

# Feinentwurf

Version 0.1

11. Januar 2010

---



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Zweck dieses Dokuments . . . . .	3
1.2	Gliederung . . . . .	3
<b>2</b>	<b>SIMPL Core</b>	<b>3</b>
2.1	Paketstruktur . . . . .	4
2.2	SIMPL Core Services . . . . .	5
2.3	SIMPLCore . . . . .	7
2.4	Datasource Plug-Ins . . . . .	7
2.5	Web Services . . . . .	7
<b>3</b>	<b>Apache ODE</b>	<b>8</b>
3.1	SIMPL DM Extension Activities . . . . .	8
3.2	SIMPL Event System . . . . .	9
3.3	Ausführung einer SIMPL DM Extension Activity . . . . .	9
3.4	SIMPL DAO . . . . .	12
<b>4</b>	<b>Eclipse</b>	<b>13</b>
4.1	BPEL DM Plug-In . . . . .	13
4.2	SIMPL Core Plug-In . . . . .	13
4.3	SIMPL Core Client . . . . .	13
<b>5</b>	<b>Kommunikation</b>	<b>13</b>
5.1	SIMPL Rahmenwerk . . . . .	13
5.2	Speicherung von Einstellungen über die Admin-Konsole . . . . .	13
	<b>Literaturverzeichnis</b>	<b>15</b>
	<b>Abkürzungsverzeichnis</b>	<b>16</b>
	<b>Abbildungsverzeichnis</b>	<b>17</b>

# Änderungsgeschichte

Version	Datum	Autor	Änderungen
0.1	13.11.2009	zoabifs	Erstellung des Dokuments.
0.2	04.01.2010	schneimi	Überarbeitung der Struktur
0.3	09.01.2010	schneimi	SIMPL Core Beschreibung

## 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 nur auf die aktuelle Iteration und wird mit den folgenden Iterationen 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 DM-Aktivitäten und das externe Auditing. (siehe[1] Kapitel 3.2)
- Kapitel 4 “Eclipse Plug-Ins” beschreibt die Implementierung der 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, der in den folgenden Abschnitten beschrieben wird.

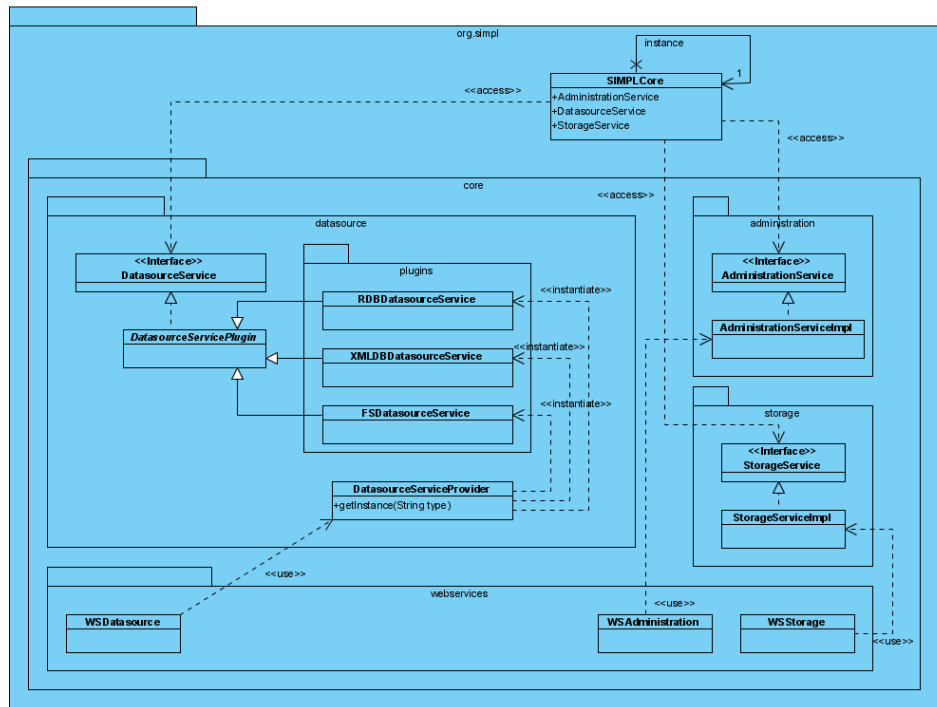


Abbildung 1: SIMPL Core Klassendiagramm

## 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 beschrieben.

### Administration Service

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

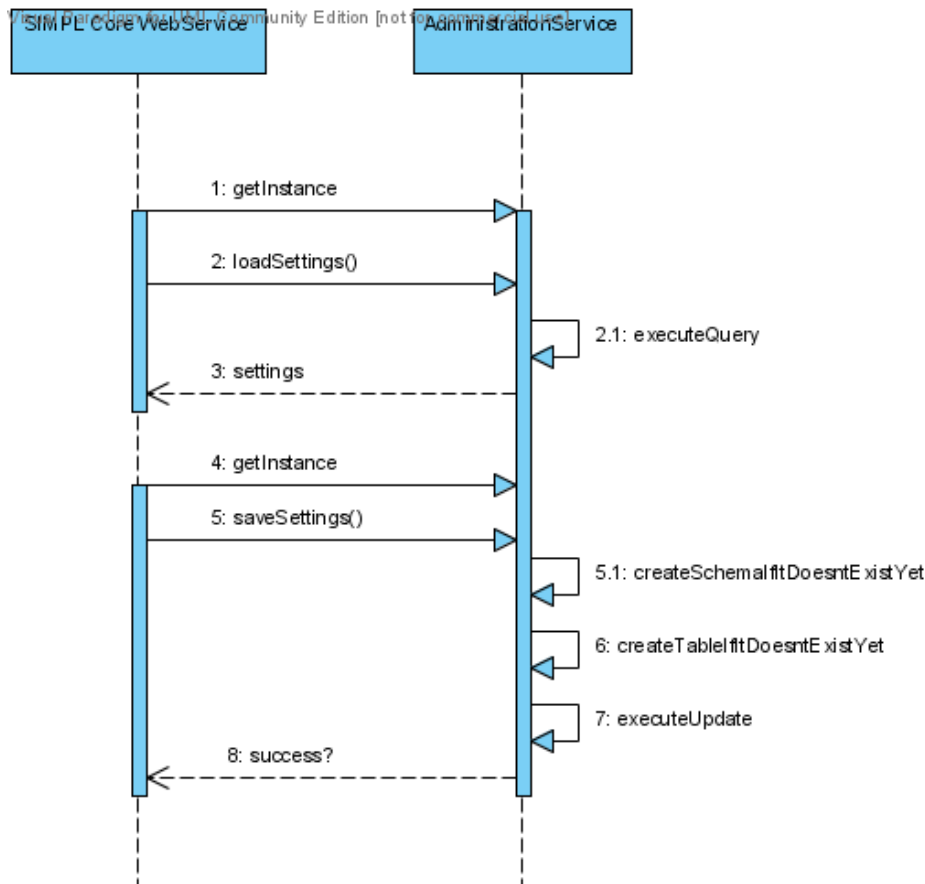


Abbildung 2: Sequenzdiagramm des AdministrationServices

Abbildung 2 zeigt die Verwendung und die Funktionalität des AdministrationService. Über die getInstance()-Methode erhält man eine Instanz des AdministrationServices mit der man dessen Funk-

tionen aufrufen kann. Mit der `loadSettings()`-Methode können Einstellungen aus der Datenbank geladen werden, dafür wird intern eine einfache Datenbankabfrage genutzt. 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 Oberpunktes, wie z.B. *General* eine gleichnamige Tabelle im Schema des Oberpunktes. 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.

## Storage Service

Der Storage Service ist für Daten eines beliebigen Services des SIMPL Cores zuständig. Der Storage Service verwaltet in einer embedded Datenbank alle Daten, die von dem Service des SIMPL Core geleitet. Und das Service des SIMPL Core kann später die Daten wieder auslesen.

Das Storage Service verhältet aenliche wie Administration Service, ein beliebiger Service des SIMPL Core benutzt die Methoden `save()` und `load()` der Storage Service Objekt, um die Data zu speichern und wieder auszulesen, muessen zuerst durch Methode `getInstance()` eine Instance erhalten.

Das Storage Service speichert die Data durch die Parametern `serviceNS`, `serviceName`, und `data` als Referenzen, wobei `serviceNS` als schemaname `serviceName` als tablename mit String Data sind. Data ist die Dateien, was man speichern moechte, durch eine `List<LinkedHashMap<String, String>>` Objekt zu organisieren. Nachdem das Service verhältet, wird die eine boolean Value zurueckgeben, um mitzuteilen, ob der Verhalten richtig durchgefuehrt wird.

Wenn das andere Service die Data durch die Storage Service auslesen moechte, braucht nur die `serviceNS` und die `serviceName` als String Data als die Parametern von der Methode `load()` eingeben. Nachdem das Service durchgefuehrt wird, die Data, die man gewuenscht werden, werden als eine `List<LinkedHashMap<String, String>>` Objekte zurueckgegeben. Wenn die Methoden `saveSettings()` und `loadSettings()` ueber den Storage Service die Data speichern und laden moechte, muss die zuerst durch die Private Methode `getConnection()` eine Connection mit der Datenbank herstellen, spaeter haengen die beiden Methode von Private Methode `getStatement()` ab, um die gewuenschte Datenbankopration zu erhalten. Nachdem die Methode fertig verhältet, wird die Methode die Operationsatz von Datenbank als eine String Objekt zurueckgeben. Wobei die Schema und Table sowie `settingName` als String Data `settings` als `List<LinkedHashMap<String, String>>` durch Parametern eingegeben werden. Hier ist die erste Element von `LinkedHashMap<>` Objekt als Primarykey gestellt. Es gibt noch Methoden `createSchemaIfItDoesntExistYet()` und `createTableIfItDoesntExistYet()`, `saveSettings()` und `loadSettings()` benutzen die Beiden, um zu bestaetigen, ob es die Schema oder Table in der Datenbank vorhandenen ist, wenn nein, dann herstellen, wenn ja, dann benutzt die vorhandenen Schema und Table. Es gibt auch die Methode `settingsAlreadyExist()`, der ist zustaeendig fuer Methode `getStatement()`, da fuer kann `getStatement()` wissen, soll Operation fuer „updaten“ oder „insert“ sein.

## Datasource Service

### 2.3 SIMPLCore

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, dort wo 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 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.

**Datasource Web Service** Der Datasource Web Service...

```
public DataObject queryData(String dsAddress, String statement, String dsType)

public boolean defineData(String dsAddress, String statement)

public boolean manipulateData(String dsAddress, String statement, DataObject data)

public List<String> getDatasourceTypes()

public List<String> getDatasourceSubTypes(String datasourceType)
```

```
public List<String> getDatasourceLanguages(String datasourceSubType)
```

```
public List<String> getDatasourceMetaData(String dsAddress, String dsType)
```

**Administration Web Service** Der Administration Web Service...

```
public boolean saveSettings(String schema, String table, String settingName, Linked-  
HashMap<String, String> settings)
```

```
public LinkedHashMap<String, String> loadSettings(String schema, String table, String  
settingName)
```

**Storage Web Service** Der Storage Web Service...

```
public boolean save(String serviceNS, String serviceName, List<LinkedHashMap<String,  
String>> data)
```

```
public List<LinkedHashMap<String, String>> load(String serviceNS, String ser-  
viceName)
```

## 3 Apache ODE

### 3.1 SIMPL DM Extension Activities

Die SIMPL DM Extension Activities (siehe Abbildung 3) haben als Hauptklasse die Klasse SIMPLActivity, welche verschiedene Funktionalitäten für alle weiteren Events anbietet. Die Extension Activities nutzen zur Ausführung der verschiedenen Datamanagement-Operationen den Datasource-Service des SIMPLCore. 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 geparkt 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. Dies 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.



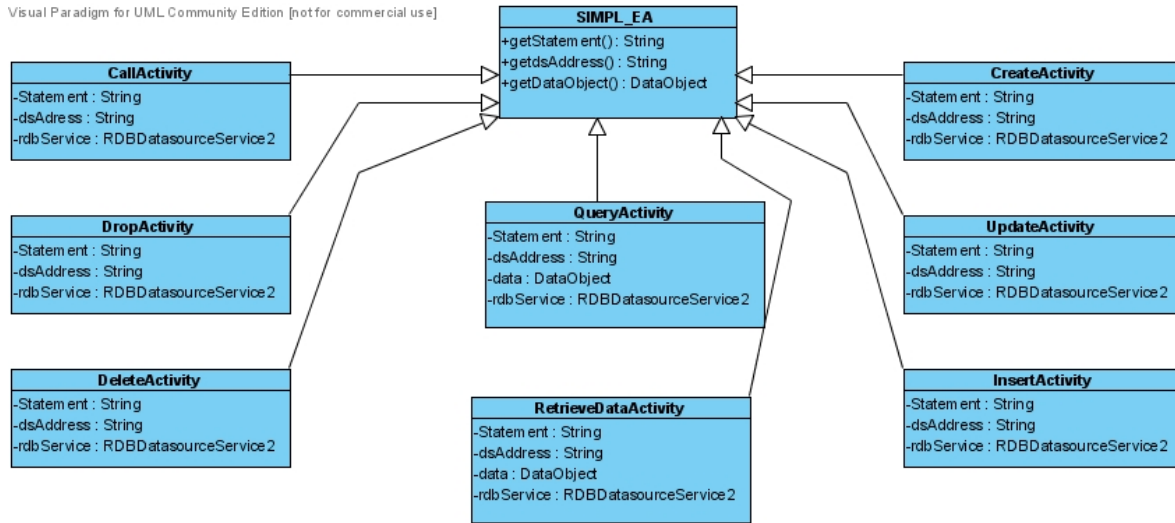


Abbildung 3: SIMPL DM Extension Activities

### 3.2 SIMPL Event System

Für die SIMPL Extension Activities wird eine Reihe von neuen Events eingeführt. Die Eventhierarchie ist in Abbildung 4 zu sehen. Die neuen Events unterteilen sich in DMEvents und ConnectionEvents, welche beide als Hauptklasse die Klasse SIMPL\_Event haben. SIMPL\_Event ist wiederum von Scope Event abgeleitet. Die neuen Events werden dadurch als Scope Events in die bestehende Event Hierarchie von ODE eingegliedert. Dies erlaubt es uns diese direkt innerhalb der Extension Activities zu nutzen und aufzurufen.

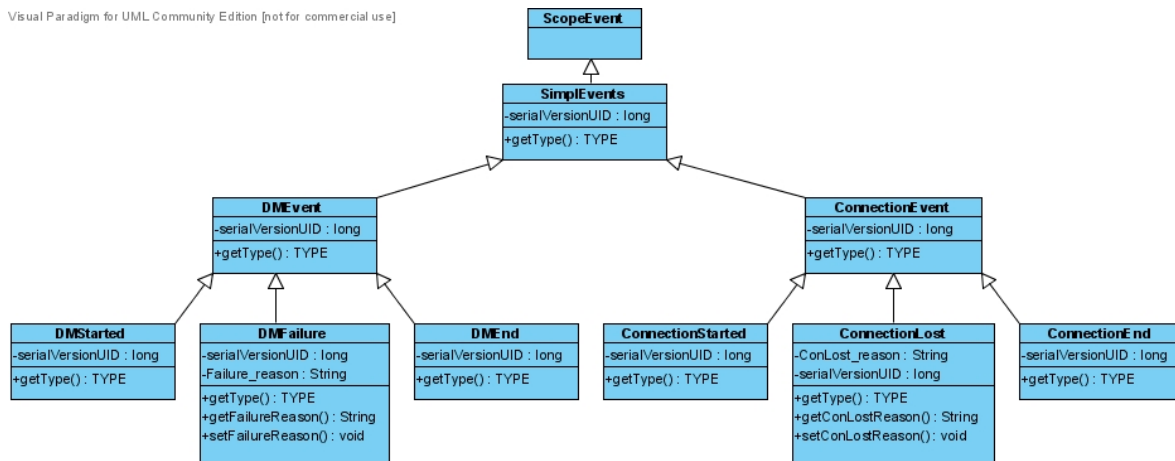


Abbildung 4: SIMPL Event System

### 3.3 Ausführung einer SIMPL DM Extension Activity

In Abbildung 5 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.

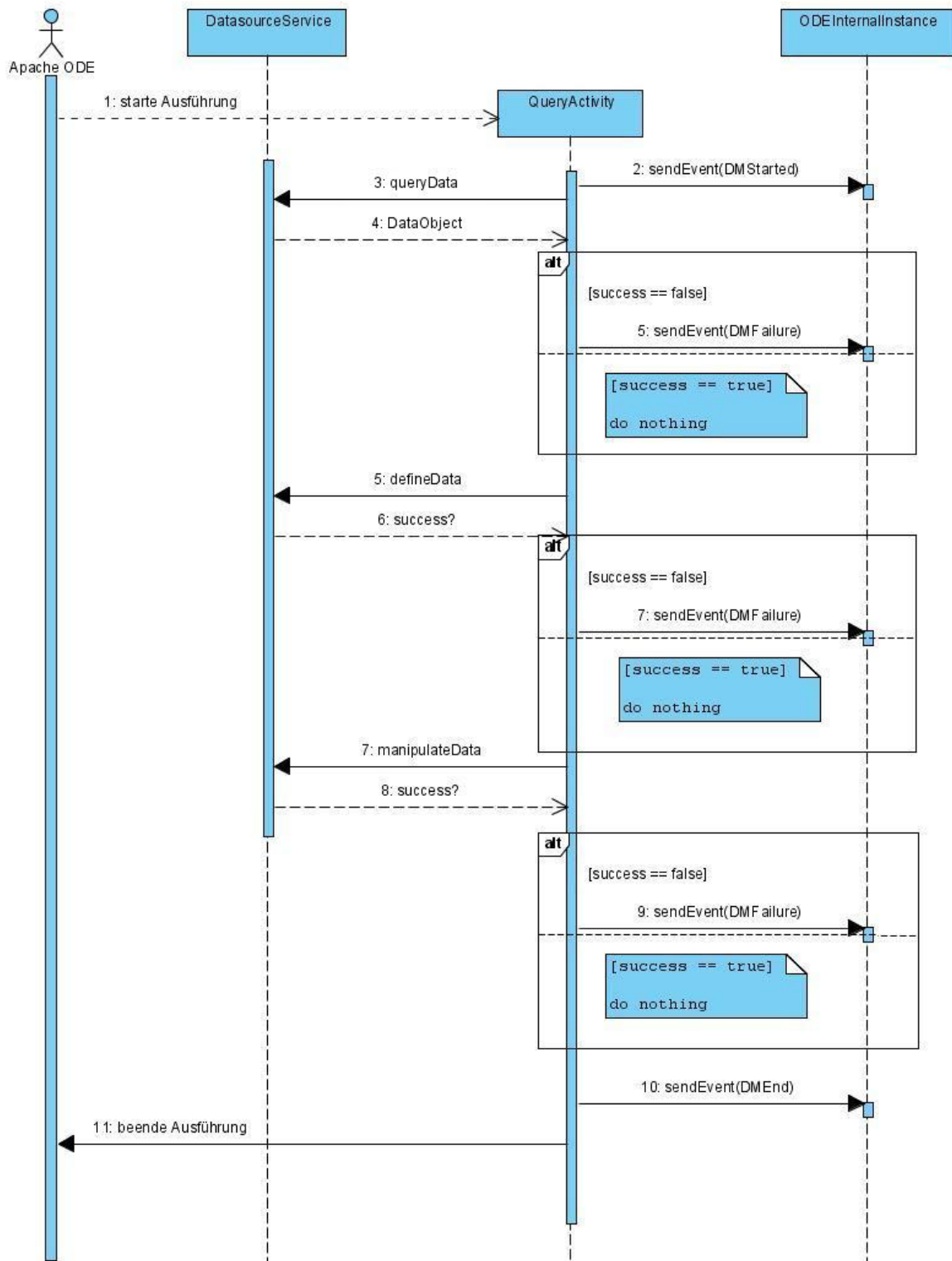


Abbildung 5: Ausführung einer SIMPL DM Extension Activity

### 3.4 SIMPL DAO

Das Simpl Dao besteht aus der Implementierung der Interfaces aus dem Paket `org.apache.ode.bpel.dao` die in den folgenden unterpunkten beschrieben werden. Das Simpl-Dao übernimmt alle Eigenschaften der Ode internen Jpa-Implementierung und erweitert diese um die Eigenschaft, Daten per SDO an den SIMPL-Core senden zu können. Das heißt, die Dao Daten werden 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.

#### **ActivityRecoveryDao**

Wird ausgeführt, wenn eine Aktivität den “recovery” Status einnimmt.

#### **BpelDAOConnectionFactory**

Die Factory verwaltet und erstellt die DaoConnection zu den angegebenen Datenquellen (standartmäßig der internen Derby Datenbank)

#### **BpelDAOConnection**

Stellt die Implementierung der Connection dar. Hier werden unter anderem BPEL Events in das Dao eingefügt und der Nachrichten austausch erstellt und verwaltet.

#### **CorrelationSetDAO**

Dieses DAO representiert ein BPEL correlation set.

#### **FaultDAO**

Wird benutzt um auf Informationen über einen Fehler zuzugreifen, der während einer Prozessausführung aufgetreten ist.

#### **MessageDAO**

Representiert eine Nachricht in der Datenbank.

#### **MessageExchangeDAO**

DAO für den Nachrichten austausch.

#### **MessageRouteDAO**

Das DAO representiert einen Nachrichten Anfrager, wie zum beispiel ein Pick oder ein receive.

#### **PartnerLinkDAO**

Das DAO representiert einen Partnerlink. Es enthält informationen über die eigene Rolle die Rolle des Partners und die Endpointreferenz.

#### **ProcessDAO**

Das DAO representiert einen laufenden Prozess. Es enthält die Prozess-Id, den Prozess-Typ, sowie die Prozessinstanzen.

## ProcessInstanceDAO

Dieses DAO repräsentiert eine Prozess-Instanz und enthält alle Daten die einer Instanz zugehörig sind. Dazu zählen Events, Scopes, sowie wartende pick und receive Aktivitäten.

## ScopeDAO

Dieses DAO repräsentiert eine Scope-Instanz. Es enthält eine ansammlung von Correlation-Sets und XML-Variablen.

## XmlDataDAO

Das DAO repräsentiert XML-Daten. Es wird dazu benutzt BPEL-Variablen zu modellieren.

# 4 Eclipse

## 4.1 BPEL DM Plug-In

## 4.2 SIMPL Core Plug-In

## 4.3 SIMPL Core Client

# 5 Kommunikation

In diesem Kapitel wird die Kommunikation im SIMPL Rahmenwerk beschrieben und wichtige Abläufe deutlich gemacht.

## 5.1 SIMPL Rahmenwerk

In Abbildung 6 wird die Kommunikation zwischen den Komponenten beschrieben und die entsprechenden Funktionsaufrufe gezeigt.

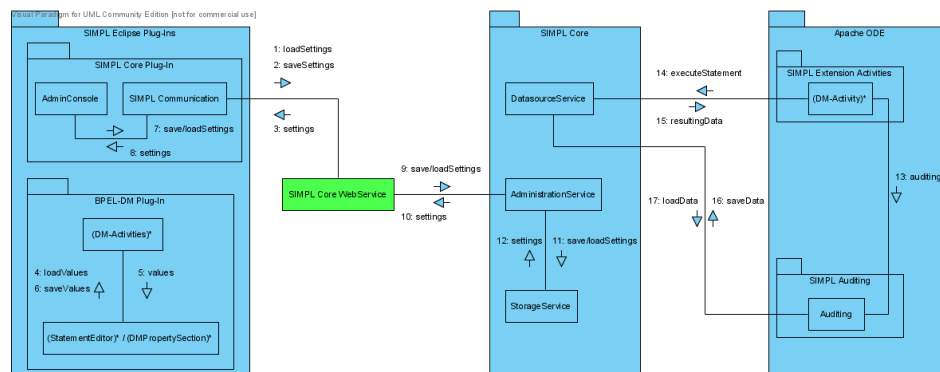


Abbildung 6: Kommunikation im SIMPL Rahmenwerk

## 5.2 Speicherung von Einstellungen über die Admin-Konsole

Abbildung 7 zeigt den Ablauf bei der Speicherung von Einstellungen der Adminkonsole und das Zusammenspiel der beteiligten Komponenten.

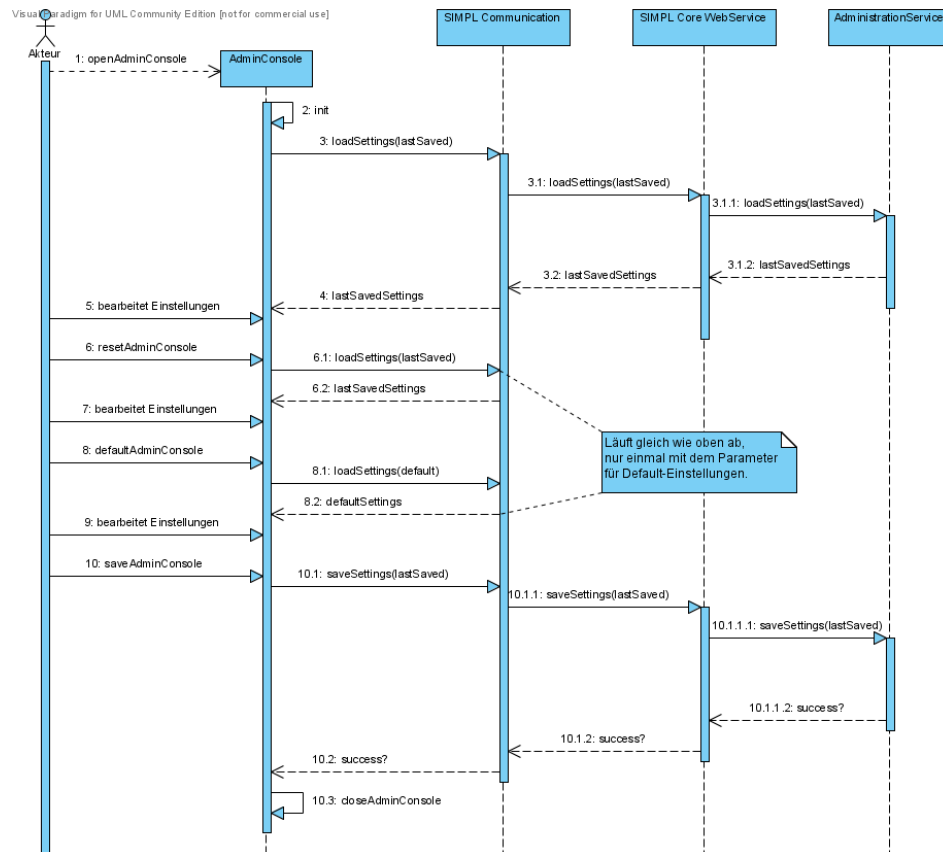


Abbildung 7: Speicherung von Einstellungen über die Adminkonsole

## Literatur

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

## Abkürzungsverzeichnis

UML	Unified Modeling Language



## Abbildungsverzeichnis

1	SIMPL Core Klassendiagramm . . . . .	4
2	Sequenzdiagramm des AdministrationServices . . . . .	5
3	SIMPL DM Extension Activities . . . . .	9
4	SIMPL Event System . . . . .	9
5	Ausführung einer SIMPL DM Extension Activity . . . . .	11
6	Kommunikation im SIMPL Rahmenwerk . . . . .	13
7	Speicherung von Einstellungen über die Adminkonsole . . . . .	14