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

Diplomarbeit Nr. 3038

Nutzung einer integrierten Datenbank zur effizienten Ausführung von Workflows

Florian Bernd Dominic Wagner

Studiengang:	Informatik
Prüfer:	Dr. habil. Holger Schwarz
Betreuer:	Dipl.-Inf. Peter Reimann
begonnen am:	16. September 2010
beendet am:	22. Februar 2011
CR-Klassifikation:	D.2.11, H.2.3, H.2.4, H.2.8, H.4.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Integration ist eines der wichtigsten Themen in unserer heutigen Welt. Von einem soziologischen Standpunkt aus betrachtet rücken Menschen aus der ganzen Welt jeden Tag näher zusammen. Dies führt manchmal zu Missverständnissen und kulturellen Konflikten. Der einzige Ausweg besteht darin, miteinander zu kommunizieren und die Lebensweise und Fähigkeiten der anderen zu respektieren. Diese Globalisierung wird nicht nur durch den immer günstiger werdenden Massenverkehr oder die überall verfügbare Telekommunikation angetrieben, auch die Informationstechnologie trägt einen großen Teil dazu bei und ist mit unserem täglichen Leben, dem Massenverkehr und der Telekommunikation verwoben wie kaum eine andere Technologie je zuvor. Mit dem omnipräsenten Internet können wir Informationen und Menschen auf der ganzen Welt, ohne eine für den Menschen spürbare Zeitverzögerung, austauschen und erreichen. Unternehmen und wissenschaftliche Einrichtungen, die über die ganze Welt verstreut sind, können ihre Daten teilen und haben dadurch die Möglichkeit global zusammen zu arbeiten. Es können Dienstleistungen angeboten werden, die vor 20 Jahren noch unvorstellbar waren und Daten wissenschaftlicher Experimente zeitnah ausgewertet werden. Die riesigen Messdaten des Compact Muon Solenoid (CMS), der Partikel Detektor des Large Hadron Colliders (LHC) in Cern, werden beispielsweise durch ein weltweites Netz von Rechnern ausgewertet [DBG+03].

Aber wie kam es überhaupt dazu und was bedeutet es Daten zu speichern und auszutauschen? - Nach den 1950ern konnte man sehr gut wissenschaftliche Probleme sowie betriebswirtschaftliche Berechnungen, z.B. Abrechnungen, mit Computern durchführen. Im Zuge der Softwarekrise in den 1970ern, kam die Notwendigkeit auf, immer größere Datenmengen getrennt von den Anwendungen abzuspeichern bzw. überhaupt Systeme zu besitzen, die große Datenmengen verwalten können. Aus dieser Notwendigkeit heraus sind die heutigen Datenbanksysteme entstanden, die Funktionen zum Speichern und Laden von Daten anbieten und die Verwaltung, wie diese Daten physikalisch auf Bändern und Festplatten gespeichert werden, übernehmen. Da die Datenbanktechnologie nun schon fast ein halbes Jahrhundert alt ist, ist sie eine ausgereifte und anerkannte Technologie innerhalb der Informatik.

Die Idee verschiedene Computer miteinander zu verbinden entstand ebenfalls in den 1970ern, hauptsächlich vom U.S. Militär vorangetrieben, die Nachrichten und Informationen zwischen ihren Außenposten austauschen wollten. Dieses Netzwerk wurde in den 1980ern erweitert, indem Universitäten aller Welt angeschlossen wurden, in Deutschland war ein Server der Universität Karlsruhe der Erste, der mit dem sog. *Internet* verbunden war. In den 1990ern

wurde das Internet öffentlich und innerhalb der letzten 10 Jahre haben sogar kleine Firmen und Privathaushalte mindestens einen Computer oder ein elektronisches Gerät, welches mit dem Internet verbunden ist.

Somit besitzen wir heutzutage ein riesiges, weltumspannendes, verteiltes und heterogenes Computersystem und eine Vielzahl von Anwendungen, die potentiell miteinander verbunden sind. Dadurch kann es einfacher, schneller und günstiger sein, neue Anwendungen durch eine Kombination bestehender Anwendungen zu realisieren. Jedoch ist die Stärke dieser verteilten Applikationen zugleich auch ihre größte Schwäche. Das Hauptproblem besteht darin, Mittel und Wege zu finden, diese verstreuten Anwendungen in einer einheitlichen Art und Weise miteinander arbeiten bzw. kommunizieren zu lassen. Eine generelle Architekturbeschreibung, welche die nötigen Voraussetzungen für ein solches System beschreibt, ist die Service Oriented Architecture (SOA) [WCL+05]. Eine allgemein anerkannte Realisierung für eine SOA sind Webservices (WSs) [WCL+05] und Workflows (WFs) [FL00]. Mit der Workflowsprache Web Services- Business Process Execution Language (WS-BPEL) [OAS07] kann man Prozesse (z.B. Geschäftsprozesse) durch Orchestrierung einzelner WSs modellieren und ausführen. Somit wird die Erstellung und die Ausführung von WFs, die Stabilität und die Geschwindigkeit der zugrunde liegenden WF-Management Systeme (WfMSe) immer wichtiger.

1.1. Motivation und Aufgaben dieser Arbeit

Nahezu alle WfMSe verwenden ein Datenbanksystem (DBS) um WF- und Prozess-Daten (z.B. Variableninhalte, Nachrichten an und von WSs und Metainformationen zu Ausführungen) zu speichern und persistent zu halten. Da Datenbanksysteme eine ausgereifte und skalierbare Technologie darstellen, ist es von Interesse, ihre Möglichkeiten und Funktionen auszunutzen, um die Arbeit von WfMSen zu verbessern. Ansatzpunkte sind die Verbesserung der Ausführungszeit von Workflow-Instanzen, ein geringer Hauptspeicherverbrauch und die Verbesserung der Stabilität sowie des Durchsatzes bei paralleler Ausführung von Instanzen. In dieser Arbeit soll geprüft werden, welche Möglichkeiten existieren, ein DBS stärker an ein WfMS anzubinden und ob dies die Leistungsfähigkeit des WfMSs erhöht und eine messbare Verbesserung der angesprochenen Eigenschaften zur Folge hat.

Hierbei konzentrieren wir uns auf die Verbesserung der Variablenzuweisung, der Bedingungsauswertung sowie der WS-Aufrufe der Workflowsprache WS-BPEL. Diese sind die meist verwendeten BPEL-Aktivitäten und somit wichtige Optimierungskandidaten. Trotz der Orientierung von WS-BPEL auf Geschäftsprozesse, ist ein aktueller Forschungsschwerpunkt WS-BPEL ebenfalls für wissenschaftliche (eScience) WFs zu verwenden [Slo07] [GSK+11]. Insbesondere um die Beschreibung aller WF-Arten zu vereinheitlichen. Wissenschaftliche WFs verarbeiten typischerweise größere Datenmengen innerhalb des WfMSs als WFs für Geschäftsprozesse, weshalb die Performanz und Stabilität der genannten WS-BPEL Aktivitäten

ebenfalls von großem Interesse sind. Um einen Prototypen entwickeln zu können, greifen wir auf die WS-BPEL OpenSource WF-Engine Apache Orchestration Director Engine[1] zurück.

1.2. Konventionen und rechtliche Hinweise

Begriffe, für die eine abkürzende Schreibweise existiert, werden bei der erstmaligen Verwendung ausgeschrieben und dahinter innerhalb runder Klammern die Abkürzung angegeben. Zusätzlich wird ein Abkürzungsverzeichnis im Anhang A (Seite 143) angegeben.

In dieser Arbeit kam Software zum Einsatz, die nicht öffentlich zur Verfügung steht und für die Lizenzen erworben werden müssen. Für den Einsatz des in dieser Arbeit entstandenen Prototyps zusammen mit bestimmten Produkten, muss eine entsprechende Lizenz erworben werden. Dies betrifft insbesondere die in der nachfolgenden Liste genannten Produkte:

- IBM DB2 UDB V9.7

Diese Arbeit enthält eine Datenbank-Auswertung. Der Autor dieser Arbeit hat die Vorbereitung und Ausführung dieser Auswertung mit besonderer Vorsicht durchgeführt. Trotzdem kann der Autor mögliche Fehler, die hierbei entstanden sind, nicht ausschließen. Aus diesem Grund übernimmt der Autor keine Verantwortung für die Korrektheit und Vollständigkeit der gesamten Auswertung und der daraus geschlossenen Erkenntnisse.

1.3. Aufbau dieses Dokuments

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Stellt wichtige informationstechnische Grundlagen zum Verständnis dieser Arbeit vor.

Kapitel 3 – Workflow Architekturen und Datenbank Integration: Gibt einen Einblick in die Archtitektur von WfMSen und stellt verwandte Arbeiten und Themen vor.

Kapitel 4 – Nutzung von Funktionen einer integrierten Workflowdatenbank: Befasst sich mit in dieser Arbeit entstandenen Konzepten sowie Konzepte aus der Literatur, die eine stärkere Integration der integrierten DB eines WfMSs ermöglichen.

Kapitel 5 – Apache ODE Architektur im Detail: Beleuchtet die Teile der Softwarearchitektur von Apache ODE, die für die Implementierung des Prototyps wichtig sind.

[1] Apache Orchestration Director Engine http://ode.apache.org

Kapitel 6 – Implementierung des Prototyps: Stellt Details zur Implementierung des Prototyps vor.

Kapitel 7 – Evaluierung des Prototyps: Dieses Kapitel beinhaltet die Laufzeit- und Hauptspeichermessungen zum Prototyp sowie eine Diskussion der Ergebnisse.

Kapitel 8 – Konzeptionelle Erweiterungen: Bettet die Ergebnisse dieser Arbeit in einen größeren Kontext ein und stellt weiterführende Arbeiten vor.

Kapitel 9 – Zusammenfassung: Fasst die Arbeit, ihre Ergebnisse und weiterführende Arbeiten zusammen.

2. Grundlagen

In diesem Kaptitel werden wir den technischen sowie wissenschaftlichen Hintergrund liefern, der nötig ist, um diese Arbeit nachzuvollziehen. Der erste Abschnitt wird sich mit der eXtensible Markup Language (XML) und Techniken befassen diese Daten zu beschreiben und zu verarbeiten. Der zweite Abschnitt befasst sich mit der in der Einleitung angesprochenen SOA und Webservices. Der dritte Abschnitt widmet sich den Workflowsprachen, insbesondere WS-BPEL. Der vierte Abschnitt soll eine kurze Zusammenfassung der Datenbanktechnologie geben und stellt die in dieser Arbeit verwendeten Datenbanksysteme und ihre Funktionalitäten, insbesondere bezüglich XML, kurz vor. Der fünfte und letzte Abschnitt dieses Kapitels stellt den für die Evaluation verwendeten Workflow-Anwendungsfall vor.

2.1. eXtensible Markup Language

Die eXtensible Markup Language (XML) [W3C08] ist eine sog. Markup-Sprache. Markup-Sprachen zeichnen sich dadurch aus, dass Informationen mit ihren zugehörigen Metainformationen verknüpft werden. In XML geschieht dies über die sog. *Tags*, diese beherbergen die Metainformation in ihrem Namen und umklammern die damit verbundene Information. Das öffnende Tag wird in spitzen Klammern geschrieben <*information*>, das schließende Tag ebenfalls in spitzen Klammern, wobei noch ein Querstrich eingeführt wird </*information*>, dazwischen befindet sich die Information (<*information*>*Hier ist die Information*</*information*>). Die Information kann sich aus Text, sowie neuen Tags zusammensetzen. Somit lässt sich XML z.B. als Baumstruktur entsprechend dem Document Object Model (DOM) [w3ca] verarbeiten.

In der Tat existieren Verwandschaftsbeziehungen zur Hyper Text Markup Language (HTML). Im Gegensatz zu HTML, wo jedes Tag eine bestimmte Bedeutung bei der graphischen Darstellung in einem Webbrowser besitzt, werden die Tags und deren Bedeutung in XML vom Benutzer vorgegeben bzw. in einer Spezifikationssprache definiert (Document Type Definition (DTD) [W3Cb] oder XML-Schema [Tho04]). Der tatsächliche Vorgänger von XML ist die Standard Generalized Markup Language (SGML) [sgm86], welche auch das DTD Format eingeführt hat. Ein *wohlgeformtes* XML-Dokument besitzt einen einzigen Wurzelknoten (also ein öffnendes und schließendes Tag des gleichen Namens) und zu jedem geöffneten Tag existiert ein schließendes Tag auf gleicher Tiefe des Baumes. Ein Beispiel für ein wohlgeformtes XML-Dokument ist in Listing 2.1 zu sehen.

Im Gegensatz zu einer Tabelle, wie man sie von relationalen Datenbanken her kennt, ist die Metainformation von der Information selbst nicht getrennt und ist somit für den Menschen besser lesbar. Trotzdem ist ein solches Dokument durch die vorhandenen Metainformation von Programmen immer noch einfach, im Gegensatz zu einem geschriebenen Text, zu verarbeiten. Ein weit wichtigerer Punkt ist jedoch, dass es jederzeit möglich ist, ein einzelnes Dokument um neue Metainformationen, durch zusätzliche oder neue Tags, und Informationen zu erweitern, ohne dass darauf arbeitende Programme geändert werden müssen. Durch diese und weitere Eigenschaften, gehört XML zu den Semi-Strukturieren Daten (siehe Tabelle 2.1).

Typ	Typische Vertreter	Änderbarkeit der Struktur
Strukturierte Daten	Datenbanktabellen	Einheitlich für alle Datensätze
Semi-Strukturierte Daten	XML	Jederzeit für einzelne Dokumente
Unstrukturierte Daten	Textdokumente	keine Einschränkungen

Tabelle 2.1.: Strukturgrad von Informationen und typische Vertreter dieser Klassen.

XML Knoten

Im Folgenden stellen wir kurz alle XML Knotentypen vor. Alle genannten Knoten finden sich ebenfalls in Listing 2.1 wieder:

Verarbeitungsanweisungen (*engl. processing instruction*) beinhalten Informationen wie das Dokument zu verarbeiten ist, z.B. ob es sich um ein HTML oder XML Dokument handelt und welche Zeichenkodierung (UTF-8, Latin1 etc.) verwendet wurde. Dieser Knoten wird durch einen Block aus Spitzen-Klammern und ein Fragezeichen gekennzeichnet. *<?xml version="1.0"encoding="UTF-8"?>*

Kommentare beschreiben zusätzliche Informationen die für das menschliche Verständnis von Interesse sind. *<!– information about the author –>*

Text Knoten beinhalten Text z.B. *Florian Wagner*

Element Knoten werden durch Tags geöffnet und geschlossen und können Textknoten und/oder weitere Elementknoten enthalten. *<exp:name>Florian Wagner</exp:name>*

Attribut Knoten können zusätzliche Informationen beherbergen, die an ein Element gebunden werden können, ohne hierfür ein Unterknoten zu definieren. Dies ist insbesondere dann nützlich, falls diese Information innerhalb der Struktur nur einmalig vorkommt. Vergleichen wir hierzu in Listing 2.1, dass ein *title* Attribut für den *document* Knoten nicht ausreicht, um mehrere Titel in verschiedenen Sprachen darzustellen. Attribute werden innerhalb eines Tags aufgenommen. *<exp:document type="thesis">*

Gemischter Inhalt Dies ist eine spezielle Form, bei dem im Unterbaum eines Elementknotens Text sowie weitere Elementknoten vorkommen. Dies ist im *document* Konten in Listing 2.1 veranschaulicht.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <exp:thesis xmlns:exp="http://www.flowsoft.de/thesis/xml">
3         <!-- information about the author -->
4         <exp:author>
5                <exp:name>Florian Wagner</exp:name>
6         </exp:author>
7         <!-- information about document -->
8         <exp:document type="thesis">
9                <exp:title lang="en">Exploiting an integrated database system to improve
                     workflow execution</exp:title>
10               <exp:title lang="de">Nutzung einer integrierten Datenbank zur effizienten
                     Ausführung von Workflows</exp:title>
11               Diese Arbeit befasst sich mit Workflow- und Datenbanksystemen.
12        </exp:document>
13 </exp:thesis>
```

Listing 2.1: Ein wohlgeformtes XML-Dokument, welches alle XML-Knotentypen beinhaltet.

XML Namensräume

Namensräume (*engl. namespaces*) wurden eingeführt, um zwischen gleichen *Tag*-Namen mit unterschiedlicher Semantik unterscheiden zu können. Vorallem beim Datenaustausch zwischen Unternehmen oder wissenschaftlichen Einrichtungen können potentiell XML Strukturen und Tagnamen identisch sein, sich überschneiden und im schlimmsten Fall eine andere Bedeutung besitzen. Um während des Auslesens der Informationen, z.B. durch ein Programm, zwischen der richtigen Semantik unterscheiden zu können, kann jedem Elementknoten ein Namensraum zugeordnet werden, der somit dessen Semantik festlegt. Die Spezifikation dieser Semantik, z.B. in Form von DTD, XML-Schema oder in Form eines Textdokuments, wird diesem Namensraum zugeordnet. Darüber hinaus ist es auch möglich innerhalb eines XML Dokuments Elemente verschiedener Namensräume zu verwenden.

Ein Namensraum wird durch einen Uniform Resource Identifier (URI) beschrieben, oftmals werden hierzu Uniform Resource Locations (URL) verwendet z.B. eine HTTP Adresse. Diese muss allerdings nicht zwingend existieren, es ist aber eine gängige Praxis hinter dieser URL die XML Spezifikation abzulegen. Typischerweise werden die Namensräume eines XML Dokuments im Wurzelknoten über die Attribute *xmlns* gefolgt von einem Doppelpunkt und dem zugeordneten Präfix angegeben. In Listing 2.1 ist also der Namensraum *http://www.flowsoft.de/thesis/xml* an den Präfix *exp* gebunden. Die Knoten dieses Namensraums werden dann durch das Präfix vor ihren Namen (durch einen Doppelpunkt getrennt) zu

diesem zugeordnet (vgl. <*exp:author*>). Es kann auch ein Standard Namensraum angegeben werden, indem lediglich das Attribut *xmlns* ohne Präfixangabe verwendet wird. Knoten ohne Präfix gehören dann automatisch zu diesem Namensraum.

Da die Nutzung von Namensräumen oft zu Verwirrungen führt, soll hier angemerkt sein, dass ein Präfix unabhängig vom Namensraum ist. Dies bedeutet, dass der Präfix nur eine abkürzende Schreibweise für das jeweils aktuelle Dokument darstellt und in einem zweiten Dokument anders heißen kann. Ebenfalls kann innerhalb einer *XPath* oder *XQuery* Anfrage (siehe Kapitel 2.1.2 und 2.1.3) der Namensraum an einen dritten Präfix gebunden werden. Wichtig ist nur, dass die Namensraum URI der Knoten identisch ist, falls sie zum gleichen Namensraum gehören.

2.1.1. XML-Schema

XML-Schema [Tho04] ist eine XML Definitionssprache, die in sich selbst definiert und im XML Format geschrieben ist. Dies ist einer der Unterschiede zur DTD, die ein eigenes Format besitzt. Ebenfalls können komplexere Strukturen als mit DTD beschrieben werden. Das Hauptmerkmal von XML-Schema liegt darin, dass man eine genaue Typisierung von Datenfeldern vornehmen kann. Es wird zwischen vordefinierten Basistypen (wie *String*, *Integer*, *Boolean* etc.) und vom Benutzer definierten, einfachen sowie komplexen Typen unterschieden. Über die komplexen Typen kann die Struktur eines XML Dokuments spezifiziert werden, durch das Einschränken von Basistypen zu simplen Typen können z.B. Enumerationen oder Zahlenräume definiert werden. Ein XML Dokument wird als valide bezeichnet, falls es wohlgeformt ist (vgl. Kapitel 2.1) und die Spezifikation erfüllt. Das XML-Schema, welches die Struktur des XML Dokument aus Listing 2.1 spezifiziert ist in Listing 2.2 zu sehen.

```
1  <xsd:schema targetNamespace="http://www.flowsoft.de/thesis/xml"
2            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3
4        <!-- Element definitions -->
5        <xsd:element name="thesis" type="thesisType"/>
6
7        <!-- Type definitions -->
8        <xsd:complexType name="thesisType">
9               <xsd:sequence>
10     <xsd:element name="author" type="authorType" minOccurs="0"
11       maxOccurs="unbounded"/>
12                    <xsd:element name="document" type="documentType" minOccurs="1"/>
13              </xsd:sequence>
14       </xsd:complexType>
15
16       <xsd:complexType name="authorType">
17              <xsd:sequence>
18                    <xsd:element name="name" type="xsd:string" minOccurs="1"/>
```

```
19                      <xsd:element name="adress" type="xsd:string" minOccurs="0"/>
20              </xsd:sequence>
21      </xsd:complexType>
22
23      <xsd:complexType name="documentType" mixed="true">
24              <xsd:sequence>
25                      <xsd:element name="title" type="titleType" minOccurs="1"
                            maxOccurs="unbounded"/>
26              </xsd:sequence>
27              <xsd:attribute name="type" type="documentTypes" use="optional" default="no"/>
28      </xsd:complexType>
29
30      <xsd:complexType name="titleType" mixed="true">
31              <xsd:attribute name="lang" type="xsd:string" use="required" default="no"/>
32      </xsd:complexType>
33
34      <xsd:simpleType name="documentTypes">
35              <xsd:restriction base="xsd:string">
36                      <xsd:pattern value="thesis|paper|manual"/>
37              </xsd:restriction>
38      </xsd:simpleType>
39
40  </xsd:schema>
```

Listing 2.2: Ein XML-Schema Dokument, welches die Struktur des XML Dokuments aus Listing 2.1 definiert.

2.1.2. XPath

Mit XPath [W3C99] ist es unter anderem möglich, einen hierarchischen Pfad innerhalb eines XML Dokuments zu durchlaufen und Unter-, Text-, Attribut- und Kommentarknoten sowie Verarbeitungsanweisungen ausgeben zu lassen. Ein einfacher Pfadausdruck sieht einem absoluten Verzeichnispfad eines Unix Betriebssystems sehr ähnlich, welches ebenfalls in einer hierarchischen Baumstruktur dargestellt wird. XPath besitzt eine Vielzahl an Möglichkeiten um Vater-, Kind- und Geschwisterknoten auszuwählen, dies geschieht über die sog. Achsen (*engl. axes*). Man kann für jede Hierarchiestufe einen sogenannten Lokalisierungsschritt durchführen. Dieser besteht aus einem einleitendem Querstrich, der Angabe der Achse, einem doppelten Doppelpunkt, einen Knotentest und aus einem optionalen Prädikats, welches innerhalb eckiger Klammern angegeben wird. Diese Lokalisierungsschritte können konkateniert werden. Ein Lokaliserungsschritt sieht also folgendermaßen aus:

```
/Achse::Knotentest[Prädikat]
```

Je nach XPath-Implementierung ist es erforderlich bzw. möglich Namensräume anzugeben und an Präfixe zu binden. Der Präfix wird dann wie im XML Dokument beim Knotentest durch einen Doppelpunkt getrennt vom Knotennamen angegeben. Einige wichtige

Achsen werden in Tabelle 2.2, die Knotentests in Tabelle 2.3, inklusive ihrer abkürzenden Schreibweisen angegeben.

Achse	selektiert	Abkürzung
child::	alle direkten Unterknoten	*wird weggelassen*
parent::	den Elternknoten	..
descendant-or-self::	den aktuellen Knoten und alle Unterknoten	//
attribute::	alle Attributknoten	@
namespace::	alle Namensraumknoten	

Tabelle 2.2.: Die XPath-Achsen mit Beschreibung und abkürzender Schreibweise.

Knotentest	selektiert
*	alle Knoten
Knotennamen	benannten Knoten
text()	Textknoten
comment()	Kommentarknoten
processing-instruction()	Verarbeitunsanweisungskonten

Tabelle 2.3.: Die XPath-Knotentests mit Beschreibung.

Um die Funktionsweise zu verdeutlichen, werden nun zwei Ausdrücke vorgestellt, mit denen Information aus dem XML Dokument aus Listing 2.1 extrahiert werden können. Der Folgende XPath-Ausdruck liefert den Name des Autors zurück:

```
/child::thesis/child::author/child::name/text()
```

Der zweite XPath-Ausdruck liefert den deutschen Titel zurück:

```
/child::thesis/child::document/child::title[attribute::lang="de"]/text()
```

Um die Anwendung dieser Ausdrücke zu vereinfachen wurden abkürzende Schreibweisen eingeführt, die in der Tabelle 2.2 aufgeführt sind. Somit lassen sich beide XPath-Ausdrucke vereinfachen zu:

```
/thesis/author/name/text()
/thesis/document/title[@lang="de"]/text()
```

Des Weiteren bietet XPath eine Reihe von Funktionen an, um Zeichenketten zu manipulieren und zu verarbeiten. Es ist zudem möglich mathematische Ausdrücke berechnen zu lassen und diese z.B. innerhalb eines Prädikats einzusetzen. Durch das Einführen von Variablen

können ebenfalls Ausdrücke evaluiert werden, die auf mehrere XML Dokumente verweisen. Für weitergehende Informationen über XPath und seine Funktionen empfehlen wir die XPath Spezifikation vom W3C [W3C99].

2.1.3. XQuery

XPath stellt eine Möglichkeit dar, Knoten und Informationen aus XML Dokumenten zu extrahieren. Allerdings ist es nicht ohne Weiteres möglich, Informationen aus mehreren XML Dokumenten zu aggregieren oder Berechnungen auf mehreren, sich wiederholenden Knoten innerhalb eines XML Dokumentes durchzuführen. Um dies zu bewerkstelligen wurde XQuery [W3C07b] eingeführt, das von einer ganzen Reihe an Query Sprachen, insbesondere *XPath* und *SQL*, inspiriert wurde. Das Hauptziel ist es, XML Dokumente in einer Mengen-orientierten Weise zu verarbeiten, wie es z.B. SQL bei relationalen Datenbanken erlaubt. Das zugrunde liegende Datenmodell besteht aus den Knotentypen, die wir von XML und XPath her kennen (vgl. Kapitel 2.1), Sequenzen (für die Mengenverarbeitung), atomaren Werten und Ausdrücken. Mit XQuery ist es so möglich einen Verbund (Join) zwischen XML Dokumenten herzustellen und deren Informationen miteinander zu verknüpfen (entsprechend einem Join auf relationalen Tabellen). Die Syntax von XQuery heißt *FLWOR* was für *For Let Where Orderby Return* steht und ist angelehnt an die *SFW* (*Select From Where*) Syntax von SQL. Der nachfolgende XQuery-Ausdruck liefert die Titel aus dem Beispiel in Listing 2.1 in alphabetischer Reihenfolge zurück:

```
for $x in fn:doc("XMLEXAMPLE.xml")/thesis/document
order by $x/title
return {$x/title/text()}
```

Für weitergehende Informationen über XQuery und seine Funktionen empfehlen wir die XQuery Spezifikation vom W3C [W3C07b].

2.1.4. pureXML

Das DBS DB2 von IBM verwendet die eigene XML-Verarbeitungssprache *pureXML* [Che07]. *PureXML* unterstützt die XQuery- und somit auch die XPath-Spezifikation. Es ist möglich diese XQuery-Anfragen in SQL-Ausdrücke einzubetten, genauso ist es umgekehrt möglich in XQuery-Anfragen relationale Daten einzubinden. Für die Einbettung von XQuery in SQL existiert die SQL-Funktion *XMLQUERY*, für die Einbettung von SQL oder den Zugriff auf ein relationales XML-Feld innerhalb eines XQuery-Ausdrucks existieren die XQuery-Erweiterungsfuntionen *db2-fn:xmlcolumn* und *db2-fn:sqlquery*. Soll ein nativer XQuery-Ausdruck anstatt einem SQL-Ausdruck verarbeitet werden, reicht die Angabe des Schlüsselworts *xquery* vor dem eigentlichen XQuery-Ausdruck. Das XQuery-Beispiel aus

Kapitel 2.1.3 ist als Anfrage an die DB2 in Listing 2.3 dargestellt.

```
1  xquery for $x in db2-fn:sqlquery("select xmlfeld from daten where id = 1")/thesis/document
2  order by $x/title
3  return {$x/title/text()}
```

Listing 2.3: XQuery-Anfrage an DB2 mit Zugriff auf ein XML-Feld einer relationalen Tabelle.

Darüber hinaus stellt *pureXML* eine eigene, in XQuery eingebettete Syntax vor, mit der XML Manipulationen möglich sind. Bisher existiert hierzu noch kein Standard, allerdings gibt es einen Kandidaten für die sog. XQuery Update Facility [W3C09], deren Syntax teilweise ähnlich zu der von *pureXML* ist. Die Syntax für die Manipulation in pureXML ist in Listing 2.4 zu sehen. Der ursprüngliche Wert des XML Dokuments wird mit der Variable *$new* verknüpft und anschließend durch die Anweisung *modify do replace value of* der Inhalt des in XPath angegebenen XML-Elements (*$new/thesis/author/name*) durch die Zeichenkette „Florian BD Wagner“ ersetzt. Der *return* Befehl gibt das veränderte Dokument zurück.

```
1  update daten set xmlfeld = XMLQUERY('
2     copy $new := $XMLFELD
3     modify do replace value of $new/thesis/author/name
4     with "Florian BD Wagner"
5     return $new')
6  where id = 1
```

Listing 2.4: Ein pureXML-Ausdruck, der ein bestehendes, in einem XML-Feld abgelegtes, XML-Dokument modifiziert.

Auf die Spalte *xmlfeld* (vom Typ XML) der Tabelle *daten* kann innerhalb des *pureXML*-Ausdrucks über den groß geschriebenen Namen *$XMLFELD* zugegriffen werden. Anstelle von *do replace* können auch neue XML-Elemente eingefügt (*do insert ... before/after ...*) oder gelöscht werden (*do delete*). Es kann auch auf mehrere XML-Felder referenziert werden, indem diese als Variablen an die *pureXML* Funktion *XMLQUERY* übergeben werden (*passing* siehe Listing 2.5). Für weitergehende Funktionalitäten von *pureXML* sei auf [Che07] verwiesen.

```
1  update daten set xmlfeld = XMLQUERY('
2     copy $new := $XMLFELD
3     modify do replace value of $new/thesis/author/name
4     with value of $XMLFELD2/thesis/author/name
5     return $new' passing (select xmlfeld from daten where id =2) as "XMLFELD2")
6  where id = 1
```

Listing 2.5: Ein pureXML-Ausdruck, der ein bestehendes XML-Dokument unter Verwendung des Inhalt eines zweiten XML-Dokuments modifiziert.

2.2. Service Oriented Architecture

Die Service Oriented Architecture (SOA) ist ein Paradigma für eine Softwarearchitektur, welche sich aus kleineren, selbstständigen Programmen zusammensetzt. Diese Programme werden *Services* oder Dienste genannt, deren Zusammensetzung (*Orchestrierung*) die neue Funktionalität generiert. So soll es schnell und mit geringen Kosten möglich sein, bestehende Software miteinander interagieren zu lassen und diese Interaktion jederzeit einfach zu verändern. Dieser Wunsch kommt insbesondere aus der Wirtschaft, ist aber ebenso für größer angelegte wissenschaftliche Experimente interessant. Ein einfaches Beispiel ist die Übernahme eines Konkurrenz-Unternehmens und der Wunsch oder die Notwendigkeit bestehende Software- und IT-Landschaften einheitlich verwenden zu können. Im Wesentlichen ergeben sich hieraus auch die weiteren Konzepte einer SOA:

- Lose Kopplung
- Service/Dienstleistungs-Vertrag
- Abstraktion
- Wiederverwendbarkeit
- Zustandslosigkeit

Eine heutzutage weit verbreitete Realisierung einer SOA, ist die Orchestrierung von Webservices (WS), die wir im Folgenden vorstellen. Die Orchestrierung der WSs erfolgt oft mit Hilfe von Workflows (WF), die wir in einem eigenen Kapitel 2.3 besprechen. Für eine tiefer gehende Beschreibung der SOA und den dazugehörigen Konzepten sei auf das Lehrbuch [WCL+05] verwiesen.

2.2.1. Webservices

Eine Reihe von Standards des World Wide Web Consortiums (W3C) und der Organization for the Advancement of Structured Information Standards (OASIS) bilden zusammen die sog. WS-* Standards. Nahezu alle WS-* Standards werden im XML Format geschrieben und in XML-Schema (siehe Kapitel 2.1.1) definiert. Das Nachrichtenaustausch Protokoll sowie die Beschreibungssprache für Webservices werden nachfolgend erläutert. Auf die Workflowsprache WS-BPEL, die eine Orchestrierung der Webservices erlaubt, wird in Kapitel 2.3.2.2 detaillierter eingegangen.

Simple Object Access Protocol

Um die Art und Weise des Nachrichtenaustausch zwischen Diensten, insbesondere WSs, zu standardisieren wurde SOAP [W3C07a] eingeführt. Dieses XML Format erlaubt es Nachrichten einheitlich zu versenden und zu empfangen. In vielen Implementierungen wird es genutzt, um direkt Operationen eines Programms aufzurufen. Hierbei werden innerhalb der SOAP Nachricht die Empfängermethode und die zu übergebenden Parameter gesendet. Liefert die Methode einen Rückgabewert, wird eine SOAP Nachricht mit dem Resultat an den Sender zurück geschickt.

In Listing 2.6 ist eine solche SOAP Nachricht dargestellt. Sie besteht aus dem Umschlag (*engl. envelope*), in dem ein Kopf (*engl. header*) und der Körper (*engl. body*) enthalten sind. Der *Header* kann zusätzliche Informationen für den Transport beinhalten, wie z.B. Informationen zur Authorisierung oder Verschlüsselung. Im *Body* wird die aufzurufende Operation, sowie ihre Parameter, deren Typen und Werte benannt. Im Beispiel wird also die Operation *getSFamilyAlignment* des WS mit dem Namensraum *http://www.dwarf.uni-stuttgart.de/ACCESS*, mit dem Parameter *superfamily_id* vom Typ Integer und dem Wert „8“, angefordert. Welche Operationen zur Verfügung stehen, an welche tatsächliche Intra- oder Internetadresse und mit welchem Transportprotokoll diese Nachricht gesendet werden muss, wird durch die Web Service Definition Language beschrieben.

```
1 <soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:acc="http://www.dwarf.uni-stuttgart.de/ACCESS">
2   <soapenv:Header/>
3   <soapenv:Body>
4     <acc:getSFamilyAlignment
          soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5       <superfamily_id xsi:type="xsd:integer">8</superfamily_id>
6     </acc:getSFamilyAlignment>
7   </soapenv:Body>
8 </soapenv:Envelope>
```

Listing 2.6: Eine SOAP Nachricht aus dem Anwendungsfall für Proteinmmodellierung (Kapitel 2.5.1.1). Die aufzurufende Methode heißt *getSFamilyAlignment* und es wird der Parameter *superfamily_id* mit Typ Integer übergeben.

Web Service Definition Language

Wie im vorhergehenden Abschnitt angesprochen wird WSDL [Chr01] verwendet um WSs ihre Operationen und den Service Endpunkt, also die Adresse an die Nachrichten gesendet werden müssen, zu definieren. WSDL wird ebenfalls in XML geschrieben und besteht im Prinzip aus folgenden sechs Elementen:

Types beschreibt die benutzerdefinierten Typen der Parameter im Abschnitt *Message* in XML-Schema.

Message beschreibt Nachrichten, durch die Angabe von Parametern und deren Typen.

PortType beschreibt die Operationen des WS, diese bestehen aus einer Eingangs- und Ausgangsnachricht die unter *Messages* definiert wurden.

Binding beschreibt das Transport-Protokoll über die die Nachrichten gesendet werden.

Port beschreibt die tatsächliche Intra- oder Internetadresse des Service Endpunkts, an den die Nachrichten gesendet werden sollen.

Service bindet alle *Ports* zusammen, somit ist es möglich Operationen eines WS auf verschiedene Endpunkte zu verteilen.

Mit diesen Informationen ist es möglich SOAP Nachrichten automatisch zu generieren und an den richtigen Endpunkt zu versenden. Ein Beispiel für eine WSDL Datei ist im Anhang C.3 (Seite 155) zu finden.

2.3. Workflowtechnologie

In den letzten 10 Jahren ist der Bedarf an WfMSen stark angewachsen. Durch die Verwendung von Workflows möchte man eine zusätzliche Abstraktionsschicht einziehen. Diese Workflows können mit weniger Aufwand und geringerer IT Expertise erstellt werden, als die zu erstellende Funktionalität in einem monolithischen Programm umzusetzen. Hinzu kommt die Möglichkeit Workflows graphisch zu repräsentieren und in einem WYSIWYG Editor bearbeiten zu können.

Mathematisch betrachtet, stellen Workflows einen gerichteten Graphen dar [FLoo], dieser besteht aus einer Menge von Knoten und einer Menge von Kanten. Die Konten stellen sog. *Aktivitäten* dar, hier werden bestimmte Aufgaben durchgeführt, z.B. das Aufrufen eines Webservice, eine Datenmanipulation oder eine Ausgabe um nur einen Bruchteil möglicher Aktivitäten zu nennen. Die Bedeutung der Kanten ist essentiell für die Art der Workflowsprache, hierauf gehen wir im Kapitel 2.3.2 näher ein. Jedoch ist allen Workflows gemein, dass sie in Richtung der Kanten von den Konten ohne Eingangskanten, bzw. speziell als Startknoten markierte Konten, bis zu Knoten ohne Ausgangskanten durchlaufen werden.

2.3.1. Workflow Management Systeme

WfMSe bestehen typischerweise aus vier Komponenten, der Build-Time, der Run-Time, der Kommunikationsinfrastruktur und aus einem Monitor Programm (siehe Abb. 2.1). Manchmal werden Monitor und Runtime miteinander verschmolzen. Ebenfalls gibt es Systeme in denen die Build-Time und/oder Monitor Komponenten fehlen bzw. unabhängig voneinander sind.

Die Build-Time Komponente ist für die Erstellung von Workflows verantwortlich, je nach System kommen hier graphische Editoren oder spezielle Texteditoren zum Einsatz. Ein Beispiel für einen graphischen Editor ist der Eclipse BPEL Designer[1]

Die Run-Time Komponente compiliert oder interpretiert den Workflow-Prozess der mit einem GUI Editor oder direkt in der WF-Sprache geschrieben wurde und ist für dessen Ausführung verantwortlich. Ein Beispiel für eine Run-Time ist die Apache Orchestration Director Engine (siehe Kapitel 3.3.1, Seite 54).

Die Kommunikationsinfrastruktur wird von der Run-Time verwendet oder die Run-Time ist in ihr eingebettet. Somit kann die Run-Time bestimmte Aktivitäten einer WF-Sprache realisieren. Dies kann z.B. das Versenden von Nachrichten an WSs oder das Ausführen von anderen Applikationen sein. Ein Beispiel für eine solche Infrastruktur ist der Apache Tomcat Server[2].

Die Monitor Komponente ist für die Überwachung und Analyse der Ausführung von Workflow-Instanzen innerhalb der Run-Time verantwortlich. Ein Beispiel ist die Monitor Perspektive des Taverna Systems (siehe Kapitel 3.3.2, Seite 55).

2.3.2. Workflow Sprachen

Workflowsprachen heben sich nicht nur durch die Funktionen der Aktivitäten voneinander ab. Ein wesentlicher Punkt indem sich Workflowsprachen unterscheiden können, ist die Bedeutung der Kanten. Man unterscheidet hierbei zwischen:

Daten-Fluss orientierten WFs - über die Kanten laufen Daten

Kontroll-Fluss orientierten WFs - die Kanten bestimmen die Ausführungsreihenfolge

[1] Eclipse BPEL Designer http://www.eclipse.org/bpel/
[2] http://tomcat.apache.org/

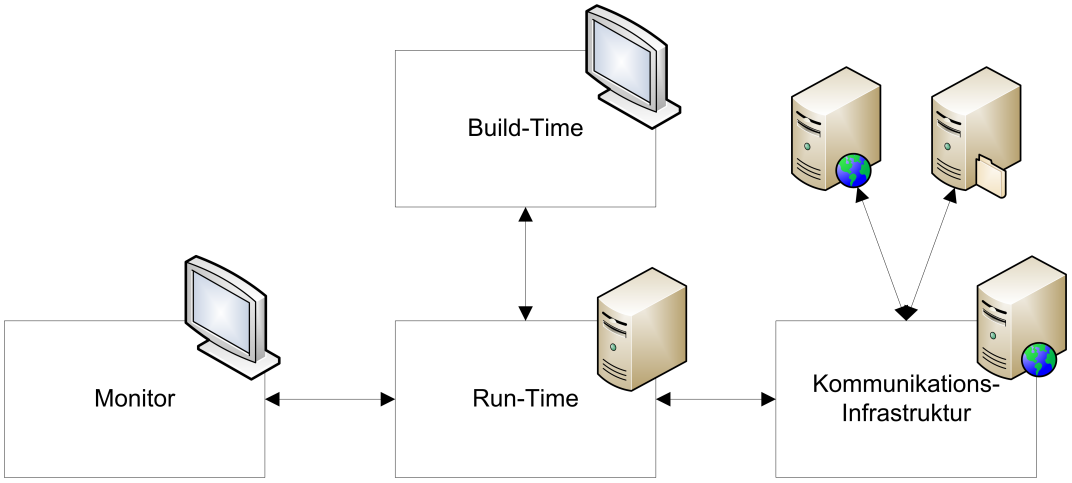


Abbildung 2.1.: Architektur eines typischen WfMSs bestehend aus einer Build-Time zur Erstellung von WFs, einer Run-Time zur Ausführung der WFs, einer Kommunikationsinfrastruktur die von der Run-Time verwendet wird um bestimmte Aktivitäten durchzuführen und einem Monitor zur Überwachung der WF-Ausführung.

Werden für wissenschaftliche Workflows häufig daten-Fluss orientierte Workflowsprachen verwendet, findet man bei der Modellierung von Geschäftsprozessen ausschließlich kontroll-Fluss orientierte Sprachen. Dies kommt daher, da bei wissenschaftlichen Experimenten Daten neu generiert und analysiert werden, die Hauptaufgabe von wissenschaftlichen Workflows stellt meist die Generierung und anschließende Analyse der Daten dar. Zudem ist die Erstellung solcher Workflows intuitiver und es existieren de facto keine Variablen in solchen Workflows. Somit ist es auch Wissenschaftlern, ohne Programmiererfahrung und ohne Kenntnisse über den Variablenbegriff möglich solche WFs zu erstellen.

Bei Geschäftsprozessen hingegen werden oft nur bestehende Daten aggregiert oder abgerufen um eine Entscheidung hervorzurufen *was* als nächstes zu tun ist, hier können kontroll-Fluss orientierte Sprachen mit ihren *if-then-else* und Schleifenkonstrukten die Arbeitsabläufe effizienter und intuitiver beschreiben.

Im Folgenden wird je eine Workflowsprache für daten- sowie kontroll-Fluss vorgestellt. Beide Sprachen basieren auf der Web Services Flow Language (WSFL) von IBM. Das Hauptaugenmerk liegt hierbei auf WS-BPEL, einerseits, da es die Workflowsprache von Apache ODE ist und somit eng mit dieser Arbeit in Verbindung steht und andererseits, da derzeit eine Standardsprache für alle Arten von Workflows gesucht wird. Hierbei ist WS-BPEL Gegenstand vieler Arbeiten und das Simulation Technology Projekt (SimTech) der Universität Stuttgart befasst sich ebenfalls mit WS-BPEL als Workflowsprache für wissenschaftliche Simulationen [Sloo7] [GHCM09] [RRS+10].

2.3.2.1. Simple Conceptual Unified Flow Language

SCUFL ist eine daten-Fluss orientierte Workflowsprache, die vom Bioinformatik-Workflowsystem Taverna [OAF+04] verwendet wird. Taverna wird ebenfalls zunehmend für medizinische und chemische WFs verwendet. Ebenfalls in XML definiert besitzt SCUFL nur sehr wenig Elemente, was die Einfachheit von daten-Fluss orientierten Sprachen unterstreicht. Es gibt nur folgende Elemente:

- Ein- und Ausgaben
- (Daten)Flusskanten
- Koordinationskanten
- Prozessoren (Aktivitäten)

Als Aktivitäten können in Java implementierte Methoden, WSDL Webservices sowie eine Vielzahl weiterer (auch lokaler) Anwendungen dienen. Für weitere Informationen über Taverna und SCUFL sei auf [OAF+04] [OLK+07] und für eine Anwendung auf [Wag10] verwiesen.

2.3.2.2. WS-Business Process Execution Language

Die Workflowsprache WS-Business Process Execution Language (WS-BPEL) wurde von IBM, Microsoft und anderen Unternehmen entwickelt und unter dem Dach der OASIS standardisiert [OAS07]. Sie ist kontroll-Fluss orientiert, arbeitet mit Variablen und zeichnet sich zusätzlich durch die Orchestrierung von WS-Aufrufen aus. Da sie in vielen WF-Systemen (IBM WebSphere Process Server[3], Microsoft BizTalk[4]) implementiert ist, ist sie der de facto Standard für die Modellierung von Geschäftsprozessen.

WS-BPEL Prozesse werden ebenfalls in XML geschrieben und die Sprache ist in XML-Schema spezifiziert, es existieren GUI-Editoren wie z.B. der BPEL-Designer [bpe]. Im Folgenden werden wir den deklarativen Aufbau eines BPEL Prozesses und anschließend die einzelnen Sprachelemente vorstellen.

[3]http://www.ibm.com/software/products/de/de/wps/
[4]http://www.microsoft.com/biztalk/

Aufbau eines BPEL-Prozesses

Ein BPEL Prozess besteht aus folgenden Abschnitten:

import In diesem Abschnitt werden WSDL und XML-Schema Definitionen eingebunden.

partnerLinks Dieser Abschnitt dient dazu, die verwendeten Webservices und Operationen einzubinden.

variables Die globalen Variablen und ihre XML-Typen werden in diesem Abschnitt definiert. WS-BPEL unterscheidet zwischen drei Klassen von XML-Typen: *WSDL-Nachrichten*, *XML-Schema-Elementen* und *XML-Schema-Typen*.

logic Die Beschreibung der eigentlichen Prozesslogik, welche auf globale *Variablen* und *partnerLinks* referenzieren kann, wird in diesem Abschnitt beschrieben.

BPEL-Sprachelemente

Die für die Beschreibung der Prozesslogik wichtigen Sprachelemente, die sog. *Aktivitäten* stellen wir kurz vor. Für alle weiteren, nicht erwähnten, Sprachelemente verweisen wir auf den Standard [OAS07].

ASSIGN ist für Zuweisungen, insbesondere Variablenzuweisungen, zuständig. Innerhalb einer ASSIGN-Aktivität können mehrere Zuweisungen erfolgen, diese werden durch COPY-Blöcke voneinander getrennt. Auf der linken Seite der Zuweisung (<*to*>) muss eine Variable stehen, auf der rechten Seite (<*from*>) sind des weiteren Literale (Initialwerte) und Ausdrücke (z.B. XPath, XQuery) erlaubt. Eine Variable kann zusätzlich um eine XPath-Selektion ergänzt werden, um nur bestimmte Teile eines XML-Dokuments zuzuweisen (Selektion auf rechter Seite), oder nur einen bestimmten Teil eines XML-Dokuments zu ersetzen (Selektion auf linker Seite). Des Weiteren können *partnerLinks* zugewiesen werden, hierfür verweisen wir ebenfalls auf den Standard.

IF ist für den klassischen Kontrollfluss zuständig. Hierbei wird ein boolscher Ausdruck einer Query-Sprache (XPath, XQuery etc.) evaluiert und entsprechend der *then* bzw. *else* Zweig durchlaufen.

FOREACH ist eine Schleife, der Start- und Endwert des Zählers wird über einen Ausdruck einer Query-Sprache ermittelt. Der Zähler wird in jedem Schleifendurchlauf um Eins erhöht, bis er den Endwert erreicht oder die optionale Bedingung zum vorzeitigen Beenden der Schleife erfüllt ist.

WHILE ist eine klassische *While*-Schleife, die solange durchlaufen wird, wie die Schleifenbedingung *wahr* ist. Die Bedingung wird vor Eintritt in den Schleifenrumpf evaluiert und ist ebenfalls ein boolscher Ausdruck einer Query-Sprache.

REPEAT UNTIL ist der *While*-Schleife sehr ähnlich, lediglich die Schleifenbedingung wird nach durchlaufen des Schleifenrumpfes evaluiert, was ein mindestens einmaliges durchlaufen des Schleifenrumpfes zur Folge hat.

ONALARM/WAIT können den Ablauf eines Prozesses pausieren. Nach einer bestimmten Zeitspanne, oder nach eintreten eines bestimmten Ereignisses, wird der Prozess fortgesetzt. Die Zeitspanne wird ebenfalls durch einen Ausdruck einer Query-Sprache bestimmt.

INVOKE ruft eine Operation eines WS auf, dieser muss dafür im Abschnitt *partnerLinks* des BPEL-Prozesses eingebunden worden sein. Es wird bestimmt welche Operation des WS aufgerufen wird, der Inhalt der Aufruf-Nachricht wird aus einer BPEL-Variablen entnommen. Der Inhalt der Ergebnis-Nachricht des WS wird ebenfalls in eine BPEL-Variable gespeichert und kann daraufhin weiter verarbeitet werden.

Allen ausgehenden Kanten aller Aktivitäten kann eine sog. *TransitionCondition* zugewiesen werden. Eine Ziel-Aktivität am Ende der Kante wird nur dann ausgeführt, falls die Bedingung der *TransitionCondition* erfüllt ist, ansonsten wird die weitere Verarbeitung an dieser Stelle gestoppt. Außerdem wird jede Aktivität und jede Variable einem sog. *Scope* zugeordnet. Diese *Scopes* sind im Wesentlichen vergleichbar mit Sichtbarkeitsbereichen von klassischen Programmiersprachen. Variablen und Aktivitäten, die z.B. innerhalb einer *FOREACH*-Schleife definiert und ausgeführt werden, sind nur innerhalb des von der Schleife definierten *Scopes* sichtbar.

Ein Beispiel für einen BPEL-Prozess ist im Anhang C.2 (Seite 151) zu finden. Es ist festzustellen, dass WS-BPEL sehr stark mit XML und seinen Verarbeitungsmöglichkeiten verwoben ist und nahezu alle Aktivitäten hiervon berührt werden.

2.3.3. Workflow Arten

In diesem Abschnitt möchten wir typische Arten von Workflows nach Ihren spezifischen Eigenschaften wie z.B. die Art ihrer Aktivitäten (und Dienste), Anwendungsbereichen, zu verarbeitende Datengrößen und der Art des Flusses klassifizieren.

2.3.3.1. Business-WFs

Typische Workflows im Geschäftsumfeld modellieren häufig wiederkehrende Arbeitsschritte eines Unternehmens [FL00]. Ein Beispiel ist der Versand von Waren nach Zahlungseingang. Der Workflow kann mehrmals am Tag ausgeführt werden, z.B. manuell durch einen Mitarbeiter ausgelöst oder automatisch zu bestimmten Uhrzeiten. Dann wird der Workflow über alle offenen Bestellungen iterieren, die Firmenkonten nach einer Zahlung durchsuchen und,

falls die zugehörige Zahlung eingegangen ist, die Daten der Lieferung an die Spedition weitergeben. Liegt keine Zahlung vor, könnte des Weiteren überprüft werden, ob eine Mahnung fällig ist und diese automatisch veranlasst werden.

Es geht bei diesen Workflows also hauptsächlich um Entscheidungen und daraus resultierenden Handlungen, hierfür sind kontroll-Fluss orientierte WF-Sprachen wie WS-BPEL ideal geeignet. Oft müssen nur identifizierende Daten, z.B. Kunden-, Auftrags- und Rechnungsnummern in der WF-Runtime gehalten werden, das Datenaufkommen in der Runtime kann also vergleichsweise gering gehalten werden. Die Laufzeiten entpsrechender WFs sind mit wenigen Ausnahmen häufig sehr kurz (innerhalb von Sekunden oder Minuten). Dafür ist die gleichzeitige, also parallele, Ausführung von Interesse, man stelle sich z.B. einen WF vor, der die Ticketbestellung für ein angesagtes Rock-Konzert abwickelt.

2.3.3.2. Wissenschaftliche-WFs

Wissenschaftliche-WFs (*engl. eScience-WFs*) finden vor allem innerhalb der Naturwissenschaften immer größere Bedeutung [Tay07]. Hauptsächlich geht es darum computergestützte Experimente oder Simulationen bzw. Auswertungen gesammelter Daten durchzuführen. Da es sich um wissenschaftliche Experimente handelt und somit oftmals keine Standardsoftware existiert, setzen sich die Berechnungen und anschließenden Analysen aus diversen Programmen zusammen. Zwischen diesen Programmen müssen ggf. Daten in ein anderes Format gebracht werden und/oder von einer Ressource auf eine andere Ressource übertragen werden um eine weitere Verarbeitung zu gewährleisten. Diese Prozesse können mit Workflows modelliert und orchestriert werden. Grundsätzlich kann man feststellen, dass der Fokus weniger auf Entscheidungen liegt, sondern auf den zu verarbeitenden Daten. Ohne Eingabedaten gibt es kein Experiment und ohne Ergebnisdaten keine Analyse. Man kann hier prinzipiell unterscheiden zwischen Workflows, die lediglich die Programme mit ihren Parametern in der richtigen Reihenfolge starten und bei denen diese Programme ihre Daten in externen Datenbanken oder auf dem Dateisystem abliegen, und Workflows, die diese Daten in der WF-Runtime verarbeiten und dort analysieren. Letztere führen also zu einem großen Datenaufkommen innerhalb der WF-Runtime. Vorallem, da diese Daten in der Regel wesentlich größer sind als bei Business-WFs. Daten-Fluss orientierte WF-Sprachen sind für eine solche Modellierung oft intuitiver anwendbar. Da man allerdings auch an einer Standardsprache für Workflows interessiert ist und sich der Datenfluss auch innerhalb eines Kontrollflusses darstellen lässt, rückt derzeit WS-BPEL ebenfalls als wissenschaftliche WF-Sprache in den Vordergrund [AMA06] [Slo07] [GHCM09] [GSK+11]. Man kann feststellen, dass Wissenschaftliche-WFs seltener, dafür mit größeren Datenmengen als Business-WFs ausgeführt werden. Eine parallele Ausführung dieser WFs macht nur in einigen speziellen Fällen Sinn.

2.3.3.3. Extraction Transformation Load-WFs

Extraction Transformation Load-WFs (ETL-WFs) orchestrieren ETL-Operationen. Diese können im Wesentlichen das Laden (Load/Retrieval) und Filtern einer Datenmenge sowie Verknüpfung (Join), Vereinigung (Union) und Zusammenführung (Merge) zweier Datenmengen sein. Die Ausführung mehrerer solcher Operationen (ETL-Prozess) kann durch Daten- oder Kontrollfluss beschrieben werden. ETL-WFs sind ETL-Prozesse, die mit Hilfe von WF-Technologie modelliert und ausgeführt werden. In einer kontroll-Fluss orientierten Sprache wie WS-BPEL werden diese ETL-Operationen als Aufruf von WSs oder in Aktivitäten eingebettete Datenverarbeitungsanweisungen realisiert. So wird z.B. der Inhalt einer Datei durch eine Anfrage an das Dateisystem in die WF-Engine geladen, oder es können SQL Anfragen an relationale DBSe gestellt werden, die entweder das Resultat (z.B. für Data Retrieval Anweisungen wie SELECT) oder eine Bestätigung der Ausführung (z.B. für DDL INSERT/UPDATE Anweisungen) an die WF-Engine liefern.

Die Arbeit [VSRM08] befasst sich mit SQL Fähigkeiten von gängigen Workflowsystemen. Hierbei werden die Ergebnisdaten eines SQL *SELECT* Ausdrucks z.B. zeilenweise in der Workflowengine verarbeitet. Somit besitzen diese Workflows ebenfalls ein erhöhtes Datenaufkommen innerhalb der WF-Runtime. Es bestehen bei solchen Workflows jedoch die Möglichkeit globale Optimierungen, z.B. durch den WF-Compiler oder einer Modelltransformation, vorzunehmen. Dadurch kann der Datentransfer zwischen Datenbank und WF-Engine zum Teil ganz vermieden werden [VSS+07]. In [RRS+10] wird ein allgemeines Gerüst (Framework) für ETL-Operationen in WF-Sprachen beschrieben und in Apache ODE prototypisch implementiert. Im Vergleich zu den Systemen aus [VSRM08] können, durch Erweiterung der SIMPL (SimTech - Information Management, Processes, and Languages) Data Mining Aktivitäten, nicht nur relationale DBS angesprochen werden, sondern alle denkbaren Datenquellen sowie Verknüpfungs- und Vereinigungsoperationen auf diesen heterogenen Datenquellen durchgeführt werden. ETL-WFs arbeiten fast ausschließlich auf Mengen von Daten, deshalb ist ihr Datenaufkommen entsprechend groß, die Daten können jedoch durch entsprechende Systeme und globale Optimierungen von der WF-Engine ferngehalten werden. Eine parallele Ausführung von ETL-WFs, zumindest bei Auswertung und Manipulation der gleichen Datensätze, führt zum gleichen Ergebnis und ist dadurch von geringerem Interesse.

2.3.3.4. Zusammenfassung

Wir fassen die Eigenschaften der vorgestellten WF-Typen in Tabelle 2.4 zusammen. Business-WFs haben typischerweise eine kurze Laufzeit und müssen ggf. auch parallel ausgeführt werden. Sie verarbeiten in der Regel kleinere Datenmengen als Wissenschaftliche- und ETL-WFs. Diese werden allerdings seltener ausgeführt und die parallele Ausführung spielt nur eine untergeordnete Rolle.

WF-Typ	Anwendungsbereich	Datengröße	Ausführungsart	Häufigkeit der Ausführung
Business	Geschäftsprozesse	klein	sequentiell / parallel	viele
Wissenschaftlich	Experimente und Simulationen	groß	oft nur sequentiell	wenige-viele
ETL	Analyse von Geschäftsdaten, Experimente und Simulationen	groß	sequentiell	wenige

Tabelle 2.4.: Vergleich der Eigenschaften und Anwendungsgebiete der verschiedenen Arten von Workflows.

2.4. Datenbanktechnologie

Die Datenbanktechnologie existiert nun seit über vier Jahrzehnten und bildet ein solides und anerkanntes Fundament innerhalb der Informatik, um große Datenmengen effizient zu speichern, zu selektieren und zu transformieren. Da diese Arbeit im Rahmen des Arbeitskreises „Anwendersoftware“ der Universität Stuttgart, der sich mit Datenbanken und Informationssystemen befasst, entstanden ist, setzen wir Grundkenntnisse in diesem Bereich voraus. Für eine Einführung in Datenbanken und Informationssysteme verweisen wir auf das Lehrbuch [AE09]. Dennoch möchten wir einige wichtige Begriffe kurz erläutern.

2.4.1. Datenbanksysteme

Ein Datenbanksystem (DBS) setzt sich aus einer Datenbank (DB) und dem Datenbank-Management-System (DBMS) zusammen. Die DB beinhaltet die Daten und Metadaten, das DBMS verwaltet die Zugriffe und Aktualisierungen auf die DB. Es existieren verschiedene Möglichkeiten Datenstrukturen zu modellieren und auf diese Daten zuzugreifen:

- Hierarchisch (Satzorientierter Zugriff)
- Relational (Mengenorientierter Zugriff)
- Objektrelational (Objekt-/Mengenorientierter Zugriff)

Einige der wichtigsten Eigenschaften, die für die Stabilität und Mehrbenutzerfähigkeit von DBMSen verantwortlich sind, sind die ACID-Eigenschaften des Transaktionskonzepts [TH01]:

Atomarität (*engl. Atomicity* „Alles oder nichts“) Die Ausführung und Veränderungen der Daten durch eine Transaktion findet entweder ganz oder gar nicht statt.

Konsistenz (*engl. Consistency*) Der Inhalt der Datenbank wird durch die Transaktion von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt. Insbesondere müssen hierbei Referenzen und Querverweise auf andere Daten der Datenbank ihre Gültigkeit beibehalten.

Isolation Sichert den logischen Einbenutzerbetrieb, obwohl mehrere Benutzer mit dem System arbeiten. Insbesondere darf innerhalb einer Transaktion nicht auf veraltete oder bereits überschriebene Daten anderer Transaktionen zugegriffen werden.

Dauerhaftigkeit (*engl. Durability*) Die Persistenz der Daten einer abgeschlossenen Transaktion muss gesichert werden, insbesondere beim Ausfall des Systems.

Am populärsten sind die relationalen Datenbanksysteme, sie finden heutzutage in fast jeder größeren Software, in der Daten gespeichert werden, Verwendung. Hierbei werden die Daten als *Zeilen* in *Tabellen* gespeichert. Die *Spalten* tragen die *Metainformation* (wie Typ und Semantik der Spalte), somit gehören relationale Datenbanken zum Typ der Strukturierten Daten (vgl. Tabelle 2.1, Seite 18).

Die Daten werden mit Hilfe der von ANSI und ISO[5] standardisierten Query-Sprache *SQL* selektiert und transformiert. Durch die massiv steigende Verwendung von Semi-Strukturierten Daten wie XML (siehe Kapitel 2.1) in den letzten 10 Jahren, ist der Bedarf an XML Verarbeitungsmöglichkeiten in Datenbanksystemen gestiegen. Hierbei gibt es zwei Ansätze:

Native-XML Datenbanken speichern und verarbeiten ausschließlich XML Dokumente mit Hilfe von XML Query-Sprachen.

XML-Enabled Datenbanken besitzen für die relationalen Tabellen einen zusätzlichen Spaltentyp *XML* indem XML Dokumente gespeichert werden können. Diese Felder können prinzipiell, z.B. mit XML Query-Sprachen, gesondert verarbeitet werden.

Im Folgenden möchten wir kurz zwei XML-Enabled Datenbanken vorstellen, die in dieser Arbeit verwendet wurden.

[5]http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45342

2.4.1.1. IBM DB2

Die *DB2* ist ein DBS der Firma *International Business Machines Corporation* (IBM) welches auf das *System R*, eine relationale Forschungsdatenbank aus Mitte der 1970er Jahre, zurückgeht. Die erste Version wurde 1983 eingeführt und wurde seitdem stetig weiterentwickelt. Seit 1997 ist die DB2 in der Version 9 verfügbar. Als Produktivsystem, welches auch in Großkonzernen eingesetzt wird, ist es ein sehr stabiles System. Es ist möglich, die DB2 als verteiltes Datenbanksystem zu verwenden. Es unterstützt eine Vielzahl der im SQL Standard vorgeschlagenen Funktionalitäten wie *User Defined Functions* (UDF), Speicherung von *Binary Large OBjects* (BLOB) und Funktionen für DataWarehousing und *OnLine Analytical Processing* (OLAP).

Als XML-Enabled DBS besitzt es seit der Version 9 den XML Spaltentyp und zahlreiche Funktionen zum Verarbeiten von XML Daten. Es ist möglich XPath und XQuery Ausdrücke alleinstehend oder innerhalb von SQL Ausdrücken zu evaluieren. Durch die von IBM eigens entwickelte Query-Sprache *pureXML* ([Che07], siehe Kapitel 2.1.4) ist es sogar möglich XML Dokumente zu transformieren, was mit XPath und XQuery nicht möglich ist.

Aktuelle Version: DB2 UDB 9.7
Verwendete Version: DB2 UDB 9.7
Offizielle Webseite: http://www.ibm.com/software/data/db2/
Historische Informationen: http://de.wikipedia.org/wiki/IBM_DB2

2.4.1.2. PostgreSQL

PostgreSQL ist ein kostenloses DBS, welches in den 1980er Jahren an der University of California in Berkeley entwickelt wurde. Seit 1997 wird es von einer OpenSource Community weiterentwickelt. Es gilt als schnelles und leichtgewichtiges DBS, das trotzdem viele Funktionen des SQL Standards implementiert.

Als XML-Enabled DBS existiert ebenfalls der XML Spaltentyp und die Möglichkeit mit der Funktion *xpath* XPath-Ausdrücke innerhalb von SQL Ausdrücken zu evaluieren. Es existiert jedoch keine Möglichkeit XQuery-Ausdrücke auszuwerten, eine eigene Transformationssprache wie *pureXML* bei IBM DB2 fehlt ebenfalls.

Aktuelle Version: PostgreSQL 9.0
Verwendete Version: PostgreSQL 8.4
Offizielle Webseite: http://www.postgresql.org/
Historische Informationen: http://de.wikipedia.org/wiki/Postgresql

2.5. Webservice und Workflow-Technologie für Proteinmodellierung

In diesem Abschnitt möchten wir aus dem Anwendungsbereich Bioinformatik einen Anwendungsfall (*engl. Use-Case*) für einen wissenschaftlichen WF (siehe Kapitel 2.3.3.2), der Daten innerhalb der WF-Runtime verarbeitet, vorstellen. Dieser wird in dieser Arbeit als Testfall für die Evaluation in Kapitel 7 dienen. In [Wag10] wurde das XML Schema *BioInformatics Interchange Format* (BIIF) als Austauschformat für Proteindaten entworfen und Webservices implementiert, die dieses Format als Ein- und Ausgabe verwenden. Außerdem wurde ein Anwendungsfall für das Workflowsystem Taverna vorgestellt und dieser ebenfalls als BPEL-Prozess modelliert. Bevor wir unseren Anwendungsfall vorstellen, gehen wir kurz auf ein paar Grundlagen ein.

2.5.1. Bioinformatik

Die Bioinformatik gehört zu den Life-Science Wissenschaften und befasst sich mit der computergestützten Forschung an biologischen Objekten und Systemen, wie etwa Proteine und ihre Rolle im Stoffwechsel von Organismen. Ein Protein besteht aus einer Verkettung von 22 möglichen Aminosäuren. Diese lineare Anordnung wird auch Primärstruktur genannt und bestimmt die Faltung in Sekundärstrukturen (α Helix, β Faltblatt und Schleifen) die wiederum die räumliche 3D Struktur des Proteins bestimmen (siehe Abb. 2.2). Die Primärstruktur eines Proteins, auch als Proteinsequenz bezeichnet, kann man als Zeichenkette darstellen (siehe Listing 2.7).

```
1 MALSQSVPFS ATELLLASAI FCLVFWVLKG LRPRVPKGLK SPPEPWGWPL LGHVLTLGKN PHLALSRMSQ RYGDVLQIRI
2 GSTPVLVLSR LDTIRQALVR QGDDFKGRPD LYTSTLITDG QSLTFSTDSG PVWAARRRLA QNALNTFSIA SDPASSSSCY
3 LEEHVSKEAK ALISRLQELM AGPGHFDPYN QVVVSVANVI GAMCFGQHFP ESSDEMLSLV KNTHEFVETA SSGNPLDFFP
4 ILRYLPNPAL QRFKAFNQRF LWFLQKTVQE HYQDFDKNSV RDITGALFKH SKKGPRASGN LIPQEKIVNL VNDIFGAGFD
5 TVTTAISWSL MYLVTKPEIQ RKIQKELDTV IGRERRPRLS DRPQLPYLEA FILETFRHSS FLPFTIPHST TRDTTLNGFY
6 IPKKCCVFVN QWQVNHDPEL WEDPSEFRPE RFLTADGTAI NKPLSEKMML FGMGKRRCIG EVLAKWEIFL FLAILLQQLE
7 FSVPPGVKVD LTPIYGLTMK HARCEHVQAR LRFSIN
```

Listing 2.7: Proteinsequenz des Proteins aus Abb. 2.2 als Zeichenkette. Quelle: [Wag10]

Proteine, welche die gleichen Eigenschaften besitzen, z.B. die gleiche chemische Reaktion katalysieren, werden zu sog. *Proteinfamilien* zusammengefasst. Oft geht dies mit einer sehr ähnlichen Proteinsequenz einher. Die *DataWarehouse Architecture for pRotein classiFication (DWARF)* [FTGP06] sucht aus externen Protein-Datenquellen wie Genbank [BKML+10] solche ähnlichen Proteinsequenzen und fasst sie in einer Datenbank zusammen. In [Wag10] wurden für das DWARF-System Webservices zum Abfragen und Verändern dieser Datenbankinhalte implementiert. Derzeit ist die CYPED[6] Datenbank [SWLP09] über diese

[6] http://www.cyped.uni-stuttgart.de/

Abbildung 2.2.: 3D Struktur eines Cytochrome P450 Proteins. Rot: α Helix, Gelb: β Faltblatt, Grün: Schleifen. Quelle: [Wag10]

Webservices auslesbar. Für weiterführende Informationen zur Biochemie sei auf das Lehrbuch [JMB07] und für Informationen zur WS und WF-Technologie für Proteinmodellierung auf [Wag10] verwiesen.

2.5.1.1. Anwendungsfall Mustersuche

Eine innerhalb der Bioinformatik annerkannte Analysemöglichkeit stellt die Mustersuche (*engl. Pattern-Matching*) dar. Hierbei wird in den als Zeichenketten gespeicherten Proteinsequenzen nach einem bestimmten Muster, z.B. in der Form eines regulären Ausdrucks, gesucht. Diese Muster dienen zum Erkennen wichtiger Regionen innerhalb einer Proteinfamilie, z.B. lassen sich so die für die chemische Reaktion wichtigen Aminosäuren lokalisieren.

Die auszuführenden Aufgaben bieten sich ideal an, um diese durch einen Workflow zu modellieren, außerdem kann man bei Verwendung von WS-BPEL die datenverarbeitenden Schritte in die WF-Engine ziehen. Aus diesem Grund ist dieser Anwendungsfall für die Messungen in Kapitel 7 von besonderem Interesse.

Für die Messungen verwenden wir das folgende Muster[7] [NG87], welches bereits in einen regulärer Ausdruck umgeformt wurde:

```
[FW][SGNH].[GD][^F][RKHPT][^P]C[LIVMFAP][GAD]
```

Wir beschreiben nun Schrittweise den zugehörigen BPEL-Prozess der graphisch in Abb. 2.3 dargestellt ist, zudem ist der in BPEL definierte Prozess im Anhang C.2 (Seite 151) zu finden.

recieveInput empfängt die Parameter ID (der Identifier der CYPED Unterfamilie die durchsucht werden soll) und PATTERN (das zu suchende Muster als regulären Ausdruck).

Initialisation initialisiert einige BPEL-Variablen und WSDL Nachrichten, insbesondere die WSDL Nachricht an den DWARF-Webservice, der als Parameter die ID der Unterfamilie benötigt.

getSuperFamilySequences ruft die Operation *getSFamilyAlignment* des Webservice *DWARF_ACCESS* auf (siehe [Wag10]).

AssignWSResponse weist das Ergebnis des WS einer BPEL-Variable zu.

ForEachProteinSequence diese Foreach Schleife wird für jede Proteinsequenz einmal durchlaufen.

PrepareProteinSequence bereinigt die aktuelle Proteinsequenz von störenden Sonderzeichen, die aus der Darstellung des WS-Resultats herrühren und speichert die bereinigte Proteinsequenz in einer BPEL-Variable ab.

IfPatternMatches prüft nun ob der reguläre Ausdruck auf die aktuelle Proteinsequenz zutrifft, falls ja wird der Bezeichner der Proteinsequenz an eine Variable vom Typ *xsd:string* konkateniert und der Zähler für positive Proteinsequenzen (die das Muster enthalten) um Eins erhöht. Andernfalls wird der Zähler für negative Proteinsequenzen (die das Muster nicht enthalten) um Eins erhöht.

PrepareOutput stellt die Antwortnachricht zusammen, bestehend aus negativem und positivem Zähler, sowie allen Bezeichnern der Proteine welche das Muster enthalten.

replyOutput sendet die Antwortnachricht.

Nach Ausführung des Workflows erhält man also eine Statistik über die Anzahl der positiven und negativen Proteinsequenzen bezüglich des vorgegebenen Musters und die Bezeichner der positiven Proteinsequenzen zurück.

[7] http://expasy.org/prosite/PDOC00081

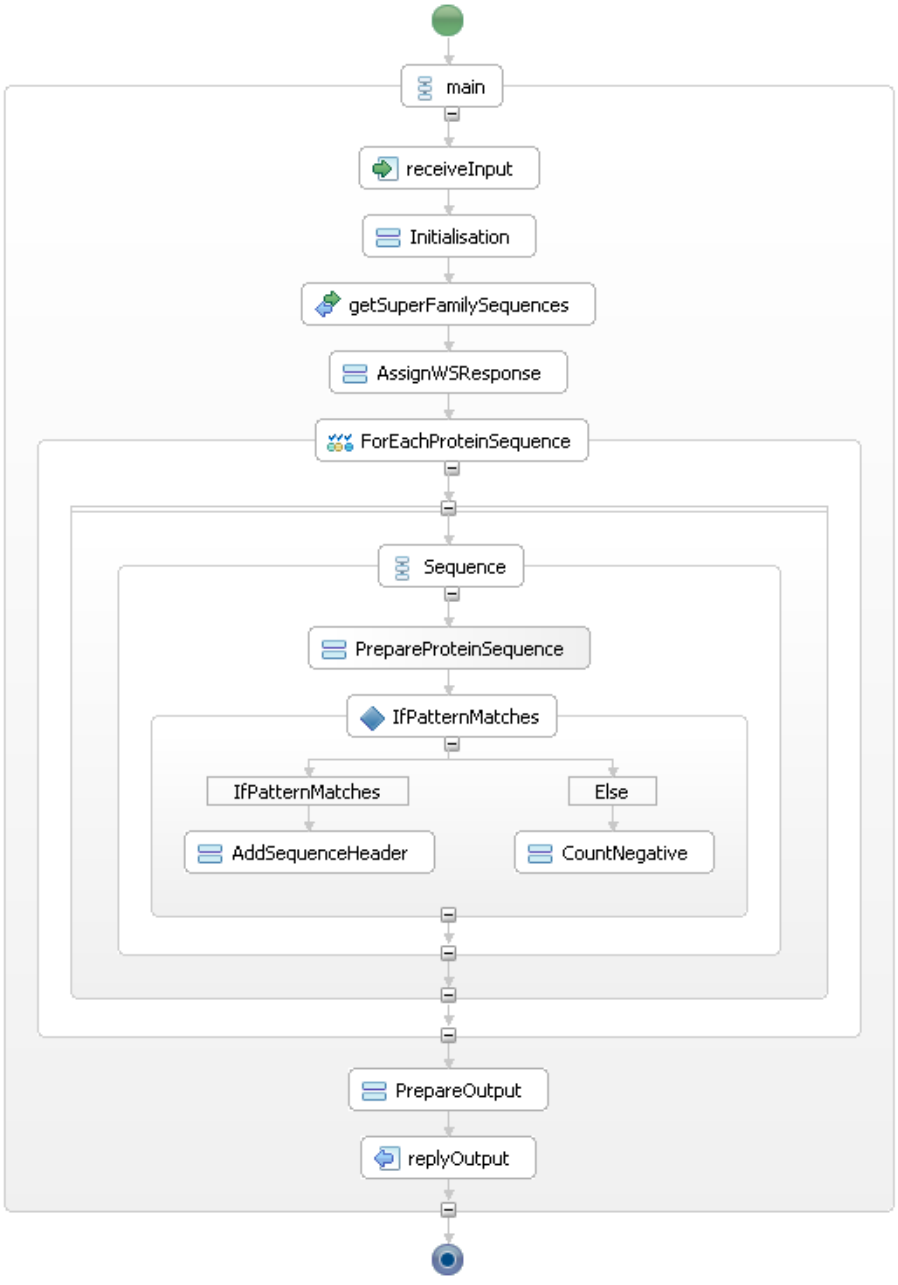


Abbildung 2.3.: Graphische Repräsentation des in WS-BPEL definierten Anwendungsfalls zur Mustersuche in Proteinsequenzen.

3. Workflow Architekturen und Datenbank Integration

Dieses Kapitel wird einen Einblick in die allgemeine Architektur von WfMSen geben. Es wird zuerst das Workflow Referenz Modell aus dem Jahr 1995 vorgestellt [Hol95], anschließend werden Möglichkeiten der Integration von Datenbanktechnologie in WfMSe [VSRM08] [RRS+10] und der Ansatz vorgestellt Datenbanken und WfMSe miteinander zu verschmelzen [AIL98] [SKDN05]. Im letzten Abschnitt dieses Kapitels wird eine Auswahl an WfMSen vorgestellt und wie diese ihre integrierte Datenbank verwenden.

3.1. Workflow Reference Model

Das Workflow Referenz Modell [Hol95] befasst sich mit dem Aufbau, Nutzen und der Funktionsweise von Workflow Management Systemen (WfMS). Der schematische Aufbau dieses Modells ist in Abb. 3.1 zu sehen. Es besteht aus der zentralen Workflow-Engine und fünf weiteren Schnittstellen. Die WF-Engine, die Prozess-Definitions Tools und die Administrations & Monitor Tools bilden das eigentliche WfMS (vgl. Kapitel 2.3.1, Seite 28). Die weiteren Schnittstellen sind für die Aufrufe der zu orchestrierenden Dienste und Programme oder anderer Workflow-Systeme zuständig. Im Folgenden werden die einzelnen Komponenten und Schnittstellen etwas näher erläutert.

Die Workflow API legt das Protokoll und die Austauschformate fest, mit denen die weiteren Komponenten mit der Workflow-Engine kommunizieren.

Die Workflow-Engine führt Instanzen der Workflows aus. Dazu navigiert sie durch die Workflow-Graphen und ruft entsprechende Dienste der Schnittstellen 2-4 auf.

Prozess-Definitions Tools werden zur Erstellung der Workflows benötigt, z.B. mit Hilfe einer GUI oder direkt in der WF-Sprache. Anschließend wird der WF der WF-Engine bekannt gemacht (*engl. deploy*), damit dieser von Anwendern oder anderen Systemen entsprechend aufgerufen werden kann. Hierbei werden die zum WF gehörenden Dokumente (z.B. die Beschreibung in einer WF-Sprache) entweder in internes Format der WF-Engine compiliert oder für die interpretative Ausführung entsprechend archiviert.

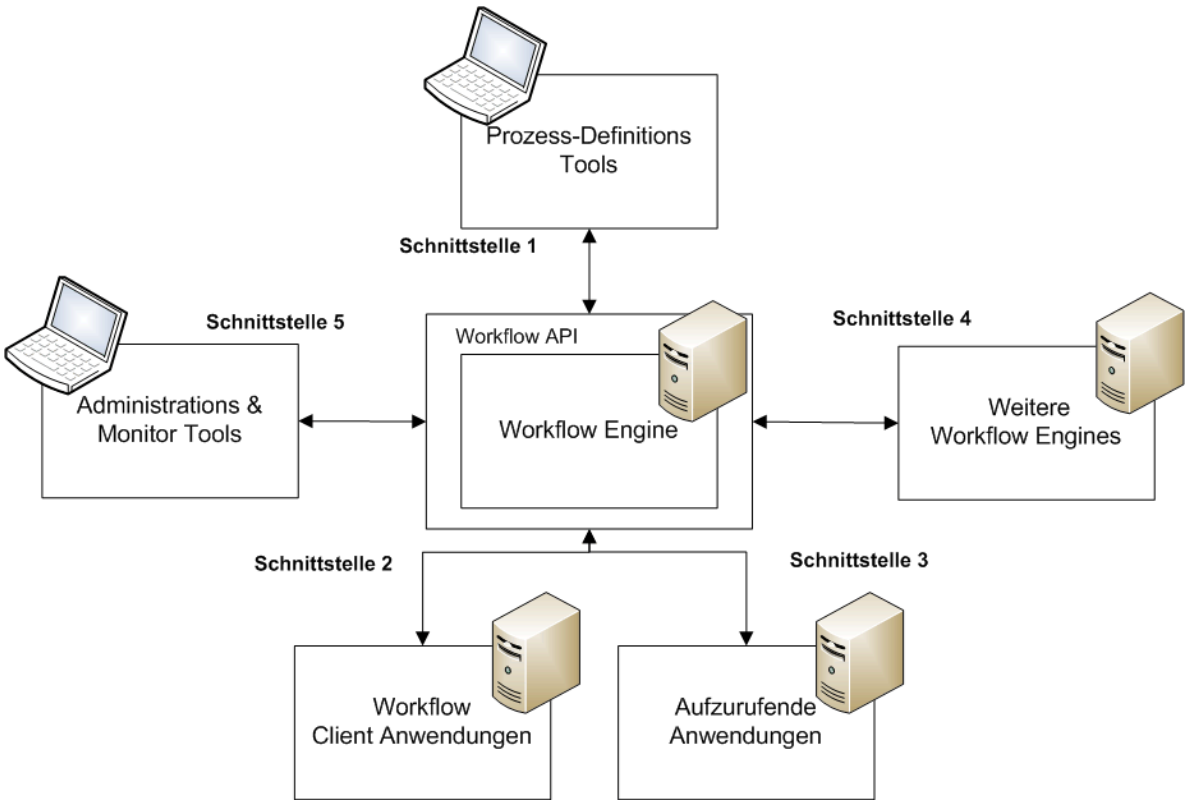


Abbildung 3.1.: Das Workflow Referenz Modell, gut zu erkennen ist die Aufteilung in Build-time (Prozess-Definition), Runtime (Workflow-Engine) und die Monitor Anwendungen (Administration & Monitor Tools). Des Weiteren die Schnittstellen zu ausführenden Programmen und anderen Workflow-Engines. Vgl. [Hol95]

Administrations & Monitor Tools stellen Funktionen zur Verfügung, mit denen das WfMS und die Ausführung von WF-Instanzen überwacht werden können.

Schnittstellen 2-4 stellen verschiedene Möglichkeiten dar, welche Dienste innerhalb eines WFs aufgerufen werden können. Dies können z.B. andere als WF definierte Prozesse sein (Schnittstelle 4) oder Programme und Dienste wie WSs (Schnittstelle 3) oder sonstige Client-Anwendungen (z.B. GUI-Anwendung zum Instanziieren eines WFs) des WfMSs (Schnittstelle 2).

In der Streitschrift [AS96] werden Probleme und Nutzen der damaligen WfMS angesprochen. Einerseits wird die Stabilität, Transaktions- und Mehrbenutzerfähigkeit (also Ausführung paralleler Instanzen) damaliger WfMS kritisiert und deren Nähe zu (*Aktiven*) Datenbanken thematisiert, welche die angesprochenen Probleme schon gelöst haben. Auf den Ansatz der Verschmelzung beider Systeme [AIL98] gehen wir im Kapitel 3.2.3 näher ein.

Heutzutage existieren WfMSe immer noch getrennt von Datenbanksystemen, allerdings werden DBSe wie in vielen anderen Softwarearchitekturen auch in WfMSen eingesetzt, um Daten zu Prozessen und laufenden Instanzen (wie Variableninhalte, Startzeiten und Anzahl laufender Instanzen etc.) persistent zu halten. Dadurch können Instanzen nach Ausfall des Systems am Punkt des Abbruchs wieder aufgenommen werden und bei Fehlern die Ausführung nachvollzogen werden (z.B. mit Monitortools). Somit kann man die zentrale Architekturkomponente der WF-Engine aus Abb. 3.1 um eine integrierte DB erweitern (siehe Abb. 3.2). Bis auf wenige Ausnahmen leichtgewichtiger WF-Engines, für z.B. Smartphones [GPW+07] [HHGR06], verwenden alle WF-Engines eine integrierte DB.

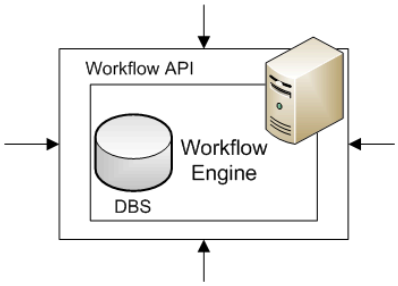


Abbildung 3.2.: Typische Workflow-Engine mit integriertem DBS für die Speicherung von Prozess und WF-Instanz Daten.

So konnten einige der in der Streitschrift angesprochenen Probleme, wie Stabilität, Fehlertoleranz, Skalierbarkeit, Performanz, Mehrbenutzerfähigkeit und Flexibilität teilweise behoben werden. In der Gesamtheit betrachtet, können aber auch heutzutage die WfMSe in den genannten Bereichen nicht mit gängigen DBSen konkurrieren. Weshalb weiterhin eine noch stärkere Integration dieser beiden Systeme von großem Interesse ist. Dies ist einer der Gründe, warum wir in dieser Arbeit untersuchen, welche Funktionen einer WF-Engine auf ihr integriertes DBS übertragbar sind und unter welchen Umständen dies zu einer verbesserten Ausführung von WF-Instanzen führt.

3.2. Arbeiten und Ansätze zur Datenbankintegration

Es soll nun ein Einblick in den Stand der Forschung zur stärkeren Integration von DBSen in WfMSe gegeben werden bzw. welche Funktionalitäten von anderen Systemen oder Ansätzen angeboten werden.

3.2.1. BPEL/SQL Funktionalität

In [VSRM08] werden die WfMSe von IBM (*WebSphere Process Server*), Microsoft (*Workflow Foundation*) und Oracle (*Oracle SOA Suite*) qualitativ auf ihre Möglichkeiten für *Inline SQL*

Support, also im Workflow eingebettete Datenverarbeitungsanweisungen, untersucht. Bis auf die *Workflow Foundation* verwenden diese Systeme WS-BPEL als WF-Sprache in jeweils erweiterter Form. Die *Workflow Foundation* stellt jedoch Import und Export Funktionen zur Verfügung, um BPEL Prozesse auf das interne Modell zu übersetzen und umgekehrt. *BPEL/SQL* bezeichnet hierbei, dass für ein WS-BPEL kompatibles WfMS entsprechende *Inline SQL*-Erweiterungen bzw. Funktionen angeboten werden.

Ausdrücklich wurden *SQL Inline* Aktivitäten untersucht, die im Gegensatz zu *SQL Adaptern* eigenständige Aktivitäten darstellen. Diese senden SQL-Anweisungen an ein DBS und werden entsprechend vom WfMS implementiert. Adapter hingegen rufen Webservices auf, die dann SQL Anfragen ausführen und die Ergebnisse in einem eigenständigen (XML) Format als SOAP Nachricht zurückgeben. Solche Adapter wurden für den Anwendungsfall der Proteinmodellierung (siehe Kapitel 2.5, Seite 38) in der Arbeit [Wag10] für das DWARF System erstellt. Die SQL Aufrufe werden bei beiden Methoden (*Inline SQL*, *Adapter*) auf externen Datenbanken ausgeführt, also nicht auf der integrierten WF-Datenbank.

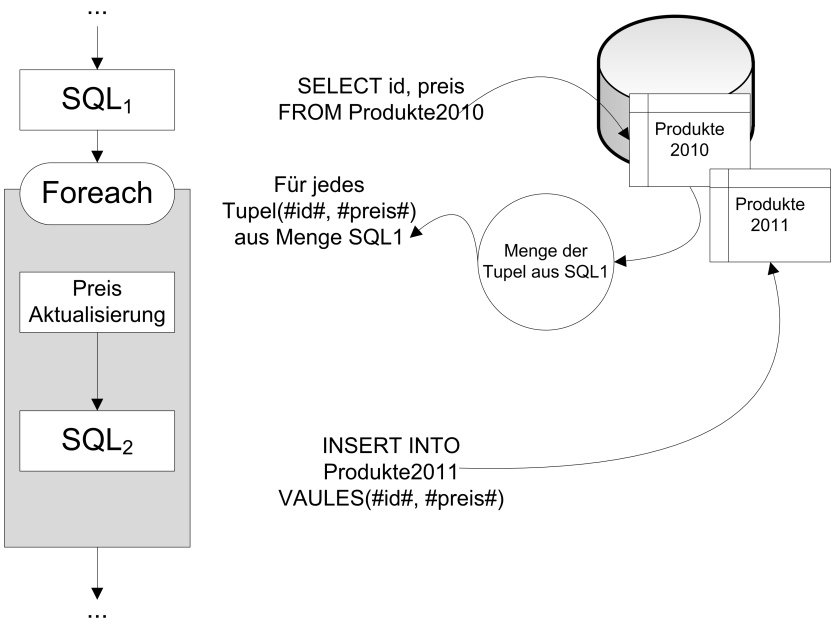


Abbildung 3.3.: Ein Beispiel-Workflow mit *Inline SQL* Aktivitäten. Hierbei werden Produktpreise einer externen Datenbank aktualisiert, indem die aktuellen Preise zuerst geladen werden, anschließend in einer *Foreach* Schleife angepasst und schließlich in die Datenbank zurückgespeichert werden.

Betrachten wir nun folgendes Workflow-Beispiel aus Abb. 3.3. Es enthält eine *Inline SQL* Aktivität (SQL_1), welche aus einer Datenbanktabelle *Produkte2010* zu allen Produkten den

Primärschlussel (id) und den Preis des Produkts ermittelt. Die Ergebnismenge wird im Workflow in einer Prozessvariable zwischengespeichert. Die *Foreach* Schleife wird für jedes Ergebnistupel in dieser Menge, also für jedes Produkt, durchlaufen. In der stilisierten Aktivität *PreisAktualisierung* wird nun der Preis des Produkts angepasst (in WS-BPEL könnte hier eine ASSIGN Aktivität oder ein Webservice Aufruf erfolgen). Anschließend wird mit der zweiten *Inline SQL* Aktivität das Produkt mit dem neuen Preis in die Tabelle *Produkte2011* eingefügt.

Die drei betrachteten Systeme unterscheiden sich hierbei, wenn man von der Art der Implementierung der *Inline SQL* Aktivitäten absieht, nur in zwei Kriterien:

1. Die Ergebnismenge der SQL_1 Aktivität wird entweder in einer temporären Tabelle der externen Datenbank (*IBM*) gespeichert und dann vollständig für die Verarbeitung in eine Prozessvariable des WF-Engine geladen, oder direkt in eine Prozessvariable und somit innerhalb der integrierten WF-Datenbank gespeichert (*Microsoft, Oracle*).
2. Die Bindung der *Inline SQL* Aktivitäten an die externe Datenbank kann statisch (*Microsoft, Oracle*) oder dynamisch (*IBM*) erfolgen.

Ein erweiterter Ansatz zu BPEL/SQL wird in [RRS+10] vorgestellt. Das dort vorgestellte *SIMPL-Framework* (SimTech - Information Management, Processes, and Languages) bietet die gleichen Möglichkeiten wie die bereits vorgestellten BPEL/SQL Ansätze, ist jedoch vom eigentlichen WfMS unabhängig und kann somit in jedes WfMS integriert werden. Es bietet insbesondere auch Zugriffsmöglichkeiten auf weitere Datenquellen, wie z.B. Dateisysteme. Das System besteht aus dem SIMPL-Kern, der in die Kommunikationsinfrastruktur des WfMSs eingebettet wird. Dort werden die Schnittstellen auf die verschiedenen Datenquellen (relationales DBS, Dateisystem etc.) als Operatoren (Data Access Operations) implementiert. Für das jeweilige WfMS können nun Aktivitäten, als sog. *Plug-Ins*, implementiert werden, die diese Operatoren aufrufen (siehe Abb. 3.4). Interessant ist hierbei die Möglichkeit, ETL-Operatoren (siehe Kapitel 2.3.3.3, Seite 34) auf heterogene Daten anzuwenden, so können z.B. Daten die im CSV-Format auf dem Dateisystem abliegen, mit Daten aus einem relationalen DBS verknüpft, vereinigt und zusammengeführt werden. Des Weiteren ist es möglich Daten aus den verschiedenen Quellen ineinander zu überführen, z.B. CSV-Daten in ein relationales Modell und umgekehrt abzuspeichern. Bisher wurde der SIMPL-Kern an die WF-Engine Apache ODE angebunden.

3.2.2. Process Graph Model Optimierung

Für die *Inline SQL* Aktivitäten aus Kapitel 3.2.1 ergeben sich globale Optimierungsmöglichkeiten, die in [VSS+07] durch den vorgestellten Process Graph Model (PGM) Optimierer durchgeführt werden. Dies steht im Gegensatz zu dem Ansatz dieser Arbeit, in der wir die integrierte DB des WfMSs nutzen möchten um während der Workflow-Ausführung

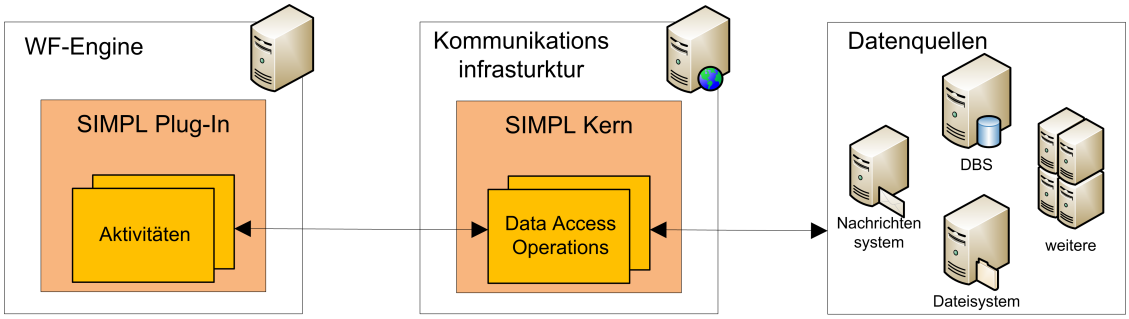


Abbildung 3.4.: Architektur des SIMPL-Frameworks. Die im WfMS implementieren *Plug-Ins* kommunizieren mit dem SIMPL-Kern, bzw. mit den dort implementierten *Data Access Operations*, die Zugriff auf eine Vielzahl von heterogenen Datenquellen erlauben. Vgl. [RRS+10]

lokale Optimierungen zu erzielen. *PGM* ist ein Graph-Modell, in den Workflows allgemein überführt werden können (also auch ein BPEL/SQL WF), hierzu berücksichtigt *PGM* die Aktivitäten als Knoten, deren Daten- sowie Kontrollflusskanten, möglich vorkommende Variablen und sog. *Partner*, also Aufrufe externer Diensten (wie WSs und DBSe). Dadurch besitzt ein *PGM*-Graph alle nötigen Informationen für eine semantisch korrekte Optimierung. Der PGM Optimierer arbeitet dann nach folgendem Schema (siehe Abb. 3.5):

1. Übersetzen eines BPEL/SQL Workflows in das PGM Modell
2. Der PGM Graph wird durch bestimmte Regeln und eine Kontrollstrategie optimiert
3. Der optimierte PGM Graph wird in ein BPEL/SQL Workflow rückübersetzt

Die *Kontrollstategie* durchläuft im Wesentlichen die vorliegende Regelmenge in einer vorgegebenen Reihenfolge und prüft die Anwendbarkeit auf den PGM-Graph. Dadurch wird gewährleistet, dass nach Anwendung einer Regel möglicherweise weitere Regeln anwendbar sind, um einen stärker optimierten Graph zu erhalten. Die Regeln bestehen jeweils aus einer *Bedingung* und einer *Aktion*, nur wenn die Bedingung erfüllt ist, werden die daran beteiligten Aktivitäten (die vom Kontrollfluss aus betrachtet vor und nach der aktuell betrachteten Aktivität auftreten können) durch die Aktion in eine entsprechend veränderte Aktivität überführt. Wichtigen und interessante Regeln sind hierbei:

Assign-Pushdown Die Verwendung einer WF-Variablen innerhalb einer SQL-Anweisung wird durch ihre vorhergehende Definition ersetzt (BPEL-ASSIGN). Dies kann insbesondere auch eine SQL-Anweisung sein, falls der Inhalt der WF-Variablen durch einen solchen Ausdruck geladen wurde.

Webservice-Pushdown Der Aufruf eines WSs wird in die SQL-Anweisung integriert, anstatt diesen über die WF-Engine auszuführen.

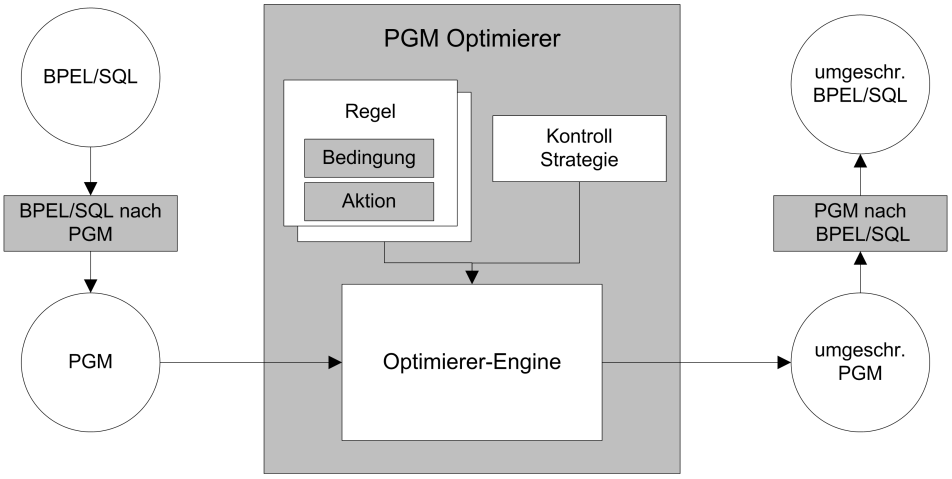
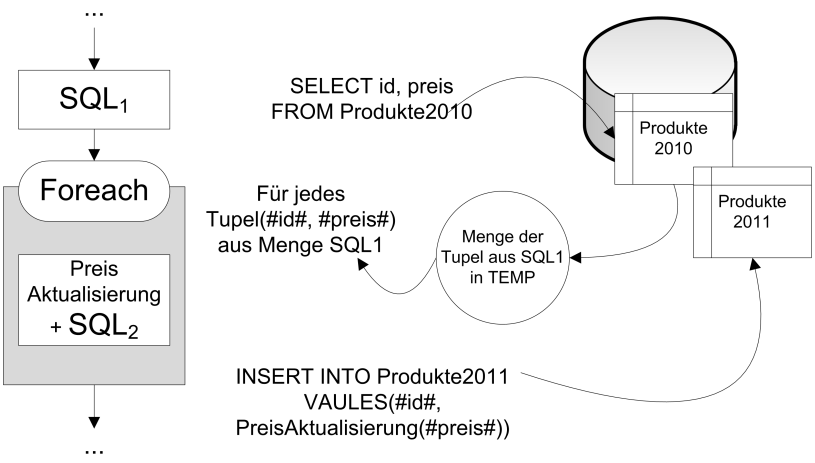


Abbildung 3.5.: BPEL/SQL WFs werden zuerst in das interne PGM Modell überführt, der PGM-Graph wird dann durch Regeln und eine Kontrollstategie optimiert und wieder in einen entsprechenden BPEL/SQL WF überführt. Vgl. [VSS+07]

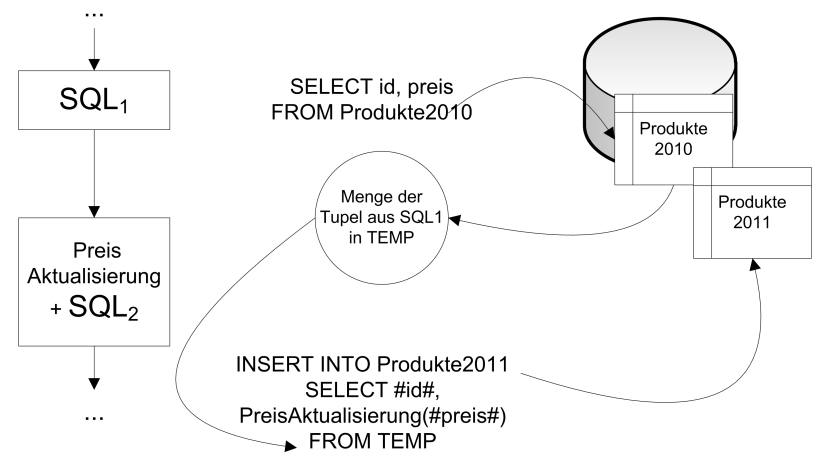
Tupel-to-Set Die tupelweise Verarbeitung innerhalb einer Schleife wird in eine SQL-Mengenoperation überführt.

Wir werden nun die grundsätzliche Funktionsweise und das Zusammenspiel ausgewählter Regeln anhand des Beispiels aus Abb. 3.3 vorstellen. Betrachten wir den ersten Optimierungsschritt (Abb. 3.6a), hier wird die verarbeitende Aktivität (*PreisAktualisierung*) in Form einer User Defined Function (UDF) innerhalb des DBSs ausgewertet. Die Funktion *PreisAktualisierung* erscheint im *INSERT* Ausdruck der *Inline SQL* Aktivität (SQL_2). Stellt diese Funktion ein Aufruf an einen Webservice dar, wird der bereits vorgestellte *Webservice-Pushdown* angewandt. Falls die Funktion (*PreisAktualisierung*) eine Zuweisung ist, wird entsprechend der Assign-Pushdown verwendet.

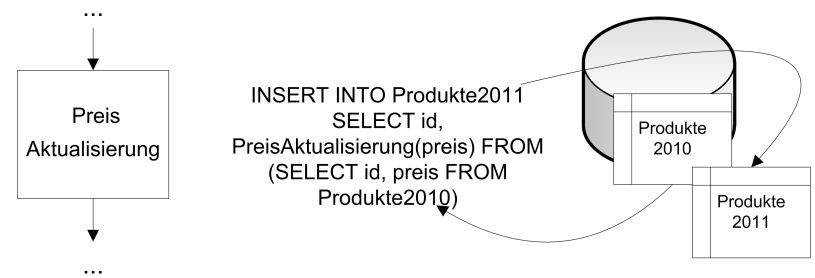
Der zweite Optimierungsschritt (Abb. 3.6b) eliminiert die *Foreach* Schleife. Dazu werden die Ergebnistupel der *Inline SQL* Aktivität SQL_1 nicht mehr tupelweise (*VALUES*) durch mehrmaliges Aufrufen von SQL_2 aus Abb. 3.6a, sondern durch eine Mengenoperation (*SELECT*) innerhalb der SQL-Anweisung SQL_2 aus der temporären Datenbanktabelle geladen. Insbesondere kann dadurch die *Foreach*-Schleife und das Laden der temporären Tabelle in eine WF-Variable eliminiert werden. Die zugehörige Regel lautet *Insert Tupel-to-Set*, die zur Gruppe der bereits angesprochenen Tupel-to-Set-Regeln gehört. Durch die Eliminierung der Schleife findet nun die Verarbeitung durch einen einzigen SQL Ausdruck mengenorientiert im DBS statt und es müssen keine Daten mehr zwischen Datenbank und WF-Engine ausgetauscht werden. Der dritte und letzte Optimierungsschritt (Abb. 3.6c) ersetzt noch die temporäre Tabelle in SQL_2 durch den SQL Ausdruck von SQL_1 da die temporäre Tabelle nicht länger benötigt wird. Die zugehörige Regel lautet *Eliminate Temporary Table*.



(a) Beispiel Prozess nach erstem Optimierungsschritt



(b) Beispiel Prozess nach zweitem Optimierungsschritt



(c) Beispiel Prozess nach letztem Optimierungsschritt

Abbildung 3.6.: Optimierung des BPEL/SQL Workflows aus Abb. 3.3

Das Papier [VSS+07] stellt noch weitere Regeln und Optimierungsstrategien vor. Messungen zeigen, dass insbesondere die *Insert Tuple-to-Set* Regel einen starken Performanz Vorteil bringt. Je nach zu verarbeitender Datengröße beträgt der Beschleunigungsfaktor der WF-Laufzeit zwischen 245 und 15000. Diese Werte sind durchaus beeindruckend und zeigen die Stärke mengenorientierter Verarbeitung innerhalb moderner DBSe.

3.2.3. Datenbank als Workflowsystem erster Klasse

Ein ganz anderer Ansatz der stärkeren Integration von Datenbank- und Workflowsystemen versucht die WF-Engine innerhalb des Datenbanksystems zu realisieren, hierbei fällt also die Implementierung einer eigenenständigen WF-Engine weg [AIL98] [SKDN05]. Dieser Ansatz wird oft mit dem Schlagwort *Aktive Datenbanken* in Verbindung gebracht [AS96] [PD99].

Vor allem bei wissenschaftlichen WFs, die stark daten-orientiert sind, spielen die effiziente Verarbeitung großer Datenmengen, die Nachvollziehbarkeit der Experimente und anschließende Analysen der Daten eine wichtige Rolle. Da nahezu alle WF-Engines über einer Datenbank aufgebaut sind, liegt die Idee nahe, die Funktionen der WF-Engine in die Datenbank zu übernehmen. So fällt einerseits die zusätzliche Kommunikation zwischen den Softwaremodulen weg und die Daten sind durch die ACID-Eigenschaften des DBSs zugleich persistent. Das Papier [AIL98] fordert DBSe als *Erste-Klasse* WfMS zu betrachten (siehe Abb. 3.7) und stellt einen Prototypen auf dem objektrelationalen DBS *Horse* vor.

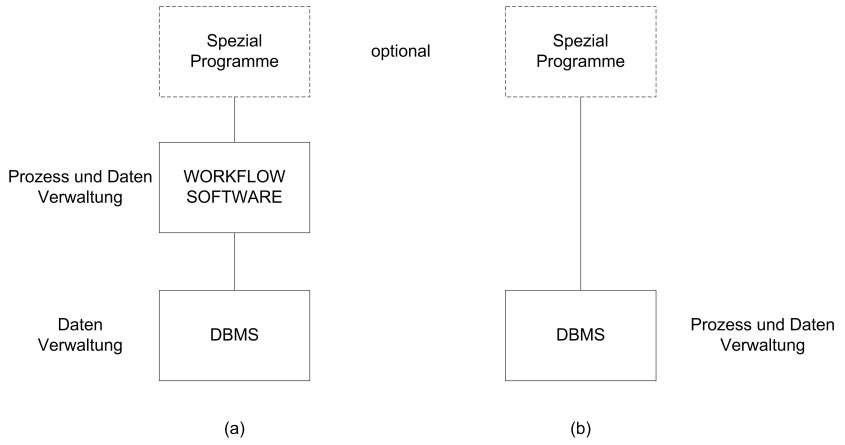


Abbildung 3.7.: Klassische WfMS Architektur (a) und DBMS als Erste-Klasse WfMS (b). Vgl. [AIL98]

Die grundlegende Idee ist, einen Workflow in ein entsprechendes Datenbankschema zu überführen, Tabellen stellen somit Aktivitäten dar. Die Daten die in diesen Aktivitäten verarbeitet oder generiert werden, werden in dieser Tabelle gespeichert. Die Verarbeitungsschritte werden über Auslöser (*engl. Trigger*) gestartet und so z.B. ein Webservice oder externes

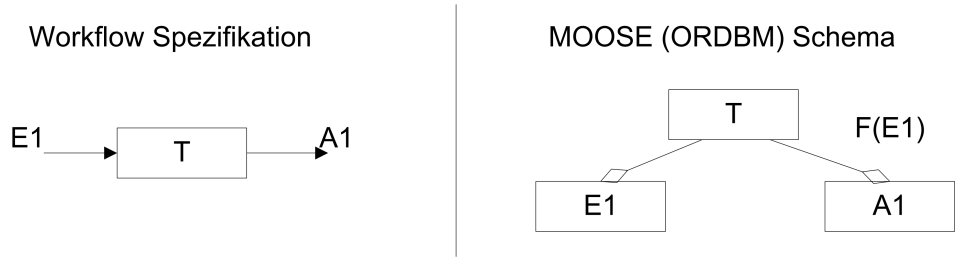


Abbildung 3.8.: Der Workflow wird in ein ORDBM Schema gebracht, die Eingabe *E1* und die Ausgabe *A1* werden als abgeleitete Tabellen der Aktivitätstabelle *T* modelliert. Einfügen von Daten in *E1* triggert die Transformation *F(E1)* und speichert das Ergebnis in Tabelle *A1* ab. Vgl. [AIL98]

Programm aufgerufen. Um nach Beenden einer Aktivität die nächste Aktivität auszulösen, wird entweder eine UDF aufgerufen (kontroll-Fluss orientiert) oder weiter zu verarbeitende Daten als Zeile in die nächste Aktivitätstabelle eingefügt, was einen Trigger zur Verarbeitung auslöst (daten-Fluss orientiert).

In Abb. 3.8 wird ein Beispiel gegeben, wie man eine Workflow Aktivität in das entsprechende Datenbankschema (Moose) der Horse Datenbank überführt. Da es sich hierbei um daten-Fluss orientierte Workflows handelt, laufen über die Workflowkanten Daten, die mit Eingabe (E) und Ausgabe (A) bezeichnet werden. Die Aktivität *T* verarbeitet die Eingabe *E1* und gibt die Ausgabe *A1* weiter. Im Moose Schema existiert die Tabelle *T*, welche die Funktionalität der Aktivität *T* repräsentiert. Die Tabellen *E1* und *A1* sind nach dem objektrelationalen Modell von *T* abgeleitet. Sobald ein Eingabedatum in Tabelle *E1* gespeichert wird, wird der entsprechende *Trigger* der Tabelle *T* aktiv, transformiert das Eingabedatum ($F(E1)$) und speichert das Ergebnis in Tabelle *A1* ab. Diese Tabelle *A1* kann nun wieder als Eingabe für eine folgende Aktivität dienen und löst deren Verarbeitungsschritt aus. Für alle weiteren Übersetzungen wie Verzweigungen, Schleifenkonstrukte und Ähnliches sei auf [AIL98] verwiesen.

Um ein solches System zu realisieren müssen nur wenige Grundvoraussetzungen an das DBMS gestellt gestellt werden:

- Interaktionsmöglichkeiten mit dem Betriebssystem wie Programmaufrufe, Laden und Speichern von Daten des Dateisystems und/oder Aufrufe von Webservices
- Bereitstellung von Auslösern (Triggern)
- Eine gewisse Ausdrucksmächtigkeit der Stored Procedures / User Defined Functions des DBMS um Datentransformationen ausführen zu können

Wie leicht zu sehen ist, können Monitor- und Analyseprogramme direkt via SQL mit diesem System kommunizieren, weitere Vorteile die man durch DBMSe von Haus aus geliefert bekommt sind:

- Transaktionssicherheit
- Mehrbenutzerfähigkeit (parallele Ausführung von Workflow Instanzen)
- Kaum Limitierung in der Größe der zu verarbeitenden Daten

Weitere Vorteile solcher solcher Systeme sind:

- Sie sind mit geringem Implementierungsaufwand zu erstellen, da alle notwendigen Komponenten vom DBS zur Verfügung gestellt werden müssen. Insbesondere müssen keine Vorkehrungen für einen Mehrbenutzerbetrieb getroffen werden
- Beinhalten Optimierungsmöglichkeiten durch Einstellungen des DBMS
- Einheitliche Sprache (wie SQL) für Analyse und Monitor Anwendungen
- Informationen zum WfMS sowie Status zu Instanzen sind sofort verfügbar

Abschließend ist zu sagen, dass solche Systeme eine echte Alternative darstellen, die in [AS96] beschriebenen Probleme von WfMSen zu lösen. Jedoch existieren auch heute noch wenig DBSe, die über ausgeprägte Interaktionsmöglichkeiten wie Aufrufe von externen Programmen oder WSs anbieten. Vorallem lässt sich ein WF-Compiler, der einen WF in das entsprechende Datenmodell bzw. Schema transformiert mit einem DBS nur schlecht implementieren. Insbesondere bietet der Ansatz keine Möglichkeit schon existierende WfMSe zu verbessern.

3.2.4. Zusammenfassung und Abgrenzung zu dieser Arbeit

Die vorgestellten Ansätze zur stärkeren Integration von DBSen in WfMSe beziehen sich entweder auf externe Datenquellen und DBSe oder verschmelzen WfMSe und DBSe zu einem einzigen System. In dieser Arbeit untersuchen wir jedoch die Nutzungsmöglichkeiten einer integrierter DB innerhalb von gängigen WfMSen. Hierbei sollen lokale Optimierungen zur Laufzeit eines Workflows zum Einsatz kommen, die sich auf Basisaktivitäten der WF-Sprache beziehen. Dies ermöglicht eine transparente Nutzung des modifizierten WfMSs für bereits existierende WFs. Ausnutzen von BPEL/SQL Aktivitäten, die auf die integrierte DB zugreifen, können die Korrektheit des WfMSs verletzen. Da diese vom Anwender geschrieben werden müssen, ist es bei einer inkorrekten Nutzung möglich Variableninhalte einer veralteten Instanz, einer parallel laufenden Instanz oder sogar eines anderen Prozesses zu lesen und zu verändern. Dies beeinträchtigt die Nachvollziehbarkeit beendeter WF-Instanzen und kann zu nicht-deterministischem und semantisch inkorrektem Verhalten von laufenden Instanzen führen. Wir konnten nach bestem Wissen bis dato in der Literatur keine konkreten Ansätze für die stärkere Nutzung der integrierten WF-DB finden.

3.3. Workflowsysteme und Engines

Wir werden nun einige ausgewählte, gängige Workflowsysteme vorstellen und in wieweit diese ihre integrierte Datenbank für die Ausführung von Workflows verwenden. Die Reihenfolge der WfMSe impliziert keine Wertung, sie wurden alphabetisch sortiert. Wir stellen jeweils die WF-Sprache, das Lizenzmodell, die unterstützten DBMSe und deren Verwendung vor und beschreiben kurz das System.

3.3.1. Apache Orchestration Director Engine

Entwickler: Apache Software Foundation
Webseite: http://ode.apache.org/
Workflow-Sprache: WS-BPEL 2.0 (Kontroll-Fluss)
Lizenz: Apache License Version 2.0
Unterstützte DBMS: Apache Derby (embedded), IBM DB2, MySQL, PostgreSQL und viele Weitere (verwendet Hibernate oder openJPA Middelware) [Mül10]
Verwendung der DB: Persistenz

Apache ODE ist eine OpenSource WF-Engine für die Ausführung von WS-BPEL Workflows. Sie ist eine der wenigen, noch existierenden OpenSource WF-Engines für WS-BPEL und unterstützt den gesamten BPEL 2.0 Standard. Sie wird für die Forschungszwecke im Rahmen des Simulation Technology (SimTech) Projekts[1] der Universität Stuttgart als Prototyp für die speziellen Bedürfnisse von Simulationsworkflows angepasst und erweitert [GSK+11]. Da ihr Quellcode offen liegt, viele DBSe unterstützt und für SimTech erweitert wird, findet sie auch in dieser Arbeit Verwendung und wird in den Kapiteln 5 und 6 näher vorgestellt. Die integrierte DB wird bisher nicht weiter ausgenutzt, außer für die Persistenz von Prozess- und Instanzdaten.

[1] http://www.simtech.uni-stuttgart.de/

3.3.2. Taverna

Entwickler: Taverna / myGrid Team
Webseite: http://www.taverna.org.uk/
Workflow-Sprache: SCUFL (Daten-Fluss)
Lizenz: Lesser General Public License (LGPL) Version 2.1
Unterstützte DBMS: Apache Derby (embedded), MySQL
Verwendung der DB: Persistenz

Taverna ist ein komplettes WfMS mit GUI Editor für die Workflow Erstellung, der WF-Engine, die Workflows ausführt, und mit einem Monitor Tool, mit welchem man die Ausführung von Workflows überwachen sowie Zwischen- und Endergebnisse einsehen kann [OAF+04]. Ursprünglich wurde es für die speziellen Bedürfnisse von Bioinformatik-Workflows entwickelt, inzwischen findet es aber auch in allen anderen LifeSciences, wie Medizin- und Chemoinformatik, Verwendung. Es zeichnet sich durch einfach zu modellierende daten-Fluss orientierte Workflows und eine Vielzahl an Diensten aus, die als Aktivitäten in einen solchen Workflow einbindbar sind. Neben der Ausführung von Webservices stehen zahlreiche eingebettete Bioinformatik-Dienste zur Verfügung. Weiterhin ist es möglich Programme auf Betriebssystem-Ebene sowie lokale, vom Anwender in Java geschriebene, Aktivitäten aufzurufen. Eine Anfrage an die Mailing-Liste[2] ergab, dass die integrierte Datenbank außer zur Persistenz nicht weiter genutzt wird, um die Workflow-Ausführung zu beschleunigen oder anderweitig zu verbessern.

3.3.3. Trident Scientific Workflow Workbench

Entwickler: Microsoft Research
Webseite: http://tridentworkflow.codeplex.com/
Workflow-Sprache: Workflow Foundation - Extensible Object Markup Language (XOML) - von und nach WS-BPEL transformierbar
Lizenz: Apache License Version 2.0
Unterstützte DBMS: Microsoft SQL Server
Verwendung der DB: Persistenz

Microsoft Trident [BJA+08] wurde als wissenschaftliche Workflow Workbench entwickelt. Die interne Repräsentation des Workflows erfolgt über die Microsoft Windows Workflow Foundation[3] und das XML Format XOML. XOML selbst ist kontroll-Fluss oriententiert, in

[2]http://www.mail-archive.com/taverna-hackers@lists.sourceforge.net/msg01341.html
[3]http://msdn.microsoft.com/en-us/netframework/aa663328

Trident besteht allerdings ebenfalls die Möglichkeit daten-Fluss orientierte WFs zu konzipieren, ähnlich zu Taverna (siehe Kapitel 3.3.2). Dies geschieht über sog. *Container*. So existieren auch in Trident eine GUI zur Erstellung der Workflows und ein Monitortool zur Überwachung der Workflow-Ausführung. Neben Kontroll-Fluss-Strukturen wie *if-then*, Auswertungen von Bedingungen etc. verfügt Trident über eine Vielzahl von Zugriffsmöglichkeiten auf verschiedene Datenquellen wie das Dateisystem (File Input, File Writer), Datenbanken (SQL Connection, Stored Procedure Executor) und für Anwendungsdomänen spezifische Datenquellen. Ebenfalls kann die Amazon S3 Storage Cloud angebunden werden. Weitere Aktivitäten ermöglichen es Diagramme zu erzeugen oder XPath-Ausdrücke auf XML-Dokumente zu evaluieren. Ebenfalls ist es möglich Webservices einzubinden.

Der Quellcode ist frei verfügbar, jedoch wird für die Ausführung von Trident eine Microsoft SQL Server Version benötigt. Microsoft rät davon ab, die kostenlose Express Version zu verwenden. Eine Anfrage[4] an das Trident Forum ergab, dass auch hier die Datenbank nur für die Persistenz verwendet wird. Eine nennenswerte Architekturentscheidung ist die ausschließliche Verwendung von UDFs und Stored Procedures zur Abstraktion der SQL Anfragen an das letztendlich verwendete interne Datenmodell innerhalb der MS SQL Datenbank.

3.3.4. WebSphere Process Server

Entwickler: IBM
Webseite: http://www.ibm.com/software/integration/wps/
Workflow-Sprache: Generalized Flow (BPEL Derivat)
Lizenz: Proprietär
Unterstützte DBMS: Apache Derby, IBM DB2, IBM Informix, Microsoft SQL Server, Oracle 10/11g
Verwendung der DB: Unbekannt, mit Sicherheit jedoch Persistenz

Der WebSphere Process Server (WPS) ist die WF-Engine von IBMs WebSphere[5] und führt Workflows aus, welche in dem erweitertem WS-BPEL Dialekt *Generalized Flow* geschrieben sind. Dieser Dialekt ist um Aktivitäten erweitert, die z.B. einen Benutzer Entscheidungen oder Tätigkeiten ausführen lassen. Aufgrund der zur Verfügung stehenden Informationen konnte nicht ermittelt werden, ob der WPS sein integriertes DBS für andere Aufgaben außer zur persistenten Speicherung verwendet. Jedoch ist das zugehörige Tabellenschema auf mehrere voneinander unabhängige Datenbanken aufgeteilt. Somit ist es möglich diese auf mehrere Datenbankserver zu verteilen, um ein skalierbareres System zu erhalten.

[4] http://tridentworkflow.codeplex.com/Thread/View.aspx?ThreadId=229518
[5] http://www.ibm.com/software/websphere/

4. Nutzung von Funktionen einer integrierten Workflowdatenbank

Nachdem wir in Kapitel 2 die nötigen Grundlagen und in Kapitel 3 den aktuellen Stand der Forschung zur Integration von WF- und DB-Systemen sowie verschiedene WfMSe vorgestellt haben, widmen wir uns nun dem Forschungsziel dieser Arbeit und der erarbeiteten konzeptionellen Ergebnisse. Diese wurden in einem Prototypen umgesetzt (siehe Kapitel 5 und 6), der anschließend auf eine verbesserte Workflow-Ausführung hin evaluiert wurde (siehe Kapitel 7).

4.1. Grundlegendes Konzept

Ziel dieser Arbeit ist es eine WF-Engine so zu verändern, dass sie Funktionalitäten, die bei der Ausführung eines WF in der WF-Runtime stattfinden, an das DBMS der integrierten DB abgibt. Die Grundidee ist hierbei, dass so Datentransfer zwischen dem DBS und der WF-Engine vermieden oder verringert werden kann, was sich potentiell in einer schnelleren Workflow-Ausführung und/oder in einem verringertem Ressourcenverbrauch des WfMSs äußern sollte.

Um das Konzept zu verdeutlichen, betrachten wir Abb. 4.1. Es ist aus Software-Engineering Gesichtspunkten üblich zwischen der eigentlichen WF-Runtime und dem DBS eine DAO-Schicht (Data Access Object) zu legen, die dann letztendlich mit dem DBS kommuniziert. So ist es ohne Modifikationen der WF-Runtime möglich andere DBMS zu unterstützen. Alle WfMSe, die diesem Konzept folgen, halten ihre Daten in der DAO-Schicht, solange diese in der Runtime nicht benötigt werden. Die DAO-Schicht kümmert sich (automatisch) um die Persistenz dieser Daten, also dass diese in der integrierten DB gespeichert werden. Dies dient der Nachvollziehbarkeit und der Überwachung der Ausführung eines WFs mit Hilfe von Monitor Anwendungen und der Möglichkeit einen WF zu pausieren sowie insbesondere nach einem Systemausfall die Ausführung des Workflows fortzusetzen (Recovery). Außerdem ist es möglich Daten der DAO-Schicht aus dem Hauptspeicher zu entfernen (Dehydratation) und bei Bedarf aus der DB zu laden (Hydratation). Dehydratation erfolgt insbesondere bei langlaufenden Prozessen, um die Ressourcen zur Ausführung anderer WFs nicht unnötig zu belegen.

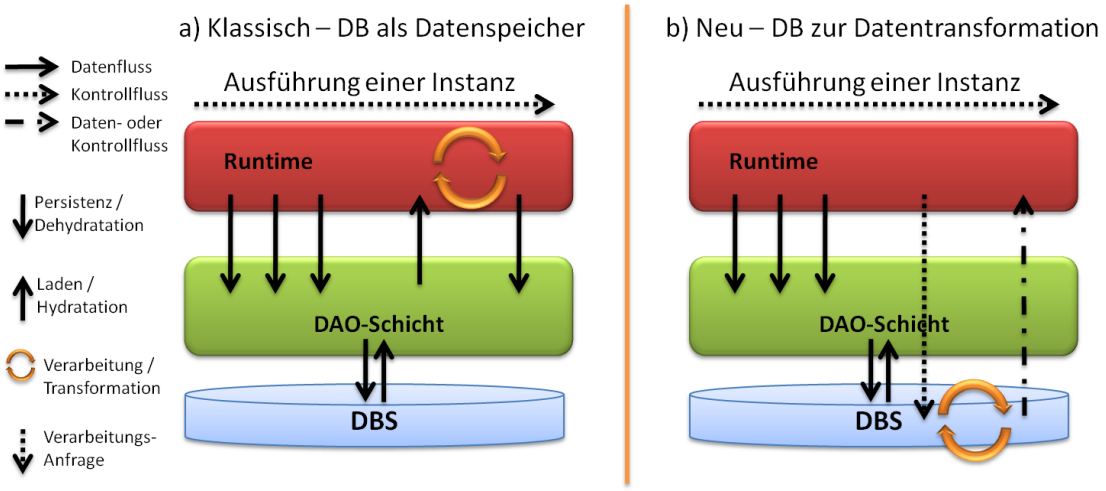


Abbildung 4.1.: Üblicherweise wird das DBS nur für die Persistenz und Speicherung von WF-Daten verwendet a). Das Konzept zur erweiterten Nutzung des DBS ist die Verarbeitung dieser Daten innerhalb der Runtime a) auf die Ebene des DBS zu übertragen b).

Betrachten wir die Zuweisung von Variablen innerhalb eines WFs als Beispiel. So wird ein Datum von der WF-Runtime aus der DAO-Schicht bzw. u.U. aus der Datenbank geladen, innerhalb der WF-Runtime zugewiesen und sofort wieder an die DAO-Schicht bzw. die DB übergeben. Die integrierte DB des WfMSs wird also nur als Datenspeicher verwendet (siehe Abb. 4.1a). Befinden sich zum Zeitpunkt der Zuweisung (Abb. 4.1b) jedoch alle Daten bereits in der DB, was z.B. durch eine garantierte Persistenz gesichert ist, kann diese Zuweisung direkt innerhalb der DB stattfinden. Mit dieser Methode können prinzipiell auch weitere Verarbeitungs- und Datentransformationsschritte (z.B. Auswertung von Bedingungen) innerhalb des DBS ausgeführt werden und falls vorhanden das Resultat oder eine Bestätigung der Ausführung an die WF-Runtime zurückgeliefert werden. Somit kann der Datentransfer zwischen den Softwareschichten verringert bzw. ganz vermieden werden.

Um diese Verlagerung der Aufgaben transparent zu halten und auch für bestehende WFs verwenden zu können, darf dies keine Veränderung im Quellcode der WFs nach sich ziehen. Somit ist die Verwendung von *Inline SQL* Aktivitäten (z.B. BPEL/SQL siehe Kapitel 3.2.1, Seite 45), die grundsätzlich auch auf der integrierten DB des WfMSs operieren könnten, nicht sinnvoll. Hieraus folgt zugleich, dass mit dem vorgestellten Konzept keine globalen Optimierungen möglich sind, diese globalen Optimierungen müssen durch entsprechende Compiler oder vorangestellte Optimierer (siehe Kapitel 3.2.2, Seite 47) erfolgen. Während der Ausführung von WFs in einer WF-Runtime sind somit nur lokale Optimierungen einzelner Aktivitäten möglich. Der durchaus plausible Ansatz die Funktionalitäten einer WF-Engine in ein DBMS vollständig zu integrieren (siehe Kapitel 3.2.3, Seite 51) und so die Runtime und DAO-Schicht aus Abb. 4.1 zu entfernen, würden den Rahmen dieser Arbeit sprengen,

die Realisierungsmöglichkeit eines entsprechenden im DBS geschriebenen WF-Compilers ist fraglich und der Ansatz liefert keinerlei Möglichkeiten bestehende WfMSe zu optimieren.

Zum aktuellen Zeitpunkt und nach bestem Wissen existiert keine Literatur zur Nutzung integrierter WF-DBen nach dem Konzept aus Abb. 4.1b. Die betrachteten WfMSe aus Kapitel 3.3 arbeiten alle nach dem klassischen Konzept aus Abb. 4.1a. Falls nicht, sind die entsprechenden Information nicht öffentlich zugänglich. Im folgenden Teilkapitel stellen wir konkrete Ansätze und Techniken vor, die unser grundlegendes Konzept aus Abb. 4.1b aufgreifen.

4.2. Pushdown Konzepte

Da die grundlegende Idee für eine stärkere Integration der integrierten DB das Herunterdrücken von Funktionalität darstellt, nehmen wir Anleihen an den Pushdown-Regeln aus [VSS+07], die das gleiche Ziel für externe DBen anstreben, und erweitern diese unter Betrachtung der Funktionalitäten einer kontroll-Fluss orientierten WF-Sprache wie WS-BPEL. Wir stellen zuerst die verschiedenen Pushdown-Konzepte bzw. Techniken vor und geben anschließend eine Hierarchie und eine entsprechende Softwarearchitektur an.

4.2.1. WebService-Pushdown

Eine Möglichkeit eine klassische Aufgabe einer WF-Engine an das DBS zu übergeben, stellt der *WebService-Pushdown* dar. Dieser ist eine Regel des PGM Optimierers (siehe Kapitel 3.2.2, Seite 47), die auch auf unseren Ansatz übertragbar ist. Hierbei wird der aufzurufende WS direkt innerhalb des DBSs aufgerufen. Dieser Aufruf kann von der WF-Engine transparent an das DBS, das die entsprechende Funktionalität unterstützt, übergeben werden. Hierbei kann ein verringerter Datentransfer zwischen WF-Runtime und DBS erzielt werden, da die Ergebnisdaten des WS direkt in die DB ohne Umweg über die WF-Runtime gespeichert werden (siehe Abb. 4.2). Allerdings sollten die Eingabedaten für den WS bereits in der DB abliegen, da sonst ein zusätzlicher Datentransfer zwischen WF-Runtime und DBS nötig wird. Der WebService-Pushdown ist für alle WfMSe, die WSs aufrufen und ein WS-fähiges DBS unterstützen, geeignet.

4.2.2. Assignment-Pushdown

Der Assignment-Pushdown ist sehr ähnlich zur Assign-Pushdown Regel des PGM Optimierers (siehe Kapitel 3.2.2, Seite 47). Während der Assign-Pushdown jedoch Referenzen auf WF-Variablen innerhalb von SQL-Anweisungen durch ihre Definition ersetzt, führt der Assignment-Pushdown ausschließlich Zuweisungen an und von WF-Variablen durch. Der

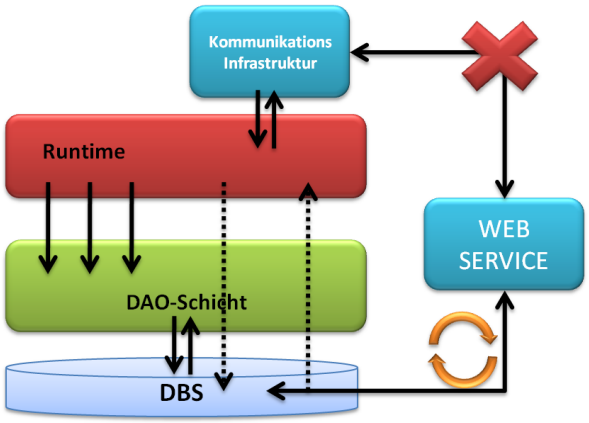


Abbildung 4.2.: Veranschaulichung des WebService-Pushdowns. Anstatt den Aufruf über die Kommunikationsinfrastruktur der WF-Runtime durchzuführen, wird der Aufruf vom DBS selbst vorgenommen. Somit wird der Datentransfer zwischen Runtime, DBS und ggf. Kommunikationsinfrastruktur vermieden.

entsprechende Verarbeitungsschritt aus Abb. 4.1b ist die Zuweisung der WF-Sprache und liefert nur eine Bestätigung an die WF-Runtime zurück ohne Variableninhalte zu übertragen. Ebenfalls müssen als Vorbedingung, wie beim WS-Pushdown, alle Variableninhalte in der integrierten DB abliegen. Eine konkrete Realisierung stellen wir in Kapitel 4.3 vor.

4.2.3. ExpressionEvaluation-Pushdown

Der ExpressionEvaluation-Pushdown wertet allgemein Ausdrücke (z.B. Mathematische) innerhalb des DBSs aus und liefert nur das Ergebnis an die WF-Runtime zurück. Dies können im Prinzip alle mögliche Typen sein, die ein entsprechender Ausdruck zurückgeben kann (z.B. Integer, Boolean uvm.). Dadurch kann das Datenvolumen, welches zwischen Runtime, DAO-Schicht und DBS ausgetauscht wird verringert werden. Der Verarbeitungsschritt in Abb. 4.1b ist somit die Auswertung dieser Ausdrücke. Im Gegensatz zum Assignment-Pushdown werden beim ExpressionEvaluation-Pushdown Daten an die WF-Runtime zurückgeliefert. Wie für WS- und Assignment-Pushdown sollten als Vorbedingung ebenfalls alle Variableninhalte schon in der integrierten DB abliegen. Eine konkrete Realisierung stellen wir ebenfalls in Kapitel 4.3 vor.

4.2.3.1. Condition-Pushdown

Der Condition-Pushdown soll Bedingungen, welche auf WF-Variablen referenzieren, innerhalb des DBSs auswerten und gibt nur noch das Resultat also *wahr* oder *falsch* an die

WF-Engine zurück, damit diese den weiteren Kontrollfluss steuern kann (z.B. IF und TransitionConditions in WS-BPEL). Somit stellt er eine konkrete Variante des ExpressionEvaluation-Pushdown dar, der boolsche Ausdrücke innerhalb des DBSs auswerten lässt. Weitere Varianten des ExpressionEvaluation-Pushdown könnten Schleifenzähler oder Zeitspannen berechnen.

4.3. Query-Pushdown

Wir möchten nun den *Query-Pushdown* einführen. Dies ist eine Technik, mit der sich, zumindest für WS-BPEL, die Konzepte Assignment- sowie ExpressionEvaluation-Pushdown gleichzeitig realisieren lassen. Hierbei werden Ausdrücke und Anfragen einer *Query*-Sprache, die nativ innerhalb eines Workflows formuliert werden können, an das DBS weitergeleitet, um dort ausgewertet zu werden.

Art der Datenstruktur	Query-Sprache	Anforderung DBS	WfMS
XML	XPath (XQuery)	XML-Enabled oder Native XML	alle WS-BPEL, Taverna, Trident uvm.
Tabellen	SQL	relationales DBS	-
Text	reguläre Ausdrücke, Information Retrieval (IR) Query-Sprache	CLOB Datenfeld, IR Query Implementierung	-
Zukünftiges Datenstruktur	zukünftige Query-Sprache	Implementierung Datenfeld und Query-Sprache	-

Tabelle 4.1.: Mögliche Ausprägungen des Query-Pushdowns.

Die Grundvoraussetzung hierfür ist, dass das DBS diese Query-Sprache implementiert und evaluieren kann. Implizit kann dies voraussetzen, dass die entsprechenden Datenstrukturen, in denen die Daten in den WFs dargestellt werden, als Datentyp im DBS abbildbar sein müssen. Die Art des Flusses der WF-Sprache spielt hierbei eine untergeordnete Rolle. Während bei kontroll-Fluss orientierten Sprachen innerhalb der Queries auf Variablen referenziert wird, kann bei einer daten-Fluss orientierten Sprache auf die eingehenden Kanten, welche mit einem Datum behaftet sind, referenziert werden. Um die möglichen Ausprägungen dieses Konzepts besser zu verstehen, sind in Tabelle 4.1 einige mögliche Ausprägungen dargestellt. Die Tabelle zeigt für verschiedene Datenstrukturen eine mögliche oder existierende Query-Sprache zur Verarbeitung dieser Struktur und die Anforderungen an das DBS, um den entsprechenden Query-Pushdown für ein WfMS zu realisieren. Wir

bezeichnen die jeweilige Ausprägung über den Namen der zugehörigen Query-Sprache (z.B. XPath-Pushdown, SQL-Pushdown etc.).

Wir können den Query-Pushdown in zwei Modi ausführen (siehe Abb. 4.3). Beim *synchronen* Modus wird der Ausdruck innerhalb des DBS ausgewertet und das Datum anschließend an die WF-Engine zurückgegeben und dort weiterverarbeitet. Dieser synchrone Modus realisiert den ExpressionEvaluation-Pushdown. Im *asynchronem* Modus wird der Ausdruck innerhalb des DBSs ausgewertet und direkt in der DB einem Datenfeld (z.B. einer WF-Variablen) zugewiesen und realisiert somit den Assignment-Pushdown. Wird das Datum des Feldes bzw. der WF-Variablen später in der Runtime benötigt, muss es entsprechen nachgeladen werden.

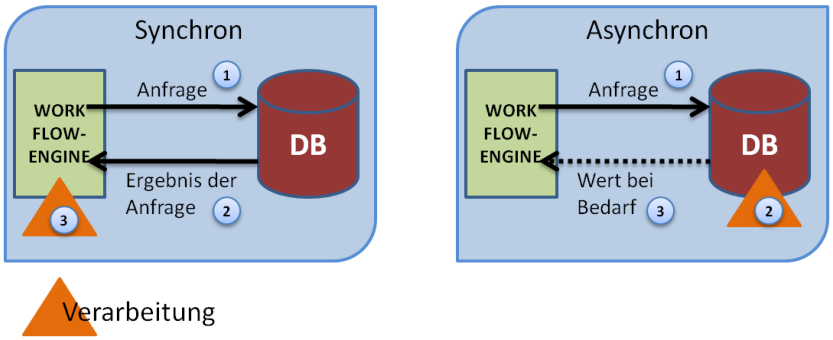


Abbildung 4.3.: Der Query-Pushdown in asynchronem oder synchronem Modus.

Für diese Arbeit ist insbesondere der *XPath-Pushdown* von Interesse, da er für alle WS-BPEL Implementierungen anwendbar ist, wir stellen ihn im Folgenden durch ein kleines Beispiel vor. Auf die anderen in Tabelle 4.1 beschriebenen Ausprägungen gehen wir nicht weiter ein, da wir Sie innerhalb dieser Arbeit nicht umsetzen werden.

4.3.1. XPath-Pushdown

Der XPath-Pushdown drückt XPath-Ausdrücke (siehe Kapitel 2.1.2, Seite 21) von der WF-Runtime Ebene auf die Datenbankebene. Betrachten wir unser XML-Beispieldokument (Listing 2.1, Seite 19) und folgende WS-BPEL *ASSIGN*-Aktivität:

```
<assign>
   <copy>
      <from>$y/document/title[@lang="de"]/text()</from>
      <to variable="x"/>
   </copy>
</assign>
```

Des Weiteren nehmen wir an, dass die Variable *y* das XML-Beispieldokument enthält und die Variable *x* vom Typ *xsd:string* ist. Somit wird also der deutsche Titel aus dem XML-Dokument *„Nutzung einer integrierten Datenbank zur effizienten Ausführung von Workflows“* an die Variable *x* zugewiesen. Man beachte, dass das Wurzel Element *thesis* im Ausdruck nicht angegeben wurde. Dies ist eine Besonderheit von *XPath-in-BPEL*-Ausdrücken, da das Element implizit durch die Typisierung bekannt ist. Wird der XPath-Pushdown synchron durchgeführt, wird der Titel an die WF-Runtime übergeben und dort der Variablen *x* zugewiesen und anschließend an die DAO-Schicht und von dort ggf. zur Persistenz an das DBS übergeben. Bei einem asynchronem XPath-Pushdown, wird der Titel innerhalb des DBSs ausgewertet und dem Datenfeld der Variable *x* innerhalb der Datenbank zugewiesen. Wird das Datum von *x* später in der WF-Runtime benötigt, muss es über die DAO-Schicht nachgeladen werden.

4.3.2. Pushdown-Hierarchie und Architekturmodell

Es ist möglich, die vorgestellten Pushdown-Konzepte einer Hierarchie zuzuordnen (siehe Abb. 4.4). Der Webservie-Pushdown steht weitgehend für sich alleine. Der Query-Pushdown kann in die asynchrone und synchrone Variante aufgeschlüsselt werden, wobei die asynchrone Variante den Assignment-Pushdown und die synchrone den ExpressionEvaluation-Pushdown realisieren kann. Der Condition-Pushdown ist als Spezialfall des ExpressionEvaluation-Pushdowns unter diesem anzuordnen. Gegebenenfalls lässt sich diese Hierarchie um weitere Konzepte erweitern. Wir werden in dieser Arbeit jedoch nur die in der Hierarchie vorgestellten Konzepte in den Prototypen umsetzen.

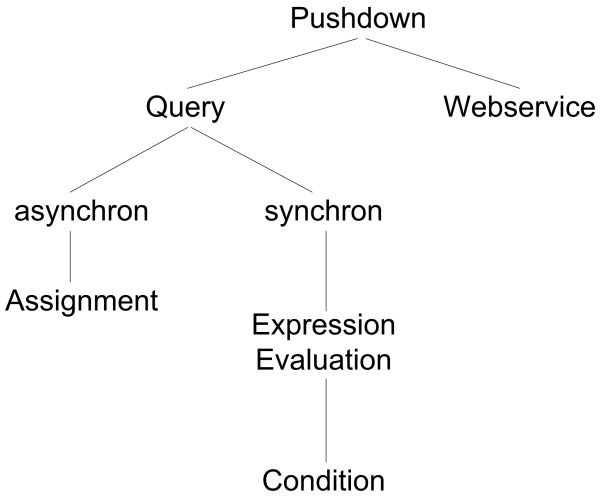


Abbildung 4.4.: Die Hierarchie der Pushdown-Konzepte. Für die Umsetzung des Prototyps auf Basis von WS-BPEL ist insbesondere die Realisierung des Query-Pushdowns und des Webservice-Pushdowns von Interesse.

Eine mögliche Softwarearchitektur, die es nicht erfordert zu große Änderungen in der WF-Engine vorzunehmen, führt neben der traditionellen DAO-Schicht eine Pushdown-Schicht ein (Abb. 4.5). In dieser Pushdown-Schicht werden, für den Query-Pushdown, die Ausdrücke des aktuell ausgeführten WFs in eine für das DBS verständliche Form gebracht und dann als Anfrage an das DBS gesendet. Für den WS-Pushdown wird die entsprechende DB-Funktion mit allen nötigen Parametern aufgerufen. Die Pushdown-Schicht kann direkt von der Runtime oder von der DAO-Schicht angesprochen und verwendet werden. Gegebenenfalls lässt sie sich auch in die DAO-Schicht integrieren.

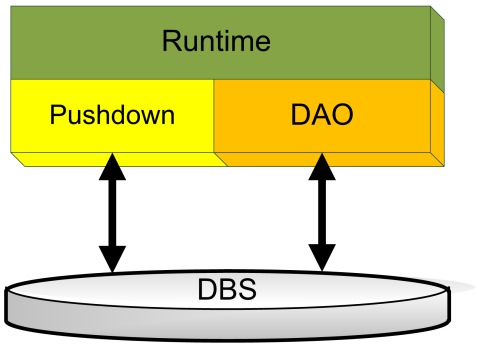


Abbildung 4.5.: Mögliche Softwarearchitektur zur Realisierung der Pushdown-Konzepte. Die Pushdown-Schicht kann von der Runtime sowie der DAO-Schicht verwendet werden und lässt sich ggf. in Letztere integrieren.

5. Apache ODE Architektur im Detail

In diesem Kapitel stellen wir die Software-Architektur von Apache ODE vor. Als OpenSource BPEL Engine und im Rahmen des SimTech Projekts untersucht [GSK+11], bietet sie sich ideal an um die Konzepte aus Kapitel 4 prototypisch zu implementieren, insbesondere da sie die dort vorgestellte Architektur Runtime-DAO-DBS besitzt. Zuerst stellen wir die allgemeine Architektur und anschließend die detailliertere Archtitektur der DAO-Schicht und die für den Prototyp wichtigen Runtime Module vor. Die Änderungen und Eingriffe, die für den Prototyp notwendig sind, werden in Kapitel 6 vorgestellt.

5.1. Gesamtarchitektur

Apache ODE ist eine reine WF-Engine, sie besitzt jedoch eingeschränkte Möglichkeiten laufende Instanzen zu überwachen, anzuhalten und fortzusetzen. Diese Funktionen sind entweder über ein Application Programming Interface (API) oder eine Webseite aufrufbar. Die in Apache ODE bekannt gemachten BPEL-Prozesse werden über WS-Aufrufe instanziiert. Aus diesem Grund muss Apache ODE in eine Kommunikationsinfrastruktur für Webservices eingebettet werden (ODE Integrationsschicht in Abb. 5.1). Typischerweise wird dazu ein Apache Tomcat Server[1] mit Axis2 verwendet, ODE kann allerdings auch in den Apache ServiceMix[2] eingebettet werden.

Apache ODE ist in Java implementiert. Die Gesamtarchitektur wird in Abb. 5.1 veranschaulicht. Betrachten wir das Schaubild zuerst von oben nach unten: Die in WS-BPEL definierten Prozesse werden durch den *ODE BPEL Compiler* zuerst in ein Java Objektschema übersetzt und anschließend serialisiert als Datei abgespeichert. Für das Instanziieren des Prozesses und seiner WS-Aufrufe müssen die entsprechenden WSDL Dateien übergeben werden. Um Initialwerte von Variablen zu generieren bzw. Zuweisungen von Literalen (XML Dokumente, die innerhalb des BPEL Prozesses definiert sind) zu validieren, werden ebenfalls die entsprechenden XML Schemata vom Compiler benötigt. Gleichzeitig wird der Prozess bekannt gemacht, ab sofort kann er instanziiert werden. Betrachten wir nun die *ODE BPEL Runtime*, sie besteht aus einer Vielzahl an Modulen, die in einem vereinfachten Architekturbild nicht alle darstellbar sind. Die Wichtigsten sind in Abb. 5.1 veranschaulicht:

[1] http://tomcat.apache.org/
[2] http://servicemix.apache.org/

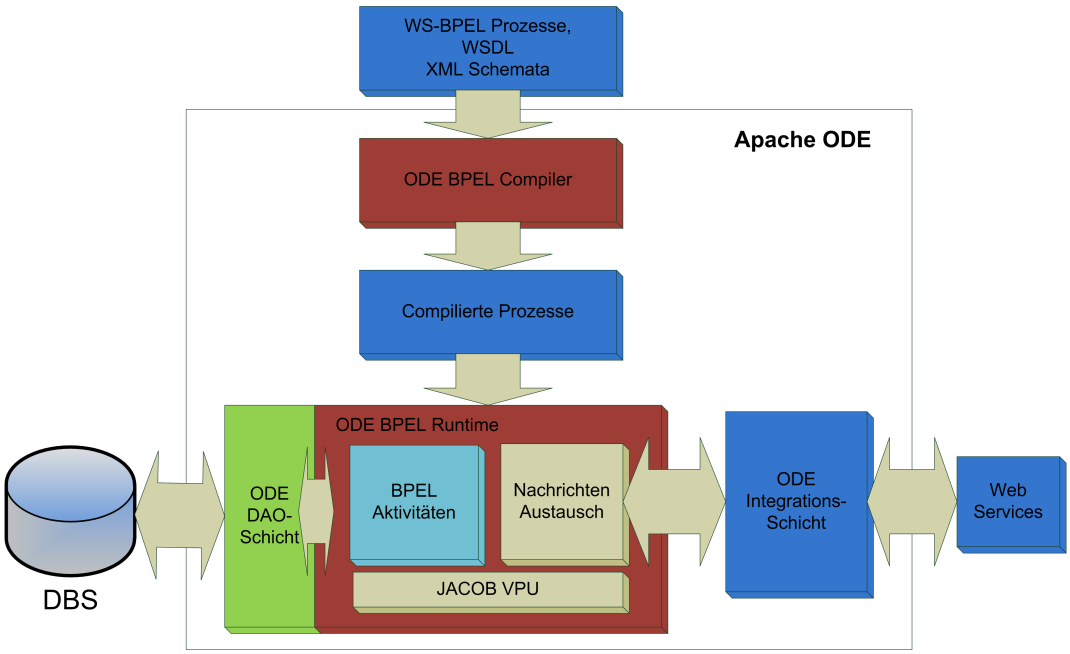


Abbildung 5.1.: Die Gesamtarchitektur von Apache ODE. WS-BPEL Prozesse werden zuerst in ein internes Schema compiliert. Die Runtime besteht aus mehreren Modulen, wobei das Fundament durch die *Jacob VPU* gebildet wird, um eine parallele Ausführung von Instanzen zu erlauben. In der Runtime werden ebenfalls die *BPEL Aktivitäten* implementiert, die potentiell auf Daten (*DAO-Schicht*) und auf WSs zugreifen (*Nachrichtenaustausch*) können. Vgl. [Apa]

JACOB VPU[3] ist eine für ODE entwickelte Virtual Processing Unit (VPU) und bildet das Fundament der Runtime Archtitektur. Sie übernimmt alle Aufgaben, um Instanzen parallel ausführen zu können (Kontextwechsel beim Warten auf Resultat eines WS, Verarbeiten von zeitgleichen Instanziierungen).

BPEL Aktivitäten werden in diesem Modul implementiert (*ASSIGN*, *FOREACH*, *IF* etc.). Dieses Modul interagiert mit den beiden Modulen *DAO-Schicht* und *Nachrichtenaustausch* um die Daten einer WF-Instanz zu verwalten und persistent zu halten sowie um mit WSs kommunizieren zu können.

ODE DAO-Schicht ist für die Speicherung und Persistenz der Prozess- und Instanzdaten verantwortlich und kommuniziert dazu mit dem DBS links in Abb. 5.1 (siehe auch Kapitel 4.1, Seite 57).

[3] http://ode.apache.org/jacob.html

Nachrichtenaustausch ist für das Senden und Empfangen von Nachrichten von und zu Webservices verantwortlich und dass eine eingehende Nachricht an die korrekte Prozess-Instanz geliefert wird. Sie interagiert mit der Integrationsschicht der Kommunikationsinfrastruktur (z.B. Axis2, ServiceMix), die ihrerseits die WS-Aufrufe verwaltet.

5.2. Detaillierte Architektur der Runtime und der Data Access Objects

Wir möchten nun detailliertere Zusammenhänge vorstellen. Zuerst werden wir die ODE Runtime genauer auflösen und anschließend die für diese Arbeit wichtigen Komponenten *OModel*, *Hibernate DAO* und *BPEL Aktivitäten* genauer betrachten.

5.2.1. ODE Runtime

Wir fächern die Runtime aus Abb. 5.1 in Abb. 5.2 noch etwas genauer auf. Auch hier wurde noch von der tatsächlichen Implementierung stark abstrahiert. Wir können die Komponenten in vier Bereiche einteilen, diese sind entsprechend farblich voneinander abgehoben. Die *DAO-Schicht* (gelb) verwaltet alle Daten zu Prozessen und Instanzen. Es kann aus drei verschiedenen Implementierungen gewählt werden:

- *Hibernate* [KBA+] ist ein DB Middleware System, welches darauf basiert, dass Datenfelder gekennzeichneter Java Objekte automatisch in einer DB persistent gemacht werden.
- *openJPA* [JPA] ist ebenfalls ein DB Middleware System und bietet im Prinzip die gleichen Funktionalitäten wie Hibernate. Die Systeme unterscheiden sich allenfalls leicht in den anbindbaren DBSen und kleineren Implementierungsdetails.
- *inMEM* implementiert die DAOs, ohne dass die entsprechenden Daten auf eine Datenbank abgebildet werden, so ist es möglich einen Prozess „inMemory“ auszuführen.

Die grünen Module kann man als *Prozessdaten und Ausführungslogik* Schicht bezeichnen. *Activity* implementiert die BPEL Konstrukte und deren Logik. Compilierte Prozesse werden im Objektmodell (*OModel*) dargestellt und repräsentieren die BPEL Aktivitäten und Konstrukte. Der *BPELRuntimeContext* hält Informationen und Zugriffsfunktionen auf die zu verwendende DAO-Schicht (Hibernate, openJPA) und die Laufzeitparameter von ODE (JDBC Einstellungen, Prozess-Dehydratation etc.). *BPELProcess* verwaltet die Informationen zu einem BPEL Prozess, wie aufzurufende Webservices und die im Prozess verwendeten Query-Sprachen (XPath, XQuery etc.). Dies hat zur Folge, dass über dieses Modul die Query Auswertung (blau) sowie die WS-Aufrufe (orange) erfolgen. Für die Evaluierung von XPath

Ausdrücken werden das *Jaxen* und *Javax* Framework verwendet, als Kommunikationsinfrastruktur kann entweder *Axis2* oder der *ServiceMix* verwendet werden (siehe Kapitel 5.1). In den folgenden Teilkapiteln stellen wir das Objektmodell, die Hibernate DAO-Schicht und die Runtime-Schicht vor.

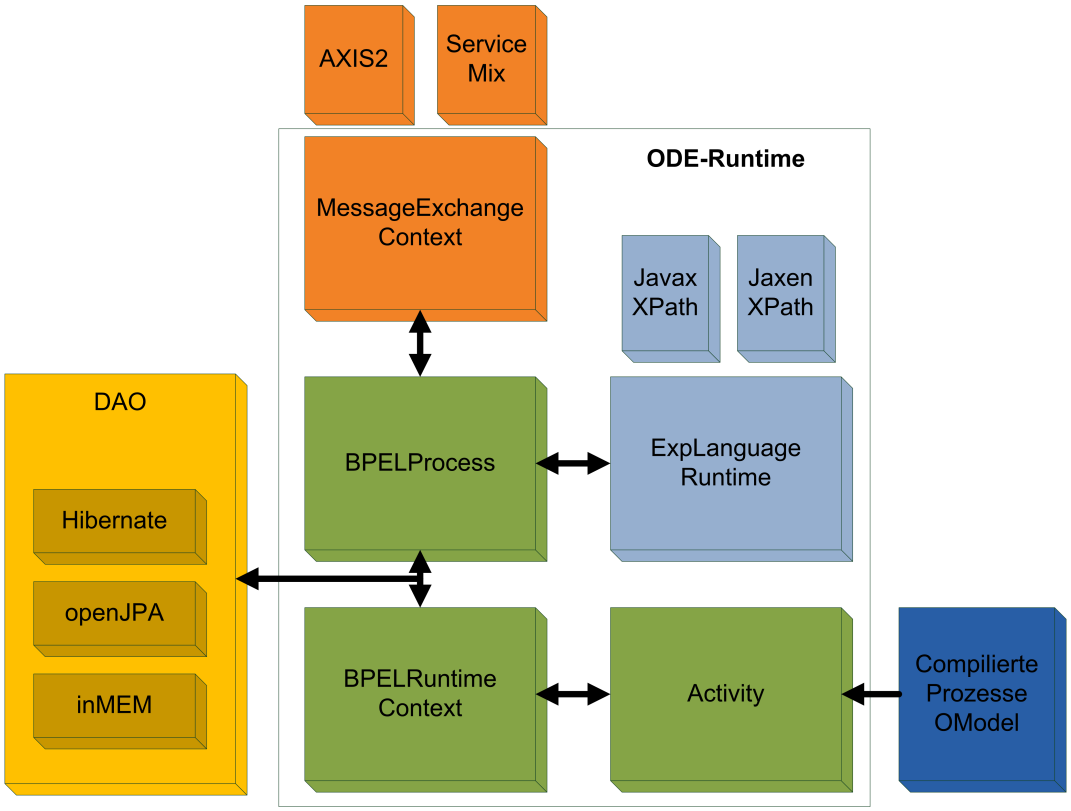


Abbildung 5.2.: Bestandteile der Apache ODE Runtime. Die BPEL *Aktivitäten* greifen auf ihre Instanzdaten über den *BPELRuntimeContext* zu. Query-Auswertungen und WS-Aufrufe erfolgen indirekt über das *BPELProcess* Modul, das die dafür nötigen Informationen trägt. Die DAO Schicht verwaltet die Prozess- und Instanzdaten.

5.2.2. OModel und BPEL Typsystem

Das *OModel* ist die Objektrepräsentation eines BPEL Prozesses, der mit Apache ODE compiliert wurde. Es gibt *OModel*-Objekte für alle BPEL Aktivitäten und weitere Konstrukte und Elemente wie *Scopes* und *Expressions*. Das *OModel* ist für das Verständnis, wie BPEL Prozesse auf Apache ODE abgebildet werden, essentiell und fördert die Lesbarkeit des Laufzeit Quellcodes der Aktivitäten und Konstrukte. Jedes BPEL-Konstrukt eines WFs wird durch den Apache ODE BPEL-Compiler in ein *OModel*-Objekt transformiert und trägt somit

die Informationen (Variablenname, WSDL-Operation, XPath-Ausdrücke etc.) aus diesem WF.

Abbildung 5.3 stellt den für diese Arbeit wesentlichen Teil des *OModel* dar:

OBase ist die Superklasse aller weitere OModel-Klassen. Die Methode *dehydrate()* erlaubt es die Informationen, die in einem OModel-Objekt zu einem konkreten BPEL-Prozess gespeichert sind, aus dem Hauptspeicher zu entfernen um Systemressourcen frei zu geben. Dies kann z.B. bei lang laufenden Prozessen mit hohen Wartezeiten sinnvoll sein.

OScope repräsentiert ein BPEL Scope, einen Sichtbarkeitsblock für Variablen, ähnlich zu Blöcken in Programmiersprachen mit statischer Namensbindung. Dieser trägt die Informationen zu allen Variablen, die in diesem Block definiert wurden.

OScope.Variable stellt eine Variablendeklaration dar. Diese beinhaltet den Namen der Variable und ihren Typ, in diesem Fall auch eine Rückreferenz auf den Block (OScope) in dem sie deklariert ist.

OVarType ist die Oberklasse der im *OModel* repräsentierten BPEL Typen, denen eine Variable angehören kann. Wir greifen das Typsystem später auf.

OActivity ist die Oberklasse für alle Aktivitäten.

OInvoke repräsentiert einen WS-Invoke. Die essentiellen Informationen sind, welche Variable die Ausgangsnachricht hält (*inputVar*), in welche Variable die Eingangsnachricht gespeichert wird (*outputVar*) und die aufzurufende WSDL Operation (*Operation*). Der Webservice selbst wird in seiner WSDL-Datei beschrieben und über BPEL partnerLinks (OPartnerLink) eingebunden.

OAssign repräsentiert die BPEL Zuweisung (ASSIGN), diese kann mehrere Copy Blöcke beinhalten (OAssign.Copy).

OAssign.Copy stellt einen Copy Block dar. Es existieren die linke Seite der Zuweisung (*to*) und die rechte Seite der Zuweisung (*from*). Die linke Seite muss auf eine Variable referenzieren, weshalb das entsprechende Interface *LValue* die *getVariable()* Methode implementieren muss. Die rechte Seite der Zuweisung (Interface *RValue*) kann eine Variable (*VariableRef*), ein Ausdruck (*Expression*) oder ein *Literal* sein. *Literale* sind Start- bzw. Initialwerte für BPEL-Variablen, können als Konstanten betrachtet werden und werden im Prozessmodell definiert, weshalb diese Werte im OModel gespeichert werden.

OExpression ist die Oberklasse für alle Query-Sprachen, die in dem System implementiert wurden. Uns reicht hier die Unterklasse *OXPath10Expression*, welche XPath1.0 Ausdrücke repräsentiert. Sie beinhaltet den XPath-Ausdruck sowie alle an dem Ausdruck beteiligten Variablen.

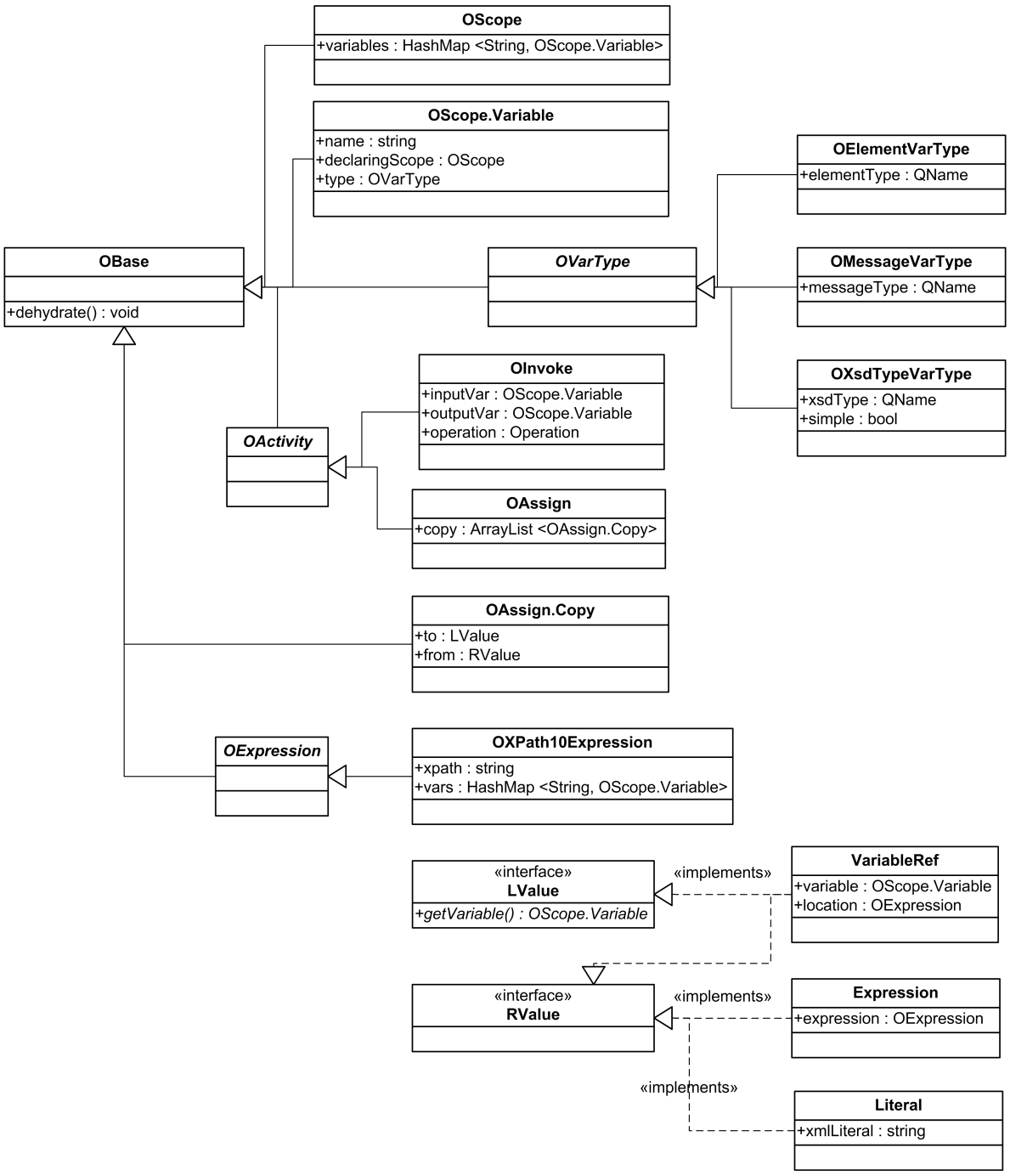


Abbildung 5.3.: Ausschnitt des *OModel* als UML Diagramm. Es zeigt die Repräsentationen der für diese Arbeit wichtigen BPEL Aktivitäten und Konstrukte wie Variablendeklaration, Variablentypen, Sichtbarkeitsbereiche, Zuweisungen, WS-Aufrufe und XPath-Ausdrücke.

In der BPEL Spezifikation [OAS07] werden folgende drei übergeordnete Variablen-Typen *WSDL Nachricht*, *XML Schema* und *XML Element* beschrieben. Diese werden entsprechend auf die *OModel*-Klassen *OMessageVarType*, *OXsdTypeVarType* und *OElementVarType* abgebildet, die alle Unterklassen von *OVarType* sind. Der jeweils zugehörige XML Schema Typ aus der Prozessdefinition wird als *Qualified Name* (QName) gespeichert. Bei Verwendung und Manipulation von XML Daten innerhalb von Apache ODE sind die intern verwendeten XML Wrapper Elemente zu den verschiedenen Typen von Interesse, diese können der Tabelle 5.1 entnommen werden. Diese Wrapper Elemente werden benötigt um entsprechende Manipulationen der XML Dokumente vorzunehmen (z.B Document-Object-Model Operationen) bzw. die Auswertungsmodule (Jaxen XPath-Evaluator) korrekt anzusteuern.

BPEL Typ	OVarType	Wrapper
WSDL Nachricht	OMessageVarType	<message/>
XML Element	OElementVarType	*Name des Elements*
XML Schema	OXsdTypeVarType - complex OXsdTypeVarType - simple	<xsd-complex-type-wrapper/> <temporary-simple-type-wrapper/>

Tabelle 5.1.: Die BPEL Variablen Typen, ihre *OModel* Repräsentation und die in der Laufzeit verwendeten Wrapper Elemente.

Es sei darauf hingewiesen, dass im *OModel* keinerlei Variableninhalte gespeichert werden. Lediglich die Werte zu Literalen innerhalb von ASSIGN-COPY Blöcken werden hier gespeichert. Die Speicherung von Variableninhalten erfolgt innerhalb der DAO-Schicht und wird im nächsten Abschnitt besprochen.

5.2.3. ODE Hibernate DAO und Tabellenschema

Nachdem wir das OModel, die darin enthaltene Deklaration und Typisierung von Variablen betrachtet haben, werden wir nun die Speicherung der Variableninhalte über die DAOs vorstellen. Wir stellen ebenfalls nur einen kleinen Ausschnitt der DAO-Schicht vor, und zwar die Schnittstellen zu Scopes und Variableninhalten. Als Implementierungsbeispiel stellen wir die Hibernate Variante vor. Einerseits, da sie im Prototyp verwendet wurde (siehe Kapitel 6) und da aus dem Hibernate Beispiel das zugehörige Tabellenschema direkt ableitbar ist.

Über die *ProcessInstanceDAO*-Schnittstelle erhält man Zugriff auf die *ScopeDAO*-Schnittstelle (siehe Abb. 5.4). Diese hält die Informationen zu den *XmlDataDAO*-Schnittstellen welche die Daten zu den Variablen beinhalten und diese über *Getter- und Settermethoden* verfügbar machen. Da die *ProcessInstanceDAO*-Schnittstelle die spätere Verbindung zum *BpelRuntimeContext* bildet, ist sie für das Gesamtbild wichtig, die konkrete Implementierung ist jedoch uninteressant.

Betrachten wir nun die Hibernate Implementierungen der Schnittstellen *ScopeDaoImpl* und *XmlDataDaoImpl*. Beides sind Unterklassen von *HibernateDao*, dort wird die aktuelle Hibernate DB-Sitzungen verwaltet und Hibernate-Methoden (*update()*) können über diese Klasse angesprochen werden.

XmlDataDaoImpl enthält ein Attribut *_node* vom Typ *W3C Node*, in dem das XML Dokument gehalten wird, sowie das Attribut *_data* vom Typ *HXmlData*. Das XML Dokument in *_node* wird, falls es größer als 256 Zeichen ist, in eine Byte-Repräsentation konvertiert und in *HXmlData _data* gespeichert. Andernfalls wird es in *HXmlData* als *_simpleValue* gespeichert. Dies wird aus Performanzgründen durchgeführt, um *String*- anstatt *BLOB*-Felder für kleine Inhalte innerhalb der DB zu verwenden. Entsprechend referenziert *ScopeDaoImpl* auf ein Objekt vom Typ *HScope* in dem z.B. der Name des Scopes abgelegt wird. Objekte von *HScope* und *HXmlData* stellen durch die Hibernate Middleware direkt Zeilen entsprechender Datenbanktabellen dar. Hibernate verwaltet die Synchronisierung, also das Speichern und Laden bzw. die Persistenz der Attribute, dieser Objekte über die *Getter-/Settermethoden* und durch Überwachung des Java Bytecodes selbstständig. Dazu müssen die Datenfelder solcher Objekte entsprechend annotiert werden (siehe Listing 5.1). Aus diesen Annotationen ergeben sich ebenfalls die Tabellenschemata für die Datenbank. Es kann auch mit Vererbung gearbeitet werden: da *HScope* sowie *HXmlData* von *HObject* abgeleitet sind, besitzen beide das *_id* Attribut. Wir erhalten aus Abb. 5.4 direkt folgendes Datenschema in Abb. 5.5 für die Datenbank.

```
1  /**
2   * @hibernate.class table="BPEL_XML_DATA"
3   */
4  public class HXmlData extends HObject {
5
6      private byte[] _data;
7      ...
8
9      /**
10      * @hibernate.property type="byte[]"
11      * @hibernate.column name="DATA" sql-type="BLOB"
12     */
13     public byte[] getData() {
14         return _data;
15     }
16
17     public void setData(byte[] data) {
18         _data = data;
19     }
20         ...
21 }
```

Listing 5.1: Beispiel für die Annotation einer Java Klasse, die von Hibernate synchronisiert werden soll.

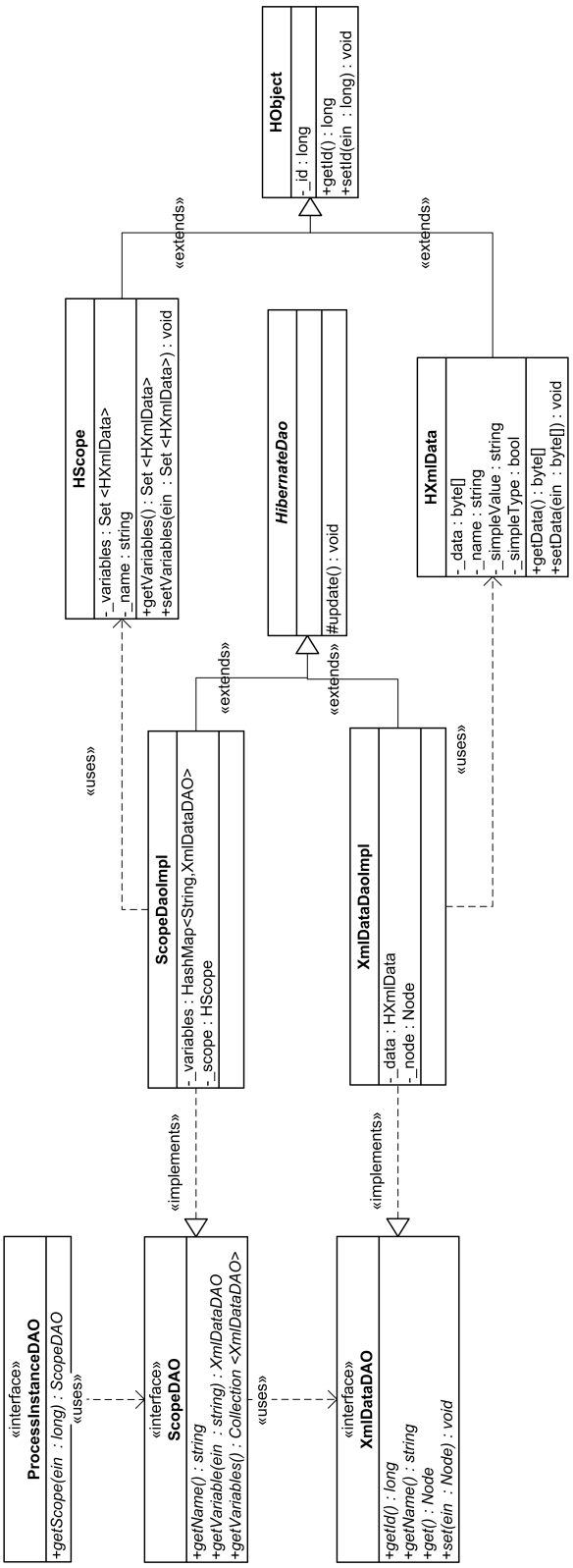


Abbildung 5.4: UML-Diagramm eines Ausschnitts der DAO-Schicht inklusive Hibernate Varianten der *ScopeDAO* und *XmlDataDAO* Schnittstellen. Dies sind alles Klassen von ODE, Hibernate selbst überwacht nur die Objekte *HScope* und *HXmlData* und synchronisiert deren Attribute mit Zeilen einer entsprechenden Datenbanktabelle.

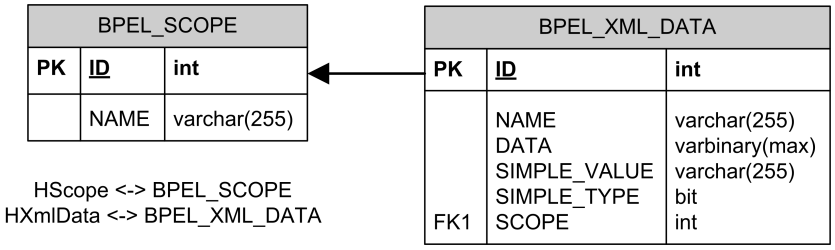


Abbildung 5.5.: Tabellenschema, welches sich durch die Hibernate Middleware direkt aus den annotierten Klassen *HScope* und *HXmlData* aus Abb. 5.4 ergibt. *HScope* wird auf die Tabelle *BPEL_SCOPE* und *HXmlData* auf die Tabelle *BPEL_XML_DATA* abgebildet.

5.2.4. BpelRuntimeContext und Aktivitäten

Um das Gesamtbild zu vervollständigen, stellen wir jetzt die Funktionsweise der Laufzeit-Aktivitäten (*ACTIVITY*) mit dem Laufzeit-Kontext (*BpelRuntimeContext*) und deren Anbindung an die DAO-Schicht und das *OModel* vor. Wir stellen die Komponenten aus Abb. 5.6 einzeln vor und beschreiben anschließend ihr Zusammenwirken. Ein konkretes Beispiel wird im Abschnitt *Ausführungsszenario* besprochen.

BpelRuntimeContext und die Implementierung *BpelRuntimeContextImpl* stellen Methoden zur Verfügung, mit denen Variableninhalte gelesen (*readVariable*) und geschrieben (*writeVariable*) werden können, diese greifen direkt auf die DAO-Schicht zu. Der *BpelRunteimContext* ist somit das Bindeglied zwischen Runtime und DAO-Schicht. Des Weiteren werden WS-Aufrufe an die Kommunikationsinfrastruktur weitergeleitet und die Auswertungsmodule für Query-Sprachen (wie XPath, XQuery etc.) den Aktivitäten zur Verfügung gestellt.

ScopeFrame implementiert die Funktionen der *BPEL-Scopes* (Blöcke). Eine Funktion ist das Auflösen einer Variable (*resolve*) entsprechend der Sichtbarkeit, die durch die im BPEL-Prozess definierten Scopes gegeben sind. Aus diesem Grund besitzt ein *ScopeFrame* Zugriff auf seinen Vater *ScopeFrame*. Darüber hinaus stellt *ScopeFrame* Methoden für das Lesen (*fetchVariableData*) und Schreiben (*writeVariable*, *commitChanges*) von Variableninhalten bereit. *ScopeFrame* ist direkt mit seiner OModel-Repräsentation verbunden (Attribut oscope).

VariableInstance ist eine Wrapperklasse für eine Variable aus dem OModel (OScope.Variable) und der ID des *Scope*, dem sie angehört.

ACTIVITY ist die Oberklasse aller implementierten BPEL-Aktivitäten. Sie beinhaltet den *ScopeFrame*, in dem sie eingebettet ist, sowie die *OModel* Repräsentation dieser Aktivität über ein Objekt der Klasse *ActivityInfo*. Ebenfalls stellt sie Methoden zum Lesen

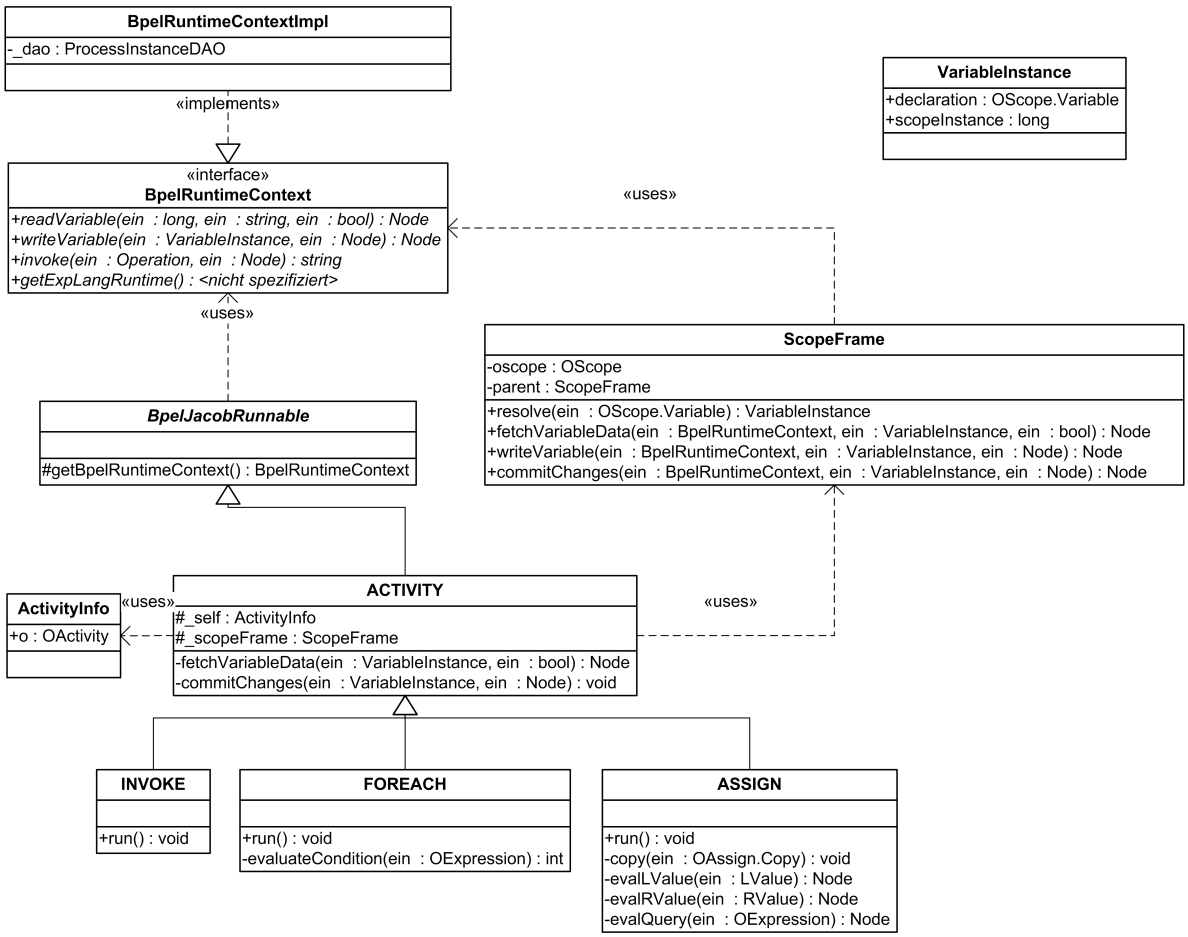


Abbildung 5.6.: Ausschnitt der Laufzeitkomponenten als UML-Diagramm. *ACTIVITY* ist an das *OModel* und die *BpelRuntimeContextImpl* an die DAO-Schicht angebunden. Zwischen diesen beiden Komponenten fungiert *ScopeFrame* als Vermittler für das Lesen und Schreiben von Variableninhalten.

(*fetchVariableData*) und Schreiben (*commitChanges*) von Variableninhalten bereit. Insbesondere verfügt sie über Zugriff auf das aktuelle *BpelRuntimeContext*-Objekt, welches für die laufende Instanz von Apache ODE gültig ist. Auf dieses kann über die Methode *getBpelRuntimeContext()*, welche von *BpelJacobRunnable* ererbt wurde, zugegriffen werden. Im Folgenden stellen wir nur die, für das Verständnis dieser Arbeit, wichtigen Aktivitäten vor. Alle abgeleiteten Aktivitäten müssen die Methode *run()* implementieren, diese wird durch die JacobVPU aufgerufen um die Aktivität zu starten.

INVOKE realisiert die Logik eines WS-Aufrufs. Zuerst wird die Variable mit der Ausgangsnachricht gelesen, diese an die *invoke*-Methode des *BpelRuntimeContext* übergeben und anschließend die Antwortnachricht des WS in die dafür vorgesehene Variable geschrieben.

FOREACH realisiert die Logik der *BPEL-Foreach* Schleife. Diese Schleife besitzt einen Start- und einen Endwert, über den ein Zähler läuft. Diese Werte werden über Query-Ausdrücke bestimmt (*evaluateCondition*).

ASSIGN realisiert die *BPEL-Assign* Logik. Hierbei werden sequentiell alle *Copy*-Blöcke durchlaufen und jeweils die Variable der linken Seite aufgelöst (*evalLValue*) sowie das Resultat des Ausdrucks oder der Inhalt der Variable der rechten Seite (*evalRValue*) und dieser Wert anschließend in die Variable der linken Seite gespeichert. Die Methode *evalQuery* wird verwendet um Query-Ausdrücke innerhalb von *evalRValue* auszuwerten.

Lesende und schreibende Zugriffe auf eine Variable innerhalb einer Aktivität finden grundsätzlich folgendermaßen statt:

1. Die Variable liegt als *OScope.Variable* vor und wird mit Hilfe von *ScopeFrame.resolve* aufgelöst und in ein Objekt von *VariableInstance* umgeschrieben.
2. Es wird auf die Schreib- und Lese-Methoden von *ACTIVITY* unter Verwendung von *VariableInstance* zugegriffen, diese geben den Aufruf an die Methoden von *ScopeFrame* weiter, die prinzipiell auch innerhalb der Aktivität direkt angesprochen werden können. Hierfür muss zusätzlich der *BpelRuntimeContext* übergeben werden.
3. *ScopeFrame* leitet die Anfrage an die Methoden zum Lesen und Schreiben von Variablen des *BpelRuntimeContext* weiter.
4. Der *BpelRuntimeContext* greift auf die konkreten Variableninhalte über die DAO-Schicht zu, überschreibt diese mit neuen Werten oder liefert den aktuellen Inhalt zurück.

Die Aktivitäten sind indirekt mit dem *OModel* über ActivityInfo und über den ScopeFrame verknüpft (siehe Kapitel 5.2.2). Die Anbindung an die DAO-Schicht erfolgt innerhalb der *BpelRuntimeContextImpl* über die *ProcessIntanceDAO* (siehe Kapitel 5.2.3).

5.2.5. Ausführungsszenario

Um die Interaktion der drei vorgestellten Schichten (*OModel*, *Hibernate-DAO* und *BpelRuntimeContext und Aktivitäten*) besser nachvollziehen zu können, werden wir hier auf das BPEL-Zuweisungsbeispiel aus Kapitel 4.3.1 (Seite 62) zurückgreifen.

```
<assign>
   <copy>
      <from>$y/document/title[@lang="de"]/text()</from>
      <to variable="x"/>
   </copy>
</assign>
```

Variable *y* enthält nach wie vor das XML-Beispieldokument aus Listing 2.1 (Seite 19). Variable *x* ist vom Typ *xsd:string*. Das entsprechende *OAssign* Objekt enthält ein *OAssign.Copy* Objekt, dessen Attribut *to* vom Typ *VariableRef* ist und den Namen der Variable *x* speichert. Das Attribut *from* ist ebenfalls vom Typ *VariableRef* und hält den Namen der Variable *y* und zusätzlich den *OXPath10Expression* Ausdruck *„/document/title[@lang="de"]/text()"*. Wir starten mit der Methode *ASSIGN.run()* und stellen die Ausführungsreihenfolge in Pseudoquellcode vor, wie sie bei Ausführung des Beispiels auftreten würden (Listing 5.2).

Man kann erkennen, dass es ein relativ langer Weg ist, bis alle Informationen für die tatsächliche Zuweisung vorliegen. Insbesondere wird der aktuelle Wert der Variablen *x* gelesen, bevor der neue Wert für *x* geschrieben wird. Dies hat mit der Möglichkeit zu tun, dass auf der linken Seite der Zuweisung ebenfalls auf einen bestimmten Pfad im XML Dokument verwiesen werden kann. Aus diesem Grund benötigt man den alten Variableninhalt zur Zuweisung innerhalb der Runtime. Beim anschließendem Speichern des neuen Werts in die DAO-Schicht, leuchtet es nicht ein, warum der neu geschriebene Wert gleichzeitig zurückgegeben wird. Insbesondere da *ASSIGN.commitChanges* diesen Wert einfach ignoriert. Wir möchten darauf hinweisen, dass der Autor dieser Arbeit nicht für den Zuweisungsmechanismus von Apache ODE, für das vorgestellte Beispiel (Listing 5.2), verantwortlich ist.

Man muss in jedem Fall darauf hinweisen, dass in Java lediglich die Referenzen der Objekte kopiert werden, es wird also tatsächlich auf dem Inhalt operiert, der sich in der DAO-Schicht befindet. Aus diesem Grund kann der vermeintliche Kommunikationsüberschuss möglicherweise sehr gering sein.

```
1  // Deklarationen
2  to : VariableRef; // Enthält Deklaration für BPEL-Variable x
3  from : VariableRef; // Enthält Deklaration für BPEL-Variable y
4  VI : VariableInstance; // Temporäre Variable
5  value, rvalue, lvalue : Node; // Temporäre Variablen für XML-Inhalte
6  // Start der Assign Aktivität
7  ASSIGN.run()
8    // Bearbeiten des COPY-Blocks, Zuweisung des from-Teils an den to-Teil
9    ASSIGN.copy(to, from)
10
11     // Bestimmen des Werts des from-Teils
12     rvalue := ASSIGN.evalRValue(from)
13       // ScopeFrame löst Variable y auf
14       VI := ScopeFrame.resolve(from)
15       // Lesen des Werts von Variable y
16       value := ASSIGN.fetchVariableData(VI)
17         // Durchreichen an BpelRuntimeContext
18         return ScopeFrame.fetchVariableData(VI)
19           return BpelRuntimeContext.readVariable(VI)
20             // Lesen der ScopeDAO des Scopes in den y eingebettet ist
21             ScopeDAO := ProcessInstanceDAO.getScope(VI.scopeInstance)
22             // Lesen der XmlDataDAO der zu y gehört
```

```
23              XmlDataDAO := ScopeDAO.getVariable(VI.declaration.name)
24              // Rückgabe des Werts von y
25              return XmlDataDAO.get
26       // Evaluieren des XPath-Ausdrucks auf den Wert von y
27       return ASSIGN.evalQuery(value, from.location)
28
29     // Bestimmen des Werts des to-Teils
30     lvalue := ASSIGN.evalLValue(to)
31       // ScopeFrame löst Variable x auf
32       VI := ScopeFrame.resolve(to)
33       // Lesen des Werts von Variable x
34       value := ASSIGN.fetchVariableData(VI)
35         // Durchreichen an BpelRuntimeContext
36         return ScopeFrame.fetchVariableData(VI)
37           return BpelRuntimeContext.readVariable(VI)
38              // Lesen der ScopeDAO des Scopes in den x eingebettet ist
39              ScopeDAO := ProcessInstanceDAO.getScope(VI.scopeInstance)
40              // Lesen der XmlDataDAO der zu x gehört
41              XmlDataDAO := ScopeDAO.getVariable(VI.declaration.name)
42              // Rückgabe des Werts von x
43              return XmlDataDAO.get
44
45     // Die eigentliche Zuweisung
46     lvalue := rvalue;
47
48     // Speichern des neuen Werts von x
49     // ScopeFrame löst Variable x auf
50     VI := ScopeFrame.resolve(to)
51     // Durchreichen an BpelRuntimeContext
52     ASSIGN.commitChanges(VI, lvalue)
53       return ScopeFrame.commitChanges(VI, lvalue)
54         return BpelRuntimeContext.writeVariable(VI, lvalue)
55           // Lesen der ScopeDAO des Scopes in den x eingebettet ist
56           ScopeDAO := ProcessInstanceDAO.getScope(VI.scopeInstance, lvalue)
57           // Lesen der XmlDataDAO der zu x gehört
58           XmlDataDAO := ScopeDAO.getVariable(VI.declaration.name)
59           // Setzen des neuen Werts von x
60           XmlDataDAO.set(lvalue)
61           // Rückgabe des neuen Werts von x
62           return XmlDataDAO.get
63
64 // Ende der Assign Aktivität
```

Listing 5.2: Pseudoquellcode der Ausführung des ASSIGN-Beispiels für Apache ODE.

5.3. Möglichkeiten für eine stärkere Nutzung der integrierten Datenbank

Nach Betrachtung der Architektur von Apache ODE und den einzelnen BPEL Aktivitäten ergeben sich nun Möglichkeiten die Pushdown-Konzepte aus Kapitel 4 anzuwenden. Für WS-Aufrufe sollte es möglich sein den WS-Pushdown einzuführen, indem die Methode *invoke* von *BpelRuntimeContextImpl* erweitert wird. Die Zuweisungslogik von *ASSIGN* sollte unter Beachtung des Typsystems (siehe Tabelle 5.1) in die *ScopeDAO*-Schicht verlagert werden können und als asynchroner Query-Pushdown (Assignment-Pushdown) direkt im DBS erfolgen. Die notwendigen Erweiterungen stellen wir in Kapitel 6 vor. Alternativ dazu kann die Zuweisungslogik in *ASSIGN* verbleiben und der synchrone Query-Pushdown (ExpressionEvaluation-Pushdown) verwendet werden, um die Berechnung des zuzuweisenden Werts in der DB durchzuführen und anschließend diesen Wert für die Zuweisung an die ODE-Runtime zu übertragen. Bedingungsevaluationen für Schleifen (*FOREACH*, *WHILE* etc.), für Kontrollstrukturen (*IF*, *SWITCH*) und Zeitberechnungen für die Verarbeitung von Ereignissen (*ONALARM*, *WAIT*) könnten über einen synchronen Query-Pushdown (ExpressionEvaluation/Condition-Pushdown) erfolgen. Ebenfalls könnte der synchrone Query-Pushdown (als Condition-Pushdown) für die Auswertung der *TransitionConditions* aller Aktivitäten verwendet werden. Eine Auflistung aller lokalen Optimierungsmöglichkeiten für WS-BPEL Aktivitäten ist der Tabelle 5.2 zu entnehmen.

WS-BPEL Aktivität/Konstrukt	ODE Klasse	Art des Pushdown	Modus des Pushdown
ASSIGN	ASSIGN	Query	asynchron & synchron
INVOKE	INVOKE	Webservice	-
FOREACH	FOREACH	Query	synchron
WHILE	WHILE	Query	synchron
REPEAT UNTIL	REPEATUNTIL	Query	synchron
IF (BPEL 2.0)	SWITCH	Query	synchron
SWITCH (BPEL 1.0)	SWITCH	Query	synchron
ONALARM	EH_ALARM	Query	synchron
WAIT	WAIT	Query	synchron
TransitionConditions	ACTIVITYGUARD	Query	synchron

Tabelle 5.2.: WS-BPEL Aktivitäten und ihre möglichen Optimierungen durch die Pushdown-Konzepte aus Kapitel 4.

6. Implementierung des Prototyps

Um die Realisierbarkeit der Konzepte aus Kapitel 4 nachzuweisen und sie auf eine Performanzsteigerung hin untersuchen zu können, wurden sie prototypisch innerhalb der WF-Engine Apache ODE umgesetzt. In diesem Kapitel stellen wir die dazu notwendigen Modifikationen und technische Details sowie typische Probleme, die bei der Umsetzung auftraten, vor.

6.1. Veränderungen an der Architektur von Apache ODE

Um den Prototyp umzusetzen, mussten zuerst mehrere Komponenten für Apache ODE festgelegt werden. Zum einen die Version von Apache ODE selbst, hier wurde entschieden die *Version 1.3.4* zu verwenden. Diese war zur Zeit der Implementierung die aktuellste 1.3 Version, die Version 2.0 wird nicht verwendet, da diese von Apache nicht fortgeführt wird. Die Einbettung von Apache ODE findet innerhalb *Apache Tomcat 6.0.29 mit Axis2* statt, da dies eine unkomplizierte, gut dokumentierte und sehr verbreitete Möglichkeit ist Apache ODE zur Verfügung zu stellen. Die Wahl für das DB Middleware System (Realisierung der DAOs) fiel auf *Hibernate 3.2.5*. Prinzipiell gibt es zwar kaum Unterschiede zwischen openJPA und Hibernate, allerdings ist es derzeit mit openJPA nicht möglich XML Daten größer als 1MB in die DB abzuspeichern. Hibernate hingegen besitzt hier keine Beschränkungen, außer die für *Binary Large OBbjects* (BLOB) üblichen, die auch für openJPA gelten. Als DBS wurde IBM DB2 UDB Version 9.7 verwendet, einerseits, da sie in [Mül10] als WF-DB empfohlen wird, und andererseits, da sie über ausreichend XML Funktionalitäten zum Speichern und Verarbeiten von XML Dokumenten verfügt. Für die Evaluierung wurde als Alternative zu DB2 noch PostgreSQL Version 8.4 mit einigen Einschränkungen hinzugezogen (siehe Kapitel 7). Die eben besprochenen Entscheidungen sind in Tabelle 6.1 zusammengefasst.

6.1.1. Änderungen am Datenmodell der integrierten Datenbank

Um die Implementierung des *XPath-Pushdowns* vorzubereiten, musste das Datenmodell bzw. das Datenbankschema aus Abb. 5.5 (Seite 74) geändert werden. Das *BLOB* Feld *DATA* der Tabelle *BPEL_XML_DATA* wurde in ein XML Feld umgewandelt, um die XML Verarbeitung innerhalb des DBSs zu ermöglichen. Dazu musste die Hibernate Annotierung für das Attribut *_data* der Klasse *HXmlData* aus Abb. 5.4 (Seite 73) von *BLOB* auf *XML* geändert werden.

Komponente	Software	Version	Begründung
WF-Engine	Apache ODE	1.3.4	aktuelle ODE Version, 2.0 wird nicht fortgeführt
Einbettung	Apache Tomcat	6.0.29	einfach durchzuführen, weit verbreitet
DB Middleware	Hibernate	3.2.5	XML Daten größer 1MB möglich
DBS	IBM DB2 UDB	9.7.0.441	empfohlen und sehr gute XML Verarbeitungsmöglichkeiten

Tabelle 6.1.: Auswahl und Begründung der verwendeten Komponenten für den Prototyp.

Tatsächlich wird in der Originalversion von Apache ODE hier ein benutzerdefinierter Typ verwendet, der die Daten aus *HXmlData* komprimiert und nur die komprimierten Daten in die DB ablegt. Diese Komprimierung wurde durch die Änderung deaktiviert, lediglich das DBS selbst könnte eine solche Komprimierung vornehmen, andernfalls wäre die Verarbeitung dieser Daten innerhalb des DBSs nicht möglich.

Um eine einheitliche Verarbeitung und Struktur der SQL/XPath-Ausdrücke zu erhalten, die im Prototyp generiert werden müssen, wurde die Unterscheidung zwischen einfachen Werten (*SIMPLE_VALUE*) und großen Werten (*DATA*) aufgehoben. Hierzu musste die *XmlDataDaoImpl* entsprechend verändert werden. Da XML Felder in DBSen nur wohlgeformte XML Dokumente enthalten dürfen, musste ein Wrapper Element verwendet werden, um XSD Einfache Typen im XML Feld *DATA* ablegen zu können. Dazu wurde das Element <*temporary-simple-type-wrapper*/> verwendet (siehe Tabelle 5.1, Seite 71). Dies wird innerhalb der ODE-Runtime bei der Verarbeitung von XSD Einfachen Typen verwendet und ermöglicht es so ohne weitere Änderungen den XPath-Pushdown ein- und auszuschalten. Alle anderen BPEL/XML-Typen besitzen innerhalb von Apache ODE schon eine eindeutiges XML Wurzelelement (siehe ebenfalls Tabelle 5.1). Damit ist Apache ODE ohne den Pushdown ebenfalls mit dem modifizierten Datenmodell lauffähig. Diese Version mit modifiziertem Datenmodell, aber ohne Pushdown-Funktionalität, nennen wir *instrumentalisierte Apache ODE*, sie wird ebenfalls für die Evaluiering (Kapitel 7) benötigt. Das veränderte Tabellenschema ist in Abb. 6.1 dargestellt.

6.1.2. Änderungen in der DAO-Schicht

Die DAO-Schicht verwaltet die Zugriffe auf die DB. Dies geschieht über SQL Anfragen an das DBS bzw. über das DB Middleware System Hibernate, welches letztendlich SQL Anfragen an das DBS kapselt und somit die Runtime unabhängig vom konkreten DBS und

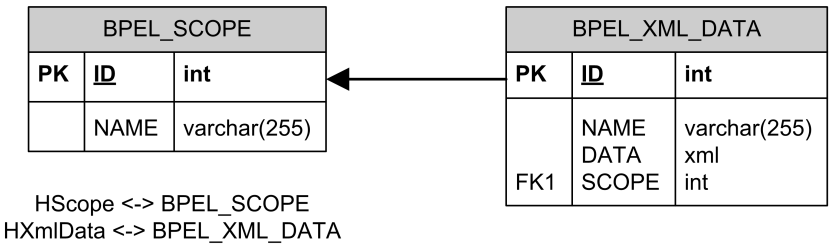


Abbildung 6.1.: Verändertes und vereinfachtes Tabellenschema für den Prototyp.

dem Datenschema macht. Deshalb ist die DAO-Schicht die Architekturschicht, in der die SQL Anfragen für den *XPath- und WS-Pushdown* generiert und an das DBS gestellt werden müssen. Abb. 6.2 zeigt das modifizierte UML-Diagramm der DAO-Schicht aus Kapitel 5.2.3. Unveränderte Klassen sind grau eingefärbt, neue Klassen und Typen sind hellblau eingefärbt, alle anderen Klassen und Schnittstellen (weiß in Abb. 6.2) wurden in irgendeiner Weise modifiziert. Private Hilfsmethoden, die benötigt werden, um die Hauptfunktionen umzusetzen, werden aus Gründen der Übersichtlichkeit nicht dargestellt. Im Folgenden gehen wir auf die einzelnen Änderungen und einige Details ein.

HibernateDao Die *HibernateDao* wird um die Methode *hibernateFlush()* erweitert, diese wird in der Methode *XmlDataDaoImpl.set()* verwendet um das Festschreiben eines Variablenwertes und somit dessen Persistenz zu erzwingen, da die Persistenz für die Realisierung des XPath- und WS-Pushdown essentiell ist.

ScopeDAO und ScopeDaoImpl Wir integrieren die in Kapitel 4 angesprochene *Pushdown*-Schicht in die *ScopeDAO*. Wir realisieren dort die vier benötigten Hauptmethoden für die Pushdown-Konzepte. Für Zuweisungen benötigen wir den asynchronen (*dataAssignByContext*) XPath-Pushdown (Assignment-Pushdown). Für Zuweisungen die noch innerhalb der WF-Runtime stattfinden sollen, aber der zuzuweisende Ausdruck im DBS ausgewertet werden soll, benötigen wir des Weiteren die synchronen XPath-Pushdown Funktionen *inDatabaseXPath* und *inDatabaseExpression* (ExpressionEvaluation-Pushdown). Insbesondere wird die synchrone XPath-Pushdown Funktion *inDatabaseExpression* ebenfalls für die Auswertung von Bedingungen und Berechnungen in anderen BPEL-Aktivitäten verwendet. Für die Realisierung des WS-Pushdown wird noch die Methode *invokeWS* benötigt. Diese vier Methoden werden in Kapitel 6.1.2.1 ausführlicher vorgestellt.

XmlDataDAO und XmlDataDaoImpl Die *XmlDataDAO* musste verändert werden, um einen veralteten Variableninhalt der in *HXmlData* gehalten wird zu kennzeichnen (*setDetached*). Dies wird benötigt, falls die Variablenzuweisung über einen WS- oder asynchronen XPath-Pushdown erfolgt. Wird der Variableninhalt dann zu einem späteren Zeitpunkt von

BpelRuntimeContext (siehe Kapitel 5.2.4, Seite 74) angefordert, muss der im Hauptspeicher gehaltene Wert zuerst mit dem Datum aus der DB aktualisiert werden.

HXmlData Die Hibernate Annotierung für *HXmlData* wurde wie in Kapitel 6.1.1 angesprochen von der komprimierten Byte Darstellung auf XML geändert. Zusätzlich wurde das Attribut *_detached* eingeführt, um die Daten als veraltet zu kennzeichnen. Diese Information wird nicht auf das Tabellenschema der Datenbank übertragen, da diese Information nur während der Laufzeit einer WF-Instanz von Interesse ist.

VariableContext Diese Klasse wird in den neuen Methoden der *ScopeDAO* verwendet. Wie in Kapitel 5.3 (Seite 79) angesprochen, muss die Zuweisungslogik, zumindest für Zuweisungen durch einen asynchronen XPath-Pushdown, auf die DAO-Schicht übertragen werden, um dort die korrekten SQL-Anfragen generieren zu können. Um den Aufruf des asynchronen XPath-Pushdown zu vereinheitlichen, wird die Wrapper Klasse *VariableContext* verwendet, die alle dafür nötigen Informationen einer Variable beinhaltet. Um ebenfalls einheitlich die Zuweisungen von Ausdrücken an Variablen zu ermöglichen, werden die *Expressions* als Pseudovariablen ohne *ID* übergeben. Insgesamt können folgende Informationen benötigt werden:

Identifier (Id) - Primärschlüssel der jeweiligen Variable in Tabelle *BPEL_XML_DATA*, für Ausdrücke *NULL*.

varType - Der Typ der jeweiligen Variable (siehe Tabelle 5.1, Seite 71) oder der Typ „Expression“, falls es sich um einen Ausdruck handelt.

type - Der XML-Typ der Variable als *Qualified Name*, für Ausdrücke *NULL*.

namespaces - Der umgebende Namensraumkontext, in dem sich die Variable oder der Ausdruck befindet, hier sind alle Namensräume und ihre etwaigen Präfixe enthalten.

path - Die Pfadselektion, für die Zuweisung von XML-Teildokumenten, bei Variablen oder der Ausdruck selbst.

exprContext - Für Ausdrücke zusätzliche Referenzen auf die im Ausdruck enthaltenen Variablen als *VariableContext*, für Variablen *NULL*.

VarType *VarType* ist eine Aufzählung der möglichen Typen, die bei der Zuweisung auftreten können. Dies ist der Typ des Attributs *varType* der Klasse *VariableContext*. Die einzelnen Typen, jeweils mit ihrem korrespondierendem OModel, lauten:

MESSAGE - OMessageVarType

ELEMENT - OElementVarType

COMPLEXXSD - OXsdTypeVarType (simple = false)

SIMPLEXSD - OXsdTypeVarType (simple = true)

EXPRESSION - Expression/OExpression

IntegratedDatabaseNativeQueries Diese *Schnittstelle* kapselt alle SQL-Anfragen und Fragmente, die für das Zusammenstellen der Anfragen an das DBS benötigt werden. Diese Schnittstelle ist nötig, um verschiedene DBSe anbinden zu können. Eigentlich wird diese Aufgabe vom DB Middleware System übernommen, jedoch verfügt das verwendete Hibernate über keine Möglichkeit die XML Verarbeitung innerhalb der eigenen *Hibernate Query Language* (HQL)[1] abzubilden. Aus diesem Grund müssen pro DBS die nativen SQL-Ausdrücke getrennt abgelegt werden. Wie Abb. 6.2 zu entnehmen ist, wurde diese Schnittstelle für IBM DB2 (*IDNDB2*) vollständig und PostgreSQL (*IDNPostgreSQL*) eingeschränkt implementiert. Diese Schnittstelle wird innerhalb von *ScopeDaoImpl* verwendet, um die konkreten SQL-Anfragen zu generieren.

[1] openJPA enthält ebenfalls keine Möglichkeit der standardisierten XML Verarbeitung

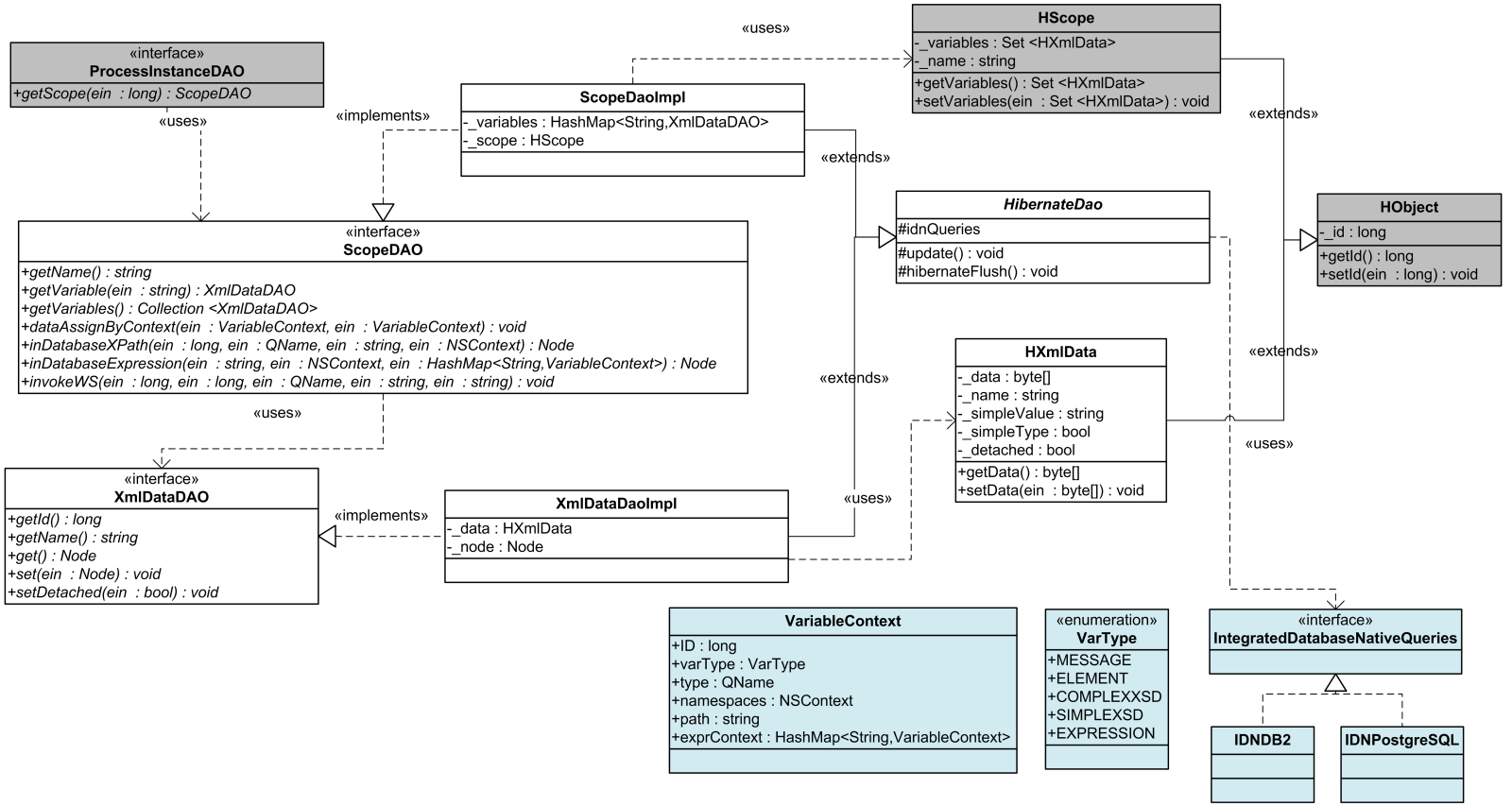


Abbildung 6.2.: UML-Diagramm eines Ausschnitts der modifizierten DAO-Schicht für den Prototypen. Graue Komponenten wurden nicht verändert, Hellblaue sind neu, weiße Komponenten wurden im Vergleich zum UML-Diagramm aus Abb. 5.4 (Seite 73) modifiziert.

6.1.2.1. Hauptmethoden von ScopeDAO

Wir stellen nun die Methoden vor, welche die Anweisungen des *WS- und XPath-Pushdown* an das DBS weitergeben. Alle vorgestellten SQL Anweisungen beziehen sich auf das DBS DB2 UDB V9.7 mit eingebettetem *pureXML*, da wir mit PostgreSQL V8.4 nicht alle Methoden realisieren können. Durch das Auflösen der Variablen durch *ScopeFrame.resolve* wird, falls noch nicht vorhanden, ein entsprechendes *XmlDataDAO* und *HXmlData* Objekt und dadurch ein Tabelleneintrag zu dieser Variable erzeugt. Deshalb müssen grundsätzlich für Zuweisungen *SQL-UPDATE* Anweisungen verwendet werden.

invokeWS ruft die *UDF* des DBSs für das Aufrufen eines WS auf.
Parameter:
inputVar (Long) - Primärschlüssel der Variable mit der Nachricht, die an den WS als Eingabe geschickt wird.
outputVar (Long) - Primärschlüssel der Variable, in die das Resultat des WS gespeichert wird.
operationNS (QName) - XML Namensraum der WS Operation.
operationName (String) - Aufzurufende WS Operation.
EPR (String) - Endpunkt URL des aufzurufenden WS.

Die *UDF*, die für den Webservice Aufruf verwendet wird lautet:

```
db2xml.soaphttpc (endpoint_url, soap_action, soap_body)
```

An diese Funktion muss der Endpunkt (*endpoint_url*), der Name der WSDL Operation (*soap_action*) und die Eingaben (*soap_body*) übergeben werden. Enthält die Nachricht bereits die Operation als Wurzel-Element, kann die Operation leer gelassen werden, diese Struktur liegt durch Apache ODE bereits vor und wir werden davon Gebrauch machen. Das vollständige SQL/pureXML Query, um den WS-Pushdown durchzuführen, ist in Listing 6.1 dargestellt. Die Parameter mit vorangestelltem Doppelpunkt sind die direkten oder modifizierten Werte der Eingangsparameter der Funktion *invokeWS*.

```
1  update bpel_xml_data set data =  db2xml.soaphttpc (':EPR', '',
2    (select XMLSERIALIZE(XMLELEMENT(
3                NAME "wsinvoke::operationName",
4                XMLNAMESPACES(':operationNS' AS "wsinvoke"),
5                XMLQUERY('declare default element namespace "*"; $DATA/message/*')
6                ) as clob)
7     from bpel_xml_data where id = :inputVar))
8  where id = :outputVar;
```

Listing 6.1: SQL/pureXML Query für den WS-Pushdown.

inDatabaseXpath stellt das SQL Query für den synchronen XPath-Pushdown innerhalb von Zuweisungen zusammen. Im Gegensatz zu *inDatabaseExpression* kann nur auf eine Variable referenziert werden und somit hauptsächlich XPath-Pfadselektionen auf ein XML-Dokument evaluiert werden (Sonderfall des ExpressionEvaluation-Pushdown).
Parameter:
varKey (Long) - Primärschlüssel der Variablen, auf die zugegriffen wird.
varType (QName) - Der XML Typ der Variable.
xPath (String) - Pfadselektion als XPath-Ausdruck.
namespaces (NSContext) - Der XML Namensraumkontext enthält alle nötigen Namensräume.
Rückgabetyp: (XML) Node

Da sich das Query nur auf eine Variable und somit auf einen XML-Schema Typen bezieht wird nur dessen Namensraum benötigt (namespaces), dieser wird in Präfix (*:prefix*) und Namensraum-URI (*:uri*) aufgetrennt. Das entsprechende SQL/pureXML Query ist in Listing 6.2 dargestellt.

```
1  select XMLQUERY('declare default element namespace "*";
2                   declare namespace :prefix=":uri";
3                   $DATA/:xPath')
4  from bpel_xml_data where id = :varKey
```

Listing 6.2: SQL/pureXML Query für den synchronen XPath-Pushdown innerhalb Zuweisungen von Variablen.

Betrachten wir das Zuweisungsbeispiel, genauer gesagt den *from* Teil aus Kapitel 4.3.1 (Seite 62):

```
<from>$y/exp:document/exp:title[@lang="de"]/text()</from>
```

Die Variable *y* enthält das XML-Beispieldokument aus Listing 2.1 (Seite 19), ihr Primärschlüssel in der Datenbanktabelle *BPEL_XML_DATA* sei *1*. Die Instanz des Queries aus Listing 6.2 für diesen Ausdruck ist in Listing 6.3 dargestellt. Der Namensraum samt Präfix wurden hinzugefügt, und der XPath-Ausdruck wurde um das Wurzel-Element (*exp:thesis*) erweitert. Dies ist notwendig, da *XPath-in-BPEL*-Ausdrücke nicht die Wurzel von XML Elementen referenzieren, da diese implizit durch die Typisierung der referenzierten Variable bekannt ist. Diese Information muss aber für die Auswertung innerhalb des DBS dem Ausdruck hinzugefügt werden. Bei Verwendung von *Message* oder *XSD* Typen müssen entsprechend die Wrapper Elemente aus Tabelle 5.1 (Seite 71) in den XPath-Ausdruck eingefügt werden.

```
1  select XMLQUERY('declare default element namespace "*";
2                   declare namespace exp="http://www.flowsoft.de/thesis/xml";
3                   $DATA/exp:thesis/exp:document/exp:title[@lang="de"]/text()')
4  from bpel_xml_data where id = 1
```

Listing 6.3: Beispielinstanz des SQL/pureXML Query aus Listing 6.2.

inDatabaseExpression stellt das SQL Query für den synchronen XPath-Pushdown eines allgemeinen XPath-Ausdrucks zusammen.
Parameter:
xPath (String) - Der zu evaluierende XPath-Ausdruck.
namespaces (NSContext) - Der XML Namensraumkontext des XPath-Ausdrucks.
exprContext (*HashMap* VariableContext) - Enthält alle Informationen zu im XPath-Ausdruck vorkommenden Variablen.
Rückgabetyp: (XML) Node

Diese Methode ist die Erweiterung von *inDatabaseXpath* auf allgemeine XPath-Ausdrücke, diese können mehr als eine Variable enthalten. Das SQL/pureXML Query muss entsprechend zusammengestellt werden. Die Basis bildet das erste Query aus Listing 6.4. Die verwendeten Namensräume werden durch die Queries zwei (Zeile 2) und drei (Zeile 3) beschrieben und anstelle des Platzhalters *:namespace* des Ersten (Zeile 1) eingesetzt. Für jede vorkommende Variable wird das vierte Query (Zeile 4) instanziiert und anstelle des Platzhalters *:variables* des Ersten eingesetzt. Die weiteren in Listing 6.4 auftretenden Platzhalter werden jeweils aus den Attributen von VariableContext zur jeweils referenzierten Variable gebildet (*:xPath*, *:id*, *:prefix* und *:uri*). Der Platzhalter *:def_namespace* wird aus dem Parameter *namespaces* abgeleitet (insofern vorhanden).

```
1  select XMLQUERY(':namespace :xPath' passing :variables) from bpel_xml_data where id = :id
2  declare default element namespace ":def_namespace";
3  declare namespace :prefix=":uri";
4  (select data from bpel_xml_data where id = :id) as ":name"
```

Listing 6.4: Aus diesen vier Teil-Queries wird das SQL/pureXML Query für den synchronen XPath-Ausdruck-Pushdown aufgebaut.

Wir erklären im Folgenden die Verwendung dieser vier Queries an einem Beispiel. Nehmen wir folgenden *XPath-in-BPEL*-Ausdruck als Beispiel:

```
concat($x/exp:author/exp:name, $y/exp:author/exp:name)
```

In den Variablen *x* und *y* sei jeweils das XML-Beispieldokument aus Listing 2.1 (Seite 19) gespeichert. Somit wird der Ausdruck die Zeichenkette *„Florian WagnerFlorian Wagner“* berechnen. Die Primärschlüssel der Variablen in der Datenbanktabelle *BPEL_XML_DATA*

seinen *1* und *2*. Als Basis dient das erste Query aus Listing 6.4. Der Platzhalter *:namespace* wird durch die Konkatenation für den Standard-Namensraum (zweites Query) und der Namensräume je Variable (drittes Query) ersetzt. Da für dieses Beispiel kein Standard-Namensraum existiert, wird der in *pureXML* mögliche Wildcard „*“ verwendet (Listing 6.5 Zeile 1). Und da beide Variablen *x* und *y* den selben Namensraum beanspruchen, kommt dieser entsprechend nur einmal vor (*exp - http://www.flowsoft.de/thesis/xml* - Listing 6.5 Zeile 2). Der Platzhalter *:xPath* des ersten Query wird entsprechend durch den modifizierten Ausdruck (mit Wurzelelementen) ersetzt (Listing 6.5 Zeile 3). Die Konkatenation des vierten Query, für alle im Ausdruck auftretenden Variablen wird im Platzhalter *:variables* ersetzt (Listing 6.5 Zeile 4-6). Damit man nur eine Zeile als Rückgabe erhält, wird der Ausdruck nur für eine beliebige im Ausdruck auftretende Variable in *BPEL_XML_DATA* zurückgegeben (*:id* im ersten Query - Listing 6.5 Zeile 1 und 7). Gegebenenfalls könnte dies auch durch eine SQL-Limitierungsanweisung erfolgen. Insgesamt folgt daraus das SQL/pureXML Query in Listing 6.5.

```
1  select XMLQUERY('declare default element namespace "*";
2                   declare namespace exp="http://www.flowsoft.de/thesis/xml";
3                   concat($x/exp:thesis/exp:author/exp:name,
                        $y/exp:thesis/exp:author/exp:name)'
4                   passing
5                   (select data from bpel_xml_data where id = 1) as "x",
6                   (select data from bpel_xml_data where id = 2) as "y")
7  from bpel_xml_data where id = 1
```

Listing 6.5: Beispiel SQL/pureXML Query für den synchronen XPath-Ausdruck-Pushdown.

dataAssignByContext ist für den asynchronen XPath-Pushdown (Assignment-Pushdown) der Zuweisung verantwortlich. Sie realisiert die Zuweisungslogik von *ASSIGN* innerhalb der DAO-Schicht.
Parameter:
lContext (VariableContext) - Linke Seite der Zuweisung.
rContext (VariableContext) - Rechte Seite der Zuweisung.

Die Methode sorgt dafür, dass innerhalb des DBSs die rechte Seite der linken Seite zugewiesen wird. *lContext* beinhaltet alle nötigen Informationen über die Variable an die zugewiesen werden soll. *rContext* stellt alle nötigen Informationen zum Auswerten der rechten Seite (Variable oder Ausdruck) zur Verfügung. Die Methode *dataAssignByContext* verwendet zahlreiche private Methoden, um den gesamten asynchronen Modus der *ASSIGN* Zuweisung zu unterstützen. Es werden die Erkenntnisse und die Aufbaulogik der SQL/pureXML Queries aus den vorangegangen Methoden *inDatabaseXpath* und *inDatabaseExpression* verwendet, um die rechte Seite der Zuweisung auszuwerten. Das Ergebnis wird dann dem XML Feld der Variablen der linken Seite innerhalb des selben SQL-Ausdrucks zugewiesen. Da eine abso-

lute Verallgemeinerung der Queries nicht möglich war, insbesondere durch die Wrapper Struktur von Apache ODE und zusätzlich deutliche Unterschiede in der XML Verarbeitung zwischen ODE und DB2 bestehen, wurde ein systematischer Ansatz gewählt, um ein lauffähiges System zu erhalten. Es können auf vier mögliche Typen der linken Seite, fünf mögliche Typen der rechten Seite zugewiesen werden. Dies ist jeweils für eine initialisierte und eine uninitialisierte linke Seite der Zuweisung möglich. Daraus ergeben sich insgesamt 40 Zuweisungsmöglichkeiten, die wir *nicht* alle im Detail besprechen.

Der Aufbau der Namensräume sowie die Veränderung der Pfadselektionen und Ausdrücke entsprechen im Wesentlichen denen aus den vorangegangen Methoden, es muss jedoch auf doppelte und gleiche Präfixe der Namensräume geachtet und diese ggf. eliminiert oder umgeschrieben werden. Wir stellen nun ein Query anhand des Beispiels aus Kapitel 4.3.1 (Seite 62) vor. Für alle anderen Möglichkeiten sei direkt auf die Implementierung verwiesen.

```
<copy>
   <from>$y/exp:document/exp:title[@lang="de"]/text()</from>
   <to variable="x"/>
</copy>
```

Die Variable *y* (Primärschlüssel *1*) enthält das XML-Beispieldokument aus Listing 2.1 (Seite 19) die Variable *x* (Primärschlüssel *2*) sei vom Typ *xsd:string*. Die Basis bildet das gleiche Query wie für den WS-Pushdown, wobei die UDF durch eine *pureXML*-Anweisung (*XMLQUERY*) ersetzt wird (siehe Listing 6.6). In dieser *pureXML*-Anweisung werden die Namensräume sowie die zu referenzierenden Variablen analog zu *inDatabaseXPath* und *inDatabaseExpression* gebildet.

```
1  update bpel_xml_data set data =
2    XMLQUERY('declare default element namespace "*";
3              declare namespace exp="http://www.flowsoft.de/thesis/xml";
4              copy $new := $DATA modify
5              do replace value of $new/temporary-simple-type-wrapper
6              with $data2/exp:thesis/exp:document/exp:title[@lang="de"]/text()
7              return $new'
8              passing
9              (select data from bpel_xml_data where id = 1 as "data2")
10             where id = 2
```

Listing 6.6: Beispiel eines SQL/pureXML Query für den asynchronen XPath-Pushdown von einem XML Element Typ an einen XSD Einfachen Typ, der initialisiert ist.

Durch den *pureXML*-Befehl *copy $new := $DATA* wird der Inhalt der zu aktualisierende Variable in *$new* gespeichert und durch den Befehl *do replace value of $new... with ...* modifiziert. Dies ist der entscheidende Schritt der Zuweisung, bei dem das Ergebnis des Ausdrucks der rechten Seite zugewiesen wird. Anschließend wird durch den Befehl *return $new* der

neue Variableninhalt zurückgegeben und an das XML-Feld von Variable *x* zugewiesen (*set data = XMLQUERY*). Ist die Variable an die zugewiesen werden soll nicht initialisiert, ändert sich der Befehl von *do replace value of* nach *do replace*. Allerdings muss dann ein wohlgeformtes XML-Dokument mit eindeutiger Wurzel zugewiesen werden und ggf. dieses Wurzelelement in den passenden Wrapper umbenannt werden (siehe Tabelle 5.1, Seite 71). Das *HXmlData* Objekt, welches von Hibernate überwacht wird, muss nach der Ausführung als *detached* markiert werden, um bei Verwendung innerhalb der Runtime aktualisiert werden zu können.

6.1.3. Änderungen in der Runtime-Schicht

In Kapitel *Änderungen in der DAO-Schicht* wurden die Änderungen in der DAO-Schicht vorgestellt, die nötig sind, um die Pushdown Konzepte zu realisieren. Diese Funktionalität muss jetzt nur noch durch die Aktivitäten aufgerufen werden, weshalb die Änderungen in der Runtime-Schicht nicht so gravierend ausfallen. Die Hauptaufgabe besteht darin, die Informationen zu Variablen und Ausdrücken aus dem *OModel* (siehe Kapitel 5.2.2, Seite 68) zu extrahieren und in geeigneter Form (direkt oder durch die Wrapper Klasse *VariableContext*) an die DAO-Schicht zu übergeben, die dann mit dem DBS kommuniziert. Das UML-Diagramm ist in Abb. 6.3 zu sehen, grau eingefärbte Klassen wurden nicht verändert. Wir gehen nun auf die veränderten Schnittstellen und Klassen ein. Tabelle 6.2 fasst die Aufrufhierarchie der Methoden aus ScopeFrame, BpelRuntimeContext und der in Kapitel 6.1.2.1 vorgestellten Methoden der ScopeDAO zusammen.

BpelRuntimeContext und BpelRuntimeContextImpl stellen vier *XPath-Pushdown* Methoden (*inDatabaseAssign*, *inDatabaseExpressionAssign*, *inDatabaseXPath*, *inDatabaseXPathExpression*) bereit, die ausschließlich von *ScopeFrame* aus aufgerufen werden. Es werden die aufgelösten Variablen als *VariableInstance (VI)* sowie XPath-Ausdrücke (*OXPath10Expression*) übergeben und die für die DAO-Schicht notwendigen Informationen extrahiert oder in Objekte der Wrapper Klasse *VariableContext* (siehe Kapitel 6.1.2) der DAO-Schicht überführt. Die Methode *inDatabaseInvoke* wird hingegen direkt durch die Aktivität *INVOKE* aufgerufen. Anschließend wird die Aufgabe entsprechend an eine der vier ScopeDAO Hauptmethoden (siehe Kapitel 6.1.2.1) weitergegeben.

ScopeFrame stellt drei *inDatabase* Methoden bereit, die von den BPEL Aktivitäten aufgerufen werden können. Hierbei führt der Aufruf von *inDatabaseAssign* zu einem asynchronen XPath-Pushdown (Assignment-Pushdown). Hierbei muss wegen des unterschiedlichen OModels für Ausdrücke (*OExpression*) und Variablen (*VariableRef*) jeweils die entsprechende Methode des BpelRuntimeContext (*inDatabaseAssign* oder *inDatabaseExpressionAssign*) aufgerufen werden. Diese vereinheitlichen dann jeweils den Aufruf an die ScopeDAO (*dataAssignByContext*). Der Aufruf von *inDatabaseXPath* führt zu

einem synchronem XPath-Pushdown für Zuweisungen und der Aufruf von *inDatabase-XPathExpression* zum allgemeinen synchronen XPath-Pushdown (ExpressionEvaluation-Pushdown), der auch für die Evaluierung von Bedingungen in den Kontroll- und Schleifenkonstrukten genutzt werden kann.

ACTIVITY und Unterklassen verwenden, bis auf *INVOKE*, innerhalb ihrer Logik die XPath-Pushdown Methoden von *ScopeFrame*. *INVOKE* greift direkt auf die Methode *inDatabaseInvoke* von *BpelRuntimeContext* zu.

ScopeFrame	BpelRuntimeContext	ScopeDAO
inDatabaseAssign	inDatabaseAssign inDatabaseExpressionAssign	dataAssignByContext
inDatabaseXPath	inDatabaseXPath	inDatabaseXPath
inDatabaseXPathExpression	inDatabaseXPathExpression	inDatabaseExpression
-	inDatabaseInvoke	invokeWS

Tabelle 6.2.: Aufrufhierarchie zwischen den XPath-Pushdown Methoden aus ScopeFrame, BpelRuntimeContext und ScopeDAO.

Durch diese Architektur, also die Implementierung der Pushdown-Methoden in BpelRuntimeContext und ScopeFrame, ist die Übertragung auf die in Tabelle 5.2 (Seite 79) genannten Aktivitäten ohne weitere Schwierigkeiten möglich. Zudem können auch zukünftige oder benutzerdefinierte Aktivitäten den *XPath-Pushdown* verwenden.

6.2. Funktionalität des Prototyps

Wir präsentieren nun die, durch die Implementierung, realisierten Funktionalitäten, stellen potentielle weiterführende Änderungen vor und berichten über konkrete Schwierigkeiten, die sich bei der Implementierung des Prototyps ergaben.

6.2.1. Realisierte Pushdown-Konzepte

Das Konzept des Query-Pushdown wurde in der Ausprägung als XPath-Pushdown in Zusammenwirken mit IBM DB2 vollständig implementiert. Der Webservice-Pushdown wurde rudimentär implementiert, hier fehlt die korrekte Logik zur Fehlerbehandlung und Fehlerweitergabe an Apache ODE. Um Messungen zur Performanz durchzuführen, eignet sich die Implementierung dennoch. Generell, auch für den WS-Pushdown, wird im Falle eines Fehlers in einer der hinzugefügten Methoden oder durch das DBS die ursprüngliche Logik

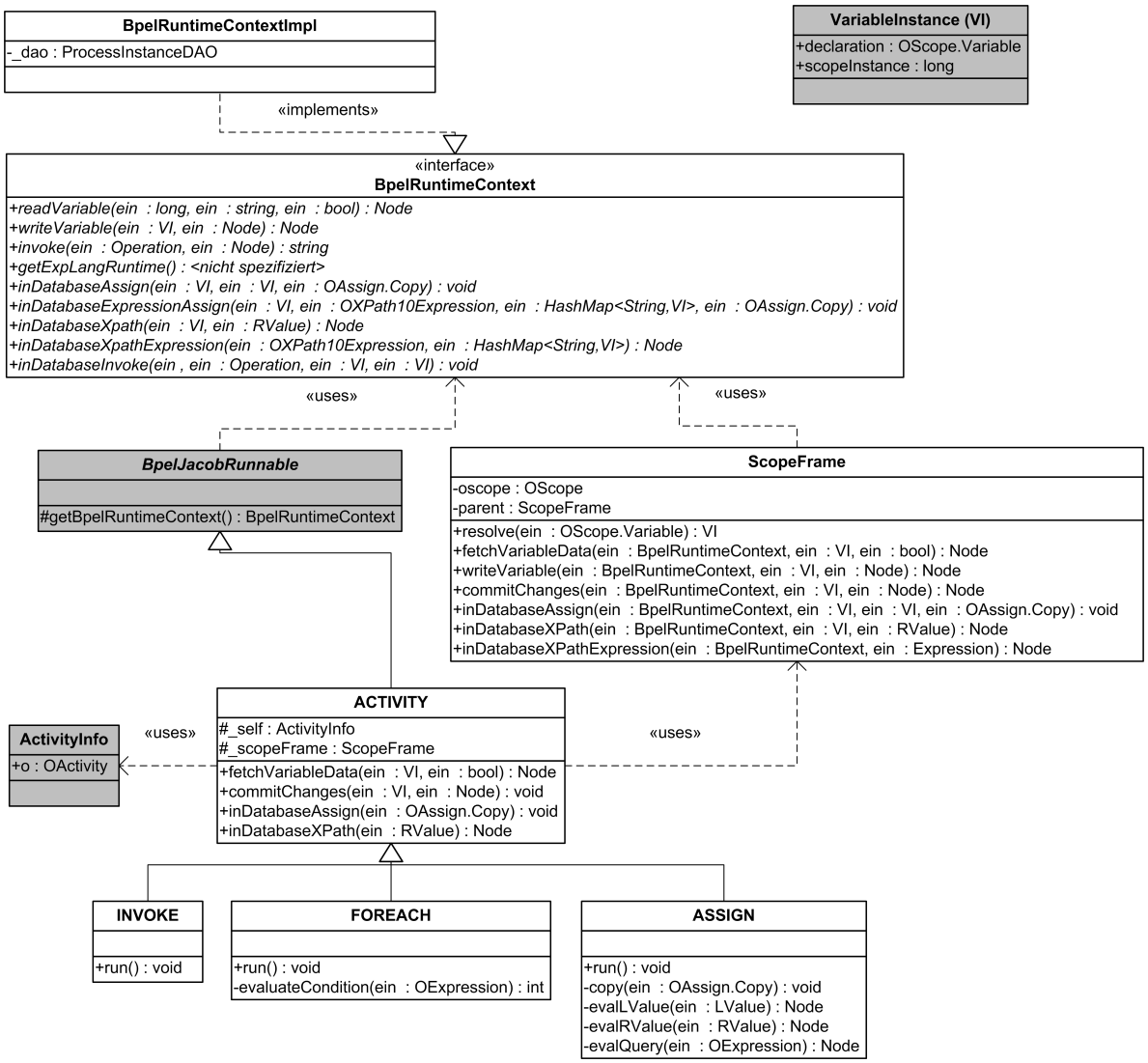


Abbildung 6.3.: UML-Diagramm der veränderten Runtime-Schicht des Prototyps. Grau gefärbte Klassen wurden nicht verändert. Zu den Lese- und Schreibmethoden auf Variablen kommen die WS- und XPath-Pushdown Methoden hinzu, diese leiten die Anfragen an die DAO-Schicht weiter.

der Aktivitäten durchlaufen. Für die in Kapitel 7 verwendeten Test- und Anwendungsfälle ergeben sich in der Implementierung des Prototyps keine Fehler, wodurch die Vergleichbarkeit der Messergebnisse gegeben ist.

Die Aktivitäten aus Tabelle 5.2 (Seite 79), die potentiell die Pushdown Konzepte verwenden können, sind nochmals, mit dem Stand ihrer Umsetzung im Prototyp, in Tabelle 6.3 angegeben.

ODE Klasse (BPEL-Aktivität)	Art des Pushdown	Modus des Pushdown	Implementiert
ASSIGN	XPath (Assignment & ExpressionEvaluation)	asynchron & synchron	ja
INVOKE	Webservice	-	ja (ohne Fehlerbehandlung)
FOREACH	XPath (ExpressionEvaluation)	synchron	ja
WHILE	XPath (Condition)	synchron	ja
REPEATUNTIL	XPath (Condition)	synchron	ja
SWITCH (IF)	XPath (Condition)	synchron	ja
EH_ALARM (ONALARM)	XPath (ExpressionEvaluation)	synchron	ja
WAIT	XPath (ExpressionEvaluation)	synchron	ja
ACTIVITYGUARD (TransitionConditions)	XPath (Condition)	synchron	ja

Tabelle 6.3.: Alle ODE Klassen, die potentiell von den Pushdown-Konzepten Gebrauch machen können und der Stand ihrer Implementierung.

Zusätzlich wurden Parameter zum Ein- und Ausschalten der verschiedenen Pushdown-Konzepte der ODE Konfigurationsdatei hinzugefügt (siehe Anhang B.4.2, Seite 148). Dadurch ist es möglich die Wirkung einzelner Pushdown-Konzepte, insbesondere der Vergleich der synchronen und asynchronen Variante für Zuweisungen, zu untersuchen. Außerdem muss man für die Verwendung des DBSs PostgreSQL V8.4 Pushdown-Konzepte, die für dieses DBS nicht umgesetzt werden konnten, deaktivieren können.

6.2.2. Technische Schwierigkeiten

Alle technischen Schwierigkeiten, die während der Implementierung auftraten, hängen direkt mit der XML Technologie oder mit der Art und Weise, wie XML Dokumente von verschiedenen Systemen und Implementierungen verarbeitet werden, zusammen. Insbesondere

muss für jedes anzubindende DBS die Möglichkeiten und Funktionen der XML Verarbeitung untersucht und getestet werden. Die Verwendung von unterschiedlichen Wrapper Elementen (siehe Tabelle 5.1, Seite 71) innerhalb der WF-Runtime verhindert die Verallgemeinerung der auszuführenden Queries erheblich.

Die größten Schwierigkeiten werden jedoch durch die XML Namensräume hervorgerufen. Diese müssen in jeder Anfrage an das DBS übergeben werden, um die XPath Ausdrücke korrekt evaluieren zu können, insbesondere wenn Präfixnotationen innerhalb der Ausdrücke verwendet werden. Hierbei können folgende Situationen auftreten:

Gleicher Präfix mit gleichem Namensraum Dies kommt bei Verwendung von zwei Variablen mit gleichem Namensraum vor, die doppelte Namensraumangabe muss unterbunden werden, da dies zu Fehlern im DBS führen kann.

Gleicher Präfix mit unterschiedlichem Namensraum Dies kann bei der Zuweisung von zwei Variablen mit unterschiedlichem Namensraum vorkommen, ein Präfix muss dabei innerhalb des Ausdrucks umgeschrieben werden.

Unterschiedlicher Präfix mit gleichem Namensraum Ist laut Spezifikation der XML Namensräume explizit erlaubt und sollte keine Fehler verursachen, die konkrete Implementierung im DBS sollte jedoch überprüft werden.

Mehrere Standard-Namensräume Dies kann ebenfalls bei einer Zuweisung zweier Variablen mit unterschiedlichem Namensraum vorkommen. Hier können je nach DBS entweder sog. *Wildcards* für Standard-Namensräume (*engl. Default Namespaces*) verwendet werden, oder es muss mindestens ein Standard-Namensraum in einen Namensraum mit Präfix überführt werden, was ebenfalls eine Anpassung des Ausdrucks nach sich zieht.

Zusätzlich kann es zu allgemeinen Schwierigkeiten mit Standard-Namensräumen kommen. In *pureXML* wird die Angabe eines Standard-Namensraums unterstützt, es kann sogar ein Wildcard eingeführt werden, der Lokalisierungsschritte ohne Präfixangabe automatisch ihrem Standard-Namensraum zuordnet. Die Angabe eines Standard-Namensraums ist in PostgreSQL hingegen nicht möglich, hier muss für die korrekte Verarbeitung des Ausdrucks extra ein Präfix für den Standard-Namensraum eingeführt werden und der Ausdruck entsprechend abgeändert werden.

Diese und weitere Probleme mit XML Namensräumen führen dazu, dass Stimmen laut werden, XML Namensräume würden nur zur Verwirrung führen und Auswertungen von XML Daten unnötig kompliziert machen und sollten spärlich eingesetzt werden [Dar05]. Allerdings sind Namensräume aus Sprachen wie WS-BPEL nicht wegzudenken, sie sind essentiell für die Typisierung der Variablen und deren Validierung nötig. Ignoriert man bei der Verarbeitung der Ausdrücke auf XML Daten generell die XML Namensräume, kann dies prinzipiell zu semantisch falschen Ergebnissen führen, was eine zwingend korrekte Verwendung der Namensräume nötig macht.

6.2.2.1. Implementierung für PostgreSQL

Für PostgreSQL konnte leider nur ein Bruchteil der Funktionalitäten umgesetzt werden. In der verwendeten Version 8.4 gibt es keine Möglichkeit einen WS aufzurufen, womit die Umsetzung des WS-Pushdown hinfällig ist. Die XML Verarbeitungsmöglichkeiten sind im Vergleich zur DB2 von IBM stark eingeschränkt. Es gibt keine Sprachelemente, mit denen es möglich ist XML Daten ähnlich zu *pureXML* oder *XQuery Update* zu transformieren. Die Evaluierung von XPath Ausdrücken erfolgt in PostgreSQL durch die Funktion *xpath*. Der Rückgabewert dieser Funktion erfolgt als Zeichenkette und wird mit zusätzlichen Anführungszeichen annotiert. Dies führt zum einen dazu, dass eine direkte Zuweisung an ein XML Feld innerhalb der Datenbank nicht möglich ist, also ein asynchroner Pushdown nicht durchführbar ist. Andererseits muss eine komplizierte Bereinigung des Rückgabewerts bei synchronen Anfragen innerhalb der DAO-Schicht erfolgen, damit die Daten im für ODE richtigen Format übergeben werden. Darüber hinaus kann die *xpath* Funktion nur auf jeweils ein XML Feld zugreifen, wodurch Evaluierungen von Ausdrücken mit mehreren Variablen ebenfalls nicht möglich sind. Zudem wird nur ein Bruchteil der XPath Spezifikation realisiert, womit im Prinzip nur noch Pfadausdrücke auf ein XML Dokument evaluierbar sind. Die einzige XPath-Pushdown Variante, die also derzeit mit PostgreSQL abgebildet werden konnte, ist der synchrone XPath-Pushdown für BPEL ASSIGN (ExpressionEvaluation-Pushdown), bei dem die rechte Seite der Zuweisung eine Variable mit Pfadselektion darstellt (siehe *inDatabaseXPath* aus Kapitel 6.1.2.1). Die Mächtigkeit der PostgreSQL-*xpath* Funktion ist auch in der aktuellen Version 9 gleich geblieben. Allerdings konnte die Version 9 nicht korrekt mit Hibernate kommunizieren, weshalb auf die Version 8.4 zurückgegriffen wurde. Falls in zukünftigen PostgreSQL Versionen die XML Verarbeitungsmöglichkeiten ausgebaut werden, ist es ggf. möglich eine vollständige Implementierung des XPath-Pushdown zu erreichen.

6.2.3. Weiterführende Modifikationen

Es sind eine Reihe weiterer Modifikationen für Apache ODE vorstellbar. Die meisten dieser Modifikationen gehen in Richtung eines Produktivsystems, einige bieten zusätzliche Funktionalität an, andere können wissenschaftlich untersucht werden. Teilweise werden wir die hier genannten Modifikationen im Kapitel 8, Konzeptionelle Erweiterungen, nochmals aufgreifen. Das Konzept aller genannten weiterführenden Modifikationen beruht nach wie vor auf der Verlagerung der Funktionen von der WF-Runtime auf das DBS (siehe Kapitel 4.1, Seite 57). Einige sind direkte weiterführende Ausprägungen des Query-Pushdown Konzepts.

Fehlerbehandlung WS-Pushdown Die Fehlerbehandlung kann noch vollständig implementiert werden. Für jedes DBS mit SOAP-Unterstützung müssen hierbei die SQL Fehler Codes auf die entsprechenden ODE Ausnahmen abgebildet werden, damit ODE oder der BPEL-Prozess entsprechend auf die Fehler reagieren können. Als Vervollständigung

der Implementierung stellt dies eine reine Erweiterung für Apache ODE dar und wird deshalb nicht in Kapitel 8 aufgeführt.

Variablen Eigenschaften BPEL-Variablen können als zusätzliche Metainformation mit sog. Eigenschaften (*engl. Variable Properties*) versehen werden. Diese Eigenschaften können ebenfalls durch Zuweisungen verändert werden. Der XPath-Pushdown für die Zuweisung dieser Eigenschaften kann analog implementiert werden, die Struktur hierfür ist bereits in der Methode *inDatabaseAssign* von *BpelRuntimeContextImpl* (siehe Abb. 6.3) vorgesehen. Die Tabelle *VAR_PROPERTIES* des ODE DB-Schemas müsste analog zu *BPEL_XML_DATA* angepasst, der *VariableContext* um die Eigenschaften erweitert und die Zuweisungslogik der ScopeDAO hinzugefügt werden. Als Funktionsvervollständigung für Apache ODE wird dies ebenfalls nicht in Kapitel 8 aufgeführt.

XML-Wrapper vereinheitlichen Speziell für den Prototypen wäre es möglich, die XML-Wrapper Elemente aller Typen von BPEL-Variablen (siehe Tabelle 5.1, Seite 71) zu vereinheitlichen und so die Implementierung des XPath-Pushdowns zu vereinfachen bzw. die Komplexität der Generierung der SQL-Anweisungen zu reduzieren. Dies wird ebenfalls nicht in Kapitel 8 diskutiert.

XQuery-Pushdown Der synchrone XQuery-Pushdown kann für IBM DB2 mit kleineren Änderungen innerhalb der Runtime-Schicht realisiert werden. Hierbei muss das *OModel* für XQuery-Ausdrücke entsprechend auf den *VariableContext* abgebildet werden. Ob die Einbettung der XQuery Ausdrücke für einen asynchronen XQuery-Pushdown innerhalb der pureXML Queries von DB2 möglich ist, muss gesondert evaluiert werden, sollte jedoch ebenfalls realisierbar sein. Diese konsequente Erweiterung des Query-Pushdown wird in Kapitel 8 aufgegriffen.

pureXML/XUpdate-Pushdown XUpdate [AL00] sowie pureXML von IBM stellen Funktionen zur Manipulation von XML Daten bereit, wie Löschen, Austauschen und Hinzufügen von XML Knoten. Es existiert derzeit noch kein Standard, allerdings sollten die Manipulationskomponenten dieser Sprachen voraussichtlich durch die XQuery Update Facility Recommendation [W3C09] ersetzt werden. Ob die direkte Manipulation von Variablen durch den Anwender innerhalb der Auswertung von z.B. Schleifenbedingung jedoch sinnvoll ist, muss gesondert diskutiert werden. Für zukünftige WfMSe und DBSe könnte dies dann auch über einen Standard wie XQuery Update erfolgen. Es sei hier darauf hingewiesen, dass für den asynchronen XPath-Pushdown (siehe Kapitel 6.1.2.1) die Manipulationskomponenten von pureXML dazu verwendet werden die BPEL ASSIGN Anweisung innerhalb des DBS auszuführen. Ob eine verschachtelte Ausführung möglich ist, muss gesondert evaluiert werden. Aufgrund der interessanten konzeptionellen Erweiterung und der Möglichkeit der Standardisierung des Query-Pushdown greifen wir dies ebenfalls in Kapitel 8 auf.

Anbindung weiterer XML-Enabled DBS Es könnten weitere XML-Enabled DBS in den Prototypen eingebunden werden. Ein zukünftiger Standard für die Manipulation von

XML Feldern und dessen Implementierung in gängigen DBSen würde die getrennte Implementierung jedoch überflüssig machen. Dies greifen wir insbesondere wegen der erneuten Auswertung zukünftiger XML-Enabled Systeme nochmals in Kapitel 8 auf.

XSD Schema Validierung Die Validierung der Variablen nach ihrer Zuweisung, wie sie im BPEL Standard vorgesehen ist, wird derzeit von Apache ODE nicht implementiert[2]. Es sollte möglich sein, die einzelnen XSD Spezifikationen in den XSD Schema Speicher der Datenbank zu laden und die Validierung innerhalb des DBSs zu veranlassen, zumindest stellt DB2 mit pureXML prinzipiell Funktionen hierzu zur Verfügung. Als direkte Funktionserweiterung von Apache ODE stellen wir dies in Kapitel 8 nicht weiter vor.

[2]http://ode.apache.org/ws-bpel-20-specification-compliance.html

7. Evaluierung des Prototyps

In diesem Kapitel stellen wir die Ergebnisse der Laufzeit- und Hauptspeichermessungen des Prototyps aus Kapitel 6 vor. In Abschnitt 7.1 stellen wir die verwendeten Testfälle, in Abschnitt 7.2 die verwendete Testumgebung vor. In Abschnitt 7.3 stellen wir die relevanten Resultate vor, deren Bedeutung sowie technische Einschränkungen des Prototyps werden in Abschnitt 7.4 diskutiert.

7.1. Vorstellung der Testfälle

Um die Auswirkungen der einzelnen Pushdown-Konzepte aus Kapitel 4 besser zu veranschaulichen, wurden Einzeltests erstellt. Diese basieren auf drei Basis-Workflows (Abb. 7.1), die sich dann in der Komplexität der XPath-Ausdrücke und der verwendeten Datengröße unterscheiden. Der erste Basis-Workflow enthält nur eine ASSIGN-Aktivität, anhand der wir Messungen für verschiedenartige Zuweisungen machen (Abb. 7.1a). Diese WFs decken somit vollständig den Assignment-Pushdown ab, sowie Teile des ExpressionEvaluation-Pushdown und sein Verhalten bei Verwendung innerhalb von Zuweisungen. Je nach XML Verarbeitungsmöglichkeiten des DBSs ist für ASSIGN nur der synchrone XPath-Pushdown (ExpressionEvaluation-Pushdown) anwendbar. Der zweite Basis-Workflow steht für die Untersuchung des ExpressionEvaluation-Pushdown (Abb. 7.1b). Dieser WF benötigt eine vorgeschaltete Zuweisung um überhaupt ein XML-Dokument im Prozessspeicher zu haben, auf den ein Ausdruck evaluiert werden kann. Da die Implementierung des synchronen

Klasse/Konzept	Art des Pushdown	**Stellvertretende** BPEL-Aktivität
Assignment	asynchroner XPath	**ASSIGN**
Expression-Evaluation	synchroner XPath	**ASSIGN**, **IF**, WHILE, FOREACH, REPEAT UNTIL, WAIT, SWITCH (BPEL 1.0)
Webservice	Webservice	**INVOKE**
Anwendungsfall	asynchroner XPath synchroner XPath Webservice	ASSIGN, FOREACH, IF, INVOKE

Tabelle 7.1.: Testfälle der Einzelmessungen im Überblick.

XPath-Pushdown (ExpressionEvaluation-Pushdown) für alle Aktivitäten (außer ASSIGN, siehe oben) gleich ist, reicht es aus, diesen für eine Stellvertreter-Aktivität, im konkreten Fall *IF*, zu untersuchen. Die nachgelagerte Zuweisung weißt zu Überprüfungszwecken einer Variablen den Wert „true“ zu, in dem Fall wenn die Auswertung der Bedingung wahr ist. Der dritte Basis-Workflow steht für die Überprüfung des Webservice-Pushdown (Abb. 7.1c). Diese WFs benötigen ebenfalls eine vorgelagerte Zuweisung um die WSDL Nachricht an den Webservice zu generieren.

Zusätzlich zu den Einzelmessungen wird der Anwendungsfall aus der Proteinmodellierung (siehe Kapitel 2.5.1.1, Seite 39), als Workflow mit datenverarbeitenden Schritten innerhalb einer Schleife, verwendet. Der zugehörige WF enthält Aktivitäten (ASSIGN, FOREACH, IF, INVOKE) zu allen drei Klassen der in den Prototyp implementierten Pushdown-Konzepten. Die Tabelle 7.1 gibt einen Überblick über die vorgestellten Testklassen.

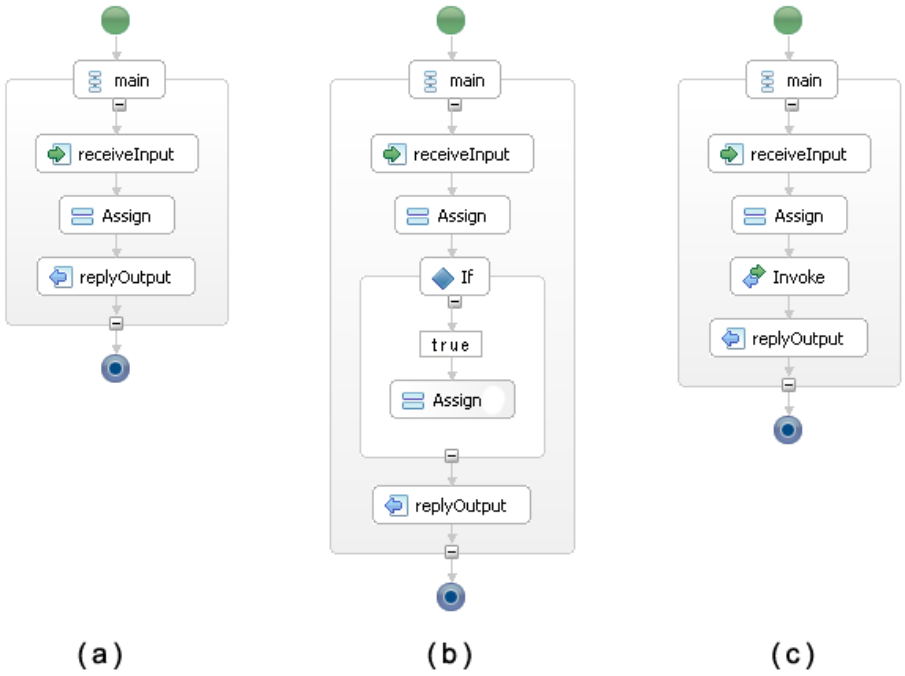


Abbildung 7.1.: Graphische Repräsentation der BPEL-Workflows für die Einzeltests.

Alle Testfälle werden jeweils mit drei unterschiedlichen Datengrößen (100kb, 500kb, 4MB) evaluiert. Hier wird jeweils ein entsprechendes XML Dokument des XML Formats BIIF aus [Wag10] verwendet. Als Beispiel ist ein solches (gekürztes) XML Dokument in Anhang C.1 (Seite 149) angegeben. Die tatsächlichen XML Dokumente unterscheiden sich nur in der Anzahl der Proteineinträge (<*seq*>-Elemente) innerhalb des <*aln*>-Elements. Zusätzlich werden die Einzeltest für den Assignment- und ExpressionEvaluation-Pushdown für verschiedene Komplexitäten des XPath-Ausdrucks evaluiert:

1. Ohne XPath-Ausdruck (nur Zuweisung einer Variable auf eine Andere)
2. XPath-Selektion des ersten <*seq*>-Elements
3. XPath-Selektion des mittleren <*seq*>-Elements
4. XPath-Selektion des letzten <*seq*>-Elements
5. Komplexer XPath-Ausdruck: Zugriff auf zwei Variablen mit gleichem Inhalt, Konkatenation der ersten Proteinsequenz der ersten Variablen mit letzter Proteinsequenz der zweiten Variablen.

Die Resultate der XPath-Selektionen (Varianten 2-4) werden gemittelt und als 'einfache XPath-Selektionen' betrachtet. Es wurden, soweit nicht anderes angegeben, pro Testfall jeweils 100 sequentielle Ausführungen durchgeführt. Aufgrund der hohen Anzahl an Testfällen wurden nur für ausgewählte Testfälle parallele Messungen vorgenommen. Hier wurden ebenfalls, falls nicht anders angegeben, 100 parallel laufende Instanzen gemessen. Aus diesen 100 Messungen wurde dann jeweils der Mittelwert gebildet (siehe auch Vorbemerkung zur Vergleichbarkeit der Messungen Kapitel 7.3.1).

7.2. Testumgebung und Durchführung

Die Messungen wurden auf dem Entwicklungssystem (siehe Anhang B, Seite 145) durchgeführt, da hier alle DBSe installiert und für die Verwendung mit Apache ODE konfiguriert waren. Außerdem ist auf diesem System die Lizenz für IBM DB2 vorhanden. Das System ist mit Windows XP Professional 32-bit (Service Pack 3) installiert, verfügt über eine zwei Kern CPU (Intel Core2Duo T7300@2GHz) sowie über 3GB Hauptspeicher und einer Grafikkarte mit eigenständigem Grafikspeicher. Die Messungen wurden jeweils für drei bzw. vier Varianten der WF-Engine Apache ODE durchgeführt:

Prototyp mit asynchronem Pushdown Im Zusammenhang mit dem Assignment-Pushdown (*ASSIGN*) und dem DBS IBM DB2 bei eingeschaltetem asynchronem XPath-Pushdown.

Prototyp mit synchronem Pushdown Verwendet ausschließlich den ExpressionEvaluation-Pushdown (*ASSIGN*, *IF*, *WHILE* etc.) in Form des synchronen XPath-Pushdown. Entsprechend werden innerhalb von Zuweisungen nur die Teilausdrücke im DBS ausgewertet aber innerhalb der Runtime zugewiesen. Diese Version wird im Zusammenhang mit dem DBSe IBM DB2 und PostgreSQL genannt.

Instrumentalisierte ODE Version und Tabellenschema des Prototyps, aber vollständig abgeschaltete XPath- und WS-Pushdown Funktionalität.

Original ODE Apache ODE in Version 1.3.4 und original Tabellenschema.

Der WS-Pushdown ist in beiden Prototyp-Varianten, soweit vom DBS unterstützt (somit nur für IBM DB2), eingeschaltet. Für die Messungen des Anwendungsfalls werden jeweils alle möglichen Pushdown-Funktionen, die PostgreSQL oder DB2 umsetzen können, eingeschaltet und diese Variante als *Prototyp* bezeichnet. Zudem wurden alle Versionen durch Zeitmessungen, für die stellvertretenden Aktivitäten aus Tabelle 7.1, erweitert. Die Zeitmessung der Aktivitäten wird jeweils in der Log-Datei von Apache Tomcat vermerkt und mit Hilfe eines Perl-Skripts ausgewertet. Hierbei wird der Durchschnitt aller Messpunkte zu dieser Aktivität über alle Instanzen hinweg berechnet. Falls nicht anders vermerkt, wurden jeweils 100 Wiederholungen des gleichen Testfalls mit Hilfe eines weiteren Perl-Skripts automatisch durchgeführt und die durchschnittliche, minimale und maximale Instanzlaufzeit durch SQL-Anfragen an das DBS berechnet. Ebenfalls wurde die Gesamtlaufzeit über die Startzeit der ersten Instanz bis Endzeit der letzten Instanz über eine SQL-Anfrage bestimmt (Apache ODE vermerkt diese Informationen in der Tabelle *BPEL_INSTANCE*). Der Hauptspeicherverbrauch wurde über die Differenz von maximal und minimal verfügbarem Hauptspeicher während einer Messung ermittelt. Hierzu wurde der 'verfügbare Hauptspeicher' durch den Leistungsmonitor von Windows XP in einer CSV Datei aufgezeichnet, die Auswertung erfolgte ebenfalls über ein Perl-Skript. Der Ablauf jeder Messung erfolgte nach folgenden Schritten:

1. Löschen aller Datenbankinhalte für die aktuelle ODE DB, sowie Löschen der aktuellen Apache Tomcat Log-Datei und Löschen der Inhalte im *processes* Ordner der verwendeten Apache ODE Version.
2. Kopieren des aktuellen Testfalls in den *processes* Ordner der aktuellen Apache ODE Version.
3. Starten der gemessenen ODE Version, bis Testfall compiliert und zur Verfügung gestellt (*deployed*) wurde.
4. Stoppen der gemessenen ODE Version, Starten der Hauptspeichermessung.
5. Starten der gemessenen ODE Version, Start der automatisierten Testfall-Ausführung (Perl-Skript).
6. Nach Ausführung aller Instanzen, Stoppen der aktuellen ODE Version.
7. Stoppen der Hauptspeichermessung.
8. Eintragen der Ergebnisse in eine Tabellenkalkulation zur Weiterverarbeitung.

Die Dateien mit den Rohdaten und den Testfällen sowie die Tabellenkalkulation mit den zusammengetragenen Ergebnissen liegen der DVD unter *[DVD]/Evaluation* bei. Im nächsten Abschnitt werden die Ergebnisse der Messungen vorgestellt und anschließend diskutiert.

7.3. Vorstellung der Messergebnisse

Da wir nur an der Tauglichkeit der Pushdown-Konzepte aus Kapitel 4 interessiert sind, werden wir nur relative Laufzeiten zu einer Bezugsversion von Apache ODE, jeweils für das gleiche DBS, vorstellen. Hierzu treffen wir im nächsten Abschnitt einige Vorbemerkungen zur Vergleichbarkeit der Messungen. Anschließend stellen wir die Ergebnisse nach den Klassen aus Tabelle 7.1 getrennt vor. Da die Hauptspeichermessungen für die Einzeltestfälle der Aktivitäten für uns keine sichtbare Aussagekraft besitzen, stellen wir diese nicht vor. Für den Anwendungsfall hingegen kann man einige Aussagen treffen und wir stellen die Resultate absolut in MB vor. Die Datenbanksysteme wurden ohne Modifikation ihrer Werkseinstellungen verwendet. Die Einzelmessungen liegen der DVD in der Datei *[DVD]/Evaluation/Evaluation_ErgebnisListe.xlsx* bei.

7.3.1. Vorbemerkung zur Vergleichbarkeit der Messungen

Bei der Implementierung des Prototyps fiel auf, dass die verwendete Hibernate-DAO der Original Apache ODE Version die Daten erst nach Ende der Instanz oder an bestimmten Stellen im Workflow (z.B. während eines INVOKE) in die Datenbank überträgt und festschreibt. Damit die Pushdown-Konzepte jedoch anwendbar sind, müssen die Daten bereits in der DB abliegen, weshalb bei jedem Aufruf der *set*-Methode der XmlDataDaoImpl (Abb. 6.2, Seite 86) ein sog. *Flush*, ein erzwungenes Eintragen der Daten in die DB durch Hibernate, innerhalb des Prototyps eingeführt werden musste (siehe auch Klasse *HibernateDao* im UML-Diagramm Abb. 6.2, Seite 86). Bei diesem *Flush* werden alle Daten, also auch Prozess- und Auditingdaten, die sich in der DAO-Schicht befinden unselektiv durch Hibernate mit der DB synchronisiert. Da es im Prototyp innerhalb der *ASSIGN* Aktivität zu Flushs kommen kann, sind die isolierten Messwerte für die ASSIGN-Aktivität nicht direkt mit den Werten der Original ODE Version vergleichbar.

Des Weiteren ergab sich eine große Diskrepanz zwischen den gemessenen durchschnittlichen Instanzlaufzeiten (ermittelt aus dem Durchschnitt der einzelnen Instanzlaufzeiten) und den berechneten durchschnittlichen Instanzlaufzeiten (ermittelt aus Gesamtlaufzeit durch Anzahl durchlaufener Instanzen) für die Original ODE Version. Um die Vermutung beweisen zu können, dass in der Originalversion, insbesondere für die kurz laufenden Testfälle, die Datenbankkommunikation erst nach Ende der Instanz erfolgt, wurde eine zusätzlich Version der Instrumentalisierten ODE erstellt. In dieser wurde der erzwungene Flush ausgesetzt, sie unterscheidet sich zur Original ODE also nur noch durch das XML Feld anstelle des BLOB Felds in der Datenbank. Wir vergleichen nun die Resultate eines Testfalls für Instrumentalisierte ODE mit Flush, Instrumentalisierte ODE ohne Flush und der Original ODE Version in Abb. 7.2. Man kann deutlich erkennen, dass die gemessene Instanzlaufzeit der Instrumentalisierten ODE ohne Flush deutlich geringer ausfällt als mit Flush, jedoch die berechnete Instanzlaufzeit weitgehend identisch ist. Der einzige Unterschied zwischen

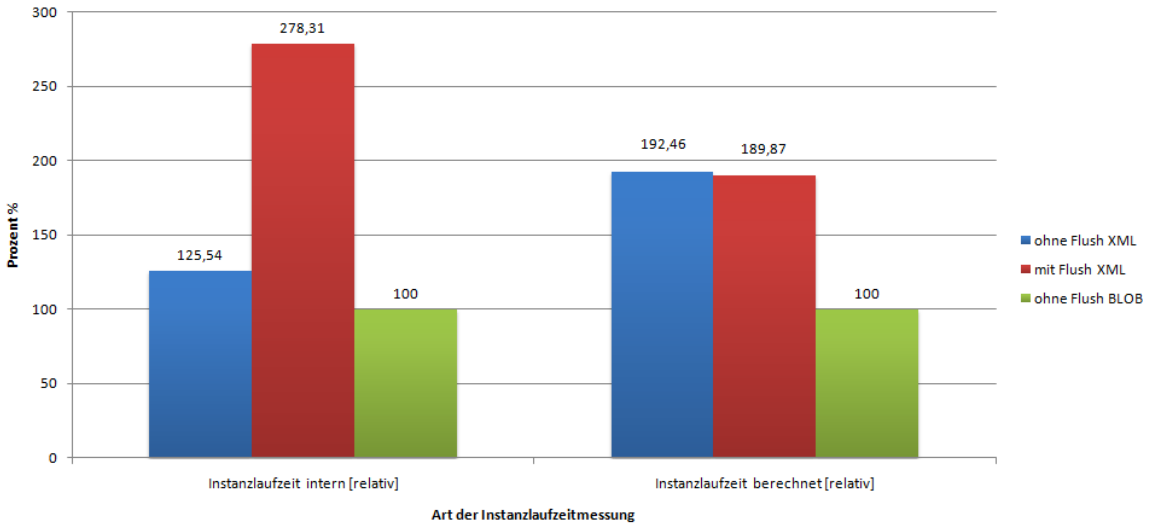


Abbildung 7.2.: Die Vergleichbarkeit der Messungen zwischen Original ODE (ohne Flush BLOB) und allen anderen Versionen ist nur für die berechnete Instanzlaufzeit gegeben, da die Datenbankkommunikation bei Versionen ohne Flush nicht in die gemessene Zeit einzelner Instanzen eingeht (linke Seite), sich jedoch in der Gesamtlaufzeit aller Instanzen widerspiegelt (rechte Seite). (DBS: IBM DB2)

Instrumentalisierter ODE ohne Flush und Original ODE ist der verwendete Typ des Datenbankfeldes. Da Original ODE schneller ist als die Instrumentalisierte ODE ohne Flush lässt sich daraus ableiten, dass möglicherweise die XML Datenbankfelder nicht so effizient sind wie die BLOB Felder der DB2 oder das Durchschreiben der Daten in ein XML Feld mehr Zeit benötigt.

Aus diesen Gründen stellen wir die Messergebnisse der Zuweisungen (Assignment- und ExpressionEvaluation-Pushdown) relativ zur Instrumentalisierten ODE (100%) dar. Als Instanzlaufzeit verwenden wir grundsätzlich die Berechnete und geben sie relativ zur Original ODE (100%) an. Ebenfalls relativ zur Original ODE stellen wir die Messungen der Webservice-Aufrufe (INVOKE) und die Messungen zum ExpressionEvaluation-Pushdown für Bedingungen (IF) dar, da in diesen Aktivitäten kein Flush im Prototyp erfolgt.

7.3.2. Zuweisungen

Im Folgenden werden wir die Ergebnisse der sequentiellen Messungen zur Zuweisung, zuerst für IBM DB2 und anschließend für PostgreSQL vorstellen. Die Messwerte zu den parallelen Messungen entsprechen weitestgehend denen der Sequentiellen, da wir aus den parallelen Messungen zu den Zuweisungen keine weiteren Schlüsse ziehen konnten, stellen

wir sie nicht vor. Als ASSIGN-Zeit bezeichnen wir die Zeit, die für die Ausführung der Aktivität ASSIGN benötigt wird. Die einfachen und komplexen XPath-Ausdrücke können der Tabelle 7.2 entnommen werden, sie werden jeweils auf ein vergleichbares XML Dokument, wie in Anhang C.1 (Seite 149) angegeben, ausgeführt.

Art des Ausdrucks	XPath-Ausdruck
ohne XPath	-
einfach	*$var*/aln/seq[position()=*erstes*,*mittleres*,*letztes*]
komplex	concat(*$var1*/aln/seq[position()=1]/aa, *$var2*/aln/seq[position()=last()]/aa)

Tabelle 7.2.: XPath-Ausdrücke verschiedener Komplexität für die Messung der Zuweisung (ASSIGN).

7.3.2.1. IBM DB2

In Abb. 7.3 ist die relative ASSIGN-Zeit, also die benötigte Zeit für die Aktivität ASSIGN, für eine Zuweisung ohne XPath-Ausdruck über die verschiedenen Datengrößen angegeben. Gut zu erkennen ist, dass der asynchrone XPath-Pushdown schneller als der synchrone XPath-Pushdown und die Instrumentalisierte Version ist. Der synchrone XPath-Pushdown und die Instrumentalisierte Version enthalten in den ASSIGN-Zeiten jeweils auch die Zeit, die für das Durchschreiben der Variablen nötig ist (*Flush*). Diese *Flush*-Zeit ist in der ASSIGN-Zeit der asynchronen Variante nicht enthalten, da hier durch die Zuweisung innerhalb der DB kein *Flush* nötig ist.

Im direkten Vergleich mit Original ODE (siehe Kapitel 7.3.1) erfolgt die Zuweisung mit dem asynchronem Pushdown für 100kb deutlich langsamer während sie für 500kb nahezu identisch ist und für 4MB wieder etwas langsamer wird. In Abb. 7.4 wird die berechnete Laufzeit dargestellt, Original ODE ist je nach Datengröße 30-90% schneller. Interessant ist, dass im Vergleich zu der ASSIGN-Zeit in Abb. 7.3 die Laufzeit für 100kb im Vergleich zu Original ODE garnicht so schlecht ist, wie die reine ASSIGN-Zeit vermuten lässt. Hingegen werden die Instanzlaufzeiten für 500kb und 4MB wieder schlechter obwohl deren ASSIGN-Zeiten relativ gesehen besser sind als für die 100kb Variante. Wir kennen den genauen Grund hierfür nicht, vermuten jedoch, dass dies ebenfalls mit der DB-Kommunikation zu tun hat.

Vergleicht man den asynchronen und synchronen XPath-Pushdown für die *einfachen XPath-Selektionen* ist die ASSIGN-Zeit für den synchronen XPath-Pushdown etwas besser (siehe Abb. 7.5), was sich durch die Einfachheit des SQL-Ausdrucks für den synchronen Fall im vgl. zum komplexeren SQL-Ausdruck im asychnronen Fall erklären lasst (siehe Kapitel 6.1.2.1, Seite 87). Allerdings sind die Instanzlaufzeiten nahezu identisch (siehe Abb. 7.6).

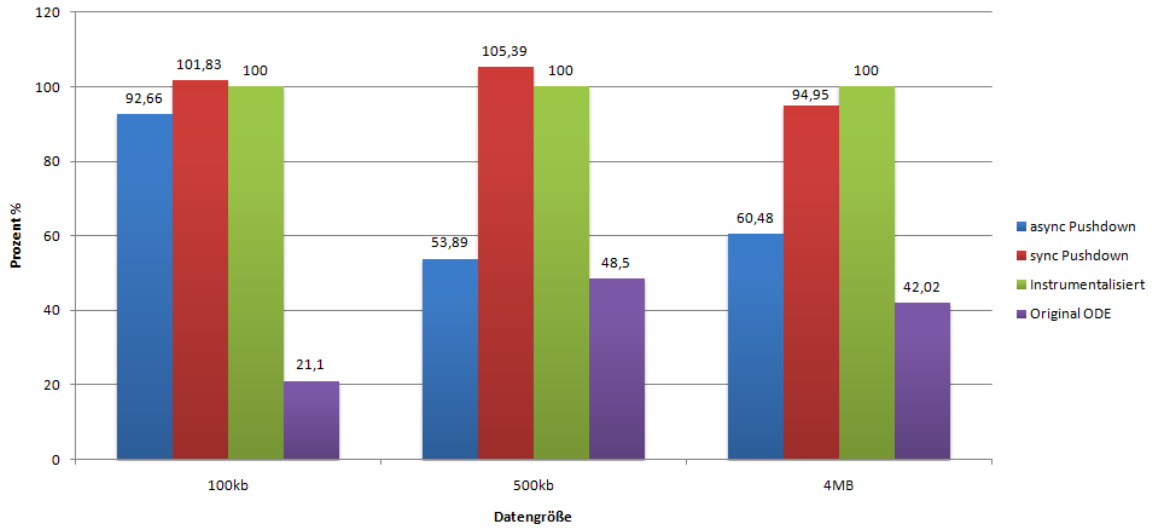


Abbildung 7.3.: Relative ASSIGN-Zeit über Datengröße für Zuweisungen ohne XPath-Ausdruck (DB2).

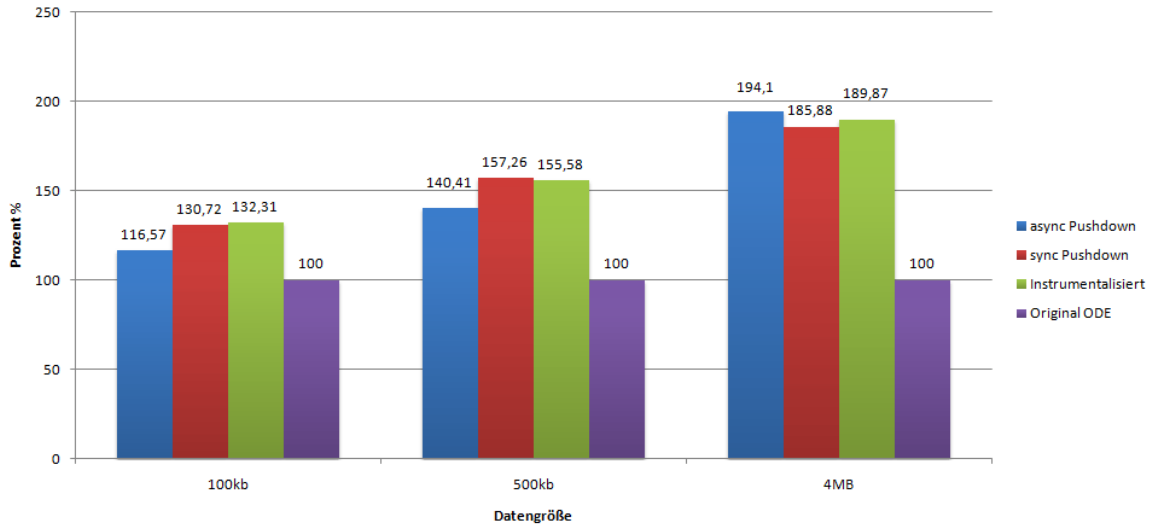


Abbildung 7.4.: Relative Laufzeit über Datengröße für Zuweisungen ohne XPath-Ausdruck (DB2).

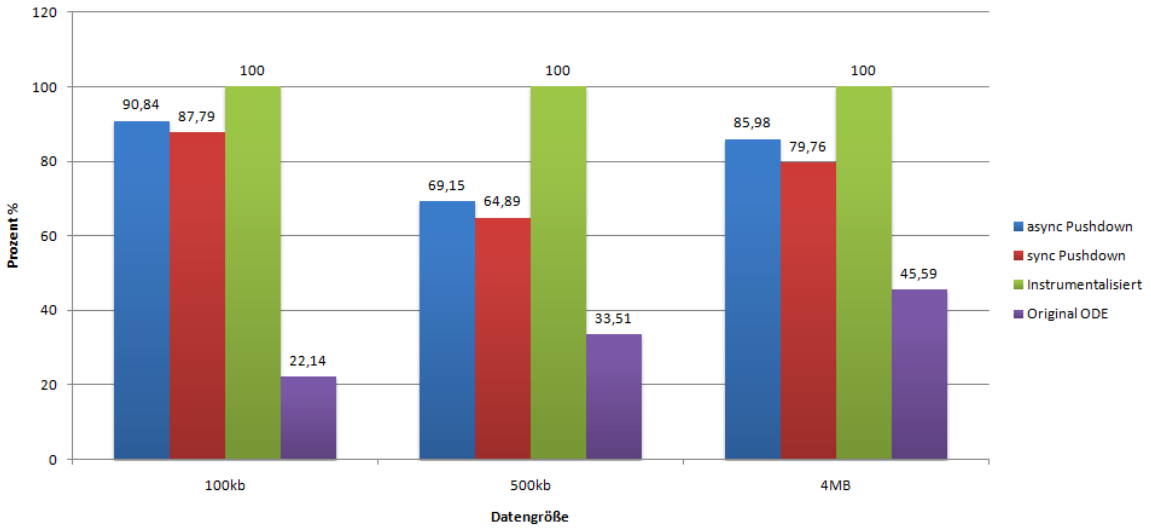


Abbildung 7.5.: Relative ASSIGN-Zeit über Datengröße für Zuweisungen einfacher XPath-Selektionen (DB2).

Vergleicht man den asynchronen und synchronen Pushdown mit Original ODE bleiben zwar die Tendenzen für die ASSIGN-Zeit und Instanzlaufzeiten bestehen, die beiden Varianten des Prototyps sind aber immer langsamer als die Original ODE.

Im Gegensatz dazu, ist die ASSIGN-Zeit und die Laufzeit des asynchronen XPath-Pushdown für komplexe Ausdrücke deutlich besser als die des Synchronen. Für größere Datenmengen zeigt der synchrone XPath-Pushdown sogar eine Verschlechterung im Vergleich zur Instrumentalisierten ODE. Hingegen kann sich der asynchrone XPath-Pushdown erstmalig für das 500kb XML Dokument gegen Original ODE durchsetzen und zeigt eine schnellere Ausführung (siehe Abb. 7.7 und 7.8). Die ASSIGN-Zeit des asynchronen Pushdown für 100kb ist etwas langsamer und die 4MB Variante geringfügig langsamer im Vergleich zur Original ODE dies spiegelt sich auch in den Instanzlaufzeiten wieder. Somit zeigt die asynchrone Variante des Prototyps insbesondere für komplexe XPath-Ausdrücke ein besseres Verhalten als bei einfachen Ausdrücken.

Mittelt man die ASSIGN-Zeiten der verschiedenen Zuweisungen, erhält man die kombinierte ASSIGN-Zeit in Abb. 7.9. Interessant ist der Vorteil der synchronen Ausführung für kleine Datenmengen, der sich bei größere Datenmengen in einen Nachteil umwandelt. Hingegen bleibt der asynchrone XPath-Pushdown immer besser als die Instrumentalisierte ODE. Insgesamt bleiben die Instanzlaufzeiten des Prototyps hinter der Original Version zurück, wobei sie bei der mittleren Datengröße von 500kb sehr nahe beieinander liegen.

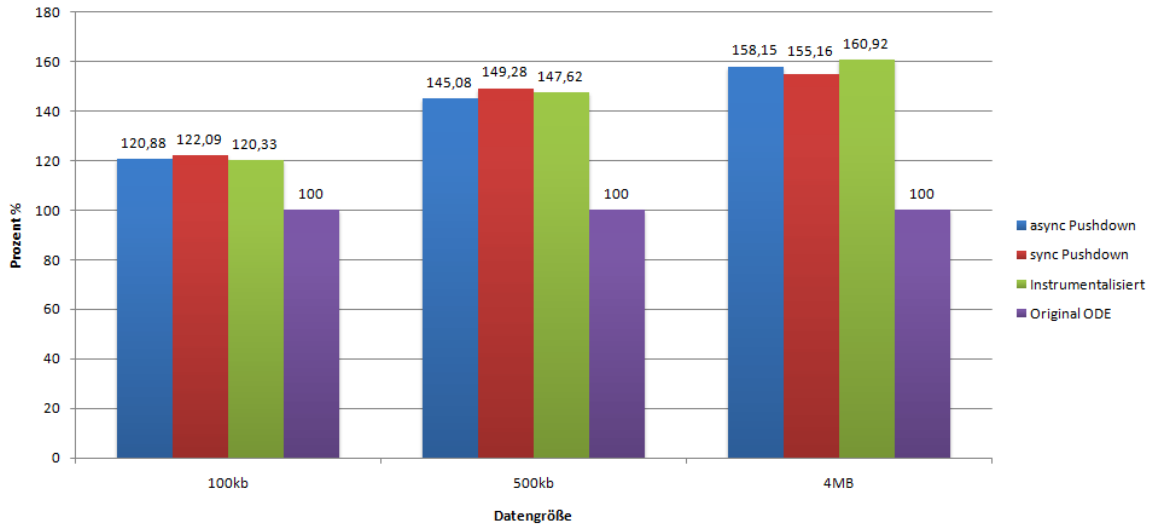


Abbildung 7.6.: Relative Laufzeit über Datengröße für Zuweisungen einfacher XPath-Selektionen (DB2).

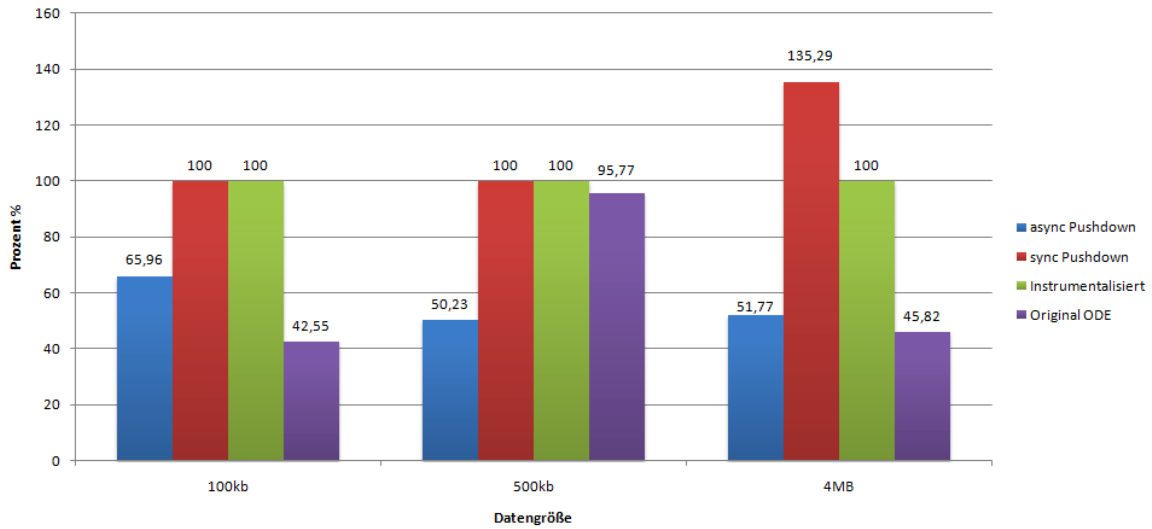


Abbildung 7.7.: Relative ASSIGN-Zeit über Datengröße für Zuweisungen komplexer XPath-Ausdrücke (DB2).

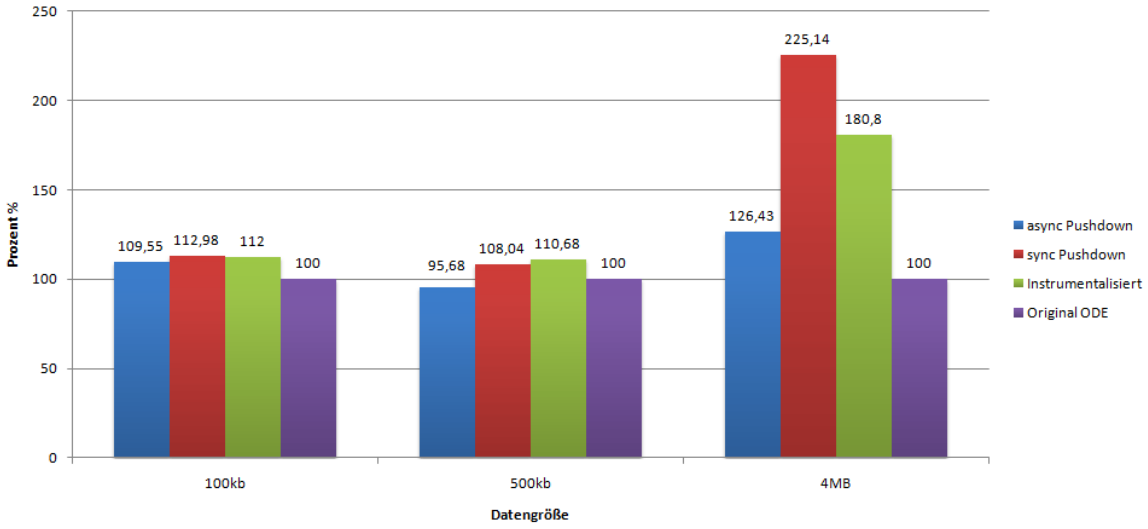


Abbildung 7.8.: Relative Laufzeit über Datengröße für Zuweisungen komplexer XPath-Ausdrücke (DB2).

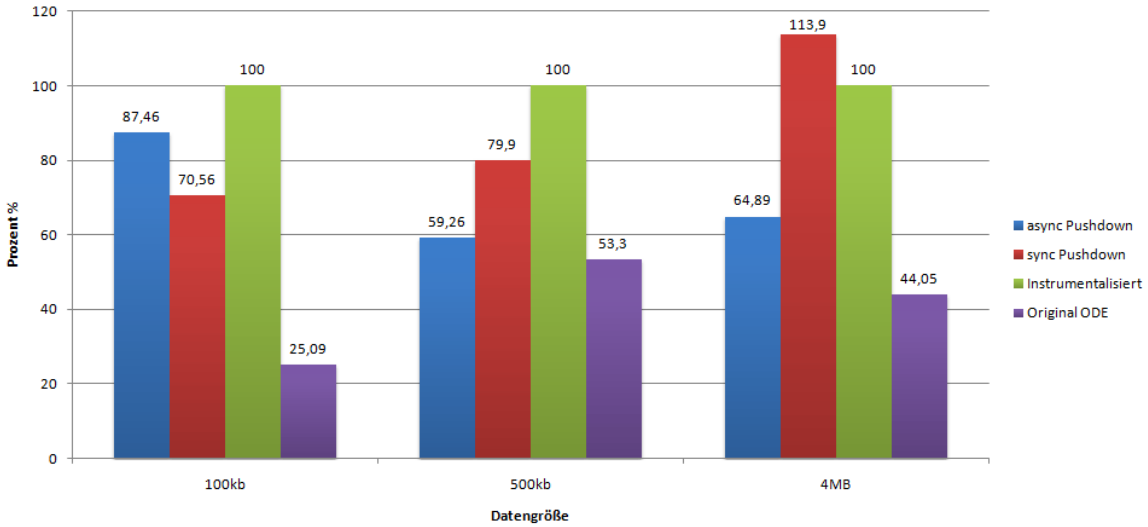


Abbildung 7.9.: Relative kombinierte ASSIGN-Zeit über Datengröße für Zuweisungen (DB2).

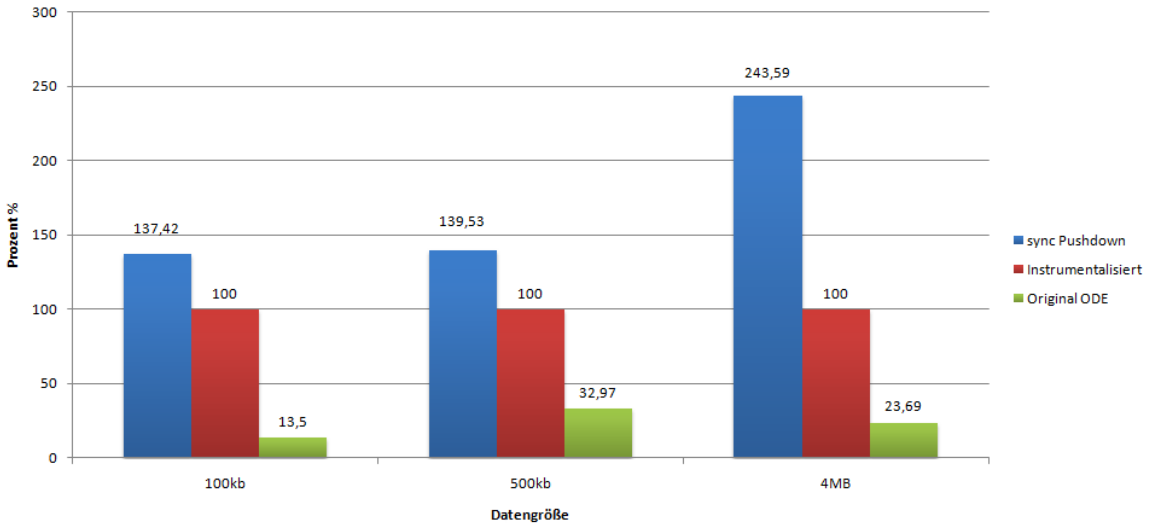


Abbildung 7.10.: Relative ASSIGN-Zeit über Datengröße für Zuweisungen ohne XPath-Ausdruck (PostgreSQL).

7.3.2.2. PostgreSQL

Im Vergleich zu IBM DB2 können wir mit PostgreSQL nur einen Bruchteil der Pushdown-Konzepte umsetzen (siehe Kapitel 6.2.2.1). Aus diesem Grund ist es nur möglich den synchronen XPath-Pushdown für Zuweisungen, ohne und für einfache XPath-Selektionen, zu evaluieren. Komplexe XPath-Ausdrücke und damit auch die Evaluierung von Bedingungen (ExpressionEvaluation/Condition-Pushdown) sowie der asynchrone XPath-Pushdown können mit dem derzeitigen Stand von PostgreSQL nicht umgesetzt werden.

Betrachtet man die ASSIGN-Zeit ohne XPath in Abb. 7.10, erkennt man generell eine schlechtere Ausführungszeit des synchronen XPath-Pushdowns zum Referenzsystem. Lediglich für die ASSIGN-Zeit bei einfachen XPath-Selektionen (Abb. 7.11) und den Testfall mit dem 100kb Dokument, lässt sich eine leichte Verbesserung feststellen. Im Vergleich zur Original ODE ist der Prototyp sowohl bei der ASSIGN-Zeit als auch bei der Laufzeit deutlich langsamer. Betrachtet man die kombinierte Instanzlaufzeit (Mittel aus ohne und einfachen XPath-Selektionen) im Vergleich zur Instrumentalisierten ODE zeigt der synchrone XPath-Pushdown für PostgreSQL eine generell schlechtere Performanz (siehe Abb. 7.12). Das Ergebnis passt zu der Beobachtung, die wir ebenfalls bei der DB2 gemacht haben: Eine Verschlechterung des synchronen XPath-Pushdowns bei zunehmenden Datenmengen. Außerdem scheint die *xpath*-Funktion von PostgreSQL noch optimierungsbedürftig zu sein, da sie insgesamt schlechter abschneidet als die pureXML-Technologie der DB2.

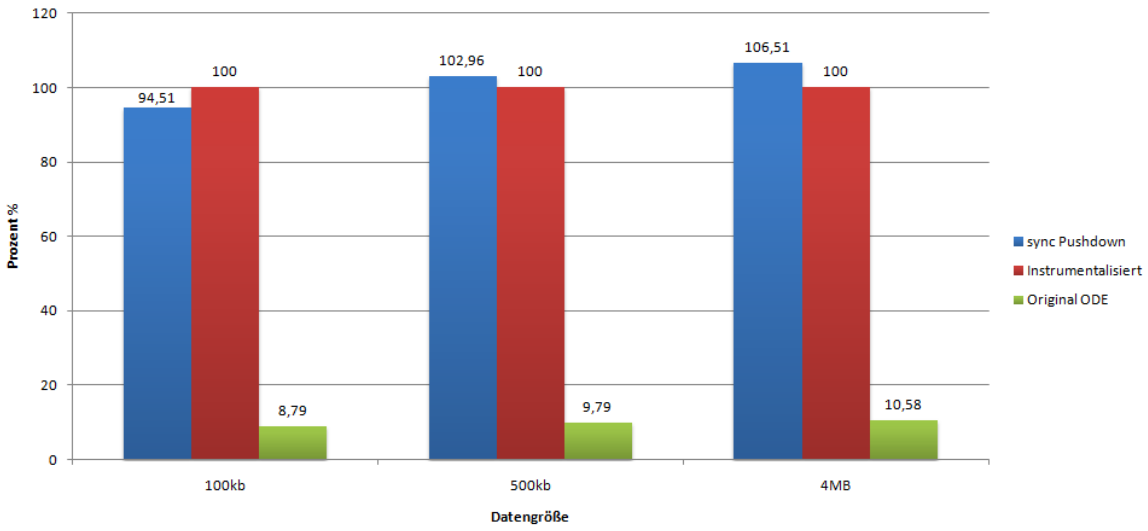


Abbildung 7.11.: Relative ASSIGN-Zeit über Datengröße für Zuweisungen einfacher XPath-Selektionen (PostgreSQL).

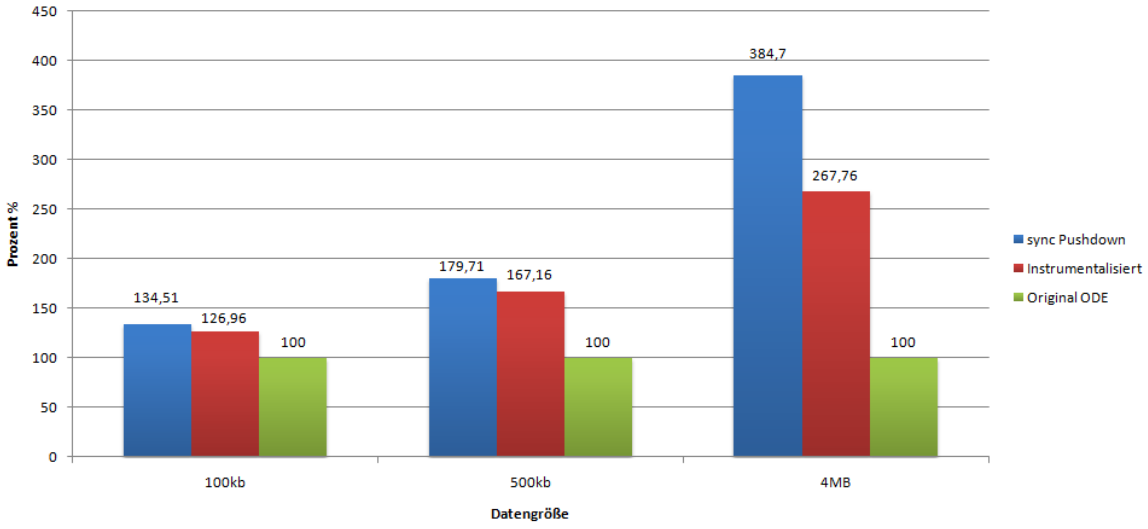


Abbildung 7.12.: Relative kombinierte Laufzeit über Datengröße für Zuweisungen (PostgreSQL).

7.3.3. Bedingungen (ExpressionEvaluation-Pushdown)

Wir stellen nun die Ergebnisse der sequentiellen und parallelen Messungen für den ExpressionEvaluation-Pushdown, anhand der Stellvertreter-Aktivität *IF* und dem DBS IBM DB2 vor. Wie in Kapitel 7.1 beschrieben, testen wir jeweils für einfache und einen komplexen XPath-Ausdruck, diese beziehen sich ebenfalls auf das Beispiel XML Dokument aus Anhang C.1 (Seite 149), welches in der Datengröße variiert wird. Die XPath-Ausrücke sind der Tabelle 7.3 zu entnehmen. Als IF-Zeit bezeichnen wir die benötigte Zeit für die Auswertung der XPath-Ausdrücke und weicht somit minimal von der benötigten Zeit der gesamten IF-Aktivität ab.

Art des Ausdrucks	XPath-Ausdruck
einfach	count(*$var*/aln/seq[position()=*erstes*,*mittleres*,*letztes*]/annotation/*) > 28
komplex	count(*$var1*/aln/seq[position()=1]/annotation/*) - count(*$var2*/aln/seq[position()=last()]/annotation/*) > 0

Tabelle 7.3.: XPath-Ausdrücke verschiedener Komplexität für die Messung der Bedingungsauswertung (IF).

Betrachten wir zunächst die Auswertung der einfachen XPath-Ausdrücke. In Abb. 7.13 ist zu sehen, dass die IF-Zeit des synchronen XPath-Pushdown immer eine schnellere Ausführung erlaubt als mit der Instrumentalisierten ODE. Allerdings ist nur für das 500kb XML Dokument der synchrone XPath-Pushdown schneller als die Original ODE Version. Für die 4MB Variante ist die IF-Zeit nur geringfügig langsamer als gegenüber der Original ODE. Vergleichen wir die Laufzeit einer Instanz in Abb. 7.14 verhalten sich alle Versionen ungefähr gleich schnell, außer für den Testfall mit 4MB, hier ist die Original Version schneller. Dies lässt sich möglicherweise auf die im WF verwendeten ASSIGN-Aktivitäten zurückführen (siehe Abb. 7.1b), um einer BPEL Variable das entsprechende XML Dokument zuzuweisen wobei ein *Flush* durchgeführt werden muss.

Betrachten wir die IF-Zeit für die Auswertung komplexer XPath-Ausdrücke, ist der synchrone XPath-Pushdown allen anderen Varianten bei jeder Datengröße überlegen (Abb. 7.15). Insbesondere bei 500kb ist der synchrone XPath-Pushdown um einen Faktor 4 schneller als Original ODE. Die Instrumentalisierte Version ist hingegen immer langsamer als die Auswertung der Original Version. Die relative Instanzlaufzeit der komplexen XPath-Ausdrücke entspricht, mit der gleichen Begründung, der der einfachen XPath-Ausdrücke, weshalb wir kein separates Schaubild angeben.

Betrachtet man die parallele Ausführung von jeweils 100 Instanzen der Testfälle, ergibt sich eine interessante Verschiebung zugunsten des synchronen XPath-Pushdowns für die IF-Zeit (siehe Abb. 7.16). Insbesondere ist die Auswertung eines komplexen XPath-Ausdrucks

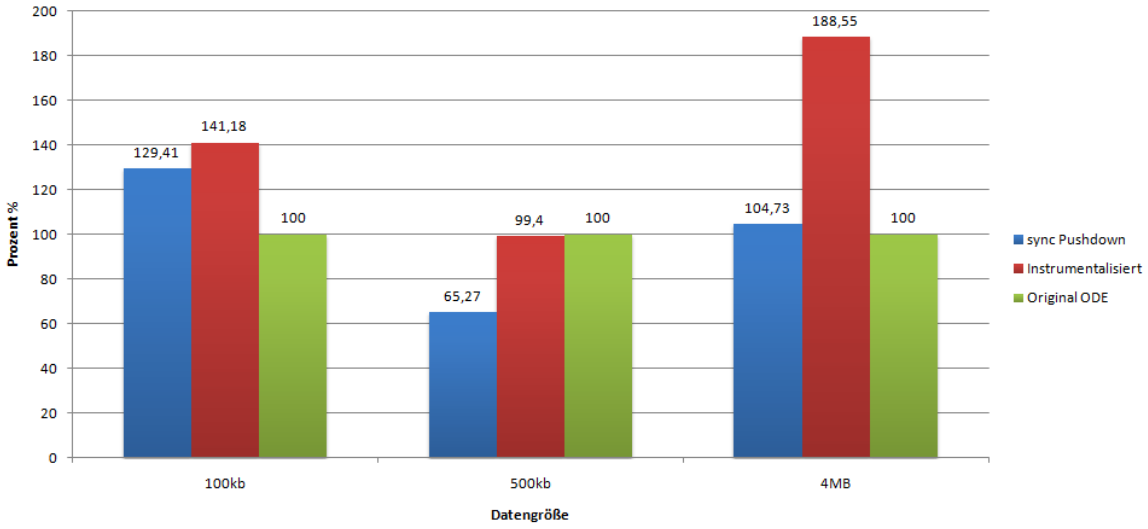


Abbildung 7.13.: Relative IF-Zeit über Datengröße für die Auswertung einfacher XPath-Ausdrücke (DB2).

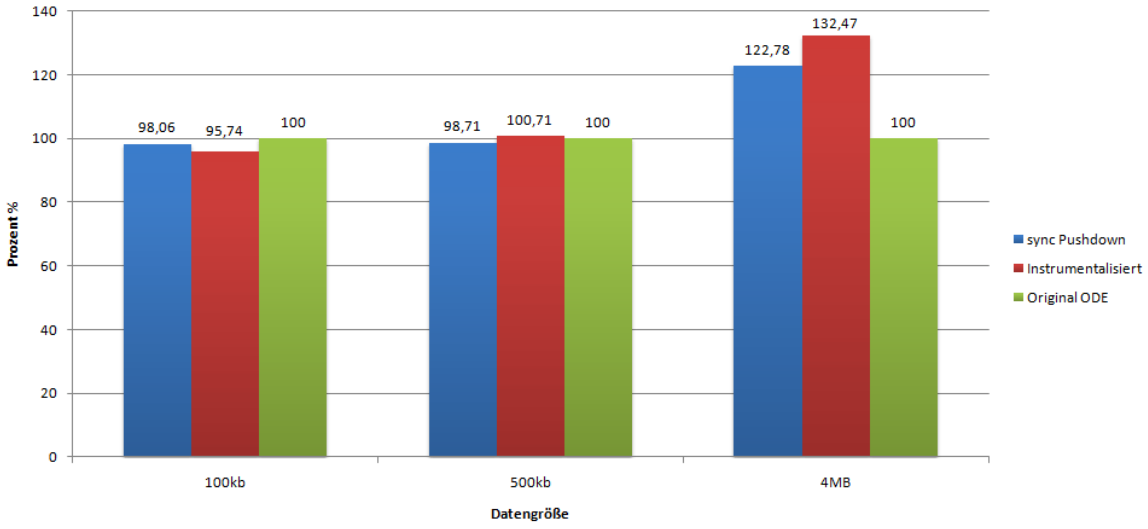


Abbildung 7.14.: Relative Laufzeit über Datengröße für die Auswertung einfacher XPath-Ausdrücke (DB2).

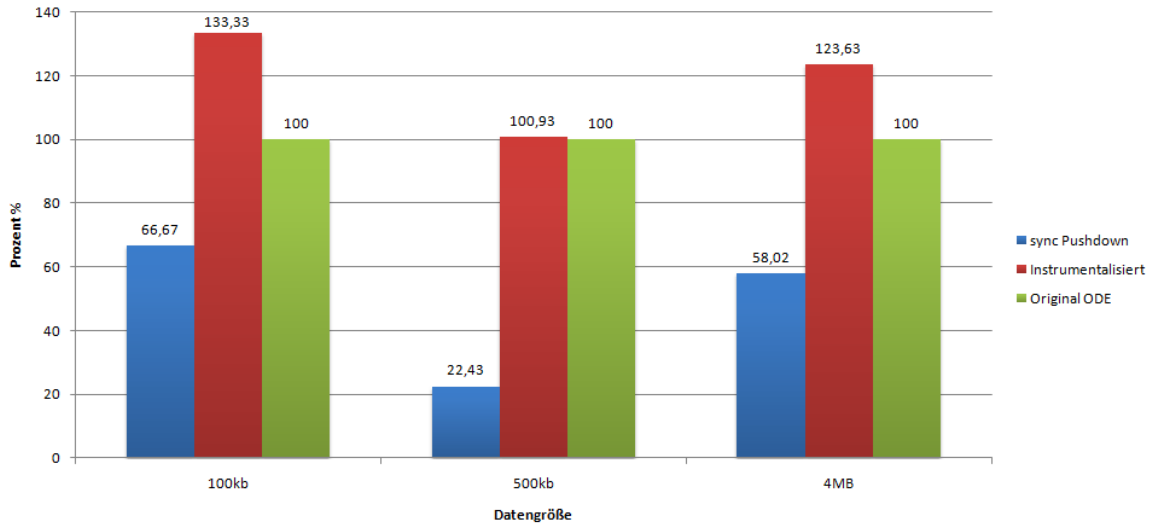


Abbildung 7.15.: Relative IF-Zeit über Datengröße für die Auswertung komplexer XPath-Ausdrücke (DB2).

auf das 4MB Dokument nun relativ am schnellsten, im Gegensatz zu den sequentiellen Messungen. Möglicherweise kommt dies durch ein effizienteres Cache-Verhalten der DB2 bei nebenläufigen Transaktionen zu Stande. Die Gesamtlaufzeiten dieser parallelen Fälle nähern sich für den synchronen XPath-Pushdown und komplexe XPath-Ausdrücke der Original ODE Version an. Allerdings ist die 4MB Variante für einfache XPath-Ausdrücke deutlich abgeschlagen gegenüber der Originalversion (siehe Abb. 7.17). Auch hier vermuten wir, dass dies durch die Effekte aus den ASSIGN-Aktivitäten, die im Testworkflow enthalten sind, herrührt. Zudem sollte erwähnt werden, dass sowohl für den Prototyp, als auch für Original ODE für die 4MB Variante keine 100 Instanzen bei paralleler Ausführung vollständig durchlaufen werden. Die Stabilität der WF-Engine lässt also bei dieser Datengröße nach. Allerdings konnten mit der Original ODE ca. zehn Instanzen mehr parallel ausgeführt werden als mit unserem Prototyp. Die Messungen wurden entsprechend für 100 Ausführungen abgeschätzt, indem die Gesamtlaufzeit durch die Anzahl der durchlaufenden Instanzen geteilt und anschließend mit 100 multipliziert wurde. Dies spiegelt nicht zwangsläufig die tatsächliche Gesamtlaufzeit für 100 parallele Instanzen wieder.

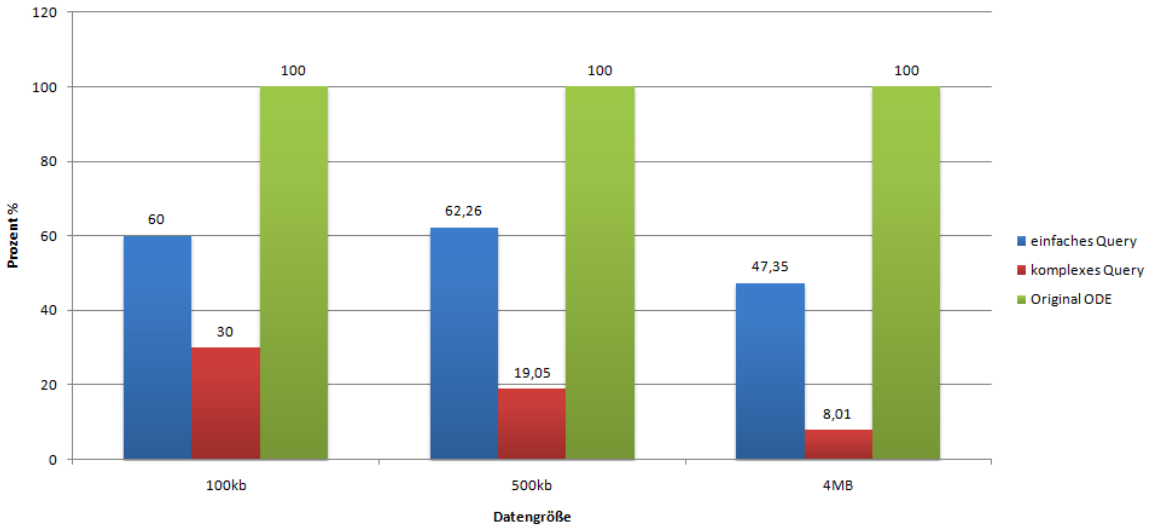


Abbildung 7.16.: Relative IF-Zeit über Datengröße für die parallele Auswertung einfacher und komplexer XPath-Ausdrücke (DB2).

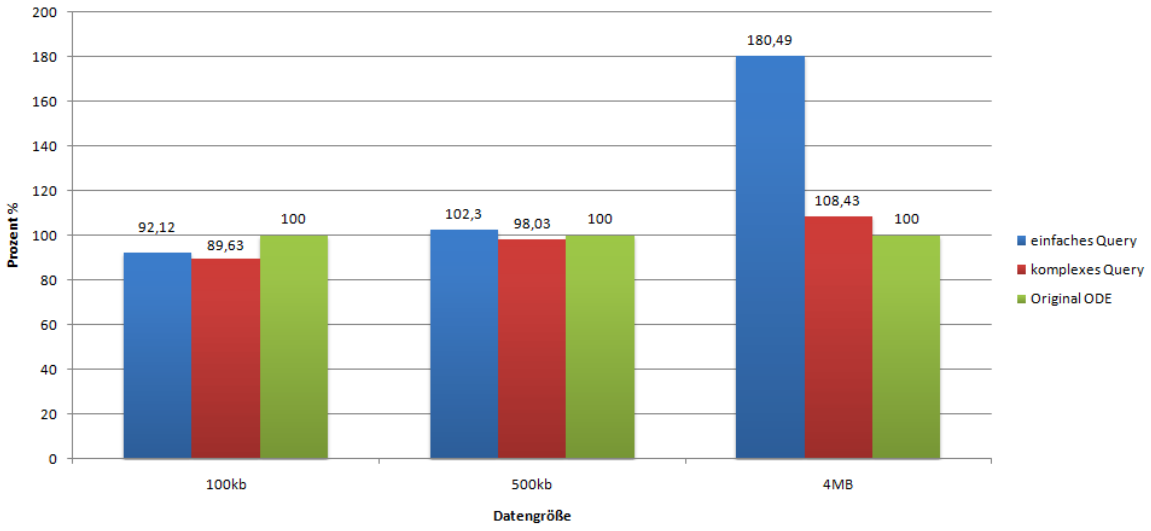


Abbildung 7.17.: Relative Gesamtlaufzeit über Datengröße für die parallele Auswertung einfacher und komplexer XPath-Ausdrücke (DB2).

7.3.4. INVOKE (Webservice-Pushdown)

In diesem Abschnitt stellen wir die Ergebnisse der Messungen zum Webservice-Pushdown vor. Der Webservice-Pushdown konnte nur für das DBS IBM DB2 umgesetzt werden (siehe Kapitel 6.1.2.1, Seite 87). Eine parallele Ausführung der Testfälle ist sowohl für Original ODE, als auch für den Prototyp nicht möglich, da Fehler beim Senden und Empfangen der SOAP Nachrichten auftreten und dabei die weitere Verarbeitung aller laufenden Instanzen in der WF-Engine unterbrochen wird. Aufgerufen wird die WSDL-Operation *getSFamilyAlignment* des Webservice *DWARF_ACCESS*, diese Operation wird ebenfalls im Anwendungsfall verwendet (siehe Kapitel 2.5.1.1, Seite 39). Die Messungen zum Anwendungsfall werden im nächsten Abschnitt vorgestellt.

Betrachten wir die INVOKE-Zeit, also die Zeit, die für die Aktivität INVOKE benötigt wird, sehen wir, dass die Zeiten für den WS-Pushdown im Vergleich zur Instrumentalisierten und Originalen ODE Version sehr dicht beieinander liegen. Die maximale Abweichung beträgt nur rund 5% (siehe Abb. 7.18). Der Webservice-Aufruf der Instrumentalisierten Version für das 4MB XML Dokument schlug jedesmal fehl, wir konnten den Fehler noch nicht lokalisieren. Für größere Dokumente können wir einen kleinen Vorteil des WS-Pushdown gegenüber der Original Version feststellen. Das Schaubild für die Laufzeiten einer Instanz (Abb. 7.19) korrelliert weitgehend mit den INVOKE-Zeiten. Lediglich die leicht schnellere Ausführung der Instrumentalisierten gegenüber der Original Version für 500kb ist nicht nachvollziehbar, dies könnte aber an Systemeinflüssen (Kontextwechsel der CPU, Netzwerklatenz etc.) während der Ausführung der Testfälle liegen, da die Resultate hier sehr dicht zusammenliegen.

Auch wenn der WS-Pushdown nur eine minimale Verbesserungen zeigt, ist er für die strikte Trennung von Daten- und Prozessverwaltung für das Architekturmodell Hybrides WfMS, welches wir in Kapitel 8 vorstellen werden, essentiell.

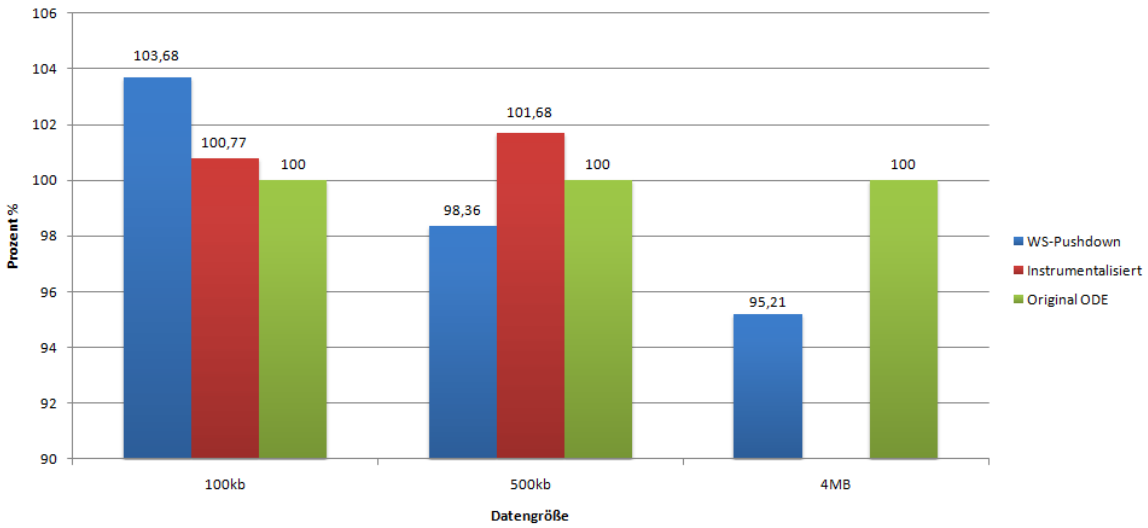


Abbildung 7.18.: Relative INVOKE-Zeit über Datengröße für den Aufruf einer WS-Operation (DB2).

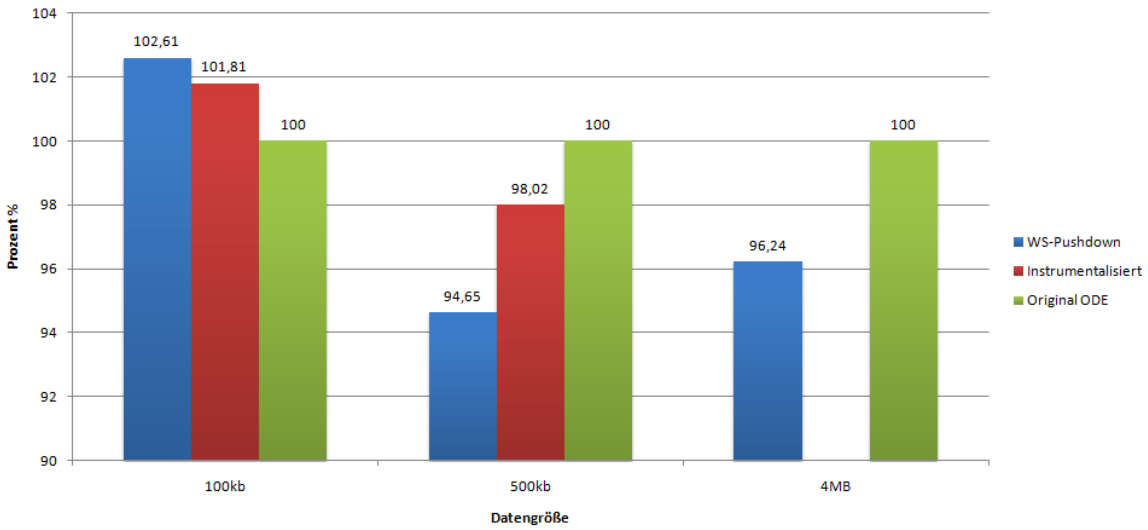


Abbildung 7.19.: Relative Instanzlaufzeit über Datengröße für den Aufruf einer WS-Operation (DB2).

7.3.5. Anwendungsfall (Simulationsworkflow)

Wir werden nun die Ergebnisse zu den Messungen des Anwendungsfalls vorstellen, dessen Funktionsweise wird in Kapitel 2.5.1.1 (Seite 39) beschrieben, der zugehörige in BPEL definierte Prozess ist im Anhang C.2 (Seite 151) und auf der DVD unter *[DVD]/Evaluation/ErgebnisseUndTestfälle/UseCase/ODEDwarfUseCase* zu finden. Der Anwendungsfall ist für die Bioinformatik relevant und verarbeitet Daten iterativ innerhalb einer Schleife. Diese iterative Datenverarbeitung ist ein grundlegendes Schema vieler Simulationen, und es werden im Rahmen des Simulation Technology (SimTech) Projekts an der Universität Stuttgart weitere Algorithmen für Simulationen vorgestellt, die eine iterative Datenverarbeitung betreiben [HDO10].

Die Testfälle werden wieder für XML Dokumente, wie in Anhang C.1 (Seite 149), mit den Datengrößen 100kb, 500kb und 4MB ausgeführt. Da über die <*seq*>-Elemente iteriert wird, können wir diese Datengrößen auch als Anzahl Schleifendurchläufe angeben, diese sind jeweils 40, 199 und 697 Iterationen. Wir haben für die sequentiellen Messungen bei 40 Iterationen den Anwendungsfall 100 mal und bei 199 Iterationen 10 mal, sowie 3 mal für 697 Iterationen hintereinander ausgeführt. Für die parallelen Messungen 50 (für 40 Iterationen), 10 (für 199 Iterationen) und 2 (für 697 Iterationen) parallel laufende Instanzen gemessen. Diese unterschiedlichen Werte wurden wegen der teils langen Laufzeiten des Anwendungsfalls, insbesondere bei vielen Interationen, und der Stabilität der Original ODE bei paralleler Ausführung verwendet. Wir werden zunächst die Resultate für das DBS IBM DB2 und anschließend für PostgreSQL vorstellen. Nur für IBM DB2 wurden parallele Tests durchgeführt, hierbei musste die INVOKE-Aktivität durch eine ASSIGN-Aktivität, in der das eigentliche WS-Resultat als Literalwert einer Variablen zugewiesen wird, ausgetauscht werden (siehe Kapitel 7.3.4).

7.3.5.1. IBM DB2

Zunächst werden wir die ASSIGN- und IF-Zeiten der sequentiellen Tests vorstellen, anschließend die Laufzeit einer Instanz und danach die Ergebnisse der parallelen Ausführung. Die ASSIGN-Zeit wird über alle gemessenen ASSIGN-Aktivitäten gemittelt, wir können somit keine Rückschlüsse auf die Performanz einzelner, im Anwendungsfall vorhandener, ASSIGN-Aktivitäten ziehen. Zu beachten ist, dass die Original ODE und die Instrumentalisiserte ODE Version im Vergleich zum Prototyp während der Verarbeitung des Anwendungsfalls mit 697 Schleifendurchläufen abbrechen. Aus diesem Grund können wir nur Vergleiche zwischen 40 und 199 Schleifendurchläufen präsentieren. Wir stellen jedoch fest, dass der Prototyp den Anwendungsfall auch für größere Datenmengen durchläuft und somit einen wesentlichen Stabilitätsvorteil aufweist.

Betrachten wir die ASSIGN-Zeit für den sequentiellen Fall, man erkennt den deutlichen Performanzvorteil des Prototyps gegenüber der Instrumentalisierten Version, der Prototyp

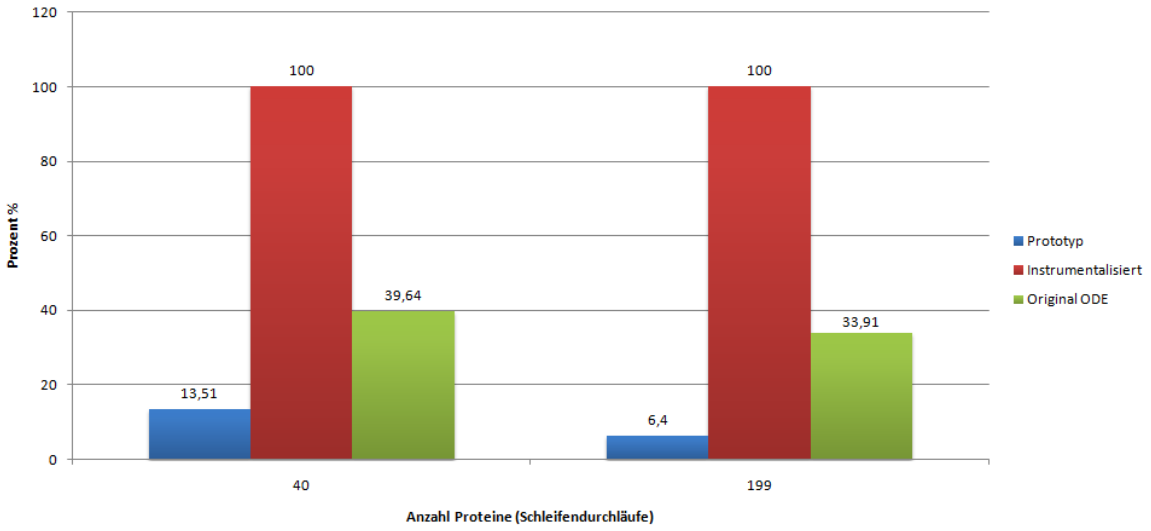


Abbildung 7.20.: Relative ASSIGN-Zeit über Anzahl Schleifendurchläufe für den Anwendungsfall (DB2).

ist 7-15 mal schneller (siehe Abb. 7.20). Sogar gegenüber Original ODE ist der Prototyp nun bei den ASSIGN-Zeiten um einen Faktor 3-5 performanter. Möglicherweise kommt der Effekt daher, da bis auf die Initialwerte der BPEL-Variablen bei der Ausführung des Anwendungsfalls keine *Flushs* erfolgen, insbesondere nicht innerhalb der Foreach-Schleife. Außerdem ist es möglich, dass die innerhalb der Schleife wiederkehrenden Zuweisungen und XPath-Auswertung vom DBS erkannt und durch einen *Query-Cache* schneller ausgeführt werden können.

Die Messungen zu den IF-Zeiten verwundern auf den ersten Blick, die Instrumentalisierte ODE ist bis zu 16 mal und der Prototyp bis zu einem Faktor von 2,5 langsamer als die Original ODE Version (siehe Abb. 7.21). Die Absolutwerte liegen jedoch im einstelligen Millisekunden Bereich, welches auch unsere Messeinheit ist. Zudem wird bei der Auswertung des zugehörigen XPath-Ausdrucks nur in einer ca. 1kb großen Zeichenkette nach einem regulären Ausdruck gesucht. Daher ist die Hauptspeicher-basierte Auswertung innerhalb Original ODE schneller, insbesondere da keine Kommunikation mit dem DBS stattfinden muss.

Bei Betrachtung der Instanzlaufzeit wird nun deutlich, dass der Prototyp eine schnellere Ausführung erlaubt. Für 40 Schleifendurchläufe ist er fast doppelt so schnell wie Original ODE und für 199 Iterationen sogar fast um einen Faktor 4 schneller[1] als die Original Version (Abb. 7.22). Die Instrumentalisierte ODE zeigt gleichförmig schlechtere Laufzeiten, was

[1]Dadurch verringert sich die mittlere Laufzeit, für eine Instanz mit 199 Iterationen auf dem Testsystem, von 7:15min auf 1:53min.

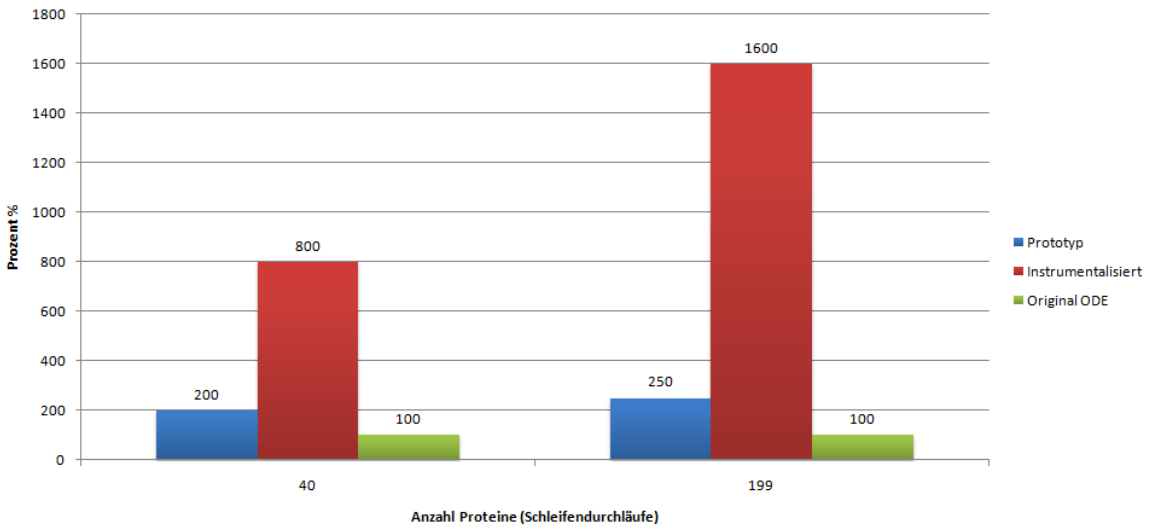


Abbildung 7.21.: Relative IF-Zeit über Anzahl Schleifendurchläufe für den Anwendungsfall (DB2).

unsere Vermutung aus Kapitel 7.3.1 bestärkt, dass der Zugriff auf XML Felder nicht so effizient ist, wie auf entsprechende BLOB Felder in der Original Version.

Während die Hauptspeichermessungen für die Einzeltests schwer zu interpretieren und nicht aussagekräftig genug waren, können wir aus den Hauptspeichermessungen des Anwendungsfalls einige interessante Schlussfolgerungen ziehen (Abb 7.23). Für den kleinen Anwendungsfall mit 40 Iterationen ist der Hauptspeicherverbrauch für alle drei Versionen fast gleich, bei 199 Iterationen erkennt man einen klaren Anstieg der Original ODE Version, um den Faktor 2-4 gegenüber Prototyp und Instrumentalisierter ODE, auf über 900MB. Die Instrumentalisierte Version benötigt in diesem Fall am wenigsten Hauptspeicher. Wir schließen daraus, dass wahrscheinlich das DB Middleware System Hibernate durch die erzwungene Persistenz in der Instrumentalisierten Version, die Objekte aus dem Hauptspeicher löschen kann, um sie bei Bedarf nachzuladen. Dieses Vorgehen ist in der Original Version nicht möglich, da sie lediglich vor der Schleifenausführung, genauer gesagt vor dem WS-Aufruf, ein Durchschreiben der Daten erzwingt und dann erst wieder beim Beenden der Instanz (siehe Kapitel 7.3.1). Der im Vergleich zur Instrumentalisierten Version erhöhte Hauptspeicher Verbrauch des Prototypen lässt darauf schließen, dass IBM DB2 für die Auswertung der XPath-Ausdrücke zusätzlichen Hauptspeicher vom System anfordet, dieser ist aber im Vergleich zur Original ODE Version immer noch geringer. Somit bleiben Ressourcen frei, die für andere Aufgaben bei der WF-Ausführung verwendet werden können, z.B. für Auditing Maßnahmen.

Zum Abschluss der Messungen für das DBS IBM DB2 betrachten wir noch die parallele Ausführung des Anwendungsfalls mit 40 Iterationen, für jeweils 10 parallel laufende Instan-

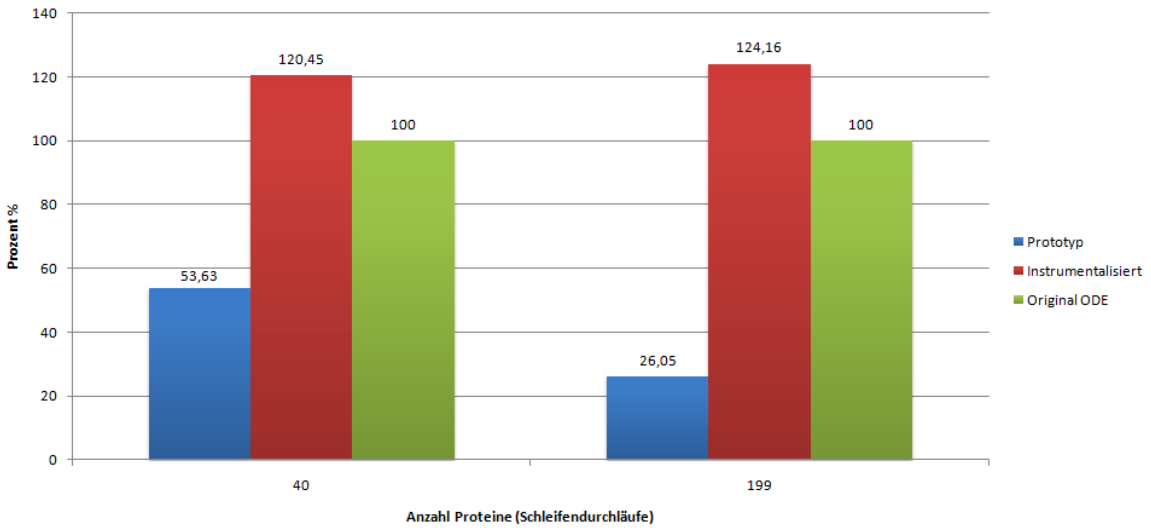


Abbildung 7.22.: Relative Laufzeit über Anzahl Schleifendurchläufe für den Anwendungsfall (DB2).

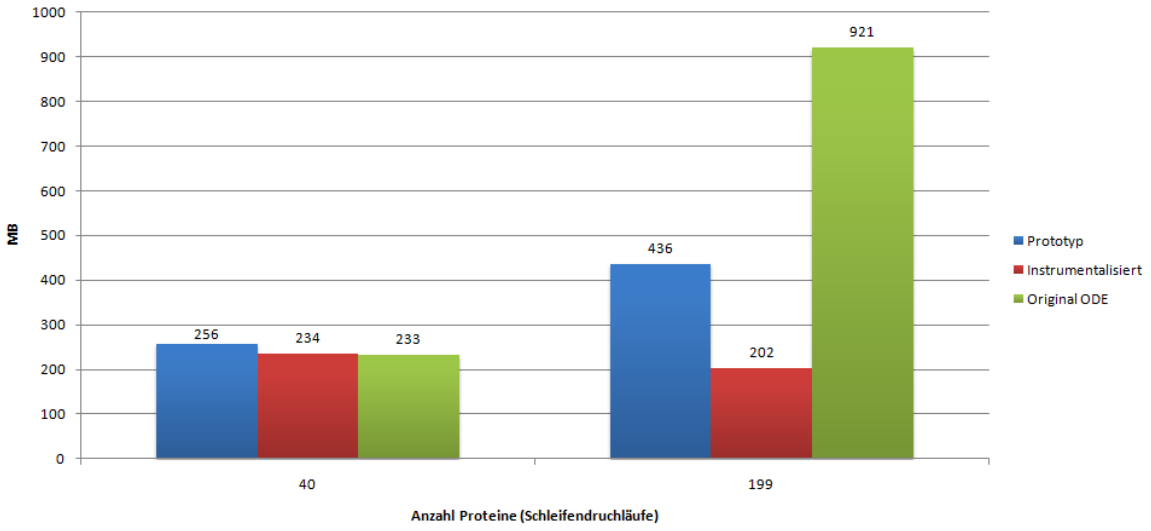


Abbildung 7.23.: Absoluter Hauptspeicherverbrauch über Anzahl Schleifendurchläufe für den Anwendungsfall (DB2).

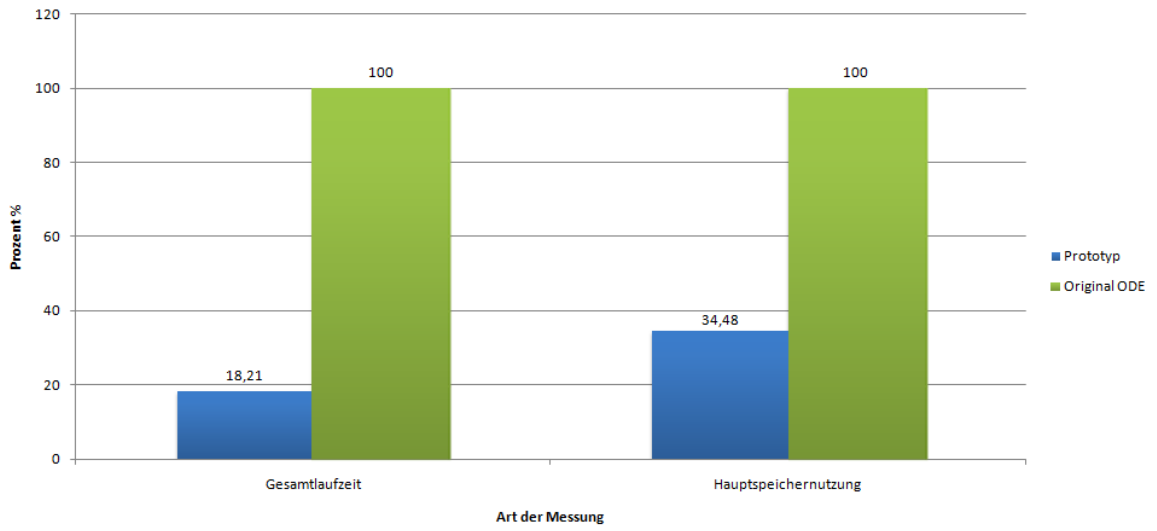


Abbildung 7.24.: Relative Laufzeit und relativer Hauptspeicherverbrauch bei paralleler Ausführung von 10 Instanzen des Anwendungsfalls mit 40 Iterationen (DB2).

zen (Abb 7.24). Man erkennt für die Gesamtlaufzeit als auch für die Hauptspeichernutzung einen deutlich Vorteil bei Verwendung des Prototyps im Gegensatz zur Original Version. Der Prototyp ist zudem stabiler, ausschließlich mit ihm ist es möglich 50 parallele Instanzen für 40 Iterationen, 10 parallele Instanzen für 199 Iterationen und 2 parallele Instanzen für 697 Iterationen auszuführen. Dabei ist insbesondere die parallele Ausführung der 697 Iterationen zu erwähnen, da die aufsummierte Gesamtlaufzeit beider Instanzen ziemlich genau der einer einzelnen sequentiellen Ausführung entspricht, was eine Durchsatzsteigerung von 100% bedeutet. Vergleicht man, jeweils für 40 Iterationen, die mittleren Instanzlaufzeiten der 10 parallel ausgeführten Instanzen mit den mittleren Instanzlaufzeiten der sequentiellen Ausführung des Prototyps und Original ODE, benötigt eine parallel ausgeführte Instanz des Prototyps im Mittel mehr als doppelt so lang wie die sequentielle Ausführung. Eine parallele Instanz der Original Version benötigt sogar die 10 fache Zeit der entsprechenden sequentiellen Ausführung. Allerdings ist die Gesamtzeit der Ausführung von 10 Instanzen im parallelen Fall durch die Nebenläufigkeit für den Prototyp ca. sechs mal und für Original ODE ca. doppelt so schnell wie die entsprechende sequentielle Gesamtlaufzeit (siehe Abb. 7.25). Somit kann bei der parallelen Ausführung der Prototyp gegenüber Original ODE sogar eine bis zu dreifach größere Durchsatzsteigerung erzielen.

7.3.5.2. PostgreSQL

Die Messwerte für den Anwendungsfall mit dem DBS PostgreSQL sind, durch die funktionalen Einschränkungen und die bereits schlechteren Ergebnisse im Vergleich zur Instrumen-

Abbildung 7.25.: Relative Gesamtlaufzeit der parallelen und sequentiellen Ausführung von 10 Instanzen des Anwendungsfalls mit 40 Iterationen (DB2).

talisierten Version für den Assignment-Pushdown (siehe Kapitel 7.3.2.2), nicht überraschend. Die ASSIGN-Zeiten des Prototyps sind nahezu mit der der Instrumentalisierten Version identisch. Dies ist darauf zurückzuführen, da lediglich die XPath-Selektionen anders behandelt werden. Die Original Version ist im Vergleich zu den beiden anderen Versionen deutlich schneller in der Ausführung (siehe Abb 7.26). Die relative Laufzeit einer Instanz korreliert entsprechend mit der ASSIGN-Zeit, Prototyp und Instrumentalisierte Version sind um einen Faktor 1,6 - 2,3 langsamer als die Original Version (siehe Abb 7.27). Der Prototyp und die Instrumentalisierte ODE verschlechtern sich bei 199 Iterationen gegenüber der Original ODE noch weiter als bei 40 Iterationen. Betrachtet man den Hauptspeicherverbrauch ergibt sich ein ähnliches Bild wie für die Auswertung mit dem DBS IBM DB2. Die Instrumentalisierte Version benötigt am wenigsten Hauptspeicher gefolgt vom Prototyp und Original ODE, zumindest für den Anwendungsfall mit 199 Iterationen (siehe Abb 7.28). Dies untermauert unsere Annahme aus Kapitel 7.3.5.1, dass durch die erzwungenen *Flushs* im Prototyp und der Instrumentalisierten ODE durch Hibernate Hauptspeicher freigegeben werden kann. Keine Version war in der Lage den Anwendungsfall mit 697 Iterationen zu durchlaufen, der Prototyp war sogar am instabilsten und durchlief schon für den Anwendungsfall mit 199 Iterationen nicht alle Instanzen.

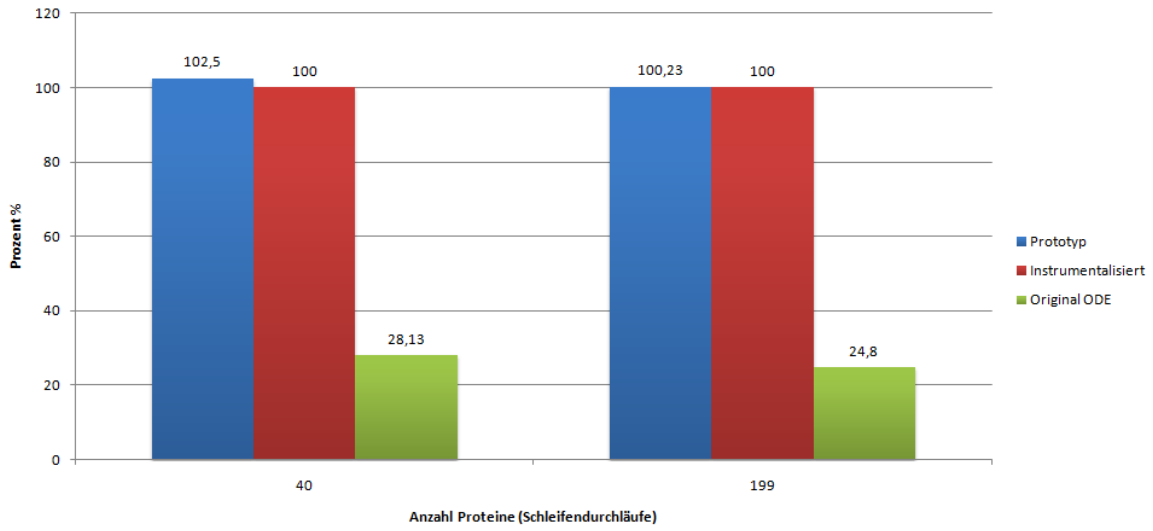


Abbildung 7.26.: Relative ASSIGN-Zeit über Anzahl Schleifendurchläufe für den Anwendungsfall (PostgreSQL).

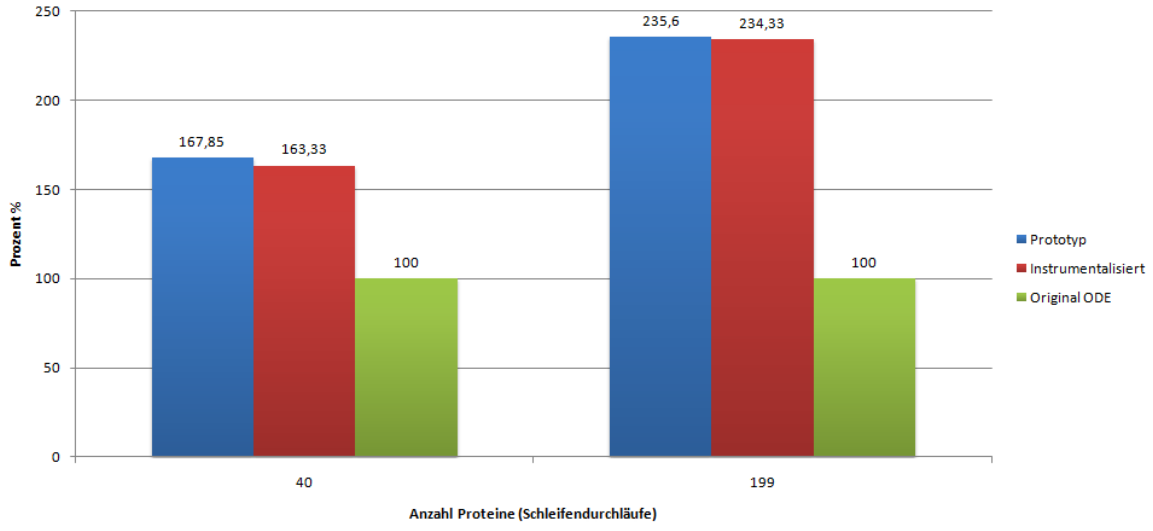


Abbildung 7.27.: Relative Laufzeit einer Instanz über Anzahl Schleifendurchläufe für den Anwendungsfall (PostgreSQL).

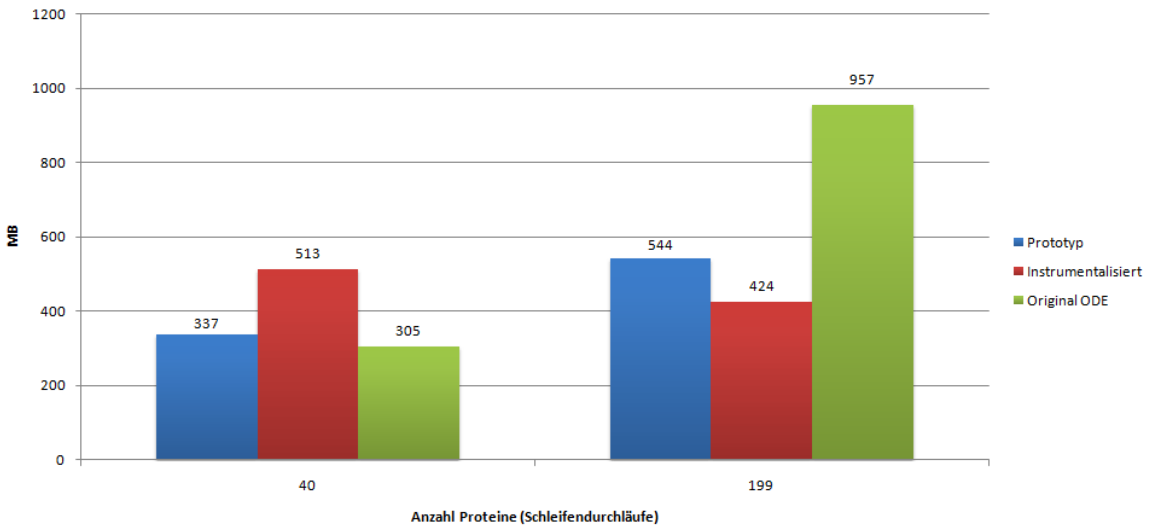


Abbildung 7.28.: Absoluter Hauptspeicherverbrauch über Anzahl Schleifendurchläufe für den Anwendungsfall (PostgreSQL).

7.4. Diskussion der Messergebnisse

Die Ergebnisse zeigen deutlich, dass die Funktionalitäten, die ein DBS anbietet, und deren effiziente Implementierung entscheidend sind, um die vorgestellten Konzepte aus Kapitel 4 erfolgreich (Durchführbarkeit sowie Performanzgewinn) umsetzen zu können. Der Prototyp mit dem DBS PostgreSQL zeigt leider durchweg eine schlechtere Performanz als die Original ODE Version, die mit PosgtreSQL arbeitet. Dies hat mehrere Gründe angefangen von den eingeschränkten XML Funktionalitäten, die nur einen Bruchteil der möglichen XPath-Pushdown Funktionen realisieren lassen und vor allem die Ausführung des interessanten asynchronen XPath-Pushdowns verhindern. Mit der DB2 war es möglich alle Funktionen des Prototyps umzusetzen, was für einige Fälle ein Performanzgewinn erbrachte.

Betrachten wir die Testfälle der einzelnen Aktivitäten, liegt der Prototyp bei der Instanzlaufzeit in fast allen Fällen hinter der Original ODE Version zurück. Wie schon in Kapitel 7.3.1 besprochen, vermuten wir hier eine aufwändigere Datenbankkommunikation in Zusammenspiel mit einer nicht so effizienten Implementierung der XML Felder im Vergleich zu den BLOB Feldern der jeweiligen DBSe. Zudem müssen alle Variableninhalte für den Prototyp bereits in der DB abgespeichert sein, das Erzwingen dieser Persistenz beeinträchtigt die Laufzeit des Prototypen. Die Persistenz wirkt sich jedoch positiv auf den Hauptspeicherverbrauch von Workflows mit Datenverarbeitung innerhalb von Schleifen aus (siehe Abb. 7.23) und könnte ggf. auch dem Original System mehr Stabilität verleihen. Wird die Persistenz der Daten erzwungen, ist der Prototyp mit dem DBS IBM DB2 jeweils performanter (Laufzeitvergleiche gegenüber der Instrumentalisierten Version). Bei den ASSIGN- und IF-Zeiten

lässt sich feststellen, dass der Prototyp bei komplexen XPath-Ausdrücke schneller wird und für die Datengröße 500kb oft die beste relative Zeit aufweist. Weitere Tests mit einem 50MB XML-Dokument ergaben, dass sich der Trend fortsetzt und die relativen Zeiten für große XML-Daten zunehmend schlechter werden. Die INVOKE-Zeit des Prototypen nimmt für zunehmende Datengrößen gegenüber der Original ODE ab und kann ab Datengrößen von 500kb grundsätzlich verwendet werden.

Das Zusammenspiel der Konzepte WS-Pushdown, Assignment-Pushdown (asynchroner XPath-Pushdown) und ExpressionEvaluation-Pushdown (synchroner XPath-Pushdown), wie sie im Anwendungsfall auftreten, ziehen eine bis zu 4-fache Steigerung der Ausführungsgeschwindigkeit sowie einen geringeren Hauptspeicherverbrauch nach sich. In diesem Fall ist auch die Datenbankkommunikation geringer, da im Wesentlichen nur Anweisungen und keine Inhalte an das DBS übergeben werden. Variableninhalte werden nur zur Initialisierung (BPEL Literale) an das DBS übergeben. Alle weiteren datenverarbeitenden Schritte finden direkt innerhalb des DBSs statt. Die Auswertung einer Bedingung liefert nur das Resultat (wahr oder falsch) und nicht die Variableninhalte zurück. Zudem können hier DB spezifische Optimierungen, z.B. ein *Query-Cache*, helfen die Ausführung zu beschleunigen. Insbesondere bei einer iterativen Verarbeitung von Daten werden innerhalb der Schleife gleichförmig strukturierte Anfragen an das DBS gestellt, die vom DB-Optimierer erkannt und so effizienter ausgeführt werden können. Die parallele Ausführung von Workflowinstanzen ist mit dem Prototyp stabiler, was wohl auf die Transaktions- und Mehrbenutzereigenschaften heutiger DBSe zurückzuführen ist. Die Persistenz der Daten ist durch den WS- und asynchronen XPath-Pushdown ohne weitere Maßnahmen gegeben wodurch prinzipiell eine Recovery nach einem Systemausfall gewährleistet wird.

Zusammenfassend ist zu sagen, dass die Wahl des DBSs entscheidend ist. Der vorgestellte Prototyp, der mit dem DBS IBM DB2 arbeitet, ist insbesondere für alle Arten von Workflows geeignet, in denen Daten innerhalb einer Schleife verarbeitet werden. Diese Verarbeitungsform ist ein wiederkehrendes Strukturelement vieler Simulationsworkflows ([HDO10], Anwendungsfall Kapitel 2.5.1.1, Seite 39) und daher von Interesse. Für ETL-Workflows (Kapitel 2.3.3.3, Seite 34) könnte sich der Prototyp möglicherweise ebenfalls eignen, da diese Workflows sehr ähnlich zu dem vorgestellten Anwendungsfall und oft noch datenintensiver sind. Dies müsste allerdings für bestimmte ETL-Muster und ETL-Operationen separat evaluiert werden. Für Business-Workflows (Kapitel 2.3.3.1, Seite 32) sowie für Simulationsworkflows mit geringem Datenaufkommen (Kapitel 2.3.3.2, Seite 33), die ähnlich zu Business-Workflows nur die Aufrufreihenfolge externe Programme orchestrieren, scheint der Prototyp im jetzigen Zustand nicht geeignet zu sein. Durch weitere Anpassungen, insbesondere an der DAO-Schicht und die Optimierung der DBS-Parameter oder der Verwendung eines Nativen-XML DBSs, könnte der Prototyp aber auch für diese Workflows zumindest eine gleich schnelle Ausführung erlauben.

7.4.1. Technische Limitierungen

Apache ODE sowie der Prototyp besitzen einige Limitierungen, die im Rahmen der Evaluation geändert oder berücksichtigt werden mussten. Zum Einen mussten hart kodierte *Timeout*-Werte für das Versenden und Empfangen von SOAP Nachrichten in beiden Versionen geändert werden, um die Testfälle ausführen zu können. Apache ODE verhält sich nach einer bestimmten Ausführungszeit einer Workflowinstanz nicht mehr ganz korrekt, z.B. wird eine vollständig erfolgreich ausgeführte Instanz nicht als *'ausgeführt'* gekennzeichnet und bleibt als *'aktiv'* markiert. Wie schon unter Kapitel 7.3.4 erwähnt, ist die parallele Ausführung von WS-Aufrufen auf die gleiche WSDL-Operation eines WS anscheinend nicht möglich, dies kann jedoch auch mit der verwendeten Kommunikationsinfrastruktur (in diesem Fall Axis2) zusammenhängen. Auch kann sowohl für den Prototyp als auch für Original Apache ODE bei Datenmengen im MB-Bereich die Anzahl parallel laufender Instanzen nicht beliebig hoch gewählt werden. Zudem kann beim gleichzeitigen Aufrufen mehrerer Instanzen des gleichen Prozesses es in beiden Versionen dazu kommen, dass eine gewisse Anzahl von Aufrufen nicht vom System verarbeitet wird und entsprechend weniger Instanzen auf der WF-Engine ausgeführt werden als angefordert. Es konnten auch keine WFs mit XML-Dokumenten die größer als 75MB waren compiliert werden, somit konnten keine Tests für sehr große Dokumente im Hundert-MB bzw. GB Bereich durchgeführt werden.

Der Prototyp mit dem DBS IBM DB2 kann standardmäßig nur WS-Aufrufe durchführen, bei denen die Antwortnachricht des WS nicht größer als 1MB ist. Dies kann durch eine veränderte Signatur der entsprechenden UDF angehoben werden[2] und wurde für die Ausführung des Anwendungsfalls mit 697 Iterationen auf 5MB erhöht. Zudem versteht *pureXML* nur einen bestimmten Dialekt von XPath-Ausdrücken, dies ist insbesondere bei der Selektion von einzelnen Elementen aufgefallen. Während Original ODE den XPath-Ausdruck *'$var/items/item[1]'* ohne Probleme auswertet und das erste <*item*>-Element des XML Dokuments in der Variable *var* zurückliefert, muss für die korrekte Bearbeitung innerhalb der DB2 der XPath-Ausdruck in *'$var/items/item[position()=1]'* umgewandelt werden. Dies wird möglicherweise in einer neuen Version der DB2 und einer XPath-Standard entsprechenden pureXML-Implementierung kein Thema mehr sein oder man könnte ggf. eine Natives XML DBS verwenden, das den XPath-Standard schon jetzt korrekt umsetzt.

[2]https://www.ibm.com/support/docview.wss?uid=swg1IZ46071

8. Konzeptionelle Erweiterungen

In diesem Kapitel werden wir, aus den Erfahrungen bei der Implementierung des Prototyps, eine Referenzarchitektur für solche Systeme vorstellen und diese in Relation zu den anderen Integrationsmöglichkeiten aus Kapitel 3.2 setzen. Im Abschnitt 8.2 werden wir weiterführende Arbeiten zur Erweiterung und zum Einsatz der Konzepte aus Kapitel 4 vorstellen.

8.1. Referenzarchitektur

Die Realisierbarkeit der Konzepte aus Kapitel 4 konnte durch den Prototypen als *Proof-of-Concept* nachgeweisen werden. Da dies ein experimenteller Ansatz ist und dieser deshalb nur prototypisch implementiert werden konnte, existiert natürlich Spielraum für eine sauberere Implementierung. Aus den Erfahrungen und Schwierigkeiten der Implementierung des Prototyps möchten wir nun in Anlehnung an Kapitel 3 eine konzeptionelle Referenzarchitektur und Implementierungsdetails vorstellen, die zu einem übersichtlicherem System und möglicherweise zu einer Verallgemeinerung der Anfragen an das DBS führen.

8.1.1. Referenzarchitektur für ein Pushdown WfMS

Die Referenzarchitektur besteht aus der *Runtime-*, *DAO-* und *Pushdown*-Schicht (siehe Abb. 8.1). Die *Pushdown*-Schicht ist für die Realisierung der Pushdown-Konzepte verantwortlich und kann von der *DAO*-Schicht sowie direkt von der *Runtime* aus aufgerufen und verwendet werden. Für die Referenzarchitektur ist es außerdem wichtig, dass das Typsystem des WfMSs über alle drei Schichten *Runtime*, *Pushdown* und *DAO* hinweg sichtbar ist (wie in Abb. 8.1 angedeutet). Dadurch ist die Verlagerung der Anwendungslogik (z.B. für Zuweisungen) von der *Runtime-* auf die *Pushdown*-Schicht ohne Weiteres möglich. Die *Pushdown*-Schicht beinhaltet den Mechanismus zur Query-Bildung und ggf. die Queries für verschiedene DBSe (solange noch keine Standard Query-Sprache für die Manipulation von XML Daten existiert). Die Aufgaben, die von der *Pushdown*-Schicht übernommen werden, sind also im Wesentlichen die in dieser Arbeit entwickelten Hauptfunktionen aus der ScopeDAO-Schicht des Prototypen (siehe Kapitel 6.1.2.1, Seite 87). Dadurch kann sowohl die *Runtime* als auch die DAO-Schicht auf die *Pushdown*-Funktionen zurückgreifen. Die DAO-Schicht ist im Prinzip nur für das Speichern von Literal-Werten für Variablen und

anderen Informationen zu Prozessen und Instanzen verantwortlich und könnte ggf. ganz eliminiert werden und deren Funktionen in die *Pushdown*-Schicht übertragen werden. Der wesentliche Unterschied zwischen *DAO*- und *Pushdown*-Schicht ist der Ort der Datenverarbeitung. Während Daten der *DAO*-Schicht in der *Runtime* verarbeitet werden, werden bei Verwendung der *Pushdown*-Schicht diese Daten ausschließlich innerhalb des DBSs verarbeitet und lediglich Resultate (wie z.B. das Ergebnis der Abbruchbedingung einer Schleife) an die *Runtime* weitergegeben. Ein einheitliches Wrapper Element für alle XML-Dokumente oder Fragmente erleichtert die Query-Erstellung, durch das Typsystem ist sowieso bekannt, um welchen Typ es sich jeweils handelt. Dies muss nicht zwangsläufig auf der Datenebene (z.B. innerhalb des XML-Dokuments) widergespiegelt werden (im Gegensatz zu Tabelle 5.1, Seite 71).

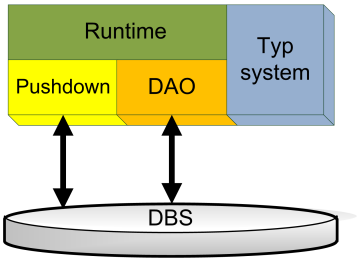


Abbildung 8.1.: Referenzarchitektur für ein Pushdown WfMS bestehend aus einem global sichtbaren Typsystem und einer Pushdown-Schicht, die von einer optionalen DAO-Schicht und der Runtime-Schicht aus aufgerufen werden kann.

8.1.2. Architekturmodell Hybrides WfMS

Wenn sowohl WS- als auch Query-Pushdown vollständig (asynchron und synchron) implementiert werden können, findet im Prinzip kein Datenaustausch von Variableninhalten zwischen Runtime und DBS statt. Lediglich Ergebnisse von Bedingungsevaluationen für Schleifen- und Kontrollstrukturen werden an die Runtime zurückgeliefert. Die Datenverarbeitung und Datenspeicherung findet ausschließlich im DBS statt. Dies stellt also eine Zwischenform, zwischen der klassischen WfMS Architektur und dem Ansatz das DBS als WfMS erster Klasse zu betrachten, dar. Wir bezeichnen dieses neue Architektur als *Hybriden Ansatz* und erweitern das Architekturmodell aus Abb. 3.7 (Seite 51) in Abb. 8.2. Der hybride Ansatz zeichnet sich dadurch aus, dass die Prozess- und Datenverwaltung strikt voneinander getrennt sind. Die Prozessverwaltung übernimmt im Wesentlichen die Navigation durch den Prozess, indem die Kontrollfluss-Entscheidungen interpretiert und ausgeführt werden. Diese strikte Trennung steht im Gegensatz zur klassischen Variante in Abb. 8.2a, bei der die Datenverwaltung sowohl innerhalb der WF-Runtime und des DBS erfolgt und der voll integrierten Variante in Abb. 8.2b, in der beide Aufgaben nur vom DBS übernommen werden.

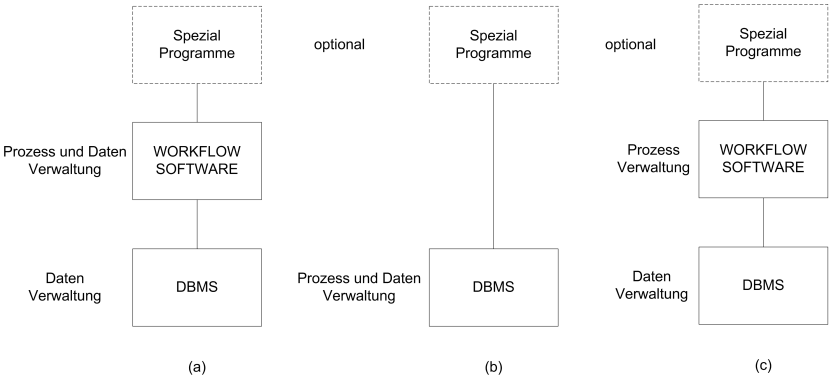


Abbildung 8.2.: Klassische WfMS Architektur (a), DBMS als Erste-Klasse WfMS (b) und der Hybride Ansatz (c). Vgl. [AIL98]

8.2. Weiterführende Arbeiten

Wir werden nun einige Erweiterungen und weitere wissenschaftliche Untersuchungsmöglichkeiten vorstellen, die mit den Pushdown-Konzepten aus Kapitel 4 im Rahmen weiterer Arbeiten untersucht werden können. Teilweise greifen wir hier Modifikationen auf, die schon direkt am Prototypen (Kapitel 6.2.3, Seite 97) vorgestellt wurden.

XQuery(-Update)-Pushdown Man sollte mit wenig Aufwand innerhalb des Prototyps den XQuery-Pushdown synchron als ExpressionEvaluation-Pushdown und asynchron als Assignment-Pushdown realisieren können. Da XQuery-Ausdrücke deutlich komplexer werden können als XPath-Ausdrücke, sind weitere Messungen dieser konkreten Pushdown-Technik interessant. Unterstützt das DBS bereits den Kandidat für die *XQuery Update Facility* [W3C09], kann der bereits implementierte XPath-Pushdown prinzipiell standardisiert werden und somit auch ohne zusätzlichen Aufwand für XQuery-Ausdrücke verwendet werden.

Unterstützung und Evaluierung weiterer XML-Enabled DBSe Man könnte noch weitere XML-Enabled DBSe, wie z.B. Microsoft SQL Server und Oracle Database, an den Prototypen anbinden und evaluieren. In Zukunft könnten die Messungen aus Kapitel 7 für XML-Enabled DBSe wiederholt werden, wenn z.B. die XML-Verarbeitung in diesen Systemen ausgereifter ist. Gegebenenfalls können dann alle in den Prototyp implementierten Pushdown-Konzepte auch durch PostgreSQL und andere DBSe realisiert werden. Insofern sich in Zukunft der XPath/XQuery-Pushdown auf XML-Enabled DBSe standardisieren lässt, müsste für eine Evaluierung der verschiedenen DBSe keine spezifische Anpassungen am Prototyp (außer die Standardisierung) mehr vorgenommen werden. Darüber hinaus könnte man den jetzigen Prototypen für verschiedene Datenbankparameter (Query-Optimierung, Anzahl Seitenpuffer etc.) testen und so seine Leistung optimieren.

(Automatische) XML-Indizierung Falls die XML-Validierung in den Prototypen implementiert wurde (siehe Kapitel 6.2.3, Seite 97) ist es, zumindest für das DBS IBM DB2, möglich XML-Knoten (Elemente, Attribute etc.) zu indizieren. Dies kann bei iterativen Zugriffen auf bestimmte Elemente des XML-Dokuments (siehe Anwendungsfall Kapitel 2.5.1.1, Seite 39) zu einer beschleunigten Ausführung der XPath/XQuery-Ausdrücken führen. Welche XML-Knoten zu indizieren sind, könnte vom Anwender vorgegeben, durch den WF-Compiler oder zur Laufzeit anhand der XPath/XQuery-Ausdrücke ermittelt werden. Letzteres könnte auch eine Funktion sein, die in die DB-Technologie von XML-Enabled DBSen, z.B. in den DB-Optimierer, aufzunehmen ist.

Natives XML-DBS Es ist denkbar innerhalb der DAO-Schicht des Prototyps, für die Speicherung und Verarbeitung von Variablen, mit einem nativen XML-DBS, wie z.B. eXist-db[1], zu kommunizieren. Hier würden die Variablen jeweils als eigenständiges Dokument oder in einer separaten XML Struktur als einzelnes XML Dokument abgespeichert, manipuliert und zugewiesen werden. Möglicherweise führt dies zu einer erheblichen Performanzsteigerung, da diese Systeme auf die Verarbeitung von XML Daten ausgelegt sind und da der zusätzliche Overhead über SQL-Anweisungen auf die XML-Felder in Tabellen zugreifen zu müssen wegfällt. Durch entsprechende Laufzeitmessungen könnte diese These belegt werden. Darüber hinaus ist es denkbar, die Prozessinformationen weiterhin in einem relationalem DBS zu halten. Die DAO-Schicht würde damit zweigeteilt sein und der Prototyp würde entsprechend zwei DBSe (Relational und Nativ XML) benötigen. Dies stellt eine Verfeinerung der Möglichkeit dar Informationen zum WfMS auf verschiedene DBen und DBSe aufzuteilen (siehe WebSphere Process Server Kapitel 3.3.4, Seite 56).

Weitere DAO Spezialisierungen Man könnte andere Datenstrukturen für die Prozessvariablen zulassen, z.B. ein relationales Schema. Diese würden sich direkt auf eine Relation eines relationalen DBSs abbilden lassen, was insbesondere für ETL-WFs, die tabellenorientierte Daten verarbeiten, zu Performanzvorteilen führen würde. Einerseits fällt dabei die XML-Konvertierung weg und andererseits können die schnellen relationalen Operatoren der integrierten DB direkt verwendet werden. Des Weiteren wäre es denkbar die Datenstruktur (XML oder Relational) für jede Prozessvariable einzeln festzulegen. Um all dies Umzusetzen benötigt man eine Reihe spezialisierter DAOs und Varianten des Query-Pushdown (siehe Tabelle 4.1, Seite 61) die auf die entsprechenden DBSe (Relational, Nativ XML) zugreifen und deren Daten verarbeiten können. Falls die Persistenz dieser Daten nicht zwingend erforderlich ist, könnten außerdem hauptspeicherbasierte DBSe zum Einsatz kommen.

Pushdown für andere WfMSe Der ExpressionEvaluation-Pushdown könnte für andere, z.B. eher datenorientierten WfMSe wie Taverna oder Microsoft Trident umgesetzt und

[1] http://exist.sourceforge.net/

evaluiert werden. Zumindest für die Aktivitäten der beiden Systeme, die XPath-Auswertungen durchführen. Außerdem könnte geprüft werden, ob die anderen Pushdown-Konzepte dort ebenfalls anwendbar sind oder auch neue Pushdown-Konzepte umsetzbar und somit in die Pushdown-Hierarchie aufzunehmen sind.

Pushdown-Optimierer Nach Evaluation weiterer Test- und Anwendungsfälle könnte in den Prototypen ein z.B. kostenbasierter Pushdown-Optimierer eingeführt werden, der nach bestimmten Kriterien entscheidet, wann die Aufgabe in der WF-Engine und wann innerhalb des DBSs durchgeführt wird. Im Anwendungsfall war z.B. die Evaluierung der IF-Bedingung durch den kleinen Inhalt der Variable innerhalb der WF-Engine schneller (siehe Kapitel 7.3.5.1, Seite 120). Allerdings müssten hierzu die Daten aus dem DBS in die WF-Engine geladen werden, was wiederum Zeit benötigt. Ob solche Effekte eine Optimierung der Instanzlaufzeit verhindern muss hierbei evaluiert werden.

Pushdown von SIMPL-Aktivitäten Eine weitere Arbeit kann sich mit der Implementierung des Pushdown für die SIMPL-Aktivitäten (siehe Kapitel 3.2.1, Seite 45) in Apache ODE befassen. Hierzu könnten UDFs und Stored Procedures für die integrierte WF-DB geschrieben werden, welche direkt mit dem SIMPL-Kern kommunizieren und so ohne Umweg durch die WF-Engine die Daten direkt miteinander austauschen. Alternativ könnte der Pushdown auch an den SIMPL-Kern erfolgen, der dabei die Informationen zur integrierten WF-DB erhält um so die Daten austauschen zu können. Dies wäre insbesondere für die SIMPL-Aktivitäten *RetrieveData* und *WriteDataBack* interessant [RRS+10].

Nexus DS Operatoren NexusDS [CEB+09] ist ein verteiltes, Datenstrom verarbeitendes System. Sein Anwendungsschwerpunkt ist die sog. Angereicherte Realität (*engl. Augmented Reality*), hierbei werden meist Bilder der realen Welt durch zusätzliche Informationen angereichert. Das System kann aus sehr heterogenen Plattformen (mobile Endgeräte, Server, Grafikkarten etc.) bestehen und eine Vielzahl von Datenquellen anbinden. Der in [CEB+09] vorgestellte Anwendungsfall erhält von einem Smartphone dessen aktuelle geographische Position (z.B. via GPS). Anhand dieser Information wird nun auf Servern und einem Grafikkarten-Cluster eine virtuelle 3D-Ansicht erstellt, die mit Informationen über interessante Orte (*POI*) und z.B. verfügbare Taxis angereichert ist. Diese 3D-Ansicht wird als 2D-Bild an das Smartphone zurückgeschickt. Solange die Anwendung auf dem Smartphone nicht abgebrochen wird, wird die Ansicht ständig aktualisiert und an das Smartphone geschickt, weshalb von einem Datenstrom System gesprochen wird. Um die verschiedenen Datenquellen in NexusDS anzusprechen, werden sog. *Operatoren* implementiert. Hier ist es denkbar entsprechende ExpressionEvaluation-Pushdown Operatoren zu realisieren, die z.B. für die Auswertung von XPath-Ausdrücken auf XML Dokumenten einer XML-(Enabled/Nativen)-Datenbank einen XPath-Pushdown vornehmen und so die Last des Gesamtsystems noch weiter zu verteilen. Insbesondere, wenn der Datenbankserver ein eigenständiges System darstellt und die Anfragen entsprechend komplex sind. Umgekehrt könnte es für Datenfluss-orientierte WfMSe

interessant sein eine Nexus DS Anwendung als Aktivität zu verwenden. Da diese ständig Daten generiert, können diese kontinuierlich in einem entsprechendem WF analysiert oder weiterverarbeitet werden.

UDFs als Alternative zu DB Middleware Man kann die Idee, UDFs und Stored Procedures anstelle von direkten SQL-Aufrufen zu verwenden (aus Microsoft Trident siehe Kapitel 3.3.3, Seite 55), dahingehend untersuchen, ob dieses Modell es erlaubt DB-Middlewaresysteme wie Hibernate und openJPA aus WfMSen zu entfernen, ohne die Flexibilität bei der Wahl des DBSs zu verlieren. Gegebenenfalls kann durch diese schlankeren Systeme bereits eine schnellere Workflow-Ausführung ermöglicht werden. Um diesen Ansatz zu überprüfen, könnte z.B. für Apache ODE eine eigene DAO geschrieben werden, die nur UDFs und Stored Procedures aufruft und Letztere entsprechend für mehrere DBSe implementiert werden. Zusätzlich kann geprüft werden, ob es möglich ist UDFs und Stored Procedures zu definieren, welche die Pushdown-Konzepte realisieren. Insbesondere könnten dann auch bei Zuweisung an Variablen SQL *INSERT*-Ausdrücke zum Einsatz kommen. Durch die Struktur der DAO-Schicht des Prototyps wurde jeweils für die Variablen der linken Seite vor der Zuweisung eine entsprechende Zeile in der Tabelle erstellt. Somit kamen nur SQL *UPDATE*-Ausdrücke zum Einsatz (siehe Kapitel 6.1.2.1, Seite 87).

Hybrider Ansatz Um den Hybriden Ansatz und die strikte Trennung von Prozess- und Datenverwaltung vollständig umzusetzen, müssten ebenfalls die Initalwerte von Prozessvariablen bereits im DBS vorliegen. In unserem Prototypen werden derzeit die BPEL-Literale über die DAO-Schicht, während der Laufzeit einer Instanz, an das DBS übergeben, was dieser strikten Trennung widerspricht. Es ist jedoch denkbar, während der *Deployment*-Phase eines Prozesses diese Initialwerte in eine dafür vorgesehene Relation in das DBS zu übertragen und beim Aufruf der entsprechenden Zuweisung diesen Initialwert innerhalb des DBSs zuzuweisen. Nach dieser Erweiterung des Prototyps würde dieser dem Hybriden Ansatz weitestgehend entsprechen. Ungeklärt ist jedoch wie die Inhalte der SOAP-Nachricht, welche für die Instanziierung eines Prozesses in Apache ODE verantwortlich ist, bei strikter Trennung von Prozess- und Datenverwaltung ohne den Umweg durch die WF-Engine in die integrierte DB gelangen sollen.

DBS als WfMS erster Klasse Falls ein System wie unter *UDFs als Alternative zu DB Middleware* beschrieben umgesetzt werden konnte, insbesondere mit Umsetzung der Pushdown-Konzepte, ist der beschriebene Ansatz von [AIL98] das DBS als WfMS erster Klasse zu betrachten nicht mehr weit. Es könnte dann untersucht werden, ob es möglich ist die Prozesslogik, z.B. für eine BPEL-Engine, vollständig in das DBS zu übertragen, und ob es möglich ist im DBS einen BPEL-Compiler bzw. Interpreter zu schreiben. Wir vermuten jedoch, dass es an dieser Stelle sinnvoller ist den von uns eingeführten *Hybriden Ansatz* (siehe Abb. 8.2c) zu verwenden.

Es existieren somit eine ganze Reihe weiterer Möglichkeiten die Integration von Daten, Datenstrukturen und DBSen mit WfMSen auszubauen und so Verbesserungen in der Laufzeit durch die Art der Datenverarbeitung und Datenübertragung zu erlangen. Insbesondere sollte je nach Datenstruktur der zu verarbeitenden Daten ein passendes Verarbeitungsmodell gewählt werden, um die optimale Leistung zu erhalten. So ist es auch denkbar andere Datenbank- und datenbanknahe Technologien einzusetzen. Als Beispiel könnten verteilte DBSe bei verteilten WfMSen zum Einsatz kommen oder *Semantic Web Frameworks*[2] bei Verwendung von Resource Description Framework-Graphen (RDF[3]) in entsprechenden WfMSen.

[2]Jena - http://jena.sourceforge.net/index.html
[3]http://www.w3.org/RDF/

9. Zusammenfassung

In dieser Arbeit haben wir uns mit der stärkeren Integration von DBSen in WfMSe befasst. Hierzu haben wir verwandte Arbeiten vorgestellt. Diese befassen sich hauptsächlich mit der Anbindung externer Datenbanken an WfMSe (BPEL/SQL, SIMPL) oder mit der Verschmelzung beider Systeme (DBS als WfMS erster Klasse). Für BPEL/SQL-WFs besteht prinzipiell die Möglichkeit der globalen Optimierung, wobei z.B. ein Ziel ist die tupelweise Verarbeitung innerhalb der WF-Runtime in eine Mengenoperation auf dem DBS zu überführen (PGM-Optimierer). Im Gegensatz dazu betrachten wir die stärkere Integration eines DBSs, das von WfMSen zur Speicherung der Prozess- und Instanzdaten verwendet wird. Hierbei soll die WF-Ausführung für den Anwender transparent bleiben, also keine Änderungen im WF nach sich ziehen. Dadurch sind wir auf lokale Optimierungen zur Laufzeit des WFs beschränkt. Die grundsätzliche Idee ist traditionelle Aufgaben, wie Zuweisungen und Webservice Aufrufe, von der Runtime-Ebene der WF-Engine auf das DBS zu übertragen. Wir haben bestehende Konzepte, um dieses „Hinunterschieben“ (Pushdown) zu ermöglichen, vorgestellt und neue Konzepte erarbeitet. Diese wurden prototypisch in der BPEL-Engine Apache ODE implementiert. Hierzu haben wir die aktuelle Software-Architektur, insbesondere die für die Implementierung wichtigen Teile, von Apache ODE vorgestellt und sind auf Implementierungsdetails und Probleme für unsere Erweiterung eingegangen. Um die Tauglichkeit der vorgestellten Pushdown-Konzepte zu überprüfen, wurde der Prototyp durch eine Reihe von Test- und einem Anwendungsfall evaluiert. Aus den vorgestellten Ergebnissen lassen sich Rückschlüsse und Voraussetzungen ableiten, die für eine verbesserte Workflow-Ausführung durch die Pushdown-Konzepte nötig sind (siehe Kapitel 9.1). Aus den Erfahrungen bei der Implementierung und der Architektur des Prototyps haben wir eine Referenzarchitektur für solche Pushdown-WfMSe vorgestellt und diese als *Hybriden-Ansatz* in Relation zu bestehenden WfMS-Architekturen gesetzt. Der Hybride-Ansatz trennt hierbei strikt die Prozess- (WF-Runtime) und Datenverwaltung (DBS) im Gegensatz zur klassischen WfMS-Architektur und stellt somit eine Mischform aus klassischer WfMS-Architektur und der Alternative DBMSe als Erste-Klasse WfMSe dar. Des Weiteren haben wir zahlreiche weitere Modifikationen und weiterführende Arbeiten vorgestellt, mit denen es möglich sein sollte die Workflow-Ausführung zu verbessern oder die Pushdown-Konzepte auch in anderen Systemen einzusetzen.

9.1. Schlussfolgerung

Wir haben erfolgreich die vorgestellten Pushdown-Konzepte in einen Prototyp umsetzen können. Hierbei fällt auf, dass für die vollständige Umsetzung die angebotene Funktionalität von DBSen eine entscheidende Rolle spielt. Während wir für das DBS IBM DB2 alle Konzepte umsetzen konnten, ging dies für PostgreSQL nur für einen sehr kleinen Teil. Die Pushdown-Konzepte selbst können für sich allein betrachtet je nach Anwendung (Komplexität der Aufgabe, Datenmenge) eine Optimierung oder Pessimierung nach sich ziehen. Dies scheint allerdings auch von der effizienten Implementierung der XML-Technologie in dem verwendeten DBS abhängig zu sein. Grundsätzlich konnten wir beim Zusammenspiel der Pushdown-Konzepte im Anwendungsfall einen deutlichen Stabilitäts- und Performanzvorteil gegenüber der Original WF-Engine feststellen. Hierbei findet die Datenmanipulation und Auswertung fast ausschließlich innerhalb des DBSs statt, während nur noch die Prozess-steuernden Anteile von der WF-Eninge verwaltet werden (*Hybrider Ansatz*). Dieser Anwendungsfall spiegelt insbesondere bestimmte Klassen von Simulationsworkflows wieder, die im Rahmen des SimTech Projekts von großem Interesse sind. Kritisch betrachtet könnte die Beschleunigung der Evaluierung von XPath-Ausdrücken innerhalb der DB2 auch durch die Implementierung der XPath-Engine in C anstatt Java entstehen. Außerdem ist nicht ausgeschlossen, dass sich die XPath-Engine in Apache ODE effizienter nutzen lässt. Wir wissen jedoch, dass auch für die XML-Felder im DBS IBM DB2 Datenbanktechnologie eingesetzt wird, die eine schnellere und stabilere Verarbeitung erlaubt. In jedem Fall sind durch das Anwenden der Pushdown-Konzepte die Daten einer WF-Instanz zu jeder Zeit persistent.

9.2. Ausblick

Diese Arbeit legt den Grundstein für zahlreiche weitere Arbeiten. Zum Beispiel kann das Konzept des WS- und ExpressionEvaluation-Pushdown als Operatoren in das NexusDS System implementiert werden. Für das SIMPL-Projekt könnten ebenfalls Pushdown-Funktionen implementiert werden, die eine direkte Interaktion des SIMPL-Kerns mit der integrierten DB des WfMSs erlauben, wodurch Daten ohne den Umweg über die WF-Engine ausgetauscht werden können. Außerdem können die Pushdown Konzepte in andere WfMSe implementiert werden. Der vorgestellte Prototyp kann um zahlreiche weitere Funktionen (siehe Kapitel 6.2.3, Seite 97) erweitert werden. Insbesondere die Anbindung und Evaluation an ein natives XML DBS klingt vielversprechend. Die Anbindung weiterer (zukünftiger) relationaler DBSe und weiterführende Auswertungen, für andere Workflow-Typen als in Kapitel 7 behandelt, sind ebenfalls von wissenschaftlichem Interesse. Weitere Arbeiten könnten sich darüber hinaus, unter der Verwendung der vorgestellten Konzepte und Erweiterungen, auch mit einer Umsetzung 'DBS als erste Klasse WfMS' für die Workflowsprache WS-BPEL befassen.

9.3. Danksagungen

Ich möchte mich bei Prof. Dr. Bernhard Mitschang und Dr. habil. Holger Schwarz für die Ermöglichung dieser Arbeit, die Bereitstellung von Arbeitsräumen und technischen Ressourcen sowie der benötigten Lizenz für das DBS IBM DB2 bedanken. Weiterer Dank gilt Dipl.-Inf. Michael Reiter für die fachlichen Korrekturen an Kapitel 3 und ein ganz besonderer Dank geht an meinen Betreuer Dipl.-Inf. Peter Reimann für seine stets guten Ideen, die den Verlauf der Arbeit geprägt haben und für seine professionellen und konstruktiven Kommentare zur Verbesserung dieser Ausarbeitung. Des Weiteren möchte ich mich bei allen Mitarbeitern der Abteilung Anwendersoftware am Institut für Parallele und Verteilte Systeme sowie der Apache ODE Mailingliste bedanken.

Zum Schluss möchte ich meiner Familie und meinen Freunden danken, die immer für mich da sind und moralische Unterstützung geleistet haben.

A. Abkürzungsverzeichnis

API Application Programming Interface

BIIF BioInformatics Interchange Format

DAO Data Access Objects

DB Datenbank

DBMS Datenbank Management System

DBS Datenbanksystem

DTD Document Type Definition

DWARF DataWarehouse Architecture for pRotein classiFication

ETL Extraction Transformation Load

GUI Graphical User Interface

OASIS Organization for the Advancement of Structured Information Standards

PGM Process Graph Model

RDF Resource Description Framework

SGML Standard Generalized Markup Language

SIMPL SimTech - Information Management, Processes, and Languages

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

UML Unified Modeling Lanuage

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

VPU Virtual Processing Unit

W3C World Wide Web Consortium

WF Workflow

WfMS Workflow Managment System

WPS WebSphere Process Server

WS Webservice

WS-BPEL WS-Business Process Execution Language

WSDL Web Service Description Language

WSFL Web Services Flow Language

WYSIWYG What You See Is What You Get (Oft im Zusammenhang mit graphischen Editoren verwendet)

XML eXtensible Markup Language

XOML Extensible Object Markup Language

B. Entwicklungsumgebung

Wir werden einige Eckdaten zur verwendeten Software und eine Installationsanleitung für den Prototypen, z.B. für weitere Auswertungen oder Implementierungsarbeiten angeben. Das Entwicklungssystem wurde mit dem Betriebssystem Windows XP Professional 32-bit betrieben und lief auf einem Intel Core2Duo T7300@2GHz Prozessor. Das System verfügte außerdem über 3GB Hauptspeicher und einer Grafikkarte mit eigenständigem Grafikspeicher.

B.1. Verwendete Software

Die verwendete Software bezieht sich immer auf Windows XP 32-bit:

- Apache Buildr 1.3.5 - http://buildr.apache.org/
- Apache ODE 1.3.4 - http://ode.apache.org
- Apache Tomcat 6.0.29 - http://tomcat.apache.org/
- BPEL-Designer (Eclipse Galileo 3.5) - http://www.eclipse.org/bpel/
- Eclipse Helios Java EE 3.6 - http://www.eclipse.org/
- IBM DB2 V9.7 (kostenpflichtige Lizenz benötigt!) - http://www.ibm.com/software/data/db2/
- Java JRE und JDK 1.6.0_23 - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- PostgreSQL 8.4 - http://www.postgresql.org/
- Ruby 1.8.7 + DevKit 3.4.5 - http://rubyonrails.org/download
- Silk Subversion 1.6.12 - http://www.sliksvn.com/en/download
- SoapUI 3.6.1 - http://www.soapui.org/
- SQuirreL SQL Client 3.1.2 - http://squirrel-sql.sourceforge.net
- Strawberry Perl 5.12.1 - http://strawberryperl.com/

B.2. Programmierumgebung

Es sollte jeweils das aktuelle Java Runtime Environment (JRE) und Java Development Kit (JDK) auf dem System installiert sein und sichergestellt werden, dass die Umgebungsvariablen *JAVA_HOME* und *JRE_HOME* korrekt auf den jeweiligen Installationspfad (*bin*-Verzeichnis) gesetzt sind.

Für die Programmierumgebung und das Compilieren des ODE Quellcodes muss Ruby 1.8.7 + DevKit 3.4.5 installiert werden und anschließend über den Kommandozeilen-Befehl *'gem install buildr -v 1.3.5 –platform mswin32'* das Buildsystem Apache Buildr installiert werden. Der Quellcode von Apache ODE 1.3.4 wurde per SVN aus dem Apache SVN Server ausgecheckt und konnte im Top-Level Quellcode Ordner mit dem Befehl *'buildr _1.3.5_ package test=no'* compiliert werden. Um die nötigen Projektinformationen für einen Import nach Eclipse Helios zu generieren, kann der Befehl *'buildr _1.3.5_ eclipse'* verwendet werden. Programmiert wurde innerhalb von Eclipse, compiliert über die Eingabeaufforderung mit Hilfe von Apache Buildr. Anschließend wurden die erzeugten Java Archive (Jar)-Dateien in den Class-Path (*[tomcatdir]/webapps/ode/WEB-INF/lib*) einer in Apache Tomcat eingebetteten Apache ODE 1.3.4 Version kopiert. Für die Einbettung von Apache ODE in Tomcat sei auf `http://ode.apache.org/war-deployment.html` verwiesen.

Der Quellcode des Prototyps liegt auf der DVD unter *[DVD]/Implementierung/Prototyp/src/ode-1.3.4-prototyp* ab. Der Quellcode der für die Evaluation modifizierten Original Apache ODE 1.3.4 liegt auf der DVD unter *[DVD]/Implementierung/ApacheODE/src/ode-1.3.4-orig* ab.

B.3. Workflow Erstellung

Die BPEL-Workflows zum Testen der Implementierung, der Testfälle sowie des Anwendungsfalls wurden mit dem auf Eclipse basierendem BPEL-Designer modelliert. Die verwendete Version konnte nicht zum automatischen bekanntmachen der Workflows, aufgrund eines Fehlers beim Kopieren des Projekts, verwendet werden. Der Prozessordner wurde daher manuell in den *processes* Ordner (*[tomcatdir]/webapps/ode/WEB-INF/processes*) von Apache ODE kopiert. Eine gute Installationsanleitung für den BPEL-Designer ist unter `http://www.se.uni-hannover.de/lehre/tutorials/BPEL-ODE-Eclipse-Getting-Started.php` zu finden. Da es zu Versionsinkompatibilitäten der BPEL-Designer-Plugins mit den Eclipse Versionen kommen kann, befindet sich die verwendete und ausführbare Version des BPEL-Designers auf der DVD unter *[DVD]/Implementierung/BPEL-Designer*.

B.4. Installation des Prototyps

Zuerst sollte das zu verwendende Datenbanksystem eingerichtet werden (siehe Abschnitt B.4.1). Außerdem muss Java JRE in der jeweils aktuellsten Version installiert sein und die Umgebungsvariable *JRE_HOME* korrekt auf den Installationsordner verweisen. Der ausführbare und in Apache Tomcat eingebettete Prototyp befindet sich auf der DVD unter *[DVD]/Implementierung/Prototyp/Ausführbar*. Es reicht aus, den Ordner auf das System zu kopieren. Jetzt müssen nur noch einige Einstellungen vorgenommen werden, die im Abschnitt B.4.1 und B.4.2 beschrieben werden.

B.4.1. Datenbank Setup

Für jedes DBS sollte eine leere Datenbank angelegt werden. Am Besten erledigt man das durch die jeweilige Steuerzentrale des DBSs. Je nach DBS müssen unterschiedliche Schemata geladen werden, dies kann ebenfalls durch die SQL-Konsole in der Steuerzentrale oder durch z.B. Squirrel-SQL erfolgen. Bei Verwendung von Squirrel-SQL müssen allerdings die passenden JDBC Treiber in den *lib*-Ordner von Squirrel-SQL kopiert werden.

IBM DB2 ODE-Schema auf DVD unter
[DVD]/Implementierung/DBSetup/ode-134-hib-db2_prototyp.sql

PostgreSQL ODE-Schema auf DVD unter
[DVD]/Implementierung/DBSetup/ode-134-hib-pgsql_prototyp.sql

Zudem muss für die Verwendung mit PostgreSQL die Hibernate-DAO im Classpath *[tomcatdir]/webapps/ode/WEB-INF/lib* des Prototypen ausgetauscht werden:

IBM DB2 Hibernate-DAO auf DVD unter
[DVD]/Implementierung/DBSetup/ode-lib/IBM DB2/ode-dao-hibernate-1.3.4.jar

PostgreSQL Hibernate-DAO auf DVD unter
[DVD]/Implementierung/DBSetup/ode-lib/PostgreSQL/ode-dao-hibernate-1.3.4.jar

Für die Verwendung von IBM DB2 mit dem Protoyp muss zusätzlich in den Classpath der Apache Tomcat Installation *[tomcatdir]/lib* die Lizenzdatei *db2jcc_license_cu.jar* eingespielt werden. Diese befindet sich innerhalb der DB2 Installation (z.B. *C:\IBM\SQLLIB\java*). Die JDBC-URL inklusive Datenbankname, Datenbankbenutzer und Passwort muss in der Konfigurationsdatei *ode-axis2.properties* abgeändert werden. Diese befindet sich unter *[tomcatdir]/webapps/ode/WEB-INF/conf*, die jeweiligen JDBC-URLs sind bereits in der Konfigurationsdatei des Prototypen enthalten.

Für die Verwendung des Webservice-Pushdown (nur für IBM DB2) muss die DB2 Datenbank mit der Zeichenkodierung *„utf8“* erstellt werden und die „Web-Services“ Funktionalität

aktiviert werden. Diese kann in der DB2 Steuerzentrale für die entsprechende DB über die Eigenschaft „Web-Services“ aktiviert und deaktiviert werden. Außerdem muss die folgende benutzerdefinierte UDF erstellt werden[1]:

```
CREATE FUNCTION db2xml.soaphttplg (
              endpoint_url VARCHAR(256),
              soapaction VARCHAR(256),
              soap_body varchar(3072))
     RETURNS clob(5M)
     LANGUAGE C PARAMETER STYLE DB2SQL
     SPECIFIC soaphttplg
     EXTERNAL NAME 'db2soapudf!soaphttpvico'
     SCRATCHPAD FINAL CALL FENCED
     NOT DETERMINISTIC CALLED ON NULL INPUT
     NO SQL EXTERNAL ACTION DBINFO;
```

B.4.2. Prototyp Einstellungen

In der Konfigurationsdatei des Prototypen können die einzelnen Pushdown-Konzepte ein- und ausgeschaltet werden. Hierzu exisiteren fünf Parameter:

ode-axis2.db.mode.enhanced (Werte: true/false) Hauptschalter, bei *false* werden alle anderen Pushdown-Einstellungen ignoriert.

ode-axis2.db.mode.enhanced.sync (Werte: true/false) Schaltet den synchronen XPath-Pushdown für Pfadselektionen ein (ExpressionEvaluation-Pushdown).

ode-axis2.db.mode.enhanced.sync.expression (Werte: true/false) Schaltet den synchronen XPath-Pushdown zusätzlich für komplexe XPath-Ausdrücke ein (ExpressionEvaluation-Pushdown).

ode-axis2.db.mode.enhanced.async (Werte: true/false) Schaltet den asynchronen XPath-Pushdown ein (Assignment-Pushdown), falls *true* findet die synchrone Auswertung innerhalb von ASSIGN-Aktivitäten nicht mehr statt.

ode-axis2.db.mode.enhanced.ws (Werte: true/false) Schaltet den Webservice-Pushdown ein oder aus.

[1] https://www.ibm.com/support/docview.wss?uid=swg1IZ46071

C. Anwendungsfall Proteinmodellierung - Mustersuche

C.1. BIIF XML Beispiel

XML Beispieldokument (gekürzt) des Resultats des Webservice-Aufrufs des Anwendungsfalls aus Kapitel 2.5 und der Evaluierung in Kapitel 7.

```
<biif>
  <date>2010-06-01</date>
  <creator>DWARF_ACCESS.pl#getHFamilyAlignment</creator>
  <description>Homologous Family Alignment of Family 1106 from CYPED</description>
  <aln>
    <seq>
      <header>AAH29014.1</header>
      <lsid>urn:lsid:dwarf.uni-stuttgart.de:p450_v2_online_091215:4568</lsid>
      <source>
        <database id="4568" name="dwarf" version="p450_v2_online_091215"/>
        <database href="http://www.ncbi.nlm.nih.gov/Genbank/index.html" id="AAH29014.1"
            name="GenBank"/>
        <database id="20809428" name="General Identifier"/>
      </source>
      <aa>MEVLGLLKFEVSGTIVTVTLLVA[...]EASPETQVPLQLESKSALGPKNGVYIKIVSR</aa>
      <annotation countGaps="no">
        <region name="p450 domain" start="1" stop="499"/>
        <region name="alphaA" start="63" stop="69"/>
        <region name="beta1_1" start="74" stop="80"/>
        <region name="beta1_2" start="86" stop="92"/>
        <region name="alphaB" start="92" stop="99"/>
        <region name="beta1_5" start="105" stop="109"/>
        <region name="alphaC" start="130" stop="144"/>
        <region name="alphaD" start="151" stop="164"/>
        <region name="alphaE" start="181" stop="194"/>
        <region name="alphaF" start="207" stop="214"/>
        <region name="alphaG" start="249" stop="260"/>
        <region name="alphaH" start="278" stop="284"/>
        <region name="alphaI" start="332" stop="361"/>
        <region name="alphaJ" start="362" stop="368"/>
        <region name="beta1_4" start="410" stop="414"/>
        <region name="beta2_1" start="416" stop="420"/>
        <region name="beta2_2" start="422" stop="426"/>
```

```
33          <region name="beta1_3" start="429" stop="433"/>
34          <region name="alphaK_1" start="438" stop="442"/>
35          <region name="Meander loop" start="446" stop="454"/>
36          <region name="Cys Pocket" start="468" stop="481"/>
37          <region name="alphaL" start="482" stop="501"/>
38          <region name="beta3_3" start="502" stop="505"/>
39          <region name="beta4_1" start="512" stop="515"/>
40          <region name="beta4_2" start="520" stop="523"/>
41          <region name="beta3_2" start="526" stop="529"/>
42          <region name="beta3_1" start="174" stop="181"/>
43          <region name="alphaK" start="391" stop="404"/>
44        </annotation>
45      </seq>
46          [...]
47      <seq>
48        <header>AAB87704.1</header>
49        <lsid>urn:lsid:dwarf.uni-stuttgart.de:p450_v2_online_091215:4572</lsid>
50        <source>
51          <database id="4572" name="dwarf" version="p450_v2_online_091215"/>
52          <database href="http://www.ncbi.nlm.nih.gov/Genbank/index.html" id="AAB87704.1"
                name="GenBank"/>
53          <database id="1698440" name="General Identifier"/>
54        </source>
55        <aa>MEVLGLLKFEVSGTIVTVTLLVA[...]EASPETQVPLQLESKSALGPKNGVYIKIVSR</aa>
56        <annotation countGaps="no">
57          <region name="p450 domain" start="1" stop="499"/>
58          <region name="alphaA" start="63" stop="69"/>
59          <region name="beta1_1" start="74" stop="80"/>
60          <region name="beta1_2" start="86" stop="92"/>
61          <region name="alphaB" start="92" stop="99"/>
62          <region name="beta1_5" start="105" stop="109"/>
63          <region name="alphaC" start="130" stop="144"/>
64          <region name="alphaD" start="151" stop="164"/>
65          <region name="alphaE" start="181" stop="194"/>
66          <region name="alphaF" start="207" stop="214"/>
67          <region name="alphaG" start="249" stop="260"/>
68          <region name="alphaH" start="278" stop="284"/>
69          <region name="alphaI" start="332" stop="361"/>
70          <region name="alphaJ" start="362" stop="368"/>
71          <region name="beta1_4" start="410" stop="414"/>
72          <region name="beta2_1" start="416" stop="420"/>
73          <region name="beta2_2" start="422" stop="426"/>
74          <region name="beta1_3" start="429" stop="433"/>
75          <region name="alphaK_1" start="438" stop="442"/>
76          <region name="Meander loop" start="446" stop="454"/>
77          <region name="Cys Pocket" start="468" stop="481"/>
78          <region name="alphaL" start="482" stop="501"/>
79          <region name="beta3_3" start="502" stop="505"/>
80          <region name="beta4_1" start="512" stop="515"/>
81          <region name="beta4_2" start="520" stop="523"/>
82          <region name="beta3_2" start="526" stop="529"/>
```

```
      <region name="beta3_1" start="174" stop="181"/>
      <region name="alphaK" start="391" stop="404"/>
    </annotation>
  </seq>
  <similarity>*****************[...]*****************************</similarity>
 </aln>
</biif>
```

Listing C.1: BIIF XML Beispiel (gekürzt)

C.2. BPEL Prozess des Anwendungsfalls für die Mustersuche

Der BPEL Prozess für den Anwendungsfall aus Kapitel 2.5.

```
<!-- ODEDwarfUseCase BPEL Process [Generated by the Eclipse BPEL Designer] -->
<bpel:process name="ODEDwarfUseCase"
        targetNamespace="http://www.dwarf.uni-stuttgart.de"
        suppressJoinFailure="yes"
        xmlns:tns="http://www.dwarf.uni-stuttgart.de"
        xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
        xmlns:ns="http://www.dwarf.uni-stuttgart.de/ACCESS"
            xmlns:biif="http://www.dwarf.uni-stuttgart.de/BIIF/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Import the client WSDL -->
    <bpel:import namespace="http://www.dwarf.uni-stuttgart.de/ACCESS"
        location="DWARF_ACCESS.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
    <bpel:import namespace="http://www.dwarf.uni-stuttgart.de/BIIF/" location="biif.xsd"
        importType="http://www.w3.org/2001/XMLSchema"></bpel:import>
    <bpel:import location="ODEDwarfUseCaseArtifacts.wsdl"
        namespace="http://www.dwarf.uni-stuttgart.de"
              importType="http://schemas.xmlsoap.org/wsdl/" />

    <!-- ================================================================= -->
    <!-- PARTNERLINKS                                                      -->
    <!-- List of services participating in this BPEL process               -->
    <!-- ================================================================= -->
    <bpel:partnerLinks>
       <!-- The 'client' role represents the requester of this service. -->
       <bpel:partnerLink name="client"
                  partnerLinkType="tns:ODEDwarfUseCase"
                  myRole="ODEDwarfUseCaseProvider"
                  />
       <bpel:partnerLink name="dwarfAccessLink" partnerLinkType="tns:dwarfAPL"
           partnerRole="dwarfAPLType"></bpel:partnerLink>
    </bpel:partnerLinks>

    <!-- ================================================================= -->
```

```
29      <!-- VARIABLES                                                    -->
30      <!-- List of messages and XML documents used within this BPEL process -->
31      <!-- ================================================================= -->
32      <bpel:variables>
33          <!-- Reference to the message passed as input during initiation -->
34          <bpel:variable name="input"
35                  messageType="tns:ODEDwarfUseCaseRequestMessage"/>
36
37          <!--
38            Reference to the message that will be returned to the requester
39            -->
40          <bpel:variable name="output"
41                  messageType="tns:ODEDwarfUseCaseResponseMessage"/>
42          <bpel:variable name="dwarfAccessLinkResponse"
                messageType="ns:getSFamilyAlignmentResponse"></bpel:variable>
43          <bpel:variable name="dwarfAccessLinkRequest"
                messageType="ns:getSFamilyAlignmentRequest"></bpel:variable>
44          <bpel:variable name="biif" type="biif:biifType"></bpel:variable>
45          <bpel:variable name="Counter" type="xsd:int"></bpel:variable>
46          <bpel:variable name="pattern" type="xsd:string"></bpel:variable>
47          <bpel:variable name="positive" type="xsd:int"></bpel:variable>
48          <bpel:variable name="negative" type="xsd:int"></bpel:variable>
49          <bpel:variable name="accessions" type="xsd:string"></bpel:variable>
50          <bpel:variable name="proteinsequence" type="xsd:string"></bpel:variable>
51      </bpel:variables>
52
53      <!-- ================================================================= -->
54      <!-- ORCHESTRATION LOGIC                                           -->
55      <!-- Set of activities coordinating the flow of messages across the -->
56      <!-- services integrated within this business process              -->
57      <!-- ================================================================= -->
58      <bpel:sequence name="main">
59
60          <!-- Receive input from requester.
61               Note: This maps to operation defined in ODEDwarfUseCase.wsdl
62               -->
63          <bpel:receive name="receiveInput" partnerLink="client"
64                  portType="tns:ODEDwarfUseCase"
65                  operation="process" variable="input"
66                  createInstance="yes"/>
67
68          <!-- Generate reply to synchronous request -->
69          <bpel:assign validate="no" name="Prepare">
70              <bpel:copy>
71                  <bpel:from part="payload" variable="input">
72                      <bpel:query
                            queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                            <![CDATA[tns:superfamily]]></bpel:query>
73                  </bpel:from>
74                  <bpel:to part="superfamily_id" variable="dwarfAccessLinkRequest"></bpel:to>
75              </bpel:copy>
```

```
76          <bpel:copy>
77              <bpel:from part="payload" variable="input">
78                  <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[tns:pattern]]> </bpel:query>
79              </bpel:from>
80              <bpel:to variable="pattern"></bpel:to>
81          </bpel:copy>
82          <bpel:copy>
83              <bpel:from>
84                  <bpel:literal xml:space="preserve">0</bpel:literal>
85              </bpel:from>
86              <bpel:to variable="positive"></bpel:to>
87          </bpel:copy>
88          <bpel:copy>
89              <bpel:from>
90                  <bpel:literal xml:space="preserve">0</bpel:literal>
91              </bpel:from>
92              <bpel:to variable="negative"></bpel:to>
93          </bpel:copy>
94          <bpel:copy>
95              <bpel:from><bpel:literal
                    xml:space="preserve">Accessions:</bpel:literal></bpel:from>
96              <bpel:to variable="accessions"></bpel:to>
97          </bpel:copy>
98      </bpel:assign>
99      <bpel:invoke name="getSuperFamilySequences" partnerLink="dwarfAccessLink"
            operation="getSFamilyAlignment" portType="ns:DWARFAccessPortType"
            inputVariable="dwarfAccessLinkRequest"
            outputVariable="dwarfAccessLinkResponse"></bpel:invoke>
100     <bpel:assign validate="no" name="AssignWSResponse">
101         <bpel:copy>
102             <bpel:from part="biif" variable="dwarfAccessLinkResponse"></bpel:from>
103             <bpel:to variable="biif"></bpel:to>
104         </bpel:copy>
105     </bpel:assign>
106     <bpel:forEach parallel="no" counterName="Counter" name="ForEachProteinSequence">
107         <bpel:startCounterValue>
108             <![CDATA[1]]>
109         </bpel:startCounterValue>
110         <bpel:finalCounterValue><![CDATA[count($biif/aln/seq)]]></bpel:finalCounterValue>
111         <bpel:scope>
112             <bpel:sequence>
113                 <bpel:assign validate="no" name="PrepareProteinSequence">
114                     <bpel:copy>
115                         <bpel:from
                                expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0">
116                             <![CDATA[replace($biif/aln/seq[position()=$Counter]/aa,"-","","i")]]>
117                         </bpel:from>
118                         <bpel:to variable="proteinsequence"></bpel:to>
119                     </bpel:copy>
```

```
120                 </bpel:assign>
121                 <bpel:if name="IfPatternMatches"><bpel:condition
                        expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0">
                        <![CDATA[matches($proteinsequence, $pattern, "i")]]> </bpel:condition>
122                 <bpel:assign validate="no" name="AddSequenceHeader">
123                     <bpel:copy>
124                         <bpel:from>
125                             <![CDATA[$positive + 1]]>
126                         </bpel:from>
127                         <bpel:to variable="positive"></bpel:to>
128                     </bpel:copy>
129                     <bpel:copy>
130                         <bpel:from>
131                             <![CDATA[concat($accessions,
                                    $biif/aln/seq[position()=$Counter]/header/text(), "; ")]]>
132                         </bpel:from>
133                         <bpel:to variable="accessions"></bpel:to>
134                     </bpel:copy>
135                 </bpel:assign>
136                 <bpel:else>
137
138                     <bpel:assign validate="no" name="CountNegative">
139                         <bpel:copy>
140                             <bpel:from>
141                                 <![CDATA[$negative + 1]]>
142                             </bpel:from>
143                             <bpel:to variable="negative"></bpel:to>
144                         </bpel:copy>
145                     </bpel:assign>
146                 </bpel:else>
147
148                 </bpel:if>
149             </bpel:sequence>
150         </bpel:scope>
151     </bpel:forEach>
152     <bpel:assign validate="no" name="PrepareOutput">
153         <bpel:copy>
154             <bpel:from>
155                 <bpel:literal xml:space="preserve"> <tns:ODEDwarfUseCaseResponse
                        xmlns:tns="http://www.dwarf.uni-stuttgart.de"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
156   <tns:positive></tns:positive>
157   <tns:negative></tns:negative>
158   <tns:acc_codes></tns:acc_codes>
159 </tns:ODEDwarfUseCaseResponse>
160 </bpel:literal>
161             </bpel:from>
162             <bpel:to variable="output" part="payload"></bpel:to>
163         </bpel:copy>
164         <bpel:copy>
165             <bpel:from variable="accessions"></bpel:from>
```

```
166             <bpel:to part="payload" variable="output">
167                 <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[tns:acc_codes]]> </bpel:query>
168             </bpel:to>
169         </bpel:copy>
170         <bpel:copy>
171             <bpel:from variable="positive"></bpel:from>
172             <bpel:to part="payload" variable="output">
173                 <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[tns:positive]]> </bpel:query>
174             </bpel:to>
175         </bpel:copy>
176         <bpel:copy>
177             <bpel:from variable="negative"></bpel:from>
178             <bpel:to part="payload" variable="output">
179                 <bpel:query
                        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[tns:negative]]> </bpel:query>
180             </bpel:to>
181         </bpel:copy>
182     </bpel:assign>
183     <bpel:reply name="replyOutput"
184           partnerLink="client"
185           portType="tns:ODEDwarfUseCase"
186           operation="process"
187           variable="output"
188           />
189   </bpel:sequence>
190 </bpel:process>
```

Listing C.2: BPEL Prozess des Anwendungsfalls für die Mustersuche

C.3. WSDL Datei zum Aufruf des BPEL-Prozesses des Anwendungsfalls für die Mustersuche

Die WSDL Datei zum Aufruf des BPEL-Prozesses innerhalb von Apache ODE für den Anwendungsfall der Mustersuche.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
      xmlns:tns="http://www.dwarf.uni-stuttgart.de"
      xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
      xmlns:wsdl="http://www.dwarf.uni-stuttgart.de/ACCESS" name="ODEDwarfUseCase"
      targetNamespace="http://www.dwarf.uni-stuttgart.de"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

```
3
4  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
5       TYPE DEFINITION - List of types participating in this BPEL process
6       The BPEL Designer will generate default request and response types
7       but you can define or import any XML Schema type and use them as part
8       of the message types.
9       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
10    <plnk:partnerLinkType name="dwarfAPL">
11    <plnk:role name="dwarfAPLType" portType="wsdl:DWARFAccessPortType"/>
12   </plnk:partnerLinkType>
13    <import location="DWARF_ACCESS.wsdl"
          namespace="http://www.dwarf.uni-stuttgart.de/ACCESS"/>
14    <types>
15        <schema xmlns="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
              elementFormDefault="qualified"
              targetNamespace="http://www.dwarf.uni-stuttgart.de">
16
17            <element name="ODEDwarfUseCaseRequest">
18                <complexType>
19                    <sequence>
20                        <element name="superfamily" type="integer"/>
21                        <element name="pattern" type="string"/>
22                    </sequence>
23                </complexType>
24            </element>
25
26            <element name="ODEDwarfUseCaseResponse">
27                <complexType>
28                    <sequence>
29                        <element name="positive" type="integer"/>
30                        <element name="negative" type="integer"/>
31                        <element name="acc_codes" type="string"/>
32                    </sequence>
33                </complexType>
34            </element>
35        </schema>
36    </types>
37
38
39 <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
40      MESSAGE TYPE DEFINITION - Definition of the message types used as
41      part of the port type defintions
42      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
43    <message name="ODEDwarfUseCaseRequestMessage">
44        <part element="tns:ODEDwarfUseCaseRequest" name="payload"/>
45    </message>
46    <message name="ODEDwarfUseCaseResponseMessage">
47        <part element="tns:ODEDwarfUseCaseResponse" name="payload"/>
48    </message>
49
50 <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
51      PORT TYPE DEFINITION - A port type groups a set of operations into
52      a logical service unit.
53      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
54
55      <!-- portType implemented by the ODEDwarfUseCase BPEL process -->
56      <portType name="ODEDwarfUseCase">
57          <operation name="process">
58              <input message="tns:ODEDwarfUseCaseRequestMessage"/>
59              <output message="tns:ODEDwarfUseCaseResponseMessage"/>
60          </operation>
61      </portType>
62
63
64  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
65      PARTNER LINK TYPE DEFINITION
66      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
67      <plnk:partnerLinkType name="ODEDwarfUseCase">
68          <plnk:role name="ODEDwarfUseCaseProvider" portType="tns:ODEDwarfUseCase"/>
69      </plnk:partnerLinkType>
70
71      <binding name="ODEDwarfUseCaseBinding" type="tns:ODEDwarfUseCase">
72          <soap:binding style="document"
73                  transport="http://schemas.xmlsoap.org/soap/http" />
74          <operation name="process">
75                  <soap:operation
76                          soapAction="http://www.dwarf.uni-stuttgart.de/process" />
77                  <input>
78                          <soap:body use="literal" />
79                  </input>
80                  <output>
81                          <soap:body use="literal" />
82                  </output>
83          </operation>
84      </binding>
85      <service name="ODEDwarfUseCase">
86          <port name="ODEDwarfUseCasePort" binding="tns:ODEDwarfUseCaseBinding">
87                  <soap:address location="http://localhost:8080/ode/processes/ODEDwarfUseCase" />
88          </port>
89      </service>
90  </definitions>
```

Listing C.3: Die WSDL Datei zum Aufruf des BPEL-Prozesses des Anwendungsfalls für die Mustersuche.

Literaturverzeichnis

[AE09] A. K. André Eickler. *Datenbanksysteme. Eine Einführung*. Oldenbourg Wissenschaftsverlag GmbH, 2009. (Zitiert auf Seite 35)

[AIL98] A. Ailamaki, Y. E. Ioannidis, M. Livny. Scientific workflow management by database management. In *Proc. Tenth Int Scientific and Statistical Database Management Conf*, pp. 190–199. 1998. doi:10.1109/SSDM.1998.688123. (Zitiert auf den Seiten 7, 9, 43, 44, 51, 52, 133 und 136)

[AL00] L. M. Andreas Laux. XUpdate Working Draft, 2000. URL http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html. (Zitiert auf Seite 98)

[AMA06] A. Akram, D. Meredith, R. Allan. Evaluation of BPEL to Scientific Workflows. In *Proc. Sixth IEEE Int. Symp. Cluster Computing and the Grid CCGRID 06*, volume 1, pp. 269–274. 2006. doi:10.1109/CCGRID.2006.44. (Zitiert auf Seite 33)

[Apa] Apache. Apache ODE Architecture. URL http://ode.apache.org/architectural-overview.html. (Zitiert auf Seite 66)

[AS96] G. Alonso, H.-J. Schek. Research Issues in Large Workflow Management Systems. In *In Proceedings of NSF Workshop on Workflow and Process Automation in Information Science*, pp. 126–132. 1996. (Zitiert auf den Seiten 44, 51 und 53)

[BJA+08] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, Y. Simmhan. The Trident Scientific Workflow Workbench. In *Proc. IEEE Fourth Int. Conf. eScience eScience '08*, pp. 317–318. 2008. doi:10.1109/eScience.2008.126. (Zitiert auf Seite 55)

[BKML+10] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, E. W. Sayers. GenBank. *Nucleic Acids Res*, 38(Database issue):D46–D51, 2010. doi:10.1093/nar/gkp1024. URL http://dx.doi.org/10.1093/nar/gkp1024. (Zitiert auf Seite 38)

[bpe] BPEL Project. Eclipse. URL http://www.eclipse.org/bpel. (Zitiert auf Seite 30)

[CEB+09] N. Cipriani, M. Eissele, A. Brodt, M. Grossmann, B. Mitschang. NexusDS: a flexible and extensible middleware for distributed stream processing. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*, IDEAS '09, pp. 152–161. ACM, New York, NY, USA, 2009. doi:http://doi.acm.org/10.1145/1620432.1620448. URL http://doi.acm.org/10.1145/1620432.1620448. (Zitiert auf Seite 135)

[Che07] W.-J. Chen. *DB2 9 pureXML Guide*. IBM, 2007. URL http://www.redbooks.ibm.com/abstracts/sg247315.html. (Zitiert auf den Seiten 23, 24 und 37)

[Chr01] F. Christensen, E. ; Curbera. Web Services Description Language (WSDL) 1.1, 2001. URL http://www.w3.org/TR/wsdl. (Zitiert auf Seite 26)

[Dar05] P. Darugar. Abolish XML namespaces? Technical report, IBM, 2005. URL http://www.ibm.com/developerworks/xml/library/x-abolns.html. (Zitiert auf Seite 96)

[DBG+03] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, (1):25–39, 2003. (Zitiert auf Seite 13)

[FL00] D. R. Frank Leymann. *Production Workflow: Concepts and Techniques*. Prentice Hall International, 2000. (Zitiert auf den Seiten 14, 27 und 32)

[FTGP06] M. Fischer, Q. K. Thai, M. Grieb, J. Pleiss. DWARF–a data warehouse system for analyzing protein families. *BMC Bioinformatics*, 7:495, 2006. doi:10.1186/1471-2105-7-495. URL http://dx.doi.org/10.1186/1471-2105-7-495. (Zitiert auf Seite 38)

[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*, GCE '09, pp. 7:1–7:10. ACM, New York, NY, USA, 2009. doi:http://doi.acm.org/10.1145/1658260.1658270. URL http://doi.acm.org/10.1145/1658260.1658270. (Zitiert auf den Seiten 29 und 33)

[GPW+07] T. Gunarathne, D. Premalal, T. Wijethilake, I. Kumara, A. Kumar. BPEL-Mora: Lightweight Embeddable Extensible BPEL Engine. In M. Calisti, M. Walliser, S. Brantschen, M. Herbstritt, C. Pautasso, C. Bussler, editors, *Emerging Web Services Technology*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pp. 3–20. Birkhaeuser Basel, 2007. URL http://dx.doi.org/10.1007/978-3-7643-8448-7_2. (Zitiert auf Seite 45)

[GSK+11] Görlach, Sonntag, Karastoyanova, Leymann, Reiter. Conventional Workflow Technology for Scientific Simulation. *To appear in: Yang, Y. (ed.); Wang, L. (ed.); Jie, W. (ed.): Guide to e-Science. Springer*, 2011. (Zitiert auf den Seiten 14, 33, 54 und 65)

[HDO10] B. Haasdonk, M. Dihlmann, M. Ohlberger. A Training Set and Multiple Bases Generation Approach for Parametrized Model Reduction Based on Adaptive Grids in Parameter Space. 2010. URL http://www.ians.uni-stuttgart.de/agh/publications/2010/HDO10/. (Zitiert auf den Seiten 120 und 128)

[HHGR06] G. Hackmann, M. Haitjema, C. Gill, G.-C. Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In A. Dan, W. Lamersdorf, editors, *Service-Oriented Computing ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pp. 503–508. Springer Berlin / Heidelberg, 2006. URL http://dx.doi.org/10.1007/11948148_47. (Zitiert auf Seite 45)

[Hol95] D. Hollingsworth. The Workflow Reference Model. 1995. (Zitiert auf den Seiten 7, 43 und 44)

[JMB07] J. L. T. Jeremy M. Berg, Lubert Stryer. *Biochemistry*. Spektrum Verlag, 2007. (Zitiert auf Seite 39)

[JPA] *OpenJPA 2.0*. URL http://openjpa.apache.org/documentation.html. (Zitiert auf Seite 67)

[KBA+] G. King, C. Bauer, M. R. Andersen, E. Bernard, S. Ebersole. *Hibernate Reference Documentation*. URL http://docs.jboss.org/hibernate/core/3.5/reference/en/html/. (Zitiert auf Seite 67)

[Mül10] C. M. Müller. *Development of an Integrated Database Architecture for a Runtime Environment for Simulation Workflows*. Diplomarbeit, Universität Stuttgart, 2010. URL http://elib.uni-stuttgart.de/opus/volltexte/2010/5232/pdf/DIP_2984.pdf. (Zitiert auf den Seiten 54 und 81)

[NG87] D. W. Nebert, F. J. Gonzalez. P450 genes: structure, evolution, and regulation. *Annu Rev Biochem*, 56:945–993, 1987. doi:10.1146/annurev.bi.56.070187.004501. URL http://dx.doi.org/10.1146/annurev.bi.56.070187.004501. (Zitiert auf Seite 40)

[OAF+04] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004. doi:10.1093/bioinformatics/bth361. URL http://dx.doi.org/10.1093/bioinformatics/bth361. (Zitiert auf den Seiten 30 und 55)

[OAS07] OASIS. Web Services Business Process Execution Language Version 2.0, OASIS Standard, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. (Zitiert auf den Seiten 14, 30, 31 und 71)

[OLK+07] T. Oinn, P. Li, D. B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, J. Zhao. Taverna myGrid: Aligning a Workflow System with the Life Sciences Community. In I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, editors, *Workflows for e-Science*, pp. 300–319. Springer London, 2007. URL http://dx.doi.org/10.1007/978-1-84628-757-2_19. (Zitiert auf Seite 30)

[PD99] N. W. Paton, O. Díaz. Active database systems. *ACM Comput. Surv.*, 31:63–103, 1999. doi:http://doi.acm.org/10.1145/311531.311623. URL http://doi.acm.org/10.1145/311531.311623. (Zitiert auf Seite 51)

[RRS+10] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL - A Framework for Accessing External Data in Simulation Workflows. 2010. (Zitiert auf den Seiten 7, 29, 34, 43, 47, 48 und 135)

[sgm86] ISO 8879:1986 Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), 1986. (Zitiert auf Seite 17)

[SKDN05] S. Shankar, A. Kini, D. J. DeWitt, J. Naughton. Integrating databases and workflow systems. *SIGMOD Rec.*, 34:5–11, 2005. doi:http://doi.acm.org/10.1145/1084805.1084808. URL http://doi.acm.org/10.1145/1084805.1084808. (Zitiert auf den Seiten 43 und 51)

[Slo07] A. Slominski. Adapting BPEL to Scientific Workflows. In I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields, editors, *Workflows for e-Science*, pp. 208–226. Springer London, 2007. URL http://dx.doi.org/10.1007/978-1-84628-757-2_14. (Zitiert auf den Seiten 14, 29 und 33)

[SWLP09] D. Sirim, F. Wagner, A. Lisitsa, J. Pleiss. The cytochrome P450 engineering database: Integration of biochemical properties. *BMC Biochem*, 10:27, 2009. doi:10.1186/1471-2091-10-27. URL http://dx.doi.org/10.1186/1471-2091-10-27. (Zitiert auf Seite 38)

[Tay07] G. S. Taylor, Deelman. *Workflows for e-Science*. Springer, 2007. (Zitiert auf Seite 33)

[TH01] E. R. Theo Härder. *Datenbanksysteme - Konzepte und Techniken der Implementierung*. Springer, 2001. (Zitiert auf Seite 36)

[Tho04] H. S. Thompson. XML Schema, 2004. URL http://www.w3.org/XML/Schema. (Zitiert auf den Seiten 17 und 20)

[VSRM08] M. Vrhovnik, H. Schwarz, S. Radeschiitz, B. Mitschang. An Overview of SQL Support in Workflow Products. In *Proc. IEEE 24th Int. Conf. Data Engineering ICDE 2008*, pp. 1287–1296. 2008. doi:10.1109/ICDE.2008.4497538. (Zitiert auf den Seiten 34, 43 und 45)

[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 *In VLDB*, pp. 615–626. 2007. (Zitiert auf den Seiten 7, 34, 47, 49, 51 und 59)

[w3ca] Document Object Model (DOM). URL http://www.w3.org/DOM/. (Zitiert auf Seite 17)

[W3Cb] W3C. Document Type Definition. (Zitiert auf Seite 17)

[W3C99] W3C. XML Path Language (XPath) Version 1.0, 1999. URL http://www.w3.org/TR/xpath/. (Zitiert auf den Seiten 21 und 23)

[W3C07a] W3C. SOAP Version 1.2, 2007. URL http://www.w3.org/TR/soap/. (Zitiert auf Seite 26)

[W3C07b] W3C. XQuery 1.0: An XML Query Language, 2007. URL http://www.w3.org/TR/xquery/. (Zitiert auf Seite 23)

[W3C08] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008. URL http://www.w3.org/TR/2008/REC-xml-20081126/. (Zitiert auf Seite 17)

[W3C09] W3C. XQuery Update Facility 1.0, 2009. URL http://www.w3.org/TR/xquery-update-10/. (Zitiert auf den Seiten 24, 98 und 133)

[Wag10] F. Wagner. Webservice und Workflow-Technologie für Proteinmodellierung, 2010. URL http://elib.uni-stuttgart.de/opus/volltexte/2010/5567/pdf/STUD_2258.pdf. (Zitiert auf den Seiten 7, 11, 30, 38, 39, 40, 46 und 102)

[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 14 und 25)

Alle URLs wurden zuletzt am 19. Februar 2011 geprüft.

Erklärung

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

(Florian Bernd Dominic Wagner)