

Spezifikation

Version 2.3

02. März 2010



Inhaltsverzeichnis

1	Einleitung	6
1.1	Zweck des Dokuments	6
1.2	Einsatzbereich und Ziele	6
1.3	Evolution des Dokuments	6
1.3.1	Iteration 1	6
1.3.2	Iteration 2	7
1.3.3	Mögliche Erweiterungen	7
1.4	Definitionen	7
1.5	Überblick	8
2	Allgemeine Beschreibung	9
2.1	Einbettung in die Systemumgebung	9
2.2	Funktionen	10
2.3	Sprache	11
2.4	Distributionsform und Installation	11
2.5	Benutzerprofile	11
2.6	Einschränkungen	11
3	Nichtfunktionale Anforderungen	12
3.1	Mengengerüst	12
3.2	Benutzbarkeit	12
3.3	Verfügbarkeit	12
3.4	Robustheit	12
3.5	Sicherheit	12
3.6	Portabilität	13
3.7	Erweiterbarkeit	13
3.8	Wartbarkeit	13
3.9	Skalierbarkeit	13
4	Benutzeroberflächen von SIMPL	14
4.1	Benutzeroberfläche des erweiterten Eclipse BPEL Designers	14
4.1.1	Erweiterungen zur Integration von Referenzvariablen in den Eclipse BPEL Designer	14
4.1.2	ODE Deployment-Deskriptor	15
4.2	Benutzeroberfläche der SIMPL Eclipse Plug-Ins	18
4.2.1	Admin-Konsole	19
4.2.2	Einstellungen der SIMPL Eclipse Plug-Ins	19
4.2.3	Hilfe	20
4.2.4	About	20
4.2.5	Nachträgliches Laden der Plug-In Daten	21
4.2.6	Data-Management-Aktivitäten	21
4.3	Benutzeroberfläche des RRS Eclipse Plug-Ins	21
4.4	Benutzeroberfläche der Datenquellen-Registry	22
4.4.1	Datenquellen-Registry Plug-In	23
4.4.2	Datenquellen-Registry Web Interface	23
5	Akteure	25
5.1	Prozess-Modellierer	25
5.2	Workflow-Administrator	25
5.3	ODE Workflow-Engine	25
5.4	Eclipse BPEL Designer	25
5.5	Reference Resolution System	25

5.6	Datenquellen-Administrator	25
6	Anwendungsfälle (Use-Cases)	26
6.1	Diagramm der Anwendungsfälle	26
6.2	Anwendungsfälle des Prozess-Modellierers	26
6.2.1	Data-Management-Aktivität erstellen	28
6.2.2	Data-Management-Aktivität bearbeiten	28
6.2.3	Data-Management-Aktivität löschen	29
6.2.4	ODE Deployment-Deskriptor erstellen	29
6.2.5	ODE Deployment-Deskriptor bearbeiten	29
6.2.6	ODE Deployment-Deskriptor löschen	30
6.2.7	Prozess auf ODE-Server deployen	30
6.2.8	Prozessinstanz starten	30
6.2.9	Strategie für das Late-Binding auswählen	31
6.3	Anwendungsfälle des Workflow-Administrators	31
6.3.1	Admin-Konsole öffnen	32
6.3.2	Auditing aktivieren	33
6.3.3	Auditing deaktivieren	33
6.3.4	Auditing-Datenbank festlegen/ändern	33
6.3.5	Globale Einstellungen festlegen/ändern	33
6.3.6	Einstellungen der Admin-Konsole speichern	34
6.3.7	Einstellungen der Admin-Konsole zurücksetzen	34
6.3.8	Default-Einstellungen der Admin-Konsole laden	34
6.3.9	Admin-Konsole schließen	35
6.3.10	Neue Referenz in RRS einfügen	35
6.3.11	Referenz aus RRS bearbeiten	36
6.3.12	Referenz aus RRS löschen	36
6.4	Anwendungsfälle der ODE Workflow-Engine	36
6.4.1	Data-Management-Aktivität ausführen	38
6.4.2	Data-Management-Aktivität zurücksetzen	39
6.5	Anwendungsfälle des Eclipse BPEL Designers	39
6.5.1	SIMPL Admin-Konsole laden	41
6.5.2	SIMPL Admin-Konsole Defaults laden	41
6.5.3	SIMPL Admin-Konsole speichern	42
6.5.4	SIMPL Extensions validieren	42
6.5.5	Datenquellen aus UDDI-Registry abrufen	42
6.5.6	BPEL-Datei transformieren	43
6.6	Anwendungsfälle des RRS	43
6.6.1	Referenz auflösen/dereferenzieren	44
6.6.2	Referenz speichern	44
6.6.3	Referenz laden	44
6.6.4	Referenz löschen	44
6.7	Anwendungsfälle der Datenquellen-Administratoren	45
6.7.1	Datenquelle in UDDI-Registry registrieren	45
6.7.2	Datenquelle aus UDDI-Registry entfernen	46
6.7.3	Datenquelle aus UDDI-Registry bearbeiten	46
6.8	Komponenten und ihre Anwendungsfälle	46

7	Konzepte und Realisierungen	49
7.1	Daten-Referenzen in BPEL (IAAS-Referenzen)	49
7.1.1	Reference Resolution System	49
7.1.2	Referenzen und Endpoint References	51
7.1.3	Realisierung von Referenzen in BPEL	52
7.2	Container-Referenzen in BPEL (IPVS-Referenzen)	59
7.3	Data-Management-Aktivitäten	62
7.3.1	Datenmanagement-Patterns	62
7.3.2	Umsetzung der Datenmanagement-Patterns	76
7.3.3	Resultierende BPEL-Aktivitäten	84
7.4	Authentifizierung und Autorisierung	89
7.4.1	Authentifizierung	89
7.4.2	Autorisierung	89
7.5	Auditing	89
7.5.1	Momentane Situation bei ODE	90
7.5.2	Auditing von SIMPL	90
7.5.3	Event Modell	90
7.6	Datenquellen Policies	94
7.6.1	Policy-Beschreibungen	94
7.6.2	Policy Defenition	95
8	Technologien und Werkzeuge	97
8.1	Technologien	97
8.2	Werkzeuge	98
	Literaturverzeichnis	99
	Abkürzungsverzeichnis	101
	Abbildungsverzeichnis	102
	Verzeichnis der Listings	103
	Tabellenverzeichnis	104

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.
1.5	05.02.2010	rehnre	Korrektur von Kapitel 7.5.
2.0	06.02.2010	hahnml	Erweiterung der Spezifikation auf die 2.Iteration.
2.1	01.03.2010	schneimi	Use-Case-Diagramme und einige Use Cases überarbeitet.
2.2	01.03.2010	schneimi, huettiwg, rehnre, hahnml	Abschließende Bearbeitung der Spezifikation.
2.3	02.03.2010	hahnml, huettiwg	Überarbeitung der Spezifikation.
2.4	27.05.2010	hahnml	Korrektur der Spezifikation anhand der Kundenkommentare zur 2.Iteration. Einfügen von Kapitel 4.1.

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 Language (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

Die vorliegende Spezifikation beschreibt die Anforderungen, die im SIMPL-Rahmenwerk umgesetzt werden, sowie einige mögliche Erweiterungen, die später noch hinzukommen können.

1.3.1 Iteration 1

Folgende Funktionen bilden die Funktionalität des SIMPL-Rahmenwerks nach der 1. Iteration:

- Umsetzung eines Plug-In System für die Anbindung von verschiedenen Datenquellen.
- Die statische Anbindung von Datenquellen, wobei vorerst nur relationale Datenbanken unterstützt werden sollen.
- Bereitstellung von generischen BPEL-Aktivitäten im Eclipse BPEL Designer für den Zugriff auf Datenquellen.

- Bereitstellung von grafischen Abfragebefehls-Editoren für die verschiedenen DM-Aktivitäten.
- Umsetzung einer grundlegenden Admin-Konsole, in der das Auditing aktiviert werden kann und die Zieldatenquelle und deren Authentifizierungsinformationen für das Auditing festgelegt werden können.

1.3.2 Iteration 2

Die Funktionalität des SIMPL-Rahmenwerks wird in der 2.Iteration um folgende Funktionen erweitert:

- 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.

1.3.3 Mögliche Erweiterungen

Im Folgenden werden einige mögliche Erweiterungen beschrieben. Auf einige dieser Erweiterungsmöglichkeiten wird in den entsprechenden Kapiteln näher eingegangen.

- Einstellung der Granularität des Auditings.
- Validierung der DM-Aktivitäten im Eclipse BPEL Designer.
- 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 7.3.3 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 7.3.3 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 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 2.2 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 gehören 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 den BPEL-DM Extension Activities, 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, Apache Derby) und Dateisysteme (CSV-Dateiformat) und kann um weitere Datenquellen, wie XML-Datenbanken (IBM DB2 mit pureXML-Technologie) 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 wissenschaftlichen 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 Communication 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 Communication Plug-In realisiert die Verbindung mit dem SIMPL Core und wird sowohl für das SIMPL Core Plug-In (Laden und Speichern von Rahmenwerkseinstellungen), als 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 (BPEL-DM Extension Activities) eingebunden. Dabei läuft die benötigte Software auf dem lokalen Rechner des Benutzers, die Datenquellen können auf verschiedene Server verteilt sein.

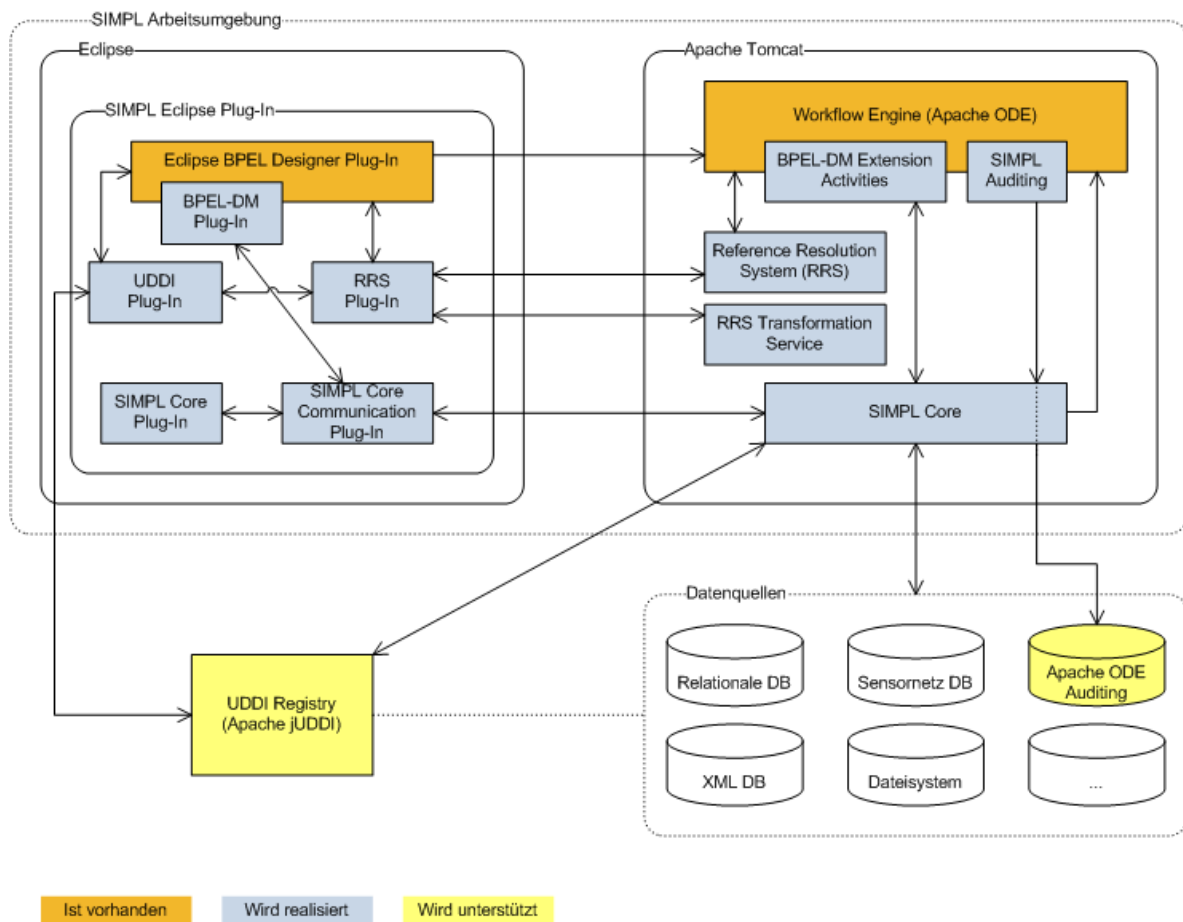


Abbildung 1: Übersicht über die Systemumgebung von SIMPL

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. Dabei können die Datenquellen sowohl statisch als auch dynamisch angebunden werden. Für die dynamische Anbindung von Datenquellen wird eine Datenquellen-Registry bereitgestellt, die es ermöglicht, dass Datenquellen im Modellierungswerkzeug einfach ausgewählt werden können oder automatisch zur Laufzeit angebunden werden. Eine nähere Beschreibung der DM-Aktivitäten wird in Abschnitt 4.2.6 gegeben.

Alle Ereignisse, die während der Laufzeit eines Prozesses auftreten (z.B. Ausführung einer DM-Aktivität), 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 Installationspakets 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 erleichtern. 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.
- Die SIMPL Eclipse Plug-Ins sind nur voll funktionsfähig, falls der SIMPL Core erreichbar ist, d.h. dass ein Apache Tomcat Server mit angebundenem SIMPL Core gestartet wurde.
- 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 7.4 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).
- Das Plug-In System des SIMPL Cores, über das Plug-Ins zur Unterstützung von weiteren Datenquellentypen einfach angebunden werden können.
- Die Verwendung des Eclipse Plug-In Mechanismus für die Admin-Konsole.

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.

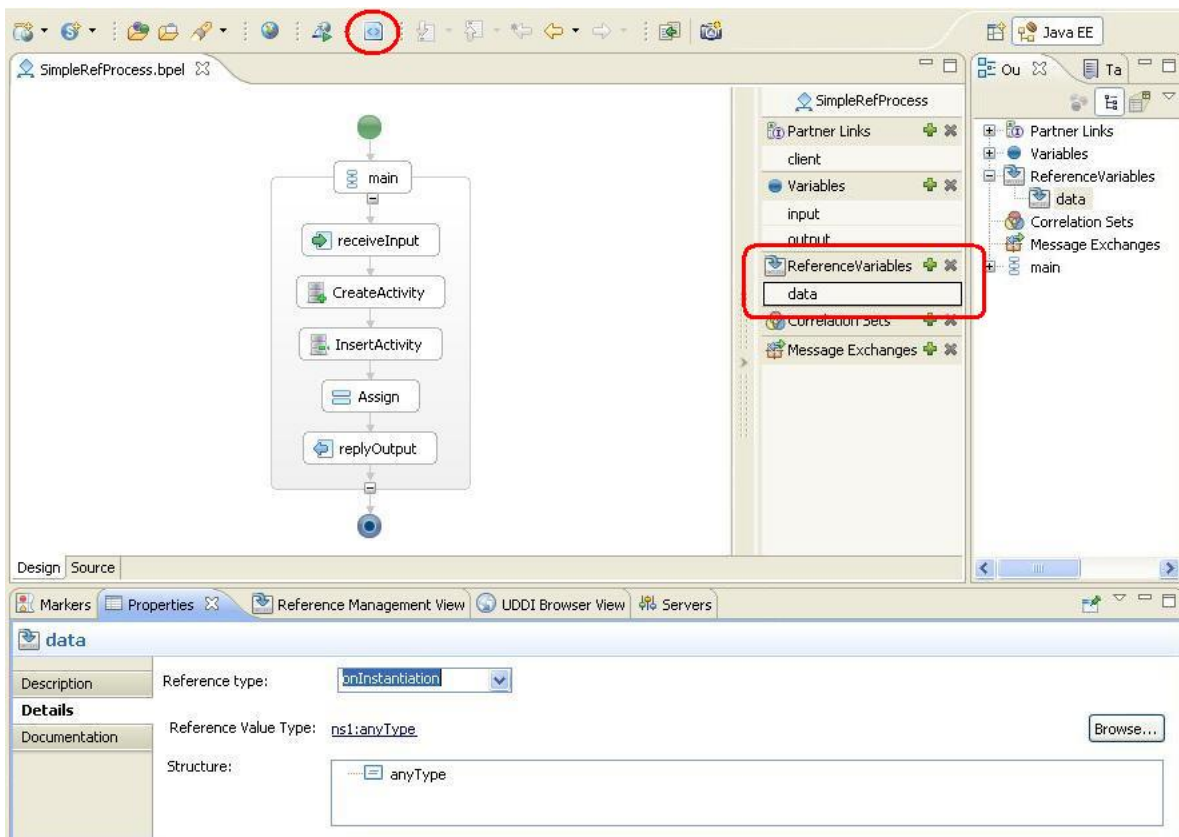


Abbildung 2: Um Referenzvariablen erweiterter Eclipse BPEL Designer

4 Benutzeroberflächen von SIMPL

In diesem Kapitel werden alle Benutzeroberflächen, die durch SIMPL bereitgestellt oder erweitert werden, beschrieben.

4.1 Benutzeroberfläche des erweiterten Eclipse BPEL Designers

Dieses Kapitel beschreibt alle Erweiterungen des Eclipse BPEL Designers und dessen Benutzeroberfläche.

4.1.1 Erweiterungen zur Integration von Referenzvariablen in den Eclipse BPEL Designer

In diesem Kapitel werden alle Erweiterungen des Eclipse BPEL Designers beschrieben, die für die Modellierung und Verwendung von Referenzvariablen in BPEL-Prozessen benötigt werden. Abbildung 2 zeigt den erweiterten BPEL Editor des Eclipse BPEL Designers. Auf die neuen Elemente wird dabei im folgenden näher eingegangen.

Das rot umrandete Symbol in der Toolbar stößt die Transformation eines Prozesses mit Referenzvariablen an. Was genau während der Transformation geschieht und wie diese durchgeführt wird, wird in Kapitel 7.1.3 beschrieben. Damit Referenzvariablen überhaupt modelliert werden können, muss ein neuer Variablentyp in BPEL eingeführt werden. Da Referenzvariablen eine ähnliche Struktur wie Standard-BPEL-Variablen haben, werden Referenzvariablen analog realisiert. Listing 1 zeigt die Struktur einer Referenzvariablen sowie deren Container (*bpel:referenceVariables*). Eine Referenzvariable hat

dabei immer einen Namen (*name*), einen Referenztyp (*referenceType*) und einen Werttyp (*valueType*). Die anderen Attribute wurden nicht in die Benutzeroberfläche integriert, sind aber im Modell vorhanden, sodass durch eine Erweiterung der Benutzeroberfläche diese Attribute verwendet werden können. Die Bedeutung der einzelnen Attribute liefert Kapitel 7.1.3.

Die Zuordnung zwischen Referenzvariablen und Referenzen wird über den Namen realisiert, d.h. eine Referenzvariable mit dem Namen “myData” wird bei der Transformation auf die gleichnamige Referenz (logischer Namen) abgebildet. Der Hintergrund dafür ist, dass die Zuweisung automatisch abläuft und keine weiteren Attribute in den Referenzvariablen zur Angabe einer Referenz benötigt werden.

Analog zu den Standard-BPEL-Variablen können durch die Erweiterung nun Referenzvariablen erzeugt, ausgewählt und gelöscht werden (siehe rot umrandeter Bereich in Abbildung 2). Nachdem eine Referenzvariable erstellt oder ausgewählt ist, können deren Attribute in der Properties-View gesetzt werden (siehe Abbildung 2 unten). Dabei kann als Referenztyp *onInstantiation* oder *fresh* ausgewählt werden und der Werttyp über den Browse-Button analog wie bei Standard-BPEL-Variablen gesetzt werden. Damit man auch Referenzvariablen in Kommunikationsaktivitäten (Invoke, Reply und Receive) verwenden kann, wurde der Eclipse BPEL Designer entsprechend erweitert, sodass auch Referenzvariablen bei der Modellierung zur Auswahl stehen. Generell sind die durchgeführten Erweiterungen nur eine Übergangslösung, da das zugrundeliegende BPEL-Modell nicht für mehrere verschiedene Variablenarten ausgelegt ist und sich dies auch in der Implementierung des Eclipse BPEL Designer niederschlägt. Eine mögliche Lösung dieses Problems würde die Umstrukturierung des zugrundeliegenden BPEL-Modells liefern. D.h. man sollte alle gemeinsamen Eigenschaften von Referenzvariablen und Standard-BPEL-Variablen in einer Oberklasse abstrahieren, sodass generell zwischen verschiedenen Variablenarten unterschieden werden kann und so auch die Implementierung übersichtlicher, konsistenter und in Zukunft einfacher erweiterbar wird.

```
<bpel:referenceVariables>
  <bpel:referenceVariable
    name = "xsd:string"
    referenceType = "bpel:ReferenceType"
    valueType = "xsd:schema"
    period = "duration"
    external = "partnerLink">
  </bpel:referenceVariable>
</bpel:referenceVariables>
```

Listing 1: Schema der ReferenceVariables- und ReferenceVariable-Elemente

4.1.2 ODE Deployment-Deskriptor

Über den ODE Deployment-Deskriptor sollen in Zukunft Daten für DM-Aktivitäten eines BPEL-Prozesses, wie z.B. Datenquellen-Informationen, Auditing-Parameter und die Zuordnung von Late-Binding Strategien und Policies, spezifiziert werden. Abbildung 3 zeigt die Benutzeroberfläche des erweiterten DD (Deployment-Deskriptors). Das erweiterte Schema des DDs zeigt Listing 2. Das Attribut *attachedUddiAddress* wird im Hintergrund auf den in den Einstellungen hinterlegten Wert der UDDI-Adresse gesetzt. Dies ist notwendig, damit in ODE zur Laufzeit die Adresse der angebundenen UDDI bekannt ist und das Late-Binding sowie statisch angebundene UDDI-Datenquellen verwendet werden können.

Im erweiterten DD können nun Datenquellen-Informationen wie die physikalische Adresse, der Typ, der Subtyp, die Abfragesprache, das Datenformat und Authentifizierungsinformationen (Benutzername, Passwort) einer Datenquelle (siehe Listing 3) an einen logischen Namen gebunden werden. Dieser logische Namen kann dann später einfach in jeder DM-Aktivität ausgewählt werden und die entsprechenden Parameter werden automatisch in der Aktivität gesetzt. Die Datenquellen-Einträge können über entsprechende Dialoge angelegt oder editiert werden (siehe Abbildung 4).

The screenshot shows a web-based configuration interface for a BPEL process named 'StaticDsProcess'. The interface has a tabbed header with 'StaticDsProcess.bpel' and 'deploy.xml'. The main content area is divided into several sections:

- General:** Contains two dropdown menus. 'This process is' is set to 'activated' and 'The auditing of this process is' is set to 'deactivated'. There is also a checkbox 'Run this process in memory' which is unchecked.
- Inbound Interfaces (Services):** A section header with a right-pointing arrow.
- Outbound Interfaces (Invokes):** A section header with a right-pointing arrow.
- Process-level Monitoring Events:** A section header with a right-pointing arrow.
- Scope-level Monitoring Events:** A section header with a right-pointing arrow.
- Data source specification:** A section with a description: 'The table contains data sources which are used in the process. Specify a name, the address, the username and the password of each data source to use them in the process.' It contains a table with columns: Name, Address, Type, Subt..., Lang..., Data..., User..., Pass..., and buttons for New, Edit, and Remove. One row is filled with: mySQL, localhost:3306/test, Data..., MySQL, SQL, RDB, test, ****.
- Activity-Data source mapping:** A section with a description: 'The table contains mappings between SIMPL DataManagement Activities and Policies to support late binding or to map the activities with the above specified data sources.' It contains a table with columns: Activity, Policy (local path), Strategy, and buttons for New, Edit, and Remove. The table is currently empty.

The bottom of the window shows the tab 'StaticDsProcess'.

Abbildung 3: Erweiterter Deployment-Deskriptor

Ebenso kann im erweiterten DD das Late-Binding von Datenquellen realisiert werden. Dazu werden über entsprechende Dialoge (siehe Abbildung 5) sogenannte Activity-Mappings angelegt oder editiert. In einem Activity-Mapping (siehe Listing 4) wird eine DM-Aktivität aus dem zugrundeliegenden Prozessmodell ausgewählt und mit einer Strategie und einer Policy-Datei verknüpft. Die Strategie kann direkt und die Policy-Datei über einen entsprechenden Datei-Browser ausgewählt werden. Die Daten der Policy-Datei werden im DD hinterlegt und können so beim Deployment ausgelesen werden. Der Inhalt der Policy-Dateien muss dabei den in Kapitel 7.6 vorgegebenen Konventionen entsprechen.

Im erweiterten DD kann auch das Auditing für das aktuell modellierte Prozessmodell aktiviert und deaktiviert werden. Der Unterschied zur (De-)Aktivierung des Auditings in der SIMPL Admin-Konsole (siehe Kapitel) liegt darin, dass hier das Auditing für einzelne Prozessmodelle (de-)aktiviert werden kann, wobei über die Admin-Konsole das Auditing für alle deployten Prozessmodelle (de-)aktiviert wird.

```
<process name = "xsd:string" attachedUddiAddress =
  "xsd:string">

  <datasources*/>
  <activityMappings*/>
  <auditing>xsd:boolean</auditing>

  standard-deployment-descriptor-elements

</process>
```


Abbildung 4: Dialog für das Hinzufügen einer Datenquellen-Beschreibung

Abbildung 5: Dialog für das Hinzufügen eines Activity Mappings

Listing 2: Schema des erweiterten Deployment-Deskriptors

```
<datasources dataSourceName = "xsd:string"
  address = "xsd:string" type = "xsd:string"
  subtype = "xsd:string" language = "xsd:string"
  userName = "xsd:string" password = "xsd:string"
  format = "xsd:string"/>
```

Listing 3: Schema der im Deployment-Deskriptor hinterlegten Datenquelleninformationen

```
<activityMappings activity = "xsd:string"
  strategy = "dd:STRATEGY_TYPE">
```

```

    <policy policyData = "xsd:string"
      localPath = "xsd:string"/>
  </activityMappings>

```

Listing 4: Schema der im Deployment-Deskriptor hinterlegten Activity Mappings

4.2 Benutzeroberfläche der SIMPL Eclipse Plug-Ins

In diesem Kapitel wird die grafische Benutzeroberfläche der SIMPL Eclipse Plug-Ins beschrieben. Abbildung 6 zeigt den erweiterten Eclipse BPEL Designer und das SIMPL Menü. In der Palette befinden sich die SIMPL DM-Aktivitäten, die wie bereits vorhandene Aktivitäten zur Modellierung von Prozessen verwendet werden können. Weiterhin wird das SIMPL Menü bereitgestellt, über das alle wichtigen Einstellungen und Informationen des SIMPL Rahmenwerks vorgenommen bzw. angezeigt werden können.

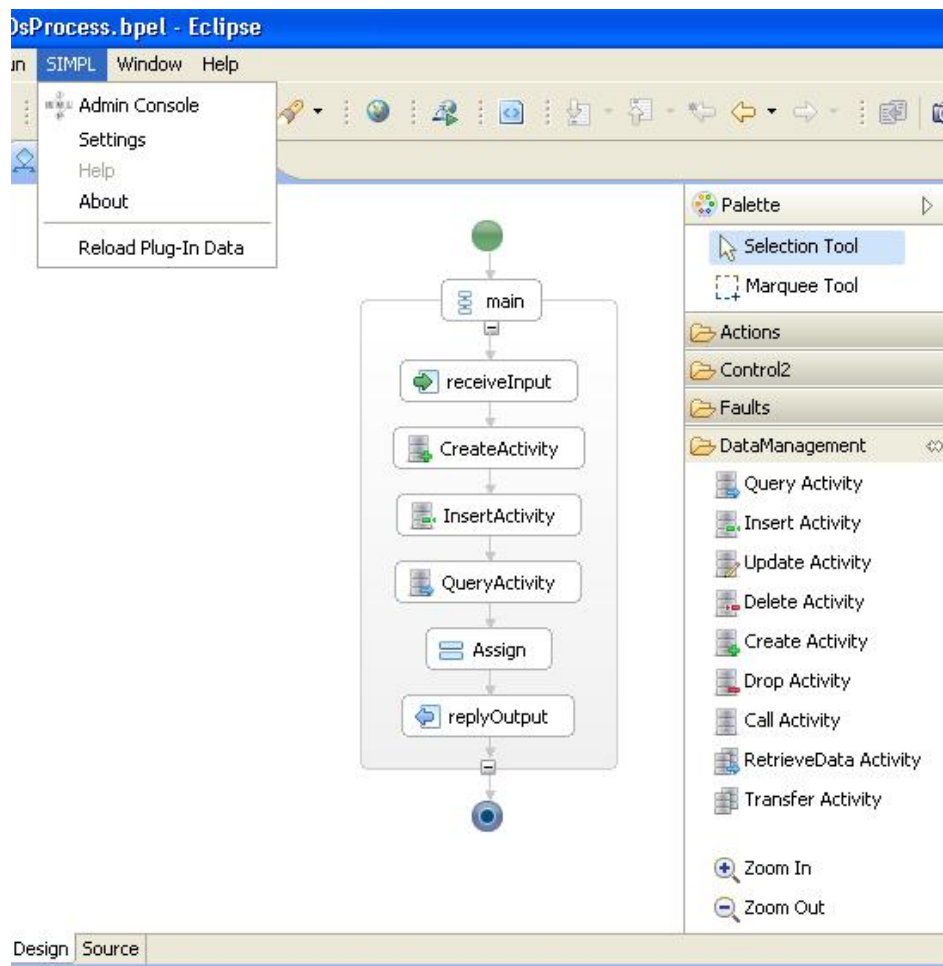


Abbildung 6: SIMPL Menü und Eclipse BPEL Designer mit einigen DM-Aktivitäten

4.2.1 Admin-Konsole

Die Admin-Konsole kann über das SIMPL Menü geöffnet werden und bietet die Möglichkeit, Einstellungen für den SIMPL Core vorzunehmen. Dazu gehört die Verwaltung des Auditings (siehe Abbildung 7).

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

Auditing

In Abbildung 7 wird der Unterpunkt “Auditing” der Admin-Konsole gezeigt. Hier kann das Auditing an- und abgeschaltet werden und eine Datenbank (analog wie im Deployment-Deskriptor) für das Auditing angegeben werden.

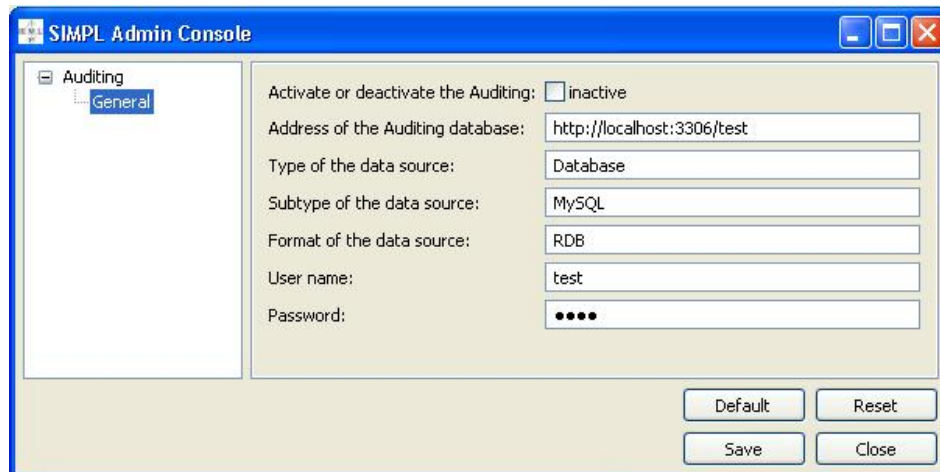


Abbildung 7: Dialog für die Einstellungen des Auditings

4.2.2 Einstellungen der SIMPL Eclipse Plug-Ins

Die Einstellungen können über das SIMPL Menü mit dem Menüpunkt “Settings” geöffnet werden und bieten die Möglichkeit, Einstellungen für die SIMPL Eclipse Plug-Ins vorzunehmen. Dazu gehört die Angabe der Adressen der SIMPL Core Web Services, der UDDI Registry, eines RRS und des Transformation Web Services (siehe Abbildung 8). Die Einstellungen werden als Eclipse Preferences integriert und stehen als solche auch über Eclipse zur Verfügung (Eclipse Menüleiste -> “Window” -> “Preferences”). Über das SIMPL Menü erreicht man direkt die SIMPL Preference Seite.

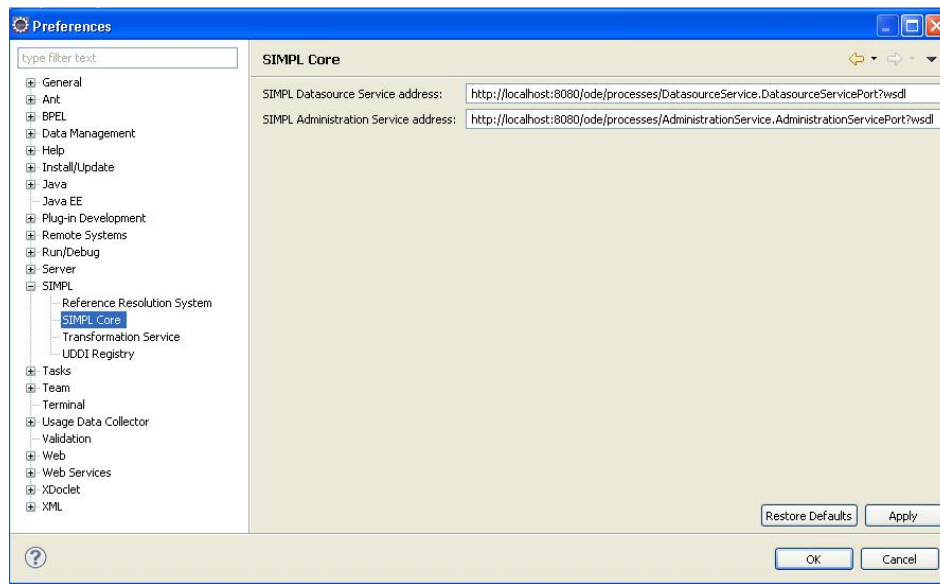


Abbildung 8: SIMPL Preference Seiten

4.2.3 Hilfe

Der Menüpunkt “Help” leitet den Benutzer auf die Eclipse Hilfe weiter. Dort stehen dem Benutzer über die Punkte “BPEL-DM Plug-In” und “SIMPL Core Plug-In” die entsprechenden Hilfe-Dokumente der SIMPL Eclipse Plug-Ins zur Verfügung. Eine genauere Beschreibung wird nachgeliefert, sobald die Umsetzung der Hilfe und deren Inhalt konkret feststeht.

4.2.4 About

Der Menüpunkt “About” öffnet das SIMPL About-Fenster, in dem Informationen über den Versionsstand, Lizenzbedingungen, die Projekt-Homepage und das Projektteam des SIMPL Rahmenwerks aufgeführt sind.

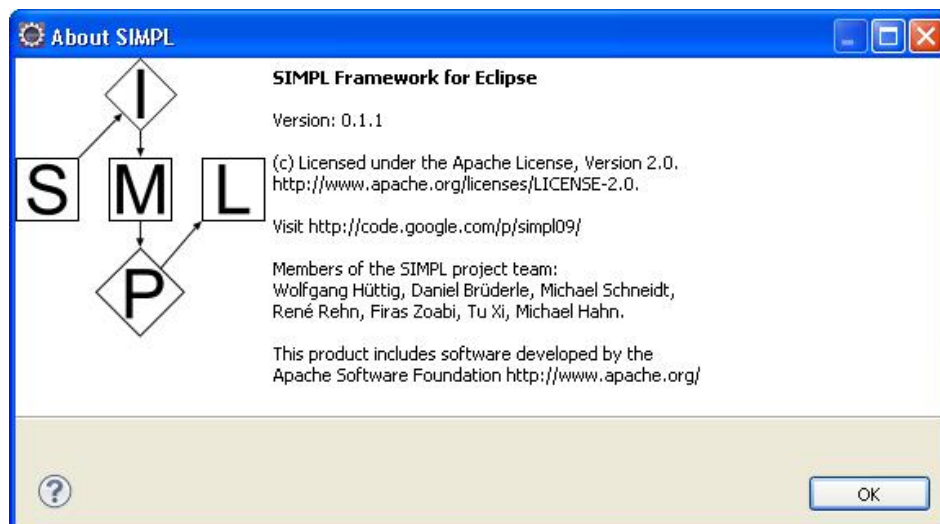


Abbildung 9: SIMPL About Fenster

4.2.5 Nachträgliches Laden der Plug-In Daten

Über den Menüpunkt “Reload Plug-In Data” in Abbildung 6 des SIMPL Menüs, können alle Daten des SIMPL Cores nachträglich abgerufen werden. Dies ist aus folgenden Gründen nötig:

- Der Deployment-Deskriptor des erweiterten BPEL Designers ruft eine Liste von unterstützten Datenquellentypen, deren Subtypen, den von diesen unterstützten Abfragesprachen und deren Datenformaten ab. Diese Informationen werden dann in den Dialogen (siehe Abschnitt 4.1.2) beim Anlegen oder Editieren von Datenquellen angezeigt und können dort ausgewählt werden.
- Das SIMPL Core Plug-In ruft die Einstellungen des SIMPL Cores ab, um diese in der Admin-Konsole anzeigen zu können.

Die Abfrage der entsprechenden Informationen geschieht bei der Initialisierung der Plug-Ins, d.h. beim Start von Eclipse. Ist zu diesem Zeitpunkt nun kein Apache Tomcat Server mit angebundenem SIMPL Core gestartet, können die Informationen nicht abgerufen werden und die beiden Plug-Ins sind nicht voll funktionsfähig. Um nun Eclipse nicht neustarten zu müssen, gibt es den Menüpunkt “Reload Plug-In Data”, mit dem die benötigten Informationen nachträglich vom SIMPL Core abgefragt werden können.

4.2.6 Data-Management-Aktivitäten

In Abbildung 10 wird die “PropertyView” am Beispiel einer Query-Aktivität (siehe Abschnitt 7.3.3) 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 eine UDDI- oder DD-Datenquelle über ihren logischen Namen ausgewählt und ein Befehl über entsprechende grafische Elemente oder auch einfach in der Text-Form erstellt. Durch die Auswahl einer Datenquelle werden alle Datenquelleninformationen automatisch in der Aktivität gesetzt.

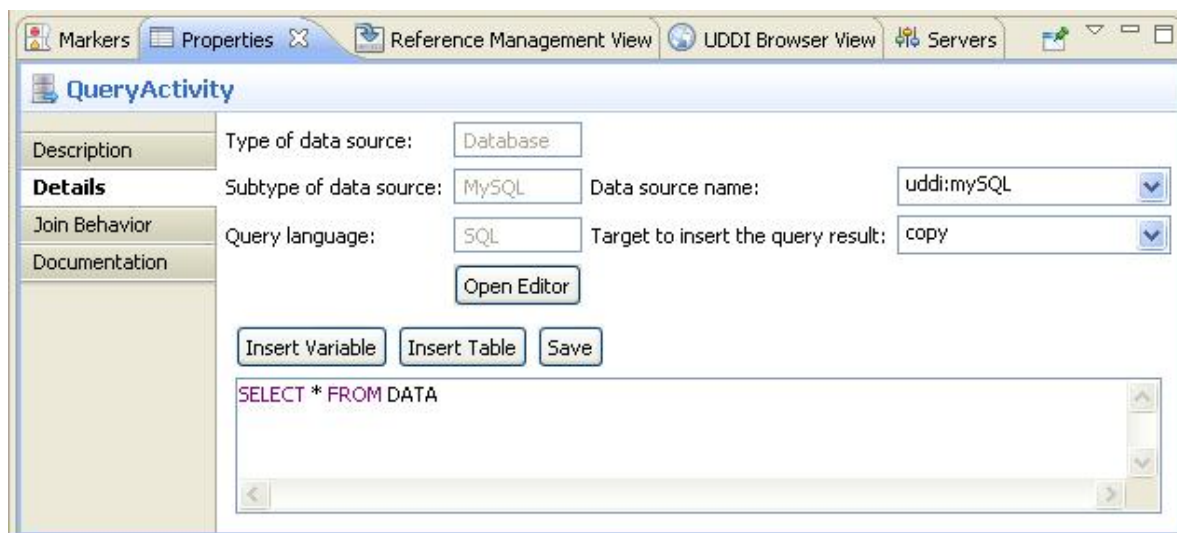


Abbildung 10: Eigenschaftsfenster einer DM-Aktivität am Beispiel einer Query Activity

4.3 Benutzeroberfläche des RRS Eclipse Plug-Ins

Das RRS Eclipse Plug-In wird als Eclipse View in Eclipse integriert. Über diese View kann das in den SIMPL Settings hinterlegte RRS verwaltet werden. Zur Verwaltung zählt das Anlegen, Bearbeiten

Name	RRS-Address	Adapter	Data source address	Statement	User name	Password
data	http://localhost:8080/axis...	RDB:DB2:SQL	localhost:50000/testdb	SELECT * FROM REFERENCE	test	****
asd	http://localhost:8080/axis...	RDB:MySQL:SQL	localhost:3306/test	SELECT * FROM test	test	****

Abbildung 11: Reference Resolution System View

Add a new reference
 Please enter the data of the new reference

Name *: myData

RRS-Address *: http://localhost:8080/axis2/services/RRSRetrievalService?wsdl

Adapter *: RDB:MySQL:SQL

DS address *: http://localhost:3306/test

Statement *: SELECT id, value FROM data

User: test

Password: test

Save

Abbildung 12: Dialog für das Hinzufügen einer Referenz

und Löschen von Referenzen eines RRSs. Abbildung 11 zeigt die View des RRS Eclipse Plug-Ins. Die verschiedenen Verwaltungsfunktionen können über das Kontextmenü oder über die Toolbar der View erreicht werden. Da ein RRS nicht nur von einem Modellierer verwendet werden kann oder auch Referenzen von Workflows oder Web Services erzeugt werden können, enthält die View einen Refresh-Button mithilfe dessen die Daten der View aktualisiert werden können.

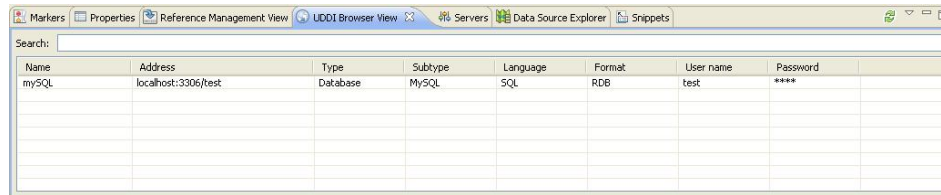
Abbildung 12 zeigt den Dialog für das Erstellen einer neuen Referenz. Dabei werden wieder alle Angaben mit einem logischen Namen verknüpft, der für die Bindung von Referenzvariablen und Referenzen benötigt wird (siehe Kapitel 4.1.1). Die RRS-Adresse wird automatisch aus den Preferences ausgelesen und ist nicht änderbar. Die Adresse des Adapters, der die Referenz auflöst, kann einfach ausgewählt werden. Der Mechanismus dafür entspricht dem des SIMPL Core, da die Architektur des RRS ähnlich strukturiert ist (Plug-In System). Die physikalische Datenquellenadresse muss zusammen mit den Authentifizierungsinformationen von Hand eingegeben werden. Für die Zukunft wäre hier die Anbindung einer Datenquellen-Registry möglich, sodass eine Datenquelle per Late-Binding oder statisch ausgewählt werden kann, ohne Daten von Hand angeben zu müssen. Der hinterlegte Abfragebefehl wird bei der Auflösung der Referenz durch den angegebenen Adapter ausgeführt und so die entsprechenden Daten zurückgeliefert.

4.4 Benutzeroberfläche der Datenquellen-Registry

In diesem Kapitel wird die Benutzeroberfläche des Eclipse Datenquellen-Registry Plug-Ins und des Datenquellen-Registry Web Interfaces beschrieben.

4.4.1 Datenquellen-Registry Plug-In

Das Datenquellen-Registry Plug-In wird als Eclipse View in Eclipse integriert. Die Datenquellen-Informationen werden aus der Datenquellen-Registry gelesen und in einer Tabelle in der View angezeigt (siehe Abbildung 13).



Name	Address	Type	Subtype	Language	Format	User name	Password
mySQL	localhost:3306/test	Database	MySQL	SQL	RDB	test	****

Abbildung 13: Datenquellen-Registry View

4.4.2 Datenquellen-Registry Web Interface

Das Web Interface soll dem Datenquellen-Administrator dabei helfen, die Datenquellen-Registry zu verwalten. Es gibt ihm die Möglichkeit, einfach über die grafische Oberfläche, neue Datenquellen anzulegen (siehe Abbildung 14). Dafür gibt er den Namen der Datenquelle und deren Adresse an und wählt jeweils aus einem Drop-Down-Menü den Typ und den Subtyp der Datenquelle aus. Durch das Drücken des “Add Policy”-Buttons, kann für die Datenquelle eine Policy-Datei, die extern modelliert und über das Textfeld ausgewählt wurde, hinzugefügt werden. Mit dem “Save”-Button werden alle durchgeführten Änderungen gespeichert.

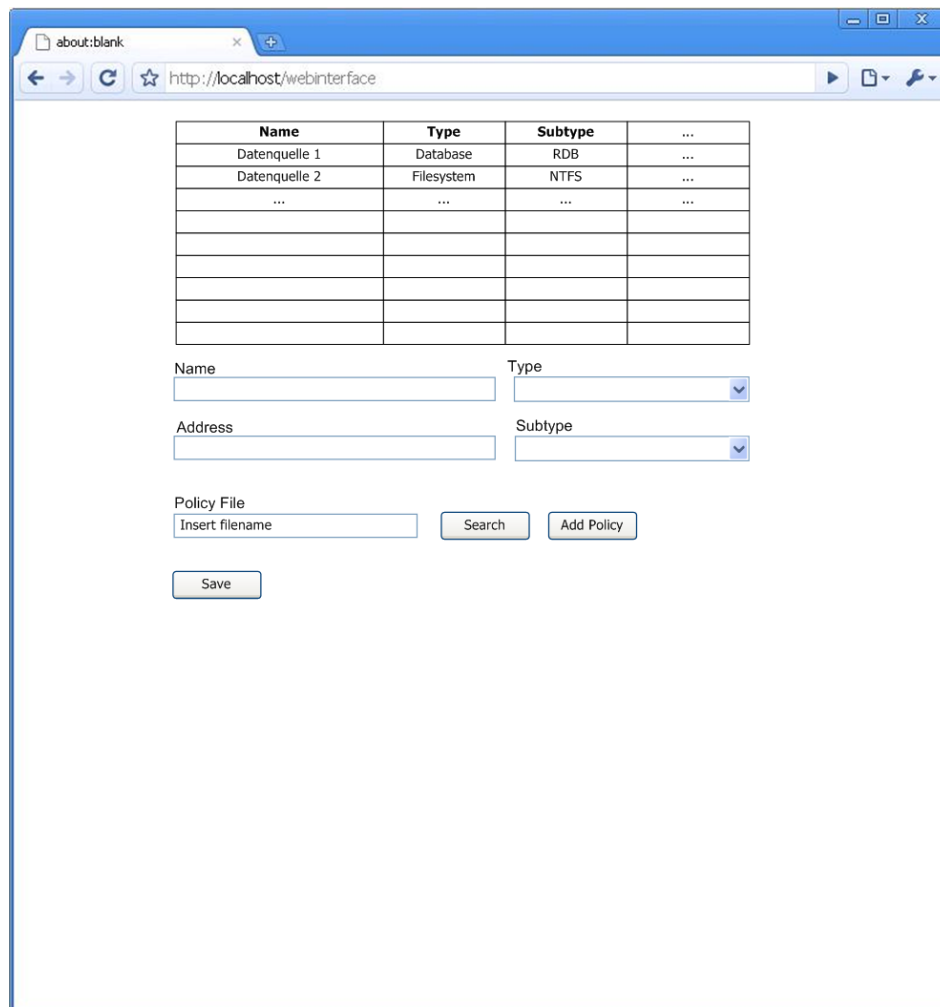


Abbildung 14: Datenquellen-Registry Webinterface

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. Weiterhin ist er für das Validieren von SIMPL Extensions zuständig.

5.5 Reference Resolution System

Das Reference Resolution System (RRS) ist ebenfalls ein durch Software realisierter Akteur. Es führt interne Anwendungsfälle aus, die von einem Benutzer durch andere Anwendungsfälle indirekt aufgerufen werden. Das RRS sorgt für das automatische Auflösen von Referenzen zur Laufzeit von Prozessinstanzen und realisiert die Verwaltung (Erstellen, Bearbeiten und Löschen) von Referenzen.

5.6 Datenquellen-Administrator

Ein Datenquellen-Administrator verwaltet Datenquellen und stellt diese über die Datenquellen-Registry anderen Nutzern zu Verfügung. Dazu kann er Datenquellen in der Datenquellen-Registry registrieren, die Eigenschaften bereits registrierter Datenquellen bearbeiten und registrierte Datenquellen auch wieder aus der Datenquellen-Registry löschen.

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 15 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 15: Anwendungsfall-Diagramm des gesamten Softwaresystems

6.2 Anwendungsfälle des Prozess-Modellierers

Ein Prozess-Modellierer kann folgende neue Anwendungsfälle (siehe Abbildung 16) ausführen:

- Data-Management-Aktivität erstellen

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

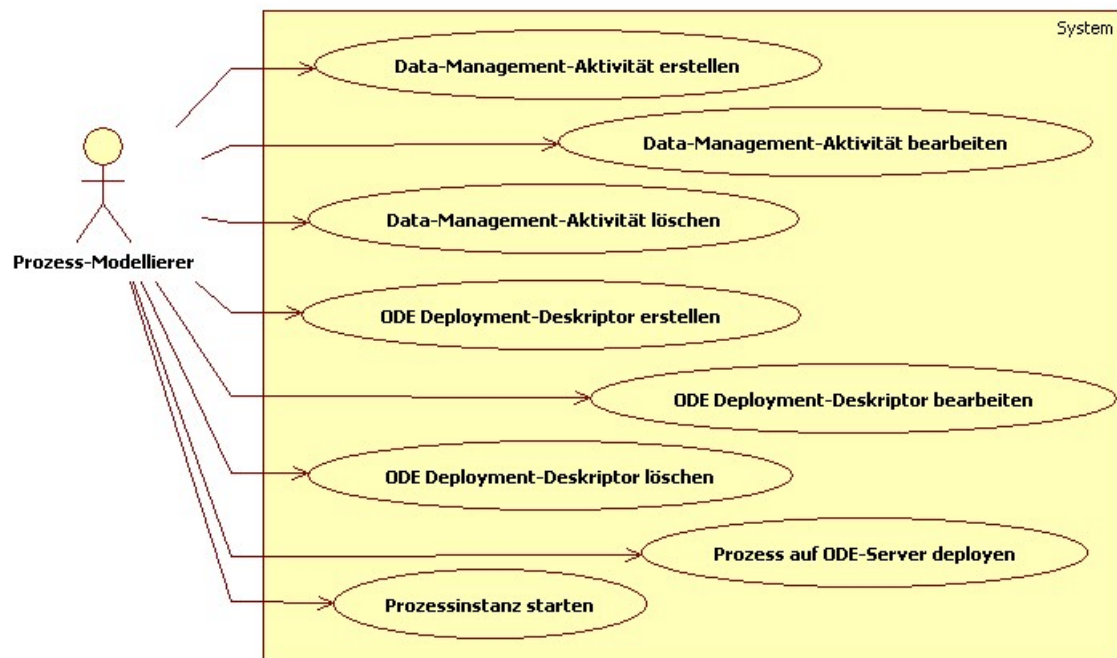


Abbildung 16: Anwendungsfall-Diagramm für den Prozess-Modellierer

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 ODE Deployment-Deskriptor erstellen

Ziel	Ein ODE Deployment-Deskriptor soll für einen Prozess erstellt werden.
Vorbedingung	Ein korrekter BPEL Prozess liegt vor.
Nachbedingung	Der ODE Deployment-Deskriptor wurde korrekt erstellt und der BPEL Prozess kann deployt werden.
Nachbedingung im Sonderfall	Der ODE Deployment-Deskriptor steht nicht zur Verfügung.
Normalablauf	1. Erstellung eines ODE Deployment-Deskriptors über File->New->Other->BPEL2.0->Apache ODE Deployment-Descriptor. 2. Konfiguration des Deployment-Deskriptors 3. Speicherung des Deployment-Deskriptors über File -> Save
Sonderfälle	Der ODE Deployment-Deskriptor kann nicht erstellt werden.

6.2.5 ODE Deployment-Deskriptor bearbeiten

Ziel	Konfiguration des ODE Deployment-Deskriptors soll geändert werden.
Vorbedingung	Ein ODE Deployment-Deskriptor wurde korrekt erstellt.
Nachbedingung	Die Änderungen sind gespeichert und der ODE Deployment-Deskriptor ist korrekt.
Nachbedingung im Sonderfall	Die Änderungen an der Konfiguration sind nicht gespeichert.
Normalablauf	1. ODE Deployment-Deskriptor über Doppelklick mit der linken Maustaste auf die Datei öffnen.
Sonderfälle	Der ODE Deployment-Deskriptor kann nicht gespeichert werden.

6.2.6 ODE Deployment-Deskriptor löschen

Ziel	Ein ODE Deployment-Deskriptor soll gelöscht werden.
Vorbedingung	Ein ODE Deployment -Deskriptor wurde korrekt erstellt.
Nachbedingung	Der ODE Deployment-Deskriptor ist gelöscht.
Nachbedingung im Sonderfall	Der ODE Deployment-Deskriptor ist nicht gelöscht.
Normalablauf	1. ODE Deployment-Deskriptor auswählen und über Edit->Delete löschen.
Sonderfälle	Der ODE Deployment-Deskriptor kann nicht gelöscht werden.

6.2.7 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, die Server-View wird angezeigt und ein ODE Deployment-Deskriptor wurde korrekt erstellt bzw. ist bereits vorhanden.
Nachbedingung	Die Prozess-Dateien wurden auf die Apache ODE Workflow-Engine kopiert und der Prozess wurde erfolgreich deployed.
Nachbedingung im Sonderfall	2a. Der ODE-Server ist nicht gestartet und der Prozess wurde nicht deployed. 2b. Der ODE-Server ist gestartet, aber der Prozess wurde nicht deployed.
Normalablauf	1. Hinzufügen des Prozesses zum ODE-Server in der Eclipse Server-View: <ul style="list-style-type: none">• Rechter Mausklick auf den ODE-Server• Auswahl des Menüpunkts "Add and Remove"• Hinzufügen der BPEL Prozess-Datei 2. Starten des ODE-Servers über rechten Mausklick und klicken auf "Start".
Sonderfälle	2a. ODE-Server startet nicht aufgrund eines Fehlers. 2b. Beim Deployen des Prozesses tritt ein Fehler auf.

6.2.8 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 "6.2.7 Prozess auf ODE-Server deployen").
Nachbedingung	Prozessinstanz wurde gestartet.
Nachbedingung im Sonderfall	Die Prozessinstanz wird verworfen.
Normalablauf	1. Instanziierung 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.

6.2.9 Strategie für das Late-Binding auswählen

Ziel	Auswahl einer Strategie für das Late-Binding einer Datenquelle in einer DM-Aktivität.
Vorbedingung	Ein BPEL-Prozess mit ODE Deployment-Deskriptor und ein WS-Policy-Dokument mit Anforderungen an die Datenquelle müssen erstellt sein. Der ODE Deployment-Deskriptor muss geöffnet sein.
Nachbedingung	Die Strategie für eine DM-Aktivität ist im ODE Deployment-Deskriptor hinterlegt.
Nachbedingung im Sonderfall	
Normalablauf	<ol style="list-style-type: none">1. Auswahl einer DM-Aktivität aus der Liste mit DM-Aktivitäten.2. Auswahl einer Strategie über das Strategie-Dropdown-Menü.3. Auswahl einer WS-Policy-Datei über den Button "WS-Policy Datei auswählen".4. Speichern des ODE Deployment-Deskriptor über das "Speichern"-Symbol in der Symbolleiste.
Sonderfälle	

6.3 Anwendungsfälle des Workflow-Administrators

Ein Workflow-Administrator kann folgende neue Anwendungsfälle (siehe Abbildung 17) 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
- Neue Referenz in RRS einfügen
- Referenz aus RRS bearbeiten
- Referenz aus RRS löschen

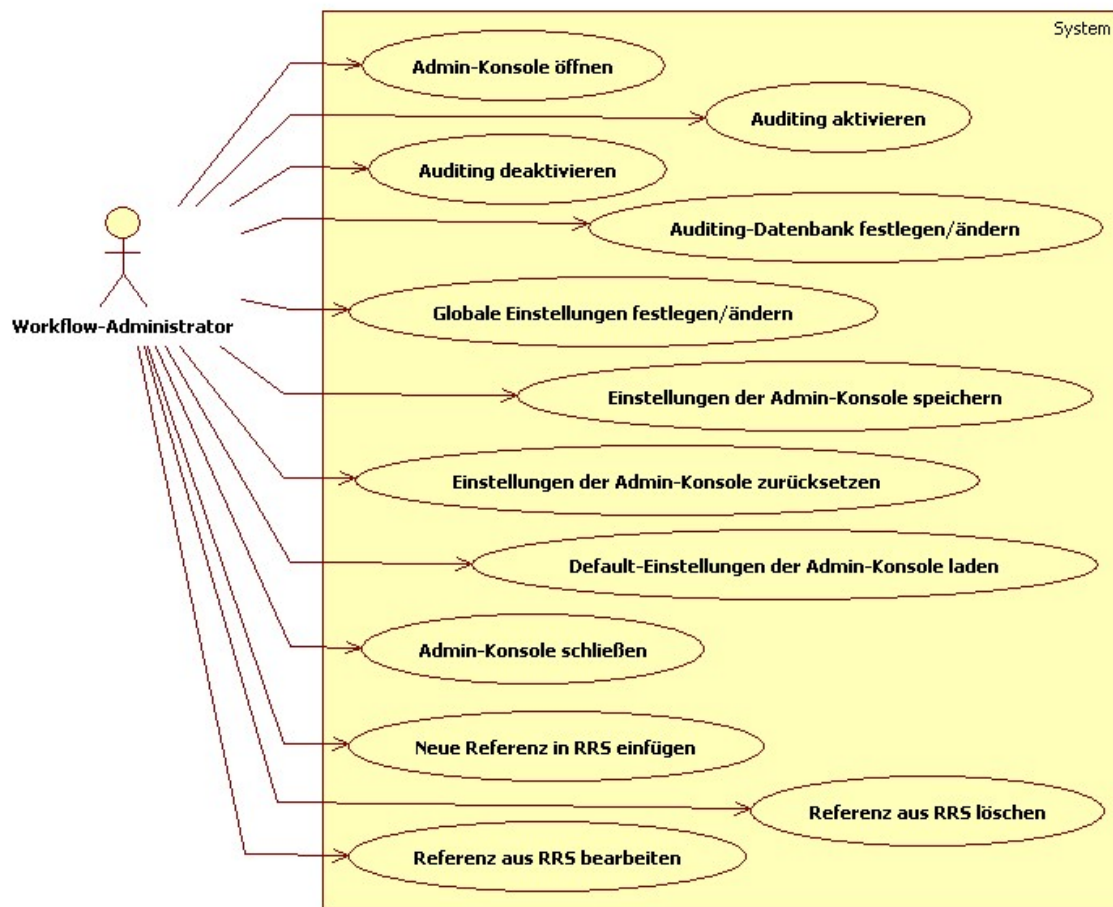


Abbildung 17: Anwendungsfall-Diagramm für den Workflow-Administrator

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	<ol style="list-style-type: none"> 1. Klick auf das SIMPL Menü in der Menüleiste. 2. Klick auf den Menüeintrag [Admin Console]. 3. Anstoßen des Anwendungsfalls "6.5.1 SIMPL 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 SIMPL 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 SIMPL 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 SIMPL 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 SIMPL 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.

6.3.10 Neue Referenz in RRS einfügen

Ziel	Eine neue Referenz soll ins RRS eingefügt werden.
Vorbedingung	Der RRS-View ist geöffnet und das RRS ist erreichbar.
Nachbedingung	Die neue Referenz wurde korrekt im RRS gespeichert und wird im RRS-View angezeigt.
Nachbedingung im Sonderfall	Es wird eine entsprechende Fehlermeldung angezeigt, die den Benutzer über die aufgetretenen Fehler informiert.
Normalablauf	1. Klick des [New] Buttons der Toolbar oder Auswahl des entsprechenden Menü-Punktes des Kontext-Menüs 2. Angabe aller Parameter der Referenz in dem entsprechenden Pop-up Fenster 3. Klick auf [Save], um die Werte der neue Referenz zu speichern und der Anwendungsfall “6.6.2 Referenz speichern” wird angestoßen
Sonderfälle	3a. Beim Speichern der neue Referenz tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist.

6.3.11 Referenz aus RRS bearbeiten

Ziel	Bearbeiten einer vorhandenen Referenz des RRS.
Vorbedingung	Der RRS-View ist geöffnet, das RRS ist erreichbar und die zu bearbeitende Referenz ist ausgewählt.
Nachbedingung	Die geänderten Werte der Referenz wurden korrekt im RRS gespeichert, die Referenz wurde aktualisiert und die aktualisierten Werte werden in der Referenzen-Tabelle angezeigt.
Nachbedingung im Sonderfall	Die bearbeitete Referenz bleibt im RRS und in der Referenzen-Tabelle unverändert und es wird eine entsprechende Fehlermeldung angezeigt, die den Benutzer über die aufgetretenen Fehler informiert.
Normalablauf	1. Klick des [Edit] Buttons der Toolbar oder Auswahl des entsprechenden Menü-Punktes des Kontext-Menüs 2. Angabe der neuen Parameter der Referenz in dem entsprechenden Pop-up Fenster 3. Klick auf [Save], um die aktualisierten Werte der Referenz zu speichern und der Anwendungsfall "6.6.2 Referenz speichern" wird angestoßen
Sonderfälle	3a. Beim Speichern der Änderungen tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist.

6.3.12 Referenz aus RRS löschen

Ziel	Löschen einer vorhandenen Referenz aus dem RRS.
Vorbedingung	Der RRS-View ist geöffnet, das RRS ist erreichbar und die zu löschende Referenz wurde ausgewählt.
Nachbedingung	Die entsprechende Referenz wurde vollständig und korrekt aus dem RRS und der Referenzen-Tabelle entfernt und kann nun nicht mehr verwendet werden.
Nachbedingung im Sonderfall	Die zu löschende Referenz bleibt im RRS und in der Referenzen-Tabelle unverändert und es wird eine entsprechende Fehlermeldung angezeigt, die den Benutzer über die aufgetretenen Fehler informiert.
Normalablauf	1. Klick des [Delete] Buttons der Toolbar oder Auswahl des entsprechenden Menü-Punktes des Kontext-Menüs und der Anwendungsfall "6.6.4 Referenz löschen" wird angestoßen.
Sonderfälle	1a. Beim Löschen der Referenz tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist.

6.4 Anwendungsfälle der ODE Workflow-Engine

Die ODE Workflow-Engine kann durch unsere Erweiterungen folgende Anwendungsfälle (siehe Abbildung 18) ausführen:

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

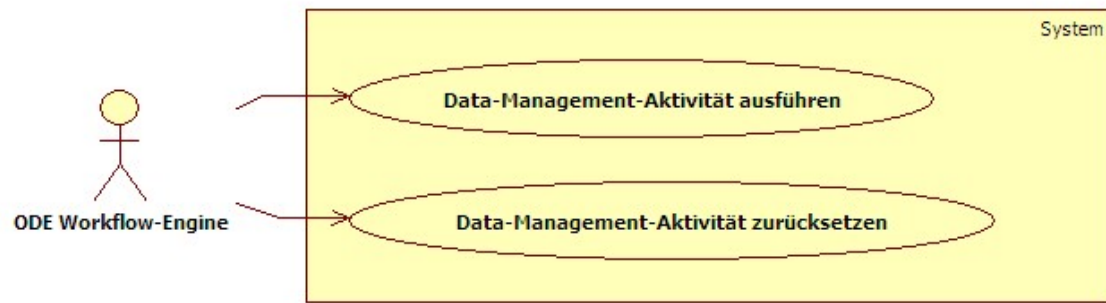


Abbildung 18: Anwendungsfall-Diagramm für die ODE Workflow-Engine

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. Die Verbindung zur Datenquelle auf der die DM-Operationen durchgeführt werden wird aufgebaut. 4. Senden der in der DM-Aktivität enthaltenen DM-Operationen zur Ausführung an die Datenquelle. 5. 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 "Complete". 6. Die Verbindung zur Datenquelle wird nach erfolgreicher Ausführung der DM-Operation geschlossen. 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" von der Vateraktivität an die entsprechende DM-Aktivität gesendet und die Aktivität wird beendet. 1a.3. Der Use Case "6.4.2 Data-Management-Aktivität zurücksetzen" wird angestoßen. 2a.1. In der Vateraktivität tritt ein Fehler auf. 2a.2. Es wird ein "Terminate_Activity Event" von der Vateraktivität an die entsprechende DM-Aktivität gesendet und die Aktivität wird beendet. 2a.3. Der Use Case "6.4.2 Data-Management-Aktivität zurücksetzen" wird angestoßen. 3. Die Verbindung zur Datenquelle konnte nicht aufgebaut werden. Es wird ein "ConnectionLost Event" ausgelöst 4a.1. Während der Durchführung der DM-Operationen tritt ein Fehler in der Datenquelle auf (z.B. ein Syntaxfehler, oder die Datenbank ist nicht erreichbar). 4a.2. Es wird ein "DMFailure" Event ausgelöst. 4a.3. Der Use Case "6.4.2 Data-Management-Aktivität zurücksetzen" wird angestoßen. 4b.1 Während der Durchführung der DM-Operation tritt ein Verbindungsabbruch zur Datenquelle auf. Es wird ein "ConnectionLost Event" ausgelöst. Zusätzlich wird ein "DMFailure Event" ausgelöst. 5a.1. In der Vateraktivität tritt ein Fehler auf. 5a.2. Es wird ein "Terminate_Activity Event" von der Vateraktivität an die entsprechende DM-Aktivität gesendet und die Aktivität wird beendet. 5a.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 19) ausführen:

- SIMPL Admin-Konsole laden
- SIMPL Admin-Konsole Defaults laden
- SIMPL Admin-Konsole speichern
- Datenquellen aus UDDI-Registry abrufen
- Datenquelle per Strategie auswählen
- BPEL-Datei transformieren
- SIMPL Extensions validieren

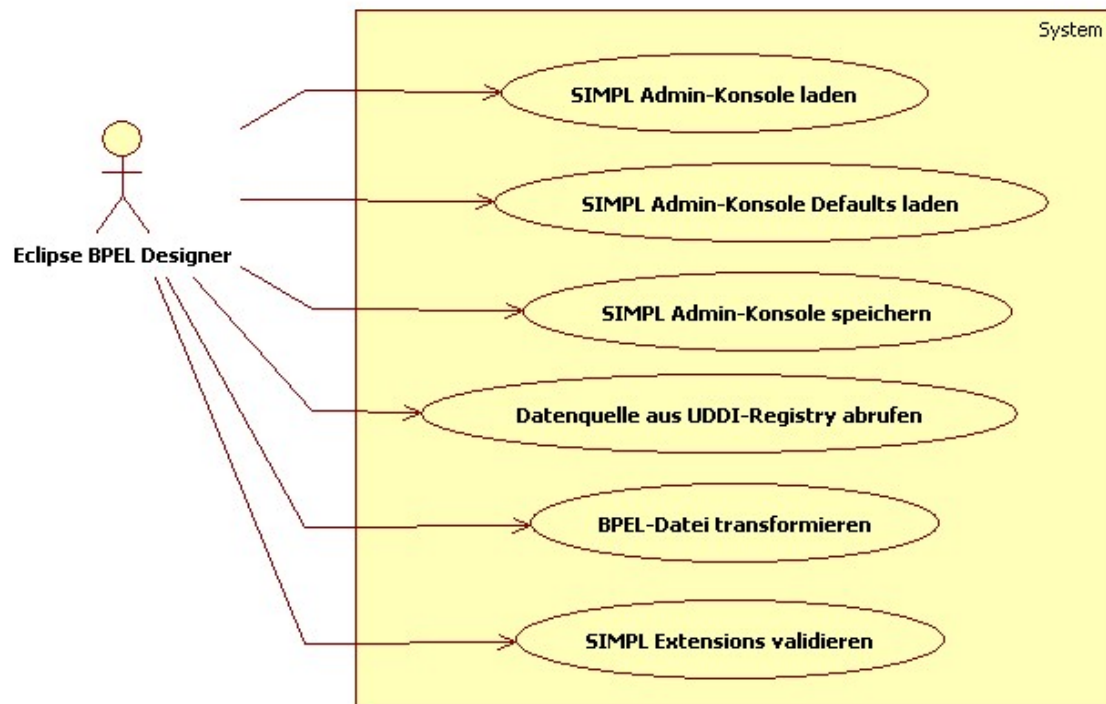


Abbildung 19: Anwendungsfall-Diagramm für den Eclipse BPEL Designer

6.5.1 SIMPL Admin-Konsole laden

Ziel	Laden der Inhalte der Admin-Konsole.
Vorbedingung	Einer der Anwendungsfälle “6.3.1 Admin-Konsole öffnen” oder “6.3.7 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 SIMPL 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. Falls zuvor keine Werte angezeigt wurden, bleiben die Felder der Admin-Konsole leer.
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 SIMPL Admin-Konsole Defaults laden

Ziel	Laden der Default-Werte der Admin-Konsole.
Vorbedingung	Der Anwendungsfall “6.3.8 Default-Einstellungen der Admin-Konsole laden” oder der Sonderfall 3a des Anwendungsfalls “6.5.1 SIMPL 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 SIMPL Admin-Konsole speichern

Ziel	Persistente Speicherung der Einstellungen der Admin-Konsole.
Vorbedingung	Der Anwendungsfall “6.3.6 Einstellungen der Admin-Konsole speichern” oder der Sonderfall <i>3b1</i> des Anwendungsfalls “6.3.9 Admin-Konsole schließen” wurde ausgeführt.
Nachbedingung	Alle Werte der Admin-Konsole wurden korrekt auf der SIMPL DB gespeichert.
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.

6.5.4 SIMPL Extensions validieren

Ziel	Validierung der SIMPL Erweiterung
Vorbedingung	1. Es wurde ein BPEL Prozess erstellt der mind. eine SIMPL Extension enthält.
Nachbedingung	Alle nicht schema-konformen Einträge werden erkannt und in der Problem-View angezeigt.
Nachbedingung im Sonderfall	
Normalablauf	1. “Validate” wird ausgeführt. 2. Fehler und Warnungen werden, falls vorhanden, in der Problem-View angezeigt.
Sonderfälle	

6.5.5 Datenquellen aus UDDI-Registry abrufen

Ziel	Abrufen aller verfügbaren Datenquellen.
Vorbedingung	Die Datenquellen-Registry ist erreichbar.
Nachbedingung	Eine Liste mit allen verfügbaren Datenquellen steht zur Verfügung.
Nachbedingung im Sonderfall	
Normalablauf	1. Mit Datenquellen-Registry verbinden 2. Verfügbare Datenquellen abfragen
Sonderfälle	

6.5.6 BPEL-Datei transformieren

Ziel	Umwandlung einer BPEL-Datei in Standard-BPEL-Quellcode.
Vorbedingung	Eine BPEL-Datei (BPEL-Prozess) muss vorhanden sein und das Reference Resolution System (RRS) muss angebunden und erreichbar sein.
Nachbedingung	Es wurde ein korrekter standard-konformer BPEL-Prozess, der die gleiche Funktionalität wie der originale Prozess besitzt und zur Verwendung von Referenzen um entsprechende Aktivitäten oder Events erweitert wurde, erzeugt.
Nachbedingung im Sonderfall	Es wurde kein neuer BPEL-Prozess erzeugt, der originale BPEL-Prozess ist unverändert und es wird eine entsprechende Fehlermeldung ausgegeben.
Normalablauf	<ol style="list-style-type: none"> 1. Auslesen des originalen Prozesses 2. Positioniertes Einfügen entsprechender Dereferenzierungsaktivitäten oder Events für Referenzen (siehe 7.1.3) 3. Generierung des neuen BPEL-standard-konformen erweiterten Prozesses
Sonderfälle	3a. Bei der Generierung tritt ein Fehler auf

6.6 Anwendungsfälle des RRS

Das RRS kann folgende Anwendungsfälle (siehe Abbildung 20) ausführen:

- Referenz auflösen/dereferenzieren
- Referenz speichern
- Referenz laden
- Referenz löschen

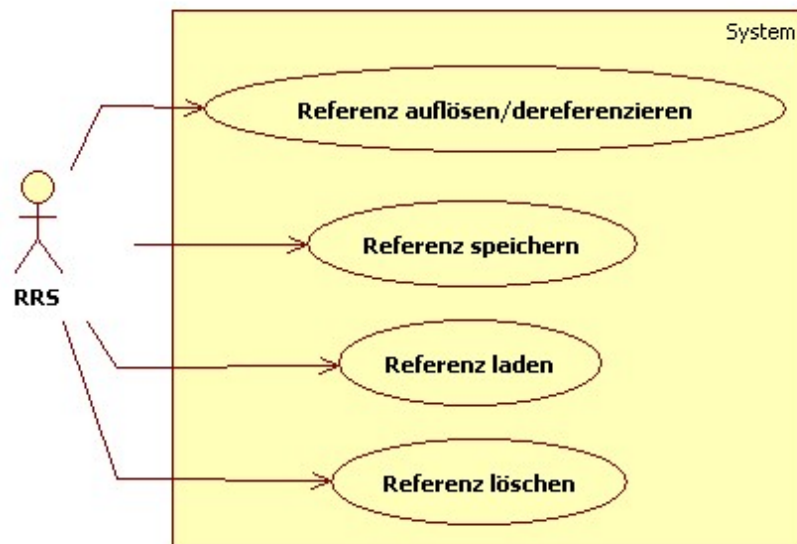


Abbildung 20: Anwendungsfall-Diagramm für RRS

6.6.1 Referenz auflösen/dereferenzieren

Ziel	Der zugrundeliegende Wert einer Referenz wird ausgelesen.
Vorbedingung	Die angegebene Referenz muss existieren und das RRS muss erreichbar sein. Ebenso muss die Datenquelle des referenzierten Werts erreichbar sein.
Nachbedingung	Der korrekte Wert einer Referenz wird geliefert.
Nachbedingung im Sonderfall	Es wird ein entsprechender Fehler geworfen, der Informationen über das zugrundeliegende Problem liefert.
Normalablauf	1. Wert der Referenz wird im RRS abgefragt
Sonderfälle	1a. Beim Abfragen des Werts tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist

6.6.2 Referenz speichern

Ziel	Die Referenz soll im RRS gespeichert werden.
Vorbedingung	Einer der beiden Anwendungsfälle “?? Neue Referenz in RRS einfügen” oder “?? Referenz aus RRS bearbeiten” wurde erfolgreich ausgeführt.
Nachbedingung	Die Referenz wurde korrekt gespeichert.
Nachbedingung im Sonderfall	Es wird ein Fehler geworfen, der Informationen über das zugrundeliegende Problem liefert.
Normalablauf	1. Die Referenz wird im RRS gespeichert
Sonderfälle	1a. Beim Speichern der Referenz tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist

6.6.3 Referenz laden

Ziel	Die Referenz soll aus dem RRS geladen werden.
Vorbedingung	Der RRS View wurde im Eclipse geöffnet.
Nachbedingung	Die Referenz wurde erfolgreich geladen und wird nun im RRS View angezeigt.
Nachbedingung im Sonderfall	Es wird ein Fehler geworfen, der Informationen über das zugrundeliegende Problem liefert.
Normalablauf	1. Die Referenz wird aus dem RRS geladen
Sonderfälle	1a. Beim Laden der Referenz tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist

6.6.4 Referenz löschen

Ziel	Die Referenz soll aus dem RRS entfernt werden.
Vorbedingung	Der Anwendungsfall “6.3.12 Referenz aus RRS löschen” wurde erfolgreich ausgeführt
Nachbedingung	Die Referenz wurde erfolgreich aus dem RRS entfernt.
Nachbedingung im Sonderfall	Es wird ein Fehler geworfen, der Informationen über das zugrundeliegende Problem liefert.
Normalablauf	1. Die Referenz wird aus dem RRS entfernt
Sonderfälle	1a. Beim Entfernen der Referenz tritt ein Fehler auf, da z.B. das RRS nicht erreichbar ist

6.7 Anwendungsfälle der Datenquellen-Administratoren

Ein Datenquellen-Administrator kann folgende Anwendungsfälle (siehe Abbildung 21) ausführen:

- Datenquelle in UDDI-Registry registrieren
- Datenquelle aus UDDI-Registry entfernen
- Datenquelle aus UDDI-Registry bearbeiten

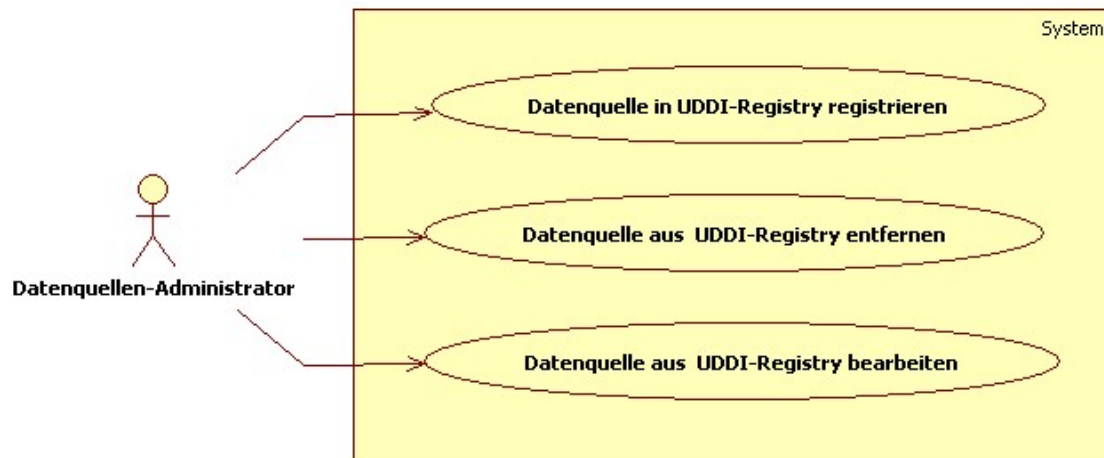


Abbildung 21: Anwendungsfall-Diagramm für die Datenquellen-Administratoren

6.7.1 Datenquelle in UDDI-Registry registrieren

Ziel	Registrierung einer Datenquelle in der Datenquellen-Registry.
Vorbedingung	Die Datenquellen-Registry ist erreichbar und der Benutzer hat die erforderlichen Rechte.
Nachbedingung	Eine neue Datenquelle wurde anhand der Benutzerangaben in der Datenquellen-Registry korrekt registriert und kann nun in Workflows verwendet werden.
Nachbedingung im Sonderfall	
Normalablauf	1. Anlegen eines neuen Datenquellenobjekts 2. Registrierung des Datenquellenobjekts in der Datenquellen-Registry
Sonderfälle	

6.7.2 Datenquelle aus UDDI-Registry entfernen

Ziel	Entfernen einer vorhandenen Datenquelle aus der Datenquellen-Registry
Vorbedingung	Die Datenquellen-Registry ist erreichbar und der Benutzer hat die erforderlichen Rechte.
Nachbedingung	Die vom Benutzer ausgewählte Datenquelle wurde korrekt und vollständig aus der Datenquellen-Registry entfernt und kann nun nicht mehr verwendet werden.
Nachbedingung im Sonderfall	
Normalablauf	1. Auswahl der Datenquelle 2. Entfernen der Datenquelle aus der Datenquellen-Registry
Sonderfälle	

6.7.3 Datenquelle aus UDDI-Registry bearbeiten

Ziel	Entfernen einer vorhandenen Datenquelle aus der Datenquellen-Registry
Vorbedingung	Die Datenquellen-Registry ist erreichbar und der Benutzer hat die erforderlichen Rechte.
Nachbedingung	Die vom Benutzer ausgewählte Datenquelle wurde korrekt und vollständig anhand der vom Benutzer durchgeführten Änderungen (Änderung der Eigenschaften, der Adresse, usw.) aktualisiert.
Nachbedingung im Sonderfall	
Normalablauf	1. Auswahl der Datenquelle 2. Bearbeiten der Werte der Datenquelle
Sonderfälle	

6.8 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	RRS / Eclipse RRS Plug-In	UDDI Webinterface / UDDI Eclipse Plug-In
Data-Management-Aktivität erstellen			x				
Data-Management-Aktivität bearbeiten			x				
Data-Management-Aktivität löschen			x				
ODE Deployment Deskriptor erstellen					x		
ODE Deployment Deskriptor bearbeiten					x		
ODE Deployment Deskriptor löschen					x		
Prozess auf ODE-Server deployen					x		
Prozessinstanz starten					x		
Strategie für das Late-Binding auswählen					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					
Referenz auflösen / dereferenzieren							
Referenz speichern						x / -	
Referenz laden						x / -	
Referenz löschen						x / -	

Anwendungsfall	SIMPL Core	SIMPL Core Plug-In	BPEL-DM Plug-In	SIMPL Extension-Activity Plug-In	Eclipse BPEL Designer Plug-In	RRS / Eclipse RRS Plug-In	UDDI Webinterface / UDDI Eclipse Plug-In
Neue Referenz in RRS einfügen						- / x	
Referenz aus RRS löschen						- / x	
Referenz aus RRS bearbeiten						- / x	
Data-Management-Aktivität ausführen				x			
Data-Management-Aktivität zurücksetzen				x			
SIMPL Admin-Konsole laden	x	x					
SIMPL Admin-Konsole Defaults laden	x	x					
SIMPL Admin-Konsole speichern	x	x					
Datenquelle aus UDDI-Registry abrufen					x		- / x
BPEL-Datei transformieren					x		
SIMPL Extensions validieren					x		
Datenquelle in UDDI-Registry registrieren							x / -
Datenquelle aus UDDI-Registry entfernen							x / -
Datenquelle aus UDDI-Registry bearbeiten							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 eines Referenzen-Konzepts für BPEL, das es ermöglicht, mit Referenzen auf Daten innerhalb von Workflows zu arbeiten oder die Identifizierung und Definition von 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-Auditing und das dafür zugrunde liegende Event-Modell beschrieben.

7.1 Daten-Referenzen in BPEL (IAAS-Referenzen)

Da bereits ein Konzept und umfassende Erkenntnisse zur Bereitstellung und Verwaltung von Referenzen in BPEL vorliegt [3], werden an dieser Stelle nur die Ergebnisse des Dokuments zusammengefasst und an die Anforderungen dieses Projekts angepasst.

In BPEL werden Daten immer in Variablen gespeichert und die Werte (*by value*) innerhalb des Workflows weitergegeben. Dies hat zur Folge, dass gerade bei großen Datenmengen, wie sie in wissenschaftlichen Workflows normal sind, die Performanz der Workflow Engine stark durch das Transportieren der Daten beeinflusst wird. Um dies zu verhindern, sollen Referenzen auf Daten eingeführt werden. Diese werden dann zwischen den einzelnen Web Services weitergegeben (Daten werden *by reference* übergeben) und die Daten bleiben, sofern sie nicht im Workflow benötigt werden, auf ihrer Datenquelle und werden dort auch bearbeitet.

Zur Umsetzung dieses Konzepts muss eine neue Art von BPEL Variablen eingeführt werden, sogenannte Referenzvariablen, die dazu genutzt werden können, auf Daten zu verweisen. Dadurch müssen nur noch Referenzen zwischen den Web Services und Workflows weitergeleitet werden, und nicht mehr die Daten selbst. Für wissenschaftliche Workflows reduziert sich dadurch der Transport von großen Datenmengen zwischen den Web Services und dem Workflow erheblich.

Zur Bereitstellung solcher Referenzen für Web Services und Workflows müssen diese extern verwaltet werden und auch extern abrufbar sein. So kann ein globales oder unternehmensweites Variablenkonzept erstellt werden, das es ermöglicht, dieselben Daten in mehreren Workflows zu nutzen und nur einmal zentral zur Verfügung zu stellen.

Die Umsetzung der Referenzen wird durch sogenannte Endpoint References realisiert, die in Abschnitt 7.1.2 beschrieben werden. Das externe Verwaltungssystem der Referenzen, das sogenannte Reference Resolution System, wird in Abschnitt 7.1.1 näher erläutert. Die Einbindung der Referenzen in BPEL (Modellierungswerkzeug & Workflow Engine) beschreibt Abschnitt 7.1.3.

7.1.1 Reference Resolution System

Abbildung 22 zeigt die Architektur des Reference Resolution Systems (RRS). Am unteren Ende der Abbildung werden die verschiedenen möglichen Nutzer des Systems gezeigt: Workflows und Web Services, die den Wert einer Referenz auslesen, Referenzen auslesen oder Referenzen im System anlegen, aktualisieren und löschen können.

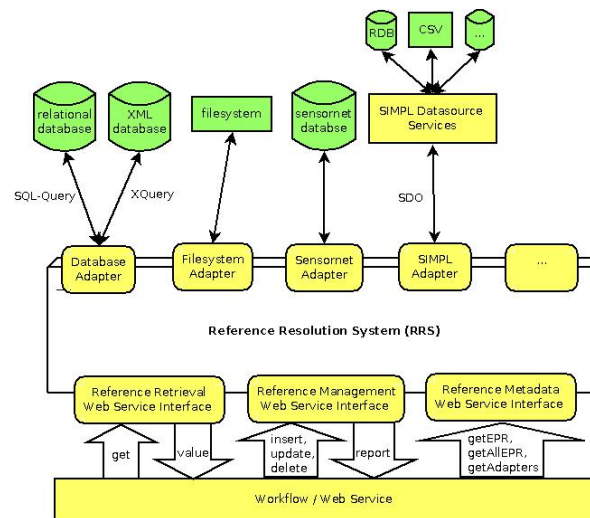


Abbildung 22: Die Architektur des Reference Resolution Systems (RRS).

Auch wenn auf eine Trennung von Nutzern nach Zugriffsrechten (Benutzer vs. Administrator) aus Gründen der Flexibilität (ermöglicht automatische Erstellung von neuen Referenzen zur Prozesslaufzeit) verzichtet wird, so werden doch zwei entsprechende Web Service Schnittstellen bereitgestellt. Dadurch ist eine spätere Erweiterung um Zugriffsrechte leicht realisierbar. Im Folgenden werden alle drei Schnittstellen des RRS kurz beschrieben:

- Die erste Schnittstelle ist der *Reference Retrieval Web Service* mit der Methode *get* und einer *Endpoint reference* (EPR) als Eingabe. Das Konzept der Endpoint References wird im Web Services Addressing Standard (siehe [25]) beschrieben und kann ohne Anpassungen für die Repräsentation einer Referenz genutzt werden. Der Rückgabewert der Methode ist der Wert, der durch die angegebene EPR referenziert wurde.
- Die zweite Schnittstelle ist der *Reference Management Web Service*, der drei Methoden bereitstellt. Die Methode *insert*, um eine neue Referenz zu erstellen. Dabei ist die Eingabe der Wert der Referenz, wo dieser Wert gespeichert wird und wie er dort wieder ausgelesen werden kann. Dies könnte beispielsweise durch entsprechende SQL-Befehle angegeben werden. Der Rückgabewert ist eine Meldung, die die neu generierte EPR enthält. Als Zweites die *update* Methode, die für die Aktualisierung von gespeicherten Werten einer Referenz benötigt wird. Dabei ist die Eingabe der neue Wert der Referenz und wie dieser Wert gespeichert wird. Der Rückgabewert ist eine Meldung, ob die Aktualisierung erfolgreich war. Die *delete* Methode löscht eine Referenz aus dem RRS, d.h. der Wert und alle sonstigen Informationen der Referenz werden aus dem System entfernt. Die Eingabe ist dabei die EPR, die entfernt werden soll und der Rückgabewert ist eine Meldung, ob die Referenz erfolgreich entfernt wurde.
- Die dritte Schnittstelle ist der *Reference Metadata Web Service*, der drei Methoden bereitstellt. Die Methode *getEPR*, um eine einzelne Referenz auslesen zu können. Dies wird bei der Transformation eines Prozessmodells benötigt (siehe Abschnitt 7.1.3). Die Methode *getAllEPR*, um alle Referenzen eines RRS auslesen zu können und diese beispielsweise in der RRS Management View in Eclipse anzeigen zu können. Die Methode *getAllAdapters*, um alle verwendbaren Adapter eines RRS abzufragen, sodass diese beispielsweise beim Anlegen neuer Referenzen angezeigt werden können.

Die Hauptkomponente ist allerdings das Reference Resolution System selbst. Es verbindet die drei Web Service Schnittstellen und bietet die Möglichkeit, eine Menge von Adaptern für die Integration verschiedenster Datenquellen anzubinden. Jeder dieser Adapter besteht dabei aus einem Lookup-Service

für gespeicherte Queries und Informationen sowie aus einem ausführenden Service, mit dem Queries auf den Datenquellen ausgeführt werden können. Anhand einer gegebenen Referenz wird dazu der passende Adapter gesucht, ausgewählt und für das Auflösen von Referenzen verwendet. Im Rahmen des Projektes wird nur ein SIMPL Adapter angeboten, der die Verbindung zu allen Datenquellen ermöglicht, die vom SIMPL Core unterstützt werden. Abbildung 22 zeigt am oberen Ende einige solcher Adapter, wobei das RRS erweiterbar ist und mit Adaptern für alle Arten von Datenquellen ergänzt werden kann.

In [3] werden mehrere mögliche Anbindungen des RRS in der Workflow-Umgebung genannt: “Ein RRS pro Workflow-Umgebung”, “Ein RRS pro Web Service” und “Ein RRS pro Datenquelle”. Im Zusammenhang mit unseren Gegebenheiten, vor allem im Hinblick darauf, dass die Workflow-Engine lokal auf dem Benutzer-Rechner ausgeführt wird, bietet sich die Variante “Ein RRS pro Workflow-Engine” an. Das RRS muss dabei nicht zwangsweise auf dem Rechner liegen auf dem auch die Workflow-Engine ausgeführt wird. Allerdings benötigt man dann eine Authentifizierung um die nachfolgenden Vorteile zu erhalten, die im Moment aus der identischen Lokalität des RRS und der Workflow-Engine resultieren:

- Jeder Prozess-Modellierer (Workflow-Administrator) kann Referenzen nach Bedarf selbst sofort erstellen und verwalten.
- Die Referenzen können nur lokal geändert werden und sind nicht global (von außen) editierbar, dadurch benötigen wir keinen Zugriffsschutz.
- Wird bei der Modellierung von BPEL-Prozessen festgestellt, dass entsprechende Referenzen benötigt werden, können diese sofort angelegt und auch sofort in der Prozess-Modellierung verwendet werden.

7.1.2 Referenzen und Endpoint References

Nachdem die Architektur des Systems beschrieben wurde, folgt die Beschreibung des wichtigsten Teils des Systems, der Referenzen selbst. Dazu wird die Repräsentation der Referenzen anhand des WS-Addressing Konzepts der *Endpoint References* (EPR's) [25] beschrieben. Das Ziel bei der Definition der Referenzen ist, diese so flexibel und erweiterbar wie möglich zu realisieren und zu versuchen die vollständige Abgeschlossenheit der EPR's zu erreichen. Vollständige Abgeschlossenheit bedeutet dabei, dass alle für die Auflösung einer Referenz benötigten Informationen in der EPR selbst hinterlegt sind. Listing 5 zeigt das Schema einer EPR und wie diese zur Repräsentation einer Referenz genutzt werden. Im Schema und allen nachfolgenden Listings werden dabei folgende BNF Konventionen verwendet: “?” bezeichnet Optionalität (0 oder 1), “*” (0 oder mehr), “+” (1 oder mehr) und “|” steht für eine Auswahlmöglichkeit.

```
<wsa:EndpointReference>
  <wsa:Address>
    xs:anyURI
  </wsa:Address>
  <wsa:ReferenceProperties>
    <rrs:resolutionSystem>
      (xs:String | xs:anyURI |
       xs:QName)
    </rrs:resolutionSystem>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
    (xs:anyURI |
     xs:any)
  </wsa:ReferenceParameters>
  <wsa:PortType>xs:QName</wsa:PortType>
```

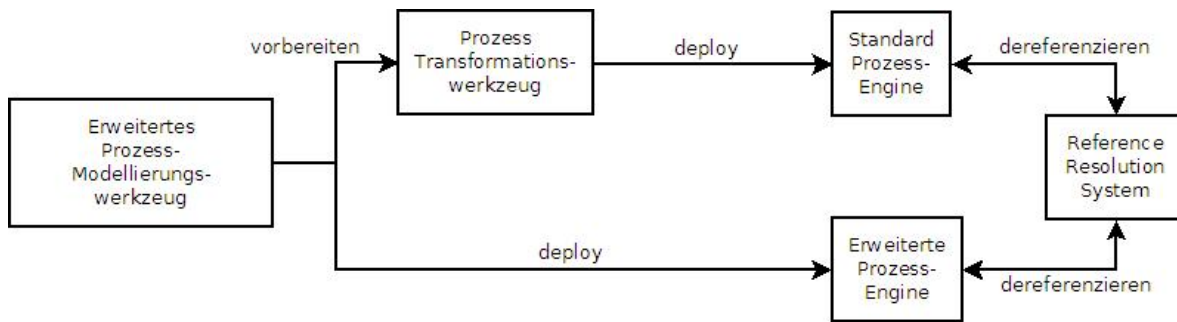


Abbildung 23: Alternativen zur Realisierung von Referenzen in BPEL, vgl. [3]

```

<wsa:ServiceName PortName="xs:NCName"??>
  xs:QName
</wsa:ServiceName>
<wsp:Policy>Policy</wsp:Policy>?
</wsa:EndpointReference>

```

Listing 5: EPR Schema [3]

Die verschiedenen Bestandteile einer EPR werden dabei für folgende Zwecke benötigt: Der **Address** Teil verweist auf den Endpunkt des RRS, das die Werte der Referenzen verwaltet.

In den **ReferenceProperties** wird der Adapter des RRS, der für das Auflösen der Referenz zuständig ist, angegeben. Alle Informationen, die der entsprechende Adapter zur Auflösung der Referenz benötigt, werden in den **ReferenceParameters** angegeben. Diese Informationen können entweder direkt angegeben werden, wie z.B. eine SQL-Query oder aber auch zentral gespeichert und anschließend über einen *Uniform Resource Identifier* (URI) referenziert werden. Beide Möglichkeiten haben Vorteile: Die direkte Angabe erlaubt es, die Query in der EPR während der Laufzeit zu ändern. Mit der Verwendung von gespeicherten Queries wird dafür sichergestellt, dass immer die gleiche Query ausgeführt wird und so niemand ungewollte Queries ausführen kann. **PortType** und **ServiceName** sind technische Parameter, die das dynamische Binden des RRS erlauben sollen. **Policy** ist optional und erlaubt das Hinzufügen von nichtfunktionalen Eigenschaften für die Ausführung der Queries. So kann z.B. *Quality of Context* (QoC) (Aktualität, Korrektheit, usw.), der für die zurückgelieferten Daten gelten soll, angegeben werden.

Weitere Details zu den Design-Entscheidungen und deren Nutzen finden sich in [3] unter "*Main Design Issues and Benefits of the Architecture*".

7.1.3 Realisierung von Referenzen in BPEL

In [3] werden zwei verschiedene Ansätze (siehe Abbildung 23) zur Integration des Referenzen-Konzepts in BPEL beschrieben und deren Vor- und Nachteile erläutert.

Für beide Konzepte muss ein entsprechendes Modellierungswerkzeug erweitert werden. In der Laufzeitumgebung unterscheiden sich dann die Konzepte. Die erste Variante besteht darin, den erweiterten BPEL-Quellcode unverändert auf einer erweiterten Workflow-Engine auszuführen. Das hat natürlich den entscheidenden Nachteil, dass nur noch eine entsprechend erweiterte Workflow-Engine diese Prozesse ausführen kann. Die zweite Variante umgeht dieses Problem, indem der veränderte BPEL-Quellcode in Standard-BPEL-Quellcode transformiert wird. Wir konzentrieren uns hier auf die Beschreibung der zweiten Variante, da diese einen universelleren Ansatz liefert, auf nahezu alle standard-konformen Workflow-Engines portiert werden kann und somit die Ausführungsumgebung des generierten BPEL-Quellcodes nicht eingeschränkt wird.

Erweiterung des Modellierungswerkzeugs

Wie bereits erwähnt, muss auf jeden Fall das Modellierungswerkzeug erweitert werden, um BPEL Prozesse mit Referenzen modellieren zu können. Zur Modellierung von solchen `<referenceVariable>` muss ein neuer Variablentyp eingeführt werden. Den Aufbau dieser Variablen zeigt das Schema in Listing 6.

```

<referenceVariable name="refName"
  valueType="xsd: schema"
  referenceType="onInstantiation | fresh |
periodic | external"
  period="duration"?
  external="partnerLink"? />*

```

Listing 6: Code eines <referenceVariable>-Schemas [3]

Dieses Schema ist eine Erweiterung des Standard BPEL Variablen-Schemas. Das Attribut **valueType** spiegelt den Datentyp der referenzierten Variable wider. Die Referenz-Variable selbst ist implizit vom Typ **xsd:EPR** und dient dazu, die eigentliche Referenz zu speichern. Über das Attribut **referenceType** kann angegeben werden, wie aktuell referenzierte Werte sein sollen, also wann und wie oft die in den Referenzen hinterlegten Werte aktualisiert werden sollen. In [3] werden vier solcher Aktualitätskonstanten vorgestellt, wobei eine Vielzahl weiterer solcher Konstanten denkbar ist. Ihre Bedeutung und Verwendung wird nachfolgend bei der Beschreibung der Modelltransformation aufgezeigt.

Transformation des Modells

Die Integration eines Modelltransformation-Zwischenschritts vor dem Deployment des Prozesses erlaubt uns, ein virtuelles Referenz-Handling für Prozesse zu realisieren, ohne dabei die den Prozess ausführende Engine zu modifizieren. Die Grundidee ist dabei, die Sprache BPEL nur im Modellierungswerkzeug, in dem wir zwischen normalen Variablen und Referenzvariablen unterscheiden, zu erweitern. Der zusätzliche Transformationsschritt generiert dabei standard-konforme BPEL Konstrukte für die Handhabung von Referenzen und speist diese in das originale Prozess-Modell ein. Für jede Referenz, die im Prozessmodell deklariert ist, werden dafür vier entsprechende Variablendeklarationen, wie in Listing 7 dargestellt, generiert.

```

<variable name="refName" type="xsd:schema"/>
<variable name="refNameEPR_Name" type="xsd:string"/>
<variable name="refNameEPR_Meta" type="xsd:EPR"/>
<variable name="refNameEPR_Ret" type="xsd:EPR"/>

```

Listing 7: Code der generierten Variablen-Deklaration, vgl. [3]

Die erste Variable wird dabei für die Haltung des eigentlichen Werts genutzt und die Zweite um den logischen Namen einer Referenz zu halten. Die beiden anderen Variablen halten die EPR der Referenz, die aus dem RRS über den Metadata Web Service abgerufen bzw. an den Retrieval Service geschickt wird. Weiterhin muss das RRS im Prozessmodell sichtbar sein. Dies wird durch die Generierung entsprechender <partnerLink>s realisiert (siehe Listing 8). Dabei muss sowohl ein Partner Link für den Metadata Web Service als auch für den Retrieval Web Service erstellt werden.

```

<bpel:partnerLink name="RRS_RET_Type"
  partnerLinkType="tns:RRS_RET_Type"
  partnerRole="get"/>

<bpel:partnerLink name="RRS_MetaData"
  partnerLinkType="tns:RRS_MD_Type"
  partnerRole="getEPR"/>

```

Listing 8: Code der RRS Partner Links

Um die Referenzen zur Laufzeit auflösen zu können, müssen die EPRs in den Prozess geladen werden. Dazu werden entsprechende Invoke-Aktivitäten auf den Metadata Web Service des RRS in das Prozessmodell eingefügt und die so eingelesenen EPRs in die *refNameEPR_Meta*-Variable gespeichert.

Zu guter Letzt zeigt Listing 9 die tatsächliche Dereferenzierung einer Referenz über den Retrieval Web Service des RRS. Der Modelltransformationsansatz speist Variablendeklarationen und Dereferenzierungsaktivitäten in den Workflow ein, um die Aktualisierung der Werte der Referenzen auszuführen. Auf den exakten Ablauf einer Transformation und deren Umsetzung wird am Ende dieses Abschnitts noch einmal genauer eingegangen.

```
<bpel:invoke name="refName_Refresh_#" partnerLink="RRS_RET_Type"
  operation="get" portType="ns:RRSRetrievalService"
  inputVariable="refNameEPR_Ret"
  outputVariable="refName">
```

Listing 9: Code einer Dereferenzierungs-Aktivität, vgl. [3]

Wie bereits weiter oben erwähnt, ermöglicht das Attribut **referenceType** dem Modellierer, eine von verschiedenen Aktualisierungsoptionen für die Werte der Referenzen auszuwählen. Im Rahmen des Projektes werden dabei nur die beiden Aktualisierungsoptionen *fresh* und *onInstantiation* umgesetzt, *periodic* und *external* sind zwei mögliche Erweiterungen. Abhängig von der Auswahl des Modellierers müssen die Dereferenzierungsaktivitäten an der entsprechenden Position im Prozess folgendermaßen eingefügt werden:

- *onInstantiation (default)*: Bei der Instanzierung des Prozesses werden die Werte vom RRS abgefragt und die Variablen entsprechend gesetzt. Diese Einstellung ist für die Definition von Konstanten, die beim Prozessstart gesetzt werden und während der ganzen Laufzeit des Prozesses unverändert bleiben, sinnvoll. Für jede Referenzvariable, die auf *onInstantiation* gesetzt ist, wird die RRS Invoke-Aktivität aus Listing 9 zu einer Sequenz, die bei der Prozessinstanzierung ausgeführt wird, hinzugefügt. Die Sequenz dieser Invokes wird dabei direkt nach dem CreateInstance-Receive des Prozessmodells eingefügt. Der Fall, dass es mehrere CreateInstance-Receives gibt, wird vernachlässigt, da er im wissenschaftlichen Bereich eher eine untergeordnete Rolle spielt.
- *fresh*: So frisch wie nur möglich - der Wert wird jedesmal abgefragt, wenn auf die Variable zugegriffen wird. Diese Einstellung ist nützlich, falls auf sich oft ändernde externe Werte, wie z.B. Sensordaten, zugegriffen werden soll. Hier muss die Dereferenzierungsaktivität direkt vor jeder Aktivität, die den Wert der Referenz liest, ausgeführt werden.
- *periodic(Erweiterung)*: Im Attribut **period** kann ein Zeitwert, wie z.B. 10 min, angegeben werden. Dieses Attribut beschreibt das maximale Alter (Zeitspanne seit letzter Aktualisierung), das ein lokal gespeicherter temporärer Wert einer Referenz haben darf. Nachdem diese Zeitspanne abgelaufen ist, wird der Wert aus dem RRS abgefragt und die temporäre Variable aktualisiert. Dafür wird ein **<onAlarm>** Element im globalen **<eventHandlers>** Element während der Transformation eingefügt. Diese Konstruktion liefert die periodische Aktualisierung von Werten über das wiederholte Abfragen dieser Werte aus dem RRS.
- *external(Erweiterung)*: Ein externes Event im Bereich der Web Service Orchestrierung ist typischerweise eine Nachricht, die an die Prozessinstanz geschickt wird (oder ein Signal, das für alle Instanzen eines Prozessmodells gültig ist). Wird dieser Wert als **referenceType** gesetzt, können Aktualisierungen der Werte von außen, durch das Senden entsprechender Nachrichten an die Prozess-Engine ausgelöst werden. Der Service, von dem solch eine Nachricht erwartet wird, kann im Attribut **external** angegeben werden. Im Transformationsschritt wird dazu ein **<onEvent>** Konstrukt im globalen **<eventHandlers>** Element eingefügt.

Da die Transformation eines Prozessmodells relativ schnell sehr komplex wird (verschachtelte Sequenzen, Flows, ...) wird die Transformation nur beispielhaft umgesetzt. Die Einschränkungen liegen dabei

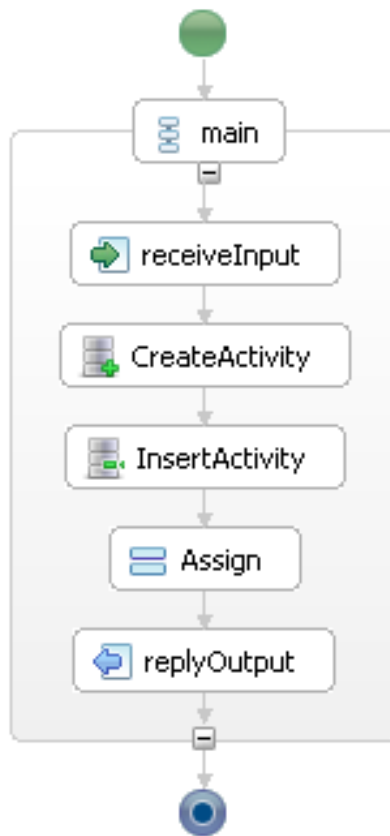


Abbildung 24: Beispiel Prozessmodell mit Referenzvariablen

darin, dass zum einen nur die oberste Sequenz des Prozessmodells transformiert wird (im BPEL Designer als “main” bezeichnet) und zum anderen eine Dereferenzierung nur in einer Assign-Aktivität mittels eines Befehls (*val(\$refName)*) angestoßen werden kann und in allen anderen Fällen die Referenz und nicht die Daten verwendet werden. Da die Transformation bereits in diesem rudimentären Status sehr komplex ist, wird diese nachfolgend Schritt für Schritt erklärt. Die oben angegebenen Beschreibungen dienen dazu als Grundlage und werden zum Teil noch weiter verfeinert.

Ein Prozessmodell wird nur transformiert falls es Referenzvariablen enthält und diese auch vollständig spezifiziert sind, d.h. ein Datentyp und eine Aktualisierungskonstante angegeben ist. Falls dies zutrifft, kann die Transformation über den in Kapitel 4.1.1 beschriebenen Button angestoßen werden, falls nicht wird eine entsprechende Fehlermeldung ausgegeben. Abbildung 24 zeigt das im Folgenden als Beispiel verwendete Prozessmodell und Abbildung 25 dessen Variablen vor der Transformation.

- Als erstes wird ein neues Projekt für die transformierten Dateien erstellt. Abbildung 26 zeigt ein BPEL-Projekt und das zugehörige neue Projekt für die transformierten Dateien, dessen Namen um das Kürzel “_TF” erweitert wird.
- Im nächsten Schritt werden nun die benötigten WSDLs des RRS (Retrieval und Metadata) über die in den Preferences hinterlegten Adressen heruntergeladen und im neu erstellten Projekt gespeichert.



Abbildung 25: Variablen eines Prozessmodells vor der Transformation

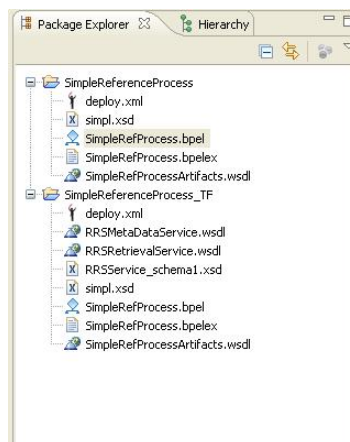


Abbildung 26: Workspace mit einem BPEL Projekt und dessen transformierter Kopie

- Nun wird die eigentliche Transformation angestoßen, d.h. die BPEL-Datei wird an den Transformation Web Service geschickt und dort transformiert:
 - Der Transformator erzeugt die RRS PartnerLinks aus Listing 8 für den Retrieval Web Service und den Metadata Web Service (siehe Abbildung 29).
 - Anschließend werden die in Listing 7 genannten Variablen erzeugt (siehe Abbildung 29).
 - Nun wird die in Abbildung 28 dargestellte “prepareEPR”-Sequence erzeugt. In dieser werden alle verwendeten EPRs aus dem RRS abgerufen und in den entsprechenden “refNameEPR_Meta”-Variablen gespeichert. Damit dies allerdings funktioniert müssen zuerst alle EPR-Namen (identisch mit den Referenzvariablenamen) in den entsprechenden “refNameEPR_Name”-Variablen hinterlegt werden. Dies geschieht über die “setEPR_Names”-Assign-Aktivität in der die Namen als fixe Werte hinterlegt werden. Nun können die EPRs über die entsprechenden “init_refNameEPR_Meta”-Invoke-Aktivitäten abgerufen werden. Dazu dienen die “refNameEPR_Name”-Variablen als Eingabevariablen und die “refNameEPR_Meta”-Variablen zur Speicherung der Rückgabewerte (EPRs). Um nun die EPRs später auflösen zu können müssen diese noch in das Nachrichtenformat des Retrieval Web Service umgewandelt werden. Dazu werden die EPRs einfach in der “copyEPRs”-Assign-Aktivität aus den “refNameEPR_Meta”-Variablen in die “refNameEPR_Ret”-Variablen kopiert.
 - Danach wird die oberste Sequenz des Prozessmodells durchlaufen und folgende Änderungen durchgeführt:
 - * Im Moment gibt es nur einen Fall in dem dereferenziert wird, diesen zeigt Abbildung 27. Dabei wird über eine XPath-Expression angegeben, dass der Wert der Referenzvariablen kopiert werden soll und nicht die EPR. Generell gilt, dass in allen anderen Fällen z.B. falls eine Referenzvariable in einer Invoke- oder Reply-Aktivität hinterlegt ist, immer die zugehörige EPR der Referenzvariablen verwendet wird und nicht die referenzierten Daten. An dieser Stelle müssen noch entsprechende Erweiterungen und ein geeignetes Konzept für die Unterscheidung zwischen EPRs und den referenzierten Daten in Kommunikationsaktivitäten erarbeitet werden.
 - * Dereferenzierung: Für alle Referenzvariablen, die als Aktualisierungskonstante *onInstantiation* gesetzt hatten, werden “refNameRefresh_0”-Invoke-Aktivitäten (siehe Listing 9) direkt nach der “prepareEPR”-Sequence eingefügt. Für alle Referenzvariablen, die als Aktualisierungskonstante *fresh* gesetzt hatten, werden “refNameRefresh_1”-Invoke-Aktivitäten (siehe Listing 9) direkt vor der jeweiligen Aktivität eingefügt. Die Zahl erhöht sich dabei für jede eingefügte “refNameRefresh_#”-Invoke-Aktivität.
 - * Sonst: Alle Referenzvariablen, die in Kommunikationsaktivitäten (Invoke, Receive und Reply) verwendet werden, werden bei der Transformation durch die entsprechenden “refNameEPR_Ret”-Variablen ausgetauscht. Ist allerdings tatsächlich gewünscht, dass eine EPR als Rückgabewert des Prozessmodells verwendet werden soll, dann muss vorerst die Prozess-WSDL noch von Hand entsprechend angepasst werden.
 - * Für die Zukunft wäre hier eine entsprechende Erweiterung sinnvoll, die zum Einen automatisch die Prozess-WSDL anpasst und zum Anderen eine Auswahlmöglichkeit zwischen referenzierten Daten und Referenzen in den jeweiligen Kommunikationsaktivitäten liefert.
- Die transformierte BPEL-Datei wird im neu erstellten Projekt gespeichert.
- Damit die beiden RRS Web Services aufgerufen werden können, müssen diese in der Prozess-WSDL bekannt gemacht werden. Dazu wird die originale WSDL-Datei eingelesen, die RRS-WSDLs importiert, die RRS Partner Link Typen definiert und die so veränderte WSDL in das neu erstellte Projekt gespeichert.

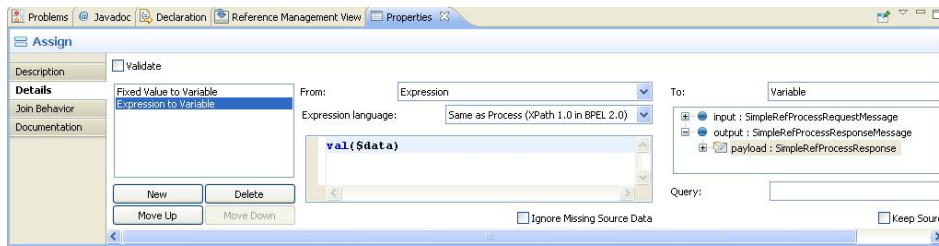


Abbildung 27: Beispiel für das Anstoßen der Dereferenzierung einer Referenzvariablen über eine Assign-Aktivität

- Zu guter Letzt werden alle unveränderten Dateien, wie z.B. XSD-Schemas, nicht transformierte BPEL-Dateien und WSDLs in das neu erstellte Projekt kopiert.
- Nun erhalten wir für unser Beispiel das in Abbildung 28 abgebildete Prozessmodell und dessen Variablen und Partner Links (siehe Abbildung 29) nach der Transformation.

7.2 Container-Referenzen in BPEL (IPVS-Referenzen)

Die Container-Referenzen sind einfache BPEL-Variablen, denen ein Datentyp mit z.T. spezieller Struktur zugewiesen wird und die alle für den Verweis auf eine Datenstruktureinheit notwendigen Informationen beinhalten. So können in BPEL-Variablen Verweise auf Datenstruktureinheiten (Tabellen, Schema, XML-Dokumente, Ordner, Dateien, usw.) von Datenquellen modelliert werden. Diese Container-Referenzen (BPEL-Variablen) können dann bei der Modellierung von Prozessen verwendet werden. Als Beispiel für einen Datentyp mit spezieller Struktur wird die Verwendung von Container-Referenzen in Abfragebefehlen von DM-Aktivitäten umgesetzt. Dafür wird der in Listing 10 dargestellte Datentyp bereitgestellt, mithilfe dessen die Angabe eines Schema- und Tabellennamens einer SQL-Abfrage in einer BPEL-Variable hinterlegt werden kann.

```
<xsd:complexType name="ContainerReferenceType">
  <xsd:sequence>
    <xsd:element name="schema" type="xsd:string"
      maxOccurs="1" minOccurs="0"/>
    <xsd:element name="table" type="xsd:string"
      maxOccurs="1" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

Listing 10: Schema des ContainerReferenceType-Datentyps

Weiterhin können natürlich auch alle einfachen Datentypen (String, Boolean, Integer, ...) für Container-Referenzen verwendet werden. Um zwischen speziellen und einfachen Container-Referenzen in den Abfragebefehlen unterscheiden zu können, werden diese entsprechend gekennzeichnet: *#bpelVariable#* signalisiert den Verweis auf eine einfache Container-Referenz und *[bpelVariable]* den Verweis auf eine speziell strukturierte Container-Referenz. Die so formatiert in den Abfragebefehlen hinterlegten BPEL-Variablen werden dann in der Workflow-Engine aufgelöst und durch die aktuellen Werte der BPEL-Variablen ersetzt.

Durch dieses Konzept lassen sich dann Prozessmodelle mit DM-Aktivitäten erstellen, die parametrisierbare Abfragebefehle enthalten. Um das Verständnis zu erhöhen, wird dies nachfolgend an einem kleinen Beispiel verdeutlicht (siehe Listing 11).

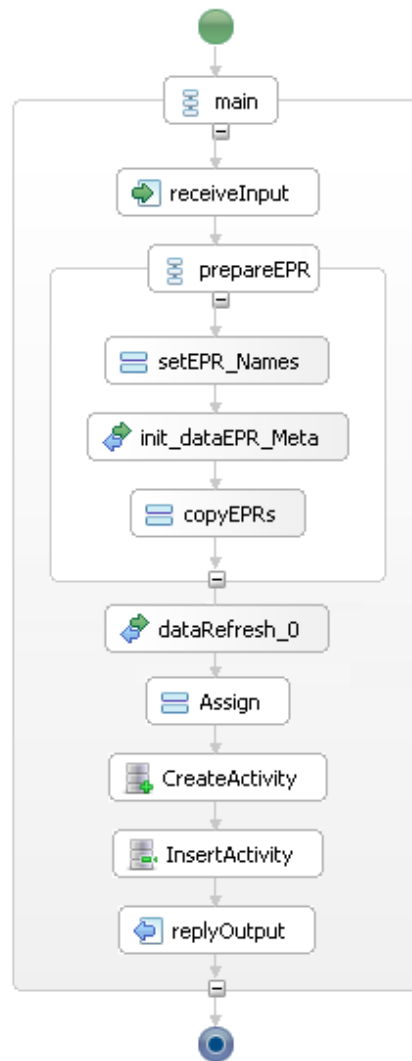


Abbildung 28: Transformiertes Beispiel Prozessmodell

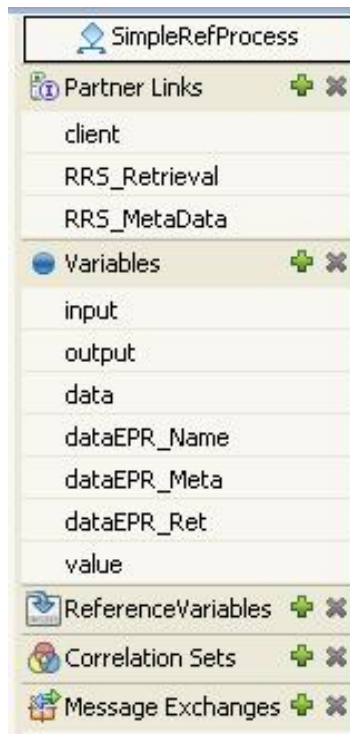


Abbildung 29: Variablen eines Prozessmodells nach der Transformation

```

1  ***
2
3  <bpel:variable name="target" type="simpl:containerReferenceType"/>
4  <bpel:variable name="columns" type="ns1:string"/>
5
6  ***
7
8  <bpel:extensionActivity>
9      <simpl:queryActivity name="QueryActivity"
10         dsKind="DB2" dsType="Database"
11         dsAddress="dd:myDB2" dsLanguage="SQL"
12         queryTarget="[target]"
13         dsStatement="SELECT #columns# FROM DATA">
14      </simpl:queryActivity>
15 </bpel:extensionActivity>
16
17 ***

```

Listing 11: Auszug aus einem Prozessmodell mit Container-Referenzen

Das Beispiel basiert auf einem einfachen Prozessmodell mit einer Query-Aktivität. Weiterhin werden zwei Container-Referenzen benötigt, eine vom Typ *string* für die Angabe der Spaltennamen einer Relation (Zeile 4) und eine vom Typ *containerReferenceType* für die Angabe der Zielrelation der Query-Aktivität (Zeile 3). Die Container-Referenzen können nun über eine Assign-Aktivität auf einen festen Wert gesetzt werden oder wie in unserem Fall aus der Eingabe des Prozesses zur Laufzeit gelesen werden. In Zeile 12 hinterlegen wir nun im queryTarget-Attribut der Query-Aktivität die Container-Referenz aus Zeile 3. In Zeile 13 hinterlegen wir im Abfragebefehl die Container-Referenz aus Zeile 4.

Nun können wir das so erstellte Prozessmodell deployen und mit geeigneten Werten für die hinterlegten Container-Referenzen instanzieren. In ODE werden dann die Container-Referenzen aufgelöst und durch die entsprechenden Werte ersetzt. So kann man ein- und dasselbe Prozessmodell mit verschiedenen Eingaben immer wieder instanzieren und ausführen ohne es neu deployen zu müssen.

7.3 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 7.3.2 identifizierten Funktionen, die für die Umsetzung der Datenmanagement-Patterns benötigt werden, werden dann in Abschnitt 7.3.3 neue benötigte BPEL-Aktivitäten abstrahiert, die dann später die entsprechenden Funktionen beinhalten und diese an die Datenquellen zur Ausführung schicken.

7.3.1 Datenmanagement-Patterns

In diesem Abschnitt werden alle Datenmanagement-Patterns, die zur Realisierung einer generischen Anbindung von verschiedenen Datenquellen benötigt werden, aufgeführt und beschrieben. Abbildung 30 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. 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.

Visual Paradigm for UML Community Edition [not for commercial use]

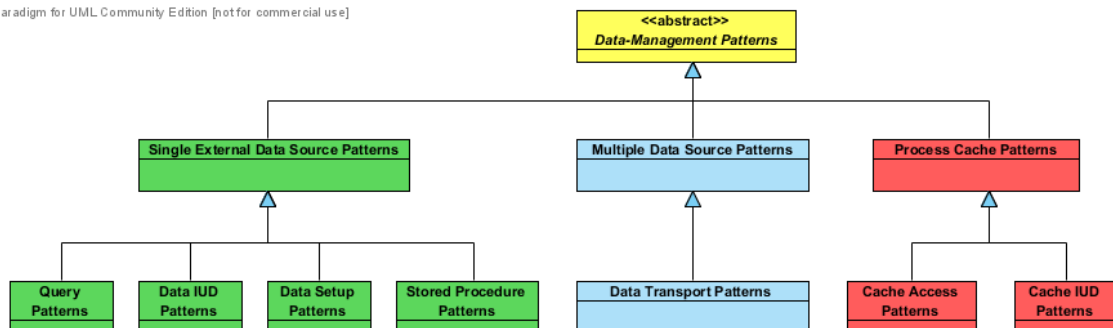


Abbildung 30: Klassenhierarchie der Datenmanagement-Patterns

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 realisieren, wobei der Prozessspeicher hier auch als eine Datenquelle angesehen wird.

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.

Visual Paradigm for UML Community Edition [not for commercial use]

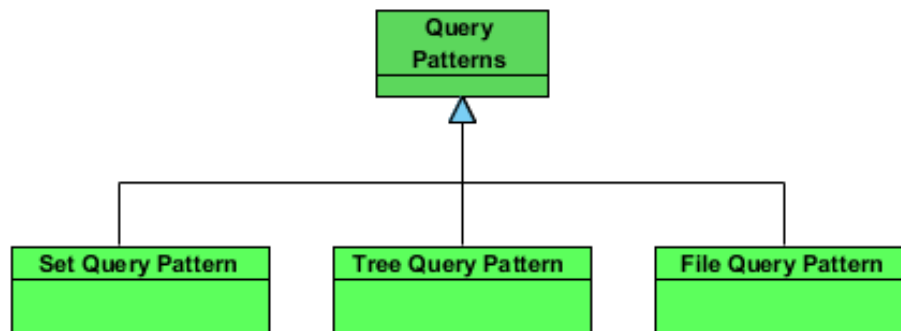


Abbildung 31: Übersicht über die Query Patterns-Klasse und deren Patterns

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 31 zu sehen ist.

- **Set Query Pattern:** Für die Anfrage von relationalen (mengenorientierten) Daten beispielsweise 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 beispielsweise mittels XQuery-Befehlen aus einer XML-Datenbank.

- **File Query Pattern:** Für die Anfrage von verschiedenartig strukturierten oder strukturlosen Daten mittels Systemaufrufen aus einem Dateisystem.

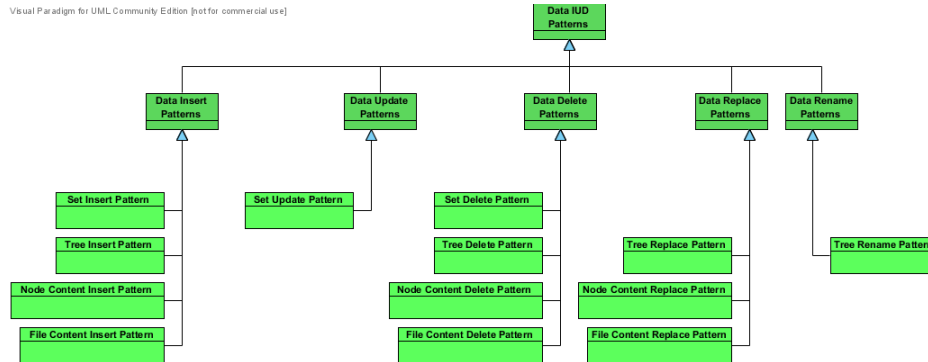


Abbildung 32: Übersicht über die Data IUD Patterns-Klasse und deren Patterns

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 7.3.2).

Abbildung 32 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 32 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 Pattern:** Diese Patterns beschreiben die Möglichkeit, auf mengenorientierten externen Daten z.B. in einem relationalen Datenbanksystem verschiedene Datenmanipulationsoperationen ausführen zu können. Zu diesen Operationen zählen das mengen- und tupelorientierte Einfügen (Set Insert Pattern), Aktualisieren (Set Update Pattern) und Löschen (Set Delete Pattern) von mengenorientierten Daten.
- **Tree Insert/Delete/Replace/Rename Pattern:** Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulationsoperationen auf Knoten, Sequenzen von Knoten oder auch Teilbäumen von baumartigen externen Daten z.B. 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 Datenmanipulationsoperationen in Knoten von baumartigen externen Daten z.B. 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 Datenmanipulationsoperationen 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.

Visual Paradigm for UML Community Edition [not for commercial use]

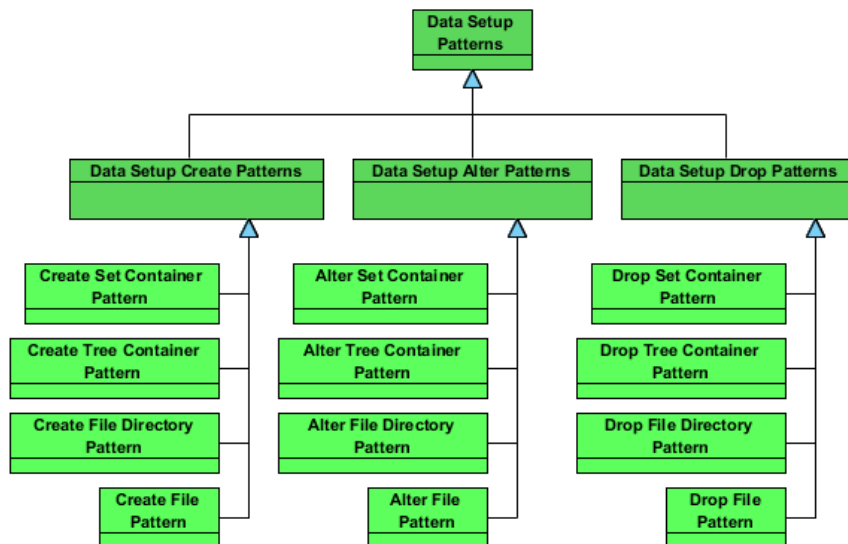


Abbildung 33: Übersicht über die Data Setup Patterns-Klasse und deren Patterns

Data Setup Patterns Die Klasse Data Setup Patterns liefert die Möglichkeit, Datenstrukturen zu ändern. So können während der Prozessausführung die Datenquellen konfiguriert oder neue Datenstruktureinheiten (Tabellen, Schema, XML-Dokumente, Ordner, Dateien, Buffer-Tabellen, usw.) erstellt, geändert oder verworfen werden.

Abbildung 33 zeigt die Data Setup Patternklasse (Data Setup Patterns), die sich in folgende drei operationsspezifische Unterklassen gliedert:

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

Alle drei Patternklassen werden durch die in Abbildung 33 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 einzelne Datenmodell.

- **Create/Alter/Drop Set Container Pattern:** Diese Patterns beschreiben die Möglichkeit, neue Relationen in einem relationalen Datenbanksystem zu erzeugen (Create Set Container Pattern), die Datenstrukturen vorhandener Relationen zu bearbeiten (Alter Set Container Pattern) oder

Relationen zu löschen (Drop Set Container Pattern). In Sensornetz-Datenbanken können mithilfe dieser Patterns auch Buffer-Tabellen erstellt und gelöscht werden (Beispiele siehe Abschnitt 7.3.2).

- Create/Alter/Drop Tree Container Pattern: Diese Patterns beschreiben die Möglichkeit, neue XML-Dokumente in einem XML-Datenbanksystem zu erzeugen (Create Tree Container Pattern), die Datenstrukturen vorhandener XML-Dokumente zu bearbeiten (Alter Tree Container Pattern) oder XML-Dokumente 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). Ein Beispiel für die Bearbeitung eines vorhandenen Ordners ist das Ändern seiner Zugriffsrechte.
- 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). Ein Beispiel für die Bearbeitung einer vorhandenen Datei ist das Ändern ihres Schreibschutzes.

Visual Paradigm for UML Community Edition [not for commercial use]

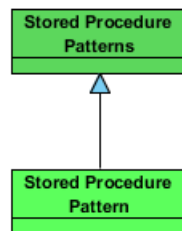


Abbildung 34: Übersicht über die Stored Procedure Patterns-Klasse und deren Patterns

Stored Procedure Patterns Abbildung 34 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.

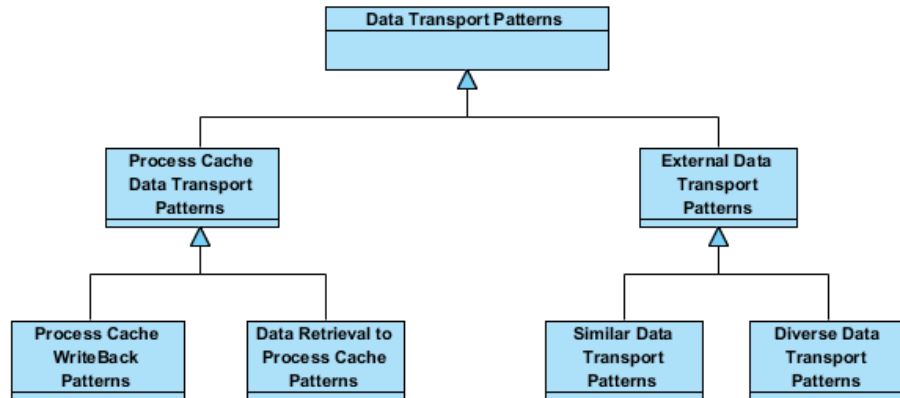


Abbildung 35: Übersicht über die Data Transport Patterns-Klasse und deren Unterklassen

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 35 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.

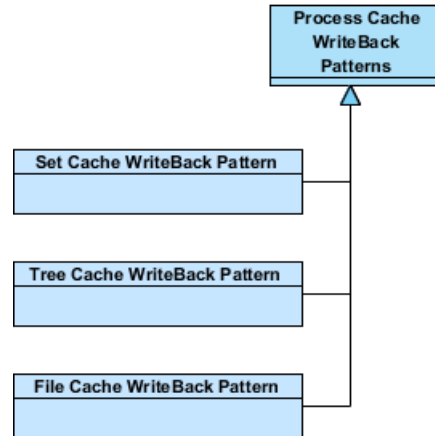


Abbildung 36: Übersicht über die Process Cache WriteBack Patterns-Klasse und deren Patterns

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

Abbildung 36 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 auf relationale Datenbanken anwendbar. Es ist nicht für die TinyDB verwendbar, da bei TinyDB keine Datenmanipulation möglich ist.
- Tree Cache WriteBack Pattern: Realisiert das Zurückschreiben von baumartigen Daten z.B. auf XML-Datenbanken.
- File Cache WriteBack. Pattern: Realisiert das Zurückschreiben von Dateiinhalten auf Dateisysteme.

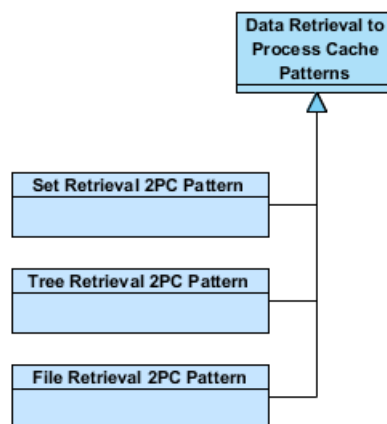


Abbildung 37: Übersicht über die Data Retrieval to Process Cache Patterns-Klasse und deren Patterns

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 37 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 z.B. 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.

Visual Paradigm for UML Community Edition [not for commercial use]

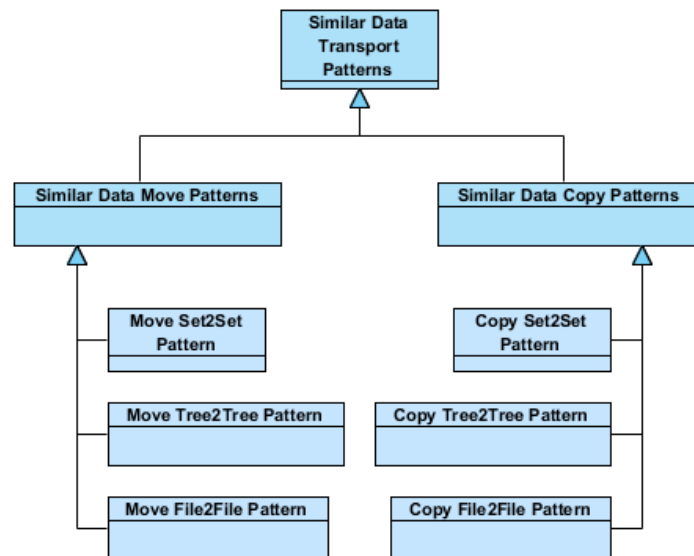


Abbildung 38: Übersicht über die Similar Data Transport Patterns-Klasse und deren Patterns

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 38 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 und die originalen Daten gelöscht werden.

- **Similar Data Copy Patterns:** Für das Kopieren von externen Daten auf externen Datenquellen, wobei die Datenstruktur und die originalen Daten unverändert bleiben.

Beide Patternklassen werden durch die in Abbildung 38 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 einzelne 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.

Visual Paradigm for UML Community Edition [not for commercial use]

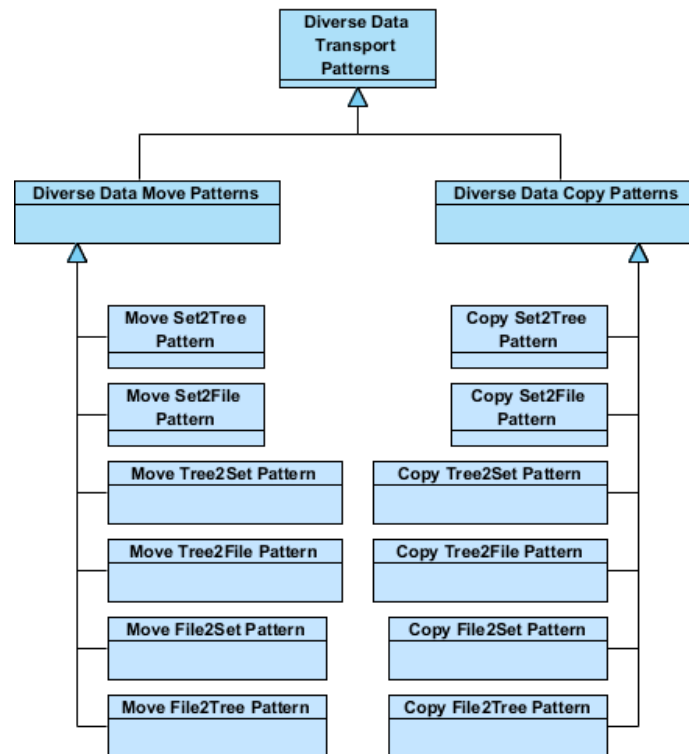


Abbildung 39: Übersicht über die Diverse Data Transport Patterns-Klasse und deren Patterns

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 39 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. Die originalen Daten werden gelöscht.
- **Diverse Data Copy Patterns:** Für das Kopieren von externen Daten auf externen Datenquellen mit veränderter Datenstruktur. Die originalen Daten bleiben unverändert.

Beide Patternklassen werden durch die in Abbildung 39 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 z.B. 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 Datenbanken, Sensornetz- 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 z.B. aus Dateisystemen auf eine relationale Datenbank als mengenorientierte 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 z.B. 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.

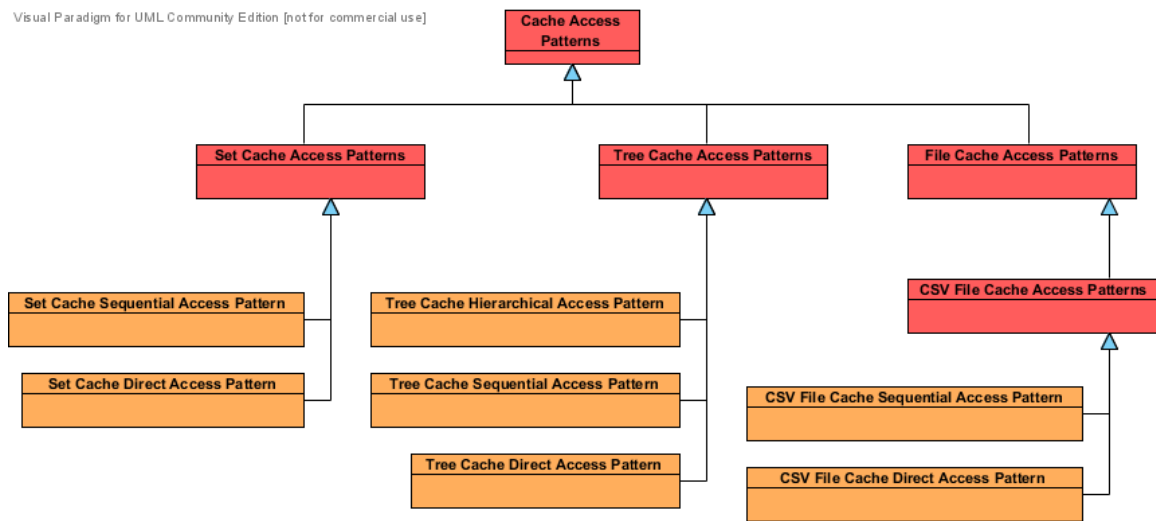


Abbildung 40: Übersicht über die Cache Access Patterns-Klasse und deren Unterklassen

Cache Access Patterns Die Klasse Cache Access Patterns beschreibt die Notwendigkeit, auf den im BPEL-Prozess erzeugten Prozessspeicher 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 40 zeigt die Cache Access Patterns, die sich in folgende drei Klassen gliedern:

- **Set Cache Access Patterns:** Die Set Cache Access Patterns liefern die Möglichkeit, auf den im BPEL-Prozess erzeugten mengenorientierten Prozessspeicher 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 Element oder umgekehrt 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 Prozessspeicher 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 auf einen Baum. Dies wird durch entsprechende Funktionen realisiert, die den linken oder rechten Geschwisterknoten bzw. den nächsten Knoten der nächst tieferen (höheren) Ebene, falls die aktuelle

Ebene bereits durchlaufen wurde, des ausgewählten Knoten liefern. Über den Wurzelknoten als Startpunkt kann jede Ebene des ganzen Baums sequentiell durchlaufen werden. Dabei kann in jedem Schritt immer nur der nächste oder vorherige Knoten aus der Knotensequenz erreicht werden.

- Tree Cache Direct Access Pattern: Liefert den direkten Zugriff auf einen einzelnen Knoten im Baum 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 Prozessspeichers 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 Separated 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 Pattern: Liefert den sequentiellen Zugriff auf die CSV-Daten, d.h. die Daten werden von der ersten Zeile bis zu der letzten Zeile zeilenweise durchlaufen, wobei in einem Schritt immer nur die vorherige oder die nachfolgende Zeile erreichbar ist.
 - * CSV File Cache Direct Access Pattern: 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.

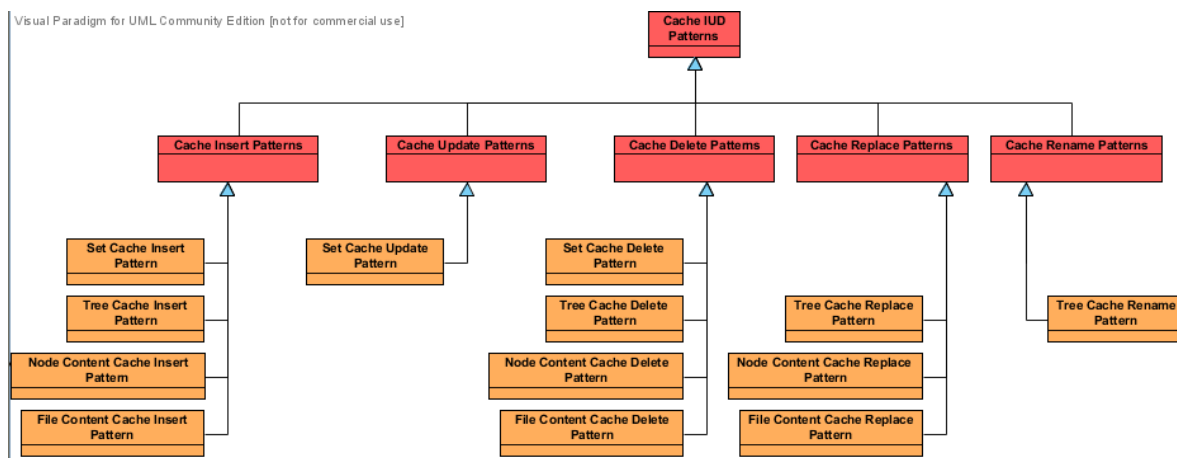


Abbildung 41: Übersicht über die Cache IUD Patterns-Klasse und deren Patterns

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

Abbildung 41 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 41 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 einzelne Datenmodell.

- **Set Cache Insert/Update/Delete Pattern:** Diese Patterns beschreiben die Möglichkeit, auf mengenorientierten internen Daten im Prozessspeicher verschiedene Datenmanipulationsoperationen ausführen zu können. Zu diesen Operationen zählen das mengen- und tupelorientierte Einfügen (Set Cache Insert Pattern), Aktualisieren (Set Cache Update Pattern) und Löschen (Set Cache Delete Pattern) von mengenorientierten Daten.
- **Tree Cache Insert/Delete/Replace/Rename Pattern:** Diese Patterns beschreiben die Möglichkeit, verschiedene Datenmanipulationsoperationen 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 Datenmanipulationsoperationen 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 Datenmanipulationsoperationen 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 Daten im Prozessspeicher.

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). Die External Data Transport Patterns werden nicht in der Tabelle aufgeführt, da sie im Rahmen dieses Projekts nicht umgesetzt werden.

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

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys.	Sensor.-DB	Cache
Set Query Pattern	x			x	
Tree Query Pattern		x			
File Query Pattern			x		
Set Insert Pattern	x				
Set Update Pattern	x				
Set Delete Pattern	x				
Tree Insert Pattern		x			
Tree Delete Pattern		x			
Tree Replace Pattern		x			
Tree Rename Pattern		x			
Node Content Insert Pattern		x			
Node Content Delete Pattern		x			
Node Content Replace Pattern		x			
File Content Insert Pattern			x		
File Content Delete Pattern			x		
File Content Replace Pattern			x		
Create Set Container Pattern	x				
Alter Set Container Pattern	x				
Drop Set Container Pattern	x				
Create Tree Container Pattern		x			
Alter Tree Container Pattern		x			
Drop Tree Container Pattern		x			
Create File Directory Pattern			x		
Alter File Directory Pattern			x		
Drop File Directory Pattern			x		
Create File Pattern			x		
Alter File Pattern			x		
Drop File Pattern			x		
Stored Procedure Pattern	x	x			
Set Retrieval 2PC Pattern	x			x	x
Set Cache Sequential Access Pattern					x
Set Cache Direct Access Pattern					x
Set Cache Insert Pattern					x
Set Cache Update Pattern					x
Set Cache Delete Pattern					x
Set Cache WriteBack Pattern	x				x

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys.	Sensor.-DB	Cache
Tree Retrieval 2PC Pattern		x			x
Tree Cache Hierarchical Access Pattern					x
Tree Cache Sequential Access Pattern					x
Tree Cache Direct Access Pattern					x
Tree Cache Insert Pattern					x
Tree Cache Delete Pattern					x
Tree Cache Replace Pattern					x
Tree Cache Rename Pattern					x
Node Content Cache Insert Pattern					x
Node Content Cache Delete Pattern					x
Node Content Cache Replace Pattern					x
Tree Cache WriteBack Pattern		x			x
File Retrieval 2PC Pattern			x		x
CSV File Cache Sequential Access Pattern					x
CSV File Cache Direct Access Pattern					x
File Content Cache Insert Pattern					x
File Content Cache Delete Pattern					x
File Content Cache Replace Pattern					x
File Cache WriteBack Pattern			x		x

7.3.2 Umsetzung der Datenmanagement-Patterns

In diesem Abschnitt wird die Realisierung der verschiedenen Datenmanagement-Patterns aus Abschnitt 7.3.1 im Zusammenhang mit den jeweiligen Datenquellentypen beschrieben. Einige Patterns werden dabei nicht oder nur für manche Datenquellen aufgrund ihrer Komplexität umgesetzt.

Relationale Datenbanken

Hier wird die Umsetzung der in Abschnitt 7.3.1 beschriebenen Patterns im Bereich der SQL-Datenbanken durch bestehende SQL-Befehle oder, falls erforderlich, durch die Definition neuer zu implementierender Funktionen beschrieben.

- Set Query Pattern: Realisierung durch SQL-SELECT.
 - Beispiel:
 - * SELECT info FROM customer
- Set Insert Pattern: Realisierung durch SQL-INSERT.
 - Beispiele:

- * INSERT INTO department VALUES ('E31', 'architecture', '00390', 'E01')
 - * INSERT INTO department SELECT * FROM old_department
- Set Update Pattern: Realisierung durch SQL-UPDATE.
 - Beispiel: UPDATE employee SET job = 'laborer' WHERE empno = '000290'
- Set Delete Pattern: Realisierung durch SQL-DELETE.
 - Beispiel: DELETE FROM department WHERE deptno = 'D11'
- Create Set Container Pattern: Realisierung durch SQL-CREATE SCHEMA, SQL-CREATE TABLE, SQL-CREATE VIEW und weitere SQL-CREATE 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 Set Container Pattern: Realisierung durch SQL-DROP.
 - Beispiele:
 - * DROP SCHEMA internal
 - * DROP TABLE tdept
 - * DROP VIEW internal.departmentView
- Alter Set Container Pattern: Wird nicht umgesetzt. Realisierung über SQL-ALTER Befehle möglich.
 - Beispiel: ALTER TABLE department DROP CONSTRAINT deptno
- Stored Procedure Pattern: Realisierung durch SQL-CALL.
 - Beispiel: CALL parts_on_hand (?, ?, ?) (? = Parameter der Prozedur)
- Set Retrieval 2PC 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 abgelegt 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 in den Prozessspeichers geladen. Eine etwas ausführlichere Beschreibung dieses 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.

- Set Cache Sequential/Direct Access Pattern: Um diese Patterns zu realisieren, müssen Methoden in BPEL bereitgestellt werden, die einen sequentiellen und direkten Zugriff auf den durch das Set Retrieval 2PC Pattern erzeugten Prozessspeicher 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 (siehe [4]) über die mengenorientierten Daten iteriert werden können.
- Set Cache Insert/Update/Delete Pattern: Erweitern die Set Cache Access Patterns Methoden um die Möglichkeiten, Tupel innerhalb der mengenorientierten Datenstruktur im Prozessspeicher 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.
- Set Cache WriteBack Pattern: Um die Daten aus dem Prozessspeicher zurück auf die originale Datenbank zu übertragen, muss eine neue BPEL-Aktivität erstellt werden, durch die der Benutzer angibt, dass die Daten zurückgeschrieben werden sollen. Diese Aktivität nutzt dann intern die Funktionalität der Data IUD Patterns und Cache Access Patterns sowie weitere 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 7.3.1 beschriebenen Patterns im Bereich der XQuery-Datenbanken durch bestehende XQuery-Befehle oder, falls erforderlich, durch die Definition neuer zu implementierender Funktionen beschrieben.

- Tree Query Pattern: Realisierung durch entsprechende FLWOR-Befehle.
 - Beispiel: `FOR $d IN fn:doc("department.xml")/employees/employee RETURN $d/name`
- Tree Insert Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility (siehe [22]).
 - Beispiel: `INSERT NODE <phone>123456</phone> AFTER fn:doc("department.xml")/employees/employee[1]/name`
- Tree Delete Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
 - Beispiel: `DELETE NODE fn:doc("department.xml")/employees/employee[1]/phone[last()]`
- Tree Replace Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
 - Beispiel: `REPLACE NODE fn:doc("department.xml")/employees/employee[1]/phone WITH fn:doc("department2.xml")/employees/employee[20]/phone`
- Tree Rename Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
 - Beispiel: `RENAME NODE fn:doc("department.xml")/employees/employee[1]/phone[2] AS "privatePhone"`
- Node Content Insert Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
 - Beispiel:

- Node Content Delete Pattern: Realisierung durch entsprechende Befehle der XQuery Update Facility.
 - Beispiel:
- Node Content Replace 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
- Create Tree Container Pattern: Wird nicht realisiert.
- Alter Tree Container Pattern: Wird nicht realisiert.
- Drop Tree Container Pattern: Wird nicht realisiert.
- Stored Procedure Pattern: Wird nicht realisiert.
- Tree Retrieval 2PC Pattern: Die Umsetzung erfolgt nach der Beschreibung des Set Retrieval 2PC Patterns für relationale Datenbanken, wobei hier XQuery-Befehle verwendet und baumartige Daten in einer BPEL-Variable abgelegt werden.
- Tree Cache Hierarchical/Sequential/Direct Access Pattern: Werden nicht realisiert.
- Tree Cache Insert/Delete/Replace/Rename Pattern: Werden nicht realisiert.
- Node Content Cache Insert/Delete/Replace Pattern: Werden nicht realisiert.
- Tree Cache WriteBack Pattern: Wird nicht realisiert.

Dateisysteme

Hier wird die Umsetzung der in Abschnitt 7.3.1 beschriebenen Patterns im Bereich der Dateisysteme durch entsprechende Systemaufrufe des dem Dateisystem zugrundeliegenden Betriebssystems beschrieben. Die Umsetzung der Patterns für Dateisysteme beschränkt sich dabei auf das CSV-Dateiformat.

- File 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 nicht realisiert.
- File Content Insert/Delete/Replace Pattern: Realisierung durch einen entsprechenden PUT- bzw. REMOVE-Befehl (RM), die intern java.io Methoden verwenden, um Inhalte in Dateien zu schreiben, zu ersetzen und aus ihnen zu löschen. Ein Ersetzen 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, ersetzt und gelöscht werden können. Es wird nur das Einfügen, Ersetzen und Löschen von vollständigen Dateiinhalten umgesetzt. Das Einfügen, Ersetzen und Löschen von Dateiinhalten wird nicht realisiert.

- Create/Alter/Drop File Pattern: Realisierung z.B. durch die Verwendung entsprechender Erzeugungsbefehle, wie MKFILE zur Erzeugung oder RMFILE für das Löschen von Dateien. Diese werden intern wieder durch java.io-Methoden realisiert. Hier sollte es auf jeden Fall möglich sein, Dateien in einem Dateisystem zu erzeugen. Ebenso sollte das Löschen auf der gleichen Ebene, d.h. von ganzen Dateien, möglich sein. Andere Operationen wie z.B. das Ändern der Zugriffsrechte von Dateien mit dem Befehl CHMOD werden nicht umgesetzt.
- Create/Alter/Drop File Directory Pattern: Realisierung ebenso wie bei den Create/Alter/Drop File Patterns, hier allerdings für Ordner, die z.B. mit dem Erzeugungsbefehl MKDIR erstellt und mit RMDIR gelöscht werden können. Weitere Operationen werden nicht realisiert.
- File Retrieval 2PC Pattern: Die Umsetzung erfolgt nach der Beschreibung des Set Retrieval Patterns für relationale Datenbanken, wobei hier entsprechende Dateisystem-Befehle (siehe Query Pattern) verwendet werden. Nur vollständige Dateiinhalt können in einer BPEL-Variable abgelegt werden. Weiterhin beschränkt sich die Umsetzung dieses Patterns auf das CSV-Dateiformat.
- CSV File Cache Sequential/Direct Access Pattern: Um diese Patterns zu realisieren, werden die CSV-Daten bei der Ausführung des File Retrieval 2PC Patterns in eine relationale Datenstruktur im Prozessspeicher überführt. Das ist ohne Probleme möglich, da die CSV-Datenstruktur sehr der einer Relation ähnelt. Dadurch können auch für diese Patterns die durch die Set Cache Access Patterns bereitgestellten Methoden verwendet werden.
- CSV File Cache Insert/Delete/Replace Pattern: Erweitern die CSV File Cache Access Patterns Methoden um die Möglichkeiten, Werte innerhalb des Datencaches des BPEL-Prozesses zu ersetzen, einzufügen und zu löschen. Die SDO API liefert dafür bereits einige Methoden, die verwendet werden können. Da auch hier nur CSV-Daten verarbeitet werden, kann aus den oben beschriebenen Gründen auch hier wieder auf die Funktionalität für mengenorientierte Daten (Set Cache Patterns) zurückgegriffen werden.
- File Cache WriteBack Pattern: Wird nur für das CSV-Dateiformat umgesetzt.

Sensornetze

Hier wird die Umsetzung der in Abschnitt 7.3.1 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]).

- Set Query Pattern: Realisierung durch ein entsprechendes SQL-SELECT der Sensornetz-Datenbank. Eine Abfrage läuft auf der Sensornetz-Datenbank dabei folgendermaßen ab: Die Sensornetz-Datenbank liest die durch den Select-Befehl angeforderten Werte aus den jeweiligen Sensoren des Sensornetzes aus, filtert diese und verknüpft sie zu einer Ergebnismenge, die dann als Ergebnis der Abfrage zurückgegeben wird. Alternativ zu normalen Abfragen besteht die Möglichkeit, sensornetzintern in einem Buffer Werte zwischenspeichern. Realisiert wird dies durch im Arbeitsspeicher von Sensoren erstellte Tabellen (siehe Create Sensor Buffer Container 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 Methoden 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 nodeid, temp EPOCH DURATION 1024
 - * 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.*
- Create/Drop Set Container 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, uint32, 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 2PC Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.
 - Set Cache Sequential/Direct Access Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.
 - Set Cache Insert/Update/Delete/Rename Pattern: Realisierung ebenso wie bei SQL-Datenbanken beschrieben.

Datentransport zwischen mehreren externen Datenquellen

Hier wird die Umsetzung der in Abschnitt 7.3.1 beschriebenen External Data Transport Patterns beschrieben. Dies sind die Similar Data Transport Patterns und die Diverse Data Transport Patterns.

- Similar/Diverse Data Move Patterns: Die Patterns dieser Klasse werden dadurch realisiert, dass entsprechende vorhandene Patterns der Quell- und Zieldatenquelle und des Prozessspeichers genutzt werden. Am Beispiel des Move Set2Set Patterns würde dies folgendermaßen aussehen:
 - Mithilfe des Set Retrieval 2PC Patterns werden die Daten aus der Quell-Datenbank abgefragt und in den Prozessspeicher geladen.
 - Mit dem Create Set Container Pattern wird auf der Ziel-Datenbank eine neue Tabelle erzeugt.
 - Anschließend werden die abgefragten Daten mit dem Set Cache WriteBack Pattern in die erstellte Tabelle auf der Ziel-Datenbank eingefügt und

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

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys. (CSV)	Sensor.-DB	Cache
Set Query Pattern	x				
Tree Query Pattern					
File Query Pattern			x		
Set Insert Pattern	x				
Set Update Pattern	x				
Set Delete Pattern	x				
Tree Insert Pattern					
Tree Delete Pattern					
Tree Replace Pattern					
Tree Rename Pattern					
Node Content Insert Pattern					
Node Content Delete Pattern					
Node Content Replace Pattern					
File Content Insert Pattern			x		
File Content Delete Pattern			x		
File Content Replace Pattern					
Create Set Container Pattern	x				
Alter Set Container Pattern					
Drop Set Container Pattern	x				
Create Tree Container Pattern					
Alter Tree Container Pattern					
Drop Tree Container Pattern					

- die Ursprungstabelle der abgefragten Daten mit dem Drop Set Container Pattern auf der Quell-Datenbank gelöscht.
- Similar/Diverse Data Copy Patterns: Die Patterns dieser Klasse werden ebenfalls dadurch realisiert, dass entsprechende vorhandene Patterns der Quell- und Zieldatenquelle und des Prozessspeichers genutzt werden. Am Beispiel des Copy Set2Set Patterns würde dies folgendermaßen aussehen:
 - Mithilfe des Set Retrieval 2PC Patterns werden die Daten aus der Quell-Datenbank abgefragt und in den Prozessspeicher geladen.
 - Mit dem Create Set Container Pattern wird auf der Ziel-Datenbank eine neue Tabelle erzeugt.
 - Abschließend werden die abgefragten Daten mit dem Set Cache WriteBack in die erstellte Tabelle auf der Ziel-Datenbank eingefügt.

Tabelle 3 zeigt eine Übersicht, welche Datenmanagement-Patterns für welche Datenquellentypen umgesetzt werden. Ein “x” markiert dabei die vollständige Umsetzung eines Patterns für einen Datenquellentyp.

Datenmanagement-Pattern	rel. DB	XML-DB	Dateisys. (CSV)	Sensor.-DB	Cache
Create File Directory Pattern			x		
Alter File Directory Pattern					
Drop File Directory Pattern			x		
Create File Pattern			x		
Alter File Pattern					
Drop File Pattern			x		
Stored Procedure Pattern	x				
Set Retrieval 2PC Pattern	x				
Set Cache Sequential Access Pattern					?
Set Cache Direct Access Pattern					?
Set Cache Insert Pattern					?
Set Cache Update Pattern					?
Set Cache Delete Pattern					?
Set Cache WriteBack Pattern					
Tree Retrieval 2PC Pattern					
Tree Cache Hierarchical Access Pattern					
Tree Cache Sequential Access Pattern					
Tree Cache Direct Access Pattern					
Tree Cache Insert Pattern					
Tree Cache Delete Pattern					
Tree Cache Replace Pattern					
Tree Cache Rename Pattern					
Node Content Cache Insert Pattern					
Node Content Cache Delete Pattern					
Node Content Cache Replace Pattern					
Tree Cache WriteBack Pattern					
File Retrieval 2PC Pattern			?		
CSV File Cache Sequential Access Pattern					?
CSV File Cache Direct Access Pattern					?
File Content Cache Insert Pattern					?
File Content Cache Delete Pattern					?
File Content Cache Replace Pattern					?
File Cache WriteBack Pattern					

7.3.3 Resultierende BPEL-Aktivitäten

In diesem Abschnitt werden die durch Abschnitt 7.3.2 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 fünf Attribute ***dsType***, ***dsKind***, ***dsLanguage***, ***dsAddress*** und ***dsStatement*** vom Typ String besitzen. Das Attribut ***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. Das Attribut ***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. Das Attribut ***dsLanguage*** dient zur Angabe der verwendeten Abfragesprache. Diese Informationen sind wichtig, um intern den richtigen SQL-Dialekt bzw. Befehlssatz für die entsprechende Datenquelle auszuwählen. Das Attribut ***dsAddress*** dient zur Angabe der Datenquellenadresse und ***dsStatement*** zur Haltung des entsprechenden Systemaufrufs bzw. SQL- oder XQuery-Befehls, der ausgeführt werden soll. Da jede der definierten Aktivitäten diese fünf Attribute besitzt, werden diese nachfolgend als ***simpl-attributes*** (vgl. standard-attributes in [4]) in den Listings angegeben und nur falls benötigt weitere aktivitätsspezifische Attribute beschrieben. Listing 12 führt die fünf Datenmanagement-Attribute noch einmal auf. Weiterhin werden die verschiedenen Ausprägungen der einzelnen Aktivitäten im Hinblick auf die zugrundeliegende Datenquelle aufgezeigt, sodass am Ende ein vollständiger Überblick aller definierten Aktivitäten und ihrer Ausprägungen vorliegt. Generell gibt es durch die verschiedenen Datenquellen keine strukturellen Unterschiede in den Aktivitäten. Dadurch spielen datenquellenspezifische Eigenschaften keine Rolle in der graphischen Oberfläche, wodurch eine einheitliche Benutzeroberfläche für alle Aktivitäten realisiert werden kann. Alle Abfragebefehle sollen über entsprechende graphische Elemente angegeben werden können, indem sie einfach "zusammengeklickt" werden können (siehe Abbildung 10). 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 Datenquellsprache bzw. der vollständige Befehlssatz der Datenquelle zur Verfügung. Der volle Sprachumfang kann dabei mindestens, wie in Abbildung 10 dargestellt, über die Angabe von Befehlen im Abfragebefehls-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 7.3.1 und 7.3.2 identifizierten Aktivitäten aufgezeigt und beschrieben. Dabei werden in den Listings der DM-Aktivitäten die in [4] beschriebenen Notationen und Definitionen verwendet.

```
dsType="xsd:string"  
dsKind="xsd:string"  
dsLanguage="xsd:string"  
dsAddress="xsd:string"  
dsStatement="xsd:string"
```

Listing 12: Listing der simpl-attributes

Query Aktivität

Diese Aktivität setzt die Query Patterns um und ermöglicht es, aus jeder beliebigen Datenquelle Daten zu lesen und diese auf derselben Datenquelle zu speichern. Dafür wird ein spezifisches Attribut *query-Target* 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 kann auch für das sensornetzinterne Einfügen von Sensordaten in Buffer-Tabellen

genutzt werden (wie auch in Abschnitt 7.3.2 beschrieben). Listing 13 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 13: Schema einer Query Aktivität

Insert Aktivität

Diese Aktivität setzt die Data Insert Patterns um. Sie ermöglicht es, Daten in eine Datenquelle einzufügen. Listing 14 zeigt das BPEL-Schema einer Insert Aktivität.

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

    standard-elements

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

Listing 14: Schema einer Insert Aktivität

Update Aktivität

Diese Aktivität setzt die Data Update/Replace/Rename Patterns um. Sie ermöglicht es, auf einer Datenquelle hinterlegte Daten zu aktualisieren. Diese Aktivität kann nicht auf Sensornetze angewendet werden. Listing 14 zeigt das BPEL-Schema einer Update Aktivität.

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

    standard-elements

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

Listing 15: Schema einer Update Aktivität

Delete Aktivität

Diese Aktivität setzt die Data Delete Patterns um. Sie ermöglicht es, Daten in Datenquellen zu löschen. Diese Aktivität kann nicht auf Sensornetze angewendet werden. Listing 16 zeigt das BPEL-Schema einer Delete Aktivität.

```

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

    standard-elements

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

```

Listing 16: Schema einer Delete Aktivität

Create Aktivität

Diese Aktivität setzt die Data Setup Create Patterns 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
 - mögliche Erweiterungen:
 - * CREATE VIEW
 - * CREATE DOMAIN
 - * CREATE INDEX
 - * CREATE TRIGGER
- mögliche Erweiterungen für Sensornetze:
 - CREATE BUFFER

Diese Aktivität wird für XQuery-Datenbanken nicht realisiert. Ebenso werden keine weiteren Befehle umgesetzt. Listing 17 zeigt das BPEL-Schema einer Create Aktivität.

```

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

    standard-elements

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

```

Listing 17: Schema einer Create Aktivität

Drop Aktivität

Diese Aktivität setzt die Data Setup Drop Patterns 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
 - mögliche Erweiterungen:
 - * DROP VIEW
 - * DROP DOMAIN
 - * DROP INDEX
 - * DROP TRIGGER
- mögliche Erweiterungen für Sensornetze:
 - DROP ALL

Diese Aktivität wird für XQuery-Datenbanken nicht realisiert. Ebenso werden keine weiteren Befehle umgesetzt. Listing 18 zeigt das BPEL-Schema einer Drop Aktivität.

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

    standard-elements

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

Listing 18: 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 wird nur für relationale Datenbanken umgesetzt. Listing 19 zeigt das BPEL-Schema einer Call Aktivität.

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

    standard-elements

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

Listing 19: Schema einer Call Aktivität

RetrieveData Aktivität

Diese Aktivität setzt die Data Retrieval to Process Cache Patterns um. Sie ermöglicht es, Daten von Datenquellen in den BPEL-Prozess zu laden. Dafür wird das zusätzliche Attribut *dataVariable* verwendet, in das eine Referenz auf eine BPEL-Variable abgelegt werden kann. Diese Aktivität wird nur für relationale Datenbanken und CSV-Dateien umgesetzt. Listing 20 zeigt das BPEL-Schema einer RetrieveData Aktivität.

```
<bpel:extensionActivity>
  <simpl:retrieveDataActivity
    standard-attributes
    simpl-attributes
    dataVariable="bpel:BPELVariableName">

    standard-elements

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

Listing 20: Schema einer RetrieveData Aktivität

Mögliche Erweiterungen:

- **WriteBack Aktivität:** Diese Aktivität setzt die Process Cache WriteBack Patterns um. Sie ermöglicht es, Daten aus dem BPEL-Prozess auf eine externe Datenquelle zu kopieren. Diese Aktivität existiert nicht für Sensornetze.
- **Alter Aktivität:** Diese Aktivität setzt die Data Setup Alter Patterns 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.
- **Transfer Aktivität:** Diese Aktivität setzt die External Data Transport Patterns um. Sie ermöglicht 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 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).

In Tabelle 4 werden noch einmal 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 und ein “m” dafür, dass eine Anwendung einer Aktivität auf diese Datenquelle theoretisch möglich ist, aber nicht umgesetzt wird. Die Markierung “-” bedeutet, dass eine Aktivität aufgrund der in Abschnitt 7.3.1 und 7.3.2 beschriebenen Umstände überhaupt nicht mit dieser 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	m	x	m
Insert Aktivität	x	m	x	-
Update Aktivität	x	m	m	-
Delete Aktivität	x	m	x	-
Create Aktivität	x	m	x	m
Drop Aktivität	x	m	x	m
Call Aktivität	x	m	-	-
RetrieveData Aktivität	x	m	x	m
WriteBack Aktivität	m	m	m	-
Alter Aktivität	m	m	m	-

7.4 Authentifizierung und Autorisierung

Dieser Abschnitt behandelt die Authentifizierung und Autorisierung beim 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. zwischenspeichern. Ziel des Rahmenwerks ist es, eine Vielzahl an Authentifizierungs- und Autorisierungsverfahren zu unterstützen und für weitere Verfahren erweiterbar zu sein.

7.4.1 Authentifizierung

Bei der Authentifizierung wird vor dem eigentlichen Zugriff zunächst ein Vertrauensverhältnis zwischen Klient 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 auf dieselbe Datenquelle 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 im Deployment-Deskriptor (siehe Kapitel 4.1.2) oder über das UDDI Web Interface (siehe Kapitel 4.4.2) für eine Datenquelle festgelegt werden.

7.4.2 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. Es wird lediglich die Autorisierung über Benutzername und Passwort unterstützt.

7.5 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.

7.5.1 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 von der ODE Engine erzeugt werden um Rückmeldung über bestimmte Aktionen die innerhalb der Engine auftreten zu geben. Bei diesen Ereignissen handelt es sich zum Beispiel um das Aktivieren eines Prozesses oder das Erzeugen einer neuen Prozessinstanz. Die Events sind in folgende fünf Gruppen eingeteilt: Instance lifecycle Events, Activity lifecycle Events, Scope Handling, Data Handling, Correlation Events. Diese Events machen es möglich zu verfolgen, was innerhalb der ODE Engine passiert und produzieren detaillierte Informationen über die Prozessausführung. Sie können mit Hilfe der Management API oder durch die Nutzung von sogenannten Event Listnern abgefragt werden.

Die Events werden momentan in der internen ODE Derby Datenbank gespeichert. Im Rahmen des SIMPL Auditing ist es geplant diese in einer externen Datenbank zu speichern.

Event Listener

Event Listener sind bestimmte Konstrukte, die es ermöglichen, bei dem Auftreten bestimmter Events eine direkte Rückmeldung zu geben, oder auf verschiedene Events zu reagieren und event-spezifische Aktionen durchzuführen. Diese Event Listener werden momentan zum Beispiel dazu benutzt um die verschiedenen event-spezifischen Informationen an das entsprechende Event DAO weiterzuleiten wo diese persistent gespeichert werden.

7.5.2 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 7). 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 abzuheften. Die so erzeugten Daten über die Prozesse, Instanzen und Events werden anschließend in einer zuvor in der Adminkonsole festgelegten Auditing-Datenbank abgelegt.

7.5.3 Event Modell

In diesem Abschnitt wird das Event Modell für das Auditing von SIMPL aufgeführt. Das SIMPL Event-Modell nutzt als Grundlage das Modell in Abbildung 42 welches ein Event Modell für die Ausführung von allgemeinen Aktivitäten ist. Da es sich dabei jedoch um ein WS-BPEL 2.0 Event-Modell handelt

sind einige Anpassungen und Änderungen notwendig. Der Grund dafür ist, dass die in diesem Modell vorkommenden Events nicht mit den Events in ODE identisch sind.

Die verschiedenen Schritte der Aktivitätsausführung die in Abbildung 42 aufgeführt sind (Inactive, Ready, Executing etc.) können bis auf eine Ausnahme für das SIMPL Event-Modell genutzt werden. Der Grund dafür ist, dass der Ablauf der Ausführung in ODE dem in diesem Modell gleicht, d. h. es werden die selben Zustände durchlaufen.

Die SIMPL Events werden nur während der direkten Ausführung der Aktivität genutzt (d.h. im Zustand "Executing" in Abbildung 42). Es genügt daher, das Event Modell darauf zu beschränken was im Zustand "Executing" passiert, da es in allen anderen Zuständen keinen neuen Events gibt.

In ODE selbst gibt es für Aktivitäten nur die fünf Events: Activity enabled, Activity execution started, Activity execution ended, Activity failure und Activity recovery. Da es in ODE keine Entsprechung der Events Complete Activity oder Activity_Completed (Übergang vom Zustand "Waiting" zu "Complete") gibt, sondern nur das Event Activity execution ended, geht ODE vom Zustand Execution direkt zum Zustand Complete über.

Das SIMPL-Event-Modell wurde daher so aufgebaut, das die Zustände "Ready" und "Complete", sowie "Faulted" übernommen wurden. Der Zustand "Executing" wird nicht explizit dargestellt, statt dessen werden die einzelnen Vorgänge die zwischen "Ready" und "Complete" auftreten näher veranschaulicht. Der Zustand "Ready" stellt daher den Anfangszustand des SIMPL-Event-Modells da und der Zustand "Complete" den Endzustand.

Zur Vereinfachung wurden im SIMPL-Event-Modell nur die SIMPL-Events aufgeführt und keine der bestehenden ODE-Events. Das darauf resultierende Event Modell ist in Abbildung 43 zu sehen. Dabei ist zu beachten, dass die Zustände die aus dem Modell in Abbildung 42 übernommen wurden in Rot dargestellt sind.

Wie im SIMPL-Event-Modell zu sehen ist gibt es zwei neue Event Klassen. 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. Es soll zukünftig, im Rahmen der Granularität möglich sein, dass diese neuen Event Klassen neben den bestehenden fünf Event Klassen, bei der Erstellung eines Deployment Deskriptors aktiviert oder deaktiviert werden können. Es ist nicht vorgesehen dies im Rahmen des Projekts umzusetzen.

Connection_Events Diese Events werden erzeugt, um verschiedene Ereignisse bei der Verbindung zu einer Datenquelle zu erfassen. Sie enthalten Informationen darüber zu welcher Datenbank eine Verbindung aufgebaut wurde (Typ der Datenbank und ihre Adresse) und wann die Verbindung aufgebaut, bzw. beendet wurde. Diese Events sind notwendig um festzuhalten wann auf eine bestimmte Datenbank zugegriffen wurde und dies eventuell in einem Monitoringtool darzustellen (z.B. Anzahl der Verbindungen zu einer bestimmten Datenbank in den letzten 24 Stunden).

- ConnectionStarted

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

- ConnectionEnd

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

- ConnectionLost

Dieses Event wird erzeugt, wenn es zu einem 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 alle relevanten Informationen zu den Aktivitäten (z.B. Name der Aktivität, die Art der Datenbank auf der die DM-Operation durchgeführt wird, das zu übergebende Statement etc.). Diese speziellen Events werden zusätzlich zu den normalen Activity_Events

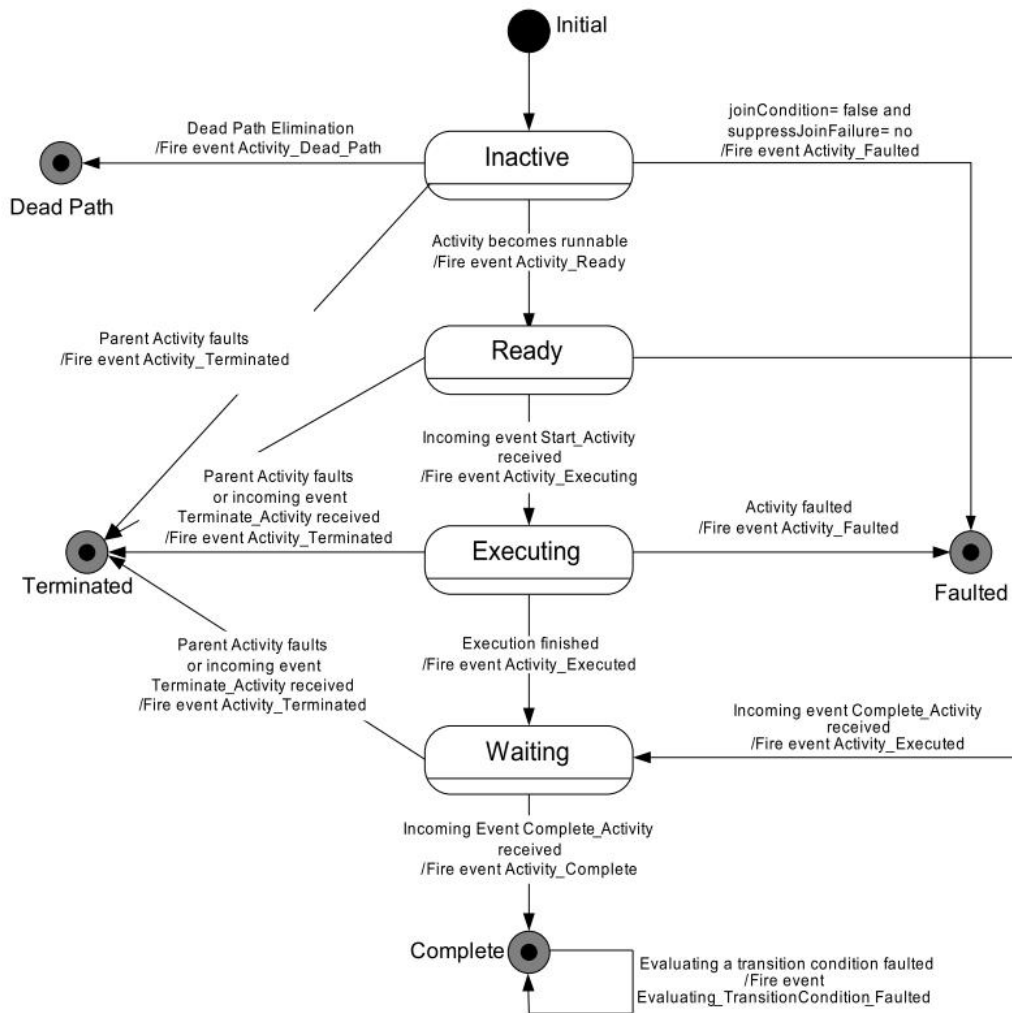


Abbildung 42: BPEL Event Modell für die Ausführung allgemeiner Aktivitäten in WS-BPEL 2.0

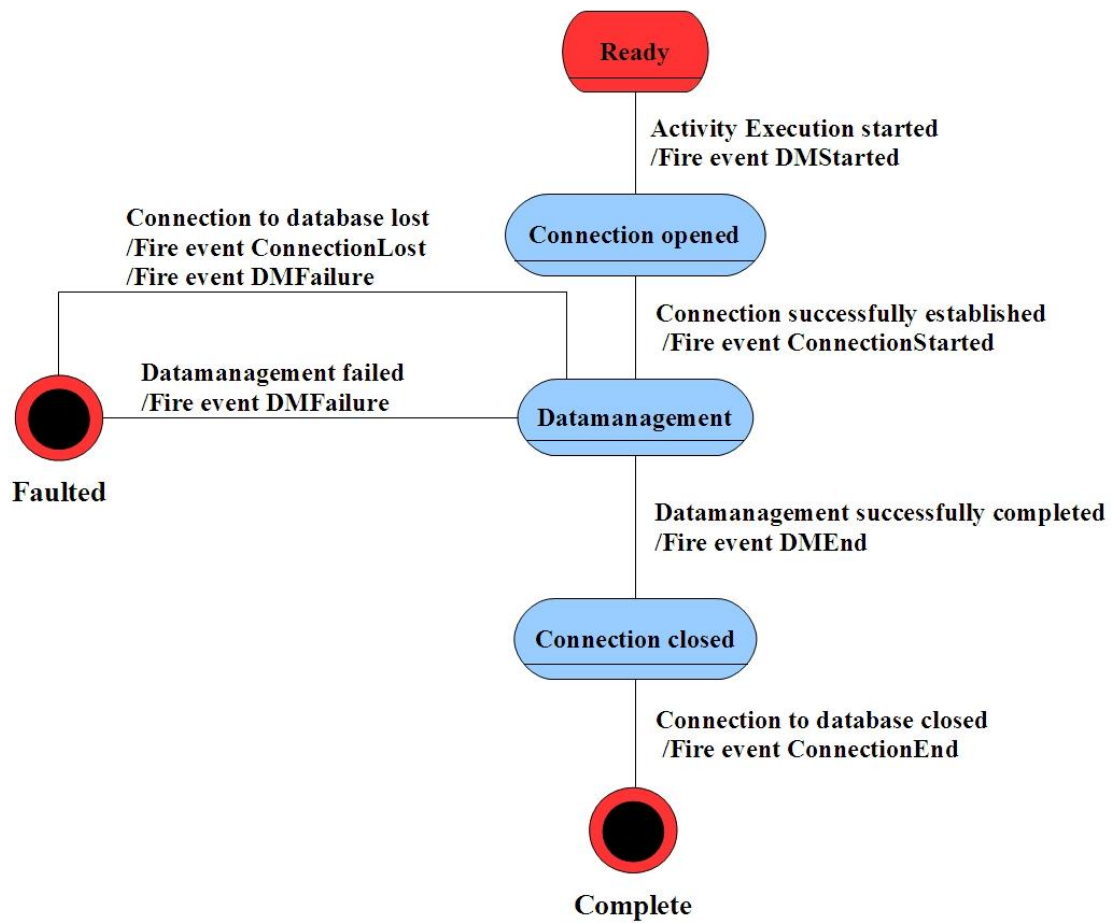


Abbildung 43: Event-Modell für die Ausführung von DM-Aktivitäten

Tabelle 5: Allgemeine Policies

Eigenschaft	Beschreibung	Einheit
responTime	Die maximale Zeit die ein Server zum Antworten brauchen darf.	ms
availability	Die Verfügbarkeit beschrieben als maximale Downtime des Servers.	hours/year
throughput	Maximaler Datendurchsatz	megabyte/sekunde
transaction	Werden Transaktionen von der Datenquelle unterstützt?	boolean

Tabelle 6: Datenbank Policies

Eigenschaft	Beschreibung	Einheit
maxQueryLength	Die maximale Länge eines Befehls der Anfragesprache	Zeichen
maxConcurrentTransaction	Die maximale Anzahl an Verbindungen, die gleichzeitig (konkurrierend) auf Daten der Datenbank zugreifen	Anzahl
maxColumSize	Maximale Spaltenzahl innerhalb der Tabelle	Anzahl
dbVersion	Die Version der Datenbank	Versionsnummer

erzeugt. Der Grund dafür diese Events zusätzlich zu erzeugen liegt darin, das wie bereits erwähnt die Möglichkeit bestehen soll, im Rahmen des Auditing nur die SIMPL-Events zu erzeugen.

- **DMStarted**
Dieses Event wird erzeugt, wenn die Ausführung einer DM-Aktivität gestartet wurde.
- **DMEnd**
Dieses Event wird erzeugt, wenn die Ausführung einer DM-Aktivität erfolgreich beendet wurde.
- **DMFailure**
Dieses Event wird erzeugt, wenn bei der Ausführung einer DM-Aktivität ein Fehler auftrat.

7.6 Datenquellen Policies

In diesem Abschnitt werden die verschiedenen Policies beschrieben, mit denen die nichtfunktionalen Anforderungen der Datenquellen modelliert werden können.

7.6.1 Policy-Beschreibungen

Die Beschreibungen der Policies gliedert sich in die Bereiche:

- allgemeine Policies, die für alle Datenquellen gelten,
- Datenbank Policies, die speziell für Datenbanken gelten und
- Filesystem Policies, die für Filesysteme gelten.

In den Policy-Dateien wird nicht unterschieden, welche Policies zu welchen Datenquellen gehören. Die Trennung erfolgt nur zur besseren Lesbarkeit.

Tabelle 7: Dateisystem Policies

Eigenschaft	Beschreibung	Einheit
maxFilenameLength	Die maximale Länge eines Dateinamens	Länge
maxPathLength	Die maximale Länge eines Dateipfades	Länge
maxFileSize	Die maximale Größe einer Datei	Megabyte

7.6.2 Policy Defenition

Folgende SIMPL Policy-Elemente stehen zur Verfügung und werden nachfolgend SIMPLPolicy-Elements genannt.

```

<responsTime value = "xsd:integer"/>
<availability value = "xsd:integer"/>
<throughput value = "xsd:integer"/>
<transaction value = "xsd:boolean"/>
<maxQueryLength value = "xsd:integer"/>
<maxConcurrentTransaction value = "xsd:integer"/>
<maxColumnSize value = "xsd:integer"/>
<dbVersion value = "xsd:String"/>
<maxFilenameLength value = "xsd:integer"/>
<maxPathLength value = "xsd:integer"/>
<maxFileSize value = "xsd:integer"/>

```

Listing 21: SIMPL Policy-Elemente

Definition:

Folgend ist die Struktur einer SIMPL Policy-Datei aufgezeigt

```
<wsp:Policy
  xmlns:simplp="http://www.example.org/simpl/policy"
  xmlns:wsp="http://www.w3.org/ns/ws-policy">

  SIMPLPolicy-Elements
</wsp:Policy>
```

Listing 22: Listing einer SIMPL Policy-Datei

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.

jUDDI[26]

jUDDI ist eine UDDI Registry und wird dazu benutzt, Datenquellen und Datenquelleninformationen zu speichern. Darüber hinaus ist es möglich, nichtfunktionale Anforderungen als WS-Policy zu definieren

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.

L^AT_EX [18]

L^AT_EX ist ein Textverarbeitungsprogramm, mit dem es möglich ist, auf einfache Art und Weise L^AT_EX-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. - 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] L^AT_EX, <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, T.: *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
- [25] Gudgin, M.; Hadley, M.; Rogers, T.: *Web Service Addressing 1.0 - Core*, W3C Recommendation. World Wide Web Consortium (W3C), 09.Mai 2006. <http://www.w3.org/TR/ws-addr-core/>, zuletzt zugegriffen am 06.02.2010
- [26] jUDDI, <http://ws.apache.org/juddi/>, zuletzt zugegriffen am 28.02.09

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

1	Übersicht über die Systemumgebung von SIMPL	10
2	Um Referenzvariablen erweiterter Eclipse BPEL Designer	14
3	Erweiterter Deployment-Deskriptor	16
4	Dialog für das Hinzufügen einer Datenquellen-Beschreibung	17
5	Dialog für das Hinzufügen eines Activity Mappings	17
6	SIMPL Menü und Eclipse BPEL Designer mit einigen DM-Aktivitäten	18
7	Dialog für die Einstellungen des Auditings	19
8	SIMPL Preference Seiten	20
9	SIMPL About Fenster	20
10	Eigenschaftsfenster einer DM-Aktivität am Beispiel einer Query Activity	21
11	Reference Resolution System View	22
12	Dialog für das Hinzufügen einer Referenz	22
13	Datenquellen-Registry View	23
14	Datenquellen-Registry Webinterface	24
15	Anwendungsfall-Diagramm des gesamten Softwaresystems	26
16	Anwendungsfall-Diagramm für den Prozess-Modellierer	27
17	Anwendungsfall-Diagramm für den Workflow-Administrator	32
18	Anwendungsfall-Diagramm für die ODE Workflow-Engine	37
19	Anwendungsfall-Diagramm für den Eclipse BPEL Designer	40
20	Anwendungsfall-Diagramm für RRS	43
21	Anwendungsfall-Diagramm für die Datenquellen-Administratoren	45
22	Die Architektur des Reference Resolution Systems (RRS).	50
23	Alternativen zur Realisierung von Referenzen in BPEL, vgl. [3]	52
24	Beispiel Prozessmodell mit Referenzvariablen	56
25	Variablen eines Prozessmodells vor der Transformation	57
26	Workspace mit einem BPEL Projekt und dessen transformierter Kopie	57
27	Beispiel für das Anstoßen der Dereferenzierung einer Referenzvariablen über eine Assign-Aktivität	59
28	Transformiertes Beispiel Prozessmodell	60
29	Variablen eines Prozessmodells nach der Transformation	61
30	Klassenhierarchie der Datenmanagement-Patterns	62
31	Übersicht über die Query Patterns-Klasse und deren Patterns	63
32	Übersicht über die Data IUD Patterns-Klasse und deren Patterns	64
33	Übersicht über die Data Setup Patterns-Klasse und deren Patterns	65
34	Übersicht über die Stored Procedure Patterns-Klasse und deren Patterns	66
35	Übersicht über die Data Transport Patterns-Klasse und deren Unterklassen	67
36	Übersicht über die Process Cache WriteBack Patterns-Klasse und deren Patterns	68
37	Übersicht über die Data Retrieval to Process Cache Patterns-Klasse und deren Patterns	68
38	Übersicht über die Similar Data Transport Patterns-Klasse und deren Patterns	69
39	Übersicht über die Diverse Data Transport Patterns-Klasse und deren Patterns	70
40	Übersicht über die Cache Access Patterns-Klasse und deren Unterklassen	72
41	Übersicht über die Cache IUD Patterns-Klasse und deren Patterns	73
42	BPEL Event Modell für die Ausführung allgemeiner Aktivitäten in WS-BPEL 2.0	92
43	Event-Modell für die Ausführung von DM-Aktivitäten	93

Verzeichnis der Listings

1	Schema der ReferenceVariables- und ReferenceVariable-Elemente	15
2	Schema des erweiterten Deployment-Deskriptors	16
3	Schema der im Deployment-Deskriptor hinterlegten Datenquelleninformationen	17
4	Schema der im Deployment-Deskriptor hinterlegten Activity Mappings	17
5	EPR Schema [3]	51
6	Code eines <code><referenceVariable></code> -Schemas [3]	54
7	Code der generierten Variablen-Deklaration, vgl. [3]	54
8	Code der RRS Partner Links	54
9	Code einer Dereferenzierungs-Aktivität, vgl. [3]	55
10	Schema des ContainerReferenceType-Datentyps	59
11	Auszug aus einem Prozessmodell mit Container-Referenzen	61
12	Listing der simpl-attributes	84
13	Schema einer Query Aktivität	85
14	Schema einer Insert Aktivität	85
15	Schema einer Update Aktivität	85
16	Schema einer Delete Aktivität	86
17	Schema einer Create Aktivität	86
18	Schema einer Drop Aktivität	87
19	Schema einer Call Aktivität	87
20	Schema einer RetrieveData Aktivität	88
21	SIMPL Policy-Elemente	95
22	Listing einer SIMPL Policy-Datei	96

Tabellenverzeichnis

1	Komponenten und ihre Anwendungsfälle	47
2	Übersicht der Datenmanagement-Patterns und der von diesen unterstützten Datenquellentypen	75
3	Übersicht über die Umsetzung der Datenmanagement-Patterns für die verschiedenen Datenquellentypen	82
4	Übersicht der resultierenden Aktivitäten und ihrer Anwendbarkeit auf verschiedene Datenquellen	89
5	Allgemeine Policies	94
6	Datenbank Policies	94
7	Dateisystem Policies	95