Studienprojekt SIMPL

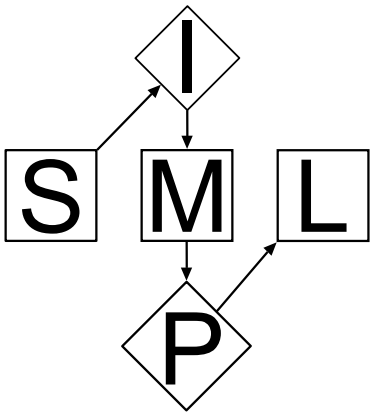
Spezifikation

Version 0.1

11. September 2009

Verfasser:

Wolfgang Huettig, Michael Hahn,Firas Zoabi, Michael Schneidt,Daniel Brüderle, René Rehn



Dok-Status: neu

QS-Status: nicht QS-geprüft

Prüf-Status: nicht geprüft

Review-Status: kein Review durchgeführt

End-Status: -

Inhaltsverzeichnis

1 Einleitung

In diesem Abschnitt 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 Richtschnur 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, 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 beruhen. Über das Rahmenwerk sollen beliebige Datenmanagement-Funktionen in einen BPEL-Prozess eingebunden werden können. Dafür werden bereits vorhandene Konzepte evaluiert und falls nötig erweitert oder angepasst. Es wird untersucht, inwiefern die Sprache BPEL ebenfalls erweitert werden muss. Für eine möglichst hohe Flexibilität soll ein dynamischer Ansatz gewählt werden, so dass erst während der Laufzeit des Systems die Datenquellen festgelegt werden können. Nichtsdestotrotz sollte auch die Möglichkeit bestehen, die Datenquellen statisch anbinden zu können. Eine Anforderung des Kunden ist, dass eine vorhandene BPEL-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-Engine ausgeführt werden können.

1.3 Definitionen

Die in der Spezifikation verwendeten Begriffe, Definitionen und Abkürzungen werden in einem separaten Begriffslexikon definiert und eindeutig erklärt. Durch das Begriffslexikon werden somit alle technischen und fachlichen Begriffe, Definitionen und Abkürzungen, die zu Missverständnissen innerhalb des Projektteams oder zwischen Projektteam und Kunde führen könnten, eindeutig definiert.

1.4 Überblick

In diesem Dokument werden alle Anforderungen des Kunden festgehalten. Nach einem kurzen Überblick über das Projekt selbst, folgen sowohl nichtfunktionale als auch funktionale Anforderungen an das zu entwickelnde Produkt. Die Beschreibung der funktionalen Anforderung, also des Funktionsumfangs, geschieht durch Use-Cases, sowie durch die Beschreibung der späteren Benutzeroberfläche. Ebenso wichtig ist die Beschreibung der nichtfunktionalen Anforderungen, die eventuelle Abhängigkeiten, Einschränkungen, Größenordnungen und Bedingungen im Bezug auf Portabilität, Robustheit, Sicherheit und weitere Aspekte beschreiben.

2 Allgemeine Beschreibung

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

2.1 Einbettung

Das SIMPL Rahmenwerk soll in die, in Abbildung 1 dargestellte, Systemumgebung eingebettet werden. Die Systemumgebung besteht dabei aus den unterschiedlichen Datenquellen, Eclipse mit dem BPEL-Designer Plug-In und einem Web-Server, wie z.B. dem Apache Tomcat, in dem das Rahmenwerk und eine Workflow-Engine (z.B. Apache ODE) ausgeführt werden. Dabei läuft die benötigte Software auf dem lokalen Rechner des Benutzers, die Datenquellen können auf verschiedene Server verteilt sein.

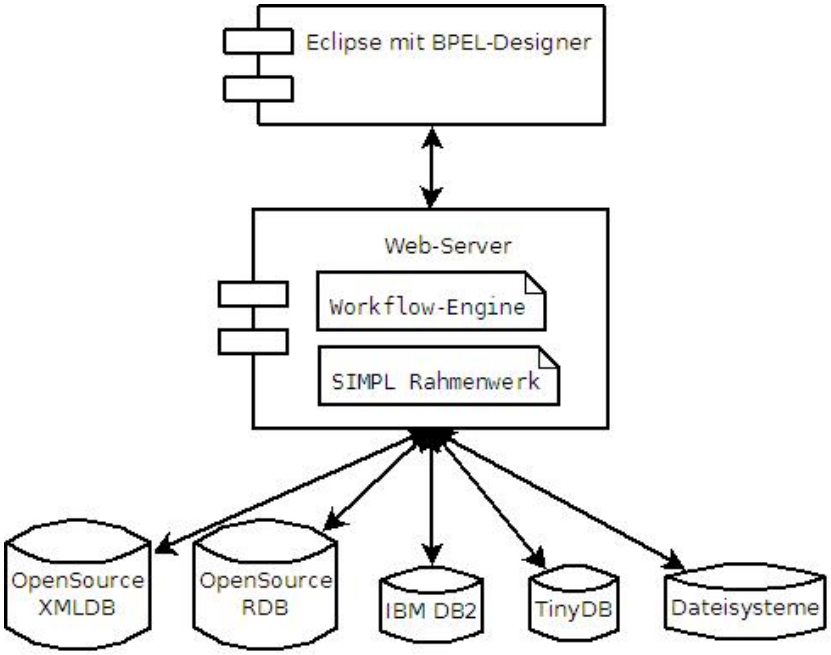


Abbildung 1: Übersicht über die Systemumgebung von SIMPL

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 als Laufzeitumgebung integriert sein. Ebenso soll die Verarbeitung von großen, heterogenen Datenmengen im Rahmen eines wissenschaftlichen Workflows möglich sein.
Um die Fehlertoleranz des Rahmenwerks zu realisieren, sollen Fehler der neu definierten Aktivitäten erkannt und durch entsprechende FaultHandler abgefangen werden können. Abgefangene Fehler werden dann durch entsprechende CompensationHandler kompensiert oder die vorher ausgeführten Aktionen rückgängig gemacht.

BPEL

Über den Eclipse BPEL Designer sollen innerhalb von BPEL-Prozessen Referenzen unterstützt werden, d.h. es soll möglich sein Referenzen (z.B. auf eine Tabelle) innerhalb eines SQL-Befehls über eine BPEL Variable anzugeben oder auch die Ergebnisse eines Queries per Referenz im Prozess bereit zu halten. Dazu ist es erforderlich, dass ein Reference Resolution System mit verschiedenen Adaptern implementiert wird. Dieses System dient dazu Referenzen/Pointer innerhalb eines BPEL Prozesses aufzulösen und so z.B. auch Daten in den Prozess-Cache zu laden. Bei Daten, die nicht für die Prozesslogik relevant sind, soll dabei die Referenz/der Pointer vom Service, zur Entlastung der Workflow-Engine, aufgelöst werden. Mithilfe eines Transformers übersetzt der BPEL Designer den BPEL-Code mit Referenzen in Standard BPEL-Code, der dann auf der Workflow-Engine ausgeführt werden kann.

BPEL Aktivitäten

Alle BPEL-Aktivitäten, die für den Umgang mit Datenquellen benötigt werden, müssen erstellt werden und sollen erweiterbar sein. Als Anhaltspunkt welche Aktivitäten benötigt werden, gelten die folgenden Datenmanagement Patterns (siehe Abschnitt 6.2 bzw. Quelle [2]: Das *Query Pattern* beschreibt die Notwendigkeit mithilfe von SQL-Befehlen externe Daten anfordern zu können. Die aus den Queries resultierenden Daten können dabei auf der Datenquelle extern gespeichert oder direkt im Prozessspeicher gehalten werden. Das *Set UID Pattern* beschreibt die Möglichkeit mengenorientiertes Einfügen (insert), Aktualisieren (update) und Löschen (delete) auf externen Daten durchführen zu können. Das *Data Setup Pattern* liefert die Möglichkeit benötigte Data Definition Language (DDL) Befehle auf einem relationalen Datenbanksystem auszuführen, um so während der Prozessausführung die Datenquelle zu konfigurieren oder neue Container (Tabellen, Schema, usw.) zu erstellen. Da die Verarbeitung von komplexen Daten meist durch Stored Procedures erfolgt, ist es bei der Verarbeitung von externen Daten unbedingt erforderlich, Stored Procedures auch aus einem Prozess heraus aufrufen zu können (*Stored Procedure Pattern*). Manchmal ist es nötig Daten innerhalb des Prozessspeichers verarbeiten zu können. Das *Set Retrieval Pattern* liefert dafür eine mengenorientierte Datenstruktur, in die man die angefragten externen Daten innerhalb des Prozessspeichers abgelegen kann. Diese Datenstruktur verhält sich dabei wie ein Cache im Prozessspeicher, der keine Verbindung zur originalen Datenquelle besitzt. Das *Set Access Pattern* beschreibt die Notwendigkeit auf den erzeugten Datencache sequentiell und direkt (random) zugreifen zu können. Das *Tuple IUD Pattern* beinhaltet einfügen (insert), aktualisieren (update) und löschen (delete) von Daten im Datencache. Das *Synchronization Pattern* realisiert die Synchronisation eines lokalen Datencaches mit der originalen Datenquelle.

Anbindung von Datenquellen

Die Anbindung von Datenquellen soll generisch sein und es sollen so viele Anbindungen wie möglich realisiert werden. Als Minimum gilt dabei die IBM DB2, mindestens zwei OpenSourceDB's (eine RDB und eine XMLDB), mindestens eine SensorDB (z.B. TinyDB) und mindestens ein Dateisystem (z.B. ext3). Es sollen auch Zugriffe auf mehrere Datenquellen innerhalb eines Prozesses möglich sein, dazu besitzt jede Aktivität eine Variable, in der der logische Namen der Datenquelle auf die zugegriffen werden soll, hinterlegt ist. Alle Anfragesprachen für den Datenbankzugriff müssen unterstützt werden, mindestens SQL und XQuery. Zusätzlich sollen auch alle SQL-Dialekte unterstützt werden. Transaktionen sollen innerhalb von DB und BPEL-Prozessen unterstützt werden. Die Konzepte sind nach Absprache mit dem Kunden frei wählbar. Es soll auch möglich sein, dass Daten aus Datenquellen aus einem Prozess auch in lokale Dateien exportiert werden können und umgekehrt soll es auch möglich sein lokale Daten zu importieren. Dabei gilt XML als Standardformat.

Die Datenbanken, die verwendet werden, existieren bereits und müssen nicht durch das Rahmenwerk erstellt werden. Weiterhin sollen aber Schema-Definitionen möglich sein und das Erstellen, Ändern und Löschen von Tabellen innerhalb der Schemas.

Das Late-binding von Datenquellen soll unterstützt werden. Dazu müssen Kriterien zur Beschreibung von Datenquellen erstellt werden. Diese Kriterien werden dann als Anforderungen durch den Be-

nutzer modelliert und als Annotationen übertragen. Die Spezifizierung der Anforderungen, die sinnvoll an eine Datenquelle gestellt werden können sollten, wird über die Definition eines Anforderungskataloges, der nach Fertigstellung mit dem Kunden abgesprochen werden muss, realisiert. Die Annotationen dienen dazu, dass Datenquellen durch definierte Strategien automatisch ausgewählt werden können. Diese Strategien, die zur Auswahl einer Datenquelle anhand der vom Benutzer angegebenen Anforderungen verwendet werden, müssen von uns definiert und anschließend mit dem Kunden abgesprochen werden. Die Auswertung der Annotationen und die Verarbeitung der darin enthaltenen Informationen soll, ebenfalls selbst definiert und anschließend mit dem Kunden abgesprochen werden. Die Adressierung der Datenbanken soll statisch über konkrete Adressen oder auch dynamisch über logische Namen mithilfe der JNDI API möglich sein.

Admin-Konsole

Die Admin-Konsole ist ein Werkzeug mit dem verschiedene Einstellungen des Rahmenwerks auch während der Laufzeit noch geändert werden können. Dazu gehört beispielsweise das An- und Abschalten des Auditings und Einstellen der Granularität. Eine solche Komponente soll für die Administration des Rahmenwerks erstellt werden.

Autorisierung und Authentifizierung

Die Autorisierung und Authentifizierung soll momentan nur für Datenquellen bereitgestellt werden, allerdings soll eine spätere Erweiterung einfach realisiert werden können. Für die Authentifizierung und Autorisierung sind verschiedene Verfahren gewünscht, sodass in jeder Situation das Bestmögliche verwendet wird. Dazu soll es möglich sein, verschiedene konkrete Verfahren flexibel anbinden zu können. Die Autorisierung und die Authentifizierung soll dabei über einen Single-Sign-On in der Instanz beim Zugriff auf Datenquellen durchgeführt werden. Autorisierungs- und Authentifizierungsparameter für einen Prozess sollen auf Datenquellenebene spezifiziert werden, d.h. bei Datenbanken über Schema und bei Dateisystemen z.B. über extra Dateien. Die Autorisierung und Authentifizierung ist bei allen Datenquellen notwendig und soll auf verschiedene Arten möglich sein.Angaben über Autorisierung und Authentifizierung sollen extra im Eclipse BPEL Designer abgefragt und als Nachricht an die Datenquelle geschickt werden.

Auditing und Monitoring

Es soll ein Auditing der Prozessausführung geben, dafür soll die Ausführungshistorie in einer angebundenen Datenbank und nicht im lokalen Speicher der Engine gespeichert werden können. Die Datenbank für das Auditing soll über das Rahmenwerk frei wählbar sein. Die Auditing-Daten werden dabei auf einer Datenbank gespeichert und nicht über mehrere verteilt. Das vorhandene Auditing von Apache ODE muss dafür um das Auditing unserer Funktionalitäten und die Möglichkeit, eine variable DB als Auditing-DB anzugeben, erweitert werden. Die Speicherungsdauer der Auditing-Daten soll variabel als Parameter übergeben werden können. Das Auditing soll standardmäßig aktiv sein. Auf Benutzerwunsch soll die Granularität des Auditings veränderbar sein oder das Auditing auch komplett abschaltbar. Dies soll auch während der Laufzeit über die Admin-Konsole steuerbar sein.

Für unsere Zwecke muss ein komplettes Monitoring erstellt werden, d.h. ein Monitoring für alle Datenquellenaktionen und die Anpassung und Erweiterung des bestehenden Monitorings von Apache ODE. Dazu soll das Event-Modell von Apache ODE für das Monitoring, d.h. welche Informationen angezeigt werden können, für unsere Erweiterungen modifiziert und mit den Kunden abgesprochen werden.

2.3 Sprache

Generell gilt, dass alle Dokumente auf Deutsch und jeder Quellcode einschließlich Kommentaren auf Englisch verfasst und ausgeliefert werden. Eine Ausnahme bilden das Handbuch und die verschiedenen

Dokumentationen der durchzuführenden 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

2.5 Benutzerprofile

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

2.6 Einschränkungen

2.7 Annahmen und Abhängigkeiten

3 Nichtfunktionale Anforderungen

In diesem Abschnitt 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

3.2 Benutzbarkeit

Die Benutzbarkeit soll sich vor allem an Nutzer mit wenig Kenntnissen im Umgang mit Workflows und BPEL richten und dafür die größtmögliche Transparenz liefern, d.h. dass die interne Prozesslogik der Software bestmöglich vom Benutzer abgeschirmt wird und er eine möglichst einfache und schnell verständliche Schnittstelle zur Software erhält, um die Verwendung von SIMPL für alle Benutzergruppen zu ermöglichen.

3.3 Robustheit

3.4 Sicherheit

3.5 Portabilität

3.6 Erweiterbarkeit

Die Erweiterbarkeit des Systems spielt 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, wird ein modularer Aufbau zugrunde gelegt und entsprechende Schnittstellen geschaffen.

3.7 Wartbarkeit

3.8 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.

3.9 ...

4 Akteure

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

4.1 Prozess-Modellierer

Der P

4.2 Workflow-Administrator

4.3 Datenquellen-Administrator

4.4 ODE Workflow-Engine

4.5 Eclipse BPEL Designer

4.6 SIMPL Core

5 Anwendungsfälle (Use-Cases)

Dieser Abschnitt beschreibt die funktionalen Anforderungen an die Software. Dazu werden alle Anwendungsfälle eines jeden Akteurs beschrieben und deren Zusammenhänge in entsprechenden Diagrammen graphisch dargestellt.

5.1 Diagramm aller Anwendungsfälle

Abbildung 2 zeigt das Diagramm aller Anwendungsfälle der gesamten Software. Dadurch wird die Funktionalität und die Akteure des späteren Gesamtsystems sichtbar.

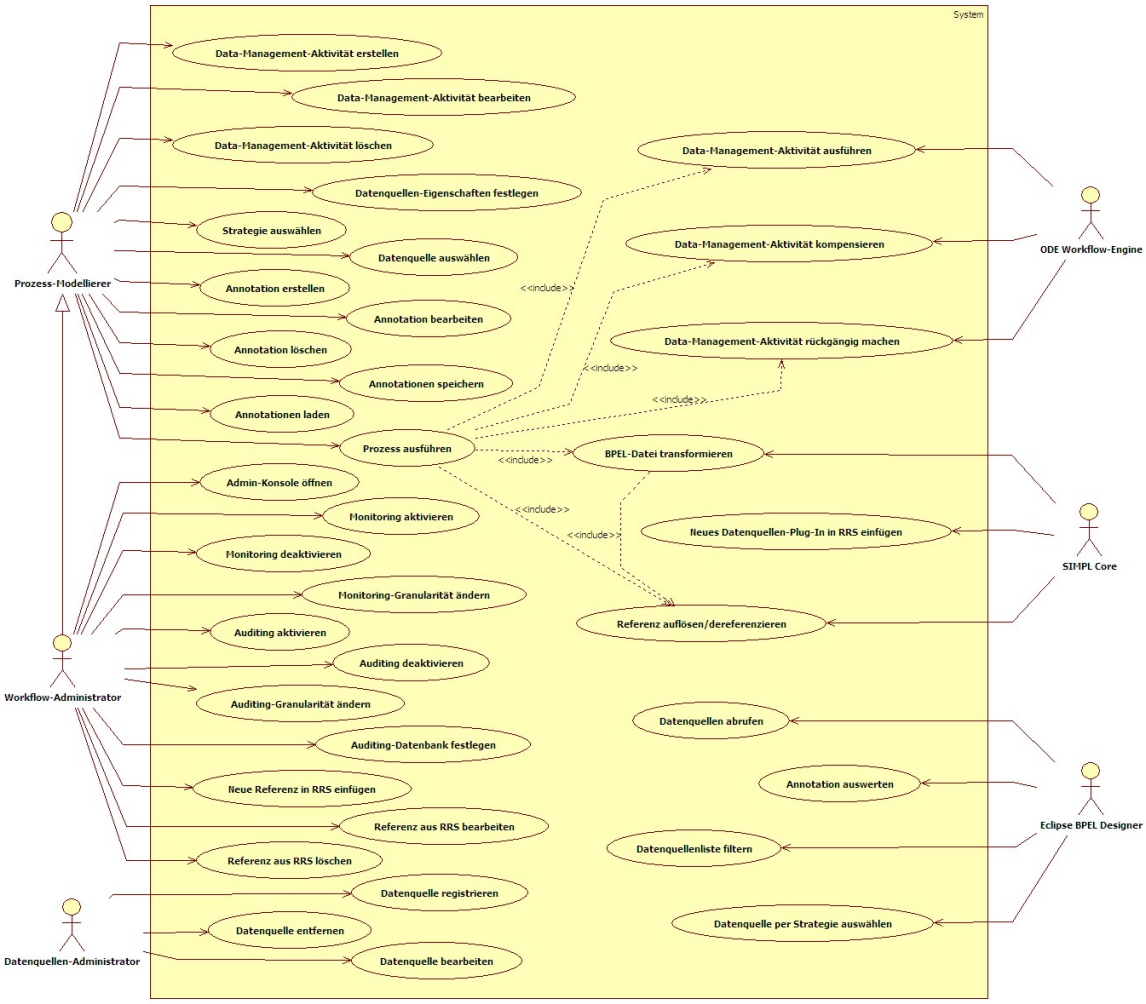


Abbildung 2: Anwendungsfall-Diagramm des gesamten Softwaresystems

5.2 Anwendungsfälle der Prozess-Modellierer

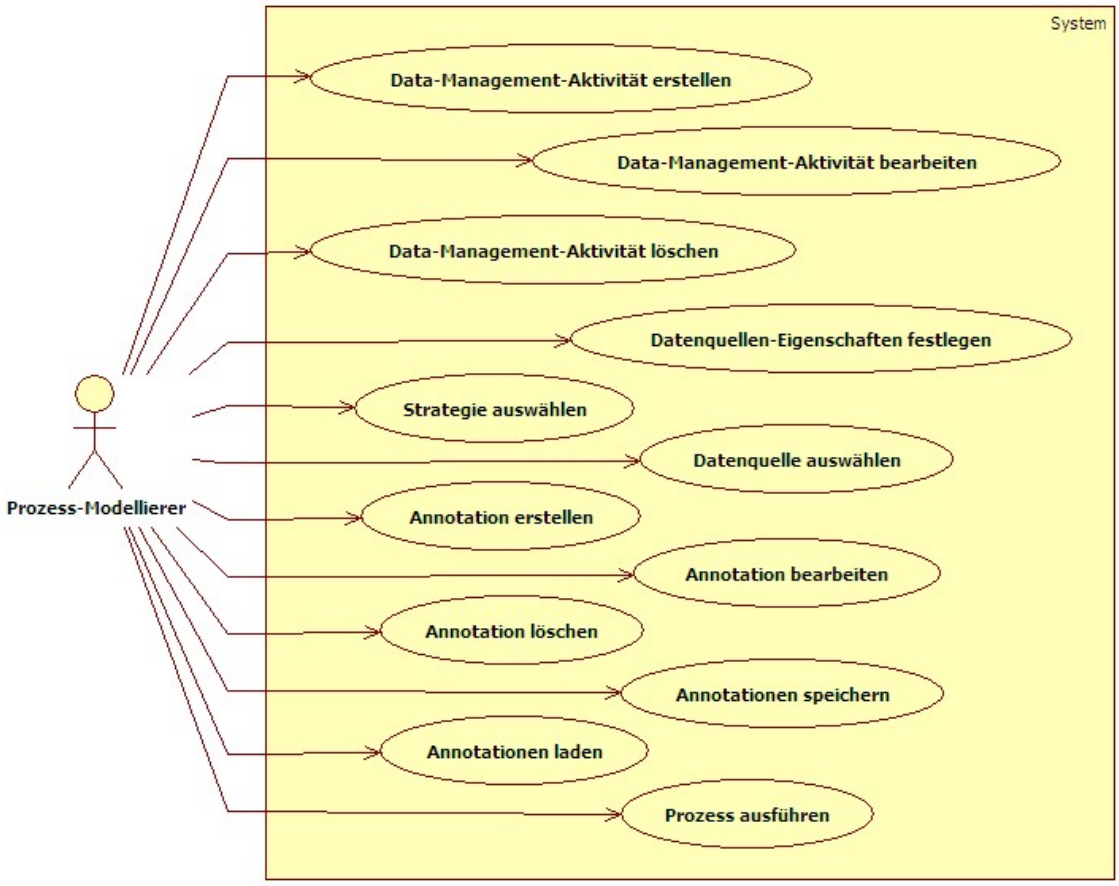


Abbildung 3: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.2.1 Data-Management-Aktivität erstellen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.2 Data-Management-Aktivität bearbeiten

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.3 Data-Management-Aktivität löschen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.4 Strategie auswählen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.5 Datenquellen-Eigenschaften festlegen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.6 Datenquelle auswählen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.7 Annotation erstellen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.8 Annotation bearbeiten

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.9 Annotation löschen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.10 Annotationen speichern

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.11 Annotationen laden

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.2.12 Prozess ausführen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3 Anwendungsfälle der Workflow-Administratoren

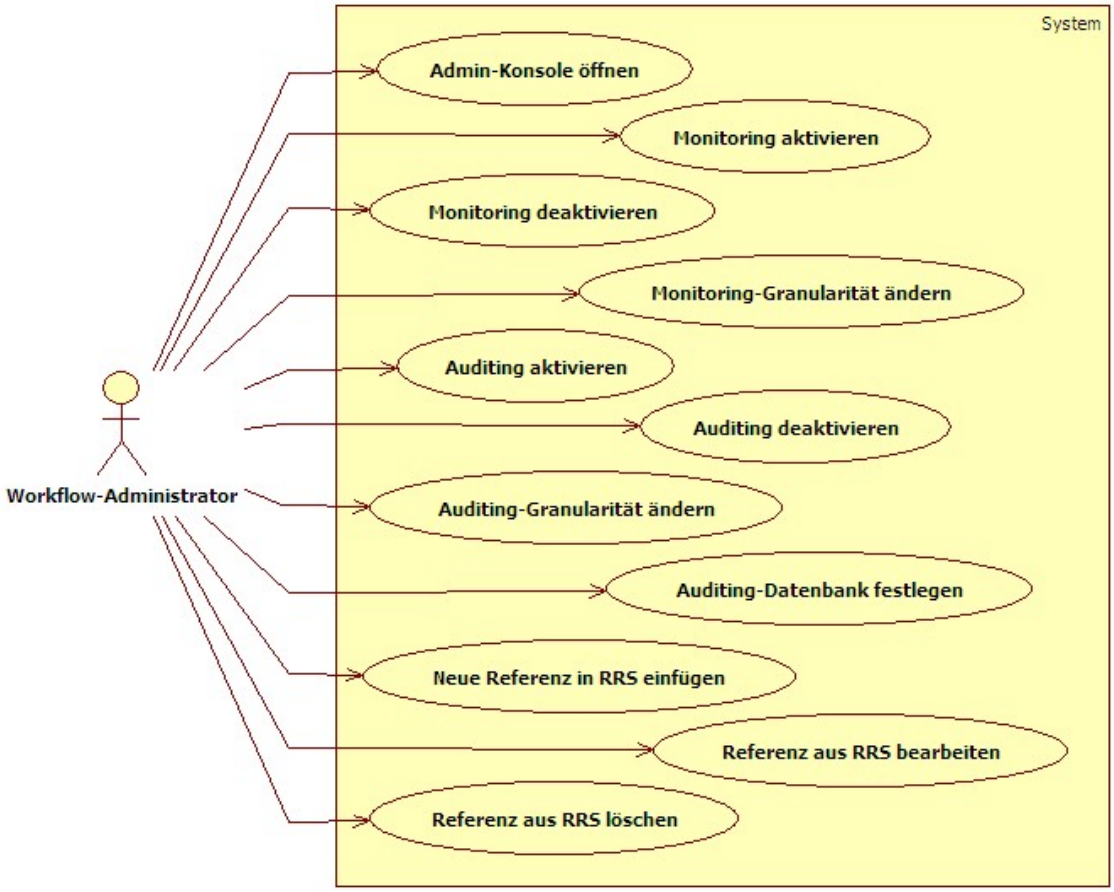


Abbildung 4: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.3.1 Admin-Konsole öffnen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.2 Monitoring aktivieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.3 Monitoring deaktivieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.4 Monitoring-Granularität ändern

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.5 Auditing aktivieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.6 Auditing deaktivieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.7 Auditing-Granularität ändern

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.8 Auditing-Datenbank festlegen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.9 Neue Referenz in RRS einfügen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.10 Referenz aus RRS bearbeiten

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.3.11 Referenz aus RRS löschen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.4 Anwendungsfälle der Datenquellen-Administratoren

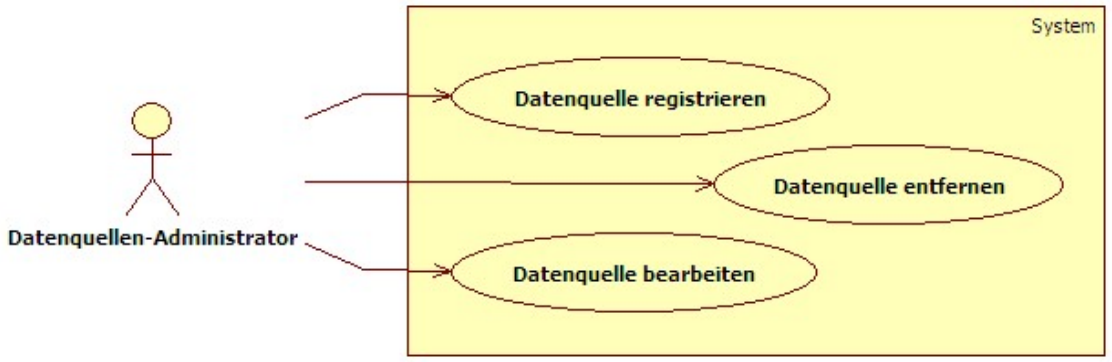


Abbildung 5: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.4.1 Datenquelle registrieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.4.2 Datenquelle entfernen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.4.3 Datenquelle bearbeiten

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.5 Anwendungsfälle der ODE Workflow-Engine

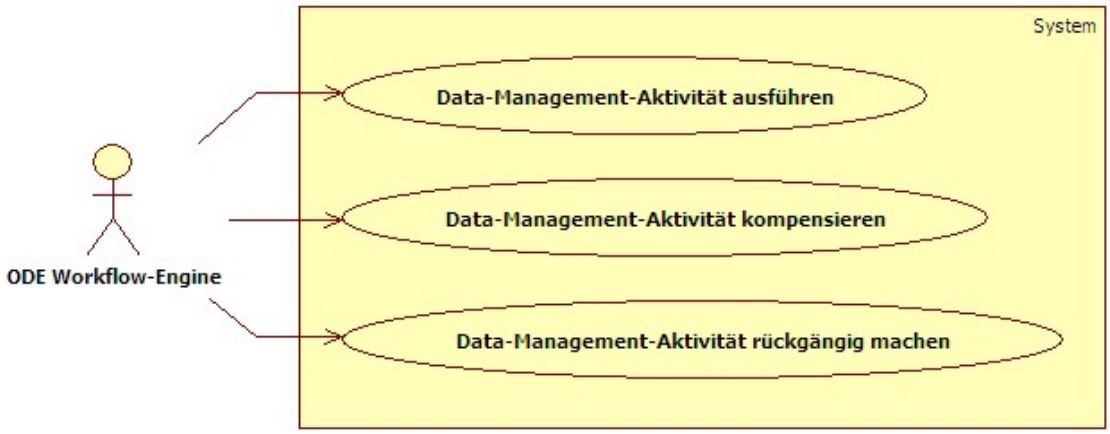


Abbildung 6: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.5.1 Data-Management-Aktivität ausführen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.5.2 Data-Management-Aktivität kompensieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.5.3 Data-Management-Aktivität rückgängig machen (rollback)

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.6 Anwendungsfälle des Eclipse BPEL Designers

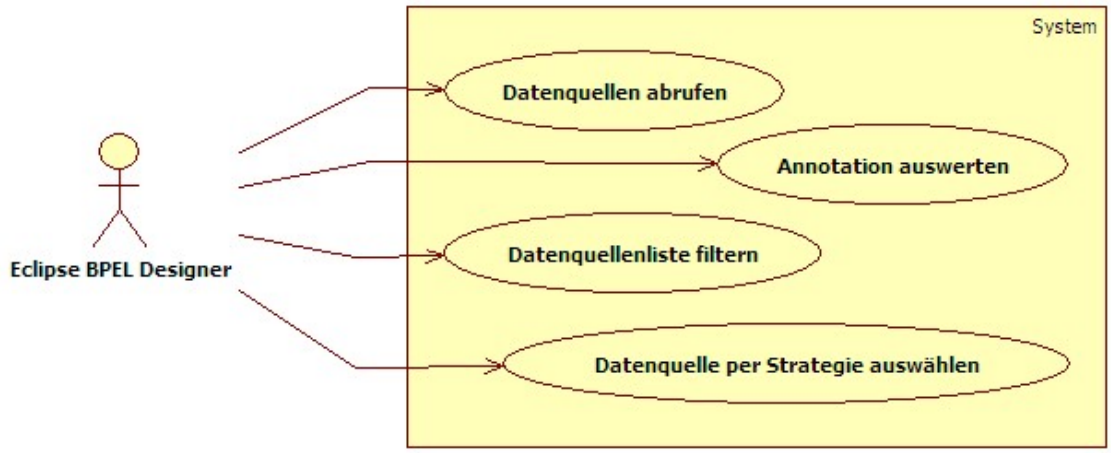


Abbildung 7: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.6.1 Datenquellen abrufen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.6.2 Annotation auswerten

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.6.3 Datenquellenliste filtern

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.6.4 Datenquelle per Strategie auswählen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.7 Anwendungsfälle des SIMPL Core

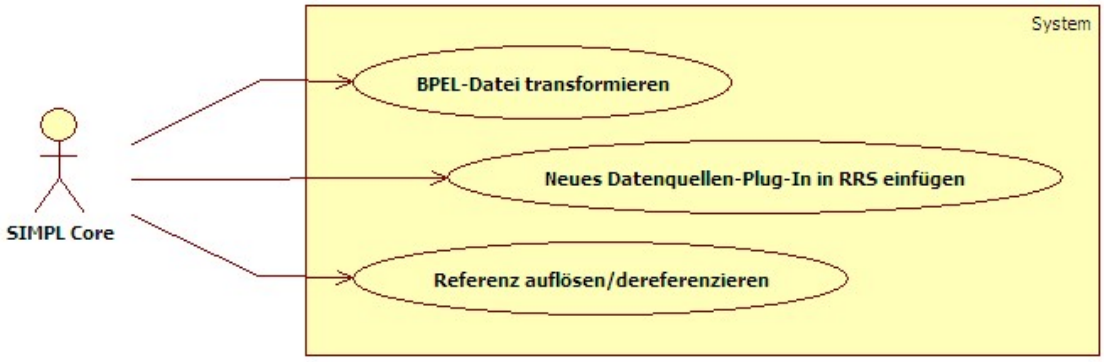


Abbildung 8: Anwendungsfall-Diagramm für die Prozess-Modelierer

5.7.1 BPEL-Datei transformieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.7.2 Neues Datenquellen-Plug-In in RRS einfügen

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

5.7.3 Referenz auflösen/dereferenzieren

Ziel	
Vorbedingung	
Nachbedingung	
Nachbedingung im Sonderfall	
Normalablauf	
Sonderfälle	

6 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 Reference Resolution Systems, dass es ermöglicht mit Referenzen innerhalb von Workflows zu arbeiten oder die Identifizierung und Definition von benötigten Datenmanagement Aktivitäten zur Realisierung einer SQL-inline Unterstützung für BPEL. Ebenso werden die benötigten Eclipse BPEL Designer Erweiterungen, sowie die Realisierung des Datenquellen-Monitorings und das dafür zugrunde liegende Event-Modell beschrieben.

6.1 Reference Resolution System

6.2 BPEL-SQL

In diesem Abschnitt werden die zu realisierenden Datenmanagement-Patterns zur Realisierung einer SQL-inline Unterstützung vorgestellt und im weiteren Verlauf deren Umsetzung für die verschiedenen Datenquellen, d.h. für Dateisysteme, Datenbanken und Sensornetze, erläutert. Aus den in Abschnitt 6.2.2 identifizierten Methoden und Funktionen werden dann in Abschnitt 6.2.3 neue benötigte BPEL-Aktivitäten abstrahiert, die dann später die entpsrechenden Methoden und Funktionen realisieren werden.

6.2.1 Datenmanagement Patterns

In diesem Kapitel werden die Datenmanagement Patterns, die zur Realisierung einer SQL-inline Unterstützung benötigt werden, aufgeführt und beschrieben (siehe Quelle [2]).

Query Pattern

Das Query Pattern beschreibt die Notwendigkeit mithilfe von SQL-Befehlen externe Daten anfordern zu können. Die aus den Queries resultierenden Daten können dabei auf der Datenquelle extern gespeichert oder direkt im Prozessspeicher gehalten werden.

Set IUD Pattern

Das Set IUD Pattern beschreibt die Möglichkeit mengenorientiertes Einfügen (insert), Aktualisieren (update) und Löschen (delete) auf externen Daten durchführen zu können.

Data Setup Pattern

Das Data Setup Pattern liefert die Möglichkeit benötigte Data Definition Language (DDL) Befehle auf einem relationalen Datenbanksystem auszuführen, um so während der Prozessausführung die Datenquelle zu konfigurieren oder neue Container (Tabellen, Schema, usw.) zu erstellen.

Stored Procedure Pattern

Da die Verarbeitung von komplexen Daten meist durch Stored Procedures erfolgt, ist es bei der Verarbeitung von externen Daten unbedingt erforderlich, Stored Procedures auch aus einem Prozess heraus aufrufen zu können.

Set Retrieval Pattern

Manchmal ist es nötig Daten innerhalb des Prozessspeichers verarbeiten zu können. Das Set Retrieval Pattern liefert dafür eine mengenorientierte Datenstruktur, in die man die angefragten externen Daten innerhalb des Prozessspeichers ablegen kann. Diese Datenstruktur verhält sich dabei wie ein Cache im Prozessspeicher, der keine Verbindung zur originalen Datenquelle besitzt.

Set Access Pattern

Das Set Access Pattern beschreibt die Notwendigkeit auf den erzeugten Datencache sequentiell und direkt (random) zugreifen zu können.

Tuple IUD Pattern

Das Tuple IUD Pattern beinhaltet einfügen (insert), aktualisieren (update) und löschen (delete) von Daten im Datencache.

Synchronization Pattern

Das Synchronization Pattern realisiert die Synchronisation eines lokalen Datencaches mit der originalen Datenquelle.

6.2.2 Umsetzung der Datenmanagement Patterns

In diesem Kapitel wird die Realisierung der acht Datenmanagement Patterns aus Abschnitt 6.2.1 im Zusammenhang mit den jeweiligen Datenquellentypen beschrieben. Es wird versucht eine einheitliche Realisierung zu erreichen, um die Usability möglichst hoch zu halten. Dafür sollen alle Befehle, die der Benutzer angeben kann, eine SQL bzw. XQuery konforme Befehlsform haben. D.h. sollte es im Moment nicht möglich sein einen benötigten Befehl in SQL oder XQuery anzugeben, so werden im Rahmen unserer Erweiterungen neue Befehlsformen erstellt, die dann auch Zugriffe auf Dateien oder Ähnliches realisieren und trotzdem nicht stark von der SQL- bzw. XQuery-Syntax abweichen.

Dateisysteme

Hier wird die Umsetzung der in Abschnitt 6.2.1 beschriebenen Patterns im Bereich der Dateisysteme durch SQL oder XQuery ähnliche Befehle, die intern durch bestehende Java-Methoden oder falls erforderlich durch die Definition neuer zu implementierender Funktionen realisiert werden, beschrieben.

- Query Pattern: Realisierung durch eine Select-Methode, die intern java.io Methoden verwendet, um Inhalte aus Dateien oder komplette Dateien aus einem Dateisystem zu lesen. Dafür muss noch ein Konzept erarbeitet werden, wie entsprechende Daten gezielt über ein SQL-Select ähnlichen Befehl oder XQuery (bei XML-Dateien oder eventuell auch Anderen) ausgewählt und abgefragt werden können.
  - Beispiel für eine CSV-Datei: SELECT * FROM *DATEINAME* WHERE *SPALTE1* LIKE 'abc', wobei Spalte1 der erste Wert aus der comma-separated-values-Liste ist.
- Set IUD Pattern: Realisierung durch entsprechende INSERT, UPDATE und DELETE-Methoden, die intern java.io Methoden verwenden, um Inhalt in Dateien zu schreiben und aus ihnen zu löschen. Ein Update wird bei Dateien intern durch das Löschen des alten Wertes und anschließendem Einfügen des neuen Wertes ausgeführt. Dafür muss noch ein Konzept erarbeitet werden, wie entsprechende Informationen gezielt über SQL ähnliche Befehle oder XQuery (bei XML-Dateien oder eventuell auch Anderen) eingefügt und gelöscht werden können.
  - Beispiel für eine CSV-Datei:
    - INSERT INTO *DATEINAME* VALUES (hans, meyer, 12345, Musterstadt) [AT END, AT BEGINING, AS *3*, SORTED, ...]
    - UPDATE *DATEINAME* SET *SPALTE3* = 'Chef' WHERE *SPALTE1* = '000290'
    - DELETE FROM *DATEINAME* WHERE *SPALTE1* = *'hans'* AND *SPALTE2* = *'meyer'*
- Data Setup Pattern: Realisierung durch die Verwendung entsprechender CREATE-Methoden, die intern durch java.io-Methoden realisiert sind. Hier sollte es auf jeden Fall möglich sein, Dateien und Ordner in einem Dateisystem erzeugen zu können. Ebenso sollte optional das Löschen auf der gleichen Ebene, d.h. von ganzen Dateien und Ordnern, möglich sein.
  - Beispiele:
    - CREATE FILE 'datei.csv' INTO 'uri (z.B. C:/)'
    - CREATE FOLDER 'test' INTO 'uri (z.B. C:/)'
    - DROP FILE 'datei.csv'
    - DROP FOLDER 'test'
- Stored Procedure Pattern: Findet im Rahmen der Dateisysteme keine Verwendung.
- Set Retrieval Pattern: Realisierung durch die Definition und Bereitstellung einer entsprechenden Activity, die Daten kapselt und im Prozessspeicher halten kann, für BPEL. Eine solche Activity liefert dann die Möglichkeit, dass in ihren Ausprägungen Ergebnisse von Queries (siehe Query Pattern), die über die SDO API abstrahiert wurden, innerhalb des Prozessspeichers abgelegt werden können. Die Firma IBM hat mit ihrer *Business Integration Suite* und dem darin enthaltenen *WebSphere Integration Developer* bereits eine Umsetzung dieses Patterns als Activity mit der Bezeichnung *"Retrieve Set Activity"* realisiert. Dabei werden in einer Retrieve Set Activity externe Daten in eine XML-Struktur innerhalb des Prozessspeichers geladen, eine etwas ausführlichere Beschreibung liefert [2].
- Set Access Pattern: Um das Set Access Pattern zu realisieren, müssen Methoden, die eine sequentielle und direkte Bearbeitung des Datencache einer RetrieveSet Activity realisieren, bereitgestellt werden. D.h. konkret, dass es möglich sein muss, mit der definierten Activity innerhalb von BPEL zu arbeiten. Die SDO API liefert dafür bereits einige Methoden die verwendet werden können. Es muss weiterhin möglich sein, dass eine entsprechende Aktivität auch in Containern, wie z.B. einer ForEach Activity, korrekt ausgeführt wird.

- Tuple IUD Pattern: Erweitert die Set Access Pattern Methoden um die Möglichkeiten Werte innerhalb der mengenorientierten Datenstruktur im Prozessspeicher zu aktualisieren, einzufügen und zu löschen. Die SDO API liefert dafür bereits einige Methoden die verwendet werden können.
- Synchronization Pattern: Um die Daten aus dem Prozesscache zurück auf die originale Datenquelle zu übertragen, muss ebenfalls eine neue BPEL-Aktivität erstellt werden, durch die der Benutzer angibt, dass die Daten zurückgeschrieben werden sollen. Diese Aktivität nutzt dann intern die im Set IUD Pattern und Set Access Pattern beschriebenen Methoden um die Daten aus dem Prozesscache auf die Datenquelle zu übertragen. Die SDO API und die DAS API liefern dafür bereits einige Methoden die verwendet werden können.

Datenbanken

Hier wird die Umsetzung der in Abschnitt 6.2.1 beschriebenen Patterns im Bereich der Datenbanken durch bestehende SQL und XQuery-Befehle oder falls erforderlich durch die Definition neuer zu implementierender Funktionen beschrieben.

- Query Pattern: Realisierung durch SQL-SELECT oder entsprechende XQuery-Befehle.
  - Beispiele:
    - SELECT Info FROM Customer
    - XQUERY db2-fn:xmlcolumn ('CUSTOMER.INFO')
- Set IUD Pattern: Realisierung durch SQL-INSERT, SQL-UPDATE, SQL-DELETE oder entsprechende XQuery-Befehle.
  - Beispiele:
    - INSERT INTO DEPARTMENT VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
    - UPDATE EMPLOYEE SET JOB = 'LABORER' WHERE EMPNO = '000290'
    - DELETE FROM DEPARTMENT WHERE DEPTNO = 'D11'
    - xquery transform copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004') modify do insert <billto country="Canada"> <street>4441 Wagner</street> <city>Aurora</city> <prov-state>Ontario</prov-state> <pcode-zip> N8X 7F8</pcode-zip> </billto> after $mycust/customerinfo/phone[last()] return $mycust
    - UPDATE Customer SET info = XMLQUERY('declare default element namespace "http://posample.org"; transform copy $newinfo := $info modify do delete ($newinfo/customerinfo/phone) return $newinfo' passing info as "info") WHERE cid = 1002
    - xquery transform copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid = 1003') modify do delete $mycust/customerinfo/phone[@type!="home"] return $mycust
- Data Setup Pattern: Realisierung durch SQL-CREATE SCHEMA, SQL-CREATE TABLE, SQL-CREATE VIEW und weitere SQL-CREATE Befehle. Falls optional auch das Löschen dieser Objekte möglich sein soll, wird dies durch SQL-DROP realisiert.
  - 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 DEPARTX

        * CREATE VIEW ADMINISTRATOR.KUND_WITH_ADR AS SELECT KUNDEN.KUNDEN_NR, KUNDEN.VORNAME, KUNDEN.NACHNAME, ADRESSEN.STRASSE, ADRESSEN.POSTLEITZAHL, ADRESSEN.STADT FROM "SYSTEM".KUNDEN AS KUNDEN, "SYSTEM".ADRESSEN AS ADRESSEN WHERE KUNDEN.KUNDEN_NR = ADRESSEN.KUNDEN_NR
        * DROP SCHEMA INTERNAL
        * DROP TABLE TDEPT
        * DROP VIEW ADMINISTRATOR.KUND_WITH_ADR

- Stored Procedure Pattern: Realisierung durch [(optional) SQL-CREATE PROCEDURE und] SQL-CALL (Aufruf über JDBC API möglich).
    - Beispiele:
        * [CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER, OUT COST DECIMAL(7,2), OUT QUANTITY INTEGER) EXTERNAL NAME 'parts!onhand' LANGUAGE JAVA PARAMETER STYLE DB2GENERAL;]
        * CALL PARTS_ON_HAND (?,?,?) *(? = Parameter der Prozedur)*

- Set Retrieval Pattern: Realisierung durch die Definition und Bereitstellung eines mengenorientierten Datentyps oder einer entsprechenden Activity, die solch einen Datentyp kapselt, für BPEL. Ein solcher Datentyp/Activity liefert dann die Möglichkeit, dass in Ausprägungen dieses Datentyps/Activity, Ergebnisse von Queries (siehe Query Pattern), die über die SDO API abstrahiert wurden, innerhalb des Prozessspeichers abgelegt werden können. Die Firma IBM hat mit ihrer *Business Integration Suite* und dem darin enthaltenen *WebSphere Integration Developer* bereits eine Umsetzung dieses Patterns als Activity mit der Bezeichnung *"Retrieve Set Activity"* realisiert. Dabei werden in einer Retrieve Set Activity externe Daten in eine XML-Struktur innerhalb des Prozessspeichers geladen, eine etwas ausführlichere Beschreibung liefert [2].

- Set Access Pattern: Um das Set Access Pattern zu realisieren, müssen Methoden, die eine sequentielle und direkte Bearbeitung der mengenorientierten Datenstruktur realisieren, bereitgestellt werden. D.h. konkret, dass es möglich sein muss, mit dem definierten mengenorientierten Datentyp innerhalb von BPEL zu arbeiten. Die SDO API liefert dafür bereits einige Methoden die verwendet werden können. Bei der Kapselung als Aktivität muss es auch möglich sein, dass die entsprechende Aktivität auch in Containern, wie z.B. einer ForEach Activity, entsprechend korrekt ausgeführt wird.

- Tuple IUD Pattern: Erweitert die Set Access Pattern Methoden um die Möglichkeiten Werte innerhalb der mengenorientierten Datenstruktur im Prozessspeicher zu aktualisieren, einzufügen und zu löschen. Die SDO API liefert dafür bereits einige Methoden die verwendet werden können.

- Synchronization Pattern: Um die Daten aus dem Prozesscache zurück auf die originale Datenquelle zu übertragen, muss ebenfalls eine neue BPEL-Aktivität erstellt werden, durch die der Benutzer angibt, dass die Daten zurückgeschrieben werden sollen. Diese Aktivität nutzt dann intern die im Set IUD Pattern und Set Access Pattern beschriebenen Methoden und ebenso SQL-Befehle um die Daten aus dem Prozesscache auf die Datenquelle zu übertragen. Die SDO API und die DAS API liefern dafür bereits einige Methoden die verwendet werden können.

Sensornetze

Hier wird die Umsetzung der in Abschnitt 6.2.1 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-Befehls Beispiele beziehen sich hier auf den SQL-Dialekt der Sensornetz-Datenbank TinyDB.

- Query Pattern: Realisierung durch ein entsprechendes SQL-SELECT der Sensornetz-Datenbank.
  - Befehlsstruktur: SELECT select-list [FROM sensors] WHERE where-clause [GROUP BY gb-list [HAVING having-list]][TRIGGER ACTION command-name[(param)]] [EPOCH DURATION integer]
  - Beispiele:
    - SELECT temp FROM sensors WHERE temp > thresh TRIGGER ACTION SetSnd(512) EPOCH DURATION 512
    - SELECT field1, field2 SAMPLE PERIOD 100 FROM name (SELECT auf Buffer-Tabelle *name*)
- Set IUD Pattern: Dieses Pattern wird von Sensornetz-Datenbanken nicht unterstützt, da Sensoren nur ausgelesen und nicht beschrieben werden können. Darum ist das Einfügen, Aktualisieren und Löschen von externen Daten auf einer Sensornetz-Datenbank nicht möglich. Es besteht lediglich die Möglichkeit sensornetzintern in einem Buffer Werte zwischenzuspeichern und den Buffer zu löschen. Realisiert wird dies durch im Arbeitsspeicher von Sensoren (siehe Data Setup Pattern) erstellte Tabellen, 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.
  - Befehlsstruktur:
    - Werte einfügen in Buffer-Tabelle: SELECT field1, field2, ... FROM sensors SAMPLE PERIOD x INTO name
- Data Setup Pattern: Es können Tabellen im Arbeitsspeicher der Sensoren erstellt werden, die dann Werte von Sensoren aufnehmen und halten können. Die Erstellung solcher Tabellen wird durch SQL-CREATE BUFFER und das Löschen des gesamten Buffers durch SQL-DROP realisiert.
  - Eine Buffer-Tabelle erstellen: CREATE BUFFER name SIZE x (field1 type, field2 type, ...)
    - *Bemerkungen:* **x** *ist die Anzahl der Zeilen,* ***type*** *ist ein Datentyp aus der Menge {uint8, uint16, uint16, int8, int16, int32} und* ***field1***, ***field2***, *usw. sind Spaltennamen, die wie der Tabellenname jeweils 8 Zeichen lang sein dürfen.*
  - Alle Buffer-Tabellen löschen: DROP ALL
- Stored Procedure Pattern: Es gibt für die TinyDB einen *stored procedures* ähnlichen Ansatz, dabei können falls eine bestimmte Bedingung gilt (z.B. Temperatur > 20°C) sogenannte *commands* aufgerufen werden. Der Code für diese Methoden wird auf die entsprechenden Sensoren übertragen und dort dann bei Bedarf ausgeführt. Ein Zugriff von außerhalb wie bei *stored procedures* ist hier allerdings nicht möglich.
  - Beispiele:
    - 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.*
- Set Retrieval Pattern: Realisierung ebenso wie bei Datenbanken beschrieben.

- Set Access Pattern: Realisierung ebenso wie bei Datenbanken beschrieben.
- Tuple IUD Pattern: Realisierung ebenso wie bei Datenbanken beschrieben.
- Synchronization Pattern: Hier gilt dasselbe wie für das Set IUD Pattern. Da es nicht möglich ist, Daten von außen in die Sensornetz-Datenbank einzubringen, wird die Realisierung dieses Patterns nicht unterstützt bzw. benötigt.

6.2.3 Resultierende BPEL-Aktivitäten

In diesem Abschnitt werden die durch Abschnitt 6.2.2 identifizierten BPEL-Aktivitäten aufgezählt und ihre Funktion noch einmal kurz beschrieben. Dazu gehört z.B. auch welche Attribute welchen Typs für die einzelnen Aktivitäten benötigt werden. Generell gilt, dass alle Aktivitäten mindestens die vier Variablen *kind*, *dqAddress* und *statement* vom Typ String und *dqReference* vom Typ EPR (endpoint reference) besitzen. Die Variable *kind* dient zur Angabe des Datenquellentyps für den die Aktivität ausgeführt wird, also ob es sich um ein Dateisystem, eine Datenbank oder ein Sensornetz handelt. Diese Auswahl ist wichtig um intern den richtigen SQL-Dialekt für die entsprechende Datenquelle auszuwählen. Die Variable *dqAddress* dient zur Angabe der Datenquellenadresse und die Variable *statement* zur Haltung des entsprechenden SQL- oder XQuery-Befehls der ausgeführt werden soll. Die Variable *dqReference* ist eine *endpoint reference* auf eine Datenquelle und dient als Alternative zur Datenquellenadresse. Diese vier Variablen besitzt jede der definierten Aktivitäten. Darum werden diese nachfolgend nicht jedesmal explizit angegeben und nur falls benötigt weitere aktivitätsspezifische Variablen beschrieben. Weiterhin werden die verschiedenen Ausprägungen der einzelnen Aktivitäten im Hinblick auf die zugrundeliegende Datenquelle aufgezeigt, so dass am Ende ein vollständiger Überblick aller definierten Aktivitäten und ihrer Ausprägungen vorliegt. Generell gibt es durch die verschiedenen Datenquellen nur wenige strukturelle Unterschiede in den Aktivitäten. Das liegt vorallem daran, dass auf datenquellenspezifische Eigenschaften bereits in der graphischen Oberfläche, also dem Eclipse BPEL Designer, eingegangen werden soll und diese so durch entsprechende Dialoge intern immer auf dieselbe Struktur abgebildet werden können. Eben dazu sollen auch alle Befehle in einer SQL-ähnlichen Syntax angegeben werden, um genau diese allgemeine Handhabung realisieren zu können. Wie diese Befehle nun intern verarbeitet werden, braucht der Benutzer nicht zu wissen und er kann dadurch schnellstmöglich mit nur "einer" Anfragesprache alle zur Verfügung gestellten Daten- und Datenquellenbefehle für alle unterstützten Datenquellen ausführen.

Select Activity

Diese Aktivität ermöglicht es aus jeder beliebigen Datenquelle Daten zu lesen. Weitere spezifische Attribute werden dafür nicht benötigt. Die Select Activity ist für alle Datenquellen gleich strukturiert und nur die entsprechenden SQL-Befehle unterscheiden sich.

Insert Activity

Diese Aktivität ermöglicht es Daten in jede Datenquelle einzufügen. Da dies für Sensornetze nicht relevant ist, wird diese Aktivität für Sensornetze für das sensornetzinterne Einfügen von Sensornetzdaten in Buffer-Tabellen genutzt (wie auch in Abschnitt 6.2.2 beschrieben).

Update Activity

Diese Aktivität ermöglicht es Daten aus Datenquellen mit externen Daten zu aktualisieren. Diese Aktivität existiert nicht für Sensornetze.

Delete Activity

Diese Aktivität ermöglicht es Daten auf beliebigen Datenquellen zu löschen. Diese Aktivität existiert nicht für Sensornetze.

Create Activity

Diese Aktivität ermöglicht es Zuordnungseinheiten (Dateien, Ordner, Tabellen, Schema, usw.) auf beliebigen Datenquellen zu erstellen. Dabei werden die folgenden Befehle zur Erstellung von Zuordnungseinheiten auf den entsprechenden Datenquellen benötigt:

- Dateisysteme
  - CREATE FOLDER
  - CREATE FILE
- Datenbanken
  - CREATE TABLE
  - CREATE SCHEMA
  - CREATE VIEW
- Sensornetze
  - CREATE BUFFER

Call Activity

Diese Aktivität ermöglicht es auf der Datenquelle hinterlegte Prozeduren auszuführen. Diese Aktivität existiert nur für Datenbanken.

RetrieveSet Activity

Diese Aktivität ermöglicht es Daten von Datenquellen in den Prozessspeicher zu laden.

WriteBack Activity

Diese Aktivität ermöglicht es Daten aus dem Prozessspeicher auf die originale Datenquelle zurückzuschreiben. Diese Aktivität existiert nicht für Sensornetze.

Optionale Aktivitäten:

- **Import Activity**: Diese Aktivität ermöglicht es lokale Daten (z.B. Simulationsparameter) des Benutzers in einen Prozess einzubinden und in diesem zu verwenden.
- **Export Activity**: Diese Aktivität ermöglicht es Daten eines Prozesses (z.B. Simulationsergebnisse) lokal auf den Benutzer-Rechner zu exportieren.
- **Move Activity**: Diese Aktivität ermöglicht es Daten zwischen beliebigen Datenquellen zu kopieren/verschieben, d.h. es können beispielsweise Daten aus einer Datei in eine Datenbank-Tabelle verschoben/kopiert werden.
- **Drop Activity**: Diese Aktivität ermöglicht es Zuordnungseinheiten auf beliebigen Datenquellen zu löschen. Dazu werden die folgenden Befehle benötigt:
  - Dateisysteme
    - DROP FOLDER
    - DROP FILE
  - Datenbanken

    - * DROP TABLE
    - * DROP SCHEMA
    - * DROP VIEW
  - Sensornetze
    - * DROP ALL

6.3 Eclipse BPEL Designer

6.4 Monitoring

6.4.1 Event-Modell

6.5 ...

7 Einzusetzende Technologien

7.1 Materialien

7.2 Werkzeuge

7.2.1 Entwicklungsumgebung

7.2.2 Sonstige Werkzeuge

8 Änderungsgeschichte

- **Version 0.1**, 11. September 2009: Erstellung des Dokuments.

Literatur

[1] Begriffslexikon

[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, April 7-12, 2008