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

Diplomarbeit Nr. 3668

Optimierung der Datenverarbeitung in Simulationsworkflows

Savas Kalyoncu

Studiengang:	Informatik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Dipl.-Inf. Peter Reimann
Beginn am:	17. Juli 2014
Beendet am:	16. Januar 2015
CR-Nummer:	D.2.11, H.2.5, H.4.1, I.6.7

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Datenintensive Workflows müssen häufig sehr große Datenmengen verarbeiten. Bestimmend für das Laufzeitverhalten ist bei solchen Workflows insbesondere die effiziente Ausführung von komplexen Datenverarbeitungsoperationen. Die Unterstützung existierender Ansätze, die es als Ziel haben, die Datenverarbeitung in datenintensiven Workflows zu optimieren, kann wesentliche Laufzeitgewinne bewirken ([Vrh11], [Zin10], [NS11]). Die Anwendung solcher Optimierungsansätze bietet sich auch bei computerbasierten Simulationen an, da Simulationsworkflows ebenfalls mit sehr großen Datenvolumina umgehen müssen. Die Größe der zu verarbeitenden Daten kann dabei zwischen mehreren GBs und zahlreichen TBs liegen. Workflows, die die Veränderung der Knochenstruktur unter verschiedenen Belastungen simulieren sind ein Beispiel für einen solchen Simulationsworkflow [K$^+$13] [RSM14b]. Ein anderes Beispiel ist ein Workflow, der bei der Proteinmodellierung verwendet wird, um durch einen Mustervergleich wichtige Bereiche innerhalb von Proteinsequenzen ausfindig zu machen [BTS].

Diese Workflows basieren auf dem Rahmenwerk SimTech Information Management, Processes and Languages (SIMPL) [RRS$^+$11]. SIMPL ist ein erweiterbares Rahmenwerk, welches für Simulationsworkflows die nahtlose Einbindung externer, heterogener Datenquellen möglich macht. Dazu stellt SIMPL eine generische Schnittstelle für Datenmanagement und -bereitstellung zur Verfügung. Zudem bietet es eine Hierarchie von Datenmanagementpatterns, mit denen Wissenschaftler komplexere Aufgaben der Datenverarbeitung bzw. -bereitstellung in Simulationsworkflows auf einfache Weise modellieren können [RSM14b]. Parametrisierungen der Patterns werden über Transformationsregeln automatisch auf ausführbare Workflowfragmente abgebildet.

In dieser Arbeit wurden Optimierungsmöglichkeiten für die Datenverarbeitung in Simulationsworkflows gesucht und anschließend verschiedene Optimierungen umgesetzt. Dazu wurde in verwandten Arbeiten zur Optimierung von datenintensiven Workflows verschiedene Optimierungsansätze betrachtet [Vrh11] [Zin10] [NS11] und ihre Übertragbarkeit auf Simulationsworkflows, insbesondere auf die in dieser Arbeit betrachteten Beispielworkflows [BTS] [RSM14b] [Boh], bewertet. Für ausgewählte Optimierungsansätze wurden anschließend konkrete Optimierungen detailliert betrachtet und entsprechende Optimierungen für Simulationsworkflows entworfen.

Außerdem wurde untersucht, ob und wie die in den Optimierungsansätzen vorgeschlagenen Optimierungen als Optimierungsregeln bei der regelbasierten Abbildung von Datenmanagementpatterns auf ausführbare Workflowfragmente Verwendung finden können. Dafür wurden Bedingungen herausgearbeitet, unter denen eine bestimmte Optimierung auf einen Teil eines Workflows angewendet werden kann. Weiterhin wurde betrachtet, für welche konkreten Patterns und auf welcher Ebene der Patternhierarchie eine bestimmte Optimierungsregel in Frage kommen könnte.

Schließlich wurden im SIMPL-Rahmenwerk für einzelne Optimierungen und die kombinierte Anwendung mehrerer Optimierungen Anwendungsszenarien bzw. Testworkflows implementiert und diese

durch anschließende Testausführungen im Hinblick auf erzielte Laufzeitverbesserungen evaluiert. Basierend auf dem Entwurf und der Evaluation dieser Anwendungsszenarien wurden abschließend mögliche Verbesserungen der Implementierung des SIMPL-Rahmenwerks, die zu einer höheren Effizienz von Simulationsworkflows führen können, vorgeschlagen.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden werden die Grundlagen dieser Arbeit beschrieben.

Kapitel 3 – Das SIMPL-Rahmenwerk: In diesem Kapitel wird das SIMPL-Rahmenwerk vorgestellt.

Kapitel 4 – Knochensimulationen und Proteinmodellierung: Die für diese Arbeit relevanten Workflows zur Simulation von Knochenstrukturveränderungen und zur Proteinmodellierung werden in diesem Kapitel beschrieben.

Kapitel 5 – Optimierung von datenintensiven Workflows: Bestehende Ansätze zur Optimierung von datenintensiven Workflows werden in diesem Kapitel betrachtet.

Kapitel 6 – Bewertung der Optimierungsansätze: Die Bewertung der bestehenden Ansätze zur Optimierung von datenintensiven Workflows hinsichtlich ihrer Übertragbarkeit auf Simulationsworkflows erfolgt in diesem Kapitel.

Kapitel 7 – Regelbasierte Optimierung von Simulationsworkflows: In diesem Kapitel werden aus dem regelbasierten PGM/F-Ansatz [Vrh11] Optimierungsregeln für Simulationsworkflows abgeleitet.

Kapitel 8 – Umsetzung und Anwendungsbeispiele für die generischen Optimierungsregeln: Hier werden entwickelte Anwendungsszenarien und die Anwendung von Optimierungsregeln auf diese beschrieben.

Kapitel 9 – Optimierungen der Transformation von Datenmanagementpatterns: Auf die Integrationsmöglichkeiten von Optimierungen in die Transformation von Datenmanagementpatterns wird in diesem Kapitel eingegangen.

Kapitel 10 – Evaluation: Hier werden die Messergebnisse für die Ausführung der entwickelten Anwendungsszenarien aufgezeigt und bewertet.

Kapitel 11 – Optimierung des SIMPL-Rahmenwerks: Auf Optimierungsideen für das SIMPL-Rahmenwerk, welche durch die Entwicklung von Anwedungsszenarien und ihrer Ausführung ermittelt wurden, wird in diesem Kapitel eingegangen.

Kapitel 12 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und gibt einen kurzen Ausblick.

2 Grundlagen

In diesem Kapitel werden grundlegende, informationstechnische Begriffe und Inhalte erklärt, die im weiteren Verlauf dieser Arbeit verwendet werden und für das Verständnis der Thematik dieser Arbeit notwendig sind.

Die in dieser Arbeit beschriebenen Workflows verwenden für ihre Ein- und Ausgabedaten ein auf XML basierendes Datenformat. Desweiteren handelt es sich bei der Workflowsprache *WS-BPEL*, die zur Definition der betrachteten Workflows benutzt wird, um eine auf XML basierende Sprache. In den folgenden Unterabschnitten wird die Struktur und die Verarbeitung von XML-Dokumenten näher beschrieben.

2.1 XML

XML ist die Abkürzung für *Extensible Markup Language*. Es bezeichnet eine erweiterbare Meta-Auszeichnungssprache zur Beschreibung von hierarchisch strukturierten Daten bzw. deren Inhalt. Die Idee, die XML und XML-Anwendungen zugrunde liegt, ist die Trennung von Inhalt, Struktur und Format von Dokumenten. Dadurch können Inhalte plattform- und anwendungsneutral ausgetauscht werden. XML wurde unter der Schirmherrschaft des World Wide Web Consortium (W3C) entwickelt [Min02] und liegt momentan in der fünften Version vor [?].

2.1.1 Bausteine eines XML-Dokuments

Den Aufbau einer einfachen XML-Datei zeigt das Listing 2.1. Es gibt strenge syntaktische Regeln für XML. Die Trennung von Inhalt, Format und Struktur von Dokumenten wird realisiert durch eine spezielle Syntax, in der alle Informationen innerhalb des Dokuments durch sogenannte *Tags* (auch "Markups")gekennzeichnet werden. Die Gesamtheit der Tags bildet das Markup.

Tags treten immer paarweise auf. Es gibt eine Start-Tag und dazu passend ein End-Tag. Start- und End-Tag umschliessen einen gewissen Inhalt des XML-Dokumentes und legen seine Bedeutung durch einen Namen fest. Das Konstrukt der Form `<tag>Inhalt</tag>` bestehend aus Start-, End-Tag und der dazwischen liegende Inhalt bilden den grundlegenden Baustein von XML-Dokumenten und werden als Elemente bezeichnet [Par00]. Tags werden innerhalb des Dokuments durch Zeichenfolgen dargestellt, welche von spitzen Klammern umschlossen werden. Die umschlossene Zeichenfolge entspricht einem Elementnamen. Start-Tag und das zugehörige End-Tag besitzen denselben Elementnamen. End-Tags werden im Gegensatz zu den Start-Tags durch ein „/“ eingeleitet. Im Beispiel ist u.a. `<Vorname>` ein Start-Tag, `</Vorname>` ein End-Tag und das Gesamtkonstrukt `<Vorname>Savas</Vorname>` ein

Listing 2.1 Ausschnitt aus einer XML-Datei

```
<Adressenliste>
  <Adresse Nr="1">
    <Name>
      <Vorname>Savas</Vorname>
      <Nachname>Kalyoncu</Nachname>
    </Name>
    <Strae>Musterstrae 2/15</Strae>
    <Ort>Ostfildern</Ort>
    <Plz>73760</Plz>
  </Adresse>
  <Adresse Nr="2">
    <Name>
      <Vorname>Peter</Vorname>
      <Nachname>Mustermann</Nachname>
    </Name>
    <Strae>Musterstrae 1</Strae>
    <Ort>Esslingen</Ort>
    <Plz>73728</Plz>
   </Adresse>
</Adressenliste>
```

Element. Start-Tags können zudem Attribute besitzen, welche aus einem Attributnamen und Attributwerten bestehen. Durch Attribute können Eigenschaften der Elemente definiert werden. Im Beispiel aus Listing 2.1 hat das erste Element *<Adresse NR="1»* ein Attribut NR mit dem Wert 1.

XML ermöglicht es dem Programmierer eigene Tags zu definieren, da im Gegensatz zu HTML die Namen für Tags und Attribute nicht vorgegeben sind. Die Definition dieser Informationen erfolgt in separaten Dateien über Spezifikationssprachen wie z.B. *XML Schema* (vergleiche Abschnitt 2.1.4) oder *Document Type Definition* (DTD) (vergleiche Abschnitt 2.1.4). Somit ermöglicht es XML beliebige Datenstrukturen zu erzeugen, deren Inhalt durch das Markup semantisch festgelegt wird.

Elemente können Text und/oder weitere Elemente als Unterelemente enthalten (Verschachtelung). Dabei muss sich das Unterelement komplett zwischen Start- und End-Tag des umgebenden Elementes befinden. Das Unterelement wird dann als Kindelement des umschliessenden Elternelementes bezeichnet. Durch diese Verschachtelung bilden XML-Dokumente eine Hierarchie, dessen oberstes Element als Wurzelement bezeichnet wird. Somit enthält das Wurzelement alle Elemente eines XML-Dokumentes. In unserem Beispiel bildet das Element <Visitenkarte> das Wurzelement. In unserem Beispiel sind <Vorname> und <Nachname> Kindelemente des Elements <Name>. Das Element <Visitenkarte> bildet das Wurzelelement des Dokuments.

2.1.2 Aufbau eines XML-Dokuments

XML-Dokumente müssen bezüglich ihrer Struktur und ihres Inhalts strenge syntaktische Vorgaben einhalten. Indem XML-Dokumente einer gewissen Grundstruktur folgen, können sie später automatisch weiterverarbeitet und ausgetauscht werden. Der Standardaufbau eines XML-Dokuments gliedert sich folgendermassen. Den ersten Bestandteil eines XML-Dokuments bildet eine XML-Deklaration.

Listing 2.2 Auflösung von Namenskonflikt durch Namensräume

```
<Mitarbeiter Name="Kalyoncu" xmlns="http://www.firma.de/Mitarbeiter">
  <Adresse>
    <Strae>Musterstrae 2</Strae>
    <Ort>Ostfildern</Ort>
    <Plz>73760</Plz>
  </Adresse>
  <Rechner xmlns="http://www.firma.de/Rechner">
    <domain>firma.de</domain>
    <Adresse>168.111.222.444</Adresse>
    <Name>Mitarbeiterrechner</Name>
  </Rechner >
</Mitarbeiter>
```

Danach folgt die Definition des XML-Schemas. Diese beiden Bestandteile sind optional. Schließlich folgt das Wurzelelement, welches im Gegensatz zu den erstgenannten Bestandteilen nicht weggelassen werden kann.

Die Deklaration in der ersten Zeile des XML-Dokuments gibt an welche XML-Version und welche Zeichenkodierung zur Speicherung des Dokumenteninhalts benutzt wird. Die XML-Deklaration ist zwar optional, kann jedoch die weitere Verarbeitung des XML-Dokuments vereinfachen. Das XML-Schema legt die Struktur des eigentlichen Inhalts des XML-Dokuments fest. Die Definition dieses Schemas kann durch Angabe einer Document Type Definiton (DTD) oder eines XML-Schemas erfolgen. Abschnitt Abschnitt 2.1.4 stellt mit XML-Schemas die aktuellere der beiden Methoden vor.

2.1.3 Namensräume

Es kann vorkommen, dass innerhalb eines XML-Dokuments oder auch bei der Kombination verschiedener auf XML basierender Dokumente identische Element- oder Attributnamen verwendet werden. Dies resultiert in Namenskonflikten, d.h. dieselben Namen besitzen mehrere Bedeutungen. In Listing Listing 2.1 ist ein Beispiel gegeben, bei dem das Element Adresse zwei Bedeutungen hat. Einerseits bezeichnet es eine Adresse einer Person, d.h. die Anschrift mit Straße und Wohnort. Andererseits steht es für die Adresse eines Rechners, d.h eine IP-Adresse. Um solche Mehrdeutigkeiten aufzulösen, werden Namensräume verwendet. Der global eindeutige Name eines Elementes oder Attributes wird durch sogenannte Qualifizierte Namen (qualified names)festgelegt. Qualified Names bestehen aus einem Präfix, der den Namensraum bezeichnet und einem lokalen Namen, der den Namen des Elements innerhalb des Namensraums bezeichnet. Der Namensraum ist durch einen Uniform Resource Identifier (URI) global eindeutig identifizierbar. Durch die Kombination aus Namensraum und einem lokalen Namen kann in unterschiedlichen Namensräumen derselbe Name angewendet werden und innerhalb desselben Namensraums ist jeder verwendete Name eindeutig.

In Listing 2.2 wurde das Konzept der Namensräume auf das Beispiel in Listing 2.2 angewandt. Hier wird als Default-Namensraum "Mitarbeiter"verwendet. Das Element "Rechner"hingegen greift auf den Namensraum "Rechnerßurück. Somit wird eindeutig zwischen Rechneradressen und den Adressen von Mitarbeitern unterschieden.

Listing 2.3 Beispiel für ein XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.firma.com"
      xmlns="http://www.firma.com" elementFormDefault="qualified">
       <xs:element name="Dokument">
            <xs:complexType>
          <xs:sequence>
           <xs:element name="Titel" type="Titel"
                  minOccurs="1" maxOccurs="1" />
              <xs:element name="Eintrag" type="Eintrag"
                  minOccurs="0" maxOccurs="unbounded" />
             </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="Titel">
        <xs:sequence>
         <xs:element name="Titel" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="Besitzer" type="xs:string" use="required" />
      </xs:complexType>
      <xs:complexType name="Eintrag">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" />
            <xs:element name="Vorname" type="xs:string" />
            <xs:element name="Anschrift" type="AnschriftTyp" />
            <xs:element name="EMail" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="AnschriftTyp">
        <xs:attribute name="Strae" type="xs:string" use="required" />
        <xs:attribute name="Plz" type="xs:string" use="required" />
        <xs:attribute name="Ort" type="xs:string" use="required" />
      </xs:complexType>
 </xs:schema>
```

2.1.4 XML Schema

XML-Schemata sind eine Weiterentwicklung von DTDs. Die Weiterentwicklung war notwendig , da DTDs gewisse Mängel aufweisen und eine begrenzte Ausdruckskraft besitzen [KE11]. Beispielsweise fehlt bei einer DTD die Kompatibilität zum Konzept der Namensräume. D.h. eine DTD kann keine Namensräume und somit keine Mehrdeutigkeiten unterscheiden. Mit PCDATA und CDATA sind die zur Auswahl stehenden Datentypen sehr eingeschränkt. Elemente können im Grunde genommen nur als Texte definiert werden.

XML-Schemas sind im Gegensatz dazu sehr ausdrucksstark. Sie nutzen selbst die XML Syntax und können somit wie jedes andere XML-Dokument geparst werden. Bei der Definiton von Elementen können die in Programmiersprachen gängigsten Datentypen wie Integer, String oder Boolean verwendet werden. XML-Schemas erlauben zudem die Definition von eigenen, einfachen Datentypen sowie von komplexeren Datentypen. Einfache Datentypen sind beispielsweise Vereinigungen und

Listen. Bei komplexen Datentypen handelt es sich um Element-Strukturen, die wiederum andere Elemente oder Attribute enthalten können.

Listing 2.3 zeigt, wie ein Schema für ein Beispiel-Dokument. In den ersten Zeilen sieht man beispielhaft die Definition eines Namensraums. Ein Dokument ist definiert als ein komplexer Datentyp repräsentiert durch eine Sequenz bestehend aus den Elementen Titel und Eintrag, welche wiederum als komplexe Datentypen definiert sind. Das Element Titel besitzt die Attribute minOccurs = ”1” und maxOccurs = ”1” und muss somit genau einmal in der Sequenz vorkommen. Das Element Eintrag besitzt die Attribute minOccurs = ”0” und maxOccurs = ”unbounded” und kann somit gar nicht, einmal oder unbegrenzt oft in der Sequenz vorkommen. Werden zur Kardinalität keine Angaben gemacht, wird sowohl minOccurs als auch maxOccurs standardmäßig auf den Wert Eins gesetzt, d. h. ein Element darf nur genau einmal in der Liste vorkommen. Da für das Vorkommen von Attributen bzw. Elementen direkt Zahlen eingegeben werden können, kann die Häufigkeit dieser Elemente bzw. Attribute im Gegensatz zu einer DTD exakt vorgegeben werden. Im Beispiel ist auch ersichtlich, dass komplexe Datentypen nicht nur Kombinationen von einfachen Datentypen sein müssen, sondern wieder aus komplexen Datenypen aufgebaut sein können. Der komplexe Datentyp ”Anschrift” ist Bestandteil des komplexen Datentyps ”Element”. Weiterhin können auf den Datentypen Einschränkungen (restrictions) definiert werden. Für die Definition solcher Restrictions kann unter anderem auf reguläre Ausdrücken zurückgegriffen werden, welche bei der Validierung des XML-Dokuments ausgewertet werden. Dies stellt ein mächtiges Werkzeug bei der Validierung von XML-Dokumenten dar. Diese Eigenschaft von XML Schema is jedoch nicht im Beispiel aus Listing 2.3 dargestellt.

2.1.5 Verarbeitung von XML

Zur Abfrage,Verwaltung und Manipulation von XML-Dokumenten stehen eine große Anzahl unterschiedlicher Sprachen zur Verfügung. In diesem Unterabschnitt werden die Sprachen *XPath* und *XQuery* kurz vorgestellt. Für weitere Details wird auf die Literatur [CD$^+$] und [BCF$^+$06] verwiesen. XPath wird verwendet um auf einzelne Elemente von XML-Dokumenten zuzugreifen. XQuery hingegen ist eine Sprache, die auf XPath basiert und komplexere Abfragen ermöglicht, die in ihrem Aufbau SQL-Anfragen bei relationalen Datenbanksystemen ähneln.

XPath

XPath ist eine vom World Wide Web Consortium (W3C) standardisierte Abfragesprache für XML-Dokumente. Einzelne Bestandteile eines XML-Dokuments können adressiert werden, indem das gesamte XML-Dokumente als ein Baum und die Elemente als Knoten aufgefasst werden. Der Wurzelelement des Baumes ist ein rein virtueller Knoten un repräsentiert keinen Bestandteil des XML-Dokuments. Bei den Knoten werden unter anderem zwischen Elementknoten, Attributknoten, Namensraumknoten und Textknoten unterschieden.

Zur Navigation innerhalb eines XML-Dokuments, d.h. der Lokalisierung von Knoten werden XPath-Ausdrücke verwendet. Diese werden zusammengesetzt aus mehreren sogenannten *Lokalisierungsschritten* , die durch "/Zeichen getrennt werden. Durch jeden Lokalisierungsschritt werden bestimmte

Knoten des XML-Dokuments selektiert. Anschließende Lokalisierungsschritte nutzen die im vorherigen Lokalisierungschritt selektierten Knoten als Referenz, um von diesen ausgehend weitere Knoten zu selektieren. Ein Lokalisierungpfad ist folgendermaßen aufgebaut:

`/Achse::Knotentest[Prädikat][...]`

Die Navigationsrichtung im Pfad wird durch die Achsen festgelegt. Sehr häufig verwendete Achsen sind beispielsweise *child*, mit denen alle direkten Unterknoten eines Knotens, *parent*, mit denen alle Elternknoten eines Knotens, *self*, mit denen der Knoten selbst und *attribute*, mit denen alle Attributknoten eines Knotens selektiert werden können. Desweiteren können mit der Achse *descendant* alle untergeordneten Knoten, mit der Achse *descendant-or-self* alle untergeordneten Knoten einschließlich der Knoten selbst, mit der Achse *ancester* alle übergeordneten Knoten und mit der Achse *ancester-or-self* alle übergeordneten Knoten einschließlich dem Knoten selbst.

Durch einen Knotentest kann die Selektion von Knoten eingeschränkt werden, indem nur Knoten referenziert werden, die eine angegebene Bedingung erfüllen. Eine weitere Einschränkung ist durch Angabe eines Prädikates möglich. Beispielsweise werden im Listing 2.1 durch den XPath-Ausdruck

`/child::Adressenliste/child::Adresse`

alle Unterknoten mit dem Namen Adresse ausgewählt.

In dieser Arbeit wird die abgekürzte Syntax von XPath verwendet. Details hierzu sind in [C+]zu finden.

Durch den XPath-Ausdruck

`/child::Adressenliste/child::Adresse[child::Plz="73760"]` wird mit Hilfe des Prädikats das Element mit dem Namen Adresse und der Postleitzahl 73760 ausgewählt. Als Ergebnis wird folgendes geliefert:

```
<Adresse Nr="1»
 <Name>
  <Vorname>Savas</Vorname>
  <Nachname>Kalyoncu</Nachname>
 </Name>
 <Straße>Musterstraße 2/15</Straße>
 <Ort>Ostfildern</Ort>
 <Plz>73760</Plz>
</Adresse>
```

XQuery

Die Sprache XQuery ist eine auf XPath basierende Sprache, die jedoch Erweiterungen anbietet, durch die komplexere Abfragen formuliert werden können. Das Hauptkonstrukt, mit dessen Hilfe XQuery-Ausdrücke formuliert werden, ist das *FLWOR*-Konstrukt. Es ist eine Abkürzung für *FOR*, *LET*, *WHERE*, *ORDER BY*, *RETURN*. Ähnlich der Abfragesprache für relationale Datenbanken SQL (vergleiche Abschnitt 2.1.4) können dabei Operationen auf den XML-Dokumenten ausgeführt werden.

Die For-Klausel, bestehend aus For und Let, iteriert mit *FOR* über eine Menge von Knoten und bindet in jedem Iterationsschritt mit *LET* den aktuellen Knoten an eine Variable. Mit der *WHERE*-Klausel wird diese Menge von Knoten durch die Definition von Bedingungen weiter eingeschränkt. Die *ORDER-BY*-Klausel ermöglicht die Sortierung der selektierten Knoten. Abschließend liefert die Result-Klausel das Resultat der gesamten Abfrage zurück. Es besteht die Möglichkeit, FLWOR-Ausdrücke sowohl verschachtelt, als auch rekursiv zu definieren. Bezogen auf das XML-Dokument in Listing 2.1 würde die FWLOR-Abfrage

```
FOR $a in doc (“Adressenliste.xml“)/Adresse WHERE $a/Plz=’73728’ RETURN $a
```

die zweite Adresse in der Adressliste mit Postleitzahl 73728 zurückliefern.

Die aktuelle Spezifikation von XQuery bietet nicht die Möglichkeit, XML-Dokumente direkt zu manipulieren. Datenbanksysteme wie z.B. MonetDB (vergleiche Abschnitt Abschnitt 2.1.4), die XQuery als Abfragesprache unterstützen, stellen jedoch entsprechende Funktionen bereit.

2.2 Datenbanken

Für die Verwaltung von Daten und den Zugriff auf sie gibt es viele verschiedene Technologien und Produkte. Datenbanksysteme (DBS) sind heutzutage eine weit verbreitete Technologie, die beispielsweise im Unternehmensbereich und im Internet verstärkt eingesetzt werden. Sie bestehen aus zwei Komponenten, dem Datenbankmanagementsystem (DBMS) und der eigentlichen Datenbank. Während das DBMS für die Verarbeitung und Verwaltung von Daten verantwortlich ist, werden die eigentlichen Daten in der Datenbank abgespeichert. Nach [KE11] gibt es vielfältige Vorzüge, die sich aus dem Einsatz eines DBS ergeben. Beispielsweise wird zum einen durch die Speicherung von Daten in einem DBS die Datenredundanz, d.h. die mehrfache Speicherung von Daten, im Gegensatz zur Datenhaltung in einem Dateisystem erheblich reduziert. Zum anderen können Nutzern, für eine sichere Verwaltung der Daten, Zugriffs- und Verwaltungsrechte auf bestimmte Daten gewährt oder verweigert werden. Desweiteren kann durch die Definition von Integritätsbedingungen verhindert werden, dass fehlerhafte Eingaben gemacht werden. Dies wiederum garantiert, dass die gespeicherten Daten sich zu jedem Moment in einem konsistenen Zustand befinden. Weitere Vorteile sind der erleichtere Zugriff auf Daten durch das Anbieten von sogenannten Abfragesprachen, der Mehrbenutzerbetrieb und die Wiederherstellung von Daten bei drohendem Datenverlust. Weitere Details bezüglich der Einsatzmöglichkeiten und Vorteile von DBSen können in [KE11] nachgelesen werden. Sehr häufig verwendete DBS sind relationale Datenbanksysteme und XML-Datenbanken. Diese werden im nächsten Abschnitt etwas näher beschrieben.

2.2.1 relationale Datenbanken und SQL

Relationale Datenbankensysteme basieren auf sogenannten Relationen. Diese sind bildlich gesehen Tabellen, in denen die Daten gespeichert werden. Die Zeilen einer Tabelle, auch als Tupel bezeichnet, entsprechen einem Datensatz. Die Spalten einer Tabelle entsprechen dabei Attributen, die Werte aus einer festgelegten Domäne annehmen können, d.h. Tupel setzen sich aus mehreren Attributwerten zusammen. Tabellen können miteinander verknüpft werden, indem sogenannte Fremd- und

Listing 2.4 Form einer üblichen SQL-Anfrage

```
SELECT attribute1, attribute2....
FROM tabelle1, tabelle2.....
WHERE bedingung1....
```

Primärschlüssel genutzt werden. Primärschlüssel sind dabei eine Menge von Attributen einer Relation, die einen Tupel der Relation eindeutig identifizieren. Fremdschlüssel hingegen, sind eine Menge von Attributen innerhalb einer Relation, die Primärschüssel innerhalb derselben oder einer anderen Relation referenzieren. Basierend auf solchen Fremdschlüssel-Primärschlüssel-Beziehungen können auf den Relationen bzw. auf den Daten mengenorientierte Operationen ausgeführt werden, d.h. Datensätze können durch Selektion, Vereinigung oder dem Schnitt verschiedener Datensätze erzeugt bzw. abgefragt werden. Solche Abfragen werden mit Hilfe der Abfragesprache Structured Query Language (SQL) formuliert. SQL ist eine weit verbreitete, deklarative Sprache. Bei deklarativen Sprachen muss der Nutzer lediglich angeben, welche Daten vom Datenbanksystem zurückgegeben werden sollen. Für die eigentiche Verarbeitung und Auswertung ist das Datenbanksystem selbst verantwortlich. Die allgemeine Form einer einfachen SQL-Anfrage ist in Listing 2.4 dargestellt.

Dabei definiert die FROM-Klausel die Tabellen, aus denen die für die Abfrage relevanten Daten bezogen werden. Durch die SELECT-Klausel werden die Attribute ausgewählt, die im Ergebnis, bzw. in den Ergebnis-Tupeln der Anfrage zurückgegeben werden sollen. Wird hier das Symbol * ausgewählt, werden alle Attribute im Resultat einbezogen. In der WHERE-Klausel wird schließlich festgelegt, welche Bedingungen die zurückzuliefernden Tupel bzw. ihre Attributwerte erfüllen müssen.

2.2.2 XML-Datenbanken

Mit der steigenden Verbreitung von XML (vergleiche Abschnitt 2.1) wurden auch Technologien entwickelt, die den Versuch unternahmen, die Technologie der Datenbanksysteme mit diesem Datenformat zu verschmelzen. [Sch03] unterscheidet bei diesen Technologien drei verschiedene Modelle, die unterschiedliche Ansätze verfolgen:

- **monolithischer** Ansatz: hier wird die gesamte Datenbank durch ein einziges, großes XML-Dokument dargestellt. Durch die Manipulation dieses XML-Dokuments werden Informationen der Datenbank hinzugefügt bzw. entfernt. Als problemtisch erweist sich hier jedoch, dass einzelne XML-Dokumente schlecht gehandhabt werden können, denn unter dem Wurzelement eines XML-Dokuments (in diesem Fall des Dokuments welches die gesamte Datenbank repräsentiert) können keine Prologs oder Document Type Definitions gespeichert werden.
- **fragmentorientierter** Ansatz: dieser Ansatz wird oft von relationalen Datenbanksystemen verfolgt, die die Verwaltung von XML-Dokumenten unterstützen. Daten werden aus XML-Sicht unverknüpft in der Datenbank gespeichert. Dabei werden sie in mehrere Fragmente aufgeteilt. Bei der Extraktion von Daten werden XML-Dokumente bzw. -Fragmente entsprechend wieder aus ihren (Unter)-Fragmenten rekonstruiert. Als problematisch bei dieser Ko-Existenz von relationaler und XML-Sicht erweist sich die angesprochene Rekonstruktion der ursprünglichen Dokumente, da ein Informationsverlust stattfinden kann (z.B. Prolog, DTD, Kommentare).

Weiterhin sind bei dieser Art der Speicherung innerhalb der Datenbank keine Beziehungen zwischen den einzelnen Fragmenten ersichtlich.

- **dokumentenorientierter** Ansatz: bei diesem Ansatz werden ganze XML-Dokumente in der Datenbank verwaltet. Die Datenbank entspricht also einer Menge von XML-Dokumenten. Einzelne Dokumente können zu benutzerspezifischen Sammlungen, sogenannte Collections, zusammengefasst werden.

Relationale Datenbanksysteme, wie z.B. Mircosoft SQL Server1, die mit XML-Dokumenten umgehen können, werden als XML-enabled Datenbanksysteme bezeichnet. Native XMLDatenbanksysteme, wie z.B. MonetDB/XQuery2[1] oder Exist3[2], verfolgen den dokumentorientierten Ansatz.

2.3 Service-Orientierte Architektur

Der Begriff einer serviceorientierten Architektur (SOA) bezeichnet keine konkrete Technologie oder Architektur, sondern einen Architekturstil, bei dem das Konzept der Dienste (Services) im Vordergrund stehen. Es gibt viele verschiedene Definitonen einer SOA, die verschiedene Aspekte in den Vordergrund stellen. Somit ist es schwierig sich auf eine universelle Definition festzulegen. Eine mögliche Definition nach [Mel10] lautet 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."

Der Begriff *Dienst* (Service) steht sowohl in dieser Definition, als auch generell bei serviceorientierten Architekturen im Vordergrund. Im weiteren Verlauf dieser Arbeit wird der Begriff Dienst verwendet. Dienste stellen ihre Funktionen über öffentlich definierte Schnittstellen bereit. Anwendungen oder Dienste können die Funktionen anderer Dienste in Anspruch nehmen. Die Einbindung von Diensten erfolgt im Rahmen einer losen Kopplung, d.h. Dienste werden erst bei Bedarf dynamisch eingebunden. Dadurch können Dienste prinzipiell gegeneinander ausgewechselt werden. Durch Kombination mehrerer Dienste können größere Dienste, wie z.B. Geschäftsprozesse aufgebaut werden (*Orchestrierung*). Dies erhöht die Wiederverwendbarkeit von Diensten. Das Auffinden von Diensten, die bestimmte Funktionen zur Verfügung stellen, erfolgt über Verzeichnisdienste (*Service Registry*). Alle verfügbaren Dienste sind in diesem Verzeichnisdienst registriert. Bei Bedarf suchen Anwendungen im Verzeichnisdienst nach Diensten und können sie dann anhand der Informationen, die vom Verzeichnisdienst zurückgeliefert werden, dynamisch einbinden. Diese Art von Architektur resultiert in einer hohen Fexibilität und einer erhöhten Wiederverwendbarkeit von Diensten, ist ist aber nur durch die Verwendung von offenen Standards und die Maschinenlesbarkeit von Schnittstellenbeschreibungen realisierbar.

[1] http://dbis.cs.tu-dortmund.de/cms/de/publications/2006/monetdb-xquery/index.html
[2] http://exist-db.org/exist/apps/homepage/index.html

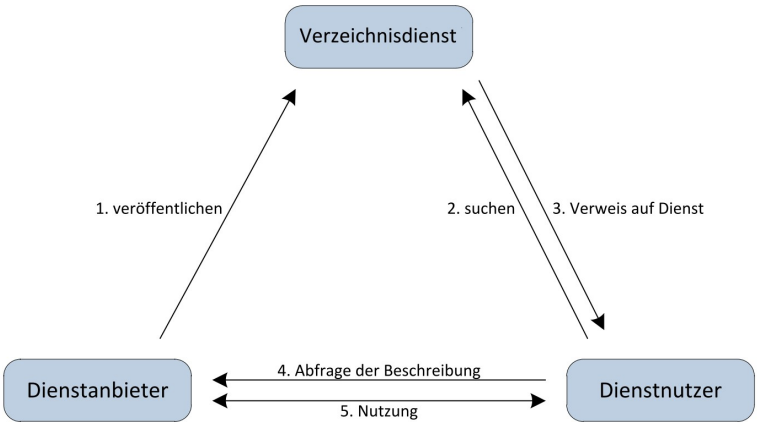


Abbildung 2.1: Das SOA-Dreieck [LR00]

2.3.1 Das SOA-Dreieck

Das SOA-Dreieck Abbildung 2.1 stellt alle Komponenten einer serviceorientierten Architektur und ihre Interaktionen dar. Es besteht aus drei Teilnehmern, dem Verzeichnisdienst, dem Dienstanbieter und dem Dienstnutzer. Der Dienstanbieter erstellt für den angebotenen Dienst eine maschinenlesbare Service-Beschreibung und registriert ihn mitsamt Metadaten im Verzeichnisdienst. Diese Metadaten beinhalten alle Informationen, die zum Auffinden und Aufrufen des Dienstes benötigt werden, wie z.B. eine abstrakte Beschreibung des Dienstes, die angebotene, öffentliche Schnittstelle, den Anbieter oder die IP-Adresse. Der Dienst wird durch die Registrierung veröffentlicht. Das Deployment, die Bereitstellung der notwendigen Infrastruktur, die Ausführung, sowie weitere Aufgaben (z.B. Sicherung des Quality of Service oder Datensicherheit) obliegen jedoch dem Dienstanbieter. Der Dienstnutzer kann nun über die in der Registry abrufbaren Daten nach einem passenden Dienst suchen. Ist ein passender Dienst gefunden, kann der Nutzer dann dessen Schnittstellenbeschreibung beim Dienstanbieter erfragen. Schließlich kann der Dienst durch den Nutzer genutzt werden.

2.3.2 Web Services

Eine Implementierungsmöglichkeit für serviceorientierte Architekturen sind unter anderem *Web Services* [BHM+04]. Webservices sind Dienste, die über eine feste URI adressiert werden können und über diese Adresse ihre Funktionalitäten zur Verfügung stellen. Nach [ABFG04] ist ein Web Service folgendermaßen definiert: „A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.“

Die Grundkonzepte für Web Services bilden dabei die Web Service Definition Language (*WSDL*), das *SOAP*-Protokoll und die Universal Description Discovery & Integration (*UDDI*). WSDL ist eine Sprache mit der die Schnittstellen von Web Services und somit der Zugriff auf diese öffentlich beschrieben wird. Eine nähere Beschreibung von WSDL findet sich in Abschnitt 2.3.4. Das SOAP-Protokoll dient

Listing 2.5 Beispiel einer SOAP-Nachricht

```
<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>
```

zum Austausch von Nachrichten mit dem Web Service. Es wird in Abschnitt 2.3.3 beschrieben. Der standardmäßige Aufbau einer Verzeichnisstruktur für Web Services wird durch die UDDI festgelegt, welche im Abschnitt 2.3.5 behandelt wird.

2.3.3 SOAP

Das Simple Object Protocol(*SOAP*) ist ein Netzwerkprotokoll, das hauptsächlich zum Austausch von Nachrichten zwischen Web Services genutzt wird. Aufgrund seiner Unabhängigkeit von Transportprotokollen kann es mit verschiedenen Transportprotokollen wie HTTP oder SMTP zum Einsatz kommen. Eine der gängigsten Kombinationen ist die Kombination von HTTP mit SOAP. SOAP-Nachrichten werden standardmäßig mit XML-Schema definiert, wobei auch andere Formate wie CSV möglich sind. Eine SOAP-Nachricht besteht aus einem Envelope, ein oder mehreren Headern und einem Body. Das Envelope bildet den Container, der die Header und den Body einer SOAP-Nachricht und Informationen über die SOAP-Spezifikation beinhaltet. Die Header sind optional und enthalten oft Informationen für den Empfänger oder notwendige Informationen, die den Zwischenstationen auf dem Transportweg vom Absender zum Empfänger der Nachricht zur Weiterverarbeitung dienen. Beispielsweise kann ein im Header enthaltenes Attribut "mustUnderstand", welches auf "true"gesetzt ist, bedingen, dass das Header-Element vom empfangenden Knoten verarbeitet werden muss. Kann er nicht weiterverarbeitet werden, kann auch die Nachricht nicht mehr weiterverarbeitet oder weitergeleitet werden. Weitere Attribute können der Datenverschlüsselung oder der Authentifizierung dienen. Der Body einer SOAP-Nachricht ist im Gegensatz zu den Headern nicht optional. Er enthält den eigentlichen Inhalt (payload) der Nachricht, der für den Empfänger relevant ist.

2.3.4 WSDL

WSDL ist eine Abkürzung für Web Service Description Language und ist eine auf XML basierende, programmiersprachenunabhängige Sprache zur Beschreibung von Web Services.

Dabei wird ein Web Service auf zwei Ebenen beschrieben (siehe Abbildung 2.2). Während auf der abstrakten Ebene die Funktionalitäten beschrieben werden, die durch den Web Service bereitgestellt

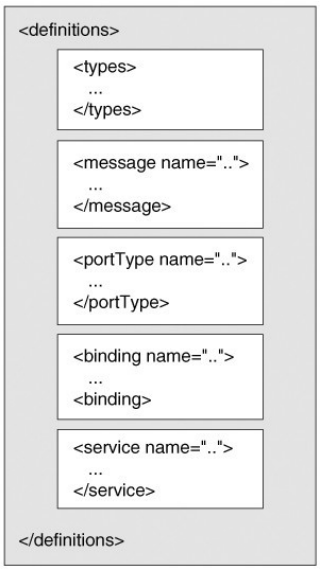


Abbildung 2.2: WSDL-Struktur [WCL+05]

werden, werden auf der konkreten Ebene technische Details beschrieben, wie der Web Service an sich bereitgestellt wird.

Auf der abstrakten Ebene werden zunächst alle verwendeten Datentypen (types) des Web Services definiert. Weiterhin werden alle Port-Typen (port-types) festgelegt. Port-Typen sind aus mehreren Operationen zusammengesetzt und bilden abstrakte Beschreibungen der Schnittstellen, welche die vom Web Service angebotene Funktionalitäten beschreiben. Schließlich werden mit Hilfe der zuvor definierten Datentypen die zu den Operationen gehörenden Input- und Output-Nachrichten (messages) beschrieben.

Dieses abstrakte Modell mit seinen Datentypen, Operationen und Schnittstellen wird erst auf der konkreten Ebene mit existierenden Protokollen und Formaten in Verbindung gesetzt. Dazu werden bindings definiert, welche die Schnittstellen des Web Services, d.h. die Port-Typen mit konkreten Datenformaten und Protokollen verknüpfen. Zudem werden ports festgelegt, welche konkrete Endpunkte darstellen und eine URL enthalten über die ein binding physikalisch erreicht werden kann. Schließlich sind es die services, die die Funktionen eines Web Services über ein oder mehrere ports eines Port-Typen nach aussen zugänglich machen.

Aufgrund der Aufteilung der Beschreibung eines Web Services in einen abstrakten und einen konkreten Teil, sowie der XML-Basiertheit, bildet WSDL eine flexible, maschinenlesbare Sprache, die nicht auf eine Programmiersprache, ein Übertragungsprotokoll oder ein Datenformat festgelegt ist.

2.3.5 UDDI

UDDI ist die Abkürzung für Universal Description Discovery and Integration. Es handelt sich hierbei um einen Standard für den Aufbau eines Verzeichnisdiensts. Die Hauptbestandteile einer UDDI bilden

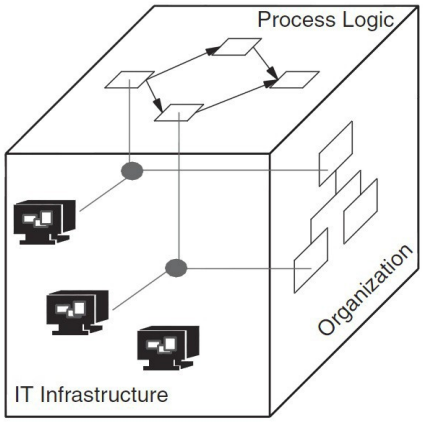


Abbildung 2.3: Die drei Workflow-Dimensionen [LR00]

die White Pages, die Yellow Pages und die Green Pages. Die White Pages enthalten, ähnlich einem Telefonbuch, Informationen über die einzelnen Dienstanbieter, die ihre Dienste im Verzeichnisdienst registriert haben. Eine Kategorisierung der einzelnen Dienste, ähnlich einem Branchenverzeichnis, bilden die Yellow Pages. Die Schnittstellenbeschreibungen der einzelnen Web Services sind in den Green Pages enthalten.

Das in Abbildung 2.1 gezeigte Architektur kann mit Web Services und einem Verzeichnisdienst, das z.B. durch UDDI aufgebaut wird, umgesetzt werden. Der Anbieter eines Dienstes erstellt eine WSDL Beschreibung für seinen Dienst und registriert diesen im Verzeichnisdienst. Der Dienstnutzer können in diesem Verzeichnisdiesnt nach einem passenden Dienst suchen und bei Bedarf die WSDL-Beschreibung eines gefundenen Dienstes anfordern. Durch eine URI, die auf die eigentliche WSDL-Beschreibung verweist kann der gewünschte Dienst eingebunden werden.

2.4 Workflowtechnologie

Ein Workflow ist ein Arbeitsablauf, der eine Menge von Arbeitsschritten umfasst, die in einer bestimmten Reihenfolge teilweise bis komplett automatisiert ausgeführt werden. Die einzelnen Arbeitsschritte werden als Aktivitäten bezeichnet. Zur Erhöhung der Wiederverwendbarkeit und Flexibilität werden oft einzelne Aktivitäten zu Komponenten zusammengefasst. Traditionell werden Workflows zur Auftragsabwicklung im Unternehmensbereich eingesetzt [LR00]. Zunehmend findet die Workflow-Technologie auch im wissenschaftlichen Bereich Einsatz, z.B. bei der Ausführung von komplexen Simulationsabläufen [LR00].

2.4.1 Dimensionen eines Workflows

Nach [LR00] kann die rechnergestützte Ausführung von Workflows mit Hilfe der drei Dimensionen **WHO**, **WHAT** und **WITH** beschrieben werden.

- **WHO**: In dieser Dimension wird beschrieben, welche Mitarbeiter oder Abteilungen innerhalb einer Organisation konkrete Aktivitäten des Workflows erledigen können. Mit Hilfe von Anfragen können geeignete Mitarbeiter bzw. Abteilungen ermittelt werden. Diese Dimension beschreibt die Organisationsstruktur des Unternehmens im Hinblick auf Abteilungen, Rollen und Personen.
- **WITH**: Die WITH-Dimension beschreibt welche IT-Ressourcen (z.B. Programme) benötigt werden, um konkrete Aktiviäten ausführen zu können.
- **WHAT**: In der WHAT-Dimension wird beschrieben, welche Aktivitäten in welcher Reihenfolge ausgeführt werden müssen. Dabei können Aktivitäten sowohl in sequentieller, als auch in paralleler Reihenfolge ausgeführt werden.

In Abbildung 2.3 spannen die Dimensionen WITH, WHAT und WHO einen dreidimensionalen Raum auf. Die Ausführung eines Workflows entspricht in diesem dreidimensionalen Raum einer Abfolge von Punkten. Der Punkt, an dem sich die drei Dimensionen treffen beschreibt, welche einzelne Aktivität mit Hilfe welcher IT-Ressource von welcher Person bzw. Programm ausgeführt wird.

2.4.2 Workflow Management Systeme

Ein Workflow Management System (WfMS) umfasst Anwendungen, welche der aktiven Steuerung von Workflows dient. Dies beinhaltet die Definition, Ausführung und Überwachung der Workflows [HH93]. Die Workflow Management Coalition (WfMC) ist ein 1993 gegründeter Verbund, dessen Ziel es ist, den Einsatz von Workflow Management Systemen zu fördern. Sie definiert ein WfMS folgendermaßen [Hol95]: A system that completely defines, manages and executes “workflows” through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Abbildung 2.4 zeigt die Architektur eines WfMS. Drei Funktionsbereiche machen die Architektur hauptsächlich aus [LR00]:

- **Build-Time**: Dieser Funktionsbereich beinhaltet alle Komponenten, die zur Erstellung und Modellierung von Workflows, sowie zur Verwaltung von Ressourcen verantwortlich sind.
- **Run-Time**: Dieser Funktonsbereich enthält alle Komponenten, die für die Ausführung von Workflow-Instanzen verantwortlich sind.
- **Data-Base**: Dieser Funktionsbereich umfasst alle Komponenten, die sich mit der Datenhaltung für Build-Time- und Run-Time-Daten befassen.

Als weiterer Funktionsbereich wird in [LR00] der Funktionsbereich Meta Model angeführt, welcher eine Unterteilung des Funktionsbereiches Build-Time darstellt. In diesem Bereich werden alle Strukturen definiert, die vom WfMS unterstützt werden.

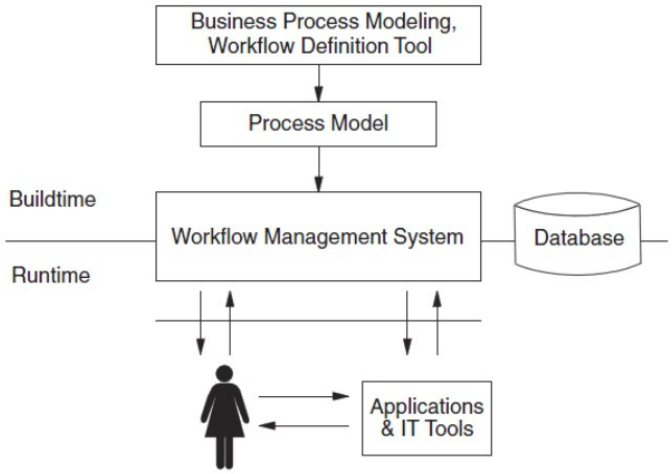


Abbildung 2.4: Die Architektur eines WfMS [LR00]

2.4.3 Workflow-Sprachen

Workflow-Sprachen ermöglichen dem Benutzer die Definition von Workflows. In diesem Abschnitt werden Workflow-Sprachen näher beschrieben. Dazu wird zunächst die Unterteilung der WS-Sprachen in zwei verschiedene Klassen erläutert. Abschließend wird WS-BPEL als Vertreter einer Workflow-Sprache vorgestellt. Der folgende Abschnitt basiert auf [RSM11].

Klassen von Workflow-Sprachen

Bei den Workflow-Sprachen wird oft zwischen zwischen den Kontrollflussorientierten und den Datenflussorientierten Sprachen unterschieden.

Datenflussorientierte Sprachen: Bei den Datenflussorientierten Sprachen liegt der Fokus auf dem Datenfluss, d.h. auf den Datenabhängigkeiten zwischen den Aktivitäten eines Workflows. Der Datenfluss entspricht dabei einem unidirektionaler Datenstrom zwischen den Aktivitäten, die durch Datenabhängigkeiten miteinander verbunden sind. Jede Aktivität besitzt eine Input-Queue, in der eingehende Daten abgelegt werden. Sobald die Input-Queue einer Aktivität mit Daten gefüllt wird, beginnt die Aktivität, ihrer Aufgabe im Workflow entsprechend, mit der Verarbeitung dieser Daten und leitet Ergebnisdaten, dem Datenfluss folgend, an die Input Queues nachfolgender Aktivitäten weiter. Diese verarbeiten ihre Eingabedaten gleichermaßen und leiten ihre Ergebnisse weiter. Datenflussorientierte Sprachen sind insbesondere für datenintensive Workflowsprachen geeignet.

Kontrollflussorientierte Sprachen: Bei den Kontrollflussorientierten Sprachen liegt der Fokus auf dem Kontrollfluss, d.h. der logischen Ausführungsreihenfolge der einzelnen Aktivitäten. Der Kontrolfluss entspricht einem gerichteten, azyklischen Graphen, wobei die Knoten des Graphen den Aktivitäten und die Kanten kausalen Abhängigkeiten zwischen den Aktivitäten repräsentieren. Jeder Knoten kann seine Aktivität nur dann ausführen, wenn die Vorgängerknoten im Graphen die

Ausführung iher Aktivitäten vollständig und erfolgreich abgeschlossen haben. Somit kann jeder Knoten höchstens einmal ausgeführt werden. Knoten können einen weiteren Graphen als Sub-Graphen enthalten, der eine gewisse prozedurale Logik definiert, z.B. eine while-Schleife. In diesem Fall kann eine Aktivität mehrmals zur Ausführung kommen. Der nächste Abschnitt befasst sich mit der kontrollflussoritentierten Sprache WS-BPEL.

WS-BPEL

Die Web Services Business Process Execution Language (WS-BPEL), oft auch nur kurz BPEL genannt, ist eine XML-basierte Sprache zur Beschreibung und Bearbeitung von Workflows [JEA+07]. Sie gilt insbesondere als de-facto Standard zur Implementierung von Geschäftsprozessen mit Hilfe von Web Services (vergleiche Abschnitt 2.3.2).

Bei WS-BPEL handelt es sich um eine kontrollflussorientierte Sprache, dessen Grundgedanke die Orchestrierung von Web Services darstellt. Dazu bietet WS-BPEL die Möglichkeit Web Services in einen Workflow einzubinden, indem Arbeitsschritte des Workflows als Web Service-Implementierungen definiert werden können. Der daraus resultierende Workflow ist ebenfalls ein Web Service und kann als Baustein weiterverwendet werden, d.h. es können neue Workflows aus bereits existierenden Workflows zusammengesetzt werden. Dieses Baukasten-Prinzip ermöglicht es, Workflows einfach zusammenzusetzen bzw. neu anzuordnen. Das erhöht einerseits die Wiederverwendbarkeit der Workflows, andererseits kann dadurch schnell auf neue Gegebenheiten reagiert werden.

WS-BPEL basiert auf den XML-Spezifikationen WSDL (vergleiche Abschnitt 2.3.4), XML Schema (vergleiche Abschnitt 2.1.4), XPath (vergleiche Abschnitt 2.3.2) und XSLT ([C+07]). WSDL wird genutzt um die Schnittstellen zu den WebServices zu beschreiben. Für die Speicherung von Daten verwendet WS-BPEL typisierte Variablen. Eine Variable kann dabei durch drei Typen definiert:

- einfacher oder komplexer XML Schema-Typ simpleType bzw. complexType
- XML Schema Element
- WSDL Message

XPath und XSLT werden zum Datenzugriff und zur Datenverarbeitung genutzt. Die Erweiterbarkeit von WS-BPEL ermöglicht für die Datenmanipulation neben den genannten Sprachen jedoch auch die Nutzung weiterer Sprachen.

Normalerweise werden bei der Beschreibung von Workflows mit WS-BPEL möglichst viele Arbeitsschritte durch Web Services implementiert, so dass bei Bereitstellung der notwendigen Daten, die Ausführung des Workflows vollständig automatisiert durch eine WfMS erfolgen kann. Sprachen wie z.B. BPEL4People [AAD+07] jedoch erweitern WS-BPEL, sodass menschliche Interaktionen als Arbeitsschritte in den Workflow eingebettet werden können.

Aktivitäten bilden die Grundlage für die Definition von Workflows mit Hilfe von WS-BPEL. Sie stellen die einzelnen Arbeitsschritte dar, aus denen sich der gesamte Workflow zusammensetzt. Aktivitäten werden in WS-BPEL in zwei Kategorien aufgeteilt, die Basisaktivitäten und die Strukturierten Aktivitäten.

Basisaktivitäten:
Bei den Basisaktivitäten handelt es sich um einfache Aktivitäten, die keine anderen Aktivitäten enhalten. Diese Aktivitäten werden als eine atomare Operation betrachtet. Zu ihnen zählen unter anderem die Aktivitäten Receive, Reply, Invoke, Assign und Extension:

- **Receive**: Die Aktivität Receive ermöglicht es einem Prozess, bei einer asynchronen Kommunikation mit einem Web Service solange mit dem Start bzw. der Fortführung des Prozesses zu warten, bis eine Nachricht von diesem Web Service eintrifft.
- **Reply**: Mit einer Reply Aktivität kann einem Partner eine Nachricht geschickt werden.
- **Invoke**: Invoke Aktivitäten ermöglichen das synchrone oder asynchrone Aufrufen von Web Services.
- **Assign**:Durch Assign Aktivitäten können ein oder mehreren Variablen Werte zugewiesen werden.
- **Extension**: Eine weitere Aktivität, die nicht unerwähnt bleiben sollte, ist die Extension Aktivität, welche das Hinzufügen benutzterdefinierter Aktivitäten möglich macht.

Strukturierte Aktivitäten:
Bei den Strukturierten Aktivitäten handelt es sich um Aktivitäten, die andere Aktivitäten enthalten können. Sie ermöglichen es durch die Zusammensetzung mehrerer Aktivitäten zu komplexen Ausführungsreihenfolgen den Kontrollfluss eines Prozesses zu beschreiben. Zu den komplexen Aktivitäten gehören unter anderem If, While, For Each, Sequence und Flow Aktivitäten.

- **If**: Durch eine If Aktivität können Workflow-Pfade definiert werden, deren Ausführung an die Erfüllung einer Bedingung gebunden ist.
- **While**: Bei einer While Aktivität wird eine Aktivität so lange ausgeführt werden, bis eine gegebene Bedingung nicht mehr erfüllt ist.
- **For Each**: Eine For Each Aktivität ermöglicht es, den Rumpf einer Schleife so oft auszuführen, bis eine definierte Anzahl von Schleifendurchläufen erreicht ist.
- **Sequence**: Innerhalb einer Sequence Aktivität werden Aktivitäten eines Workflows sequentiell ausgeführt.
- **Flow**: Durch Flow Aktivitäten können Aktivitäten parallel oder einem definierten Ablaufgraphen folgend ausgeführt werden.

In einem BPEL-Prozess werden Web Services aufgerufen. Die Einbindung von Web Services erfolgt in WS-BPEL über *Partner Links*. Für jede Aktivität, die einen Web Service aufruft, ist ein Partner Link definiert. Dieser legt fest, wie mit dem Web Service kommuniziert wird. Partner Links werden mit Hilfe der WSDL-Erweiterung PartnerLinkTypes aus dem Namespace: „plink“ erzeugt. und werden charakterisiert durch einen Namen, einen Typ und ein oder mehreren Roles, die den beiden Parteien des Partner Links ihre Rolle innerhalb der Interaktion zuweisen. Jeder von einer Rolle unterstützte Operation wird durch einen *Port-Typ* (entsprechend der WSDL portTypes) dargestellt. Bei Instanziierung des Prozessmodels wird jeder Role ein konkreter Web Service-Endpunkt zugeordnet.

Listing 2.6 Aufbau eines Partner Links

```
< partnerLinks >
  < partnerLink name = "Fluggesellschaft"
   partnerLinkType = "wsdl:FLugPLT"
   myRole = "VerfugbarkeitAbfragen"
   partnerRole = "FlugService" />
</ partnerLinks >
```

Listing 2.7 Grundstruktur eines BPEL-Dokuments

```
<process name =" meinProzess "> <! -- Prozess -->
 targetNamespace =" anyURI "
 <extensions > <! -- Erweiterungen -->
 </ extensions >
 <imports /> <! -- externe Abhaengigkeiten -->
 < partnerLinks > <! -- Dienste -->
 </ partnerLinks >
 < messageExchanges >
 </ messageExchanges >
  <variables > <! --Daten -->
 </ variables >
 < correlationSets >
 </ correlationSets >
 < faultHandlers > <! -- Fehlerbehandlung -->
 </ faultHandlers >
 < eventHandlers > <! -- Ereignisbehandlung -->
 </ eventHandlers >
 <sequence name =" meineSeq "> <! -- Ablaufsequenz -->
 <receive >
 <assign > <! -- Aktivitaeten -->
 <reply >
 </ sequence >
</ process >
```

WS-BPEL folgt damit dem WSDL-Grundkonzept der Trennung von abstrakten Informationen und konkreten technischen Details, indem die Kommunikation mit Partnern durch abstrake WSDL-Schnittstellen definiert wird, es jedoch keinerlei Informationen darüber liefert, wie die konkreten Bindings aussehen und welche konkreten Services die Prozessinstanzen verwenden [JEA+07].

In 2.6 wird die Interaktion mit einer Fluggesellschaft definiert. Der BPEL-Prozess kann über diesen Partner Link die Verfügbarkeit eines Fluges abfragen (myRole) und erwartet eine Antwort von der Fluggesellschaft (partnerRole).

In Listing 2.7 ist beispielhaft die Grundstruktur für einen BPEL-Dokument abgebildet. Als Wurzelelement besitzt jedes BPEL-Dokument das Element *Process*. Dieser Process-Container ist das äußerste Konstrukt eines BPEL-Dokuments und enthält einen Definitionsteil und die Aktivitäten.

In den ersten beiden Zeilen des Beispiels wird der Name des BPEL-Dokuments und der target-Namespace für den Prozess definiert. Diese Definitionen sind für die Ausführung des Workflows erforderlich.

WS-BPEL ist blockstrukturiert, d.h. ein BPEL-Dokument kann eine verschachtelte Struktur von Scope-Elementen enthalten, die lokale Aktivitäten, Variablen, Partner Links und verschiedene Handler definiert. Durch *Scope*-Elemente können Gültigkeitsbereiche für z.B. Variablen und Partner Links definiert werden. Alle in einem Scope enthaltenen Aktivitäten bilden eine transaktionale Einheit. Als Handler stehen *Event Handler*, *Compensation Handler* und *Fault Handler* zur Auwahl. Diese können Scope-Elementen zugeordnet werden und enthalten Aktivitäten, die bei einer Aktivierung des Handlers ausgeführt werden. Mit Hilfe von Event Handlern kann auf auf auftretende Ereignisse reagiert werden. Fault Handler ermöglichen eine Fehlerbehandlung, die auf bei der Ausführung von Aktivitäten auftretende Exceptions reagieren kann. Compensation Handler hingegen können die Auswirkungen bereits abgeschlossener Aktivitäten zwar nicht rückgängig machen, sie können sie jedoch kompensieren, indem entsprechende Kompensationshandlungen ausgeführt werden. Diese Kompensationshandlungen ermöglichen lang-andauernde Transaktionen.

Die wesentlichen Vorteile von BPEL bei der Modellierung und Ausführung von wissenschaftlichen Workflows sind der modulare Aufbau, die Flexibilität im Umgang mit generischen XML-Datentypen und der späten Anbindung von Diensten an den Workflow, sowie Möglichkeiten zu Fehler- und Ereignisbehandlung und Fehlerkompensation. Für ein tiefgehenderes Studium der syntaktischen Elemente von WS-BPEL wird auf die Quelle [ACD+03] und [JEA+07] verwiesen.

2.4.4 Klassifizierung von Workflows

Folgende Abschnitte befassen sich mit den verschiedenen Arten von Workflows und basieren auf [Wag11] und [RSM11] soweit nicht anders angegeben. In Abbildung 2.5 erfolgt eine Klassifizierung von Workflows nach ihrer Datenintensität und danach, welche Probleme sie lösen. Zu den *Orchestration Workflows* gehören Business Workflows, die Geschäftsprozesse durch die Verknüpfung verschiedener, heterogener Anwendungen realisieren bzw. automatisieren [LR00]. Auch *Simulation Management Workflows*, die auf ähnliche Weise die Interaktionen mit Simulationsanwendungen und Ressourcen koordinieren, die für Simulationsprozesse Berechnungen durchführen und Daten verwalten [GSK+11], gehören zur Gruppe der *Orchestration Workflows*. Hauptaugenmerk *Datenintensiver Workflows* hingegen ist es, große Mengen an Daten verarbeiten zu können. Dabei können die Daten heterogen sein und verteilt vorliegen. Die Verarbeitung der Daten kann entweder auf externen Ressourcen oder innerhalb des Workflows erfolgen. Die interne Verarbeitung macht das Laden der Daten in den Kontext des Workflow erforderlich.

Zu den datenintensiven Workflows gehören *Data Analysis Workflows*, *Data Modeling Workflows* und *ETL Workflows*. Mit Hilfe von *Data Analysis Workflows* können Daten visualisiert oder Erkenntnise aus der Analyse von Daten gewonnen werden. *Data Modeling Workflows* ermöglichen die Erzeugung von Modellen um gewisse Problemstellungen zu beschreiben oder bestimme Muster innerhalb dieser Modelle aufzudecken. Die dritte Untergruppe von datenintensiven Workflows sind die *ETL-Workflows*. Sie versorgen Anwendungen oder andere übergeordnete Workflows mit Daten, indem Extraktions-, Transformations- und Lade-Prozesse durchgeführt werden.

Zur Gruppe der *Scientific Workflows* werden *Simulation Management Workflows*, *Data Analysis Workflows* sowie *Data Modeling Workflows* zusammengefasst. Im nächsten Abschnitt werden die Business Workflows, die Scientific Workflows und die ETL-Workflows näher beschrieben.

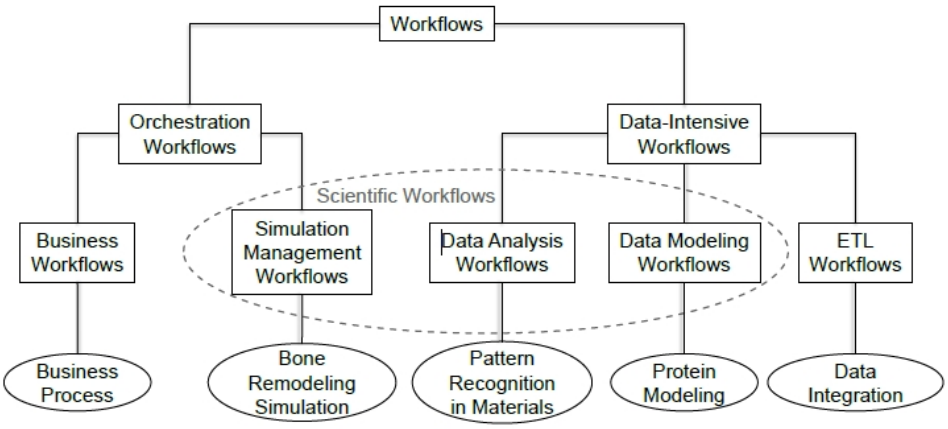


Abbildung 2.5: Klassifizierung von Workflows nach [RSM11]

- **Business Workflows**: Business Workflows werden verwendet, um häufig auftretende Arbeitsabläufe im Unternehmensbereich zu modellieren und automatisieren [LR00]. Beispiele für solche Workflows sind der Versand von Waren nach dem Eingang von Zahlungen oder die Bearbeitung eines Kreditantrages im Bankenbereich. Business Workflows basieren somit auf Entscheidungen und dazugehörigen Handlungen. Da oft nur relativ kleine Datenmengen verarbeitet werden müssen, ist der Einsatz von Kontrollflussorientieren Workflowsprachen hier geeignet. Weiterhin bietet sich aufgrund der geringen Laufzeiten der Workflows die parallele Ausführung dieser Workflows an.
- **Scientific Workflows**: Workflows finden auch im wissenschaftlichen Umfeld verstärkt Einsatz [TDG+07]. Solche wissenschaftlichen Workflows werden, wie oben bereits erwähnt, unter dem Begriff Scientific Workflows zusammengefasst [RSM11]. Nach [GSK+11] bietet der Einsatz von Workflows in der Wissenschaft folgende Nutzen:
  - Workflows unterstützen die Wissensweitergabe, indem sie dem Wissenschaftler-Team als Dienst zur Verfügung stehen.
  - Mit Hilfe von Workflows wird eine Analyse von Ergebnissen durch die Community ermöglicht.
  - Workflows können mit großen Datenmengen umgehen (z.B. Daten von Sensoren).
  - Workflows können in heterogenen und verteilten Umgebungen ausgeführt werden (Unterstützung verschiedener Plattformen und Programmiersprachen). Dies sind Bedingungen, unter denen wisschenschaftliche Berechnungen oft ausgeführt werden.
  - Durch die Automatisierung der Schritte während des Workflow-Designs und ihrer Ausführung, können sich Wissenschaftler auf die Lösung ihrer eigentlichen, wissenschaftlichen Probleme konzentrieren.

  - Durch Workflows können wissenschaftliche Simulationen parallel und automatisiert durchgeführt werden.

Weniger als Entscheidungen und daraus resultierende Handlungen, stehen bei Scientific Workflows die rechnerunterstützte Ausführung von Simulationen und Berechnungen und die damit einhergehende Verwaltung großer Datenmengen im Vordergrund [Wag11]. Der Einsatz von verschiedensten Programmen zur Berechnung und Analyse von Simulationsdaten resultiert in einer großen Zahl von Datenformatskonvertierungen und einem regen Datentransfer.

Mit Hilfe der Workflow-Technologie können diese Arbeitsabläufe modelliert werden. Dabei können Daten entweder in den Workflow-Kontext geladen und innerhalb des Workflows zur Verfügung gestellt oder an externe Programme (z.B. Web Services) für die Ausführung von Berechnungen und Analysen übergeben werden. Obwohl sich aufgrund des hohen Datenverkehrs hier datenflussorientierte Workflows besonders anbieten, hat sich die kontrollflussorientierte Workflowsprache WS-BPEL (vgl. Abschnitt Abschnitt 2.4.3) durchgesetzt. Dies begründet sich durch eine weite Verbreitung von WS-BPEL als Standard-Sprache bei der Modellierung und Orchestration von Workflows und der Möglichkeit, Datenfluss innerhalb von WS-BPEL einbetten zu können [Slo07] [GHCM09].

Häufig wird der Begriff *Simulationsworkflow* als Synonym für oben erwähnte Simulation Management Workflows verwendet (siehe Abbildung 2.5). In [GSK+11] z.B. werden Simulationsworkflows als eine Teilmenge der Scientific Workflows beschrieben, die typischerweise als Kompositionen langlaufender, numerischer Berechnungen zur Realisierung mathematischer Simulationsmodelle umgesetzt werden, (z.B. basierend auf partiellen Differentialgleichungen). Im weiteren Verlauf dieser Arbeit wird ebenso der Begriff Simulationsworkflow statt Simulation Management Workflow verwendet.

- **ETL Workflows**: Extract Transfer Load (ETL) Operationen werden verwendet, um von verschiedenen Ressourcen (z.B. Data Marts) Daten zu extrahieren und diese dann Anwendungen und Programmen (z.B. Data Warehouses) zur Verfügung zu stellen. Mit Hilfe von ETL Workflows (ETL) können solche ETL Operationen orchestriert und ausgeführt werden. Dabei werden diese Operationen durch Aktivitäten innerhalb des Workflows dargestellt, die es ermöglichen, Daten von Datenressourcen zu laden (LOAD), Daten von verschiedenen Datenmengen zu verknüpfen (JOIN), zu vereinen (UNION) oder zusammenzuführen (MERGE).

  Workflows, die es ermöglichen, direkt und nahtlos auf beliebige, externe Datenquellen zuzugreifen und ETL Operationen auszuführen, werden im SIMPL-Rahmenwerk [RRS+11] (vergleiche Kapitel 3) beispielsweise durch die Erweiterung von WS-BPEL um sogenannte Datenmanagement-Aktivitäten realisiert. So kann beispielsweise der Inhalt einer Tabelle aus einer relationalen Datenbank direkt abgefragt oder der Inhalt einer Datei aus einem Dateisystems direkt in den Workflow-Kontext geladen werden. Eine nähere Betrachtung des SIMPL-Rahmenwerks findet in Kapitel Kapitel 3 statt.

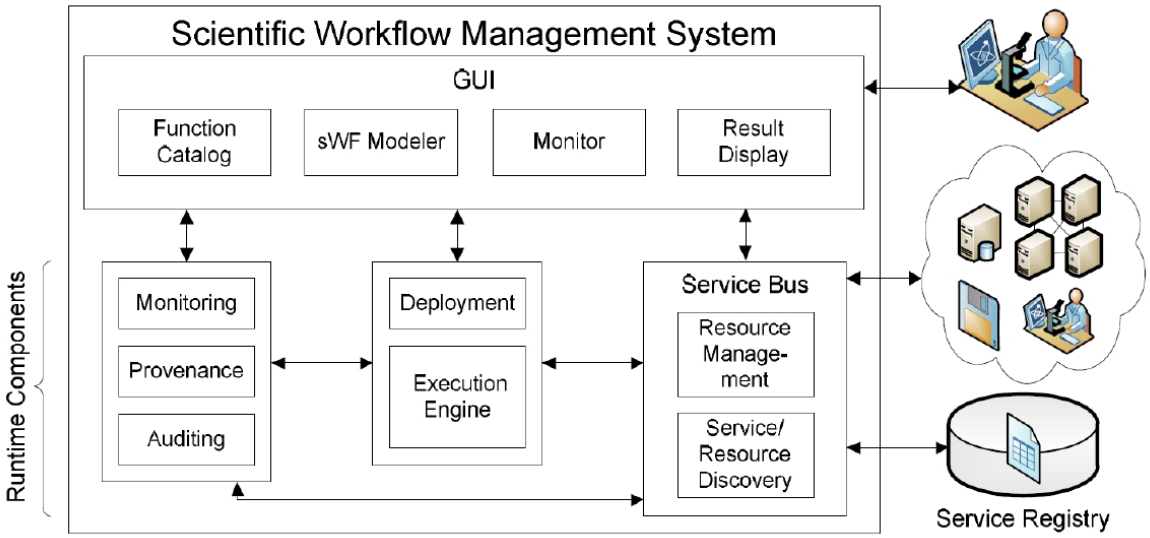


Abbildung 2.6: Architektur eines sWFMS [GSK+11]

2.4.5 Scientific Workflow Management Systeme

In AbschnittAbschnitt 2.1.4 wurde die Architektur eines WfMS beschrieben. Scientific Workflows verwenden eine Architektur, die eine Erweiterung der Architektur eines WfMS darstellt, ein Scientific Workflow Management System (sWfMS). Die in Abbildung 2.1 dargestellte Architektur basiert auf der in [LR00] definierten Workfowtechnolgie für Business und Production Workflows. Im folgenden Abschnitt wird die Architektur etwas näher beschrieben. Bei den Komponenten wird unterschieden zwischen Komponenten für das Graphical User Interface (GUI) und den Laufzeitkomponenten.

Zu den **GUI-Komponenten** gehört der *Scientific Workflow Modeler (sWF Modeler)*, mit dessen Hilfe Workflows modelliert und Deployment-Informationen erzeugt werden. Eine weitere Komponente ist der *Function Catalog*, welcher eine Liste von verfügbaren Diensten und eine Menge von Funktionen bereitstellt, die bei der Modellierung eines Workflows eingesetzt werden können. Die *Monitor Component* bietet dem Nutzer die Möglichkeit zur Überwachung der Ausführung von Workflow-Instanzen und Erkennung unerwarteter Ereignisse bzw. Fehler während der Ausführung. Durch das *Result-Display* werden dem Benutzer die Zwischen- und Endergebnisse der ausgeführten Workflows in einem geeigneten Format zur Verfügung gestellt.

Zu den **Laufzeit-Komponenten** gehört die *Deployment Component*, welche die Workflow-Modelle in interne Repräsentationen abbildet und sie auf der Execution Engine installiert, welche für die Ausführung von Workflow-Instanzen verantwortlich ist. Eine weitere Komponente ist die *Auditing Component*, die alle Ereignisse aufzeichnet, die während der Ausführung von Workflows und Aktivitäten auftreten, z.B. den Startzeitpunkt eines Workflows. Die *Monitor Component* nutzt diese Informationen um den aktuellen Status von laufenden Workflows anzuzeigen. Detaillierte Informationen, die über die Auditing Informationen hinausgehen und benötigt werden, um Ausführungen

von Workflows reproduzieren zu können, werden von der *Provenance Component* aufgezeichnet. Der *Service Bus* sucht und selektiert passende Dienste, welche die Aktivitäten eines Workflows implementieren. Desweiteren ist er für die Nachrichtenzustellung, die Durchführung von Datentransformationen und die Anbindung von externen Ressourcen an den Worfklow verantwortlich. Alle Metadaten über externe Ressourcen und Dienste werden von der *Resource Management* verwaltet. Die *Resource Discovery* benutzt diese Metadaten oder externe Verzeichnisse um eine Liste zu erstellen, die alle für den Nutzter in Frage kommenden Dienste und Ressourcen beinhaltet. Im Falle eines Fehlers bei der Ausführung können mit Hilfe dieser Liste alternative Dienste oder Ressourcen ausgewählt werden.

2.5 Simulationen und FEM

In diesem Abschnitt werden zunächst Simulationen im Allgemeinen erklärt, indem Begriffe wie Simulation, Multisimulation und Simulationsrahmenwerke definiert werden (Abschnitt 2.5.1). Danach wird in Abschnitt 2.5.2 mit der Finite Element Methode ein numerisches Verfahren zur Lösung von partiellen Differentialgleichungen vorgestellt, welches bei der Knochensimulation Einsatz findet.

2.5.1 Simulationen

Der folgende Abschnitt basiert auf [Dor11]. Andere Quellen werden explizit angegeben. In Abschnitt Abschnitt 2.4.4 wird erklärt, dass Scientific Workflows unter anderem die Ausführung von Simulationen umfassen. Im Folgenden wird der Begriff einer Simulationen, einer Multi-* Simulation und eines Simulationsrahmenwerkes geklärt.

Definition Simulation

Es gibt viele verschiedene Definitionen des Begriffes Simulation, die aus vielen verschiedenen Fachbereichen stammen und somit verschiedene Aspekte in den Mittelpunkt setzen. [Har96] liefert folgende abstrakte Definiton:

„ *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.*“

Simulation ermöglichen es demnach, das Verhalten eines Systems oder Prozesses nachzuahmen. Dabei spielt die Verbindung zwischen Modellen und Simulationen eine große Rolle. Die Ausführung einer Simulation basiert auf der Erstellung eines validen Modells, das parametrisiert wird, bevor der Simulationslauf gestartet wird. Durch Ausführung von Experimenten an dem Modell können dann Erkenntnisse über das abgebildete System gewonnen werden. Ein valides Modell ist nach [Sta73] stets das Abbild eines natürlichen oder künstlichen Originals, wobei das Original selbst wieder ein Modell sein kann. Zudem werden in einem validen Modell nicht alle Attribute des dargestellten Originals erfasst, sondern nur vom Modellierer ausgewählte Attribute des Originals. Des Weiteren sind Modelle

nicht nur ein Modell von etwas, sondern erfüllen für jemanden einen bestimmten Zweck innerhalb eines Zeitintervalls.

Nach [Har96] können mit Hilfe von Simulationen Details über Systeme ermittelt werden, die sonst nicht experimentell erfasst werden können (z.B. die Evolution von Galaxien); Experimente ersetzt werden, die experimentell nicht untersucht werden können (z.B. die Enstehung von Galaxien); heuristisch Hypothesen und Theorien entwickelt werden, die Durchführung von Experimenten unterstützt und reale Prozesse veranschaulicht und verständlich gemacht werden.

Multi-* Simulation

Dieser Abschnitt basiert auf [Rei]. Multi-* Simulationen sind Simulationen, die verschiedene wissenschaftliche Bereiche kombinieren. Dabei werden mindestens zwei der folgenden Bereiche kombiniert:

- **Multi-Domänen**: Diese Simulationen verwenden Methoden, die aus verschiedenen Fachbereichen stammen, beispielsweise werden Modelle aus der Mechanik und der Biologie kombiniert.
- **Multi-Skalen**: Für die Simulation kommen unterschiedliche Zeit- und /oder Raumskalen zum Einsatz.
- **Multi-Physiken**: Unterschiedliche physikalische Gesetze kommen in der Simulation zum Einsatz.
- **Multi-Tools**: Im Rahmen der Simulation werden unterschiedliche Tools verwendet.

Die Simulation realer Objekte oder Prozesse bietet sich oft für den Einsatz von Multi-* Simulationen an, da hier häufig die Verknüpfung von Simulationsmodellen verschiedener Bereiche erforderlich ist. Erst durch die Kombination verschiedener Simulationsmodelle innerhalb eines Simulationsworkflows können realere Simulationen mit präziseren Ergebnissen erzielt werden [K$^+$13].

Für die Erstellung geeigneter Simulationsmodelle muss die Modellphysik durch Definition aller relevanten physikalischen Gesetze festgelegt werden. Diese physikalischen Gesetze basieren dabei auf mehreren, kontinuierlichen Raum- und Zeit-Dimensionen und werden häufig durch Differentialgleichungen dargestellt. Da kontinuierliche Informationen jedoch nur äußerst schwer von Computern verarbeitet werden können, findet eine Umwandlung der Differentialgleichungen in lineare Gleichungssysteme statt. Desweitern werden die kontinuierlichen Dimensionen in Teilabschnitte unterschiedlicher Granularität unterteilt. Insgesamt wird mit Hilfe dieser das kontinuierliche Verhalten des modellierten Systems diskretisiert. Die Granularität der Teilabschnitte einer Dimensionen wiederum wird über die Auswahl geeigneter geigneter Raum- und Zeitskalen festgelegt. Beispielsweise können für die Aufteilung der Dimension Zeit unterschiedliche Skalen ausgewählt werden, sodass ein Teilabschnitt der Simulation wenige Nano-Sekunden bis hin zu mehreren Jahren umfasst.

Simulationsrahmenwerke

Der folgende Abschnitt basiert aud [Mül10], [FS] und [RRS+11]. Simulationsrahmenwerke dienen der rechnergestützten Ausführung von Simulationen. Beispiele für Simulationsrahmenwerke sind PANDAS oder SIMPL.

PANDAS ist eine Abkürzung für Porous Media Adaptive Nonlinear finite element solver based on Differential Algebraic Systems und stellt ein Simulationsrahmenwerk für Berechnungen bei porösen Medien dar. Das Problem der porösen Medien tritt in verschiedensten Engineeringbereichen auf. Modelle für poröse Modelle sind beispielsweise das interaktive Verhalten eines verformten Skeletts. Desweiteren können chemische und elektrochemische Phänomene in die Simulationsmodelle mit einfließen. Die Problemlösung basiert dabei auf der Finiten Element Methode, welches im folgenden Abschnitt erläutert wird. Dem Benutzer stehen sowohl eine interaktive, als auch eine Batch-Anwendung zur Verfügung. Weiterhin können Ergebnisse online visualisiert werden.

Das SimTech-Information Management, Processes and Languages (**SIMPL**) ist ein erweiterbares Rahmenwerk, welches für Simulationsworkflows eine generische Schnittstelle für Datenmanagement und -bereitstellung zur Verfügung stellt. Dadurch ermöglicht es die nahtlose Einbindung externer, heterogener Datenquellen in Simulationsworkflows. SIMPL abstrahiert über heterogenen Datenressourcen, indem es ihre Schnittstellen durch logische Deskriptoren, generische Datenzugriffsoperationen und Metadaten zur Abbildung auf konkrete Datenzugriffsmechanismen, vereinheitlicht. Durch eine Erweiterung der Workflowsprache WS-BPEL, können zusätzliche BPEL-Aktivitäten für das Datenmanagement und die Datenbereitstellung innerhalb Simulationsworkflows definiert werden. Eine weitere Abstraktion sind die Data Management Patterns, welche beispielsweise die Definition von ETL-Operationen ermöglichen. Eine genauere Betrachtung des SIMPL-Rahmenwerks findet in Kapitel Kapitel 3 statt.

2.5.2 Finite Element Methode FEM

Folgender Abschnitt basiert auf [Ari12], [Dor11] und [ZT05]. Bei der Finite Elemente Methode (FEM) handelt es sich um ein numerisches Verfahren zur Lösung von partiellen Differentialgleichungen. Da viele Simulationsmodelle aus dem wissenschaftlichen Bereich auf Differentialgleichungen basieren, wird FEM oft zur Lösung unterschiedlichster Problemstellungen verwendet. Das FEM-Verfahren wird beispielsweise in Knochensimulationen (vergleiche 4.1) verwendet, um die Verformung einer Knochenstruktur bei unterschiedlichen Belastungen auf den Knochen zu simulieren. Je nach Problemstellung kommen verschiedene Ansätze zum Einsatz, beispielsweise der Ritz-Ansatz bei Körperverformungen oder der Galerkin-Ansatz für zeitabhängige Probleme. Beim Galerkin-Ansatz wird die Lösung für ein Grundproblem, also das zu simulierende Modell, durch eine endliche Anzahl von sogenannten Elementen angenähert. Elemente sind die Teilprobleme, die durch Aufteilung des Grundproblems erzeugt werden. Aus der endlichen Anzahl der Elemente leitet sich der Name Finite Elemente Methode ab.

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

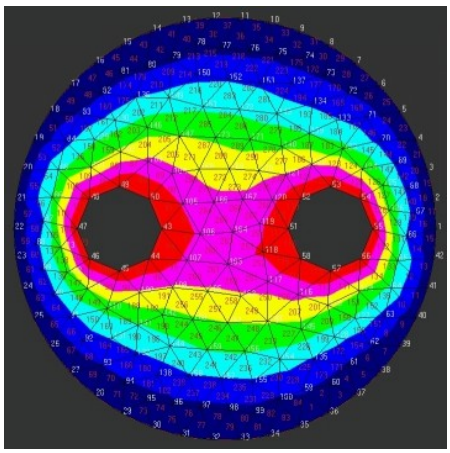


Abbildung 2.7: Aufteilung des Gesamtmodells in Elemente [3]

Alle Elemente werden mit Hilfe eines numerischen Verfahrens innerhalb eines Gitters miteinander vernetzt und besitzen eine einfache, geometrische Grundform, beispielweise Drei- oder Rechtecke. Dieses Netzwerk nähert das zu simulierende Modell an, indem die Elemente in endlichen Zeitschritten t_0 bis t_n diskretisiert werden. Zur Festlegung der Reihenfolge der Berechnungen werden alle Elemente innerhalb des Gitters eindeutig nummeriert. An bestimmten Stellen im Gitter, den sogenannten Knotenpunkten, werden Gleichungen aufgestellt. Der Zustand eines Elements zum Zeitpunkt t_i kann somit durch eine Matrix von Gleichungen, die sich auf dieses Element beziehen, beschrieben werden. Die globale Matrix aller Gleichungen stellt schließlich den Zustand des gesamten, zu simulierenden Modells zum Zeitpunkt t_i dar. Mit Hilfe eines iterativen Verfahrens können dann die Zeitschritte t_0 bis t_n berechnet werden. In Abbildung 2.7 ist als Beispiel ein Modell abgebildet, das in Elemente aufgeteilt wurde, welches die Grundform eines Dreiecks besitzen. Für ein tiefergehendes Verständnis des FEM-Verfahrens wird als Lektüre auf [ZT05] verwiesen.

2.6 Ontologien

Der folgende Abschnitt basiert auf [Stu09]. Eine Ontologie ist eine abstrakte Sicht auf Dinge und deren Beziehungen untereinander und ermöglicht es, das Wissen über einen bestimmten Weltausschnitt in vereinfachter Form zu beschrieben. Eine häufig zitierte Definition für Ontologien lautet [Gru93]: „An ontology is an explicit specification of a conceptualisation“ [Gru93] Bei einer Konzeptualisierung handelt es sich hier um Beziehungen zwischen Objekten, Konzepten und anderen Gegenständen einer gemeinsamen Domäne. Aufgrund ihrer Eigenschaften, eignen sich Ontologien auch für die Darstellung und Verarbeitung durch Informations-Systeme. Hierfür müssen Ontologien durch Modellierungssprachen definiert werden. Eine der der wichtigsten Modellierungssprachen, welche insbesondere im Bereich des Semantic Web als Standard gilt, ist die Web Ontology Language (OWL). OWL basiert auf dem Resource Description Framework (RDF) und RDF-Schema und nutzt die XML-Syntax. Sie ist vom World Wide Web Consortium (W3C) definiert worden. Das RDF-Schema

basiert auf sogenannten RDF-Tripeln, welche Ressourcen einer Domäne und deren Beziehungen beschreiben. Diese Tripel bestehen aus Subjekt (die zu beschreibende Ressource), Prädikat (Eigenschaft der Ressource) und Objekt (Wert des Prädikats). Um mit Hilfe von Owl genauere Aussagen über Ressourcen formulieren zu können, ist ein entsprechendes Vokabular erforderlich. Die Hauptkonstrukte des Vokabulars sind z.B.**owl:class**, **owl:subClassof**, **owl:thing**, **owl:ObjectProperties**, **owl:SubObjectPropertyOf**, **owl:DataObjectProperties**. Mit *owl:class* können Klassen definiert werden und mit *owl:subClassof* können Klassen als Unterklassen von bestehenden Klassen deklariert und somit Klassenhierarchien aufgebaut werden. Mit Hilfe von *owl:ObjectProperties* können Beziehungen zwischen den Individuen von Klasse und mit *owl:SubObjectPropertyOf* ebenso Hierarchien von Beziehungen definiert werden. Individuen einer Klasse werden mit *owl:thing* definiert, wobei die Zuordnung zu einer Klasse über *rdf:type* festgelegt wird. Neben den Beziehungen zwischen Individuen von Klassen können zusätzlich Beziehungen zwischen Individuen von Klassen und Daten (z.B. Literalwerten für Strings oder Integer) definiert werden. Ferner können komplexere Operationen auf den Klassen angewandt werden, wie z.B. Vereinigung, Schnitt oder Komplementärmenge. Für ein tiefergehendes Studium von Owl wird auf die Literatur [G+12] verwiesen.

3 Das SIMPL-Rahmenwerk

In diesem Kapitel wird das SIMPL-Rahmenwerk vorgestellt, welcher als Grundlage für in diese Arbeit betrachteten Workflows dient. Dafür wird zunächst im Abschnitt 3.1 die Architektur des SIMPL-Rahmenwerks vorgestellt. Anschliessend wird in Abschnitt 3.2die im Rahmen des SIMPL-Rahmenwerks erstellte Worklflow-Sprache BPEL-DM beschrieben. Im Abschnitt 3.3 wird abschliessend das dazugehörende Konzept der *Data Management Patterns* erläutert und die Transformation dieser Patterns in ausführbare Workflow-Fragmente beschrieben. Die folgenden Abschnitte basieren auf [RRS+11], [RS14] und [RSM14b].

3.1 Architektur des SIMPL-Rahmenwerks

Bei der Ausführung von Simulationen werden grosse Mengen von Daten mit und zwischen den Simulationsanwendungen ausgetauscht. Dabei werden oft unterschiedliche, gegebenenfalls proprietäre Dateiformate aus verschiedenen, heterogenen Datenquellen verwendet, die entsprechende Datentransformationen in ein für die Simulation geeignetes Format erfordern. Dies resultiert in einem Mehraufwand für den Wissenschaftler, das insbesondere bei komplexen Simulation ins Gewicht fällt. Eine adäquate Datenverwaltung und -bereitstellung sind notwendig, um mit der bei Simulationen anfallenden Informationsflut umgehen zu können.

Das SimTech-Information Management, Processes and Languages (SIMPL) ist ein erweiterbares Rahmenwerk, welches in Simulationsworkflows die nahtlose Einbindung beliebiger, externer Datenquellen ermöglicht, indem es einen einheitlichen Zugriff erlaubt. Hierfür stellt SIMPL eine generische Schnittstelle für das Management und die Bereitstellung von Daten zur Verfügung. Für die Definiton des Datenmanagements bietet SIMPL dem Modellierer zusätzliche Workflow-Aktivitäten, sowie sogenannte Data Management Patterns (vergleiche Abschnitt 3.3)zur Definition von typischen, komplexeren Aktivitäten der Datenbereitstellung, wie z.B. ETL-Operationen.

Das SIMPL-Rahmenwerk stellt eine Erweiterung einer sWfMS dar (vergleiche Abschnitt 2.4.5). Abbildung 3.1 zeigt, wie das SIMPL-Rahmenwerk konkret die Architektur eines sWfMS erweitert, wobei nur die Elemente einer sWfMS dargestellt werden, die für SIMPL relevant sind:

- Die Komponente **SIMPL Core** ist im Service Bus eingebettet und stellt einheitliche, logische Schnittstellen für den Zugriff auf heterogene Datenquellen zur Verfügung.
- Die Komponente Simulation Artifact Registry wurde erweitert um **Metadaten**, die die Abbildung der einheitlichen Schnittstellen auf konkrete Zugriffsmechanismen definieren.

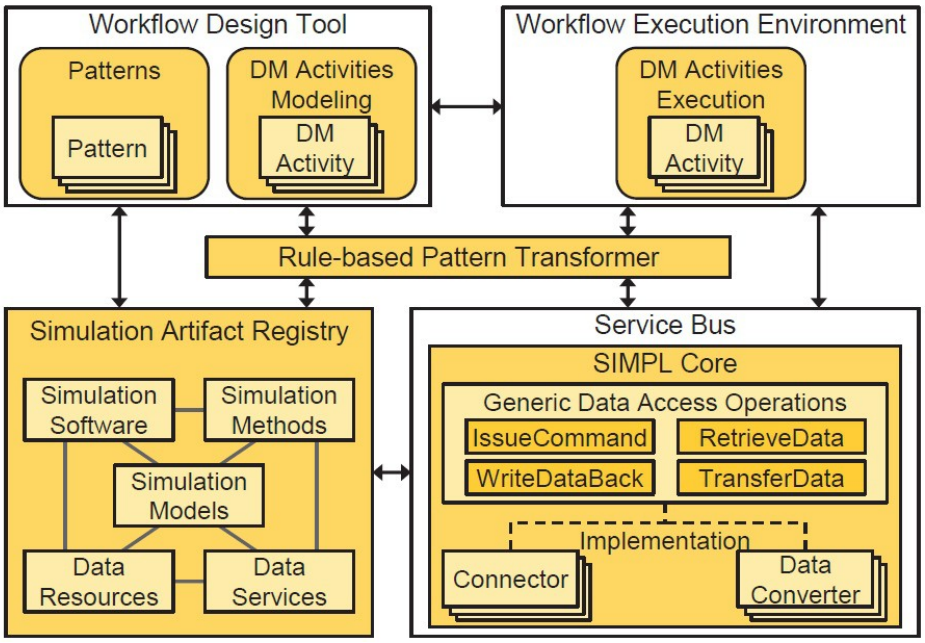


Abbildung 3.1: Die Integration des SIMPL-Rahmenwerks in die sWfMS-Architektur [RS14]

- Das **DM Activities Modeling Plugin** und **DM Activities Execution Plugin** erweitern das Workflow Design Tool bzw. die Workflow Execution Environment des sWfMS. Während das DM Activities Modeling Plugin zusätzliche Data Management Aktivitäten (vergleiche Abschnitt 3.2) bereitstellt, ist das DM Activities Execution Plugin für die Ausführung dieser zusätzlichen Aktivitäten verantwortlich. Diese neuen Aktivitäten können entweder direkt in den Simulationsworkflow eingefügt oder innerhalb ETL Workflows eingebettet werden.
- Das **Patterns Plugin** ist eine weitere Erweiterung des Workflow Design Tools, die den Modellierer des Workflows bei der Definition der Data Management Operationen durch die Bereitstellung von abstrakten, sogenannten Data Management Patterns (vergleiche Abschnitt 3.3) unterstützt. Diese Patterns ermöglichen die Modellierung von typischen Aufgaben der Datenbereitstellung in Simulationsworkflows.
- Für den Einsatz von Data Management Patterns enthält das SIMPL Rahmenwerk einen **regelbasierten Pattern Transformer**. Diese Komponente greift auf die **Simulation Artifact Registry** zu, um Data Management Patterns in ausführbare Workflow-Fragmente umzuwandeln.

3.2 BPEL-DM

Die Workflow-Sprache BPEL (vergleiche Abschnitt 2.4.3) ist ein de-facto Standard für die Definition und Ausführung von Geschäftsprozessen. BPEL wird auch für die Modellierung und Ausführung von

wissenschaftlichen Workflows vorgeschlagen [AMA06]. Aufgrund dieser Vorzüge von BPEL wurde die Business Process Execution Language extension for Data Management (**BPEL-DM**) definiert, die BPEL um weitere Typen von Aktivitäten erweitert. Diese neuen Aktivitäten werden als *Data Management Aktivitäten* (DM) bezeichnet und repräsentieren Aufgaben im Workflow, die Operationen auf Datenquellen beinhalten. Dabei ermöglichen sie einen direkten und nahtlosen Zugriff von einem Simulationworkflow aus auf externe Datenquellen.

Die grundlegenden Aktivitätstypen von BPEL-DM sind *IssueCommand*, *RetrieveData*, *WriteDataBack* und *TransferData*. Diese Aktivitäten rufen den SIMPL-Core auf und senden ihm die auszuführende Datenoperation. Der SIMPL-Core übernimmt dann den Zugriff und die Ausführung der Operation auf der heterogenen Datenquelle.

Im weiteren Verlauf dieser Arbeit wird für ein System, welches Daten speichern und verwalten kann, der Begriff *Datenquelle* verwendet. Das sind beispielsweise Datenbanken oder Dateisysteme. Die eigentlichen Operationen, die auf der Datenquelle ausgeführt werden, werden als *DM commands* bezeichnet. Beispiele sind SQL-Befehle oder Shell-Kommandos.

Als Eingabeparameter erhalten die DM Aktivitäten eine spezielle BPEL-Variable *data source reference variable*, welche die Datenquelle referenziert, auf der die eigentliche Anweisung ausgeführt werden soll. Solch eine Referenz ist ein *logical data source descriptor*, der entweder den logischen Namen der Datenquelle oder eine (nicht)-funktionale Anforderungsbeschreibung beinhaltet. Durch den logischen Namen einer Datenquelle wird genau eine Datenquelle referenziert. Die Beschreibung dieser Datenquelle ist im Resource Management hinterlegt. Mithilfe einer Anforderungsbeschreibung kann eine geeignete Datenquelle erst zur Laufzeit gesucht und eingebunden werden.

Jede Datenquelle verwaltet mehrere Datencontainer (*data container*). Diese entsprechen referenzierbaren Sammlungen Daten, beispielsweise Tabellen in einer relationalen Datenbank oder Dateien in einem Dateisystem. Mithilfe von *data container reference variables* können einzelne Container über logische Namen referenziert werden. Das Resource Management bildet die logischen Namen auf ihre konkreten Identifikatoren ab. *Data set variables* dienen als Zielcontainer, um Daten in einen Workflow zu laden. Der Aufbau dieser Variablen wird durch XML Schema Definitionen festgelegt und berücksichtigt die Unterschiede der verschiedenen Datenquellen. Beispielsweise werden tabellenbasierte Daten, wie Daten in relationalen Datenbanken oder CSV-basierten Dateien, in einer *XML RowSet* Struktur gespeichert.

Im folgenden Abschnitt werden die einzelnen DM Aktivitäten näher beschrieben.

- **IssueCommand**: Die IssueCommand Aktivität kann verwendet werden um Daten zu manipulieren und zu definieren, d.h. Datenstrukturen, wie z.B. die Struktur einer Datenbanktabelle, zu erstellen, zu verwerfen oder zu ändern. Neben einer Referenz auf eine Datenquelle erwartet diese Aktivität als Eingabeparameter einen Dm command, welche auf der spezifizierten Datenquelle ausgeführt wird. Die Execution Engine erwartet eine Benachrichtigung darüber, ob die Operation mit den Erfolg bzw ohne Erfolg abgeschlossen wurde. Bei einem Erfolg wird die Ausführung des Workflows gemäß seines Kontrollflusses fortgesetzt. Bei einem Misserfolg kann eine Fehlerbehandlung bzw. Fehlerkompensation gestartet werden.

- **RetrieveData**: Die RetrieveData Aktivität dient zum Laden von Daten. Sie erwartet genau wie die IssueCommand Aktivität einen DM command als Eingabeparameter. Diese Anweisung muss Daten generieren, wie z.B. ein SELECT-Befehl oder ein Dateipfad. Bei erfolgreicher Ausführung der Anweisung werden die generierten Daten an die Execution Engine zurückgeschickt. Ein weiterer Eingabeparameter ist bei dieser Aktivität eine data set variable, in der die generierten Daten dann gespeichert werden. Im Fehlerfall kann wieder eine entsprechende Fehlerbehandlung angestossen werden.
- **WriteDataBack**: Die WriteDataBack Aktivität ist das Gegenstück der RetrieveData Aktivität. Sie schreibt Daten zurück, d.h. sie speichert Daten aus dem Kontext des Workflows in einer Datenquelle ab. Als Eingabeparameter erwartet sie dazu eine data set variable sowie eine data container reference variable. Die Daten, die durch die erste Variable spezifiziert werden, 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.
- **TransferData**: Die TransferData Aktivität ermöglicht Datentransfers. Sie erwartet einen DM command als Eingabeparameter. Diese Anweisung muss Daten generieren, wie z.B. ein SELECT-Befehl oder ein Dateipfad. Als weitere Eingabeparameter erwartet sie dazu Referenzen auf zwei Datenquellen sowie zwei data container reference variables. Die Daten, die durch die Anwendung der Dm command auf dem ersten Container (referenziert durch die erste Variable)spezifiziert werden, werden im Container, der durch die zweite Variable referenziert wird, gespeichert.

3.3 Data Management Patterns und ihre Transformation

Die folgenden Abschnitte basieren auf [RSM14], [RS13] und [RS14].

3.3.1 Konzept der Datamanagementpatterns

Obwohl das SIMPL-Rahmenwerk mittels DM Aktivitäten einen einheitlichen Zugriff auf verschiedene Datenquellen bietet, müssen Wissenschaftler in ihren Workflowmodellen immer noch viele Details bezüglich der Implementierung der Datenbereitstellung spezifizieren, wie z.B. die Festlegung von Datenservices bzw. die Definiton von Anforderungsbeschreibungen für diese oder die Definition von Datenmanagementoperationen für DM-Aktivitäten (unter Verwendung der Befehlssprachen der zu Grunde liegenden Datenressourcen). Um Wissenschaftler von der Spezifikation von Implementierungsdetails zu befreien, sowie sie dazu zu befähigen, mit den Sprachen ihrer Simulationsmodelle zu arbeiten, statt mit Sprachen der Workflow- oder Datenmodellierung, wurde das Workflow Design Tool des SIMPL-Rahmenwerks um weitere Komponente erweitert, die ein *Pattern* -basiertes Design von Workflows unterstützt. Es bietet dabei eine erweiterbare Liste von sogenannten Patterns, die als Schablonen und Bausteine für typische Datenbereitstellungsoperationen in Simulationsworkflows dienen. In Abschnitt Abschnitt 3.1 wurde die SIMPL-Architektur näher beschrieben. Dabei sind in der Abbildung 3.1 bereits alle Erweiterungen veranschaulicht, die für das pattern-basierte Design

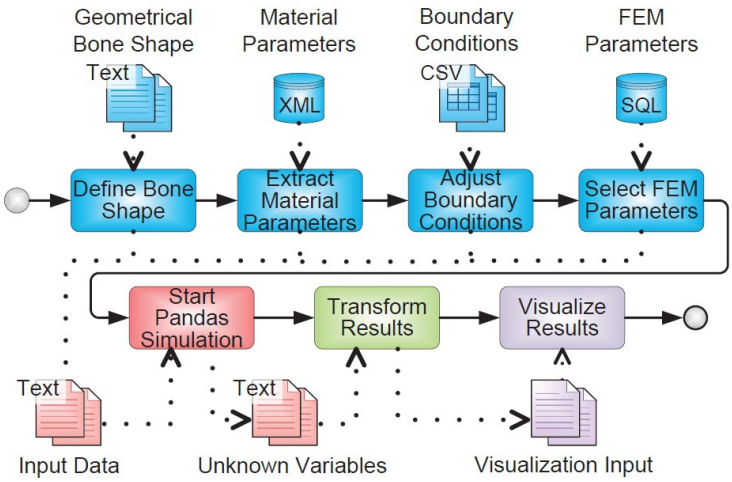


Abbildung 3.2: Workflow für die Simulation von Veränderungen der Knochenstruktur als Grundlage für die Patterntransformation [RS14]

notwendig bzw. relevant sind, wie z.B. der regelbasierte *Pattern Transformer* oder die *Simulation Artifact Registry*. Jedes Pattern umfasst mehrere fein-granulare Arbeitsschritte des Workflows. Dadurch sinkt die Anzahl der für den Wissenschaftler sichtbaren Arbeitsschritte und die Workflows werden übersichtlicher. Die Einführung von Patterns erleichtert die Modellierung von wissenschaftlichen Workflows erheblich. Der Wissenschaftler muss dabei lediglich einige, wenige, geeignete, abstrakte Patterns aussuchen, parametrisieren und kombinieren, anstatt die einzelnen Aufgaben und Implementierungsdetails des Workflows festlegen zu müssen. Die Parameter entsprechen dabei Begriffen oder Konzepten, welche die Wissenschaftler bereits aus ihren Simulationsmodellen und -methoden kennen. Mit Hilfe einer Menge von Transformationsregeln werden die parametrisierten Patterns in ausführbare Workflows transformiert.

Im Folgenden wird die Idee der Patterns anhand eines Simuationsworkflows zur Knochensimulation erläutert. Die Abbildung 3.2 stellt den Workflow vereinfacht dar. Das Pandas-Rahmenwerk benötigt als Eingaben die geometrische Form des Knochens, einige Materialparameter, Randbedingungen, wie z.B.der zeitabhängige Druck von außen auf das Knochengelenk und bestimmte FEM-spezifische Parameter (z.B. Interpolationsfunktionen zwischen Knotenpunkten im FEM-Gitter). Die ersten vier Schritte befassen sich damit, diese Daten aus vielfältigen Datenquellen (unstrukturierte Text-Dateien, XML Datenbanken, CSV-basierte Dateien und SQL-Datenbanken) herauszufiltern und sie in proprietäre Text-Dateien zu überführen, die das Pandas-Rahmenwerk verarbeiten kann. Anschließend wird mit diesen Input-Daten die Pandas-Berechnung ausgeführt, die zeitabhängigen Unbekannten der Simulation, wie z.B. die Knochendichte, berechnet und diese in weiteren Text-Dateien gespeichert. Zum Schluss werden die Resultate der Simulation in ein Format transformiert, welche durch ein zu Grunde liegendes Visualisierungstool eingelesen werden können. Durch die Visualisierung können die Wissenschaftler dann die Simulationsergebnisse interpretieren und auswerten.

In Abbildung 3.3 ist dargestellt, wie Patterns eingesetzt werden können um den Beispiel-Workflow aus Abbildung 3.2 zu vereinfachen. Die ersten vier Arbeitsschritte, die in der Abbildung blau gefärbt sind,

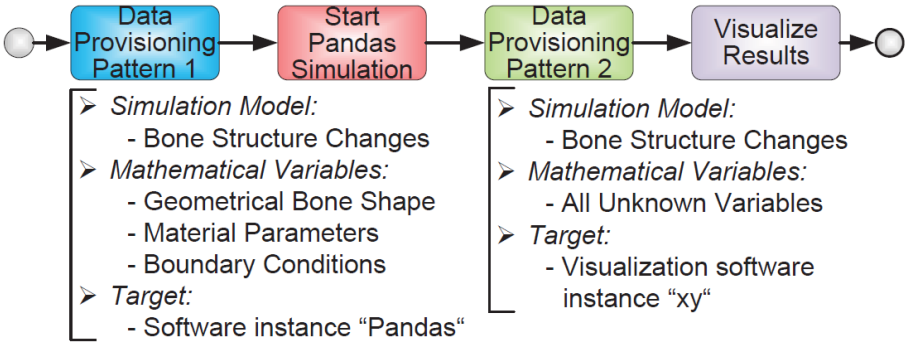


Abbildung 3.3: Die parametrisierten Patterns für die fünf Datenbereitstellungsoperationen (blau und grün) aus Abbildung 3.3 [RS14]

werden durch ein *Data Provisioning Pattern* ersetzt. Ebenso wird der grün gefärbte Arbeitsschritt, der die Ergebnisse der Pandas-Rahmenwerks transformiert, durch ein *Data Provisioning Pattern* ersetzt. Dadurch wird die Anzahl der für die Wissenschaftler sichtbaren Arbeitsschritte von insgesamt fünf auf zwei gesenkt. Als Eingabeparameter für beide *Data Provisioning Patterns* dienen ein Simulationsmodell und die Menge seiner mathematischen Variablen (z.B. konkrete Instanzen der geometrischen Knochenform, der Materialparameter und der Rahmenbedingungen bei der Datenbereitstellung für die Pandas-Berechnungen oder unbekannte Variablen bei der Ergebnisvisualisierung. Die Endpunkte der Datenbereitstellung sind Referenzen auf entsprechende Software-Instanzen, wie z.B. die Instanzen von Pandas oder die Instanzen eines Visualisierungs-Werkzeugs.

3.3.2 Hierarchie von Datenmanagementpatterns

Die Pattern-Transformation erfolgt gemäß einer Pattern-Hierarchie über mehrere Abstraktionsebenen. Parametrisierte Patterns werden durch Transformationsregeln in konkretere Patterns, Schablonen oder Dienste und letztendlich in ausführbare Workflow-Fragmente transformiert. Alle Patterns und Dienste werden abhängig von ihrem Implementierungsgrad von Personen mit unterschiedlichen Qualifikationen, wie z.B. Wissenschaftlern, Datenverwaltungsexperten oder Workflow-Ingenieuren, bereitgestellt. Durch die Abstraktionsebenen wird somit eine Trennung von Aufgabengebieten erzielt. Aufgrund ihrer Beziehung zu Simulationsmodellen, können z.B. die *Data Provisioning Patterns* aus Abbildung 3.3 sehr gut von Wissenschaftlern festgelegt werden. Im folgenden Abschnitt wird die Hierarchie der Patterns, welche in 3.5 gezeigt ist, näher beschrieben. Je höher ein Data Management Pattern sich in dieser Hierachie befindet, desto mehr Details bzgl. der zugrundeliegenden Datenmanagementoperationen und Befehlssprachen werden in diesem Pattern abstrahiert und desto stärker werden Informationen verdichtet. Dementsprechend müssen Wissenschaftler mit steigender Hierarchieebene auch über immer weniger Detailwissen verfügen. Gleichzeitig nimmt mit steigender Abstraktionsebene auch die Nähe zwischen den Parametern der Patterns und den Sprachen der jeweiligen Simulationsmodelle zu und die Nähe zu den Sprachen der Workflow- oder Datenmodellierung entsprechend ab. In Abbildung 3.4 werden die Parameterwerte und Simulations-Artefakte ihrem Abstraktionsgrad entsprechend klassifiziert. In umgekehrter Richtung, müssen mit sinkendem

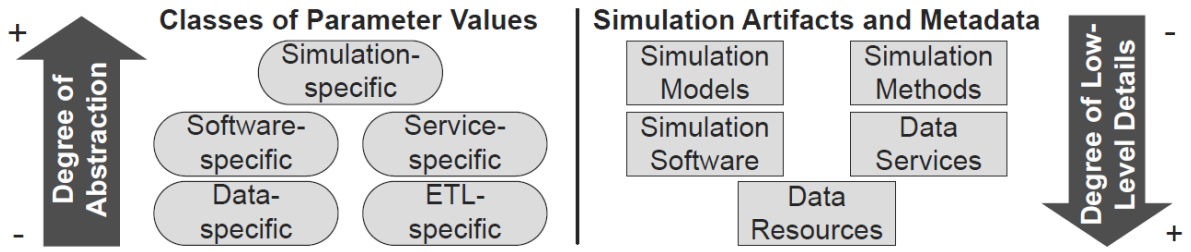


Abbildung 3.4: Die Klassifizierung der Pattern-Parameterwerte [RSM14b]

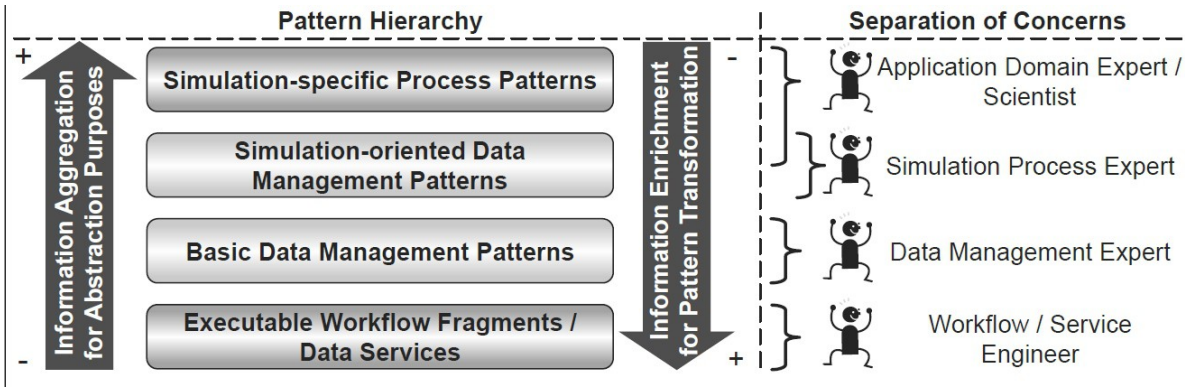


Abbildung 3.5: Die Hiearchie der Data Management Patterns [RSMC14]

Abstraktionsgrad, die Patterns der höheren Hierarchiebenen mit Informationen angereichert werden, indem sie durch Transformationsregeln auf Patterns darunterliegender Hierarchiebenen bzw. auf Datenservices und Workflow-Fragmente abgebildet werden.

Die erste Ebene bilden die **ausführbaren Workflow-Fragmente** und Services für das Datenmanagement, sogenannte **Datenservices**, auf welche die Patterns letztendlich durch die rekursive Anwendung von Transformationsregeln abgebildet werden. Sie enthalten sehr viele Implementierungsdetails und setzen die Patterns höherer Hierarchiebenen um. Hier können Workflow und Web-Service-Ingenieure ihr Wissen bereitstellen. Auf der nächsten Ebene folgen die **grundlegenden Data Management Patterns**. Sie abstrahieren von den Implementierungsdetails der darunterliegenden Ebene und setzten die *Simulation-oriented-Data Management Patterns* der nächsthöheren Ebene um. Ein Beispiel sind *Data Transfer and Transformation Patterns*, durch die *Simulation-oriented Data Provisioning Patterns* implementiert werden können. *Data Iteration Patterns* hingegen iterieren über einer Datenmenge S und führen eingebettete Operationen für einzelne Teilmengen von S aus. Bei den Werten der Parameter kann es sich um daten-, dienst- oder ETL-spezifische Werte handeln. Experten für Datenmanagement können für diese Ebene Parameterwerte bereitstellen.

Die nächste Ebene umfasst die **Simulation Oriented Data Management Patterns**, welche die darunterliegenden *grundlegenden Patterns* als Basis nutzen. Die Parameterwerte besitzen densel-

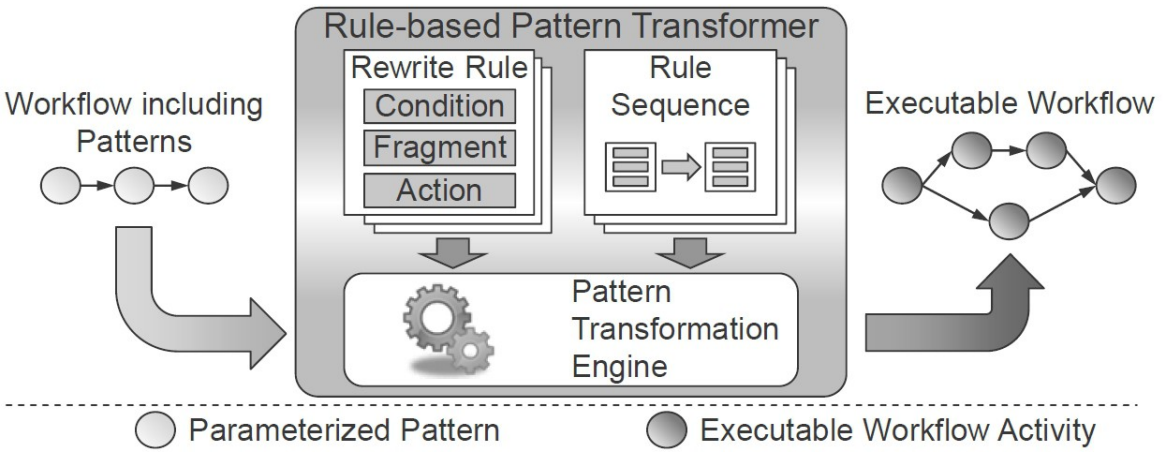


Abbildung 3.6: Das Verarbeitungsmodell der Patterntransformation [RSM14b]

ben Abstraktionsgrad wie die Parameterwerte der *Simulation-specific-Data Management Patterns*, haben jedoch einen stärkeren Bezug zu Anwendungsfällen in der Datenverwaltung. D.h. die Parametrisierung kann immer noch seitens der Wissenschaftler erfolgen, besser jedoch von Experten für Simulationsprozesse. Ein Beispiel sind die *Simulation-oriented Data Provisioning Pattern*, welche Abstraktionen für die Bereitstellung von Daten für Simulationsberechnungen oder Visualisierungen von Ergebnisdaten darstellen, z.B. als Teil der *Simulation Model Realization* oder *Simulation Result Interpretation Patterns* der nächsthöheren Hierarchiebene. Die hier für die Parameter bereitzustellenden Daten sind *Simulationsmodelle* oder eine Menge der *mathematischen Variablen* eines solchen Simulationsmodells. Ein weiteres Beispiel sind die *Simulation-oriented Data Interoperability Pattern*, welche Datenabhängigkeiten zwischen zwei Simulationsmodellen pber simulationsspezifische Parameterwerte definieren. Das *Parameter Sweep Pattern* schließlich unterstützt Prozesse, die auf einer Liste von simulationsspezifischen Parametern iterieren und für jeden Parameter in dieser Liste eine Operation ausführen.

Die oberste Ebene mit dem höchsten Abstraktionsgrad bilden die **Simulation-specific Process Patterns**. Diese Patterns haben den stärksten Bezug zu Simulationen und repräsentieren Anwendungsfälle, die im Interessensgebiet der Wissenschaftler liegen. Wissenschaftler müssen nur einige, wenige *Simulation-Specific Process Patterns* kombinieren, um Simulationsprozesse zu erstellen. Dabei entsprechend wenige*simulationsspezifische Parameter* spezifiziert werden, die domänenspezifischen Simulationsartefakten entsprechen, welche den Wissenschaftler bereits vertraut sind (z.B. Simulationsmodelle, die mathematischen Variablen dieser Simulationsmodelle oder Simulationsmethoden wie FEM (vergleiche

3.3.3 Transformation von Datenmanagementpatterns

In diesem Abschnitt wird die Patterntransformation beschrieben. Abbildung 3.6 zeigt das Verarbeitungsmodell der Patterntransformation. Der *Pattern Transformer* enthält eine erweiterbare Menge von Transformationsregeln und wandelt Workflows, welche Datenmanagementpatterns enthalten,

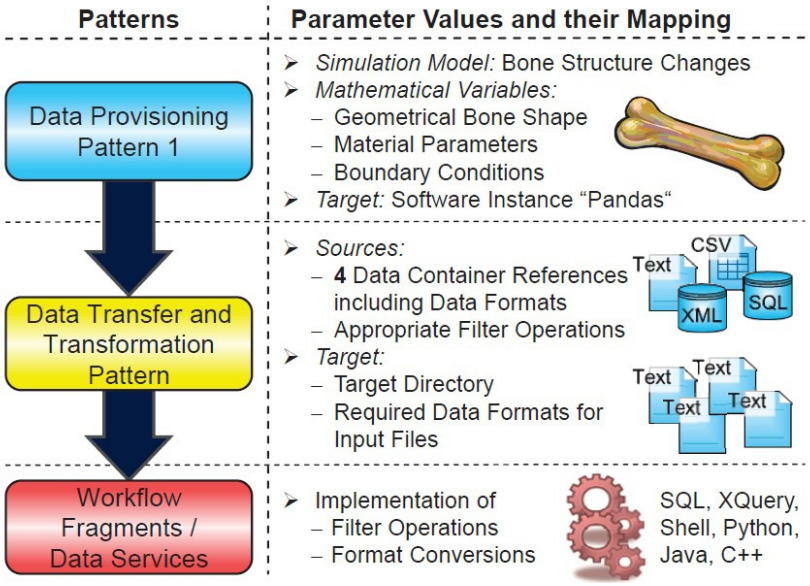


Abbildung 3.7: Regelbasierte Transformation des Data Provisioning Patterns aus Abbildung 3.7 [RS14]

mit Hilfe einer *Transformation Engine* in ausführbare Workflows um. Er erhält als Eingabe den auf der linken Seite der Abbildung dargestellten Workflow, in dem mehrere Patterns eingebettet sind, traversiert diesen und sucht für jedes gefundene Pattern eine geeignete Transformationsregel. Dabei besteht jede Regel aus einem *Bedingungs-*, einem *Fragment-* und einem *Aktionsteil*. Der Bedingungsteil spezifiziert die Bedingungen, die erfüllt sein müssen, um den restlichen Teil der Regel anzuwenden. Der Fragmentteil identifiziert in einer *Workflow Fragment Libary* ein Template für ein zu verwendendes Workflow-Fragment. Der Aktionsteil fügt diesem Fragment Implementierungsdetails hinzu und bildet eventuell Parameter höheren Abstraktionslevels auf solche niedrigeren Levels ab. Abschließend wird das Pattern in das Workflow-Fragment abgebildet. Dieser ist entweder ausführbar oder enthält weitere Patterns. Trifft Letzteres zu, muss die Transformation rekursiv fortgesetzt werden bis ein ausführbarer Workflow vorliegt. Desweiteren existiert für jedes Ebene der Hierarchie der Datenmanagementpatterns eine *Rule Sequence*, welche jedem Pattern dieser Ebene Transformationsregeln zuordnet und die Reihenfolge der Anwendung von Regeln bestimmt. Die erste Regel der Rule Sequence, dessen Bedingungen erfüllt sind, wird ausgeführt und die restlichen Regeln ignoriert. In Abbildung 3.7 ist die Transformation des ersten *Data Provisioning Patterns* des Beispiel-Workflows aus Abbildung 3.2 veranschaulicht.

Der erste Transformationssschritt bildet das *Data Provisioning Pattern* in ein *Data Transfer and Transformation Pattern* ab. Dabei werden durch die Transformationsregel drei mathematische Variablen auf drei *data container reference variables* abgebildet, welche jene Datencontainer referenzieren, in denen entsprechende Eingabedateien enthalten sind. Die FEM-Parameter für das übergeordnete *Data Provisioning Pattern* werden automatisch, ohne das Hinzutun eines Wissenschaftlers, dem Pattern hinzugefügt. Dazu werden Metadaten genutzt, welche die Abhängigkeiten zwischen Simulationsmodellen und Simulationmethoden beschreiben und somit die SQL-Datenbank, welche die FEM-Parameter ent-

hält, implizit bestimmen. Desweiteren fügt die Transformationsregel mittels Filteroperationen, welche geeignete Daten aus den Datencontainern beziehen, Implementierungsdetails hinzu. Beispielsweise wird im Datencontainer *directory* das Ziel des Datentransfers, wo das Pandas-Tool seine Eingabedaten erwartet und die für das Pandas-Tool geeigneten Datenformate spezifiziert. Aufgrund seinem starken Bezug zu Datenressourcen und Datentransformationsoperationen wird das *Data Transfer and Transformation Pattern* typischerweise von Datenmanagement-Experten spezifiziert. Anschließend wird durch eine weitere Transformation das *Data Transfer and Transformation Pattern* in ein ausführbares Workflow-Fragment oder einen Datenservice überführt. Diese Fragmente müssen mittels SQL- oder XML-Befehlen, gegebenenfalls sogar mittels Skript- und Programmiersprachen alle Filteroperationen und Datenformatskonvertierungen implementieren. Deshalb eignen sich insbesondere Workflow- oder Service-Ingenieure für die Bereitstellung solcher Workflow-Fragmente und Dienste.

4 Knochensimulationen und Proteinmodellierung

In diesem Kapitel werden die für diese Arbeit relevanten Anwendungsfälle für wissenschaftliche Workflows (vergleiche Abschnitt 2.1.4) vorgestellt. Zu Beginn wird das Szenario der Knochensimulation beschrieben, bei der die Strukturveränderung von menschlichen Knochen bei wechselnden Belastungen simuliert wird. Dabei wird in Abschnitt 4.1 zunächst ein Kopplungsworkflow beschrieben, der eine Multisimulation von Knochenveränderungen darstellt, indem er eine biomechanische und eine systembiologische Simulation vereint. Anschliessend wird näher auf die BPEL-DM-Realisierungen sowohl vom biomechanischen als auch vom systembiologischen Workflow eingegangen (Abschnitt 4.2). Als zweites Anwendungsszenario wird in Abschnitt 4.3 ein Workflow aus dem Bereich der Bioinformatik zur Proteinmodellierung vorgestellt, bei dem mit Hilfe einer Ähnlichkeitssuche (Pattern-Matching) nach einem bestimmten Muster, beispielsweise wichtigen Regionen innerhalb von Proteinsequenzen, gesucht wird.

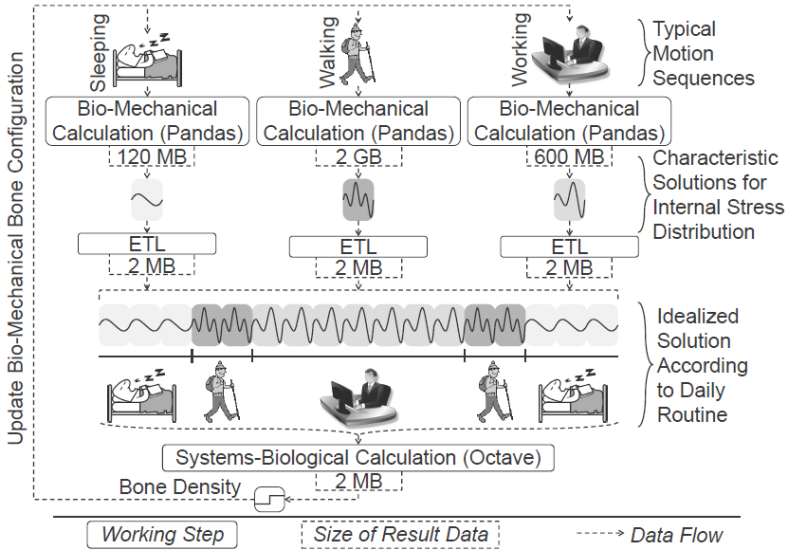


Abbildung 4.1: Konzept der gekoppelten Simulation der Veränderung der Knochenstruktur [RSM14b]

4.1 Knochensimulation mit Pandas und GNU Octave

Der folgende Abschnitt basiert auf [Ari12], [Pie12], [Dor11] und [RSM14a]. Im Folgenden wird eine Multi-* Simulation vorgestellt (vergleiche Abschnitt 2.5.1), die durch die Berücksichtigung biomechanischer als auch systembiologischer Prozesse zwei verschiedene Domänen kombiniert. Während die biomechanische Simulation die mechanische Veränderung der Knochenstruktur auf der Gewebeebene simuliert, bestimmt die systembiologische Simulation Prozesse des Gewebeauf und -abbaus, d.h. Veränderungen des Knochengewebes auf zellulärer Ebene. Die biomechanische Simulation konzentriert sich dabei hauptsächlich auf Massenbewegungen zwischen den porösen Medien und den darin enthaltenen Flüssigkeiten. Da der Workflow zwei verschiedene Simulationen mit unterschiedlichen Granularitätsebenen kombiniert, wird für die Simulation insgesamt ein präziseres Endergebnis erzielt.

Für den biomechanischen Workflow wird das Pandas Rahmenwerk verwendet. Der systembiologische Workflow wird durch die GNU Octave Rechenumgebung ausgeführt. Pandas verwendet für die Simulation auf makroskopischer Ebene eine größere Raumskala als GNU Octave für die Simulation auf mikroskopischer Ebene. Desweiteren besitzt das biomechanische Modell eine große Zeitskala, welche z.B. in Tagen abläuft und das systembiologische Modell hat eine kleinere Zeitskala, die zum Beispiel in Minuten oder Stunden abläuft.

Die Rahmenbedingungen der gesamten Simulation werden bestimmt von der Art und Weise in der sich die relevante Person bewegt. Das wiederum ist abhängig von den täglichen Routinen der Person, beispielsweise die Routinen mehrerer typischer Arbeitstage. In der Simulation werden tägliche Routinen dargestellt durch eine Zusammensetzung von repräsentativen Bewegungssequenzen, wie z.B. Gehen, Schlafen und Arbeiten.

Abbildung 4.1 zeigt das Konzept der Kopplung von zwei Simulationsmodellen zur Ausführung der Simulation. Die biomechanische Simulation berechnet zu Beginn für jede Bewegungssequenz die Veränderung der Knochenstruktur. Die systembiologische Simulation setzt anschließend die Ergebnisse der einzelnen Berechnungen dem typischen Arbeitstag entsprechend zu einem Tagesablauf zusammen. Basierend auf den Berechnungen der systembiologischen Simulation für diesen Tagesablauf wird wiederum die biomechanische Simulation für den nächsten Arbeitstag angepasst. Dieser Ablauf wird solange wiederholt bis alle zu berechnenden Arbeitstage berechnet wurden.

Der folgende Abschnitt basiert auf [RSM14a]. In Abbildung 4.2 ist der mit BPEL-DM umgesetzte *Kopplungsworkflow* abgebildet. Er beginnt zunächst mit dem Aufruf eines Repository-Dienstes, der eine Liste von Tagesroutinen lädt, die mehrere Bewegungssequenzen beinhalten. Anschließend iteriert eine BPEL Foreach-Aktivität über dieser Liste und führt jeweils für jede Bewegungssequenz die biomechanische und die systembiologische Simulation aus.

Die biomechanische Simulation kann für jede Bewegungssequenz einer Tagesroutine unabhängig ausgeführt werden, was die Parallelisierung dieser Simulation auf mehreren Rechnern ermöglicht. Dazu ruft der Workflow einen Repository-Dienst auf, der eine Liste von verfügbaren Pandas-Rechnern enthält. Durch einen XQuery-Befehl wird jede Bewegungssequenz der aktuellen Tagesroutine einem Pandas-Rechner zugeordnet, auf dem die Verarbeitung dieser Bewegungssequenz stattfinden soll. Anschließend wird für jede Bewegungssequenz parallel ein Pandas Simulation Service gestartet.

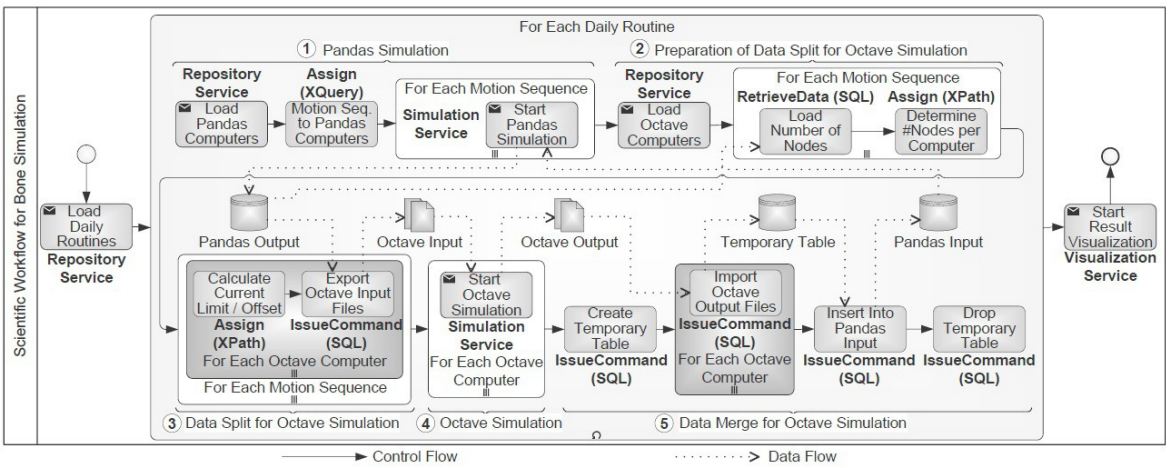


Abbildung 4.2: BPEL-DM-Workflow zur gekoppelten Simulation von Strukturveränderungen in Knochen [RSM14a]

In der Berechnungsphase berechnet Pandas mit einer sequentiellen Schleife für die ersten n Zeitschritte die mechanischen Belastungsverteilungen im Knochen und speichert sie in einer Datenbanktabelle (Pandas Output) ab. Die Daten werden dabei in die einzelnen Zeitschritte sowie in Tausend oder Millionen Elementen eines FEM-Gitters strukturiert. Für die Interpolationsberechnungen der Belastungsverteilung im Knochen werden jedem Element des FEM-Gitters mehrere Gausspunkte als Stützstellen zugeordnet. Für jeden Zeitschritt und jeden Gausspunkt speichert Pandas Werte zu zehn Variablen des biomechanischen Simulationsmodells.

Die folgende systembiologische Simulation benötigt nur die Werte des letzten berechneten Zeitschritts und nur zwei der zehn Variablen. Deshalb erfolgen anschließend entsprechende Filterungen der von Pandas generierten Ausgabedaten. Aufgrund der feinen Granularität und Rechenintensivität der systembiologischen Simulation, wird sie zudem parallelisiert und es findet eine Aufteilung der Daten auf mehrere Rechner und Instanzen von Octave statt.

Der Kopplungsworkflow steuert diese Filterung und Aufteilung der Daten. Dazu lädt er eine Liste aller verfügbaren Octave-Rechner aus einem Repository und bestimmt die Aufteilung der Daten auf diese Rechner gemäß der Vorgaben der Wissenschaftler, beispielsweise entsprechend den Gausspunkten des FEM-Gitters, wie sie in der Pandas-Datenbanktabelle gespeichert sind.

Über eine SIMPL RetrieveData-Aktivität holt sich der Workflow dazu zunächst die ID des letzten berechneten Zeitschritts aus der Datenbanktabelle zu Gausspunkten. Dabei wird eine SQL SELECT-Anweisung abgesetzt, die mit Hilfe einer Aggregatfunktion die maximale Zeitschritt-ID ermittelt. Zusätzlich wird ein Filterprädikat bzgl. der aktuellen Simulations-ID verwendet, da in der Tabelle zu Gausspunkten Daten für mehrere Simulationen gespeichert sein können.

Für jede Bewegungssequenz wird anschließend mit Hilfe einer RetrieveData-Aktivität die Gesamtzahl der relevanten Gausspunkte in eine Workflow-Variable geladen. Dazu wird eine SQL-Select-Anweisung genutzt, welche eine entsprechende Aggregatfunktion sowie Filterprädikate nach der Simulations-ID

und nach dem Zeitschritt beinhaltet. Die nächste BPEL Assign-Aktivität verwendet einen XPath-Ausdruck um die Anzahl der Gausspunkte pro Octave-Rechner zu bestimmen.

Anschließend setzt die eigentliche Aufteilung der Pandas-Ausgabedaten ein, wobei in einer parallelen Dateniteration eine IssueCommand-Aktivität für jeden Octave-Rechner den Export der Daten aus der Datenbanktabelle in eine CSV-Datei durchführt. Für jede Bewegungssequenz und für jeden Octave-Rechner berechnet zunächst eine Assign-Aktivität durch eine XPath-Anweisung die Limit- und Offset-Werte für die Extraktion der richtigen Gausspunkte aus der Pandas-Datenbanktabelle. Diese Werte basieren auf der im vorherigen Schritt bestimmten Gesamtzahl der Gausspunkte und der Anzahl der pro Octave-Rechner zu verarbeitenden Gausspunkte. Die für den Export zuständige IssueCommand-Aktivität enthält eine SQL-Anweisung, welche mit Hilfe von Filterprädikaten nach dem Zeitschritt und der Simulations-ID sowie nach den zuvor berechneten Limit- und Offset-Werten die richtigen Gausspunkte und Bewegungssequenz selektiert. Des Weiteren beinhaltet dieser SQL-Befehl eine Projektion auf über alle Zeitschritte berechnete Durchschnittswerte relevanter Spaltenwerte der Gausstabelle. Diese Tabellenspalten repräsentieren die relevanten Variablen des biomechanischen Simulationsmodells, die zwischen und Pandas und Octave ausgetauscht werden.

Anschließend startet der Kopplungsworkflow eine neue Instanz des systembiologischen Simulationsworkflows. Dieser stellt die notwendigen Plattformen und Softwarepakete für Octave bereit und kopiert in der Datenbereitstellung die CSV-Datei auf den jeweiligen Octave- Rechner. Jeder Octave Rechner berechnet mit der erhaltenen CSV-Datei die geänderten Werte der Gausspunkte und speichert sie in einer weiteren CSV-Datei ab. Nach der parallelen Ausführung aller Octave-Simulationen importiert der systembiologische Workflow in der Nachbearbeitungsphase die resultierenden CSV-Dateien in eine Datenbanktabelle (Pandas Input), aus der Pandas sie auslesen kann. Dazu wird zunächst eine temporäre Tabelle erstellt, in welche die nachfolgende IssueCommand-Aktivität mit Hilfe von SQL Import-Anweisungen die Ausgabedaten jedes Octave-Rechners importieren kann. Anschließend kopiert eine mengenbasierte SQL-Import-Anweisung die Daten der tempörären Tabelle in die Pandas Input Datenbanktabelle. Danach wird die temporäre Tabelle mit Hilfe einer SQL-Drop-Anweisung wieder gelöscht und der systembiologische Workflow beendet.

Nach Beendigung des systembiologischen Workflows importiert der Kopplungsworkflow die in der Datenbanktabelle Pandas Input enthaltenen Daten in die Datenbank der Gausspunkte, womit die Knochenkonfiguration des biomechanischen Modells für den nächsten Zeitschritt angepasst wird. Der biomechanische Simulationsworkflow wiederholt diesen Prozess, bis alle Zeitschritte der Simulation betrachtet wurden. Zusätzlich zu den mechanischen Belastungsverteilungen speichert der Workflow auch die Knochenstrukturen für alle Zeitschritte in einem Pandas-spezifischen Dateiformat. Wurden alle Tagesroutinen verarbeitet, werden die Ausgabedaten abschließend in Datenformate transformiert, mit denen das von den Wissenschaftlern gewünschte Visualisierungstool arbeiten kann, und bei Bedarf auf den Rechner dieses Tools kopiert.

4.2 Der systembiologische und der biomechanische Workflow

Im Folgenden werden die Implementierungen des systembiologischen und des biomechanischen Workflows näher beschrieben. Das in beiden Workflows enthaltene Workflow-Fragment der Datenbereitstellung wird in Abschnitt 4.2.3 erklärt. Dieser Abschnitt basiert auf [Boh].

4.2.1 Der biomechanische Workflow

Sowohl der biomechanische, als auch der systembiologische Simulationsworkflow sind in die Phasen Bereitstellung, Berechnung und Nachbearbeitung gegliedert. Die Bereitstellungsphase wird genutzt, um zunächst notwendige Plattformen zu erzeugen. Dies beinhaltet insbesondere die Erzeugung von Verzeichnisstrukturen auf den jeweiligen Rechnern, in welche die nachfolgenden Aktivitäten Softwarepakete für Pandas bzw. Octave installieren sowie Eingabedaten kopieren können. Durch diese Eingabedaten wird das entsprechende Simulationsmodell beschrieben. Dazu extrahiert der Workflow Daten aus mehreren heterogenen Datenquellen (relationale Datenbanken, strukturierte und unstrukturierte Textdokumente) und wandelt diese in Dateiformate um, mit denen Pandas arbeiten kann.

Abbildung 4.3 zeigt den Aufbau des biomechanischen Workflows. Zu Beginn des Workflows wird die Webservice Methode PlatformProvisioning aufgerufen, welche parallel zwei neue Simulationsinstanzen erzeugt und für jede dieser Instanzen eine SimID zurückliefert. Nach dem PlatformProvisioning wird über den Methodenaufruf PreparePandasSource für jede Pandas Instanz der Pandas Sourcecode bereitgestellt. Nach diesen beiden Aufrufen erfolgt die eigentliche Datenbereitstellungsphase. Hierfür wird das *Simulation Oriented Data Provisioning Pattern* verwendet. Es stellt beiden Simulationsinstanzen die für ihre Simulation notwendigen Eingabedaten bereit. Als Parameter erhält sie Nach der Datenbereitstellung wird durch den Aufruf CompilePandas der Pandas Sourcecode kompiliert. Anschließend werden durch den Aufruf der Methode StartPandas beide Instanzen gestartet. Die folgenden Methoden werden nicht mehr parallel aufgerufen. Es wird zunächst die Methode RunCmd aufgerufen, welche die Commandfiles der beiden Simulationsinstanzen ausführt. Das Ergebnis der Simulation wird von Pandas automatisch in der dafür bereitgestellten SQL Datenbank gespeichert. Nach abgeschlossener Ausführung der Simulation werden durch den Aufruf der Methode Stop Pandas die Pandas Instanzen beendet und der biomechanischeWorkflow gibt eine Rückmeldung über den korrekten Ablauf an den aufrufenden Workflow.

4.2.2 Der systembiologische Workflow

In diesem Abschnitt wird der systembiologischen Workflow beschrieben, der durch den Kopplungsworkflow aufgerufen wird, um die Prozesse des Gewebeauf und -abbaus, d.h. Veränderungen des Knochengewebes auf zellulärer Ebene zu simulieren. Dieser Workflow verwendet den Octave Webservice. Abbildung 4.4 zeigt den Aufbau des systembiologischen Workflows. Zu Beginn des Workflows erfolgt im Prepare Simulation Schritt ein Web-Service-Aufruf, der eine neue Simulationsinstanz erzeugt. Anschließend folgt die Datenbereitstellungsphase, in der durch das *Simulation Oriented Data Provisioning Pattern* die notwendigen Simulationsdaten bereitgestellt werden. Durch den Aufruf von StartProgram wird die Octave Simulation gestartet. Nach Abschluss der Simulation werden die Ausgabedaten an den Kopplungsworkflow zurückgegeben.

4.2.3 Das Workflow-Fragment der Datenbereitstellungsphase

Die Datenbereitstellungsphase wird durch das *Simulation Oriented Data Provisioning Pattern* repräsentiert. Als Eingabeparameter erhält dieses Pattern das Simulationsmodell, die von dem jeweiligen

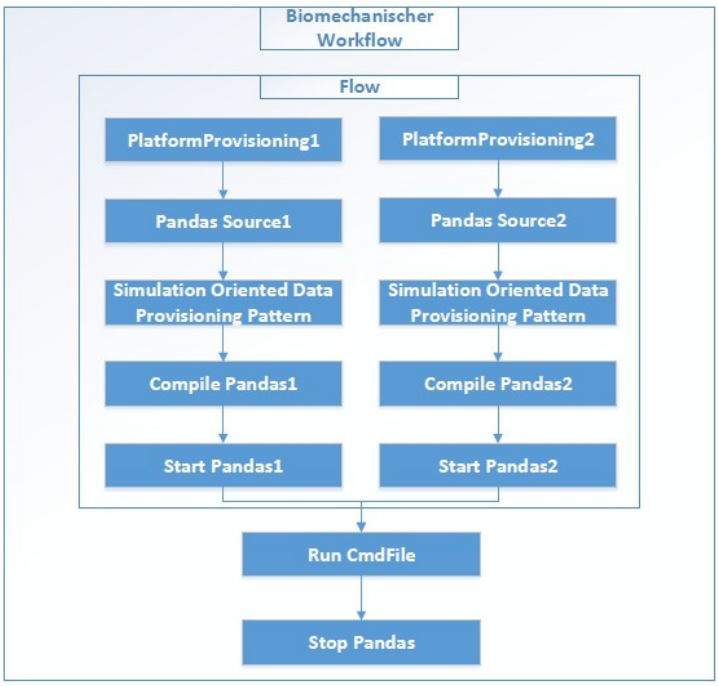


Abbildung 4.3: Der biomechanische Workflow [Boh]

Modell benötigten mathematischen Variablen und die Ziel-Softwareinstanz, auf der die Simulation durchgeführt wird. In dieser Phase werden die für die jeweilige Simulation notwendigen Dateien auf den Pandas bzw. Octave-Rechner kopiert. Diese ergeben sich aus den mathematischen Variablen, die das Simulationsmodell, den spezifizierten Knochen und die Randbedingungen der Bewegungssequenz definieren. In der zum jeweiligen Simulationsmodell gehörenden Ontologie (BoneTissue bzw. BoneCell) sind den mathematischen Variablen konkrete Instanzen zugeordnet, welche den Namen und den konkreten Dateipfad der Eingabedatei spezifizieren.

Das relevante, ausführbare Workflow-Fragment, welches aus der Transformation des *Simulation Oriented Data Provisioning Patterns* resultiert, ist für den biomechanischen und den systembiologischen Workflow identisch. Eingeleitet wird das Fragment durch einen scope, der eine Sequenz enthält, die wiederum eine Assign Aktivität InitContainerReferenceList sowie einen weiteren scope umfasst. Die Assign-Aktivität erstellt eine *data container reference list*, welche die zu den mathematischen Variablen gehörenden *data container reference variables* enthält.

Der innere Scope enthält eine Sequenz, welche wiederum eine Assign-Aktivität InitLocalContainerReferenceList und eine For-Each-Aktivität mit eingebetteter Assign-Aktivität getPath und TransferData enthält. InitLocalContainerReferenceList erzeugt eine weitere *data container reference list*, die eine Kopie der zuvor erstellten Liste darstellt. Diese Kopie dient der For-Each-Aktivität als Iterationsmenge und jede in ihr enthaltene Datencontainerreferenzvariable enthält zusätzlich die *data source reference* der zugehörigen, konkreten Datenquelle. Die Assign-Aktivität getPath weist einer Variable *currentContainer* die im aktuellen Iterationsschritt referenzierte *data container reference variable* und der

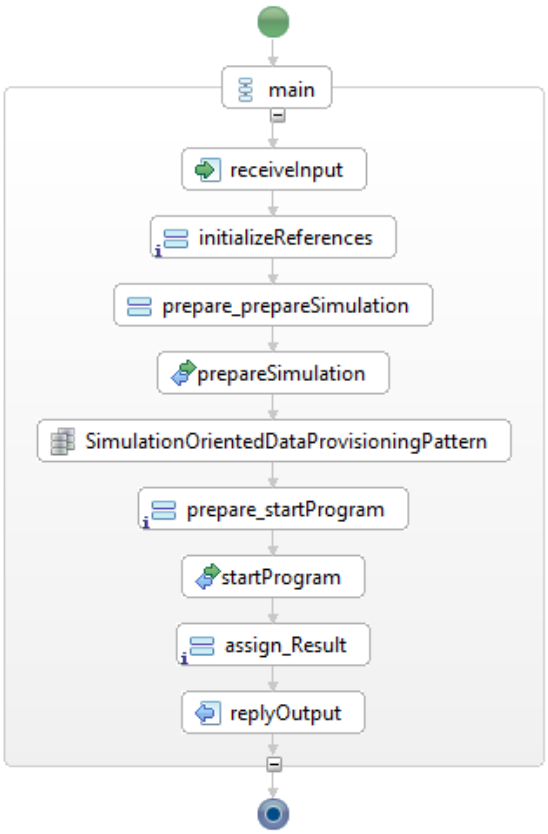


Abbildung 4.4: Der systembiologische Workflow [Boh]

als Datenquelle von *currentcontainer* spezifizierten *data source reference variable scopeDataSource* die entsprechenden, aktuellen Werte zu. Die TransferData-Aktivität kopiert schließlich die relevante Eingabedatei des durch *currencontainer* spezifizierten Datencontainers in der *scopeDataSource* auf das Zielverzeichnis im Pandas- bzw. Octave-Rechner.

4.3 Proteinmodellierung

Im Folgenden wird eine Workflow vorgestellt, der bei der Modellierung von Proteinen Einsatz findet [BTS]. Dieser Workflow wird zur Klasse der Datenmodellierungsworkflows zugeordnet, die genutzt werden um unter anderem Modelle aus gegeben Problemstellungen zu extrahieren, die anschließend für Berechnungen herangezogen werden können. Es werden dabei unterschiedliche Modelle erzeugt und auf ihre Eigenschaften hin untersucht.

Der Proteinmodellierungsworkflow vergleicht Proteinsequenzen auf ihre Ähnlichkeit mit gewissen Mustern. So wird versucht wichtige Bereiche innerhalb von Proteinsequenzen oder -familien ausfindig zu machen, beispielsweise Aminosäuren, die für gewisse chemische Reaktionen von Bedeutung sind.

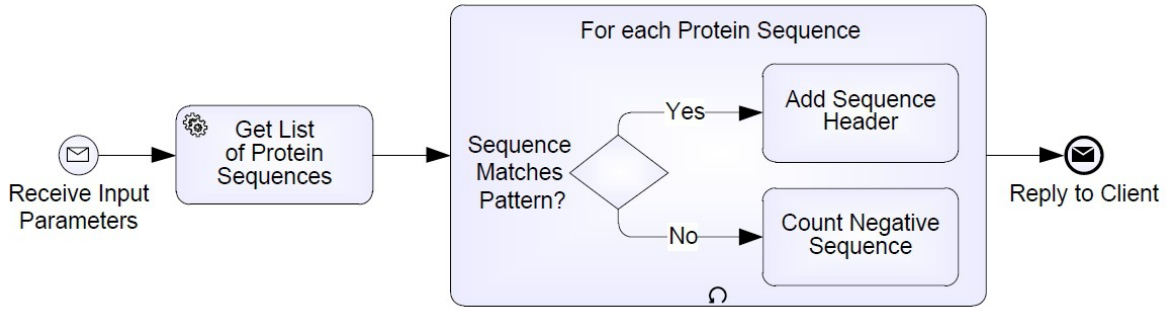


Abbildung 4.5: Workflow zur Proteinmodellierung in Business Process Modeling Notation (BPMN) [RSM11]

Dadurch können Proteine ermittelt werden, die gewisse chemische oder biologische Problemstellungen lösen können.

In Abbildung 4.5 ist ein Beispiel eines solchen Proteinmodeelierungsworklfows in BPMN-Notation veranschaulicht. Er erhält vom Nutzer zunächst Eingabeparameter, wie z.B. Referenzen auf die Datenressurcen, in denen die zu unersuchenden Proteinsequenzen gespeichert sind. Anschließend ruft der Workflow einen Dienst auf, der eine Liste der relevanten Proteinsequenzen lädt. Diese Liste wird in einer Variable im Workflow-Kontext gespeichert. Der Workflow iteriert über diese Liste und sucht z.B. mit Hilfe eines regulären Ausdrucks nach einem gewissen Muster innerhalb der Proteinsequenz. Wird das Muster ausfindig gemacht, fügt der Workflow den Header der Proteinsequenz einer Liste hinzu, welcher die Liste aller hener Header enthält, deren zugehörige Proteinsequenz das Muster aufwies. Diese Liste ist ebenso im Workflow-Kontext als Variable gespeichert. Falls das Muster nicht gefunden wird, wird ein Zähler, der die Anzahl der Proteinsequenzen festhält, welche das Muster nicht aufwiesen, um eins erhöht. Nachdem alle Proteinsequenzen durchgearbeitet wurden, sendet der Workflow sowohl die Liste der Header der positiven Proteinsequenzen, als auch die Anzahl der negativen Proteinsequenzen an den Client zurück.

5 Optimierung von datenintensiven Workflows

Datenintensive Workflows müssen häufig mit sehr großen Datenmengen umgehen. Insbesondere bei computerbasierten Simulationen im wissenschaftlichen Bereich (vergleiche Abschnitt 2.4.4) kann die Größe der zu verarbeitenden Daten zwischen mehreren GBs und zahlreichen TBs liegen. Mit den Workflows zur Simulation von Veränderungen der Knochenstruktur und der Proteinmodellierung wurden in Kapitel 4 zwei Beispiele für solche Simulationsworkflows vorgestellt. Insbesondere die Laufzeit der datenverarbeitenden Operationen ist bestimmend für die Gesamtleistung dieser Workflows.

Eine effiziente Datenverarbeitung und die Unterstützung von Optimierungsansätzen sind somit für Laufzeitverbesserungen solcher Simulationsworkflows unumgänglich. Optimierungen beziehen sich jedoch nicht ausschliesslich auf Laufzeitverbesserungen. Zudem ist es wünschenswert, dass Simulationsworkflows ein Design aufweisen, das übersichtlich ist und auf die wissenschaftliche Funktionalitäten fokussiert, während technische Details, wie z.B. spezifische Zugriffsmethoden auf Datenressourcen oder die Bereitstellung von notwendigen Daten den Wissenschaftlern verborgen bleiben bzw. vereinfacht werden. Dies ermöglicht es den Wissenschaftlern sich bei der Erstellung von Workflows auf ihre eigentliche wissenschaftliche Arbeit zu konzentrieren. Häufig geht ein gutes Workflow-Design bzw. -Design-Paradigma mit Laufzeitverbesserungen einher. Beispielsweise werden durch die Reduktion der datenverarbeitenden Aktivitäten in Workflows sowohl leichter verständliche, übersichtlichere Workflows erzeugt, als auch die Kommunikation und der Datenverkehr mit den zu Grunde liegenden Datenressourcen reduziert und folglich Laufzeitverbesserungen erzielt.

Da sich neben dem Unternehmensbereich mittlerweile auch im wissenschaftlichen Bereich die kontrollflussorientierte Workflow-Sprache WS-BPEL als Standard-Beschreibungssprache für Workflows durchgesetzt hat und die in dieser Arbeit betrachteten Beispiel-Workflows mit kontrollflussorientierten Sprachen beschrieben wurden, werden in dieser Arbeit die Optimierungsvorschläge insbesondere im Hinblick auf ihr Optimierungspotential bezüglich solcher kontrollflussorientierter Workflow-Sprachen beurteilt.

In diesem Kapitel werden existierende Optimierungsansätze für datenintensive Workflows betrachtet (Abschnitt 5.1, Abschnitt 5.2 und Abschnitt 5.3). Dabei wird jeweils der Optimierungsansatz vorgestellt und seine Grundideen kurz erklärt.

5.1 Regelbasierte Optimierung von datenintensiven Workflows

Der folgende Abschnitt basiert auf [VSS+07] und [Vrh11] Beim ersten Ansatz handelt es sich um eine Optimierungsstrategie für datenintensive Workflows, die auf Workflowsprachen basieren, bei denen Datenverarbeitungsoperationen bzw. SQL-Anweisungen in Form von *dedizierten SQL-Aktivitäten*

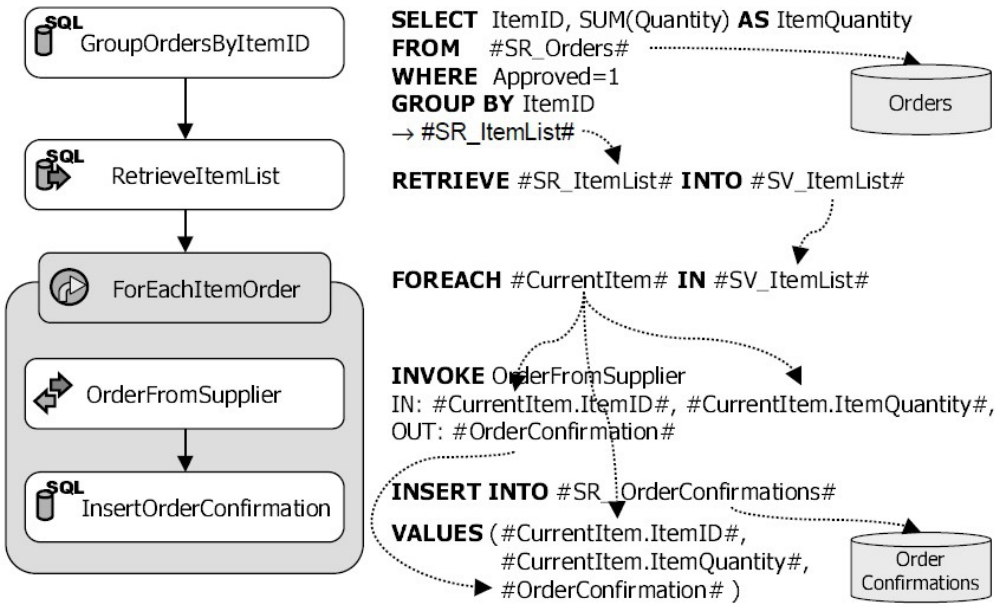


Abbildung 5.1: Beispiel-BPEL/SQL Workflow [VSS+07]

auf der Ebene der Prozesslogik integriert werden können. Der Optimierungsansatz ist insbesondere auf die Optimierung datenintensiver Geschäftsprozesse ausgerichtet und dementsprechend werden in [VSS+07] zur Veranschaulichung der Optimierunsmöglichkeiten *Geschäftsprozesse* betrachtet, die typische Arbeitsabläufe im Unternehmensbereich darstellen. Die Grundidee ist, die zu einem Workflowmodell gehörenden Datenverarbeitungsoperationen und -muster unter Verwendung von *Restrukturierungsregeln* derart zu modifizieren, dass deren effizientere Verarbeitung durch ein Workflow- bzw. Datenbanksystem ermöglicht wird. Die Optimierungsstragie schlägt unter anderem vor, möglichst viele dieser SQL-Aktivitäten direkt im darunterliegenden relationalen Datenbanksystem zu verarbeiten. Dabei werden Daten zwischen den Aktivitäten nur als Referenzen übergeben. Im Folgenden wird einleitend zunächst anhand eines Optimierungsszenarios veranschaulicht, wie die regelbasierte Optimierung abläuft (Abschnitt 5.1.1). Anschliessend werden die Grundlagen dieses Optimierungsansatzes beschrieben (Abschnitt 5.1.2). Zum Abschluss werden die wichtigsten Komponenten bei der Restrukturierung von Workflows erläutert (Abschnitt 5.1.3).

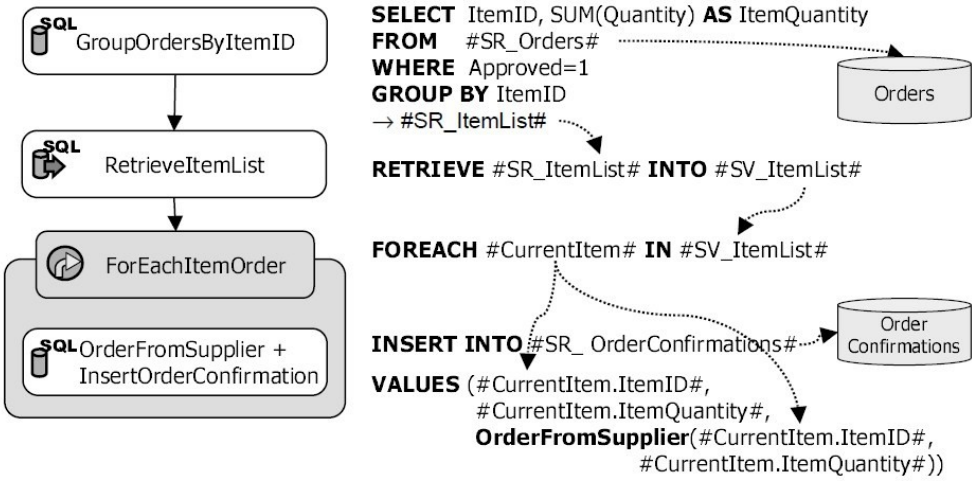
5.1.1 Regelbasierte Optimierung eines Beispiel-Workflows

Bei dem Beispielszenario handelt es sich um die Bearbeitung von Bestellungen (Abbildung 5.1). Dabei nimmt der Prozess eine Menge von Bestellungen an und entscheidet, ob sie direkt bearbeitet werden sollen oder nicht. Ist Letzteres der Fall, muss ein Angestellter eine Bestellung gemäss der Geschäftspolitik bewilligen. Wurde die Bestellung abgesegnet, schickt der Prozess Benachrichtigungen über abgelehnte Bestellungen raus. Die verbleibenden Bestellungen werden gleichzeitig verarbeitet. Dieser Teil des Gesamtszenarios, wie z.B. die automatische Verarbeitung von bewilligten Bestellungen ist im linken Teil der Abbildung 5.1 abgebildet. Der Prozess wurde dabei in BPEL-SQL modelliert. Die Aktivität GroupOrdersByItemIDs erzeugt eine Menge von Bestellungen, wobei für jede Bestellung eine *itemID* und die erforderliche Menge angegeben ist. Eine Aktivität ForEachItemOrder iteriert in einer Schleife über diese Bestellungen. Innerhalb dieser Schleife wird ein Web Service OrderFromSupplier aufgerufen, der die Bestellungen an die Lieferanten weiterleitet. Diese wiederum geben Auskunft darüber, ob sie die nachgefragten Waren liefern können oder nicht. Diese Bestätigungen seitens der Lieferanten werden über eine Aktivität InsertOrderConfirmation persistent gespeichert.

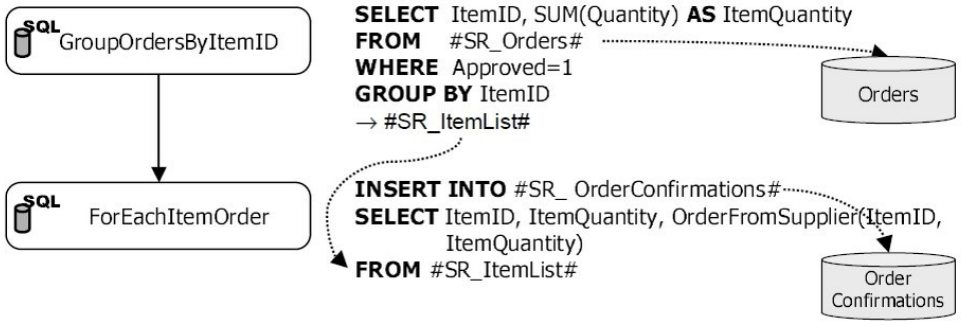
Im rechten Teil der Abbildung 5.1 sind die entsprechenden SQL-Anweisungen und zusätzliche Informationen zum Beispielprozess angegeben. BPEL Variablen werden dabei von Raute-Zeichen umfasst. Alle Bestellungen sind in der Tabelle Orders gespeichert, welche durch die Set Referenz Variable *SR Orders* referenziert wird. Die Aktivität GroupOrdersByItemID selektiert mit Hilfe einer Select-Anweisung einzelne Tupel aus der Menge der bestätigten Bestellungen und speichert sie in in der Set Referenz-Variable *SR ItemList*. Durch die Aktivität RetrieveItemList werden diese Bestellungen aus *SR ItemList* in die Set Variable *SV ItemList* geschrieben. Durch die Aktivität ForEachItemOrder beginnt ein Schleifendurchlauf, der in jedem Iterationsschritt ein Tupel aus *SV ItemList* ausliest und an die Variable *CurrentItem* bindet. Die Attribute ItemID und ItemQuantity der Variable *CurrentItem* werden als Eingabewerte an die Aktivitäten OrderFromSupplier und InsertOrderConfirmations übergeben. Die Letztere schreibt mit Hilfe einer Insert-Anweisung die Ergebnisse des Web-Service-Aufrufs in eine Tabelle *OrderConfirmations*. Diese Tabelle wiederum wird von einer Set Referenz Variable *OrderConfirmations* referenziert.

In Abbildung 5.2 wird die schrittweise Restrukturierung und der schliesslich aus den Transformationen resultierende Workflow dargestellt. Im ersten Schritt wird der Web-Service-Aufruf OrderFrom-Supplier mit der SQL Anweisung der Aktivität InsertOrder- Confirmation verschmolzen. Dies wird ermöglicht durch die Definition einer benutzerdefinierten Funktion OrderFromSupplier (siehe Teilschritt a)). Der resultierende Workflow dient als Grundlage für den folgenden Transformationsschritt, bei dem die tupelorientierte Schleifenoperation in eine einzelnde mengenorientierte SQL-Aktivität umgewandelt wird. Somit wird das gesamte Schleifenkonstrukt der Aktivität ForEachItemOrder durch eine SQL-Aktivität ersetzt (siehe Teilschritt b)). Beim letzten Transformationsschritt wird diese SQL-Aktivität mit der Aktivität GroupByItemID verschmolzen (siehe Teilschritt c)).

Als Endresultat ergibt sich eine Aktivität, die dieselbe Funktion erfüllt wie der ursprüngliche Workflow. Der bedeutende Unterschied liegt jedoch darin, dass alle mengenorientierten Datenverarbeitungsoperationen durch das darunterliegende Datenbanksystem innerhalb einer SQL-Anweisung ausgeführt werden. Durch diese Massnahmen können unabhängig vom verwendeten DBMS, wie in [Vrh11] nachgewiesen, erhebliche Verbesserungen in der Laufzeit erzielt werden. Dies ist zuruckzuführen auf ein im Vergleich zum ursprünglichen Workflow geringeres Datenvolumen, das zwischen der



(a) Process 'Order Processing' After First Optimization Step



(b) Process 'Order Processing' After Second Optimization Step

(c) Process 'Order Processing' After All Optimization Steps

Abbildung 5.2: schrittweise Optimierung des Beispiel-Workflows [VSS+07]

Ebene des DBMS und der Ebene des Workflows übertragen wird. Des Weiteren ist insgesamt der Verarbeitungsaufwand durch die Ausführung einer einzelnen SQL-Anweisung geringer als bei der Verarbeitung einer Menge von SQL-Anweisungen. Schliesslich ergeben sich durch den neuen Workflow weitere Optimierungsmöglichkeiten für das Datenbanksystem.

5.1.2 Grundlagen der regelbasierten Optimierung von datenintensiven Workflows

Vhrovnik stellt in seiner Arbeit ein Rahmenwerk für die Optimierung der Datenverarbeitung in Geschäftsprozessen vor. Dieses Rahmenwerk basiert auf *Restrukurierungs-Regeln*, welche Datenverarbeitungsoperationen in datenintensiven Geschäftsprozessen derart transformieren, dass eine effizientere Verarbeitung von Daten ermöglicht wird, ohne die ursprüngliche Ausführungssemantik des Prozesses zu verändern. Die Regeln wiederum basieren auf einem semi-prozeduralen *Graphenmodell*, das sowohl Kontroll- als auch Datenflussabhängigkeiten des Geschäftsprozesses explizit darstellt. Des Weiteren wird eine mehrstufige *Kontrollstrategie* für den Optimierungsprozess definiert. Als Grundlage für die Optimierungsszenarien werden Workflows betrachtet, die in der Workflow-Sprache **BPEL-SQL** beschrieben sind. Bei dieser Sprache handelt es sich um eine Erweiterung der Workflow-Sprache BPEL. Datenverarbeitungs-Operationen werden als dedizierte Aktivitäten, sogenannte *SQL-spezifische Aktivitätstypen* definiert und können somit auf der Ebene der Prozesslogik in die Workflows eingebunden werden. D.h. SQL-Funktionalität wird nicht, wie es in BPEL üblich ist, in einem Web-Service gekapselt, sondern in die Workflowbeschreibung eingebettet. Dadurch gehört die auszuführende SQL-Anweisung direkt zur Beschreibung der Aktivität und ist somit Bestandteil eines Workflowmodells.

Zu den grundlegenden Konzepten von BPEL/SQL gehören neben SQL-spezifischen Aktivitätstypen auch mengenbasierte Datenstrukturen zur Verarbeitung relationaler Datenmengen. Zur Realisierung einer mengenbasierten Datenverarbeitung auf der Workflowebene bietet BPEL/SQL folgende SQL-spezifische Aktivitätstypen: *<sql> Aktivitäten* führen SQL-Anweisungen auf einem relationalen Datenbanksystem aus. Es werden unter anderem Anfragen und Aufrufe von Stored-Procedures unterstützt. Die im Workflow definierten BPEL-Variablen können als Eingabeparameter einer SQL-Anweisung verwendet werden, d.h. <sql> Aktivitäten bieten Lese- und Schreibzugriff auf BPEL-Variablen. Die Beschreibung einer <sql> Aktivität enthält eine SQL-Anweisung und das Datenbanksystem, auf dem die SQL-Anweisung ausgeführt werden soll. Zur Laufzeit wird die SQL-Anweisung einer <sql> Aktivität an das zu Grunde liegende Datenbanksystem gesendet und dort ausgeführt. Bei einer SQL-Anfrage wird die resultierende Ergebnismenge nicht an die <sql> Aktivität zurückgesendet, sondern im Datenbanksystem materialisiert. Eine *<retrieveSet>* Aktivität lädt eine Anfrageergebnismenge in einen Workflow. Bei diesem Materialisierungsschritt werden die relationalen Daten in einer mengenbasierten XML-RowSet-Datenstruktur bereitgestellt. Variablen, die an Tabellen in relationalen Datenbanksystemen gebunden werden, werden als *Set-Referenz-Variablen* bezeichnet. Als *set variables* werden Variablen für mengenorientierte Datenstrukturen bezeichnet, die Tabellen repräsentieren, welche im Datenbanksystem materialisiert werden. Ein Materialisierungsschritt erfolgt in einer <retrieveSet> Aktivität, die relationale Daten lokal im Workflow in Set-Variablen verfügbar macht.

Jede Ausführung einer <sql> Aktivität verursacht Ausführungskosten auf der Workflow und Datenebene. Um die Laufzeit eines BPEL/SQL-Workflows zu reduzieren, müssen folglich insbesondere die Ausführungskosten der darin eingebetteten <sql> Aktivitäten minimiert werden. Dies lässt sich

erreichen, indem suboptimal modellierte Datenverarbeitungoperationen, die auf <sql> Aktivitäten basieren, identifiziert und durch effizientere Strukturen ersetzt werden. Die verschiedenen Optimierungstechniken des Rahmenwerks beseitigen unter Nutzung des Wissens über die Ausführungslogik eines Workflowmodells, suboptimale Datenverarbeitungsstrukturen in einem datenintensiven Workflowmodell. Dieses Wissen über die Ausführungslogik eines Workflowmodells steht einem einzelnen Datenbanksystem hingegen nicht zur Verfügung. Zu diesen Optimierungstechniken zählen beispielsweise die Kombination bzw. Parallelisierung von SQL-Anweisungen sowie das Zusammenfassen von SQL-Anweisungen, Web- Service-Aufrufen und Zuweisungsoperationen. Darüber hinaus können Kontrollflussmuster in *effizientere Strukturen* umgewandelt werden. Ebenso kann durch eine geeignete Optimierung der SQL-Anfragen eine Minimierung der von einem Datenbank- an ein Workflowsystem zu übertragenden Ergebnismengen erreicht werden. Des Weiteren kann es gewinnbringend sein, SQL-Anweisungen mithilfe von Stored-Procedures von der Workflow auf die Datenebene zu verlagern. Dazu werden geeignete Datenverarbeitungsoperationen in einer Workflowbeschreibung durch entsprechende Stored-Procedures ersetzt, welche die entsprechenden Datenverarbeitungsoperationen direkt auf der Datenebene effizient ausführen können.

5.1.3 Komponenten bei der regelbasierten Optimierung

Als Grundlage für die Definition und Anwendung der Restrukturierungsregeln auf einen datenintensiven Workflow dient das sogenannte *Prozessgraphenmodell* (PGM). Restrukturierungsregeln werden auf dieser internen Repräsentation eines Workflowmodells angewendet. Dabei werden in einem PGM nur die für eine Optimierung relevanten Teile eines Workflows dargestellt. Die Architektur des PGM/F-Systems ist in Abbildung 5.3 dargestellt. Zentrale Komponente ist der *PGM-Optimierer*, der für die heuristische Benutzung der Restrukturierungsregeln auf eine PGM-Repräsentation verantwortlich ist. Diese Komponente greift auf die verfügbare *Regelbasis* zurück und steuert die Nutzung dieser Regelbasis mit Hilfe einer *Kontrollstrategie*.

Jede Restrukturierungsregel besteht aus zwei Teilen: Einem *Bedingungs-* und einem *Aktionsteil*. Der *Bedingungsteil* definiert die Bedingungen bezüglich Daten-, Kontrollfluss und Kommunikationsabhängigkeiten, die gelten müssen, damit eine Regel auf ein Workflowmodell korrekt angewendet werden kann. Des Weiteren stellen diese Bedingungen sicher, dass die Ausführungssemantik des ursprünglichen Workflowmodells nicht verändert wird. Im *Aktionsteil* einer Regel sind die Transformationsschritte definiert, die auf jene Teile des Workflowmodells angewandt werden, welche den Bedingungsteil einer Regel erfüllen. Ferner entscheidet der PGM-Optimierer mit Hilfe der *Kontrollstrategie* darüber, auf welche Teile eines Workflowmodells und in welcher Reihenfolge die Restrukturierungsregeln angewendet werden sollen.

Der Optimierungsablauf beginnt, indem ein Workflowmodell in die interne *PGM-Repräsentation* transformiert wird. Der gewählten Kontrollstrategie folgend, wendet der PGM-Optimierer anschliessend die Restrukturierungsregeln an, indem er Muster innerhalb einer PGM-Repräsentation ausfindig macht, welche den Bedingungsteil einer Regel erfüllen. Diese werden gemäss dem Aktionsteil einer Regel transformiert. Da eine PGM-Repräsentation Aktivitäten mit ihren Daten-, Kontrollfluss- und Kommunikationsabhängigkeiten explizit darstellt, können Muster einfach identifiziert werden. Zuletzt wird die aus dem Optimierungsprozess resultierende PGM-Repräsentation in ein entsprechendes Workflowmodell zurücktransformiert. Der *PGM-Optimierer* akzeptiert auch Beschreibungen von

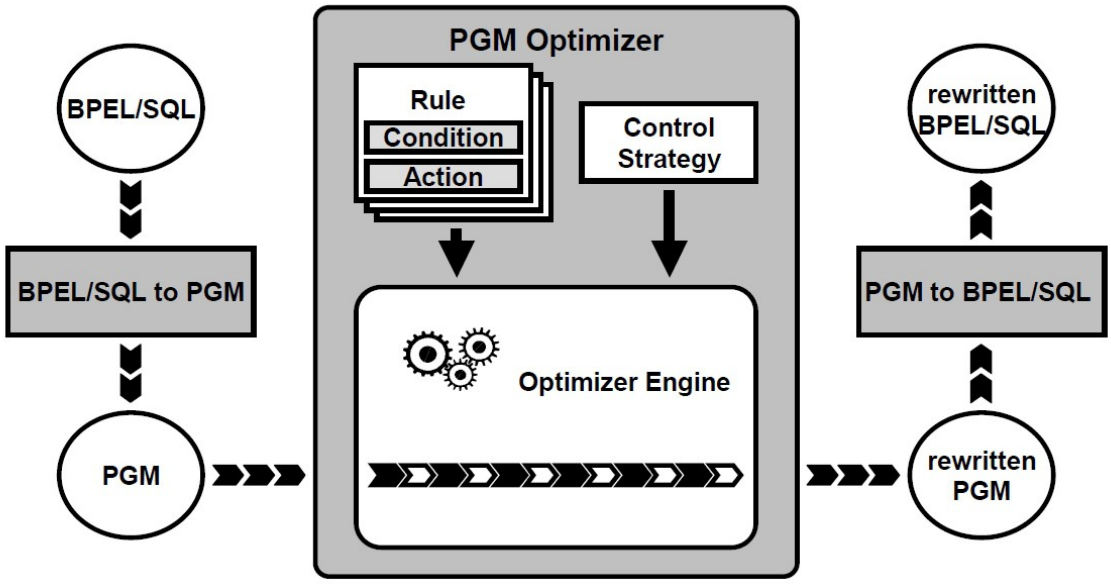


Abbildung 5.3: Architektur des PGM/F [VSS+07]

SQL-basierten Stored- Procedures, die in *SQL/PSM* definiert werden, als mögliche Eingaben. SQL/PSM ist eine Erweiterung von SQL, die prozedurale Kontrollflusskonstrukte einführt, mit denen SQL-Anweisungen verbunden werden können. Des Weiteren kann der beschriebene Optimierungsansatz auch auf die Definition SQL-basierter Stored-Procedures angewendet werden.

Das *Prozessgraphenmodell* ist eine generische Repräsentation, das bekannten Query Graph Modellen ähnelt und eine abstrakte und sprachneutrale Darstellung prozedural modellierter Workflowmodelle erlaubt. Eine PGM-Repräsentation entspricht einem gerichteten Graphen, dessen Knoten Worklfow-Aktivitäten repräsentieren und dessen Kanten Daten-, Kontrollfluss- und Kommunikationsabhängigkeiten zwischen Aktivitäten und externen Systemen (z.B. relationale Datenbanksysteme oder Web Services) explizit darstellen. Zur Definition von Kontrollflussmustern wie z.B. Joins oder Splits werden spezielle Aktivitäten bereitgestellt. Die Ausführung von SQLAnweisungen oder Aufrufe von Stored-Procedures können über SQL-Aktivitäten dargestellt werden. Invoke-Aktivitäten hingegen repräsentieren Web-Service-Aufrufe Daten, die innerhalb eines Workflows verarbeitet werden, werden in Variaben gespeichert. Ein PGM-Graph enthält somit alle für die Definition, Implementierung und Anwendung der Restrukturierungsregeln notwendigen Informationen. Des Weiteren kann das PGM einfach erweitert werden, beispielsweise um es an weitere Erweiterungen von BPEL oder weitere Konstrukte anderer Workflowsprachen anzupassen. Für BPEL-Erweiterungen müssen Abbildungen von BPEL/SQL zu PGM und PGM zu BPEL/SQL abgeändert werden, für andere Workflowsprachen müssen komplett neue Abbildungen definiert werden. Falls eine neue Workflowsprache weitere Optimierungsregeln zulässt, können diese neuen Optimierungsregeln zur Regelbasis hinzugefügt werden.

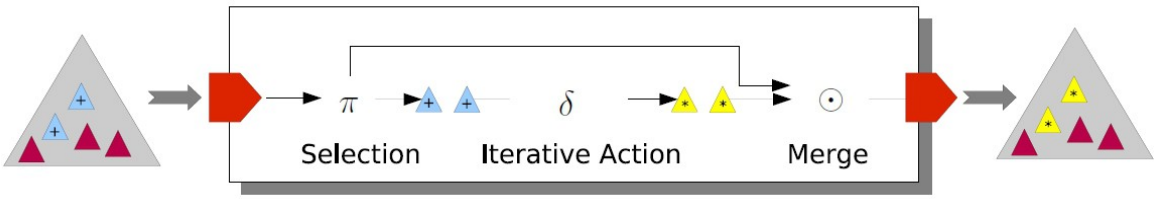


Abbildung 5.4: Konzept eines COMAD actors [MBZL09]

Bei den **Restrukturierungsregeln** handelt es sich sowohl um neue als auch um bereits existierende Ansätze zur Optimierung von Datenverarbeitungsoperationen, die auf den Kontext datenintensiver Workflows übertragen werden können. Die verschiedenen Optimierungsregeln werden im Kapitel Kapitel 7 kurz beschrieben.

5.2 Design und Optimierung von wissenschaftlichen Workflows

Dieser Abschnitt basiert auf [Zin10] und [MBZL09]. Beim zweiten Ansatz, der in [Zin10] vorgestellt wird, werden Vorschläge für das Design und die Optimierung von wissenschaftlichen Workflows gemacht. Die Vorschäge stützen sich vor allem auf einem Design-Paradigma **Virtual Data Assembly Line** (**VDAL**), welches in Abschnitt 5.2.1 vorgestellt wird. Anschliessend werden zwei der auf VDAL basierenden Optimierungsvorschläge vorgestellt (Abschnitt 5.2.2 und Abschnitt 5.2.3).

5.2.1 Virtual Data Assembly Line

Der Entwurf von wissenschafltichen Workflows mit datenflussorientieren Sprachen resultiert oft in unübersichtlichen, unflexiblen Workflows. Z.B. müssen innerhalb der Workflows häufig parameterreiche Funktionen aufgerufen und komplexe, benutzerdefinierte Datenstrukturen verwendet werden, woraus sich komplexe Verknüpfungen ergeben. Dazu trägt auch bei, dass IF- oder Schleifen-Konstrukte „händisch“, durch mehrere Aktivitäten und Datenflussstränge konstruiert werden müssen. Des Weiteren wird der Workflow bei verteilter Ausführung von wissenschaftlichen Workflows durch Aktivitäten, welche die verteilte Ausführung koordinieren, noch komplexer. Das Design-Paradigma VDAL soll diesen Herausforderungen entgegnen. Dabei handelt es sich um eine Erweiterung von COMAD ([MB05], [MBL06]), bei dem wissenschaftliche Workflows als Pipelines von sogenannten (*actors*, dargestellt werden, die auf einem Strom von Daten arbeiten. *Actors* sind Methoden, die Input- und Output-Ports besitzen und über datenübertragende Kanten, den *channels* miteinander verbunden sind. Der *Read Scope* und der *Write Scope* eines *Actors* legen fest, welche Daten aus einem Datenstrom (z.B. eine XML-Datei) gelesen und wo Daten im Datenstrom wieder eingefügt werden.

Bei VDAL besitzt jeder *actor* einen einzigen Input- sowie einen einzigen Output-Port, so dass VDAL-Workflows aus einer **linearen** Aneinanderreihung von *actors* bestehen. Durch diese Pipeline fliessen

Daten in Form von XML-ähnlichen, baumartigen Datenstrukturen, sogenannten **Collections**, welche ähnlich einem SAX-Parser mit XML-Sprachen wie XPath verarbeitet werden. Die *actors* werden in eine separate Schicht **Configuration Shell** verpackt, in der festgelegt werden kann, wie sie mit den an ihnen vorbeifliessenden Daten interagieren. Durch diese Eigenheiten werden komplexe Verknüpfungen und komplexe Datenstrukturen vermieden.

5.2.2 Ausnutzung von Datenparallelismus

In [Zin10] wird ein Ansatz vorgeschlagen, der *Datenparallelismus* ausnutzt, um VDAL-Pipelines zu optimieren. Dabei werden neue Berechnungsstrategien für das **Map-Reduce**-Verfahren verwendet. Bei VDAL fliessen XML-Strukturen durch eine XML-verarbeitende Pipeline. In Abbildung 5.5 ist eine solche Pipeline gezeigt, welche aus fünf Schritten besteht. Jeder Schritt arbeitet ihrem *scope* entsprechend auf bestimmtem Fragmenten (Unterbäumen) der Daten, die er durch XPath-Ausdrücke bestimmt (*scope match*). Der erste Schritt z.B. hat den *scope* B, d.h. er kann innerhalb aller Unterbäume mit B als Wurzel beliebige Veränderungen durchführen. Der Schritt mit *scope* D jedoch kann nur Veränderungen auf den Blättern der Baumstrukur ausführen, die in der Abbildung unten links abgebildet ist.

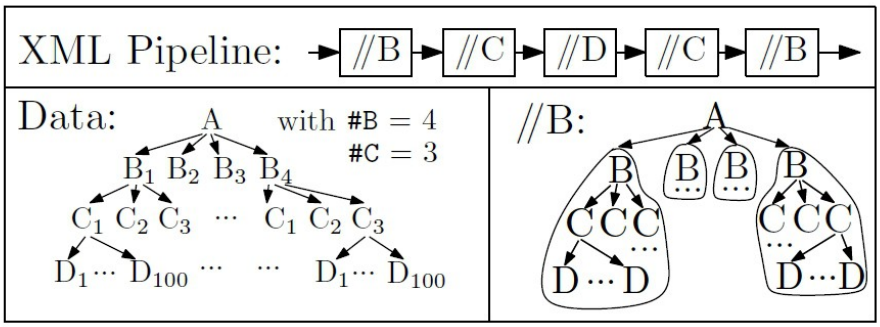


Abbildung 5.5: Beispiel einer XML-Pipeline [Zin10]

Um Datenparallelismus auszunutzen, werden *scope matches* auf sogenannte *work pieces* abgebildet, die dann vom *MapReduce*-Verfahren parallel verarbeitet werden. Unten rechts in der Abbildung 5.5 ist veranschaulicht, wie die Daten für den *scope* B partitoniert werden (für die Schritte 1 und 5). Ein naiver Einsatz des *MapReduce*-Verfahrens jedoch kann zu Flaschenhälsen in den Phasen Splitting und Regrouping des parallelen *MapReduce*-Verfahrens führen, wenn dies durch eine einzelne, globale Task erledigt wird.

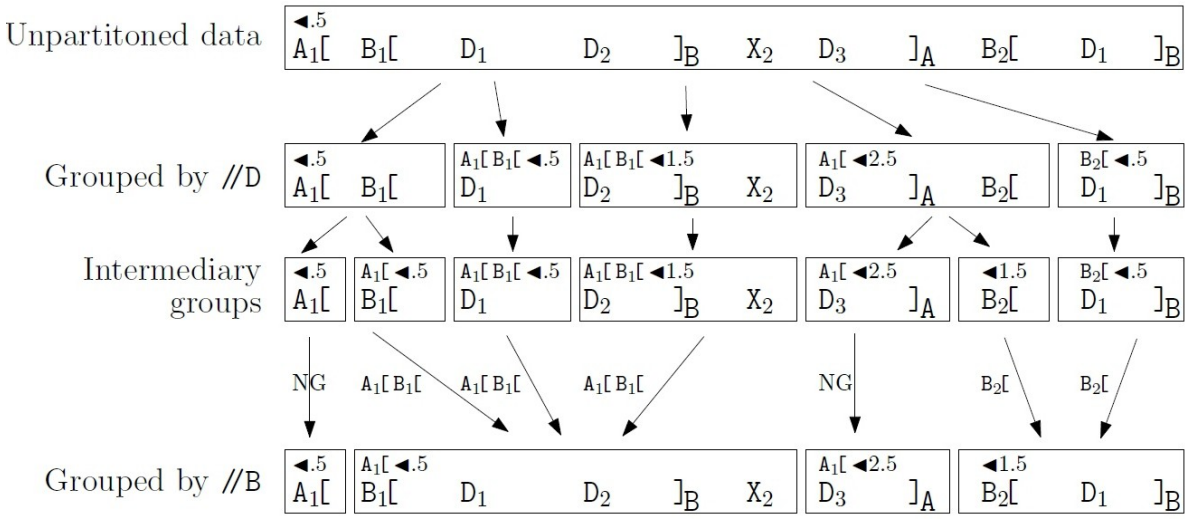


Abbildung 5.6: Die parallele Strategie: Veränderung der Fragmentierung von //D zu //B [Zin10]

Es werden drei Parallelisierungsstrategien vorgestellt, welche für jede Aktivität in der XML Pipeline ein MapReduce-Verfahren anwenden, der die Zwischenschritte *split*, *map* und *reduce* beinhaltet. Der **naiven Ansatz** wendet im Grunde genommen, das MapReduce-Verfahren unverändert auf XML-Daten an, so dass das Splitting und Regrouping Flaschenhälse darstellen. Beim **XMLFS-Ansatz** wird der Flaschenhals in der Phase Reduce vermieden, indem XML-Strukturen auf eine verteiltes Datei-System abgebildet werden, kein explizites Grouping mehr notwendig ist. Die Aufteilung der Daten stellt jedoch immer noch einen Flaschenhals dar. Beim **Parallelen Ansatz** werden die Phasen Splitting und Grouping unter Nutzung der Gruppier- und Sortier-Fähigkeiten von MapReduce parallel ausgeführt, wobei Daten je nach dem *read scope* der Nachfolgeaktivität in Fragmente gruppiert werden. Abbildung 5.6 zeigt ein Beispiel, bei dem die Veränderung der Fragmentierung von *scope* D zu *scope* B verdeutlicht wird. Dabei erfolgt der Datenstrom als Tokensequenz. Die Fragmente sind als Rechtecke dargestellt und durch einen Schlüssel, bestehend aus einer ID (z.B. Dezimalzahl, die um 0.5 kleiner ist als die ID des ersten Tokens im Fragment) und dem Pfadnamen, durch den der erste Token des Fragments erreicht werden kann, eindeutig identifizierbar. Beim Übergang von Zeile 2 zu Zeile 3 (Map-Phase) erfolgt das parallele Splitting der Fragmente und von Zeile 3 zu Zeile 4 (Reduce-Phase) werden die Fragmente anhand ihrer Schlüsselwerte nach scope B gruppiert.

Abbildung 5.7 zeigt eine Auswertung der verschiedenen Strategien hinsichtlich der Ausführungszeit im Vergleich zu einer seriellen Ausführung eines Beispiel-Workflows. Alle drei vorgeschlagenen Strategien übertreffen die serielle Ausführung. Enthalten die Eingabedaten nur eine *Collection*, liegt die Geschwindigkeitssteigerung bei etwa dem 13-fachen der Ausführungszeit der seriellen Ausführung. Bei 10 Collections ist die Ausführung etwa 20-mal schneller.

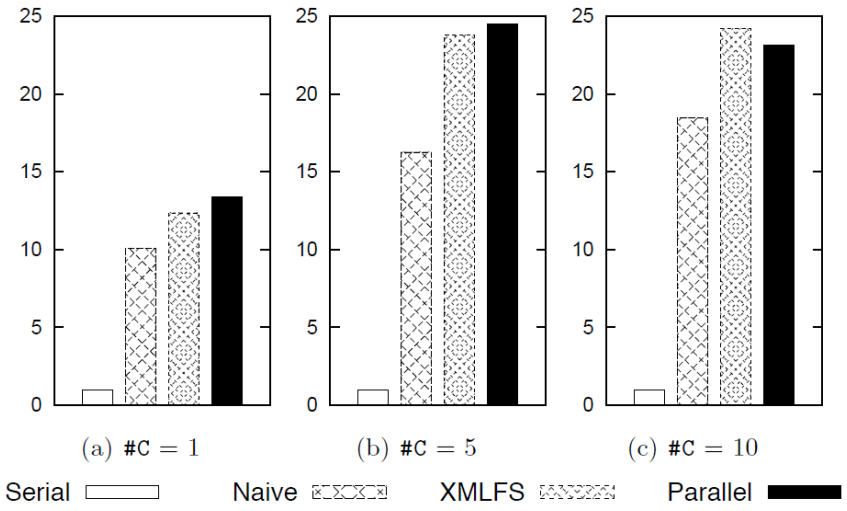


Abbildung 5.7: Auswertung der Ausführungszeiten von serieller Ausführung im Vergleich zu MapReduce-Strategien [Zin10]

5.2.3 Minimierung des Datentransfers

Beim Design-Paradigma VDAL gestaltet sich als Problem, dass Daten an *actors* geschickt werden, selbst wenn diese den Grossteil dieser Daten nicht bearbeiten müssen und weiterleiten werden. Ein weiterer Optimierungsansatz namens **X-Scissor** (**X-CSR**) soll diesen (unnötigen) Datenverkehr minimieren.

Hierzu werden VDAL Workflows um weitere Komponenten (*distributors* und *mergers*) erweitert, die den Datenstrom dynamisch aufteilen und Daten an die richtigen Abnehmer senden. Hierzu wird mit Hilfe der Workflow-Informationen analysiert, wo im Workflow welche Teile des Inputs zuerst verwendet werden.

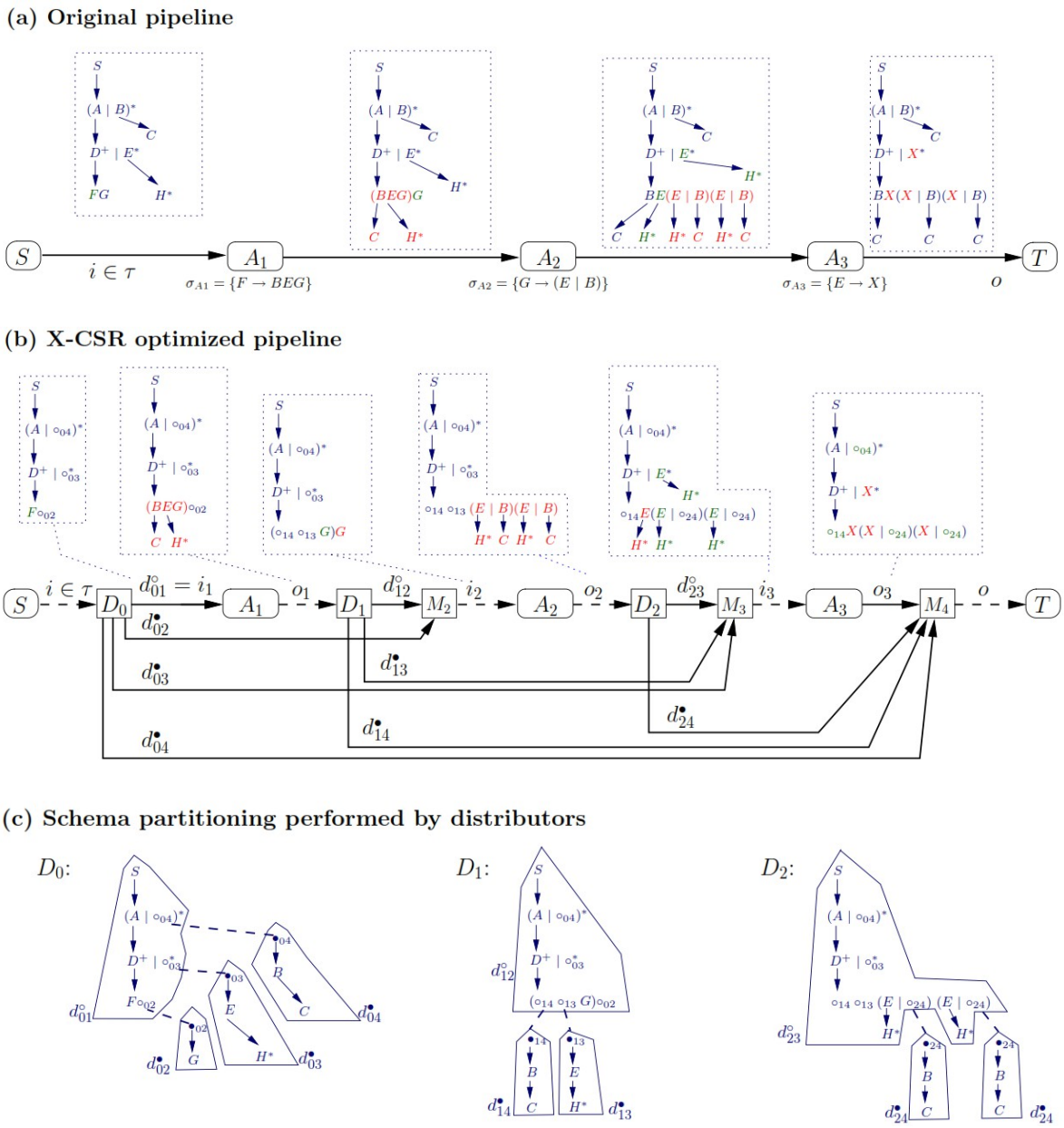


Abbildung 5.8: Das Ausschneiden von Datenfragmenten (X-Scissor) [Zin10]

Beim Aufteilen des Haupt-Datenstroms werden an den Trennpunkten nummerierte, sogenannte *hole* Markierungen in den Haupt-Datenstrom eingefügt. Vom Haupt-Datenstrom abgespaltene Fragmente werden entsprechend mit nummerierten, sogenannten *filler* Markierungen gekennzeichnet. Somit können beim Wiedereinfügen der Fragmente in den Haupt-Datenstrom, *hole* und *filler* Markierungen zugeordnet und die ursprüngliche Reihenfolge des Fragmente wiederhergestellt werden.

Das Verfahren wird in der Abbildung 5.8 veranschaulicht. In a) ist dabei eine Pipeline bestehend aus den Aktivitäten A1, A2 und A3 und ein Input-Schema gezeigt. Teilabbildung b) zeigt die optimierte Pipeline mit den relevanten Schemata. In c) werden schliesslich die von den *distributors* generierten Schemapartitionierungen gezeigt. Dabei sind *hole*-Markierungen als durchsichtige Kreise, *filler*-Markierungen als schwarz ausgefüllte Kreise dargestellt. Da der *read scope* von A1 F und der relevante Pfad, der zu F führt, S/A/D/F ist, wird der entsprechende Teilbaum ohne die Teilbäume B/C und E/H an A1 geschickt. Knoten G ist jedoch im *scope* des actors A2. Der *distributor* D_0 schneidet den Teilbaum, der G enthält, aus und sendet ihn direkt vor den *actor* A2, wo er in den Datenstrom eingefügt werden kann. Ebenso wird der Teibaum, die E enthält von A1 und A2 ignoriert und direkt vor A3 geschickt, da E in dessen *read scope* liegt. B- und C-Daten liegen in keinem *read scope* und werden an das Ende des Hauptdatenstroms geschickt.

Die durch die Optimierungen erzielbaren Leistungssteigerungen wurden mit Hilfe von Beispielworkflows gemessen und werden in Abbildung 5.9 dargestellt. Dabei wurden ein serieller, ein paralleler, sowie ein gemischter Datenfluss betrachtet und die Ausführungszeiten, sowie die Ersparnisse beim Datenverkehr analysiert. Generell ist zu beoabachten, dass bei allen drei Datenflussvarianten erhebliche Einsparungen beim Datenverkehr (bis zu 56%) und erheblich kürzere Ausführungszeiten (bis zu 69%) gegenüber einem unpotimierten Workflow erzielt werden.

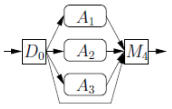
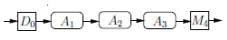
	Scenario	Input Data / Input Workflow	Actual Dataflow	Data Shipped (MB) orig.	Data Shipped (MB) opt.	Exec. Time (sec) orig.	Exec. Time (sec) opt.
(a) Parallel	$A_1 : \langle \mathtt{a} \rangle \mapsto \langle \mathtt{u} \rangle$ $A_2 : \langle \mathtt{b} \rangle \mapsto \langle \mathtt{v} \rangle$ $A_3 : \langle \mathtt{c} \rangle \mapsto \langle \mathtt{w} \rangle$	$\mathtt{s}[\ (\mathtt{a}[\mathtt{z}]\ \mathtt{b}[\mathtt{z}]\ \mathtt{c}[\mathtt{z}]\ \mathtt{w}[\mathtt{z}]) * i\ ]$ / $A_1 \to A_2 \to A_3$		$80i$	$35i$ -56%	$\approx 3.6i$	$\approx 1.1i$ -69%
(b) Serial	$A_1 : \langle \mathtt{a} \rangle \mapsto \langle \mathtt{b} \rangle$ $A_2 : \langle \mathtt{b} \rangle \mapsto \langle \mathtt{c} \rangle$ $A_3 : \langle \mathtt{c} \rangle \mapsto \langle \mathtt{w} \rangle$	$\mathtt{s}[\ \mathtt{a}[\mathtt{z}] * 4i\ ]$ / $A_1 \to A_2 \to A_3$		$80i$	$80i$ 0%	$\approx 3.6i$	$\approx 2.6i$ -28%
(c) Mixed	$A_1 : \langle \mathtt{a} \rangle \mapsto \langle \mathtt{b} \rangle$ $A_2 : \langle \mathtt{b} \rangle \mapsto \langle \mathtt{v} \rangle$ $A_3 : \langle \mathtt{c} \rangle \mapsto \langle \mathtt{w} \rangle$	$\mathtt{s}[\ (\mathtt{a}[\mathtt{z}] * 2i)\ (\mathtt{c}[\mathtt{z}] * 2i)\ ]$ / $A_1 \to A_2 \to A_3$		$80i$	$50i$ -38%	$\approx 3.6i$	$\approx 2.2i$ -39%

Abbildung 5.9: Auswertung der Ausführungszeit und Reduktion des Datenverkehrs von X-CSR im Vergleich zur Standard-Ausführung [Zin10]

5.3 Deep Business Optimization

Ein weiterer, interessanter Optimierungsansatz wird in [NS11] beschrieben und widmet sich der Optimierung von Geschäftsprozessen. Bei diesem Ansatz wird eine Plattform vorgestellt, welche Analysten von Geschäftsprozessen die (semi-)automatische Aufdeckung von Optimierungsmöglichkeiten und die Anwendung von Optimierungstechniken ermöglicht. Dabei wird ein Katalog von formalisierten Optimierungstechniken kombiniert mit Datenanalyse und -integration. Das Hauptziel der Geschäftsprozessoptimierung ist generell die Auswahl des richtigen Prozessdesigns und die Anwendung der geeignetsten Optimierungstechniken auf einem existierenden Prozess. Bei konventioneller Geschäftsprozessoptimierung wird jedoch oft die Integration von heterogenen Prozess- und

operationalen Daten vernachlässigt, da davon ausgegangen wird, dass die relevanten Daten sich auf einer, einzelnen Datenquelle befinden. Ferner werden bei der Analyse des Prozesses und seiner Daten häufig nur einfache Messwerte erfasst und es kommen keine Data-Mining-Technologien zum Einsatz. Optimierungstechniken werden zudem isoliert voneinander, mit Bezug auf unterschiedliche Domänen und mit Hilfe verschiedener Formalismen definiert. Dadurch wird die Kombination verschiedener Optimierungstechniken erschwert.

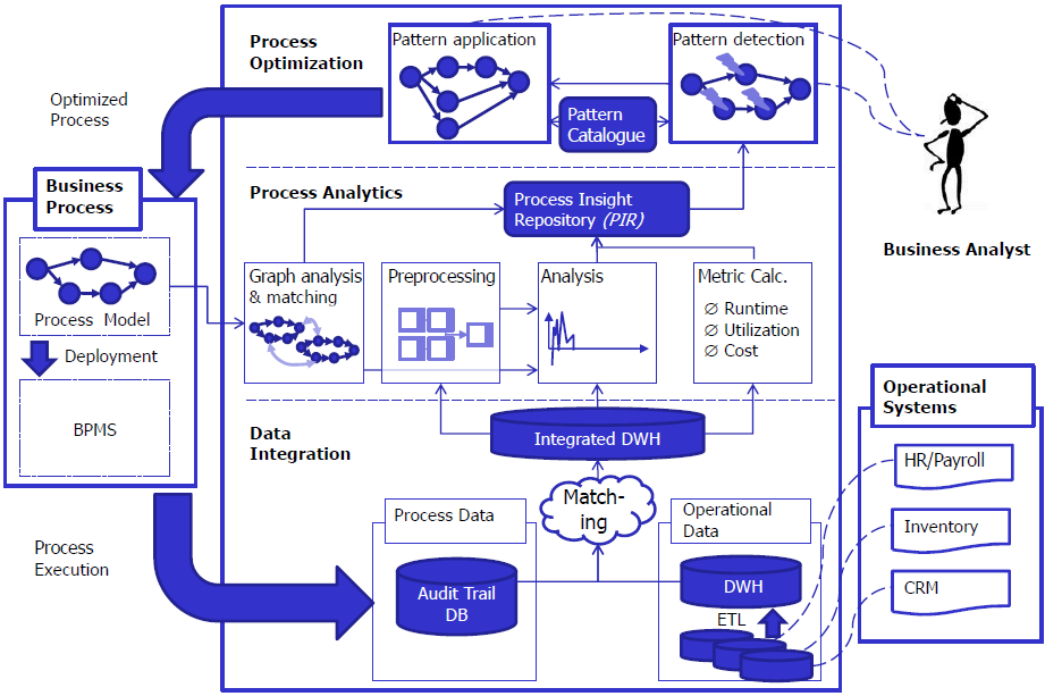


Abbildung 5.10: Übersicht dBOP-Plattform [NS11]

Um genannten Herausforderungen zu entgegnen, wird eine Architektur **deep Business Optimization Platform (dBOP)** vorgestellt, welche aus 3 integrierten Ebenen besteht (Abbildung 5.10)

- **Datenintegrationsebene**: Hier werden die Daten aus heterogenen Datenquellen integriert. Dabei liegt das Hauptaugenmerk auf der Integration von Prozess- und operationalen Daten.
- **Analyse-Ebene**: Basierend auf den Resultaten einer Graphenanalyse und dem Wissen über anzuwendende Optimierungsmuster werden in dieser Ebene mit Hilfe von Messungen und Data-Mining-Techniken Prozessdaten verarbeitet und analysiert.
- **Optimierungsebene**: Mit Hilfe einer Optimierungs-Engine und einem Katalog aus Optimierungsmustern werden auf dieser Ebene semiautomatisch Optimierungsmöglichkeiten des Prozesses aufgedeckt und die am besten geigneten Optimierungen angewandt.

Durch die Integration von Prozess- und operationalen Daten kann eine tiefgehende Prozessanalyse stattfinden. Dabei werden vom Analysten die bei beiden Datenmengen übereinstimmenden

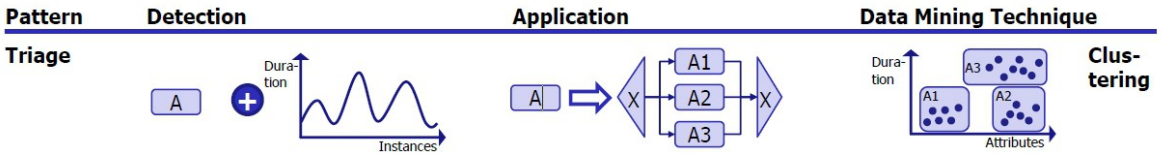


Abbildung 5.11: Das Triage-Pattern [NS11]

Attribute festgelegt und auf andere Prozesselemente übertragen (z.B. durch Zuweisungen). Die integrierten Daten werden schliesslich in einem gemeinsamen Data Warehouse gespeichert. In der Analyse-Phase wird das Prozessmodell und dessen Ausführungsverhalten analysiert. Dabei werden Übereinstimmungen mit anderen Prozessmodellen ermittelt und einfache Messungen durchgeführt um interessante Teilbereiche des Prozesses ausfindig zu machen. Für diese wird anschliessend mit Hilfe von Data-Mining-Technologien eine tiefergehende Analyse durchgeführt . Um die Auswahl der Data-Mining-Technologien einzugrenzen, sind dem ausgewählten Bereich, dem sogenannten Optimierungspattern intern eine bestimmte Menge von passenden Data-Mining-Algorithmen zugeordnet. Ein Beispiel ist in Abbildung 5.11 gezeigt. Das Triage-Pattern beschreibt den Sachbestand, dass manchmal bei der Ausführung einer Aktivität mehrere Varianten entstehen. Für eine grössere Transparenz spaltet das Pattern die Aktivität auf. Hier prüft der Analyzer zunächst, ob das Verhalten der Aktivität z.B. in Bezug auf Lebensdauer einen Grund für die Verwendung des Patterns liefert. Dann prüft ein Clustering-Algorithmus, der auf verschiedenen Teilmengen der Eingabedaten der Aktivität angesetzt wird, ob identifizierbare Varianten existieren, die sich genügend voneinander unterscheiden. Falls dies zutrifft, kann das Pattern angewandt werden.

Die Optimierungsphase verwendet die Resultate der Analyse-Phase und eine Menge von formalisierten Optimierungstechniken für die semi-automatisierte Aufdeckung und Implementierung von Optimierungsmöglichkeiten. Der Analyst legt als ersten Schritt die Hauptziele für eine Optimerung fest (z.B. kürzere Laufzeit oder Minimierung der Kosten). Der Optimierer versucht anschliessend im Prozess Instanzen der Optimierungspatterns zu finden, die in der Analysephase ermittelt wurden. Zum Schluss wird dem Analysten ein optimierter Prozess präsentiert, bei dem die Reihenfolge der Patterns so gewählt ist, dass mit grösster Wahrscheinlichkeit die optimalen Ergebnisse erzielt werden. Die zur Auswahl stehenden Optimierungspatterns werden in einem Katalog verwaltet und auf einen gemeinsamen Formalismus abgebildet. Im Katalog sind die Patterns z.B. nach den Veränderungen, die sie im Prozess implementieren, dem Grad mit dem sie Optimierungsziele unterstützen, nach der Prozessphase, in der sie eingesetzt werden können und nach ihren Einschränkungen insbesondere in Bezug auf Ausführungsreihenfolge bei Kombination mit anderen Patterns kategorisiert.

Für den Formalismus der Optimierungspatterns wird ein graph-basiertes Metamodell verwendet, das auf dem Prozessmodellgraph basiert, der in [LR00] vorgestellt wurde. Details sind ebenso dort zu finden. Vereinfacht beschrieben besteht ein PM-Graph aus einem Tupel (V, O, N, C, E, , o) wobei V eine endliche Menge von Prozessdatenelementen (Variablen), O eine endliche Menge von operationalen Daten, N eine endliche Menge von Prozessknoten, welcher die Menge von Aktivitäten NA, den Start- und den Endknoten, die Terminierungsknoten NT und die Kontrollknoten NC enthält (XOR and AND Fork/Join), C eine endliche Menge von Bedingungen, E N N C eine Menge von (Kontroll-) Konnektoren, : N C G $\longrightarrow$ (V) die Abbildung der Menge der Eingabedaten, mit (V) als Potenzmenge

Listing 5.1 Das Activity Eliminate Pattern [NS11]

```
Detection
Require: Similarity threshold T  [0, 1], Activity nodes NA of process graph G
Ensure: All found pattern instances
 instances = {}
 for all act  NA do
  for all succ  Successors (act,EC) do
   if SIM (act, succ)  T then
     instances = instances  newInstance (act, succ, SIM (act,succ))
   end if
  end for
 end for
return instances

Application
Require: Graph G, all instances, selected instance, analyst input
Ensure: The optimized (improved) process graph Gopt
 Gopt = G
 if confirms (instance, analyst) then
   DeleteNode (inst.succ)
   RemoveDependents (instances, inst)
end if
return Gopt
```

und o : N G → (V) die Abbildung der Ausgabedaten. Mit diesem Metamodell können einfache Optimierungspatterns formuliert werden. Als Beispiel ist das *Activity Elimination Pattern* in Listing 5.1 gezeigt. Dem Pattern liegt die Idee zu Grunde, dass eine hohe Ähnlichkeit von Prozessaktivitäten auf eine Redundanz deuten kann. Mit einer Ähnlichkeitsfunktion wird die Ähnlichkeit zweier Knoten berechnet, wobei Werte im Wertebereich zwischen 1 (hohe Ähnlichkeit) und 0 (geringe Ähnlichkeit) möglich sind. Bei ähnlichen Aktivitäten muss der Analyst entscheiden, ob die Aktivitäten überflüssig sind. Falls dieser die Überflüssigkeit bestätigt, wird die Aktivität aus dem Prozess entfernt.

6 Bewertung der Optimierungsansätze

Im Kapitel 5 wurden unterschiedliche Optimierungsansätze für datenintensive Workflows vorgestellt, mit denen teilweise erhebliche Leistungssteigerungen erzielt werden konnten, was aus den Auswertungen der jeweiligen experimentellen Testprojekte hervorgeht. Auch für Simulationsworkflows bieten diese Ansätze gegebenenfalls ein erhebliches Optimierungspotenzial, welches es auszunutzen gilt. In diesem Kapitel soll für die im vorherigen Kapitel betrachteten Ansätze erörtert werden, ob sie für die Anwendung auf datenintensive Simulationsworkflows, insbesondere auf die für diese Arbeit relevanten Beispielworkflows geeignet sind. Die Evaluierung des PGM-F-Rahmenwerks findet in Abschnitt 6.1 statt. Anschliessend folgt Abschnitt 6.2 die Bewertung des VDAL-Ansatzes. Zum Abschluss wird in Abschnitt 6.3 der dBOP-Ansatz evaluiert.

6.1 Bewertung des PGM-F-Optimierungsansatzes

Bei den in Abschnitt 5.1 vorgestellten Optimierungsszenarien werden hauptsächlich Geschäftsprozesse, d.h. Workflows aus dem Unternehmensbereich behandelt. Die den Optimierungsszenarien in Abschnitt 5.1 zu Grunde liegenden Workflows sind in einer *kontrollflussorientierten* Workflowsprache beschrieben. Trotz der grossen Datenmengen, die in wissenschaftlichen Workflows verarbeitet werden, werden mittlerweile auch sehr viele Simulationsworkflows mit Hilfe kontrollflussorientierter Sprachen wie z.B. BPEL beschrieben, wie auch die in dieser Arbeit betrachteten Beispielworkflows. Des Weiteren werden Workflows mit *eingebetteten Datenverwaltungs-Operationen* betrachtet. Diese Operationen werden auf den zu Grunde liegenden Datenressourcen ausgeführt, jedoch auf der Workflow-Ebene durch entsprechende Aktivitäten repräsentiert. Somit sind sie aktiver und auch oft laufzeitbestimmender Teil der Workflows. Die Optimierungsregeln zielen verstärkt auf die Optimierung solcher Aktivitäten ab. Für datenintensive Simulationsworkflows, die ähnliche, datenverarbeitende Aktivitäten enthalten, bieten sich die Regeln prinzipiell an. Bei den eingebetteten Operationen handelt es sich in Abschnitt 5.1 insbesondere um SQL-Anweisungen und den Workflows liegt als Workflowbeschreibungssprache *BPEL/SQL* sowie *SQL/PSM* zu Grunde. Da die in dieser Arbeit betrachteten Workflows mit BPEL/DM beschrieben sind bzw. werden, stellt dies einen Vorteil dar. Denn sowohl bei BPEL/SQL, als auch bei *BPEL/DM* handelt es sich um Erweiterungen von WS-BPEL. Somit ist eine grundlegende Ähnlichkeit der verwendeten Sprachen gegeben.

BPEL-DM erweitert WS-BPEL um spezifische Aktivitätstypen, sogenannte Data Management(DM) Aktivitäten, welche Workflow-Aufgaben darstellen, in die Datenverwaltungsoperationen eingebettet sind. Diese Operationen werden nahtlos auf den zu Grunde liegenden Datenressourcen ausgeführt. Im Folgenden werden die verschiedenen Aktivitäten und Konstrukte der beiden BPEL-Erweiterungen

aufgelistet, die die Realisierung von Datenverarbeitungsoperationen auf der Workflowebene ermöglichen. Unter einem jeweiligen Gesichtspunkt werden dabei die Konstrukte zusammengefasst, die Ähnlichkeiten aufweisen.

- BPEL/SQL bietet **<sql>** Aktivitäten an, mit denen SQL-Anweisungen auf einem relationalen Datenbanksystem ausgeführt werden. Unterstützt werden neben Anfragen, Datenmanipulations- (DML) und Datendefinitionsanweisungen (DDL) auch Aufrufe von Stored-Procedures. Dabei ist es möglich die im Workflow definierten BPEL-Variablen als Eingabeparameter innerhalb einer SQLAnweisung zu verwenden. In der Beschreibung einer <sql> Aktivität ist eine SQL-Anweisung und das Datenbanksystem enthalten, auf dem die SQL-Anweisung ausgeführt werden soll. Die Ergebnismenge einer Anfrage wird nicht an die <sql> Aktivität zurückgesendet, sondern im Datenbanksystem materialisiert, auf welche über eine Referenz zugegriffen werden kann. Den <sql> Aktivitäten in BPEL/SQL entspricht in BPEL-DM die Aktivität **IssueCommand**. Mit einer IssueCommand-Aktivität können auf der Datenressource ebenso Datenmanipulations- und -definitionssanweisungen ausgeführt werden. Neben einer BPEL-Variable *data source reference*, welche die zu Grunde liegende Datenressource spezifiziert, erwartet sie einen DM command als Eingabeparameter. Dieser wird auf der spezifizierten Datenressource ausgeführt und muss dementsprechende in der Befehlssprache der jeweiligen Datenressource definiert sein (z.B. SQL, XQuery, Shell usw.).
- Anfrageergebnismengen können in BPEL/SQL mit Hilfe von **<retrieveSet>** Aktivitäten in einen Workflow geladen werden. Die materialisierten, relationalen Daten werden dabei in einer mengenbasierten XML-RowSet-Datenstruktur im Workflow-Kontext bereitgestellt. Dieser <retrieveSet> Aktivität in BPEL/SQL entspricht die BPEL-DM-Aktivität **RetrieveData**, welche als Eingabeparameter einen DM command (z.B. eine SELECT-Anweisung, ein Pfadname) erhält, der auf der spezifizierten Datenressource ausgeführt wird. Der DM command muss als Resultat Daten zurückliefern. Nach der Ausführung werden die Ergebnisdaten an die Workflow Execution Engine zurückgeschickt und in einer durch einen weiteren Eingabeparameter spezifizierten data set variable gespeichert.
- Für eine **logische Referenzierung** des Datenbankssystem können in BPEL/SQL qualifizierende Bezeichner verwendet werden. Die logische Referenz wird dann zum Deploymentzeitpunkt eines Workowmodells an ein konkretes physisches Datenbanksystem gebunden. In BPEL-DM werden Datenressourcen durch *data source reference variables* referenziert. Dies sind logische Referenzen, bei denen es sich entweder um logische Namen oder um Dokumente handelt, die funktionale oder nichtfunktionale Anforderungen der Datenressource beschreiben. Logische Namen werden vom SIMPL-Core zur Laufzeit des Workflows aufgelöst.
- Für die Materialisierung einer relationalen Tabelle werden in BPEL/SQL **Set-Variablen** zur Verfügung gestellt. Bei diesen Variablen handelt es sich um XML-RowSet-Datenstrukturen, in denen die relationalen Daten nach der Ausführung einer <retrieveSet> Aktivität lokal im Workflow verfügbar gemacht werden. In BPEL-DM dienen **data set variables** als Zieldatencontainer, um Daten in den Workflow-Kontext zu laden. Der Inhalt dieser Variablen wird über geeignete XML Schemadefinitionen spezifiziert. Für tabellen-basierte Daten, wie z.B. von einer SQL-Datenbank oder von CSV-Dateien werden hier ebenso XML RowSet Datenstrukturen verwendet.

- Relationale Tabellen können in einem BPEL/SQL-Workflow an sogenannte **Set-Referenz**-Variablen gebunden werden. Diese sind Zeiger auf eine relationale Tabelle. Eine Ergebnis-Referenzmenge kann für eine nachfolgende <sql> Aktivität als Eingabe dienen. Auf diese Weise können relationale Daten über Aktivitäts- und Workflowgrenzen hinweg referenziert (by-reference) anstatt materialisiert (by-value) übergeben werden. Dadurch können hohe Leistungsgewinne erzielt werden. Datenkonstrukte, wie z.B Tabellen in einem Datenbanksystem, Collections und XML-Dokumente in XML-Datenbanken oder Dateien in Dateisystemen können in BPEL-DM an Datencontainer gebunden werden. Dabei ist jeder Datencontainer eine identifzierbare Sammlung von Daten. Datenressourcen können mehrere solcher Datencontainer verwalten. Die logische Referenzierung solcher Datencontainer ist über logische Namen oder den lokalen Bezeichner möglich, welche in **data container reference variables** gespeichert werden. Der SIMPL Core löst diese Rfrenzen mit Hilfe der Simulation Artifact Registry dann auf. *Data container reference variables* können als Parameter in DM commands von IssueCommand oder RetrieveData verwendet werden, z.B. in der From-Klausel einer SQL SELECT-Anweisung.

Insgesamt stellt BPEL-DM eine mächtigere Sprache dar als BPEL/SQL. Bis auf einige wenige Ausnahmen, die für die in dieser Arbeit betrachteten Optimierungen weniger relevant sind, können alle Ausdrücke, die in BPEL/SQL möglich sind, auch in BPEL/DM formuliert werden. Darüber hinaus können mit BPEL/DM jedoch neben SQL-Datenbanken auch auf anderen Datenressourcen wie XML-Datenbanken oder Dateisystemen Datenverarbeitungsoperationen ausgeführt werden. Mit Hilfe einer IssueCommand-Aktivität können im Grunde alle Befehle ausgeführt werden, die auch von der zu Grunde liegenden Datenressource unterstützt werden. Des Weiteren können mit TransferData-Aktivitäten Dateien zwischen zwei Datenressourcen kopiert werden. Die WriteDataBack-Aktivität ermöglicht das Schreiben von Daten von der Workflow- auf die Datenebene. Ein weiterer Vorteil von BPEL-DM ist, dass Datenquellen durch logische Namen zur Laufzeit bestimmt werden können, während dies bei BPEL/SQL bereits zum Deployment-Zeitpunkt passieren muss. Was BPEL-DM (bisher) nicht zur Verfügung stellt, sind Aktivitäten, welche die flexible Definition von Transaktionsgrenzen zulassen. Solch eine Aktivität wird in BPEL/SQL bereitgestellt. Sie enthält eine Liste von <sql> und <retrieveSet> Aktivitäten, die in einer Transaktion ausgeführt werden sollen.

Durch die Optimierungsregeln werden Kontrollflussmuster in effizientere Strukturen umgewandelt. Die zu optimierenden Strukturen basieren dabei grösstenteils auf SQL-basierten Anweisungen. Beispielsweise wird eine tupelbasierte SQL-Anweisung in eine einzelne, mengenbasierte Anweisung umgewandelt und somit das Entfernen eines For-Schleifenkonstrukts aus dem Workflow erreicht. *Ähnliche Kontrollflussmuster* und Konstrukte sind jedoch, unabhängig von den zu Grunde liegenden Datenquellen, auch in anderen kontrollflussorientierten Simulationsworkflows zu finden. Somit dürfte sich die Erarbeitung ähnlicher Restrukturierungen vergleichsweise einfach gestalten. Da BPEL-DM verschiedene, heterogene Datenquellen abdeckt, muss geklärt werden, inwiefern der auf SQL basierende Optimierungsansatz auf *nicht SQL-basierte Sprachen*, wie z.B. XQuery bei XML-Datenbanken oder Shell-Kommandos bei Dateisystemen, ausgeweitet werden kann.

Die starke *Fundierung des Ansatzes auf SQL* kann eventuell problematisch sein. Während die Übertragbarkeit auf andere relationale Sprachen und die Sprache XQuery, welche als Korrespondenz zu SQL im Bereich der XML-Datenbanken angesehen wird, einfach realisierbar scheint, könnte sich die Entwicklung von Optimierungen bei Shell-Sprachen problematischer gestalten. Dagegen kann die *Regelbasiertheit* des Optimierungsansatzes als Vorteil betrachtet werden, da somit eine strikte

Definiton der Anwendungsfälle, sowie eine Strukturierung der Optimierungsmethoden vorgegeben ist, die in einen daraus zu entwickelnden Optimierungsansatz übernommen werden können. Ein weiterer Vorteil ist insbesondere, dass entsprechende Optimierungsregeln unter Umständen in die ebenso regelbasierte Transformation von Patterns eingebunden werden können.

Zusammenfassend lässt sich sagen, dass die in Abschnitt 5.1 vorgestellten Optimierungsvorschläge wegen ihres Bezugs zu der Workflowsprache BPEL/SQL eine geeignete Grundlage zur Erarbeitung von ähnlichen Optimierungsvorschlägen für kontrollflussbasierte Simulationsworkflows bilden, insbesondere für Simulationsworkflows, die mit BPEL/SQL-ähnlichen Workflow-Sprachen wie BPEL/DM beschrieben wurden, sodass ähnlich grosse Optimierungsgewinne zu erwarten sind. Aufgrund dieser Überlegungen wird der Optimierungsansatz aus [Vrh11] in Kapitel 7 bei der Entwicklung von Optimierungen für Simulationsworkflows, insbesondere für die in dieser Arbeit betrachteten Beispiel-Workflows, als Grundlage dienen. Da es sich bei Simulationsworkflows um datenintensive Workflows handelt, denen neben SQL-Datenbanksystemen verschiedenste Datenressourcen zu Grunde liegen können, z.B. Dateisysteme oder XML-Datenbanken und dies die Einbindung unterschiedlichster Datenverwaltungsoperationen und datenressourcen-spezifischer Befehle erfordert, muss der PGM-F-Optimierungsansatz an diese neuen Anwendungsszenarien angepasst werden, indem er auf nicht SQL-basierte Sprachen, wie z.B. XQuery bei XML-Datenbanken oder Shell-Kommandos bei Dateisystemen ausgeweitet wird.

6.2 Bewertung des VDAL-Optimierungsansatzes

Der Ansatz aus Abschnitt 5.2 stellt mit *Virtual Data Assembly Lines* ein Paradigma vor, mit dessen Hilfe wissenschaftliche Workflows erstellt werden können, indem bereits existierende Ideen aus dem Bereich der Datenfluss-Netzwerke und Prinzipien aus der Verarbeitung von XML-Daten verwendet werden. Hinsichtlich der Übertragbarkeit der Optimierungsvorschläge dieses Ansatzes auf die in dieser Arbeit betrachteten Simulationworkflows ist ein wichtiger Faktor, dass in [Zin10] *datenintensive* Workflows aus dem wissenschaftlichen Bereich betrachtet werden. Die Zugehörigkeit der Simulationsworkflows zu den wissenschaftlichen Workflows [RSM11] und die Tatsache, dass diese ebenso mit grossen Datenmengen umgehen müssen, legt den Einsatz der Optimierungvorschläge für Simulationsworkflows nahe.

VDAL betrachtet Workflows jedoch mit dem Fokus auf ihren Datenfluss. D.h. Workflows, die mit *datenflussorientierten* Sprachen beschrieben wurden, bieten sich für die Anwendung dieses Ansatzes an. Im Gegensatz dazu wurden die in dieser Arbeit betrachteten Workflows mit kontrollflussorientierten Sprachen beschrieben. Aufgrund dieser *Diskrepanz* der verwendeten Beschreibungssprachen, ist ein direkter Zusammenhang zwischen den betrachteten Workflows und den Optimierungsvorschlägen nicht direkt ersichtlich bzw. eine direkte Übertragung der Optimierungsvorschläge nicht ohne Weiteres realisierbar. Die Herausforderungen, denen durch die Einführung des VDAL-Ansatzes begegnet werden soll, sind typische Problemstellungen, die sich bei Verwendung datenflussorientierter Modelle und Beschreibungssprachen beim Workflow-Design ergeben. Die meisten Ideen des VDAL-Paradigmas, welche den Entwurf von Workflows vereinfachen sollen, werden bereits durch kontrolflussorientierte Sprachen abgedeckt. Parameterreiche Funktionen und benutzerdefinierte

Datenstrukturen äussern sich hier z.B. nicht in einer Vielzahl zusätzlicher Aktivitäten und Verknüpfungen, da Datenflussbhängigkeiten durch Parameter und Variablen implizit modelliert werden.

Während VDAL als primäres Ziel hat, das Design von Workflows zu optimieren, werden in [Zin10] zwei auf VDAL basierende Optimierungsvorschläge vorgestellt, die die Ausführungszeiten der Workflows verringern bzw. den Datenverkehr zwischen den einzelnen Aktivitäten auf ein Minimum reduzieren sollen. Eine direkte Übertragung dieser Optimierungsvorschläge auf die relevanten Beispielworkflows dieser Arbeit, würde die Überführung selbiger in datenflussorientierte Modelle erfordern und wäre aufwendig. Ebenso aufwendig wäre die Überarbeitung der Optimierungsansätze, sodass sie auch für Kontrollflüsse geeignet wären. Die den Optimierungsvorschlägen zu Grunde liegenden Ideen können jedoch weiterverwendet werden. Der Vorschlag zur Nutzung von *MapReduce*-Verfahren zur Ausnutzung von Datenparallelität bietet sich z.B. bei einer abstrakten Betrachtung für weitere Untersuchungen an. Der Parallelisierungs-Ansatz soll in dieser Arbeit jedoch nicht weiter untersucht werden. Zum Einen wird in Vr ovhnik's Arbeit ebenso eine Optimierungsregel zur Parallelisierung vorgeschlagen. Zum Anderen wurde der Einsatz des MapReduce-Rahmenwerks für den Kopplungsworkflow bereits in [Ges] bearbeitet. Dem Ansatz zur *Minimierung des Datenverkehrs* liegt als Hauptgedanke eine *Filterung* der zwischen den Workflow-Aktivitäten übertragenen Daten zu Grunde. Eine solche Art von Datenfilterung kommt beim Ansatz von Vrovhnik in ähnlicher Form bei der Optimierungsregel des *Predicate-Pushdown* zum Einsatz. Dementsprechend wird dieser Optimierungsvorschlag auch nicht weiterverfolgt.

Zusammenfassend kann gesagt werden, dass die in Abschnitt 5.2 vorgestellten Optimierungsvorschläge eine zu enge Bindung zu datenflussorientierten Repräsentationsformen von Workflows aufweisen. Von dem Standpunkt aus betrachtet, dass kontrollflussorientierte Sprachen wie WS-BPEL verstärkt zur Modellierung von Workflows verwendet werden [AMA06], [Slo07], [GHCM09], und insbesondere die für diese Arbeit relevanten Beispielworkflows mit WS-BPEL-ähnlichen Sprachen realisiert wurden, sind die Optimierungsvorschläge für diese Arbeit weniger geeignet. Generell ist jedoch eine Betrachtung dieser Optimierungsvorschläge in zukünftigen Arbeiten nicht uninteressant.

6.3 Bewertung des dBOP-Optimierungsansatzes

Dieser Ansatz weist Ähnlichkeiten zum PGM-F-Ansatz auf, der in Abschnitt 5.1 betrachtet wurde. Ein Workflow wird hier ebenso als graphenbasiertes Modell beschrieben, auf dem für eine Optimierung interessante Teilbereiche ausfindig gemacht werden. Zudem werden Optimierungen hier auch in Form von Regeln, bestehend aus einem Bedingungs- und einem Aktionsteil, in einer Regelbasis bereitgestellt. Im Vergleich zur PGM-F-Optimierung werden hier zur Unterstützung der Entscheidung über die Regelanwendung Data-Mining-Technologien angewandt. Hierfür sind den bestimmten Optimierungspatterns innerhalb des Optimierers bestimmte Data-Mining-Algorithmen zugeordnet. Des Weiteren sind die Regeln innerhalb der Regelbasis ebenfalls kategorisiert, im Gegensatz zum PGM-F-Rahmenwerk jedoch nicht nur nach den Veränderungen, die sie verursachen, sondern mitunter auch nach den Optimierungszielen, die sie unterstützen. Ferner findet ebenso wie im PGM-F-Rahmenwerk eine automatisierte Optimierung eines Workflows statt, wobei der Optimierer die zu optimierenden Teilbereiche und die Reihenfolge der Regelanwendung bestimmt. Am Anfang des Prozesses kann der Analyst hier jedoch Optimierungsziele vorgeben, nach denen sich die Optimierung richten soll, wobei

kostenbasierte Ziele verwendet werden können. Zudem hat der Analyst die Möglichkeit einzelnen Transformationen zuzustimmen oder abzulehnen, bevor die Transformationen endgültig ausgeführt werden.

Dieser Ansatz stellt eine auf Geschäftsprozesse fokussierende Komplettlösung für eine Prozessoptimierung dar, welche eine Interaktion mit dem Benutzer einschliesst, sowie Optimierungsentscheidungen basierend auf komplexeren Data-Mining-Analyseergebnissen fällt, in welche zudem operationale Daten einfliessen. Für die Untersuchungen in dieser Arbeit ist dieser Ansatz zu umfangreich und würde den Rahmen dieser Arbeit sprengen. Zudem sind viele der Ideen im PGM-F-Ansatz zu finden. Deshalb wird dieser Ansatz nicht weiterverfolgt.

7 Regelbasierte Optimierung von Simulationsworkflows

In diesem Kapitel werden Optimierungen für Simulationsworkflows vorgestellt, die den Optimierungsansatz aus Abschnitt 5.1 als Grundlage nehmen. Da der Grossteil der Simulationsworkflows im Rahmen des an der Universität Stuttgart entwickelten Prototyps des SIMPL-Rahmenwerks in der Workflow-Sprache BPEL-DM realisiert worden sind, werden sich die hier vorgestellten Optimierungsideen dementsprechend an dieser Workflow-Sprache orientieren. Hierfür wird zunächst in Abschnitt 7.1 die Regelbasis des PGM/F-Rahmenwerks sowie die jeweiligen Optimierungsregeln kurz beschrieben. Dabei wird nicht auf Details der jeweiligen Optimierungen eingegangen. Vielmehr werden die Grundideen beschrieben, welche anschliessend als Grundlage für den Abschnitt 7.2 dienen, in dem generische Optimierungsregeln für BPEL-DM-Workflows hergeleitet werden.

7.1 Die Regelbasis und Optimierungsregeln des PGM/F-Rahmenwerks

Die Regelbasis des PGM/F enthält alle Regeln, die zur Optimierung der Datenverarbeitunsoperationen in BPEL/SQL-Workflows verwendet werden und die Restrukturierung von suboptimalen Workflowfragmenten definieren. In Abbildung 7.1 ist die Hierarchie der verschiedenen Regelklassen dieser Regelbasis abgebildet.

Im Folgenden wird eine Auswahl der verschiedenen Optimierungsregeln des PGM/F-Rahmenwerks vorgestellt und kurz erläutert. Dabei werden die Aspekte bezüglich des PGM/F-Modells so gut wie möglich aussen vor gelassen, da ein solches Modell nicht Teil der in dieser Arbeit vorgestellten Optimierungsideen sein soll und den Rahmen dieser Arbeit sprengen würde. Vielmehr sollen die den Restrukturierungsregeln zu Grunde liegenden Ideen verdeutlicht werden. Für tiefergehende Details bezüglich der Optimierungsregeln wird auf die Lektüre [Vrh11] verwiesen.

7.1.1 Die Regelklasse Tuple-To-Set

Diese Regelklasse bezieht sich auf Schleifen, die auf einer gegebenen Menge iterieren und für jedes Tupel der Menge eine SQL-Aktivität ausführen. Diese Regeln ersetzen solch eine Schleife und die SQL-Aktivität im Schleifenrumpf durch eine einzelne SQL-Aktivität, welche die Semantik der gesamten Schleife abdeckt. Durch die Tansformation einer tupel-orientierten SQL-Anweisung in eine mengenorientiere SQL-Anweisung, wird die iterative Ausführung der SQL-Aktivität uberflüssig

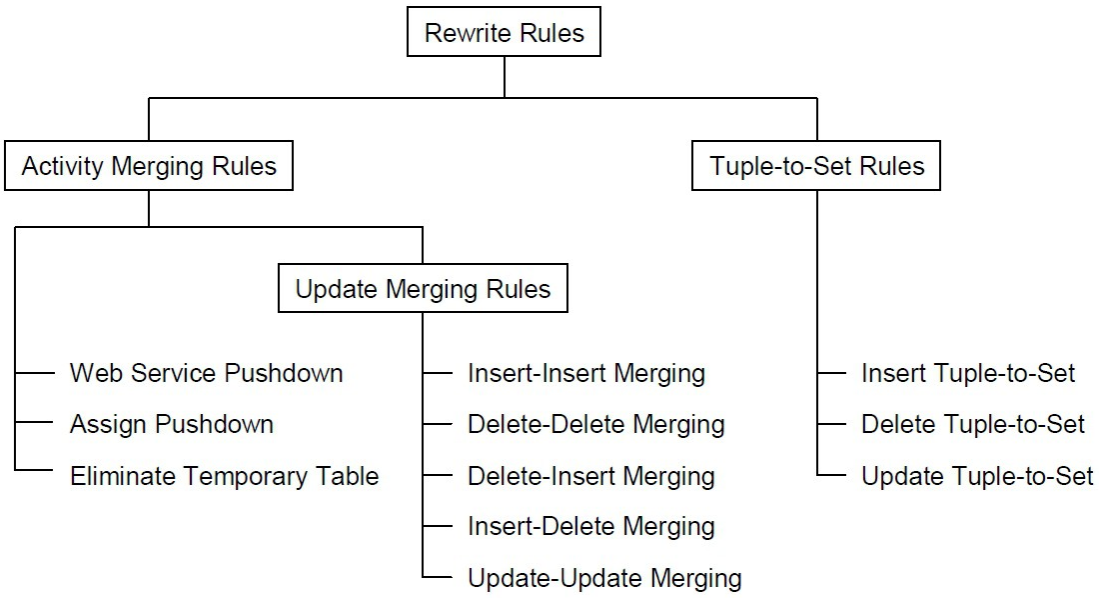


Abbildung 7.1: Klassifikation der Regeln der Regelbasis von PGM/F [Vrh11]

und die For-Each-Aktivität kann aus dem Workflow entfernt werden. Diese Optimierung führte im PGMF-F-Rahmenwerk zu erheblichen Laufzeitgewinnen [Vrh11].

Je nach Art einer DML-Anweisung wird zwischen einer *Insert-*, *Update-* und *Delete-Tuple-To-Set-Regel* unterschieden. Auf die Details der unterschiedlichen Varianten wird hier nicht weiter eingegangen. In Abbildung 7.2 ist ein Beispiel für die Anwendung der *Insert-Tuple-To-Set-Regel* gezeigt. Die SQL-Aktivität a_i schreibt die Resultate eine Anfrage $expr_i$ in eine *set variable* v_{set}. Die ForEach-Aktivität a_j iteriert über v_{set} und stellt in jedem Iterationslauf die aktuell, referenzierte Zeile in v_{row} bereit. Die SQL-Aktivität a_k führt anschliessend eine INSERT-Anweisung aus, wobei die Tupel aus v_{row} als Eingabeparameter dienen. Nach Anwendung der *Tuple-To-Set-Regel* wird das beschriebene Datenverarbeitungsmuster durch eine SQL-Aktivität a_k^* ersetzt, welche alle relevanten Tupelwerte gleichzeitig durch ein mengenbasiertes INSERT verarbeitet.

7.1.2 Die Regelklasse Activity-Merging

Ziel der Regelklasse *Activity-Merging* ist es, eine Datenverarbeitungsoperation, die in einer Assign- oder SQL-Aktivität definiert ist, mit einer folgenden datenabhängigen SQL-Anweisung zu verschmelzen. Dabei wird die datenabhängige SQL-Anweisung derart transformiert, dass die zuvor getrennt ausgeführten Datenverarbeitungsoperationen zu einer einzelnen Datenverarbeitungsoperation kombiniert werden. Das Verschmelzen von Aktivitäten kann dabei nur durchgeführt werden, falls dadurch keine Datenflussabhängigkeiten zu anderen Aktivitäten beeinflusst werden. Entsprechend der unterschiedlichen Aktivitäts- bzw. SQL-Anweisungstypen, die miteinander kombiniert werden können,

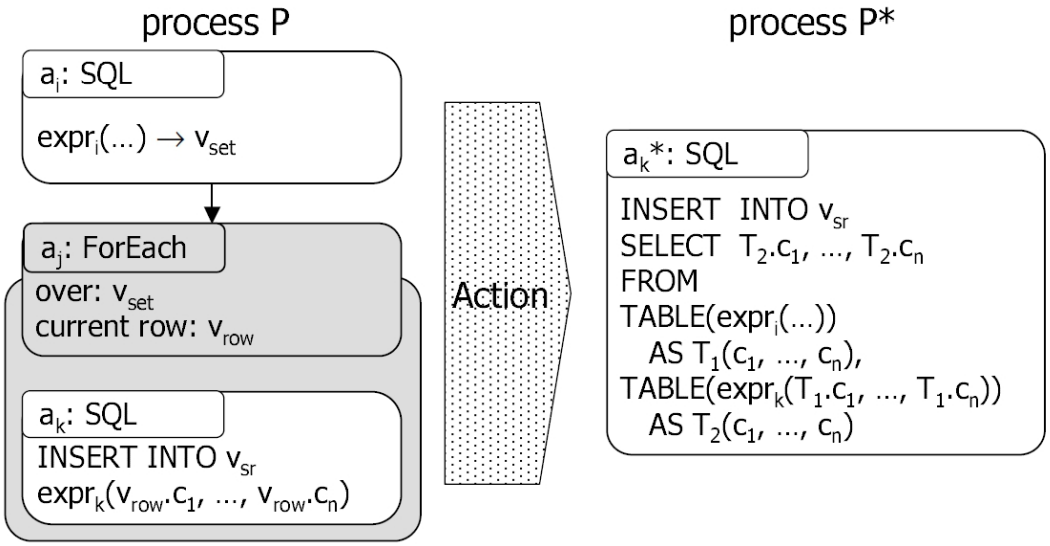


Abbildung 7.2: Beispiel für die Tuple-To-Set-Regel [VSS+07]

setzt sich die Regelklasse *Activity-Merging* aus den Regeln *Assign- Merging* und *Eliminate-Temporary-Table* und zum anderen aus den Regelunterklassen *Query-Merging* und *Update-Merging* zusammen. In den folgenden Abschnitten werden beispielhaft einige der Regeln beschrieben. Details bezüglich aller Regeln sind in [Vrh11] nachzulesen.

Assign Merging

Die *Assign-Merging-Regel* eliminiert eine Variablenzuweisung, die in einer Assign-Aktivität definiert ist, indem die Zuweisung direkt in einer abhängigen SQL-Aktivität durchgeführt wird. In Abbildung 7.3 ist ein Beispiel für die Anwendung der *Assign-Merging-Regel* gezeigt. Eine Assign-Aktivität a_i schreibt den Wert einer Variable v_i in die Variable v_j. Die anschliessende SQL-Aktivität SQ_j ermittelt aus einer Tabelle ApprovedOrders jene Bestelldaten, die preislich über diesem Wert liegen. Aufgrund der exklusiven Schreib-Lese-Datenabhängigkeit zwischen a_i und a_j basierend auf v_j kann die Assign-Merging-Regel angewandt werden, welche das Vorkommen der Variablen v_j in SQ_j direkt durch Variable v_i. Die nun überflüssige Assign-Aktivität a_i kann entfernt werden.

Die Regelklasse Update-Merging

Gleichartige Typen von SQL-Aktivitäten, die sequentiell DML-Anweisungen auf derselben Tabelle ausführen, lassen sich zu einer einzelnen Aktivität verschmelzen. Dies kann beispielsweise durch Verwendung geeigneter Mengenoperationen oder Anpassung von Klauseln realisiert werden. Bei der Verschmelzung der Aktualisierungsoperationen muss jedoch sichergestellt sein, dass durch die gleichzeitige Ausführung der Aktualisierungen dasselbe Resultat erzielt wird, wie durch ihre ursprüngliche, sequentielle Ausführung.

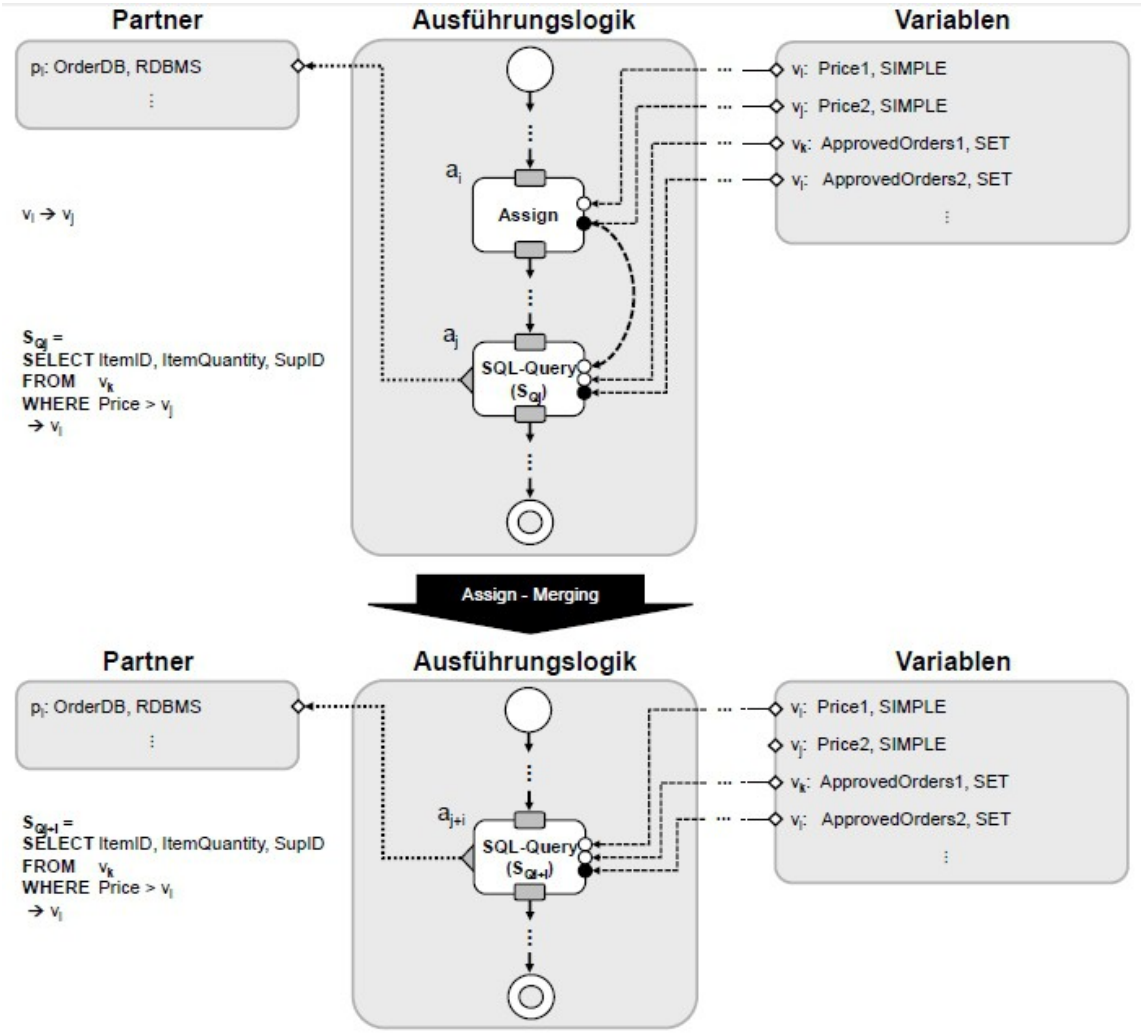


Abbildung 7.3: Beispiel für die Assign-Merging-Regel

Bei der **Insert-Insert-Merging-Regel** z.B. werden zwei Insert-Anweisungen, mit einer UNION-ALL-Operation zu einer einzelnen Aktualisierungsoperation kombiniert. Die **Delete-Delete-Merging-Regel** dagegen, kombiniert zwei Delete-Anweisungen zu einer einzelnen Delete-Anweisung, indem die Selektionsprädikate in der Where-Klausel beider Anweisungen mittels einer OR-Verknüpfung verschmolzen werden. Beim **Update-Update-Merging** schliesslich werden zwei Update-Anweisungen durch Kombination der SET-Klauseln beider DML-Anweisungen zu einer Update-Anweisung verschmolzen.

Die WebService-Pushdown-Regel

Eine **Web-Service-Pushdown**-Regel verlagert einen Web-Service-Aufruf von der Workflow- auf die Datenebene. Dabei wird der Web-Service-Aufruf in eine SQL-Anweisung eingebettet und bei der Verarbeitung der SQL-Anweisung vom Datenbanksystem ausgeführt. Damit verbunden ist eine Umwandlung einer BPEL-Invoke- in eine SQLQuery-Aktivität. Voraussetzung für die Anwendbarkeit dieser Regel ist die Unterstützung der Durchführung von Web-Service-Aufrufen durch das Datenbanksystem. Es existieren keine Standards seitens der Datenbanksysteme für den Aufruf von Web Services. Üblicherweise werden sogenannte User-Defined-(Table-)Functions (im Folgenden WS-UDTFs genannt) angeboten. Dabei handelt es sich um benutzerdefinierte Funktionen, die über SOAP/HTTP Web-Services aufrufen können. Hierzu müssen alle Eingabeparameter des Web-Service-Aufrufs von der Workflow- auf die Datenebene und das Ergebnis des Web-Service-Aufrufs an das Datenbanksystem übertragen werden.

In [Vrh11] wurde zudem eine der *Web-Service-Pushdown-Regel* vom Prinzip her ähnliche Regel für SQL Stored Procedures vorgestellt. Bei dieser **Stored Procedure Pushdown-Regel** werden statt Web-Service-Aufrufen, beliebige Workflow-Konstrukte auf die Datenebene, genauer auf eine Stored Procedure verlagert. Dies bietet sich insbesondere dann an, wenn Activity-Merging bzw. Tuple-To-Set-Regeln nicht erfolgreich angewendet werden konnten.

Eliminate-Temporary-Table

Die **Eliminate-Temporary-Table-Regel** entfernt in SQL-Aktivitäten die Verwendung von temporären Tabellen innerhalb von SQL-Anweisungen. Dabei bezeichnet eine temporäre Tabelle eine Anfrageergebnismenge, welche in BPEL/SQL durch eine *result set reference* referenziert wird und für jede Prozessinstanz zu Beginn der Prozessausführung erstellt und nach ihrer Ausführung entfernt wird. Durch Anwendung der Regel wird eine solche Referenz auf einen *result set* in einer SQL-Anweisung durch die Anfrage ersetzt, die diesen *result set* als Ergebnis zurückliefert. Dadurch reduzieren sich die Kosten, die für die Lebenszyklusverwaltung der Tabellen auf Datenebene und für die SQL-Verarbeitung auf Workflow-Ebene anfallen.

Ein Beispiel für die Anwendung der *Eliminate-Temporary-Table-Regel* ist in 5.2 im letzen Optimierungsschritt von b) zu c) zu sehen. Die Aktivität GroupOrdersByItemID führt hier eine SQL-Anfrage auf der Datenbanktabelle Orders aus und speichert (eine Referenz) auf das Anfrageergebnis in der BPEL-Varablen *SRItemList*. Die anschliessende Aktivität ForEachItemOrder führt anschliessend unter lesender Referenzierung von *SRItemList* eine SQL-Anfrage auf der Tabelle OrderConfirmations aus. Durch die Anwendung der Regel wird diese Referenz in der SQL-Anfrage von ForEachItemOrder durch die SQL-Anfrage von GroupOrdersByItemID ersetzt. Die Aktivität GroupOrdersByItemID und *SRItemList* werden entfernt. Aus ForEachItemOrder geht eine eine neue Aktivität OrderProcessing hervor.

7.1.3 Query-Merging

Das zwischen einem Workflow- und einem Datenbanksystem zu übertragende Datenvolumen kann für die Laufzeit eines datenintensiven Workflows entscheidend sein. Durch eine Reduzierung der Grösse von Anfrageergebnismengen, die von der Daten- auf die Workflowebene übertragen werden müssen, lassen sich Übertragungs- und Materialisierungskosten senken. Deshalb sollte auf unnötige Übertragungen von Anfrageergebnismengen verzichtet werden. Dies kann durch die Regeln *Predicate-Pushdown*, *Select-Into-Merging* und *Select-Merging* erreicht werden. Die Regel *Predicate Pushdown* wird nachfolgend vorgestellt. *Select-Into-Merging* und *Select-Merging* werden nicht weiter betrachtet, da diese Optimierungen sehr eng gebunden sind an eine SQL-Syntax und lediglich im Zusammenspiel mit einer Select-Into-Anfrage Sinn machen, welche zuvor aus der Anwendung einer Web-Service-Pushdown-Regel hervorgegangen ist.

Die Predicate-Pushdown-Regel

Bei der Ermittlung von Anfrageergebnissen kann durch frühzeitige Ausführung von Selektionsoperationen die Grösse von Zwischenergebnissen verkleinert und somit hohe Laufzeitgewinne erzielt werden. Entsprechende Regeln findet man z.B. in der klassischen SQL-Anfrageoptimierung.

In Abbildung 7.4 ist ein Beispiel für die Anwendung der *Predicate-Pushdown-Regel* auf einem Workflow gezeigt. Es existiert eine SQL-Query-Aktivität a_i, welche durch eine Anfrage SQ_i auf der der Tabelle ApprovedOrders (v_i) eine Bestellmenge bestimmt, deren Bestellwert eine bestimmte Summe übersteigt und das Anfrageergebnis in einer SET-Variablen ApprovedOrders2 v_k materialisiert. Die anschliessende Assign-Select-Aktivität a_j wendet auf v_k ein Selektionsprädikat an, sodass nur Bestellungen bestimmter Lieferanten herausgefiltert werden. Aufgrund der exklusiven Schreib-Lese-Datenabhängigkeit zwischen a_i und a_j basierend auf v_k wird die Predicate-Pushdown-Regel angewendet und das Selektionsprädikat von a_j nach a_i verschoben. Hierfür wird eine neue SQLAnfrage SQ_{i+j} abgeleitet, welche die relevanten Tupel durch die Verknüpfung der Selektionsprädikate von SQ_i und SQ_j mit einem logischen AND bestimmt. Durch die Transformation werden a_j und v_k entfernt und die Selektionsoperation von der Workflow- auf die Datenebene verschoben, was zu einer Verkleinerung der Anfrageergebnismenge, sowie einer Reduktion der Übertragungs- und Materialisierungskosten führt.

Variante der Predicate-Pushdown-Regel

Eine **Variante der Predicate-Pushown-Regel** kann auf folgendes Datenverabeitungsmuster, wie in Abbildung 7.5 gezeigt, angewandt werden: Es existiert eine SQL-Query-Aktivität a_i, die eine Anfrage SQ_i auf einer Tabelle (v_i) ausführt und das Ergebnis in einer SET-Variablen v_j speichert, welche wiederum einem folgenden LOOP-Kontrollflussmuster als Iterationsmenge dient. Eine Set-Iterator-Aktivität a_k stellt in jedem Iterationsschritt das nächste Tupel aus v_j bereit. Anschliessend entscheidet eine XOR-Split-Select-Aktivität a_l anhand eines Attributwertes dieses Tupels, ob es sich für eine Weiterverarbeitung qualifiziert oder nicht.

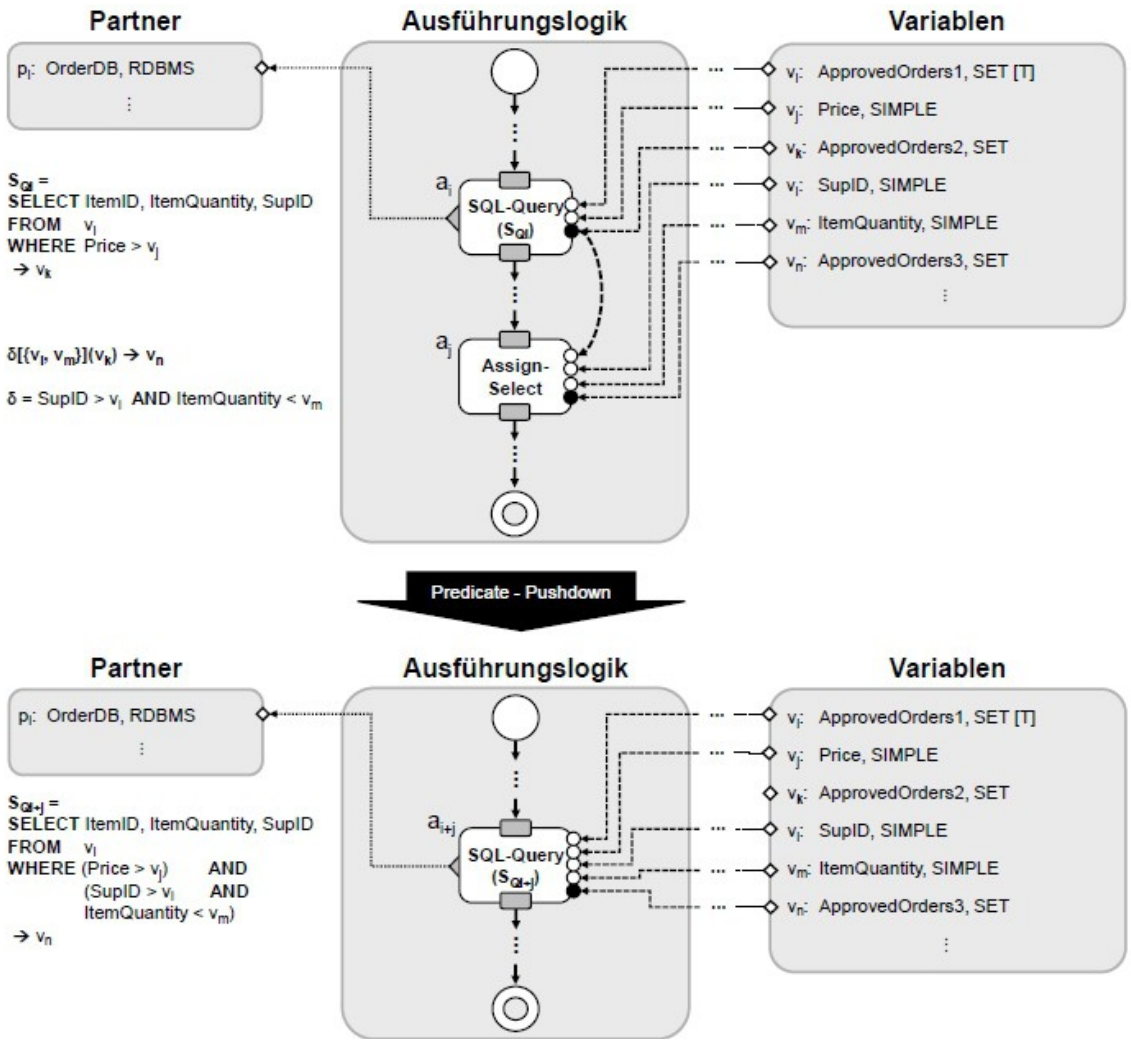


Abbildung 7.4: Beispiel für die Predicate-Pushdown-Regel [Vrh11]

Durch die XORSplit-Select-Aktivität erfolgt eine späte, ineffiziente Filterung der materialisierten Datenmenge, welche Daten enthält, die auf der Workflowebene nicht benötigt werden und somit unnötige Materialisierungs- und Übertragungskosten verursachen. Die Ausführungskosten für das LOOP-Kontrollflussmuster sind proportional zur Grösse dieser Iterationsmenge.

Für eine Verbesserung der Laufzeit müssen durch eine frühzeitig Filterung überflüssige Daten herausgefiltert und die Iterationsmenge verkleinert werden. Dies kann durch Kombination des Selektionsprädikats der XORSplit-Select-Aktivität mit dem Selektionsprädikat der Where-Klausel der SQL-Anfrage SQ_i erreicht werden. Zudem entfällt die Ausführung der XOR-Split-Select-Aktivität und der Schleifenrumpf wird vereinfacht, so dass gegebenenfalls eine *Tuple-To-Set-Regel* ermöglicht wird.

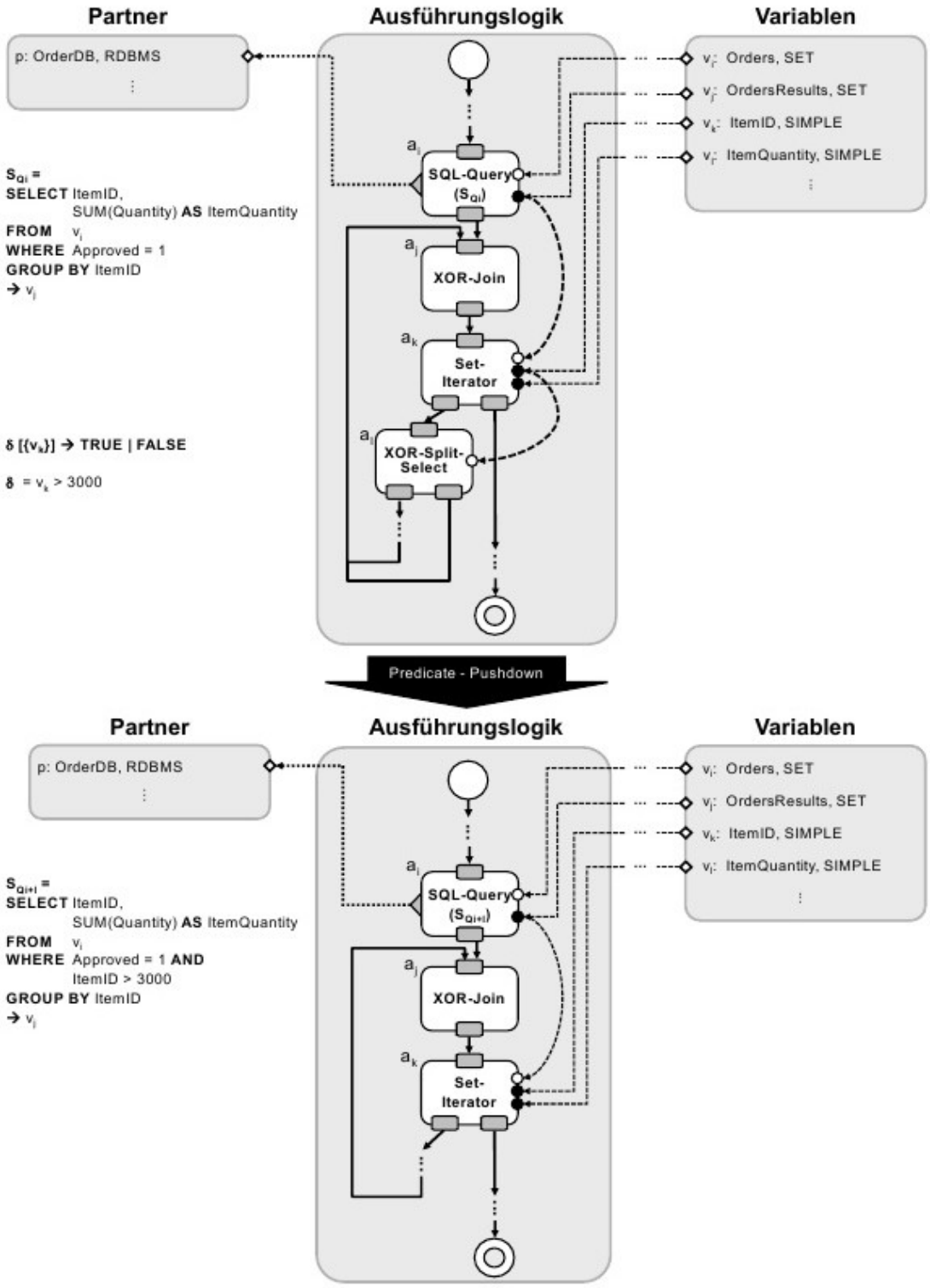


Abbildung 7.5: Beispiel für die Variante der Predicate-Pushdown-Regel [Vrh11]

7.2 Optimierungsregeln für Simulationsworkflows

In diesem Abschnitt werden aus den im letzten Abschnitt beschriebenen Optimierungsregeln des PGM/F-Rahmenwerks abgeleitete, abstrakte Optimierungsregeln für Simulationsworkflows beschrieben. Bei der Formulierung der Regeln wird berücksichtigt, dass innerhalb eines Simulationsworkflows unterschiedliche Datenressourcen und Datenformate und somit unterschiedliche Befehlssprachen eingesetzt werden können. Die vorgeschlagenen Regeln können als Richtlinien verstanden werden, sodass Abweichungen fallspezifisch möglich sind. Im Vordergrund steht die Umsetzung der Grundidee der Optimierungsregeln.

7.2.1 Die Element-To-Set-Regel

Bei der **Element-To-Set-Regel** handelt es sich um die Optimierungsregel, die der *Tuple-To-Set-Regel* des PGM/F-Rahmenwerks entspricht und sie generalisiert. Wie die *Tuple-To-Set-Regel*, hat auch die *Element-To-Set-Regel* das Ziel, das ineffiziente Datenverarbeitungsmuster eines sequentiellen Durchlaufens einer Iterationsmenge und der mehrfachen Anwendung einer Datenverarbeitungsoperation auf den einzelnen *Elementen* dieser Menge aufzulösen. Bei den Elementen der Menge kann es sich hier neben Tupeln einer relationalen Tabelle je nach zu Grunde liegender Datenressource, z.B. auch um Dateien in einem Dateisystem oder Knoten einer XML-Struktur handeln.

Die *Element-To-Set-Regel* überführt also eine datensatz- bzw. dateibasierte Ausführung einer Anweisung in eine semantisch äquivalente, **mengenbasierte** Ausführung. Voraussetzung für die Transformation ist, dass die Befehlssprache, in der die neue Anweisung formuliert wird, die mengenbasierte Erfassung der relevanten Daten ermöglicht. Dies kann z.B. bei Dateisystemen häufig durch Verzeichnisnamen und bei XML-Datenbanken durch XQuery-FLWOR-Ausdrücke erreicht werden. Durch die Transformation wird das ineffiziente Datenverarbeitungsmuster des Schleifenkonstrukts sowie alle damit verbundenen Konstrukte zur Vorbereitung der Schleifendurchläufe gelöscht bzw. direkt durch die mengenbasierte Ausführung ersetzt. Diese Transformation einer Schleife auf der Workflow-Ebene in eine mengenbasierte Anweisung auf der Datenebene kann eventuell grosse Laufzeitgewinne bewirken. Das Grundmuster der *Element-To-Set-Regel* ist in Abbildung 7.6 dargestellt. Der Workflow kann optional am Anfang eine Aktivität bzw. mehrere Aktivitäten enthalten, welche die Ausführung des nachfolgenden Schleifenkonstruktes vorbereiten. Dabei handelt es sich z.B. um eine RetrieveData-Aktivität oder eine Assign-Aktivität, welche die Iterationsmenge lädt bzw. als Initialwert einer Variablen zuweist. Diese Art von Aktivität(en) wird im restlichen Teil dieser Arbeit als **datenbereitstellende Aktivität** bezeichnet. Diese datenbereitstellende Aktivität ist sowohl vor der Anwendung der *Element-To-Set-Regel*, als auch danach optional. Dies wird durch die gepunktete Umrandung der Aktivität in der Abbildung 7.6 deutlich gemacht. Ein Beispiel hierfür ist in Abschnitt 8.9 gegeben, wo nach der Regelanwendung mit Hilfe einer Assign-Aktivität eine *data container reference list* in einer Variablen des Typs String bereitgestellt werden muss, bevor sie später auf Kommandozeilenebene verarbeitet werden kann.

Um eine *Element-To-Set-Regel* korrekt anwenden zu können, müssen folgende **Regelbedingungen** erfüllt sein:

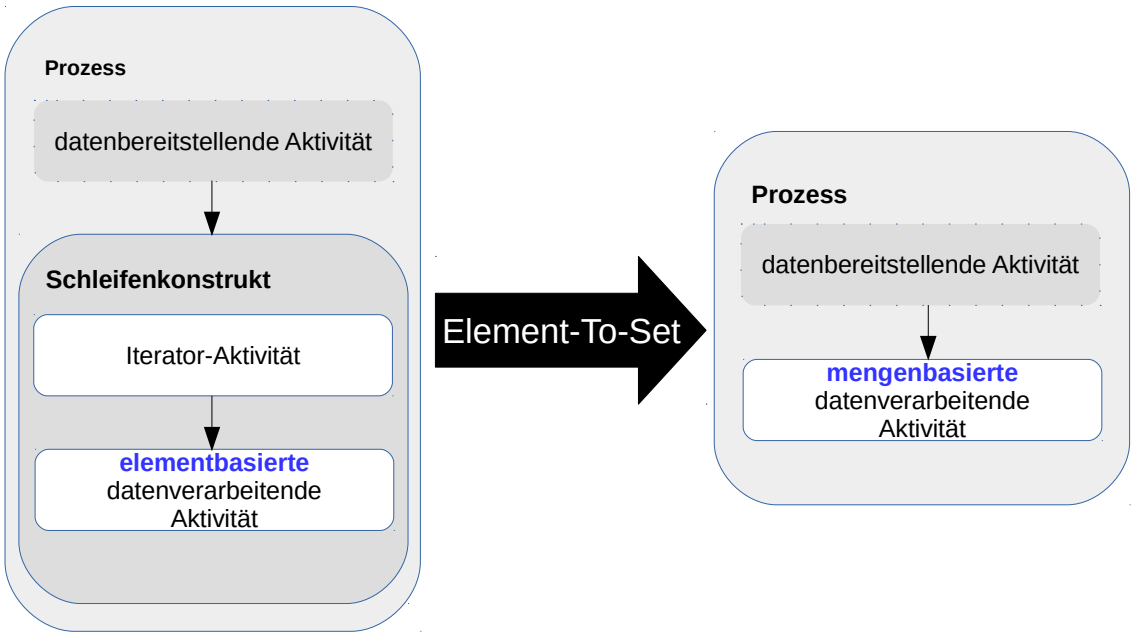


Abbildung 7.6: Das Grundmuster der Element-To-Set-Regel

- Das Datenverarbeitungsmuster, auf das die *Element-To-Set-Regel* angewendet werden kann, beginnt mit einer Aktivität, die eine *For-Schleife* realisiert.
- Dieses wiederum enthält eine Aktivität oder mehrere Aktivitäten, welche eine iterative Verarbeitung der zu verarbeitenden Daten- bzw. Dateimenge ermöglicht. Dabei wird in jedem Schleifendurchlauf z.B. ein Zähler aktualisiert oder der aktuell zu bearbeitende Datensatz bzw. die aktuell zu bearbeitende Datei in entsprechende Variablen geladen. Diese Art von Aktivität wird in dieser Arbeit als **Iterator-Aktivität** bezeichnet.
- Innerhalb des Schleifenrumpfes wird eine weitere Aktivität ausgeführt, welche eine Datenoperation **elementbasiert**, d.h. auf dem aktuellen Datensatz bzw. der aktuellen Datei, definiert. Dabei verarbeitet sie die jeweils aktuellen Werte, die in den von der Iterator-Aktivität geschriebenen Variablen bereitgestellt werden. Diese Aktivität wird in dieser Arbeit als **datenverarbeitende Aktivität** bezeichnet.
- Es existiert innerhalb des Schleifenrumpfes nur eine datenverarbeitende Aktivität, sodass die Iterationsmenge während des Schleifendurchlaufs unverändert bleibt.
- Es kann in der Befehlssprache der Datenressource, welcher der auszuführenden, elementbasierten Datenverarbeitunsoperation zu Grunde liegt, eine mengenbasierte Anweisung formuliert werden, welche die ursprünglich, iterativ durchlaufene Menge vollständig und ausschliesslich erfassen und die Datenverarbeitungsoperation auf dieser Menge ausführen kann.

Sind obige Bedingungen erfüllt, können als **Regelaktion** folgende Transformationsschritte durchgeführt werden:

- Die elementbasierte Ausführung der Datenoperation wird in eine mengenbasierte Ausführung umgewandelt. Dazu wird aus der elementbasierten Anweisung der datenverarbeitenden Aktivität eine mengenbasierte Anweisung abgeleitet, indem z.B. die Anfrage der datenbereitstellenden Aktivität mit der ursprünglichen Anweisung der datenverarbeitenden Aktivität verschmolzen wird. Die mengenbasierte Anweisung erfasst alle relevanten Daten vollständig und ausschliesslich, sodass auf der Workflow-Ebene eine Iteration über der Iterationsmenge nicht mehr notwendig ist.
- Die resultierende Anweisung wird von einer neuen mengenbasierten, datenverarbeitenden Aktivität ausgeführt, welche die Semantik des gesamten Schleifenkonstrukts im ursprünglichen Workfow (d.h. der Iteratoraktivität, der datenverarbeitenden Aktivität und gegebenenfalls der datenbereitstellenden Aktivität) abdeckt.
- Der Kontrollfluss im Workflow muss wie folgt angepasst werden: Zunächst werden alle Aktivitäten entfernt, die in Bezug zum Schleifenkonstrukt standen. Dabei handelt es sich um die Aktivität, die das Schleifenkonstrukt an sich repräsentiert, die Iterator-Aktivität(en), sowie gegebenenfalls die datenbereitstellende Aktivität.
- Die Variablen, in welche zuvor die aus dem Workflow entfernten Aktiviäten geschrieben haben, werden ebenfalls entfernt, wenn sie von anderen Aktivitäten im Workflow nicht mehr referenziert werden.

7.2.2 Die Regelklasse Activity-Merging

Diese Regelklasse entspricht der gleichnamigen Regelklasse des PGM/F-Rahmenwerks und generalisiert sie. Ziel der *Activity Merging-Regeln* ist es, zwei Aktivitäten miteinander zu verschmelzen, die sequentiell jeweils eine Datenverarbeitungsoperation ausführen. Ist eine Verschmelzung möglich, wird die nachfolgende, von der ersten Anweisung abhängige Anweisung derart transformiert, dass die beiden zuvor getrennt ausgeführten Datenverarbeitungsoperationen zu einer einzelnen kombiniert werden. Zu beachten ist, dass durch die Verschmelzung keine Datenflussabhängigkeiten zu anderen Aktivitäten beeinflusst werden dürfen. So kann z.B. eine Assign-Aktivität nicht vollständig entfernt werden, falls sie eine weitere Variablenzuweisung durchführt, die bei einer dritten Aktivität endet. Des Weiteren muss die ursprüngliche Ausführungssemantik erhalten bleiben. Diese wird in [Vrh11] ausführlich diskutiert. Folgende Regeln gehören zu der Menge der *Activity-Merging- Regeln*: Die Regeln *Web-Service-Pushdown*, *Assign-Merging*, *Eliminate-Temporary-Container*, *Predicate-Pushdown*, sowie die Regeln der Regelunterklasse *Update-Merging*.

Um eine *Activity-Merging-Regel* korrekt anwenden zu können, müssen folgende **Regelbedingungen** erfüllt sein:

- Es müssen zwei Aktivitäten existieren, die jeweils eine Datenverarbeitungsoperation ausführen.
- Bei der ersten Aktivität kann es sich entweder um eine Aktivität handeln, die eine Variablenzweisung ausführt, einen Web-Service aufruft oder direkt auf einer Datenressource arbeitet bzw. Daten abfragt (z.B. eine RetrieveData-Aktivität). Die zweite Aktivität muss eine Anweisung auf einer Datenressource repräsentieren.

- Durch das Verschmelzen der Aktivitäten werden keine anderen Datenabhängigkeiten im Workflow beeinflusst.
- Ferner müssen für die unterschiedlichen *Activity-Merging Regeln* unter Umständen weitere spezifische Bedingungen erfüllt sein. Diese werden in den jeweiligen Abschnitten besprochen.

Sind obige Bedingungen erfüllt, können die Datenverarbeitungsoperationen verschmolzen werden. Dabei werden als **Regelaktion** folgende Transformationsschritte durchgeführt:

- Durch die Verschmelzung wird die zweite Datenverarbeitungsoperation in eine neue Operation überführt.
- Aus der zweiten Aktivität wird eine neue Aktivität abgeleitet, welche diese neue Operation ausführt.
- Anschliessend kann die erste Aktivität aus der Ausführungslogik entfernt werden.
- Dabei müssen Kontrollfluss- und Datenabhängigkeiten wie folgt angepasst werden: Der direkte Vorgänger und Nachfolger der entfernten Aktivität müssen miteinander verbunden werden.
- Für die Variablen, die von der entfernten Aktivität referenziert wurden, müssen alle Datenabhängigkeiten neu berechnet werden.
- Variablen und Datenressourcen, die nach der Regelanwendung weder von der neuen Aktivität noch von einer anderen Aktivität referenziert werden, werden entfernt.

In den folgenden Unterabschnitten werden einige zur Regelklasse *Activity-Merging* gehörende Optimierungsregeln beschrieben. Die Beschreibung der ebenfalls zur selben Klasse gehörenden *Predicate-Pushdown-Regel* und seiner Varianten erfolgt aus Übersichtlichkeitsgründen separat in Abschnitt 7.2.3 und in Abschnitt 7.2.4.

Die Assign Merging-Regel

Diese Regel entspricht der gleichnamigen Regel des PGM/F-Rahmenwerks und generalisiert sie. Die *Assign-Merging-Regel* eliminiert eine Variablenzuweisung, die in einer Aktivität definiert ist, indem die Zuweisung direkt in einer abhängigen, nachfolgenden Aktivität durchgeführt wird. Sind beide Anweisungen in derselben Befehlssprache formuliert, ist die Substitution in der Regel problemlos realisierbar. Sind die verwendeten Befehlssprachen jedoch unterschiedlich, muss fallspezifisch geprüft werden, ob solch eine Substitution durchführbar ist. Angenommen eine Assign-Aktivität ermittelt aus einer vorliegenden Workflow-Variable mit Hilfe eines XPath-Ausdruckes einen bestimmten Wert und weist diesen einer Variablen w zu. Arbeitet nun eine datenabhängige Anweisung auf einer XML-Datenbank unter Verwendung der Befehlssprache XQuery und referenziert dabei w, so kann die in der Befehlssprache XPath formulierte Definition von w problemlos in die XQuery-Anweisung eingebettet werden, da XQuery alle XPath-Befehle vollständig unterstützt. Handelt es sich bei der datenabhängigen Aktivität jedoch um eine IssueCommand-Aktivität, die einen Shell-Befehl auf einem Dateisystem ausführt, so kann der XPath-Befehl nicht nahtlos in den Shell-Befehl eingebettet werden und es muss versucht werden diesen in Shell-Sprache umzuformulieren.

Um eine *Assign-Merging-Regel* korrekt anwenden zu können, müssen folgende **Regelbedingungen** erfüllt sein:

- Es existiert eine Aktivität a_i, die eine Variablenzuweisung von einer Variablen v zu einer Variablen w ausführt. Dabei wird z.B. ein Literalwert, ein Variablenwert oder ein XPath-Ausdruck, der bestimmte Werte berechnet, zugwiesen.
- w wird in der Anweisung einer nachfolgenden Aktivität a_j exklusiv gelesen.
- Die in w eingesetzte Definition der Variablen v kann in die Anweisung von a_j verschoben werden. Dazu muss entweder die Definition der Variablen v mit der Befehlssprache der Anweisung von a_j **kompatibel** sein. Kompatibilität bedeutet in diesem Zusammenhang dass die Kombination beider Befehlssprachen innerhalb einer Anweisung durch die zu Grunde liegende Datenressource unterstützt wird. (z.B. SQL-Datenbanken, die XPath-Ausdrücke unterstützen) oder die Definition des Werts von v muss in der Befehlssprache der Anweisung von a_j **umformulierbar** sein.

Sind obige Bedingungen erfüllt, können als **Regelaktion** folgende Transformationsschritte durchgeführt werden:

- Das Vorkommen der Variablen w in der Anweisung der Aktivität a_j wird entweder direkt durch die Definition der Variable v oder durch eine Umformulierung ersetzt und somit eine neue Anweisung abgeleitet.
- Die Aktivität a_i wird aus dem Workflow entfernt, falls sie keine weiteren Zuweisungen ausführt (vgl. Teilbedingung 3 beim Activity Merging), die bei dritten Aktivitäten enden. Zudem wird die Variable w gelöscht, falls sie nicht mehr von anderen Aktivitäten im Workflow referenziert wird.

Die Regelklasse Update-Merging

Diese Regelklasse entspricht der gleichnamigen Regelklasse des PGM/F-Rahmenwerks und generalisiert sie. Das Ziel der Regeln dieser Regelklasse ist es, Aktivitäten, die Aktualisierungoperationen auf derselben Datenressource ausführen, durch geeignete Kombination ihrer Anweisungen zu einer einzelnen Aktivität zu verschmelzen. Die Kombination der einzelnen Anweisungen erfolgt dabei durch Verwendung datenquellspezifischer Trennzeichen (z.B. & bei Shell-Befehlen), geeigneter Mengenoperationen (z.B. UNION ALL bei SQL), Anpassungen von Klauseln oder FOR-Anweisungen (XQuery). Durch die gleichzeitige Ausführung können Optimierungseffekte erzielt werden, indem beispielsweise vor der Verschmelzung ausgeführte Berechnungsschritte nicht mehr wiederholt in zwei verschiedenen Anweisungen ausgeführt werden. Ferner kann bei gegebener Unterstützung der zu Grunde liegenden Datenressource eine parallele Ausführung von Teilen der Anweisungen auf der Datenebene zu einer beschleunigten Verarbeitung führen.

Bei der Verschmelzung der Aktivitäten muss sichergestellt sein, dass durch die gleichzeitige Ausführung der Aktualisierungsoperationen auf der Datenressource dasselbe Resultat erzielt wird, wie durch ihre ursprünglich sequentielle Ausführung. Um die Ausführbarkeit einer resultierenden, kombinierten Anweisung sicherzustellen müssen beide Aktivitäten auf derselben Datenressource arbeiten.

Alternativ kann es sich auch um zwei Datenressourcen sein, die ihre Daten selbständig miteinander austauschen können (z.B. MySQL Cluster). Je nach Aktualisierungsoperation wird hier zwischen einer *Insert-Insert-*, einer *Delete-Delete-Merging-* und einer *Update-Update-Merging-Regel* unterschieden. Im Folgenden werden die *Insert-Insert-*, sowie die *Delete-Delete-Merging-Regel* beschrieben. Aufgrund der grossen Ähnlichkeit der Regeln wird die *Update-Update-Merging-Regel* nicht behandelt.

- **Insert-Insert-Merging-Regel:**

  Diese Regel kombiniert zwei Insert-Anweisungen, die auf bestimmten Datensätzen bzw. Datencontainern ausgeführt werden, mit einer Kombinations-Operation zu einer einzelnen Insert-Anweisung. Dies setzt voraus, dass die verwendete Befehlssprache solche eine Kombinationsoperation anbietet. Bei Dateisystemen z.B. können zwei Echo-Befehle, die sequentiell in dieselbe Textdatei schreiben, auf einfache Weise durch einen Echo-Befehl ersetzt werden, indem dieser beide einzufügenden Texte als Parameter erhält. In XQuery können mit der XQuery-Upate-Facility 1.0 mehrere Knoten eingefügt werden (vergleiche [RCD+11]). Die Realisierbarkeit der Kombination ist oft fallspezifisch von der Semantik abhängig. Beispielsweise ist es in XQuery nicht möglich innerhalb desselben Update-Statements nach dem ersten Knoten und dem vierten Knoten in einem XML-Dokument jeweils neue Knoten einzufügen. Besteht ein XML-Dokument nur aus zwei Knoten und müssen nach jedem Knoten jeweils neue Knoten eingefügt werden, so kann dies mit einem einzelnen FLWOR-Ausdruck realisiert werden.

- **Delete-Delete-Merging-Regel:** Die *Delete-Delete-Merging-Regel* kombiniert zwei Delete-Anweisungen, die nacheinander auf bestimmten Datencontainern bzw. Datensätzen ausgeführt werden, mit Hilfe einer Kombinations-Operation zu einer einzelnen Delete-Anweisung. Dies setzt voraus, dass die verwendete Befehlssprache solche eine Kombinationsoperation bereistellt. Im einfachsten Fall können z.B. zwei Löschoperationen, welche zwei Dateien auf einem Unix-Dateisystemen löschen, zu einem DEL-Befehl kombiniert werden, indem dieser beide Dateinamen als Parameter erhält. Die Realisierbarkeit der Kombination ist oft fallspezifisch von der Semantik abhängig. Beispielsweise ist es in XQuery nicht möglich in einem XML-Dokument mit zehn Knoten innerhalb desselben Update-Statements den ersten und dritten Knoten zu löschen. Würde die Anforderung jedoch lauten, alle zehn Knoten im aktuellen XML-Dokument zu löschen, wäre dies mit einem einzelnen Ausdruck realisierbar.

Die Eliminate-Temporary-Container-Regel

Diese Regel entspricht der *Eliminate-Temporary-Table-Regel* des PGM/F-Rahmenwerks. Sie entfernt redundante, temporäre Datencontainer aus einem Workflow, deren gespeichertes Zwischenergebnis von genau einer abhängigen Anweisung referenziert wird. Der Begriff temporärer Datencontainer bezeichnet in diesem Zusammenhang neben temporären, relationalen Tabellen auch XML-Rowset-Strukturen, Verzeichnisse auf Dateisystemen und BPEL-Variablen, in denen Daten bzw. Dateien zwischengespeichert werden.

Ein Beispiel wäre hier z.B. eine Retrieve-Data-Aktivität, welche eine Anfrage auf einer XML-Datenbank ausführt und das Anfrageergebnis in eine Workflow-Variable v des Typs tXMLDataFormat abspeichert. Führt nun eine anschliessende Aktivität auf derselben XML-Datenbank, jedoch nicht zwangsweise auf demselben XML-Dokument eine weitere Anfrage aus und referenziert dabei v lesend, so kann

durch Anwendung der *Eliminate-Temporary-Container-Regel* die Variable *v* in der Anweisung der zweiten Aktivität direkt durch ihre Definition aus der ersten RetrieveData-Aktivität ersetzt werden. Dadurch werden die erste RetrieveData-Aktivität und die Variable *v* überflüssig und können entfernt werden.

Um eine *Eliminate-Temporary-Container-Regel* korrekt anwenden zu können, müssen folgende **Regelbedingungen** erfüllt sein:

- Es existiert eine Aktivität a_i, die mit einer Anweisung DM_i einen zuvor leeren Datenbehälter Temp(referenziert durch eine Variable v_i) mit Daten füllt.
- Diese wird anschliessend von einer weiteren Aktivität a_j mit einer Anweisung DM_j exklusiv gelesen.
- Bei einem Verschmelzen der Anweisungen muss die Berechnung der Zwischenergebnisse in die Anweisung von a_j verschoben werden können. Dazu muss entweder die Definition der Zwischenergebnisse mit der Befehlssprache der Anweisung von a_j **kompatibel** sein. Kompatibilität bedeutet in diesem Zusammenhang dass die Kombination beider Befehlssprachen innerhalb einer Anweisung durch die zu Grunde liegende Datenressource unterstützt wird, oder die Definition der Zwischenerebnisse muss in die Befehlssprache der Anweisung von a_j **umformulierbar** sein.

Sind obige Bedingungen erfüllt, können als **Regelaktion** folgende Transformationsschritte durchgeführt werden:

- Die Datenabhängigkeit wird aufgelöst, indem das Vorkommen von v_i in der Anweisung DM_j der Aktivität a_j direkt durch die definierende Anweisung DM_i der Aktivität a_i ersetzt wird. Als Ergebnis erhält man die neue Anweisung DM_{i+j}
- Die Aktivität a_j wird durch eine Aktivität a_{i+j} ersetzt, welche diese neue Anweisung ausgeführt DM_{i+j} ausführt.
- Durch den Substitutionsschritt wird die Referenzierung des temporären Datencontainers überflüssig. Die Variable v_i, sowie die Aktivität a_i werden entfernt.

Die WebService-Pushdown-Regel

Diese Regel entspricht der *Web-Service-Pushdown-Regel* des PGM/F-Rahmenwerks und generalisiert sie. Sie verlagert einen Web-Service-Aufruf von der Workflow- auf die Datenebene. Dabei wird der Web-Service-Aufruf in eine Anweisung der zu Grunde liegenden Datenressource eingebettet und zusammen mit der Anweisung ausgeführt.

Voraussetzung für die Anwendbarkeit dieser Regel ist somit die *Unterstützung von Web-Service-Aufrufen durch die zu Grunde liegende Datenressource.* Bei SQL-Datenbanksystemen wird dies üblicherweise durch die Bereitstellung von *WS-UDTF*s bewerkstelligt. Bei Unix-Dateisystemen können

Listing 7.1 XQuery-Befehl, das einen Web Service aufruft

```
import module namespace http = "http://expath.org/ns/http-client" at "http-client.xqm";
let $url:= 'http://neo.jpl.nasa.gov/risk/'
let $req := <http:request href="{$url}" method="get"/>
return
    http:send-request($req)[1]
```

z.B. Tools wie *Curl* ([Ste]) und *GNU WGet*[1] verwendet werden, um Web-Services auf Kommandozeilenebene aufzurufen. Mit beiden Tools können Daten mit URL-Syntax ohne Benutzerinteraktion übertragen werden, insbesondere auch HTTP POSTs und GETs durchgeführt werden, was Web-Service-Aufrufe auf SOAP/HTTP-Grundlage ermöglicht. In [EFS08] wird eine *Web Service Facility* als Erweiterung für XQuery vorgeschlagen, welche den Aufruf eines Web-Services und die Weiterverarbeitung der Ergebnisse in einem XQuery-Ausdruck erlaubt. Des Weiteren unterstützt z.B. das native Datenbanksystem *existDB* eine *Expath HTTP Client Library*[2], mit der in XQuery-Anweisungen Http-Requests gesendet und somit auch Web Services aufgerufen werden können. Das Beispiel in Listing 7.1 zeigt, wie mit Hilfe der URL des Web Services und der gewünschten HTTP-Methode GET ein Web Service aufgerufen und aus dem Ergebnis nur das erste Element selektiert wird. D.h. aus der HTTP-Response des Web-Services können mit benutzerdefninerten XQuery-Funktionen verschiedene Elemente herausgefiltert bzw. kombiniert werden.

Es müssen folgende **Regelbedingungen** erfüllt sein, damit eine *Web-Service-Pushdown-Regel* korrekt angewendet werden kann:

- In einem Simulationsworkflow muss eine Aktivitat a_i existieren, die auf der Workflow-Ebene eine Web-Service- Operation aufruft (z.B. eine Invoke-Aktivität).
- Diese Aktivitat besitzt *n* Eingabeparameter, die als Eingabeparameter für den aufzurufenden Web-Service dienen und *m* Ausgabeparameter, in welche die Ergebnisparameter der aufgerufenen Web-Service-Operation geschrieben werden. Diese Ergebnisparameter werden in Workflow-Variablen *v* geschrieben.
- *Des Weiteren muss auf der Datenebene eine spezifische Funktion, ähnlich einer WS-UDTFs, oder ein Befehlstyp vorhanden bzw. definierbar sein, welche die direkte Ausführung von Web Service-Aufrufen auf der Datenebene gewährleistet.* Dies beinhaltet die Konstruktion einer SOAP-Anfragenachricht aus den Eingabeparametern dieser Funktion und eventuell aus weiteren Informationen aus der WSDL-Beschreibung des aufzurufenden Web-Services (wie z.B. dem Service-Endpoint oder dem Namen der aufzurufenden Operation) und ihre Übermittlung, sowie die Extraktion des Anfrageergebnisses aus der SOAP-Antwortnachricht und ihre Abbildung auf entsprechende Strukturen der zu Grunde liegenden Datenressource.
- Eine nachfolgende Aktivität a_j liest die von a_i geschriebene Variable *v* exklusiv.

Sind obige Bedingungen erfüllt, konnen als **Regelaktion** folgende Transformationsschritte durchgeführt werden:

[1] https://www.gnu.org/software/wget/
[2] http://www.expath.org/ modules/http-client/

- Die Aktivität a_i wird entfernt und im Workflow durch eine neue Aktivität a_{i+j} ersetzt.
- Alle ursprünglichen Daten- und Kontrollflussabhängigkeiten zwischen der entfernten Aktivität und anderen Aktivitäten im ursprünglichen Workflow bleiben erhalten.
- Die Aktivität a_{i+j} wird mit der Datenressource verknüpft, auf dem der Web-Service-Aufruf ausgeführt werden soll.
- Sie führt eine Anweisung aus, in der eine Funktion bzw. Befehl eingebettet ist, welche die Web-Service-Operation auf Datenebene aufruft.
- Diese Funktion liest die *n* Eingabeparameter von a_{i+j} und die *m* Ausgabeparameter der Web-Service-Operation werden in *m* Ausgabeparameter von a_{i+j} geschrieben.

7.2.3 Die Predicate-Pushdown-Regel

Diese Regel entspricht der gleichnamigen *Predicate-Pushdown-Regel* des PGM/F-Rahmenwerks und generalisiert sie. Die Optimierungsidee einer frühzeitigen Ausführung von Selektionsoperationen zur Verkleinerung der Grösse von Zwischenergebnissen lässt sich ebenso auf die Optimierung datenintensiver Simulationsworkflows übertragen. Unter dem Begriff Selektions-Operation ist hier nicht zwangsläufig eine SQL-SELECT-Anweisung zu verstehen, sondern ein Überbegriff, der vielmehr die **Filterung** der relevanten Datenmenge bezeichnet, beispielsweise durch die Ausführung einer XPath-Anweisung.

Für eine Anwendung der *Predicate-Pushdown-Regel* müssen folgende **Regelbedingungen** erfüllt sein:

- Es existiert eine datenbereitstellende Aktivität a_i, die die eine Anfrage auf einer Datenressource ausführt und das Anfrageergebnis in einer Workflow-Variable *v* speichert. Optional wird *v* urch eine Zuweisungsoperation ein Literalwert (fixed value) zugewiesen (z.B. die Initialisierung einer *data container reference list*). Das Anfrageergebnis bzw. die Initialmenge wird in der Workflow-Variablen *v* zur Verfügung gestellt.
- Die Variable *v* wird von einer nachfolgenden Aktivität a_j exklusiv gelesen. Dabei führt a_j eine Selektionsoperation aus, welche Daten aus *v* filtert.
- Für das Verschieben der Selektionsoperation in die datenbereitstellende Aktivität a_i muss die Befehlssprache, in der die von a_i ausgeführte Anweisung formuliert ist, **kompatibel** sein mit der Befehlssprache, in der die Selektionsoperation von a_j formuliert ist. Ist aufgrund einer Inkompatibilität der Befehlssprachen eine direkte Einbettung nicht möglich, muss die Möglichkeit bestehen die Selektionsoperation in der Befehlssprache der ersten Anweisung **umformulieren** zu können.

Sind obige Bedingungen erfüllt, konnen als **Regelaktion** folgende Transformationsschritte durchgeführt werden:

- Durch die Regelanwendung wird aus a_i eine neue Aktivität a_{i+j} abgeleitet, die eine Anweisung ausführt, welche durch die Verschiebung der Selektionsoperation von a_j in die Anweisung von a_i entsteht.

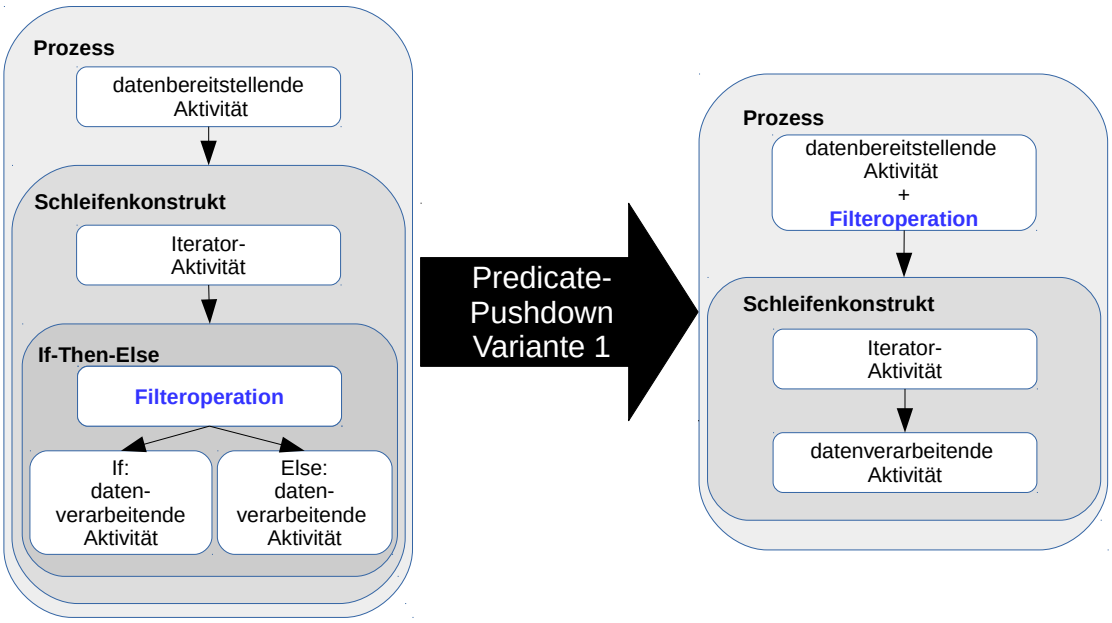


Abbildung 7.7: Das Grundmuster der ersten Variante der Predicate-Pushdown-Regel: Verschieben der Filteroperation in datenbereitstellende Aktivität

- Durch das Verschieben der Selektionsoperation von der Workflow- auf die Datenebene, wird a_j überflüssig und kann entfernt werden.
- Ferner wird die Variable v entfernt, falls sie nicht von anderen Aktivitäten im Workflow referenziert wird.

7.2.4 Varianten der Predicate-Pushdown-Regel

Im Folgenden werden Varianten der *Predicate-Pushdown-Regel* beschrieben. Dabei entspricht die erste vorgestellte Variante der entsprechenden gleichnamigen Variante des PGM/F-Rahmenwerks. Zudem wird eine zusätzliche, *zweite Variante der Predicate-Pushdown-Regel* vorgestellt, welche im Rahmen dieser Arbeit entwickelt wurde und auf der ersten Variante basiert. Die Motivation für dies lag darin, dass in bestimmten Fällen, in denen eine datenbereitstellende Aktivität in BPEL-DM-Worklows nicht existiert bzw. in denen aufgrund Inkompabilitäten der Befehlssprachen ein Verschieben der Selektionsoperation in die datenbereitstellende Aktivität nicht möglich ist, die *erste Variante der Predicate-Pushdown-Regel* nicht Anwendung finden kann. Die zweite Variante der Regel erlaubt es hier, dennoch Transformationen auszuführen und bildet somit die Grundlage für die Anwendung weiterer Optimierungsregeln.

Erste Variante der Predicate-Pushdown-Regel

In Abbildung 7.7 ist das Grundmuster der ersten Variante der Predicate-Pushdown-Regel gezeigt. Die **erste Variante** der *Predicate-Pushdown-Regel* bezieht sich auf ein Datenverarbeitungsmuster, das mit einer Aktivität beginnt, welche mit einer Anfrage auf einer Datenressource Daten in den Worklfow-Kontext lädt. Alternativ wird mit einer Zuweisungsoperation eine Workflow-Variable initialisiert. Die geladenen Daten dienen einem nachfolgenden For-Konstrukt als Iterationsmenge. Im Schleifenrumpf existiert eine *Iterator-Aktivität*, welche in jedem Schleifendurchlauf den nächsten Datensatz der Datenmenge in einer Variablen bereitstellt. Anschliessend folgt eine If-Aktivität, welche mit Hilfe ihrer Bedingungsanweisung entscheidet, ob sich der vorliegende Datensatz für eine Weiterverarbeitung qualifiziert oder nicht.

Durch die If-Aktivität im Schleifenrumpf erfolgt auf der Iterations-Datenmenge erst spät eine Filterung. Dadurch enthält die zu Beginn des Workflows geladene Iterationsmenge Daten, die auf der Workflow-Ebene nicht benötigt werden. D.h. im Falle einer Anfrage entstehen unnötig Übertragungskosten von der Daten- auf die Workflowebene. Zudem sind die Ausführungskosten des Schleifenkonstrukts proportional abhängig von der Grösse der Iterationsmenge.

Bei der ersten Variante der *Predicate-Pushdownregel* wird die Filteroperation in der Bedingungsanweisung der If-Aktivität in die datenbereitstellende Aktivität vor der Ausführung des Schleifenkonstrukts verschoben. Dadurch wird aus der datenbereitstellenden Aktivität eine neue Aktivität abgeleitet. Diese frühzeitige Filterung kann zu einer Verkleinerung der Iterationsmenge führen, so dass nur solche Daten in der zu iterierenden Datenmenge enthalten sind, welche sich für eine Weiterverarbeitung qualifizieren. Dies resultiert in einer schnelleren Ausführung der For-Schleife. Zudem wird die If-Aktivität im Schleifenrumpf überflüssig und kann entfernt werden. Dies wiederum bildet gegebenenfalls die Grundlage für die Anwendung der *Element-To-Set-Regel*, durch die das gesamte Schleifenkonstrukt entfernt werden kann.

Zweite Variante der Predicate-Pushdown-Regel

Die **zweite Variante** der *Predicate-Pushdown-Regel* bezieht sich auf die Verlagerung der Bedingungsanweisung, die in einer If-Aktivität ausgeführt wird, in die datenverarbeitende Aktivität im If- bzw. Else-Zweig bzw. in eine vorhergehende Aktivität in der Ausführungsreihenfolge des Workflows, wobei es sich nicht um die datenbereitstellende Aktivität handelt. Das Grundmuster dieser Regel ist in Abbildung 7.9 gezeigt. Diese Variante bietet sich an, wenn vor dem Schleifenkonstrukt keine datenbereitstellende Aktivität existiert, in welche die Filteroperation verschoben werden könnte. Dies ist häufig der Fall, wenn in einem Workflow Dateien auf Dateisystemen bearbeitet werden (z.B. im Anwendungsszenario in Abschnitt 8.7). Das Enfernen des If-Konstrukts ist dennoch wünschenswert, um den Workflow zu vereinfachen und die Anwendung weiterer Optimierugnsregeln zu ermöglichen.

Ein anderers Anwendunsszenario ist, wenn zwar eine datenbereitstellende Aktivität am Anfang des Workflows existiert, es jedoch nicht möglich ist, die Filteroperation in diese Aktivität zu verschieben. Dieses Szenario wird für einen Test-Workflow in Abschnitt 8.6 noch einmal erörtert.

Für die *zweite Variante der Predicate-Pushdown-Regel* werden zwei Optionen festgelegt.

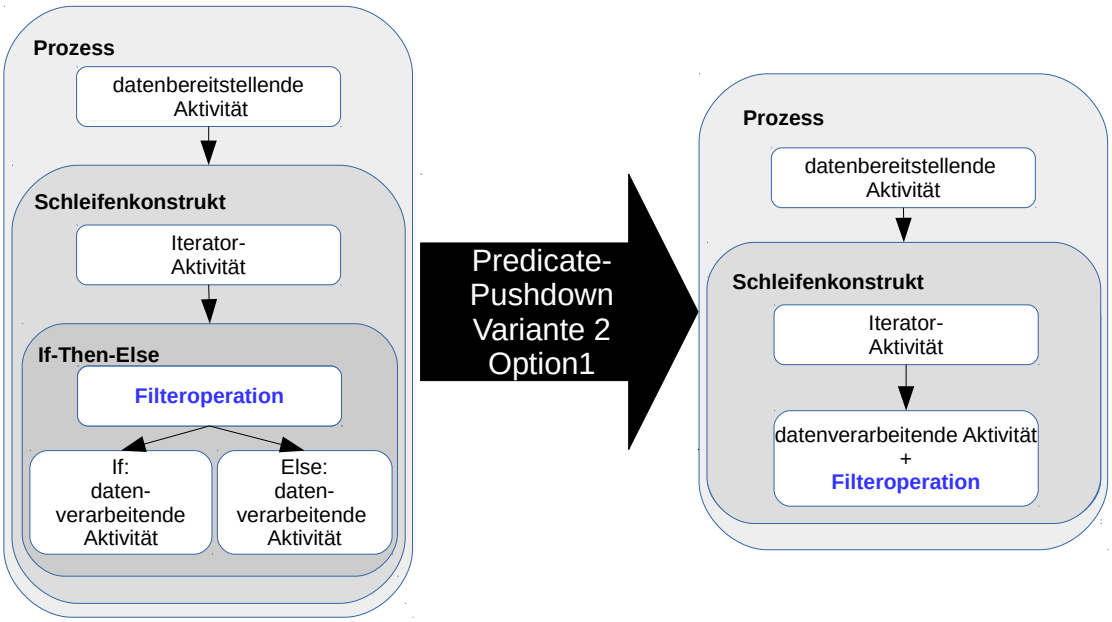


Abbildung 7.8: Das Grundmuster der ersten Option der zweiten Predicate-Pushdown-Regelvariante: Verschieben der Filteroperation in direkt nachfolgende Aktivität

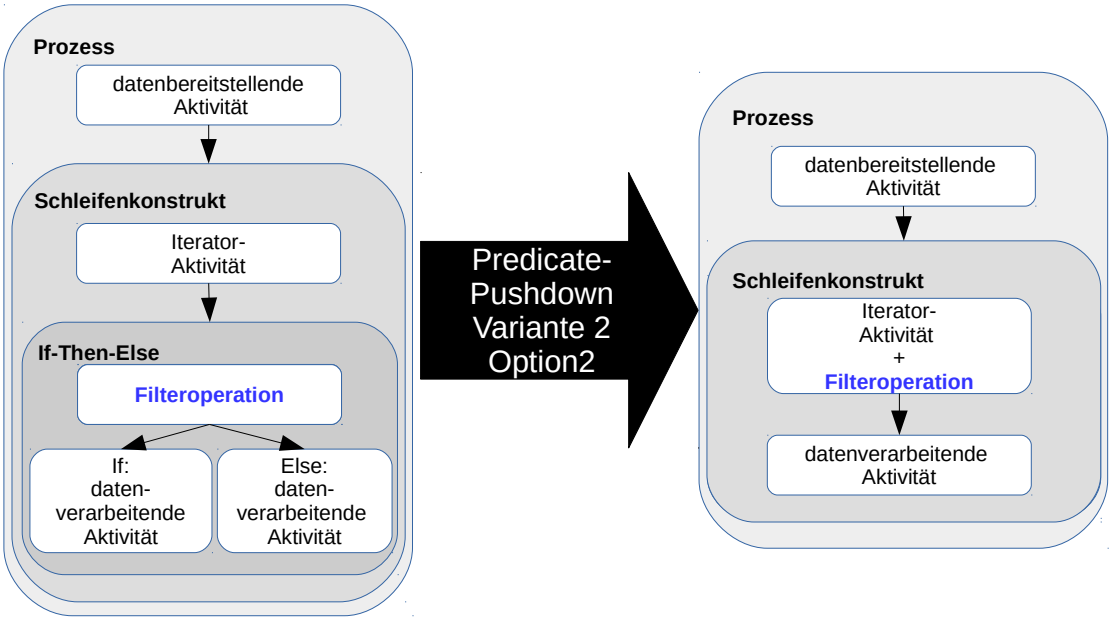


Abbildung 7.9: Das Grundmuster der zweiten Option der zweiten Predicate-Pushdown-Regelvariante: Verschieben der Filteroperation in direkt vorhergehende Aktivität

- **Option 1:** Bei der *ersten Option der zweiten Predicate-Pushdownregelvariante* wird die Filteroperation in der Bedingungsanweisung der If-Aktivität in die direkt nachfolgende, datenverarbeitende Aktivität(en) in der Ausführungsreihenfolge des Workflows verschoben. So wird aus der If-Aktivität eine neue Aktivität erzeugt (z.B. eine IssueCommand-Aktivität), die eine Anweisung ausführt, welche die Semantik sowohl des If-Zweigs, als auch des Else-Zweigs abdeckt. Das If-Konstrukt wird überflüssig und kann entfernt werden. Das Grundmuster dieser Regel ist in Abbildung 7.8 gezeigt.
- **Option 2:** Bei der *zweiten Option der zweiten Predicate-Pushdownregelvariante* wird die Filteroperation in die direkt vorhergehende Aktivität in der Ausführungsreihenfolge des Workflows verschoben. Dabei handelt es sich häufig um die Iterator-Aktivität innerhalb des Schleifenrumpfs. Dadurch wird aus der vorhergehenden Aktivität eine neue Aktivität abgeleitet, die eine Anweisung ausführt, welche durch das Verschieben der Filteropartion der If Aktivität entstanden ist. Das If-Konstrukt wird überflüssig und kann entfernt werden.

Ziel dieser Regel ist es, durch das Verschieben der Filterung von einem If-Konstrukt in eine vor- oder nachgelagerte Aktivität eine Reduktion der Anzahl der Aktivitäten zu bewirken. Im Bezug auf das Laufzeitverhalten des Workflows können durch die Anwendung dieser Regelvariante meistens direkt keine Optimierungseffekte erzielt werden, der Workflow kann durch diesen Transformationsschritt jedoch vereinfacht werden und die Grundlage für die Anwendung weiterer Optimierungsregeln wie z.B. der *Element-To-Set-Regel* bilden.

Hier gilt ebenso wie für die *erste Variante der Predicate-Pushdown-Regel*, dass ein Verschieben der Anweisung einer If-Aktivität in eine andere Anweisung nur dann möglich ist, wenn die beiden Befehlsprachen, in denen die jeweiligen Anweisungen formuliert sind, miteinander kompatibel sind bzw. es eine Möglichkeit gibt die Anweisung der If-Aktivität in der Befehlssprache der Aktivität zu formulieren, in der sie eingebettet werden soll.

8 Umsetzung und Anwendungsbeispiele für die generischen Optimierungsregeln

In diesem Kapitel wird die genaue Funktionsweise ausgewählter, in Kapitel 7 beschriebener Optimierungsregeln veranschaulicht. Dazu wird zunächst die Anwendung bestimmter Regeln für eigens erstellte Anwendungsbeispiele beschrieben. In den folgenden Abschnitten wird jeweils die Anwendung der *Web-Service-Pushdown-Regel* (Abschnitt 8.1), der *Assign-Merging-Regel* (Abschnitt 8.2), der *Delete-Delete-Merging-Regel* Abschnitt 8.3, der *Eliminate-Temporary-Container-Regel* (Abschnitt 8.4), der *Element-To-Set-Regel* (Abschnitt 8.5)und der *Predicate-Pushdown-Regel* (Abschnitt 8.6) auf ein Anwendungsbeispiel betrachtet. Anschliessend wird in Abschnitt 8.7 die kombinierte Anwendung der *Predicate-Pushdown-Regel* mit der *Element-To-Set-Regel* aufgezeigt. Zum Abschluss wird die Anwendung von Optimierungsregeln auf die für diese Arbeit relevanten Beispiel-Worklflows der Proteinmodellierung (Abschnitt 8.8) und der biomechanischen bzw. systembiologischen Simulation (Abschnitt 8.9) erläutert.

Da die neuen Optimierungsregeln aus Regeln für BPEL/SQL-Workflows abgeleitet wurden, gestaltet sich ihre Anwendung auf BPEL-DM-Workflows, welche Operationen auf SQL-Datenbanken ausführen, verhältnismässig intuitiv. Aus diesem Grund wurden bei den Umsetzungen SQL-Datenbanksysteme so gut wie gar nicht betrachtet und stattdessen auf Anwendungsszenarien mit Dateisystemen und XML-Ressourcen fokussiert.

8.1 WebService-Pushdown

In folgendem Beispiel soll die Anwendung der **WebService-Pushdown-Regel** veranschaulicht werden. Als Anwendungsszenario dient dabei ein BPEL-DM-Workflow, der eine simple Funktion erfüllt. Er ruft einen öffentlichen Web-Service globalweather [1] auf, der einen aktuellen Wetterreport für eine gegebene Stadt zurückliefert. Das Resultat des Web-Service-Aufrufes wird anschliessend mit Hilfe eines IssueCommand-Befehls in eine Text-Datei WSResult.txt auf dem lokalen Unix-System geschrieben. In Abbildung 8.1 links ist der BPEL-DM-Prozess-Graph für den konventionellen, unoptimierten Workflow abgebildet.

Eingeleitet wird der Workflow durch eine simulationStart-Aktivität, welches innerhalb des SIMPL-Rahmenwerks in allen Simulationsworkflows als generische, den Workflow einleitende Aktivität eingefügt wird. Es folgt eine Assign-Aktivität InitializeReferences, welche allen im Workflow verwendeten Referenzen auf Datenquellen und Datencontainern Anfangswerte zuweist. Diese Aktivitäten

[1] http://www.webservicex.net/globalweather.asmx?op=GetWeather

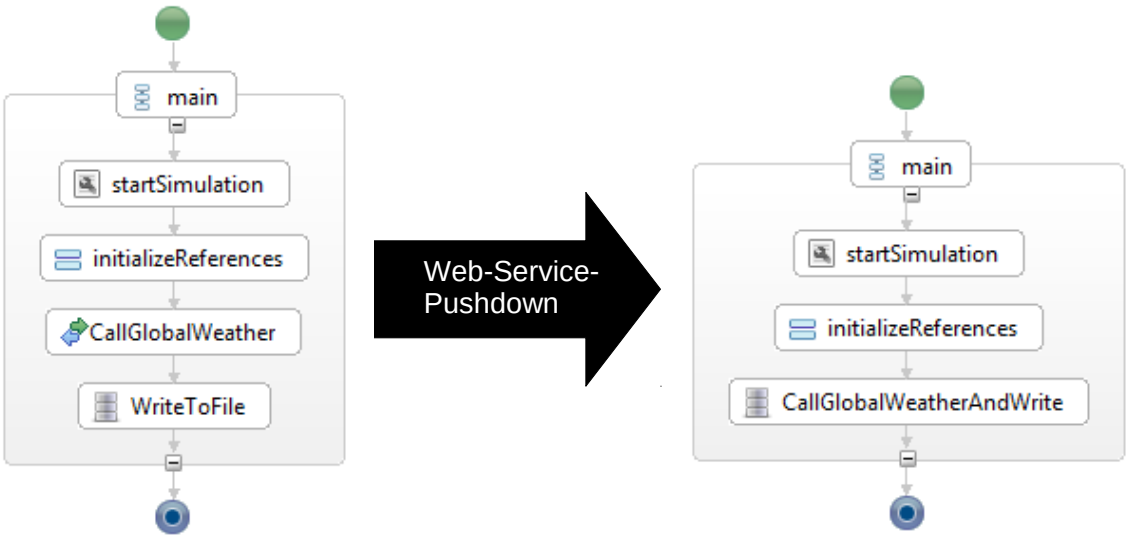


Abbildung 8.1: Anwendung der WS-Pushdown-Regel auf einen Beispielworkflow, der die Werte eines WebService-Aufrufes in eine Text-Datei schreibt

Listing 8.1 String-Wert für Variable weatherreportToWrite

```
xmlns="http://www.webserviceX.NET">
<?xml version="1.0" encoding="utf-16"?>
      <CurrentWeather>
        <Location>Milano / Linate, Italy (LIML) 45-26N 009-17E 103M</Location>
        <Time>Dec 18, 2014 - 04:50 PM EST / 2014.12.18 2150 UTC</Time>
        <Wind> from the W (260 degrees) at 2 MPH (2 KT):0</Wind>
        <Visibility> greater than 7 mile(s):0</Visibility>
        <Temperature> 42 F (6 C)</Temperature>
        <DewPoint> 42 F (6 C)</DewPoint>
        <RelativeHumidity> 100%</RelativeHumidity>
        <Pressure> 30.21 in. Hg (1023 hPa)</Pressure>
        <Status>Success</Status>
      </CurrentWeather>
```

stellen Standard-Aktivitäten dar, die in allen Anwendungsbeispielen verwendet werden und werden daher im restlichen Teil dieser Arbeit für die Betrachtung von Anwendungsbeispielen ignoriert.

Das eigentlich relevante Datenverarbeitungsmuster beginnt mit einer Invoke-Aktivität *Callglobal-weather*, welche die Operation GetWeather des Web-Services *globalweather* aufruft. Das Resultat des Web-Service-Aufrufes wird in eine Workflow-Variable *weatherreportToWrite* des Typs String gespeichert. Es handelt sich dabei um eine Auflistung von Wetterdaten für eine bestimmte Stadt. Ein Beispiel für einen Wert von *weatherreportToWrite* ist in Listing 8.1 gezeigt.

Auf die Invoke-Aktivität folgt eine IssueCommand-Aktivität *WritetoFile*, die den in Listing 8.2 gezeigten DM command auf dem zu Grunde liegenden Unix-Dateisystem ausführt. Es wird ein Unix-Shell-Command ausgeführt, der den Wert der Variable *weatherreportToWrite* lesend referenziert, in eine

Listing 8.2 DM command der IssueCommand-Aktivität WriteToFile

```
xmlstring="#weatherreportToWrite#" &&
      wsresult="$(echo"$xmlstring" | xmlstarlet sel -t -v "/CurrentWeather/text()" -)" &&
            echo "$wsresult" >> "/opt/testWSPushdown/WSResult.txt"
```

Listing 8.3 DM command der IssueCommand-Aktivität CallglobalweatherAndWrite

```
wsresult="$(curl -H "Content-Type: text/xml; charset=utf-8"
   -H "SOAPAction:http://www.webserviceX.NET/GetWeather"
   -d@soap-request.xml
    http://www.webservicex.net/globalweather.asmx |
            xmlstarlet sel -t -v "/CurrentWeather/text()" - )" &&
                  echo "$wsresult" >> "/opt/testWSPushdown/WSResult.txt"
```

Shell-Variable *xmlstring* schreibt und daraus mit einem Xpath-Ausdruck das Element CurrentWeather selektiert. Dazu wird das Tool *xmlstarlet* [Gru12] verwendet, welches das Parsen von XML-Strukturen von der Kommandozeile ermöglicht. Das Resultat wird mit Hilfe eines Echo/Befehls in die Text-Datei *WSResult.txt* auf dem lokalen Unix-Dateisystem gespeichert.

Es existiert mit der Invoke-Aktivität *Callglobalweather* eine Aktivität, die auf der Workflow-Ebene eine Web-Service- Operation aufruft. In diesem Fall besitzt die Aktivität keine Eingabeparameter und einen Ausgabeparameter *weatherreportToWrite*, in welchen der Ergebnisparameter der aufgerufenen Web-Service-Operation geschrieben wird. Basierend auf der Variable *weatherreportToWrite* besteht zwischen der Invoke-Aktivität und der IssueCommand-Aktivität eine Schreib-Lese-Datenabhängigkeit. Für die Anwendbarkeit der *Web-Service-Pushdown-Regel* muss es zudem möglich sein, auf der Datenebene, d.h. in diesem Fall im Unix-Dateisystem, eine Funktion bzw. Anweisung zu definieren, welche den Web-Service von der Datenebene aus aufruft. Diese Bedingung wird z.B. durch Verwendung des Unix-Tools *cURL* [Ste] erfüllt.

Der beschriebene Workflow erfüllt somit die Bedingungen für die Anwendung der *Web-Service-Pushdown-Regel*. Der resultierende Workflow ist in Abbildung 8.1 rechts dargestellt. Durch die Anwendung der Regel wird die Invoke-Aktivität, sowie die Variable *weatherreportToWrite* entfernt und der Web-Service-Aufruf in den DM command der IssueCommand-Aktivität verschoben, so dass eine neue IssueCommand-Aktivität *CallglobalweatherAndWrite* abgeleitet wird (Listing 8.3). Dabei wird mit dem Tool *cURL* der Web-Service auf Kommandozeilenebene aufgerufen. Der Request an den Web Service, der in Listing 8.4 gezeigt ist, ist eine SOAP-Message und enthält die Eingabeparameter *CityName* und *Country Name*. Er wird an den WebService globalWeather gesendet und anschliessend dessen Resultat direkt mit STDIN an das Tool *xmlstarlet* weitergeleitet. Dieses parst das XML-Resultat und filtert daraus das gewünschte Element CurrentWeather heraus. Das Resultat des xmlstarlet-Tools wird abschliessend in die Shell-Variable *wsresult* zugewiesen und anschliessend mit Hilfe eines Echo-Befehls in die Textdatei WSResult.txt geschrieben. Durch die Transformation wird nun der Web-Service-Aufruf auf der Datenebene ausgeführt und die Resultate direkt in *WSResult.txt* geschrieben, ohne dass eine Invoke-Aktivität auf der Workflow-Ebene notwendig ist.

Listing 8.4 SOAP-Request für WebService Globalweather

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetWeather xmlns="http://www.webserviceX.NET">
      <CityName>Milan</CityName>
      <CountryName>Italy</CountryName>
    </GetWeather>
  </soap:Body>
</soap:Envelope>
```

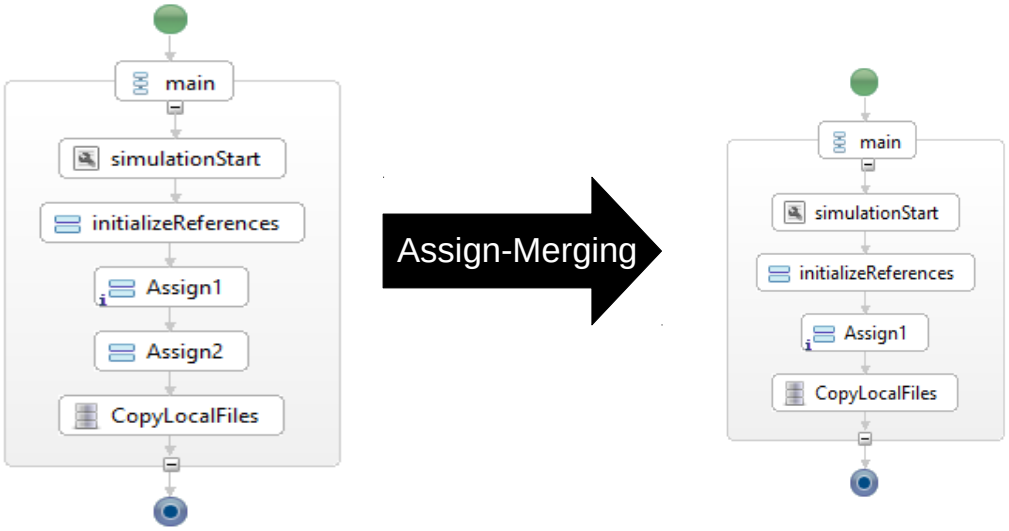


Abbildung 8.2: Anwendung der Assign-Merging-Regel auf einen Beispielworkflow

8.2 Assign-Merging

In diesem Beispiel soll die Anwendung der **Assign-Merging-Regel** veranschaulicht werden. Als Anwendungsszenario dient dabei ein BPEL-DM-Workflow, welcher auf einem Unix-Dateisystem Dateien lokal kopiert. In Abbildung 8.2 links ist der BPEL-DM-Prozess-Graph für den konventionellen, unoptimierten Workflow abgebildet.

Das relevante Datenverarbeitungsmuster beginnt mit einer Assign-Aktivität Assign1, welche einer Variable *maxvalue* einen Literalwert 10 zuweist. Eine zweite Assign-Aktivität Assign2 weist einer Variable *filenumber* den Wert von *maxvalue* zu. Eine folgende IssueCommand-Aktivität CopyLocalFiles referenziert *filenumber* lesend und führt den in Listing 8.5 gezeigten DM command aus. Alle Dateien mit Dateiname VisInputFile und einer Nummerierung von 1 bis *filenumber* werden in das Verzeichnis /opt/assgnmergetest/ kopiert.

Listing 8.5 DM command der IssueCommand-Aktivität CopyLocalFiles

```
cp /opt/100016transfertest/VisInputFile{1..#fileNumber#}.csv "/opt/assgnmergetest/"
```

Listing 8.6 DM command der IssueCommand-Aktivität CopyLocalFiles nach dem Assign-Merging

```
cp /opt/100016transfertest/VisInputFile{1..#maxvalue#}.csv "/opt/assgnmergetest/"
```

Die Bedingungen zur Anwendung der *Assign-Merging-Regel* sind hier erfüllt: Es existieren mit Assign2 und CopyLocalFiles zwei Aktivitäten, die jeweils Datenverarbeitunsoperationen ausführen, wobei Assign2 eine Variablenzuweisung durchführt und diese Variable anschliessend von CopyLocalFiles exklusiv gelesen wird. Zudem gibt es zwischen den Aktivitäten keine Inkompatibilitäten bzgl. der Befehlssprachen, da hier lediglich eine Variable eingefügt werden muss.

Durch Anwendung der Regel wird *filenumber* im DM command der IssueCommand-Aktivität direkt durch seinen Wert, in diesem Fall also *maxvalue* ersetzt (Listing 8.6). Dadurch werden Assign2 und *filenumber* überflüssig und können entfernt werden. Der resultierende Workflow ist in Abbildung 8.2 rechts dargestellt. Hinsichtlich der Laufzeit wird diese Optimierung keine grosse Wirkung haben, der Workflow wird hiermit jedoch vereinfacht und die Anwendung weiterer Regeln ermöglicht.

8.3 Delete-Delete-Merging

Als Anwendungsszenario für die **Delete-Delete-Merging-Regel** wurde ein BPEL-DM-Workflow entworfen, der in einem Verzeichnis auf einem lokalen Dateisystem zuerst alle Dateien mit Dateiendung CSV löscht und anschliessend alle Dateien mit Dateiendung TXT löscht. Das relevante Datenverarbeitungsmuster beginnt mit einer IssueCommand-Aktivität *DeleteCSVFiles*, welche einen Shell-Befehl ausführt, der im Verzeichnis auf einem lokalen Windows-Dateisystem alle Dateien mit Dateiendung csv löscht (siehe Listing 8.7). Die anschliessende IssueCommand-Aktivität*DeleteTXTFiles* wiederum löscht mit einem ähnlichen Befehl alle Dateien mit Dateiendung txt aus demselben Verzeichnis.

Die Aktivitäten DeleteFiles1 und DeleteFiles2 können durch Anwendung der *Delete-Delete-Merging-Regel* zu einer einzelnen IssueCommand-Aktivität DeleteFilesCombined kombiniert werden. Diese kombinierte Aktivität vereinigt die zuvor nacheinander ausgeführten DM commands, indem sie ihre Löschoperationen in einem einzelnen Befehl ausdrückt (Listing 8.8). DeleteFilesCombined wird aus DeleteFiles2 abgeleitet. DeleteFiles1 hingegen ist nicht mehr erforderlich und kann aus dem Workflow entfernt werden. Durch die Transformation wurde der Workflow vereinfacht, indem die Anzahl der Aktivitäten reduziert wurde. Die hiermit erzielten Laufzeitverbesserungen sind in Kapitel 10 zu finden.

8.4 Eliminate-Temporary-Container-Regel

In diesem Beispiel soll die Anwendung der **Eliminate-Temporary-Container-Regel** gezeigt werden. Als Anwendungsszenario dient dabei ein Workflow, bei dem Dateien von einem Linux-Rechner

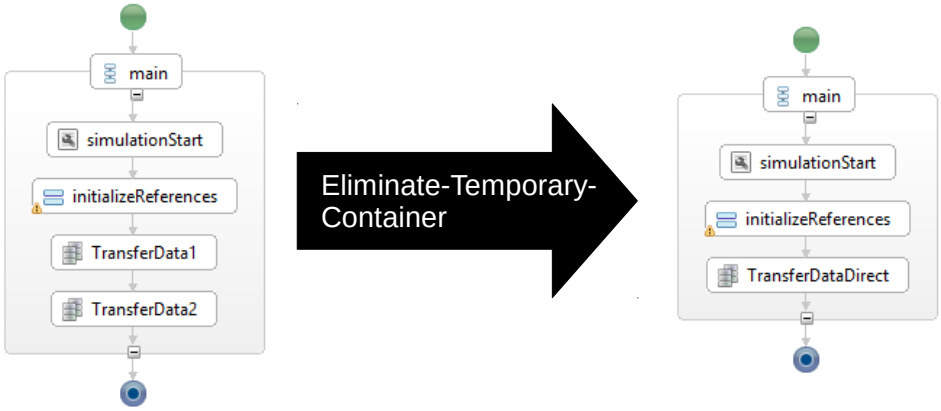


Abbildung 8.3: Anwendung der Eliminate-Temporary-Container-Regel auf einen Beispielworkflow

auf einen Windows-Rechner und anschliessend von dort wieder auf einen anderen Linux-Rechner kopiert werden, d.h. das Windows-Verzeichnis wird als Zwischenspeicher verwendet. In Abbildung 8.2 links ist der BPEL-DM-Prozess-Graph für den unoptimierten Workflow abgebildet. Das relevante Datenverarbeitungsmuster beginnt mit einer TransferData-Aktivität TransferData1, welche Dateien von einem Linux-Rechner in ein Verzeichnis auf einem Windows-Rechner kopiert. Dabei wird als DM command der TransferData-Aktivität der Name des Quellverzeichnisses verwendet. Alternativ könnte hier als DM command aber auch eine *data container reference variable* verwendet werden. Das Zielverzeichnis wird über eine *data container reference variable tempdir* referenziert. Eine anschliessende TransferData-Aktivität TransferData2 kopiert die Dateien auf einen weiteren Linux-Rechner indem es in ihrem DM-command *tempdir* lesend referenziert.

Die Bedingungen zur Anwendung der *Eliminate-Container-Regel* sind hier erfüllt: Es existiert mit TransferData1 eine Aktivität, welche einen temporären Container *tempdir* mit Dateien füllt, welcher anschliessend von TransferData2 lesend referenziert wird. Somit besteht zwischen TransferData1 und TransferData2 basierend auf *tempdir* eine exklusive Schreib-Lese-Bedingung.

Durch Anwendung der Regel wird *tempdir* im DM command von TransferData2 direkt durch seine Berechnung ersetzt. Da in diesem Fall keine Berechnung stattfindet, werden hier lediglich die zu transferierenden Dateien referenziert. In diesem Fall wird also der im DM command von TransferData1 gegebene Verzeichnisname verwendet. Aus TransferData2 wird somit eine neue Aktivität TransferDataDirect abgeleitet. TransferData1, die *data source reference variable*, welche den Windows-Rechner referenziert und *tempdir* werden überflüssig und können entfernt werden. Der resultierende Workflow ist in Abbildung 8.3 rechts dargestellt.

8.5 Element-To-Set-Regel

In folgendem Beispiel soll die Anwendung der **Element-To-Set-Regel** veranschaulicht werden. Als Anwendungsszenario dient dabei ein BPEL-DM-Workflow, der Dateien aus einem Ordner auf einem lokalen Windows-Dateisystem in einen Ordner auf einem entfernten Unix-Dateisystem transferiert.

Listing 8.7 DM command der IssueCommand-Aktivität DeleteCSVFiles

```
del E:\visSavas2\folder\*.csv
```

Listing 8.8 DM command der IssueCommand-Aktivität DeleteFilesCombined

```
del E:\visSavas2\folder\*.csv ; E:\visSavas2\folder\*.txt
```

Alle zu transferierenden Dateien besitzen denselben Präfix in ihren Dateinamen. An diesem Präfix ist jeweils eine Zahl als Nummerierung angehängt ist, d.h. alle Dateien sind durchnummeriert. Im Beispiel unten werden 100 Dateien mit dem Präfix VisInputFile und Nummerierung von 0 bis 99 transferiert.

In Abbildung 8.4 links ist der BPEL-DM-Prozess-Graph für den unoptimierten Workflow abgebildet. Der gezeigte Workflow erfüllt alle Voraussetzungen des **Bedingungsteils** der *Element-To-Set-Regel* und ermöglicht somit ihre Anwendung. Es fehlt eine optionale, datenbereitstellende Aktivität, da die Dateien nicht in den Workflow-Kontext geladen werden. Das eigentlich relevante Datenverarbeitungsmuster beginnt mit einem Schleifenkonstrukt, das in WS-BPEL durch eine Foreach-Aktivität repräsentiert wird. Es enthält als Schleifenrumpf eine Assign-Aktivität Assign und eine TransferData-Aktivität. Eine umschliessende Sequence-Aktivität sorgt dafür, dass diese beiden Aktivitäten nacheinander

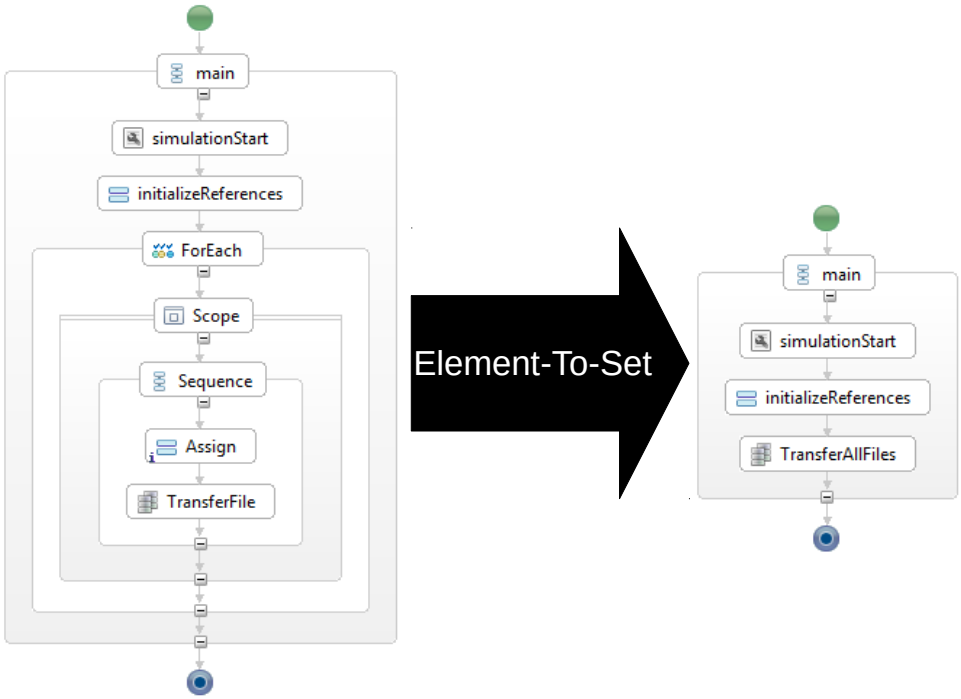


Abbildung 8.4: Anwendung der Element-To-Set-Regel auf einen Beispielworkflow zur schleifenbasierten Übertragung von Dateien

Listing 8.9 DM command der TransferData-Regel nach Anwendung der Element-To-Set-Regel

```
E:\vis\in\
```

ausgeführt werden. Umschlossen werden alle diese Aktivitäten wiederum von einer Scope-Aktivität, die den Schleifenrumpf einleitet. Die ForEach-Aktivität enthält eine BPEL-Variable *Counter*, in der der Index des aktuellen Schleifendurchlaufs gespeichert wird. Beginnend bei 0 als Startwert, wird der Wert der Variable *Counter* in jedem Schleifendurchlauf um eins erhöht, bis der Endwert 99 erreicht ist. In jedem Schleifendurchlauf wird jeweils eine Datei selektiert und transferiert.

Die Assign-Aktivität stellt die *Iterator-Aktivität* dar. Sie kopiert bei jedem Iterationsschritt den aktuellen Wert von *Counter* in die BPEL-Variable *filenumber*, sodass im *i*-ten Iterationsschritt jeweils die *i*-te Datei im Ordner referenziert werden kann. Es folgt eine TransferData-Aktivität, welche mit ihrer DM command den eigentlichen, datenverarbeitenden Befehl ausführt. Dieser besteht aus dem absoluten Pfad-Namen der jeweiligen Datei, d.h. dem Namen des Verzeichnisses, in dem die Datei liegt und dem eigentlichen Dateinamen. Dieser wiederum hat die Form *VisInputFile#fileNumber#.csv*, d.h. er besteht aus dem String VisInputFile, an dessen Ende die Variable *filenumber* angehängt ist. Da die Transfer-Data-Aktivität der einzige Leser der Variablen *filenumber* ist, existiert zwischen der Transfer-Data-Aktivität und der Assign-Aktivität der For-Schleife eine exklusive Schreib-Lese-Datenabhängigkeit.

Der **Aktionsteil** der *Element-To-Set-Regel* sieht es vor, einen datensatz- bzw. dateibasierten Befehl durch einen einzelnen, **mengenbasierten** Befehl zu ersetzen, der alle zuvor von der Schleife iterierten Daten auf einmal erfasst. Dies setzt voraus, dass zusätzlich zu den oben erfüllten Voraussetzungen ein solcher mengenbasierter Befehl gefunden werden kann. Durch Verwendung des Pfadnamens des relevanten Verzeichnisses, der alle relevanten Dateien enthält als DM command der TransferData-Aktivität führt der transformierte Workflow dieselbe Funktion aus wie der ursprüngliche Workflow, wenn sich im relevanten Verzeichnis und allen enthaltenen Unterordnern ausschliesslich Dateien mit Dateiname VisInputFile und Nummerierung von 0 bis 99 befinden. Alternativ könnte statt dem konkreten Pfadnamen auch eine *data container reference variable* verwendet, werden, die vom SIMPL Core auf das genannte Verzeichnis aufgelöst wird.

Abbildung 8.4 zeigt rechts den resultierenden Workflow nach Anwendung der Regel. Der Kontrollfluss im Workflow wurde wie folgt angepasst: Aus der Regelanwendung geht eine einzelne TransferData-Aktivität hervor, welcher einen mengenbasierten DM command ausführt, der alle Dateien auf einmal erfasst. Dabei handelt es sich um den Pfadnamen des Verzeichnisses, in dem alle zu transferierenden Dateien liegen(siehe Listing 8.9). Die ForEach-Aktivität, die Assign-Aktivität und die ursprüngliche TransferData-Aktivität werden entfernt. Anschliessend wird der direkte Vorgänger der ForEach-Aktivität mit der neuen TransferData-Aktivität verbunden. Des Weiteren wird die Variable *Counter* entfernt, da sie von der neuen TransferData-Aktivität nicht referenziert wird.

Die Semantik des neuen Workflows stimmt jedoch nicht in allen möglichen Fällen mit dem ursprünglichen überein, da bei Verwendung des Verzeichnisnamens *alle* Dateien in diesem Verzeichnis übertragen werden, selbst wenn diese nicht einen Dateinamen VisInputFile mit Nummerierung von 0 bis 99 haben.

Listing 8.10 DM command einer IssueCommand-Aktivität nach Anwendung der Element-To-Set-Regel

```
FOR %G IN (E:\vis\in\VisInputFile*.csv) DO copy %G E:\vis\out
```

Listing 8.11 DM command einer IssueCommand-Aktivität nach Anwendung der Element-To-Set-Regel mit festgelegter Anzahl von Schleifendurchläufen

```
FOR /L %G IN (1 1 100) DO COPY E:\vis\in\VisInputFile%G.csv E:\vis\out
```

Die TransferData-Aktivität akzeptiert als DM command entweder einen Dateipfad oder eine *data container reference variable*, die wiederum einen Dateipfad beschreibt . Dadurch können jedoch keine komplexeren Befehle verwendet werden, wie z.B. die Definition einer Art regulärer Ausdrücke, mit denen Dateien selektiert werden, deren Dateinamen einem gewissen Muster entsprechen(z.B. durch Verwendung des Asterisk-Symbols). Mit Hilfe des Tools *find* könnte z.B. eine Befehl *dir /B | find /V "VisInputFile"* definiert werden, der alle Dateien aussucht, die im Dateinamen den String VisInputFile enthalten. Diese Einschränkungen der TransferData-Aktivität können umgangen werden, wenn stattdessen eine **IssueCommand-Aktiviät** verwendet wird. Das Spektrum der verwendbaren Shell-Befehle ist hier bedeutend größer, genaugenommen können damit alle Befehle abgeschickt werden, die das zu Grunde liegende Betriebssystem unterstützt. Beispielsweise könnte für diesen Fall ein *For*-Befehl verwendet werden, der alle Dateien der Form *VisInPutFile*.csv* in das Zielverzeichnis kopiert (siehe Listing 8.10). Zudem kann in einem solchen For-Befehl die Anzahl der Schleifendurchläufe festegelegt werden, was der Semantik des ursprünglichen Workflows gleichkommen würde (siehe Listing 8.11). Hier wird eine Variable *G* als Laufparameter der Schleife festgelegt und beginnend mit dem Startwert 0 in jedem Iterationsschritt um eins erhöht, bis sie den Endwert 99 erreicht.

Sollen die Dateien wie beim Original-Workflow auf einen entfernten Rechner übertragen werden, muss eine separate **Authentifikation** erfolgen. Hier kann beispielsweise das Tool *sshpass* für einen Dateitransfer unter Verwendung der SSH-Schnittstelle, sowie der Command-Line-Befehl *scp* (Secure Copy Protocol) verwendet werden. Die Besonderheit an *sshpass* liegt darin, dass die Authentifizierung in einem nicht interaktiven Modus erfolgt. Somit können die Daten mit einer einzelnen IssueCommand-Aktivität übertragen werden (Listing 8.12). Voraussetzung für die Einsetzbarkeit von *sshpass* ist, dass es auf allen beteiligten Rechnern installiert sein muss. Auf Windows-Rechnern muss hierfür zusätzlich eine Unix-SSH-Umgebung wie z.B. *Cygwin* installiert sein.

Alternativ kann statt dem Kommandozeilen-Befehl *scp* der Befehl *pscp* verwendet werden. Hierzu muss das Tool *Putty* und anstelle einer Passwortauthentifikation eine Authentifikation mit Paaren von *privaten und öffentlichen Schlüsseln* zum Einsatz kommen. Auf dem lokalen System wird mit einem Key-Generator jeweils ein privater und ein öffentlicher Schlüssel erzeugt. Während der private

Listing 8.12 DM command der IssueCommand-Aktivität unter Verwendung des sshpass Tools nach Anwendung der Element-To-Set-Regel

```
FOR /L %G IN (1 1 100)
      C:\cygwin\bin\sshpass.exe -p "openstack" SCP E:\vis\in\VisInputFile%G.csv
          openstack@192.168.209.210:/opt/transfertest/
```

Listing 8.13 DM command der IssueCommand-Aktivität Element-To-Set-Regel unter Verwendung des Putty Tools und dem Befehl PSCP

```
FOR /L %G IN (1 1 100) DO PSCP E:\vis\in\VisInputFile%G.csv
    openstack@192.168.209.210:/opt/transfertest/
```

Listing 8.14 XPath-Befehl zur Selektion der aktuellen Datencontainer-Referenz aus containerRefList

```
$containerRefList//GenericContainerReference[number($contNumber)]/*
```

Schlüssel auf dem lokalen Rechner gespeichert wird, muss der öffentliche Schlüssel auf alle Rechner kopiert werden, auf die Dateien übertragen werden soll. Nach diesen Schritten ist eine passwortlose, nicht-interaktive Authentifikation zwischen den beteiligten Rechnern möglich. Ein möglicher Befehl für dieses Fallbeispiel ist in Listing 8.13 gezeigt. Diese Methode ist im Hinblick auf Sicherheitsaspekte der obigen passwortbasierten Authentifikation vorzuziehen, da keine Passwörter über Netzwerke übertragen werden.

8.6 Predicate-Pushdown-Regel

In folgendem Beispiel wird die Anwendung der **Predicate-Pushdown-Regel** gezeigt, wobei die verschiedenen in Abschnitt 7.2.4 beschriebenen *Varianten* der Regel veranschaulicht werden. Als Anwendungsszenario dient dabei ein BPEL-DM-Workflow, der eine *data container reference list* lädt und für jede einzelne *data container reference variable* überprüft, ob der Dateiname der entsprechenden Datei im Datencontainer einem vorgegebenen Namen entspricht. Falls der Test positiv ausfällt, wird die entsprechende Datei auf einen entfernten Unix-Rechner kopiert(die gleichnamige, bereits existierende Datei wird überschrieben).

In Abbildung 8.5 ist der BPEL-DM-Prozess-Graph für den unoptimierten Workflow abgebildet. Eine Assign-Aktivität InitContainerList simuliert das Laden der *data container reference list*, indem es einer Variable *containerRefList* des Typs *tdataContainerReferenceList* einen Initialwert zuweist. Dabei handelt es sich um eine Liste von *GenericContainerReferences*, welche als Abstraktion für *data container reference variables* dienen.

Beim unoptimierten Workflow wird ein Schleifenkonstrukt verwendet, um die Aufgabe des Datentransfers zu erfüllen. Dazu wird durch eine Assign-Aktivität getCounter zunächst der aktuelle Schleifenindex in der Variable *contNumber* gespeichert. Mit dieser wird im *i*-ten Schleifendurchlauf die *i*-te *data container reference variable* aus *containerRefList* referenziert und ihr Wert einer Variable *currentcontainer* des Typs *FileSystemDataContainerReferenceType* zugewiesen (siehe Listing 8.14). Eine If-Aktivität IfFileNameEqVisInputFile3 prüft anschliessend mit einem XPath-Ausdruck, ob der Name der in *currentcontainer* enthaltenen Datei dem String VisInputFile3 entspricht (Listing 8.15). Ist dies der Fall, überträgt die folgende TransferData-Aktivität die Datei auf den entfernten Unix-Rechner. Andernfalls erfolgt kein Dateitransfer. Dies wird durch eine Empty-Aktivität im Else-Zweig der If-Aktivität dargestellt. Der DM command von TransferData besteht dabei lediglich aus der *data container reference variable* currenContainer.

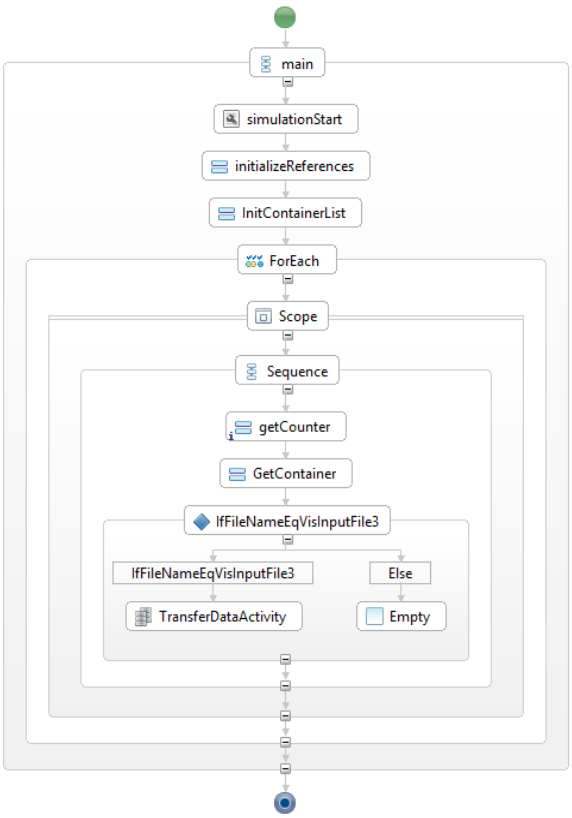


Abbildung 8.5: unoptimierter Beispielworkflow zur schleifenbasierten Übertragung von Datencontainer-Inhalten nach Auswertung einer Filter-Operation

Listing 8.15 Bedingungsanweisung der If-Aktivität IfFileNameEqVisInputFile3

```
boolean(string($currentContainer//file/text()) = 'VisInputFile3.csv')
```

Das Workflow-Fragment im Rumpf der For-Schleife erfüllt die Voraussetzungen für eine Anwendung der **Predicate-Pushdown-Regel**. Mit der If-Aktivität wird eine Art Filterung der Dateien in *containerRefList* durchgeführt, indem nur Dateien jener Datencontainer übertragen, welche die Bedingungsanweisung der If-Aktivität erfüllen. Da die Filteroperation im Rumpf einer Schleife erfolgt, muss eine der *Varianten der Predicate-Pushdown-Regeln* zum Einsatz kommen.

Aufgrund der Existenz einer datenbereitstellenden Aktivität bietet sich hier die **1. Variante der Pushdown-Regel** an, welche die Filteroperation, die erst spät im Schleifenrumpf erfolgt, in InitContainerList verschieben würde. Dadurch würde die *containerRefList* von vornherein mit den korrekten *data container reference variables* befüllt werden. Hierzu müsste ein XPath-Ausdruck gefunden werden, der alle *data container reference variables* in *containerRefList* selektiert, die eine Datei mit dem

Listing 8.16 xsl-Skript, das alle Knoten entfernt, welche sich nicht zur Weiterverarbeitung qualifizieren

```
<xsl:template match="@*|node()">
   <xsl:copy>
     <xsl:apply-templates select="@*|node()"/>
   </xsl:copy>
  </xsl:template>
  <xsl:template match="GenericContainerReference[WindowsLocalDataContainerReference/file !=
      'VisInputFile3.csv']"/>
</xsl:stylesheet>
```

Listing 8.17 copy-Anweisung der Assign-Aktivität InitCorrectContainerList

```
<bpel:copy>
  <bpel:from>
   <![CDATA[bpel:doXslTransform('ConrefListTransform.xsl', $containerRefList)]]>
   </bpel:from>
   <bpel:to variable="containerRefList"></bpel:to>
 </bpel:copy>
```

Dateinamen VisInputFile3.csv enthalten. Zudem muss das Ergebnis dieses XPath-Ausdrucks erneut der Variable *containerRefList* zugewiesen werden.

Es konnte jedoch keine XPath-Anweisung gefunden werden, die dies bewerkstelligen kann. Weder XPath 1.0, noch XPath 2.0 verfügen über die Fähigkeit, bestehende XML-Dokumente zu **manipulieren**, z.B. durch Lösch-, Aktualisierungs- und Einfügeoperationen. Als Workaround kann hier ein **XSLT (Extensible Stylesheet Language Transformations)**-Skript zum Einsatz kommen ([C$^+$07]). Mit Hilfe von XSLT kann ein XML-Dokument z.B. derart umgeformt werden, dass nur die Knoten übrig bleiben, die eine gewisse Bedingung erfüllen oder es können gezielt Knoten hinzugefügt, gelöscht oder aktualisiert werden. Die BPEL-Spezifikation ermöglicht das Einbinden von XSL-Transformationen in einen BPEL-Prozess [2]. Hierfür muss ein XSL-Skript definiert werden, das die Manipulationsoperationen enthält. Anschliessend wird innerhalb einer Assign-Aktivität die Funktion *doXslTransform* aufgerufen, welche das zuvor definierte XSL-Skript auf einem als Parameter angegebenen XML-Dokument ausführt.

In diesem Fall wird hierzu aus der ursprünglichen Assign-Aktivität eine neue Assign-Aktivität *InitCorrectContainerList* abgeleitet. Diese führt ein XSL-Skript aus, welche zwei *template*-Anweisungen anwendet (siehe Listing 8.16). Die erste Anweisung selektiert den Inhalt aller Knoten der Knotenmenge und die zweite Anweisung sucht alle Knoten aus, deren Element *file* nicht VisInputFile3.csv entspricht. Da der body der zweiten template-Anweisung bewusst leer gelassen ist, werden diese Knoten im Endergebnis nicht berücksichtigt.

Listing 8.17 wiederum zeigt die *copy*-Anweisung von *InitCorrectContainerList*, welche *conRefList* die bereits manipulierte Liste als Wert zuweist. Abbildung 8.6 zeigt den Workflow nach Anwendung der *ersten Variante der Predicate-Pushdown-Regel*. Der Kontroll- und Datenfluss im Workflow wurde wie

[2] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

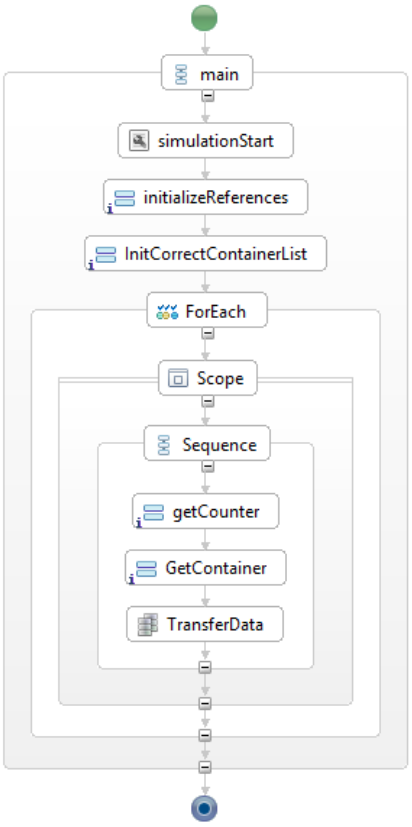


Abbildung 8.6: Der resultierende Workflow nach Anwendung der ersten Variante der Predicate-Pushdown-Regel: Verschieben der Filteroperation in datenbereitstellende Aktivität vor der Schleifenausführung

folgt angepasst. Die If-Aktivität, sowie die Empty-Aktivität wurden entfernt. Die Aktivität TransferData bleibt unverändert, ist aber nun direkter Nachfolger von getContainer. Aus InitContainerList wurde eine neue Assign-Aktivität *InitCorrectContainerList* abgeleitet. Die Filterung der Daten, in diesem Fall die Auswahl der korrekten Container wurde von der If-Aktivität in die Assign-Aktivität *InitCorrectContainerList* verschoben. Da diese Methode mit der XSL-Transformation nicht fehlerlos zum Laufen gebracht werden konnte, werden hier weitere Vorgehensweisen beschrieben, von denen zwei implementiert wurden.

Wenn die Liste der *data container reference variables* nicht durch ein Assign-Aktivität initialisiert wird, sondern auf einer XML-Datenbank abgelegt ist, muss die Liste mit Hilfe einer RetrieveData-Aktivität in den Workflow-Kontext geladen werden. Da der SIMPL-Prototyp für eine RetrieveData-Aktivität mit einer XML-Datenbank als zu Grunde liegender Datenressource automatisch XQuery als Befehlssprache festlegt, kann durch einen XQuery-Ausdruck beim Laden der Daten gleichzeitig auf einfache Weise die Filterung ausgeführt werden. Eine mögliche XQuery-Anfrage ist in Listing 8.18 gezeigt.

Die **2. Variante der Predicate-Pushdown-Regel** kann eingesetzt werden, wenn die Filteroperation, wie bei der ersten Variante, im Rumpf einer Schleife ausgeführt wird. Die Aktivität, die die

Listing 8.18 XQuery-Anfrage für RetrieveData-Aktivität, bei der die Selektion bereits beim Laden stattfindet

```
$containerRefList//GenericContainerReference[//file= 'VisInputFile3.csv']
```

Listing 8.19 DM command der IssueCommand-Aktivität nach Anwendung der Option 1 der zweiten Variante der Predicate-Pushdown-Regel

```
IF #dateiname# == VisInputFile3.csv COPY E:\vis\in\#dateiname# E:\vis\out
```

Filteroperation ausführt, ist dabei in einer Abhängigkeitsbeziehung mit der nachfolgenden und/oder vorhergehenden Aktivität im Schleifenrumpf. In diesem Fallbeispiel ist TransferData abhängig von der Filter-Operation der If-Aktivität. Die Filter-Operation wiederum ist abhängig von den Variablen, deren Werte von getContainer geschrieben werden.

In ersten Fall(**Option 1**) müsste die Bedingungsanweisung der If-Aktivität in den DM command der TransferData-Aktivität verschoben werden. Eine Filteroperation kann in einer TransferData-Aktivität nicht ausgeführt werden, da diese nur *data container reference variables* oder Pfadnamen als Befehle unterstützt. Bei Nutzung einer *IssueCommand*-Aktivität können im Gegensatz zur TransferData-Aktivität IF-Then-Else-Konstrukte verwendet werden(siehe Listing 8.19).

Die zweite Mögichkeit (**Option 2**) wäre das Verschieben der Bedingungsanweisung in die Assign-Aktivität getContainer. Hierfür musste eine XPath-Ausdruck gefunden werden, der ein If-Then-Else realisiert (siehe Listing 8.22). Die Realisierung des If-Then-Else-Konstrukts ist in **XPath 2.0** möglich, da Apache ODE offiziell die Sprache XPath 2.0 unterstützt und diese ein vordefiniertes If-Then-Else-Konstrukt anbietet. Dies erfordert, dass im Wurzel-Element des BPEL-Prozesses die in Listing 8.20 gezeigten Informationen eingefügt werden. Listing 8.21 zeigt eine möglichen copy-Anweisung für die Umsetzung des IF-Konstrukts in der Assign-Aktivität GetContainer. In jedem Iterationsschritt wird *currentContainer* mit einem leeren Container initialisiert. Nur bei positiver Auswertung der If-Bedingung wird dieser Initialwert von *currentContainer* mit dem Wert des aktuell referenzierten Datencontainers überschrieben, d.h. in TransferData wird anschliessend der Inhalt eines leeren Containers übertragen

Neben der XPath 2.0-Lösung, wird im folgenden Abschnitt ebenso die XPath 1.0-Lösung vorgestellt, da BPEL-Workflows immer noch standardmässig mit XPath 1.0 arbeiten und ein problemloses Funktionieren aller XPath 2.0-Ausdrücke nicht gewährleistet ist. Ferner soll dadurch aufgezeigt werden, dass die erfolgeiche Umsetzung von Optimierungsregeln sehr eng mit den Eigenheiten der zu Grunde liegenden Datenressourcen und einsetzbaren Befehlssprachen verknüpft ist.

Da **XPath 1.0** keinen vordefinierten IF-Then-Else-Ausdruck unterstützt, wird ein Workaround verwendet, der die String-Funktion *concat* verwendet. Dabei wird die Tatsache genutzt, dass die Funktion *number()* auf *true()* angewendet den Wert 1 liefert bzw. auf *false()* angewendet den Wert 0 liefert.

Listing 8.20 Erweiterung des BPEL-Prozesses um XPath2.0

```
queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
```

Listing 8.21 Befehl der Assign-Aktivität getContainer nach Anwendung der Option 2 der zweiten Variante der Predicate-Pushdown-Regel

```
<bpel:copy>
    <bpel:from>
     <![CDATA[
       if (string($containerRefList//GenericContainerReference[number($Iterator)]//file/text()) =
            "VisInputFile3.csv")
                  then $containerRefList//GenericContainerReference[number($Iterator)]/*
                  else $currentContainer
     </bpel:from>
   <bpel:to variable="currentContainer"></bpel:to>
</bpel:copy>
```

Zudem kommt die *Division durch 0* zum Einsatz. Bei Division von 1 durch 0 wird als Ergebnis der String *Infinity* zurückgeliefert. Falls die Prüfung des Dateinamens auf Gleichheit mit *VisInputFile3.csv* zu *false()* ausgewertet wird, ist *("1", Infinity)* das erste Argument der *concat*-Funktion. Dieses Argument wiederum entspricht einem leeren String, da der String „1“ eine endliche Länge hat. Falls die Bedingung zu *true()* ausgewertet wird, ist das erste Argument der concat-Funktion substring("1", 1) und das Ergebnis ist der String„1“. Abhängig von der Auswertung des Bedingungsausruckes erhält die Funktion *concat* also entweder einen nichtleeren String oder einen leeren String als Argument. Insgesamt erzeugt der Xpath-Ausdruck den String 1 wenn die Bedingung zu *true()* ausgewertet wird, andernfalls, d.h. wenn die Negation der Bedingung zu *true()* ausgewertet wird, wird der leere String „“ erzeugt.

Der resultierende String wird durch Anwendung der Funktionen *boolean()* und *number()* in die Zahl 1 oder die Zahl 0 umgewandelt. Die Funktion *boolean()* liefert bei einem leeren String *false()*, bei jedem anderen String mit Länge grösser 0 *true()* zurück. Durch die Anwendung der Funktion *number()* wiederum wird der Wert 1 bzw der Wert 0 erzeugt. Anschliessend wird dieser Wert mit dem aktuellen Schleifeindex *Iterator* multipliziert, sodass entweder der Schleifenindex bei der Referenzierung der entsprechenden *data container reference variable* verwendet wird oder der Wert 0. Da die Referenzierung von Knoten in XML-Dokumenten durch XPath bei 1 beginnt und der Schleifenindex bei 0 anfängt, wird der Wert noch um eins erhöht. Der referenzierte Datencontainer wird der Variable *currentcontainer* zugewiesen. In jedem Schleifendurchlauf wird der Inhalt von *currentcontainer* transferiert, d.h. es wird entweder keine Datei übertragen (der erste Container in *containerRefList*) oder die Datei im aktuell referenzierten Datencontainer. Deshalb funktioniert diese Methode nur, wenn in der zu übertragenen *data container reference list* als erste Containerreferenz eine Referenz auf einen leeren Container enthalten ist bzw. nachträglich eingefügt werden kann.

Zu beachten ist, dass das beschriebene XPath 1.0-If-Konstrukt nur Strings als Ausgabewert zurückliefert. Das Verfahren ist dadurch sehr eingeschränkt einsetzbar. Es wäre z.B nicht möglich, im If-Zweig (erster substring) oder im Else-Zweig (zweiter substring) des Konstrukts direkt Kommandos zu definieren, die bei entsprechender Auswertung ausgeführt werden können. Die Anweisungen würden als Strings und nicht als ausführbare Befehle interpretiert. Nur über den Umweg über Strings und Stringvariablen können Befehle dann z.B. in einer IssueCommand-Aktivität ausgeführt werden.

Der resultierende Workflow nach Anwendung der *Predicate-Pushdown-Regel* sieht bei beiden Lösungen gleich aus (Abbildung 8.7). Die If-Aktivität, sowie die Empty-Aktivität wurden entfernt. Die Filterung

Listing 8.22 XPath-Ausdruck von getContainer nach Anwendung der Predicate-Pushdown-Regel

```
$containerRefList//GenericContainerReference[round(number(
($iterator * number(boolean
      (concat(
      substring('1', number(
            1 div number(boolean($containerRefList//GenericContainerReference[number($Iterator +
                1)]//file/text() ="VisInputFile3.csv")))),
      substring('', number(
            1 div
                number(not(boolean($containerRefList//GenericContainerReference[number($Iterator
                + 1)]//file/text() ="VisInputFile3.csv"))))))))
 + 1 ))]/*
```

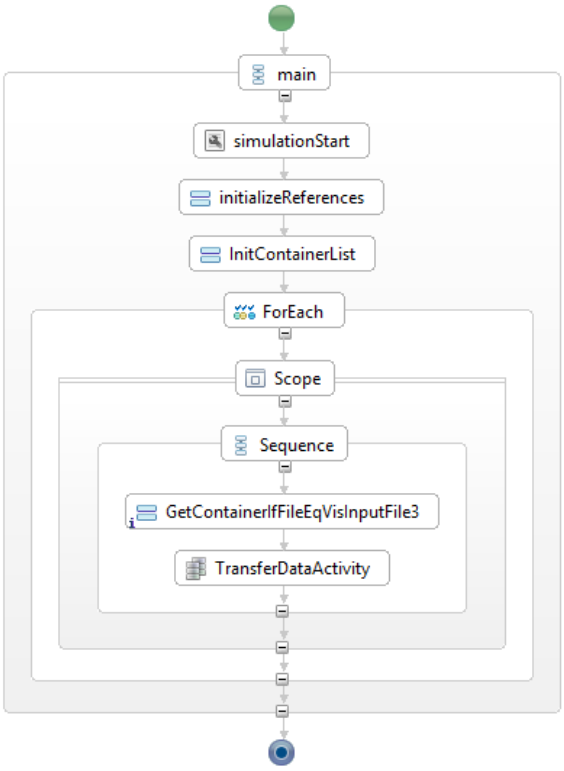


Abbildung 8.7: Workflow nach Anwendung der Predicate-Pushdown-Regel

Listing 8.23 Bedingungsanweisung der If-Aktivität IfFileNumberLss5OrGtr15

```
$fileNumber < 5 or $fileNumber > 15
```

der Daten, in diesem Fall die Auswahl der korrekten Container, wurde von der ursprünglichen Aktivität in die Assign-Aktivität GetContainerIfFileEqVisInputFile3 verschoben. Aus TransferData wurde eine neues TransferData abgeleitet, welches direkter Nachfolger von GetContainerIfFileEqVisInputFile3 ist. Durch diese Transformationen können keine grossen Laufzeitverbesserungen erzielt werden, da weiterhin alle *data container reference variables* der ursprünglichen *data container reference list* durchlaufen werden. Es wurde jedoch die Anzahl der Aktivitäten im Workflow verringert und die eventuelle Anwendung weiterer Optimierungsregeln ermöglicht.

Obwohl sowohl mit der XPath 1.0-, als auch mit der XPath 2.0-Lösung der Predicate Pushdown erreicht werden kann, ist die XPath-Version die benutzerfreundlichere Version, da sie aufgrund der typischen If-Then-Else-Struktur intuitiv verständlich ist.

8.7 Predicate-Pushdown + Element-To-Set

In diesem Abschnitt soll die kombinierte Anwendung der **Predicate-Pushdown-** mit der **Element-To-Set-Regel** veranschaulicht werden. Der folgende BPEL-DM-Workflow iteriert über die Dateien in einem lokalen Ordner in einem Windows-Dateisystem. Alle Dateien im relevanten Ordner haben einen Dateinamen der Form *VisInputFile + Nummerierung* und es befinden sich insgesamt 20 Dateien mit Nummerierung 0 bis 19 im Ordner. Nach Ausführung des Workflows sollen alle Dateien mit einer Nummerierung kleiner als 5 und grösser als 15 aus dem Verzeichnis gelöscht werden.

In Abbildung 8.8 ist der BPEL-DM-Prozess-Graph für den unoptimierten Workflow abgebildet. Eine For-Each-Aktivität repräsentiert das Schleifenkonstrukt, welches über die Dateien iteriert und im Schleifenrumpf eine Sequence-Aktivität enthält. Diese wiederum umschliesst eine Assign-Aktivität GetFilenumber und eine If-Aktivität IfFileNumberLss5OrGtr15. GetFilenumber weist einer Workflow-Variable *filenumber* den aktuellen Wert des Schleifenindexes zu. In der If-Aktivität wird mit Hilfe eines einfachen XPath-Ausdruckes überprüft, ob der aktuelle Schleifenindex kleiner als 5 oder grösser als 15 ist (siehe Listing 8.23). Ist dies der Fall, wird die Datei mit der entsprechenden Nummerierung über einen IssueCommand-Befehl im If-Zweig der If-Aktivität aus dem Ordner gelöscht. Ansonsten erfolgt keinerlei Aktion. Dies wird durch eine Empty-Aktivität im Else-Zweig der If-Aktivität dargestellt.

Als mögliche Ansatzpunkte für die Anwendung von Optimierungsregeln bietet sich die *Element-To-Set-Regel* zur Entfernung des Schleifenkonstrukts sowie die *Predicate-Pushdown-Regel* zur Entfernung des If-Konstrukts an. Dabei muss die *Predicate-Pushdown-Regel* vor der *Element-To-Set-Regel* verwendet werden, so dass der Schleifenrumpf vereinfacht und der Einsatz der *Element-To-Set-Regel* ermöglicht wird. Das Workflow-Fragment im Rumpf der For-Schleife erfüllt alle Voraussetzungen, die für die Anwendung der *Predicate-Pushdown-Regel* erfüllt sein müssen.

Mit der If-Aktivität wird eine Filterung der Dateien im lokalen Windows-Ordner durchgeführt. Da die Filteroperation in einer If-Aktivität im Rumpf einer Schleife ausgeführt wird, muss eine der *Varianten der Predicate-Pushdown-Regeln* zum Einsatz kommen. Da keine datenbereitstellende Aktivität existiert,

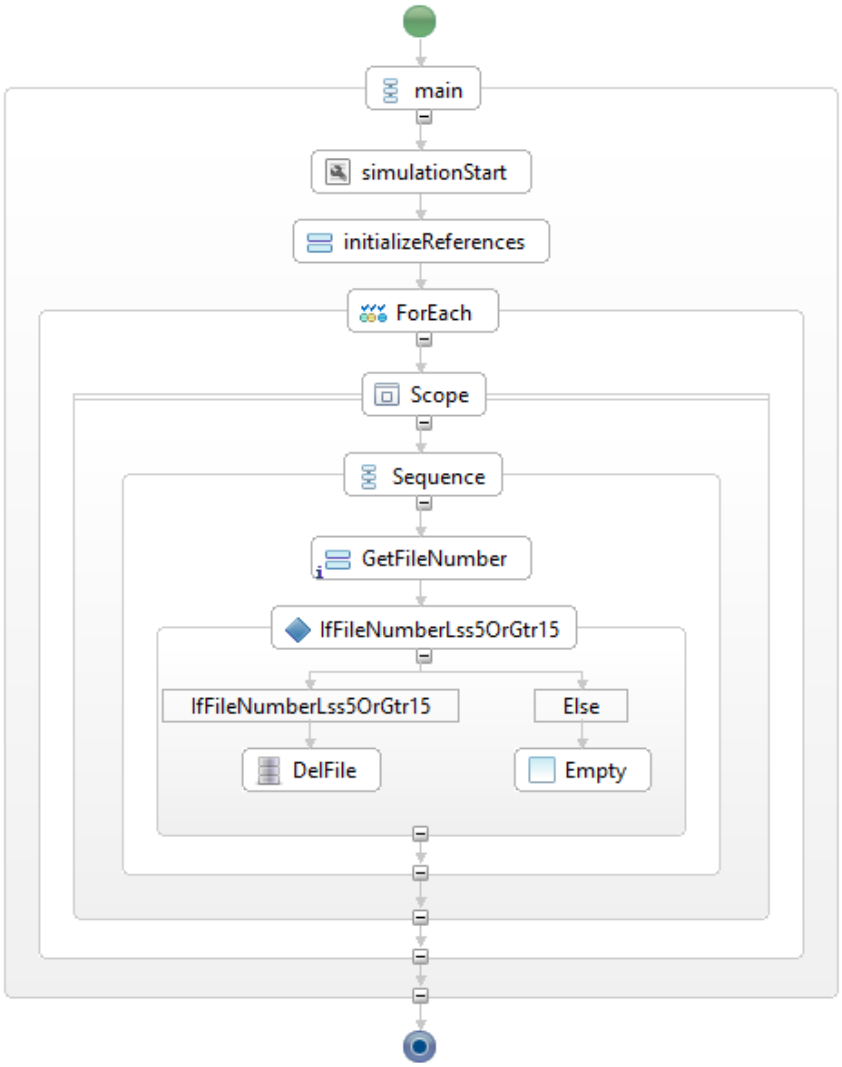


Abbildung 8.8: unoptimierter Beispielworkflow zum Löschen von Dateien mit Nummerierung < 5 und > 15

Listing 8.24 Zuweisungsanweisung der Assign-Aktivität GetFileNumberToDelete nach Anwendung der Predicate-Pushdown-Regel

```
if ($Counter < 5 or $Counter > 15)
then $Counter
else -100
```

Listing 8.25 Bedingungsanweisung der If-Aktivität IfFileNumberLss5OrGtr15 nach Verschieben der Filteroperation in die vorhergehende Assign-Aktivität

```
SET delete=1 & (IF #fileNumber# GTR 5 IF #fileNumber# LSS 15 SET delete=) & IF DEFINED delete
    (DEL C:\tmp\out\VisInputFile#fileNumber#.csv)
```

muss die **zweite Variante der Predicate-Pushdown-Regel** angewendet werden. Dabei ist die IssueCommand-Aktivität abhängig von der Filter-Operation der If-Aktivität. Die Filter-Operation wiederum ist abhängig von der Variablen *fileNumber*, dessen Wert von getFileNumber geschrieben wird.

Die **2. Variante der Predicate-Pushdown-Regel** sieht es vor, die Filter-Operation in die nachfolgende Aktivität zu verschieben, die von dieser Filter-Operation abhängt (*Option 2*) oder in die vorhergehende Aktivität, von der die Filteroperation selbst abhängig ist (*Option 1*). Für die **2. Option** könnte z.B. ein ähnlicher XPath 1.0-Ausdruck, wie in Abschnitt 8.6 beschrieben, zum Einsatz kommen. Dieser Lösungsweg wird hier jedoch nicht weiterverfolgt, da sie eine eher unpraktische, Methode darstellt. Bei einem XPath2.0 Ausdruck würde sich das Verschieben einfach gestalten. Hierfür wird aus der ursprünglichen Assign-Aktivität eine neue Assign-Aktivität GetFileNumberToDelete abgeleitet, welche die Zuweisungsanweisung in Listing 8.24) ausführt. Hier wird der Variable *fileNumber* der Wert des aktuellen Schleifenindexes nur dann zugewiesen, wenn das Ergebnis der Auswertung der If-Bedingung postitiv ausfällt, andernfalls wird ein negativer Wert zugewiesen, in der Annahme dass keine Datei eine negative Nummerierung besitzt. Der resultierende Workflow ist in Abbildung 8.9 dargestellt. Die If-Aktivität, sowie die Empty-Aktivität wurden entfernt. Aus der Assign-Aktivität GetFileNumber wurde eine neue Assign-Aktivität *GetFileNumberToDelete* abgeleitet, welche nun direkter Vorgänger der IssueCommand-Aktivität DelFile ist. Die Filterung der Dateien wurde von der ursprünglichen Aktivität in diese neue Assign-Aktivität verschoben.

Der Vollständigkeit halber, wird die **1. Option** kurz besprochen. Hier müsste die Filteroperation der If-Aktivität in den DM command der IssueCommand-Aktivität verschoben werden (siehe Listing 8.25). Für diesen Befehl ist ein logischer OR-Operator notwendig. Da Windows-Shell keinen vordefinierten OR-Operator bereitstellt, wird als Workaround ein logisches AND der Teilbedingungen konstruiert (Aneinanderreihung von zwei IF-Befehlen), der bei positiver Auswertung negiert wird. Für die Negation wird eine Variable *delete*, welche zuvor mit dem Wert 1 initialisiert wurde, der Wert null zugewiesen. Die Löschoperation wird schliesslich nur dann ausgeführt, falls *delete* definiert ist.

Abbildung 8.10 zeigt den Workflow nach Anwendung der *zweiten Variante der Predicate-Pushdown-Regel*. Die If-Aktivität, sowie die Empty-Aktivität wurden entfernt. Aus der IssueCommand-Aktivität DelFile wurde eine neue IssueCommand-Aktivität *IfFileNumberLss5OrGtr15* abgeleitet, welche direkter Nachfolger von getFileNumber ist. Die Filterung der Dateien wurde von der ursprünglichen Aktivität in die neue IssueCommand-Aktivität verschoben. Dadurch werden keine grossen Laufzeitverbesserungen

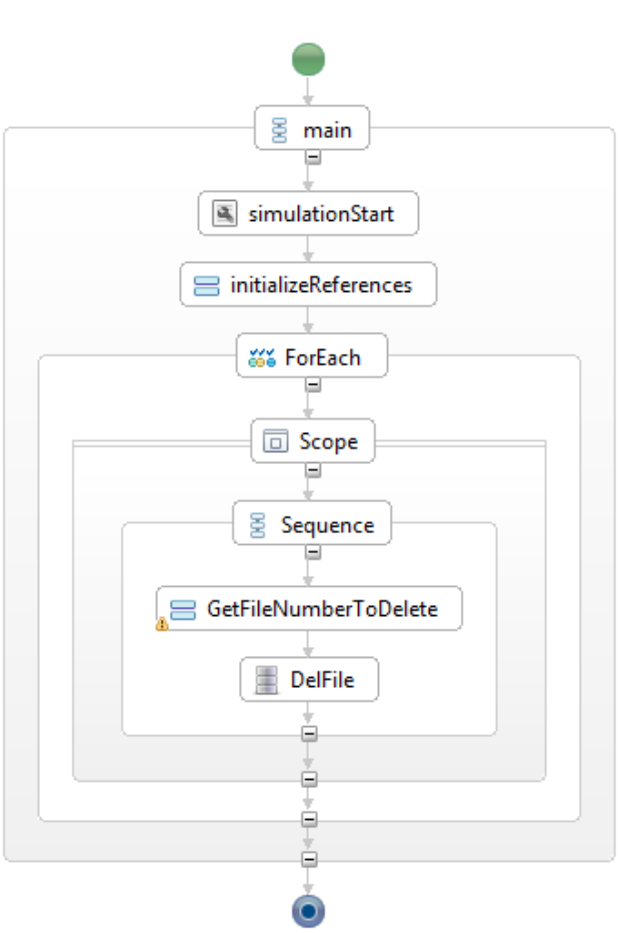


Abbildung 8.9: Der resultierende Workflow nach Anwendung der Predicate-Pushdown-Regel

erzielt werden können. Es wurde jedoch die Anzahl der Aktivitäten im Workflow verringert und die eventuelle Anwendung weiterer Optimierungsregeln ermöglicht.

Die Erläuterungen werden anhand des Workflow aus Abbildung 8.10 fortgesetzt. Die Transformation im anderen Fall gestaltet sich im Großen und Ganzen ähnlich. Der Workflow erfüllt die Voraussetzungen für die Anwendung der **Element-To-Set-Regel**. Innerhalb der Schleife liefert die Assign-Aktivität getFileNumber mit der Variablen *fileNumber* die in jeder Schleifeniteration notwendigen Information um die nächste Datei referenzieren zu können. Die IssueCommand-Aktivität stellt die eigentliche, datenverarbeitende Aktivität dar. Basierend auf der Variablen *fileNumber*, existiert zwischen ihr und der Assign-Aktivität getFileNumber eine exklusive Schreib-Lese-Datenabhängigkeit.

Abbildung 8.11 zeigt den Workflow nach Anwendung der *Element-To-Set-Regel*. Aus der Regelanwendung geht eine einzelne IssueCommand-Aktivität hervor, welche von der ursprünglichen IssueCommand-Aktivität abgeleitet wurde. Sie führt einen DM command aus, der mit einer For-Schleife alle Dateien innerhalb des relevanten Ordners durchläuft, wobei in jedem Schleifendurchlauf

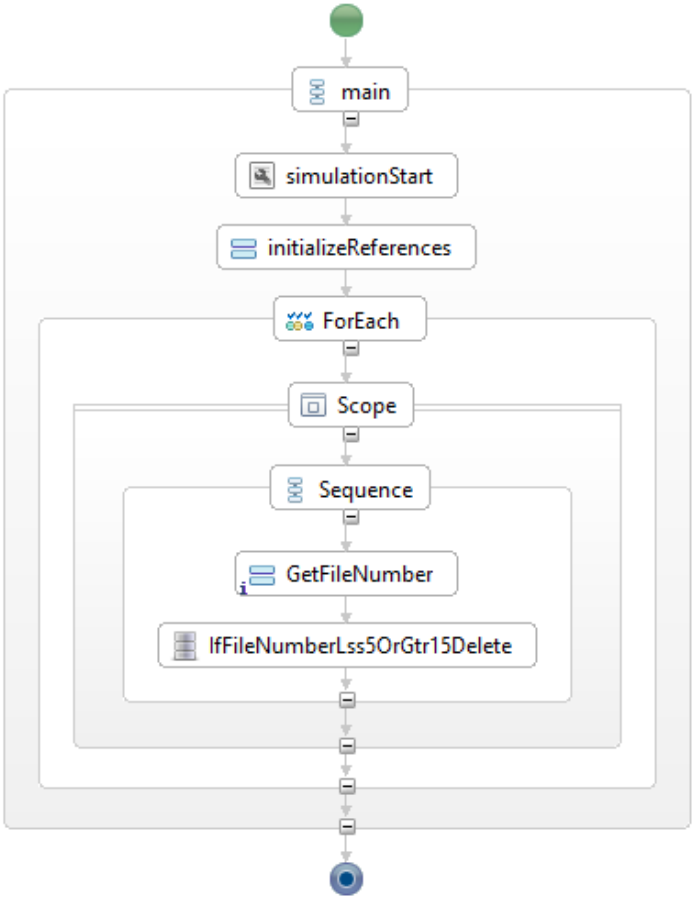


Abbildung 8.10: Workflow nach Anwendung der zweiten Variante der Predicate-Pushdown-Regel: Verschieben der Filteroperation in die abhängige IssueCommand-Aktivität

Listing 8.26 DM command der If-Aktivität nach Anwendung der Predicate-Pushdown- und Element-To-Set-Regel

```
cmd /q /e:on /v:on /c
      "for %a in (C:\tmp\out\VisInputFile*.csv) do
            (set "n=%~na" & set /a "n=!n:~12!">nul &
            set "delete=1" & (if !n! gtr 5 if !n! lss 15 set "delete=") &
            if defined delete (del "%a"))"
```

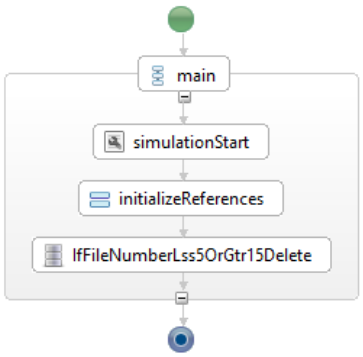


Abbildung 8.11: Workflow nach Anwendung der Element-To-Set-Regel

einer Variablen *a* die aktuell referenzierte Datei zugewiesen wird. Aus *a* wird die Dateinummer herausgefiltert und der Variable *n* zugewiesen(Listing 8.26). Anschliessend wird geprüft, ob die Dateinummer kleiner als 5 oder grösser als 15 ist. Der logische Or-Operator OR wird erneut aus einem logischem AND und einer Negation gebildet. Vorteil dieses Shell-Befehls ist, dass die Anzahl der zu durchlaufenden Dateien nicht im Vornhinein bekannt sein muss. Zudem wird ein numerischer Vergleich durchgeführt, was der Semantik des ursprünglichen Workflows entspricht, bei dem die Dateinummer durch die Variable *fileNumber* numerisch verglichen wird. Des Weiteren wurde die ForEach-Aktivität, GetFileNumber und IfFileNumberLss5OrGtr15 entfernt. Der direkte Vorgänger der ForEach-Aktivität bzw. der direkte Nachfolger der Assign-Aktivität innerhalb der For-Schleife wurden mit der neuen IssueCommand-Aktivität verbunden. Die Variable *fileNumber* wurde entfernt, da sie in der neuen IssueCommand-Aktivität nicht mehr referenziert wird. Der Workflow hat als Resultat eine einfachere Struktur mit weniger Aktivitäten. Ferner wurde die Iteration von der Workflow- auf die Datenebene verschoben. Dadurch können gegebenenfalls hohe Laufzeitgewinne erzielt werden.

Würde man von der ursprünglichen Semantik abweichen, könnte ebenso ein Vergleich der Dateinamen als Test verwendet werden. Dann müsste z.B zunächst die Menge der Dateien mit Dateinamen VisInputFile5 bis VisInputFile15 ermittelt und anschliessend beim Durchlaufen aller Dateien nur jene Dateien gelöscht werden, deren Namen sich nicht in dieser Menge befindet.

8.8 Optimierung des Proteinmodellierungsworkflows

In diesem Abschnitt soll untersucht werden, ob und wie Optimierungsregeln genutzt werden können, um den in Abschnitt 4.3 betrachteten Beispielworkflow zur Proteinmodellierung zu optimieren. Für die Untersuchungen wurde der Workflow mit Hilfe der Workflow-Sprache BPEL-DM unter Verwendung der XML-Datenbank *MonetDB/Xquery* als zu Grunde liegende Datenquellen folgendermassen

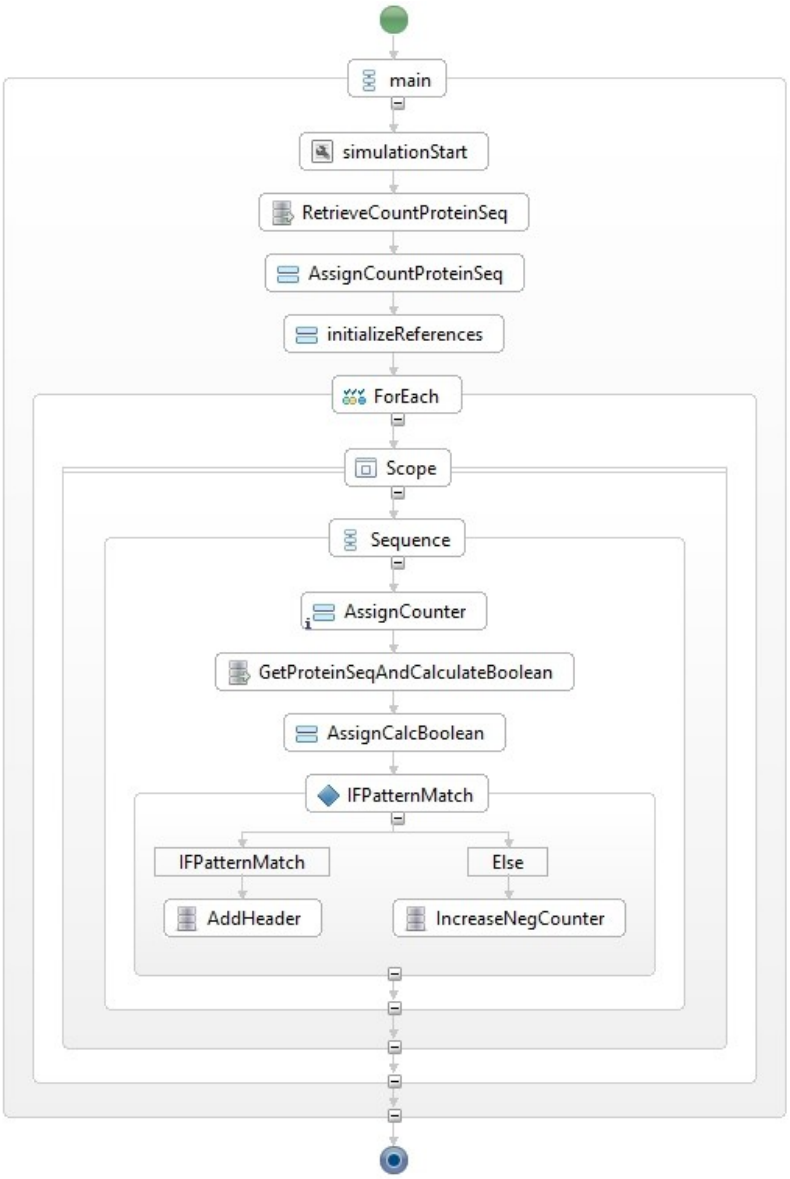


Abbildung 8.12: BPEL-DM-Workflow zur Proteinmodellierung

Listing 8.27 XPath-Ausdruck zur Ermittlung der Anzahl der Proteinsequenzen

```
number(count(doc("Proteinsequenzen.xml")/seqlist/seq))
```

Listing 8.28 XPath-Ausdruck zur Selektion des Elements mit Anzahl der Proteinsequenzen aus countProtSeqXML

```
number(countProtSeqXML/datacontent/text())
```

umgesetzt(siehe Abbildung 8.12). Das relevante Datenverarbeitungsmuster beginnt mit einer Retrieve-Aktivität RetrievCountProtSeqs, die mit Hilfe eines XPath-Ausdruckes aus einer XML-Datenbank bzw. aus einer XML-Datei, welche eine Liste von Proteinsequenzen enthält, die Anzahl der zu durchsuchenden Proteinsequenzen bezieht. Dieser XPath-Ausdruck ist in Listing 8.27 abgebildet. Der ermittelte Wert wird in einer SIMPL-Variablen *countProtSeqXML* des Datentyps *tXMLDataformat* gespeichert. Bei diesem Datenformat handelt es sich um den Standard-Datentyp im SIMPL-Rahmenwerk zur Speicherung von XML-Daten [Pie11]. Aufgrund der Struktur dieses Datenformats muss die Anzahl der Proteinsequenzen in einer Assign-Aktivität mit Hilfe eines XPath-Ausdruckes (siehe Listing 8.28) aus dem Element *data* herausgefiltert und in einer Integer-Variablen *countProtSeq* gespeichert werden.

Es folgt eine For-Each-Aktivität, welche die zuvor ermittelte Anzahl der Proteinsequenzen abzüglich Eins als Endwert des Schleifenindexes *Counter* verwendet. Beginnend bei Null als Startwert, wird der Wert der Variable *Counter* in jedem Schleifendurchlauf um eins erhöht, bis der Endwert erreicht ist. Der Schleifenrumpf wird eingeleitet durch einen scope, der wiederum eine sequence enthält. Die erste Aktivität in der sequence ist eine Assign-Aktivität AssignCounter. Sie kopiert bei jedem Iterationsschritt den aktuellen Wert des *Counters* in die BPEL-Variable *seqnumber*, welche die folgenden RetrieveData-Aktivität GetProteinSeqAndCalcBoolean nutzt, um im *i*-ten Iterationsschritt jeweils die *i*-te Proteinsequenz aus der Liste der Proteinsequenzen zu laden. Der DM command von GetProteinSeqAndCalcBoolean wendet einen XPath-Ausdruck (siehe Listing 8.29) auf die *i*-te Proteinsequenz an, der mit Hilfe der XPath 2.0-Funktion *matches* [3] innerhalb der Sequenz nach einem bestimmten Muster sucht. Für die Suche wird ein regulärer Ausdruck verwendet, der in einer Variablen *pattern* des Typs String gespeichert ist. Das Ergebnis des XPath-Ausdruckes wird erneut in einer Variable des Datentyps tXMLDataFormat abgespeichert. Die anschliessende Assign-Aktivität AssignCalcBoolean filtert mit Hilfe des XPath-Ausdrucks in Listing 8.30 den eigentlichen Booleschen Wert aus dieser Variable heraus und speichert ihn in einer Variablen des Typs Boolean ab.

Es folgt eine If-Aktivität IfPatternMatch, die mit Hilfe dieses Boolean Wertes den Kontrollfluss in zwei verschiedene Pfade aufteilt. Falls der Boolean Wert *true()* ist, d.h. die Proteinsequenz das Muster dem durch den regulären Ausdruck definierten Muster entspricht, wird der Header der Proteinsequenz

Listing 8.29 DM command der Aktivität GetProteinSeqAndCalcBoolean, das den i-ten Proteinsequenz lädt und einen regulären Ausdruck auf diesen anwendet

```
matches(doc("Proteinsequenzen.xml")/seqlist/seq[$Counter], $pattern, "i")
```

[3] matches

Listing 8.30 Anweisung der Aktivität AssignCalcBoolean, der den Boolean-Wert des Resultats der Mustersuche herausfiltert und der Variable protSeqBoolean zuweist

```
boolean($protSeqBooleanXML/documentContent/text())
```

Listing 8.31 Die Bedingungsanweisung von IfPatternMatch nach Anwendung der Assign-Merging-Regel

```
(boolean($protSeqBooleanXML/documentContent/text())= true())
```

durch eine IssueCommand-Aktivität in eine XML-Datei geschrieben, die sich auf derselben XML-Datenbank befindet wie die Liste der Proteinsequenzen. Sie enthält nach Ausführung des gesamten Workflows die Header aller Proteinsequenzen, in denen das Muster ausfindig gemacht werden konnte. Ist der Boolean Wert *false()*, wird in eine weitere XML-Datei auf derselben XML-Datenbank ein Zähler für die Gesamtzahl der Proteinsequenzen, die das Muster nicht enthalten, um eins erhöht. Wurde über alle zu durchsuchenden Proteinsequenzen iteriert, wird die Schleife verlassen und der Kontrollfluss an die der Schleife folgende Aktivität weitergeleitet.

Ein erster Ansatzpunkt für die Anwendung einer Optimierungsregel wäre die Entfernung des Schleifenkonstrukts durch Anwendung der *Element-To-Set-Regel*. Hier muss zunächst der Schleifenrumpf durch Anwendung weiterer Optimierungsregeln vereinfacht werden. Ein erster Schritt hierfür ist die Verschiebung der Assign-Aktivität AssignCalcBoolean in die nachfolgende If-Aktivität IfPatternMatch mit Hilfe der **Assign-Merging-Regel**. Die Bedingungen zur Anwendung der Regel sind erfüllt: Es existieren mit AssignCalcBoolean und IfPatternMatch zwei Aktivitäten, die jeweils Datenverarbeitunsoperationen ausführen, wobei AssignCalcBoolean eine Variablenzuweisung durchführt, bei welcher der Wert eines XPath-Ausdruckes einer Variable *protSeqBoolean* zugewiesen wird, welche anschliessend in der Bedingungsanweisung von IfPatternMatch exklusiv gelesen wird. Somit existiert eine exklusive Schreib-Lese-Datenabhängigkeit zwischen AssignCalcBoolean und IfPatternMatch basierend auf *protSeqBoolean*. Zudem sind die Befehlssprachen der relevanten Aktivitäten kompatibel.

Durch Anwendung der Regel wird die Zuweisungsanweisung von AssignCalcBoolean in die abhängige Bedingungsanweisung von IfPatternMatch verschoben, indem die Variable *protSeqBoolean* durch ihre eigene Definition ersetzt wird. In diesem Fall durch die XPath-Anweisung von AssignCalcBoolean. Aus der If-Aktivität wird eine neue If-Aktivität abgeleitet, welche die Bedingungsanweisung in Listing 8.31 ausführt. Die Variable *protSeqBoolean* wird entfernt. Anschliessend kann ebenso die Aktivität AssignCalcBoolean entfernt werden. Der direkte Vorgänger von AssignCalcBoolean, GetProtSeqAndCalcBoolean, wird mit seinem direkten Nachfolger IfPatternMatch verbunden. Durch die Verschmelzung der Assign- mit der If-Aktivität werden keine Datenflussabhängigkeiten zu anderen Aktivitäten beeinflusst.

Nach Anwendung der *AssignMerging-Regel* besteht zwischen IfPatternMatch und GetProtSeqAndCalcBoolean eine Datenabhängigkeit, basierend auf der Variablen *protSeqBooleanXML*. GetProtSeqAndCalcBoolean lädt Daten, auf denen das nachfolgende IfPatternMatch eine selektive Operation ausführt, ähnlich einem Prädikat in SQL-Anweisungen. Desweiteren sind die IssueCommand-Aktivitäten im If- und im Else-Zweig der If-Aktivität von dessen Bedingungsanweisung abhängig. Es bietet sich die Anwendung der **Predicate-Pushdown-Regel** an. Da die Filteroperation in einer If-Aktivität im

Listing 8.32 DM command der IssueCommand-Aktivität GetBooleanPatternMatchAndWriteResult nach Anwendung der zweiten Variante der Predicate-Pushdown-Regel: Verschieben der Filteroperation in nachfolgende, abhängige Aktivitäten

```
let $h := doc("Proteinsequenzen.xml")/seqlist/seq[$seqnumber]/header/text()
let $negcounter := doc("ProteinsequencesCountNegative.xml")/negcounter[1]
if (string(boolean($proteinseqBooleanXML/documentContent/text())) = "true" )
then do insert <header>"{$h}"</header>
as last into doc("ProteinsequencesHeaders.xml")/headerlist/
else do replace value of $negcounter with number($negcounter) + 1
```

Rumpf einer Schleife erfolgt, muss eine der *Varianten der Predicate-Pushdown-Regeln* zum Einsatz kommen. Die *erste Variante der Predicate-Pushdown-Regel* kann hier nicht angewendet werden, da hier keine datenbereitstellende Aktivität existiert. Die RetrieveData-Aktivität RetrieveCountProteinSeq vor dem Schleifenkonstrukt berechnet lediglich die Gesamtzahl der im relevanten XML-Dokument enthaltenen Proteinsequenzen.

Als weitere Möglichkeit bietet es sich hier nun an, die **zweite Variante der Predicate-Pushdown-Regel** anzuwenden. Dabei sind beide IssueCommand-Aktivitäten IncreaseNegCounter und AddPosHeader abhängig von der Filter-Operation der If-Aktivität. Die Filter-Operation wiederum ist abhängig von der Variablen *proteinseqBooleanXML*, dessen Wert von GetProteinSeqAndCalculateBoolean geschrieben wird. Die **zweite Option** schlägt vor, die Bedingungsanweisung der If-Aktivität in die nachfolgende, datenabhängige Aktivität zu verschieben. Hier müsste die Bedingungsanweisung theoretisch sowohl in den DM command der IssueCommand-Aktivität im If-Zweig, als auch in die IssueCommand-Aktivität im Else-Zweig verschoben werden. In der Praxis bedeutet dies, dass eine neue IssueCommand-Aktivität GetBooleanPatternMatchAndWriteResult erzeugt werden muss, die einen DM command ausführt, welcher die Semantik der gesamten If-Aktivität abdeckt (Listing 8.32). Würde man hier für den IF- und den Else-Zweig konsequent zwei IssueCommand-Aktivitäten erzeugen, die in beliebiger Reihenfolge sequentiell ausgeführt werden, so könnten diese später durch die Activity-Merging-Regel verschmolzen werden, womit man erneut eine einzelne IssueCommand-Aktivität erhalten würde. Die Abbildung 8.13 zeigt den Workflow nach Anwendung der zweiten Option der *zweiten Predicate-Pushdown-Regelvariante*.

Nach der Transformation ist die neue IssueCommand-Aktivität direkter Nachfolger der Aktivität GetProteinSeqAndCalculateBoolean. Diese Konstellation erfüllt die Voraussetzungen für die Anwendung der **Eliminate-Temporary-Container-Regel**. GetProteinSeqAndCalculateBoolean bezieht Daten aus der zu Grunde liegenden Datenquelle und weist sie einer Variable *proteinseqBooleanXML* zu. Die folgende IssueCommand-Aktivität GetBooleanPatternMatchAndWriteResult referenziert diese Variable lesend in ihrem DM command. Des Weiteren sind die Befehlssprachen der beiden relevanten Anweisungen kompatibel.

Eine Anwendung der *Eliminate-Temporary-Container-Regel* ersetzt *proteinseqBooleanXML* im DM command der IssueCommand-Aktivität durch ihre eigene Definition. Dadurch wird *proteinseqBooleanXML* überflüssig und kann entfernt werden. Ferner wird die Aktivität GetProteinSeqAndCalculateBoolean entfernt und durch die Ableitung einer neuen DM command eine neue **IssueCommand-Aktivität** PatternMatchAndWriteResult erzeugt. Dies ist nötig, da nach Auswertung der Filteroperation entweder das XML-Dokument *ProteinsequencesHeaders* oder *ProteinsequencesCountNegative* auf der

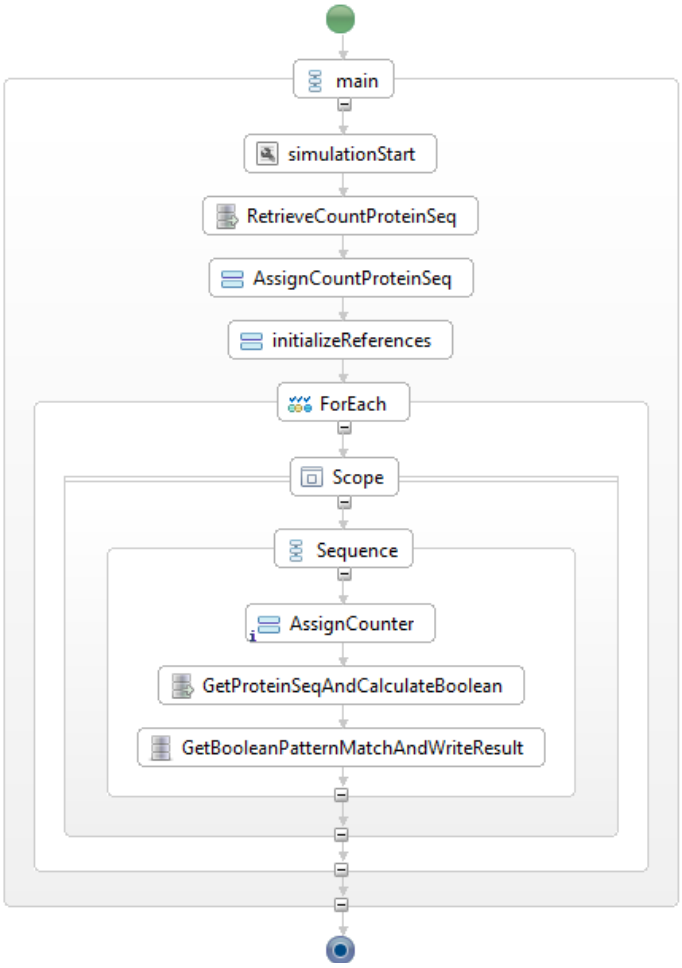


Abbildung 8.13: Der Proteinmodellierungsworkflow nach Anwendung der zweiten Variante der Predicate-Pushdown-Regel: Verschieben der Filteroperation in nachfolgende, abhängige Aktivitäten

zu Grunde liegenden Datenressource modifiziert werden müssen. Dies ist mit einer RetrieveData-Aktivität nicht durchführbar. Der DM command von PatternMatchAndWriteResult wird in Listing 8.33 gezeigt und der resultierende Workfow ist in Abbildung 8.14 dargestellt. Beim DM command handelt es sich um einen XQuery-Ausdruck, der auf der zu Grunde liegenden XML-Datenbank ausgeführt wird. Falls die Proteinsequenz das gesuchte Muster enthält, wird ihr Header im XML-Dokument *ProteinsequencesHeaders.xml* hinzugefügt. Andernfalls wird in der XML-Datei *ProteinsequencesCount-Negative.xml* die Anzahl der negativen Proteinsequenzen um eins erhöht. Im Gegensatz zum DM command in Listing 8.32 wird nun das Pattern-Matching-Verfahren direkt ausgeführt und nicht mehr das Resultat aus der Variable *proteinseqBooleanXML* ausgelesen.

Bei Anwendung der **ersten Option** der 2. Predicate-Pushdown-Regelvariante muss die Bedingungsanweisung der If-Aktivität in den DM command der RetrieveData-Aktivität GetProteinSeqAndCalcu-

Listing 8.33 DM command der IssueCommand-Aktivität PatternMatchAndWriteResult, nach Anwendung der Predicate-Pushdown-Regel: sowohl Option 1 als auch Option 2

```
let $h := doc("Proteinsequenzen.xml")/seqlist/seq[$seqnumber]/header/text()
let $negcounter := doc("ProteinsequencesCountNegative.xml")/negcounter[1]
 if (matches(doc("Proteinsequenzen.xml")/seqlist/seq[$seqnumber], $pattern, "i") //Hier wird das
     Pattern-Matching-Verfahren direkt ausgefhrt
  then do insert <header>"{$h}"</header>
   as last into doc("ProteinsequencesHeaders.xml")/headerlist/
 else
   do replace value of $negcounter with number($negcounter) + 1
```

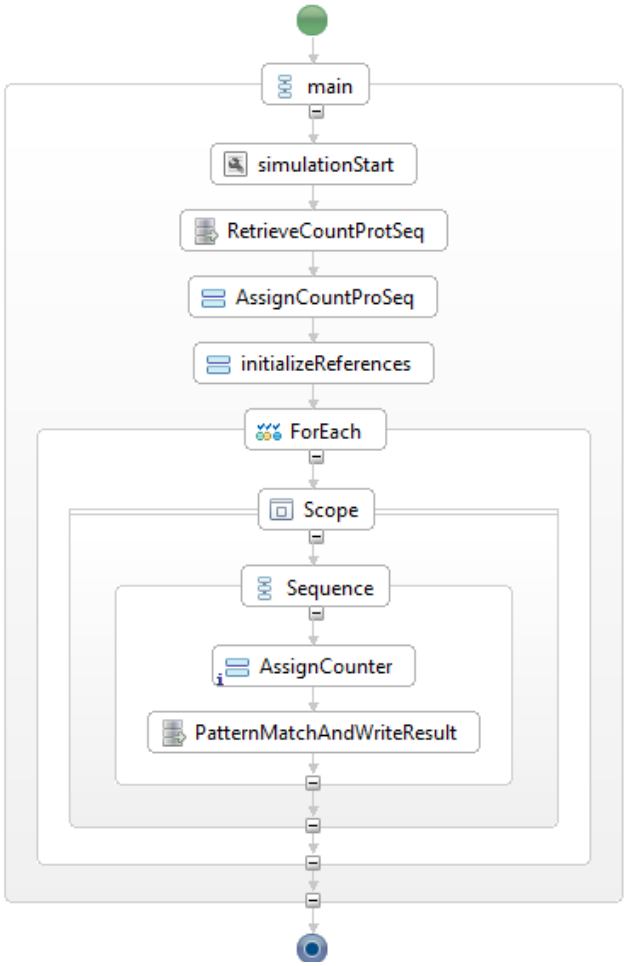


Abbildung 8.14: Der Proteinmodellierungsworkflow nach Anwendung der zweiten Variante der Predicate-Pushdown-Regel: sowohl Option 1 als auch Option 2

lateBoolean verschoben werden, wobei die Anweisungen der IssueCommand-Aktivitäten im If- und im Else-Zweig der If-Aktivität berücksichtigt werden müssen. Hierfür muss die GetProtSeqAndCalcBoolean wieder in eine **IssueCommand-Aktivität** umgewandelt werden, da die XML-Dokumente *ProteinsequencesHeaders.xml* oder *ProteinsequencesCountNegative.xml* modifiziert werden müssen. Der DM command für die IssueCommand-Aktivität PatternMatchAndWriteResult ist derselbe wie der, der bei Anwendung der zweiten Option der Predicate-Pushdown-Regelvariante erzeugt wurde (siehe Listing 8.33. Als Resultat der Anwendung der Regel entsteht der in Abbildung 8.14 bereits gezeigte Workflow. Durch das Verschieben der Bedingungsanweisung wurde IfPatterMatch zusammen mit den dazugehörenden IssueCommand-Aktivitäten in seinem If- und Else-Zweig aus dem Workflow entfernt und der direkte Vorgänger von IfPatterMatch, in diesem Fall PatternMatchAndWriteResult, mit seinem direkten Nachfolger verbunden. Zudem wurden die RetrieveData-Aktivität GetProteinSeqAndCalculateBoolean und die Variable *ProtSeqBooleanXML* entfernt, da sie durch die neue Aktivität nicht mehr referenziert wird. Sowohl durch *Option 1* als auch *Option 2* werden semantisch äquivalente Workflows generiert. Im Vergleich zur *zweiten Option* der zweiten Predicate-Pushdown-Regelvariante wurde mit der *ersten Option* der Zielworkflow jedoch in nur einem Transformationsschritt erreicht. Im Hinblick auf die Verständlichkeit der Transformation ist aufgrund der schrittweisen Vorgehensweise jedoch die *zweite Option* vorzuziehen.

Nach Anwendung der *Predicate-Pushdown-Regel* bietet sich die Anwendung der **Element-To-Set-Regel** an. Im ursprünglichen Workflow aus Abbildung 4.5 extrahiert die erste Aktivität die Liste der zu durchsuchenden Proteinsequenzen. In dieser BPEL-DM-Umsetzung des Workflows wird hier die Anzahl der Proteinsequenzen aus der Liste der Proteinsequenzen ermittelt und somit die maximale Anzahl der Schleifendurchläufe festgelegt. Die anschliessende For-Each-Aktivität entspricht der Schleifen-Aktivität, die Assign-Aktivität AssignCounter entspricht der Iterator-Aktivität und die Issue-Command-Aktivität PatternMatchAndWriteResult der datenverarbeitenden Aktivität der Regelbedingungen. Da PatternMatchAndWriteResult der einzige Leser der Variablen *seqnumber* ist, existiert zwischen PatternMatchAndWriteResult und der Assign-Aktivität der For-Schleife AssignCounter eine exklusive Schreib-Lese-Datenabhängigkeit. Damit sind alle Regelbedingungen erfüllt, um dem Aktionsteil der*Element-To-Set-Regel* anzuwenden.

Aus der Regelanwendung geht eine einzelne Issue-Command-Aktivität PatternMatchProtSeqList hervor, welche von der ursprünglichen Aktivität PatternMatchAndWriteResult abgeleitet wurde. Sie führt einen mengenbasierten XQuery-Befehl aus, der die Funktionalität der Operationen aller im Schleifenrumpf des ursprünglichen Workflows ausgeführten Aktivitäten abdeckt (siehe Listing 8.34). Er referenziert innerhalb eines Befehls alle Proteinsequenzen innerhalb der Liste der Proteinsequenzen und führt einen Patternvergleich aus. Dem Resultat des Patternvergleichs entsprechend wird entweder das XML-Dokument *ProteinsequencesHeaders* oder *ProteinsequencesCountNegative* aktualisiert. Der Einsatz eines XQuery-Befehls ermöglicht die mengenbasierte Verarbeitung der Daten, so dass auf eine iterative Schleife und alle mit ihr verbundenen Aktivitäten verzichtet werden kann.

Somit werden die RetrieveData-Aktivität RetrieveCountProtSeq, die Assign-Aktivität AssignCountProtSeq ausserhalb der Schleife, die ForEach-Aktivität an sich, die Assign-Aktivität AssignCounter und die Issue-Command-Aktivität PatternMatchAndWriteResult aus dem Workflow entfernt. Anschliessend wird der direkte Vorgänger der Retrieve-Data-Aktivität RetrieveCountProtSeq bzw. der direkte Nachfolger der Issue-Command-Aktivität PatternMatchAndWriteResult innerhalb der For-Schleife mit der neuen Issue-Command-Aktivität PatternMatchProtSeqList verbunden. Des Weiteren

Listing 8.34 Dm command der Aktivität PatternMatchProtSeqList

```
let $negcounter := doc("ProteinsequencesCountNegative.xml")/negcounter[1]
for $seq in doc("Proteinsequences.xml")/seqlist/seq
 $h in $seq/header/text()
      return
    if (matches ($seq, $pattern, "i"))
           then
           do insert <header>"{$h}"</header>
      as last into doc ("ProteinsequencesHeaders.xml")/headerlist
        else
           do replace value of $negcounter with number($negcounter) + 1
```

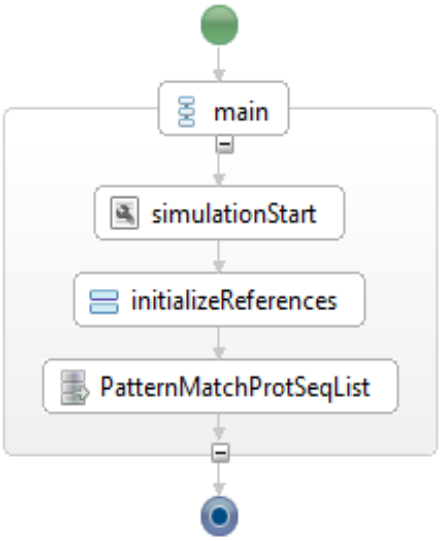


Abbildung 8.15: Der Proteinmodellierungsworkflow nach Anwendung Predicate-Pushdown- sowie der Element-To-Set-Regel

werden die Variablen *countProtSeqXML*, *countProtSeq* und *seqnumber* entfernt, da die zu verarbeitende Datenmenge nicht mehr sequentiell durchlaufen werden muss. Der resultierende Workflow ist in Abbildung 8.15 dargestellt. Als Endresultat ergibt sich ein Proteinmodellierungsworkflow, der mit insgesamt 4 Aktivitäten deutlich weniger Aktivitäten aufweist als der ursprüngliche Workflow mit 14 Aktivitäten. Zudem wird eine laufzeitaufwendige, schleifenbasierte Verarbeitung der relevanten Datenmenge vermeidet.

Wäre der Zähler der negativen Proteinsequenzen nicht ein Element in einem XML-Dokument, das auf der XML-Datenbank abgelegt ist, sondern ein in einer Workflow-Variable gespeicherter Integer-Wert, könnte der XQuery-Ausdruck der IssueCommand-Aktivität dieser Workflow-Variable keinen Wert zuweisen, da in XQuery-Ausdrücken generell keine Zuweisungen an globale Variablen definiert werden können. Hier könnte der Workflow so abgeändert werden, dass in der Schleife lediglich das XML-Dokument *ProteinsequencesHeaders.xml* für die positiven Proteinsequenzen modifiziert wird. Nach dem Schleifendurchlauf könnte anschliessend eine zusätzliche Retrieve-Data-Aktivität

die Anzahl der Header in ProteinsequencesHeaders von der bereits berechneten Gesamtzahl der Proteinsequenzen subtrahieren und so die Anzahl der negativen Proteinsequenzen ermitteln. Das Ergebnis würde in einer Workflow-Variablen des Typs *tXMLDataformat* abgelegt werden und wäre somit im Workflow-Kontext zugreifbar.

8.9 Optimierung des biomechanischen/systembiologischen Workflows

Sowohl beim biomechanischen als auch beim systembiologischen Workflow stellt die Datenbereitstellungsphase das für die Optimierung relevante Workflow-Fragment dar. Da bei beiden Workflows der Aufbau dieser Datenbereitstellungsphase exakt gleich ist, werden im Folgenden nur die Optimierungsmöglichkeiten für den biomechanischen Workflow diskutiert. Alle Beobachtungen können auf den systembiologischen Workflow übertragen werden. Da während der Bearbeitung dieser Diplomarbeit kein ausführbarer Workflow zur Verfügung stand, wurde ein Testworkflow erstellt, der grob die Datenbereitstellungsphase im biomechanischen bzw. systembiologischen Simulationsworkflow nachahmt. In Abbildung 8.16 rechts ist dieser Workflow gezeigt. Details aus dem Original-Workflow wie die Erzeugung einer *scopeDataSource* und dessen Aktualisierung in jedem Schleifendurchlauf wurden weggelassen. Alle zu übertragenden Dateien VisInputFile1 bis VisInputFile n liegen auf einem Unix-System in Verzeichnissen /opt/in1 bis /opt/in n. Das relevante Workflow-Fragment im ausführbaren Workflow wird eingeleitet durch eine Assign Aktivität InitContainerList. Diese erstellt eine *data container reference list conRefList*, welche die zu den zu transferierenden Dateien gehörenden *data container reference variables* enthält. Anschliessend wird in einer Schleife über dieser Liste iteriert. In jedem Iterationsschritt werden mit einer Assign-Aktivität getCounter der aktuelle Schleifenzähler ermittelt, sowie mit einer Assign-Aktivität GetContainer einer Variable *currentContainer* die Werte der aktuell referenzierten *data container reference variable* zugewiesen. Die TransferData-Aktivität TransferCurrentInputFile kopiert schliesslich die relevanten Dateien in das Zielverzeichnis.

Als Optimierungsmöglichkeit bietet sich hier die **Element-To-Set-Regel** an, um die schleifenbasierte Übertragung der Dateien durch einen **mengenbasierten** Befehl zu ersetzen. Dabei kann InitContainerList als datenbereitstellende Aktivität, getCounter und GetContainer als Iterator-Aktivitäten und TransferCurrentInputFile als eigentliche, datenverarbeitende Aktivität betrachtet werden. Da die TransferCurrentInputFile der einzige Leser der Variablen *currenContainer* ist, existiert zwischen ihr und GetContainer eine exklusive Schreib-Lese-Datenabhängigkeit. Zudem besteht zwischen GetContainer und InitContainerList basierend auf *containerRefList* eine exklusive Schreib-Lese-Datenabhängigkeit. Eine Anwendung der *Element-To-Set-Regel* unter Verwendung einer TransferData-Aktivität ist möglich, wenn nur eine Datenquelle für *datasource* angegeben werden muss, d.h. wenn alle Datencontainer auf derselben Datenressource liegen. Des Weiteren muss eine mengenbasierte Anweisung gefunden werden können, welche alle relevanten Dateien vollständig und ausschliesslich erfassen kann.

In diesem Fall könnte beispielsweise der Dateipfad eines *Überordners* verwendet werden. Dies funktioniert jedoch nur, falls der ausgesuchte Überordner ausschliesslich relevante Dateien enthält. Dieses Konzept wird bei der Betrachtung der Datenmanagementpatterns in Kapitel 9 erneut aufgegriffen. Dort wird ebenso besprochen, wie eine Optimierung aussehen kann, wenn zwar alle Dateien auf

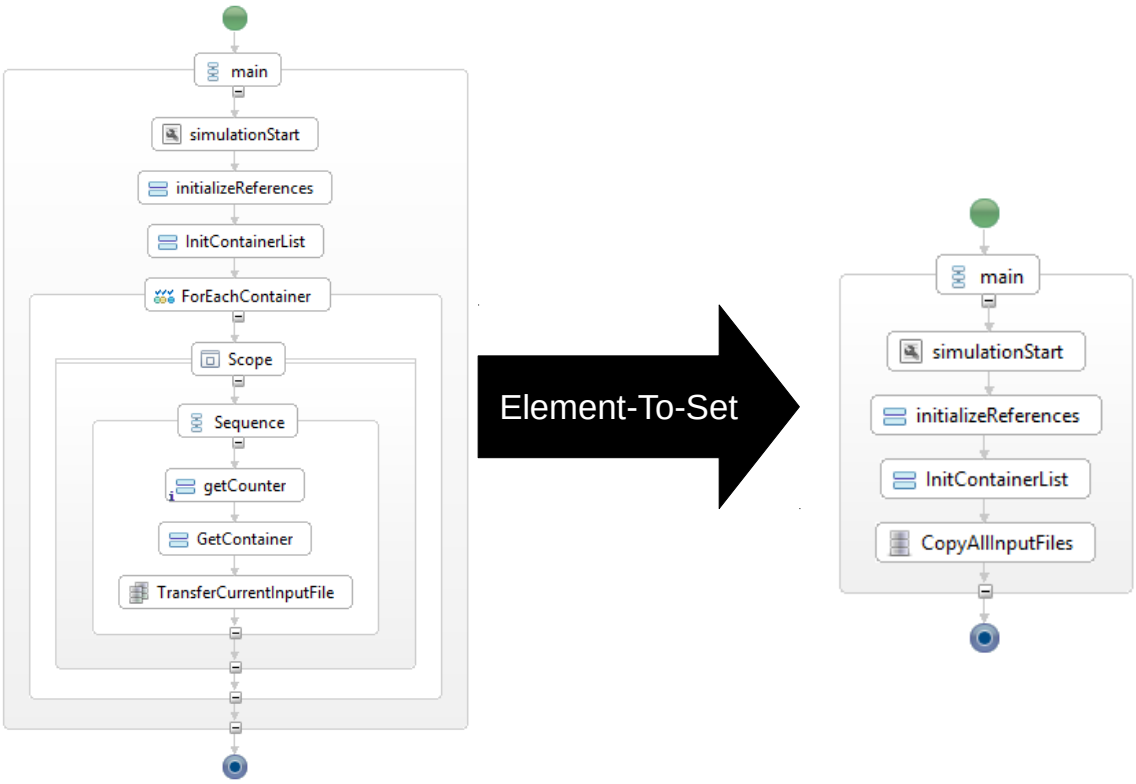


Abbildung 8.16: Testworkflow, der die Grundfunktion der Datenbereitstellungsphase des biomechanischen Workflows nachahmt, links vor Anwendung der Element-To-Set-Regel und rechts nach Anwendung der Element-To-Set-Regel

Listing 8.35 DM command der IssueCommand-Aktivität zur Erfassung aller Pandas-Eingabedateien

```
conRefListvar="#conRefLString#" &&
allpaths="$(echo "$conRefListvar" | xmlstarlet sel -t -m
"//GenericContainerReference/*" -v "directory" -v "file" -o " " -)"
&& sshpass -p "openstack" scp $allpaths "openstack@10.0.0.16:/opt/transfertest/"
```

derselben Datenressource liegen, jedoch nur für Teilmengen der Dateien Überordner gefunden werden können.

Neben der Verwendung von Überordnern, wurde zusätzlich eine alternative Umsetzung der *Element-To-Set-Regel* entwickelt. Diese basiert auf der Idee, *data container reference lists* als String bereitzustellen, und diesen dann auf Kommandozeilenebene zu parsen. Durch Anwendung der Regel entsteht ein optimierter Workflow, der in Abbildung 8.16 links gezeigt ist. Hier wurde das gesamte Schleifenkonstrukt durch eine IssueCommand-Aktivität CopyAllInputFiles ersetzt. In InitContainerList wird der Inhalt von *conRefList* als String-Wert einer Workflow-Variablen *conRefLString* zugewiesen. Im DM command von TransferInputFiles wird der Inhalt von *conRefLString* wiederum einer Shell-Variable *conRefListvar* zugewiesen, d.h. die Workflowvariable wird auf Kommandozeilenebene zur Verfügung

Listing 8.36 DM command der IssueCommand-Aktivität für den Beispiel-Biomechanischen Workflow

```
sshpass -p "openstack" scp [currentContainer] openstack@192.168.209.227:/opt/100016transfertest/
```

gestellt (siehe Listing 8.35). Dadurch kann sie anschliessend mit dem XML-Kommandozeilen-Tool *xmlstarlet* geparst werden. Dabei werden durch den Parameter *m* alle GenericContainerReferences selektiert. Im Ausgabe-Template (eingeleitet durch Parameter *v*) wird festgelegt, dass für jede GenericContainerReference jeweils das Element *directory* und das Element *file* getrennt durch eine Leerstelle zurückgeliefert werden soll, d.h. für jede Eingabedatei wird der absolute Pfadname konstruiert. Der Gesamtstring wird anschliessend in einer Variable *allpaths* gespeichert, mit dessen Hilfe schliesslich in einem einzelnen Kopier-Befehl alle Dateien kopiert werden können. Im Vergleich zum ursprünglichen Workflow wird hier ein Schleifenkonstrukt vermieden. Zudem muss kein *currentContainer* erstellt werden, dem in jedem Schleifendurchlauf aktuellen Werte zugewiesen werden müssen. Diese Lösung bietet sich z.B. an, wenn alle Eingabedateien zwar auf derselben Datenquelle, jedoch auf verschiedenen Laufwerken liegen, so dass keinerlei Überordner verwendet werden können. Im Testworkflow wurden mit *sshpass scp* Dateien zwischen einem lokalen Unix- und einem entfernten Unix-Rechner kopiert. Da *xmlstarlet* auch für Windows verfügbar ist, dürfte die Lösung dort ähnlich aussehen. Lauzeiten für diese Umsetzung der *Element-To-Set-Regel* sind in Kapitel 10 zu finden.

Eine weitere, alternative Optimierungsmöglichkeit für den Original-Workflow, unabhängig von der *Element-To-Set-Regel* ist die Verwendung einer IssueCommand- anstelle der TransferData-Aktivität. Diese Optimierungsmöglichkeit war nicht unmittelbar aus der Struktur des Workflows ersichtlich. Bei Testläufen mit dem Workflow machte sich jedoch bemerkbar, dass die TransferData-Aktivität in jedem Iterationsschritt die aktuell referenzierte Datei auf den lokalen Windows-Rechner kopierte, bevor er ihn auf das Zielverzeichnis auf dem Unix-Rechner übertrug. Dies ähnelt dem Szenario in Abschnitt Abschnitt 8.4, wobei hier keine explizite, sondern eine implizite Verwendung eines temporären Datencontainers vorliegt. Aufgrund dieser Beobachtung wurde die TransferData-Aktivität durch eine IssueCommand-Aktivität ersetzt, welche den DM command in Listing 8.36 verwendet, um die relevante Datei zwischen den Unix-Rechnern zu kopieren. Dabei wird das Tool *sshpass* verwendet um eine nicht-interaktive Authentifikation beim Transfer von Dateien mit *scp* zwischen den zwei Unix-Rechnern zu realisieren. Nach dieser Transformation erfolgte ein direkter Dateitransfer ohne Zwischenspeicherung der Dateien auf dem lokalen Windows-Rechner, sodass diese Transformation der Umsetzung der Grundidee der **Eliminate-Temporary-Container-Regel** gleichkommt. Lauzeiten für diese Umsetzung der *Element-To-Set-Regel* sind in Kapitel 10 zu finden.

9 Optimierungen der Transformation von Datenmanagementpatterns

In diesem Kapitel wird untersucht, ob und wie die in vorigen Kapiteln betrachteten Optimierungsregeln und Optimierungen im Allgemeinen in die Transformation von Datenmanagementpatterns des SIMPL-Rahmenwerks integriert werden können. Dabei werden unter anderem Bedingungen herausgearbeitet, unter denen eine bestimmte Optimierung auf einen Teil eines Workflows angewendet werden kann und schliesslich auf konkreten Patterns bestimmte Optimierungsregeln eingesetzt.

In Abschnitt 9.1 wird zunächst die Grundidee für die Optimierung während der Patterntransformation erklärt. Anschließend wird in Abschnitt 9.2 das *Simulation-Oriented Data Provisioning Pattern* in Abschnitt 9.3 das *Data Transfer and Transformation Pattern*, in Abschnitt 9.4 das *sequentiellen Data Iteration Pattern* und in Abschnitt 9.5 das *komplexe Container-To-Container-Pattern* auf Optimierungsmöglichkeiten hin untersucht. Zum Abschluss werden in Abschnitt 9.6 Ideen für die Optimierung des Kupplungsworkflows und seiner Datenmanagementpatterns beschrieben.

9.1 Untersuchung der Transformation von Datenmanagementpatterns

Die in Abschnitt 5.1 genauer betrachtete, regelbasierte Optimierung von datenintensiven Workflows bietet sich als Optimierungsansatz insbesondere für die Optimierung von Datenmanagementpatterns an, da hier ebenso eine **Regelbasiertheit** vorliegt. Die Strukturierung der Transformationen von Datenmanagementpatterns in Form von Regeln mit einem Bedingungs- und einem Aktionsteil, sowie die Ähnlichkeit des Transformationsprozesses könnte die Integration der im Kapitel 7 und 8 definierten und umgesetzten Optimierungsregeln vereinfachen.

In Abbildung 9.1 sind zum Vergleich das Prozessmodell der regelbasierten Transformation von Datanmanagementpatters (oben) und das Prozessmodell der heuristischen, regelbasierten Optimierung (unten) aus [Vrh11] gemeinsam dargestellt. Beide Prozesse weisen einen ähnlichen Verlauf auf:

- Bei der **Patterntransformation** traversiert der Pattern Transformer einen Workflowgraph und versucht bei Vorliegen eines korrekt parametrisierten Datenmanagentpatterns eine Transformationsregel anzuwenden. Für jedes Pattern existiert eine *Regelsequenz*, welche die für dieses Pattern generell anwendbaren Regeln umfasst. Diese legt zudem die Reihenfolge fest, in der die Regeln auf Anwendbarkeit überprüft werden. Jede Regel besteht aus einem *Bedingungsteil*, bei dessen Erfüllung der restliche Teil der Regel auf dem Pattern ausgeführt wird. Mit Hilfe einer Anfrage an eine *Workflow Fragment Library* bestimmt der *Fragment-Teil* der Regel ein

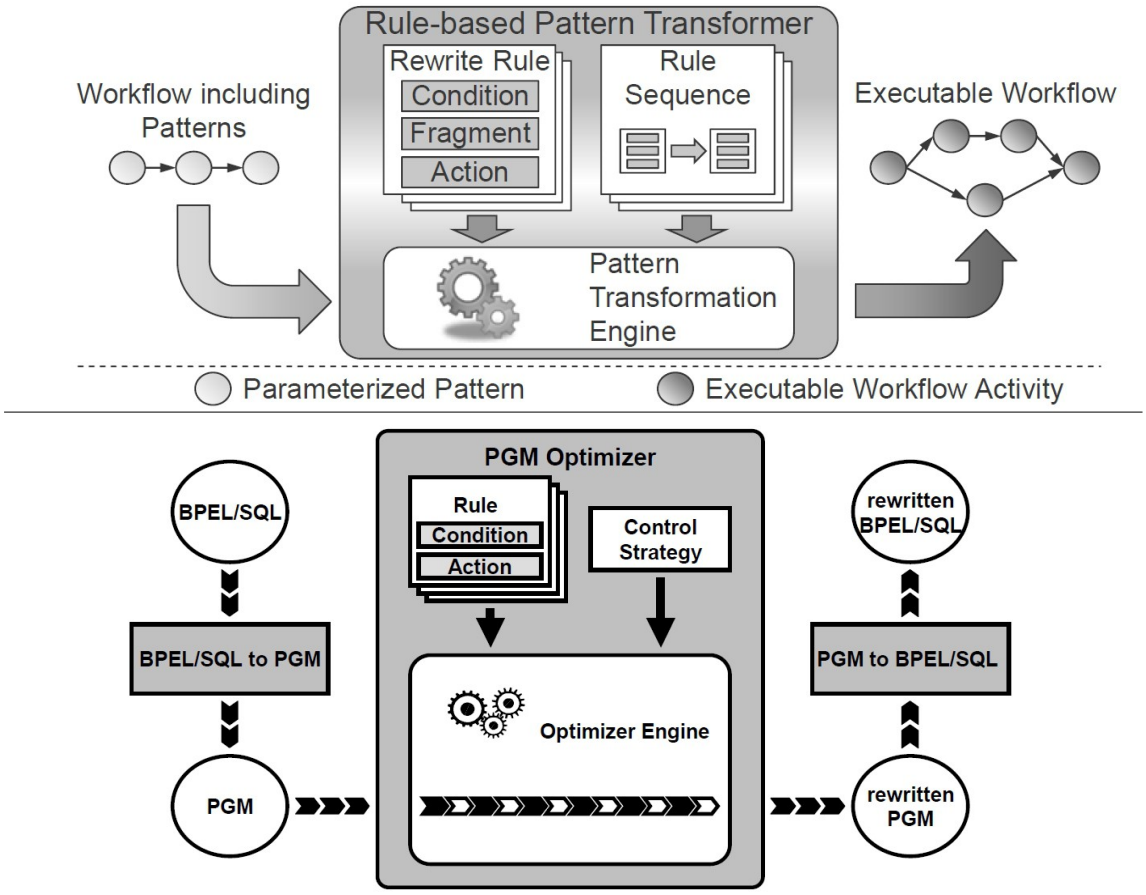


Abbildung 9.1: regelbasierte Transformation von Datanmanagementpatters vs. heuristische, regelbasierte Optimierung aus
citereimann2014pattern [Vrh11]

Workflow-Fragment-Template. Der *Aktionsteil* der Regel fügt diesem anschliessend Implementierungsdetails hinzu. Durch die Regelanwendung wird schliesslich das Pattern im Workflow durch dieses Workflow-Fragment ersetzt. Der resultierende Workflow deckt dabei dieselbe Semantik ab, wie der ursprüngliche Workflow.

- Der regelbasierte, heuristische **PGM-Optimierer** traversiert einen vorliegenden Workflowgraphen (der zuvor von BPEL-SQL in PGM umgewandelt wurde) und versucht bei jedem gefundenen Datenverarbeitungsmuster eine Optimierungsregel aus seiner *Regelbasis* anzuwenden. In einer *Kontrollstrategie* ist definiert, auf welche Teile eines Workflowmodells und in welcher Reihenfolge die Restrukturierungsregeln angewendet werden. Jede Regel besteht aus einem *Bedingungsteil*, bei dessen Erfüllung die im textitAktionsteil der Regel festgelegten Transformationsschritte ausgeführt werden. Der resultierende Workflow deckt dabei dieselbe Semantik ab, wie der ursprüngliche Workflow.

Bei beiden Ansätzen kommt also eine ähnliche Architektur zum Einsatz, die auf einem bestehenden Workflowgraphen die für eine Optimierung relevanten Teilbereiche ermittelt und mit Hilfe von Regeln Transformationen auf diesen ausführt. Dabei sind die Regeln in Gruppen organisiert, in der die Reihenfolge ihrer Überprüfung auf Anwendbarkeit festgelegt ist. Als Ergebnis wird ein veränderter Graph zurückgeliefert. Ein wesentlicher Unterschied ist, dass beim PGM/F-Ansatz mit einem ausführbaren, unoptimierten Workflow gestartet wird und als Resultat ein optimierter, ausführbarer Workflow zurückgeliefert wird. Dabei besteht der Ausgangs-Workflow in der Regel aus vielen Aktivitäten und der resultierende Workflow hingegen besitzt deutlich weniger Aktivitäten.

Im Gegensatz dazu wird bei der Patterntransformation mit einem abstrakten, nicht ausführbaren Workflow mit wenigen Workflow-Schritten (Patterns) begonnen und versucht für jedes eingebettete Pattern ein passendes Workflow-Fragment zu finden, bis alle Workflow-Fragmente ausführbar sind und ein ausführbarer Workflow zurückgeliefert wird. Die Patterntransformation berücksichtigt keine Optimierungseffekte, sodass resultierende Workflows im Hinblick auf Laufzeitleistung unoptimiert sein können. Dies spiegelt sich auch in den Regeln wider. Während beim PGM/F-Ansatz komplexe Restrukturierungen definiert sind, besitzen die Regeln beim Patterntransformation einen Fragment-Teil, der lediglich nach passenden Workflow-Fragmenten sucht. Im Aktionsteil der Regeln werden nur kleine Anpassungen oder Ergänzungen des Fragments festgelegt.

Die Integration von Optimierungen in die Patterntransformation kann grundsätzlich auf zwei verschiedene Weisen erfolgen. Die erste Variante wäre es, den PGM/F-Ansatz konsequent auf die Optimierung von Datenmanagementpatterns zu übertragen. D.h. es müssten bei der Patterntransformation vordefinierte, unoptimierte Workflow-Fragmente durch komplexe Restrukturierungen aktiv in optimierte Fragmente umgewandelt werden. Eine zweite Variante wäre es, in der Workflow Fragment Library sowohl optimierte, als auch unoptimierte Workflow-Fragmente vorzuhalten und je nach Szenario dann auf das entsprechende Fragment abzubilden. Hier besteht die Patterntransformation, wie zuvor, aus einer Auswahl der passenden Workflow-Fragmente und die Erweiterung dieses Fragments um Implementierungsdetails. In den Diskussionen in den folgenden Abschnitten wird eher der zweite Ansatz verfolgt. D.h. falls ein Pattern auf ein optimiertes Fragment abgebildet werden kann, dann wird dieses der Workflow Fragment Library, sowie die dazugehörige, neue Transformationsregel der Regelsequenz hinzugefügt. In den Regelsequenzen wird dann zuerst auf Anwendbarkeit der optimierten Regel, und erst wenn diese nicht anwendbar ist, auf Anwendbarkeit der unoptimierten Regel geprüft.

Generell ist es von Vorteil, bei der Untersuchung von Datenmanagementpatterns mit den Patterns der höheren Ebenen der Hierarchie der Datenmanagementpatterns (vergleiche Abschnitt 3.3.2) zu beginnen. Eine *Top-Down*-Herangehensweise erlaubt die frühzeitige Einplanung der möglichen Anwendung von Optimierungen und Optimierungsregeln. Je früher Optimierungsregeln angewendet werden können, desto früher werden durch Transformationen optimierte Datenmanagementpatterns erzeugt, welche als Grundlage für weitere Transformationssschritte dienen. Auf diese Weise können sich positive Optimierungsgewinne von oben nach unten in der Hierarchie der Datenmanagementpatterns fortpflanzen.

9.2 Das Simulation Oriented Data Provisioning Pattern

Das **Simulation Oriented Data Provisioning Pattern** gehört in der Hierarchie der Datenmanagementpatterns zu den *Simulation-Oriented Data Management Patterns*. Es stellt eine Abstraktion für Prozesse der Datenbereitstellung dar. Da die Datenbereitstellungsphase bei dem in Abschnitt 4.2 beschriebenen biomechanischen bzw. systembiologischen Workflow die für die Optimierung relevante Phase darstellt, stellen die folgenden Überlegungen zugleich Optimierungsmöglichkeiten für diese Workflows dar. Die relevanten Daten werden in Form eines *Simulationsmodells* (z.B. beim biomechanischen Workflow ein Modell zur Simulation der Veränderung der Knochenstruktur bei unterschiedlicher Belastung) und den zu diesem Simulationsmodell gehörenden *mathematischen Variablen* bereitgestellt. Zudem wird das Ziel der Datenbereitstellung (z.B. Software-Instanz oder Dienst) festgelegt. Den mathematischen Variablen sind in einer zum Simulationsmodell gehörenden *Ontologie* konkrete Instanzen zugeordnet, welche den Namen und den Dateipfad für die eigentlichen Eingabedateien spezifizieren.

Die Transformationskette beginnend beim *Simulation Oriented Data Provisioning Pattern* über mehrere Transformationsschritte bis hin zum ausführbaren Workflow wird im Folgenden beschrieben und ist in Abbildung 9.2 für den biomechanischen Workflow dargestellt. Das *Simulation Oriented Data Provisioning Pattern* wird im ersten Schritt auf Instanzen der **Data Transfer und Transformation Pattern** abgebildet (Abbildung 9.2, a)), wobei für die Instanzen der mathematischen Variablen mit Hilfe der entsprechenden Ontologie jeweils *data container reference variables* definiert werden, die Referenzen zu den Dateien speichern, in welchen die Instanzen der mathematischen Variablen beschrieben sind. Dies ist möglich, da jeder mathematischen Variable in der Ontologie ein konkreter Dateipfad zu einer Eingabedatei zugeordnet ist. Das *Data Transfer and Transformation Pattern* wird in einem nächsten Transformationsschritt in eine Assign-Aktivität InitContainerList sowie ein **sequentielles Data Iteration Pattern** mit eingebettetem *Container-To-Container-Pattern* abgebildet (Abbildung 9.2, b)). InitContainerList weist die Liste der *data container reference variables* einer einzelnen Variable zu und das *sequentielle Data Iteration Pattern* nutzt diese Liste anschliessend als Iterationsmenge. In einem letzten Transformationsschritt wird das *sequentielle Data Iteration Pattern* in eine Assign-Aktivität InitLocalContainerRenferenceList und eine For-Each-Aktivität, sowie das Container-To-Container-Pattern in eine Transfer-Data-Aktivität überführt. InitLocalContainerRenferenceList erzeugt dabei eine lokale Kopie der gegebenen *data container reference list*. Der resultierende Workflow ist in Abbildung 9.2, c) gezeigt. Da dieser Workflow zum Zeitpunkt der Bearbeitung dieser Diplomarbeit nicht lauffähig war, wurde der in 8.16 gezeigte Workflow als Umsetzung entworfen. Generell bildet das Schleifenkonstrukt den laufzeitaufwendigsten Teil der Datenbereitstellungsphase, welches letztendlich aus der mehrstufigen Transformation des *Simulation-Oriented-Data Provisioning Patterns* hervorgegangen ist.

Erhält das *Simulation Oriented Data Provisioning Pattern* als Eingabeparameter z.B. 500 Instanzen von mathematischen Variablen, muss im nächsten Schritt das *Data Transformation and Transfer Pattern* mit 500 *data container reference variables* umgehen, das *sequentielle Data Iteration Pattern* wiederum über einer *data container reference list* mit 500 *data container reference variables* iterieren, d.h. die For-Each-Aktivität im ausführbaren Workflow-Fragment 500 Datencontainer einzeln auf ein Zielverzeichnis kopieren. Eine frühzeitige Optimierung könnte beispielsweise die Anzahl der *data container reference variables* minimieren bzw. im Idealfall auf eine *data container reference variable*

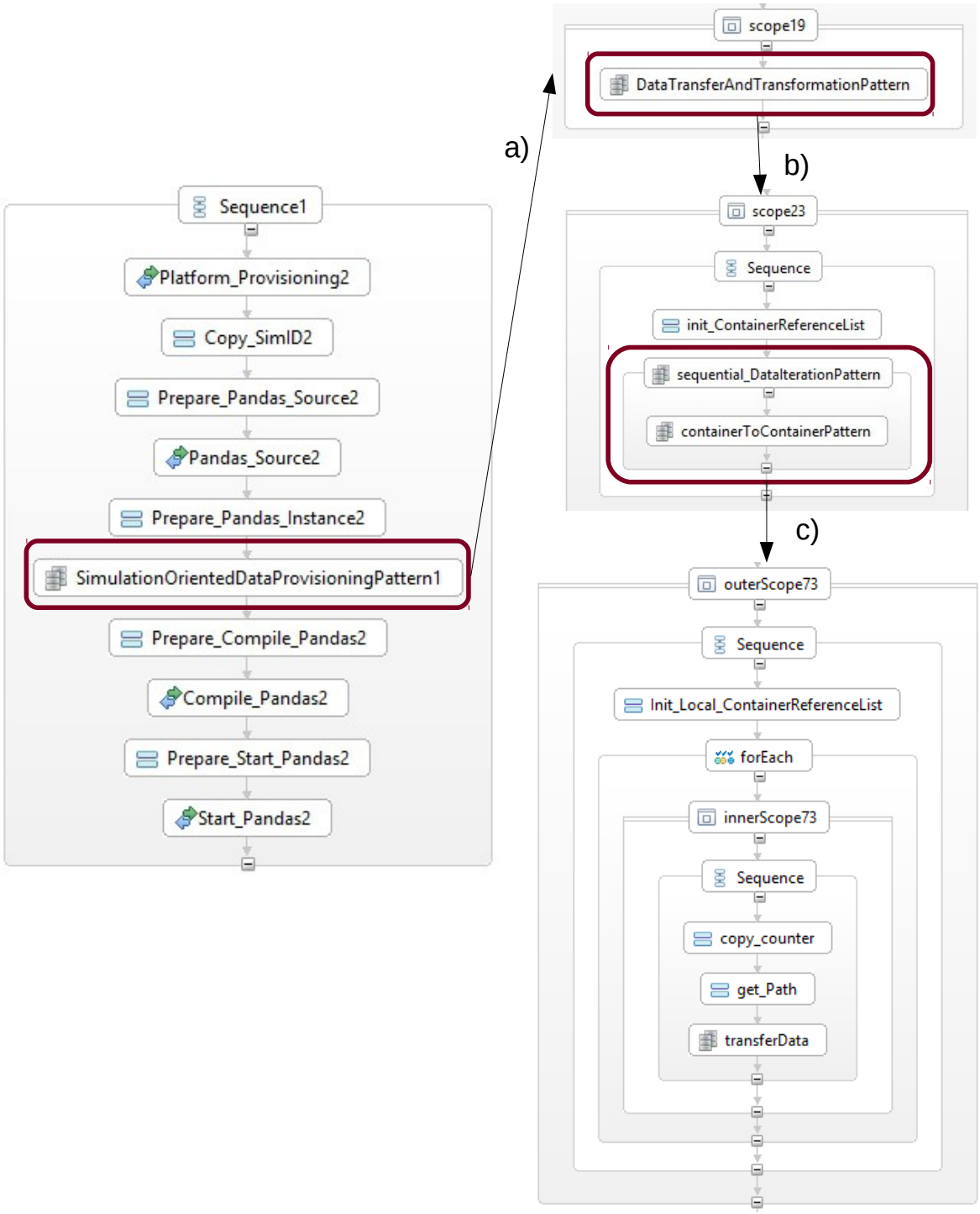


Abbildung 9.2: Die Transformationskette vom Workflow-Fragment zur Datenbereitstellung im biomechanische Workflow bis hin zum ausführbaren Workflow-Fragment

Listing 9.1 Code-Ansatz für die Methode MapDirNamesToFileNames

```
Methode MapDirNamesToFileNames:

Map<String,Set<String>> dir2filenames = new HashMap<>();
for( File f: filesToCopy ){
   String name = f.getName();
   String dir = f.getParent();
   Set<String> names = dir2filenames.get( dir );
   if( names == null ){
      names = new HashSet<>();
      dir2filenames.put( dir, names );
   }
   names.add( name );
}
```

reduzieren„ die alle 500 betrachten Dateien auf einmal beschreibt. Eine Möglichkeit wäre es hier, Datencontainer zu **gruppieren**, wobei z.B. der Dateipfad als Kriterium verwendet werden könnte. So könnte gleich zu Beginn, nachdem die Parametrisierung des *Simulation Oriented Data Provisioning Patterns* abgeschlossen ist, überprüft werden, ob die zu den mathematischen Variablen gehörenden Dateien im selben Ordner liegen. Ist das der Fall, muss für diese Dateien nur ein Datencontainer angelegt werden. Liegen die Daten zwar nicht im selben Ordner, aber in Unterordnern, welche denselben Überordner besitzen, dann kann ein einzelner Datencontainer erzeugt werden, wobei der Dateipfad dieses Überordners als Dateipfad verwendet wird.

Voraussetzung für die Verwendung eines Überordners ist, dass dieser nur zu übertragende Dateien enthält. Andernfalls werden Dateien mitübertragen, die eigentlich nicht dafür vorgesehen waren. Kann für eine Datei kein Überordner verwendet werden, muss weiterhin sein absoluter Dateipfadname verwendet werden. Das Idealziel wäre es, alle relevanten Dateien über einen einzelnen Dateipfad eines Überordners zu erfassen. Falls dies nicht möglich ist, sollten die Dateien so weit wie möglich in einzelne Gruppen gruppiert werden, sodass zumindest die Anzahl der Schleifendurchläufe reduziert werden kann. Für die Realisierung dieser Idee müsste die Transformationsregel, welche das *Simulation Oriented Data Provisioning Pattern* auf ein *Data Transfer and Transformation Pattern* abbildet, abgeändert werden. Sie müsste eine Methode **ReduceCountContainers** aufrufen, welche die Funktion hat, für alle Instanzen von mathematischen Variablen, die durch den Nutzer als Parameter übergeben wurden, so wenig wie möglich Referenzen auf Datencontainer zu erzeugen. Dabei würden die Informationen verwendet werden, die in der entsprechenden Ontologie hinterlegt sind, die zu dem als Parameter übergebenen Simulationsmodell gehört.

Mit Hilfe von einem Code-Ansatz soll hier die ungefähre Funktionsweise der Methode deutlich gemacht werden. Als erster Schritt erzeugt die Methode *ReduceCountContainers* eine Liste *filesToTransfer* der Namen aller zu übertragenden Dateien. Dazu müssen in der entsprechenden Ontologie des Simulationsmodells die Eingabedateien gefunden werden, die den zugehörigen mathematischen Variablen zugeordnet sind. ReduceCountContainers ruft anschliessend zwei Unterfunktionen *MapDirNamesToFileNames* (Listing 9.1) und *CreateGroups* (Listing 9.2) auf:

- Die Methode **MapDirNamesToFileNames** erzeugt einen HashMap *dir2filenames*, der eine Zuordnung von Pfadnamen zu einer Menge von Dateinamen vornimmt, d.h. die Dateinamen

Listing 9.2 Code-Ansatz für die Methode CreateGroups

```
for( String dirname: dir2filenames.keySet() ){
   File[] entries = new File( dirname ).listFiles();
   if( entries.length > dir2filenames.get( dirname ).size() ){
      for( String name: dir2filenames.get( dirname ){
         String pathname = new File( dirname, name ).getAbsolutePath();
         groupedpaths.add(pathname);
      }
   } else {
      groupedpaths.add(dirname);
   }
}
```

werden auf die Verzeichnisnamen abgebildet, in denen sie sich befinden. Sie iteriert hierfür auf der Liste *filesToTransfer* und erstellt für den Verzeichnisnamen der aktuell referenzierten Datei einen Eintrag in der Hashtabelle, falls dieser nicht bereits existiert, wobei der Verzeichnisname als Schlüssel und eine Menge bestehend aus dem Dateinamen als Wert dient. Existiert der Eintrag bereits, wird der Dateiname zur existierenden Menge der Dateinamen hinzugefügt. Nach dem Durchlauf der For-Schleife existiert für jedes Verzeichnis, welches zu transferierende Dateien aus *filesToTransfer* enthält, ein Eintrag in der Hash-Tabelle.

- Die Funktion **createGroups** ist für die Erzeugung von Gruppen verantwortlich. Hierzu iteriert die Funktion über den Schlüsselwerten der zuvor erzeugten Hash-Tabelle und lädt die Namen aller in diesem Verzeichnis real enthaltenen Dateien. Enthält das Array mehr Dateinamen als der entsprechende Eintrag in der Hash-Tabelle bedeutet dies, dass das relevante Verzeichnis neben den zu transferierenden Dateien weitere Dateien enthält und somit nicht als Überordner verwendet werden kann. In diesem Fall wird der absolute Pfadname der jeweiligen Dateien zur Liste der zu übertragenden Verzeichnisse hinzugefügt(z.B. eine ArrayListe *groupedpaths*). Andernfalls wird das vorliegende Verzeichnis als Überordner zur Liste der zu übertragenden Verzeichnisse hinzugefügt. Für eine weitere Gruppierung der Verzeichnisse muss das Ganze eventuell rekursiv wiederholt werden.

Im **Worst Case** befinden sich alle Eingabedateien in unterschiedlichen Verzeichnissen, die nicht durch Überordner zusammengefasst werden können, so dass, dem Verhalten der ursprünglichen Transformationsregel entsprechend, für jede relevante mathematische Variable jeweils eine *data container reference variable* erzeugt werden muss.

Im **Idealfall** hingegen können alle relevanten Dateien mit Hilfe eines Überordners erfasst werden und es wird nur eine einzelne *data container reference variable* als Eingabeparameter für das *Data Transfer and Tranformation Pattern* erzeugt. Bei einer entsprechenden Optimierung der Transformation des *Data Transfer and Transformation Patterns* könnte dies dann auf ein einzelnes *Container-To-Container-Pattern* abgebildet werden (siehe Abschnitt 9.3).

Können statt allen relevanten Dateien nur einige der Dateien mit Hilfe von Überordnern gruppiert werden, wird für jede Gruppe und jede Datei, die nicht gruppiert werden konnte, jeweils eine *data container reference variable* erzeugt. Das *Data Transfer and Transformation Pattern* wird in eine Sequenz bestehend aus einer Assign-Aktivität InitContainerList und einem *sequentiellen Data Iteration Pattern*

abgebildet, wobei jedoch die Grösse seiner Iterationsmenge verkleinert wurde. Zur Umsetzung der oben beschriebenen Idee wird bei der Transformation das *Simulation Oriented Data Provisioning Pattern* weiterhin auf das ursprüngliche Workflow-Fragment abgebildet, welcher aus einem *Data Transfer and Transformation Pattern* besteht. Die Transformationsregel muss aber dahingehend angepasst werden, dass bei jeder Transformation die Methode ReduceCountContainers aufgerufen und die Anzahl der *data container reference variables* berechnet bzw. minimiert wird.

Eine weitere Optimierungsregel wäre die Umsetzung der in Abschnitt 8.9 besprochenen Idee, bei der das XML-Dokument der *data container reference list* mit *xmlstarlet* auf Kommadozeilenebene geparst wird. Hier müsste die Transformationsregel das Pattern auf eine Assign-Aktivität und eine IssueCommand-Aktivität abbilden. Statt für jede mathematische Variable eine *data container reference variable* zu erzeugen, würde die Assign-Aktivität direkt einen String erzeugen, der die Liste der *data container reference variables* als XML-Struktur erhält. In der IssueCommand-Aktivität könnte das XML-Dokument geparst und die notwendigen Kopieroperationen mengenbasiert ausgeführt werden. Diese Regel könnte in der Ausführungsreihenfolge der Regelsequenz nach der Regel mit den Überordnern eingeordnet werden, so dass sie nach einem unerfolgreichem Versuch des Findens eines einzelnen Überordners ausgeführt werden könnte. Denn es ist davon auszugehen, dass prinzipiell die Verwendung von Überordnern eine bessere Laufzeit liefert, als die xmlstarlet-Variante. Bei den erzielten Messergebnissen der Laufzeit (10), wurden momentan jedoch mit der xml-Starlet-Methode bessere Laufzeiten erzielt. Dies ist vermutlich auf die Implementierung der TransferData-Aktivität zurückzuführen.

9.3 Das Data Transfer and Transformation Pattern

Das **Data Transfer and Transformation Pattern** beschreibt ein Datenverarbeitungsmuster, bei dem Daten aus einer oder mehreren Datenquellen in eine oder mehrere Datensenken transferiert werden (Abbildung 9.3 [RS+13]). In der in Abschnitt Abschnitt 3.3 vorgestellten Hierarchie befindet sich dieses Pattern auf der Ebene der grundlegenden Datenmanagementpatterns. Der Datentransfer wird durch einen ETL-Prozess umgesetzt, sodass z.B. Filterungs-, Aggregations-, Datenformatkonvertierungs-, Vereinigungs- oder Verbund-Operationen erfolgen können. Die folgenden Betrachtungen beziehen sich auf die im SIMPL-Rahmenwerk vorliegende Implementierung, welche für die Verwendung in den Knochensimulationsworkflow ausgelegt ist. Es stellt ein Datenverarbeitungsmuster dar, bei dem Daten aus einer oder mehreren Datencontainern in einer oder mehreren Datenquellen auf genau einen Datencontainer in einer Datensenke kopiert werden.

Die Transformation des Pattern wurde bereits in Abschnitt 9.2 beschrieben. Im Hinblick auf das Laufzeitverhalten ist im resultierenden Worklow-Fragment das *sequentielle Data Iteration Pattern* von Interesse. Eine Optimierung wäre hier in dem Sinne möglich, dass die Anzahl der*data container reference variables* bei der Transformation des *Data Transformation and Transfer Patterns* so weit wie möglich reduziert wird. Dadurch verkleinert sich die Iterationsliste des *sequentiellen Data Iteration Patterns* und es werden letztendlich weniger Schleifendurchläufe ausgeführt.

Dies kann, wie auch in Abschnitt 9.2, durch die **Gruppierung** von Pfadnamen mit Hilfe von Überordnern erreicht werden. Dazu kann durch die Regelsequenz eine der in Abschnitt 9.2 beschriebenen ähnliche Methode *ReduceCountContainers*, aufgerufen werden. Für die Ermittlung der möglichen

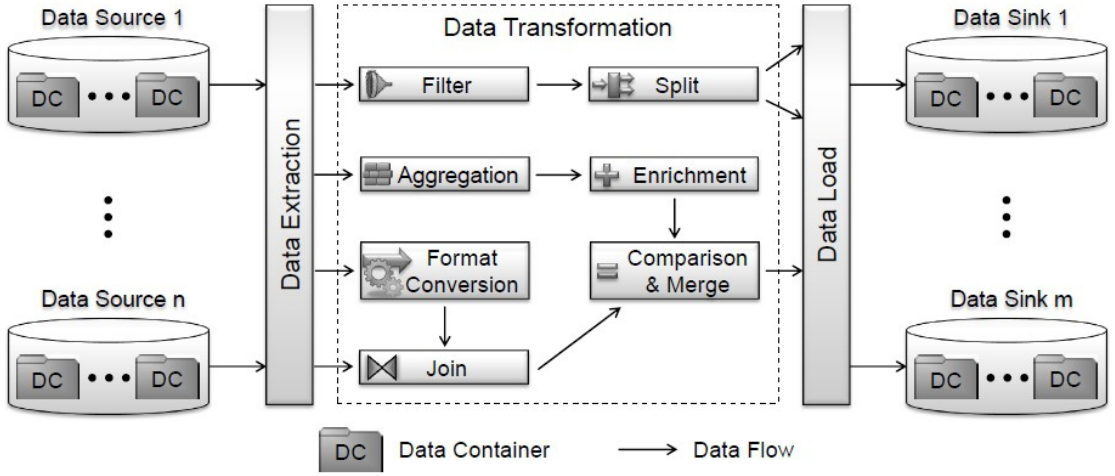


Abbildung 9.3: Das Data Transfer and Transformation Pattern [RS+13]

Überordner muss hier statt auf Instanzen von mathematischen Variablen und die in der Ontologie des entsprechenden Simulationsmodells zu diesen Instanzen hinterlegten Informationen, auf die Informationen der als Parameter des Patterns übergebenen *data container reference variables* zurückgegriffen werden. Dabei muss jeweils zur Ermittlung des Dateipfades einer zu transferierenden Eingabedatei das Element *directory* der Datencontainerreferenz selektiert werden. Für diese Dateipfade wird anschliessend versucht, Überordner zu ermitteln, die ausschliesslich zu übertragende Dateien enthalten.

Im **Idealfall** kann die Anzahl der *data container reference variables* auf eine *data container reference variable* reduziert werden und das *Data Transformation and Transfer Pattern* kann durch die Transformationsregel auf ein einzelnes *Container-To-Container-Pattern* abgebildet werden, da eine Iteration nicht notwendig ist.

Im **Worst Case** befinden sich alle Eingabedateien in unterschiedlichen Verzeichnissen, die nicht durch Überordner zusammengefasst werden können, sodass bei der Patterntransformation eine *data container reference list* erzeugt werden muss, welche für alle relevanten Dateien jeweils eine *data container reference variable* enthält. Dies entspricht dem Verhalten der ursprünglichen Transformationsregel des *Data Transfer and Transformation Patterns*.

Ist das *Data Transfer and Transformation Pattern* eingebettet in ein Workflow-Fragment, welches aus der Transformation eines in der Patternhierarchie auf einer höheren Ebene liegenden Datenmanagementpatterns hervorgegangen ist (wie z.B. *Simulation Oriented Data Provisioning Pattern*), so muss die Berechnung von Überordnern nicht erneut erfolgen, da dies bereits bei der vohergehenden Transformation stattgefunden hat. Dies könnte z.B. durch einen bei der Transformation übergebenen Flag *optimized* erreicht werden. Somit kann die aufgerufene Methode zunächst prüfen, ob das Flag gesetzt ist und bei gesetzten Flag den Optimierungsversuch abbrechen, ansonsten den Optimierungsversuch ausführen. Es müsste untersucht werden, ob dies auf andere Weise realisiert werden kann/sollte.

Zur Umsetzung der oben beschriebenen Idee muss zusätzlich zur ursprünglichen Transformationsregel des *Data Transfer and Transformation Patterns* eine neue Transformationsregel implementiert und vor der ursprünglichen Regel in die Regelsequenz eingefügt werden.

- Falls das Pattern als Eingabeparameter nur eine einzelne *data container reference variable* erhält bzw. durch Aufruf der Methode ReduceCountContainers ein **einzelner Überordner** berechnet wird, so wird das Pattern durch die neue Transformationsregel in ein einzelnes *Container-To-Container-Pattern* abgebildet, welche mit Hilfe des berechneten Überordners alle relevanten Dateien ins Zielverzeichnis überträgt.
- Für den Fall, bei dem mehrere *data container reference variables* als Eingabeparameter übergeben wurden bzw. durch den Aufruf der Methode ReduceCountContainers **mehrere Überordner** berechnet werden, muss die Regelsequenz die ursprüngliche Transformationsregel des Patterns ausführen, welche das Patterns in eine Sequenz aus Assign-Aktivität und *sequentiellem Data Iteration Pattern* mit eingebettetem *Container-To-Container-Pattern* abbildet. Das *sequentielle Data Iteration Pattern* iteriert dabei auf den einzelnen *data container reference variables*.

Auch hier könnte als zweite Optimierungsregel nach der Regel, welche Überordner verwendet, die xmlstarlet-Optimierung eingeordnet werden, sodass das Pattern auf eine Assign-Aktivität und eine IssueCommand-Aktivität abgebildet wird. Statt einer Variable des Typs *tdataContainerReferenceList* einen Wert zuzuweisen, würde die Assign-Aktivität hier einen String-Wert, der die Liste der *data container reference variables* als XML-Struktur enthält und diesen einer Variablen des Typs String zuweisen.

9.4 Das sequentielle Data Iteration Pattern mit eingebettetem TransferData

Das **sequentielle Data Iteration Pattern** erwartet als Eingabeparameter eine Liste mit *data container reference variables* sowie den Namen einer *data container reference variable*, die bei der Transformation des Patterns lokal erzeugt werden soll. Zudem kann der Modellierer einen Namen angeben, der dem Zähler der ForEach-Schleife zugewiesen werden soll. Diie Transformation des Patterns wurde in 9.2 bereits beschrieben und ist in 9.2 für den biomechanischen Workflow gezeigt.

Legt man die Konstellation der Aktivitäten zu Grunde, bietet sich hier die Anwendung der **Element-To-Set-Regel** an, um die laufzeitaufwendige Iteration durch einen mengenbasierten Befehl zu ersetzen. Dieses Szenario wurde bereits auf der Ebene der ausführbaren Workflows in Abschnitt 8.9 erörtert. Die Idee der *Element-To-Set-Regel* kann schon während der Patterntransformation umgesetzt werden. Dazu kann durch die Regel eine der in Abschnitt 9.2 beschriebenen ähnliche Methode *ReduceCount-Containers*, aufgerufen werden. Für die Ermittlung der möglichen Überordner muss hier statt auf mathematische Variablen und die Information einer Ontologie auf die als Parameter des Patterns übergebene *data container reference list* zurückgegriffen werden. Dabei muss jeweils zur Ermittlung des Dateipfades einer zu transferierenden Datei das Element *directory* der entsprechenden *data container reference variable* aus der *data container reference list* selektiert werden.

Zur Umsetzung der oben beschriebenen Idee muss zusätzlich zur ursprünglichen Transformationsregel des *sequentiellen Data Iteration Patterns* eine neue Transformationsregel implementiert und in die Regelsequenz der Ebene der *Basic Data Management Patterns*, der das *sequentielle Data Iteration Pattern* zugeordnet ist, eingefügt werden:

- Falls das Pattern als Eingabeparameter eine *data container reference list* mit nur einer einzelnen *data container reference variable* erhält bzw. durch Aufruf der Methode *ReduceCountContainers* ein **einzelner Überordner** berechnet wird, so wird das Pattern durch die neue Transformationsregel in eine einzelne TransferData-Aktivität abgebildet, welche mit Hilfe des berechneten Überordners alle relevanten Dateien ins Zielverzeichnis überträgt.
- Für den Fall, bei dem eine *data container reference list* mit mehreren *data container reference variables* als Eingabeparameter übergeben wurde bzw. durch den Aufruf der Methode ReduceCountContainers **mehrere Überordner** berechnet werden, muss die Regelsequenz die ursprüngliche Transformationsregel des Patterns ausführen, welche das Pattern auf eine Sequenz aus Assign-Aktivität und For-Each-Aktivität mit eingebettetem TransferData abbildet. Die For-Each-Aktivität iteriert dabei auf den einzelnen *data container reference variables* der Liste.

9.5 Das komplexe Container-To-Container-Pattern

Das **komplexe Container-to-Container Pattern** wird verwendet, wenn neben dem Datentransfer von Verzeichnis zu Verzeichnis, zusätzlich eine Datenformatkonvertierung stattfinden muss. Die im SIMPL-Rahmenwerk vorliegende Implementierung wird auf einen Scope abgebildet, der aus einer Sequenz besteht, die wiederum eine Assign Aktivität *InitIntermediateContainer*, ein *einfaches Container-to-Container Pattern* sowie ein *Data Format Conversion Pattern* umfasst. Somit erfolgt zuerst der Transfer der referenzierten Datei bzw. des referenzierten Objekts und anschließend die Konvertierung in ein anderes Datenformat.

Die Transformation des *komplexen Container-To-Container-Patterns* ist in Abbildung 9.4. *InitIntermediateContainer* erzeugt zwei *data container reference variables intermedateDirectory* und *intermediateContainer* des Typs *FileSystemDataContainerReferenceType*. Das *einfache Container-To-Container-Pattern* kopiert die relevante Datei in das durch die erste *data container reference variable intermediateDirectory* spezifizierte Verzeichnis und das anschliessende *Data Format Conversion Pattern* referenziert mit Hilfe der zweiten *data container reference variable intermediateContainer* die zu konvertiertende Datei, die durch das *einfache Container-To- Container Pattern* zuvor in das Zielverzeichnis kopiert wurde.

Eine Optimierung wäre hier möglich, indem die Funktion des *einfachen Container-To-Container-Patterns*, sowie des *Data Format Conversion Patterns* verschmolzen wird. Dies würde prinzipiell der Anwendung der **Activity-Merging-Regel** bzw. dessen Unterklasse der*Update-Merging-Regel* gleichkommen. Bei einer solchen Verschmelzung muss sichergestellt sein, dass dieselbe Semantik abgedeckt wird, wie durch die ursprünglich sequentielle Ausführung der Operationen.

Für die Verschmelzung der beiden Patterns müsste also eine geeignete Aktivität gefunden werden, welche die zu übertragende Datei direkt in das Zielverzeichnis kopieren und gleichzeitig eine Datenformatskonvertierung durchführen kann. In diesem Fall könnte das *Data Transfer and Transformation*

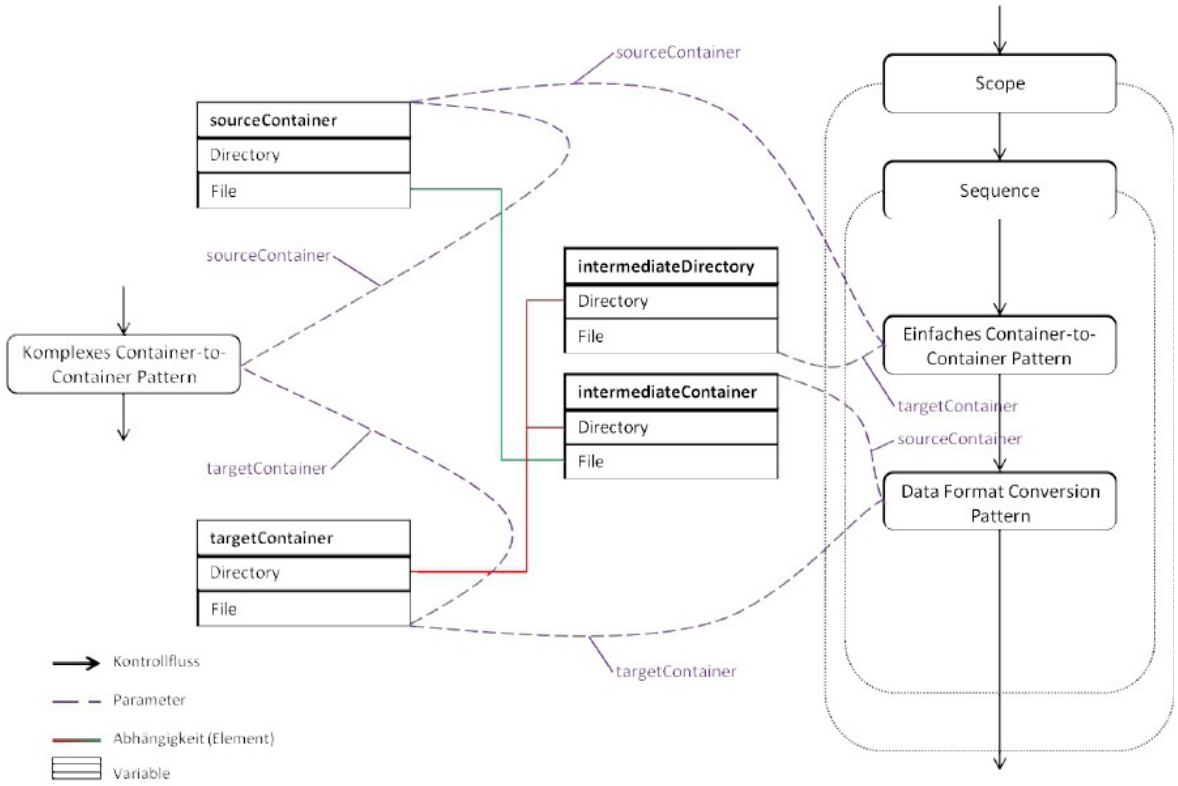


Abbildung 9.4: Transformation des komplexen Container-To-Container Patterns [Pie12]

Pattern auf eine einzelne IssueCommand-Aktivität abgebildet werden. Die Assign-Aktivität, die zu Beginn zwei lokale *data container reference variables intermediateDirectory* und *intermediateContainer* erzeugt, würde überflüssig werden. Denn *intermediateDirectory* wird erzeugt, um die relevante Datei mit Hilfe des *Container-To-Container-Patterns* auf das Zielverzeichnis zu kopieren, *intermediateContainer* hingegen wird erzeugt, um die Datei in das entsprechende Format zu konvertieren.

Findet der Dateitransfer innerhalb desselben, lokalen Dateisystems statt, so kann die oben beschriebene Idee einfach umgesetzt werden. Wird die relevante Datei auf ein entferntes Dateisystem übertragen, z.B. von einem lokalen auf ein entferntes Unix-System, so muss das Skript, welches die Datenformatskonvertierung ausführen soll, *remote* aufgerufen werden. Unter Verwendung des Tools *sshpass* kann die relevante Datei mit einer nicht-interaktiven Authorisation auf das entfernte Dateisystem übertragen und das Skript über das Tool *ssh remote* aufgerufen werden. Als Eingabeparameter erhält das Skript den Dateinamen. Ein möglicher Code-Ansatz für den DM command der IssueCommand-Aktivität ist in Listing 9.3 gezeigt. Der Name des Skripts sowie der Datei- und der Pfadname des Zielverzeichnisses müssten aus den jeweiligen Parametern des *komplexen Container-To-Container-Patterns* bestimmt und bei der Patterntransformation entsprechend ersetzt werden. Durch diese Umsetzung wird mit einer IssueCommand-Aktivität die Funktion des gesamten, ursprünglichen Workflow-Fragments erfüllt. Dieser Optimierungsvorschlag wird im Hinblick auf die Laufzeit des entsprechenden Workflows

Listing 9.3 Ansatz für DM command, der eine Datei auf ein entferntes Unix-System kopiert und ein Skript zur Datenformatskonvertierung aufruft

```
C:\cygwin\bin\sshpass.exe -p "openstack" SCP E:\vis\in\?filename
    openstack@192.168.209.210:?testdirectory
&&
SSH openstack@192.168.209.210:?testdirectory ?script ?filename
```

keinen grossen Einfluss haben. Durch die Vereinfachung des Workflows wird jedoch die Grundlage für die Anwendung weiterer Optimierungsregeln gelegt.

Eine weitere Möglichkeit einer Optimierung, unabhängig von den heuristischen Optimierungsregeln, besteht bei diesem Pattern darin, die Reihenfolge der Ausführung der Aktivitäten zu ändern. Eine Datenformatkonvertierung vor dem Datentransfer macht dann Sinn, wenn das neue Datenformat eine geringere Datengröße aufweist als das ursprüngliche Datenformat und somit das zu übertragende Datenvolumen geringer ist. Eine solche Optimierung würde jedoch erfordern, dass vor der Transformation des komplexen *Container-To-Container-Patterns* in ein ausführbares Workflow-Fragment fallspezifisch geprüft wird, welches der beiden relevanten Datenformate die geringere Datengrösse besitzt und somit besser für den Datentransfer geeignet ist. Es müssten in der Regelsequenz zwei Abbildungsregeln angelegt werden, wobei eine Regel auf ein Workflow-Fragment abbildet, bei der die Datenkonvertierung vor dem Datentransfer stattfindet bzw. andersherum. Bei dieser Optimierung handelt es sich im Gegensatz zu den erarbeiteten Optimierungsregeln nicht um eine heuristische, sondern um eine kostenbasierte Optimierung.

9.6 Der Kopplungsworkflow

In diesem Abschnitt soll der in Abschnitt 4.1 beschriebene Kopplungsworkflow auf Optimierungsmöglichkeiten, insbesondere in Bezug auf ihre Datenmanagementpatterns hin untersucht werden. Dieser Workflow wurde in [RSM14b] in eine konkrete Pattern-Hierarchie eingebettet(siehe Abbildung 9.5), welche die verwendeten Datenmanagementpatterns und ihre Abbildungen von Hierarchieebene zu Hierarchiebene bis hin zu ausführbaren Workflow-Fragmenten zeigt.

Auf der obersten Ebene der *Simulation Specific Process Patterns* befinden sich ein *Simulation Model Coupling Pattern*, zwei *Simulation Model Realization Patterns* und ein *Simulation Result Interpretation Pattern*. Durch das *Simulation Model Coupling Pattern* wird das übergeordnete Simulationsmodell spezifiziert und es wird festgelegt, wie zwei Simulationsmodelle miteinander verknüpft sind. Die *Simulation Model Realization Patterns* repräsentieren konkrete Simulationsmodelle, die realisiert werden sollen. Mit Hilfe des *Simulation Result Interpretation Patterns* wird spezifiziert, wie die Ergebnisse visualisiert werden sollen.

Mit einem Transformationsschritt werden die drei Patterns dieser Ebene (das Simulation Model Coupling Pattern und die beiden Simulation Model Realization Patterns) auf ein *Parameter Sweep* mit eingebetteten *Parameter Sweep Pattern* und *Simulation Oriented Data Interoperability Pattern* der darunterliegenden Ebene der *Simulation Oriented Data Management Patterns* abgebildet.

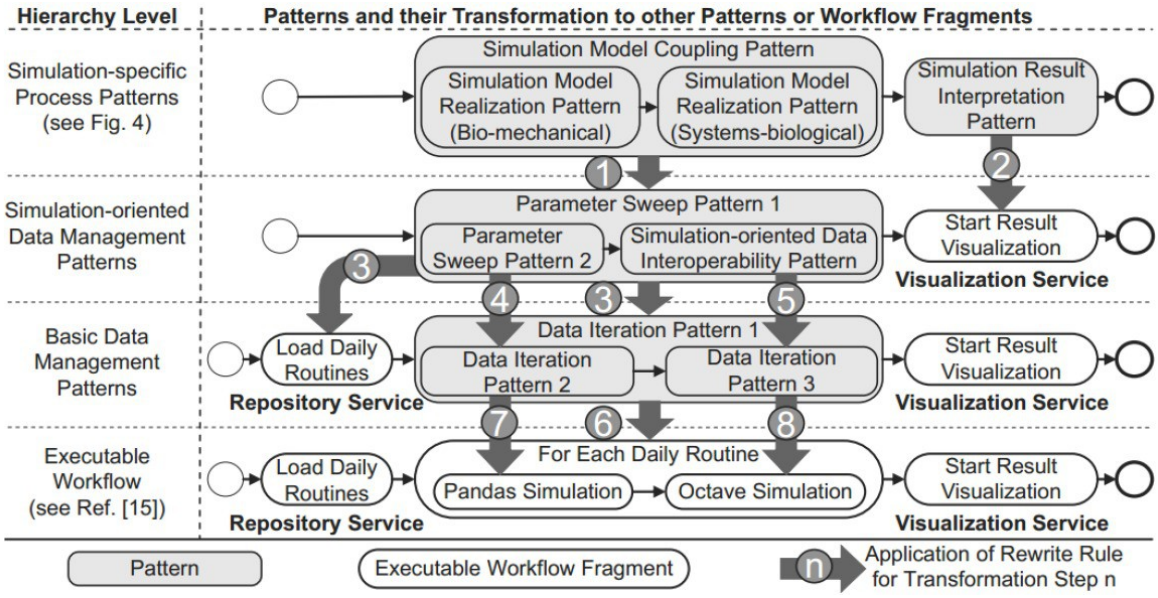


Abbildung 9.5: Der Kopplungsworkflow aus der Perspektive der Pattern-Hierarchie aus [RSM14b]

Das äussere *Parameter Sweep Pattern* implementiert die Verknüpfungsstrategie, die als Parameter des *Simulation Model Coupling Pattern* spezifiziert wurde. Es iteriert sequentiell auf der Liste der Tagesroutinen und führt für jeden Tag die in ihr eingebetteten zwei Patterns aus. Das zweite *Parameter Sweep Pattern* iteriert parallel auf der Liste der Bewegungssequenzen für den aktuellen Tag und führt in jedem Iterationsschritt einen Service aus, der das biomechanische *Simulation Model Realization Pattern* implementiert.

Das anschliessende *Simulation-oriented Data Interoperability Pattern* stellt eine Abstraktion der Verknüpfung des biomechanischen und des systembiologischen Simulationsmodells dar, indem es die Relationen zwischen ihren mathematischen Variablen definiert. Das *Simulation Result Interpretation Pattern* wird im zweiten Transformationsssschritt, ohne die gesamte Patternhierarchie zu durchlaufen, direkt in ein ausführbares Workflow-Fragment transformiert. Dabei handelt es sich um einen passenden, Service der zur Visualisierung der Resultate aufgerufen wird.

Die Patterns dieser Ebene werden nun in drei Transformationsschritten auf *Basic Data Management Patterns* der nächsttieferen Hierarchieebene abgebildet. Das äußere *Parameter Sweep Pattern* wird zunächst in ein *Sequentielles Data Iteration Pattern* transformiert. Anschließend wird das innere *Parameter Sweep Pattern* in ein *Paralleles Data Iteration Pattern* umgewandelt. Das *Simulation-oriented Data Interoperability Pattern* wird schliesslich in ein weiteres *Paralleles Data Iteration Pattern* überführt.

Nach den Transformationsschritten 3 bis 5 besteht der Workflow nur noch aus 3 *Basic Data Management Patterns* und zwei Web-Service-Aufrufen. Die Patterns der *Basic Data Management Patterns* Ebene werden in drei Transformationsschritten in ausführbare Workflow-Fragmente transformiert. Als Resultat der Transformationen entsteht der Workflow, der in Abschnitt 4.1 beschrieben wurde und in Abbildung 4.2 dargestellt wird. Der in dieser Abbildung gezeigte Schritt 1 des Workflows wird

in der Abbildung 9.5 durch das Workflow-Fragment Pandas-Simulation und die Schritte 2, 3, 4 und 5 durch das Workflow-Fragment Octave-Simulation repräsentiert.

Ein Optimierungsgedanke wäre beispielsweise die Anwendung der **Eliminate-Temporary-Container-Regel** in der *Data Merge-Phase* der Octave-Simulation. Für den Fall, dass diese Regel in dieser Phase des Workflows angewendet werden kann, können die Ausgabedaten der jeweiligen Octave-Simulation direkt in die Pandas-Input-Datenbank geschrieben werden, sodass der Aufwand für die Erzeugung, die Verwaltung und das Löschen der in Abbildung 4.2 gezeigten zusätzlichen temporären Datenbanktabelle (Temporary Table) vermieden werden kann. Die Idee der *Eliminate-Temporary-Container-Regel* kann bereits bei der Transformation des *Parallelen Data Iteration Patterns* auf das entsprechende Workflow-Fragment berücksichtigt werden, sodass fallspezifisch auf ein Workflow-Fragment mit oder ohne temporäre Tabelle abgebildet wird.

Eine andere Optimierungsidee ist es, mit der **Element-To-Set-Regel** die Schleife aufzulösen, welche in der *Phase Data Split for Octave Simulation* über der Menge der Octave-Rechner iteriert und für jeden Rechner mit Hilfe von Limit und Offset-Werten sowie dem PostgreSQL-Befehl COPY relevante Zeilen aus der Pandas-Datenbank jeweils in eine CSV-Datei exportiert. Ein SQL-Befehl, der gleichzeitig alle relevanten Zeilen einer Tabelle auf verschiedene CSV-Dateien kopiert, konnte im Rahmen dieser Diplomarbeit jedoch nicht gefunden werden. In [Vrh11] wird für Schleifen, die nicht aufgelöst werden können, vorgeschlagen, diese mittels einer Stored-Procedure (**Stored Procedure Pushdown**) zu definieren, um die Laufzeit und Datetransfer- und -Materialisierungkosten zu reduzieren. Hierzu müsste eine Funktion definiert werden, welche auf der Datenebene ausgeführt wird und eindeutige Dateinamen für die CSV-Dateien berechnet, die Schematransformationen vom Pandas- in das Octave-Format durchführt und schleifenbasiert alle Export-Befehle erledigt. Dadurch könnte das *parallele Data Iteration Pattern* auf ein Worklflow-Fragment abgebildet werden, das anstelle einer Schleife über den Octave-Rechnern eine einzelne, mengenbasierte IssueCommand-Aktivität enthält, welche wiederum die Stored-Procedure aufruft. Voraussetzung für diese Optimierungsregel ist die Definierbarkeit einer geeigneten Stored Procedure, was z.B. vom zu Grunde liegenden Datenbanksystem abhängt. Alternativ könnte in der mengenbasierten IssueCommand-Aktivität der PostgreSQL-Befehl COPY TO PROGRAM [1] verwendet werden, um den relevanten Inhalt aus der SQL-Datanbank an ein Skript zu übergeben, der dann on-the-fly (z.B. mit *awk*) unterschiedliche Dateinamen für die csv-Ausgabedateien berechnet.

Eine letzte Optimierungsidee befasst sich mit der der Berechnung der Anzahl der Gitterelemente sowie der Gausspunkte pro Element in der Phase *Preparation of Data Split for Octave*. Hier können die relevanten Tabellenzeilen für einen bestimmten Zeitschritt und eine bestimmte SimulationsID aus der Pandas-Datenbanktabelle z.B. in einen XML-RowSet geladen und anschliessend eine Berechnungsfunktion auf diesen Zeilen ausgeführt werden ([Pie11]). Alternativ kann die Berechnung der Werte direkt auf der Pandas-Datenbanktabelle ausgeführt wird [RSM14b]. Dies kommt der Anwendung einer **Predicate-Pushdown-Regel** gleich, so dass das entsprechende *parallele Data Iteration Pattern* auf ein Workflow-Fragment abgebildet werden kann, bei der die Berechnung der Gitterelemente und Gausspunkte je Gitterelement in einem Schritt durchgeführt wird.

[1] http://www.postgresql.org/docs/current/static/sql-copy.html

10 Evaluation

In diesem Kapitel werden einige der in Kapitel 7 vorgestellten und in Kapitel 8 beschriebenen Implementierungen der Optimierungsregeln für Simulationsworkflows anhand von Laufzeitmessungen evaluiert. In Abschnitt 10.1 wird zunächst die für die Laufzeitmessungen eingesetzte Testumgebung beschrieben. Anschliessend werden die verwendeten Testworkflows genannt. Zum Abschluss werden die für diese Testworkflows erzielten Messergebnisse präsentiert und diese kurz bewertet. Zum Schluss folgt in Abschnitt 10.4 eine kurzes Fazit zu den Ergebnissen.

10.1 Testumgebung

Alle Messungen wurden auf einem System ausgeführt, das eigens für die Evaluation zur Verfügung gestellt wurde. Es bestand aus einem Server mit zwei Sechs-Kern-Prozessoren Intel Xeon E5-2630 @2,3GHz, insgesamt 16 x 16 GiB RAM (256GB) Hauptspeicher und einem 10GB Ethernet Controller. Auf diesem Server wurden mit Hilfe von Openstack drei virtuelle Linux-Instanzen und virtuelle eine Windows-Instanz eingerichtet. Jede Instanz verfügte über vier physische, und somit acht virtuelle CPU Cores sowie 32 GB RAM. Als Betriebssystem waren auf den Linux-Instanzen Ubuntu 13.10 und auf der Windows-Instanz eine 64 Bit Version von Windows Server 2012 R2 Standard installiert. Auf der Windows-Instanz wurde als Simulationsworkflowmanagementsystem der SIMPL-Prototyp mit der Workflow-Engine Apache ODE 1.3.5 verwendet. Als Laufzeitumgebung kam Apache-Tomcat Version 7.0 zum Einsatz. Bei den in dieser Arbeit mit Linux- bzw. Windows-Rechner titulierten Rechnern handelte es sich um beschriebene Linux- bzw. Windows-Instanzen.

10.2 Testworkflows

Um die Wirkung der Optimierungsregeln zu analysieren, wurden ausgewählte Workflows mit den oben genannten Datenvolumina ausgeführt. Für die **Element-To-Set-Regel** wurde der Workflow aus Abschnitt 8.5, für die **Predicate-Pushdown-Regel** der Workflow aus Abschnitt 8.7 (2. Variante der Regel Option 1 und 2), für die **Delete-Delete-Merging-Regel** der Workflow aus Abschnitt 8.3, für die **Assign-Merging-Regel** der Workflow aus Abschnitt 8.2 und für die Eliminate-Temporary-Container-Regel die Workflows aus Abschnitt 8.4 und Abschnitt 8.9 verwendet. Die kombinierte Anwendung von **Predicate-Pushdown**- und **Element-To-Set-Regel** wurde anhand des Workflows aus Abschnitt 8.7 evaluiert. Für die Beispiel-Szenarien der systembiologischen und der biomechanischen Knochensimulation (vgl.Abschnitt 4.2) standen keine lauffähigen Implementierungen zur Verfügung. Um die Datenbereitstellungsphase dieser Workflows zu optimieren, wurde in Abschnitt 8.9 ein eigener Workflow entworfen und dieser für Evaluierungen eingesetzt. Das Szenario der Proteinmodellierung

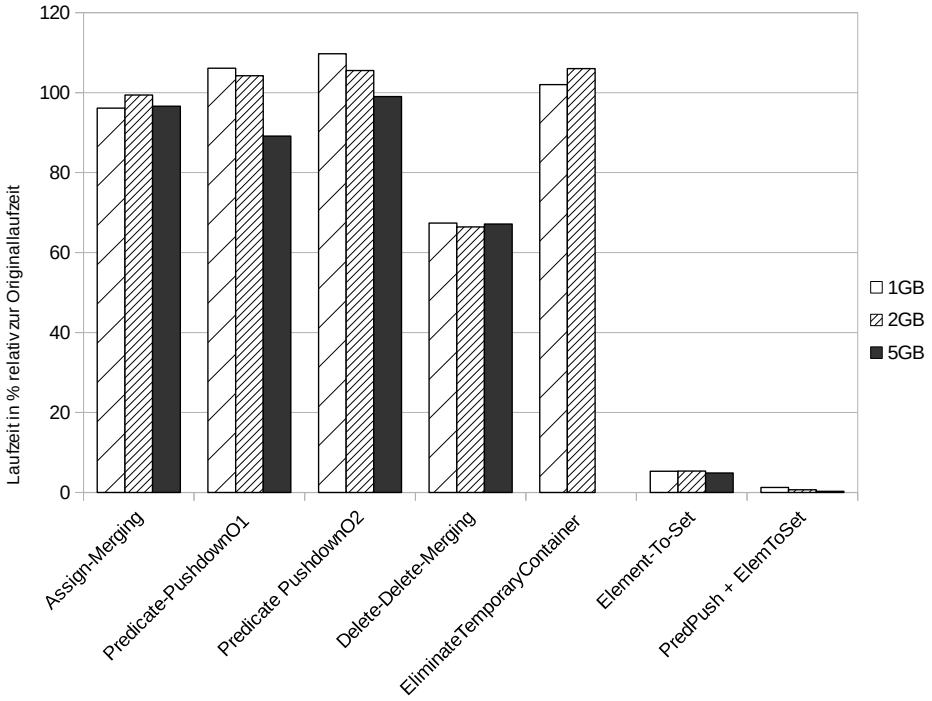


Abbildung 10.1: Messergebnisse der verschiedenen Optimierungsregeln, im Vergleich zum entsprechenden, unoptimierten Workflow

(vgl. Abschnitt 4.3) hingegen, konnte aufgrund technischer Probleme bei der Anbindung von XML-Datenbanken bei den Evaluierungen nicht berücksichtigt werden. Hier wurde nur konzeptionell ein Testworkflow und Optimierungen dafür entworfen.

10.3 Messergebnisse

In diesem Abschnitt werden die Messergebnisse der einzelnen Szenarien vorgestellt. Zu beachten ist, dass Messergebnisse generell von den Eigenheiten der in den Workflows verwendeten Datenressourcen (Dateisysteme, XML-Datenbanken etc.) abhängen und somit nur Hinweise dafür liefern, welche Optimierungsgewinne durch die Regelanwendungen erzielt werden können. Die durschnittlichen Laufzeiten der optimierten Workflows werden in Prozent relativ zur durchschnittlichen Laufzeit des entsprechenden, unoptimierten Workflows angegeben.

Die Messergebisse der optimierten Workflows sind in Abbildung 10.1 dargestellt. Bei der *Assign-Merging-Regel* konnten nur geringe Leistungssteigerungen im Bereich von etwa 5 % erzielt werden, was sich damit begründen lässt, dass hier lediglich eine Variablenzuweisung wegfällt, die zu Grunde liegende Datenressource jedoch immer noch dieselbe Anweisung ausführt. Die Einsparung einer Assign-Aktivität auf Workflowebene fällt kaum ins Gewicht. Diese Regel trägt zwar nicht zu grossen

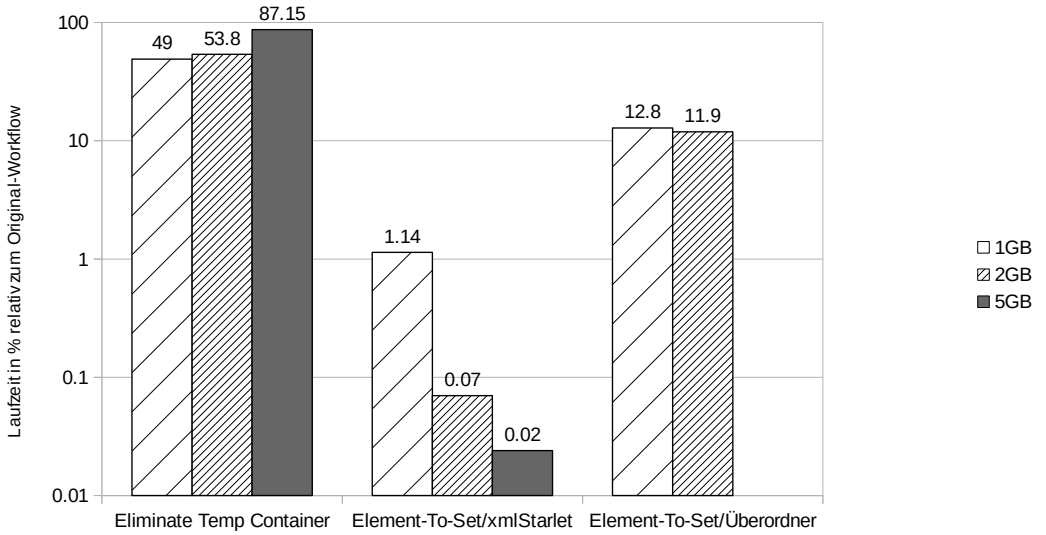


Abbildung 10.2: Resultate der Optimierungen des Testworkflows für die Datenbereitstellung im biomechanischen/systembiologischen Workflow

Laufzeitgewinnen bei, bereitet aber gegebenenfalls die Anwendung weiterer Optimierungsregeln vor.

Bei der *zweiten Variante der Predicate-Pushdown-Regel*, sowohl bei Option 1 als auch Option 2, wurden bei Datenvolumina von 1 GB und 2 GB jedoch zunächst eine Verschlechterung der Laufzeit im Bereich von bis zu 7 % festgestellt. Bei steigenden Datenvolumina jedoch konnten insbesondere bei der Option 1 Laufzeitverbesserungen bis zu 11 % erreicht werden. Bei der Option 1 wurde bei 5GB lediglich eine Verbesserung um 1% erzielt. Dies ist darauf zurückzuführen, dass es sich hierbei um die zweite Variante der *Predicate-Pushdown-Regel* handelt, welche durch Entfernen eines If-Then-Else-Konstruktes lediglich eine Vereinfachung des Workflows vornimmt. Mit der *Standard-Predicate-Pushdown-Regel*, bei der in der Regel die zwischen Workflow- und Datenebene übertragenen Datenmengen reduziert werden, hätten vermutlich grössere Laufzeitverbesserungen erzielt werden können. Höhere Laufzeitgewinne konnten bei der *Update-Merging-Regel*, in diesem Fall der *Delete-Delete-Merging-Regel*, erzielt werden. Die Leistungssteigerungen lagen hier bei rund 33% gegenüber dem Original-Workflow. Dies kann damit begründet werden, dass nach der Regelanwendung nur eine umfangreiche anstatt wie vorher zwei Delete-Operation ausgeführt wurden. Bei der *Eliminate-Temporary-Container-Regel* wurde zunächst der Workflow aus Abschnitt 8.4 getestet. Hier konnten keine wesentlichen Verbesserungen der Laufzeit, teilweise sogar Verschlechterungen festgestellt werden. Dies ist darauf zurückzuführen, dass, obwohl das explizite Zwischenverzeichnis entfernt wurde, aufgrund der Implementierung der TransferData-Aktivität immer noch eine Zwischenspeicherung der zu kopierenden Dateien auf dem lokalen Rechner erfolgte. Die Messergebnisse der *Eliminate-Temporary-Container-Regel* für den Workflow aus Abschnitt 8.9 werden weiter unten besprochen. Die Element-To-Set-Regel lieferte

erwartungsgemäß Laufzeitverbesserungen bis zu 95%, was durch die Entfernung der Iteration begründet werden kann. Beim der Predicate-Pushdown- und Element-To-Set-*Regelkombination* wurde im Vergleich zum Original-Workflow eine fast 99%-ige Verbesserung bei 5GB Datenvolumen erzielt. Dies rechtfertigt die Implementierung der zweiten Variante der Predicate-Pushdown-Regel, welche einzeln kaum ins Gewicht fällt, hier jedoch die Grundlage für die Anwendung der *Element-To-Set-Regel* bildet.

In Abbildung 10.2 sind in einer logarithmischen Skala noch einmal die Laufzeitresultate für die Optimierungen des Testworkflows, der für die Datenbereitstellung in der biomechanischen bzw. systembiologischen Simulation entworfen wurde, gezeigt. Dabei sind die Laufzeiten wieder in Relation zum Originalworkflow (100%) angegeben. Der erste, optimierte Workflow unterscheidet sich vom Original-Workflow nur darin, dass er anstelle einer TransferData-Aktivität eine IssueCommand-Aktivität verwendet. Dies kommt der Anwendung der **Eliminate-Temporary-Container** gleich, da keine Zwischenspeicherung mehr stattfindet. Der zweite, optimierte Workflow setzt, wie in 8.9 beschrieben, die **Element-To-Set-Regel mit xmlstarlet** um. Beim dritten, optimierten Workflow wurde die **Element-To-Set-Regel mit einem Überordner** umgesetzt (mit Hilfe einer TransferData-Aktivität).

Die beste Laufzeit wurde mit xmlstarlet erzielt. Der mit einem Überordner optimierte Workflow war hier zwar langsamer, dies ist jedoch auf die Implementierung der TransferData-Aktivität zurückzuführen. Denn prinzipiell ist die Angabe eines Verzeichnisnamens die schnellste Methode um Dateien zu übertragen und bei der xmlStarlet-Methode findet zusätzlich ein Parsing statt, welches bei XML-Strukturen mit enorm vielen Knoten zu entsprechende langen Laufzeiten führen würde. Da der erste, optimierte Workflow immer noch über die Dateien iteriert, ist seine Laufzeit im Vergleich zu den beiden anderen, optimierten Workflow sehr hoch. Im Vergleich zum Original-Workflow wurde dennoch eine Laufzeitverbesserung von bis zu 50% erzielt.

10.4 Fazit

Insgesamt konnten durch die Optimierungen ähnliche Ergebnisse erzielt werden, wie in [Vrh11]. Auch hier bewirkte die Vermeidung von Schleifenkonstrukten, in diesem Fall durch die *Element-To-Set-Regel*, die grössten Optimierungsgewinne. Zudem konnten durch die *Eliminate-Temporary-Container-Regel* (implizit durch Verwendung einer IssueCommand-Aktivität) und die *Delete-Delete-Merging-Regel* signifikante Laufzeitverbesserungen erzielt werden. Erneut den Ergebnissen in [Vrh11] entsprechend, waren die Laufzeitsteigerungen bei der *Assign-Merging-Regel* nicht hoch. Bei der zweiten Variante der *Predicate-Pushdown-Regeln*, sowohl bei Option 1 als auch Option 2, sowie der *Eliminate-Temporary-Container-Regel* (Entfernen eines expliziten Zwischenordners) wurden sogar minimale Laufzeitverschlechterungen gemessen. Wie am Beispiel der Kombination von *Predicate-Pushdown* und *Element-To-Set-Regel* gezeigt werden konnte, sind diese Regeln jedoch als Enabler weiterer Optimierungsregeln notwendig.

Zusammenfassend kann aufgrund der Messergebnisse gesagt werden, dass die Erarbeitung und Anwendung von Optimierungsregeln im Rahmen dieser Diplomarbeit durchaus sinnvoll war. Die oben getesteten Optimierungsregeln und Anwendungsszenarien stellen, im Hinblick auf die verschiedensten Datenressourcen und Befehlssprachen, die bei Simulationsworkflows zum Einsatz kommen

können, nur einen kleinen Ausschnitt von den Optimierungsmöglichkeiten solcher Workflows dar. Die Messergebnisse geben jedoch einen guten Eindruck darüber, dass viele der in [Vrh11] vorgeschlagenen Ideen auch für Laufzeitoptimierungen in Simulationsworkflows genutzt werden können.

11 Optimierung des SIMPL-Rahmenwerks

In diesem Kapitel sollen kurz die generellen Optimierungsvorschläge für das SIMPL-Rahmenwerk aufgezählt werden, die sich insbesondere auf den Beobachtungen stützen, die bei der Erstellung und nach der Evaluation der Testworkflows gemacht wurden. Dabei werden in Abschnitt 11.1 zunächst Spracherweiterungen für das SIMPL-Rahmenwerk und anschliessend in Abschnitt 11.2 Verbesserungsvorschläge bzgl. der TransferData-Aktivität diskutiert.

11.1 Spracherweiterungen

Bei der Erstellung der Testworkflows und der Anwendung der in Abschnitt 7.2 abgeleiteten Optimierungsregeln ergaben sich die grössten Schwierigkeiten durch die Verwendung von unterschiedlichen Programmier- bzw. Anfragsprachen innerhalb desselben Workflows. Dies wiederum ist zuruckzuführen auf die Verwendung unterschiedlicher Datenressourcen bzw. -formate im selben Workflow. Im Anwendungsbeispiel in Abschnitt 8.7 wird z.B. mit Hilfe eines XML-Dokuments, der *data container reference list*, der Dateitransfer von einem lokalen Windows-Dateisystem auf ein entferntes Unix-System gesteuert. Die Verarbeitung des XML-Dokuments erfordert dabei eine XML-Programmiersprache wie XPath und der Dateitransfer erfolgt mit Hilfe eines Windows Shell-Befehls. Die Kompatibilität der verwendeten Sprachen ist für die Anwendbarkeit der Optimierungsregeln von Bedeutung. Unter Kompatibilität wird hier die Möglichkeit verstanden, einen Befehl, der in einer Programmier- bzw. Befehlssprache formuliert ist, in eine weitere Programmier- bzw. Befehlssprache (um)formulieren zu können, sodass die Semantik der ursprünglichen Formulierung abgedeckt wird. Es stellte keine grosse Herausforderung dar, in einer beliebigen Sprache formulierte Anweisungen mit einer XML-Anfragesprache wie XQuery umzuformulieren. Dies ist zurückzuführen auf die grosse Ausdruckskraft der Sprache. Andersherum bereitete dies jedoch Probleme. Da die Optimierungsregeln Workflowfragemente restrukturieren und diese in der Regel auf dem Verschieben von Anweisungen zwischen Aktivitäten bzw. der Verschmelzung von Aktivitäten basieren, mussten häufig solche Umformulierungen zwischen Befehlssprachen erfolgen.

Bei Workflows im SIMPL-Rahmenwerk und BPEL-Prozessen im Allgemeinen herrscht eine grosse Dominanz der XML-Sprachen vor. Je stärker ausdrucksstarke XML-Sprachen unterstützt werden, desto grösser sind folglich die Möglichkeiten für Umstrukturierungen von SIMPL-Workflows. Apache ODE unterstützt offiziell XPath 2.0. Der Eclipse BPEL Designer arbeitet jedoch standardmässig mit XPath 1.0 und von einem Funktionieren aller XPath 2.0-Ausdrücke in Apache ODE konnte nicht mit Sicherheit ausgegangen werden. Deshalb wurde bei den Anwendungsbeispielen zum Grossteil mit XPath 1.0 gearbeitet. Dadurch ergaben sich häufig Probleme bei der Anwendung der Optimierungsregeln. Eine vollständige und fehlerlose Unterstützung von XPath 2.0 wird in dieser Hinsicht Erleichterungen bringen. Bei IF-Then-Else-Konstrukten in Assign-Aktivitäten z.B. erlaubte XPath

2.0 die Verwendung eines konventionelles If-Then-Else-Konstrukts, während in XPath 1.0 dies nur über Umwege realisierbar war. Auch die Tatsache, dass Assign-Aktivitäten Variablen nur einen einzelnen Wert zuweisen können, verursachte Probleme, z.B. wenn die durch einen XPath- oder XQuery-Ausdruck selektierte Knotenmenge zugewiesen werden sollte. Hier könnten beispielsweise BPEL Extension Assign Operations implementiert werden, welche komplexere Zuweisungen bei Optimierungen erlauben. Angesprochene Problemstellungen beziehen sich auf die Workflowsprache BPEL oder Apache ODE als BPEL-Engine. Eine Optimierung des SIMPL-Rahmenwerks könnte gegebenenfalls alleine schon durch die Verwendung einer anderen BPEL-Engine erreicht werden.

Ebenso könnten bei Dateisystemen Optimierungen vereinfacht werden, wenn BPEL-DM-Aktivitäten wie TransferData oder IssueCommand, eine Unterstützung für mächtigere Shell-Sprachen wie z.B. PowerShell anbieten würden.

11.2 TransferData-Aktivität

Des Weiteren stellte häufig die TransferData-Aktivität eine Herausforderung bei Optimierungsbemühungen dar. Dies ist darauf zurückzuführen, dass diese Aktivität nur *data container reference variables* und Pfadnamen als Befehle akzeptiert. Diese Aktivität könnte gegebenenfalls überarbeitet werden, sodass das Spektrum der von diesen Aktivitäten akzeptierten Befehle(z.B. die Definiton von Patterns wie *.csv oder *.*) bzw. Befehlssprachen erweitert wird, um so die Anwendung von Optimierungsregeln zu vereinfachen. So müsste eine TransferData-Aktivität nicht jedesmal in eine semantisch äquivalente IssueCommand-Aktivität umgewandelt werden.

Bei Implementierungen mit der TransferData-Aktivität, insbesondere bei Untersuchungen der Testausführungen des in dieser Arbeit erstellten biomechanischen Testworkflows fiel auf, dass eine einzelne TransferData-Aktivität bei Dateitransfers die zu kopierenden Dateien lokal zwischenspeichert, bevor sie sie auf das endgültige Zielverzeichnis überträgt und somit Laufzeitverluste generiert und nebenbei Speicherplatz auf dem lokalen Rechner belegt. Dies war sogar dann der Fall, wenn die Dateien von einem remote Rechner auf den lokalen Rechner übertragen wurden. D.h. die Dateien wurden auf dem Zielrechner zwischengespeichert, bevor sie in das Verzeichnis auf eben diesem selben Zielrechner transferiert wurden. Im Gegensatz dazu konnte dies bei Einsatz einer IssueCommand-Aktivität, welche die Dateien z.B. unter Verwendung von *sshpass* und *scp* überträgt, nicht festgestellt werden. Demzufolge ist es erstrebenswert, die TransferData-Aktivität auch in dieser Hinsicht der IssueCommand-Aktivität anzugleichen, sodass bei Dateitransfers keine impliziten Zwischenverzeichnisse verwendet werden und ein direkter Dateitransfer erfolgt.

12 Zusammenfassung und Ausblick

Datenintensive Workflows müssen in der Regel mit grossen Datenmengen umgehen. Für die Laufzeitleistungen solcher Workflows ist insbesondere die Leistung der Datenverarbeitungsoperationen massgebend. Ebenso sind Wissenschaftliche Workflows vom Umgang mit grossen Datenmengen geprägt. Existierende Optimierungsansätze für wissenschaftliche sowie generell datenintensive Worklfows versuchen mit Hilfe unterschiedlicher Verfahren das Laufzeitverhalten solcher Workflows zu optimieren. Simulationsworkflows, insbesondere jene Workflows, die mit Hilfe des SIMPL-Rahmenwerks realisiert wurden, haben als datentintensive Workflows ebenso mit Laufzeitproblemen zu kämpfen und eignen sich für die Anwendung von Optimierungsansätzen. Dementsprechend ist eine Übertragung existierender Ansätze auf diese Workflows angebracht.

Die Verwendung von Datenmanagementpatterns erleichtert Wissenschaftlern die Arbeit mit Simulationsworkflows, indem wiederkehrende Datenverarbeitungsmuster bei der Modellierung von Simulationsworkflows als Schablonen bzw. Bausteine bereitgestellt werden. Diese Bausteine umfassen mehrere feingranulare Arbeitsschritte von Simulationsworkflows und müssen lediglich parametrisiert und zusammengesetzt werden. Nach erfolgter Parametrisierung werden Datenmanagementpatterns durch Transformationsregeln in ausführbare Workflow-Fragmente überführt. Die Anwendung von Optimierungsregeln auf diesen Datenmanagementpatterns kommt der Optimierung typischer Datenverarbeitungsmustern gleich und ist ebenso Ziel der Optimierungsbemühungen.

In dieser Arbeit wurden Optimierungsmöglichkeiten für die Datenverarbeitung in Simulationsworkflows untersucht und einzelne Optimierungsregeln umgesetzt. Dazu wurden in Kapitel 5 zunächst verwandte Arbeiten zur Optimierung von datenintensiven Workflows näher betrachtet und in diesen nach möglichen Optimierungsansätzen für die Datenverarbeitung in Simulationsworkflows gesucht. In Kapitel Kapitel 6 folgte eine Bewertung der zuvor betrachteten Optimierungsansätze hinsichtlich ihrer Übertragbarkeit auf Simulationsworkflows, insbesondere auf die in dieser Arbeit betrachteten Beispielworkflows (Abschnitt 4.1, Abschnitt 4.2 und Abschnitt 4.3).

In Kapitel 7 wurden basierend auf dem Optimierungsansatz aus Abschnitt 5.1 Optimierungen für Simulationsworkflows abgeleitet. Dabei wurden die ursprünglichen Optimierungsregeln für die Anwendbarkeit auf Simulationsworkflows, insbesondere auf BPEL-DM-Workflows angepasst. Aufgrund der unterschiedlichen Datenressourcen, die BPEL-DM-Workflows zu Grunde liegen können, erforderte dies eine Abstrahierung der Original-Regeln. Insgesamt konnten hier aufgrund der Ähnlichkeit der Sprachen BPEL/SQL und BPEL-DM, sowie der Anwendungsgebiete sehr viele Parallelen gezogen und ähnliche Anwendungsszenarien eingegrenzt werden, auf denen die Optimierungsregeln Anwendung finden können. Ferner wurden, angelehnt an die ursprüngliche Variante der Predicate-Pushdown-Regel, neue Predicate-Pushdown-Regeln entworfen.

Für einige dieser Optimierungen wurden anschliessend in Kapitel 8 Anwendungsszenarien entworfen und die Anwendung der Optimierungen auf diese Szenarien, sowie die dadurch erzielten Resultate

beschrieben. Aufgrund technischer Komplikationen bei der Anbindung von XML-Datenbanken an das SIMPL-Rahmenwerk, bezogen sich die Betrachungen bzw. Implementierungen vermehrt auf Workflows mit Dateisystemen als zu Grunde liegende Datenquellen. So wurde z.B. keine Umsetzung der *Predicate-Pushdown-Regel* entworfen, bei der die von der Daten- auf die Workflowebene übertragene Datenmenge reduziert wird. Dies wäre theoretisch mit einer RetrieveData-Aktivität und einer entsprechend angepassten XQuery-Anweisung auf der XML-Datenbank einfach und intuitiv realisierbar gewesen. Bei der konzeptionellen Implementierung des Proteinmodellierungsworkflows, welche auf XML-Datenbanken arbeitet und bei der Arbeit mit XML-Dokumenten(wie z.B. *data container reference lists*) liess sich auch beobachten, dass sich die Umsetzung von Optimierungsregeln mit XQuery, aufgrund dessen Ausdrucksstärke und aufgrund seiner Verwandtheit mit SQL verhältnismässig einfach gestaltet. Die Umsetzung der in [Vrh11] vorgeschlagenen Regeln kann hier fast Eins zu Eins erfolgen. Im Gegensatz dazu war die Implementierung von Testworkflows mit Shell-Sprachen mit grösseren Herausforderungen verbunden. Insbesondere für die Datenbereitstellungsphase des biomechanischen/systembiologischen Workflows wurden in Abschnitt 8.9 zwei interessante Optimierungsmöglichkeiten entworfen, welche die Idee der *Tuple-To-Set-Regel* [Vrh11] umsetzen.

Im anschliessenden Kapitel 9 wurde die Verwendbarkeit der in den Optimierungsansätzen vorgeschlagenen Optimierungen als Optimierungsregeln bei der regelbasierten Abbildung von Datenmanagementpatterns auf ausführbare Workflowfragmente erörtert. Hierzu wurde in Abschnitt 9.1 ein Vergleich der Transformationsprozesse des PGM/F-Rahmenwerks und der Datenmanagementpatterns angestellt. Anschliessend wurden für konkrete Patterns Optimierungsmöglichkeitene diskutiert. Dabei wurden insbesondere die Transformationsketten des *Simulation Oriented Data Provisioning Patterns* und des Kopplungsworkflows (Abschnitt 4.1) näher untersucht.

In Kapitel 10 erfolgte anschliessend die Evaluierung der in Kapitel 7 hergeleiteten Optimierungsregeln für Simulationsworkflows hinsichtlich ihrer Optimierungsgewinne. Dazu wurden die Messergebnisse der Testworkflows verwendet, die in Kapitel 8 entworfen wurden. Diese wurden mehrfach, für verschiedene Datenvolumina ausgeführt und ihre Laufzeiten dokumentiert. Basierend auf dieser Evaluation und den Herausforderungen bei der Implementierung der Testworkflows in Kapitel 8 wurden anschliessend im Kapitel Kapitel 11 mögliche Verbesserungen der Implementierung des SIMPL-Rahmenwerks vorgeschlagen, die zu einer höheren Effizienz von Simulationsworkflows führen können.

Ausblick

In dieser Arbeit wurden Optimierungsregeln für Simulationsworkflows erarbeitet und ihre Anwendung mit Hilfe von Beispielworkflows getestet. Um die Optimierungsregeln für beliebige BPEL-DM Workflows ausnutzen zu können, könnte für diese in zukünftigen Arbeiten eine Automatisierung implementiert werden. Dies könnte ähnlich dem Transformationsprozess der Datenmanagementpatterns umgesetzt werden, indem in der Simtech View der Bedieneroberfläche des Eclipse BPEL Designers entsprechende Buttons bereitgestellt werden, die bei Anklicken durch den Benutzer, einen Optimierungsprozess starten. Dabei könnte ein Parser über den Workflowgraphen laufen und Datenverarbeitungsmuster ausfindig machen, die unoptimiert sind und einer Optimierung bedürfen.

Bei Vorliegen eines solchen unoptimierten Workflow-Fragments, könnte die entsprechende Optimierungsregel aus der Regelbasis der dazugehörigen Kontrollstrategie ausgeführt und als Resultat ein restrukturierter, optimierter Workflow zurückgeliefert werden.

Für den Proteinmodellierungsworkflow wurde aufgrund technischer Komplikationen bei der Anbindung von XML-Datenbanken an das SIMPL-Rahmenwerk nur konzeptionell ein rudimentärer Workflow und dafür Optimierungen entworfen. Dieser könnte in einer zukünftigen Arbeit ausimplementiert und evaluiert werden.

Zudem wurden für die Optimierungsregeln und insbesondere die Umsetzungen mit Testworkflows Dateisysteme und XML-Datenbanken betrachtet. In zukünftigen Arbeiten könnten diesbezüglich weitere Datenressourcen, z.B. aus dem NoSQL-Bereich untersucht werden.

Des Weiteren wurden konzeptionell Optimierungen für die Transformation von bestimmten Datenmanagementpatterns erarbeitet und vorgeschlagen, welche in Form von neuen Transformationsregeln in die Patterntransformation eingebunden werden könnten. Dazu müssten diese und weitere Regeln ausimplementiert und in die Regelsequenz der entsprechenden Datenmanagementpatterns hinzugefügt werden, so dass sie bei der automatisierten Ausführung der Patterntransformation zur Verfügung stehen. Somit könnte z.B. durch einen Klick des Nutzers die Transformation gestartet und zunächst eine Abbildung auf ein optimiertes Workflow-Fragment überprüft werden, andernfalls die Standardregel verwendet werden.

Zudem könnten, wie bereits in Kapitel 11 angesprochen, die Implementierungen einiger, vom SIMPL-Rahmenwerk bereitgestellter BPEL-DM-Aktivitäten wie z.B. die TransferData oder die Assign-Aktivität überarbeitet werden. Ziel dieser Überarbeitungen könnte es sein, das Spektrum der von diesen Aktivitäten akzeptierten Befehle bzw. Befehlssprachen zu erweitern bzw. komplexere Zuweisungsoperationen zu ermöglichen, um so die Anwendung von Optimierungsregeln zu vereinfachen. Die TransferData-Aktivität könnte auch dahingehend verbessert werden, dass sie wie eine IssueCommand-Aktivität Dateien ohne Zwischenspeicherung direkt kopiert.

Eine weitere, zukünftige Aufgabe wäre es, die in 8.9 beschriebene Idee, welche Überordner zur Gruppierung verwendet, um den Laufzeitaufwand von Schleifenkonstrukten zu vermeiden oder zumindest zu minimieren, auf eine nächsthöhere Hierarchieebene zu ziehen. So könnten mathematische Variablen gruppiert und die dem *Simulation Oriented Data Provisioning Pattern* übergebene Menge mathematischer Variablen reduziert werden.

Literaturverzeichnis

[AAD+07] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, et al. WS-BPEL extension for people (BPEL4People). *V1. 0*, 2007. (Zitiert auf Seite 28)

[ABFG04] D. Austin, A. Barbir, C. Ferris, S. Garg. Web services architecture requirements. *W3C Working Group Notes*, 2004. (Zitiert auf Seite 22)

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, 2003. (Zitiert auf Seite 31)

[AMA06] A. Akram, D. Meredith, R. Allan. Evaluation of BPEL to scientific workflows. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, Band 1, S. 269–274. IEEE, 2006. (Zitiert auf den Seiten 43 und 79)

[Ari12] S. Aristidou. *Abstraktionsunterstutzung fur die Definition des Datenmanagements in Simulationsworkflows*. Diplomarbeit, Stuttgart, Universitat Stuttgart, Diplomarbeit, 2012, 2012. (Zitiert auf den Seiten 37 und 52)

[BCF+06] S. Boag, D. Chamberlin, M. Fernndez, D. Florescu, J. Robie, J. Simon. XQuery 1.0: an XML query language W3C candidate recommendation, 3 November (2005). *Dostupné na: http://www. w3. org/TR/xquery*, 2006. (Zitiert auf Seite 17)

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard. Web services architecture. 2004. (Zitiert auf Seite 22)

[Boh] A. M. Bohrn. *Pattern-basierte Definition der Datenbereitstellung für Simulationen zu Strukturänderungen in Knochen*. Bachelor-arbeit. (Zitiert auf den Seiten 6, 11, 54, 56 und 57)

[BPSM+98] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 1998. (Zitiert auf Seite 13)

[BTS] J. Berg, J. Tymoczko, L. Stryer. Biochemistry. Sixth. 2007. (Zitiert auf den Seiten 11 und 57)

[C+] W. W. W. Consortium, et al. XML Path Language (XPath), Version 1.0. W3C Recommendation, 16 November 1999. (Zitiert auf Seite 18)

[C+07] W. W. W. Consortium, et al. Xsl transformations (xslt) version 2.0. 2007. (Zitiert auf den Seiten 28 und 114)

[CD+] J. Clark, S. DeRose, et al. XML path language (XPath) version 1.0. W3C recommendation, 1999. *World Wide Web Consortium, http://w3c. org/TR/xpath*. (Zitiert auf Seite 17)

[Dor11] R. Dormien. Service-Bus-Erweiterung um Pandas-basierte Simulationen in Workflows zu nutzen. 2011. (Zitiert auf den Seiten 35, 37 und 52)

[EFS08] K. S. Esmaili, P. M. Fischer, J. Simeon. XQuery 1.0 Web Services Facility (Proposal), 2008. (Zitiert auf Seite 96)

[G+12] O. W. Group, et al. W.: OWL 2 web ontology language: document overview. W3C Recommendation (27 October 2009), 2012. (Zitiert auf Seite 39)

[Ges] A. Gessler. *MapReduce to Couple a Bio-mechanical and a Systems-biological Simulation*. Bachelor-arbeit. (Zitiert auf Seite 79)

[GHCM09] T. Gunarathne, C. Herath, E. Chinthaka, S. Marru. Experience with adapting a WS-BPEL runtime for eScience workflows. In *Proceedings of the 5th Grid Computing Environments Workshop*, S. 7. ACM, 2009. (Zitiert auf den Seiten 33 und 79)

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993. (Zitiert auf Seite 38)

[Gru12] M. Grushinskiy. XMLStarlet command line XML toolkit, 2002. *Retrieved May*, 15, 2012. (Zitiert auf Seite 105)

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. Conventional workflow technology for scientific simulation. In *Guide to e-Science*, S. 323–352. Springer, 2011. (Zitiert auf den Seiten 6, 31, 32, 33 und 34)

[Har96] S. Hartmann. The world as a process. In *Modelling and simulation in the social sciences from the philosophy of science point of view*, S. 77–100. Springer, 1996. (Zitiert auf den Seiten 35 und 36)

[HH93] D. Hollingsworth, U. Hampshire. Workflow management coalition the workflow reference model. *Workflow Management Coalition*, 68, 1993. (Zitiert auf Seite 26)

[Hol95] D. Hollingsworth. Workflow management coalition. *The Workflow Reference Model*, S. 22–23, 1995. (Zitiert auf Seite 26)

[JEA+07] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, et al. Web services business process execution language version 2.0. *OASIS standard*, 11:11, 2007. (Zitiert auf den Seiten 28, 30 und 31)

[K+13] R. Krause, et al. Scientific Workflows for Bone Remodelling Simulations. *Applied Mathematics and Mechanics*, 13(1), 2013. (Zitiert auf den Seiten 11 und 36)

[KE11] A. Kemper, A. Eickler. *Datenbanksysteme: Eine Einführung*. Oldenbourg Verlag, 2011. (Zitiert auf den Seiten 16 und 19)

[LR00] F. Leymann, D. Roller. Production workflow: concepts and techniques. 2000. (Zitiert auf den Seiten 6, 22, 25, 26, 27, 31, 32, 34 und 73)

[MB05] T. M. McPhillips, S. Bowers. An approach for pipelining nested collections in scientific workflows. *ACM Sigmod Record*, 34(3):12–17, 2005. (Zitiert auf Seite 66)

[MBL06] T. McPhillips, S. Bowers, B. Ludäscher. Collection-oriented scientific workflows for integrating and analyzing biological data. In *Data Integration in the Life Sciences*, S. 248–263. Springer, 2006. (Zitiert auf Seite 66)

[MBZL09] T. McPhillips, S. Bowers, D. Zinn, B. Ludäscher. Scientific workflow design for mere mortals. *Future Generation Computer Systems*, 25(5):541–551, 2009. (Zitiert auf den Seiten 6 und 66)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis*. Springer-Verlag, 2010. (Zitiert auf Seite 21)

[Min02] S. Mintert. XML & Co. *Die W3C-Spezifikationen für Dokumentenund Datenarchitektur. Edition W3C. de. München, Boston, San Francisco etc.: Addison-Wesley*, 2002. (Zitiert auf Seite 13)

[Mül10] C. M. Müller. *Development of an integrated database architecture for a runtime environment for simulation workflows*. Diplomarbeit, 2010. (Zitiert auf Seite 37)

[NS11] F. Niedermann, H. Schwarz. Deep business optimization: Making business process optimization theory work in practice. In *Enterprise, Business-Process and Information Systems Modeling*, S. 88–102. Springer, 2011. (Zitiert auf den Seiten 6, 7, 8, 11, 71, 72, 73 und 74)

[Par00] T. Partl. XML-Extensible Markup Language. *Version September*, 2000. (Zitiert auf Seite 13)

[Pie11] H. A. Pietranek. *Erweiterung von SIMPL und BPEL-DM zur Unterstützung weiterer Typen von Datenquellen*. Diplomarbeit, 2011. (Zitiert auf den Seiten 126 und 151)

[Pie12] H. A. Pietranek. *Datenmanagementpatterns in multi-skalaren Simulationsworkflows*. Studienarbeit, 2012. (Zitiert auf den Seiten 8, 52 und 148)

[RCD+11] J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, J. Siméon. XQuery update facility 1.0. *W3C Rec., March*, 2011. (Zitiert auf Seite 94)

[Rei] P. Reimann. Data Provisioning for Simulation Workflows. (Zitiert auf Seite 36)

[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. Citeseer, 2011. (Zitiert auf den Seiten 11, 33, 37 und 41)

[RS+13] P. Reimann, H. Schwarz, et al. Datenmanagementpatterns in Simulationsworkflows. In *BTW*, S. 279–293. 2013. (Zitiert auf den Seiten 8, 144 und 145)

[RS14] P. Reimann, H. Schwarz. Simulation workflow design tailor-made for scientists. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, S. 49. ACM, 2014. (Zitiert auf den Seiten 6, 41, 42, 44, 45, 46 und 49)

[RSM11] P. Reimann, H. Schwarz, B. Mitschang. Design, implementation, and evaluation of a tight integration of database and workflow engines. *Journal of Information and Data Management*, 2(3):353, 2011. (Zitiert auf den Seiten 6, 27, 31, 32, 58 und 78)

[RSM14a] P. Reimann, H. Schwarz, B. Mitschang. Data patterns to alleviate the design of scientific workflows exemplified by a bone simulation. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, S. 43. ACM, 2014. (Zitiert auf den Seiten 6, 52 und 53)

[RSM14b] P. Reimann, H. Schwarz, B. Mitschang. A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, S. 21–38. Springer, 2014. (Zitiert auf den Seiten 6, 8, 11, 41, 47, 48, 51, 149, 150 und 151)

[Sch03] H. Schöning. XML und Datenbanken. *Hanser Fachbuchverlag (November 2002)*, 2003. (Zitiert auf Seite 20)

[Slo07] A. Slominski. Adapting BPEL to scientific workflows. In *Workflows for e-Science*, S. 208–226. Springer, 2007. (Zitiert auf den Seiten 33 und 79)

[Sta73] H. Stachowiak. {Allgemeine Modelltheorie}. 1973. (Zitiert auf Seite 35)

[Ste] D. Stenberg. curl-transfer a URL. *U RL http://curl. haxx. se/docs/manpage. html.(l ast visit: Jul. 2012)*. (Zitiert auf den Seiten 96 und 105)

[Stu09] H. Stuckenschmidt. *Ontologien: Konzepte, Technologien und Anwendungen*. Springer Berlin, 2009. (Zitiert auf Seite 38)

[TDG+07] I. J. Taylor, E. Deelman, D. Gannon, M. Shields, et al. *Workflows for e-Science*. Springer-Verlag London Limited, 2007. (Zitiert auf Seite 32)

[Vrh11] M. Vrhovnik. *Optimierung datenintensiver Workflows: Konzepte und Realisierung eines heuristischen, regelbasierten Optimierers*. Dissertation, 2011. (Zitiert auf den Seiten 7, 8, 11, 12, 59, 61, 78, 81, 82, 83, 85, 87, 88, 91, 137, 138, 151, 156, 157 und 162)

[VSS+07] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Maier, T. Kraft. An approach to optimize data processing in business processes. In *Proceedings of the 33rd international conference on Very large data bases*, S. 615–626. VLDB Endowment, 2007. (Zitiert auf den Seiten 6, 7, 59, 60, 62, 65 und 83)

[Wag11] F. Wagner. Nutzung einer integrierten Datenbank zur effizienten Ausführung von Workflows. In *BTW Workshops*, S. 145–149. 2011. (Zitiert auf den Seiten 31 und 33)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. (Zitiert auf den Seiten 6 und 24)

[Zin10] D. Zinn. *Design and optimization of scientific workflows*. Dissertation, UNIVERSITY OF CALIFORNIA DAVIS, 2010. (Zitiert auf den Seiten 6, 11, 66, 67, 68, 69, 70, 71, 78 und 79)

[ZT05] O. C. Zienkiewicz, R. L. Taylor. *The finite element method for solid and structural mechanics*. Butterworth-heinemann, 2005. (Zitiert auf den Seiten 37 und 38)

Alle URLs wurden zuletzt am 15. 01. 2015 geprüft.

Erklärung

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

Ort, Datum, Unterschrift