in der Property Section**
dsStatement	dmCommand	Data management command:
dsIdentifier	dataResource	Data resource:
dsType	-	Type of data resource:
dsKind	-	Subtype of data resource:
dsLanguage	-	Command language of data resource:

Tabelle 7.1.: Parametrisierung der IssueCommand Aktivität

Parametername (alt)	**Parametername (neu)**	**Beschreibung in der Property Section**
dataVariable	targetVariable	Target variable:

Tabelle 7.2.: Parametrisierung der RetrieveData Aktivität

Parametername (alt)	**Parametername (neu)**	**Beschreibung in der Property Section**
queryTarget	targetContainer	Target container to insert the query result:

Tabelle 7.3.: Parametrisierung der QueryData Aktivität

Parametername (alt)	Parametername (neu)	Beschreibung in der Property Section
writeTarget	targetContainer	Target container to write the data:
dataVariable	dataVariable	Data variable:

Tabelle 7.4.: Parametrisierung der WriteDataBack Aktivität

Parametername (alt)	Parametername (neu)	Beschreibung in der Property Section
dsStatement	dmCommand	Data management command for data source:
dsIdentifier	dataSource	Data source:
dsType	-	Type of data source:
dsKind	-	Subtype of data source:
dsLanguage	-	Command language of data source:
targetDsIdentifier	dataSink	Data sink:
targetDsType	-	Type of data sink:
targetDsKind	-	Subtype of data sink:
targetDsLanguage	-	Command language of data sink:
targetDsContainer	dataSinkContainer	Target container to insert the data:

Tabelle 7.5.: Parametrisierung der TransferData Aktivität

Um die Parametrisierung der DM Aktivitäten zu ändern, wurde das Ecore[1] Model im Paket *org.eclipse.bpel.simpl.model* geändert. Dieses Modell bestimmt die Attribute, die eine DM Aktivität aufweist. Des Weiteren wurden der Serialisierer (die Klasse *DataManagementActivitySerializer* im Paket *org.eclipse.bpel.simpl.model*) sowie der Deserialisierer (die Klasse *DataManagementActivityDeserializer* im Paket *org.eclipse.bpel.simpl.model*) angepasst.

7.2.2. Datenmanagementpatterns

Um die in Kapitel 6 erläuterten Datenmanagementpatterns in das SIMPL Rahmenwerk bzw. dessen prototypische Umsetzung zu integrieren, musste das Ecore Model im Paket

[1] http://www.eclipse.org/modeling/emf/?project=emf

org.eclipse.bpel.simpl.model erweitert werden. Das erweiterte Modell definiert die folgenden Datenmanagementpatterns:

- Container-to-Container Pattern
- Data Format Conversion Pattern
- Sequential Data Iteration Pattern
- Multiple Data Transfer Pattern
- Parallel Data Iteration Pattern
- Get Time Step Pattern

Damit der Modellierer diese Patterns in einen Workflow einfügen kann, wurde die View *Palette* um Datenmanagementpatterns erweitert (vgl. Abbildung 7.1). Der Modellierer kann das gewünschte Pattern, genau wie eine DM Aktivität, auswählen und in den Workflow einfügen (*Drag & Drop*).

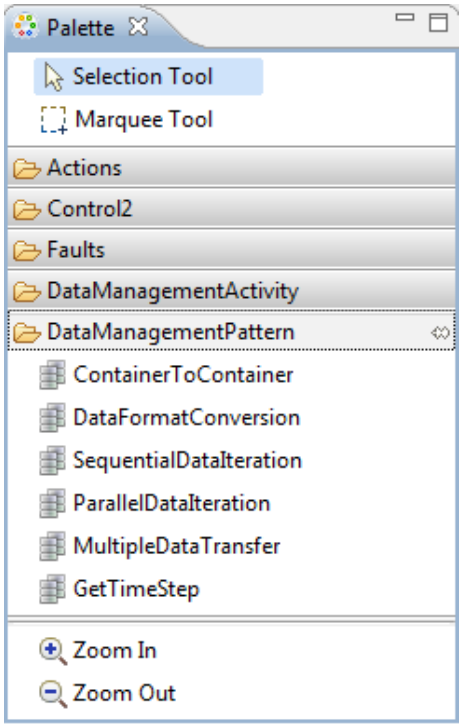

Abbildung 7.1.: Die View *Palette* wurde um Datenmanagementpatterns ergänzt

Des Weiteren wurden für die einzelnen Datenmanagementpatterns die zugehörigen *Property Sections* im Paket *org.eclipse.bpel.simpl.ui.properties* implementiert. Abbildung 7.2 zeigt beispielhaft die *Property Section* für das Container-to-Container Pattern.

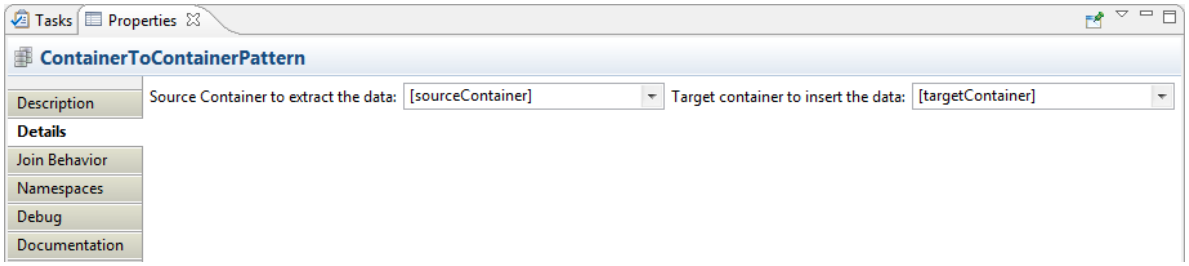

Abbildung 7.2.: Die *Property Section* für das Container-to-Container Pattern

Die Tabellen 7.6 bis 7.11 zeigen die Parametrisierung der Datenmanagementpatterns sowie die Beschreibung der Parameter in den *Property Sections*.

Parametername	**Beschreibung in der Property Section**
sourceContainer	Source Container to extract the data:
targetContainer	Target container to insert the data:

Tabelle 7.6.: Parametrisierung des Container-to-Container Patterns

Parametername	**Beschreibung in der Property Section**
sourceContainer	Source Container to extract the data:
targetContainer	Target container to insert the converted data:

Tabelle 7.7.: Parametrisierung des Data Format Conversion Patterns

Parametername	**Beschreibung in der Property Section**
data	The data to be iterated:
currentContainer	Name of the local data container reference variable:
activity	(Dieser kann in der *Property Section* nicht geändert werden.)
counterName	The name of the counter variable:

Tabelle 7.8.: Parametrisierung des sequentiellen Data Iteration Patterns

Parametername	**Beschreibung in der Property Section**
containerReferences	(Dieser Parameter wird in der *Property Section* nicht beschrieben.)
targetContainer	Target container to insert the data:

Tabelle 7.9.: Parametrisierung des Multiple Data Transfer Patterns

Parametername	**Beschreibung in der Property Section**
data	The data to be iterated:
referenceList	Nodes:
mode	Mode:
simulationId	Simulation Id:
timeStep	Time Step:
counterName	The name of the counter variable:
activity	(Dieser kann in der *Property Section* nicht geändert werden.)

Tabelle 7.10.: Parametrisierung des parallelen Data Iteration Patterns

Parametername	**Beschreibung in der Property Section**
timeStep	Time step:
targetVariable	Variable to insert the concrete time step value:
data	Data:
simulationId	Simulation Id:

Tabelle 7.11.: Parametrisierung des Get Time Step Patterns

Zu beachten ist, dass das Multiple Data Transfer Pattern (vgl. Tabelle 7.9) einen Parameter mit dem Namen *containerReferences* beinhaltet. In Abschnitt 6.2.5 wurde jedoch erläutert, dass bei der Modellierung eine *data container reference list* erzeugt wird, die als Element des Multiple Data Transfer Patterns gespeichert wird. Zum einen hat sich dieser Ansatz erst im Laufe der Implementierung der Patterns als beste Möglichkeit erwiesen, zum anderen wurde im Rahmen dieser Arbeit keine Möglichkeit gefunden das Ecore Model entsprechend anzupassen. Deshalb werden bei der umgesetzten Implementierung mithilfe des Parameters *containerReferences* die Namen der durch den Modellierer ausgewählten *data container reference variables* gespeichert und durch ein Leerzeichen getrennt. Erst bei der Transformation des Patterns wird die *data container reference list*, die den Platzhalter *?listContent* ersetzt, generiert (dazu wurde die in Abschnitt 6.2.5 vorgestellte Transformationsregel entsprechend angepasst).

In den Tabellen 7.8 und 7.10 ist ersichtlich, dass der Parameter *activity* nicht in der *Property Section* geändert werden kann. Das liegt daran, dass dieser Parameter der eingebetteten Operation entspricht, die das jeweilige Pattern ausführt. Der Workflow-Modellierer kann diese Operation im graphischen Editor ändern, indem er Aktivitäten per *Drag & Drop* hinzufügt oder Aktivitäten löscht. Der zugehörige Parameter bzw. das Kindelement des jeweiligen Patterns im XML-Code wird dann automatisch angepasst.

Damit der Modellierer die Transformation der Datenmanagementpatterns zur Modellierungszeit starten kann, wurden weitere Anpassungen an der GUI vorgenommen. Der Menüleisteneintrag *SIMPL* wurde um die Menüeinträge *Transform all patterns* und *Transform all patterns (no recursion)* erweitert (vgl. Abbildung 7.3). Klickt der Nutzer auf den Eintrag *Transform all patterns*, werden alle Datenmanagementpatterns im geöffenten Workflow ersetzt. Sollte ein Datenmanagementpattern auf ein Workflow-Fragment abgebildet werden, welches wiederum weitere Patterns enthält, so werden auch diese Datenmanagementpatterns rekursiv ersetzt. Klickt der Nutzer hingegen auf den Eintrag *Transform all pattern (no recursion)*, werden eingebettete Patterns **nicht** rekursiv ersetzt. Des Weiteren wurden der Toolbar Buttons hinzugefügt, die ebenfalls diese Möglichkeiten der Pattern Transformation bieten.

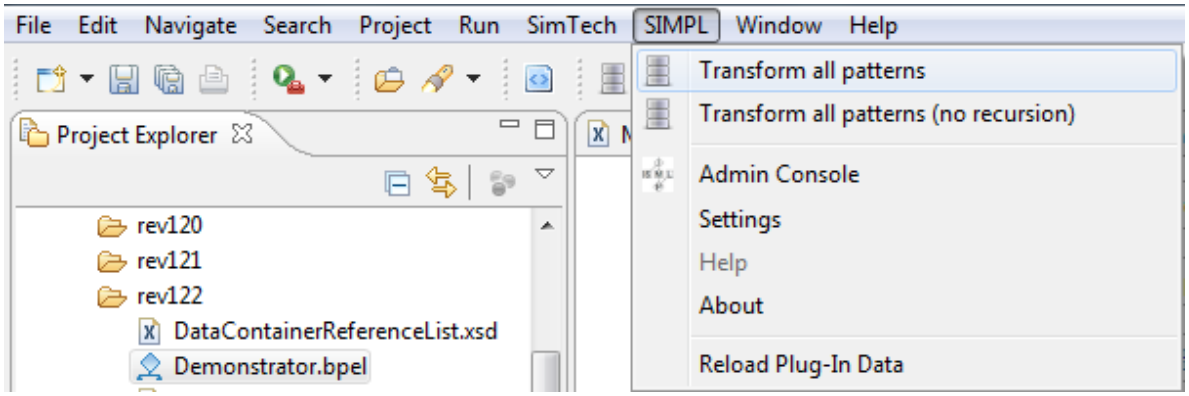

Abbildung 7.3.: Der Menüleisteneintrag *SIMPL*

Öffnet der Nutzer das Kontextmenü eines Datenmanagementpatterns durch einen Rechtsklick, beinhaltet das Menü vier Einträge, die die Pattern Transformation betreffen: *Transform all patterns*, *Transform all patterns (no recursion)*. *Transform **this** pattern* und *Transform **this** pattern (no recursion)*. Die erste beiden Möglichkeiten der Pattern Transformation wurden bereits erläutert. Klickt der Nutzer auf den Eintrag *Transform this pattern*, wird das ausgewählte Datenmanagementpattern transformiert. Auch hier werden wiederum rekursiv alle eingebetteten Datenmanagementpatterns ersetzt. Wird hingegen auf den Menüeintrag *Transform this pattern (no recursion)* geklickt, wird das ausgewählte Datenmanagementpattern transformiert. Eingebettete Patterns werden in diesem Fall nicht ersetzt. Auf die konkrete Umsetzung der Pattern Transformation wird in Abschnitt 7.4 eingegangen.

7.3. Erweiterungen am Resource Management

In Abschnitt 5.4.1.2 wird erläutert, dass ein Data Format Conversion Pattern eine Datenformatkonvertierung durchführt. Im Allgemeinen kann solch eine Konvertierung auf unterschiedliche Weise erfolgen: es können die Methoden des Service Bus genutzt, ein Script gestartet oder ein Web Service bzw. RESTful Service aufgerufen werden.

Das Data Format Conversion Pattern der Pandas Post-Processing Phase wird z.B. auf ein Workflow-Fragment abgebildet, das mithilfe einer IssueCommand Aktivität ein Script startet. Die Datenformatkonvertierung in der Pandas-Matlab Kopplung hingegen erfolgt über einen Web Service Aufruf.

Bisher konnten im Resource Management nur *Data Transformation Services* registriert werden, die Daten aus einem Workflow Datenformat in ein anderes Workflow Datenformat transformieren. Metadaten über diese Services wurden in der Tabelle **Datatransformationservices** gespeichert (vgl. Abschnitt 5.2). Ab sofort werden diese Data Transformation Services *Service Bus Data Transformation Services* genannt, da sie innerhalb des Service Bus deployed sind. Der Begriff *Data Transformation Service* wird als Oberbegriff für alle Services bzw. Programme verwendet, die Daten z.B. über ETL Operationen transformieren können. In dieser Arbeit wird lediglich auf die Umsetzung für die ETL Operation *Datenformatkonvertierung* eingegangen.

Im Rahmen dieser Diplomarbeit wurde das Resource Management erweitert. Nun können *Service Bus Data Transformation Services*, *Data Transformation Scripts* sowie *Data Transformation Web Services* registriert werden. Auf deren konkrete Speicherung in der PostgreSQL Datenbank wird im Folgenden eingegangen. Zu beachten ist, dass in dieser Implementierung *RESTful Services* nicht unterstützt werden.

Das in Abschnitt 6.2.3 erläuterte *Data Transformation Script* transformiert eine Datei, die im *TecPlot*-Format vorliegt, in das *VTK*-Format. Bei diesen Formaten handelt es sich nicht um Workflow Datenformate, sondern um sogenannte externe Datenformate. Bisher konnten solche externen Formate **nicht** im Resource Management registriert werden. Nun werden bei der Initialisierung des Resource Managements im Schema *Simpl_Definitions* die folgenden drei Tabellen für Datenformate erzeugt:

- **Data_Formats(**
  Id, Name, Kind**)**
- **Workflow_Dataformat_Types(**
  Id, XSD_Type**)**
- **External_Dataformats(**
  Id, Documentation**)**

Bei der Registrierung der Datenformate wird das *Partitionierungsmodell* berücksichtigt. D.h., dass die zu speichernden Metadaten auf mehrere Tabellen aufgeteilt werden. Eigenschaften, welche die Workflow Datenformate und die externen Datenformate aufweisen, werden in der Tabelle **Data_Formats** gespeichert. Zusätzliche Metadaten werden separat in eigenen Tabellen gespeichert. Wenn ein Workflow Datenformat oder ein externes Datenformat registriert

wird, wird ein neuer Eintrag in der Tabelle *Data_Formats* erzeugt. Mithilfe des Attributs *Id* wird dem Datenformat eine eindeutige Id zugewiesen. Der Name des Datenformats wird mithilfe des Attributs *Name* gespeichert. Das Attribut *Kind* bestimmt, ob es sich um ein Workflow Datenformat oder um ein externes Datenformat handelt. Bei einem Workflow Datenformat wird das zugehörige XML Schema in der Tabelle **Workflow_Dataformat_Types** gespeichert. Das Attribut *Id* entspricht der Id des Datenformats und verweist somit auf den entsprechenden Eintrag in der Tabelle **Data_Formats**. Bei einem externen Datenformat hingegen werden zusätzliche Information über das Datenformat mithilfe des Attributs *Documentation* in der Tabelle **External_Dataformats** gespeichert.

Für die Speicherung der *Data Transformation Services* werden bei der Initialisierung des Resource Managements im Schema *Simpl_Resources* die im Folgenden beschriebenen Tabellen angelegt. Auch hier wird wiederum das Partionierungsmodell angewendet:

- **Data_Transformation_Services(**
  Id, Name, Input_Dataformat, Output_Dataformat, Direction_Input_Output, Direction_Output_Input, Kind**)**
- **Service_Bus_Data_Transformation_Services(**
  Id, Implementation**)**
- **Data_Transformation_Scripts(**
  Id, Call, Relative_Path, Output_Container_Required**)**
- **Data_Transformation_Web_Services(**
  Id, Port_Type, Operation, Service, Port, Address, Synchronous, WSDL**)**

Eigenschaften, die die verschiedenen Typen von *Data Transformation Services* teilen, werden in der Tabelle **Data_Transformation_Services** gespeichert. Die einzelnen Attribute werden bereits in Abschnitt 5.2 erläutert. Zu beachten ist, dass auch hier jedem *Data Transformation Service* eine eindeutige Id zugewiesen wird. Bei einem *Service Bus Data Transformation Service* wird ein Verweis auf die konkrete Implementierung mithilfe des Attributs *Implementation* in der Tabelle **Service_Bus_Data_Transformation_Services** festgehalten. Wenn z.B. die WriteDataBack Operation des SIMPL Cores aufgerufen wird und die zu speichernden Daten ein anderes Workflow Datenformat aufweisen als der zugehörige Konverter erwartet, wird nach einem geeigneten *Service Bus Data Transformation Service* gesucht, der die Daten in ein Workflow-Datenformat, welches der zugehörige Konverter verarbeiten kann, transformiert und die Implementierung entsprechend aufgerufen. Bei einem *Data Transformation Script* speichert das Attribut *Call* in der Tabelle **Data_Transformation_Scripts** den Script-Aufruf. Das Ausgabeverzeichnis, in dem konvertierte Objekte gespeichert werden, wird mithilfe des Attributs *Relative_Path* festgehalten. Das Attribut *Output_Container_Required* spezifiziert, ob beim Script-Aufruf auch noch eine Referenz für die Ausgabe angegeben werden muss. In der Tabelle **Data_Transformation_Web_Services** werden zusätzliche Metadaten über *Data Transformation Web Services* gespeichert (es gibt die Attribute *Port_Type*, *Operation*, *Service*, *Port*, *Address* und *Snychronous*). Die zugehörige WSDL-Beschreibung des Web Service wird durch das Attribut *WSDL* festgehalten. Die Werte der anderen Attribute könnten theoretisch auch aus der WSDL extrahiert werden. Die separate Speicherung der Informationen hat jedoch einen entscheidenen Vorteil. Bei der Transformation eines Datenmanagementpatterns

muss nicht jedes mal die WSDL eingelesen und geparst werden, wenn einzelne Platzhalter durch Informationen aus dieser WSDL ersetzt werden müssen.

Zur Verwaltung der Objekte wurde der Resource Management Web Service um CRUD (Create, Read, Update und Delete) Operationen für die einzelnen Objekte erweitert und das Resource Management Web Interface entsprechend angepasst. Ebenso kann nun im Resource Management ein geeigneter *Data Transformation Service* anhand eines Eingabe- und Ausgabeformats gesucht werden.

Ebenso können im Resource Management nun zusätzliche Metadaten über einzelne Datenformate hinterlegt werden. Dazu wird bei der Initialisierung des Resource Managements die Tabelle **Data_Format_Metadata** im Schema *Simpl_Definitions* erzeugt:

- **Data_Format_Metadata(**
  Id, Data_Format, Term, Keys**)**

Durch das Attribut *Data_Format* wird ein Datenformat (z.B. das externe Datenformat *Gausspoint_Instance_Table*) aus der Tabelle **Data_Formats** referenziert. Das Attribut *Term* speichert einen Term (z.B. *Gausspoints*). Mithilfe des Attributs *Keys* können Spaltennamen in einer XML-Struktur hinterlegt werden (z.B. *<tableMetaData> <column>sid</column> <column>elementnr</column> <column>gaussnr</column> </tableMetaData>*). Auch hier wurde der Resource Management Web Service um CRUD Operationen erweitert. Des Weiteren bietet der Web Service die Möglichkeit, ein *DataFormatMetadata* Objekt anhand eines Datenformats und eines Terms zu suchen. Auch das Resource Management Web Interface wurde entsprechend angepasst.

In Abschnitt 6.3.1 wird erläutert, dass das parallele Data Iteration Pattern auf ein Workflow-Fragment abgebildet wird, in dem der Platzhalter *?keys* ersetzt werden muss. Mithilfe der im Resource Management gespeicherten Metadaten über Tabellen kann nun dieser Platzhalter ersetzt werden (das Attribut *Keys* bestimmt die zu berücksichtigenden Spalten). Des Weiteren kann beim Get Time Step Pattern mithilfe dieser Metadaten der konkrete Zeitschritt für einen Ausdruck, wie z.B. *first* oder *last*, ermittelt werden (vgl. Abschnitt 6.3.2).

Des Weiteren wurde das Resource Management noch an einer anderen Stelle erweitert. Bei dessen Initialisierung wird im Schema *Simpl_Definitions* nun die Tabelle **XML_Schemas** erzeugt:

- **XML_Schemas(**
  Name, XSD_Type**)**

In dieser Tabelle können XML Schemas gespeichert werden. Der Name eines Schemas wird mithilfe des Attributs *Name* festgehalten und das Attribut *XSD_Type* umfasst das eigentliche Schema. Dies geschieht aus folgendem Grund: beim Multiple Data Transfer Pattern z.B. wird eine *data container reference list* erzeugt. Wenn der Nutzer nun ein Multiple Data Transfer Pattern modelliert, kann automatisch das zugehörige XML Schema, das diese Liste definiert, aus dem Resource Managment geladen, im Projekordner bereitgestellt und importiert werden. Die Zuordnung zwischen Datenmanagementpattern und benötigtem XML Schema findet im Serialisierer (die Klasse *DataManagementPatternSerializer* im Paket *org.eclipse.bpel.simpl.model*) statt. Der Serialisierer wird u.a. immer dann aufgerufen, wenn ein Datenmanagementpattern

dem Workflow hinzugefügt wird. Beim Multiple Data Transfer Pattern wird nun noch das Schema mit dem Namen *DataContainerReferenceList* aus dem Resource Management geladen und importiert. Auch hier wurde das Resource Management Web Interface entsprechend angepasst.

7.4. Transformation der Datenmanagementpatterns

In diesem Abschnitt wird die implementierte Pattern Transformation betrachtet. Möchte der Modellierer *alle* Datenmanagementpatterns eines im erweiterten Eclipse BPEL-Designer geöffneten Workflows transformieren, muss dieser auf einen Menüeintrag mit dem Namen *Transform all patterns* oder *Transform all patterns (no recursion)* klicken (vgl. Abschnitt 7.2.2). Die zugehörige *Action* bzw. der *Handler* ruft die Methode *transformAllPatterns* der Klasse *PatternTransformationImplementation* im Paket *org.eclipse.bpel.simpl.ui.pattern.transformation* auf. Diese Methode prüft, ob es sich bei dem geöffneten Dokument um ein BPEL-Dokument handelt und fragt den Modellierer, ob eine neue Revision des Projekts angelegt werden soll. Dies geschieht, da Änderungen der Pattern Transformation **nicht** rückgängig gemacht werden können. Anschließend ruft diese Methode die Methode *traverse* der Klasse *WorkflowGraphTraverser* im Paket *org.eclipse.bpel.simpl.ui.pattern.transformation.traverser* auf. Die *traverse* Methode verarbeitet den Workflow bzw. das eigentliche XML-Dokument mithilfe der JDOM API[1]. Das Dokument wird eingelesen und als Baum im Hauptspeicher bereitgestellt. Mithilfe der Methode *visit* werden die Elemente dieser Strukur bzw. des Baums besucht. Dazu wird dieser Methode das Wurzelelement übergeben. Es wird geprüft, ob es sich bei dem aktuell betrachteten Element um ein Datenmanagementpattern (*isDataManagementPattern*) handelt. Ist dies *nicht* der Fall, wird für jedes Kindelement erneut die Methode *visit* aufgerufen. Dieser wird das jeweilige Kindelement als Eingabeparameter übergeben. Auf diese Weise werden alle Elemente des Baums rekursiv besucht.

Wenn es sich bei dem aktuell betrachteten Element um ein Datenmanagementpattern handelt, wird die Methode *transform* der Klasse *PatternTransformer* im Paket *org.eclipse.bpel.simpl.ui.pattern.transformation.transformer* aufgerufen. Diese Methode lädt zunächst die Kontrollstrategie für das zu transformierende Pattern.

Eine Kontrollstrategie muss das in Listing 7.3 dargestellte Interface implementieren. Dieses Interface beinhaltet u.a. die Methoden *hasNextRule* und *getNextRule*. Die erste Methode liefert einen Wahrheitswert. Dieser bestimmt, ob schon alle Transformationsregeln auf Anwendbarkeit geprüft wurden. Die zweite Methode liefert die jeweils nächste Transformationsregel. Eine Transformationsregel wiederum muss das in Listing 7.4 dargestellte Interface implementieren. Die Methode *evaluateConditionPart* überprüft, ob die Transformationsregel auf das Datenmanagementpattern angewendet werden darf. Durch die Methode *applyActionPart* wird der spezifizierte Action Part ausgeführt. Für jedes Datenmanagementpattern wurde also eine Klasse, die die Kontrollstrategie beschreibt implementiert. Des Weiteren wurde für jede der in Kapitel 6 definierten Transformationsregeln eine Klasse implementiert.

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

Listing 7.3 Interface Beschreibung einer Kontrollstrategie

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

  public boolean hasNextRule();

  public TransformationRule getNextRule();
}
```

Listing 7.4 Interface Beschreibung einer Transformationsregel

```
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;
}
```

Die Methode *transform* der Klasse *PatternTransformer* lädt also die zugehörige Kontrollstrategie. Solange nicht alle Regeln dieser Kontrollstrategie auf Anwendbarkeit geprüft wurden und keine der bisher betrachteten Regeln anwendbar war, wird die jeweils nächste Transformationsregel der Kontrollstrategie geladen und der Condition Part evaluiert. Wenn keine Transformationsregel anwendbar ist, wird eine *Exception* geworfen und dem Modellierer mitgeteilt, dass für das Pattern keine geeignete Transformationsregeln gefunden werden konnte. Ansonsten wird der Action Part der gefundenen Transformationsregel ausgeführt. Die Methode *applyActionPart* gibt einen Baum bzw. das Wurzelelement eines Baums zurück. Das Element, das das zu transformierende Datenmanagementpattern im ursprünglichen Baum referenziert, wird durch das neue Wurzelelement ersetzt. Auf diese Weise wird ein Datenmanagementpattern durch ein generiertes Workflow-Fragment ersetzt. Dieses Workflow-Fragment kann wiederum weitere Datenmanagementpatterns beinhalten. Sollen auch diese ersetzt werden, wird das Wurzelelement des eingesetzten Teilbaums wieder der Methode *transform* der Klasse *WorkflowGraphTraverser* übergeben. Auf diese Weise werden rekursiv alle eingebetteten Patterns ersetzt. Sollte keine Transformationsregel anwendbar sein, wird die Transformation der Datenmanagementpatterns abgebrochen und dem Nutzer wird mitgeteilt, für welches Pattern keine Transformationsregel gefunden werden konnte. Im Wesentlichen entsprechen die implementierten Algorithmen den in Abschnitt 5.4.2 vorgestellten Algorithmen (vgl. Algorithmus 5.1 und 5.2) für die Pattern Transformation.

Möchte der Modellier hingegen nicht alle, sondern nur *ein* Datenmanagementpattern ersetzen, muss dieser auf den Menüeintrag *Transform this pattern* oder *Transform this pattern (no recursion)* klicken. Die zugehörige Action ruft in diesem Fall nicht die Methode *transformAllPatterns*, sondern die Methode *transformSinglePattern* der Klasse *PatternTransformationImplemenation* auf. Diese Methode ruft wiederum die Methode *findElementByActivity* der Klasse *WorkflowGraphTraverser* auf, die das XML-Dokument mithilfe der JDOM API einliest und im resultierenden Baum *das* Element sucht, das dem gewünschten Datenmanagementpattern

entspricht. Danach wird dieses Pattern mithilfe der Methode *transform* der Klasse *Pattern-Transformer* genau wie im anderen Fall (wenn alle Patterns transformiert werden sollen) transformiert.

Wenn die Transformation der Datenmanagementpatterns beendet wurde, wird der im Hauptspeicher gehaltene und durch die Transformation der Patterns modifizierte Baum wiederum in ein XML-Dokument verwandelt und persistent gespeichert.

Um wiederkehrende Aufgaben einfacher bewerkstelligen zu können, wurden in der Klasse *PatternTransformationUtils* im Paket *org.eclipse.bpel.simpl.ui.pattern.transformation.util* Methoden implementiert, die öfters benötigt werden. So wurden Methoden implementiert, die in einem JDOM-Baum den Initialwert einer Variablen suchen, das Datenformat einer Variablen bestimmen und in Bezug auf Data Transformation Web Services die WSDL des Prozesses sowie den Deployment Descriptor erweitern.

Zu beachten ist, dass die implementierte Transformationsregel für das Data Format Conversion Pattern der Pandas-Matlab Kopplung nicht alle Bedingungen überprüft, die in Abschnitt 6.3.3 für dieses Pattern definiert werden. Es müsste *Condition 5* evaluiert werden, die prüft, ob in dem Verzeichnis, in dem die zu importierende Eingabedatei liegt, noch zwei weitere Dateien (die die Primärschlüssel sowie die Namen der Variablen beinhalten) liegen. Des Weiteren müssen diese Primärschlüssel und Variablennamen in der referenzierten Tabelle vorkommen. Diese Bedingungen können jedoch nicht so einfach überprüft werden, da die Dateien auf einem entfernten Rechner liegen könnten bzw. bei der Transformation des Patterns zur Modellierungszeit noch gar nicht existent sind. In Abschnitt 8.2 wird eine mögliche Lösung dieses Problems vorgestellt, die auf zusätzlichen Metadaten im Resource Management basiert.

In Bezug auf die in Abbildung 5.9 dargestellte Architekur des Abbildungsmechanismus implementiert die Klasse *WorkflowGraphTraverser* die Komponente **Workflow Graph Traverser**. Die Klassen, die die in Listing 7.3 und 7.4 dargestellten Interfaces implementieren, entsprechen der Komponente **Ruleset and Control Strategies**. Die Komponente **Pattern Transformer Engine** wird von der Klasse *Pattern Transformer* implementiert. In dieser wird die Kontrollstrategie geladen und die Transformationsregeln werden auf deren Anwendbarkeit geprüft. Ebenfalls wird von dieser Klasse die Komponente **Workflow Fragment Replacer** implementiert, da, wenn eine geeignete Transformationsregel gefunden wurde, das ursprüngliche Datenmanagementpattern ersetzt wird. Die einzelnen XML-Dateien bzw. Workflow-Fragmente, die innerhalb der Transformationsregeln geladen werden, entsprechen der Komponente **WF-Library**.

8. Evaluation

In Kapitel 6 werden die im Rahmen dieser Diplomarbeit definierten Workflow-Fragmente und Transformationsregeln vorgestellt. In Kapitel 7 werden u.a. Erweiterungen der GUI sowie des Resource Managements erläutert und auf die konkrete Umsetzung der Pattern Transformation eingegangen. Dieses Kapitel bewertet die geänderten Zugriffsmechanismen (vgl. Abschnitt 8.1) und die einzelnen Workflow-Fragmente und Transformationsregeln hinsichtlich ihrer universellen Einsetzbarkeit (vgl. Abschnitt 8.2). Des Weiteren werden auf Verbesserungen durch eine zukünftig mögliche Datenflussanalsyse bzw. Transformation der Patterns zur Laufzeit eingegangen (vgl. Abschnitt 8.3) und weitere Optimierungsmöglichkeiten (vgl. Abschnitt 8.4) erläutert. Zuletzt wird in Abschnitt 8.5 erneut auf das in Abschnitt 4.1 erläuterte ChemShell Szenario eingegangen, um zu diskutieren, inwieweit der in dieser Arbeit vorgestellte Ansatz auf weitere multi-skalare Simulationsworkflows angewendet werden kann.

8.1. Bewertung der geänderten Zugriffsmechanismen

In Abschnitt 7.1 wird erläutert, dass im Rahmen dieser Arbeit u.a. der SIMPL Core um die generische Operation *TransferData* erweitert wurde. Der einzige Konnektor, der diese Operation momentan implementiert, ist der SSHConnector. Wenn eine Datei, wie z.B. in der Pandas Preprocessing-Phase (vgl. Abschnitte 4.2 und 5.4.3), auf einen entfernten Rechner kopiert werden soll, ruft die Execution Engine die generische TransferData Operation des SIMPL Cores auf. Diese ruft wiederum die TransferData Operation des SSHConnectors auf. Durch dieses Vorgehen konnte die Performanz des Workflows verbessert werden. Vor den durchgeführten Änderungen rief die Execution Engine zuerst die generische RetrieveData Operation und dann die WriteDataBack Operation des SIMPL Cores auf. Es erfolgten also *zwei* Web Service Aufrufe. Zum einen mussten dadurch häufiger und somit insgesamt mehr Daten übertragen werden, zum anderen dauerte der gesamte Vorgang länger (vor allem wenn der Web Service auf einem anderen Rechner ausgeführt wurde). Des Weiteren wurde eine Datei in der in Abschnitt 6.1 beschriebenen XML RowSet Struktur gespeichert, aus dieser wieder extrahiert und im Tomcat-Verzeichnis zwischengespeichert. Erst dann konnte der eigentliche Dateitransfer erfolgen. Nun wird die Datei weder zwischengespeichert noch transformiert, sondern direkt kopiert. Auch dies führt zu einer weiteren Verbesserung der Performanz.

Jedoch funktioniert der implementierte Ansatz nur für Dateien, die mithilfe des SSHConnectors auf einen entfernten Rechner kopiert werden. Um den Datentransfer bei beliebigen multi-skalaren Simulationsworkflows, die andere Daten transferieren bzw. andere Typen von

Datenressourcen nutzen, zu verbessern, müssen evtl. weitere Möglichkeiten des Dateitransfers in das SIMPL-Rahmenwerk integriert werden. Des Weiteren müssten diese Möglichkeiten miteinander verglichen werden, um zu bestimmen, unter welchen Bedingungen welche Möglichkeit am besten geeignet ist. Metadaten über die verschiedenen Möglichkeiten könnten im Resource Management gespeichert werden. Die generische TransferData Operation des SIMPL Cores würde dann, mithilfe dieser Metadaten, die beste Option bestimmen und die Ausführung der Operation des jeweiligen Konnektors und ggf. Konverters steuern. Neben einer Optimierung für den Datentransfer stellt dies auch eine Abstraktionsunterstützung für den Workflow-Modellierer dar, da dieser nicht mehr selbst zwischen diesen verschiedenen Möglichkeiten des Datentransfers wählen muss.

8.2. Bewertung der Workflow-Fragmente und Transformationsregeln

In den Abschnitten 6.2 und 6.3 werden Workflow-Fragmente mit Platzhaltern und Transformationsregeln für die verschiedenen Datenmanagementpatterns, der im Rahmen dieser Arbeit betrachteten Anwendungsfälle (der Pandas-Workflow sowie die Pandas-Matlab Kopplung), definiert. Dabei wurde das Ziel verfolgt, die betrachteten Anwendungsfälle zu unterstützen und gleichzeitig die Workflow-Fragmente und Transformationsregeln **so generisch wie möglich** zu halten. Im Folgenden werden die Beschränkungen der einzelnen Workflow-Fragmente noch einmal kurz zusammengefasst, um dem Leser einen Gesamteindruck zu vermitteln, wie generisch diese sind. Für die konkreten Gründe der Beschränkungen wird auf die jeweiligen Abschnitte verwiesen:

- Das Fragment **simplC2C.xml**: Das einfache Container-to-Container Pattern des Pandas-Workflows wird auf dieses Workflow-Fragment abgebildet (vgl. Abschnitt 6.2.1). Das Fragment bzw. die zugehörige Transformationsregel kann immer dann eingesetzt werden, wenn Daten aus einem Datencontainer an einem referenzierten Ort bereitgestellt werden sollen. Dies entspricht der Definition des einfachen Container-to-Container Patterns (vgl. Abschnitt 5.4.1.1). Das Workflow-Fragment sowie die zugehörige Transformationsregel für dieses Pattern sind somit universell einsetzbar.
- Das Fragment **complexC2C.xml**: Das komplexe Container-to-Container Pattern des Pandas-Workflows wird auf dieses Workflow-Fragment abgebildet (vgl. Abschnitt 6.2.2). Es kann nur dann verwendet werden, wenn die *data container reference variables*, die durch die Eingabeparameter *sourceContainer* und *targetContainer* spezifiziert werden, Verzeichnisse bzw. Dateien innerhalb eines Dateisystems referenzieren.
- Das Fragment **dataFormatConversion_script.xml**: Das Data Format Conversion Pattern des Pandas-Workflows wird auf dieses Fragment abgebildet (vgl. Abschnitt 6.2.3). Dieses Fragment kann immer dann eingesetzt werden, wenn die *data container reference variables*, die durch die Eingabeparameter *sourceContainer* und *targetContainer* spezifiziert werden, Verzeichnisse bzw. Dateien innerhalb eines Dateisystems referenzieren und im Resource Management ein geeigentes *Data Transformation Script* gefunden wird. Ferner muss dieses Script die konvertierte Datei in dem Verzeichnis bereitstellen, in dem auch die Eingabedatei liegt.

- Das Fragment **sequentialDIP.xml**: Der Pandas-Workflow umfasst ein sequentielles Data Iteration Pattern, welches auf dieses Workflow-Fragment abgebildet wird (vgl. Abschnitt 6.2.4). Dieses Fragment kann immer dann eingesetzt werden, wenn der Parameter *data* eine *data container reference list* referenziert und alle *data container reference variables* dieser Liste dasselbe oder kein Datenformat aufweisen.
- Das Fragment **mdt.xml**: Das Multiple Data Transfer Pattern des Pandas Workflows wird auf dieses Fragment abgebildet (vgl. Abschnitt 6.2.5). Für dieses gibt es keine besonderen Restriktionen. Jedoch kann es passieren, dass das eingebettete sequentielle Data Iteration Pattern nicht abgebildet werden kann, wenn z.B. *data container reference variables* ausgewählt wurden, die Objekte in einer Datenbank referenzieren. Zu beachten ist, dass dieses Pattern in der Pandas Postprocessing-Phase nicht eingesetzt werden kann, da dieses Pattern auf ein Workflow-Fragment abgebildet wird, das u.a. ein sequentielles Data Iteration Pattern mit eingebettetem Container-to-Container Pattern beinhaltet. In Abschnitt 6.2.4 wird jedoch erläutert, dass als eingebettete Operation noch eine Assign Aktivität ausgeführt werden muss, um die Referenzen zu vervollständigen.
- Die Fragmente **parallelDIP_choice.xml**, **parallelDIP_gausspoint_table.xml** und **parallelDIP_gausspoint_instance_table.xml**: Das parallele Data Iteration Pattern der Pandas-Matlab Kopplung wird, je nach gewählten Eingabeparametern, auf eines dieser Fragmente abgebildet (vgl. Abschnitt 6.3.1). Diese Fragmente bzw. die zugehörigen Transformationsregeln berücksichtigen Besonderheiten der Kopplungsphase der Pandas-Matlab Kopplung. So können die Fragmente z.B. nur dann eingesetzt werden, wenn der Nutzer eine bestimmte eingebettete Operation (bzw. Operationen) modelliert hat. Darüber hinaus werden die Web Services in diesem Workflow-Fragment proprietär eingesetzt. Des Weiteren wurde auch das Pattern an sich an diesen Anwendungsfall angepasst. Der Nutzer kann die Parameter *referenceList*, *mode*, *simulationId* und *timeStep* spezifizieren, da diese in Bezug auf die Aufteilung des Simulationsraums benötigt werden. In anderen Szenarien werden diese Parameter eventuell nicht oder andere Parameter benötigt. In Zukunft könnte überlegt werden, wie solche Datenaufteilungen generalisert werden könnten und welche Parameter auch in anderen Anwendungsfällen benötigt werden. Weiterhin könnte untersucht werden, welche Abhängigkeiten es hierbei in Bezug auf das sequentielle Data Iteration Pattern gibt.
- Die Fragmente **getTimeStep_variable.xml**, **getTimeStep_literal.xml** und **getTimeStep_first_last.xml**: Das Get Time Step Pattern der Pandas-Matlab Kopplung wird je nach gewählten Eingabeparametern auf eines dieser Fragmente abgebildet (vgl. Abschnitt 6.3.2). Nur für das Fragment *getTimeStep_first_last.xml* gibt es eine Einschränkung: der Parameter *data* muss einer *data container reference variable* entsprechen, die eine Tabelle in einer relationalen Datenbank referenziert, da innerhalb der Transformationsregel ein SQL-Ausdruck generiert wird. Des Weiteren muss der Wert des Parameters *timeStep first* oder *last* lauten, da die Transformationsregel nur diese Ausdrücke auf einen Term abbilden kann. In Zukunft könnten solche Ausdrücke ebenfalls im Resource Management registriert und mithilfe der dort gespeicherten Metadaten auf einen Term sowie andere Informationen, die für das Auslesen des Zeitschritts nötig sind,

abgebildet werden. Dann wäre die Transformationsregel noch generischer, da auch andere Ausdrücke, als *first* oder *last*, vom Modellierer angegeben werden könnten.

- Das Fragment **dataFormatConversion_importWS.xml**: Das Data Format Conversion Pattern der Pandas-Matlab Kopplung wird auf dieses Fragment abgebildet (vgl. Abschnitt 6.3.3). Dieses Fragment kann nur dann eingesetzt werden, wenn die Datenformate der Eingabeparameter *sourceContainer* und *targetContainer CSV* und *GausspointInstanceTable* lauten. In Abschnitt 7.4 wird erläutert, dass die zugehörige Transformationsregel nicht prüft, ob die referenzierte Datei sowie die Tabelle kompatibel zueinander sind, da diese Überprüfung schwierig bzw. überhaupt nicht durchführbar ist (wenn z.B. die referenzierte Datei zur Modellierungszeit noch nicht existiert). In Zukunft könnte evaluiert werden, ob das Datenformat solcher CSV-Dateien nicht anders benannt werden sollte, da das Datenformat *CSV* recht generisch ist. Es könnte ein neues Format definiert werden. Des Weiteren könnte das Resource Management dahingehend erweitert werden, dass Eigenschaften bzw. Bedingungen definiert werden können, die ein entsprechendes Objekt, das diesem Datenformat entspricht, aufweisen muss. Auf diese Weise wird die Transformationsregel wiederum generisch, da *Condition 5* (vgl. Abschnitt 6.3.3) nicht mehr überprüft werden muss, da diese Bedingungen automatisch für das Format gelten.

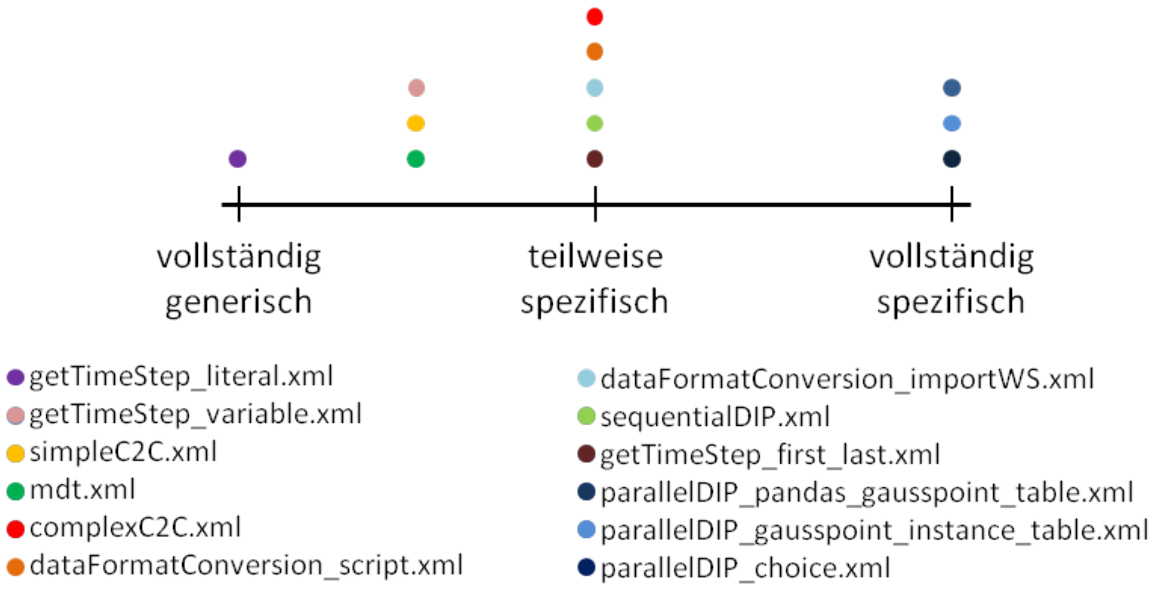

Abbildung 8.1.: Grad der universellen Einsetzbarkeit der einzelnen Workflow-Fragmente

Abbildung 8.1 ordnet die einzelnen Workflow-Fragmente dem Grad der universellen Einsetzbarkeit zu. Obwohl in obiger Auflistung erläutert wird, dass es für die Workflow-Fragmente *getTimeStep_variable.xml*, *simpleC2C.xml* und *mdt.xml* keine Restriktionen gibt, verdeutlicht

diese Abbildung, dass die Fragmente nicht vollständig generisch sind. Das liegt an folgendem Grund: es ist nicht immer klar, welchen Wert eine Variable hat, da im Rahmen dieser Diplomarbeit keine Datenflussanalyse durchgeführt wird und innerhalb der Transformationsregeln nur die Initialwerte der Variablen ausgelesen werden (vgl. Abschnitt 6.2.1). Dadurch wird die universelle Einsetzbarkeit aller Workflow-Fragmente bzw. der zugehörigen Transformationsregel eingeschränkt, da zwischen der Initialisierung der Variablen und dem Datenmanagementpattern keine Wertezuweisung erfolgen darf. In Abschnitt 8.3 wird diese Problematik erneut aufgegriffen. Das einzige Workflow-Fragment, das davon ausgenommen und vollständig generisch ist, ist das Fragment *getTimeStep_literal.xml*, da in der zugehörigen Transformationsregel keine Variablenwerte ausgelesen werden müssen und die Problematik somit nicht auftritt.

Jedoch gibt es bei allen weiteren Workflow-Fragmenten, bis auf die Workflow-Fragmente für das parallele Data Iteration Pattern der Pandas-Matlab Kopplung, da diese auf ein Anwendungsszenario zugeschnitten sind, nur geringfügige Einschränkungen. Daher sind diese zumindest teilweise generisch und können somit auch in anderen Anwendungsszenarien für die Transformation eines Datenmanagementpatterns eingesetzt werden.

Im Rahmen dieser Diplomarbeit wird erläutert, dass der Modellierer eines Workflows auch ETL Patterns, wie z.B. das Data Format Conversion Pattern, modellieren kann. In Zukunft könnte evaluiert werden, ob diese Möglichkeit nicht wieder aus der Modellierungsumgebung entfernt werden sollte. In den betrachteten Anwendungsfällen werden, mithilfe von Data Format Conversion Patterns, Dateien, die im TecPlot-Format vorliegen, in das VTK-Format konvertiert (Pandas-Workflow) und CSV-Dateien in eine Datenbank importiert (Pandas-Matlab Kopplung). Im letzteren Fall werden z.B. Daten über Rechnergrenzen hinweg konvertiert. Des Weiteren werden Daten durch die Konvertierung in einem anderen Datencontainer bereitgestellt. Diese Trennung zwischen Container-to-Container Pattern und ETL Pattern könnte den Modellierer verwirren. Dieser weiß eventuell nicht, welches Pattern das Richtige ist. Wenn es keine ETL Patterns in der Modellierungsumgebung geben würde, könnte der Modellierer nur das Container-to-Container Pattern bzw. andere Datentransfer- und -transformationspatterns auswählen. Dem Nutzer wird die Modellierung demzufolge erleichtert. Die im Rahmen dieser Arbeit erläuterten Datenformatkonvertierungen oder zukünftig auch weitere ETL Operationen wären dann Bestandteil der jeweiligen Datentransfer- und -transformationspatterns. Solche ETL Operationen werden dann entweder implizit über die Datenformate der beteiligten Container angegeben oder aber durch die Angabe der Operation, z.B. einer Filter-Operation, explizit definiert.

8.3. Die Notwendigkeit einer Datenflussanalyse bzw. einer Transformation zur Laufzeit

In Abschnitt 8.2 wird erläutert, dass selbst die Workflow-Fragmente, die keinen besonderen Restriktionen unterliegen, aufgrund der Tatsache, dass keine Datenflussanalyse durchgeführt wird, nicht vollständig universell einsetzbar sind. Das Problem ist, dass Variablenwerte zwischen der Initialisierung und dem Datenmanagementpattern nicht verändert werden

dürfen, da ansonsten falsche Werte bei der Pattern Transformation berücksichtigt werden. Im Folgenden werden nun einzelne Patterns bzw. Workflow-Fragmente erneut betrachtet und es wird erläutert, welche Verbesserungen eine Datenflussanalyse bzw. eine Transformation zur Laufzeit, bewirken würde.

Workflow-Fragmente (wie z.B. die Fragmente *simpleC2C.xml* und *complexC2C.xml*), deren zugehörige Transformationsregeln Werte referenzierter Variablen prüfen (wie z.B. ob das Datenformat gleich ist) könnten, wenn eine Datenflussanalyse zur Modellierungszeit durchgeführt würde, auch dann eingesetzt werden, wenn Variablenwerte während der Ausführung des Workflows verändert werden würden.

Eine Transformation der Datenmanagementpatterns zur Laufzeit würde die universelle Einsetzbarkeit noch weiter erhöhen, da dann z.B. auch Variablen referenziert werden könnten, denen ein Wert von externen Komponenten, wie z.B. durch einen Web Service, zugewiesen wird. Dasselbe gilt für das Multiple Data Transfer Pattern: bisher werden bei der Erzeugung der *data container reference list* lediglich die Initialwerte der Variablen berücksichtigt, was die universelle Einsetzbarkeit einschränkt.

Auch das Workflow-Fragment *sequentialDIP.xml* für das sequentielle Data Iteration Pattern könnte häufiger eingesetzt werden. Die zugehörige Transformationsregel hat zur Bedingung, dass alle Container der referenzierten *data container reference list* (vgl. Abschnitt 6.2.4) dasselbe oder kein Datenformat aufweisen. Dies ist nötig, da in diesem Workflow-Fragment eine lokale *data container reference variable* erzeugt und initalisiert wird. Wird diese Variable nicht initialisiert, können eingebettete Patterns unter Umständen nicht abgebildet werden. Des Weiteren wird eine lokale Kopie der referenzierten *data container reference list* erzeugt, die die jeweiligen *data source references* enthält. Eine Transformation zur Laufzeit würde diese Probleme beseitigen bzw. vereinfachen, da solche Variableninitialisierungen und Workarounds sehr wahrscheinlich entfallen würden. Eingebettete Patterns könnten bei jeder Iteration erneut transformiert werden und somit die in der Iteration gültigen Werte berücksichtigen.

Generell würde eine Datenflussanalyse bei der Transformation der Datenmanagementpatterns zur Modellierungszeit die universelle Einsetzbarkeit aller Fragmente erhöhen, die Variablenwerte auslesen, da Variablenwerte auch noch nach der Initialisierung geändert werden könnten und bei der Transformation berücksichtigt werden würden. Noch besser wäre allerdings eine Transformation zur Laufzeit, da die Patterns in diesem Fall auch Variablen referenzieren könnten, denen ein Wert beispielsweise durch einen Web Service zugewiesen wird.

Jedoch ergibt sich bei einer Transformation der Datenmanagementpatterns zur Laufzeit ein Nachteil, da diese Transformation die Dauer der Workflow-Ausführung beeinflusst. In weiteren Arbeiten könnte der Nutzen einer Transformation der Patterns zur Laufzeit diesem Nachteil gegenübergestellt werden.

8.4. Weitere Optimierungsmöglichkeiten

In Abschnitt 3.7 wird erläutert, dass die Kontrollstrategie die Reihenfolge bestimmt, in der die einzelnen Transformationsregeln auf Anwendbarkeit geprüft werden. Bei der im Rahmen dieser Diplomarbeit erarbeiteten Implementierung ist die Reihenfolge hart codiert. In den meisten Fällen werden Transformationsregeln, die weniger Bedingungen evaluieren, zuerst geprüft (vgl. Abschnitt 6.4).

In Zukunft könnten Optimierungsentscheidungen (vgl. [VSS+07]) berücksichtigt bzw. deren Nutzen evaluiert werden. So könnten z.B. Optimierungsentscheidungen in den Transformationsregeln und/oder Kontrollstrategien getroffen werden. Im Folgenden wird jeweils ein Beispiel beschrieben.

In Abschnitt 6.2.3 wird erläutert, dass das Data Format Conversion Pattern des Pandas Workflows auf ein Fragment abgebildet wird, in dem zuerst der Dateitransfer und dann die Datenformatkonvertierung durchgeführt wird. Dies geschieht aus folgendem Grund: Dateien im TecPlot-Format sind in der Regel kleiner als ins VTK-Format konvertierte Dateien. Damit muss eine insgesamt kleinere Datenmenge übertragen werden. In Zukunft könnten Transformationsregeln dynamisch entscheiden, welche Aktivtität zuerst ausgeführt wird. Je nachdem, ob die ursprüngliche Datei oder die konvertierte Datei kleiner ist. Diese Optimierungsentscheidung kann analog zu den Optimierungsregeln aus [VSS+07] im Bedingungsteil der jeweiligen Regel repräsentiert werden.

Ebenso könnten Kontrollstrategien Optimierungsentscheidungen beinhalten. So könnten z.B. Transformationsregeln, deren resultierende Workflow-Fragmente eine bessere Performanz (können in kürzerer Zeit ausgeführt werden) aufweisen, jedoch nicht immer einsetzbar sind, zuerst geprüft werden. Erst danach werden Transformationsregeln, die eine schlechtere Performanz der Workflow-Fragmente verursachen, dafür aber häufiger eingesetzt werden können, geprüft.

Ein weiterer Performanz-Vorteil wäre möglich, wenn Bedingungen nicht mehrmals geprüft werden würden. Müssen mehrere Transformationsregeln für ein Datenmanagementpattern evaluiert werden, kann es passieren, dass innerhalb der Regeln gleiche Bedingungen mehrmals evaluiert werden (wenn z.B. zwei Transformationsregeln die selben oder teilweise die selben Bedingungen überprüfen, wie es bei den Regeln für das einfache und komplexe Container-to-Container Pattern der Fall ist (vgl. Abschnitte 6.2.1 und 6.2.2)). Um dies zu verhindern, könnten in Zukunft in den Transformationsregeln Bedingungsblöcke definiert werden. Wenn solch ein Bedingungsblock evaluiert wurde, wird dies durch die Kontrollstrategie festgehalten. Weitere Transformationsregeln, die ebenfalls diese Blöcke enthalten, ignorieren diese Blöcke dann, da diese bereits evaluiert wurden. Auf diese Weise kann der Condition Part einer Transformationsregel, wenn enthaltene Bedingungsblöcke schon evaluiert wurden, schneller überprüft werden.

Eine weitere Möglichkeit, um Bedingungen nicht mehrmals zu evaluieren, wäre die Definition hierarchischer Regeln. In so einem Fall würden im Action Part einer Transformationsregel weitere Regeln oder sogar eine weitere Kontrollstrategie, die wiederum mehrere Regeln umfasst, definiert werden. Der Vorteil ist, dass bei diesen eingebetteten Regeln, die äußeren

Bedingungen nicht nochmal definiert und evaluiert werden müssen und somit die mehrfache Bedingungsevaluation vermieden wird.

8.5. Datenmanagementpatterns in Bezug auf ChemShell

Abschließend wird noch einmal der Ansatz der Datenmanagementpatterns auf konzeptioneller Ebene betrachtet. Es wird erläutert, wie die in Abschnitt 5.4.1 formalisierten Datenmanagementpatterns im beschriebenen ChemShell Szenario (vgl. Abschnitt 4.1) genutzt werden könnten. Dabei werden die Ergebnisse der Definition der einzelnen Workflow-Fragmente aus Kapitel 6 berücksichtigt. Auf die einzelnen Details wird jedoch nicht eingangen.

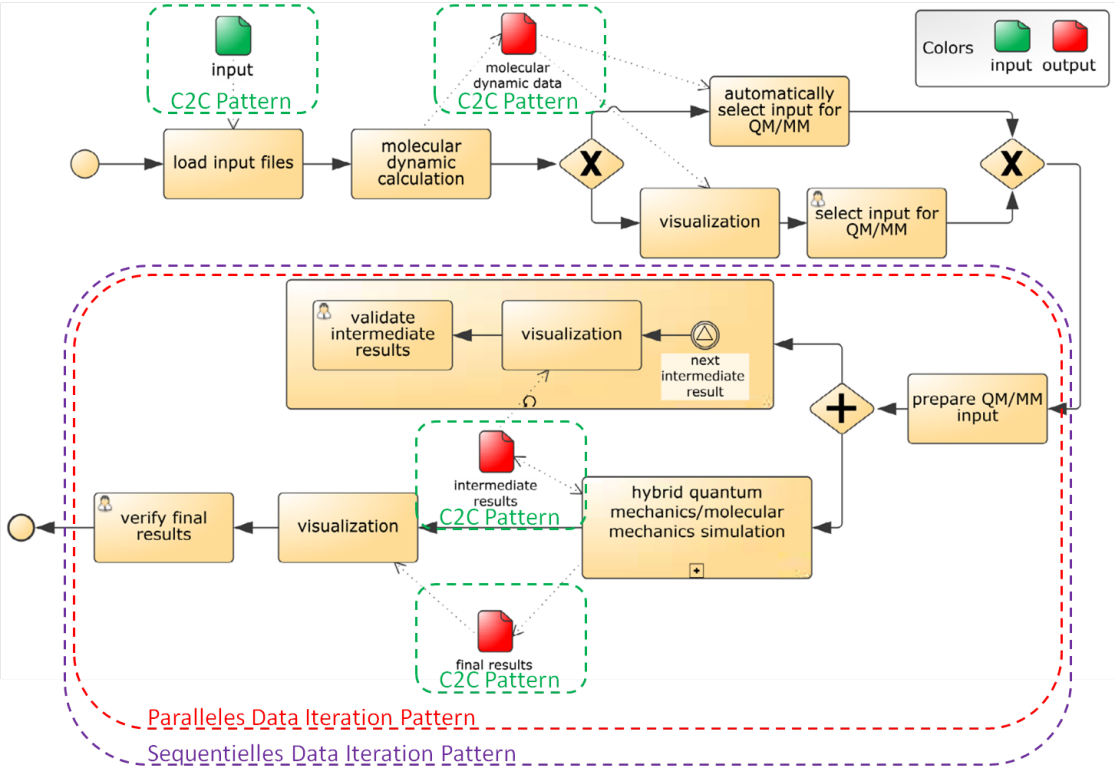

Abbildung 8.2.: Datenmanagementpatterns im ChemShell Workflow (in Anlehnung an [Mü10])

In Abschnitt 2.4.3.2 wird erläutert, dass mithilfe von ChemShell hybride quantenmechanische/molekularmechanische Simulationen durchgeführt werden können. Das Konzept eines Simulationsworkflows, der die Konversion eines Glutamats in Methyl-Aspartat simuliert, wird in Abbildung 4.1 dargestellt. Abbildung 8.2 zeigt diesen Workflow mit möglichen Datenmanagementpatterns. Die verschiedenen Einsatzmöglichkeiten werden im Folgenden

erläutert. Zu beachten ist allerdings, dass im Rahmen dieser Diplomarbeit keine tiefergehende Evaluation möglich war, da die Implementierung des Workflows noch nicht ausgereift war und des Weiteren nicht diesem konzeptionellen Ablauf entspricht.

An mehreren Stellen des Workflows werden Daten geladen bzw. Zwischen- oder Endergebnisse gespeichert. Bei solchen Datenoperationen könnten Container-to-Container Patterns eingesetzt werden (grüne Markierungen in der Abbildung). So könnte z.B. die Aktivität *load input files*, die eine Proteindatei aus einer Datenbank lädt, durch ein Container-to-Container Pattern ersetzt werden.

Des Weiteren könnte eine paralleles Data Iteration Pattern in diesen Workflow integriert werden. Bevor die hybride quantenmechanische/molekularmechanische Simulationen ausgeführt wird, muss der Nutzer Teile des Simulationsraums selektieren (*automatically select input for QM/MM* bzw. *select input for QM/MM*), auf denen die nachfolgende Simulation ausgeführt wird. Es wird nicht der gesamte Simulationsraum berücksichtigt, da bei solch einer Simulation versucht wird die Schrödinger Gleichung zu lösen, was in der Regel sehr rechen- und datenintensiv ist. Ein paralleles Data Iteration Pattern (rote Markierung in der Abbildung) könnte auch hier über eine Datenmenge S iterieren. Diese Datenmenge S beschreibt z.B. den gesamten Simulationsraum und wird gemäß der vom Nutzer festgelegten Aufteilung des Simulationsraums oder weiterer Parallelisierungskriterien in n $(n > 0)$ Teilmengen aufgeteilt und auf n Instanzen bzw. Rechner verteilt. Zu beachten ist, dass entgegen zur Pandas-Matlab Kopplung diese n Teilmengen in ihrer Vereinigung nicht unbedingt die gesamte Datenmenge S ergeben müssen. Die einzelnen Instanzen führen dann die quantenmechanische/molekularmechanische Simulation auf der jeweiligen Teilmenge aus. Anschließend werden die Ergebnisse der n Instanzen in die urspüngliche Datenmenge integriert. In obigem Workflow bzw. in obiger Abbildung wird nur eine Iteration ausgeführt. Dieses Vorgehen würde durch das parallele Data Iteration Pattern ebenfalls unterstützt werden. Jedoch bietet das Pattern auch die Möglichkeit, wie gerade beschrieben, mehrere Iterationen durchzuführen und somit mehrere Teile des betrachteten Simulationsraums parallel zu untersuchen.

Da im Rahmen dieser Arbeit keine tiefergehende Evaluation möglich war, müsste in Zukunft noch evaluiert werden, ob die Aktivitäten (*automatically select input for QM/MM* bzw. *select input for QM/MM*) Bestandteil der eingebetteten Operation des parallelen Data Iteration Patterns sein müssten. Des Weiteren werden beim erläuterten Vorgehen in jeder Iteration neue Zwischenergebnisse visualisiert und durch den Nutzer validiert (der Block, der die Aktivitäten *visualization* und *validate intermediate results* beinhaltet). Dieser Block entspricht ebenfalls einer Schleife. In Zukunft könnte evaluiert werden, ob auch an dieser Stelle ein Data Iteration Pattern eingesetzt werden könnte. Darüber hinaus müssten die Aktivitäten *visualization* und *verify final results*, die bei jeder Iteration die Endergebnisse in Bezug auf die jeweilige Teilmenge visualisieren und durch den Nutzer evaluieren lassen, nicht Bestandteil der eingebetteten Operation des parallelen Data Iteration Patterns sein. Stattdessen könnten auch die Endergebnisse in Bezug auf **alle** Teilmengen nach der Ausführung des parallelen Data Iteration Patterns visualisiert und durch den Nutzer evaluiert werden.

Dieses parallele Data Iteration Pattern könnte wiederum in ein sequentielles Data Iteration Pattern (violette Markierung in der Abbildung) eingebettet werden. Dieses sequentielle Data Iteration Pattern iteriert wie oben über die Datenmenge S, die den gesamten Simulationsraum

beschreibt. Es selektiert gemäß den Vorgaben des Nutzers jedesmal eine Teilmenge $S_i \subseteq S$, die alle relevanten Teile des Simulationsraums umfasst. Das eingebettete parallele Data Iteration Pattern wird auf diese Teilmenge S_i angewendet und führt die gerade erläuterte Simulation mithilfe der n Instanzen aus. Zu beachten ist, dass die Vereinigung dieser Teilmengen des eingebetteten parallelen Data Iteration Patterns diesmal zwar der Teilmenge S_i, aber nicht zwingenderweise der Gesamtmenge S entspricht. Anschließend werden die jeweiligen Ergebnisse $S_{i'}$ wieder in die Gesamtmenge S integriert. Nachdem das parallele Data Iteration Pattern ausgeführt wurde, wird der Nutzer durch das sequentielle Data Iteration Pattern gefragt, ob die Resultate seinen Anforderungen genügen. Wenn dies der Fall ist, wird das sequentielle Data Iteration Pattern beendet und somit auch der gesamte Workflow. Wenn der Nutzer mit den Ergebnissen nicht zufrieden sein sollte, kann dieser wiederum einen anderen (z.B. einen größeren) Teil des Simulationsraums bestimmen, auf den dann das eingebettete parallele Data Iteration Pattern angewendet wird. Auf diese Weise werden die Ergebnisse bei jeder Iteration des sequentiellen Data Iteration Patterns weiter verfeinert.

9. Zusammenfassung und Ausblick

SIMPL bietet die Möglichkeit, aus Simulationsworkflows heraus, auf beliebige Datenquellen über vereinheitlichte Schnittstellen zuzugreifen. Ein Bestandteil des SIMPL-Rahmenwerks sind die im Rahmen dieser Arbeit betrachteten Datenmanagementpatterns. Dabei handelt es sich um vorgefertigte Datenmanagement-Operationen, die nur noch parametrisiert werden müssen. Auf diese Weise wird eine zusätzliche Abstraktionsebene geschaffen.

In dieser Arbeit wurde der in [Ari12] entwickelte Ansatz erweitert und wenn nötig angepasst, um auf multi-skalare Simulationen, wie z.B. die Pandas-Matlab Kopplung, angewendet werden zu können. Konkret wurde Folgendes geleistet:

- In Bezug auf Dateisysteme wurden die Zugriffsmechanismen von SIMPL überarbeitet bzw. erweitert. Unix-basierte Dateisysteme werden unterstützt und Daten werden direkt, ohne temporäre Speicherung im Tomcat-Verzeichnis, durch den SSHConnector auf einen entfernten Rechner kopiert (vgl. Abschnitt 7.1).
- Die in [Ari12] definierten Workflow-Fragmente und Transformationsregeln für das *Container-to-Container Pattern*, das *Data Format Conversion Pattern* und das *sequentielle Data Iteration Pattern* wurden überarbeitet. Dabei wurde das Ziel verfolgt, die einzelnen Workflow-Fragmente so generisch wie möglich zu definieren (vgl. Abschnitte 6.2.1 - 6.2.4).
- Im Pandas-Workflow wurde ein neues Pattern, das sogennante *Multiple Data Transfer Pattern vorgestellt* (vgl. Abschnitt 6.2.5). Dieses Pattern ermöglicht es dem Modellierer n $(n > 0)$ Container auszuwählen, die an einem referenzierten Ort bereitgestellt werden. Es bietet damit eine weitere Abstraktion für diesen Anwendungsfall als die bisher betrachteten Patterns.
- In einem weiteren Schritt wurde die Pandas-Matlab Kopplung analysiert und ein *paralleles Data Iteration Pattern* im Data-Manager Workflow identifiziert (vgl. Abschnitt 6.3.1). Dieses Pattern teilt den Simulationsraum in n $(n > 0)$ Teilmengen auf, führt eine vom Nutzer definierte Operation, in diesem Fall die Matlab-Simulation, auf der jeweiligen Teilmenge aus und fasst die Ergebnisdaten wiederum in einer Datenmenge zusammen, die dann wiederum als Eingabe für die Pandas-Simulation genutzt werden kann. Für das parallele Data Iteration Pattern der Pandas-Matlab Kopplung wurden im Rahmen dieser Arbeit drei Workflow-Fragmente sowie zugehörige Transformationsregeln definiert.
- Des Weiteren wurden in den Workflows der Pandas-Matlab Kopplung noch ein weiteres *Data Format Conversion Pattern* sowie das neue *Get Time Step Pattern* identifiziert (vgl. Abschnitte 6.3.2 und 6.3.3). Das Letztere liefert für einen Ausdruck wie z.B. *first* oder *last*

eine konkrete Zahl, die den entsprechenden Zeitschritt in einer Datenbank repräsentiert. Dadurch können die Problemstellungen, die durch diese Patterns gelöst werden, auch in anderen Anwendungsszenarien durch den Einsatz der Patterns gelöst werden.

- Bisher fanden alle Entwicklungen bzgl. der Datenmanagementpatterns nur auf konzeptioneller Ebene statt. Im Rahmen dieser Arbeit wurde auch die prototypische Umsetzung des SIMPL-Rahmenwerks um Datenmanagementpatterns erweitert (vgl. Kapitel 7). Der Modellierer kann Patterns modellieren und diese zur Modellierungszeit auf ausführbare Workflow-Fragmente transformieren. Ebenso wurde der Pandas-Workflow sowie die Workflows der Pandas-Matlab Kopplung auf *data container references* umgestellt.
- Auch das Resource Management wurde erweitert (vgl. Abschnitt 7.3). Nun können im Resource Management die folgenden drei Arten von *Data Transformation Services* gespeichert werden: *Service Bus Data Transformation Services*, *Data Transformation Scripts* und *Data Transformation Web Services*. Zudem können externe Datenformate, Metadaten über einzelne Tabellen und XML-Schemas im Resource Management registriert werden.
- In Kapitel 8 wird der im Rahmen dieser Diplomarbeit erarbeitete Ansatz bzgl. der Datenmanagementpatterns evaluiert. Es wird u.a. erläutert, dass die definierten Workflow-Fragmente und Transformationsregeln, bis auf die Fragmente für die Pandas-Matlab Kopplung, zumindest teilweise generisch sind und in anderen Anwendungsfällen eingesetzt werden können. Des Weiteren werden Verbesserungen und Optimierungsmöglichkeiten dargelegt, um die universelle Einsetzbarkeit der einzelnen Workflow-Fragmente weiter zu erhöhen und die Performanz der Pattern Transformation zu verbessern.

Mithilfe der durchgeführten Änderungen und Erweiterungen konnte der Ansatz der Datenmanagementpatterns auf multi-skalare Simulationen übertragen werden. In erster Linie wurde in dieser Arbeit die Pandas-Matlab Kopplung betrachtet. Bei dieser Kopplung werden nun alle Datenmanagementschritte durch Datenmanagementpatterns repräsentiert. Des Weiteren werden in Kapitel 8 Einsatzmöglichkeiten von Datenmanagementpatterns in einem Chemshell Simulationsworkflow erläutert. Für weitere Arbeiten bieten sich insbesondere die folgenden Fragestellungen bzw. Aufgaben an:

- In Abschnitt 8.1 wird erläutert, dass in Zukunft eventuell weitere Möglichkeiten des Dateitransfers in das SIMPL-Rahmenwerk integriert und diese miteinander verglichen werden sollten. Metadaten über die einzelnen Möglichkeiten könnten im Resource Management gespeichert werden. Die TransferData Operation des SIMPL Cores würde die jeweils beste Möglichkeit bestimmen und die Ausführung der Operation des jeweiligen Konnektors und ggf. Konverters steuern. In weiteren Arbeiten könnte der Nutzen dieser Idee weiter evaluiert und die Idee ggf. in das SIMPL-Rahmenwerk integriert werden.
- In Abschnitt 5.4.1.3 wird erläutert, dass das *parallele Data Iteration Pattern* drei Phasen umfasst: die *Split Stage*, die *Operation Stage* sowie die *Merge Stage*. In der *Operation Stage* wird die vom Nutzer definierte Operation auf die einzelnen Teilmengen angewendet. Bei den Transformationsregeln für das parallele Data Iteration Pattern der

Pandas-Matlab Kopplung ist diese Unterteilung nicht möglich (vgl. Abschnitt 6.3.1). Teile der vom Nutzer definierten Operation müssen vor dem eigentlichen *Data Split* ausgeführt werden, um das Arbeitsverzeichnis auf dem jeweiligen Matlab-Rechner zu erzeugen. In Zukunft könnte evaluiert werden, ob es beim parallelen Data Iteration Pattern neben der eingebetteten Operation noch Aktivitäten zur Vorbereitung des *Data Splits* und zur Nachbereitung des *Data Merge* geben sollte. Außerdem könnte die Modellierungsumgebung insofern verbessert werden, als dass diese Operationen inklusive des Data Splits und Data Merges etwas intuitiver bei der Modellierung der Patterns angegeben werden kann.

- In Zukunft könnten weitere Anwendungsfälle, wie z.B. ChemShell-Simulationen, betrachtet werden. Zum einen könnte versucht werden, die universelle Einsetzbarkeit der einzelnen Workflow-Fragmente zu erhöhen, zum anderen könnte untersucht werden, wann Fragmente generisch sein sollten und wann eher nicht.
- Ein weiteres Problem besteht darin, dass bei einer Transformation der Datenmanagementpatterns zur Modellierungszeit nicht immer klar ist, welchen Wert eine Variable hat (vgl. Abschnitt 6.2.1). Bei der verwirklichten Implementierung werden die Initialwerte der Variablen berücksichtigt. In Zukunft könnte eine Datenflussanalyse durchgeführt werden, um den Variablenwert zu bestimmen. Dann würden auch Werteänderungen während des Workflows berücksichtigt werden.
- Des Weiteren könnte eine Pattern Transformation zur Laufzeit umgesetzt werden. Dies hätte den Vorteil, dass die Workflow-Fragmente, auf welche die Patterns abgebildet werden, bis kurz vor der Ausführung des Datenmanagementpatterns geändert werden könnten. Darüber hinaus könnten Datenmanagementpatterns auch Variablen referenzieren, denen ein Wert von externen Komponenten, wie z.B. durch einen Web Service, zugewiesen wird (vgl. Abschnitt 8.3).
- In Abschnitt 7.3 wird erläutert, dass der Begriff *Data Transformation Services* für alle Services bzw. Programme verwendet wird, die Daten z.B. über ETL Operationen transformieren können. In dieser Arbeit wird lediglich auf die Umsetzung für die ETL Operation Datenformatkonvertierung eingegangen. In Zukunft könnte eine entsprechende Metadaten-Unterstützung für weitere ETL Operationen umgesetzt werden.
- Es könnte evaluiert werden, ob ETL Patterns aus der Modellierungsumgebung entfernt und dafür als Bestandteil der Datentransfer- und -transformationspatterns definiert werden sollten, um den Modellierer nicht zu verwirren. Dieser kann aufgrund der Vielzahl an Patterns eventuell nicht entscheiden, welches das für sein Szenario geeignete Pattern ist (vgl. Abschnitt 8.2).
- Bei der Ausführung der Workflows der Pandas-Matlab Kopplung werden Operationen der in [Dor11] beschriebenen Interfaces aufgerufen. Im Rahmen dieser Arbeit wurden die einzelnen Workflows auf *data container references* umgestellt. Jedoch können den Operationen der Interfaces diese *data container references* nicht übergeben werden. Bei der Ausführung der Workflows werden solche Referenzen in den Workflows aufgelöst. Das geht in den betrachteten Anwendungsfällen, da *immer* Objekte in einem Dateisystem referenziert werden. In Zukunft könnte man auch die Web Services auf

data container references umstellen und dabei das Metadaten Management von SIMPL und dieser Web Services konsolidieren.

- In Abschnitt 6.2 wird erläutert, dass bei der Pattern Transformation davon ausgegangen wird, dass Datencontainer stets über deren lokale Bezeichner referenziert werden und nicht im Resource Management registriert sind. Zukünftig könnten auch Referenzen, die einen logischen Namen beinhalten, unterstützt werden. Bei der Pattern Transformation könnten, wenn ein Datencontainer mithilfe eines logischen Namens referenziert wird, die im Resource Management gespeicherten Metadaten geladen und bei der Evaluation des Condition Parts (wenn z.B. das Datenformat überprüft werden muss) bzw. der Ausführung des Action Parts einer Transformationsregel verwendet werden. In weiteren Arbeiten könnte diesbezüglich untersucht werden, welche Abhängigkeiten bzw. Probleme sich durch ein solches Vorgehen ergeben würden.
- In Zukunft könnten Optimierungsentscheidungen (vgl. Abschnitt 8.4) in das SIMPL-Rahmenwerk integriert werden. Diese könnten Bestandteil der Transformationsregeln und/oder Kontrollstrategien sein.
- Wenn ein Datenmanagementpattern transformiert werden soll, kann es passieren, dass erst die n-te Transformationsregel einer Kontrollstrategie anwendbar ist. Vorher wurden also $n-1$ Regeln geprüft, die nicht anwendbar waren. Wenn die Transformationsregeln teilweise die gleichen Bedingungen prüfen, werden gleiche Bedingungen mehrmals geprüft. In Zukunft könnten Bedingungsblöcke (vgl. Abschnitt 7.4) eingeführt werden. Mehrere Bedingungen werden dann zu einem Block zusammengefasst. Beim Übergang zur nächsten Transformationsregel einer Kontrollstrategie, könnte dieser in so einem Fall mitgeteilt werden, welche Bedingungsblöcke schon evaluiert wurden. Diese Bedingungen müssen dann nicht erneut geprüft werden. Ebenso könnten hierarchische Regeln definiert werden. In einem solchen Fall würden im Action Part einer Transformationsregel weitere Regeln oder sogar eine weitere Kontrollstrategie definiert werden. Der Vorteil ist, dass bei diesen eingebetteten Regeln, die äußeren Bedingungen nicht nochmal definiert und evaluiert werden müssen.
- Wenn keine Transformationsregel der Kontrollstrategie auf ein zu transformierendes Datenmanagementpattern anwendbar ist, kommt es zur sogenannten *Escalation* (vgl. Abschnitt 5.4.2). Bei der im Rahmen dieser Arbeit verwirklichten Implementierung wird die Pattern Transformation in so einem Fall abgebrochen und dem Nutzer der Grund mitgeteilt. In Zukunft könnten auch andere Maßnahmen ergriffen werden. So könnte z.B. der Nutzer ein eigenes Workflow-Fragment modellieren. Außerdem kann das Konzept der *Escalation* auch als Bestandteil von Transformationsregeln verwendet werden, z.B. wenn in einigen Regeln Informationen fehlen, die vom Nutzer abgefragt werden müssen.

A. Struktur von WS-BPEL

Listing A.1: Grundlegende Struktur von WS-BPEL [JE07]

```
<process name="NCName" targetNamespace="anyURI" queryLanguage="anyURI" ?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">

  <extensions>?
    <extension namespace="anyURI" mustUnderstand="yes|no" />+
  </extensions>

  <import namespace="anyURI"?
   location="anyURI"?
   importType="anyURI" />*

  <partnerLinks>?
    <!-- Note: At least one role must be specified. -->
    <partnerLink name="NCName"
    partnerLinkType="QName"
    myRole="NCName"?
    partnerRole="NCName"?
    initializePartnerRole="yes|no"?>+
    </partnerLink>
  </partnerLinks>

  <messageExchanges>?
    <messageExchange name="NCName" />+
  </messageExchanges>

  <variables>?
    <variable name="BPELVariableName"
    messageType="QName" ? type="QName" ?
    element="QName" ?>+
    from-spec?
    </variable>
  </variables>

  <correlationSets>?
    <correlationSet name="NCName" properties="QName-list" />+
  </correlationSets>

  <faultHandlers>?
    <!-- Note: There must be at least one faultHandler -->
    <catch faultName="QName"?
      faultVariable="BPELVariableName"?
```

```
      faultMessageType="QName" | faultElement="QName" )>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
  </faultHandlers>

  <eventHandlers>?
    <!-- Note: There must be at least one onEvent or onAlarm. -->
    <onEvent partnerLink="NCName"
       portType="QName"?
       operation="NCName"
       (messageType="QName" | element="QName" )?
       variable="BPELVariableName"?
      messageExchange="NCName"?>*
      <correlations>?
        <correlation set="NCName" initiate="yes|join|no" ? />+
      </correlations>
      <fromParts>?
        <fromPart part="NCName" toVariable="BPELVariableName" />+
      </fromParts>
      <scope...>...</scope>
    </onEvent>
    <onAlarm>*
      <!-- Note: There must be at least one expression. -->
      (
      <for expressionLanguage="anyURI" ?>duration-expr</for>
      |
      <until expressionLanguage="anyURI" ?>deadline-expr</until>
      )?
      <repeatEvery expressionLanguage="anyURI" ?>
        duration-expr
      </repeatEvery>?
      <scope...>...</scope>
    </onAlarm>
  </eventHandlers>
  activity
</process>
```

B. Workflow-Fragmente

Listing B.1: Das Workflow-Fragment *simplC2C.xml*

```
<bpel:extensionActivity>
  <simpl:transferDataActivity dataSource="?dataSource" dataSourceCommand="?dataSourceCommand"
      dataSink="?dataSink" dataSinkContainer="?dataSinkContainer" name="Transfer_Data"/>
</bpel:extensionActivity>
```

Listing B.2: Das Workflow-Fragment *complexC2C.xml*

```
<bpel:scope name="Scope?randomNumber">
  <bpel:sequence>
    <bpel:assign validate="no" name="Init_Intermediate_Container">
      <bpel:copy>
        <bpel:from variable="?targetContainerName"/>
        <bpel:to variable="intermediateDirectory"/>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>
          <bpel:literal xml:space="preserve">?dataFormat</bpel:literal>
        </bpel:from>
        <bpel:to variable="intermediateDirectory">
          <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
            <![CDATA[dataFormat]]>
          </bpel:query>
        </bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from variable="?targetContainerName"/>
        <bpel:to variable="intermediateContainer"/>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>
          <![CDATA[string($?sourceContainerName/file/text())]]>
        </bpel:from>
        <bpel:to variable="intermediateContainer">
          <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
            <![CDATA[file]]>
          </bpel:query>
        </bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:extensionActivity>
      <simpl:containerToContainerPattern sourceContainer="?sourceContainer"
          targetContainer="[intermediateDirectory]" name="Transfer_Data"/>
    </bpel:extensionActivity>
```

```
  <bpel:extensionActivity>
    <simpl:dataFormatConversionPattern sourceContainer="[intermediateContainer]"
        targetContainer="?targetContainer" name="DataFormatConversionPattern"/>
  </bpel:extensionActivity>
</bpel:sequence>
<bpel:containerReferenceVariables>
  <bpel:containerReferenceVariable name="intermediateContainer"
      type="simpl:FileSystemDataContainerReferenceType">
    <bpel:from>
      <bpel:literal xml:space="preserve">
        <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
          <dataSourceReferenceVariable>?dataSourceReferenceVariable
          </dataSourceReferenceVariable>
          <dataFormat>?dataFormat</dataFormat>
        </simpl:container>
      </bpel:literal>
    </bpel:from>
  </bpel:containerReferenceVariable>
  <bpel:containerReferenceVariable name="intermediateDirectory"
      type="simpl:FileSystemDataContainerReferenceType">
    <bpel:from>
      <bpel:literal xml:space="preserve">
        <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
          <dataSourceReferenceVariable>?dataSourceReferenceVariable
          </dataSourceReferenceVariable>
          <dataFormat>?dataFormat</dataFormat>
        </simpl:container>
      </bpel:literal>
    </bpel:from>
  </bpel:containerReferenceVariable>
</bpel:containerReferenceVariables>
<bpel:variables>
  <bpel:variable name="intermediateContainer"
      type="simpl:FileSystemDataContainerReferenceType">
    <bpel:from>
      <bpel:literal xml:space="preserve">
        <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
          <dataSourceReferenceVariable>?dataSourceReferenceVariable
          </dataSourceReferenceVariable>
          <dataFormat>?dataFormat</dataFormat>
        </simpl:container>
      </bpel:literal>
    </bpel:from>
  </bpel:variable>
  <bpel:variable name="intermediateDirectory"
      type="simpl:FileSystemDataContainerReferenceType">
    <bpel:from>
      <bpel:literal xml:space="preserve">
        <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
          <dataSourceReferenceVariable>?dataSourceReferenceVariable
          </dataSourceReferenceVariable>
```

```
          <dataFormat>?dataFormat</dataFormat>
        </simpl:container>
      </bpel:literal>
    </bpel:from>
  </bpel:variable>
 </bpel:variables>
</bpel:scope>
```

Listing B.3: Das Workflow-Fragment *dataFormatConversion_script.xml*

```
<bpel:scope name="Scope?randomNumber">
 <bpel:sequence>
  <bpel:assign validate="no" name="Init_Intermediate_Container">
   <bpel:copy>
    <bpel:from variable="?sourceContainerName"/>
    <bpel:to variable="intermediateDirectory"/>
   </bpel:copy>
   <bpel:copy>
    <bpel:from>
     <![CDATA[string('')]]>
    </bpel:from>
    <bpel:to variable="intermediateDirectory">
     <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      <![CDATA[file]]>
     </bpel:query>
    </bpel:to>
   </bpel:copy>
  </bpel:assign>
  <bpel:extensionActivity>
   <simpl:issueCommandActivity name="Start_Conversion" dataResource="?dataResource"
       dmCommand="cd [intermediateDirectory] &amp;amp; ?script ?sourceContainer"/>
  </bpel:extensionActivity>
 </bpel:sequence>
 <bpel:containerReferenceVariables>
  <bpel:containerReferenceVariable name="intermediateDirectory"
      type="simpl:LocalDataContainerReferenceType">
   <bpel:from>
    <bpel:literal xml:space="preserve">
     <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
      <dataSourceReferenceVariable>?dataSourceReferenceVariable
      </dataSourceReferenceVariable>
      <dataFormat>?dataFormatDir</dataFormat>
     </simpl:container>
    </bpel:literal>
   </bpel:from>
  </bpel:containerReferenceVariable>
 </bpel:containerReferenceVariables>
 <bpel:variables>
  <bpel:variable name="intermediateDirectory" type="simpl:LocalDataContainerReferenceType">
   <bpel:from>
    <bpel:literal xml:space="preserve">
     <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.example.org/simpl simpl.xsd ">
      <dataSourceReferenceVariable>?dataSourceReferenceVariable
```

```
          </dataSourceReferenceVariable>
          <dataFormat>?dataFormatDir</dataFormat>
        </simpl:container>
      </bpel:literal>
    </bpel:from>
  </bpel:variable>
 </bpel:variables>
</bpel:scope>
```

Listing B.4: Das Workflow-Fragment *sequentialDIP.xml*

```
<bpel:scope name="OuterScope?randomNumber"
    xmlns:conRef="http://org.simpl.core.data.container.reference.list/">
  <bpel:sequence>
    <bpel:assign validate="no" name="Init_Local_ContainerReferenceList">
      <bpel:copy>
        <bpel:from>
          <bpel:literal>
            <dataSourceReference xmlns:simpl="http://www.example.org/simpl"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.example.org/simpl simpl.wsdl">
              <name/>
              <requirements/>
              <strategy/>
            </dataSourceReference>
          </bpel:literal>
        </bpel:from>
        <bpel:to variable="scopeDataSource"/>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>
          <bpel:literal xml:space="preserve">
            <conRef:datacopyContainerReferenceList>?listContent
            </conRef:datacopyContainerReferenceList>
          </bpel:literal>
        </bpel:from>
        <bpel:to variable="copyContainerReferenceList"/>
      </bpel:copy>
    </bpel:assign>
    <bpel:forEach parallel="no" counterName="?counterName" name="ForEach">
      <bpel:startCounterValue>
        <![CDATA[1]]>
      </bpel:startCounterValue>
      <bpel:finalCounterValue>
        <![CDATA[round(count($copyContainerReferenceList/ GenericContainerReference))]]>
      </bpel:finalCounterValue>
      <bpel:completionCondition/>
      <bpel:scope name="InnerScope?randomNumber">
        <bpel:sequence>
          <bpel:assign validate="no" name="Get_Path">
            <bpel:copy>
              <bpel:from>
                <![CDATA[$copyContainerReferenceList/GenericContainerReference
                    [number($?counterName)]/*]]>
              </bpel:from>
```

```
          <bpel:to variable="?currentContainerName"/>
        </bpel:copy>
        <bpel:copy>
          <bpel:from>
            <![CDATA[string($?currentContainerName/dataSourceReference/name)]]>
          </bpel:from>
          <bpel:to variable="scopeDataSource">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
              <![CDATA[name]]>
            </bpel:query>
          </bpel:to>
        </bpel:copy>
        <bpel:copy>
          <bpel:from>
            <![CDATA[string($?currentContainerName/dataSourceReference/requirements)]]>
          </bpel:from>
          <bpel:to variable="scopeDataSource">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
              <![CDATA[requirements]]>
            </bpel:query>
          </bpel:to>
        </bpel:copy>
        <bpel:copy>
          <bpel:from>
            <![CDATA[string($?currentContainerName/dataSourceReference/strategy)]]>
          </bpel:from>
          <bpel:to variable="scopeDataSource">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
              <![CDATA[strategy]]>
            </bpel:query>
          </bpel:to>
        </bpel:copy>
      </bpel:assign>?operation</bpel:sequence>
    </bpel:scope>
  </bpel:forEach>
</bpel:sequence>
<bpel:dataSourceReferenceVariables>
  <bpel:dataSourceReferenceVariable name="scopeDataSource"
      type="simpl:DataSourceReferenceType">
    <bpel:from>
      <bpel:literal xml:space="preserve">
        <dataSourceReference xmlns:simpl="http://www.example.org/simpl"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.example.org/simpl simpl.wsdl">
          <name/>
          <requirements/>
          <strategy/>
        </dataSourceReference>
      </bpel:literal>
    </bpel:from>
  </bpel:dataSourceReferenceVariable>
</bpel:dataSourceReferenceVariables>
<bpel:containerReferenceVariables>
  <bpel:containerReferenceVariable name="?currentContainerName"
      type="simpl:LocalDataContainerReferenceType">
```

```
      <bpel:from>
        <bpel:literal xml:space="preserve">
          <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              stringPattern="directory/file" xsi:schemaLocation="http://www.example.org/simpl
              simpl.xsd ">
            <dataSourceReferenceVariable>scopeDataSource</dataSourceReferenceVariable>
            <dataFormat>?dataFormat</dataFormat>
          </simpl:container>
        </bpel:literal>
      </bpel:from>
    </bpel:containerReferenceVariable>
  </bpel:containerReferenceVariables>
  <bpel:variables>
    <bpel:variable name="scopeDataSource" type="simpl:DataSourceReferenceType">
      <bpel:from>
        <bpel:literal xml:space="preserve">
          <dataSourceReference xmlns:simpl="http://www.example.org/simpl"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.example.org/simpl simpl.wsdl">
            <name/>
            <requirements/>
            <strategy/>
          </dataSourceReference>
        </bpel:literal>
      </bpel:from>
    </bpel:variable>
    <bpel:variable name="?currentContainerName" type="simpl:LocalDataContainerReferenceType">
      <bpel:from>
        <bpel:literal xml:space="preserve">
          <simpl:container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              stringPattern="directory/file" xsi:schemaLocation="http://www.example.org/simpl
              simpl.xsd ">
            <dataSourceReferenceVariable>scopeDataSource</dataSourceReferenceVariable>
            <dataFormat>?dataFormat</dataFormat>
          </simpl:container>
        </bpel:literal>
      </bpel:from>
    </bpel:variable>
    <bpel:variable name="copyContainerReferenceList"
        type="conRef:tDataContainerReferenceList"/>
  </bpel:variables>
</bpel:scope>
```

Listing B.5: Das Workflow-Fragment *mdt.xml*

```
<bpel:scope name="Scope?randomNumber"
    xmlns:conRef="http://org.simpl.core.data.container.reference.list/">
  <bpel:sequence>
    <bpel:assign validate="no" name="Init_ContainerReferenceList">
      <bpel:copy>
        <bpel:from>
          <bpel:literal xml:space="preserve">
            <conRef:dataContainerReferenceList
                xmlns:conRef="http://org.simpl.core.data.container.reference.list/"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
```

```
        </bpel:literal>
      </bpel:from>
      <bpel:to variable="containerReferenceList"/>
    </bpel:copy>
    <bpel:copy>
      <bpel:from>
        <bpel:literal xml:space="preserve">
          <conRef:dataContainerReferenceList
              xmlns:conRef="http://org.simpl.core.data.container.reference.list/">?listContent
          </conRef:dataContainerReferenceList>
        </bpel:literal>
      </bpel:from>
      <bpel:to variable="containerReferenceList"/>
    </bpel:copy>
  </bpel:assign>
  <bpel:extensionActivity>
    <simpl:dataIterationPattern containerReferenceList="containerReferenceList"
        currentContainer="currentContainer" name="Sequential_DataIterationPattern">
      <bpel:extensionActivity>
        <simpl:containerToContainerPattern sourceContainer="currentContainer"
            targetContainer="?targetContainer" name="ContainerToContainerPattern"/>
      </bpel:extensionActivity>
    </simpl:dataIterationPattern>
  </bpel:extensionActivity>
 </bpel:sequence>
 <bpel:variables>
  <bpel:variable name="containerReferenceList" type="conRef:tDataContainerReferenceList"/>
 </bpel:variables>
</bpel:scope>
```

Listing B.6: Das Workflow-Fragment *getTimeStep_variable.xml*

```
<bpel:assign validate="no" name="assign">
 <bpel:copy>
  <bpel:from variable="?sourceVariable"/>
  <bpel:to variable="?targetVariable"/>
 </bpel:copy>
</bpel:assign>
```

Listing B.7: Das Workflow-Fragment *getTimeStep_literal.xml*

```
<bpel:assign validate="no" name="assign">
 <bpel:copy>
  <bpel:from>
   <bpel:literal xml:space="preserve">?literal</bpel:literal>
  </bpel:from>
  <bpel:to variable="?targetVariable"/>
 </bpel:copy>
</bpel:assign>
```

Listing B.8: Das Workflow-Fragment *getTimeStep_first_last.xml*

```
<bpel:scope name="Scope?randomNumber">
 <bpel:sequence>
```

```
    <bpel:extensionActivity>
      <simpl:retrieveDataActivity name="getTimeStep" dataResource="?dataSource"
          targetVariable="timeStepData" dmCommand="?dmCommand"/>
    </bpel:extensionActivity>
    <bpel:assign validate="no" name="assign">
      <bpel:copy>
        <bpel:from>
          <![CDATA[$timeStepData/table/row[position()=1]/column[position()=1]/text()]]>
        </bpel:from>
        <bpel:to variable="?targetVariable"/>
      </bpel:copy>
    </bpel:assign>
  </bpel:sequence>
  <bpel:variables>
    <bpel:variable name="timeStepData" type="simpl:tRelationalDataFormat"/>
  </bpel:variables>
</bpel:scope>
```

Listing B.9: Das Workflow-Fragment *dataFormatConversion_importWS.xml*

```
<bpel:scope name="import?randomNumber"
     xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
     xmlns:ws="http://transformer.csv.to.table.simpl.org">
  <bpel:sequence>
    <bpel:assign validate="no" name="prepare_import">
      <bpel:copy>
        <bpel:from>
          <bpel:literal xml:space="preserve">
            <tns:importCSVFileToTable xmlns:tns="http://transformer.csv.to.table.simpl.org"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
              <dataSourceReference>
                <name>name</name>
                <strategy>strategy</strategy>
                <requirements>requirements</requirements>
              </dataSourceReference>
              <fileSystemDataContainerReference stringPattern="">
                <directory>directory</directory>
                <file>file</file>
                <dataSourceReferenceVariable>dataSourceReferenceVariable
                </dataSourceReferenceVariable>
                <dataFormat>dataFormat</dataFormat>
              </fileSystemDataContainerReference>
              <tableDataContainerReference stringPattern="">
                <schema>schema</schema>
                <table>table</table>
                <dataSourceReferenceVariable>dataSourceReferenceVariable
                </dataSourceReferenceVariable>
                <dataFormat>dataFormat</dataFormat>
              </tableDataContainerReference>
            </tns:importCSVFileToTable>
          </bpel:literal>
        </bpel:from>
        <bpel:to part="parameters" variable="importCSVFileToTableRequest"/>
      </bpel:copy>
      <bpel:copy>
```

```
      <bpel:from variable="?dataResource"/>
      <bpel:to part="parameters" variable="importCSVFileToTableRequest">
        <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
          <![CDATA[dataSourceReference]]>
        </bpel:query>
      </bpel:to>
    </bpel:copy>
    <bpel:copy>
      <bpel:from variable="?container"/>
      <bpel:to part="parameters" variable="importCSVFileToTableRequest">
        <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
          <![CDATA[fileSystemDataContainerReference]]>
        </bpel:query>
      </bpel:to>
    </bpel:copy>
    <bpel:copy>
      <bpel:from variable="?table"/>
      <bpel:to part="parameters" variable="importCSVFileToTableRequest">
        <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
          <![CDATA[tableDataContainerReference]]>
        </bpel:query>
      </bpel:to>
    </bpel:copy>
  </bpel:assign>
  <bpel:assign validate="no" name="initializePartnerLink">
    <bpel:copy>
      <bpel:from>
        <bpel:literal xml:space="preserve">
          <sref:service-ref xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref">
            <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
              <Address>?wsAddressImport</Address>
            </EndpointReference>
          </sref:service-ref>
        </bpel:literal>
      </bpel:from>
      <bpel:to variable="endpoint"/>
    </bpel:copy>
    <bpel:copy>
      <bpel:from variable="endpoint"/>
      <bpel:to partnerLink="TransformationWS_ID?wsIDPartnerLink"/>
    </bpel:copy>
  </bpel:assign>
  <bpel:invoke name="importCSVFileToTable"
      partnerLink="TransformationWS_ID?wsIDPartnerLink" operation="importCSVFileToTable"
      portType="ws:TableToCSVTransformer" inputVariable="importCSVFileToTableRequest"
      outputVariable="importCSVFileToTableResponse"/>
</bpel:sequence>
<bpel:variables>
  <bpel:variable name="importCSVFileToTableRequest" messageType="ws:importCSVFileToTable"/>
  <bpel:variable name="importCSVFileToTableResponse"
      messageType="ws:importCSVFileToTableResponse"/>
  <bpel:variable name="endpoint" element="sref:service-ref"/>
</bpel:variables>
<bpel:partnerLinks>
```

```
    <bpel:partnerLink name="TransformationWS_ID?wsIDPartnerLink"
        partnerLinkType="tns:TransformationWS_ID?wsIDPartnerLink"
        partnerRole="TransformationWS_ID?wsIDProvider"/>
  </bpel:partnerLinks>
</bpel:scope>
```

Literaturverzeichnis

[Ari12] S. Aristidou. *Abstraktionsunterstützung für die Definition des Datenmanagements in Simulationsworkflows*. Diplomarbeit Nr. 3235, Universtät Stuttgart, 2012. (Zitiert auf den Seiten 6, 12, 38, 59, 61, 63, 73, 78, 79, 80, 81, 84, 85, 87, 90, 91, 93, 97, 99 und 151)

[BBB+05] J. Beatty, H. Blohm, C. Boutard, S. Brodsky, M. Carey, J.-J. Dubray, R. Ellersick, M. Ho, A. Karmarkar, D. Kearns, R. Le Brettevillois, M. Nally, R. Preotiuc-Pietro, M. Rowley, S. Smith, H. Yan. Service Data Objects For Java Specification Version 2.01, 2005. URL http://www.oracle.com/technetwork/testcontent/sdo-specification-java-v2-131073.pdf. (Zitiert auf Seite 72)

[BBC+10] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon. XML Path Language (XPath) 2.0 (Second Edition), 2010. URL http://www.w3.org/TR/2010/REC-xpath20-20101214/. (Zitiert auf Seite 18)

[BCF+10] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon. XQuery 1.0: An XML Query Language (Second Edition), 2010. URL http://www.w3.org/TR/2010/REC-xquery-20101214/. (Zitiert auf Seite 18)

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard. Web Services Architecture, 2004. URL http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. (Zitiert auf Seite 22)

[BKNT10] C. Baun, M. Kunze, J. Nimis, S. Tai. *Cloud Computing: Web-basierte dynamische IT-Services*. Springer, 2010. (Zitiert auf Seite 22)

[BPSM+08] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008. URL http://www.w3.org/TR/2008/REC-xml-20081126/. (Zitiert auf Seite 15)

[CMRW07] R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2007. URL http://www.w3.org/TR/2007/REC-wsdl20-20070626/. (Zitiert auf den Seiten 8, 23 und 24)

[Dor11] R. Dormien. *Service-Bus-Erweiterungen um Pandas-basierte Simulationen in Workflows zu nutzen*. Diplomarbeit Nr. 3127, Universtät Stuttgart, 2011. (Zitiert auf den Seiten 6, 34, 61, 62, 109, 111 und 153)

[Foua] Apache Software Foundation. Apache ODE. URL http://ode.apache.org/. (Zitiert auf den Seiten 26 und 65)

[Foub] The Eclipse Foundation. BPEL Designer Project. URL http://www.eclipse.org/bpel/. (Zitiert auf Seite 65)

[FS] Pandas Coupled FEM Solver. Offical Website. URL http://www.get-pandas.com/. (Zitiert auf Seite 37)

[GHM+07] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2007. URL http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. (Zitiert auf den Seiten 8 und 22)

[Gro] PostgreSQL Global Development Group. PostgreSQL. URL http://www.postgresql.org/. (Zitiert auf Seite 67)

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. *Conventional Workflow Technology for Scientific Simulation*. in: Guide to E-Science, Springer, 2011. (Zitiert auf den Seiten 6, 29 und 33)

[Har96] S. Hartmann. *The World as a Process: Simulations in the Natural and Social Sciences*. in: Simulation and Modelling in the Social Sciences from the Philosophy of Scinece Point of View, Springer, 1996. (Zitiert auf den Seiten 34 und 35)

[IEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12.1990, 1990. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159342&isnumber=4148. (Zitiert auf Seite 34)

[JE07] D. Jordan, J. 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 den Seiten 8, 30, 31, 65 und 155)

[KE11] A. Kemper, A. Eikler. *Datenbanksysteme: Eine Einführung*. Oldenbourg Wissenschaftsverlag, 2011. (Zitiert auf den Seiten 7, 15, 18, 20, 39, 40, 41, 43 und 45)

[KME12] R. Krause, B. Markert, W. Ehlers. Remodelling Processes in Bones: A Biphasic Porous Media Model. *Applied Mathematics and Mechanics*, 12(1):79–80, 2012. (Zitiert auf den Seiten 6 und 36)

[LR99] F. Leymann, D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall International, 1999. (Zitiert auf den Seiten 6, 11, 24, 25, 27, 28 und 32)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. Spektrum Akademischer Verlag, 2010. (Zitiert auf den Seiten 6, 8, 20, 21, 22, 23, 26, 30 und 31)

[Mü10] C. M. Müller. *Development of an Integrated Database Architecture for a Runtime Environment for Simulation Workflows*. Diplomarbeit Nr. 2984, Universtät Stuttgart, 2010. (Zitiert auf den Seiten 6, 7, 37, 59, 60 und 148)

[Rei11] P. Reimann. *SimTech Milestone Report: Data Provisioning for Scientific Workflows*. Universität Stuttgart, 2011. (Zitiert auf den Seiten 6 und 55)

[RM11] P. Reimann, B. Mitschang. Data Provisioning for Scientific Workflows, Poster-Präsentation auf dem vierten SimTech Status Seminar, Bad Boll, Deutschland, 2011. (Zitiert auf den Seiten 6 und 56)

[RRS+11] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL - A Framework for Accessing External Data in Simulation Workflows. *Gesellschaft für Informatik (ed.): Datenbanksysteme für Business, Technologie und Web*, pp. 534–553, 2011. (Zitiert auf den Seiten 6, 11, 30, 33, 47, 48, 51, 52, 53, 54, 60 und 61)

[RSM11] P. Reimann, H. Schwarz, B. Mitschang. Design, Implementation, and Evaluation of a Tight Integration of Database and Workflow Engines. *A. H. F. Laender (ed.), M. M. Moro (ed.): Journal of Information and Data Management*, Vol. 2(3):353–368, 2011. (Zitiert auf den Seiten 6, 27 und 28)

[RSRM] P. Reimann, H. Schwarz, M. Reiter, B. Mitschang. *Data Provisioning Techniques for Simulation Workflows*. Unveröffentlichter Bericht des Instituts für Parallele und Verteilte Systeme, Universtät Stuttgart. (Zitiert auf den Seiten 6, 36, 60, 73, 74, 75, 76 und 77)

[Rut09] J. Rutschmann. *Generisches Web Service Interface um Simulationsanwendungen in BPEL-Prozesse einzubinden*. Diplomarbeit Nr. 2895, Universtät Stuttgart, 2009. (Zitiert auf Seite 37)

[Sch03] H. Schöning. *XML und Datenbanken: Konzepte und Systeme*. Carl Hanser Verlag, 2003. (Zitiert auf den Seiten 8, 15, 18, 41, 42, 43 und 45)

[Seb10] T. J. Sebestyen. *XML: Einstieg für Anspruchsvolle*. Addison-Wesley, 2010. (Zitiert auf den Seiten 15 und 18)

[SIM] SIMPL. SIMPL Framework (Source Code). URL http://code.google.com/p/simpl09/. (Zitiert auf den Seiten 8, 65, 71 und 72)

[Sta73] H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973. (Zitiert auf Seite 34)

[TDGS06] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2006. (Zitiert auf Seite 28)

[VOH+07] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, U. Yalçinalp. Web Services Policy 1.5 - Framework, 2007. URL http://www.w3.org/TR/2007/REC-ws-policy-20070904/. (Zitiert auf Seite 71)

[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. *Proc. of the 33rd International Conference on Very Large Data Bases (VLDB), Wien, Österreich*, pp. 615–626, 2007. (Zitiert auf den Seiten 78, 79 und 147)

[Wag11] F. B. D. Wagner. *Nutzung einer integrierten Datenbank zur effizienten Ausführung von Workflows*. Diplomarbeit Nr. 3038, Universität Stuttgart, 2011. (Zitiert auf den Seiten 27 und 38)

[ZTZ05] O. C. Zienkiewicz, R. Taylor, J. Zhu. *The Finite Element Method - Its Basis and Fundamentals*. Butterworth-Heinemann, 2005. (Zitiert auf Seite 38)

Alle URLs wurden zuletzt am 04.09.2012 geprüft.

Abkürzungsverzeichnis

API	Application Programming Interface
BPEL	Business Process Execution Language
BPEL-DM	Business Process Execution Language extension for Data Management
C2CP	Container-to-Container Pattern
CRUD	Create, Read, Update, Delete
CSV	Comma-Separated Values
DBS	Datenbanksystem
DBVS	Datenbankverwaltungssystem
DFCP	Data Format Conversion Pattern
DIP	Data Iteration Pattern
DM	Datenmanagement (von engl. data management)
DMP	Data Merge Pattern
DSP	Data Split Pattern
DTD	Document Type Definition
DTTP	Data Transfer and Transformation Pattern
ETL	Extract, Transform, Load
FEM	Finite Element Methode
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JDBC	Java Database Connectivity
MDTP	Multiple Data Transfer Pattern
MM	Molekularmechanik
OASIS	Organization for the Advancement of Structured Information Standards
ODE	Orchestration Director Engine
QM	Quantenmechanik
QM/MM	Quantenmechanik/Molekularmechanik
SIMPL	SimTech - Information Management, Processes and Languages
SOA	Serviceorientierte Architektur
SQL	Structured Query Language
SSH	Secure Shell
sWfMS	Scientific Workflow Management System
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium

WfMC	Workflow Management Coalition
WS	Web Service
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
XML	Extensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XSD	XML Schema

Erklärung

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

(Henrik Andreas Pietranek)