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

Studienarbeit Nr. 2449

Pattern-basierte Kopplung eines biomechanischen und eines systembiologischen Simulationsmodells

Victor Riempp

Studiengang:	Informatik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Dipl.-Inf. Peter Reimann
Beginn am:	19. Dezember 2013
Beendet am:	28. Juli 2014
CR-Nummer:	H.2.5, H.4.1, I.6.7

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Nachdem sich Workflows im Unternehmensumfeld durchgesetzt haben, finden sie auch immer mehr Verwendung im wissenschaftlichen Bereich. Dort werden sie vor allem genutzt, um Simulationen durchzuführen. Dabei können die Wissenschaftler auf die solide Software für Workflow-Systeme zurückgreifen, die über die Jahre für die Unternehmen entwickelt wurde [RS13].

Allerdings haben Simulationen im allgemeinen andere Anforderungen an ein Workflow Management System, als Unternehmen. Zum einen müssen Simulationen oft große Datenmengen verarbeiten und erzeugen oft auch große Datenmengen. Diese Stammen dabei meist aus unterschiedlichen Quellen und liegen in unterschiedlichen Formaten vor. Zum anderen müssen die Datenmengen meist aufwändig Bereitgestellt und Konvertiert werden. Um dies zu vereinheitlichen, wurde mit SIMPL (SimTech – Information Management, Processes, and Languages) eine Erweiterung der Scientific Workflow Management Systems vorgeschlagen, welche sich diesen Problemen widmen und den einheitlichen Zugriff auf verschiedene Datenquellen ermöglicht. Des weiteren werden von SIMPL auch Datenmanagementpatterns unterstützt, mit denen Wissenschaftler ohne große Kenntnisse im Datenmanagementbereich ihre Simulationen erzeugen können [RRS+11].

Die in [KSR+11] vorgeschlagene Simulation, die ein biomechanisches Simulationsmodell und ein systembiologisches Simulationsmodell miteinander verbindet, stellt ebenfalls hohe Anforderungen an ein Scientific Workflow Management System, was die Datenbereitstellung und Konvertierung betrifft, da das biomechanische Simulationsmodell für die Berechnungen Pandas[1] genutzt wird und für das systembiologische Simulationsmodell GNU Octave[2] verwendet wird. Beide Systeme erwarten jeweils unterschiedliche Daten in unterschiedlichen Formaten als Eingabe und müssen hin und her konvertiert werden.

Im Rahmen dieser Studienarbeit wurde zunächst der Kopplungsworkflow, der diese beiden Simulationsmodelle, welche getrennt als Workflow definiert wurden, unter Verwendung des SIMPL-Prototyp umgesetzt. Im zweiten Schritt wurde der SIMPL-Prototyp erweitert, sodass die Kopplung auf allen ebenen der Pattern-Hierarchie unterstützt wird und die nötigen Pattern und ihre Umformungsregeln vorhanden sind.

Die Studienarbeit ist wie folgt aufgebaut: Zunächst werden im Kapitel 2 die, zum besseren Verständnis der Arbeit nötigen, Grundlagen beschrieben. Das SIMPL-Rahmenwerk wird im 3. Kapitel genauer betrachtet und vorgestellt. Das 4. Kapitel stellt verschiedene Konzepte für den Kopplungsworkflow vor und beschreibt wie dieser schließlich umgesetzt wurde. Im 5. Kapitel wird anschließend die Erweiterung des SIMPL-Prototyps zur Unterstützung der Pattern erläutert. Zum Abschluss wird

[1] http://www.mechbau.uni-stuttgart.de/pandas/index.html
[2] http://www.gnu.org/software/octave/

im 6. Kapitel evaluiert, inwieweit, die Verwendung von Pattern in diesem Simulationsworkflow zur Vereinfachung der Modellierung beitragen. Im 7. Kapitel wird die Arbeit noch in einem kurzen Fazit zusammengefasst.

2 Grundlagen

2.1 Simulation

Im Allgemeinen versteht man unter einer Simulation nach [Har96] einen Prozess, der einen anderen Prozess imitiert.

Dabei basiert die Simulation auf einem abstrakten Modell dieses Prozesses, dem Simulationsmodell.

Dieses Modell muss den Prozess soweit vereinfachen, dass nur noch seine essentiellen Elemente vorhanden sind und dabei jedoch für den selben Input optimaler weise den selben Output erzeugen oder diesem sehr ähnelt.

Anhand des Simulationsmodells kann nun die Reaktion des Prozesses auf verschiedene Inputs überprüft werden. Diese Überprüfung kann, entweder durch physikalische Experimente (z.b. Crashtests bei Autos) oder durch Überlegungen und Berechnungen erfolgen. Wenn man für die Berechnungen Computer nutzt, dann spricht man auch von einer Computersimulation.

In [Har96] werden außerdem fünf Einsatzgebiete für Simulationen kategorisiert:

1. **Simulation als eine Technik:**
   Simulationen, mit denen Experimente simuliert werden können, die wir sonst nicht durch führen könnten, weil die Zeitskala zu klein ist (z.b. für nukleare Reaktionen) oder zu groß ist (z.b. die Entwicklung von Galaxien).

2. **Simulation als eine heuristisches Werkzeug:**
   Durch das Durchführen mehrerer Simulationen mit verschiedenen Parametern und das anschließende Vergleichen ihrer Ergebnisse, können unbekannte Zusammenhänge und einfachere Regeln für das Modell abgeleitet werden, was oft auch zu einfacheren Modellen führt.

3. **Simulation als Ersatz für ein Experiment:**
   Simulationen, mit denen Experimente simuliert werden können, die wir sonst nicht durchführen könnten, weil sie praktisch unmöglich sind (z.b. die Entstehung von Galaxien), weil sie theoretisch unmöglich sind (z.b. wie eine Welt wäre, in der die Ladung eines Elektrons eine andere wäre) oder ethisch unmöglich sind (z.b. die Auswirkungen einer Einkommensteuererhöhung).

4. **Simulation als ein Werkzeug für Experimentatoren:**
   Simulationen werden auch häufig benutzt, um neue Experimente zu generieren, die sinnvollsten Experimente und deren bester Aufbau auszuwählen und um anschließend ihre Ergebnisse auszuwerten.

5. **Simulation als eine pädagogisches Werkzeug:**
   Des weiteren können Simulationen benutzt werden, um ein besseres Verständnis für ein reales Problem zu gewinnen, indem man die Auswirkungen der Änderungen verschiedener Parameter auf eine Simulation betrachtet und so die Problematik zu Verdeutlichen.

2.2 Service-Orientierter Architektur

Bei Service-Orientierter Architektur (Service-Oriented Architecture) handelt es sich um einen Software-Architektur Stil, der von [Mel10] wie folgt definiert wurde:

„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 Wiederverwendung ermöglicht."

Als Dienste werden hierbei Programme und Softwarekomponenten bezeichnet, die von anderen Genutzt werden können.

Das zentrale Merkmale einer Service-Orientierten Architektur ist die lose Kopplung der Diensten, mit einer Bindung, die erst zur Laufzeit erfolgt. Dieses dynamische Einbinden findet mithilfe eines Verzeichnisdienstes statt, bei dem sich Dienste registriert haben müssen und bei dem vorhandene Dienste abgefragt werden können, um sie dann einbinden und benutzen zu können.

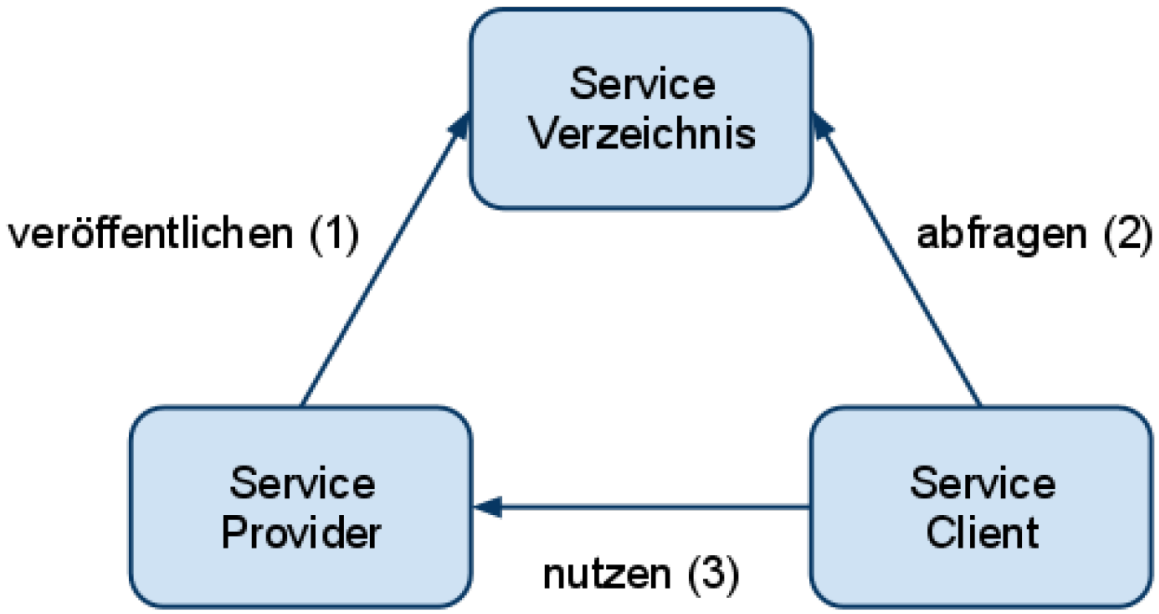

Abbildung 2.1: Das SOA-Dreieck (nach [Dor11])

Wie in Abbildung 2.1 zu sehen ist, gibt es im Bereich der Service-Orientierten Architektur drei verschiedene Rollen, deren Zusammenspiel als SOA-Dreieck bezeichnet wird.

- **Der Service Provider**, welcher Dienste zur Verfügung stellt
- **Das Service Verzeichnis**, in dem Dienste veröffentlicht und später vorhandene Dienste abgefragt werden können.
- **Der Service Client**, welcher sich passende Dienste aus dem Verzeichnis aussucht und anschließend nutzt.

Dabei ist die Verwendung von offenen Standards eine wichtige Voraussetzung dafür, eine breite Akzeptanz für Service-Orientierte Architektur zu ermöglichen, da es sonst unweigerlich zu Interoperabilitätsproblemen käme und damit auch keine lose Kopplung mehr möglich wäre. Diese Standards müssen zudem Einfachheit und Sicherheit bieten [Mel10].

2.3 Web Service

Web Services sind eine Implementierung der Service-Orientierten Architektur, deren Grundlagen die drei Standards SOAP, WSDL und UDDI bilden.

Vom World Wide Web Consortium (W3C) wird ein Web Service in [BHM+04] folgendermaßen definiert:

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

Die nachfolgenden Teilabschnitte basieren auf [BHM+04].

2.3.1 SOAP

SOAP ist ein Nachrichtenformat auf XML-Basis, welches auch zur Kommunikation zwischen Web Services genutzt wird. SOAP ist dabei unabhängig von einem Transportprotokoll. Ursprünglich stand SOAP für *Simple Object Access Protocol*, dieses Akronym wurde allerdings durch die Version 1.2 abgelegt, da es nicht mehr zu der Anwendung von SOAP im Web Service Bereich passt.

Eine SOAP-Nachricht besteht hierbei aus einem SOAP-Envelope, einem SOAP-Header und ein SOAP-Body enthalten muss. Auf dem Weg vom Sender zum Empfänger kann eine SOAP-Nachricht verschiedene Zwischenknoten passieren, welche neue Header einfügen, andere Header löschen und sogar die Nachricht selbst ändern können.

Was ein Knoten mit einer SOAP-Nachricht macht hängt vor allem von ihrem SOAP-Header ab. Dieser enthält beliebig viele Header-Blocks, welche an unterschiedliche Knoten-Arten gerichtet sein können und Transformationsanweisungen enthalten können. So werden spezielle Routing-Anweisungen als auch verschiedene Sicherheitsaspekte wie Verschlüsselung und Signieren auch über SOAP-Header

implementiert. Wenn ein Knoten ein Header-Block nicht verarbeiten kann, wird eine Fehlermeldung erzeugt.

Der SOAP-Body enthält die eigentliche Nachricht, die der Empfänger erhalten soll, und muss ein wohlgeformtes XML-Dokument sein.

2.3.2 WSDL

Die *Web Service Description Language* (WSDL) ist ein öffentlicher, XML basierter Standard des W3C, der zum Beschreiben der Schnittstellen von Web Services dient und in [CCMW01] genauer beschrieben wird. WSDL ist dabei weder von einer Plattform, noch von einem Protokoll oder einer Programmiersprache abhängig. Ein WSDL-Dokument lässt sich hierbei in mehre Bereiche aufspalten:

Types, Messages und PortTypes bilden den abstrakten Teil während Bindings und Services eine konkrete Implementierung repräsentieren.

Im abstrakten Teil wird hierbei definiert, was der Web Service anbietet, welche Nachrichten er erwartet und welche Variablen sie enthalten müssen. Im konkreten Teil wird beschrieben, wo man eine konkrete Implementierung des Web Service findet und welches Protokoll man benutzen muss, um mit ihm zu interagieren.

2.3.3 UDDI

Bei *Universal Description Discovery and Integration* (UDDI) handelt es sich um ein Standart für Verzeichnisdienste für Web Services. UDDI besteht dabei aus den White Pages, Yello Pages und Green Pages. In den White Pages sind die einzelnen Dienstanbieter aufgeführt. In den Yello Pages werden sie in verschiedene Kategorien einsortiert. In den Green Pages sind die einzelnen Web Services aufgeführt und beschrieben.

2.4 Workflow

Unter einem Geschäftsprozess (Business Process) wird im Allgemeinen eine Folge von Schritten, die ausgeführt werden müssen, um ein geschäftliches Ziel zu erreichen, verstanden. Das geschäftliche Ziel hierbei ist oft Gewinn oder der Ausbau der eigenen Marktposition, kann jedoch auch völlig anders lauten (z.b. bei Non-Profit-Organisationen) [OF06].

Ein Prozessmodell ist die Struktur eines Geschäftsprozesses und beschreibt die einzelnen Schritte, die in ihm enthalten sein können. Dabei können auch Verzweigungen vorhanden sein und auch Arbeitsschritte vom Menschen in der Realität können als Schritt angesehen werden. Eine Instanz dieses Modells ist ein konkreter (Geschäfts-) Prozess [LR99].

Ein Prozessmodell, welches nur Schritte enthält, die vom Computer ausgeführt werden, wird als Workflowmodell bezeichnet. Eine Instanz eines solchen Workflowmodells wird nun als Workflow bezeichnet (Siehe Abbildung 2.2).

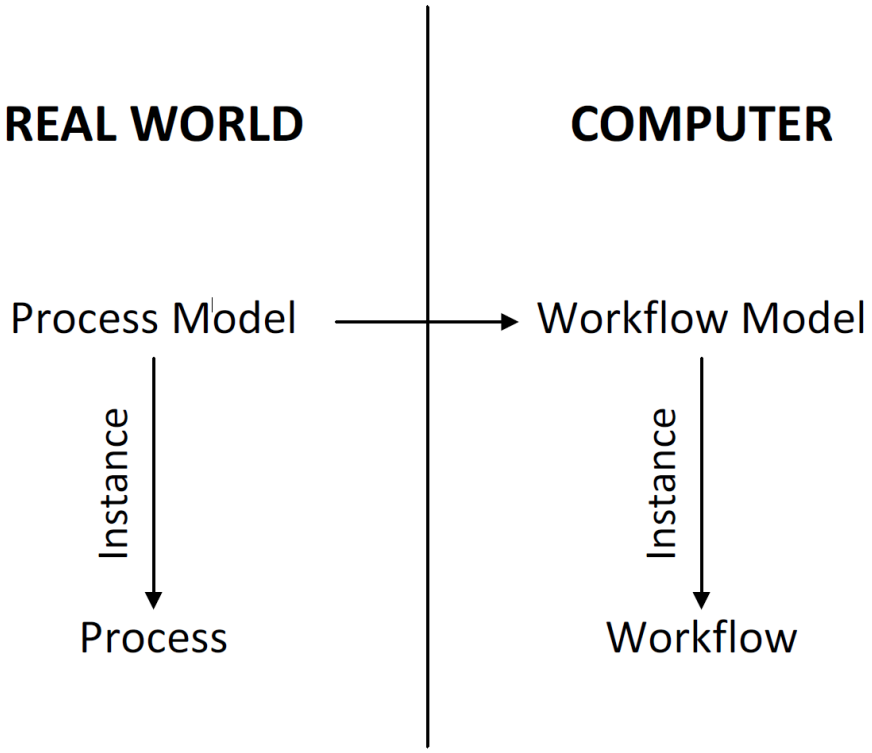

Abbildung 2.2: Zusammenhang zwischen Geschäftsprozess und Workflow [LR99]

2.5 BPEL

Da die Business Process Executiion Language (BPEL) ein Teil der WS-* Spezifikation ist, lautet ihr eigentlicher Name WS-Business Process Execution Language (WS-BPEL). Jedoch hat sich die kürzere Bezeichnung BPEL durchgesetzt. BPEL ist als industrieller Standard von OASIS definiert [Jor07] und hat sich weitgehend gegen konkurrierende Spezifikationen durchgesetzt [Pie12].

Listing 2.1 Die erste Ebene eines BPEL-Dokuments nach [Mel10]

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

BPEL ist eine auf XML basierende Sprache zur Beschreibung von Workflows, wobei alle Aktivitäten als Web Service implementiert werden müssen und auch der Workflow selbst als Web Service zur Verfügung gestellt wird. So lassen sich verschiedene Workflows ohne großen Aufwand einfach

kombinieren [Dor11]. In den folgenden Abschnitten beziehe ich mich, sofern nicht anders angegeben auf [Mel10].

Damit Web Services aufgerufen werden können, müssen diese zuerst definiert werden. Hierzu dienen die so genannten *partnerLinks* , diese werden wie in Listing 2.2 gezeigt definiert und bestehen aus den folgenden Elementen:

- **Name**, mit dem der partnerLink im gesamten Workflow referenziert werden kann.
- **partnerLinkType**, ein spezifischer Kommunikationstype, der im aufgerufenen Web Service so definiert sein muss.
- **myRole**, die Rolle mit der der Workflow mit dem Web Service kommuniziert.
- **partnerRole**, die Rolle mit der der Web Service mit dem Workflow kommuniziert.

Listing 2.2 Ein PartnerLink nach [Mel10]

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

Die, unter *variables* definierten, Variablen sind globale Variablen, auf die aus dem gesamten Workflow zugegriffen werden kann. Variablen können hierbei ein *WSDL message type*, ein *XML Schema simple type* oder ein *XML Schema element* sein.

Mit der Hilfe von Scopes (Geltungsbereiche) können auch lokale Variablen definiert werden, welche bei gleichem Namen die globalen Variablen verdecken.

In BPEL sind auch einige Basisaktivitäten und mehre strukturierte Aktivitäten definiert, welche es erlauben, den Ablauf des Workflows zu steuern.

Für ein genaueren Einblick in BPEL sei an dieser Stelle auf die Spezifikation [Jor07] verwiesen.

2.6 Scientific Workflow Management Systems

Ein Scientific Workflow Management Systems (sWfMSs) ist nach [RRS+11] ein Workflow Management System, das auf Workflow-Technologie aufbaut und speziell für die Erstellung und Ausführung von wissenschaftlichen Simulationen optimiert ist. Dieser Abschnitt basiert dabei auf [RRS+11], [GSK+11] und [Pie12].

Dabei ist zu beachten, dass der Fokus in einem solchen System auf das einfache erstellen und überwachen eines Workflows ausgerichtet ist. Das Ziel dabei ist es zu ermöglichen, dass auch Nutzern mit wenige Informatikkenntnissen schnell ein ausführbaren Workflow erzeugen können, während sich das Management System im Hintergrund um die vielen Implementierungsdetails kümmert.

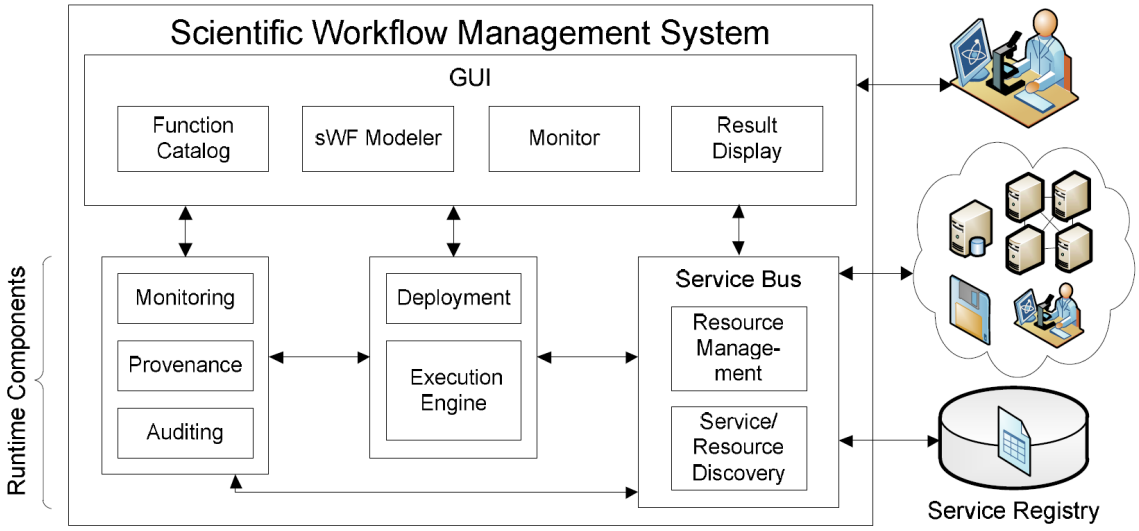

Abbildung 2.3: Architektur eines Scientific Workflow Management Systems, von [GSK+11]

Ein Scientific Workflow Management Systems kann in die GUI- und die Runtime-Komponenten aufgeteilt werden.

Zu den GUI-Komponenten zählt man:

- **Function Catalog** enthält eine Liste von vorhandenen Services und vorgefertigten Funktionen, die sich einfach zu einem ausführbaren Workflow kombinieren lassen.
- **sWF Modeler** hilft beim erstellen der Workflow-Spezifikationen und der Deployment-Informationen für den Workflow.
- **Monitor** ermöglicht das überwachen der Ausführung des Workflows.
- **Result Display** visualisiert die Ergebnisse des Workflows.

Zu den Runime Komonenten zählt man:

- **Deployment** überführt den Workflow in eine von der Execution Engine ausführbare Form.
- **Execution Engine** führt die übergebenen Workflows aus.
- **Auditing** speichert ab, was während der Ausführung eines Workflows passiert.
- **Monitoring** zeigt an, was die Auditing Komponente erfasst, um den Status eines Workflows schon währenddessen Ausführung an zu zeigen.
- **Provenance** speichert weitere Daten ab, die während der Ausführung eines Workflows anfallen, um die Wiederholbarkeit des Workflows sicherzustellen.
- **Service Bus - Ressource Management** dient zur Speicherung von Metadaten über externe Ressourcen und Dienste.

- **Service Bus - Discovery** ermöglicht das finden von passenden Diensten und Ressourcen über die vom Ressource Management bereitgestellten Metadaten.

3 SIMPL

SIMPL (SimTech – Information Management, Processes, and Languages) ist ein erweiterbares Rahmenwerk das auf Scientific Workflow Management Systems (Siehe Abschnitt 2.6) aufbaut und dieses um den vereinfachten und vereinheitlichten Zugriff auf Datenquellen erweitert. Soweit nicht anders genannt beziehe ich mich in diesem Kapitel auf [RRS+11].

3.1 Motivation

Das Benutzen von Workflow-Technologie mithilfe von Scientific Workflow Management Systemen für die Ausführung von Simulationen hat den Vorteil, dass man hierbei auf bewährter, standardisierter und verbreiteter Software aufbaut. Hierfür ist es natürlich erst einmal Nötig, die eigentlichen Komponenten und Rahmenwerke der Simulationen in Web Services zu Kapseln und für das das Scientific Workflow Management System bereit zu stellen. Dies kann jedoch von Mitarbeitern mit Informatikkenntnissen erledigt werden, während sich die Wissenschaftler um die Erstellung ihrer Simulation und deren Parameter kümmern können. Diese Trennung der Kompetenzen führt zur Entlastung der Wissenschaftler.

Oft kommt es jedoch vor, dass für eine Simulation viele Daten aus unterschiedlichen Datenquellen benötigt werden. Eine Datenquelle kann hierbei eine SQL-Datenbank, ein Dateisystem, eine einzelne Datei oder ein Web Service sein. Hierbei kann man nicht davon ausgehen, dass diese Daten ähnlich strukturiert sind oder auf die selbe weiße codiert sind. Hierfür muss man die Daten zuerst konvertieren.

An diesem Punkt setzt SIMPL an, in dem es den Zugriff auf die Datenquellen abstrahiert und vereinheitlicht. Außerdem bietet es die Möglichkeit Muster zu definieren und diese im Funktionskatalog zu speichern.

3.2 Architektur

SIMPL erweitert die Architektur eines Scientific Workflow Management Systems (Siehe Abschnitt 2.6) um ein einheitliches Datenmanagement. Um dies zu ermöglichen müssen verschiedene Komponente erweitert werden:

- **SIMPL Core** stellt einheitliche logische Schnittstellen bereit, um beliebige Datenquellen einzubinden und ist direkt in den Service Bus eingebunden.

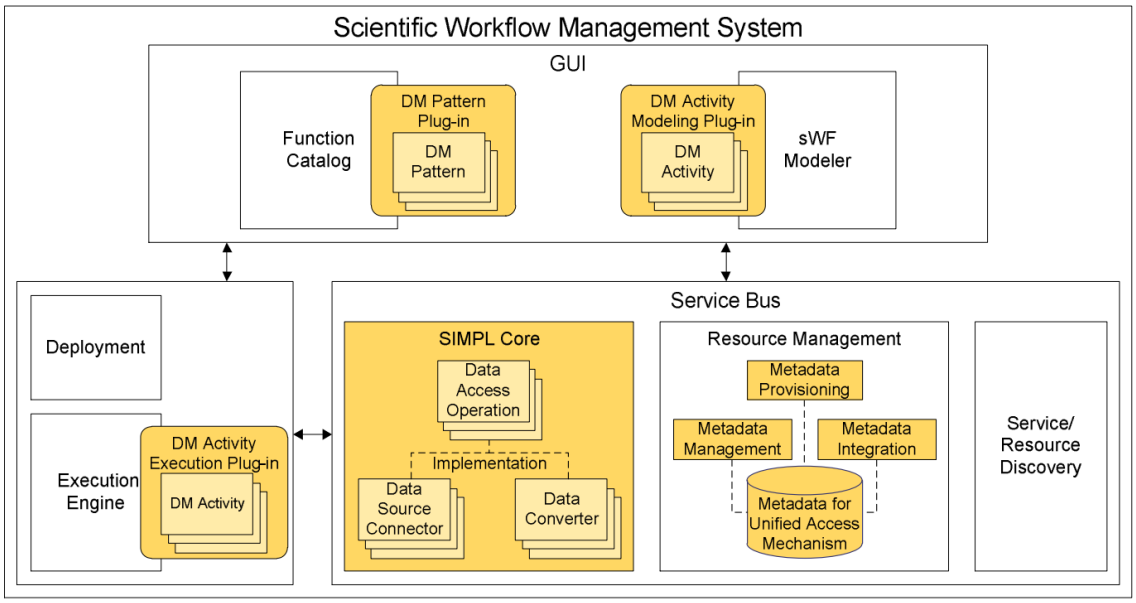

Abbildung 3.1: Erweiterung von sWfMSs durch SIMPL, nach [RRS+11]

- **Resource Management** wurde erweitert, um dort auch Metadaten speichern zu können, die nötig sind, um die logischen Schnittstellen mit ihren jeweiligen konkreten Zugriffsmethoden verbinden zu können.
- **Datenmanagement Activity Modeling Plug-in** stellt eine Erweiterung des sWF Modeller dar, die es ermöglicht alle Aktivitäten aus BPEL-DM zu nutzen.
- **Datenmanagement Activity Execution Plug-in** erweitert die Execution Engine, sodass die BPEL-DM Aktivitäten auch zur Laufzeit ausgeführt werden können.
- **Datenmanagement Pattern Plug-in** erweitert den Funktionskatalog, sodass vorgefertigte Patterns darin enthalten sind.

3.3 BPEL-DM

Da für SIMPL mehre neue BPEL Konstrukte nötig sind, wurde BPEL um zusätzliche Datenmanagementaktivitäten erweitert. Diese Erweiterungen werden unter BPEL-DM (Business Process Execution Language extension for Data Management) zusammen gefasst.

Jede DM-Aktivität benötigt als Eingabeparameter eine Variable des Typs *data source reference variable*. Diese Referenzvariable ist dabei ein *logical data source descriptor* verweist also entweder auf den (logischen) Name der Datenquelle, auf der die jeweilige Aktivität ausgeführt werden soll oder beschreibt die Anforderungen an eine beliebige Datenquelle, sodass diese zur Laufzeit dynamisch ausgewählt werden kann.

Die Daten einer Datenquelle werden in Container aufgeteilt, die jeweils mit einer *data container reference variable* referenziert werden können. Ein Container umfasst bei einer relationalen Datenbank zum Beispiel eine einzelne Tabelle.

Um Daten in eine Datenquelle speichern zu können, muss der Zielcontainer dieser Aktion als *data set variable* definiert sein. Dieser definiert, wie die Daten strukturiert sein müssen und wie diese abgespeichert werden müssen [Dor11].

BPEL-DM enthält drei grundlegenden Aktivitäten:

- **IssueCommand**: Dient dem Ausführen von Kommandos auf der Datenquelle und bietet dabei die Möglichkeit Daten zu ändern, zu erstellen oder zu löschen.
- **RetrieveData**: Dient dem Abfragen von Daten aus der Datenquelle.
- **WriteDataBack**: Dient dem Speichern von Daten in der Datenquelle.

3.4 Datenmanagementpatterns

Ausgehend von den 3 grundlegenden Aktivitäten, die in BPEL-DM definiert sind, können weitere Datenmanagementpatterns definiert werden, sodass diese anschließend bei der Modellierung verwendet werden können. Die Definierten Datenmanagementpatterns werden hierbei in ein ausführbares Workflowfragment überführt [Pie12].

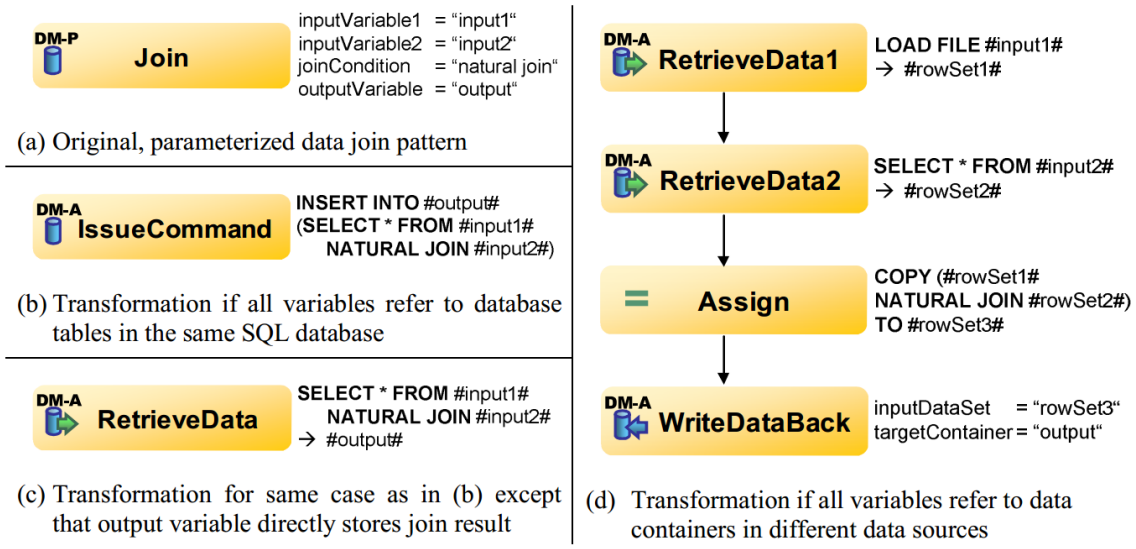

Abbildung 3.2: Beispielhafte Realisierung des Join Patterns [RRS+11]

Ausgehend von den Parametern, die in dem Datenmanagementpattern definiert wurden, können diese sehr unterschiedlich ausfallen, was in Abbildung 3.2 gut zu sehen ist, obwohl sie dasselbe Verhalten repräsentieren.

Da die Transformation der Datenmanagementpatterns von den verwendeten Datenquellen und anderer Parameter abhängt, kann dies unter Umständen auch erst beim Deployment geschehen.

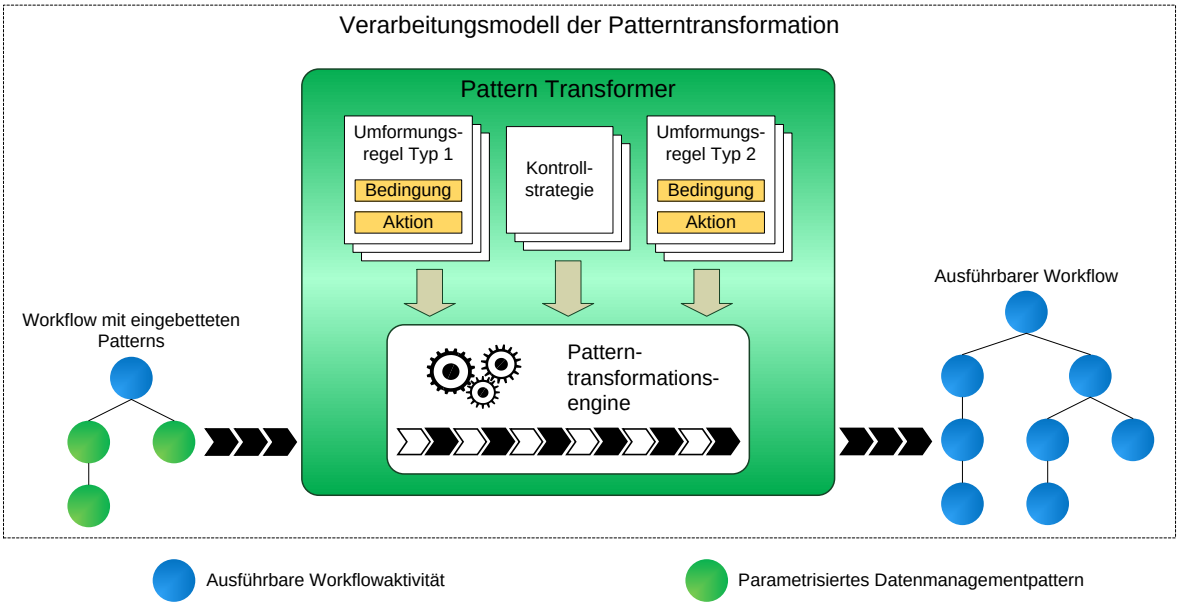

Abbildung 3.3: Regelbasierte Pattern Transformation [Rei11]

Die Transformation wird vom so genannten Pattern Transformer ausgeführt, welcher die Transformer Engine enthält. Des weiteren werden im Pattern Transformer die benötigten Umformungsregeln und Kontrollstrategien hinterlegt, die von der Patterntransformationsengine benutzt werden, um die Patterns in einen ausführbaren Workflow zu überführen [RSM14b]

3.5 Patternhierarchie

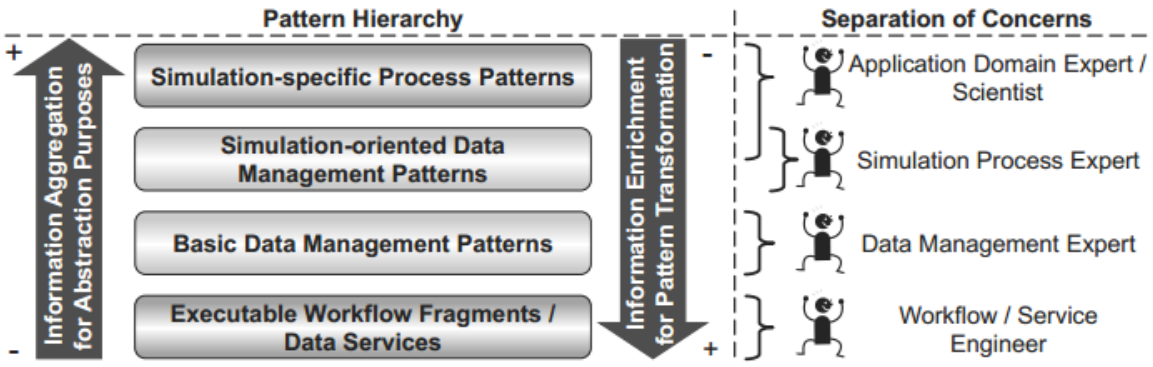

Abbildung 3.4: Pattern Hierarchie, aus [RSM14b]

Durch die Patterntransformation entsteht eine Patternhierarchie, da abstrakte Pattern in weniger abstrakte Pattern überführt werden, bevor sie diese im nächsten Schritt weiter transformiert, so lange, bis ein ausführbarer Workflow entstanden ist.

Die verschiedenen Ebenen in dieser Hierarchie ermöglichen hierbei eine einfache Trennung der Aufgabenbereiche. Damit wird erreicht, dass verschiedene Experten auf ihren jeweiligen Stufen der Hierarchie die Patterns und deren Transformationsregeln definieren ohne, dass sie sich in den anderen Bereichen auskennen müssen.

Das Ziel hierbei ist, dass die Wissenschaftler für ihre Simulationsworkflows nur auf der obersten Ebene arbeiten können und dabei unabhängig von den Implementierungsdetails sich um ihre Simulation kümmern können, während verschiedene Experten die benötigte Patternhierarchie mit ihren Transformationsschritten bereitstellen und so dafür sorgen, dass die abstrakt definierte Simulation ausgeführt werden kann (Siehe Abbildung 3.4).

Wie in Abbildung 3.5 zu sehen ist, können die Parameter der verschiedenen Ebenen zu bestimmten Parameter-Klassen zu geordnet werden und enthalten ein unterschiedliches Niveau der letztlich nötigen Implementierungsdetails [RSM14b].

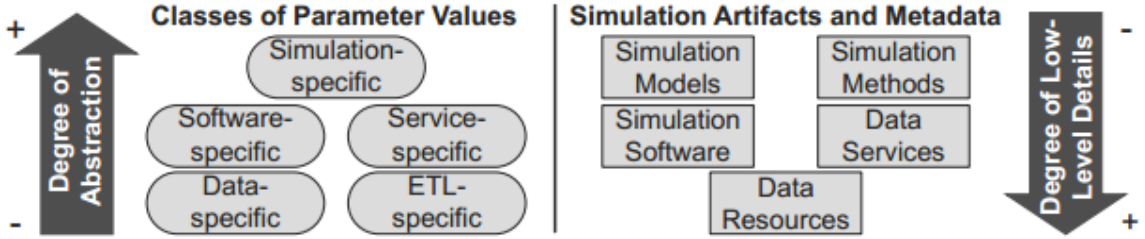

Abbildung 3.5: Abstraktion der Pattern-Parameter in der Pattern Hierarchie, aus [RSM14b]

3.6 Prototyp

SIMPL wurde prototypisch implementiert, als Basis wurde hierbei BPEL (Siehe Abschnitt 2.5) verwendet. Für die Benutzeroberfläche wurde Eclipse mit dem Plug-In Eclipse BPEL Designer[1] Version 0.8.0 verwendet und erweitert. Ausgeführt werden die Workflows mithilfe von Apache Orchestration Director Engine[2] (ODE)in Version 1.3.5. Zum Speichern der Metadaten der Simulationsartefakte wird die Datenbank PostgreSQL[3] Version 9.2 genutzt und für die regelbasierte Transformation der Datenmanagementpatterns wird die JRuleEngine[4] verwendet [RS14].

[1] http://www.eclipse.org/bpel/
[2] http://ode.apache.org/
[3] http://www.postgresql.org/
[4] http://jruleengine.sourceforge.net/

4 Kopplungsworkflow

Die Grundidee für die Kopplung einer biomechanischen und einer systembiologischen Simulation kommt von Robert Krause und wird in [KSR+11] beschrieben. Für die biomechanischen Simulation wird Pandas und für die systembiologischen Simulation wird Matlab benutzt. Die Daten müssen jeweils in anderen Formaten bereit gestellt werden.

In [KAK+13] wird die Verwendung eines Simulationsworkflows zur Realisierung der Simulation vorgeschlagen, weil es für Fachfremde leichter ist hier die benötigten Änderungen an den Simulationen vorzunehmen.

4.1 Alter Kopplungsworkflow

In [Dor11] wurde erstmals ein konkreter Simulationsworkflow für die oben beschriebene gekoppelte Simulation vorgestellt. Dieses wurde in [RS13] verfeinert und wird in dieser Sektion vorgestellt:

Der Simulationsworkflow, wie in [RS13] beschrieben (Abbildung 4.1), ist in drei Workflows aufgeteilt:

- Biomechanischer Workflow
- Kopplungsworkflow
- Systembiologischer Workflow

Zuerst wird der biomechanische Workflow ausgeführt. Dieser führt die Berechnungen in Pandas durch und ruft den Kopplungsworkflow auf. Anschließend wartet der Biomechanischeworkflow bis dieser Abgeschlossen ist und führt die abschließende Nachbearbeitung durch.

Der Kopplungsworkflow konvertiert nun, die vom biomechanischen Workflow erzeugten Daten in das vom systembiologischen Workflow benötigte Format, bestimmt die Anzahl der zur Verfügung stehenden Rechner, teilt die Daten entsprechend auf und startet für jede Instanz den systembiologischen Workflow. Anschließend wartet der Kopplungsworkflow bis alle systembiologische Workflows ihre Berechnungen abgeschlossen haben und stellt ihre Ergebnisse dem biomechanischen Workflow zur Verfügung.

Der systembiologische Workflow führt die Berechnungen mit Octave aus und gibt die Daten zurück an den Kopplungsworkflow.

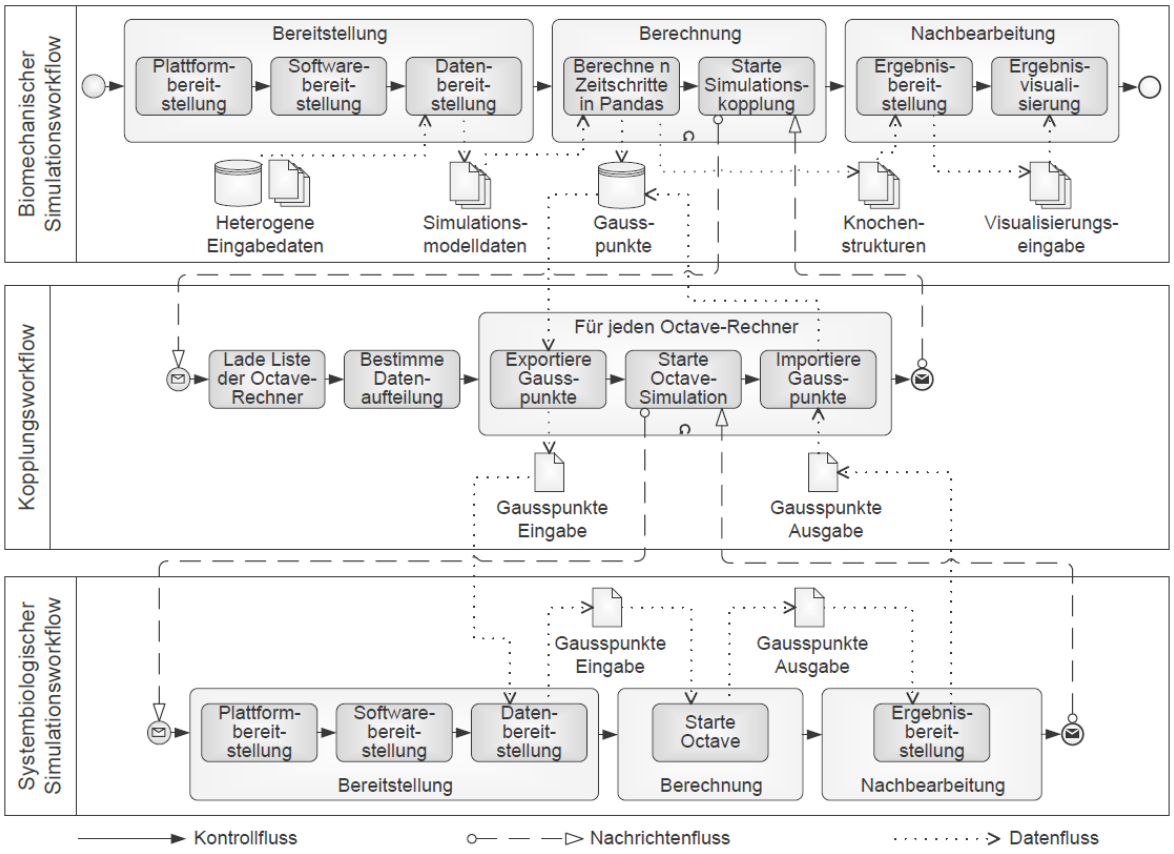

Abbildung 4.1: Ursprünglicher Kopplungsworkflo [RS13]

4.2 Neue Version des Kopplungsworkflow

In [RSM14a] wurde ein anderer Simulationsworkflow vorgeschlagen. Dieses wurde im Rahmen dieser Arbeit ausgearbeitet sowie implementiert und wird im folgenden genauer beschrieben.

Auch dieser Simulationsworkflow ist in drei Workflows aufgeteilt:

- Biomechanischer Workflow
- Kopplungsworkflow
- Systembiologischer Workflow

Allerdings wird hier der Kopplungsworkflow zuerst ausgeführt und ruft, sowohl den biomechanischen, als auch den systembiologischen Workflow auf. Dies erlaubt die Veränderung eines der beiden Workflows, ohne das der Kopplungsworkflow angepasst werden muss, sofern sich die Eingabeparameter und die Ausgabeparameter nicht ändern. Zudem lässt sich so auch der biomechanische Workflow eigenständig ohne Änderungen nutzen, sofern das gewünscht ist.

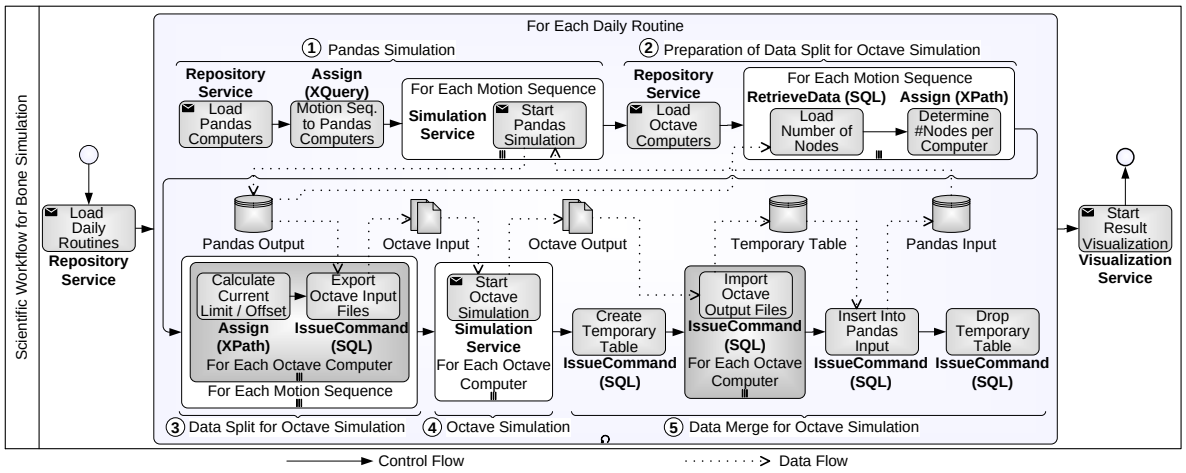

Abbildung 4.2: Neue Version des Kopplungsworkflow [RSM14a]

Der Aufbau dieses Kopplungsworkflow ist in Abbildung 4.2 zu sehen und wird im folgenden genauer beschrieben. Zunächst besteht der Kopplungsworkflow aus den folgenden drei Aktivitäten:

- **LoadDailyRoutines**

  Diese Aktivität läd eine Liste von Daily Routines von einem angegebenen Web Service. Als Daily Routine wird nach [KAK+13] hierbei das Bewegungsverhalten bezeichnet, welchem der Knochen dem Tag über ausgesetzt ist. Zur Vereinfachung der Berechnung werden die Daily Routines durch mehre Motion Sequences repräsentiert, die jeweils einen gewissen Bewegungsablauf repräsentieren, zum Beispiel schlafen, gehen, arbeiten. Für diese Motion Sequences sind jeweils Werte gespeichert, welche dann in der Berechnung benutzt werden können ohne direkt Übergeben werden zu müssen.

- **ForEachDailyRoutine**

  Diese For-Schleife iteriert nun über die zuvor geladenen Daily Routines und führt für jede von ihnen die beiden Simulationen aus. Hierzu sind eine Vielzahl von Aktivitäten in der For-Schleife gekapselt.

- **Start Result Visualization**

  Diese Aktivität startet nun einen anderen Workflow oder Web Service, der die erzeugten Ergebnisse aufbereitet und für den Nutzer geeignet darstellt.

Die Aktivitäten in der ForEachDailyRoutine-Schleife lassen sich in 5 Gruppen einteilen:

- **Pandas Simulation**

  In dieser Gruppe wird der biomechanische Workflow für die aktuelle Daily Routine gestartet. Hierzu wird zuerst eine Liste von verfügbaren Pandas-Computern abgerufen, welche für die Pandas-Simulation genutzt werden können. Danach werden, die in der Daily Routine enthaltenen, Motion Sequences auf die verschiedenen verfügbaren Pandas-Computer aufgeteilt und anschließend auf jedem dieser Pandas-Computer der biomechanische Workflow gestartet.

- **Preparation of Data Split for Octave Simulation**

  In dieser Gruppe werden zuerst die verfügbaren Oktave-Conputer abgerufen, welche für die Oktave-Simulation genutzt werden können und anschließend die Anzahl, der vom biomechanischen Workflow für eine spezielle Motion Sequence berechneten Nodes, bestimmt und zuletzt noch die Anzahl der Nodes pro Oktave-Computer berechnet.

- **Data Split for Octave Simulation**

  In dieser Gruppe werden die vom biomechanischen Workflow erzeugten Daten auf die verschiedenen Oktave-Computer aufgeteilt.

- **Octave Simulation**

  In dieser Gruppe wird nun für jeden dieser Oktave-Computer der systembiologische Workflow gestartet.

- **Data Merge for Octave Simulation**

  In dieser Gruppe werden, die von den verschiedenen Instanzen des systembiologischen Workflow erzeugten Daten, importiert und anschließend mit den Eingabedaten für Pandas vereinigt, sodass Pandas die Ergebnisse der systembiologischen Simulation im nächsten Iterationsschritt benutzen können.

4.3 Umgesetzter Kopplungsworkflow

Die neue Version des Kopplungsworkflows, wie in Abschnitt 4.2 beschrieben, wurde von mir im SIMPL-Prototyp (siehe Abschnitt 3.6) detailliert und prototypisch implementiert und wird im folgenden genauer beschrieben.

Da verschiedene Elemente, welche im Kopplungsworkflow benutzt werden sollten, aber nicht direkt für die Datenbereitstellungen oder Datenintegration relevant sind und noch nicht vorhanden waren, mussten diese prototypisch ersetzt werden. Dies betraf vor allem die Repository Services für DailyRoutines, PandasComputers und OctaveComputers, welche durch einfache Assing-Aktivitäten simuliert werden. Diese geben einen festen Wert für die entsprechenden Variablen vor. Sobald die entsprechenden Repository Services vorhanden sind, genügt es die Assing-Aktivitäten mit einer ensprechenden Invoke-Aktivität zu ersetzen.

Des weiteren fehlen der biomechanische, der systembiologische und der visualisierungs Workflow. An der Stelle der ersten beiden werden Web Services aufgerufen, welche die Ergebnisdaten von der jeweiligen Simulation als Zufallswerte generieren, während für den letzten nur ein Platzhalter definiert wurde.

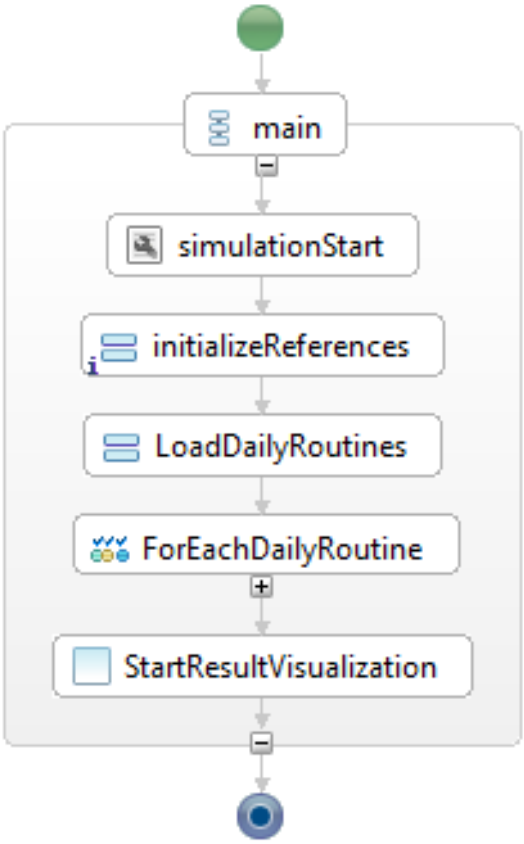

Abbildung 4.3: Oberste Ebene des Kopplungsworkflow

4.3.1 Oberste Ebene

Die oberste Ebene (Abbildung 4.3) des umgesetzten Kopplungsworkflow weißt zwei Aktivitäten mehr auf, als der modellierte Kopplungsworkflow. Zum einen die *simulationStart*-Aktivität, welche in jedem SIMPL-Workflow enthalten sein muss und zum anderen die *initializeReferences*-Aktivität, die nötig ist um die Variablen, die für die SIMPL-Datenmanagementaktivitäten gebraucht werden zu initialisieren.

Die Variable *PandasDBRef* vom Datentyp *simpl:DataSourceReferenceType* muss die nötigen Metadaten enthalten, um eine Datenquelle eindeutig zu identifizieren oder um eine beliebige Datenquelle anhand ihrer Spezifikationen auswählen zu können. Jede Datenmanagementaktivität benötigt eine solche Variable als Parameter.

Die Variable *NumberOfNodes* vom Datentyp *simpl:tRelationalDataFormat* kann als Ziel für *RetrieveData*-Aktivitäten dienen, welche das Ergebnis ihrer Abfrage in ihnen speichert. Obwohl diese Variable, bei der Verwendung als Ziel einer Abfrage überschrieben wird, muss sie vorher initialisiert worden sein.

Die Aktivität *ForEachDailyRoutine*, die noch weitere Aktivitäten kapselt, wird im folgenden Abschnitt 4.3.2 genauer betrachtet.

Die Aktivität *StartResultVisualization* ist nur als Platzhalter eingefügt, da zum jetzigen Zeitpunkt noch kein entsprechender Workflow vorhanden ist.

Listing 4.1 Struktur der Daily Routines

```
<complexType name="dailyroutines">
      <sequence>
            <element name="routine" type="dr:routine" maxOccurs="unbounded"
                  minOccurs="1"></element>
      </sequence>
</complexType>

<complexType name="routine">
      <sequence>
            <element name="motion_sequence" type="dr:motion_sequence" maxOccurs="unbounded"
                minOccurs="1"></element>
      </sequence>
</complexType>

<simpleType name="motion_sequence">
      <restriction base="string"></restriction>
</simpleType>
```

Die Assign-Aktivität *LoadDailyRoutines* simuliert einen RepositoryService, welcher eine Liste von Daily Routines, mit ihren jeweiligen Motion Sequences, zurückliefert. Dazu wird die Variable *daily_routines* des von mir entworfenen Datentyp *dailyroutines* (siehe Listing 4.1) mit entsprechenden Werten initialisiert (z.b. Listing 4.2)

Listing 4.2 Beispiel für den Inhalt der Variable daily_routines

```
<dr:dailyroutines xmlns:dr="http://de.ustutt.simtech/daily_routines">
      <dr:routine>
            <dr:motion_sequence>Sleeping</dr:motion_sequence>
            <dr:motion_sequence>Walking</dr:motion_sequence>
            <dr:motion_sequence>Working</dr:motion_sequence>
      </dr:routine>
      <dr:routine>
            <dr:motion_sequence>Sleeping</dr:motion_sequence>
            <dr:motion_sequence>Walking</dr:motion_sequence>
            <dr:motion_sequence>Working</dr:motion_sequence>
            <dr:motion_sequence>Working</dr:motion_sequence>
            <dr:motion_sequence>Walking</dr:motion_sequence>
            <dr:motion_sequence>Sleeping</dr:motion_sequence>
      </dr:routine>
</dr:dailyroutines>
```

4.3.2 ForEachDailyRoutine

In der ForEachDailyRoutine-Aktivität, die wie ihr Name schon verrät vom Typ ForEach ist und über die geladenen Daily Routines iteriert, werden die eigentlichen Berechnungen dieser kombinierten Simulation ausgeführt.

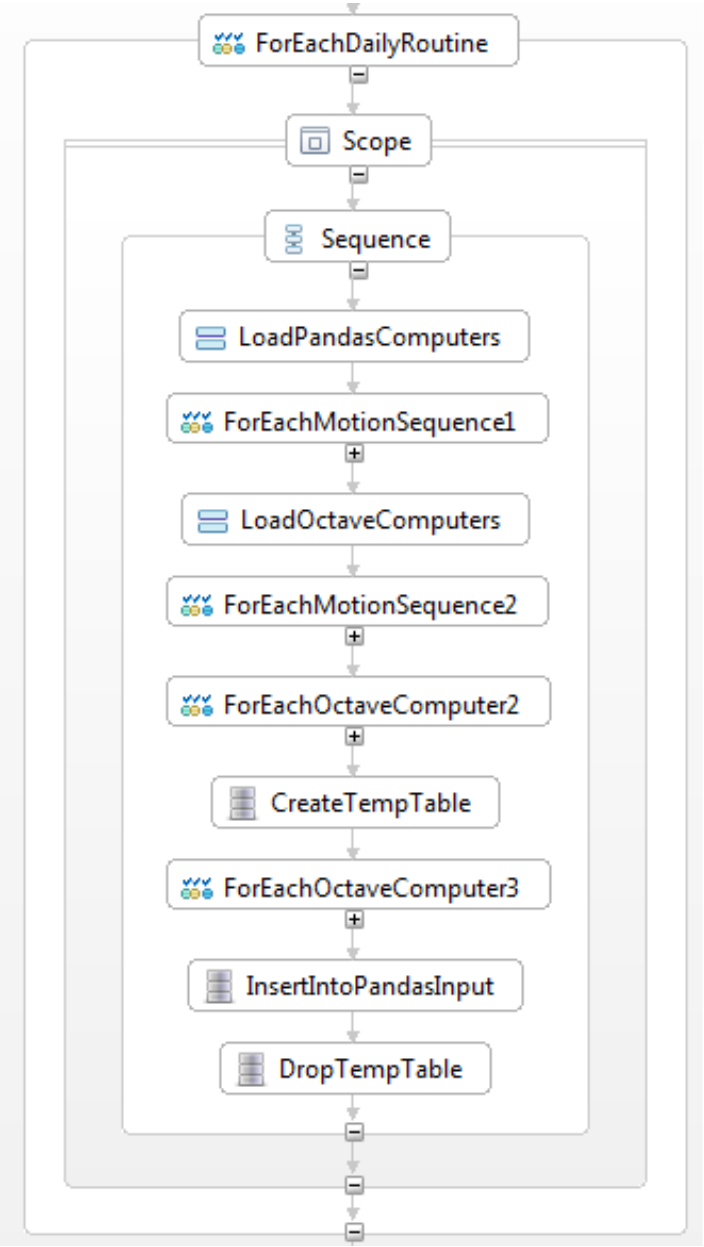

Abbildung 4.4: Übersicht über ForEachDailyRoutine

Wie schon in Abschnitt 4.2 beschrieben, können, die hier enthaltenen Aktivitäten, in mehre Gruppen eingeteilt werden (Siehe Abbildung 4.2). Allerdings ist es sinnvoller die Gruppen Preparation of

Data Split for Octave Simulation und Data Split for Octave Simulation zu vereinen, denn die in der ersten Gruppe berechneten Werte, werden jeweils in nur für einen Schleifendurchlauf in der zweiten Gruppe benötigt. Daher können diese auch direkt dort berechnet werden wodurch es möglich ist eine komplette ForEach-Schleife einzusparen.

PandasSimulation

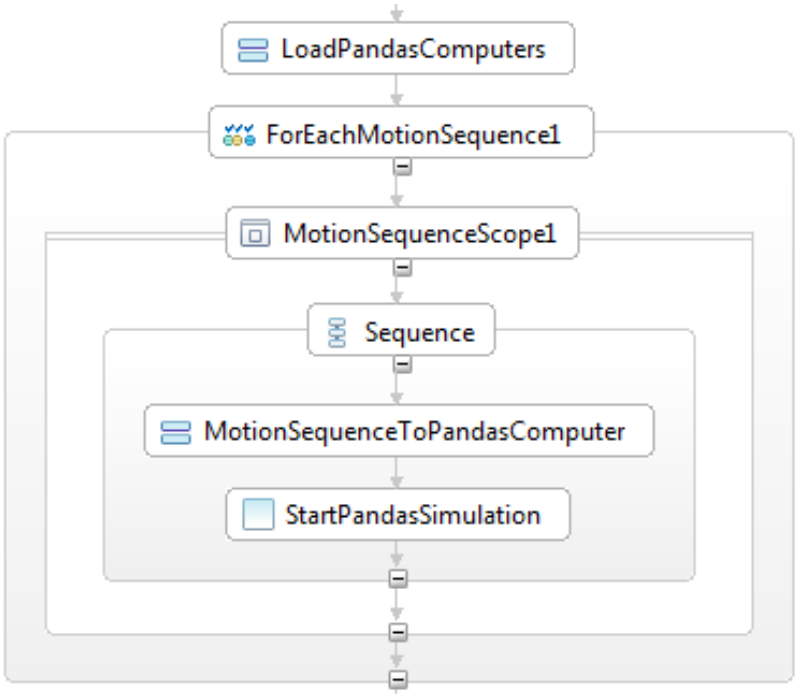

Abbildung 4.5: Übersicht über die Gruppe PandasSimulation

Die Gruppe PandasSimulation (Abbildung 4.5) bereitet die Pandas Simulationen vor, dh. sie verteilt die *Motion Sequences* auf die verfügbaren *Pandas Computer* und führt anschließend die Simulationen parallel aus.

Dazu enthält die Assign-Aktivität *LoadPandasComputers*, welche ein Repository Service simuliert, welcher die IP-Adressen aller verfügbaren Pandas-Instanzen zurückliefert. Hierzu wird die Variablen *pandas_computers* des von mir entworfenen Datentyps *ip-list* (siehe Listing 4.5) mit den entsprechenden Werten initialisiert (z.B. Listing 4.4).

Im Unterschied zum neuen Kopplungsworkflow-Modell wurde hier die Aktivität *MotionSequencesToPandasComputer* ihres Plurals beraubt und in die ForEach-Schleife integriert, da dort die einzelnen Motion Sequences leichter, nämlich mit einer einfachen Moduloberechnung, auf die Pandas Computers verteilt werden können (siehe Listing 4.3).

Die Aktivität *StartPandasSimulation* ist nur als Platzhalter eingefügt, an ihrer Stelle wird ein Web Service aufgerufen, welcher die Ergebnisdaten von Pandas als Zufallsdaten generiert.

Listing 4.3 Zuweisung einer Motion Sequence zu einem Pandas Computer

```
$pandas_computers/pc:ip[number($MotionSequence1Counter mod (count($pandas_computers/pc:ip)))]
```

Listing 4.4 Beispiel für den Inhalt der Variable pandas_computers

```
<pc:ip-list xmlns:pc="http://de.ustutt.simtech/pandas_computers">
      <pc:ip>192.168.1.40</pc:ip>
      <pc:ip>192.168.1.41</pc:ip>
</pc:ip-list>
```

Listing 4.5 Definition des Datentyps ip-list

```
<complexType name="ip-list">
      <sequence>
            <element name="ip" type="oc:ip" maxOccurs="unbounded"
                  minOccurs="1"></element>
      </sequence>
</complexType>

<simpleType name="ip">
      <restriction base="string"></restriction>
</simpleType>
```

Preparation for Octave Simulation

Die Gruppe Preparation for Octave Simulation (Abbildung 4.6) fasst die beiden Gruppen *Preparation of Data Split for Octave Simulation* und *Data Split for Octave Simulation* zusammen. Aufgabe der Aktivitäten dieser Gruppe ist es, alles Nötige Vorzubereiten, um die Octave Simulation starten zu können. Hierzu müssen die Ergebnisse der Pandas Simulation ausgelesen, auf die verfügbaren Pandas Computer aufgeteilt und anschließend noch in das von Octave benutzte CSV-Format umgewandelt werden.

Die *LoadOctaveComputers*-Aktivität simuliert einen Repository Service, welcher die IP-Adressen aller verfügbaren Octave-Instanzen zurückliefert. Hierzu wird die Variable *octave_computers* des von mir entworfenen Datentyps *ip-list* (siehe Listing 4.5) mit den entsprechenden Werten initialisiert (z.B. Listing 4.6).

Listing 4.6 Beispiel für den Inhalt der Variable octave_computers

```
<oc:ip-list xmlns:oc="http://de.ustutt.simtech/octave_computers">
      <oc:ip>192.168.1.42</oc:ip>
      <oc:ip>192.168.1.43</oc:ip>
</oc:ip-list>
```

Anschließend wird mit der *ForEachMotionSequence2*-Aktivität über alle Motion Sequences iteriert und mit der RetrieveData-Aktivität *LoadNumberOfNodes* die Anzahl der Nodes pro Element und die totale Anzahl der Nodes aus dem Ergebnis der Pandas-Simulation ausgelesen (Siehe Listing 4.7).

Listing 4.7 SQL-Abfrage zur Bestimmung der Anzahl an Nodes

```
SELECT COUNT(Distinct node) AS number_nodes_per_element, Count((element, node)) AS total_nodes
    FROM bonesimulation.pandasresults WHERE timestep = 1
```

In der *DetermineNumberOfNodes*-Aktivität werden die Ergebnisse der Abfrage in den Variablen *NumberOfNodesPerElement* und *NumberOfNodesTotal* vom Datentyp *integer* abgespeichert. Dies wird dadurch notwendig, dass die RetrieveData-Aktivität ein XML-Dokument als Ergebnis in der Variable NumberOfNodes abspeichert, im Nachfolgenden, werden die beiden Werte aber als Integer benötigt.

Nun wird mit der *ForEachOctaveComputer1*-Aktivität über alle verfügbaren Octave Computer iteriert. Es wird jeweils der Offset und das Limit, der vom jeweiligen Octave Computer zu verarbeitenden Nodes bestimmt und anschließend die Daten entsprechend aufgeteilt und ins CSV-Format umgewandelt und für Octave exportiert.

Der *Offset* ist zu Beginn jeweils 0 und bei jedem Iterationsschritt wird das vorherige Limit dazu addiert.

Die Berechnung des *Limits* ist in Listing 4.8 mit Hilfe von etwas Pseudocode beschrieben. Auf diese Weise werden die Nodes so gleichmäßig wie möglich auf die verschiedenen Octave-Instanzen verteilt.

Listing 4.8 Berechnung des Limits

```
if(($NumberOfNodesTotal - $Offset) mod count($octave_computers/oc:ip/text()) != 0)
      $NumberOfNodesPerElement div count($octave_computers/oc:ip/text())
else
      ($NumberOfNodesPerElement div count($octave_computers/oc:ip/text()))+1
Endif
```

Zuletzt erfolgt noch der eigentliche Export der Datan aus der Datenbank. Zuerst muss die Variable *FileName* den Name und Pfad der zu erzeugenden Datei als Wert zu gewiesen bekommen. Anschließend wird eine IssueCommand-Aktivität verwendet, die den in Listing 4.9 gezeigten SQL-Befehl ausführt, welcher die von Octave benötigten Daten zwischen Offset und Limit, in die durch FileName definierte Datei abspeichert.

Listing 4.9 Export der Octave Input Files

```
COPY (SELECT ((element-1)*20)+node AS id, AVG(tensile_stress) AS tensile_stress,
    AVG(growth_energy) growth_energy, AVG(solidity) AS solidity FROM bonesimulation.pandasresults
    GROUP BY element, node ORDER BY element, node LIMIT #Limit# OFFSET #Offset# ) TO \'
    #FileName# \' DELIMITER AS \' \' CSV HEADER
```

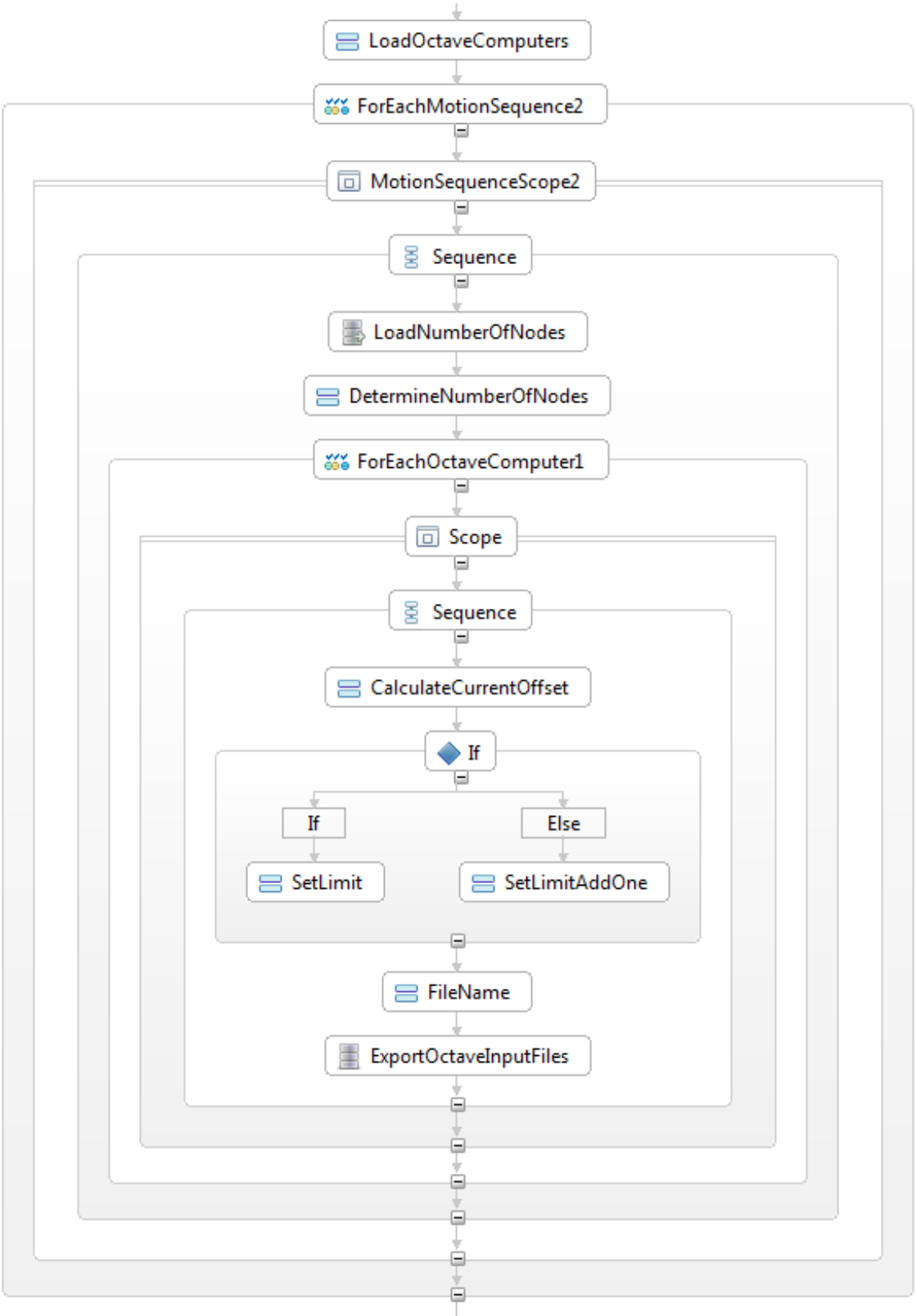

Abbildung 4.6: Übersicht über die Gruppe Preparation for Octave Simulation

Octave Simulation

Die Gruppe Octave Simulation (Abbildung 4.7) führt nun einfach die Octave-Simulation auf jedem verfügbaren Octave-Computer paralell aus.

Die Aktivität *StartOctaveSimulation* ist nur als Platzhalter eingefügt, da zum jetzigen Zeitpunkt noch kein entsprechender Workflow vorhanden ist.

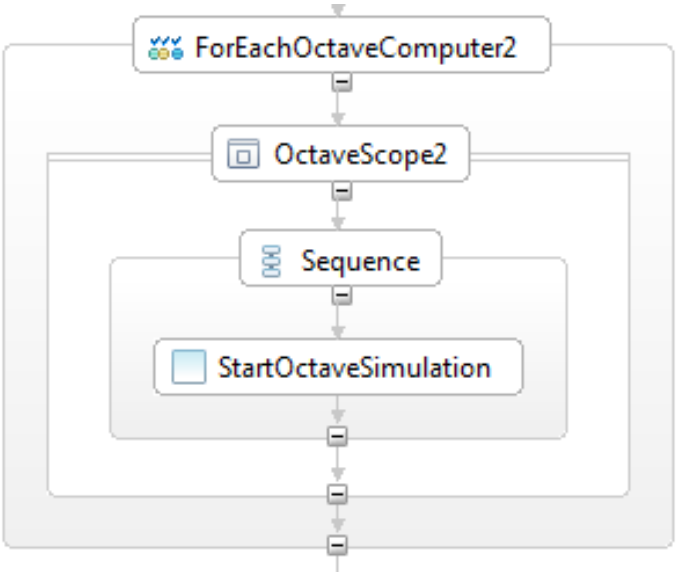

Abbildung 4.7: Übersicht über die Gruppe Octave Simulation

Data Merge for Octave Simulation

Die Gruppe Data Merge for Octave Simulation (Abbildung 4.8) importiert, die von den Octave-Instanzen erzeugten Output Files und integriert die dadurch gewonnenen Daten in die Pandas-Datenbank, sodass Pandas im nächsten Schritt mit den, durch Octave verfeinerten Daten, weiter rechnen kann.

Hierzu ist wird zunächst in der *CreateTempTable*-Aktivität eine neue Tabelle in der Datenbank angelegt (siehe Listing 4.10).

Listing 4.10 Erstellen der temporären Tabelle

```
CREATE TABLE bonesimulation.intermediateresults (id integer, growth_energy_production double
    precision, solidity double precision, CONSTRAINT pk_intermediateresults PRIMARY KEY (id))
    WITH (OIDS = FALSE);
     ALTER TABLE bonesimulation.intermediateresults OWNER TO postgres;
```

Als nächstes wird über alle Octave-Instanzen iteriert und jeweils die erzeugte Datei in die temporäre Tabelle eingelesen. Dazu muss zuerst die Variable FileName entsprechend gesetzt werden, anschlie-

ßend wird mit der *ImportOctaveOutputFiles*-Aktivität die benötigte Datenbankanweisung ausgeführt (siehe Listing 4.11).

Listing 4.11 Import des Octave Outputs

```
COPY bonesimulation.intermediateresults FROM \'#FileName#\' DELIMITER AS \' \' CSV HEADER
```

Anschließend müssen nun die Daten aus der temporären Tabelle umgeformt und in die Tabelle mit den Pandas-Zwischenergebnisse eingefügt werden. Dies wird durch die *InsertIntoPandasInput*-Anweisung durchgeführt (siehe Listing 4.12).

Listing 4.12 Transformation und Einfügen in Pandas

```
INSERT INTO bonesimulation.octaveresults (cycle, element, node, growth_energy_production,
    solidity)(SELECT 1, (((id -1) / #NodesPerComputer#) + 1), (((id-1) % #NodesPerComputer#) +1),
    growth_energy_production, solidity FROM bonesimulation.intermediateresults ORDER BY id)
```

Zu guter Letzt wird jetzt nur noch die temporäre Tabelle durch die *DropTempTable*-Aktivität gelöscht (siehe Listing 4.13);

Listing 4.13 Transformation und Einfügen in Pandas

```
DROP TABLE bonesimulation.intermediateresults;
```

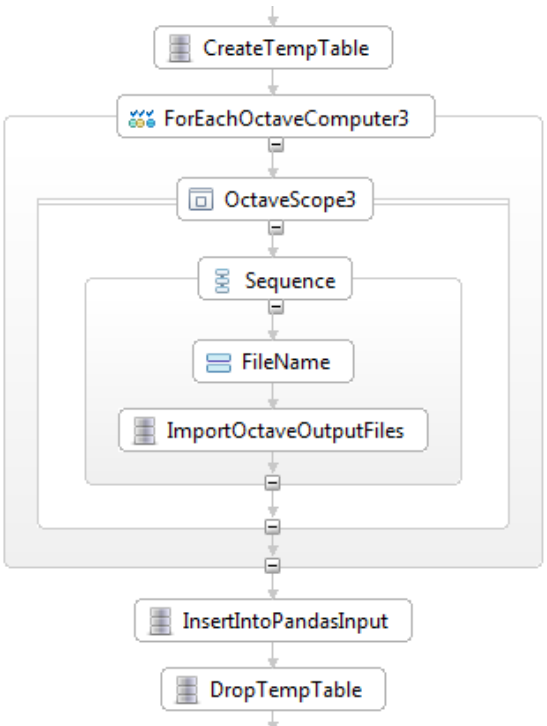

Abbildung 4.8: Übersicht über die Gruppe Data Merge for Octave Simulation

5 Patterns

In diesem Kapitel wird nun ausgehend, von dem im Kapitel 4 entworfenen und anschließend detailliert prototypisch implementierten, Kopplungsworkflow die Pattern-Hierarchie aufgebaut. Das Ziel hierbei ist, dass durch sinvolle Transformationsregeln und einem passend parametrisierten Workflow auf der obersten eben der Pattern-Hierarchie ein ausführbarer Workflow entsteht. Im ersten Abschnitt wird der konzeptionelle Zusammenhang der einzelnen Pattern beschrieben und im darauffolgenden die Implementierung der Patterns und ihre Transformationen genauer erläutert.

5.1 Kozeptioneller Entwurf

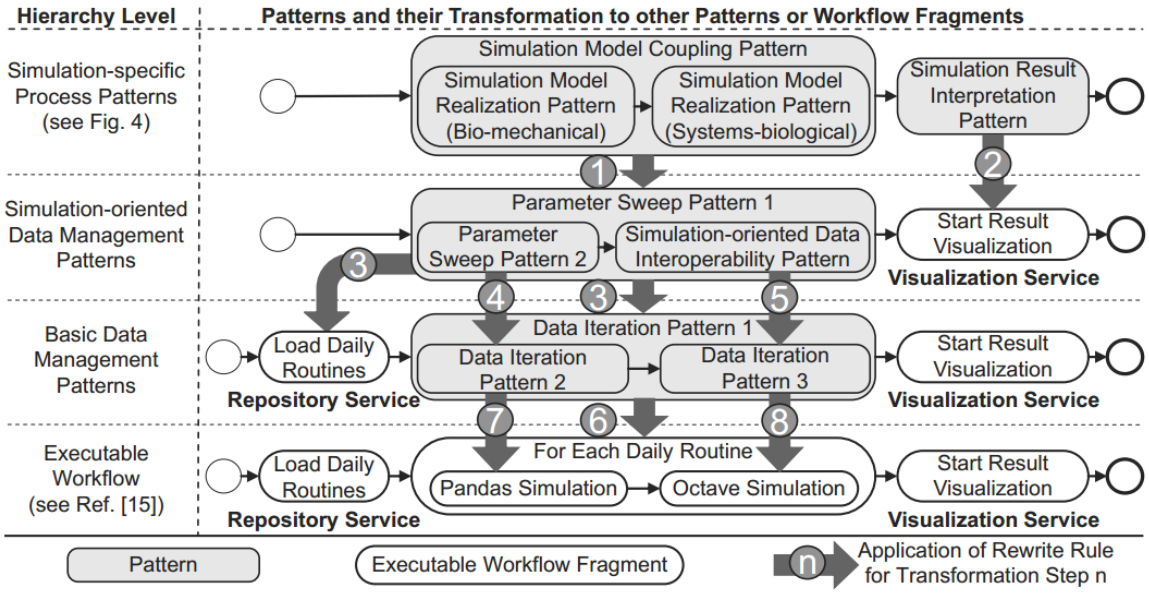

Abbildung 5.1: Einbettung des Kopplungsworkflow in die Pattern-Hierarchie aus [RSM14b]

In [RSM14b] wurde eine konkrete Pattern-Hierarchie für den Kopplungsworkflow und der grobe Zusammenhang der einzelnen Patterns beschrieben.

Wie in Abbildung 5.1 zu sehen ist, befindet sich auf der obersten Ebene der Pattern-Hierarchie die **Simulation-specific Process Patterns**. In diesem Fall sind sind es die Folgenden:

- **Simulation Model Coupling Pattern** legt das übergeordnete Simulationsmodell und die Verbindungsstrategie der einzelnen Simulationsmodelle fest.
- **Simulation Model Realization Pattern** geben ein konkretes Simulationsmodell an, das realisiert werden soll.
- **Simulation Result Interpretation Pattern** spezifiziert, wie die Ergebnisse visualisiert werden sollen.

Diese Patterns werden nun in zwei Schritten auf die nächste Ebene der Pattern-Hierarchie, den **Simulation-oriented Data Management Patterns**, transformiert.

Im ersten Schritt wird das *Simulation Model Coupling Pattern* auf ein *Parameter Sweep Pattern* abgebildet, das erste *Simulation Model Realization Pattern* auf ein darin enthaltenes *Parameter Sweep Pattern* und das zweite *Simulation Model Realization Pattern* auf ein *Simulation-oriented Data Interoperability Pattern* abgebildet.

Im zweiten Schritt wird das *Simulation Result Interpretation Pattern* direkt auf den Aufruf eines passenden Web Service abgebildet.

Dadurch sind nur noch drei Patterns von zwei verschiedenen Arten vorhanden:

- **Parameter Sweep Pattern** ist eine Abstraktion des *Data Iteration Pattern*, mit dem Fokus auf der einfachen sequentiellen oder parallelen Iteration über eine Parameterliste.
- **Simulation-oriented Data Interoperability Pattern** ist ebenfalls eine Abstraktion des *Data Iteration Pattern*, hier liegt der Fokus jedoch auf dem Verbinden der zwei Simulationsmodellen.

Die Patterns dieser Ebene werden nun in drei Schritten auf die Ebene der **Basic Data Management Patterns** transformiert.

Zuerst wird das äußere *Parameter Sweep Pattern* in ein *Data Iteration Pattern* transformiert. In der Abbildung 5.1 ist aus Gründen der Übersichtlichkeit nicht zu sehen, dass es sich hierbei um das **Sequential Data Iteration Pattern** handelt.

Anschließend wird das innere *Parameter Sweep Pattern* ebenfalls in ein *Data Iteration Pattern* umgewandelt, allerdings handelt es sich bei diesem um ein **Parallel Data Iteration Pattern**.

Zum Schluss wird nun noch dass *Simulation-oriented Data Interoperability Pattern* in ein weiteres *Parallel Data Iteration Pattern* umgewandelt.

Bei jedem diesen Schritte werden außerdem noch weitere ausführbare Aktivitäten erstellt, die der Übersichtlichkeit halber weg gelassen wurde. Diese werden im nächsten Abschnitt genauer betrachtet.

Die Patterns der *Basic Data Management Patterns* Ebene werden nun ebenfalls in drei Schritten in ausführbare **Workflow-Fragmente** umgewandelt. Alle *Data Iteration Pattern* hier stellt eine Abstraktion der **ForEach**-Aktivität dar, in die sie entsprechend transformiert werden, so dass der Workflow keine Pattern sondern nur noch ausführbare Aktivitäten enthält.

5.2 Implementierung

In diesem Abschnitt wird nun beschrieben in wie weit die Patterns und ihre Transformationen neu implementiert oder abgeändert werden mussten. Die Simulation-specific Process Patterns der obersten Ebene wurden hierbei ausgelassen, da sie nicht direkt mit der Datenbereitstellung oder der Datenintegration zusammen hängen.

5.2.1 Parameter Sweep Pattern

Das Parameter Sweep Pattern gehört zur Ebene der Simulation-oriented Data Management Patterns und ist eine Generalisierung des Data Iteration Patterns. Ziel des Parameter Sweep Patterns ist also über eine Liste von Parametern zu iterieren und in jedem Schritt die angegebene Operation auszuführen.

Dieses Pattern hat folgende Eingabeparameter für seine Anwendungsfälle im Kopplungsworkflow:

- **Parameter List** ist eine Liste von simulationsspezifischen Parametern, über die Iteriert wird.
- **Parallel** gibt an, ob die Liste parallel oder sequentiell iteriert werden soll.

Des weiteren enthält dieses Pattern genau eine Aktivität, die bei jedem Iterationsschritt ausgeführt wird. Diese Aktivität kann natürlich beliebig viele weitere Aktivitäten beinhalten.

Der Condition Part der Transformationsregel enthält folgende Bedingungen:

- **1. Bedingung:** Die Parameter Liste darf nicht leer sein.
- **2. Bedingung:** Das Pattern muss genau eine Aktivität enthalten.

Sind diese Bedingungen erfüllt wird der Action Part der Transformationsregel ausgeführt:

- **1. Aktion:** Laden des Workflow Fragments
- **2. Aktion:** Abhängig von der Art der ausgewählten Parameter List, wird entweder die Variable, über die Iteriert werden soll direkt mit Werten gefüllt, oder über ein Aufruf erzeugt, der die entsprechenden Werte zurück liefert.
- **3. Aktion:** Abhängig vom Zustand des Parameters Parallel ein SequentialDataIterationPattern oder durch ParallelDataIterationPattern erzeugt und mit der erzeugten Variable als Parameter ausgestattet.

5.2.2 Simulation-oriented Data Interoperability Pattern

Das Simulation-oriented Data Interoperability Pattern gehört zur Ebene der Simulation-oriented Data Management Patterns und ist eine Generalisierung des Data Iteration Pattern mit dem Fokus auf die parallele Iteration einer Simulation und der dafür vorher nötigen Datenaufteilung und der anschließenden Datenvereinigung.

Dieses Pattern hat folgende Eingabeparameter für seine Anwendungsfälle im Kopplungsworkflow:

- **FirstSimulationModel** enthält das erste Simulationsmodell, dessen Ergebnisdaten verwendet werden sollen.
- **SecondSimulationModel** enthält das zweite Simulationsmodell, welches ausgeführt werden soll.
- **DataSplitParameters** enthält eine Liste von Relationen, die Angibt welcher Parameter des ersten Simulationsmodells auf welchen des zweiten übertragen werden muss.
- **DataMergeParameters** enthält eine Liste von Relationen, die Angibt, welcher Parameter des zweiten Simulationsmodells, nach Ablauf der zweiten Simulation, auf welchen Parameter des ersten Simulationsmodells zurück übertragen werden muss.
- **PartitionMode** bestimmt wie die Parameter aufgeteilt werden müssen.

Der Condition Part der Transformationsregel enthält folgende Bedingungen:

- **1. Bedingung:** Das FirstSimulationModel und das SecondSimulationModel müssen jeweils ein gültiges Simulationsmodell sein.
- **2. Bedingung:** Die DataSplitParameters liste darf nicht leer sein und die ihr enthaltenen Parameter müssen mit einem Teil des Outputs des FirstSimulationModel übereinstimmen.

Sind diese Bedingungen erfüllt wird der Action Part der Transformationsregel ausgeführt:

- **1. Aktion:** Laden des Workflow Fragments
- **2. Aktion:** Bestimme anhand der Simulationsmodelle und der DataSplitParameters, wie diese Aufgeteilt werden müssen.
- **3. Aktion:** Bestimme anhand der Simulationsmodelle und der DataMergeParameters, wie diese anschließend Vereinigt werden müssen.

5.2.3 Sequential Data Iteration Pattern

Das Sequential Data Iteration Pattern gehört zur Ebene der Basic Data Management Patterns und ist dabei eine Generalisierung der ForEach-Aktivität mit dem Fokus auf der sequentiellen Iteration über eine Datenmenge.

Dieses Pattern hat folgende Eingabeparameter für seine Anwendungsfälle im Kopplungsworkflow:

- **DataSet** Eine Liste von Elementen, über die Iteriert werden soll.

Des weiteren enthält dieses Pattern genau eine Aktivität, die bei jedem Iterationsschritt ausgeführt wird. Diese Aktivität kann natürlich beliebig viele weitere Aktivitäten beinhalten.

Der Condition Part der Transformationsregel enthält folgende Bedingungen:

- **1. Bedingung:** DataSet muss eine nicht leere Liste sein.
- **2. Bedingung:** Das Pattern muss genau eine Aktivität enthalten.

Sind diese Bedingungen erfüllt wird der Action Part der Transformationsregel ausgeführt:

- **1. Aktion:** Laden des Workflow Fragments
- **2. Aktion:** Ersetzen des Platzhalters ?FinalCounterValue durch die Anzahl der Elemente im DataSet.

5.2.4 Parallel Data Iteration Pattern

Das Parallel Data Iteration Pattern gehört zur Ebene der Basic Data Management Patterns und ist dabei eine Generalisierung der ForEach-Aktivität mit dem Fokus auf der parallelen Iteration über eine Datenmenge.

Dieses Pattern hat folgende Eingabeparameter für seine Anwendungsfälle im Kopplungsworkflow:

- **DataSet** Eine Liste von Elementen, über die Iteriert werden soll.

Des weiteren enthält dieses Pattern genau eine Aktivität, die bei jedem Iterationsschritt ausgeführt wird. Diese Aktivität kann natürlich beliebig viele weitere Aktivitäten beinhalten.

Der Condition Part der Transformationsregel enthält folgende Bedingungen:

- **1. Bedingung:** DataSet muss eine nicht leere Liste sein.
- **2. Bedingung:** Das Pattern muss genau eine Aktivität enthalten.

Sind diese Bedingungen erfüllt wird der Action Part der Transformationsregel ausgeführt:

- **1. Aktion:** Laden des Workflow Fragments
- **2. Aktion:** Ersetzen des Platzhalters ?FinalCounterValue durch die Anzahl der Elemente im DataSet.

6 Bewertung

In diesem Kapitel werden die in Kapitel 5 beschriebenen und neu implementierten Patterns, in Bezug auf den Grad der Abstraktionsunterstützung auf den verschiedenen Ebenen der Pattern-Hierarchie und auf ihre generische Einsetzbarkeit bewertet.

6.0.5 Abstraktionsunterstützung

Das erstellen neuer Simulationen sollte mit Hilfe der eingeführten Pattern deutlich vereinfacht werden, in diesem Abschnitt wird anhand des in Kapitel 4 beschriebenen und von mir implementierten Kopplungsworkflow betrachtet, in wie weit dies der Fall ist.

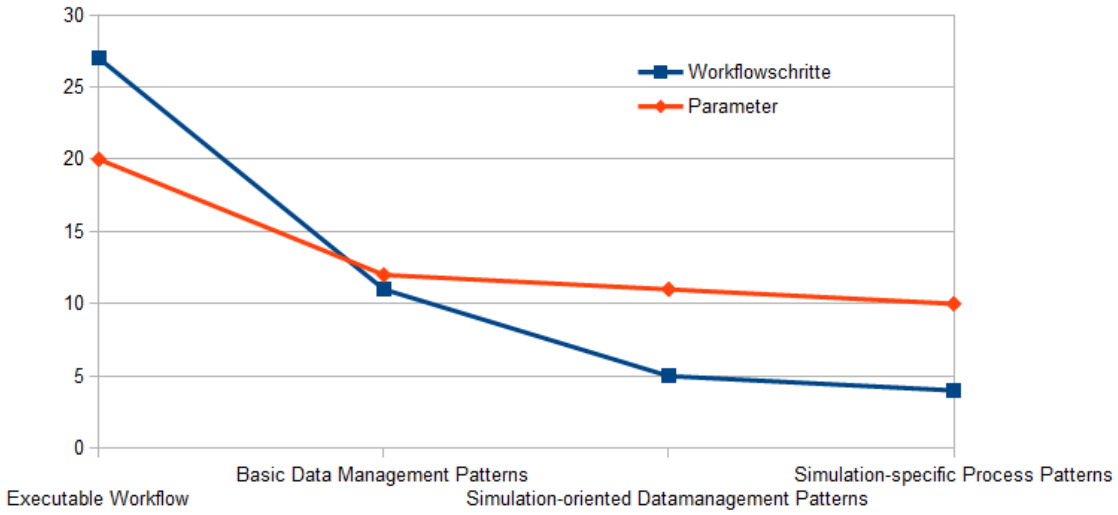

Abbildung 6.1: Anzahl der Workflow Schritte und Paramter

Wie in Abbildung 6.1 zu sehen ist, nimmt sowohl die Anzahl der Parameter, als auch die Anzahl der Workflowschritte ab, je höher man in der Pattern-Hierarchie ist.

Es ist zu beobachten, dass die Anzahl der Parameter, ab der Basic Data Management Patterns Stufe nur noch sehr langsam abnimmt. Ein Grund hierfür ist sicherlich, dass eine gewisse Anzahl an Parametern nötig ist, um das gewünschte Verhalten der Simulation zu beschreiben, sofern nicht auf die generische Einsetzbarkeit verzichtet werden soll.

Die Anzahl der Workflowschritte hingegen kann besser abstrahiert werden, so dass auf der Simulation-specific Process Patterns Ebene nur noch die 4 sehr abstrakte Workflowschritte übrig sind.

Obwohl die Anzahl der Parameter zwischen den höheren Stufen nur noch sehr langsam abnimmt und die Abstraktion des Workflows dadurch kaum zu nimmt, nimmt die Abstraktion der meisten Parameter jedoch zu. Während die Parameter auf der Basic Data Management Patterns Ebene sich direkt auf bestimmte Daten beziehen, beziehen sich die Parameter auf der Simulation-oriented Data Management Patterns Ebene schon auf Simulationsmodelle und die Beziehungen der abstrakten Simulationsparameter untereinander. Auf der Simulation-specific Process Patterns Ebene werden nur noch abstrakte, für das Simulationsmodell wichtige Parameter definiert, welche in der Terminologie der Simulation gehalten sind, und völlig unabhängig von der Workflow-Technologie sind.

Die Abstraktionsunterstützung wird also sowohl durch eine reduzierung der nötigen Workflowschritte, als auch durch die Umwandlung der datenspezifischen Parameter in simulationsspezifische Parameter erreicht und führt dazu, dass die Generierung neuer Simulationen oder die Änderung bestehender Simulationen für einen Benutzer deutlich vereinfacht wird.

6.0.6 Generische Einsetzbarkeit

Die in Kapitel 5 beschriebenen Patterns sollen nun auf ihre Wiederverwendbarkeit in anderen Workflows untersucht werden.

- **Parameter Sweep Pattern**
  Das Parameter Sweep Pattern bekommt eine Liste von Parametern übergeben und kann dabei mit beliebigen Parametern umgehen, sofern für sie eine Transformationsregel angeben wurde, in der Beschrieben wird, wie die vom Data Iteration Pattern benötigte Liste von Elementen erzeugt, geladen oder abgerufen wird. Diese Transformationsregel kann für neue Parameter leicht hinzugefügt werden und abgesehen davon kann das Parameter Sweep Pattern in allen Bereichen verwendet werden.
- **Simulation-oriented Data Interoperability Pattern**
  Das Simulation-oriented Data Interoperability Pattern kann beliebige Simulationsmodelle und Parameterrelationen verarbeiten, sofern sie im entsprechenden Format übergeben werden. Neue PartitionModes müssen jedoch implementiert werden und können nicht generisch angegeben werden.
- **Sequential Data Iteration Pattern**
  Das Sequential Data Iteration Pattern iteriert über beliebige in XML angegebene Liste und ist damit so generisch wie möglich gehalten.
- **Parallel Data Iteration Pattern**
  Das Parallel Data Iteration Pattern iteriert ebenfalls über beliebige in XML angegebene Liste und ist damit auch so generisch wie möglich gehalten.

7 Fazit

Pattern bieten eine gute Möglichkeit, um den Einsatz von Workflows für wissenschaftliche Simulationen zu vereinfachen und damit zu fördern. Sie erlauben durch ihre unterschiedliche Abstraktionsebenen eine Aufteilung der Zuständigkeiten, so dass sich die Experten der verschiedenen Ebenen jeweils auf ihren Bereich Fokussieren können.

Im Rahmen dieser Arbeit wurde in Kapitel 4 ein Kopplungsworkflow für eine biomechanische und ein systembiologiche Simulation erstellt und detailliert prototypisch implementiert. Dieser Kopplungsworkflow orchestriert die Ausführung dieser beiden Simulationen und sorgt dafür, dass die Daten zwischen den unterschiedlichen Formaten konvertiert und auf die jeweiligen Instanzen aufgeteilt wird.

Anschließend wurde in Kapitel 5 mehre Datenmanagementpatterns entworfen, die dazu dienen den implementierten Kopplungsworkflow zu abstrahieren und auf den höheren Ebenen der Pattern Hierarchie zu repräsentieren. Diese Pattern wurden anschließend im SIMPL-Prototyp möglichst generisch implementiert, so dass sie auch für andere Workflows benutzt werden können.

Literaturverzeichnis

[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 11)

[CCMW01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1., 2001. URL http://www.w3.org/TR/2001/NOTEwsdl-20010315. (Zitiert auf Seite 12)

[Dor11] R. Dormien. *Service-Bus-Erweiterung um Pandas-basierte Simulationen in Workflows zu nutzen.* Diplomarbeit, Universität Stuttgart, 2011. (Zitiert auf den Seiten 4, 10, 14, 19 und 23)

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. Conventional Workflow Technology for Scientific Simulation. *Guide to e-Science*, S. 323, 2011. (Zitiert auf den Seiten 4, 14 und 15)

[Har96] S. Hartmann. The World as a Process: Simulations in the Natural and Social Sciences. *Simulation and Modelling in the Social Sciences from the Philosophy of Scinece Point of View*, 1996. (Zitiert auf Seite 9)

[Jor07] J. Jordan, D.; Evdemon. Web Services Business Process Execution Language Version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. (Zitiert auf den Seiten 13 und 14)

[KAK+13] R. Krause, F. Allgöwer, D. Karastoyanova, B. Leymann, Frank andMarkert, B. Mitschang, P. Reimann, M. Reiter, D. Schittler, S. Schmitt, S. Waldherr, W. Ehlers. Scientific workflows for bone remodelling simulations. *Proceedings in Applied Mathematics and Mechanics*, 2013. (Zitiert auf den Seiten 23 und 25)

[KSR+11] R. Krause, D. Schittler, M. Reiter, S. Waldherr, F. Allgöwer, D. Karastoyanova, F. Leymann, B. Markert, W. Ehlers. Bone remodelling: A combined biomechanical and systems-biological challenge. *Proceedings in Applied Mathematics and Mechanics*, 11(1):99–100, 2011. (Zitiert auf den Seiten 7 und 23)

[LR99] F. Leymann, D. Roller. *Production Workflow: Concepts and Techniques.* Prentice Hall International, 1999. (Zitiert auf den Seiten 4, 12 und 13)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis.* Spektrum Akademischer Verlag, 2010. (Zitiert auf den Seiten 4, 10, 11, 13 und 14)

[OF06] M. Osterloh, J. Frost. *Prozessmanagement als Kernkompetenz: wie Sie Business Reengineering strategisch nutzen können.* Springer, 2006. (Zitiert auf Seite 12)

[Pie12] H. A. Pietranek. *Datenmanagementpatterns in multi-skalaren Simulationsworkflows*. Diplomarbeit, Universität Stuttgart, 2012. (Zitiert auf den Seiten 13, 14 und 19)

[Rei11] P. Reimann. SimTech Milestone Report: Data Provisioning for Scientific Workflows. Technischer Bericht, Technischer Bericht, Universität Stuttgart, 2011. (Zitiert auf den Seiten 4 und 20)

[RRS+11] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL-A Framework for Accessing External Data in Simulation Workflows. In *BTW*, S. 534–553. 2011. (Zitiert auf den Seiten 4, 7, 14, 17, 18 und 19)

[RS13] P. Reimann, H. Schwarz. Datenmanagementpatterns in Simulationsworkflows. In *BTW*, S. 279–293. 2013. (Zitiert auf den Seiten 4, 7, 23 und 24)

[RS14] P. Reimann, H. Schwarz. Simulation Workflow Design Tailor-Made for Scientists. *SSDBM*, 2014. (Zitiert auf Seite 21)

[RSM14a] P. Reimann, H. Schwarz, B. Mitschang. Data Patterns to Alleviate the Design of Scientific Workflows Exemplified by a Bone Simulation. *SSDBM*, 2014. (Zitiert auf den Seiten 4, 24 und 25)

[RSM14b] P. Reimann, H. Schwarz, B. Mitschang. A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design, 2014. Unveröffentlichter Bericht des Instituts für Parallele und Verteilte Systeme. (Zitiert auf den Seiten 4, 20, 21 und 37)

Alle URLs wurden zuletzt am 25. 07 2014 geprüft.

Erklärung

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

Ort, Datum, Unterschrift