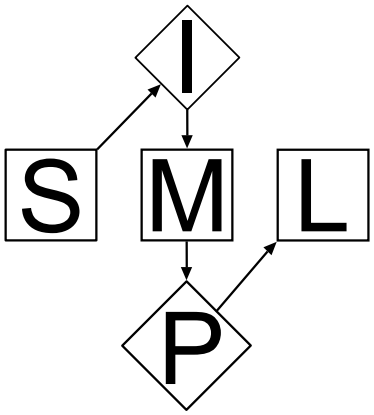
Studienprojekt SIMPL

Feinentwurf

Version 0.1

13. Januar 2010



Inhaltsverzeichnis

Änderungsgeschichte

Version	Datum	Autor	Änderungen
0.1	13.11.2009	zoabifs	Erstellung des Dokuments
0.2	04.01.2010	schneimi	Überarbeitung der Struktur, Kapitel 1
0.3	09.01.2010	schneimi	Kapitel 2
0.4	12.02.2010	schneimi	Kapitel 5
0.5	12.01.2010	rehnre	Kapitel 3.1, 3.2, 3.3
0.6	12.01.2010	huettiwg	Kapitel 3.4
0.7	12.01.2010	bruededl	Kapitel 4
0.8	12.01.2010	hahnml	Diagramme überarbeitet

1 Einleitung

Dieses Kapitel erklärt den Zweck des Dokuments, den Zusammenhang zu anderen Dokumenten und gibt dem Leser einen Überblick über den Aufbau des Dokuments.

1.1 Zweck dieses Dokuments

Der Feinentwurf beschreibt Details der Implementierung der Komponenten, die im Grobentwurf [1] in Kapitel 3 vorgestellt wurden. Die Komponenten werden ausführlich beschrieben und ihre Funktionalität durch statische und dynamische UML-Diagramme visualisiert. Der Feinentwurf bezieht sich im Gegensatz zum Grobentwurf aktuell nur auf die erste Iteration und wird mit der zweiten Iteration vervollständigt. Grobentwurf und Feinentwurf bilden zusammen den Gesamtentwurf des SIMPL Rahmenwerks.

1.2 Gliederung

Der Feinentwurf gliedert sich in die folgenden Kapitel.

- Kapitel 2 “SIMPL Core” beschreibt die Implementierung des SIMPL Cores und seinen Web Services. (siehe [1] Kapitel 3.1)
- Kapitel 3 “Apache ODE” beschreibt die Implementierung der Datamanagement-Aktivitäten (DM-Aktivitäten) und das externe Auditing. (siehe[1] Kapitel 3.2)
- Kapitel 4 “Eclipse Plug-Ins” beschreibt die Implementierung der Eclipse Plug-Ins, die für das SIMPL Rahmenwerk realisiert werden. (siehe[1] Kapitel 3.3)
- Kapitel 5 “Kommunikation” beschreibt die Kommunikation der Komponenten im SIMPL Rahmenwerk auf Funktionsebene.

2 SIMPL Core

Abbildung 1 zeigt den Aufbau des SIMPL Cores mit Paketstruktur, Klassen und Interfaces, sowie deren Zusammenhänge über Verbindungspfeile, die in den folgenden Abschnitten beschrieben werden. Als Services werden dabei allgemein die Dienste des SIMPL Cores bezeichnet. Manche dieser Dienste werden nach Außen über Web Services verfügbar gemacht und werden, wenn diese explizit gemeint sind, auch als solche bezeichnet wie z.B. Administration Web Service.

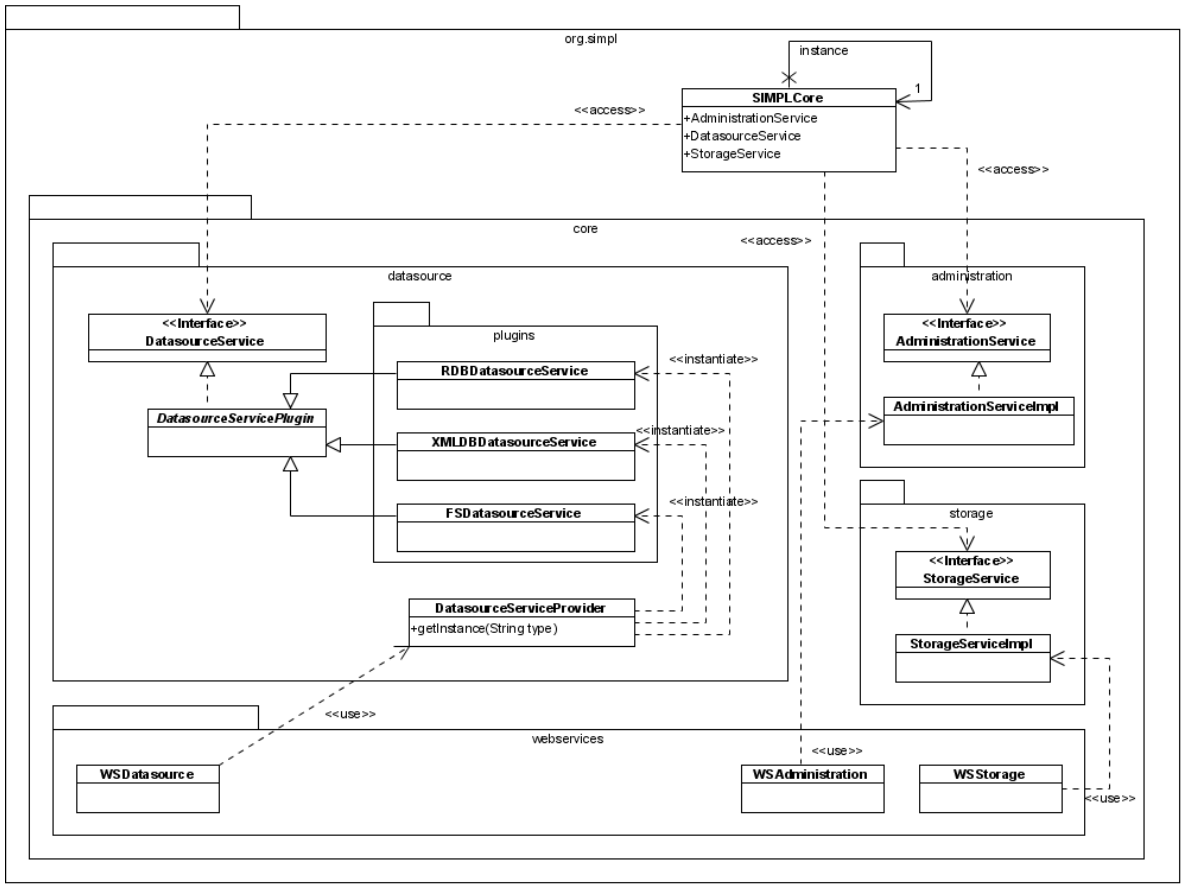


Abbildung 1: SIMPL Core Klassendiagramm

2.1 Paketstruktur

Der SIMPL Core besitzt folgende Paketstruktur, die sich in einen Kernbereich, sowie Bereiche für die Dienste (Services) und Web Services aufteilt.

org.simpl.core

Hier befinden sich zentrale Klassen des SIMPL Cores, die auf die Dienste des SIMPL Core zurückgreifen, wie z.B. die SIMPLCore-Klasse, die in Kapitel 2.3 näher beschrieben wird.

org.simpl.administration

Hier befinden sich alle Klassen zur Realisierung des Administration Service ([1] 3.3.1).

org.simpl.storage

Hier befinden sich alle Klassen zur Realisierung des Storage Service ([1] 3.3.5).

org.simpl.datasource

Hier befinden sich alle Klassen zur Realisierung des Datasource Service ([1] 3.3.5).

org.simpl.datasource.plugins

Hier befinden sich die Plug-Ins für den Datasource Service, die für die verschiedenen Datenquellentypen entwickelt werden. Falls sich die einzelnen Plug-Ins auf mehrere Klassen verteilen, werden diese zusätzlich auf eigene Unterpakete verteilt. Das Plug-In-System wird in Kapitel 2.4 näher beschrieben.

org.simpl.webservices

Hier befinden sich die Web Services des SIMPL Core, die den Zugriff von Außen auf den SIMPL Core ermöglichen. Alle Klassen werden mit JAX-WS-Annotationen versehen und können als Webservices in Apache ODE deployt werden.

2.2 SIMPL Core Services

In diesem Abschnitt werden die Dienste des SIMPL Cores und ihre Funktionsweise beschrieben.

2.2.1 Administration Service

Der Administration Service ist für die Verwaltung der Einstellungen der Admin-Konsole des SIMPL Core Eclipse Plug-Ins zuständig. Die Einstellungen der Admin-Konsole werden dabei über das SIMPL Core Client Plug-In an den Administration Service übermittelt oder angefordert. Die auf diese Weise zentral im SIMPL Core hinterlegten Einstellungen können dann bei Bedarf direkt von anderen SIMPL Core Diensten, die diese Informationen benötigen, ausgelesen werden. Zur persistenten Speicherung der Einstellungen und weiterer Daten wird eine eigene eingebettete Apache Derby (Embedded Derby) Datenbank verwendet, die vom gesamten SIMPL Core genutzt wird.

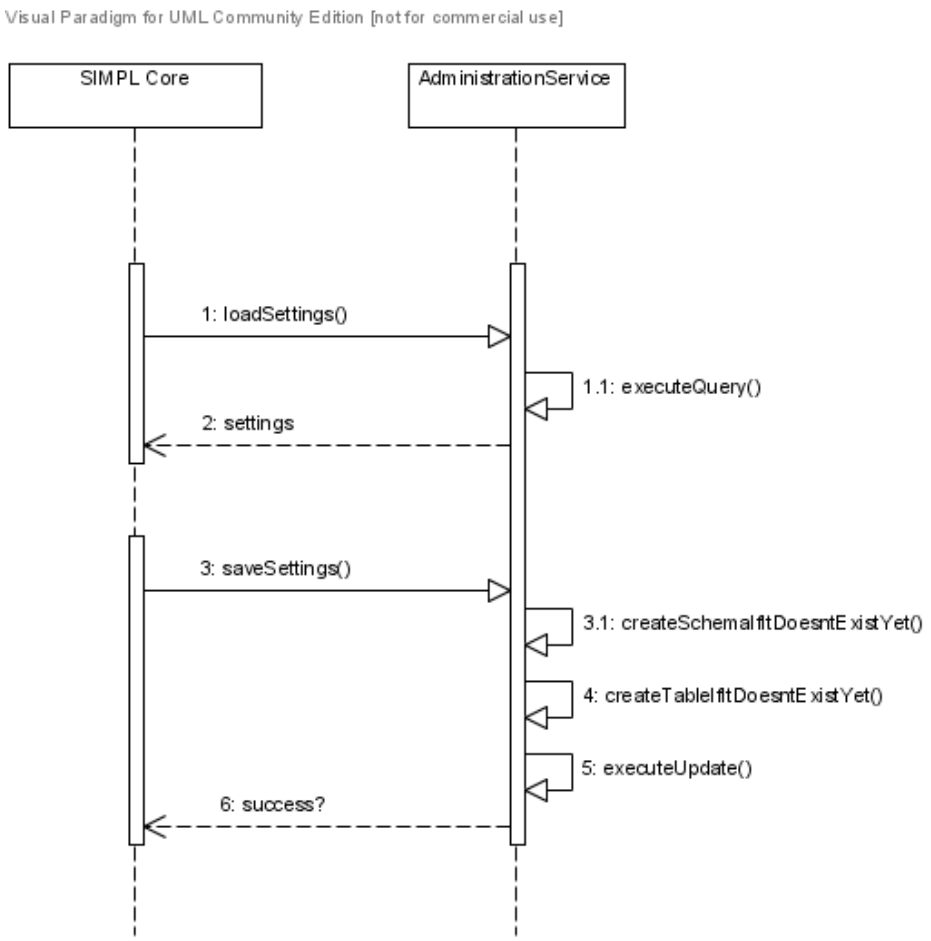


Abbildung 2: Sequenzdiagramm des Administration Services

Abbildung 2 zeigt die Verwendung und die Funktionalität des Administration Service. Mit der loadSettings()-Methode können Einstellungen aus der Datenbank geladen werden, dafür wird intern eine einfache Datenbankabfrage genutzt. Die Einstellungen werden dabei als HashMap zurückgeliefert und auch so beim Speichern übergeben, damit sowohl die Bezeichnung der Einstellung wie auch ihr Wert zu jeder Zeit verfügbar ist. Zur Identifizierung verschiedener Einstellungen, wie z.B. Standard-Einstellungen und zuletzt gespeicherten Einstellungen, besitzt jede Einstellung eine eindeutige Id. So kann später die Admin-Konsole um das Laden und Speichern von benutzerspezifischen Preset-Einstellungen ergänzt werden.

Die Struktur der Datenbank orientiert sich direkt am Aufbau der Admin-Konsole, da hier immer Ober- und Unterpunkte zusammengehören, wurde auf der Datenbank diese Beziehungen durch die Strukturierung mit Schemata und Tabellen umgesetzt. So gibt es für jeden Oberpunkt, wie z.B. *Auditing* ein gleichnamiges Schema und für jeden Unterpunkt eines Oberpunkts, wie z.B. *General* eine gleichnamige Tabelle im Schema des Oberpunkts. Daraus ergibt sich der genaue Pfad einer, in der Datenbank gespeicherten, Einstellung aus der Auswahl in der Admin-Konsole. Mit der saveSettings()-Methode können Einstellungen in einer entsprechenden Tabelle eines Schemas gespeichert werden. Dazu wird zuerst überprüft, ob das zu den Einstellungen gehörige Schema bereits existiert oder noch erzeugt werden muss und anschließend, ob die Tabelle bereits existiert oder noch erzeugt werden muss. Die Tabelle wird dabei direkt aus den übergebenen Einstellungen automatisch erzeugt, indem die Einstellungsnamen als Spaltennamen verwendet werden. Wenn nun Schema und Tabelle vorhanden

sind wird überprüft, ob die zu speichernde Einstellung bereits vorhanden ist und nur noch aktualisiert werden muss oder ob die Einstellung neu angelegt, also eine neue Zeile eingefügt werden muss.

2.2.2 Storage Service

Der Storage Service ist für die Verwaltung von Daten aller SIMPL Core Services zuständig. Dafür nutzt er ebenfalls die eingebettete Apache Derby Datenbank. Seine Architektur und Funktionalität ist der des Administration Services sehr ähnlich, mit dem Unterschied, dass die Struktur der zu speichernden Daten und ihre Quelle sich zur Laufzeit ständig ändern können. Auch der Storage Service nutzt das Prinzip die Daten nach ihrer Herkunft mit Schemata und Tabellen zu strukturieren. Da hier aber keine natürliche Struktur wie beim Administration Service vorliegt, wird eine entsprechende Gliederung durch die Zugehörigkeit der Services erzeugt. So werden alle Daten von Services, die etwas mit Sicherheit zu tun haben unter dem Schema *Security* in entsprechende Tabellen gespeichert, wobei die Tabellennamen aus den Klassennamen der Services erzeugt werden. Die Zugehörigkeit eines Services wird dabei in seiner Implementierung als Konstante hinterlegt und kann so einfach zur Laufzeit genutzt werden. Daraus ergibt sich wieder ein eindeutiger Pfad zu den, in der Datenbank gespeicherten, Daten eines jeden SIMPL Core Services. Die Daten werden auch wie im Administration Service wieder als HashMaps verarbeitet, um sowohl die Bezeichnung wie auch den Wert immer verfügbar zu haben. Im Storage Service wird allerdings immer der erste in der HashMap hinterlegte Wert als Id der zugehörigen Datenbanktabelle interpretiert.

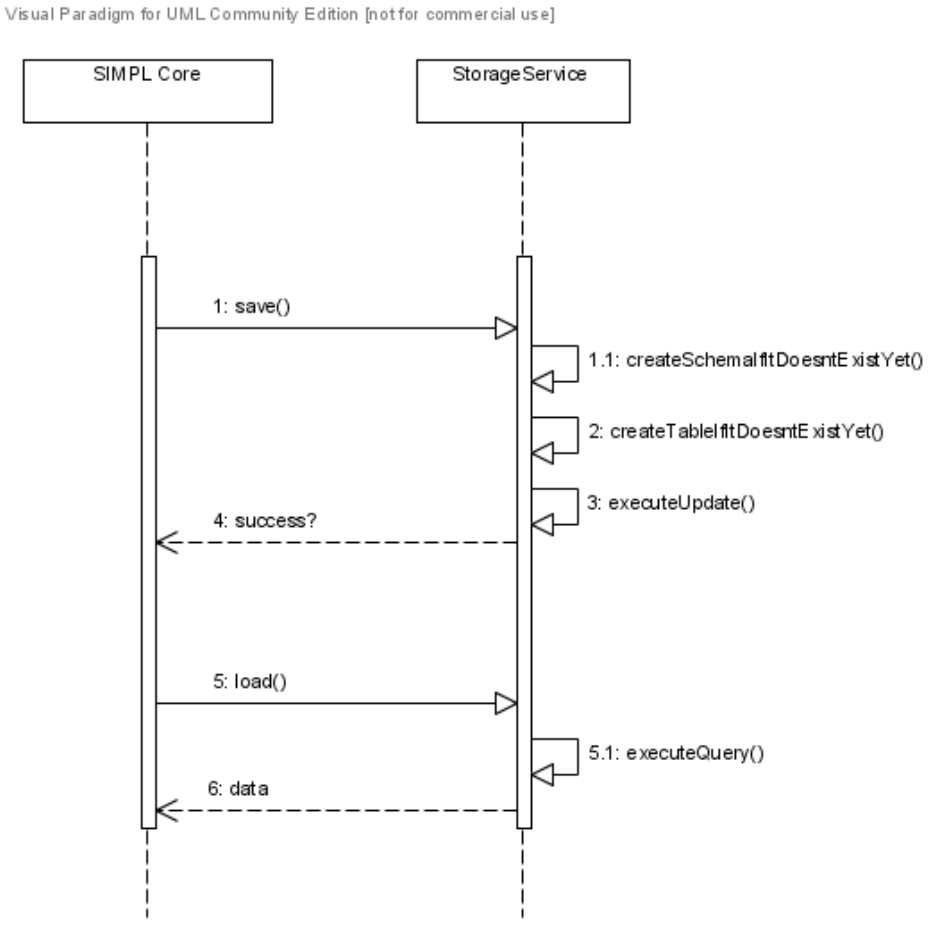


Abbildung 3: Sequenzdiagramm des Storage Services

Abbildung 3 zeigt die Verwendung und die Funktionalität des Storage Services. Mit der load()-Methode können wieder Daten aus der Datenbank geladen werden und die save()-Methode speichert alle Daten eines Services nach der selben Vorgehensweise wie die saveSettings()-Methode des Administration Services.

2.2.3 Datasource Service

Der Datasource Service stellt die Verbindung zu den verschiedenen Datenquellen her. Dazu gibt es gibt ihn in verschiedenen Ausprägungen, die zu den verschiedenen Typen von Datenquellen über ein Plug-In System bereitgestellt werden. Die Instanz eines Datasource Service kann nur über den Datasource Service Provider (DatasourceServiceProvider) angefordert werden, der für die Verwaltung der Plug-Ins zuständig ist. Eine genaue Beschreibung des Plug-In Systems folgt in Kapitel 2.4. Folgende privaten (private) und öffentlichen (public) Funktionen werden von einem Datasource Service realisiert:

private openConnection() Öffnet eine Verbindung zu einer Datenquelle und liefert ein Verbindungsobjekt über das mit der Datenquelle kommuniziert werden kann.

private closeConnection() Schließt die Verbindung zu einer Datenquelle und bestätigt den Verbindungsabbau mit einem Rückgabewert.

public queryData() Ermöglicht die Anforderung von Daten von einer Datenquelle durch ein Statement der entsprechenden Anfragesprache wie z.B. ein SELECT-Statement in der Anfragesprache SQL. Die Rückgabe der Daten erfolgt als SDO.

public defineData() Wird verwendet um Datenstrukturen über ein Statement zu definieren wie z.B. das Erstellen von Tabellen in einer Datenbank und liefert eine Bestätigung als Rückgabewert.

public manipulateData() Wird verwendet um bestehende Daten zu manipulieren bzw. zu aktualisieren. Die Funktion erhält Daten in Form eines SDO und ein Statement mit dem die Verwendung der Daten beschrieben wird. Der Erfolg der Operation wird mit einem Rückgabewert bestätigt.

2.3 Die SIMPLCore Klasse

Die SIMPLCore-Klasse bildet den zentralen Zugriffspunkt auf alle Dienste des SIMPL Cores für den Zugriff auf Klassenebene. Damit die Instanzen der Dienste nur einmal existieren und nicht bei jedem Zugriff erneut erstellt werden, ist die Klasse als Singleton ([1] Kapitel 3.3) ausgelegt. Diese Klasse wird von den Apache ODE Extension Activities (siehe 3.1) benutzt um DM-Aktivitäten auszuführen, aber auch innerhalb des SIMPL Cores, wenn sich Dienste gegenseitig verwenden.

2.4 Datasource Plug-Ins

Die Unterstützung verschiedener Typen von Datenquellen wird durch Datasource Plug-Ins realisiert um eine Erweiterungsmöglichkeit zu garantieren. Dies wird durch die Bereitstellung einer abstrakten Klasse erreicht, von der sich die Plug-Ins ableiten lassen. Mit der Reflection API von Java ist es möglich, die Plug-Ins zur Laufzeit zu erkennen und zu verwenden, ohne dass bestehender Code angepasst werden muss.

DatasourceService (interface) Das DatasourceService-Interface schreibt alle Funktionen vor, die von den DatasourceServices (Plug-Ins) implementiert werden müssen.

DatasourceServicePlugin (abstract class) Bei der DatasourceServicePlugin-Klasse handelt es sich um eine abstrakte Klasse, die an das DatasourceService-Interface gebunden ist und damit das Grundgerüst für einen Datasource Service bildet. Ein Plug-In muss diese Klasse erweitern und wird dadurch gezwungen das DatasourceService-Interface zu implementieren.

DatasourceServiceProvider Über den DatasourceServiceProvider kann mit der Methode getInstance() die Instanz eines DatasourceService angefordert werden. Dies geschieht über eine eindeutige Typkennzeichnung (z.B. RDB, XML, ...), mit der die entsprechende Instanz der Klasse über die Class.forName-Methode der Java Reflection API erzeugt und ausgeliefert wird.

2.5 Web Services

Die Web Services werden mit den JAX-WS annotierten Klassen wie folgt bereitgestellt. Zunächst wird mit Hilfe des Befehls wsgen.exe (..\Java\jdk1.6.0_14\bin\wsgen.exe), eine WSDL-Datei zu einer Klasse erzeugt. Die WSDL-Datei wird anschließend zusammen mit der kompilierten Klasse als JAR-Datei in Apache ODE hinterlegt (..\Tomcat 6.0\webapps\ode\WEB-INF\servicejars) und wird damit beim Start von Apache Tomcat von Apache ODE als Web Service bereitgestellt.

Komplexe Objekte wie z.B. HashMaps, die intern von den SIMPL Core Diensten zur Ausführung benötigt werden, werden als String serialisiert an die Web Services übergeben und in der Form auch als Rückgabeparameter empfangen. Bei der Deserialisierung, werden die Objekte wiederhergestellt und können als solche verwendet werden. Eine Ausnahme bilden die SDO Objekte, die bereits über eine XML Darstellung verfügen und in dieser direkt übermittelt werden können.

2.5.1 Datasource Web Service (WSDatasource)

Der Datasource Web Service bietet eine Schnittstelle zu allen Ausprägungen des Datasource Service im SIMPL Core. Die Funktionen des Datasource Web Service, entsprechen den öffentlichen Funktionen der Datasource Services, erhalten aber als zusätzlichen Parameter die eindeutige Typkennzeichung (siehe 2.4 DatasourceServiceProvider) der angeforderten Datenquelle, über die bei einem Zugriff intern der entsprechende Datasource Service angesprochen werden kann. Zusätzlich besitzt der Datasource Web Service eine Funktion um Metadaten von einer Datenquelle anzufordern, die beispielsweise von Eclipse zur Auswahl in der GUI benötigt werden, wie z.B. die existierenden Tabellen in einer Datenbank.

2.5.2 Administration Web Service (WSAdministration)

Der Administration Web Service ist die direkte Schnittstelle des Administration Service nach Außen und besitzt daher die gleichen Funktionen wie dieser, mit dem Unterschied, dass komplexe Parameter, aus oben genannten Gründen, als String-Parameter gehandhabt werden.

2.5.3 Storage Web Service (WSStorage)

Der Storage Web Service bietet anderen Diensten von Außen die Möglichkeit Einstellungen zu speichern und zu laden. Für den Storage Web Service gilt das gleich wie für den Administration Service, nur dass dieser die direkte Schnittstelle zum Storage Service darstellt.

3 Apache ODE

In diesem Kapitel wird auf die Erweiterungen die an Apache ODE vorgenommen werden eingegangen. Dies beinhaltet die SIMPL DM Extension Activities, das SIMPL Event System, sowie das SIMPL DAO. Es wird auf die verschiedenen Funktionalitäten als auch auf deren Implementierung eingegangen.

3.1 SIMPL DM Extension Activities

Die SIMPL DM Extension Activities (siehe Abbildung 4) haben als Hauptklasse die Klasse SIMPLActivity, welche verschiedene Funktionalitäten für alle weiteren Extension Activities anbietet. Die Extension Activities nutzen zur Ausführung der verschiedenen Data-Management-Operationen den Datasource Service des SIMPL Cores. Die Implementierung der Extension Activities wird folgendermaßen umgesetzt:

Zunächst muss eine neue Aktivität von der Klasse „AbstractSyncExtensionOperation“ abgeleitet werden und die dadurch vererbten Methoden müssen implementiert werden. Die Methode „runsync“ ist hierbei für die eigentliche Ausführung der neuen Aktivität verantwortlich. Dafür ist die Nutzung der beiden Parameter „context“ und „element“ notwendig. Mit “context” hat man die Möglichkeit auf BPEL-Variablen und weitere Konstrukte die im Prozess vorhanden sind zuzugreifen. Der Inhalt des BPEL-Prozess-Dokuments wird in Nodes geparst um ein objektbasiertes Modell des BPEL Prozesses zu erzeugen. Mit „element“ ist es möglich auf die verschiedenen Eigenschaften dieser Nodes zuzugreifen und mit ihnen zu arbeiten.

Weiterhin ist es notwendig ein eigenes ExtensionBundle zu implementieren. Das ExtensionBundle ist notwendig, damit ODE weiß aus welchen Extension Activities besteht und sie zur Laufzeit ausgeführt werden können. Die Implementierung wird erreicht durch das Ableiten einer neuen Klassen von „AbstractExtensionBundle“. In dieser Klasse müssen nun in der Methode „registerExtensionActivity“ alle Klassen die für die Extension Activity von Bedeutung sind mit Hilfe von „registerExtensionOperation“ bei ODE registriert werden.

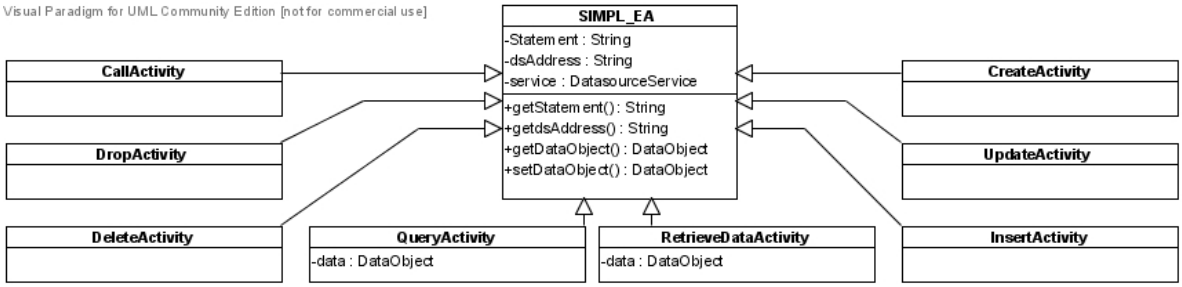


Abbildung 4: SIMPL DM Extension Activities

3.2 SIMPL Event System

Für die SIMPL Extension Activities wird eine Reihe von neuen Events eingeführt. Die Klassenhierarchie der Events ist in Abbildung 5 zu sehen. Die neuen Events unterteilen sich in DMEvents und ConnectionEvents, welche beide als Hauptklasse die Klasse SIMPLEvent haben. SIMPLEvent ist wiederum von Scope Event abgeleitet. DMEvents sind dabei für alle Ereignisse die während der Ausführung einer DM-Aktivität zuständig, während ConnectionEvents Rückmeldung über den Status der Verbindung geben. Die neuen Events werden dadurch als Scope Events in die bestehende Event Hierarchie von ODE eingegliedert. Dies erlaubt es uns diese direkt innerhalb der ExtensionAcitvities zu nutzen und aufzurufen.

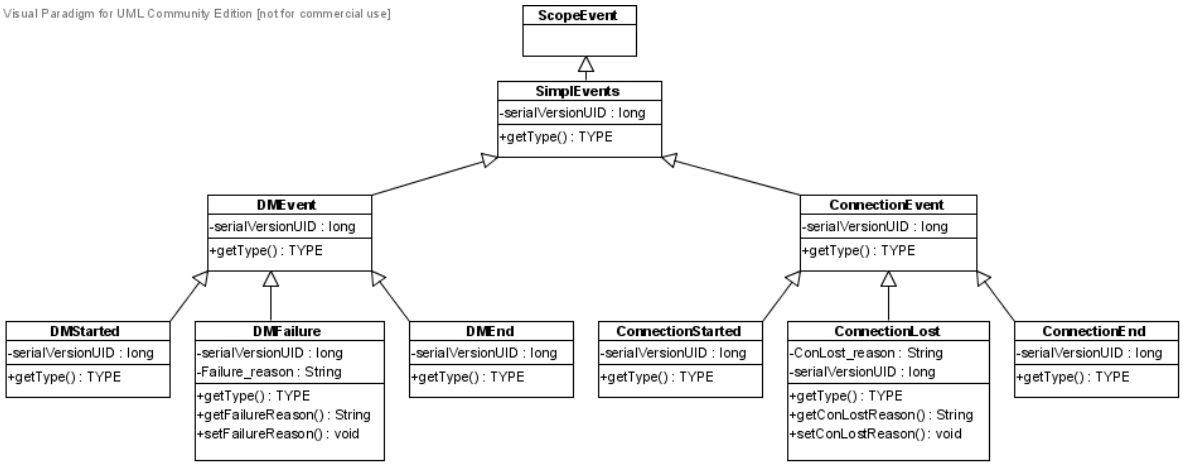


Abbildung 5: SIMPL Event System

3.3 Ausführung einer SIMPL DM Extension Activity

In Abbildung 6 wird die Ausführung einer Query-Activity, mit den während der Ausführung auftretenden Events, aufgezeigt. Hierbei ist zu erwähnen, dass die Query-Activity folgendermaßen durchgeführt wird:

1. Mit Hilfe der queryData-Methode werden die Daten aus der aktuellen Datenquelle gelesen und als DataObject zurückgegeben
2. Mit Hilfe der defineData-Methode wird eine neue Tabelle in der aktuellen Datenbank erzeugt
3. Mit Hilfe der manipulateData-Methode wird das unter 1. erzeugte DataObject in der in 2. erzeugten Tabelle abgespeichert

Die Events “DMStarted” und “DMEnd” werden zu Beginn bzw. am Ende der Ausführung erzeugt. Das Event “DMFailure” wird erzeugt falls die Rückmeldungsvariable “success” auf false gesetzt wurde.

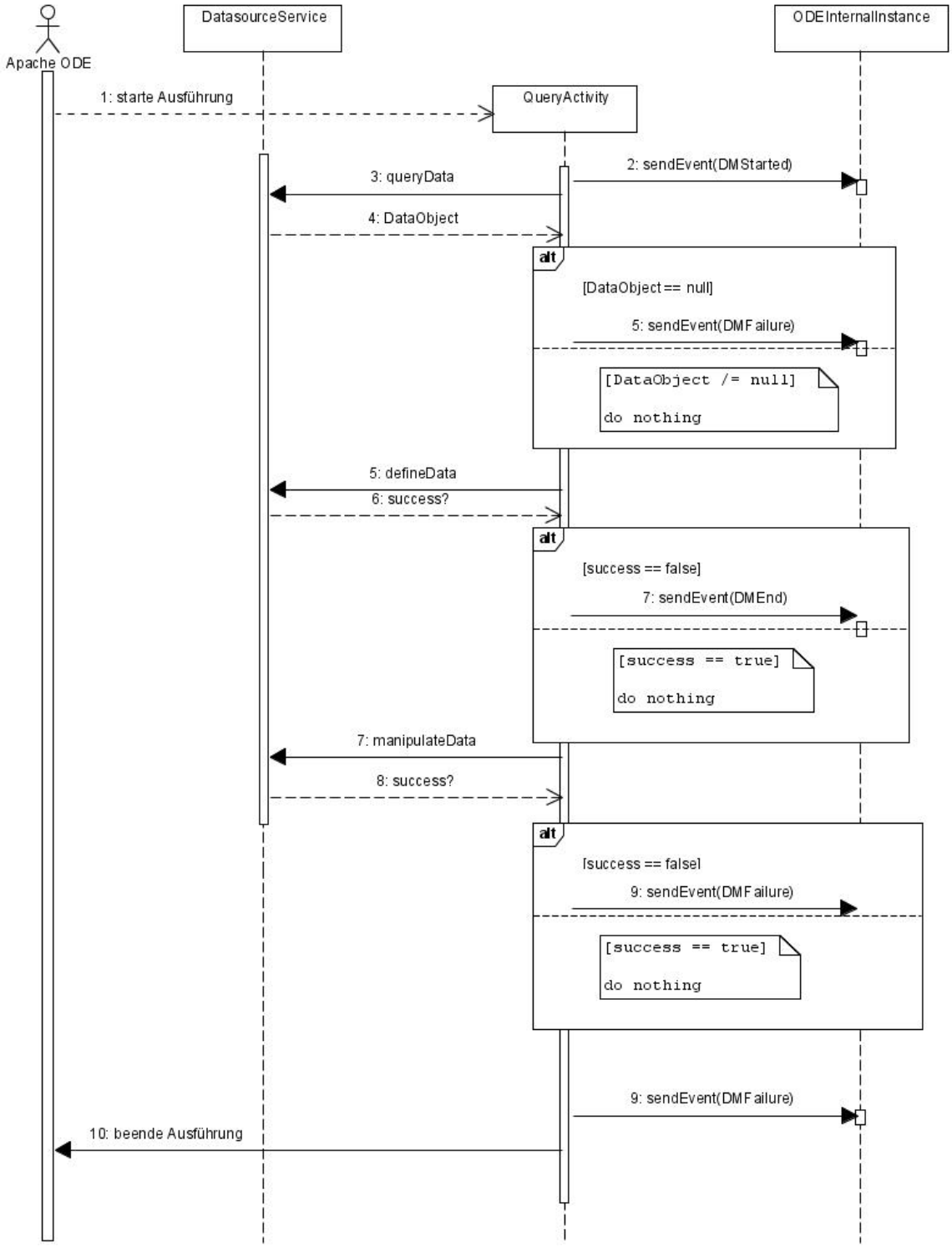


Abbildung 6: Ausführung einer SIMPL DM Extension Activity

3.4 SIMPL DAO

Das SIMPL Data Access Object (DAO) besteht aus der Implementierung der Interfaces aus dem Packet org.apache.ode.bpel.dao die in den folgenden Unterpunkten beschrieben werden. Das DAO wird dafür verwendet, wichtige Daten der Prozessausführung aufzuzeichnen und persistent zu speichern. (Siehe hierfür [1] 3.2.2).

Das SIMPL DAO übernimmt alle Eigenschaften der ODE internen Java Persistence API (JPA)-Implementierung und erweitert diese um die Eigenschaft, Daten per Service Data Object (SDO) an den SIMPL Core senden zu können. Von dort aus können die Daten, wie zum Beispiel Informationen über ausgeführte Prozesse(Name, Id, Typ), in beliebigen Datenquellen gespeichert werden. Die Übertragung der Daten findet dabei direkt in den setter Methoden der DAOs statt. Die DAO Daten werden trozdem auch weiterhin in der internen Apache Derby Datenbank gespeichert und von dort gelesen. Datentransfers an den SIMPL Core und damit verbundene beliebige Datenquellen, können nur schreibend, jedoch nicht lesend erfolgen.

3.4.1 DAO's

BpelDAOConnectionFactory

Die Factory verwaltet und erstellt die DAOConnection zu den ang-egebenen Datenquellen (standardmäßig zu der internen Derby Datenbank).

BpelDAOConnection

Die BpelDAOConnection stellt eine Verbindung zu den DAO's her. Hier werden zum Beispiel BPEL Events in das DAO eingefügt, Prozesse erstellt und Prozessinstanzen verwaltet, über die auf die Restlichen DAO's zugegriffen werden kann.

ActivityRecoveryDAO

Das ActivityRecoveryDao wird ausgeführt, wenn eine Aktivität den "recovery" Status einnimmt.

CorrelationSetDAO

Das CorrelationSetDAO wird ausgeführt, wenn in BPEL ein Correlation Set erstellt wird. Correlation Sets ermöglichen die Kommunikation einer Prozessinstanz mit seinen Partnern.

FaultDAO

Das FaultDao wird erstellt wenn ein Fehler in der Proessausführung passiert. Über dieses Dao kann auf die Informationen bezüglich des Fehlers, zum Beispiel der Name und der Grund für den fehler, zugegriffen werden.

MessageDAO

Repräsentiert eine Nachricht in der Datenbank.

MessageExchangeDAO

Das MessageExchangeDAO ist für den Austausch von Nachrichten zuständig..

MessageRouteDAO

Das MessageRouteDAO repräsentiert einen Nachrichten Anfrager, wie zum Beispiel ein Pick oder ein Receive.

PartnerLinkDAO

Das PartnerLinkDAO repräsentiert einen Partnerlink. Es enthält Informationen über die eigene Rolle, die Rolle des Partners und die Endpoint-Referenz.

ProcessDAO

Das ProcessDAO repräsentiert einen laufenden Prozess. Es enthält die Prozess-Id, den Prozess-Typ und die Prozessinstanzen.

ProcessInstanceDAO

Das ProcessInstanceDAO repräsentiert eine Prozess-Instanz und enthält alle Daten die einer Instanz zugehörig sind. Dazu zählen Events, Scopes, sowie wartende Pick- und Receive-Aktivitäten.

ScopeDAO

Das ScopeDAO repräsentiert eine Scope-Instanz. Es enthält eine Ansammlung von Correlation-Sets und XML-Variablen.

XmlDataDAO

Das XmlDataDAO representiert XML-Daten und wird dazu benutzt Bpel-Variablen zu modelieren.

3.4.2 DAO Java Persistence API (JPA)

Das Dao-Jpa ist eine Dao Implementierung, die auf dem Apache Open JPA basiert. Dieses stellt Funktionalitäten zur persistenten Speicherung auf relationalen Datenspeichern zur Verfügung. Über annotierte Variablen können somit die Dao Daten komfortabel in der ODE internen Derby Datenbank gespeichert werden.

3.4.3 DAO Lebenszyklus

Beim starten von Apache Tomcat wird auch ODE und somit der darin enthaltene BPEL-Server gestartet. Sofort wird die in OdeServer Klasse enthaltene init Methode aufgerufen. Diese ruft wiederrum die initDao Methode auf. Dort wird die DaoConnectionFactory geladen, welche zuvor in der OdeConfig-Properties definiert oder aus der Axis2.properties geladen wurde. Über die ConnectionFactory werden DaoConnections erstellt und bereitgestellt, mit deren Hilfe direkt auf die DAOs zugegriffen werden kann. Dies geschieht an den Stellen in Ode, wo die den DAOs entsprechenden BPEL Konstukte ausgewertet werden. So wird auf die ProcessDao zum Beispiel aus der ODEProcess Klasse aufgerufen, um die Prozessdaten persistent zu speichern.

4 Eclipse

Das SIMPL Rahmenwerk besteht aus der vorhandenen Eclipse IDE, dem Eclipse BPEL Designer Plug-In sowie die drei zu erstellenden Plug-Ins BPEL-DM Plug-In, SIMPL Core Plug-In und SIMPL Communications. Im Rahmen des Feinentwurfes wird die Anbindung an die vorhandenen Komponenten sowie die zu erstellenden Komponenten näher erläutert.

4.1 BPEL DM Plug-In

Mit dem BPEL-DM Plug-In werden die bestehenden Aktivitäten des Eclipse BPEL Designer Plug-Ins um die DM-Aktivitäten ergänzt. Das Plug- In läßt sich grob in die Pakete User-Interface, Modell und Erweiterung gruppieren. Das User-Interface Paket (org.eclipse.bpel.ui) sorgt für die grafische Darstellung in Eclipse und repräsentiert somit das zugrundeliegende Datenmodell. Dieses befindet sich im Paket org.eclipse.bpel.simpl.model. Für die Darstellung von Erweiterungen wird beispielhaft ein SQL Plug-In (Paketorg.eclipse.bpel.simpl.ui.sql) verwendet. Im folgenden Diagramm sind die Pakete und Klassen des kompletten Plug-Ins dargestellt um eine leichten Überblick zu ermöglichen. Diese Pakete und Klassen werden in den folgenden Unterkapitel näher erläutert.

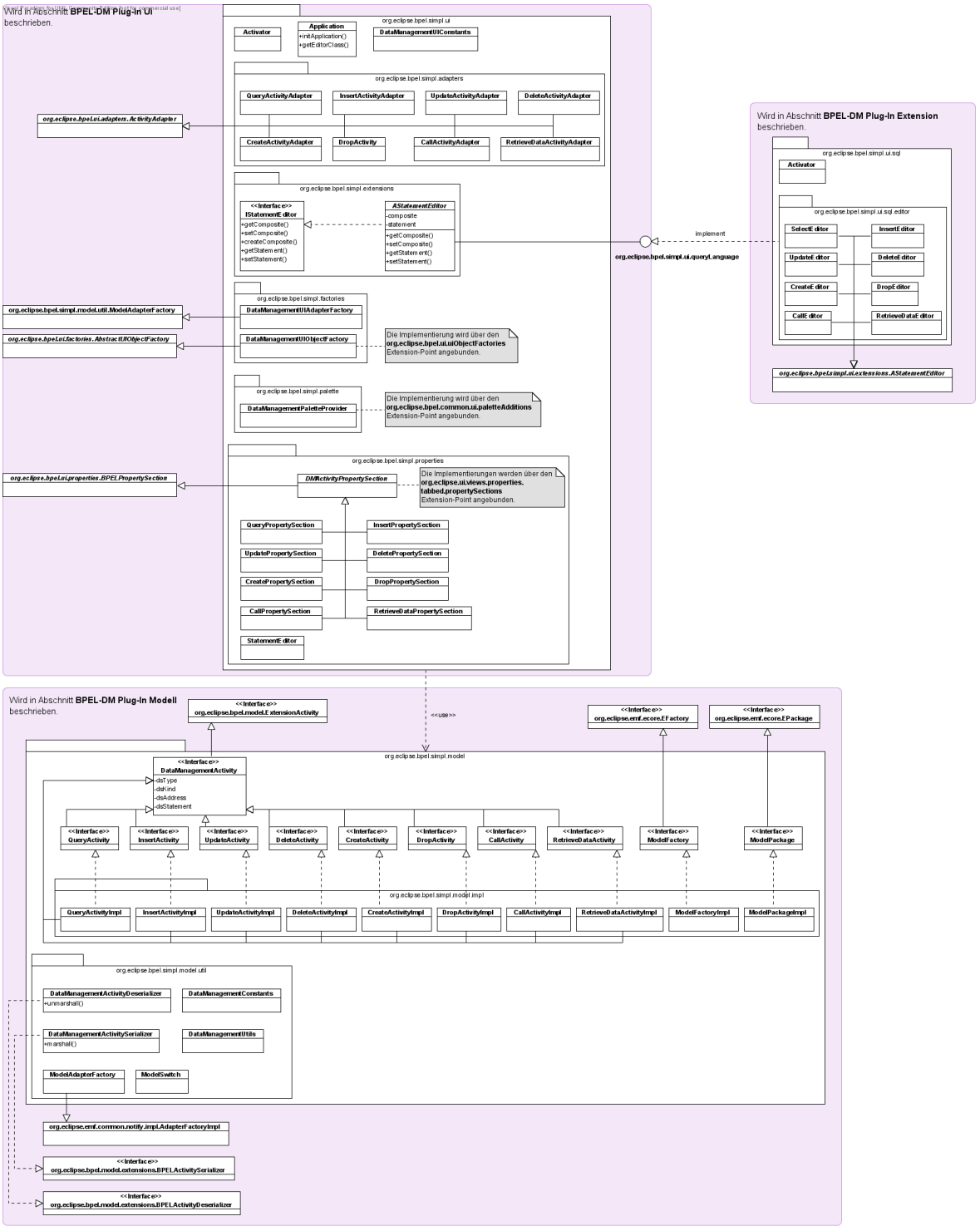


Abbildung 7: BPEL DM Plug-In

4.1.1 BPEL DM Plug-In User-Interface

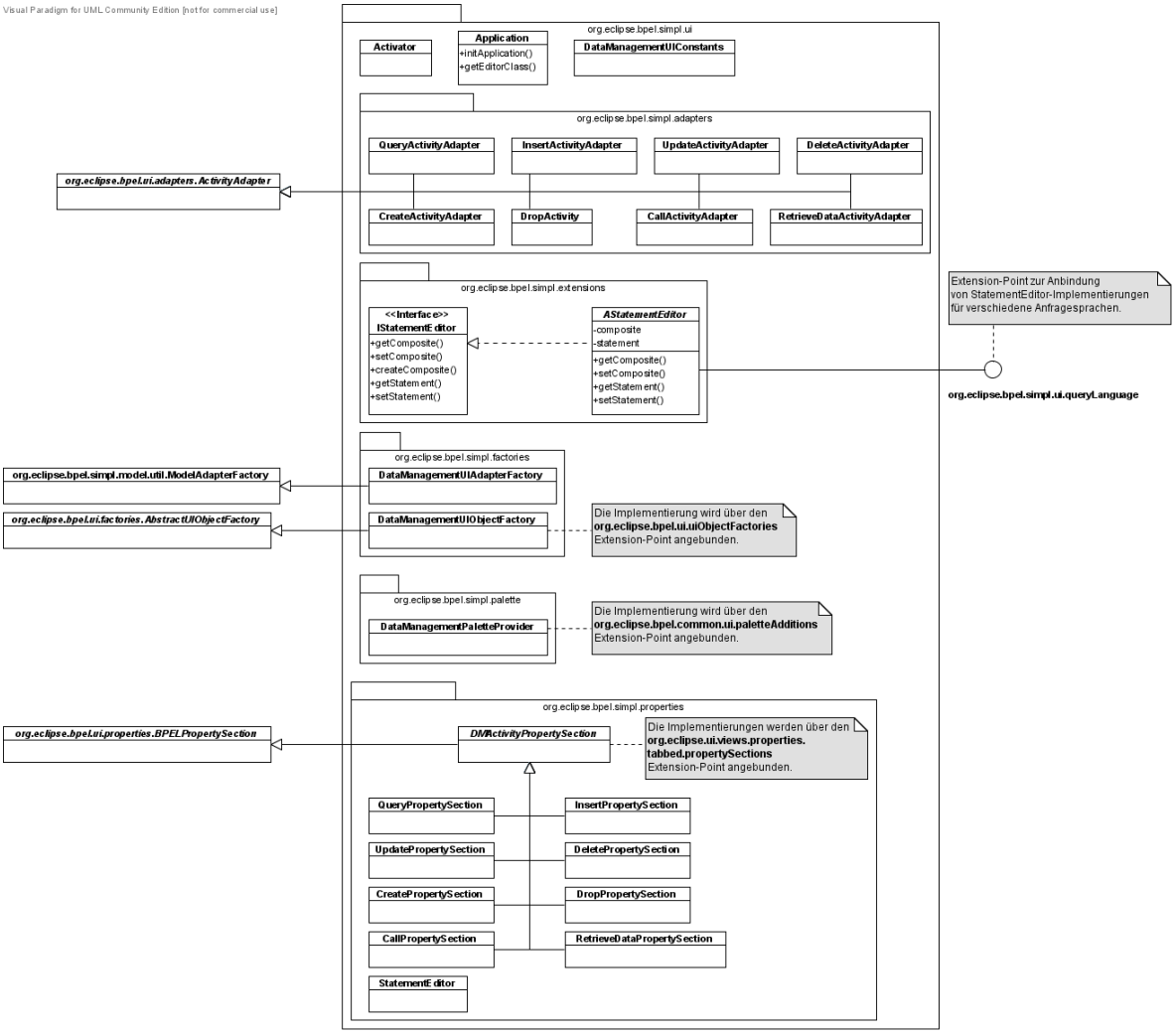


Abbildung 8: BPEL DM Plug-In User-Interface

Die Abbildung 8 zeigt den Aufbau der Darstellung des Datenmodells.

org.eclipse.bpel.simpl.adapters

Die Paket mit den verschiedenen Adapter-Klassen ist die Verbindung zu den Aktivitäten von BPEL.

org.eclipse.bpel.simpl.extensions

Die Paket beinhaltet ein Interface für die Statement Editoren für Erweiterungen. Das Interface I

org.eclipse.bpel.simpl.factories

Die beiden Klassen erzeugen die Adapter Objekte und normale Objekte. Die Anbindung erfolgt über den Extension Point org.eclipse.bpel.ui.uiObjectFactories.

org.eclipse.bpel.simpl.palette

Beinhaltet die grafische Palette der Aktivitäten für den BPEL Designer.. Anbindung erfolgt über den Extension Point org.eclipse.bpel.common.ui.paletteAdditions.

org.eclipse.bpel.simpl.properties

Beinhaltet die Anzeige der Eigenschaften der jeweiligen DM Aktivitäten. Anbindung erfolgt über den Extension Point org.eclipse.ui.views.properties.tabbed.propertySections.

4.1.2 BPEL DM Plug-In Modell

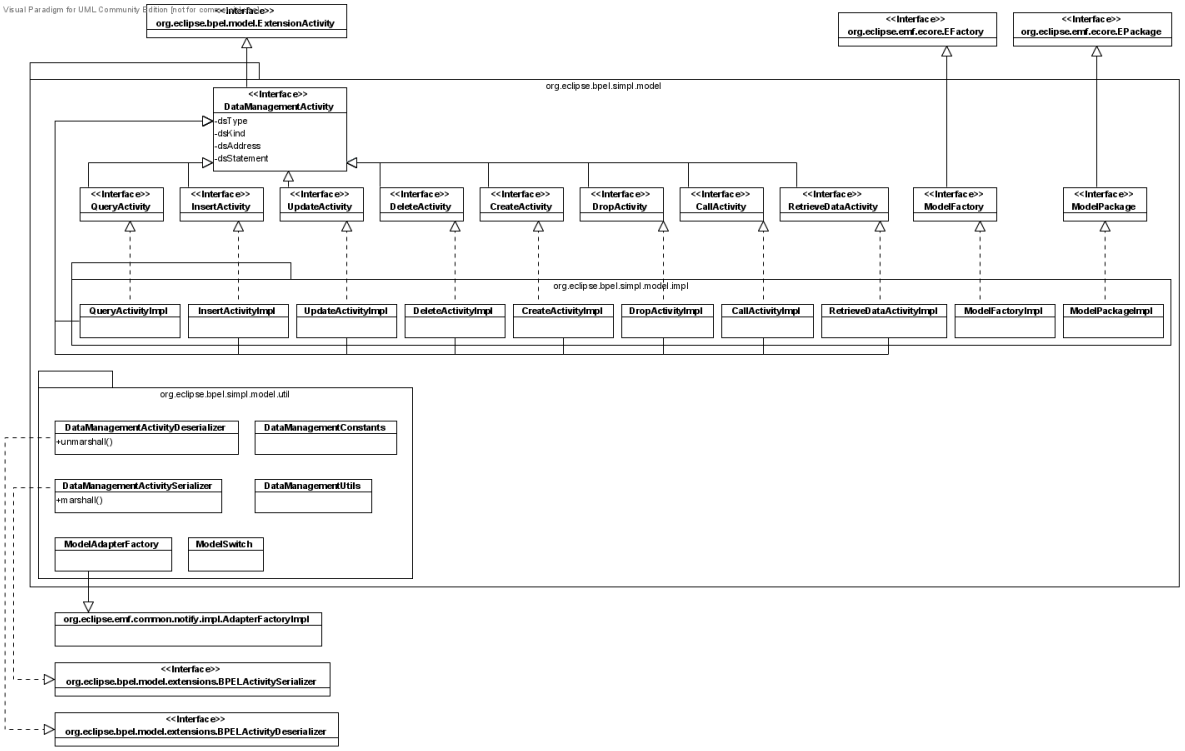


Abbildung 9: BPEL DM Plug-In Modell

Das Paket org.eclipse.bpel.simpl.model beinhaltet das Datenmodell. An oberster Stelle steht die DataManagementActivity-Klasse (Interface) welche mit den Variablen dsType, dsKind, dsAddress und dsStatement die gemeinsame Schnittmenge der Aktivitäten bildet. Diese werden an die Kindklassen wie z.B. QueryActivity (Interface) vererbt und um schnittstellenspezifische Eigenschaften erweitert. Die konkrete Realisierung erfolgt dann im Paket org.eclipse.bpel.simpl.model z.B. in der Klasse QueryActivityImpl. Im Paket org.eclipse.bpel.simpl.model.util befinden sich Zubehörklassen wie Serializer und Deserializer.

4.1.3 BPEL DM Plug-In Extension Modell

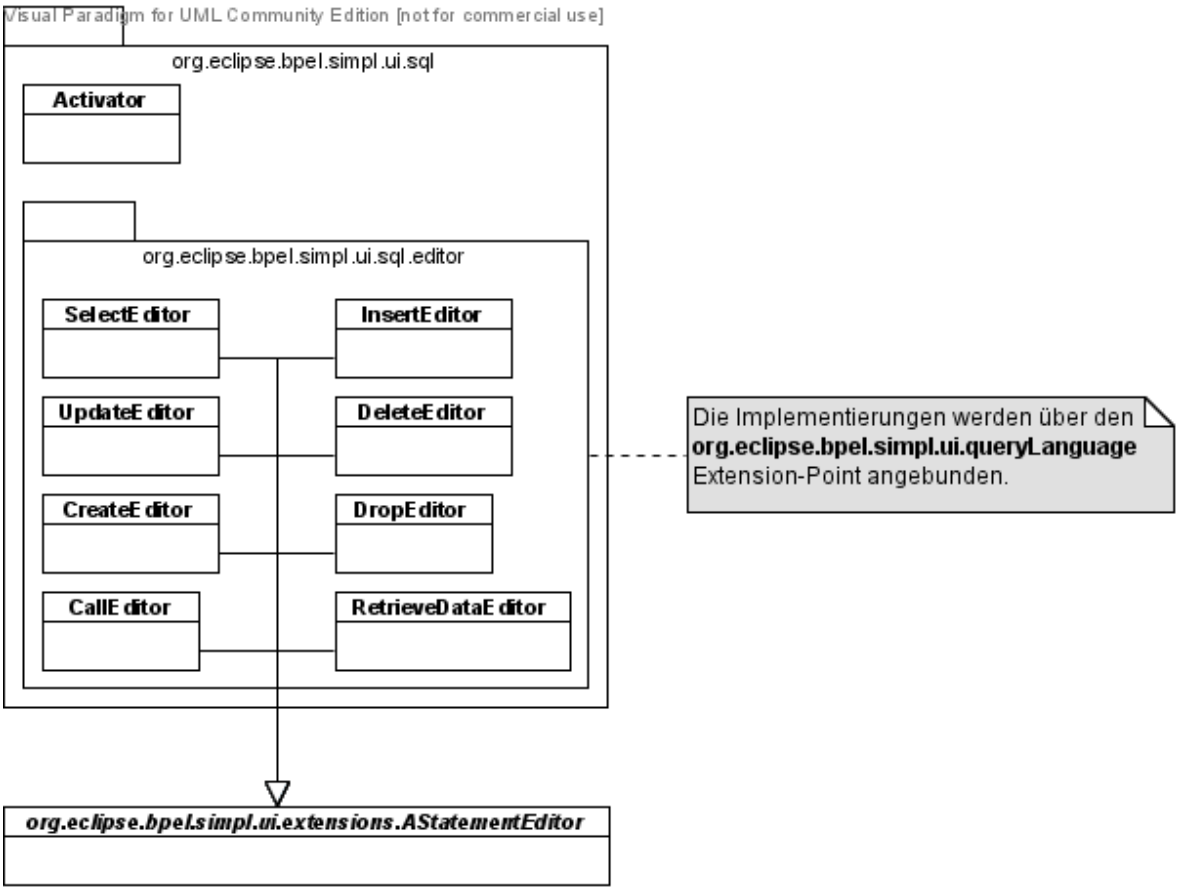


Abbildung 10: BPEL DM Plug-In Extension Modell

Das Paket org.eclipse.bpel.simpl.ui.sql steht beispielhaft für eine Erweiterung um um eine Benutzeroberfläche für die Anfragesprache SQL. Enthalten ist ein Aktivator sowie die Auswahlmöglichkeiten für den Statement Editor org.eclipse.bpel.simpl.ui.extensions.AStatementEditor. Jede Erweiterung kann am Extension Point org.eclipse.bpel.simpl.ui.queryLanguage angeschlossen werden.

4.2 SIMPL Core Plug-In

Das SIMPL Core Plug-In besteht aus dem Paket Admin-Konsole und SIMPL Core Client. Jede Kommunikation des SIMPL Core Plug-In mit dem SIMPL Core wird durch SIMPL Core Client realisiert.

Admin-Konsole

Durch diese grafischen Benutzeroberfläche wird SIMPL verwaltet. Die Admin-Konsole besteht nur aus Extension Points um eine größtmögliche Flexibilität hinsichtlich der späteren Nutzung zu erreichen. Das bedeutet die Einträge die bereits bei der Auslieferung von SIMPL in der Admin-Konsole vorhanden sind, wurden auch durch Extension Points realisert und können gegebenfalls leicht ausgetauscht werden.

Weitere Einträge können am Extension Points ACItem hinzugefügt werden. Bei der Auslieferung sind die Funktionen Auditing und Global Settings bereits durch Extension Points eingebunden.

Pakete, Klassen und wichtige Methoden

org.eclipse.simpl.core.globalSettings

Beinhaltet die "Activator"-Klasse mit den Methoden :
stop(BundleContext context)
start(BundleContext context)
Activator getDefault()
Activator()

org.eclipse.simpl.core.globalSettings.ui

Beinhaltet die Klasse "AuthentificationComposite" mit den Methoden :
saveSettingsToBuffer()
saveSettings(String parentItem, String item, String settingName)
loadSettingsFromBuffer()
loadSettings(String parentItem, String item, String settingName)
haveSettingsChanged()
createComposite(Composite composite)

org.eclipse.simpl.core.auditing

Beinhaltet die Klasse "Activator" mit den Methoden :
Activator getDefault()
Activator()
start(BundleContext context)
stop(BundleContext context)

org.eclipse.simpl.core.auditing.ui

Beinhaltet die Klasse "AuditingGeneralComposite" mit den Methoden :
loadSettings(String parentItem, String item, String settingName)
loadSettingsFromBuffer()
saveSettings(String parentItem, String item, String settingName)
saveSettingsToBuffer()

4.3 RRS Plug-In

Mit dem RRS Plug-In wird die Verbindung zum Reference Resolution System (RRS) geschaffen. Dadurch können Referenzen verwaltet werden und falls Referenzen bei der Modellierung verwendet wurden, eine Modelltransformation vor dem Deployment durchgeführt werden. Dazu benutzt das Plug-In den RRSAdministrationService und den RRSTransformationService. Die Verwaltung geschieht dabei über den Storage Service vom SIMPL Core. Weitere Details werden in der nächsten Iteration folgen.

4.4 SIMPL Core Client Plug-In

Das SIMPL Core Client Plug-In stellt die Verbindung zu den SIMPL Core Web Services her und bietet den anderen Eclipse Plug-Ins somit die Möglichkeit diese zu verwenden. Die Funktionalität für den Zugriff auf die Web Services wird mit Hilfe des Befehls wsimport (..\Java\jdk1.6.0_14\bin\wimport.exe) über die WSDL-Schnittstellen generiert und wird um die Serialisierung und Deserialisierung der komplexen Parameter erweitert.

5 Kommunikation

In diesem Kapitel wird die Kommunikation zwischen den Komponenten des SIMPL Rahmenwerks beschrieben und wichtige Abläufe deutlich gemacht.

5.1 SIMPL Rahmenwerk

In Abbildung 11 wird die Kommunikation zwischen den Komponenten mit entsprechenden Funktionsaufrufen gezeigt. Über das SIMPL Core Client Plug-In wird die Kommunikation der anderen SIMPL Eclipse Plug-Ins zum SIMPL Core hergestellt. Über die Web Services des SIMPL Cores werden Metdaten zu Datenquellen angefordert (11, 21: getDatasourceMetaData) und Einstellungen gespeichert (1: saveSettings) und geladen (1: loadSettings). Dazu werden von den Web Services die Dienste des SIMPL Cores verwendet und die Anfragen entsprechend weitergeleitet. Apache ODE kann die Dienste des SIMPL Cores direkt ansprechen, da sich der SIMPL Core im Classpath von Apache ODE befindet. Dort werden die DM-Aktivitäten (DM-Activities) über den DatasourceService ausgeführt, wozu die drei Methoden manipulateData, defineData und queryData (19, 18, 14) zur Verfügung stehen, die in Kapitel 2.2.3 bereits beschrieben wurden. Für das SIMPL Auditing benötigen die SIMPL DAOs ebenfalls Zugriff auf den DatasourceService, um die Auditing Daten zu speicher. Die Auditing Daten entstehen unter anderem bei der Ausführung der DM-Aktivitäten (13: auditing) und lösen eine Speicherung über die SIMPL DAOs aus (16: saveData).

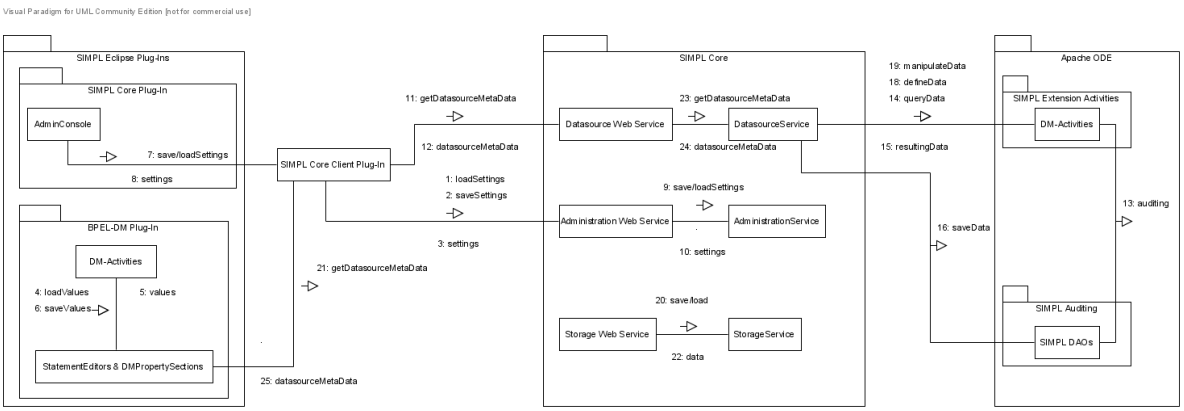


Abbildung 11: Kommunikation im SIMPL Rahmenwerk

Literatur

[1] *Grobentwurf v1.5.* Stupro-A SIMPL (2009)

Abkürzungsverzeichnis

API	Application Programming Interface
BPEL	Business Process Execution Language
DAO	Data Access Object
DM	Data-Management
GUI	Graphical User Interface
JAX-WS	Java API for XML - Web Services
ODE	Orchestration Director Engine
SDO	Service Data Object
SIMPL	SimTech: Information Management, Processes and Languages
SQL	Structured Query Language
UML	Unified Modeling Language
WS	Web Service

Abbildungsverzeichnis