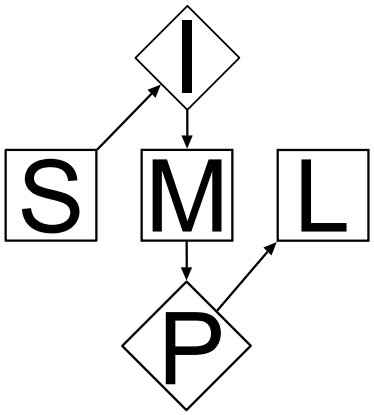
Studienprojekt SIMPL

Spezifikation

Version 1.3

13. November 2009



Inhaltsverzeichnis

Änderungsgeschichte

Version	Datum	Autor	Änderungen
0.1	11.09.2009	hahnml	Erstellung des Dokuments.
0.2	14.09.2009	zoabfs, hahnml, xitu	Kapitel 3 eingefügt.
0.3	17.09.2009	hahnml, zoabisfs	Kapitel 7 eingefügt.
0.4	21.09.2009	hahnml, zoabisfs, rehnre	Kapitel 5 und 6 eingefügt.
1.0	28.09.2009	hahnml, zoabisfs, schneimi, bruededl, huettiwg, rehnre	Abschließende Überarbeitung des Dokuments.
1.1	19.10.2009	hahnml, schneimi, rehnre	Korrektur der Spezifikation nach dem Review mit den Kunden.
1.2	13.11.2009	hahnml	Korrektur der 1.Iteration der Spezifikation.
1.3	13.11.2009	huettiwg	Abschließende Überarbeitung der 1. Iteration der Spezifikation.
1.4	22.01.2010	hahnml	Korrektur der Spezifikation anhand von weiteren Kommentaren der Kunden.

1 Einleitung

In diesem Kapitel wird der Zweck dieses Dokuments sowie der Einsatzbereich und die Ziele der zu entwickelnden Software beschrieben. Weiterhin werden die in diesem Dokument verwendeten Definitionen erläutert und ein Überblick über das restliche Dokument gegeben.

1.1 Zweck des Dokuments

Diese Spezifikation ist die Grundlage für alle weiteren Dokumente, die im Rahmen dieses Projekts entstehen. In ihr sind sämtliche Anforderungen an die zu entwickelnde Software festgelegt. Sie muss stets mit den anderen Dokumenten, insbesondere mit dem Entwurf und der Implementierung, konsistent gehalten werden. Die Spezifikation dient den Team-Mitgliedern als Grundlage und Richtlinie für die Entwicklung der Software und den Kunden als Zwischenergebnis zur Kontrolle.

Zum Leserkreis dieser Spezifikation gehören:

- Die Entwickler der Software,
- die Kunden und
- die Gutachter der Spezifikationsreviews.

1.2 Einsatzbereich und Ziele

Das Entwicklungsteam soll ein erweiterbares und generisches Rahmenwerk für die Modellierung und Ausführung von Workflows erstellen, welches den Zugriff auf nahezu beliebige Datenquellen ermöglichen soll. Bei den Datenquellen kann es sich beispielsweise um Sensornetze, Datenbanken und Dateisysteme handeln. Der Schwerpunkt soll klar auf wissenschaftlichen Workflows liegen, in denen es möglich sein muss, große heterogene Datenmengen verarbeiten zu können. Als Modellierungs- und Ausführungssprache für die hier betrachteten Workflows dient die Business Process Execution Langauge (BPEL, [4]). Über das Rahmenwerk sollen beliebige Datenmanagement-Funktionen in einen BPEL-Prozess eingebunden werden können. Dafür werden bereits vorhandene Konzepte evaluiert, wie z.B. die Sprache BPEL, und falls nötig erweitert und angepasst. Für eine möglichst hohe Flexibilität soll ein dynamischer Ansatz gewählt werden, so dass auch erst zur Laufzeit des Systems die Datenquellen festgelegt werden können. Nichtsdestoweniger sollte auch die Möglichkeit bestehen, die Datenquellen statisch anbinden zu können. Eine Anforderung des Kunden ist, dass eine vorhandene BPEL-Workflow-Engine sowie ein vorhandenes Modellierungstool um diese gewünschten Funktionen erweitert bzw. angepasst werden. Die BPEL-Prozesse sollen mit dem entsprechenden Modellierungstool spezifiziert und mit der BPEL-Workflow-Engine ausgeführt werden können.

1.3 Evolution des Dokuments

Das Projekt SIMPL ist in zwei Iterationsstufen aufgeteilt. Die vorliegende Spezifikation beschreibt die Anforderungen, die in der ersten Iterationsstufe des Rahmenwerks umgesetzt werden sollen und stellt dessen Grundfunktionalität dar. Die Spezifikation und damit die Beschreibung der weitergehenden Funktionalität wird in späteren Iterationen vervollständigt.

Darüber hinaus wird in weiterführenden Kundengesprächen über die Präferenzordnung von optionalen Implementierungen diskutiert, welche in den entsprechenden Iterationen umgesetzt werden sollen. Abschnitt 1.3.1 nennt die Basisfunktionalitäten des Rahmenwerks, die in der ersten Iteration umgesetzt werden. Abschnitt 1.3.2 liefert einen Ausblick auf die Funktionalitäten, die in den weiteren Iterationen umgesetzt werden sollen.

1.3.1 Basisfunktionalität

Folgende Funktionen bilden die Basisfunktionalität des SIMPL-Rahmenwerks:

- Umsetzung eines Plug-In System für die Anbindung von verschiedenen Datenquellen.
- Die statische Anbindung von Datenquellen, wobei vorerst nur relationale Datenbanken unterstützt werden sollen.
- Eine grundlegende Admin-Konsole, die Funktionen wie das An- und Ausschalten des Auditings und die Eingabe globaler Einstellungen für das Rahmenwerk bereitstellt.
- Bereitstellung von generischen BPEL-Aktivitäten im Eclipse BPEL Designer für den Zugriff auf Datenquellen.

1.3.2 Ausblick

In diesem Abschnitt geht es darum, welche Funktionalität in weiteren Iterationen hinzugefügt wird. Die folgende Liste enthält die Punkte, die auf jeden Fall in einer der folgenden Iterationen umgesetzt werden:

- Anbindung von Dateisystemen am Beispiel des CSV-Dateiformats.
- Unterstützung von Referenzen in BPEL (siehe [3]), damit auf Daten auch per Referenz im Workflow zugegriffen werden kann. Dies wird aus Gründen der Performanz benötigt, da die Datenübergabe zwischen Workflow und Web Service standardmäßig per Wert erfolgt, was bei großen Datenmengen (bis zu Gigabytes oder gar Terabytes im wissenschaftlichen Bereich) zu erheblichen Performanzeinbußen führt.
- Bereitstellung einer Datenquellen-Registry, mit Hilfe derer Datenquellen über den Eclipse BPEL Designer (siehe [10]) manuell durch den Benutzer oder dynamisch durch das SIMPL Rahmenwerk ausgewählt werden können.
- Unterstützung einer automatischen Auswahl von Datenquellen zur Laufzeit. Dabei kann der Benutzer Anforderungen an Datenquellen formulieren und eine Strategie auswählen, mit deren Hilfe eine Datenquelle, die seine Anforderungen erfüllt, ausgewählt wird.

Im Folgenden wird beschrieben, was von uns (unter Absprache mit dem Kunden) als optionale Anforderungen definiert wurden. Diese werden umgesetzt, wenn der zeitliche Rahmen des Projekts es zulässt:

- Einstellung der Granularität des Auditings.
- Implementierung eines Monitorings, welches das bestehende Auditing nutzt, um dessen Daten in definierten Abständen auszulesen und dem Benutzer zur Beobachtung und Überwachung von BPEL-Prozessen anzuzeigen.

1.4 Definitionen

Die in der Spezifikation verwendeten Begriffe, Definitionen und Abkürzungen werden in einem separaten Begriffslexikon eindeutig definiert und erklärt. Dadurch werden Missverständnisse innerhalb des Projektteams oder zwischen Projektteam und Kunde vermieden.

Auf alle in Abschnitt 7.1.3 beschriebenen BPEL-Aktivitäten wird in diesem Dokument mit dem Sammelbegriff Data-Management-Aktivität verwiesen. Wird also von einer Data-Management-Aktivität gesprochen, ist indirekt eine oder mehrere dieser Aktivitäten gemeint. Über die Definition und Verwendung dieses Sammelbegriffs soll lediglich die Allgemeingültigkeit der getroffenen Aussagen für jede der in Abschnitt 7.1.3 beschriebenen Aktivitäten erreicht werden. Nachfolgend wird weiterhin für den Begriff Data-Management-Aktivität die Abkürzung DM-Aktivität verwendet.

1.5 Überblick

In diesem Dokument soll die zu entwickelnde Software spezifiziert werden. Dazu werden in Kapitel 2 die spätere Systemumgebung, die Kernfunktionen, die Sprache und weitere Aspekte der Software beschrieben. So erhält der Leser einen Überblick über die Funktionalität der Software und deren Verwendung. Weiterhin werden die Ziele und Aufgaben, die für die Realisierung der Software bestehen, aufgezeigt. Anschließend werden die vom Kunden genannten Anforderungen durch die Kapitel 3, 5, 6 und die vorläufige Benutzeroberfläche in Kapitel 4 aufgezeigt. Dabei werden durch die nichtfunktionalen Anforderungen qualitative (Robustheit, Portabilität, usw.) und quantitative (Mengengerüst) Anforderungen an die Software spezifiziert. Die Anwendungsfälle in Kapitel 6 beschreiben die funktionalen Anforderungen. Es werden also konkret die Funktionen, die die Software enthalten soll und z.T. schon in der Übersicht der Kernfunktionalität in Abschnitt 1.3.1 aufgeführt sind, beschrieben. Im Anschluss folgen in Kapitel 7 die Beschreibungen und Definitionen einiger Konzepte bzw. Ansätze, die zur Umsetzung der gewünschten Funktionalität benötigt werden. Am Ende des Dokuments werden in Kapitel 8 die verwendeten Werkzeuge und Technologien, die für die Erstellung der Software benötigt werden, vorgestellt.

2 Allgemeine Beschreibung

Dieses Kapitel liefert allgemeine Informationen über die zu entwickelnde Software. Dazu gehöhren beispielsweise die Beschreibung der späteren Systemumgebung, die wichtigsten Funktionen, die verwendete Sprache und Informationen über den Benutzerkreis der Software.

2.1 Einbettung in die Systemumgebung

Das SIMPL Rahmenwerk, bestehend aus dem SIMPL Core, den SIMPL Eclipse Plug-Ins und dem SIMPL Extension-Activity Plug-In, soll in die in Abbildung 1 dargestellte Systemumgebung eingebettet werden. Die Systemumgebung besteht dabei aus Eclipse mit dem BPEL Designer Plug-In, einem Web Server, wie dem Apache Tomcat (siehe [7]), und den Datenquellen, auf die zugegriffen wird. Der SIMPL Core und eine Workflow-Engine (Apache Orchestration Director Engine (ODE), siehe [6]) werden auf dem Web Server ausgeführt. SIMPL unterstützt relationale Datenbanken (MySQL, IBM DB2) und kann um weitere Datenquellen, wie XML-Datenbanken (IBM DB2 mit pureXML-Technologie), Dateisysteme oder Sensornetz-Datenbanken (TinyDB) ergänzt werden. Eine Sensornetz-Datenbank (TinyDB) speichert die Daten von meist kabellos vernetzten Sensoren und liefert so eine zentrale und einheitliche Zugriffsmöglichkeit auf Sensordaten. Der SIMPL Core läuft als Web Service auf dem Web Server und liefert u.a. die Funktionalität, die während der Laufzeit von Prozessen mit DM-Aktivitäten benötigt wird. Durch den SIMPL Core werden generell weitgehend alle Funktionalitäten, die von einem beliebigen Ansatz für den Zugriff auf Datenquellen von wissentschaftlichen Workflows benötigt werden, geliefert. Dadurch ergibt sich die Möglichkeit in der Workflow-Engine so wenig wie nötig implementieren zu müssen. Diese Aufteilung ermöglicht es, entsprechende zukünftige Ansätze für den Datenzugriff relativ leicht umzusetzen bzw. bereits entwickelte Ansätze auch relativ leicht in andere Workflow-Laufzeit- und Modellierungsumgebungen zu integrieren. Dieser Umstand erhöht die Portabilität und die Erweiterbarkeit des Rahmenwerks beträchtlich. Die SIMPL Eclipse Plug-Ins bestehen aus drei separaten Plug-Ins, dem SIMPL Core Plug-In, dem BPEL-DM Plug-In und dem SIMPL Core Client Plug-In. Das SIMPL Core Plug-In erweitert die grafische Oberfläche von Eclipse, um Einstellungen für das SIMPL Rahmenwerk vornehmen zu können. Das BPEL-DM Plug-In erweitert das Eclipse BPEL Designer Plug-In, um Prozesse mit DM-Aktivitäten modellieren zu können. Durch diese Trennung ist es möglich, auch nur eines der beiden Plug-Ins in Eclipse einzubinden und z.B. für das jeweils andere Plug-In eigene Implementierungen zu nutzen. Das SIMPL Core Client Plug-In realisiert die Verbindung mit dem SIMPL Core und wird sowohl für das SIMPL Core Plug-In (Laden und Speichern von Rahmenwerks-Einstellungen), wie auch für das BPEL-DM Plug-In (Laden aller vom SIMPL Core unterstützten Datenquellen) benötigt. Die Ausführungslogiken der jeweiligen DM-Aktivitäten werden in der Workflow-Engine implementiert und in diese als Plug-In (SIMPL Extension-Activity Plug-In) eingebunden. Dabei läuft die benötigte Software auf dem lokalen Rechner des Benutzers, die Datenquellen können auf verschiedene Server verteilt sein.

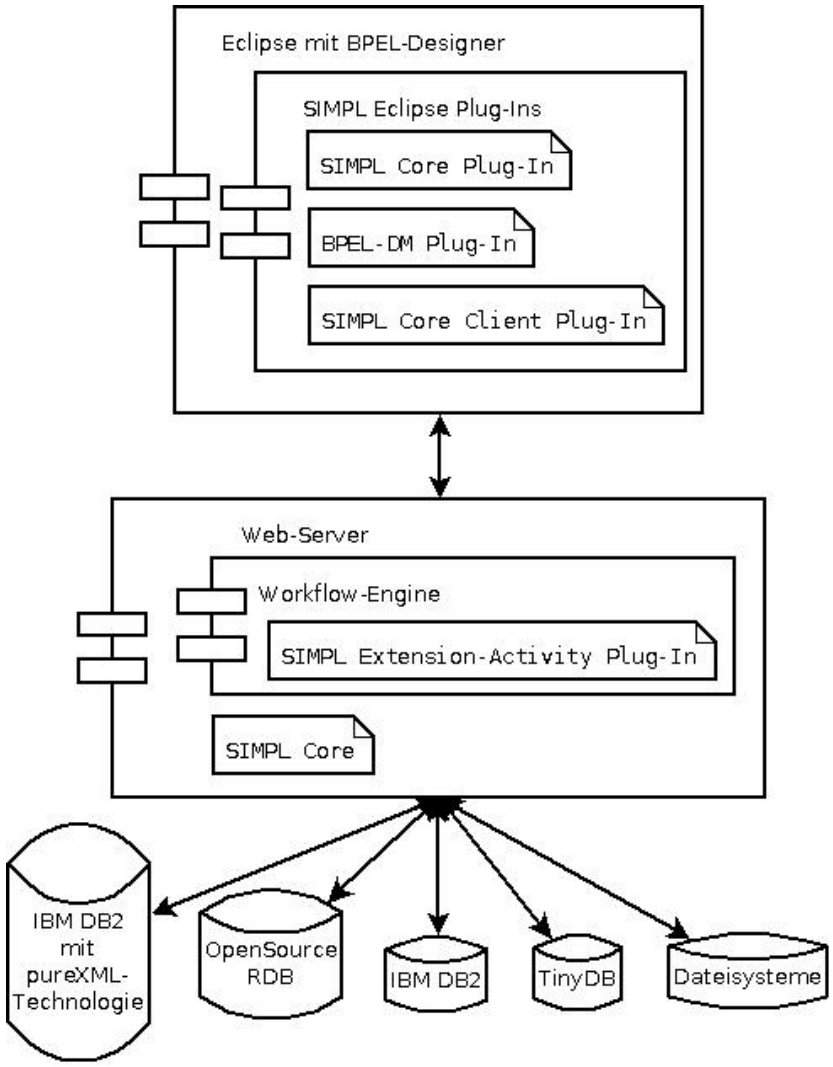


Abbildung 1: Übersicht über die Systemumgebung von SIMPL

2.2 Funktionen

In diesem Abschnitt folgen die wichtigsten Funktionen des Rahmenwerks, die später dessen Kernfunktionalität bilden sollen.

Das Rahmenwerk soll als Eclipse Plug-In verwendet werden und mit der Laufzeitumgebung integriert sein. Es soll die Verarbeitung von großen, heterogenen Datenmengen im Rahmen eines wissenschaftlichen Workflows ermöglichen.

Die vorhandenen BPEL-Aktivitäten des Eclipse BPEL Designers werden dazu um neue Aktivitäten für die Verwaltung von Daten erweitert (DM-Aktivitäten). Mit deren Hilfe wird die Anbindung von Datenquellen in BPEL-Prozessen vereinfacht. In der ersten Iteration werden die Datenquellen über ihre physikalische Adresse angebunden. In einer späteren Iteration wird eine Datenquellen-Registry bereitgestellt. Diese ermöglicht die Realisierung einer grafischen oder automatischen Auswahlmöglichkeit für Datenquellen. Eine nähere Beschreibung der DM-Aktivitäten wird in Abschnitt 7.1 gegeben.

Alle Ereignisse, die während der Laufzeit eines Prozesses auftreten (z.B. Zugriff auf eine Datenquelle), werden durch ein Auditing in einer vom Nutzer definierten Datenbank gespeichert.

Für die Verwaltung des SIMPL-Rahmenwerks und zur Änderung von Einstellungen auch während der Laufzeit von Prozessen wird eine Admin-Konsole implementiert. Über diese kann zum Beispiel das Auditing an und ausgeschaltet und eine Datenbank zur Speicherung der Auditinginformationen angegeben werden.

2.3 Sprache

Generell gilt, dass alle Dokumente auf Deutsch und jeder Quellcode einschließlich Kommentaren auf Englisch verfasst und ausgeliefert werden sollen. Eine Ausnahme bilden das Handbuch und die verschiedenen Dokumentationen der von uns durchgeführten Erweiterungen, wie z.B. die Erweiterungen von Apache ODE oder dem Eclipse BPEL Designer. Diese Dokumente werden auf Deutsch und auf Englisch verfasst, um sie einem breiteren Leserkreis zur Verfügung stellen zu können.

2.4 Distributionsform und Installation

Das Rahmenwerk wird als Teil eines großen Installationpakets ausgeliefert. Dieses Installationspaket besteht aus allen Programmen, die für die Verwendung des Rahmenwerks benötigt werden. Dazu gehört ein Modellierungstool (Eclipse BPEL Designer), ein Web Server (Apache Tomcat), der eine Workflow-Engine ausführen kann, eine Workflow-Engine (Apache ODE) und natürlich das Rahmenwerk selbst. Mithilfe des Installationspakets ist es möglich, viele Einstellungen bereits vorzudefinieren und dem Benutzer die Installation zu erleichern. Das Installationspaket wird dabei als RAR-Archiv zusammen mit allen wichtigen Dokumenten auf einer CD/DVD ausgeliefert. So können nachträgliche Erweiterungen/Korrekturen des Rahmenwerks mithilfe der Dokumentationen leichter realisiert werden. Die Installation der einzelnen Komponenten wird dann anhand der mitgelieferten Installationsanleitung durchgeführt. Nähere Informationen liefert [1].

2.5 Benutzerprofile

Die Benutzer sind im Normalfall Wissenschaftler und Ingenieure. Sie haben meist keine bis wenig Vorkenntnisse in den Bereichen Workflow und Informatik und stellen so entsprechende Anforderungen an die Benutzbarkeit des Rahmenwerks (siehe Kapitel 3).

2.6 Einschränkungen

Für die Erstellung und Verwendung des SIMPL Rahmenwerks gelten die folgenden Einschränkungen:

- Als Workflow-Modellierungssprache dient die Business Process Execution Language (BPEL).
- Die Modellierung von BPEL-Prozessen ist an das Modellierungswerkzeug Eclipse BPEL Designer gebunden.
- Die Ausführung von BPEL-Prozessen ist an die Workflow-Engine Apache ODE gebunden. Das Auditing einer Prozessausführung wird nur für Apache ODE bereitgestellt.
- Als Programmiersprache für das SIMPL Rahmenwerk kommt Java zum Einsatz.

3 Nichtfunktionale Anforderungen

In diesem Kapitel werden die nichtfunktionalen Anforderungen an die zu entwickelnde Software beschrieben. Dafür werden die entsprechenden Software-Qualitäten aufgeführt und ihre Bedeutung für die zu entwickelnde Software erläutert.

3.1 Mengengerüst

Das Mengengerüst beinhaltet alle quantifizierbaren Anforderungen an das Rahmenwerk:

- Für die Speicherung der Auditing-Daten kann nur eine Datenbank gleichzeitig ausgewählt sein.
- Alle laufenden Prozesse einer Workflow-Engine können zu einem Zeitpunkt gemeinsam nur Daten in Höhe des Speichers, der durch das Betriebssystem zur Verfügung gestellt wird, halten.

3.2 Benutzbarkeit

Die Benutzbarkeit soll sich vor allem an Nutzer mit wenig Kenntnissen im Umgang mit Workflows und BPEL richten und die dafür größtmögliche Transparenz liefern. Das bedeutet, dass die interne Prozesslogik der Software bestmöglichst vom Benutzer abgeschirmt wird. Dadurch erhält der Benutzer eine einfache und schnell verständliche Schnittstelle zur Software.

3.3 Verfügbarkeit

Die Verfügbarkeit des Rahmenwerks soll mindestens so hoch sein, dass sie der Verfügbarkeit der späteren Systemumgebung entspricht oder diese übersteigt, damit durch die Verwendung des Rahmenwerks keine zusätzlichen Ausfallzeiten entstehen.

3.4 Robustheit

Unter Robustheit ist hier zu verstehen, dass selbst wenn es zu ungünstigen Bedingungen kommt, SIMPL weiterhin weitgehend fehlerfrei verwendet werden kann. Ungünstige Bedingungen sind dabei z.B. der Ausfall des Servers oder Probleme bei der Verbindung mit Datenquellen. Dazu sollen Prozesse fehlerfrei ausgeführt werden und entsprechend korrekte Ergebnisse liefern oder das Rahmenwerk sicher beendet werden können. Nach dem Neustart des Rahmenwerks müssen evtl. auch die Prozessinstanzen, die zum Zeitpunkt des Fehlers ausgeführt wurden, neu gestartet werden, falls kein Recovery möglich ist.

Benutzereingaben werden nicht vom Rahmenwerk überprüft. Fehlerhafte Eingaben resultieren während dem Deployment bzw. dem Prozessablauf jedoch immer in stabilen Zuständen, die über entsprechende Fehlermeldungen dem Benutzer mitgeteilt werden. Anhand der Fehlermeldungen kann der Benutzer seine Eingaben korrigieren und den Prozess neu deployen bzw. ausführen.

3.5 Sicherheit

Da das Rahmenwerk nur lokal ausgeführt wird und alle lokalen Benutzer momentan die gleichen Rechte besitzen, wird vorerst auf Authentifizierungs- und Autorisierungsmaßnahmen für den Zugriff auf das Rahmenwerk verzichtet. Um eine spätere Realisierung zu vereinfachen, werden bereits jetzt die Rollen Prozess-Modellierer und Workflow-Administrator (siehe Kapitel 5) definiert. Die Authentifizierung und Autorisierung bei Datenquellen wird in Abschnitt 7.2 beschrieben.

3.6 Portabilität

Die Portabilität des SIMPL Eclipse Plug-Ins ist durch die Integration in Eclipse gewährleistet. Der SIMPL Core wird dahingehend implementiert, dass er in allen Java unterstützenden Web Containern lauffähig ist.

3.7 Erweiterbarkeit

Die Erweiterbarkeit des Systems ist eine zentrale Anforderung, da es über einen langen Zeitraum genutzt und in Zukunft um die Anbindung weiterer Datenquellen, Konzepte für den Datenzugriff und den Umgang mit weiteren Datenformaten ergänzt werden soll. Um die Erweiterbarkeit des Systems zu gewährleisten, werden ein modularer Aufbau zugrunde gelegt und entsprechende Schnittstellen geschaffen. Beispiele für den modularen Aufbau des Rahmenwerks sind:

- Die Aufteilung des Rahmenwerks in SIMPL-Core, erweiterter Workflow-Engine und erweitertes Eclipse Plug-In (siehe Abschnitt 2.1).
- Der SIMPL-Core an sich wird modular aufgebaut sein, damit er selbst erweitert werden kann.

3.8 Wartbarkeit

Durch eine qualitativ hochwertige Dokumentation und ein strukturiertes, geplantes und sauberes Entwicklungsvorgehen soll eine hohe Wartbarkeit erreicht werden. Dazu werden alle Dokumente entsprechend gepflegt und laufend aktualisiert. Weiterhin werden nach jedem Entwicklungsintervall Tests durchgeführt und deren Ergebnisse protokolliert.

3.9 Skalierbarkeit

Die Skalierbarkeit des Systems muss eine sehr flexible Infrastruktur erlauben, da die Computersysteme, auf denen SIMPL später ausgeführt wird, in ihrer Leistung sehr weit auseinander gehen können, d.h. vom normalen Desktop-Computer bis zum Supercomputer kann und soll alles möglich sein. Weiterhin soll die Skalierbarkeit des Rahmenwerks garantieren, dass für die Verarbeitung von größer werdenden Datenmengen die benötigten Ressourcen höchstens in der gleichen Größenordnung steigen.

4 Benutzeroberfläche des SIMPL Eclipse Plug-Ins

In diesem Kapitel wird die grafische Benutzeroberfläche des SIMPL Eclipse Plug-Ins beschrieben. Abbildung 2 zeigt den erweiterten Eclipse BPEL Designer. In der Palette befinden sich die SIMPL DM-Aktivitäten, die wie bereits vorhandene Aktivitäten zur Modellierung von Prozessen verwendet werden können. Für SIMPL wird ein Menü bereitgestellt, über das alle wichtigen Einstellungen und Informationen des SIMPL Rahmenwerks vorgenommen bzw. angezeigt werden können.

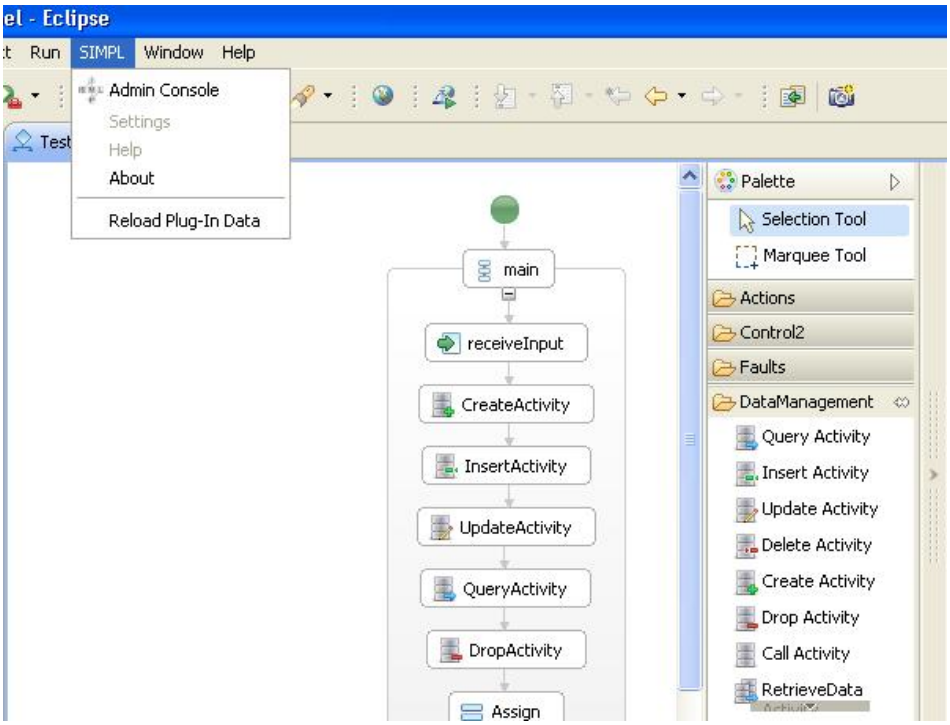


Abbildung 2: SIMPL Menü und Eclipse BPEL Designer mit einigen DM-Aktivitäten

4.1 Admin-Konsole

Die Admin-Konsole kann über das SIMPL Menü geöffnet werden und bietet die Möglichkeit, Einstellungen für das Rahmenwerk vorzunehmen. Dazu gehört beispielsweise die Angabe von globalen Einstellungen und die Verwaltung des Auditings (siehe Abbildung 3).

Folgende Schaltflächen stehen durchgängig zur Verfügung:

- [Default]: Laden der Standard-Einstellungen
- [Save]: Speichern aller durchgeführten Änderungen
- [Reset]: Zurücksetzen aller durchgeführten Änderungen auf den letzten Speicherstand
- [Close]: Schließen der Admin-Konsole und Verwerfen aller Änderungen

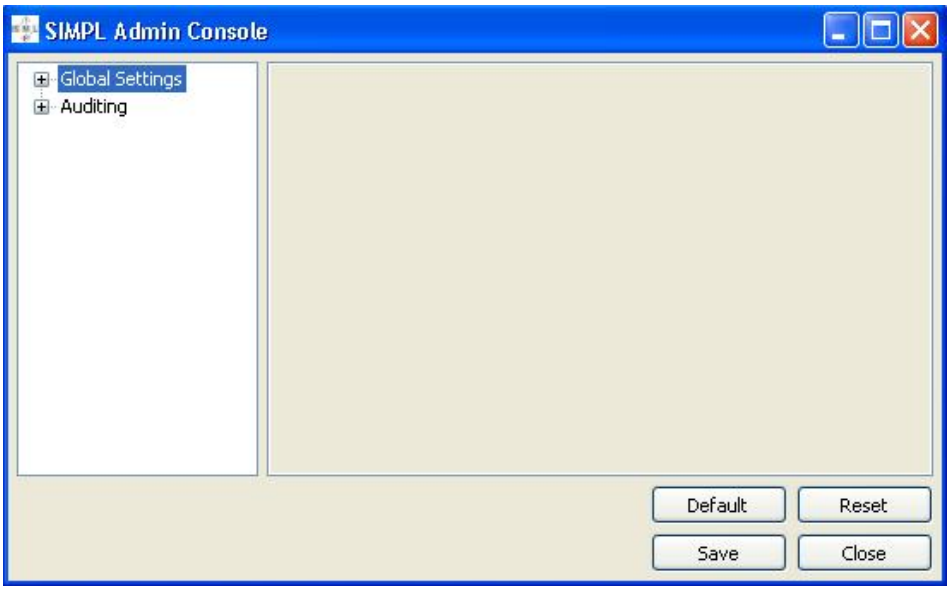


Abbildung 3: Admin-Konsole

4.1.1 Globale Einstellungen

In Abbildung 4 wird der Unterpunkt "Global Settings" der Admin-Konsole gezeigt. Hier können Standardwerte für die Authentifizierung bei Datenquellen angegeben werden.

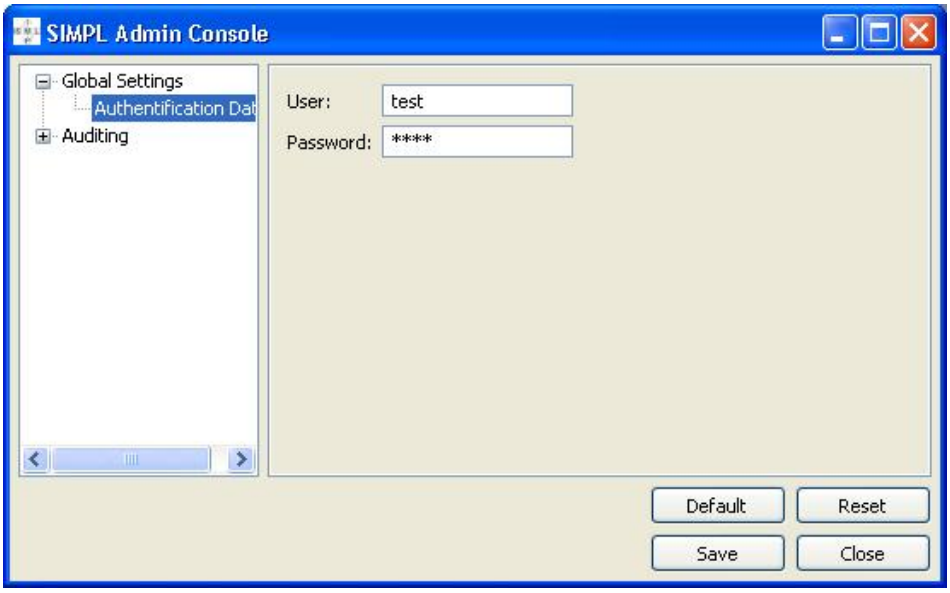


Abbildung 4: Dialog der globalen Eigenschaften

4.1.2 Auditing

In Abbildung 5 wird der Unterpunkt "Auditing" der Admin-Konsole gezeigt. Hier kann das Auditing an- und abgeschaltet werden und eine Datenbank für das Auditing angegeben werden.

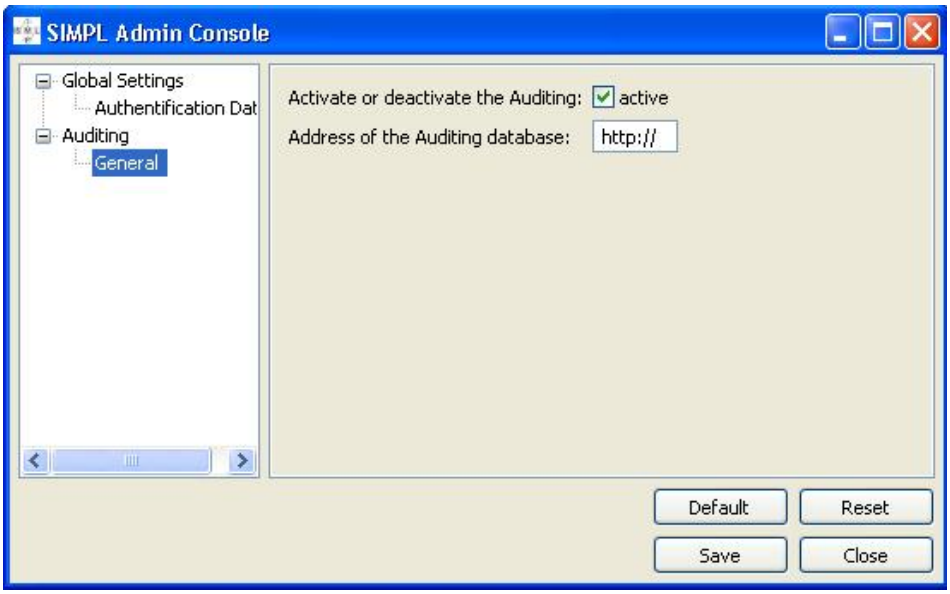


Abbildung 5: Dialog für die Einstellungen des Auditings

4.2 Data-Management-Aktivitäten

In Abbildung 6 wird die "PropertyView" am Beispiel einer Query-Aktivität (siehe Abschnitt 7.1.3) gezeigt. Hier kann die im Prozess ausgewählte Aktivität parametrisiert werden. Das bedeutet, dass die Aktivität hier mit Inhalt gefüllt wird, wie z.B. der Zieldatenquelle oder dem Befehl, der auf dieser ausgeführt werden soll. Dazu wird die Art der Datenquelle ausgewählt, ein Befehl über entsprechende grafische Elemente erstellt und die physikalische Adresse der Datenquelle angegeben. Falls die Checkbox "Show resulting statement" gesetzt ist, wird der durch die grafischen Elemente definierte Befehl im Textfeld "Resulting statement" angezeigt. Im Textfeld wird dabei der Befehl in seiner vollen Länge angezeigt, genau so wie er auf der Datenquelle ausgeführt wird.

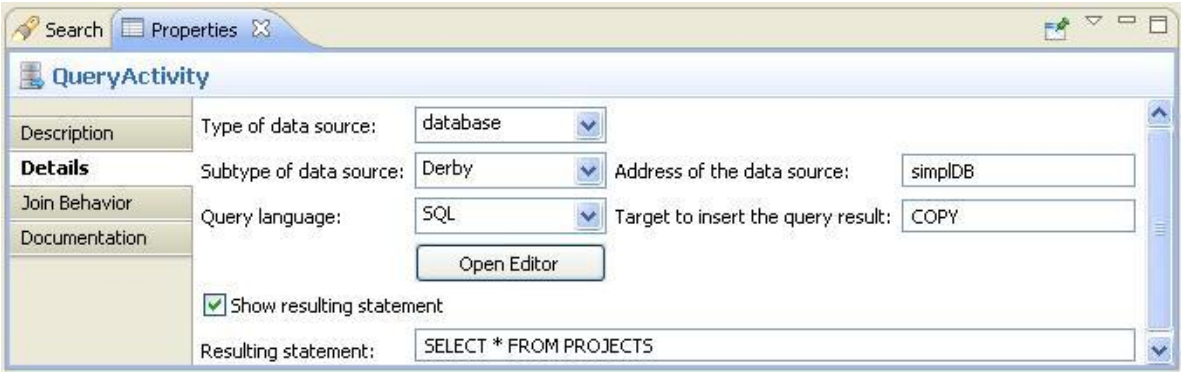


Abbildung 6: Eigenschaftsfenster einer DM-Aktivität am Beispiel einer Query Activity

5 Akteure

In diesem Kapitel werden die einzelnen Akteure der Software beschrieben und ihre Abhängigkeiten untereinander definiert.

5.1 Prozess-Modellierer

Ein Prozess-Modellierer besitzt Fachwissen (z.B. aus der Biologie), das er bei der Modellierung eines wissenschaftlichen Workflows verwendet. Zur Modellierung der Workflows nutzt er den Eclipse BPEL Designer. Dazu kann er beispielsweise BPEL-Aktivitäten erstellen, bearbeiten und auch löschen. Hat der Prozess-Modellierer den BPEL-Prozess fertig modelliert, kann er diesen im Anschluss auf einer Workflow-Engine deployen und danach ausführen lassen.

5.2 Workflow-Administrator

Ein Workflow-Administrator ist eine Spezialisierung des Prozess-Modellierers. D.h. er kann alle Anwendungsfälle des Prozess-Modellierers und noch weitere administrative Anwendungsfälle ausführen. Seine Kenntnisse liegen eher im technischen Bereich, wie z.B. bei der Konfiguration des Rahmenwerks. Er kann beispielsweise über die Admin-Konsole während der Prozesslaufzeit das Auditing an- und abschalten. Ebenso legt er die Datenbank für das Speichern der Auditing-Daten fest.

5.3 ODE Workflow-Engine

Die ODE Workflow-Engine ist ein durch Software realisierter Akteur. Sie führt interne Anwendungsfälle aus, die von einem Benutzer durch andere Anwendungsfälle indirekt aufgerufen werden. Durch die Erweiterungen kann sie DM-Aktivitäten ausführen und zurücksetzen.

5.4 Eclipse BPEL Designer

Der Eclipse BPEL Designer ist ebenfalls ein durch Software realisierter Akteur. Er führt interne Anwendungsfälle aus, die von einem Benutzer durch andere Anwendungsfälle indirekt aufgerufen werden. Der Eclipse BPEL Designer sorgt für das Laden der Einstellungen der Admin-Konsole und das Speichern dieser nach Änderungen.

6 Anwendungsfälle (Use-Cases)

Dieses Kapitel beschreibt die funktionalen Anforderungen an die Software. Dazu werden alle Anwendungsfälle eines jeden Akteurs, die durch das SIMPL Rahmenwerk neu hinzukommen, beschrieben und deren Zusammenhänge in entsprechenden Diagrammen graphisch dargestellt. D.h. das bereits durch vorhandene Software (z.B. Eclipse BPEL Designer) realisierte Anwendungsfälle nicht aufgeführt und beschrieben werden. Für die persistente Speicherung der Admin-Konsolen Einstellungen wird vorerst eine Apache Derby Datenbank verwendet, die im nachfolgenden als SIMPL DB bezeichnet wird.

6.1 Diagramm der Anwendungsfälle

Abbildung 7 zeigt das Diagramm aller Anwendungsfälle der gesamten Software. Dadurch sollen die Funktionalität und die Akteure des späteren Gesamtsystems sichtbar werden. Die einzelnen Anwendungsfälle der verschiedenen Akteure werden in den folgenden Abschnitten näher beschrieben.

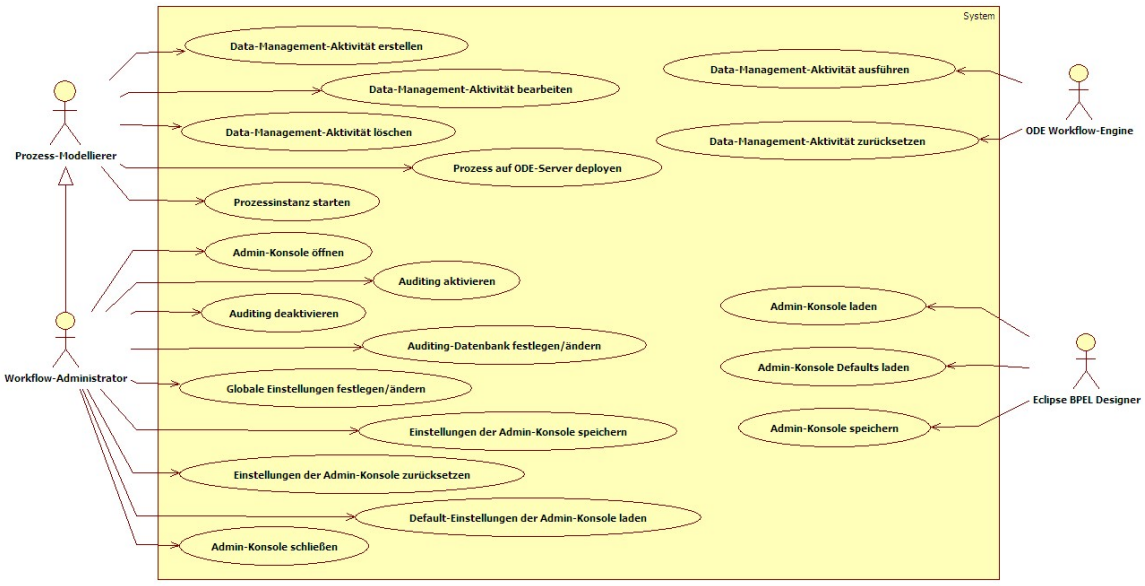


Abbildung 7: Anwendungsfall-Diagramm des gesamten Softwaresystems

6.2 Anwendungsfälle des Prozess-Modellierers

Ein Prozess-Modellierer kann folgende neue Anwendungsfälle (siehe Abbildung 8) ausführen:

- Data-Management-Aktivität erstellen
- Data-Management-Aktivität bearbeiten
- Data-Management-Aktivität löschen
- Prozess auf ODE-Server deployen
- Prozessinstanz starten

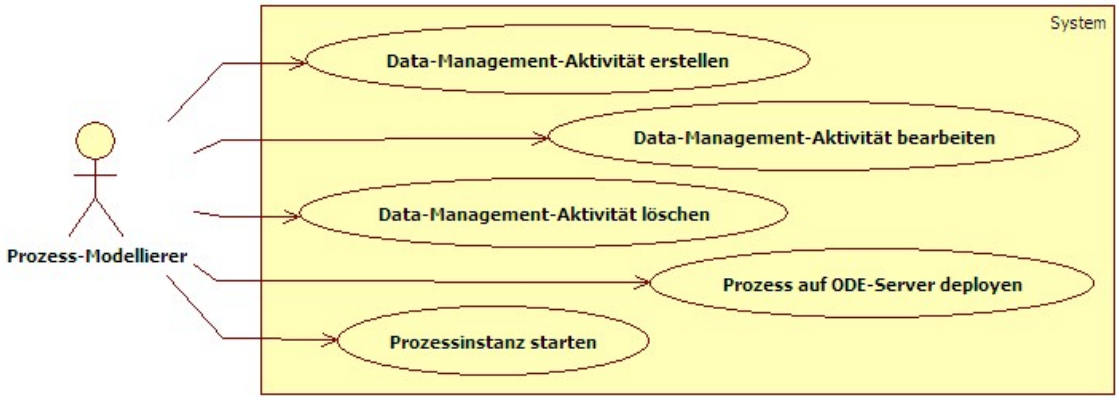


Abbildung 8: Anwendungsfall-Diagramm für den Prozess-Modellierer

6.2.1 Data-Management-Aktivität erstellen

Ziel	Erstellung einer neuen DM-Aktivität.
Vorbedingung	Ein vorhandener BPEL-Prozess ist im Eclipse BPEL Designer geöffnet und die BPEL Designer-Palette wird angezeigt.
Nachbedingung	Die erstellte DM-Aktivität wurde an der selektierten Position in den Prozess eingefügt, und der vom Benutzer eingegebene Name wird angezeigt.
Nachbedingung im Sonderfall	Die erstellte DM-Aktivität wurde an der selektierten Position in den Prozess eingefügt, und der vom Eclipse BPEL Designer vorgeschlagene Name wird angezeigt.
Normalablauf	1a. Selektion einer DM-Aktivität aus der BPEL Designer Palette durch Auswahl mit der linken Maustaste und anschließend Selektion der Stelle des Prozesses, an der die ausgewählte DM-Aktivität eingefügt werden soll, mit der linken Maustaste. 1b. (alternativ) Drag&Drop einer DM-Aktivität aus der Palette an die gewünschte Stelle im Prozess. 2. Eingabe eines Aktivitätsnamens durch den Benutzer.
Sonderfälle	2a. Der angezeigte Namensvorschlag wird vom Benutzer bestätigt.

6.2.2 Data-Management-Aktivität bearbeiten

Ziel	Bearbeitung der Eigenschaften einer vorhandenen DM-Aktivität.
Vorbedingung	Ein vorhandener BPEL-Prozess mit mindestens einer DM-Aktivität ist im Eclipse BPEL Designer geöffnet.
Nachbedingung	Alle durchgeführten Änderungen der Eigenschaften der ausgewählten DM-Aktivität wurden korrekt übernommen und werden in der "Properties-View" angezeigt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Selektierung einer DM-Aktivität aus dem Prozess durch Auswahl mit der linken Maustaste. 2. Öffnen der "Properties-View" der DM-Aktivität um ihre Eigenschaften anzuzeigen. 3. Änderung der Eigenschaften der DM-Aktivität.
Sonderfälle	keine

6.2.3 Data-Management-Aktivität löschen

Ziel	Löschen einer DM-Aktivität.
Vorbedingung	Ein vorhandener BPEL-Prozess mit mindestens einer DM-Aktivität ist im Eclipse BPEL Designer geöffnet.
Nachbedingung	Die ausgewählte DM-Aktivität wurde vollständig und korrekt aus dem Prozess gelöscht. Alle ausgehenden und eingehenden Links werden entsprechend dem Standardverhalten des Eclipse BPEL Designers neu gesetzt.
Nachbedingung im Sonderfall	keine
Normalablauf	1a. Selektierung einer DM-Aktivität aus dem Prozess durch Auswahl mit der linken Maustaste. 1b. Öffnen des Kontextmenüs durch rechten Mausklick auf eine DM-Aktivität. 2a. Löschen der DM-Aktivität durch drücken der "entf"-Taste. 2b. Mausklick auf den Menüpunkt "Delete" des Kontextmenüs.
Sonderfälle	keine

6.2.4 Prozess auf ODE-Server deployen

Ziel	Deployen eines Prozesses auf der Apache ODE Workflow-Engine.
Vorbedingung	Ein Prozess ist im Eclipse BPEL Designer geöffnet, die Apache ODE Workflow-Engine korrekt in Eclipse eingebunden und die Server-View wird angezeigt.
Nachbedingung	Die Prozess-Dateien wurden auf die Apache ODE Workflow-Engine kopiert und der Prozess wurde erfolgreich deployed.
Nachbedingung im Sonderfall	3a. Der ODE-Server ist nicht gestartet und der Prozess wurde nicht deployed. 3b. Der ODE-Server ist gestartet, aber der Prozess wurde nicht deployed.
Normalablauf	1. Erstellung eines ODE Deployment-Deskriptors über File->New->Other->BPEL2.0->Apache ODE Deployment Descriptor. 2. Hinzufügen des Prozesses zum ODE-Server in der Eclipse Server-View: • Rechter Mausklick auf den ODE-Server • Auswahl des Menüpunkts "Add and Remove" • Hinzufügen der BPEL Prozess-Datei 3. Starten des ODE-Servers über rechten Mausklick und klicken auf "Start".
Sonderfälle	3a. ODE-Server startet nicht aufgrund eines Fehlers. 3b. Beim Deployen des Prozesses tritt ein Fehler auf.

6.2.5 Prozessinstanz starten

Ziel	Start einer Prozessinstanz eines Prozessmodells auf der Apache ODE Workflow-Engine.
Vorbedingung	Das Prozessmodell wurde erfolgreich auf der Apache ODE Workflow-Engine deployt (siehe Anwendungsfall "6.2.4 Prozess auf ODE-Server deployen").
Nachbedingung	Prozessinstanz wurde gestartet.
Nachbedingung im Sonderfall	Die Prozessinstanz wird verworfen.
Normalablauf	1. Instanzierung des Prozessmodells und Start der Prozessinstanz durch Senden einer entsprechenden SOAP-Nachricht an den Endpunkt des Prozesses.
Sonderfälle	1a. Fehler beim Starten der Prozessinstanz, z.B. durch eine SOAP-Nachricht mit falschem Inhalt.

6.3 Anwendungsfälle des Workflow-Administrators

Ein Workflow-Administrator kann folgende neue Anwendungsfälle (siehe Abbildung 9) ausführen:

- Admin-Konsole öffnen
- Auditing aktivieren
- Auditing deaktivieren
- Auditing-Datenbank festlegen/ändern

- Globale Einstellungen festlegen/ändern
- Einstellungen der Admin-Konsole speichern
- Einstellungen der Admin-Konsole zurücksetzen
- Default-Einstellungen der Admin-Konsole laden
- Admin-Konsole schließen

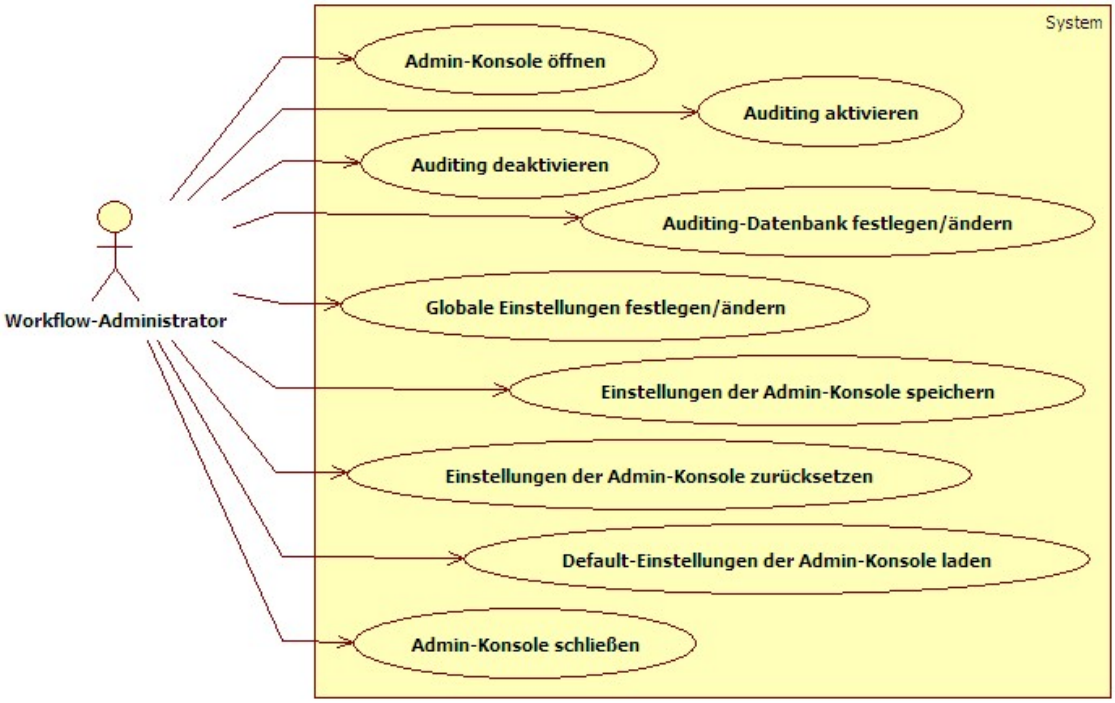


Abbildung 9: Anwendungsfall-Diagramm für den Workflow-Administrator

6.3.1 Admin-Konsole öffnen

Ziel	Öffnen der Admin-Konsole.
Vorbedingung	Eclipse mit dem SIMPL Core Plug-In und dem SIMPL Core Client Plug-In ist geöffnet.
Nachbedingung	Wenn der angestoßene Anwendungsfall erfolgreich ausgeführt wurde, wird die Admin-Konsole mit einer Liste aller Menüpunkte (Auditing, Global Settings) angezeigt und kann verwendet werden.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Klick auf das SIMPL Menü in der Menüleiste. 2. Klick auf den Menüeintrag [Admin Console]. 3. Anstoßen des Anwendungsfalls “6.5.1 Admin-Konsole laden” des Eclipse BPEL Designers.
Sonderfälle	keine

6.3.2 Auditing aktivieren

Ziel	Auditing von SIMPL aktivieren.
Vorbedingung	Die Admin-Konsole ist geöffnet, der Unterpunkt “Auditing” wird angezeigt und das Auditing-Häkchen ist nicht gesetzt.
Nachbedingung	Das Auditing-Häkchen ist gesetzt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Setzen des Auditing-Häkchens.
Sonderfälle	keine

6.3.3 Auditing deaktivieren

Ziel	Auditing von SIMPL deaktivieren.
Vorbedingung	Die Admin-Konsole ist geöffnet, der Unterpunkt “Auditing” wird angezeigt und das Auditing-Häkchen ist gesetzt.
Nachbedingung	Das Auditing-Häkchen ist nicht gesetzt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Zurücksetzen des Auditing-Häkchens.
Sonderfälle	keine

6.3.4 Auditing-Datenbank festlegen/ändern

Ziel	Festlegung/Änderung einer Datenbank für das Speichern der Auditing-Informationen.
Vorbedingung	Die Admin-Konsole ist geöffnet und der Unterpunkt “Auditing” wird angezeigt. Das Auditing “Häkchen” ist gesetzt.
Vorbedingung im Sonderfall	Die Admin-Konsole ist geöffnet und der Unterpunkt “Auditing” wird angezeigt. Das Auditing “Häkchen” ist nicht gesetzt.
Nachbedingung	Die festgelegte/geänderte Datenbank für das Auditing wird in der Admin-Konsole angezeigt.
Nachbedingung im Sonderfall	Die festgelegte/geänderte Datenbank für das Auditing wird in der Admin-Konsole angezeigt. Es erscheint ein Hinweis, dass das Auditing nicht aktiv ist, und die Datenbank somit keine Daten erhält.
Normalablauf	1. Angabe der Datenbank-Adresse über einen Unified Resource Locator (URL) (optional Auswahl einer Datenbank über die Datenbank-Registry).

6.3.5 Globale Einstellungen festlegen/ändern

Ziel	Festlegung/Ändern der globalen Einstellungen.
Vorbedingung	Die Admin-Konsole ist geöffnet und der Unterpunkt “Global Settings” wird angezeigt.
Nachbedingung	Die festgelegten/geänderten globalen Einstellungen werden in der Admin-Konsole angezeigt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Eingabe/Änderung der Werte in entsprechenden Textfeldern.
Sonderfälle	keine

6.3.6 Einstellungen der Admin-Konsole speichern

Ziel	Festlegung der in der Admin-Konsole getätigten/geänderten Einstellungen und deren Speicherung in der SIMPL DB.
Vorbedingung	Die Admin-Konsole wird angezeigt, und es wurde mindestens ein Wert geändert.
Nachbedingung	Wenn der angestoßene Anwendungsfall erfolgreich ausgeführt wurde, wurden alle Werte der Admin-Konsole korrekt gespeichert und alle veralteten Werte mit den Neuen überschrieben. Die Festlegungen/ Änderungen wurden in den Einstellungen entsprechend übernommen.
Nachbedingung im Sonderfall	1a.&1b. Der Benutzer erhält eine Fehlermeldung, die ihn über den entsprechenden Fehler informiert. Die Änderungen werden verworfen und so die Werte auf den letzten Speicherstand zurückgesetzt.
Normalablauf	1. Klick auf den Button [Save]. 2. Anwendungsfall "6.5.3 Admin-Konsole speichern" des Eclipse BPEL Designers wird angestoßen.
Sonderfälle	1a. Das Auditing-Häkchen ist gesetzt, und vom Benutzer wurde keine Auditing-Datenbank zum Speichern der Auditing-Daten festgelegt. 1b. Das Auditing-Häkchen ist gesetzt und eine Auditing-Datenbank angegeben. Die angegebene Auditing-Datenbank kann aber nicht verwendet werden, da sie z.B. nicht erreichbar ist oder die Authentifizierung fehlgeschlagen ist.

6.3.7 Einstellungen der Admin-Konsole zurücksetzen

Ziel	Zurücksetzen des Inhalts der Admin-Konsole auf die zuletzt gespeicherten Werte.
Vorbedingung	Die Admin-Konsole wird angezeigt, und es wurde mindestens ein Wert geändert.
Nachbedingung	Wenn der angestoßene Anwendungsfall erfolgreich ausgeführt wurde, wurden alle geänderten Werte der Admin-Konsole auf die zuletzt gespeicherten Einstellungen zurückgesetzt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Klick auf den Button [Reset]. 2. Anstoßen des Anwendungsfalls "6.5.1 Admin-Konsole laden" des Eclipse BPEL Designers.
Sonderfälle	keine

6.3.8 Default-Einstellungen der Admin-Konsole laden

Ziel	Laden der Standardwerte in der Admin-Konsole.
Vorbedingung	Die Admin-Konsole wird angezeigt.
Nachbedingung	Wenn der angestoßene Anwendungsfall erfolgreich ausgeführt wurde, wurden alle Werte der Admin-Konsole auf die gespeicherten Standardwerte zurückgesetzt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Klick auf den Button [Default]. 2. Anstoßen des Anwendungsfalls "6.5.2 Admin-Konsole Defaults laden" des Eclipse BPEL Designers.
Sonderfälle	keine

6.3.9 Admin-Konsole schließen

Ziel	Schließen der Admin-Konsole.
Vorbedingung	Die Admin-Konsole wird angezeigt. Es wurden keine Änderungen in der Admin-Konsole seit dem letzten Speichervorgang durchgeführt.
Vorbedingung im Sonderfall	Die Admin-Konsole wird angezeigt. Es wurde mindestens ein Wert in der Admin-Konsole seit dem letzten Speichervorgang geändert.
Nachbedingung	1a. Die Admin-Konsole wurde geschlossen.
Nachbedingung im Sonderfall	3b1. Der Anwendungsfall "6.5.3 Admin-Konsole speichern" des Eclipse BPEL Designers wird angestoßen. Wenn der angestoßene Anwendungsfall erfolgreich ausgeführt wurde, wurden alle Festlegungen/Änderungen in den Einstellungen entsprechend übernommen und gespeichert. 3b2. Die Admin-Konsole wurde geschlossen und alle Änderungen wurden verworfen. 3b3. Die Admin-Konsole wird weiterhin angezeigt und alle geänderten Werte bleiben erhalten.
Normalablauf	1a. Klick auf den Button [Close]. 1b. Klick auf den Button [Close]. 2b. Es öffnet sich ein Dialog mit einer Sicherheitsabfrage, ob der Benutzer die durchgeführten Änderungen speichern möchte. 3b1. Klick auf den Button [Yes] des Dialogfensters. 3b2. Klick auf den Button [No] des Dialogfensters. 3b3. Klick auf den Button [Cancel] des Dialogfensters.

6.4 Anwendungsfälle der ODE Workflow-Engine

Die ODE Workflow-Engine kann durch unsere Erweiterungen folgende Anwendungsfälle (siehe Abbildung 10) ausführen:

- Data-Management-Aktivität ausführen
- Data-Management-Aktivität zurücksetzen

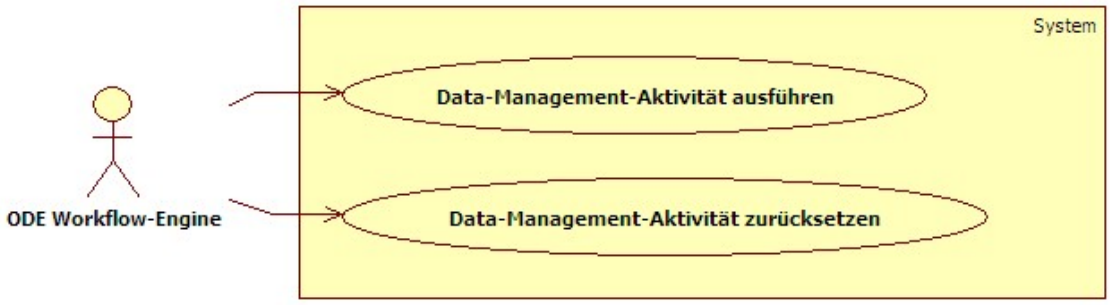


Abbildung 10: Anwendungsfall-Diagramm für die ODE Workflow-Engine

6.4.1 Data-Management-Aktivität ausführen

Ziel	Ausführen einer DM-Aktivität.
Vorbedingung	Es wurde ein Prozess, der mindestens eine DM-Aktivität enthält, deployed und es wurde eine Instanz des Prozesses erzeugt. Eine DM-Aktivität wurde über den Kontrollfluss des Prozesses erreicht.
Nachbedingung	Die DM-Aktivität und die darin enthaltene Datenmanagementoperation wurden erfolgreich ausgeführt.
Nachbedingung im Sonderfall	1a. Die Aktivität befindet sich im Endzustand “Terminated”. 1b. Die Aktivität befindet sich im Zustand “Waiting”. 2a. Die Aktivität befindet sich im Endzustand “Terminated”. 3a. Die Aktivität befindet sich im Endzustand “Faulted”. 4a. Die Aktivität befindet sich im Endzustand “Terminated”.
Normalablauf	1. Die DM-Aktivität befindet sich im Zustand “Ready”. 2. Die Ausführung der DM-Aktivität wird gestartet. 3. Senden der in der DM-Aktivität enthaltenen DM-Operationen zur Ausführung an die Datenquelle. 4. Die DM-Operation wurde in der Datenquelle erfolgreich ausgeführt, und die Ausführung der DM-Aktivität ist damit beendet. Die DM-Aktivität befindet sich im Zustand "Waiting". 5. Es wird ein “Complete_Activity Event” an die DM-Aktivität geschickt. 6. Die DM-Aktivität wird als “Complete” gekennzeichnet. 7. Die nächste(n) Aktivität(en) im Kontrollfluss wird(/werden) initialisiert.
Sonderfälle	1a.1. In der Vateraktivität tritt ein Fehler auf. 1a.2. Es wird ein “Terminate_Activity Event” ausgelöst und die Aktivität wird beendet. 1a.3. Der Use Case “6.4.2 Data-Management-Aktivität zurücksetzen” wird angestoßen. 1b.1. Es wird ein “Complete_Acitivity Event” an die DM-Aktivität gesendet. Dieses Event wird geschickt, wenn die Aktivität im Zustand “Ready” blockiert wurde, allerdings kein “Start_Activity” Event gesendet wurde. 1b.2. Es wird ein “Activity_Executed Event” ausgelöst. 2a.1. In der Vateraktivität tritt ein Fehler auf. 2a.2. Es wird ein “Terminate_Activity Event” ausgelöst und die Aktivität wird beendet. 2a.3. Der Use Case “6.4.2 Data-Management-Aktivität zurücksetzen” wird angestoßen. 3a.1. Während der Durchführung der DM-Operationen tritt ein Fehler in der Datenquelle auf (z.B. ein Syntaxfehler, oder die Datenbank ist nicht erreichbar). 3a.2. Es wird ein “Activity_Faulted Event” ausgelöst. 3a.3. Der Use Case “6.4.2 Data-Management-Aktivität zurücksetzen” wird angestoßen. 4a.1. In der Vateraktivität tritt ein Fehler auf. 4a.2. Es wird ein “Terminate_Activity Event” ausgelöst. 4a.3. Der Use Case “6.4.2 Data-Management-Aktivität zurücksetzen” wird angestoßen.

6.4.2 Data-Management-Aktivität zurücksetzen

Ziel	Rückgängig machen einer laufenden DM-Aktivität, so dass der Zustand der Datenquelle vor Ausführung der DM-Aktivität wiederhergestellt wird.
Vorbedingung	Im Normalablauf einer DM-Aktivität tritt ein Fehler auf (siehe Sonderfälle 1a.1, 2a.1, 3a.1 und 4a.1 des Anwendungsfalls "6.4.1 Data-Management-Aktivität ausführen").
Nachbedingung	Der Zustand vor Ausführung der DM-Aktivität wurde erfolgreich wiederhergestellt.
Nachbedingung im Sonderfall	keine
Normalablauf	1. Alle Änderungen in der Datenquelle werden zurückgesetzt.
Sonderfälle	keine

6.5 Anwendungsfälle des Eclipse BPEL Designers

Der Eclipse BPEL Designer kann durch unsere Erweiterungen folgende Anwendungsfälle (siehe Abbildung 11) ausführen:

- Admin-Konsole laden
- Admin-Konsole Defaults laden
- Admin-Konsole speichern

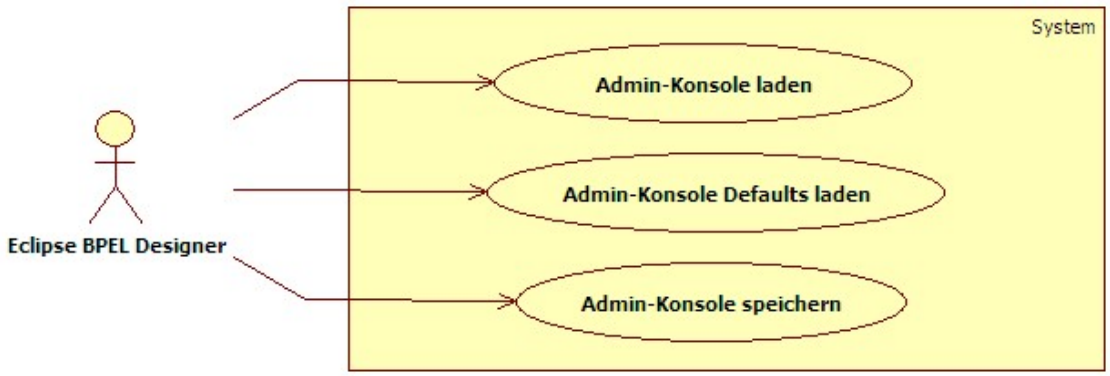


Abbildung 11: Anwendungsfall-Diagramm für den Eclipse BPEL Designer

6.5.1 Admin-Konsole laden

Ziel	Laden der Inhalte der Admin-Konsole.
Vorbedingung	Einer der Anwendungsfälle “Admin-Konsole öffnen” oder “Einstellungen der Admin-Konsole zurücksetzen” wurde ausgeführt.
Nachbedingung	Alle Werte der SIMPL DB mit den aktuellen Einstellungen der Admin-Konsole wurden geladen und werden im Fenster “Admin Console” und dessen Unterfenstern angezeigt.
Nachbedingung im Sonderfall	3a. Der Anwendungsfall “6.5.2 Admin-Konsole Defaults laden” wird angestoßen. Wenn dieser Anwendungsfall erfolgreich ausgeführt wurde, wurden die hinterlegten Default-Einstellungen geladen und werden im Fenster “Admin Console” und dessen Unterfenstern angezeigt. 3b. Die in der Admin-Konsole angezeigten Werte bleiben unverändert.
Normalablauf	1. Laden aller zuletzt gespeicherten Werte aus der SIMPL DB. 2. Füllen der Felder der Admin-Konsole mit den geladenen Werten.
Sonderfälle	1a. Beim Laden der Werte tritt ein Fehler auf, da z.B. die SIMPL DB nicht erreichbar ist. 2a. Es wird eine entsprechende Fehlermeldung angezeigt und der Benutzer wird über ein Dialogfenster gefragt, ob er die in der SIMPL DB hinterlegten Default-Einstellungen laden möchte oder nicht. 3a. Klick auf den Button [Yes]. 3b. Klick auf den Button [No].

6.5.2 Admin-Konsole Defaults laden

Ziel	Laden der Default-Werte der Admin-Konsole.
Vorbedingung	Der Anwendungsfäll “6.5.2 Default-Einstellungen der Admin-Konsole laden” oder der Sonderfall *a* des Anwendungsfalls “6.5.1 Admin-Konsole laden” wurde ausgeführt.
Nachbedingung	Alle Werte der SIMPL DB mit den Default-Einstellungen der Admin-Konsole wurden geladen und werden im Fenster “Admin Console” und dessen Unterfenstern angezeigt.
Nachbedingung im Sonderfall	Im Quellcode hinterlegte Default-Werte werden geladen und im Fenster “Admin Console” und dessen Unterfenstern angezeigt.
Normalablauf	1. Laden aller Default-Werte aus der SIMPL DB. 2. Füllen der Felder der Admin-Konsole mit den geladenen Werten.
Sonderfälle	1a. Beim Laden der Werte tritt ein Fehler auf, da z.B. die SIMPL DB nicht erreichbar ist.

6.5.3 Admin-Konsole speichern

Ziel	Persistente Speicherung der Einstellungen der Admin-Konsole.
Vorbedingung	Der Anwendungsfall "Einstellungen der Admin-Konsole speichern" wurde ausgeführt.
Nachbedingung	Alle Werte der Admin-Konsole wurden korrekt auf der SIMPL DB.
Nachbedingung im Sonderfall	1a. Alle geänderten Werte der Admin-Konsole werden nicht in der SIMPL DB gespeichert. Es wird eine entsprechende Fehlermeldung angezeigt und ein erneuter Speicherversuch kann durchgeführt werden. Die Werte werden solange in der Admin-Konsole angezeigt, bis diese geschlossen wird.
Normalablauf	1. Speichern der Werte auf der SIMPL DB.
Sonderfälle	1a. Beim Speichern der Werte tritt ein Fehler auf, da z.B. die SIMPL DB nicht erreichbar ist.

6.6 Komponenten und ihre Anwendungsfälle

In diesem Abschnitt werden in Tabelle 1 alle Anwendungsfälle den verschiedenen Komponenten des SIMPL Rahmenwerks, die an der Ausführung der entsprechenden Funktionalität beteiligt sind, zugeordnet. Die mit "x" markierte Komponente ist dabei an der Ausführung des Anwendungsfalls beteiligt. Es ist durchaus möglich, dass Anwendungsfälle durch mehrere Komponenten ausgeführt werden. Zum Beispiel wird das Auditing in der Admin-Konsole (SIMPL Core Plug-In) aktiviert, die Ausführung des Auditings wird allerdings durch den SIMPL Core realisiert.

Tabelle 1: Komponenten und ihre Anwendungsfälle

Anwendungsfall	**SIMPL Core**	**SIMPL Core Plug-In**	**BPEL-DM Plug-In**	**SIMPL Extension-Activity Plug-In**	**Eclipse BPEL Designer Plug-In**
Data-Management-Aktivität erstellen			x		
Data-Management-Aktivität bearbeiten			x		
Data-Management-Aktivität löschen			x		
Prozess auf ODE-Server deployen					x
Prozessinstanz starten					x
Admin-Konsole öffnen		x			
Auditing aktivieren		x			
Auditing deaktivieren		x			
Auditing-Datenbank festlegen/ändern		x			
Globale Einstellungen festlegen/ändern		x			
Einstellungen der Admin-Konsole speichern		x			
Einstellungen der Admin-Konsole zurücksetzen		x			
Default-Einstellungen der Admin-Konsole laden		x			
Admin-Konsole schließen		x			
Data-Management-Aktivität ausführen				x	
Data-Management-Aktivität zurücksetzen				x	
Admin-Konsole laden	x	x			
Admin-Konsole Defaults laden	x	x			
Admin-Konsole speichern	x	x			

7 Konzepte und Realisierungen

In diesem Kapitel werden alle Konzepte, die für SIMPL benötigt werden, und deren Realisierung erläutert. Dazu zählt z.B. die Beschreibung und Definition der benötigten DM-Aktivitäten, die zur Realisierung einer generischen Unterstützung von verschiedenen Abfragesprachen, wie z.B. der Structured Query Language (SQL) oder der XML Query Language (XQuery), für BPEL benötigt werden. Weiterhin werden Authentifizierung und Autorisierung im Rahmen von SIMPL erläutert. Darüber hinaus wird am Ende dieses Kapitels die Realisierung des Datenquellen-Auditings und das dafür zugrunde liegende Event-Modell beschrieben.

7.1 Data-Management-Aktivitäten

In diesem Abschnitt werden die zu realisierenden Datenmanagement-Patterns zur Umsetzung einer generischen Unterstützung verschiedener Abfragesprachen für BPEL vorgestellt und im weiteren Verlauf deren Realisierung für die verschiedenen Datenquellen, d.h. für Dateisysteme, Datenbanken und Sensornetze, erläutert. Aus den in Abschnitt 7.1.2 identifizierten Funktionen, die für die Umsetzung der Datenmanagement-Patterns benötigt werden, werden dann in Abschnitt 7.1.3 neue benötigte BPEL-Aktivitäten abstrahiert, die dann später die entsprechenden Funktionen beinhalten und diese an die Datenquellen zur Ausführung schicken.

7.1.1 Datenmanagement-Patterns

In diesem Abschnitt werden alle Datenmanagement-Patterns, die zur Realisierung einer generischen Anbindung von verschiedenen Datenquellen benötigt werden, aufgeführt und beschrieben. Abbildung 12 zeigt die oberen Ebenen (0. Ebene bis 2.Ebene) der Klassenhierarchie der verschiedenen Datenmanagement-Patterns. Diese Hierarchie kann bei Bedarf beliebig um weitere Patternklassen oder Patterns ergänzt werden. Vorerst werden allerdings nur die Patterns und Patternklassen beschrieben, die auch umgesetzt werden. Die nachfolgenden Beschreibungen orientieren sich an dieser Hierarchie und erläutern sie gleichzeitig etwas detaillierter. Darüber hinaus werden nachfolgend weitere Pattern-Klassen und Patterns eingeführt, die die Hierarchie ab der 2. Ebene verfeinern und beschreiben. Die Klassenhierarchie wird dabei von oben nach unten durchlaufen und ist aus Gründen der Lesbarkeit auf mehrere Diagramme aufgeteilt, die in den entsprechenden Beschreibungen folgen. Weiterhin wird in den Beschreibungen der Datenmanagement-Patterns auf deren Anwendbarkeit auf verschiedene Datenquellentypen Bezug genommen. Die Datenquellentypen sind dabei relationale Datenbanken, XML-Datenbanken, Dateisysteme und Sensornetz-Datenbanken. Die Beschreibungen im Zusammenhang mit Sensornetz-Datenbanken stützen sich dabei auf die TinyDB [14], die eine Schnittstelle für das Abfragen von Sensordaten mit einer SQL-ähnlichen Sprache bereitstellt.

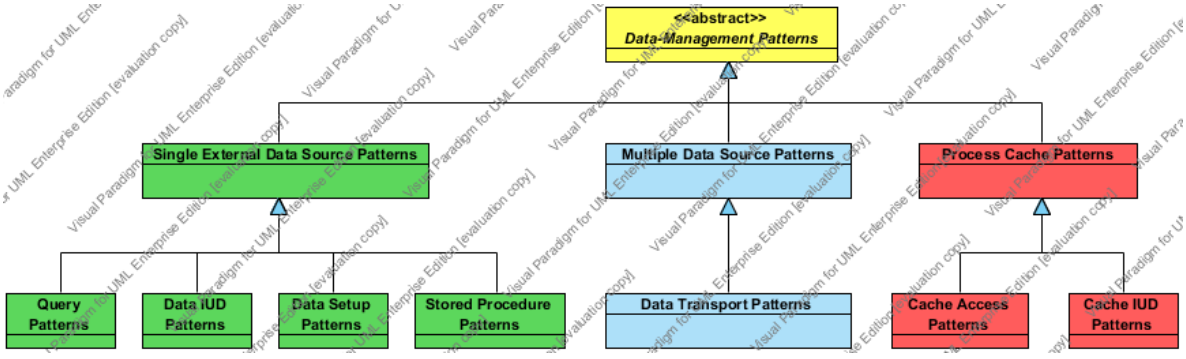


Abbildung 12: Klassenhierarchie der Datenmanagement-Patterns

Die Datenmanagement-Patterns gliedern sich hinsichtlich des Verarbeitungsortes der von ihnen zu verarbeitenden Daten in drei grundlegende Klassen:

- **Single External Data Source Patterns**, die die Verarbeitung von Daten in genau einer externen Datenquelle realisieren,
- **Process Cache Patterns**, die die Verarbeitung von Daten in genau einer internen Datenquelle, dem Prozessspeicher bzw. Cache realisieren und
- **Multiple Data Source Patterns**, die die Verarbeitung von Daten in mehr als einer Datenquelle, wobei der Prozessspeicher hier auch als eine Datenquelle angesehen wird, realisieren.

Daraus folgt direkt der weitere Aufbau der Klassenhierarchie, denn alle Unterklassen der Klasse **Single External Data Source Patterns** erfüllen ebenso deren grundlegende Eigenschaft. D.h. dass auch alle Unterklassen Daten nur extern in genau einer Datenquelle verarbeiten können. Etwas konkreter bedeutet dies, dass Daten nur auf einer externen Datenquelle beispielsweise eingefügt, gelöscht, aktualisiert oder abgefragt werden können. Wobei die abgefragten Daten auch in der gleichen Datenquelle gespeichert werden, von der sie abgefragt wurden, z.B. in einen temporären Datencontainer. Genau dasselbe gilt für die Unterklassen der Klasse **Process Cache Patterns** mit dem Unterschied, dass diese eben nur Daten in bzw. aus dem Prozessspeicher einfügen, abfragen, löschen, usw. können.

In der nächsten Ebene (2.Ebene) der Klassenhierarchie werden die Klassen dann bereits anhand der benötigten Funktionalitäten weiter verfeinert (vgl. [2]). Dazu gehören beispielsweise die Möglichkeiten, Daten abzufragen (**Query Patterns**), neue Datenstrukturen zu erstellen, zu ändern und zu verwerfen (**Data Setup Patterns**) oder auch Funktionalitäten, um Daten zwischen verschiedenen Datenquellen zu verschieben oder zu kopieren (**Data Transport Patterns**).

Alle weiteren Ebenen (ab der 3.Ebene) verfeinern die Klassenhierarchie noch weiter oder liefern die letztendlich durch BPEL-Aktivitäten umzusetzenden Patterns. Die nachfolgenden Beschreibungen orientieren sich an den sieben Klassen der 2.Ebene und erläutern die verschiedenen Ausprägungen (Patterns) bzw. die Verfeinerungen (Unterklassen) dieser Klassen, in den weiteren Ebenen.

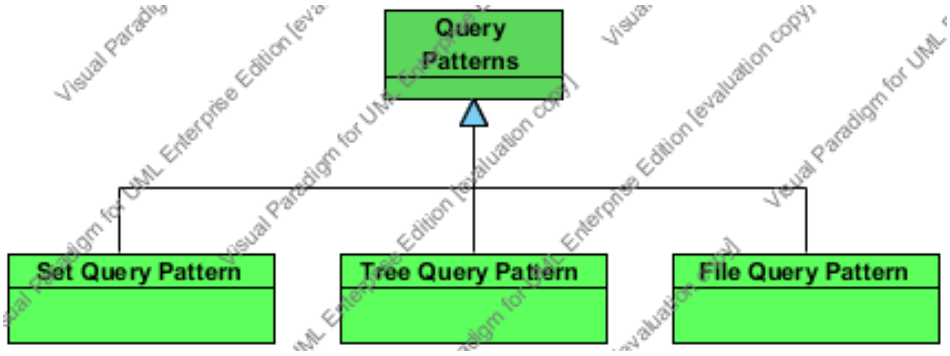


Abbildung 13: Übersicht über die Query Patterns-Klasse und deren Patterns

Query Patterns Die Klasse Query Patterns beschreibt die Notwendigkeit, mithilfe von entsprechenden Befehlen, wie z.B. SQL-Befehlen, XQuery-Befehlen oder Systemaufrufen bei Dateisystemen, externe Daten anfordern zu können. Die aus den Queries resultierenden Daten werden dabei auf der Datenquelle gespeichert, von der sie auch abgefragt wurden. Da sich die verschiedenen Datenquellentypen in der Art der Anfrage und der beteiligten Datenstrukturen/Datenmodelle unterscheiden, gibt es drei verschiedene Ausprägungen (Patterns) der Klasse, wie in Abbildung 13 zu sehen ist.

- Set Query Pattern: Für die Anfrage von relationalen (mengenorientierten) Daten mittels SQL-Befehlen aus einer relationalen Datenbank oder mittels SQL-ähnlichen Befehlen aus einer Sensornetz-Datenbank (TinyDB).

- Tree Query Pattern: Für die Anfrage von baumartigen Daten mittels XQuery-Befehlen aus einer XML-Datenbank.
- File Query Pattern: Für die Anfrage von verschiedenartig strukturierten oder strukturlosen Daten mittels Systemaufrufen aus einem Dateisystem.

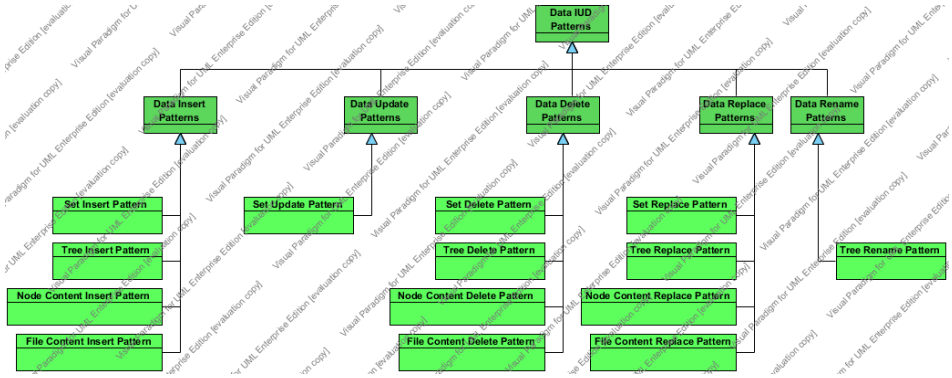


Abbildung 14: Übersicht über die Data IUD Patterns-Klasse und deren Patterns

Data IUD Patterns Die Klasse Data IUD Patterns beschreibt die Möglichkeit, auf Daten verschiedene Datenmanipulations-Operationen ausführen zu können. Die Daten und somit auch die möglichen Datenmanipulations-Operationen unterscheiden sich nach der zugrundeliegenden Datenquelle und Datenstruktur. Die Patterns dieser Klasse können für relationale Datenbanksysteme, XML-Datenbanksysteme und Dateisysteme verwendet werden. Die Sensornetzdatenbank TinyDB unterstützt das Manipulieren von Daten nicht und dadurch sind die Data IUD Patterns nicht für die TinyDB verwendbar (Beispiele siehe Abschnitt 7.1.2).

Abbildung 14 zeigt die Data IUD Patternklasse, die sich in folgende fünf operationsspezifische Unterklassen gliedert:

- **Data Insert Patterns** für das Einfügen neuer Daten.
- **Data Update Patterns** für das Aktualisieren vorhandener Daten durch beispielweise Aktualisierungsroutinen.
- **Data Delete Patterns** für das Löschen vorhandener Daten.
- **Data Replace Patterns** für das Ersetzen vorhandener Daten durch neue Daten.
- **Data Rename Patterns** für das Umbenennen vorhandener Daten.

Alle fünf Patternklassen werden durch die in Abbildung 14 gezeigten Patterns umgesetzt, die sich aufgrund der beteiligten Datenstrukturen/Datenmodelle unterscheiden. Im folgenden werden diese Patterns zusammengefasst beschrieben, d. h. es folgt eine Beschreibung für jedes Datenmodell.

- Set (Insert/Update/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, auf mengenorientierten externen Daten in einem relationalen Datenbanksystem verschiedene Datenmanipulations-Operationen ausführen zu können. Zu diesen Operationen zählen das mengen- und tupelorientierte Einfügen (Set Insert Pattern), Aktualisieren (Set Update Pattern), Löschen (Set Delete Pattern) und Ersetzen (Set Replace Pattern) von Daten. Im Gegensatz zu anderen SQL-Datenbanken sind diese Patterns nicht für die TinyDB verwendbar. Das liegt daran, dass in die TinyDB nur interne Daten, die aus dem Sensornetz selbst kommen, eingefügt werden können. Dadurch können externe Daten weder eingefügt, aktualisiert noch gelöscht werden.

- Tree (Insert/Delete/Replace/Rename) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen auf Knoten, Sequenzen von Knoten oder auch Teilbäumen von baumartigen externen Daten in einem XML-Datenbanksystem ausführen zu können. Zu diesen Operationen zählen das Einfügen (Tree Insert Pattern), Ersetzen (Tree Replace Pattern), Umbenennen (Tree Rename Pattern) und Löschen (Tree Delete Pattern) von Knoten, Sequenzen von Knoten und Teilbäumen.
- Node Content (Insert/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen in Knoten von baumartigen externen Daten in einem XML-Datenbanksystem ausführen zu können. Zu diesen Operationen zählen das Einfügen (Node Content Insert Pattern), Ersetzen (Node Content Replace Pattern) und Löschen (Node Content Delete Pattern) von Werten und Attributwerten innerhalb von Knoten.
- File Content (Insert/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen innerhalb von Dateien eines Dateisystems ausführen zu können. Zu diesen Operationen zählen das Einfügen (File Content Insert Pattern), Ersetzen (File Content Replace Pattern) und Löschen (File Content Delete Pattern) von Dateiinhalten.

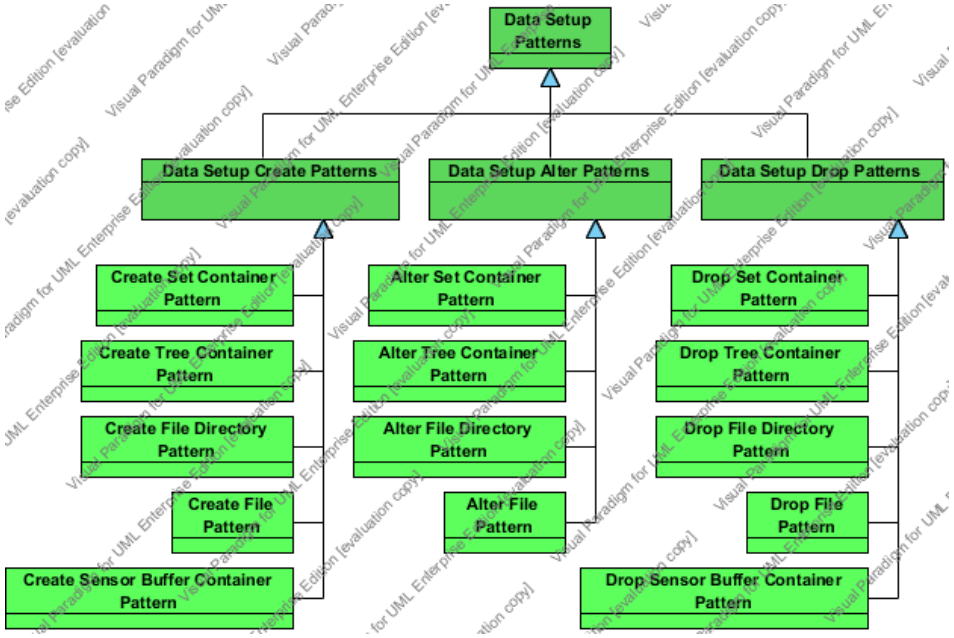


Abbildung 15: Übersicht über die Data Setup Patterns-Klasse und deren Patterns

Data Setup Patterns Die Klasse Data Setup Patterns liefert die Möglichkeit, Data Definition Language (DDL) Befehle auf relationalen Datenbanksystemen, XML-Datenbanksystemen, Dateisystemen und Sensornetz-Datenbanksystemen auszuführen. So können während der Prozessausführung die Datenquellen konfiguriert oder neue Container (Tabellen, Schema, XML-Dokumente, Ordner, Dateien, Buffer-Tabellen, usw.) erstellt, geändert oder verworfen werden.

Abbildung 15 zeigt die Data Setup Patternklasse (Data Setup Patterns), die sich in folgende drei operationsspezifische Unterklassen gliedert:

- **Data Setup Create Patterns**: Für das Erstellen neuer Container.
- **Data Setup Alter Patterns**: Für das Ändern bereits vorhandener Container.

- **Data Setup Drop Patterns**: Für das Verwerfen vorhandener Container.

Alle drei Patternklassen werden durch die in Abbildung 15 gezeigten Patterns umgesetzt, die sich aufgrund der beteiligten Datenstrukturen/Datenmodelle unterscheiden. Im folgenden werden diese Patterns zusammengefasst beschrieben, d. h. es folgt eine Beschreibung für jedes Datenmodell.

- (Create/Alter/Drop) Set Container Pattern: Diese Patterns beschreiben die Möglichkeit neue Relationen in einem relationalen Datenbanksystem zu Erzeugen (Create Set Container Pattern), vorhandene Relationen zu Bearbeiten (Alter Set Container Pattern) oder zu Löschen (Drop Set Container Pattern).

- (Create/Alter/Drop) Tree Container Pattern: Diese Patterns beschreiben die Möglichkeit neue XML-Dokumente in einem XML-Datenbanksystem zu Erzeugen (Create Tree Container Pattern), vorhandene XML-Dokumente zu Bearbeiten (Alter Tree Container Pattern) oder zu Löschen (Drop Tree Container Pattern).

- (Create/Alter/Drop) File Directory Pattern: Diese Patterns beschreiben die Möglichkeit neue Ordner in einem Dateisystem zu Erzeugen (Create File Directory Pattern), vorhandene Ordner zu Bearbeiten (Alter File Directory Pattern) oder zu Löschen (Drop File Directory Pattern).

- (Create/Alter/Drop) File Pattern: Diese Patterns beschreiben die Möglichkeit neue Dateien in einem Dateisystem zu Erzeugen (Create File Pattern), vorhandene Dateien zu Bearbeiten (Alter File Pattern) oder zu Löschen (Drop File Pattern).

- (Create/Drop) Sensor Buffer Container Pattern: Diese Patterns beschreiben die Möglichkeit neue Buffer-Tabellen in einem Sensornetz-Datenbanksystem zu Erzeugen (Create Sensor Buffer Container Pattern) oder vorhandene Buffer-Tabellen zu Löschen (Drop Sensor Buffer Container Pattern) (Beispiele siehe Abschnitt 7.1.2).

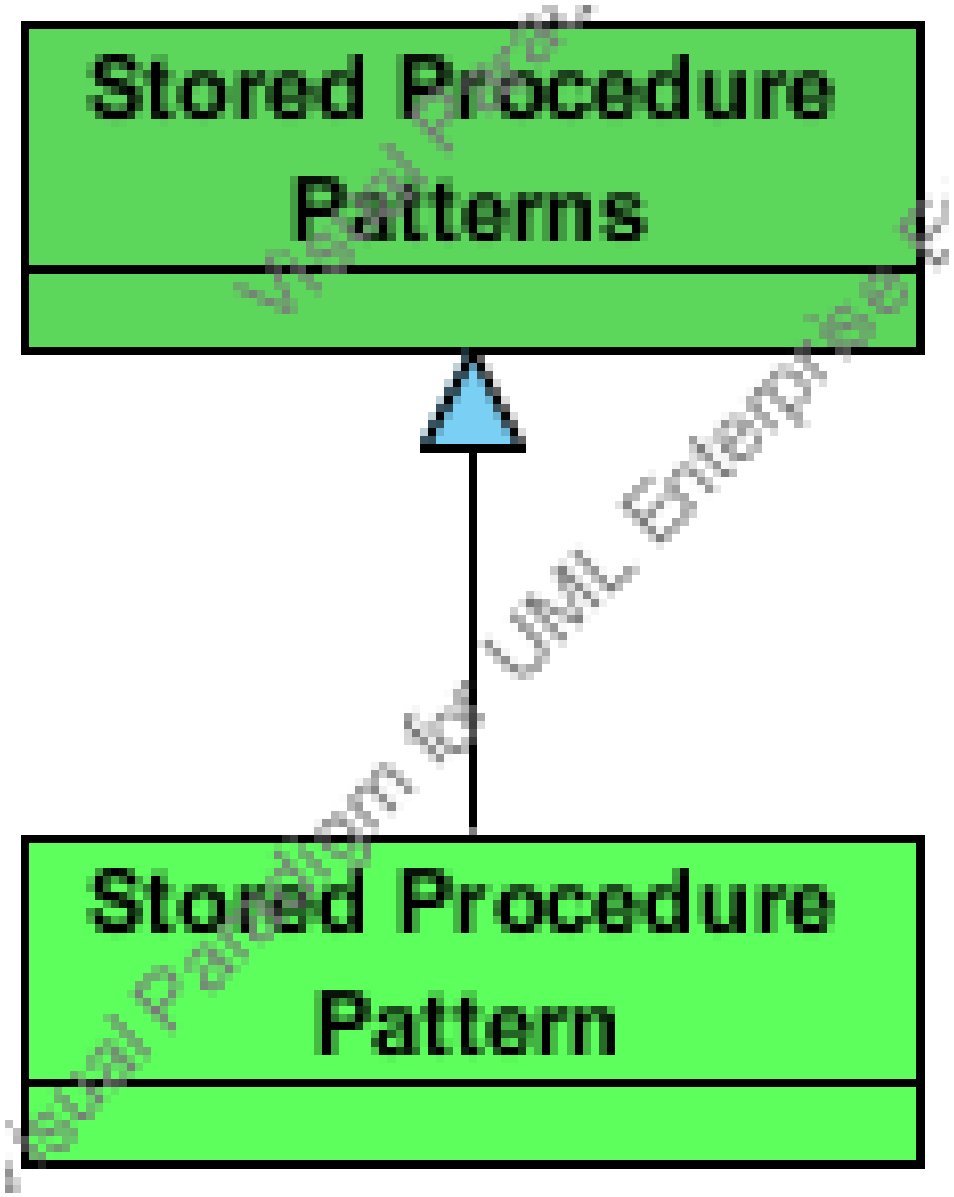


Abbildung 16: Übersicht über die Stored Procedure Patterns-Klasse und deren Patterns

Stored Procedure Patterns Abbildung 16 zeigt die Klasse Stored Procedure Patterns, die im Moment nur eine Ausprägung hat, das Stored Procedure Pattern. Trotzdem wurde für die Verarbeitung von Stored Procedures eine extra Klasse eingeführt, so dass man in Zukunft weitere Patterns dieser Klasse hinzufügen kann, wenn dafür Bedarf besteht. Das Stored Procedure Pattern kann für relationale und XML-Datenbanksysteme verwendet werden. Es wird benötigt, da die komplexe Verarbeitung von Daten meist durch Stored Procedures realisiert wird, und es so für die Verarbeitung von externen Daten unbedingt erforderlich ist, dass Stored Procedures auch aus einem Prozess heraus aufgerufen werden können. Dateisysteme und Sensornetze mit einer TinyDB unterstützen dieses Pattern nicht, da es dort keinen vergleichbaren Ansatz zu Stored Procedures gibt.

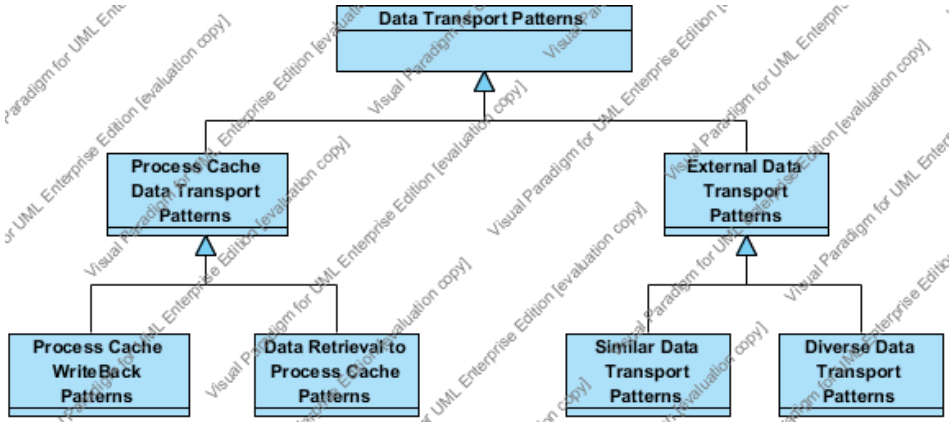


Abbildung 17: Übersicht über die Data Transport Patterns-Klasse und deren Unterklassen

Data Transport Patterns Die Klasse Data Transport Patterns liefert die Möglichkeit Daten zwischen zwei Datenquellen oder auch zwischen einer Datenquelle und einem Cache (Prozessspeicher) zu verschieben oder zu kopieren.

Abbildung 17 zeigt die Data Transport Patternklasse (Data Transport Patterns), die sich in folgende zwei logische Unterklassen gliedert:

- **Process Cache Data Transport Patterns**: Für das Kopieren von externen Daten in den Prozessspeicher (Data Retrieval to Process Cache Patterns) und das Kopieren von internen Daten aus dem Prozessspeicher auf eine externe Datenquelle (Process Cache WriteBack Patterns).
- **External Data Transport Patterns**: Für das externe Kopieren oder Verschieben von externen Daten in derselben Datenstruktur, wie z.B. das Kopieren von relationalen Daten in eine relationale Datenbank (Similar Data Transport Patterns) und für das externe Kopieren oder Verschieben von externen Daten aus einer Datenstruktur in eine Andere, wie z.B. das Kopieren von relationalen Daten in eine XML-Datenbank (Diverse Data Transport Patterns).

Diese beiden Unterklassen gliedern sich in weitere vier Unterklassen, die im folgenden in extra Abschnitten näher beschrieben werden.

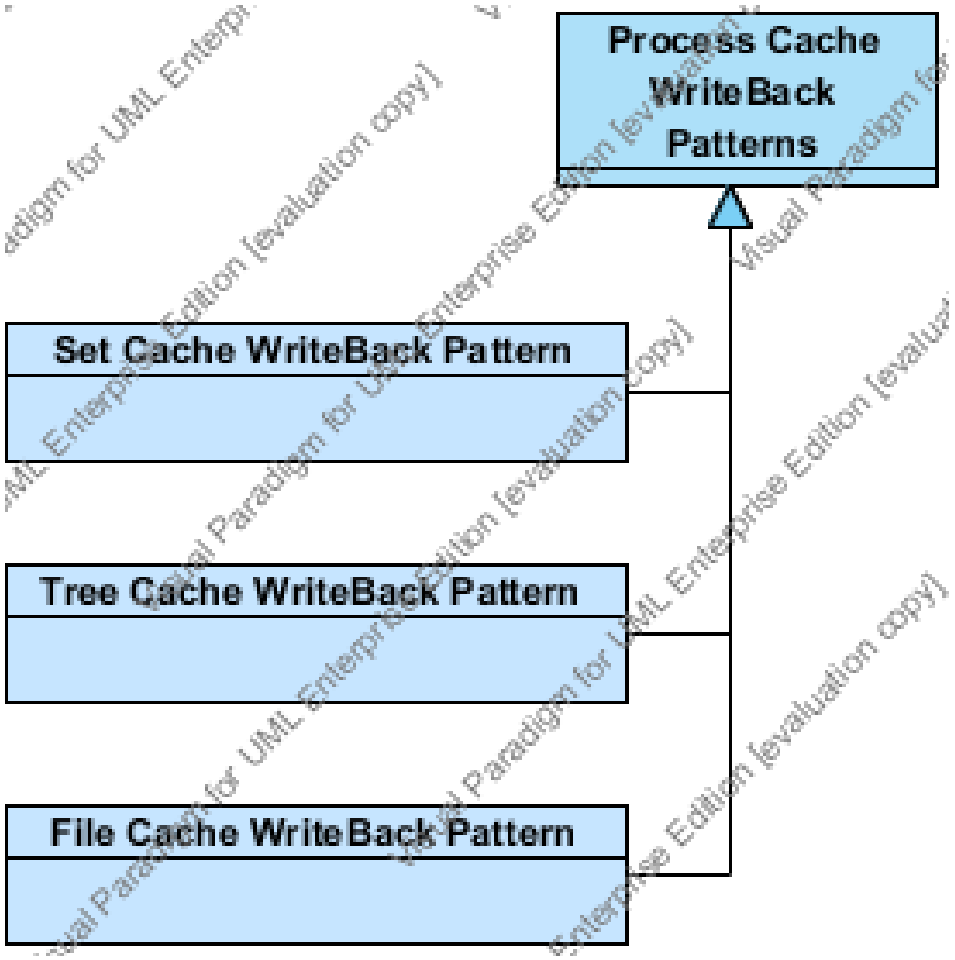


Abbildung 18: Übersicht über die Process Cache WriteBack Patterns-Klasse und deren Patterns

Process Cache WriteBack Patterns Die Klasse Process Cache WriteBack Patterns realisiert die Synchronisation bzw. das Zurückschreiben eines lokalen Datencaches auf die originale Datenquelle.

Abbildung 18 zeigt die Process Cache WriteBack Patterns, die sich in folgende drei Patterns gliedern:

- Set Cache WriteBack Pattern: Realisiert das Zurückschreiben von mengenorientierten Daten. Dieses Pattern ist ebenfalls nicht für die TinyDB verwendbar, da bei TinyDB keine Datenmanipulation möglich ist.
- Tree Cache WriteBack Pattern: Realisiert das Zurückschreiben von baumartigen Daten
- File Cache WriteBack. Pattern: Realisiert das Zurückschreiben von Dateien.

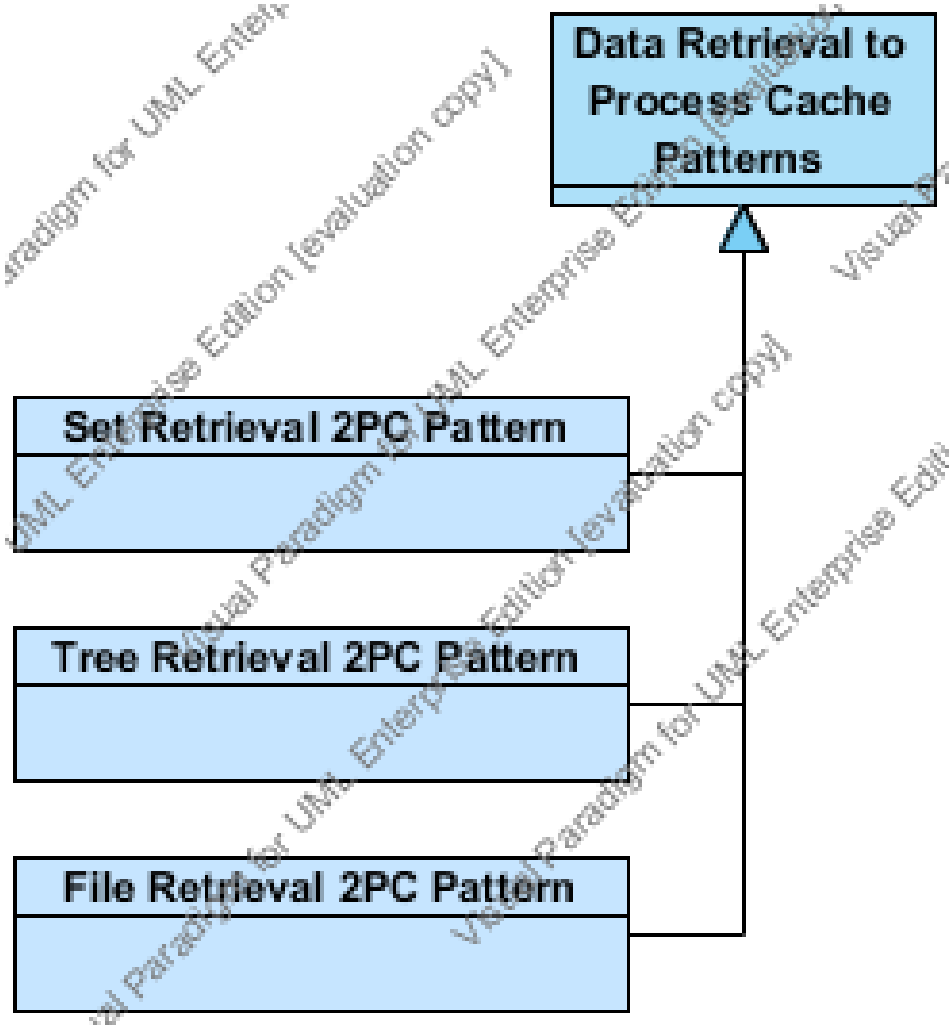


Abbildung 19: Übersicht über die Data Retrieval to Process Cache Patterns-Klasse und deren Patterns

Data Retrieval to Process Cache Patterns Die Klasse Data Retrieval to Process Cache Patterns kann für alle Datenquellentypen verwendet werden. Sie dient dazu, externe Daten in einen BPEL-Prozess zu laden, um diese dort verarbeiten zu können. Die verschiedenen Ausprägungen der Data Retrieval to Process Cache Patterns aus Abbildung 19 liefern dafür entsprechende Datenstrukturen, in die man die angefragten externen Daten innerhalb des BPEL-Prozesses ablegen kann. Diese Datenstrukturen verhalten sich dabei wie Caches im BPEL-Prozess, die keine Verbindung zur originalen Datenquelle besitzen.

Die Data Retrieval to Process Cache Patterns gliedern sich in folgende drei Patterns:

- Set Retrieval 2PC Pattern: Das Set Retrieval 2PC Pattern liefert eine mengenorientierte Datenstruktur, um externe mengenorientierte Daten im Prozessspeicher abzulegen. Es ist für relationale Datenbanken und Sensornetze mit TinyDB anwendbar.
- Tree Retrieval 2PC Pattern: Das Tree Retrieval 2PC Pattern ist für XML-Datenbanken anwendbar und liefert eine baumartige Datenstruktur, um externe baumartige Daten im Prozessspeicher abzulegen.
- File Retrieval 2PC Pattern: Das File Retrieval 2PC Pattern ist für Dateisysteme anwendbar und

liefert eine Datenstruktur, in die man aus einem Dateisystem abgefragte Daten innerhalb des BPEL-Prozesses ablegen kann.

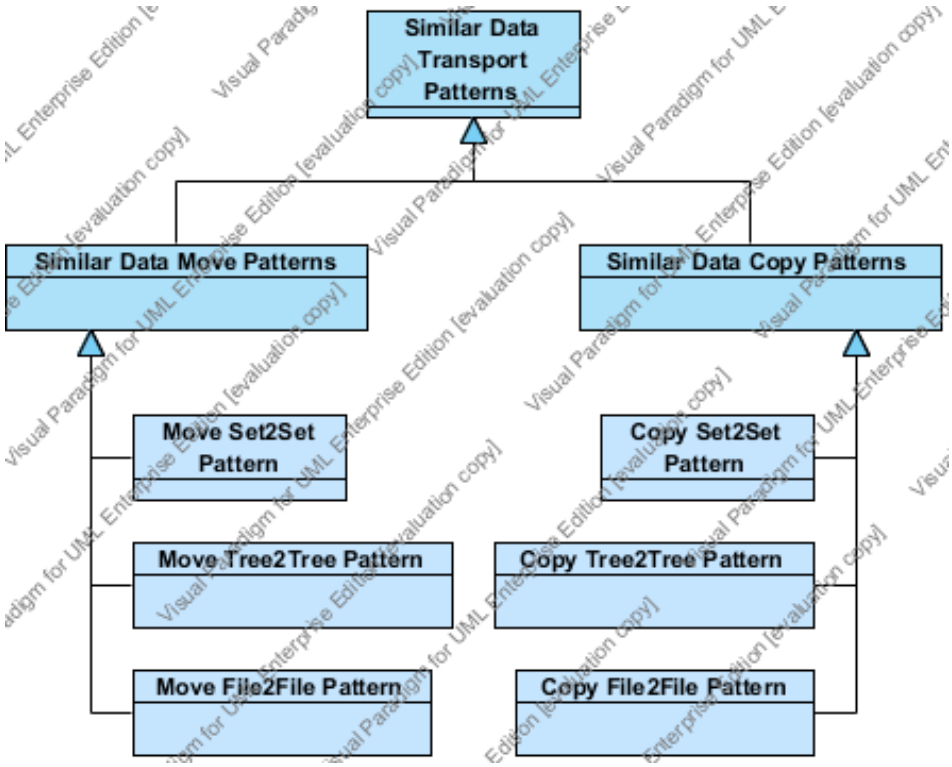


Abbildung 20: Übersicht über die Similar Data Transport Patterns-Klasse und deren Patterns

Similar Data Transport Patterns Die Klasse Similar Data Transport Patterns kann für alle Datenquellentypen verwendet werden. Sie dient dazu externe Daten von externen Datenquellen auf andere externe Datenquellen zu verschieben oder zu kopieren, wobei die Datenstruktur der zu verschiebenden/kopierenden Daten unverändert bleibt.

Abbildung 20 zeigt die Similar Data Transport Patterns, die sich in folgende zwei Klassen gliedern:

- **Similar Data Move Patterns**: Für das Verschieben von externen Daten auf externen Datenquellen, wobei die Datenstruktur unverändert bleibt.
- **Similar Data Copy Patterns**: Für das Kopieren von externen Daten auf externen Datenquellen, wobei die Datenstruktur unverändert bleibt.

Beide Patternklassen werden durch die in Abbildung 20 gezeigten Patterns umgesetzt, die sich aufgrund der beteiligten Datenstrukturen/Datenmodelle unterscheiden. Im folgenden werden diese Patterns zusammengefasst beschrieben, d. h. es folgt eine Beschreibung für jedes Datenmodell.

- (Move/Copy) Set2Set Pattern: Liefert die Möglichkeit externe mengenorientierte Daten von einer externen Datenquelle auf eine andere externe Datenquelle zu Kopieren oder zu Verschieben. Die Datenstruktur der Daten bleibt dabei unverändert mengenorientiert. Dieses Pattern ist für relationale Datenbanken anwendbar.
- (Move/Copy) Tree2Tree Pattern: Liefert die Möglichkeit externe baumorientierte Daten von einer externen Datenquelle auf eine andere externe Datenquelle zu Kopieren oder zu Verschieben. Die

Datenstruktur der Daten bleibt dabei unverändert baumorientiert. Dieses Pattern ist für XML-Datenbanken anwendbar.

- (Move/Copy) File2File Pattern: Liefert die Möglichkeit externe Daten aus Dateien von einer externen Datenquelle auf eine andere externe Datenquelle zu Kopieren oder zu Verschieben. Die Datenstruktur der Daten bleibt dabei unverändert. Dieses Pattern ist für Dateisysteme anwendbar.

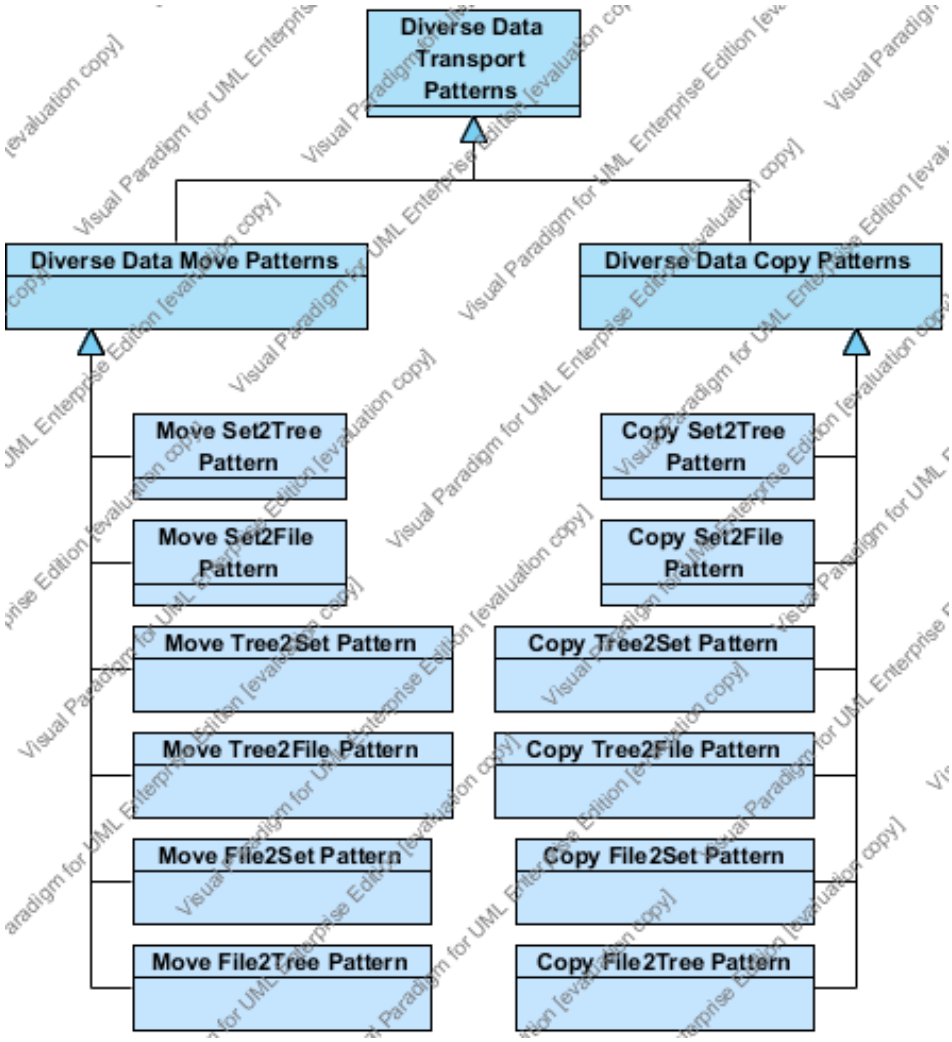


Abbildung 21: Übersicht über die Diverse Data Transport Patterns-Klasse und deren Patterns

Diverse Data Transport Patterns Die Klasse Diverse Data Transport Patterns kann für alle Datenquellentypen verwendet werden. Sie dient dazu externe Daten von externen Datenquellen auf andere externe Datenquellen zu verschieben oder zu kopieren, wobei sich die Datenstruktur der zu verschiebenden/kopierenden Daten ändert, d.h. die Daten werden in eine andere Datenstruktur überführt.

Abbildung 21 zeigt die Diverse Data Transport Patterns, die sich in folgende zwei Klassen gliedern:

- **Diverse Data Move Patterns**: Für das Verschieben von externen Daten auf externen Datenquellen mit veränderter Datenstruktur.

- **Diverse Data Copy Patterns**: Für das Kopieren von externen Daten auf externen Datenquellen mit veränderter Datenstruktur.

Beide Patternklassen werden durch die in Abbildung 21 gezeigten Patterns umgesetzt, die sich aufgrund der beteiligten Datenstrukturen/Datenmodelle unterscheiden. Im folgenden werden nur drei dieser Patterns beschrieben, da bei den verbleibenden drei Patterns jeweils nur eine umgekehrte Datenmodell-Transformation durchgeführt wird.

- (Move/Copy) Set2Tree Pattern: Liefert die Möglichkeit externe mengenorientierte Daten aus einer relationalen Datenbank oder einer Sensornetz-Datenbank auf eine XML-Datenbank als baumartige Daten zu Kopieren oder zu Verschieben. Dieses Pattern ist für relationale, Sensortnetz und XML-Datenbanken anwendbar, wobei die relationalen bzw. Sensornetz-Datenbanken nur als Quelle und die XML-Datenbanken nur als Ziel der Kopier- bzw. Verschiebe-Operation verwendet werden können.

- (Move/Copy) File2Set Pattern: Liefert die Möglichkeit externe Daten aus Dateisystemen auf eine relationale Datenbank als relationale Daten zu Kopieren oder zu Verschieben. Dieses Pattern ist für Dateisysteme und relationale Datenbanken anwendbar, wobei die Dateisysteme nur als Quelle und die relationalen Datenbanken nur als Ziel der Kopier- bzw. Verschiebe-Operation verwendet werden können. Die Sensornetz-Datenbank TinyDB kann hier nicht als Ziel angegeben werden, da auf ihr keine Datenmanipulation möglich ist.

- (Move/Copy) File2Tree Pattern: Liefert die Möglichkeit externe Daten aus Dateisystemen auf eine XML-Datenbank als baumartige Daten zu Kopieren oder zu Verschieben. Dieses Pattern ist für Dateisysteme und XML-Datenbanken anwendbar, wobei die Dateisysteme nur als Quelle und die XML-Datenbanken nur als Ziel der Kopier- bzw. Verschiebe-Operation verwendet werden können.

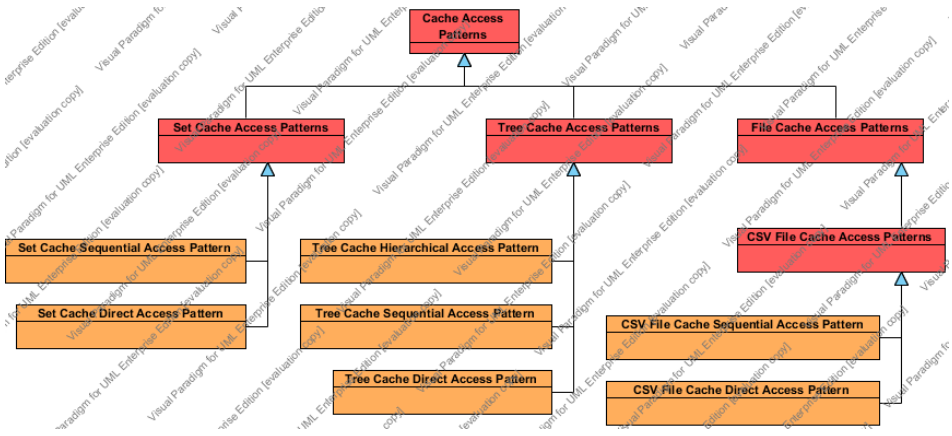


Abbildung 22: Übersicht über die Cache Access Patterns-Klasse und deren Unterklassen

Cache Access Patterns Die Klasse Cache Access Patterns beschreibt die Notwendigkeit, auf den im BPEL-Prozess erzeugten Datencache in geeigneter Weise zugreifen zu können. Die weiteren Unterklassen unterscheiden sich durch die zugrundeliegende Datenstruktur der Daten im Prozessspeicher. Die einzelnen Patterns dieser Klassen liefern verschiedene Zugriffsmöglichkeiten auf die Daten im Prozessspeicher. Da diese Zugriffsmöglichkeiten je nach Datenstruktur variieren, wird auf diese in den nachfolgenden Beschreibungen näher eingegangen.

Abbildung 22 zeigt die Cache Access Patterns, die sich in folgende drei Klassen gliedern:

- **Set Cache Access Patterns**: Die Cache Set Access Patterns liefern die Möglichkeit, auf den im BPEL-Prozess erzeugten mengenorientierten Datencache sequentiell und direkt (wahlfrei) zugreifen zu können.
  - Set Cache Sequential Access Pattern: Liefert den sequentiellen Zugriff auf die mengenorientierte Datenstruktur, d.h. die Relation wird von ihrem ersten Element bis zu ihrem Letzten schrittweise durchlaufen, wobei in einem Schritt immer nur das vorherige oder das nächste Element der Relation erreicht werden kann.
  - Set Cache Direct Access Pattern: Liefert den direkten (wahlfreien) Zugriff auf die mengenorientierte Datenstruktur, d.h. jedes beliebige Element der Relation kann direkt z.B. über einen Index ausgewählt werden.
- **Tree Cache Access Patterns**: Die Tree Cache Access Patterns liefern die Möglichkeit, auf den erzeugten baumorientierten Datencache hierarchisch, sequentiell und direkt (wahlfrei) zugreifen zu können.
  - Tree Cache Hierarchical Access Pattern: Liefert den hierarchischen Zugriff, der durch entsprechende Funktionen realisiert wird, die die Kindknoten oder den Vaterknoten des ausgewählten Knoten liefern. Über den Wurzelknoten als Startpunkt kann so der ganze Baum hierarchisch in beiden Richtungen (Wurzel -> Blatt vs. Blatt ->Wurzel) durchlaufen werden.
  - Tree Cache Sequential Access Pattern: Liefert den sequentiellen Zugriff, der durch entsprechende Funktionen realisiert wird, die die Geschwisterknoten oder den Kind- bzw. Vaterknoten des ausgewählten Knoten liefern. Über den Wurzelknoten als Startpunkt kann jede Ebene des ganzen Baums sequentiell in beiden Richtungen (Wurzel -> Blatt vs. Blatt ->Wurzel) durchlaufen werden.
  - Tree Cache Direct Access Pattern: Liefert den direkten Zugriff beispielsweise durch die Angabe eines XPath-Ausdrucks (siehe[24]) mit dem der entsprechende Knoten im Baum gesucht und zurückgegeben wird, falls ein solcher Knoten existiert.
- **File Cache Access Patterns**: Die File Cache Access Patterns liefern die Möglichkeit, auf Dateiinhalte eines erzeugten Datencache in geeigneter Weise (z.B. sequentiell oder wahlfrei) zugreifen zu können. Die Klasse kann durch weitere dateiformatspezifische Unterklassen verfeinert werden. Im Moment wird nur das Dateiformat Comma Seperated Values (CSV) näher betrachtet und beschrieben.
  - CSV File Cache Access Patterns: Die CSV File Cache Access Patterns liefern die Möglichkeit, auf den erzeugten CSV-Datencache sequentiell und direkt (wahlfrei) zugreifen zu können.
    - CSV File Cache Sequential Access Patterns: Liefert den sequentiellen Zugriff auf die CSV-Daten, d.h. die Daten werden von der ersten Zeile bis zu der Letzten zeilenweise durchlaufen, wobei in einem Schritt immer nur die vorherige oder die nachfolgende Zeile erreichbar ist.
    - CSV File Cache Direct Access Patterns: Liefert den direkten (wahlfreien) Zugriff auf die CSV-Daten, d.h. jede beliebige Zeile der Daten kann direkt z.B. über einen Index ausgewählt werden.

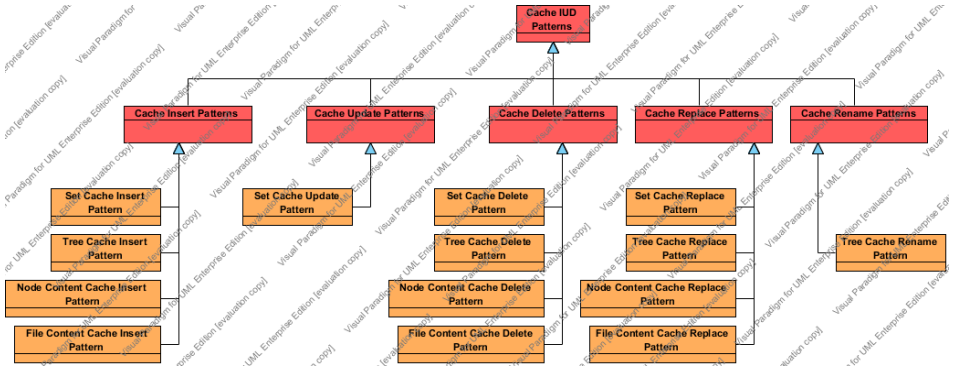


Abbildung 23: Übersicht über die Cache IUD Patterns-Klasse und deren Patterns

Cache IUD Patterns Die Klasse Cache IUD Patterns beschreibt die Notwendigkeit, dass in einen Datencache auch Daten eingefügt, aktualisiert, umbenannt, ersetzt und wieder aus diesem gelöscht werden können.

Abbildung 23 zeigt die Cache IUD Patternklasse, die sich in folgende fünf operationsspezifische Unterklassen gliedert:

- **Cache Insert Patterns** für das Einfügen neuer Daten in den Prozessspeicher.
- **Cache Update Patterns** für das Aktualisieren vorhandener Daten im Prozessspeicher durch beispielweise Aktualisierungsroutinen.
- **Cache Delete Patterns** für das Löschen vorhandener Daten im Prozessspeicher.
- **Cache Replace Patterns** für das Ersetzen vorhandener Daten im Prozessspeicher durch neue Daten.
- **Cache Rename Patterns** für das Umbenennen vorhandener Daten im Prozessspeicher.

Alle fünf Patternklassen werden durch die in Abbildung 23 gezeigten Patterns umgesetzt, die sich aufgrund der beteiligten Datenstrukturen/Datenmodelle unterscheiden. Im folgenden werden diese Patterns zusammengefasst beschrieben, d. h. es folgt eine Beschreibung für jedes Datenmodell.

- Set Cache (Insert/Update/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, auf mengenorientierten internen Daten im Prozessspeicher verschiedene Datenmanipulations-Operationen ausführen zu können. Zu diesen Operationen zählen das mengen- und tupelorientierte Einfügen (Set Cache Insert Pattern), Aktualisieren (Set Cache Update Pattern), Löschen (Set Cache Delete Pattern) und Ersetzen (Set Cache Replace Pattern) von Daten.
- Tree Cache (Insert/Delete/Replace/Rename) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen auf Knoten, Sequenzen von Knoten oder auch Teilbäumen von baumartigen internen Daten im Prozessspeicher ausführen zu können. Zu diesen Operationen zählen das Einfügen (Tree Cache Insert Pattern), Ersetzen (Tree Cache Replace Pattern), Umbenennen (Tree Cache Rename Pattern) und Löschen (Tree Cache Delete Pattern) von Knoten, Sequenzen von Knoten und Teilbäumen.
- Node Content Cache (Insert/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen in Knoten von baumartigen internen Daten im Prozessspeicher ausführen zu können. Zu diesen Operationen zählen das Einfügen (Node Content Cache Insert Pattern), Ersetzen (Node Content Cache Replace Pattern) und Löschen (Node Content Cache Delete Pattern) von Werten und Attributwerten innerhalb von Knoten.

- File Content Cache (Insert/Delete/Replace) Pattern: Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulations-Operationen innerhalb von Daten, die aus Dateien ausgelesen wurden, im Prozessspeicher ausführen zu können. Zu diesen Operationen zählen das Einfügen (File Content Cache Insert Pattern), Ersetzen (File Content Cache Replace Pattern) und Löschen (File Content Cache Delete Pattern) von Dateiinhalten.

Tabelle 2 zeigt eine Übersicht aller Datenmanagement-Patterns, die für den Umgang mit verschiedenen Datenquellentypen benötigt werden, und welche Patterns für welche Datenquellentypen verwendet werden können (durch "x" markiert).

Tabelle 2: Übersicht der Datenmanagement-Patterns und der von diesen unterstützten Datenquellentypen

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys.	Sensor.-DB	Cache

7.1.2 Umsetzung der Datenmanagement-Patterns

In diesem Abschnitt wird die Realisierung der verschiedenen Datenmanagement-Patterns aus Abschnitt 7.1.1 im Zusammenhang mit den jeweiligen Datenquellentypen beschrieben. Einige Patterns werden dabei für manche Datenquellen aufgrund ihrer Komplexität erst in einer späteren Iterationsstufe umgesetzt.

Relationale Datenbanken

Hier wird die Umsetzung der in Abschnitt 7.1.1 beschriebenen Patterns im Bereich der SQL-Datenbanken durch bestehende SQL-Befehle oder, falls erforderlich, durch die Definition neuer zu implementierender Funktionen beschrieben.

- Query Pattern: Realisierung durch SQL-SELECT.

  - Beispiel:
    * SELECT info FROM customer

- Set IUD Pattern: Realisierung durch SQL-INSERT, SQL-UPDATE, SQL-DELETE.
  - Beispiele:
    * INSERT INTO department VALUES ('E31', 'architecture', '00390', 'E01')
    * INSERT INTO department SELECT * FROM old_department
    * UPDATE employee SET job = 'laborer' WHERE empno = '000290'
    * DELETE FROM department WHERE deptno = 'D11'

- Data Setup Pattern: Realisierung durch SQL-CREATE SCHEMA, SQL-CREATE TABLE, SQL-CREATE VIEW und weitere SQL-CREATE Befehle. Weiterhin sollte auch das Löschen dieser Objekte möglich sein, dies wird durch SQL-DROP realisiert. Momentan noch optional ist die Bearbeitung entsprechender Objekte über SQL-ALTER Befehle.
  - Beispiele:
    * CREATE SCHEMA internal AUTHORIZATION admin
    * CREATE TABLE tdept (deptno CHAR(3) NOT NULL, deptname VARCHAR(36) NOT NULL, mgrno CHAR(6), admrdept CHAR(3) NOT NULL, PRIMARY KEY(deptno)) IN internal
    * CREATE VIEW internal.departmentView AS SELECT deptno, deptname FROM internal.tdept
    * DROP SCHEMA internal
    * DROP TABLE tdept
    * DROP VIEW internal.departmentView

- Stored Procedure Pattern: Realisierung durch SQL-CALL.
  - Beispiele:
    * CALL parts_on_hand (?,?,?) *(? = Parameter der Prozedur)*

- Set Retrieval Pattern: Realisierung durch die Definition und Bereitstellung einer entsprechenden BPEL-Aktivität. Diese Aktivität liefert die Möglichkeit, dass durch einen SQL-SELECT Befehl abgefragte Daten (siehe Query Pattern) in den BPEL-Prozess geladen und in einer entsprechenden BPEL-Variable ablegt werden können. Dazu müssen neue Variablentypen bereitgestellt werden, die den Strukturen der zu haltenden Daten entsprechen, also mengenorientierte Daten halten können. Die Daten werden dazu über das *Service Data Objects Application Programming Interface* (SDO API) abstrahiert. Wie in [2] beschrieben, wurde dieses Konzept bereits durch die *Business Integration Suite* und dem darin enthaltenen *WebSphere Integration Developer* mit der Bezeichnung *"Retrieve Set Activity"* realisiert. Dabei werden in einer Retrieve Set Activity externe Daten in eine XML-Struktur innerhalb des BPEL-Prozesses geladen. Eine etwas ausführlichere Beschreibung diese Ansatzes liefert [2]. Unser Ansatz beschränkt sich zunächst auf die Möglichkeiten, die durch die SDO API und die *Data Access Services* (DAS) API bereitgestellt werden.

- Cache Set Access Pattern: Um das Set Access Pattern zu realisieren, müssen im Projektverlauf Methoden in BPEL bereitgestellt werden, die einen sequentiellen und direkten Zugriff auf den durch eine RetrieveData Aktivität erzeugten Datencache ermöglichen. Die SDO API liefert dafür bereits einige Methoden, die verwendet werden können und auf die wir uns zunächst beschränken. Weiterhin soll bei einem sequentiellen Zugriff auch mit einer ForEach-Aktivität über die mengenorientierten Daten iteriert werden können.

- Cache Set IUD Pattern: Erweitert die Cache Set Access Pattern Methoden um die Möglichkeiten, Tupel innerhalb der mengenorientierten Datenstruktur im BPEL-Prozess zu aktualisieren, einzufügen und zu löschen. Die SDO API liefert dafür bereits einige Methoden, die verwendet werden können und auf die wir uns zunächst beschränken.
- Cache Set WriteBack Pattern: Um die Daten aus dem Prozesscache zurück auf die originale Datenbank zu übertragen, muss ebenfalls eine neue BPEL-Aktivität erstellt werden, durch die der Benutzer angibt, dass die Daten zurückgeschrieben werden sollen. Diese Aktivität nutzt dann intern die im Set IUD Pattern und Cache Set Access Pattern beschriebenen Methoden und ebenso SQL-Befehle, um die Daten aus dem Prozesscache auf die Datenbank zu übertragen. Die SDO API und die DAS API liefern dafür bereits einige Methoden, die verwendet werden können und auf die wir uns zunächst beschränken.

XML-Datenbanken

Hier wird die Umsetzung der in Abschnitt 7.1.1 beschriebenen Patterns im Bereich der XQuery-Datenbanken durch bestehende XQuery-Befehle oder, falls erforderlich, durch die Definition neuer zu implementierender Funktionen beschrieben.

- Query Pattern: Realisierung durch entsprechende FLWOR-Befehle.
- Beispiel:
  - FOR $d IN fn:doc(“department.xml”)/employees/employee RETURN $d/name
- Tree IUD Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility (siehe [22]).
  - Beispiele:
    - INSERT NODE <phone>123456</phone> AFTER fn:doc("department.xml")/employees/employee[1]/name
    - REPLACE NODE fn:doc("department.xml")/employees/employee[1]/phone WITH fn:doc("department2.xml")/employees/employee[20]/phone
    - RENAME NODE fn:doc("department.xml")/employees/employee[1]/phone[2] AS "privatePhone"
    - DELETE NODE fn:doc("department.xml")/employees/employee[1]/phone[last()]
- Node Content IUD Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
  - Beispiel:
    - REPLACE VALUE OF NODE fn:doc("department.xml")/expectedSales/total WITH fn:doc("department.xml")/expectedSales/total + 50000
- Data Setup Pattern: Wird erst in einer späteren Iteration realisiert.
- Stored Procedure Pattern: Wird erst in einer späteren Iteration realisiert.
- Tree Retrieval Pattern: Die Umsetzung erfolgt nach der Beschreibung des Set Retrieval Patterns für relationale DB (siehe 7.1.2), wobei hier XQuery-Befehle verwendet und baumartige Daten in einer BPEL-Variable abgelegt werden.
- Cache Tree Access Pattern: Wird erst in einer späteren Iteration realisiert.
- Cache Tree IUD Pattern: Wird erst in einer späteren Iteration realisiert.
- Cache Tree WriteBack Pattern: Wird erst in einer späteren Iteration realisiert.

Dateisysteme

Hier wird die Umsetzung der in Abschnitt 7.1.1 beschriebenen Patterns im Bereich der Dateisysteme durch entsprechende Systemaufrufe des dem Dateisystem zugrundeliegenden Betriebssystems beschrieben.

- Query Pattern: Realisierung durch einen GET-Befehl, der intern java.io Methoden verwendet, um Inhalte aus Dateien oder komplette Dateien aus einem Dateisystem zu lesen. Für das Abfragen von Dateiinhalten muss noch ein Konzept erarbeitet werden, wie entsprechende Daten gezielt über einen Befehl ausgewählt und abgefragt werden können. Eine Möglichkeit wäre z.B. einen Filter-Dialog zur Angabe von Suchparametern über die GUI bereitzustellen, der es ermöglicht Dateiinhalte zu suchen und zurückzugeben. Das Abfragen von Dateiinhalten wird erst in einer der weiteren Iteration realisiert.
- File Content IUD Pattern: Realisierung durch einen entsprechenden PUT- bzw. REMOVE-Befehl (RM), die intern java.io Methoden verwenden, um Inhalte in Dateien zu schreiben und aus ihnen zu löschen. Ein Update wird bei Dateien intern durch das Löschen des alten Wertes und anschließendem Einfügen des neuen Wertes ausgeführt. Dafür muss noch ein Konzept erarbeitet werden, wie entsprechende Informationen gezielt über einen Befehl eingefügt und gelöscht werden können. In der ersten Iterationsstufe wird vorerst nur das Einfügen bzw. Löschen von vollständigen Dateiinhalten umgesetzt werden.
- Data Setup Pattern: Realisierung durch die Verwendung entsprechender Erzeugungsbefehle, wie MKDIR und MKFILE zur Erzeugung von Ordnern und Dateien. Diese werden intern wieder durch java.io-Methoden realisiert. Hier sollte es auf jeden Fall möglich sein, Dateien und Ordner in einem Dateisystem zu erzeugen. Ebenso sollte das Löschen auf der gleichen Ebene, d.h. von ganzen Dateien und Ordnern, möglich sein.
- File Retrieval Pattern: Die Umsetzung erfolgt nach der Beschreibung des Set Retrieval Patterns für relationale DB (siehe 7.1.2), wobei hier entsprechende Dateisystem-Befehle (siehe Query Pattern) verwendet werden und in der ersten Iterationsstufe nur vollständige Dateiinhalte in einer BPEL-Variable abgelegt werden können.
- Cache File Access Pattern: Um das Cache File Access Pattern zu realisieren, müssen im Projektverlauf Methoden, die in einer geeigneter Weise (z.B. sequentiell oder wahlfrei) den Zugriff auf den durch das File Retrieval Pattern erzeugten Datencaches ermöglichen, bereitgestellt werden. Die SDO API liefert dafür bereits einige Methoden, die verwendet werden können. In der ersten Iterationsstufe werden, die im Datencache hinterlegten, vollständigen Dateiinhalte als reiner Fließtext interpretiert und können auch nur als solcher verarbeitet werden. Durch die Java API stehen so entsprechende Fließtextoperationen zur Verarbeitung der Dateiinhalte zur Verfügung.
- Cache File IUD Pattern: Erweitert die Cache File Access Pattern Methoden um die Möglichkeiten, Werte innerhalb des Datencaches des BPEL-Prozesses zu aktualisieren, einzufügen und zu löschen. Die SDO API liefert dafür bereits einige Methoden, die verwendet werden können. In der ersten Iterationsstufe werden auch hier nur entsprechende Fließtextoperationen (append, split, etc.) durch die Java API bereitgestellt.
- Cache File WriteBack Pattern: Wird ebenso wie das Cache Set WriteBack Pattern bei relationalen Datenbanken umgesetzt (siehe 7.1.2). Die Umsetzung dieses Patterns wird erst in einer späteren Iteration realisiert, da zuerst Erkentnisse im Bereich der relationalen Datenbanken gesammelt werden und mit diesen dann ein Ansatz für Dateisysteme erarbeitet wird.

Sensornetze

Hier wird die Umsetzung der in Abschnitt 7.1.1 beschriebenen Patterns im Bereich der Sensornetze und deren Datenbanken durch bestehende sensornetzspezifische SQL-Befehle oder, falls erforderlich,

durch die Definition neuer zu implementierender Funktionen beschrieben. Alle SQL-Befehlsbeispiele beziehen sich hier auf den SQL-Dialekt der Sensornetz-Datenbank TinyDB (siehe [14]).

- Query Pattern: Realisierung durch ein entsprechendes SQL-SELECT der Sensornetz-Datenbank. Weiterhin besteht die Möglichkeit, sensornetzintern in einem Buffer Werte zwischenzuspeichern. Realisiert wird dies durch im Arbeitsspeicher von Sensoren erstellte Tabellen (siehe Data Setup Pattern), die mit momentanen Sensorwerten gefüllt werden können, um später die Daten zeitversetzt abrufen zu können. Die gewünschten Werte werden dabei mit einem entsprechenden SQL-SELECT Befehl in den Buffer geschrieben. Die TinyDB liefert noch ein weiteres Konzept für Abfragen. Dabei können, falls eine bestimmte Bedingung gilt (z.B. Temperatur $> 20°C$), sogenannte *commands* aufgerufen werden, die dann bestimmte Aktionen anstoßen. Der Code für diese Methoden wird auf die entsprechenden Sensoren übertragen und dort dann bei Bedarf ausgeführt.

  - Befehlsstruktur:
    - SELECT select-list [FROM sensors] WHERE where-clause [GROUP BY gb-list [HAVING having-list]][TRIGGER ACTION command-name[(param)]] [EPOCH DURATION integer]
    - Werte in Buffer-Tabelle einfügen: SELECT field1, field2, ... FROM sensors SAMPLE PERIOD x INTO name
  - Beispiele:
    - SELECT temp FROM sensors WHERE temp > thresh TRIGGER ACTION SetSnd(512) EPOCH DURATION 512
    - SELECT field1, field2 FROM sensors SAMPLE PERIOD 100 INTO name
      (Einfügen von Werten, über SELECT, in die Buffer-Tabelle *name*)
    - SELECT temp FROM sensors WHERE temp > thresh TRIGGER ACTION SetSnd(512) EPOCH DURATION 512
      - *Bemerkungen: Hier wird alle 512ms die Temperatur an den Sensoren abgefragt und falls diese einen gewissen Wert übersteigt, wird über den* ***command*** *SetSnd(512) für 512ms ein Signalton ausgegeben.*

- Data Setup Pattern: Es können Tabellen im Arbeitsspeicher der Sensoren erstellt werden, die dann Werte von Sensoren aufnehmen und halten können. Die Erstellung solcher Tabellen wird durch SQL-CREATE BUFFER und das Löschen des gesamten Buffers durch SQL-DROP realisiert.

  - Eine Buffer-Tabelle erstellen: CREATE BUFFER name SIZE x (field1 type1, field2 type2, ...)
    - *Bemerkungen:* **x** *ist die Anzahl der maximal möglichen Zeilen,* ***type1****,* ***type2****, usw. sind Datentypen aus der Menge {uint8, uint16, uint16, int8, int16, int32} und* ***field1****,* ***field2****, usw. sind Spaltennamen, die wie der Tabellenname jeweils 8 Zeichen lang sein dürfen.*
  - Alle Buffer-Tabellen löschen: DROP ALL

- Set Retrieval Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.

- Cache Set Access Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.

- Cache Set IUD Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.

Datenquellen-unabhängige Patterns

Hier wird die Umsetzung der in Abschnitt 7.1.1 beschriebenen datenquellen-unabhängigen Patterns beschrieben. Dies sind die Data Move Patterns, also das Move Pattern, das Import Pattern und das Export Pattern. Diese Patterns werden erst in einer späteren Iteration umgesetzt.

- Move Pattern: Das Move Pattern wird dadurch realisiert, dass die je nach Datenquellentyp benötigten datenquellen-abhängigen Patterns verwendet werden. So werden z.B. um Daten aus einem Dateisystem in eine relationale Datenbank zu verschieben, zuerst die Daten über das File Retrieve Pattern des Dateisystems in den Prozess geladen, dort verarbeitet und dann durch die Verwendung des Set IUD Patterns in die Datenbank in eine entsprechende Tabelle eingefügt.
- Import Pattern: Das Import Pattern wird dadurch realisiert, dass das File Retrieval Pattern, mit dem lokalen Dateisystem als Datenquelle, verwendet wird.
- Export Pattern: Das Export Pattern wird dadurch realisiert, dass durch die Verwendung des Data Setup Patterns eine neue Datei auf dem lokalen Dateisystem erzeugt wird und die Daten des Caches über das Cache File WriteBack Pattern in diese Datei eingefügt werden. Eine alternative Realisierung wäre über das Auslesen des Caches (Cache File Access Pattern) und das Einfügen der Daten in eine lokale Datei über das File Content IUD Pattern möglich.

Tabelle 3 zeigt eine Übersicht welche Datenmanagement-Patterns für welche Datenquellentypen umgesetzt werden. Ein “x” markiert dabei die vollständige Umsetzung in der ersten Iterationsstufe, ein “t” für die teilweise bzw. prototypische Umsetzung in der ersten Iterationsstufe, ein “(x)” die Umsetzung in einer späteren Iterationsstufe und ein “o” die generell optionale Umsetzung eines Patterns für einen Datenquellentyp.

Tabelle 3: Übersicht über die Umsetzung der Datenmanagement-Patterns für die verschiedenen Datenquellentypen

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys.	Sensor.-DB	Cache
Query Pattern	x	x	t	x	
Set IUD Pattern	x				
Tree IUD Pattern		x			
Node Content IUD Pattern		x			
File Content IUD Pattern			t		
Data Setup Pattern	t	(x)	t	x	
Stored Procedure Pattern	x	(x)			
Move Pattern	(x)/o	(x)/o	(x)/o	(x)/o	
Import Pattern					o
Export Pattern					o
Set Retrieval Pattern	t			t	
Tree Retrieval Pattern		t			
File Retrieval Pattern			t		
Cache Set Access Pattern					t
Cache Tree Access Pattern					(x)
Cache File Access Pattern					t
Cache Set IUD Pattern					t
Cache Tree IUD Pattern					(x)
Cache File IUD Pattern					t
Cache Set WriteBack Pattern	t/o				
Cache Tree WriteBack Pattern		(x)/o			
Cache File WriteBack Pattern			(x)/o		

7.1.3 Resultierende BPEL-Aktivitäten

In diesem Abschnitt werden die durch Abschnitt 7.1.2 identifizierten BPEL-Aktivitäten aufgezählt und ihre Funktion noch einmal kurz beschrieben. Dazu gehört z.B. auch, welche Attribute welchen Typs für die einzelnen Aktivitäten benötigt werden. Generell gilt, dass alle Aktivitäten mindestens die vier Variablen ***dsType***, ***dsKind***, ***dsAddress*** und ***dsStatement*** vom Typ String besitzen. Die Variable ***dsType*** dient zur Angabe des Datenquellentyps, für den die Aktivität ausgeführt wird, also ob es sich um ein Dateisystem, eine Datenbank oder ein Sensornetz handelt. Die Variable ***dsKind*** dient zur Angabe der genaueren Datenquellenart, d.h. um was für ein Dateisystem, was für eine Datenbank oder welche Art von Sensornetz es sich handelt. Diese Informationen sind wichtig, um intern den richtigen SQL-Dialekt bzw. Befehlssatz für die entsprechende Datenquelle auszuwählen. Die Variable ***dsAddress*** dient zur Angabe der Datenquellenadresse und die Variable ***dsStatement*** zur Haltung des entsprechenden Systemaufrufs bzw. SQL- oder XQuery-Befehls, der ausgeführt werden soll. Da jede der definierten Aktivitäten diese vier Variablen besitzt, werden diese nachfolgend als ***simpl-attributes*** (vgl. standard-attributes in [4]) in den Listings angegeben und nur falls benötigt weitere aktivitätsspezifische Variablen beschrieben. Listing 1 führt die vier Datenmanagement-Variablen nocheinmal auf. Weiterhin werden die verschiedenen Ausprägungen der einzelnen Aktivitäten im Hinblick auf die zugrundeliegende Datenquelle aufgezeigt, so dass am Ende ein vollständiger Überblick aller definierten

Aktivitäten und ihrer Ausprägungen vorliegt. Generell gibt es durch die verschiedenen Datenquellen nur wenige strukturelle Unterschiede in den Aktivitäten. Das liegt vor allem daran, dass auf datenquellenspezifische Eigenschaften bereits in der graphischen Oberfläche, also dem Eclipse BPEL Designer, eingegangen werden soll und diese so durch entsprechende Dialoge intern immer auf dieselbe Struktur abgebildet werden können. Eben dazu sollen alle Befehle auch über entsprechende graphische Elemente angegeben werden können, indem sie einfach "zusammengeklickt" werden können (siehe Abbildung 6). Dadurch soll eine einfachere Handhabung realisiert werden, damit der Benutzer schnellstmöglich und ohne detaillierte Kenntnisse der Abfragesprache alle zur Verfügung gestellten Daten- und Datenquellenbefehle für alle unterstützten Datenquellen modellieren kann. Trotz der Einschränkungen, die ohne Zweifel durch die grafische Erstellung der Befehle bestehen, steht dem Benutzer der volle Sprachumfang der jeweiligen Datenquellensprache bzw. der vollständige Befehlssatz der Datenquelle zur Verfügung. Der volle Sprachumfang kann dabei mindestens, wie in Abbildung 6 dargestellt, über die Angabe von Befehlen im "Resulting statement"-Textfeld genutzt werden. So weit es möglich ist, wird allerdings versucht, den vollen Sprachumfang bereits durch die grafische Modellierung der Befehle abzudecken.

Nachfolgend werden alle, durch die Abschnitte 7.1.1 und 7.1.2 identifizierten, Aktivtäten aufgezeigt und beschrieben. Dabei werden in den Listings der DM-Aktivitäten die in [4] beschriebenen Notationen und Definitionen verwendet. Die Schemata der DM-Aktivitäten werden im Verlauf der 1. Iteration vervollständigt.

```
dsType="xsd:string"
dsKind="xsd:string"
dsAdress="anyURI"
dsStatement="xsd:string"
```

Listing 1: Listing der simpl-attributes

Query Aktivität

Diese Aktivität setzt das Query Pattern um und ermöglicht es, aus jeder beliebigen Datenquelle Daten zu lesen und diese auf der Datenquelle zu speichern. Dafür wird ein spezifisches Attribut *queryTarget* benötigt. In diesem Attribut kann der Prozessmodellierer das Ziel (Tabellenname, Dateiname, etc.) hinterlegen, wo die abgefragten Daten auf der Datenquelle gespeichert werden sollen, um diese an einer anderen Stelle im Prozess verwenden zu können. Die Query Aktivität ist für alle Datenquellen gleich strukturiert, und nur die entsprechenden Query-Befehle unterscheiden sich je nach Datenquelle. Diese Aktivität wird auch für das sensornetzinterne Einfügen von Sensordaten in Buffer-Tabellen genutzt (wie auch in Abschnitt 7.1.2 beschrieben). Listing 2 zeigt das BPEL-Schema einer Query Aktivität.

```
<bpel:extensionActivity>
    <simpl:queryActivity standard-attributes
             simpl-attributes queryTarget="xsd:string">

        standard-elements

    </simpl:queryActivity>
</bpel:extensionActivity>
```

Listing 2: Schema einer Query Aktivität

Insert Aktivität

Diese Aktivität setzt die Data IUD Patterns um. Sie ermöglicht es, Daten in eine Datenquelle einzufügen. Listing 3 zeigt das BPEL-Schema einer Insert Aktivität.

```
<bpel:extensionActivity>
    <simpl:insertActivity standard-attributes
             simpl-attributes>

        standard-elements

    </simpl:insertActivity>
</bpel:extensionActivity>
```

Listing 3: Schema einer Insert Aktivität

Update Aktivität

Diese Aktivität setzt die Data IUD Patterns um. Sie ermöglicht es, auf einer Datenquelle hinterlegte Daten zu aktualisieren. Diese Aktivität kann nicht auf Sensornetze angewendet werden. Listing 3 zeigt das BPEL-Schema einer Update Aktivität.

```
<bpel:extensionActivity>
    <simpl:updateActivity standard-attributes
             simpl-attributes>

        standard-elements

    </simpl:updateActivity>
</bpel:extensionActivity>
```

Listing 4: Schema einer Update Aktivität

Delete Aktivität

Diese Aktivität setzt die Data IUD Patterns um. Sie ermöglicht es, Daten in Datenquellen zu löschen. Diese Aktivität kann nicht auf Sensornetze angewendet werden. Listing 5 zeigt das BPEL-Schema einer Delete Aktivität.

```
<bpel:extensionActivity>
    <simpl:deleteActivity standard-attributes
             simpl-attributes>

        standard-elements

    </simpl:deleteActivity>
</bpel:extensionActivity>
```

Listing 5: Schema einer Delete Aktivität

Create Aktivität

Diese Aktivität setzt das Data Setup Pattern um. Sie ermöglicht es, Zuordnungseinheiten (Dateien, Ordner, Tabellen, Schemata, usw.) auf beliebigen Datenquellen zu erstellen. Dabei werden die folgenden Befehle zur Erstellung von Zuordnungseinheiten auf den entsprechenden Datenquellen unterstützt:

- Dateisysteme:
  - MKDIR folder

  - MKFILE file

- Datenbanken:
  - CREATE TABLE
  - CREATE SCHEMA
  - CREATE VIEW
  - optional:
    * CREATE DOMAIN
    * CREATE INDEX
    * CREATE TRIGGER

- Sensornetze:
  - CREATE BUFFER

Diese Aktivität wird für XQuery-Datenbanken nicht realisiert. Ebenso werden in dieser Iterationsstufe keine weiteren Befehle umgesetzt. Listing 6 zeigt das BPEL-Schema einer Create Aktivität.

```
<bpel:extensionActivity>
    <simpl:createActivity standard-attributes
             simpl-attributes>

         standard-elements

    </simpl:createActivity>
</bpel:extensionActivity>
```

Listing 6: Schema einer Create Aktivität

Drop Aktivität

Diese Aktivität setzt das Data Setup Pattern um. Sie ermöglicht es, Zuordnungseinheiten auf beliebigen Datenquellen zu verwerfen. Dazu werden die folgenden Befehle unterstützt:

- Dateisysteme:
  - RMDIR folder
  - RM file

- Datenbanken:
  - DROP TABLE
  - DROP SCHEMA
  - DROP VIEW
  - optional:
    * DROP DOMAIN
    * DROP INDEX
    * DROP TRIGGER

- Sensornetze:

– DROP ALL

Diese Aktivität wird für XQuery-Datenbanken nicht realisiert. Ebenso werden in dieser Iterationsstufe keine weiteren Befehle umgesetzt. Listing 7 zeigt das BPEL-Schema einer Drop Aktivität.

```
<bpel:extensionActivity>
    <simpl:dropActivity standard-attributes
             simpl-attributes>

         standard-elements

    </simpl:dropActivity>
</bpel:extensionActivity>
```

Listing 7: Schema einer Drop Aktivität

Call Aktivität

Diese Aktivität setzt das Stored Procedure Pattern um. Sie ermöglicht es, auf der Datenquelle hinterlegte Prozeduren auszuführen. Diese Aktivität kann vorerst nur für relationale Datenbanken verwendet werden. Listing 8 zeigt das BPEL-Schema einer Call Aktivität.

```
<bpel:extensionActivity>
    <simpl:callActivity standard-attributes
             simpl-attributes>

         standard-elements

    </simpl:callActivity>
</bpel:extensionActivity>
```

Listing 8: Schema einer Call Aktivität

RetrieveData Aktivität

Diese Aktivität setzt die Data Retrieval Patterns um. Sie ermöglicht es, Daten von Datenquellen in den BPEL-Prozess zu laden. Listing 9 zeigt das BPEL-Schema einer RetrieveData Aktivität.

```
<bpel:extensionActivity>
    <simpl:retrieveDataActivity
             standard-attributes
             simpl-attributes>

         standard-elements

    </simpl:retrieveDataActivity>
</bpel:extensionActivity>
```

Listing 9: Schema einer RetrieveData Aktivität

Optionale Aktivitäten:

- **Import Aktivität**: Diese Aktivität setzt das Import Pattern um. Sie ermöglicht es, lokale Daten (z.B. Simulationsparameter) des Benutzers in einen Prozess einzubinden und in diesem zu verwenden.
- **Export Aktivität**: Diese Aktivität setzt das Export Pattern um. Sie ermöglicht es, Daten eines Prozesses (z.B. Simulationsergebnisse) lokal auf den Benutzer-Rechner zu exportieren.
- **Move Aktivität**: Diese Aktivität setzt das Move Pattern um. Sie ermöglicht es, Daten zwischen beliebigen Datenquellen zu kopieren/verschieben, d.h. es können beispielsweise Daten aus einer Datei in eine Datenbank-Tabelle verschoben/kopiert werden.
- **WriteBack Aktivität:** Diese Aktivität setzt die Cache WriteBack Patterns um. Sie ermöglicht es, Daten aus dem BPEL-Prozess auf die originale Datenquelle zurückzuschreiben, nachdem die Daten zuvor über eine RetrieveData Aktivität in den Prozessspeicher geladen wurden. Diese Aktivität existiert nicht für Sensornetze.
- **Alter Aktivität:** Diese Aktivität setzt das Data Setup Pattern um. Sie ermöglicht es, bereits bestehende Zuordnungseinheiten (Tabellen, Schemata, usw.) nachträglich gezielt zu verändern und an veränderte Anforderungen anzupassen. Es ist z.B. möglich, in eine Tabelle neue Spalten einzufügen, bestehende Spalten neu zu benennen oder zu verwerfen.

In Tabelle 4 werden nocheinmal alle resultierenden DM-Aktivitäten und ihre Verwendbarkeit im Bezug auf die verschiedenen Datenquellen aufgezeigt. Dabei steht ein "x" für die vollständige Anwendbarkeit einer Aktivität auf eine Datenquelle, "(x)" steht für die teilweise Verwendbarkeit (z.B. in einer Move Aktivität können Sensornetze nur als Quelle nicht als Ziel angegeben werden) und "o" steht dafür, dass diese Aktivität erst in einer späteren Iterationen auf eine Datenquelle angewendet werden kann. Ein "-" steht dafür, dass eine Aktivität nicht mit einer entsprechenden Datenquelle verwendet werden kann.

Tabelle 4: Übersicht der resultierenden Aktivitäten und ihrer Anwendbarkeit auf verschiedene Datenquellen

Aktivität	RDB	XML-DB	Dateisysteme	Sensornetze (TinyDB)
Query Aktivität	x	x	x	x
Insert Aktivität	x	x	o	-
Update Aktivität	x	x	o	-
Delete Aktivität	x	x	o	-
Create Aktivität	x	o	x	x
Drop Aktivität	x	o	x	x
Call Aktivität	x	o	o	-
RetrieveData Aktivität	x	o	o	x
Move Aktivität	x	x	x	(x)
WriteBack Aktivität	(x)	(x)	(x)	(x)
Alter Aktivität	o	o	o	-

7.2 Authentifizierung und Autorisierung

Dieser Abschnitt behandelt die Authentifizierung und Autorisierung bei dem Zugriff auf Datenquellen und die dabei eingesetzten Technologien und Konzepte. Bei beiden Verfahren handelt es sich um Verfahren auf Seite der Datenquellen, d.h. es findet keine Autorisierung und Authentifizierung der Benutzer innerhalb eines Prozesses statt. Das Rahmenwerk bietet dem Benutzer lediglich die Möglichkeit,

benötigte Authentifizierungs- und Autorisierungsinformationen anzugeben und ggf. zwischenzuspeichern. Ziel des Rahmenwerks ist es, eine Vielzahl an Authentifizierungs- und Autorisierungsverfahren zu unterstützen und für weitere Verfahren erweiterbar zu sein.

7.2.1 Authentifizierung

Bei der Authentifizierung wird vor dem eigentlichen Zugriff zunächst ein Vertrauensverhältnis zwischen Benutzer und Datenquelle hergestellt. Je nach Verfahren wird dabei das Vertrauen einseitig oder auch beidseitig hergestellt. Für das Rahmenwerk soll dabei das Konzept des Single Sign On (SSO) angewendet werden, das nach einer erfolgreich durchgeführten Authentifizierung verhindert, dass bei weiteren Zugriffen eine erneute Authentifizierung durchgeführt werden muss. Unabhängig von der Realisierung, soll das für den Benutzer auch bedeuten, dass die Authentifizierungsinformationen nicht in jeder DM-Aktivität zu einer Datenquelle redundant angegeben werden müssen. Vorerst wird nur ein Verfahren mit Benutzername und Passwort realisiert. Benutzername und Passwort können in der ersten Iteration zunächst nur in den Globalen Einstellungen (siehe 4.1.1) für alle Datenquellen festgelegt werden. In einer weiteren Iterationsstufe wird die separate Angabe dieser Daten für jede Datenquelle möglich sein.

7.2.2 Autorisierung

Eine Autorisierung findet meist nach einer erfolgreichen Authentifizierung statt und überprüft, ob der Zugriff bzw. die Operation, die ausgeführt werden soll, berechtigt ist. Dazu werden Zugriffsrechte überprüft, die als Regeln formuliert auf Seite der Datenquellen verwaltet werden. In der ersten Iteration wird lediglich die Autorisierung über Benutzername und Passwort unterstützt.

7.3 Auditing

In diesem Abschnitt wird eine Beschreibung des geplanten Auditing von SIMPL gegeben. Dazu wird zunächst auf das bestehende Auditing und Monitoring von ODE eingegangen.

7.3.1 Momentane Situation bei ODE

Das Auditing und Monitoring von ODE wird durch sogenannte Events und Event Listener sowie die Management API realisiert.

Management API

Die Management API der ODE Engine macht es möglich, zahlreiche Informationen abzurufen. So ist es beispielsweise möglich, zu überprüfen, welche Prozessmodelle gerade eingesetzt werden und welche Prozessinstanzen ausgeführt werden oder beendet sind. Dies ist besonders für das Monitoring wichtig, da es hierdurch möglich ist, spezifische Informationen zu einem bestimmten Prozess oder einer bestimmten Instanz abzurufen. Dies kann zum Beispiel das Erstellungsdatum des Prozesses oder der Instanz sein, eine Liste der Events, die für diesen Prozess oder diese Instanz generiert wurden, und vieles mehr.

Events

Events sind Ereignisse, die auftreten, wenn bestimmte Aktionen innerhalb der ODE Engine durchgeführt werden, z.B. das Aktivieren eines Prozesses oder das Erzeugen einer neuen Prozessinstanz. Diese Events machen es möglich, zu verfolgen, was innerhalb der ODE Engine passiert, und produzieren detailierte Informationen über die Prozessausführung. Sie können mit Hilfe der Management API oder durch die Nutzung von Event Listenern abgefragt werden.

Event Listener

Event Listener sind bestimmte Konstrukte, die es ermöglichen, bei Auftreten bestimmter Events eine direkte Rückmeldung zu geben, oder auf verschiedene Events zu reagieren und event-spezifische Aktionen durchzuführen.

7.3.2 Auditing von SIMPL

Im BPEL Designer ist der Zugriff auf das Auditing und die Auditingeinstellungen unter dem entsprechenden Menüpunkt möglich. Hier kommt man zum entsprechenden Interface, in dem sich verschiedene Einstellungen für das Auditing tätigen lassen (siehe Abbildung 5). Es ist möglich, das Auditing zu aktivieren oder zu deaktivieren. Weiterhin ist es möglich, die Datenbank für das Speichern der Auditing-Daten festzulegen. Das Auditing ist standardmäßig aktiviert, kann allerdings über die Adminkonsole deaktiviert und auch erneut aktiviert werden.

Für das Auditing von SIMPL werden die von ODE erzeugten Daten direkt an den SIMPL Core übergeben und weiterverarbeitet, anstatt diese wie bisher in der virtuellen Datenbank von ODE abzulegen. Die so erzeugten Daten über die Prozesse, Instanzen und Events werden anschließend in einer zuvor in der Adminkonsole festgelegten Auditing-Datenbank abgelegt.

7.3.3 Event Modell

In diesem Abschnitt wird das Event Modell für das Auditing von SIMPL aufgeführt und erläutert. In Abbildung 25 ist das Event Modell für die Ausführung von DM-Aktivitäten zu sehen. Darin enthalten sind die verschiedenen WS-BPEL Events. Auf diesem Modell aufbauend wird analyisert, welche Events in ODE hinzugefügt werden müssen. Das Modell baut auf dem Modell in Abbildung 24 auf und beschreibt die Besonderheiten, die während der Ausführung (im Zustand “Executing” in Abbildung 24) auftreten. Die Zustände “Ready” und “Waiting” wurden im Event Modell der Ausführung einer DM-Aktivität mit Rot gekennzeichnet, um ersichtlich zu machen, dass dieses Event Modell nur ein Ausschnitt aus dem Modell in 24 ist und die Ausführung vor “Ready” und nach “Waiting” der in diesem Modell gleicht.

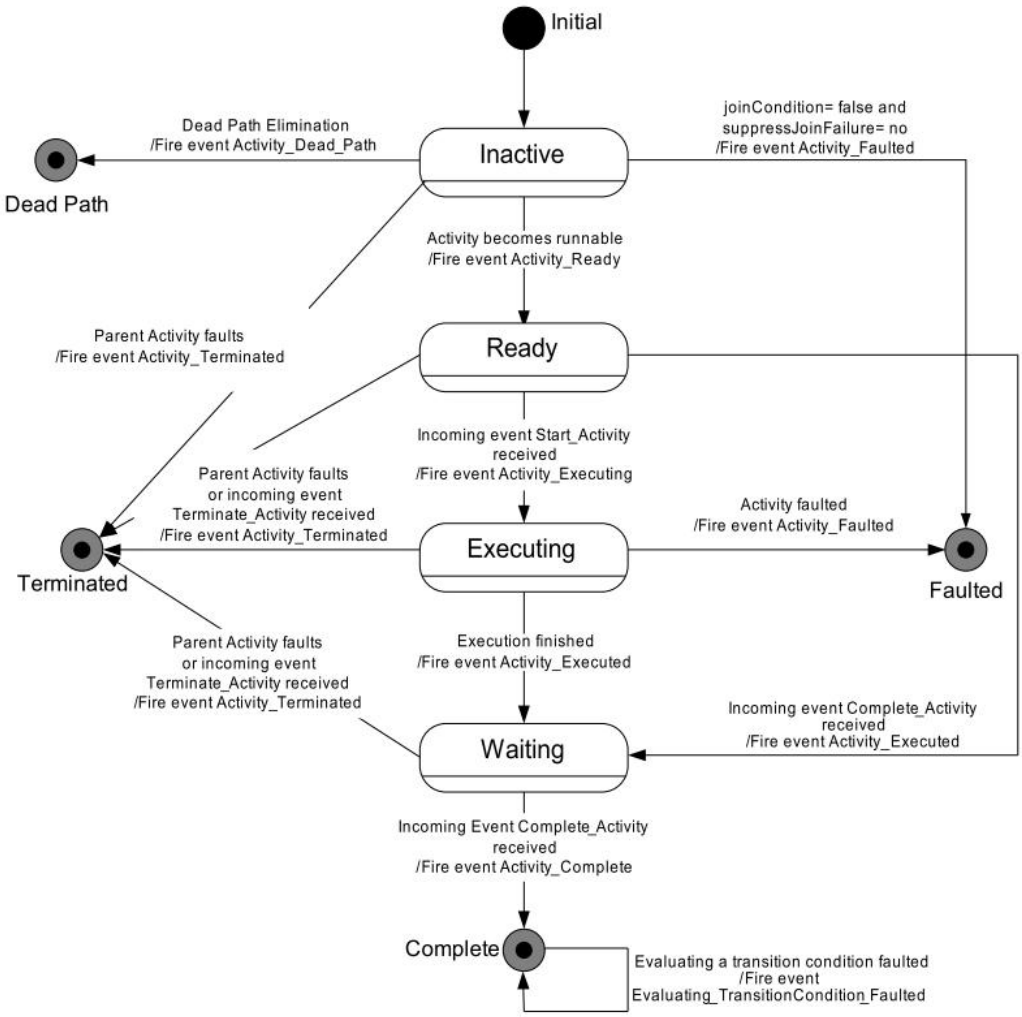


Abbildung 24: BPEL Event Modell für die Ausführung allgemeiner Aktivitäten in WS-BPEL 2.0

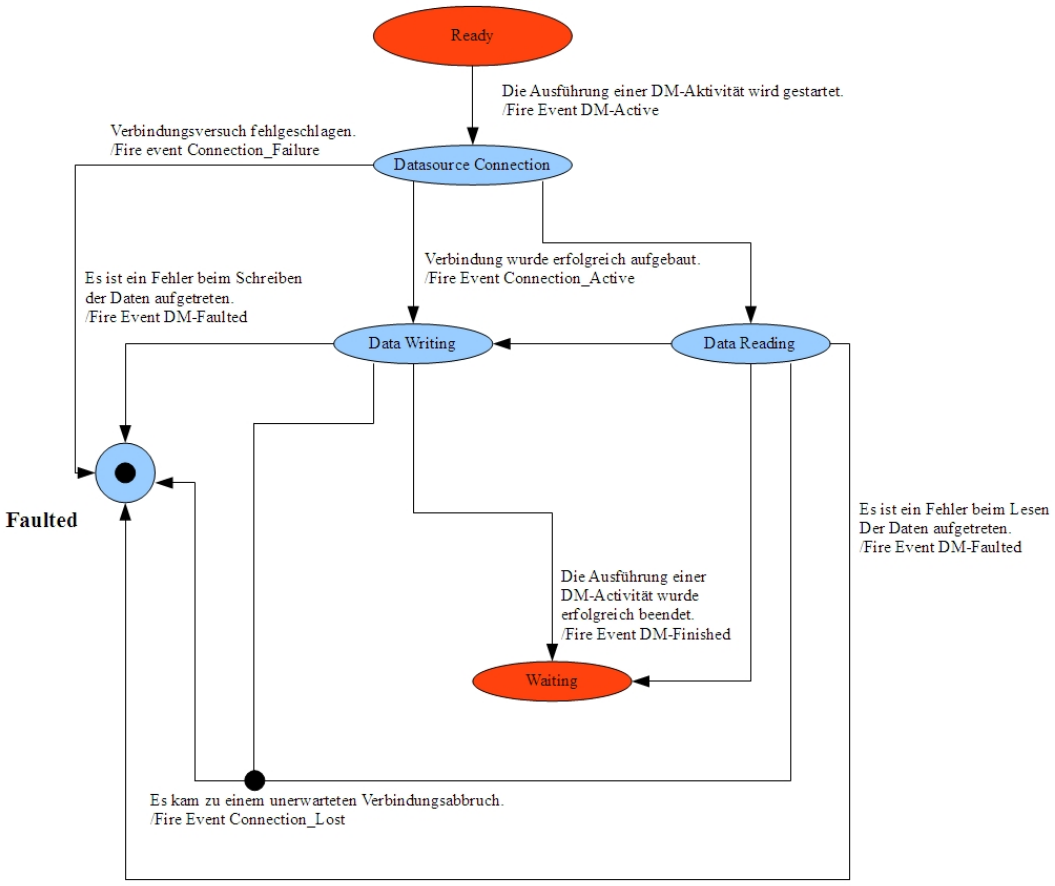


Abbildung 25: Event-Modell für die Ausführung von DM-Aktivitäten

Aus den Events im BPEL Event Modell der Ausführung folgt, dass zwei neue Event Klassen hinzugefügt werden müssen. Dies sind die Event Klassen Connection_Events und DM_Events. Jede dieser beiden Event Klassen verfügt über eine Anzahl von einzelnen Events, die nachfolgend genauer erläutert werden.

Connection_Events Diese Events werden erzeugt, um verschiedene Ereignisse bei der Verbindung zu einer Datenquelle zu erfassen.

- Connection_Failure_Event

Dieses Event wird erzeugt, wenn ein Verbindungsversuch fehlschlägt.

- Connection_Active_Event

Dieses Event wird erzeugt, wenn erfolgreich eine Verbindung aufgebaut wurde.

- Connection_Closed_Event

Dieses Event wird erzeugt, wenn die Verbindung zu einer Datenquelle beendet wurde.

- Connection_Lost_Event

Dieses Event wird erzeugt, wenn es zu einen unerwarteten Abbruch der Verbindung kommt. Beispielsweise wenn es zu einem Timeout der Verbindung kommt.

DM_Events Die DM_Events sind für die Ereignisse einer DM-Aktivität zuständig. Hier wird nicht unterschieden, um welche Art von DM-Aktivität es sich handelt. Für alle DM-Aktivitäten werden dieselben Events erzeugt. Diese Events enthalten jeweils Informationen über den Namen und den Typ der Aktivität. Diese speziellen Events werden zusätzlich zu den normalen Activity_Events erzeugt, um dem Nutzer die Möglichkeit zu bieten sich nur die Events der DM-Aktivitäten und nicht aller Aktivitäten anzeigen zu lassen.

- DM_Finished

Dieses Event wird erzeugt, wenn die Ausführung einer DM-Aktivität erfolgreich beendet wurde.

- DM_Started

Dieses Event wird erzeugt, wenn die Ausführung einer DM-Aktivität gestartet wurde.

- DM_Failure

Dieses Event wird erzeugt, wenn bei der Ausführung einer DM-Aktivität ein Fehler auftrat.

8 Technologien und Werkzeuge

In diesem Kapitel folgt ein Überblick über die im Projekt verwendeten Technologien und Werkzeuge.

8.1 Technologien

In der Entwicklung von SIMPL werden die folgenden Technologien zum Einsatz kommen:

Apache Axis2 [5]

Axis2 ist eine Simple Object Access Protocol (SOAP-) Engine zur Konstruktion von darauf basierenden Web Services und Client-Anwendungen.

Apache ODE [6]

Apache ODE ist eine BPEL-Workflow-Engine. Es wird sowohl WS-BPEL 2.0 als auch BPEL4WS 1.1 unterstützt, um Prozesse in einer SOA auszuführen. Zudem ist ein Deployment von Prozessen zur Laufzeit (Hot Deployment) möglich sowie die Analyse und Validierung von Prozessen.

Apache Tomcat [7]

Apache Tomcat ist ein Web Container, der auch einen kompletten Web Server beinhaltet.

Apache Tuscany DAS [8]

Data Access Services ermöglichen den einheitlichen Zugriff auf Datenquellen, indem eine zentrale Schnittstelle für das Lesen und Schreiben von heterogenen Daten bereitgestellt wird. Die Daten werden dabei in Service Data Objects gekapselt und so von ihrer Quelle unabhängig gemacht.

Apache Tuscany SDO [9]

Service Data Objects stellen eine einheitliche API zur Verfügung, mit der die Handhabung verschiedener heterogener Daten und Datenquellen vereinfacht wird.

Eclipse BPEL Designer [10]

Der Eclipse BPEL Designer ist ein Eclipse Plugin für die Modellierung von BPEL-Prozessen. Er verfügt über eine leicht verständliche GUI und über eine Syntaxprüfung für BPEL-Prozesse. Zudem bietet der Eclipse BPEL Designer in Kombination mit Apache ODE eine Schritt für Schritt Ausführung von Prozessen.

IBM-DB2 [11]

Die IBM-DB2 ist ein Hybriddatenserver, mit dem sowohl die Verwaltung von XML-Daten als auch von relationalen Daten möglich ist.

Java 6 [12]

Java ist eine objektorientierte Programmiersprache von Sun Microsystems, mit der sich plattformunabhängige Programme entwickeln lassen.

JAX-WS [13]

JAX-WS ist eine Java API zur Erstellung von Web Services, die für das Deployment Java Annotationen verwendet.

TinyDB [14]

TinyDB ist ein Datenbanksystem für Sensornetze und bietet eine SQL-ähnliche Schnittstelle.

Web Services

Web Services sind eigenständige Software-Einheiten, die sich selbst beschreiben und die ihre Dienste über ein Netzwerk, wie beispielsweise das Internet, bereitstellen. Damit werden verteilte Anwendungen möglich, die flexibel auf sich ändernde Anforderungen angepasst werden können. Die Schnittstellen von Web Services werden mit WSDL (siehe [21]) beschrieben und können damit unabhängig von Betriebssystem, Plattform und Programmiersprache verwendet werden.

8.2 Werkzeuge

Es werden folgende Werkzeuge eingesetzt, um den Entwicklungsvorgang und die Dokumentation zu unterstützen:

Apache Maven [15]

Maven ist ein Projekt-Management-Tool zur standardisierten Erstellung und Verwaltung von Java-Programmen. Mit Maven ist es möglich, viele Schritte, die die Entwickler normalerweise von Hand erledigen müssen, zu automatisieren.

Eclipse [16]

Eclipse ist ein Integrated Development Environment (IDE) für Java und auch andere Programmiersprachen. Für SIMPL wird Eclipse in der Version 3.5 (Galileo) verwendet.

Hudson [17]

Hudson ist ein webbasiertes System zur kontinuierlichen Integration von Softwareprojekten.

LyX [18]

LyX ist ein Textverarbeitungsprogramm, mit dem es möglich ist, auf einfache Art und Weise LaTeX-Dokumente zu erstellen.

PDF-XChange Viewer [19]

Mit dem PDF-XChange Viewer lassen sich PDF-Dateien nicht nur öffnen, lesen und drucken, sondern zusätzlich Kommentare, Notizen und Markierungen vornehmen.

Subversion [20]

Subversion ist eine Software zur Versionskontrolle und eine Weiterentwicklung von CVS. Es wird genutzt, um mehreren Nutzern den gleichzeitigen Zugriff auf und ein gleichzeitiges Bearbeiten von verschiedenen Dateien und Dokumenten zu ermöglichen.

Literatur

[1] SIMPL Angebot, Version 1.2, 25.September 2009

[2] Vrhovnik, M.; Schwarz, H.; Radeschütz, S.; Mitschang, B.: *An Overview of SQL Support in Workflow Products*. In: Proc. of the 24th International Conference on Data Engineering (ICDE 2008), Cancún, México, 7.April - 12. April 2008

[3] Wieland, M.; Görlach, K.; Schumm, D.; Leymann, F.: *Towards Reference Passing in Web Service and Workflow-based Applications*. In: Proc. of the 13th IEEE Enterprise Distributed Object Conference (EDOC 2009), Auckland, New Zealand, 31.August - 04.September 2009.

[4] Jordan, D.; Evdemon, J.: *Web Services Business Process Execution Language Version 2.0*, OASIS Standard. Organization for the Advancement of Structured Information Standards (OASIS), 11.April 2007. http://docs.oasis-open.org/ wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, zuletzt zugegriffen am 04.11.2009

[5] Axis2, http://ws.apache.org/axis2/, zuletzt zugegriffen am 17.10.2009

[6] Apache ODE, http://ode.apache.org/, zuletzt zugegriffen am 17.10.2009

[7] Apache Tomcat, http://tomcat.apache.org/, zuletzt zugegriffen am 17.10.2009

[8] Apache Tuscany DAS, http://tuscany.apache.org/das-overview.html, zuletzt zugegriffen am 17.10.2009

[9] Apache Tuscany SDO, http://tuscany.apache.org/sdo-overview.html, zuletzt zugegriffen am 17.10.2009

[10] Eclipse BPEL Designer, http://www.eclipse.org/bpel/, zuletzt zugegriffen am 17.10.2009

[11] IBM-DB2, http://www.ibm.com/db2/, zuletzt zugegriffen am 17.10.2009

[12] Java 6, http://java.sun.com/, zuletzt zugegriffen am 17.10.2009

[13] JAX-WS, https://jax-ws.dev.java.net/, zuletzt zugegriffen am 17.10.2009

[14] TinyDB, http://telegraph.cs.berkeley.edu/tinydb/, zuletzt zugegriffen am 17.10.2009

[15] Apache Maven, http://maven.apache.org/, zuletzt zugegriffen am 18.10.2009

[16] Eclipse, http://www.eclipse.org/, zuletzt zugegriffen am 18.10.2009

[17] Hudson, https://hudson.dev.java.net/, zuletzt zugegriffen am 18.10.2009

[18] LyX, http://www.lyx.org/, zuletzt zugegriffen am 18.10.2009

[19] PDF-XChange Viewer, http://pdf-xchange-viewer.softonic.de/, zuletzt zugegriffen am 18.10.2009

[20] Subversion, http://subversion.tigris.org/, zuletzt zugegriffen am 18.10.2009

[21] WSDL, http://www.w3.org/standards/techs/wsdl, zuletzt zugegriffen am 04.11.2009

[22] XQuery Update Facility, http://www.w3.org/standards/techs/xquery, zuletzt zugegriffen am 06.11.2009

[23] Steinmetz, Th.: *Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE*. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 2729, 02.August 2008.

[24] XPath, http://www.w3.org/standards/techs/xpath, zuletzt zugegriffen am 30.01.2010

Abkürzungsverzeichnis

API	Application Programming Interface
BPEL	Business Process Execution Language
DAS	Data Access Service
DDL	Data Definition Language
DM	Data Management
FLWOR	FOR, LET, WHERE, ORDER, RETURN
IDE	Integrated Development Environment
IUD	INSERT, UPDATE, DELETE
ODE	Orchestration Director Engine
RDB	Relational Database
RRS	Reference Resolution System
SDO	Service Data Object
SIMPL	SimTech: Information Management, Processes and Languages
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSO	Single Sign On
UDDI	Universal Description, Discovery and Integration
URI	Unified Resource Identifier
URL	Unified Resource Locator
WS	Web Service
WSDL	Web Service Description Language
XML	Extensible Markup Language
XPath	XML Path Language
XQUERY	XML Query Language

Abbildungsverzeichnis

Listings

Tabellenverzeichnis