Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3208

Extended Business Rules

Jian Liu

Studiengang : Informatik

Prüfer : Prof. Dr.-Ing. habil. Bernhard Mitschang

Betreuer : M.Sc. Florian Niedermann

begonnen am : 2. Mai 2011

beendet am : 1. Nov 2011

CR-Klassifikation : H4.1, H2.4, H3.3

Inhaltsverzeichnis

Abbildungsverzeichnis

Listing

Tabelle

1. Einleitung

Geschäftsprozesse sind die wichtige Bestandteile in der industriellen Praxis. Ein Geschäftsprozess definiert die Abläufen und Prozeduren im Unternehmen. Normalerweise besteht er aus eine Folge von Einzeltätigkeiten, die zum Erreichen eines geschäftliches oder betriebliches Ziel bei einem Arbeitsschritt ausgeführt werden. Er besitzt oft einen Beitrag für die Wertschöpfung eines Unternehmens. Idealerweise stellt das Ziel mit einem höheren Wert als das Ursprünglichen dar. Heutzutage werden die Produktzyklen immer kürzer und der Konkurrenzdruck unter den Firmen immer größer, deshalb wird die schneller Anpassung von der Geschäftsprozesse als die Konkurrenz immer wichtiger.

1.1. Motivation

In Geschäftsprozessen werden so genannt Business Rule verwendet, um zum Zeitpunkt die Ausführung des Geschäftsprozesses zu prüfen, ob bestimmte Regeln/ Integritätsbedingungen erfüllt sind und, je nach Ergebnis, unterschiedliche Folgeaktionen auszulösen. Beispiele für solche Regeln sind :

- In einem Produktionsprozess, kann eine Regel sein „Wenn die gesamte Produktionsmenge 20 Einheiten überschreitet, muss zusätzlicher Lagerplatz gebucht werden“

- In einem Versicherungsprozess kann eine Regel sein „Wenn das Gesamtrisiko des Versicherten 100 Punkte überschreitet, muss ein Manager befragt werden“

Normale Business Rules können sich dabei nur auf Daten beziehen, die im Prozessmodell selbst stehen (also z.B. in den BPEL Variablen). Dies reicht für viele Anwendungen aus, ist aber für andere Anwendungen zu sehr eingeschränkt. Wie in Abbildung 1 zu sehen, korrespondieren die Prozessdaten i.d.R. zu verschiedenen,

sogenannten operativen Daten. Dies sind Daten, die in einer anderen Datenquelle, z.B. in einem Data-Warehouse stehen. Diese Daten können für Business Rules ebenfalls relevant sein. Im Beispiel in Abbildung 1, dass aus einer Autovermietung stammt, könnte z.B. eine Regel sein „Wenn der Kunde noch keine Vermietungen hat (Rentals = 0) und ein Luxusauto mieten möchte, dann muss eine spezielle Prüfung durchgeführt werden“. Eine solche Regel kann nicht mit normalen Business Rules beschrieben werden, da sowohl die Rentals des Kunden als auch der Typ des Autos nicht in den Prozessdaten stehen. Dafür werden Extended Business Rules benötigt, welche neben der Beschreibung von Regeln auch die Möglichkeit bieten, zusätzliche (operative) Daten einbinden können.

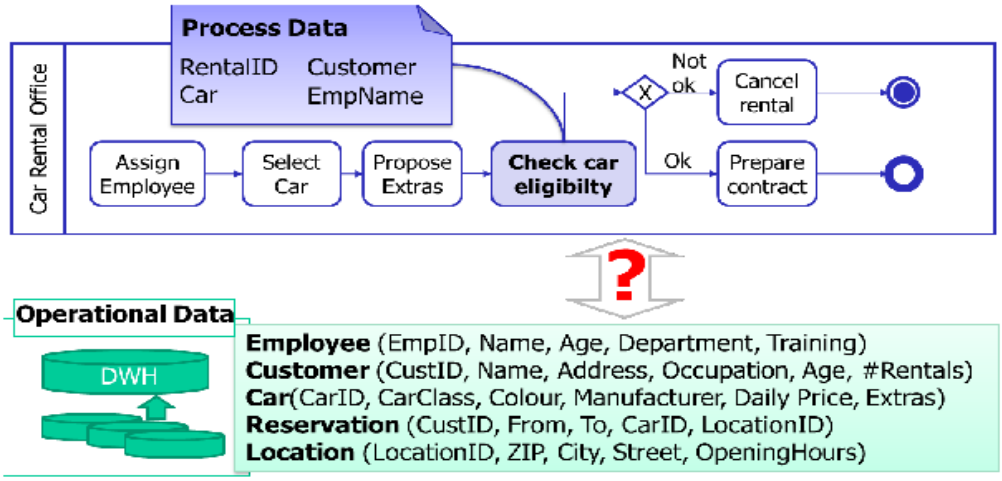


Abbildung 1: Prozessdaten und operative Daten

1.2. Aufgabe dieser Arbeit

Diese Arbeit hat zur Aufgabe, eine Komponente zur Anwendung der Extended Business Rules in den Geschäftsprozesse zu entwerfen.

Mittels der Verbindung mit der Datenbank kann die Komponente die opativen Daten zur Prüfung verwenden. Darüber hinaus muss eine Verbindung mit einem Geschäftsprozess eingerichtet werden, damit dieser Prozess auf die verschiedenen Resultaten reagieren kann. Schließlich wird ein Beispiel zum Testen genommen.

1.3. Aufbau der Arbeit

Diese Arbeit besteht aus sieben Kapiteln, welche im Weiteren folgendermaßen gegliedert sind:

- **Kapitel 2 – Grundlagen:** In diesem Kapitel werden die Grundlagen erläutert, die für das Verständnis dieser Arbeit notwendig sind (also z.B. BPEL, Business Rules, WS und XML).

- **Kapitel 3 – Konzept :** In diesem Kapitel wird die Konzept für den Extended Business Rule Manager gegeben.

- **Kapitel 4** – **Verwendete Technologie :** Die im Rahmen dieser Arbeit verwendete Technologie wird in diesem Kapitel beschrieben.

- **Kapitel 5** – **Implementierung:** In diesem Kapitel wird die Implementierung beschrieben und auch erklärt.

- **Kapitel 6 – Evaluation:** In diesem Kapitel wird der Extended Business Rule Manager ausgewertet.

- **Kapitel 7** – **Zusammenfassung und Ausblick:** In diesem Kapitel werden eine Zusammenfassung der Diplomarbeit und ein Ausblick über Verbesserungsmöglichkeiten gegeben.

2. Grundlagen

In diesem Kapitel sollen alle Grundlagen geklärt werden, die für das Verständnis dieser Arbeit notwendig sind. Kapitel 2.1 führt Business Rules ein und erklärt die wesentlichen Konzepte. In Kapitel 2.2 werden XML und XML-Schema kurz vorgestellt. Die Architektur von Web Service wird in Kapitel 2.3 gezeigt. Kapitel 2.4 beschreibt WS-BPEL, die für die Erzeugung eines Geschäftsprozesses relevant ist. Schließlich in Kaptiel 2.5 wird sogenannte DBOP Plattform vorgestellt.

2.1. Business Rules

Business Rules sind die Aussage in Geschäftsprozessen, die zur Kontrolierung der Struktur von Geschäftsprozessen angewendet werden. Sie beschreiben die Operationen, die Definitionen und die Beschränkungen, die in Geschäftsprozesse verwendet werden, um ein Ziel zu erreichen[DBR00].
Es gibt vier Kateorien von Business Rule:

- **Definitions of business terms**

  Die Sprache zur Definition von Business Rules ist das grundlegendste Element von Business Rules. Die Definition von Business Terms ist auch ein Business Rule, die die Sprache in einer bestimmten Anwendungsdomäne beschreibt (also z.B. Die Definition von Car und Customer in Abbildung 1). Deswegen führt die Definiton von Business Terms auch die Kategorie von Business Rules ein [DBR00].

- **Facts relating terms to each other**

  Die Struktur einer Organisation kann durch eine Reihe von Fakten beschrieben werden. Die Ordnung, die von den Kunden gesteuert wird, ist auch ein Business Rule. Fakten können als natürliche Sprachsätze oder als Beziehungen, Attribute und Verallgemeinerungsstrukturen in einem

graphischen Modell dokumentiert werden (also z.B. Ein Kunde mietet ein Auto) [DBR00].

- **Constraints**

  In der Geschäftsprozessen sind Constraints Bedingungen, die vom Wert einer Variablen erfüllt werden müssen. Die Aktion kann anhand des Wertes unterschiedlich reagieren (also z.B. Manche Daten werden modifiziert oder nicht)[DBR00].

- **Derivations**

  Die Weise, auf der das Wissen in einer Form in anderes Wissen möglicherweise in einer anderen Form verwandelt werden können (z.B. Die Inferenz oder die mathematische Kalkulation)[DBR00].

In dieser Arbeit wird Business Rule immer als dritte Kategorie betrachtet.

2.2. XML und XML-Schema

In dieser Arbeit werden Business Rules in XML-Schema definiert. Jedes Business Rule entspricht einer XML-Datei.

2.2.1. XML

XML("erweiterbare Auszeichnungssprache") ist eine Auszeichnungssprache, die von von W3C herausgegebene XML-Spezifikation definiert wird. Man verwendet XML, um die Daten, die hierarchisch strukturiert werden, in Form von Textdaten darzustellen. XML wird oft für den Austausch von Daten über das Internet angewerdet. Ein XML-Dokument besteht aus Elemente, die rekursiv umfassen oder parallel existieren[EML] . Listing 1 zeigt ein Beispiel eines XML-Dokuments.

```
<?xml version="1.0" encoding="UTF-8" ?>
<verzeichnis>
        <titel>Titel</titel>
        <eintrag>
            <stichwort>wort</stichwort>
            <eintragstext>text</eintragstext>
    </eintrag>
</verzeichnis>
```

Listing 1 : Ein Beispiel von XML-Dokument

Namen der Strukturelemente

Die Namen der Strukturelemente kann man belieblig auswählen. Die Strukturelemente (XML-Element) kann unterschiedliche Daten(nicht nur Text, sondern auch Tabelle und Grafik)umfassen und bescheiben. In XML-Anwendung werden die Daten und ihre Repräsentation getrennt bearbeitet. Das ist vorteilhaft für den Austauch von Daten zwischen zwei Computersysteme[EML] .

XML-Regel

- In einem XML-Dokument existiert genau ein Wurzelelement. Ein Wurzelelement ist das äußerste Element in der jeweiligen XML-Dokument[EML] .

- Alle Elemente mit Inhalt werden von einem Beginn- und einem End-Auszeichner repräsentiert. Wenn Elemente keine Inhalt besitzen, wird nur einem Auszeichner benötigt(z.B. <eintrag / >)[EML] .

- Alle Elemente müssen vor der End-Auszeichner des entsprechenden Elternelements oder der Beginn-Auszeichner eines Geschwisterelements geschlossen werden[EML] .

- Mehrere Attribute mit demselben Namen von einem Element sind verboten[EML] .

Aufbau eines XML-Dokuments

Es gibt einen physischen und einen logischen Aufbau in den XML-Dokumente.

```
<?xml version="1.0">
<!DocType book [
   <!ENTITY Titel "Gone with the Wind">   ⟹ Physischer Aufbau
]>
<book>
<name>&Titel; </name>                     ⟹ Logischer Aufbau
</book>
```

Listing 2 : Physischer und logischer Aufbau in einem XML-Dokument

Physischer Aufbau

- Entitäten: Die mögliche Entitäten sind die Hauptdatei des XML-Dokuments und Zeichenketten, die über Entitätenreferenzen eingebunden sind[EML] .

- Zur Spezifikation von XML-Version, Zeichenkodierung und Verarbeitbarkeit ohne Dokumenttypdefinition kann eine XML-Deklaration optional angewendet werden[EML] .

- Zur Spezifikation von Entitäten und dem erlaubten logischen Aufbau kann eine Dokumenttypdefinition optional verwendet werden[EML] .

Logischer Aufbau

Der logische Aufbau ist hierarchisch und ebenso wie eine Baumstruktur. Er besteht aus:

- Elemente mit einem Beginn- und einem End-Auszeichner oder mit nur einem Auszeichner.

- Attribute für zusätzliche Informationen über Elemente bei einem Beginn-Auszeicher oder einem selbst abgeschlossenen Auszeicher.

- Verarbeitungsanweisungen

- Kommentare

- Text, der sowohl als normaler Text als auch in Form von eines CDATA-Abschnittes auftreten kann.

In einem XML-Dokument existiert genau ein Element, das auf der obersten Ebene liegt. Die anderen Elemente werden von dem ersten Element enthält. Diese Elemente werden auch durch Angabe eines XML-Namensraum eindeutig sichergestellt. Das heißt, dass es keine Doppeldeutigkeiten gibt[EML] .

XML-Schema werden angewendet, um die Dokumenttypdefinitionen zu spezifizieren, damit keine Entitäten definiert werden können.

Programmgesteuerter Zugriff auf XML-Dokumente

Zum Einlesen oder Generierung von XML-Dokumenten werden XML-Prozessoren verwendet. Die XML-Prozessoren bestehen aus APIs und unterstützen drei grundlegende Verarbeitungsmodelle[EML] .

- DOM : Mit DOM-API wird XML-Dokument als Baumstruktur repräsentiert. Jede

Bestandteile der Baumstruktur kann beliebig einzeln zugreifen. Mit DOM-API Kann man nicht nur XML-Dokumente einlesen, sondern auch die Baumstruktur manipulieren. Außerdem wird das Zurückschreiben der Baumstruktur in ein XML-Dokument auch erlaubt[EML][DOM] .

- SAX : Mit SAX-API wird ein XML-Dokument als sequentiellen Datenstrom eingelesen. Das Verarbeitungsprinzip ist ähnlich wie das Konzept einer Pipeline[SAX].

- Pull-API : Daten werden mit XML-Pull-API sequenziell verarbeitet. XML-Pull-API ist hoch speichereffizient und leicht für Programmer als das SAX-API[EH].

In der Arbeit wird JDOM (ähnlich wie beim DOM) verwendet, um die XML-Dokumente zu bearbeiten.

2.2.2. XML-Schema

XML-Schema (XSD) wird zur Definition von Strukturen für XML-Dokumente verwendet. Datentypen und XML-Dokumente werden durch XML-Schema in einer komplexen Schemasprache beschrieben.

Datentypen

In XML-Schema gibt es einfachen Datentypen und Komplexen Datentypen. Einfache Datentypen sind die von XML-Schema angebotenen atomaren Datentypen, die zum Teil auch in anderen Typsystemen unterstützt werden[XS] .

Einfache Typen

- xs: string
- xs: decimal

- xs: integer
- xs: float
- xs: boolean
- xs: date
- xs: time
- QName
- anyURL
- language
- ID
- IDREF

Listing 3 : Einfache Typen

Darüber hinaus gehören Listen und Unions zu den einfachen Typen.

Komplexe Typen

Komplexe XML-Datentypen definieren die komplexe Elementenstrukuren, die weitere Elemente (Kindelemente) und Attribute beinhalten.

Die Kindelemente eines komplexen Typs können auf drei Weise kombiniert werden.

- xs: sequence : Beliebig viele Kindelemente können folgend aufgelisted werden. Jedes Element kann auch beliebig auftreten. Die Attributen minOccurs und maxOccurs zeigen die Anzahl des Auftretens für ein Element. Falls die beide Attributen nicht vorhanden sind, wird der Default-Wert 1 verwendet[XS] .
- xs: choice : Ein Element kann aus einer Liste von Elemente ausgewählt

werden.

```
<xs:complexType name="example">
        <xs:choice>
        <xs:element name="name_1" type="xs:string"/>
        <xs:element name="name_2" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
```

Listing 4 : Ein Beispiel von Komlexen Typ xs : choice

In diesem Beispiel enthält der Komplexe Typ „example“ entweder das Kindelement „name_1“ oder „name_2“.

- xs: all : ähnlich wie xs: sequence, aber die Anzahl des Auftretens von der Kindelemente betragt entweder 0 oder 1. Das heißt, die Attributen maxOccurs und minOccurs der Kinderelemente dürfen 0 oder 1 annehmen[XS] .

XML-Schema erlaubt die Benutzer, neuen Typen zu definieren. Aber in dieser Arbeit braucht man keine neuen Typen zu definieren, deswegen wird dieser Teil nicht erwähnt.

Verwendung von XML-Schema

Namensraum wird durch das Attribut „schemaLocation“ angegeben, sodass man das zu verwendeten XML-Schema finden kann. Falls kein Namesraum im XML-Dokument definiert ist, muss man das Attribut „noNamespaceSchemaLocation“ verwenden[XS] .

2.3. Web Services

Ein Web Service ist eine Softwareanwendung zur Realisierung der sogenannten Service-Oriented Architecture(SOA). Es unterstützt die Zusammenarbeit zwischen verschiedenen Anwendungsprogrammen in einem Netzwerk, die auf unterschiedlichen Plattformen laufen.

Die Service-Oriented Architecture (SOA) ist ein Architektursmuster, der sich mit der losen Kopplung und der dynamischen Bindung zwischen Services befasst. Mit dem sogenannten SOA-Dreieck wird das Basisprinzip erläutert

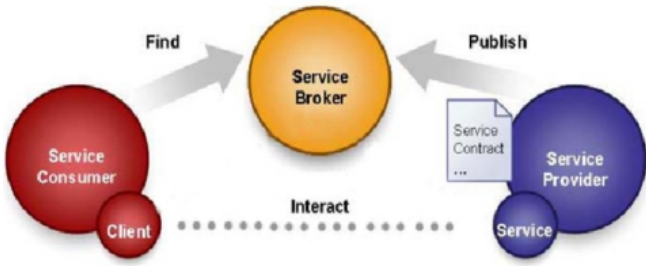


Abbildung 2 : SOA-Dreieck nach [BV08]

In dem SOA-Dreieck stehen der Service Provider, Broker und Consumer. Der Service Provider zeigt in dieser Architektur, dass die Dienste angeboten wird. Der Service Consumer zeigt, dass ein Benutzer einige Dienst finden möchte. Aber Am Anfangen weiß der Service Consumer nicht, wo er diese Dienste finden kann. Um diese Dienste zu finden, wird der sogenannter Service Broker angewendet. In diesem Bestandteil registriert der Service Provider seinen Dienste und speichert die Informationen von diesen Diensten , die diese Dienste beschreiben. Dann kann der Service Consumer hier die verfügbaren Dienste nachfragen und die Informationen für die Verbindung der Dienste erhalten[BV08] .

Die populärsten Arten von Diensten für SOA sind sogenannte Web Services.

Beschreibung von Web Services

Zur Beschreibung von Web Services kommt die Web Service Description Language(WSDL) zum Einsatz. Es handelt sich dabei um ein XML Format. Anhand dieser XML Format können die funktionalen Charakteristiken von Web Services beschrieben werden. Auf dieser Weiser können die Nachrichten zwischen Web Services ausgetauscht werden, die unabhängig von einem Nachtrichtenformat als auch Netzwerkprotokoll sind.

Services werden durch sieben XML-Hauptelemente definiert:

- **Types** – Definitionen der Datentypen und werden für die Definition der Messages Elemente benutzt.

- **Message** – Durch das Message Element wird Aufbau einer Nachricht beschrieben, die zwischen Services ausgetauscht werden kann.

- **Operation** – Zur Beschreibung einer Aktion von einem Service,Mit dem Operation Elemente werden auch die Nachtrichten gezeigt, die als Eingabe von einer Operation erwartet.

- **Port Type** – Eine Menge von abstrakten Arbeitsschritten und definiert alle unterstützten Aktionen eines Endpunktes.

- **Binding** – Bestimmt ein konkrete Übertragungsprotokoll und Datenformat. Im Falle des SOAP Nachrichtenformats hat man die Auswahl zwischen dem Remote Procedure Call (RPC) Style und dem Document Style.

- **Port** – Zur Definition von einem einzelnen Endpunkt (Binding und

Netzwerkadresse)

- **Service** – Fasst mehrere Port Elementen zusammen.

SOAP

SOAP ist ein Protokoll zur Beschreibung des Aufbaus von XML-basierter Nachrichten über ein Computernetzwerk in einer einfachen und erweiterbaren Weise, die zwischen Web Services ausgetauscht werden können. Eine SOAP Nachricht besteht aus einem SOAP Envelope, in der es ein SOAP Header und ein SOAP Body gibt. Der SOAP Header besteht aus Meta-Informationen, die zum Routing, zu Verschlüsslung oder zu Transaktionsidentifizierung verwendet werden. Dieser Element ist aber optional. Der SOAP Body dagegen ist notwendig. In diesem stehen die eigentlichen Nutzdaten der Nachricht [SOAP] .

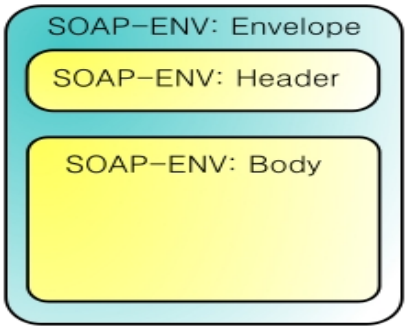


Abbildung 3 : SOAP-Struktur nach [SOAP]

2.4. WS-BPEL

Die WS-Business Process Execution Language (BPEL) ist eine Sprache zur Beschreibung von Geschäftsprozessen. Es bezieht sich auf eine erweiterbare Workflow-Sprache[WSBPEL]. Mittels dieser Sprache wird die Geschäftsprozesslogik

beschrieben, d.h. Die Reihenfolge und Ausführungsbedingungen der Aufrufe externer Web Services.

Mit BPEL ist es möglich,dass der graphenbasierten oder blockbasierten Arbeitsablauf erstellt wird. BPEL ermöglicht auch die rekursive Kombination von Web Services.

Ein BPEL-Prozess ist blockstruktuiert, d.h. Er besteht aus einer Reihe von lokalen Umgebungen (Scopes), die sich verschachteln. In einem Scope werden die Beziehungen zu externen Partner, die Deklarationen für die Datenverarbeitung, verschiedenen Handler und Aktivitäten beinhaltet. Der Scope, der in dem äußersten Gebiet liegt, bildet die Prozessdefinition eines BPEL-Prozesses[SOA] .

Durch BPEL können nur Kontrollflüsse abgebildet werden, deswegen Daten in BPEL durch Variablen gespeichert werden müssen, damit Daten übertragt werden können. Der Typ der Variable wird durch WSDL Message Typen, XML Schema Element oder XML Schema Typen definiert. Durch XML Schema Typen können sowohl simplen Typen als auch komplexen Typen definiert werden.Um die Daten, die von einer Variablen enthalten werden, modifizieren zu können kommt XPath zur Verwendung. XPath ist eine Ausdruckssprache,die die Konten in einem XML Baum adressieren können.

In einem BPEL-Prozess gibt es grundlegende Aktivitäten und strukturierte Aktivitäten. Die grundlegende Aktivitäten sind nicht aus anderen Aktivitäten aufgebaut. Dagegen können die strukturierte Aktivitäten andere Aktivitäten beinhalten[SOA] .

Basis Activities:

- **Invoke** – Mittels der invoke Aktivität können Web Services synchron oder asynchron aufgerufen werden, die von Partnern angeboten werden.

- **Assign** – Mittels der assign Aktivität können Daten entweder von einer Variable in eine andere Variable kopiert oder mittels eines Ausdrucks

aufgebaut und zu einer Variable hinzugefügt werden.

- **Receive** – Mittels der receive Aktivität kann eine Nachricht eines Partners empfangen werden, in dem sie selbst Operationen für die Partner bereitstellt. Diese Aktivität ist eine blockierende Aktivität, die solange wartet, bis eine Nachricht ankommt.

- **Reply –** Mittels der reply Aktivität kann eine Nachricht gesendet werden.

- **Pick** – Die pick Aktivität bietet eine nicht-deterministische Auswahl in einem Prozess an. Diese wird durch externe Ereignisse (Message, Zeitpunkt) entscheidet.

Structured Aktivities:

- **If** – Mittels der if Aktivität kann ein Prozess auf Bedingung basiert weiter laufen. Um diese Bedingung zu erstellen, kommt XPath zu Einsatz.

- **Sequence** – In der sequence werden die Aktivitäten sequentiell abgearbeitet.

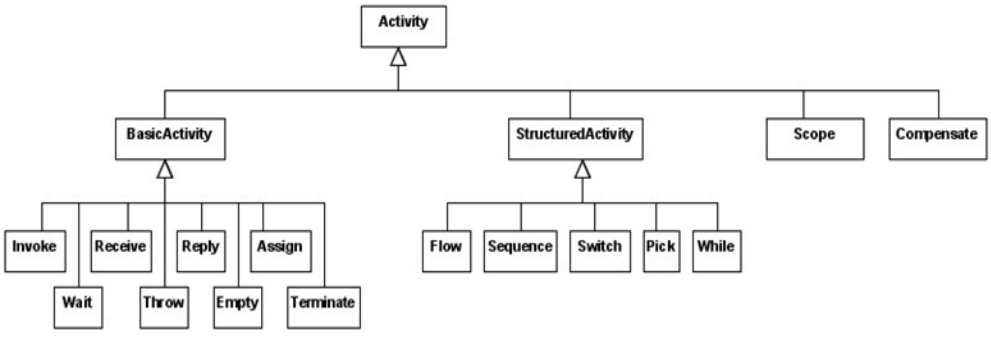


Abbildung 4 : Aktivitäten nach [SOA]

Partner Links

Die Interagierung von einem Prozess mit den Services werden in BPEL als Partner Links realisiert. Für ein Partner gibt es einen partnerLinkType. In den Partner Links können myRole und partnerRole definiert werden, von denen ein Partner Link mindestens einer dieser beiden Rollen definiert haben muss. Die eigene Rolle in einem Prozess wird durch myRole verdeutlicht und die Rolle des Partners durch partnerRole.

2.5. DBOP Plattform

Wenn ein Analytiker einen Prozess analysieren, versucht er so viele Daten in dem Prozess wie möglich zu finden, um den Mängel zu finden und dann eine Verbesserung durchzuführen. Es gibt viele Nachteile mit diesem Ansatz. Zuerst ist ein Analytiker nicht in der Lage, alle Verbesserungsmöglichkeit besonders in komplexen Prozessen zu identifizieren. Zweitens fehlt es eine Integrationsfähigkeiten von Daten, deswegen kann der Analytiker nicht alle für Verbesserung relevanten Daten erhalten. D.h. Mache Verbesserungsmöglichkeit können ignoriert werden. Drittens ist während des Entwurfs die Optimierung abhängig von statischen Modellen oder von Daten die durch Simulation generiert wurden. Die Erfahrungen, die durch Optimierung vorherigen Prozesse erhaltet werden, können nicht in dem neu erzeugten Prozess angewendet. Schließlich erfordert der Analytiker viel Zeit und Ressourcen, um alle Verbesserungsmöglichkeiten zu finden. Um diese Nachteile zu beseitigen, wird ein Deep Business Optimization Platform (dBOP) benötigt, die Optimierung während Entwurfs, Durchführung und Analyse von Prozesse unterstützt[NRM].

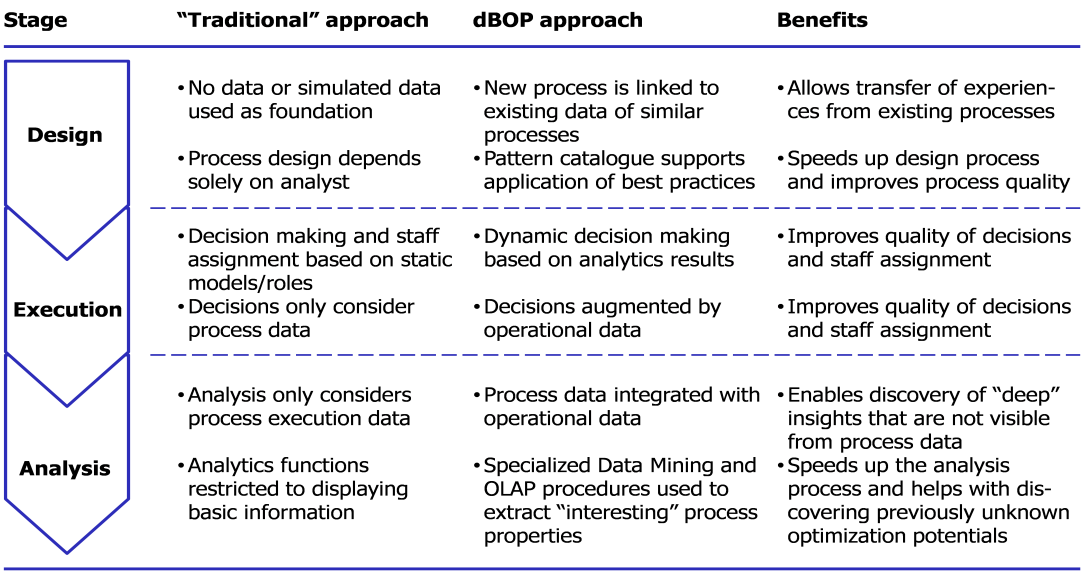


Abbildung 5 : Vorteile von DBOP im Vergleich zu traditionellen Ansatz nach[NRM]

DBOP besteht aus drei Schichten :

- **Data Integration** – Die für die Prozesse relevanten Daten können in verschiedenen Datenquellen verteilt. Diese Daten können entweder Prozess-Daten oder operationalen Daten sein. Prozess-Daten sind die in dem Geschäftsprozess übertragende Daten. Operationale Daten sind die anderen Daten, die in anderen Datenquelle gespeichert werden. In dieser Schicht wird die Daten, die für die Prozesse wichtig sind, integriert. Dies wird durch eine halbautomatische Anpassung realisiert [NRM].

- **Process Analytics –** Hier wird die für einen Prozess relevanten Daten aus der integrierten Daten-Schicht entnommen, damit die Optimierung sinnvoll ist. Die Resultat nach Analyse wird zur Optimierung von dem Prozess in sogenannten „Process Insight Repository" gespeichert. Darüber hinaus werden process matching capabilities und static process graph analysis methods in dieser Schicht umgefasst [NRM].

- **Process Optimization –** Die Optimierung von Prozesse wird anhand der Resultat nach Analyse mittels Optimierung-Muster durchgeführt [NRM].

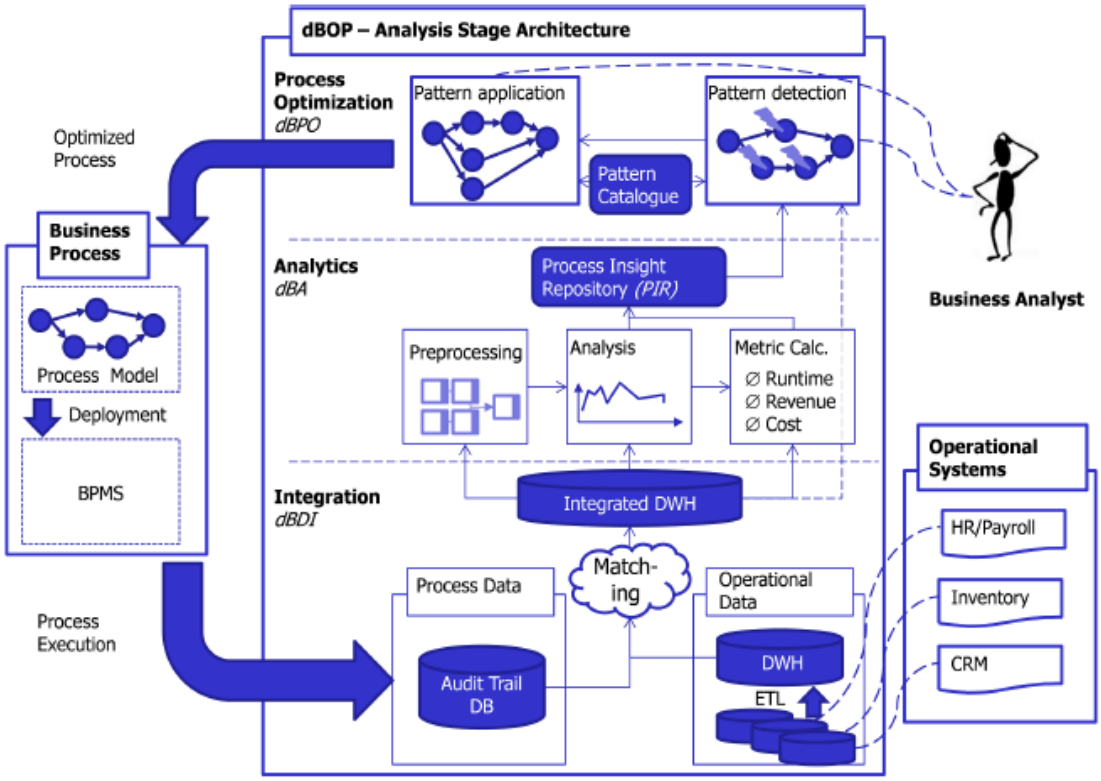


Abbildung 6 : deep Business Optimization Platform nach [NRM]

3. Konzept

In diesem Kapitel handelt sich es um den Entwurf des Extended Business Rule Managers und der Struktur der Regeln. Als erstes stellt Kapitel 3.1 die Architektur des Extended Business Rule Managers vor. Als nächstes wird in Kapitel 3.2 die Struktur der Regeln beschrieben.

3.1. Architektur

Bevor der Extended Business Rule Manager vorgestellt wird, ist es erforderlich, dass der Art der Anwendungen des Business Rule in BPEL-Prozesse zu verdeutlichen ist. Dazu wird ein einfaches Beispiel benutzt.

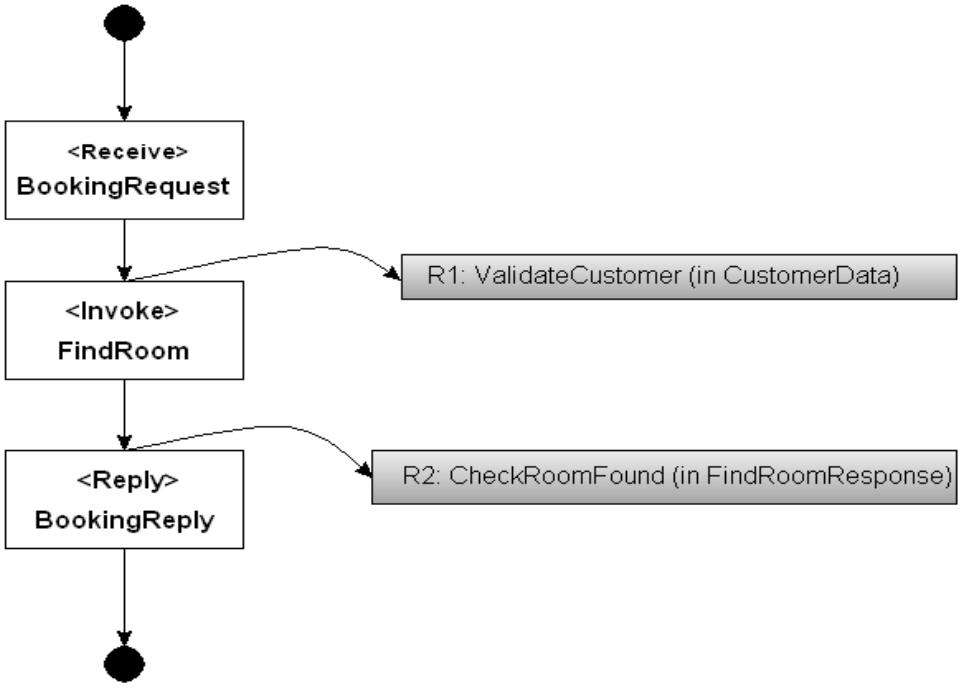


Abbildung 7 : Hotelbestellung Prozess

Dieser Prozess zeigt einen einfachen Vorgang von Hotelbestellung. Am Anfang kommt Eine Anfrage für Hotelbestellung an. Dann wird ein Web Service aufgerufen,

das ein freies Zimmer finden. Wenn ein Zimmer frei ist, wird Eine Antwort mit Zimmer-Nummer zurückgesendet. In diesem Prozess müssen zwei Bedingungen geprüft werden:Die Gültigkeit von Daten der Kunde vor die Aktivität <Invoke> und die Existenz von freien Zimmer vor der Aktivität <Reply>. Diese Bedingungen werden als die Regeln R1 und R2 dargestellt. Für die Anwendung von diesen Business Regeln in BPEl Engine ist eine Komponente erforderlich, die die Regeln mit BPEL Engine verbinden, damit die Regeln zur Laufzeit eines Prozesses geprüft werden. Diese Komponente muss in der Lage sein, die Parameter aus einem BPEL-Prozess zu empfangen, die in der Komponente integrierten Regeln mit den Parameter zu prüfen und schließlich die entsprechende Resultat zu BPEL-Prozesse zurückzugeben. Dies ist prinzipiell auf zwei Arten möglich – entweder kann die BPEL-Engine selbst erweitert werden oder es wird eine externe Komponente verwendet. In dieser Arbeit wird diese Komponente als externe Komponente realisiert.

Als externe Komponente wird ein Web Service implementiert. Der Web Service greift auf die Regel zu. Der Web Service wird vor jeder Prozessaktivitäten aufgerufen.

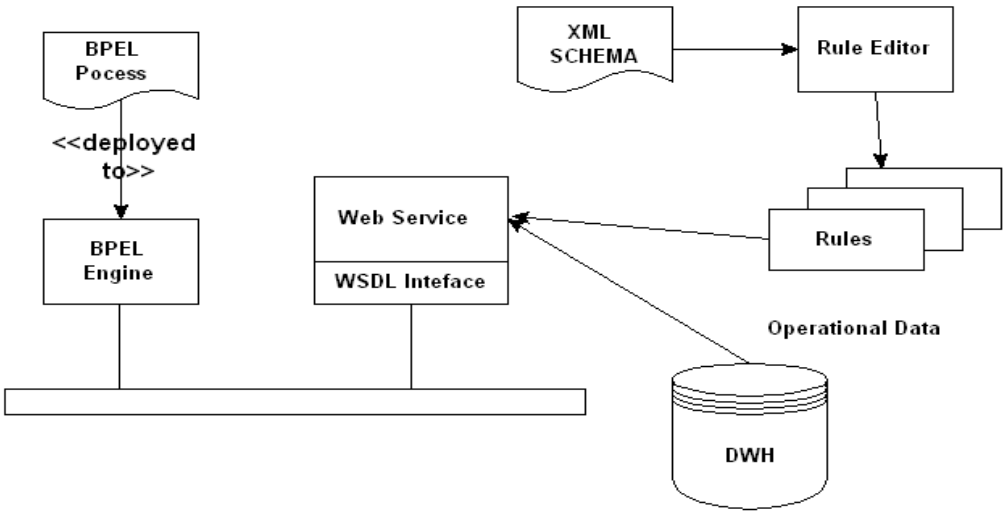


Abbildung 8 : Architektur des Extended Business Rule Managers

Abbildung 8 zeigt die Architektur des Extended Business Rule Managers. Ein Extended Business Rule Manager besteht aus drei Komponente: ein Web Service ein Regel-Editor und ein Data-Warehouse. Der Regel-Editor bietet den Benützern die

Fähigkeit an, die neuen Regeln nach einem XML-Schema zu erstellen. Diese erstellten Regeln werden von dem Web Service aufgerufen. Darüber hinaus stehen die in diesen Regeln definierten operativen Daten in einem Data-Warehouse, daher ist eine Verbindung zwischen den Web Service und einen Data-warehouse auch notwendig. Diese Verbindung wird durch Hibernate realisiert.

Für die Beschreibung der Business Rules wird eine spezielle Sprache benötigt, die sowohl die Regeln selbst beschreiben kann als auch die erforderliche Verknüpfung mit operativen Daten. Für die Beschreibung der Sprachen bietet sich ein XML-Format an.

3.2. Struktur der Regeln

Ein Business Rule muss mindestens drei Teile enthalten: Daten, Bedingungen und Aktionen.

- Daten - In diesem Teil können nicht nur die von BPEL-Prozesse gesendeten Daten sondern auch die sogenannten operativen Daten stehen. Darüber hinaus werden alle für die Findung der Wert der operativen Daten relevanten Daten auch hier gezeigt (also z.B. Join_Process und Join_Operational).
- Bedingungen - In diesem Teil sollen die Bedingungen anhand von den in dem Regel definierten Daten erstellt werden(also z.B. Rental == 0) .
- **Aktionen -** In diesem Teil kann ein Verhalten selbst definiert werden. (z.B. wenn die Bedingung in R2 erfüllt ist. wird eine Zimmer-Nummer zurückgesendet.)

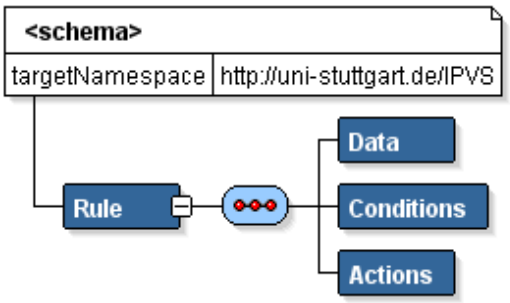


Abbildung 9 : Struktur der Regeln

3.3. Extender Rules und „normale“ Rules

Ein „normale“´Rule enthält nur die Prozessdaten. In Vergleich dazu stehen in einem Extenden Rule auch die Namen der benötigen operativen Daten. Die Werte der benötigen operativen Daten werden nach den in diesem Regel stehenden Prozessdaten in einem Data-Warehouse gesucht und in dem folgenden „Conditions“ Teil weiter benutzt.

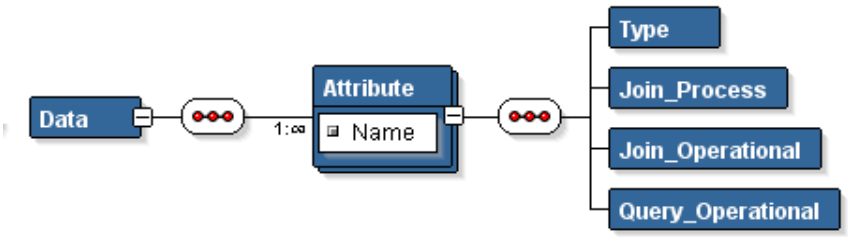


Abbildung 10 : Struktur des „Data“ Teiles in einem Extenden Business Rule

Das Element „Type“ zeigt die Type der bezüglichen Daten. Das Element „Join_Operational“ definiert die Prozessdaten. Das Element „Join_Process“ definiert den konkrete Ort in einem Data-Warehouse. Die beide sind zur Findung der Werte der operativen Daten „Query_Operational“ relevant.

4. Verwendete Technologien

In diesem Kapitel sollen die verwendete Technologien kurz vorgestellt werden,welche von Bedeutung für diese Arbeit sind.

4.1. Standard Widget Toolkit

SWT (Standard Widget Toolkit) bietet eine Bibliothek für die Erstellung von grafischer Oberflächen an, die in Java-Plattform angewendet wird. Um die grafischen Elemente zu erstellen, besucht SWT die nativen Bibliothek des Betriebssystems durch JNI (Java Native Interface). Diese ist ähnlich wie die Benutzung von spezifischen Betriebssystem-APIs für die Programmierung.

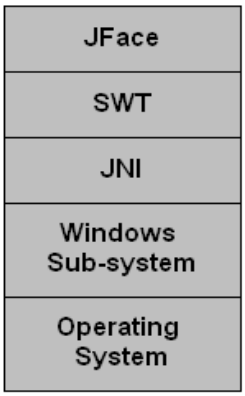


Abbildung 11 : Struktur von SWT

SWT ist ein in Java geschriebener Werkzeug und einzigartig für jede Plattform. In der „Windows“ Plattform kann SWT effizient angewendet. Aber in machen Java basierten Plattform ist SWT nicht effizient, weil SWT unterschiedliche native Bibliothek für unterschiedliche Plattform benutzt.

Verwendung in dieser Arbeit

Im Rahmen dieser Arbeit kommt SWT für Windows in der Version 3.6.1 zum Einsatz

und wird für die Bereitstellung einer grafischen Benutzeroberfläche zur Definition neuer Regeln verwendet.

4.2. JDOM

JDOM (Java Document Object Model) ist eine XML-Darstellung in Java und kommt als eine API zur Arbeit mit XML zum Ersatz. Bei JDOM wird Ein XML-Dokument als einen JDOM-Baum im Speicher repräsentiert. Jeder XML-Knoten darin wird durch eine spezifische Java-Klasse dargestellt. Es gibt zehn grundlegenden Java-Klassen zur Repräsentation des JDOM Dokuments, die in dem Paket org.jdom stehen. Hier werden nur die drei wichtigsten klassen kurz vorgestellt [CU] :

- **Document –** JDOM Dokumente werden durch die „Document“ Klasse erzeugt. Ein vollständiges JDOM Dokument fasst in der Regel „DocType“, „ProcessingInstruction“, einem Wurzelelement und „Comment“ um.

- **Element –** Jedes Element in dem Dokument wird durch die „Element“ Klasse abgebildet. Mit dem Wurzelelement gibt es die Möglichkeit, auf die anderen Elemente des Dokumentenbaums zuzugreifen.

- **Attribute –** Die Attribute von Elementen können durch diese Klasse erzeugt.

Um die XML-Dateien einlesen und auch ausgeben zu können, werden die Klassen XML Outputter und SAXBuilder benötigt, die in den Paketen org.jdom.output und org.jdom.input liegen.[CU]

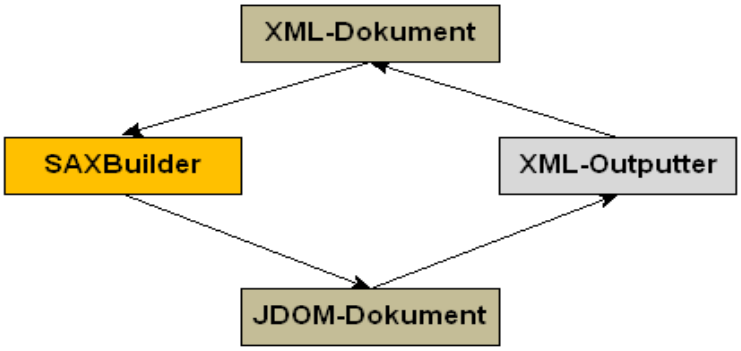


Abbildung 12 : Erzeugung von XML-Dokument und JDOM-Dokument

XML Outputter gibt ein JDOM Dokument in Form von Byte-Fluss aus. Dagegen erzeugt SAX Builder ein JDOM Dokument aus Dateien, Byte-Fluss oder URLs durch einen SAX Parser.

Verwendung in dieser Arbeit

Im Rahmen dieser Arbeit kommt JDOM in der Version 1.1.1 zum Einsatz und wird für die Erstellung und den Einlesen neuer Regeln verwendet.

4.3. Objektrelationales Mapping

Hibernate ist ein Objekte-Relationen-Mapper unter den open-source Systemen, der eine Verbindung zwischen einer Java-Anwendung und einer Datenbank auf der Basis von JDBC und SQL realisiert. Mittels Hibernate können die Objekte mit Attributen und Methoden in relationalen Datenbanken gespeichert und aus die Datensätzen erzeugt werden. Die Relationen zwischen Objekten können auch durch Datenbank realisiert werden.

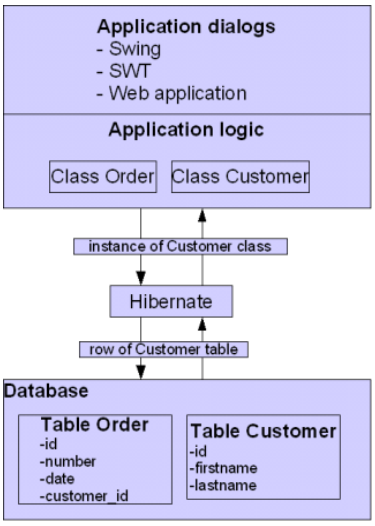


Abbildung 13 : Mapping zwischen Objekte und Realtion durch Hibernate[SH08]

Konfiguration

Hibernate muss mit einem Datenbankmanagementsystem (DBMS) verbinden. Dafür muss das vom Objekte-Relationen-Mapper verwendete JDBC benötigt werden : Den Treiber, die Verbindungs-URL und Authentifizierungsdaten. Hibernate kann nicht nur einen Art von DBMS verwenden, deswegen braucht man auch den SQL-Dialekt angeben, der zur Unterscheidung von verschiedenen DBMS verwendet wird. Die Konfiguration wird in der Datei hibernate.cfg.xml gespeichert. Ein Beispiel von hibernate.cfg.xml sieht so aus [BM06] :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
                "http://hibernate.sourceforge.net/hibernate-configuration- 3.0.dtd">

<hibernate-configuration>

<session-factory>

        <!-- Database connection settings -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
        <property name="connection.url">jdbc:mysql://localhost/hibernate</property>
        <property name="connection.username">root</property>
        <property name="connection.password">r</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.OracleDialect</property>

        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="hibernate.show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>

        <mapping resource="com/Example.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Listing 5 : Ein Beispiel von hibernate.cfg.xml

Objektrelationale Abbildung

Die Abbildung zwischen Objekte und Dantenbanktabellen wird mittels einer XML-Datei (*.hbm.xml) oder mit Java-Annotation realisiert. Dafür kommt vier Hibernate Tools zum Einsatz: „hbm2java“, „hbm2dll“, „Xdoclet“ und „Middlegen“. „Hbm2java“ kann eine Java-Klasse mittels einer XML-Datei automatisch generiert. „Hbm2dll“ bietet die Möglichkeit, eine Tabelle in Datenbank auch mittels einer XML-Datei automatisch zu erzeugen. „Middlegen“ kann eine XML-Datei nach der Tabelle in Datenbank generieren. Mit Hilfe von „XDoclet“ kann eine XML-Datei nach die entsprechende Java-Klasse automatisch erzeugt. Diese Hibernate Tools stehen in „hibernate-tools.jar“[HE].

Um Hibernate zu verwenden, sind drei Klassen wichtig : SessionFactory, Session und Transaction. SessionFactory wird einmal in einer Anwendung erzeugt und lädt

die Konfiguration und die Abbildung. Session kann die Java-Programme mit den Hibernate-Diensten verbinden, damit die Methoden für Insert-, Update-, Delete- und Query-Operationen in den Java-Programme angewendet werden. Transaction bildet JDBC-Transaktionen ab[HE].

Verwendung in dieser Arbeit

Im Rahmen dieser Arbeit kommt Hibernate in der Version 3.3.2 zum Einsatz und wird für die Abfrage von operativen Daten und auch die Erstellung von Logging-Tabelle in Oracle Datenbank verwendet.

4.4. Oracle Database 10g Express Edition

Oracle Database Express Edition ist eine relationale Datenbank. Sie bietet die Möglichkeit, die Daten zu speichern, zu benutzen oder sogar zu modifizieren. Dafür wird sogenannt JDBC (The Java Database Connectivity) zum Zugriff und zur Manipulierung von Daten verwendet.

Verbindung mit Oracle Datenbank

JDBC ist ein Datenbankzugangsprotokoll, mit der eine Verbindung mit Datenbank realisiert werden kann, damit die SQL-Sätze ausgeführt werden können und die Daten dann in Datenbank zuzugreifen sind. Die Java-Bibliothek liefert nur ein JDBC-API(java.sql), dagegen muss JDBC so entworfen werden, dass der Treiber die Spezialisierung für eine besondere Datenbank anbieten kann. Oracle Datenbank XE liefert solche Unterstützungen. Zur Entwicklung von Anwendungen für den Client kommen die Paketen „JDBC Thin Driver“, „Oracle Call Interface Driver“, „oracle.sql“ und „oracle.jdbc“ zum Einsatz. Diese Paketen erweitern das JDBC[2DPJDG07].

Oracle JDBC Thin Driver

„JDBC Thin Driver“ ist geeignet für die Web-basierte Anwendungen und Applets,weil es dynamisch von einer Webseite wie ein Java-Applet herunterladen werden können. Es wird in Java geschrieben und kann JDK 1.4.x und 1.5.x unterstützen. Außerdem ist „JDBC Thin Driver“ unabhängig von den Plattformen, daher wird keine zusätzliche Komponente dafür benötigt. „JDBC Thin Driver“ kommuniziert sich mit einem Service mittels sogenannten „SQL*Net“[2DPJDG07].

Oracle.sql

Das Paket „oracle.sql“ kann den direkter Zugriff von Daten in Form von SQL unterstützen. Dieses besteht aus Klassen, die eine Abbildung zwischen Java -Datentypen und SQL-Datentypen liefern. Z.B. Die Type „Charaktere“ wird in die Type „Chars“ in Java umgewandelt[2DPJDG07].

Oracle.jdbc

Die Schnittstellen in dem Paket „oracle.jdbc“ definieren die Erweiterungen für die Schnittstellen in dem Paket „java.sql“. Diese Erweiterung liefern Zugang zu Oracle SQL-Formatdaten und auch Zugang zu anderen Oracle spezifischen Merkmalen einschließlich Oracle Leistungsverbesserungen[2DPJDG07].

Verwendung in dieser Arbeit

Oracle Database 10g Express Edition kommt im Rahmen dieser Arbeit zum Einsatz. Alle von BPEL-Prozesse benötigen operativen Daten und auch die Informationen für die Anmeldung von Business Regeln werden in dieser Datenbank gespeichert.

4.5. Apache Axis

Apache Axis ist eine SOAP-Engine, mit der Web Services und Client-Anwendungen aufgebaut werden können. Axis bietet die Möglichkeit an, Web Services als auch auf

WSDL basierenden Clients aus reinen Java Klassen zu generieren. Zudem werden synchrone als auch asynchrone Aufrufen von Web Services in Axis unterstützt.

Axis unterstützt während der Entwicklung von Web Services den sogenannten Bottom-Up und Top-Down Ansatz.

- **Bottom-Up Ansatz :** Web Services werden aus der bereits erstellten Java Klassen (POJO) bereitgestellt. Axis generiert automatisch die WSDL-Dokumente als auch einen entsprechenden Client. Danach können diese verwendbaren Web Services in Apache Tomcat eingespielt werden [MT] .

- **Top-Down Ansatz :** In Gegensatz zum Bottom-Up Ansatz müssen zunächst die WDSL-Dokumente erzeugt. Daraus werden die Java Klassen generiert, in den die Methoden für die in den WSDL-Dokumente definierten Operationen von Web Services auch gleichzeitig erzeugt werden. Danach müssen diese Methoden auch mit Logik ausgefüllt werden [MT] .

Verwendung in dieser Arbeit

Mittels Axis, der in WTP (Web Tool Platform) Integriert wird, werden alle im Rahmen dieser Arbeit notwendigen Web Services nach dem Bottom-Up Ansatz entwickelt.

4.6. Apache ODE

Apache ODE (Apache Orchestration Director Engine) ist eine Laufzeitumgebung für das Ausführen von Geschäftsprozessen, die auf WS-BPEL basiert [ODE10].

Die ODE kann nicht nur die Ausführung von kurzlebigen Prozesse sondern auch von langlebige Prozesse unterstützen, die mit Web Services orchestriert werden. Mittels der ODE können die Interaktion mit Web Services, die Sendung und der Empfangen von SOAP-Nachrichten, die Datenmanipulationen und auch die Fehlerbehandlungen realisiert werden. Darüber hinaus unterstützt die ODE ein Hot-Deployment von WS-BPEL Paketen und auch bietet ein Management-API zum

Monitoren der laufenden Prozesse an[ODE10].

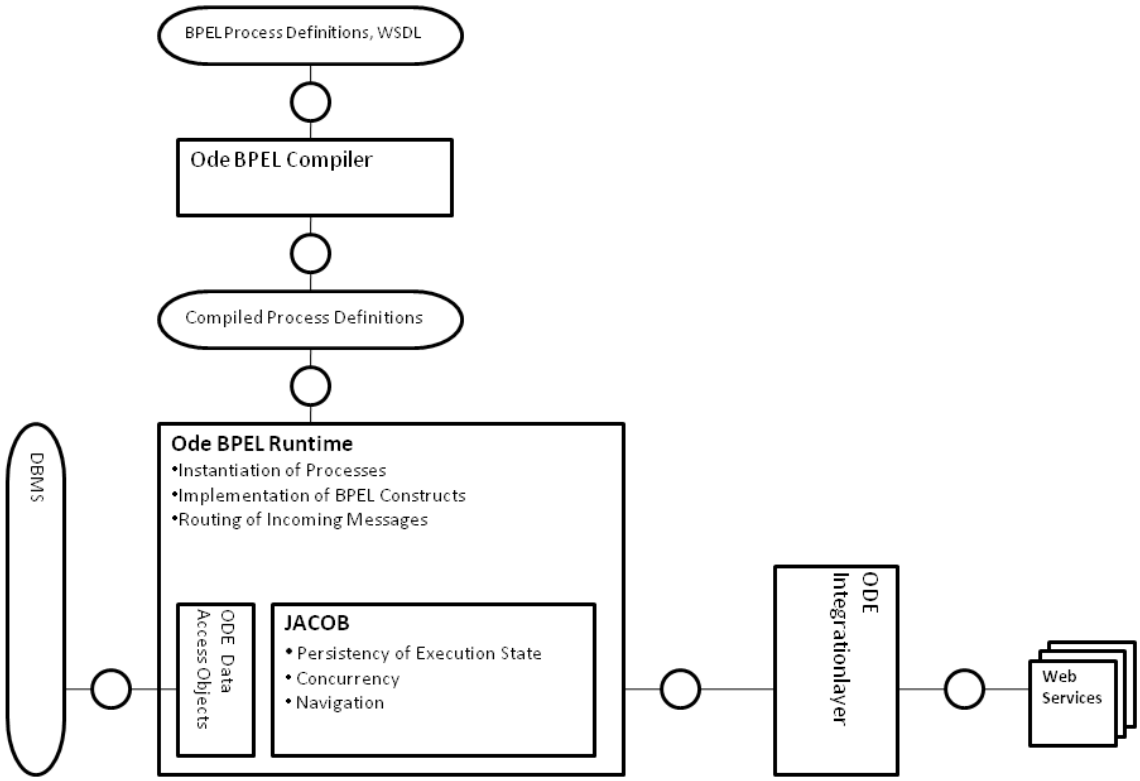


Abbildung 14 : Die Architektur von ODE[ODE]

Deployment von Prozessen in ODE

Um ein erfolgreiches Deployment ausführen zu können, werden einige relevanten Dateien benötigt, die in einem Ordner zusammengefasst werden : die Prozess-Definition, der Deployment Descriptor und die WSDL-Dateien. Beim Deployment Descriptor geht es um ein XML-Dokument mit dem Name „deploy.xml“. In diesem Arbeit werden alle Prozesse, die WSDL-Dateien für diese Prozesse als auch für Web Services in einem Projekt umfasst[ODE10].

Das Deployment von BPEL-Prozesse in ODE ist einfach. Für das Deployment wird nur der Ordner, in dem die relevanten Dateien liegen, in den /WEB-INF/processes

Ordner der ausgepackten WAR Distribution von ODE kopiert. Weil die ODE ein Hot-Deployment von WS-BPEL Paketen unterstützt, können die Prozesse in ODE zur Laufzeit verändert und dann geladen werden. Wenn das Deployment erfolgreich ausgeführt wird, können die richtigen Informationen in Konsole gezeigt werden. Wenn nicht , wird auch ein Hinweise über die möglichsten Fehlerursachen geliefert. Manchmal kann ein Fehler noch auftreten, wenn alle Dateien in dem Ordner richtig sind. Auf diesem Fall braucht man nur einen vorher erfolgreich eingesetzte Prozess nochmal in ODE aufzustellen. Danach kann der benötige Prozess erfolgreich deployt werden.

Verwendung in dieser Arbeit

Im Rahmen dieser Arbeit kommt die ODE in der Version 1.3.4 zum Einsatz und wird für das Ausführen von BPEL-Prozessen verwendet.

4.7. Apache Tomcat

Apache Tomcat ist eine Implementierung für den Java Servlet und Java Server Pages (JSP)Technologien und bietet eine Umgebungen für in die Java geschriebenen Web Services an. Er besteht aus dem Servlet-Container Catalina und dem Connector Coyote.

Verzeichnisstruktur

Das Verzeichnis CATALINA_HOME ist das Root-Verzeichnis von Apache Tomcat. Es zeigt den Ort, in dem der Tomcat installiert wird. Alle anderen statischen Verzeichnisse stehen in diesem Verzeichnis. Die nicht statischen Verzeichnisse, die zur Laufzeit entstehen, werden in dem CATALINA_BASE Verzeichnis abgelegt. Für mehrere unterschiedlich konfigurierte Instanzen in Tomcat gibt es auch mehrere unterschiedlichen CATALINA_BASE Verzeichnisse. Das garantierte, dass mehrere Instanzen nur einen Tomcat brauchen[TALT] .

Webanwendung

Eine Webanwendung ist eine Anwendung, die durch eine bestimmte Verzeichnis-Hierarchie und die entsprechenden Dateien definiert wird. Eine Webanwendung kann entweder direkt in Catalina verwendet werden oder in einem Web-Archiv (WAR) eingepackt werden. Das eingepackt Web-Archiv wird beim Deployment automatisch ausgepackt[TALT] .

Deployment von Webanwendungen

Um eine Webanwendung ausführen zu können, braucht man noch ein Servlet Container, in dem eine Webanwendung eingesetzt wird. Das Deployment von Webanwendungen in Tomcat ist ähnlich wie das Deployment von BPEL-Prozesse. Für das Deployment werden die Webanwendungen in den /webapps Ordner kopiert. Wenn die Webanwendungen in den Web-Archiv eingepackt werden, werden die Web-Archiv in den /webapps Ordner automatisch ausgepackt. Tomcat bietet auch eine andere Möglichkeit, eine Webanwendung einzusetzen: die Manager-Anwendung. Mittels der Manager-Anwendung kann man die Webanwendungen über ein Web-Interface hinzufügen, löschen oder neue WAR-Dateien aufspielen[TALT] .

Verwendung in dieser Arbeit

Im Rahmen dieser Arbeit kommt Tomcat in der Version 5.0 zum Einsatz und wird für das Deployment der Webanwendungen und Apache ODE bereitgestellten Web-Archive verwendet.

4.8. Genutzte Werkzeuge

Zum Erstellen der BPEL-Prozesse und der in Java geschriebenen Web Services kommt im Rahmen dieser Arbeit die Eclipse IDE in der Version 3.6.2 mit installierter Web Tools Platform und BPEL Visual Designer zum Einsatz. Zur Entwicklung der grafischen Benutzeroberfläche wird SWT in der Eclipse IDE verwendet.

5. Implementierung

In diesem Kapitel gibt es drei Hauptaufgaben, die vorgestellt werden müssen: Bereitstellung einer grafischen Benutzeroberfläche zur Definition Business Regeln nach vordefinierten XML-Schema, Umsetzung des Business Rule Managers als ein Java Web Service, Testen der erstellten Web Services in einem Geschäftsprozess. Als erstes stellt Kapitel 5.1 die Regel-Editor vor. Als nächstes wird in Kapitel 5.2 und 5.3 der Business Rule Manager beschrieben. Schließlich wird in Kapitel 5.4 und 5.5 der Business Rule Manager in einem Geschäftsprozess getestet.

5.1. Regel-Editor

5.1.1. XML-Schema von Regeln

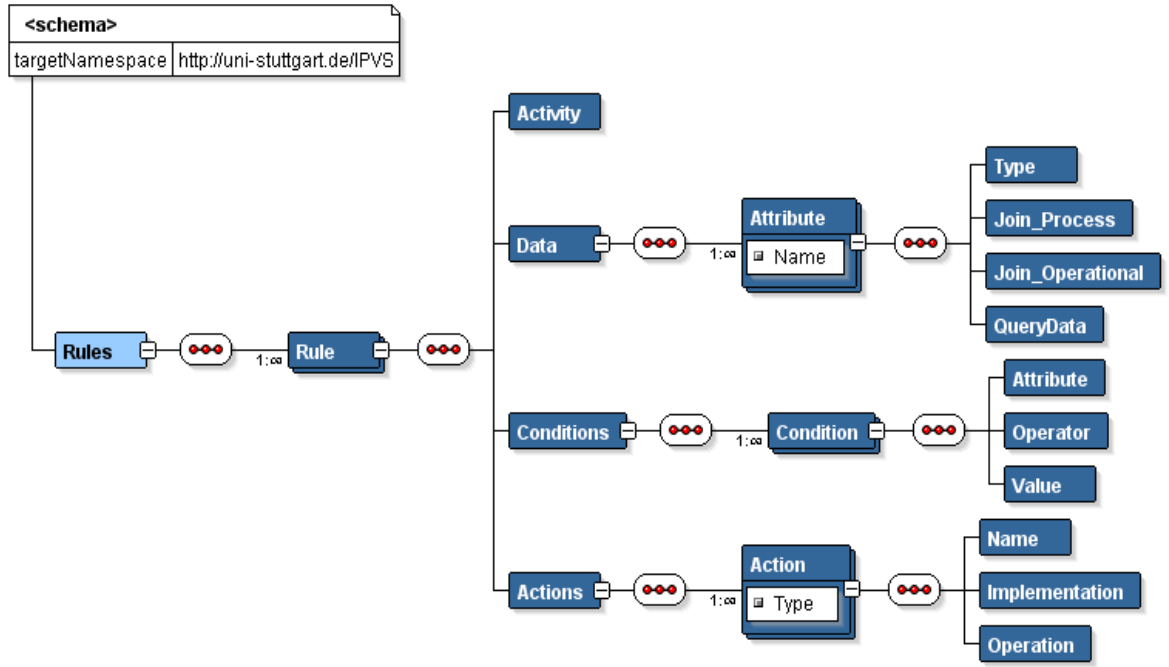


Abbildung 15 : XML-Schema von Extended Rules

Bevor die grafische Benutzeroberfläche bereitgestellt wird, muss das XML-Schema für Extenden Business Rules erstellt wird. In dem vorliegenden Kapitel ist die

wichtigste Struktur von einem Extenden Business Rule vorgestellt worden. Ein Extenden Business Rule muss mindestens drei Haupteile enthalten: „Data“, „Conditions“ und „Action“.
In einem „Data“ Element liegt mindestens ein „Attribute“ Element, das aus vier Elemente besteht: Type, „Join_Process“, „Join_Operational“ und „QueryData“. Das „Type“ Element zeigt den Typ von der benötigen Daten. Das Element „Join_Operational“ definiert die benötigen Prozessdaten. Das Element „Join_Process “ definiert den konkrete Ort in einem Data-Warehouse. Die beide sind zur Findung der Werte der operativen Daten „QueryData“ relevant.

Ein „Conditions“ Element besteht aus mehrere „Condition“ Element. In Jedem „Condition“ Element gibt es drei Bestandteile: „Attribute“, „Operator“ und „Value“. Der Inhalt des „Attribute“ Elementes entspricht dem Inhalt vom „QueryData“ Element. Mit den „Operator“ und „Value“ Elemente kann ein relationaler Ausdruck wie „Rental == 0 “ erzeugt wird. In dieser Arbeit können nur die Operatoren „==“,
„!=“, „>=“, „<=“, „>“ und „<“ unterstützt werden.

Für das „Action“ Element gibt es verschiedene Möglichkeiten. Aber es ist nicht der Schwerpunkt von dieser Arbeit. Wir nehem an, wenn alle Bedingungen in dem Bestandteil „Conditions“ erfüllt sind, wird ein boolischer Wert „true“ als die Ausgabe zurückgesendet.wenn nicht, wird ein boolisher Wert „false“ als die Ausgabe zurückgesendet.

Das Element „Activity“ beinhaltet den Namen der Aktivität, für die die Regel definiert ist.

5.1.2 Regelausführung

Zur Ausführung der Regeln in dem XML-Format werden die Paketen „JDOM.jar“ und „org.eclipse.swt.jar“ benötigt.

Mittels dem „org.eclipse.swt.jar“ Paket werden die grafische Benutzeroberfläche

erstellt, in der zwei Tabelle liegen. Eine davon wird für die Eingabe der benötigen Daten erstellt. Die andere wird für die Konstruktion des relationalen Ausdruckes erzeugt. Nach die in zwei Tabelle gespeicherten Daten kann eine Regel als eine XML-Datei mittels des APIs XMLoutputter, der in dem „JDOM.jar“Paket besteht, ausgegeben werden.

Der Vorgang von Erstellung einer Regel sieht so aus:

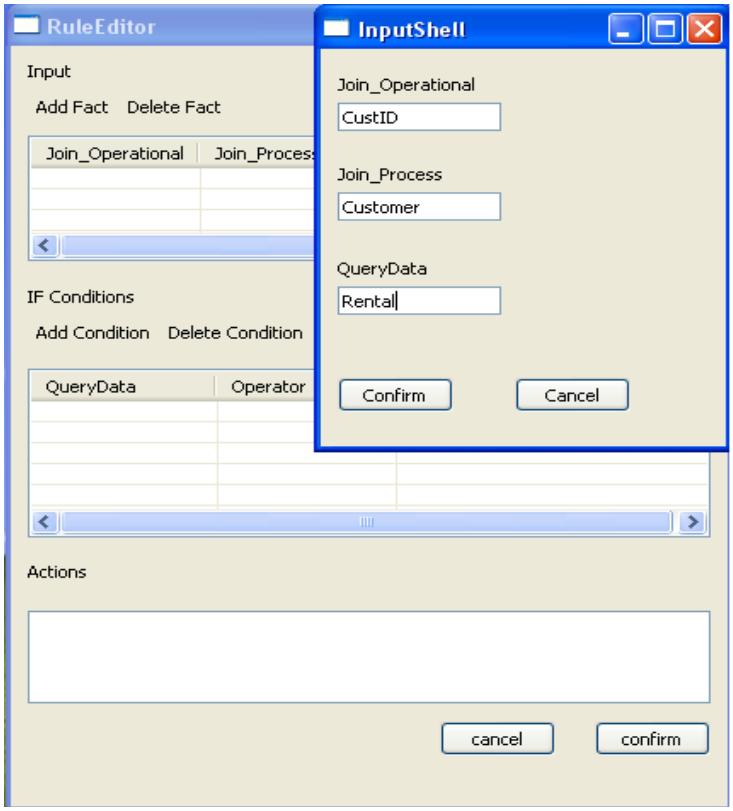


Abbildung 16 : Eingabe der Daten für das Element „Attribute“

Bei Drücken vom „Add Fact“Button wird ein Fenster für die Eingabe der Daten für das Element „Attribute“ aufgerufen. Nach der Eingabe werden die Daten in der ersten Tabelle gespeichert:

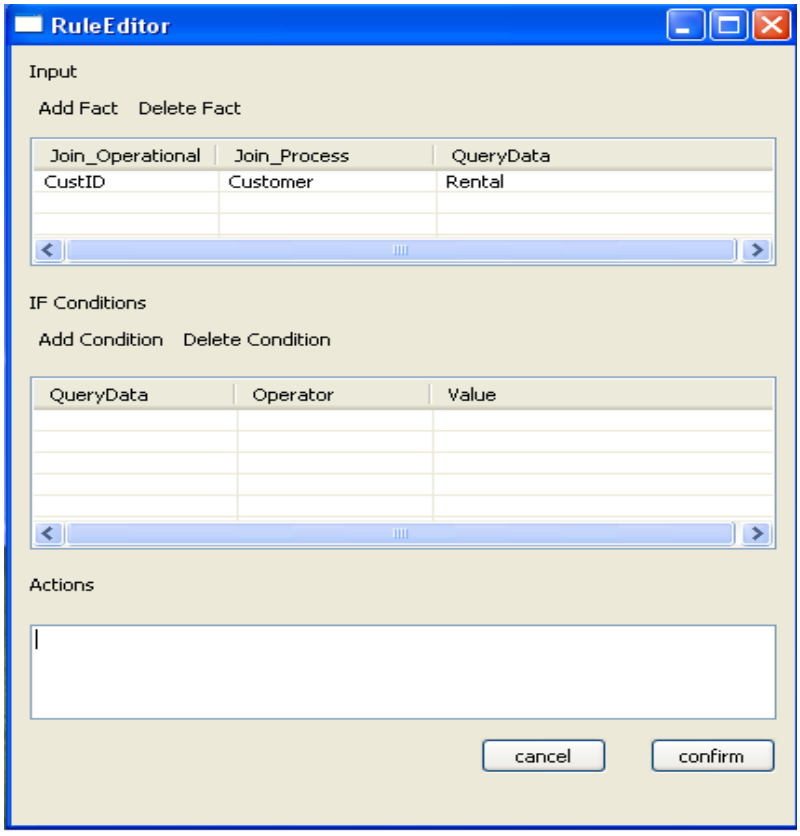


Abbildung 17 : Die gespeicherten Daten in der ersten Tabelle

Bei Drücken vom „Add Condition“ Button wird ein Fenster für die Aufbau des relationalen Ausdruckes für das Element „Condition“ aufgerufen. Nach der Eingabe werden die Daten in der zweiten Tabelle gespeichert:

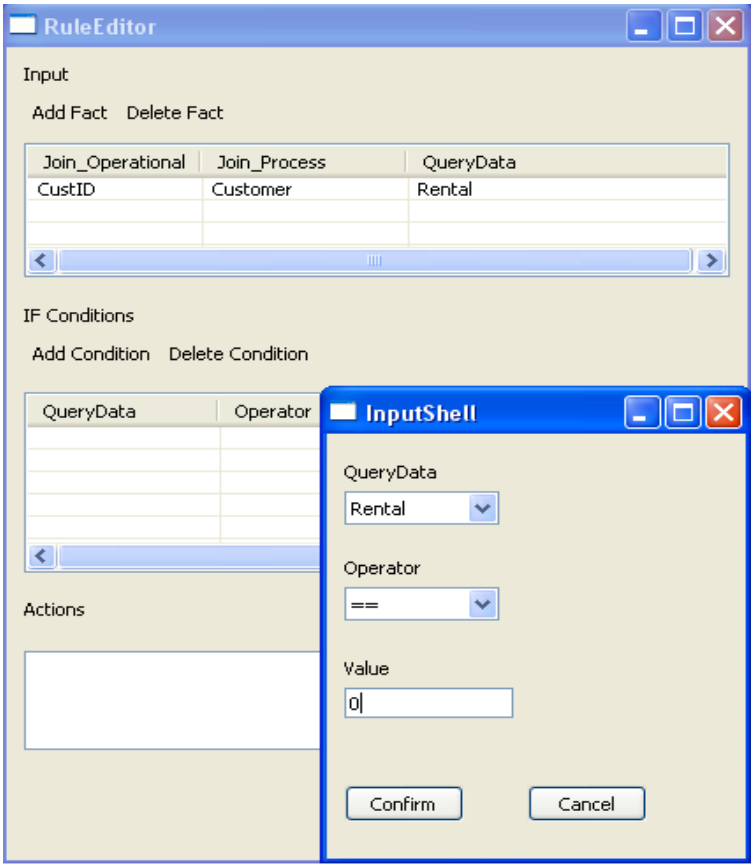


Abbildung 18 : Erstellung der Bedingungen

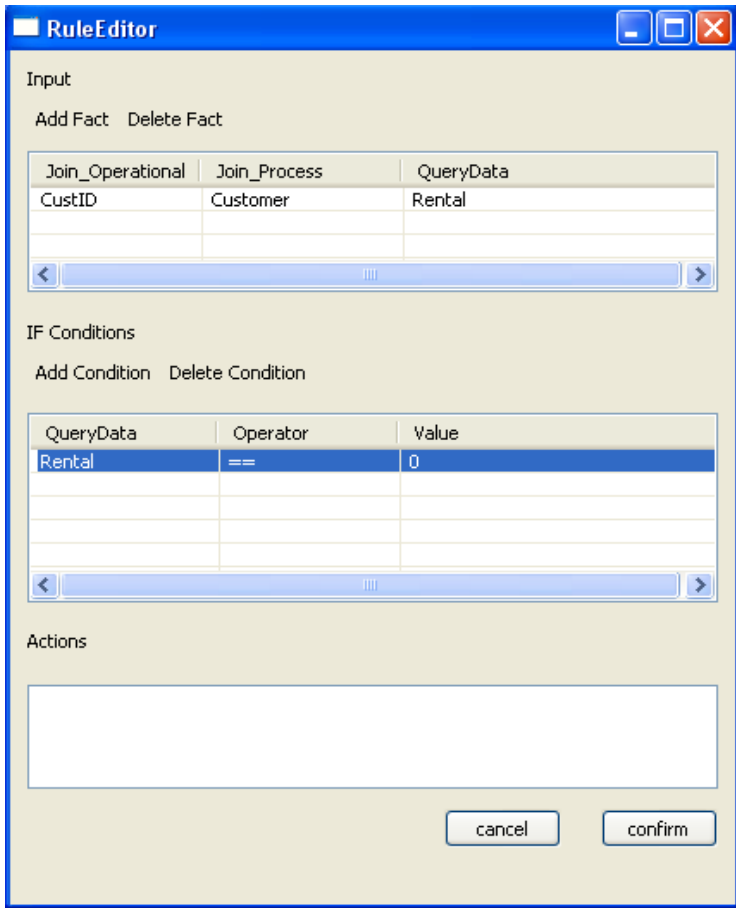
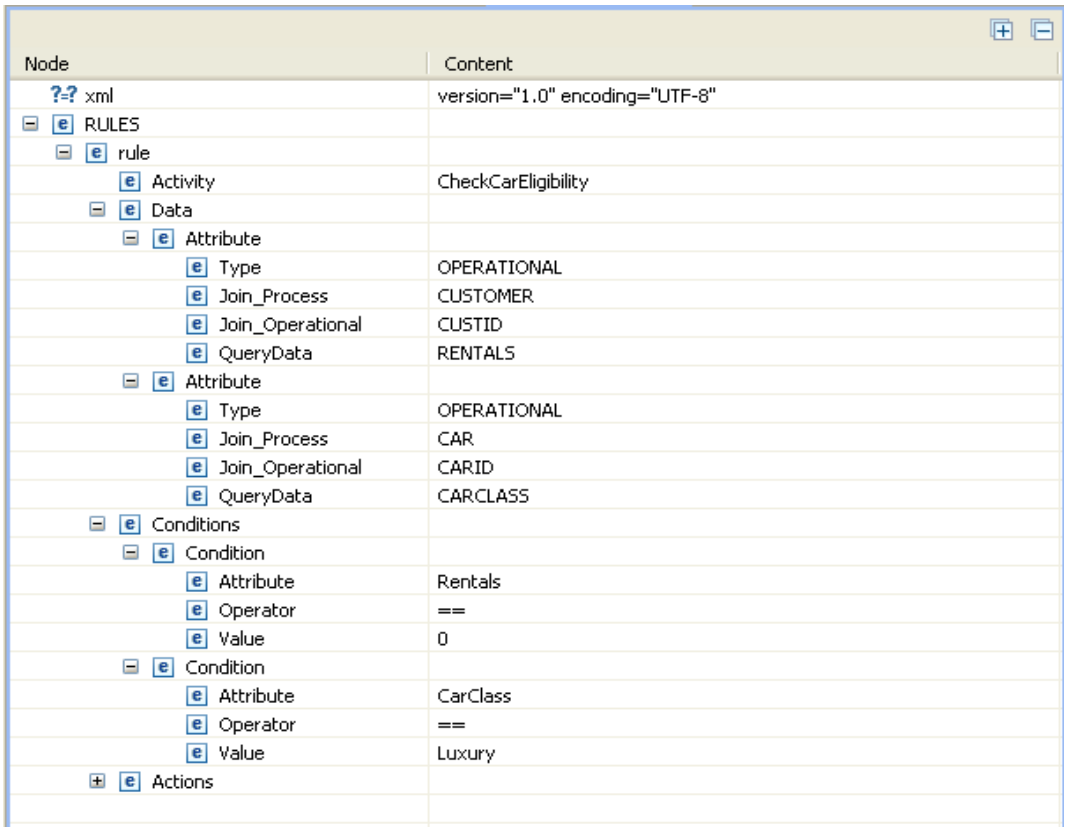


Abbildung 19 : Die gespeicherten Daten in der zweiten Tabelle

Bei Drücken vom Button „confirm“ wird die Regel erzeugt. Bei der erfolgreichen Erzeugung der Regel wird die Struktur der erzeugt Regel in dem XML-Format in der Konsole gezeigt:

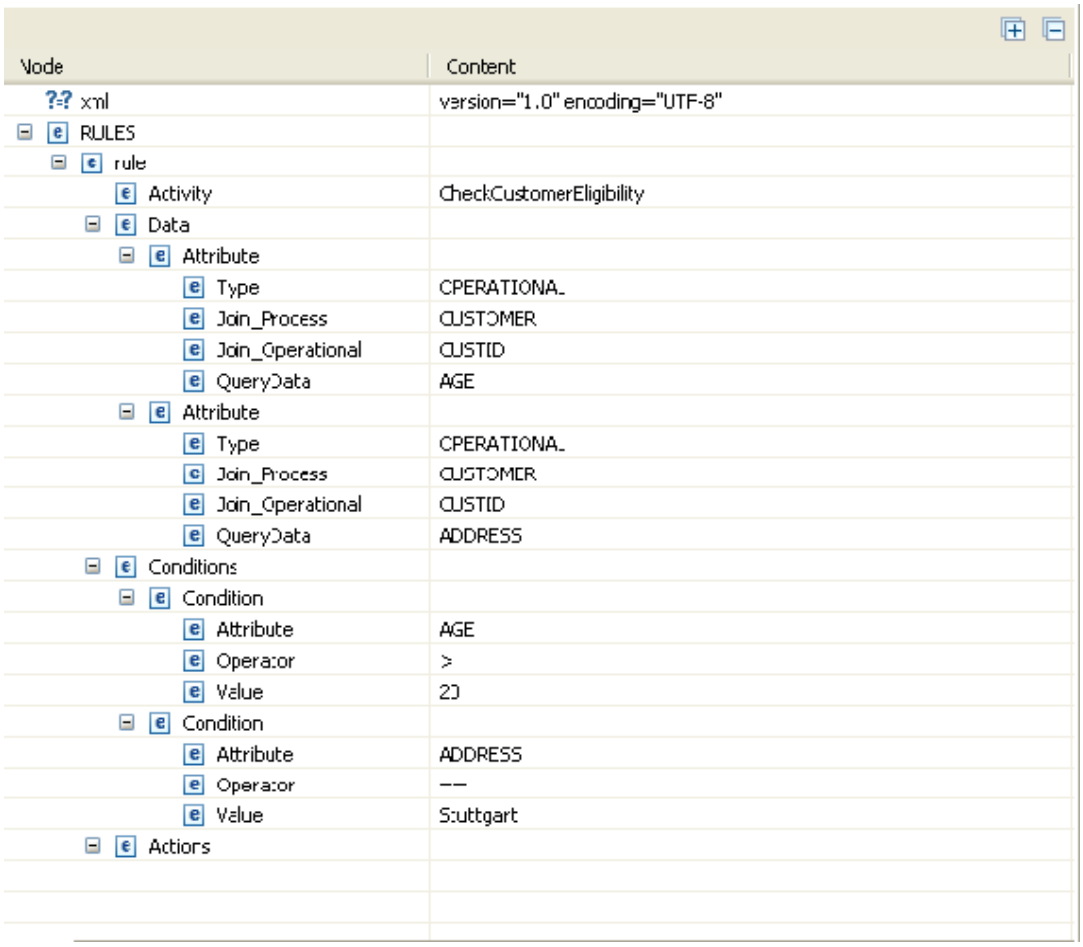
Abbildung 20 : Informationen in der Konsole

Zum Testen der erstellten Web Services in einem Geschäftsprozess werden zwei Regeln in dieser Arbeit definiert : Rules.xml und RuleCustomer.xml. Bei Rules.xml werden die Bedingungen „Rentals == 0“ und „CarClass == „Luxury““ (Der Kunde hat noch keine Vermietungen und möchtet ein Luxusauto mieten) geprüft. Bei RuleCustomer.xml werden die Bedingungen „Age > 20“ und „Address == Stuttgart“ (Der Kunde muss mindestens 21 Jahr alt sein und in Stuttgart Wohnen) geprüft. Dazu muss auch die Werte der entsprechenden operativen Daten „Rentals“, „CarCalss“, „Age“ und „Address“ nach den empfangenen Prozessdaten in der Datenbank gefunden werden.



Node	Content
xml	version="1.0" encoding="UTF-8"
RULES	
rule	
Activity	CheckCarEligibility
Data	
Attribute	
Type	OPERATIONAL
Join_Process	CUSTOMER
Join_Operational	CUSTID
QueryData	RENTALS
Attribute	
Type	OPERATIONAL
Join_Process	CAR
Join_Operational	CARID
QueryData	CARCLASS
Conditions	
Condition	
Attribute	Rentals
Operator	==
Value	0
Condition	
Attribute	CarClass
Operator	==
Value	Luxury
Actions	

Abbildung 21 : Rules.xml



Node	Content
?-? xml	version="1.0" encoding="UTF-8"
RULES	
rule	
Activity	CheckCustomerEligibility
Data	
Attribute	
Type	OPERATIONAL
Join_Process	CUSTOMER
Join_Operational	CUSTID
QueryData	AGE
Attribute	
Type	OPERATIONAL
Join_Process	CUSTOMER
Join_Operational	CUSTID
QueryData	ADDRESS
Conditions	
Condition	
Attribute	AGE
Operator	>
Value	20
Condition	
Attribute	ADDRESS
Operator	==
Value	Stuttgart
Actions	

Abbildung 22 : RuleCustomer.xml

5.2. Verbindung mit Datenbank durch Hibernate

Nachdem die Business Regeln erzeugt geworden sind, muss ein Web Service als einen Regel-Prüfer erstellt wird, der drei Aufgaben besitzt. Eine von drei Aufgaben ist Findung der Werte von den in der Business Regel definierten operativen Daten in einer Datenbank(Oracle Database 10g). Dazu muss eine Verbindung mit der Datenbank eingerichtet werden. Diese Verbindung wird durch Hibernate realisiert.

Die wichtigsten Paketen von Hibernate sind „Hibernate3.jar“, „jta.jar“, „slf4j-api-1.5.8.jar“, „antlr-2.7.6.jar“, „commons-collections-3.1.jar“, „dom4j-1.6.1.jar“ und „javassist-3.9.0.jar“, die in dem Ordner WebContent\WEB-INF\lib des

entsprechenden Web Service hingelegt werden müssen. Außerdem ist „Oracle JDBC Driver“ auch nötig, der mit dem Namen „ojdbc.jar“ von Oracle Datebase 10g angeboten wird. Da die Ausgabe von Web Services als eine XML-Datei gesendet wird, muss das Paket „JDOM.jar“ auch eingefügt werden.

Zur Einrichtung der Verbindung muss zuerst die Datei „hibernate.cfg.xml“ erzeugt werden. Die Konfiguration von Hibernate ist eigentlich die Konfiguration für Sessionfactory.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property name="dialect">org.hibernate.dialect.OracleDialect</property>

        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
        <property name="connection.username">user</property>
        <property name="connection.password">11111111</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.OracleDialect</property>

        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>

        <!-- Disable the second-level cache  -->
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider
        </property>
```

```
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>

        <mapping resource="com/entity/cn/Employee.hbm.xml"/>
        <mapping resource="com/entity/cn/Customer.hbm.xml"/>
        <mapping resource="com/entity/cn/Car.hbm.xml"/>
        <mapping resource="com/entity/cn/Reservation.hbm.xml"/>
        <mapping resource="com/entity/cn/Location.hbm.xml"/>
        <mapping resource="com/entity/cn/Rulelogtable.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Listing 6 : hibernate.cfg.xml

Die Erläuterungen dafür sind:

- **dialect :** Er wird von verschiedenen Datenbanken eindeutig bestimmt.
- **connection.driver_class :** Der benötigen JDBC Driver wird hier bestimmt.
- **connection.url :** Verbindungs-URL.
- **connection.username und connection.password :** Für Anmeldung der Datenbanken.
- **connection.pool_size :** Die Größe des Verbindungs-Pools.
- **current_session_context_class :** Auf welcher Weiser wird session erzeugt (thread oder jta).

- **cache.provider_class :** Er bestimmt, ob der Cache durch Hibernate unterstütz wird.
- **show_sql :** Er bestimmt, ob die Sql-Sätze in der Konsole ausgedrückt werden.

- **hbm2ddl.auto :** Er bestimmt, ob die Datenbank bei jeder Durchlauf neu aufgebaut wird. (Pass auf, dass der Wert dieser Eigenschaft in dieser Arbeit „Update“ sein muss. Sonst können die wichtigen Daten in der Datenbank verloren gehen.

- **mapping resource:** Die Adresse von „* .hbm.xml“ Dateien, die die Struktur der Tabellen und die Zusammenhänge zwischen die Tabelle beschreiben.

Die „hibernate.cfg.xml“ Datei muss in dem Orden /src gestellt werden.
Nach der Konfigurierung von Hibernate müssen die entsprechenden „*.hbm.xml“ für die Oracle Datenbank erstellt werden, damit die Datenbank aufzubauen ist und danach auch die entsprechenden Klassen für die Tabellen erzeugt werden. In dieser Arbeit werden sechs Tabellen erstellt : Employee(EmpID, Name, Age, Department, Training) , Customer(CustID, Name, Address, Occupation, Age, Rentals), Car(CarID, CarClass, Colour, Manufacturer, DailyPrice, Extras), Location(LocationID, Zip, City, Street), Reservation(ResID, CustID, Von, Nach, CarID, LocationID), RuleLogTable(RuleID, TimeStart, ProcessName, RuleMassage, RuleStatus, RuleName). Dafür müssen auch sechs entsprechenden „.hbm.xml“ Dateien erstellt werden.

```
<?xml version="1.0" encoding='UTF-8'?>

<!DOCTYPE hibernate-mapping PUBLIC

                                    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

                                    "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd" >
```

```
<hibernate-mapping package="com.entity.cn">

    <class name="Rulelogtable" table="RULELOGTABLE">

        <id name="RuleID" column="RULEID" type="java.lang.Integer" >
            <generator class="native">
            <param name="sequence">RULE_SEQ_ID</param>
            </generator>
        </id>

        <property name="TimeStart" column="TIMESTART" type="java.lang.String"
        not-null="true" />

        <property name="ProcessName" column="PROCESSNAME"
type="java.lang.String"
        not-null="true" />

        <property name="RuleMassage" column="RULEMASSAGE"
type="java.lang.String"/>

        <property name="RuleStatus" column="RULESTATUS" type="java.lang.String"
        not-null="true"/>

        <property name="RuleName" column="RULENAME" type="java.lang.String"
        not-null="true"/>

    </class>


</hibernate-mapping>
```

Listing 7 : Rulelogtable.hbm.xml

Diese Datei zeigt die Abbildungen zwischen Java Klassen und die Tabellen. Jede Klasse entspricht einer Tabelle in der Datenbank. Jede Variable in einer Klasse entspricht einer Reihe der entsprechenden Tabelle. Bei der Generierung von „ID“ gibt es auch einige Strategien :

- **assigned :** Der primäre Schlüssel „ID“ wird durch anderen Programmen bestimmt.

- **Hilo :** Der primäre Schlüssel wird durch den hi/lo Algorithmus bestimmt.

- **Seqhilo :** Ähnlich wie Hilo

- **increment :** Der primäre Schlüssel ist abhängig von einer Variable, die um 1 zunimmt.

- **Identity :** Die Generierung von „ID“ ist abhängig von der Datenbank (Die Datenbank bietet eine Strategie für die Generierung von „ID“) an.

- **Sequence :** Die Datenbank bietet einen sogenannten „Sequence“ Mechanismus, davon die Generierung von „ID“ anhängt.

- **Native :** Die Strategie kann „Hilo“ , „identity“ oder „sequence“ sein.

- **Foreign :** Die Generierung des primäre Schlüssel ist abhängig von dem primären Schlüssel in einer anderen Tabelle.

In dieser Datei wird der „Sequence“ Mechnismus angewendet. Dafür wird ein „Sequence“ in der Oracle Datenbank mit dem Namen „RULE_SEQ_ID“ erstellt. „RULE_SEQ_ID“ wird so erstellt : Es fängt mit 1 an, endet mit 99999999999999 und nimmt um 1 zu.

Min Value	1
Max Value	999999999999999999999999999
Increment By	1
Cycle Flag	N
Order Flag	N
Cache Size	20
Last Number	161

Abbildung 23 : RULE_SEQ_ID

In anderen Dateien kommt die Strategie „native“ zum Einsatz.

Darüber hinaus kann die konkreten Relationen zwischen Tabellen in dieser Datei definiert werden. In dieser Arbeit sind diese Relationen nicht nötig, da jede Anfrage innerhalb einer Tabelle durchgeführt wird.

Mittels der erstellten sechs „*.hbm.xml“ Dateien können auch die entsprechenden Java Klassen (Java POJO) und die Tabellen erzeugt werden. Die Erzeugung der Java Klassen und der Tabellen kann durch die Hibernate-Tools „hbm2java“ und „hbm2dll“ realisiert werden.

Es gibt insgesamt sechs Java Klassen, die erzeugt werden. Für jede Reihe einer Klasse werden zwei Funktionen „getter“ und „setter“angeboten, die zur Anfrage und Modifizierung der Daten angewendet.

Nachdem die oben vorgestellten Dateien und die Tabelle erstellt worden sind, kann die Verbindung mit Datenbank durch SessionFactory realisiert werden. SessionFactory ist eine Klasse in Hibernate, die die Konfiguration und die Abbildungen lädt. Durch SessionFactory können Session erzeugt werden. Session verbindet die Java-Programme mit den Hibernate Diensten, damit die Anfrage der operativen Daten und die „Insert“ Operation in dieser Arbeit durchgeführt werden.

5.3. Regel-Prüfer

Die zweite Aufgabe für die Einrichtung von einem Web Service als einen Regel-Prüfer ist die Erstellung der Funktionen. In dieser Arbeit muss ein Regel-Prüfer drei Funktionen anbieten : Erstens muss die Werte der in der Regeln definierten operativen Daten in der Datenbank gefunden werden. Zweitens muss die in der Regeln definierten Bedingungen nach den Werte der operativen Daten geprüft werden. Drittens muss die Informationen für die aufgerufenen Regeln in einer Tabelle (RULELOGTABLE) gespeichert werden, damit der Status von der aufgerufenen Regel deutlich ist. Die drei Funktionen werden durch die folgenden Methoden realisiert :

- **CreateQuery(Element e, String InputValue) :** Ein SQL-Satz wird anhand der Eingabe in Form von „Select Join_Process.QueryData From Join_Process Where Join_Process.Join_Opertional = InputValue“ erzeugt und danach durch die Klasse „SQLQuery“ in der Datenbank verwendet.

- **check(Element e) :** Wenn die Bedingungen in einer Regel erfüllt werden, wird eine Ausgabe mit dem Wert „true“ zrückgesendet, sonst „false“.

- **InsertRuleLogingTable(String ProcessName, String RuleStatus, String RuleName) :** Die Informationen für die aufgerufenen Regeln werden durch die Methoden „setter“ von Java POJO in der Tabelle „RULELOGTABLE“ gespeichert.

Nach der Erstellung der Funktionen kann die Prüfung der Bedingungen durchgeführt werden (muss auch eine Schnittstelle zur Verwendung der drei Funktionen erstellt werden). Dann gibt es noch die letzte Aufgabe : Erzeugung von dem Web Service.
Zur Erzeugung von dem Web Service muss die Apache Tomcat 5.0 voraus installiert werden. Dann kann ein Server in Eclipse erzeugt werden. Die Konfiguration von dem Server sieht so aus :

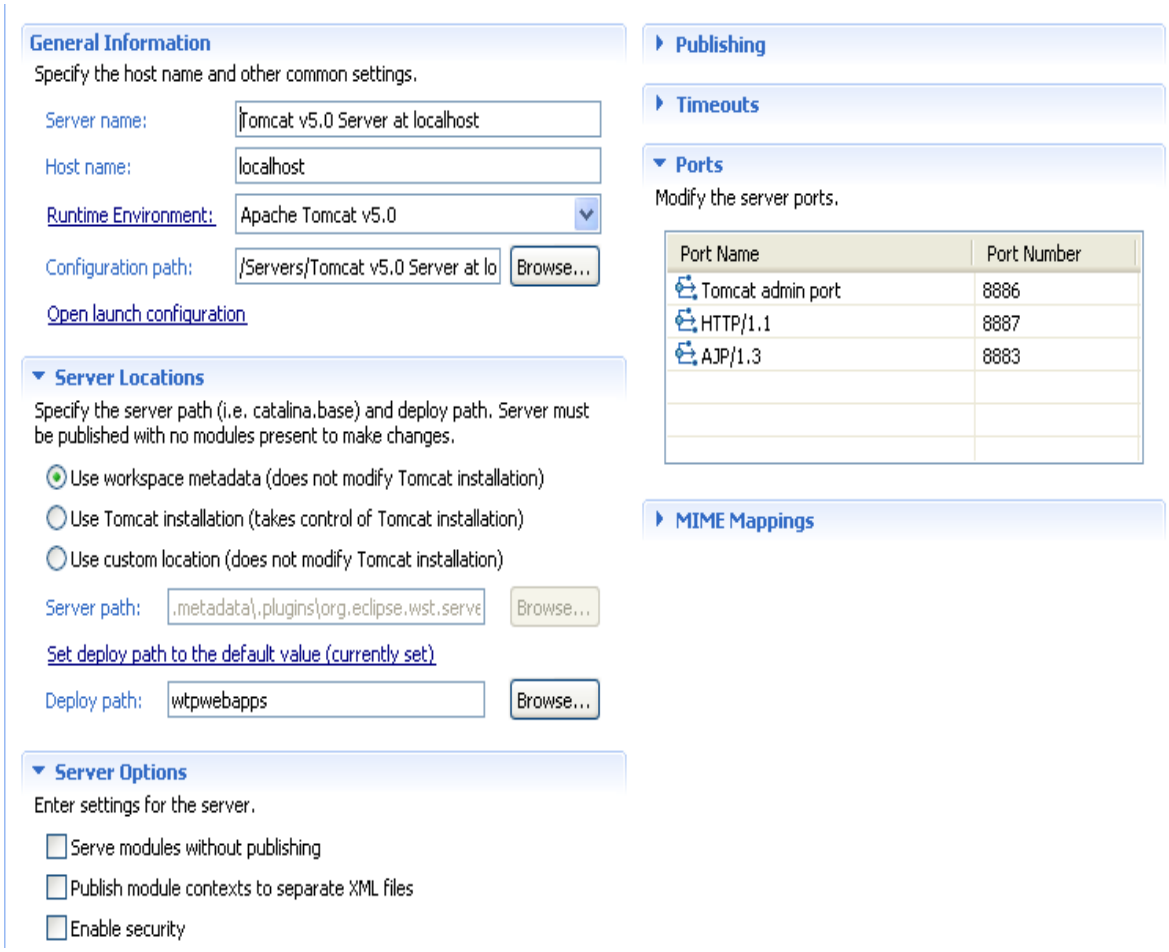


Abbildung 24 : Die konfiguration von Tomcat v5.0 Server

Pass auf, dass die „Port Number“ von „Tomcat admin port“, HTTP/1.1 und AJP/1.3 modifiziert werden, da die Standardeinstellung „8080“ für „Port Number“ von Oracle Datenbank besetzt wird.

Dann kommt die WTP (Web Tool Platform) zum Einsatz. Mittels WTP kann ein dynamisches Web-Projekt namens „CheckRules“ erstellt werden. Danach Kann ein Web Service nach dem Bottom-Up Ansatz durch die folgenden Schritte erzeugt werden :

1. Importiere die bereits vorhandene „hibernat.cfg.xml“ Datei in den Ordner /src und die anderen Dateien (Java Klassen (POJO) und „*.hbm.xml“) in das

entsprechenden Paket (Z.B. /src/com/entity).

2. Wähle die „CheckRule.java“ Datei und Öffne File -> New->Other...->Web Services->Web Service

3. Verschiebe den Schieberegler bis zu der „Strart Service“ Position und den Client-Regler bis zu der „Test Client“ Position. Danach wähle auch „Monitor the Web Service“ und Klicke auf „Finish“.

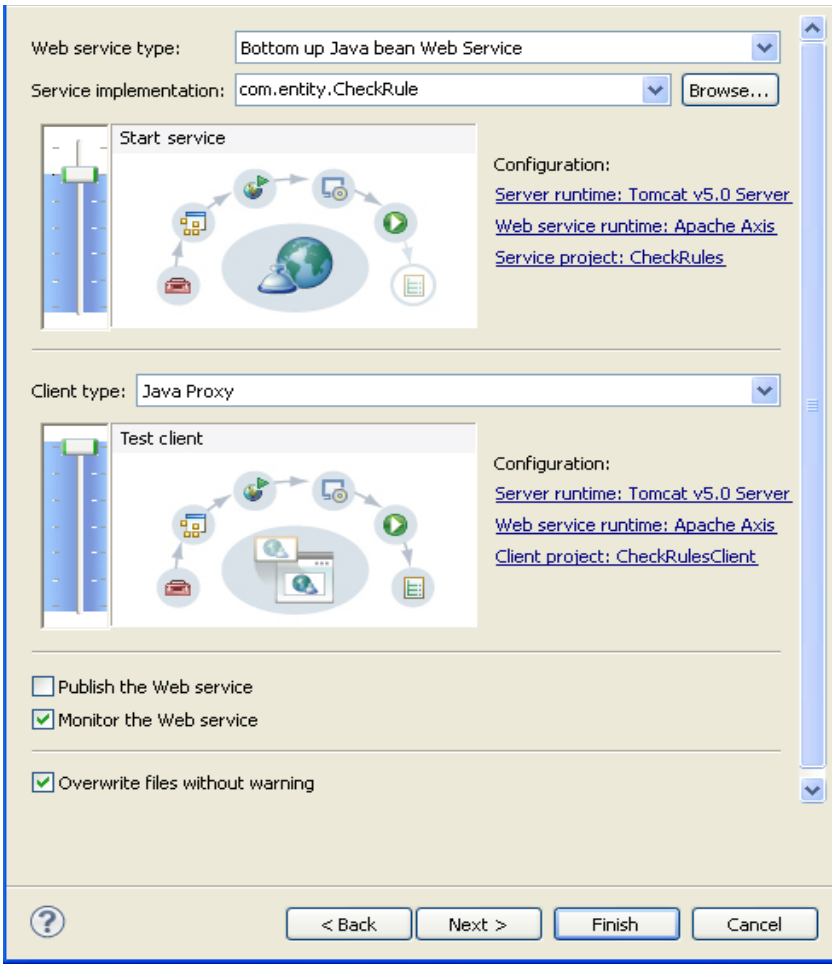


Abbildung 25 : Konfiguration für den Web Service

In Hintergrund passiert es : Zuerst werden die Webanwendung für den Web Service erzeugt und die nötigen Paketen von Axis in diese Webanwendung kopiert.

Anschließend wird die ausgewählte Java Klasse in Abhängigkeit von der Eingaben in die Webanwendung deployt und das WSDL-Dokument als auch der Client mit einer JSP-Seite für den Web Service automatisch erzeugt. Diese Webanwendung wird dann auf dem Tomcat-Server eingespielt.

Zum Testen des erzeugten Web Services fügen wir die „CheckRules“ und „CheckRulesClient“ Projekte in den Tomcat v5.0 Server ein. Im Web Browser werden die angebotene Methode beim Klicken von „Method.jsp“ gezeigt. Nach der Auswahl der Hauptfunktion (query) können die benötigen Daten eingegeben werden.

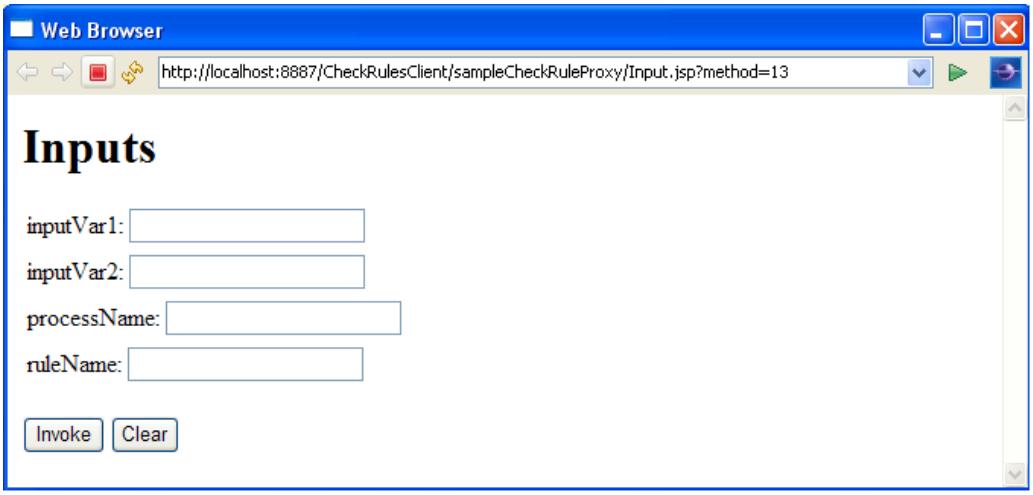


Abbildung 26 : Fenster für die Eingabe

Um den Web Service zu testen, speichern wir voraus einige Daten in den Tabellen „Customer“ und „Car“ in der Oracle Datenbank.

EDIT	CUSTID	NAME	ADDRESS	OCCUPATION	AGE	RENTALS
	2	Mike	Frankfurt	Student	20	1
	1	Peter	Stuttgart	Manager	32	0
					row(s) 1 - 2 of 2	

EDIT	CARID	CARCLASS	COLOUR	MANUFACTURER	DAILYPRICE	EXTRAS
	1	Luxury	Black	BMW	20000	-
	2	UnLuxury	White	VW	2000	-
					row(s) 1 - 2 of 2	

Abbildung 27 : Die Daten in den Tabellen Customer und Car

Wir geben „InputVar1 = 1“(CustID=1), „InputVar2 =1“(CarID=1), „processName =CarRenatlProcess“ und „ruleName = Rules“ ein. Nach der Bedingungen in „Rules.xml“ soll die Resultat „true“ im Web Browser ausgegeben werden.

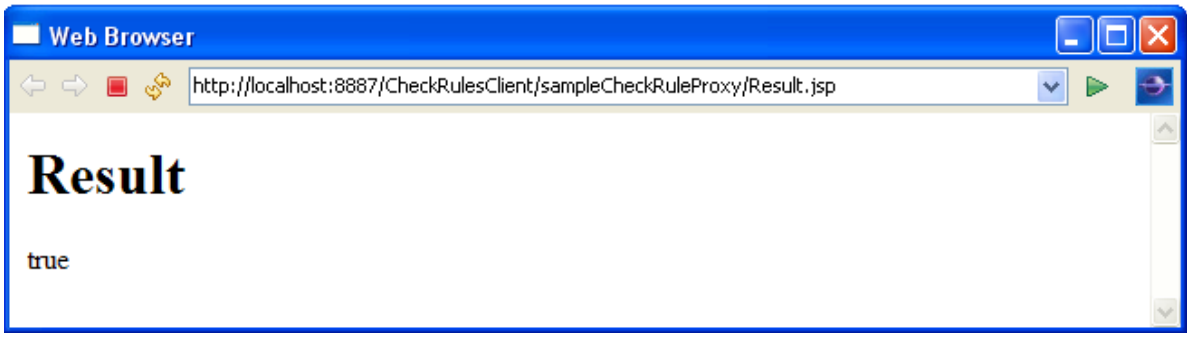


Abbildung 28 : Fenster für die Ausgabe

Die Informationen in der Konsole sind :

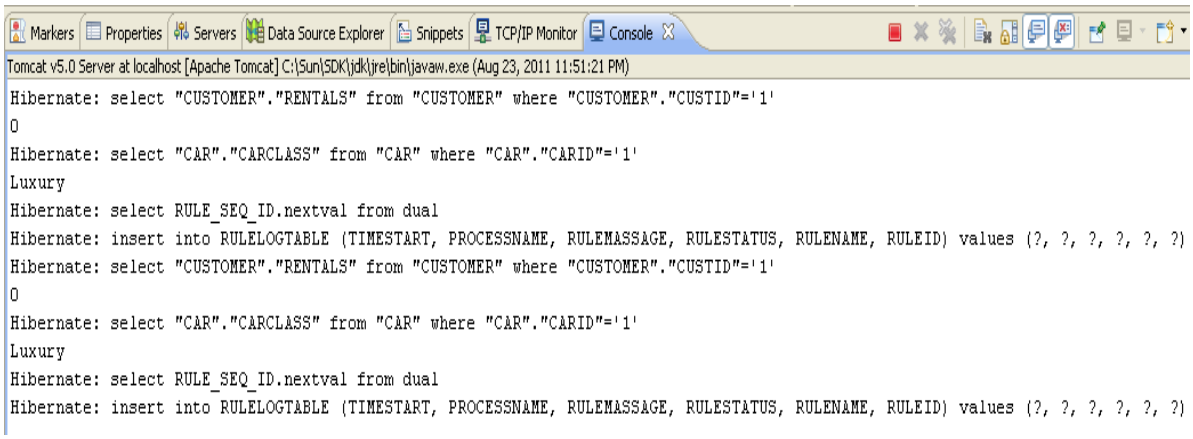


Abbildung 29 : Informationen in der Konsole

Auf der gleichen Weise kann ein Web Service für die Regel „RuleCustomer.xml“ generiert werden.

Pass auf, dass die zwei Web Services gleich sind. Der einzige Unterschied ist die eingelesene Regel. Dieser wird durch „Location“ und „RuleName“ realisiert.

```
Document document = builder.build( new File ( Location + RuleName + ".xml" ) )
```

5.4. Einsetzen in einen einfachen Geschäftsprozess

In diesem Abschnitt soll aufgezeigt werden, wie die Web Services innerhalb eines BPEL-Prozesses einsetzt werden können.

Wir nehmen als Beispiel einen synchronischen BPEL-Prozess „CarRentalProcess“, der diese Web Services als die Extended Business Rule Manager aufrufen.
In der Abbildung 30 ist der BPEL-Prozess „CarRentalProcess“ dargestellt, in dem versucht wird, die vorhandenen Web Services aufzurufen.

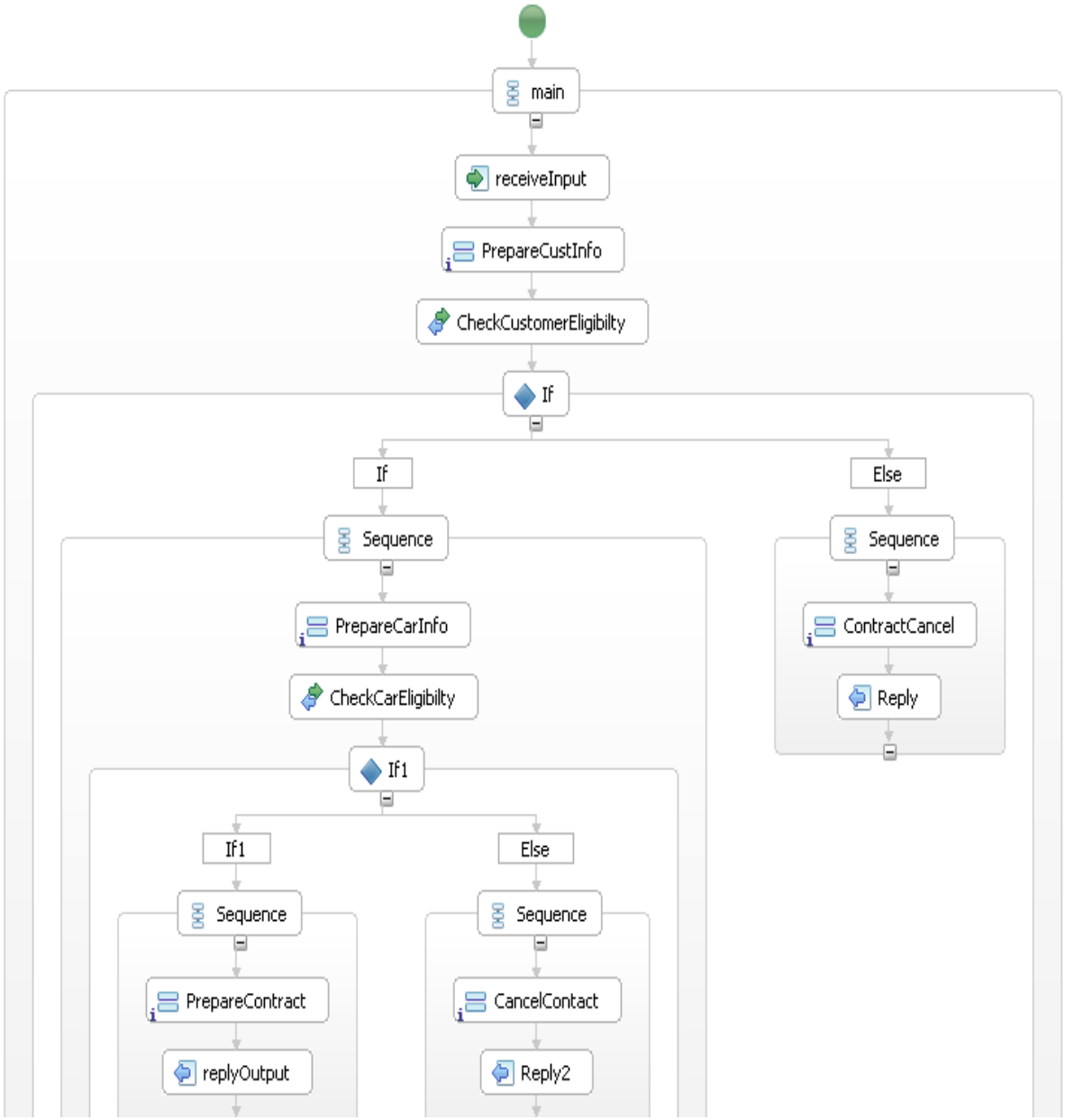


Abbildung 30 : CarRentalProcess

Dieses Beispiel soll im Folgenden genauer betrachtet werden.

Die erste Aktion in diesem BPEL-Prozess ist der Empfangen von Benutzereingaben(CustID, CustName, EmpName, CarID, ProcessName), um später die synchronen Request-Response Operationen für den Extended Business Rule Manager verwenden zu können. Als Nächstes folgt die Aktivität für die Vorbereitung der Requeat-Response Operation, in der die Initialisierung der Request-

Nachricht (CheckCustomerPLRequest) für die Requeat-Response Operation durchgeführt wird. In dieser Nachricht muss zum einen die CustID und CarID eingetragen werden, als auch die Namen von dem Geschäftsprozess und der geprüften Regel. Die Definition der Request-Response Operation besteht in der Aktivität „CheckCustomerEligibilty“. Dazu wird der globale Partnerlink „CheckCustomerPL“ mit der entsprechenden PartnerLinkType festgelegt werden, in den myRole und partnerRole zu definieren sind. Dieser Partnerlink basiert auf das WSDL-Dokument des vorhandenen Web Service für die Regel „RuleCustomer.xml“, das voraus in diesen BPEL-Prozess importiert wird. In der Abbildung 31 wird das WSDL-Dokument des Web Service gezeigt.

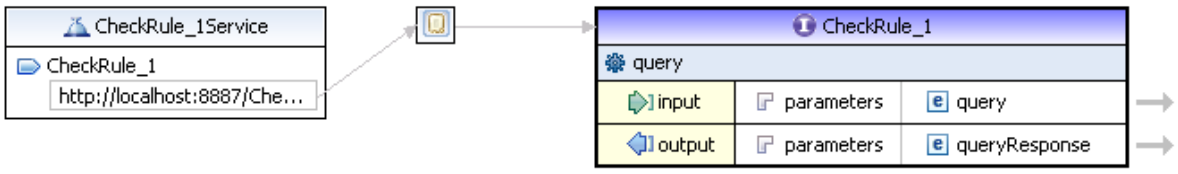


Abbildung 31 : CheckRule_1.wsdl

Dieses WSDL-Dokument enthält eine Schnittstelle namens „CheckRule_1“. In dieser Schnittstelle steht auch eine Operation „query“, die zum Aufrufen der Funktionen in dem Web Service verwendet wird. Zudem gibt es auch die Request-Response Nachrichten.

Um die Operation in dem Web Service aufrufen zu können, muss man die entsprechenden Request-Response Nachrichten in dem Geschäftsprozess definieren. Die sind entweder automatisch oder nach der Abbildung von WSDL zu erzeugen. Hier werden die Request-Nachricht namens „CheckCustomerPLRequest“ und die Response-Nachricht namens „CheckCustomerResponse“ automatisch generiert. In der Abbildung 32 ist die Struktur der beiden Nachrichten.

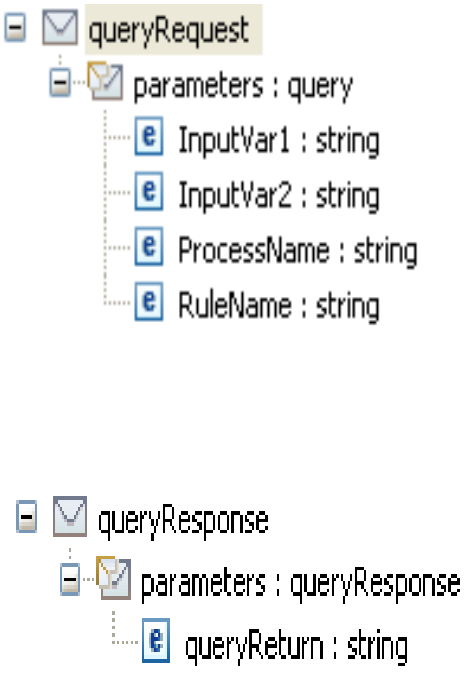


Abbildung 32 : die Struktur von CheckCustomerPLRequest und CheckCustomerResponse

Hat man diese Nachricht in der Aktivität „PrepareCustInfor“ generiert, ruf man die Operation „query“ des Web Service auf, um die ausgewählt Regel zu prüfen als auch die Informationen für die aufgerufenen Regeln in der Datenbank zu speichern. Im Falle dieser Operation gibt es nur zwei Mögliche Response-Nachrichten, die entweder mit „true“ oder mit „false“ generiert werden. Wenn die Bedingungen nach der Eingaben nicht erfolgreich erfüllt werden, wird dieser Geschäftsprozess sofort abgebrochen und in der Response-Nachricht von dem Geschäftsprozess die Information „Contract canceled!“ gespeichert und danach gesendet. Sonst wird eine weitere Aktivität „CheckCarEligibilty“ durchgeführt. Diese Logik kann durch eine If Aktivität mit der entsprechenden Xpath-Anweisung realisiert werden :

```
$CheckCarPLResponse.parameters/ns0:queryReturn="true"
```

Bei der Aktivität „CheckCarEligibilty“ ist die Situation ähnlich wie bei „CheckCustomerEligibily“. In dieser Aktivität wird ein globaler Partnerlink namens „CheckCarPL“ definiert, der die gleiche Operation besetzt. Die Request-Response Nachrichten „CheckCarPLRequest“ und „CheckCarPLResponse“ für diese Operation haben auch die gleiche Struktur wie „CheckCustomerPLRequest“ und „CheckCustomerResponse“.

Nach der Initialisierung der Request-Nachricht „CheckCarRLRequest“ in der Aktivität „PrepareCarInfor“ kann die Operation des Web Service für die Regel „Rules.xml“ aufgerufen werden, damit die aufgerufene Regel geprüft und die Information für diese Regel gespeichert wird. Wenn die Response-Nachricht „CheckCarPLResponse“ eine Werte mit „true“ empfängt, wird die Response-Nachricht von dem Geschäftsprozess mit der Information „Succeed to sign a Contract!“ gesendet, sonst mit „Contract canceled!“. Eine If Aktivität wird hier nochmal angewendet. Die entsprechende Xpath-Anweisung sind :

```
$CheckCarPLResponse.parameters/ns0:queryReturn="true"
```

Pass auf, dass die Response-Nachricht von dem Geschäftsprozess vor den Reply Aktivitäten initialisiert werden muss.

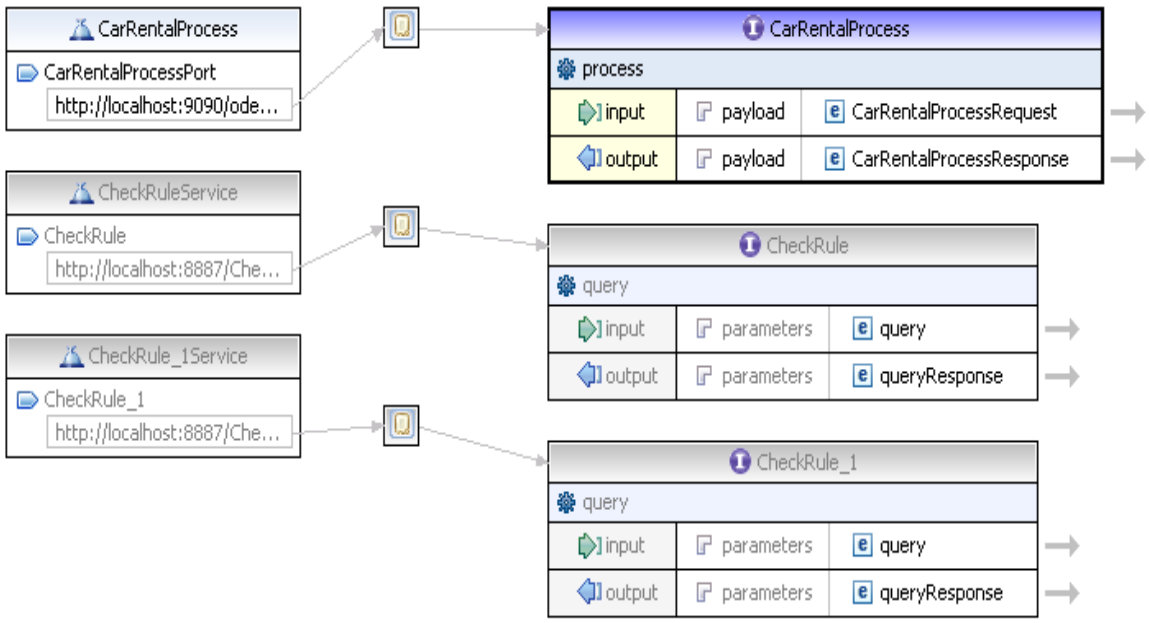


Abbildung 33 : CarRentalProcessArtifacts.wsdl

Nach der Erstellung des Geschäftsprozesses müssen auch die Adresse von dem Service für den Prozess, die Type von Binding und PortType in dem entsprechenden WSDL-Dokument definiert werden. In der Abbildung 33 ist das WSDL-Dokument von dem Geschäftsprozess „CarRentalProcess“. In diesem werden sie so definiert:

Binding :

Name : CarRentalProcessBinding

PortType : CarRentalProcess

Protocol : SOAP

Service :

Service Name : CarRentalProcess

Port Name : CarRentalProcessPort

Binding : CarRentalProcessBinding

Address : http://localhost:9090/ode/processes/CarRentalProcess

Protocol : SOAP

Außerdem wird die Request-Nachricht „CarRentalRequest“ auch neu definiert:

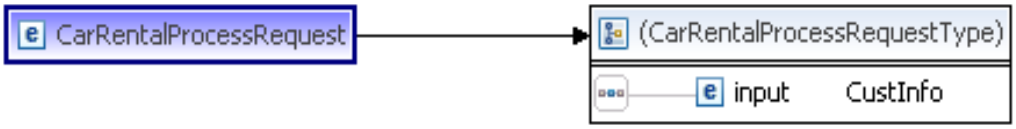


Abbildung 34 : CarRentalProcessRequest

Wobei ist die Type „CustInfo“ ist eine Kompexe Type, die aus fünf Elemente besteht.

Für das Deployment des Prozesses gibt es noch zwei Aufgaben : Einrichtung von Apache ODE Server und Erzeugung von dem Deployment Descriptor „deploy.xml“.

Die Einrichtung von Apache ODE Server ist einfach. Man braucht nur die heruntergeladene „ode.war“ Datei in den Ordner Tomcat\webapps zu legen und die „startup.bat“ Datei in dem Ordner Tomcat\bin zu klicken. Dann kann die ODE automatisch deployt werden. In Eclipse kann man danach den entsprechenden Apache ODE Server einrichten.

Der Deployment Descriptor „deploy.xml“ ist durch die Auswahl von „File ->New ->Other->Apache ODE Deployment Descriptor“ zu erzeugen.

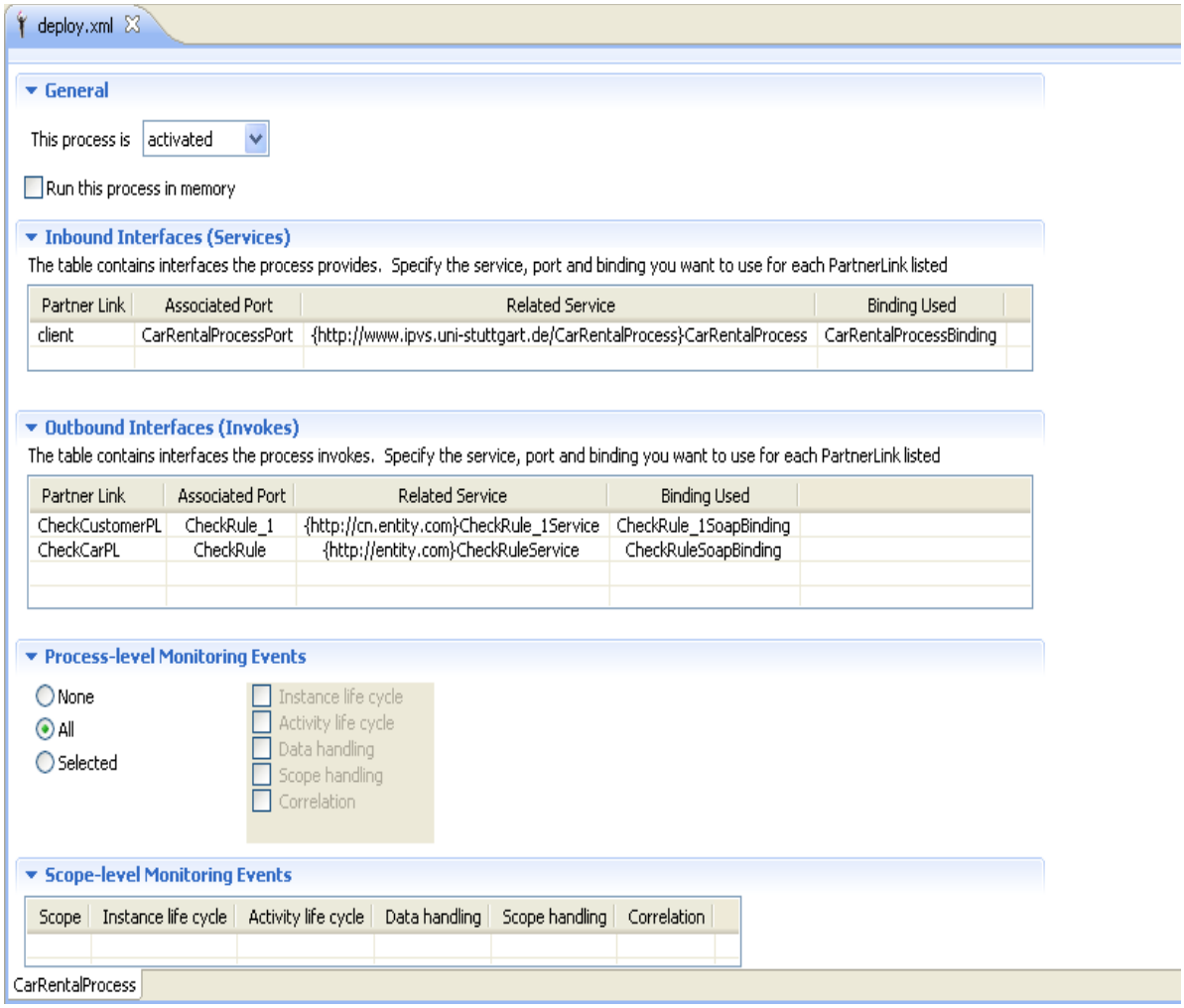


Abbildung 35 :deploy.xml

In dem Deployment Descriptor werden alle relativen Services ausgewählt, damit der Apache ODE Server weiß, welche Services zum Einsatz kommen.

Dann kann der Prozess durch den Einfügen des entsprechenden Projektes in den Apache ODE Server deployt werden.

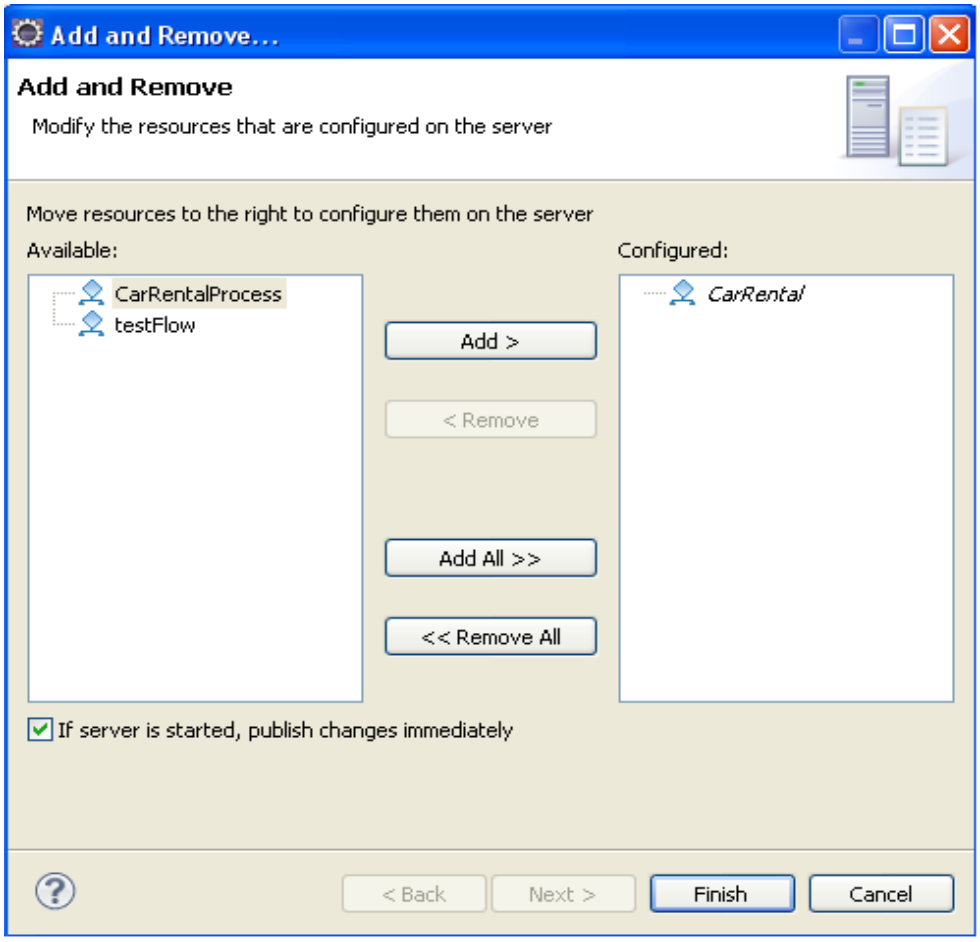


Abbildung 36 : Depoyment von dem Geschäftsprozess

Beim erfolgreichen Deployment werden die Informationen wie „Deployment of artifact CarRental successful“ in der Konsole gezeigt.

5.5. Testen

Der Web-Services-Explorer, der innerhalb WTP integriert wird, ist zum Testen des Geschäftsprozesses zu verwenden.

Der Web-Services-Explorer kann entweder über das Kontextmenü einer WSDL-Datei (Web-Services -> Test with Web Services Explorer) oder über das Menü " Run -> Launch the Web Services Explorer " gestartet werden.

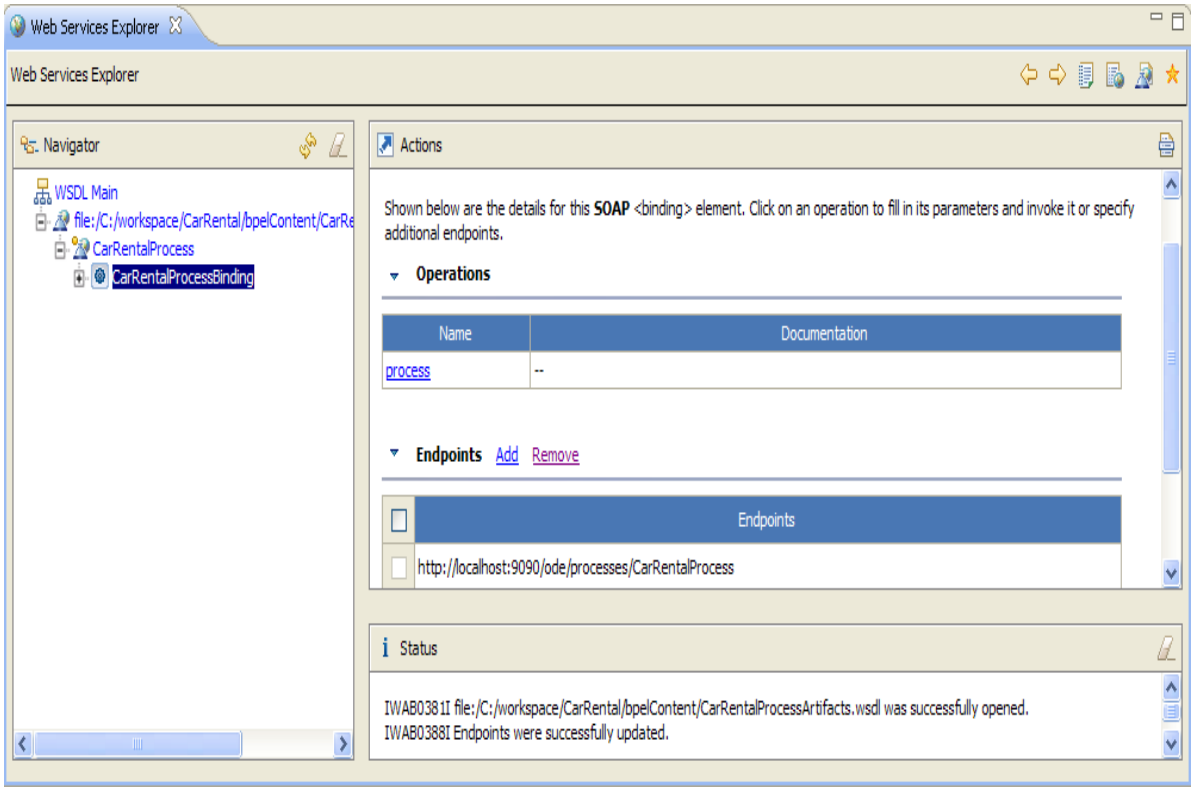


Abbildung 37 : Web Services Explorer

Nach der Auswahl der angebotenen Operation kann man die benötigen Daten eintragen und die entsprechende Resultat betrachten.

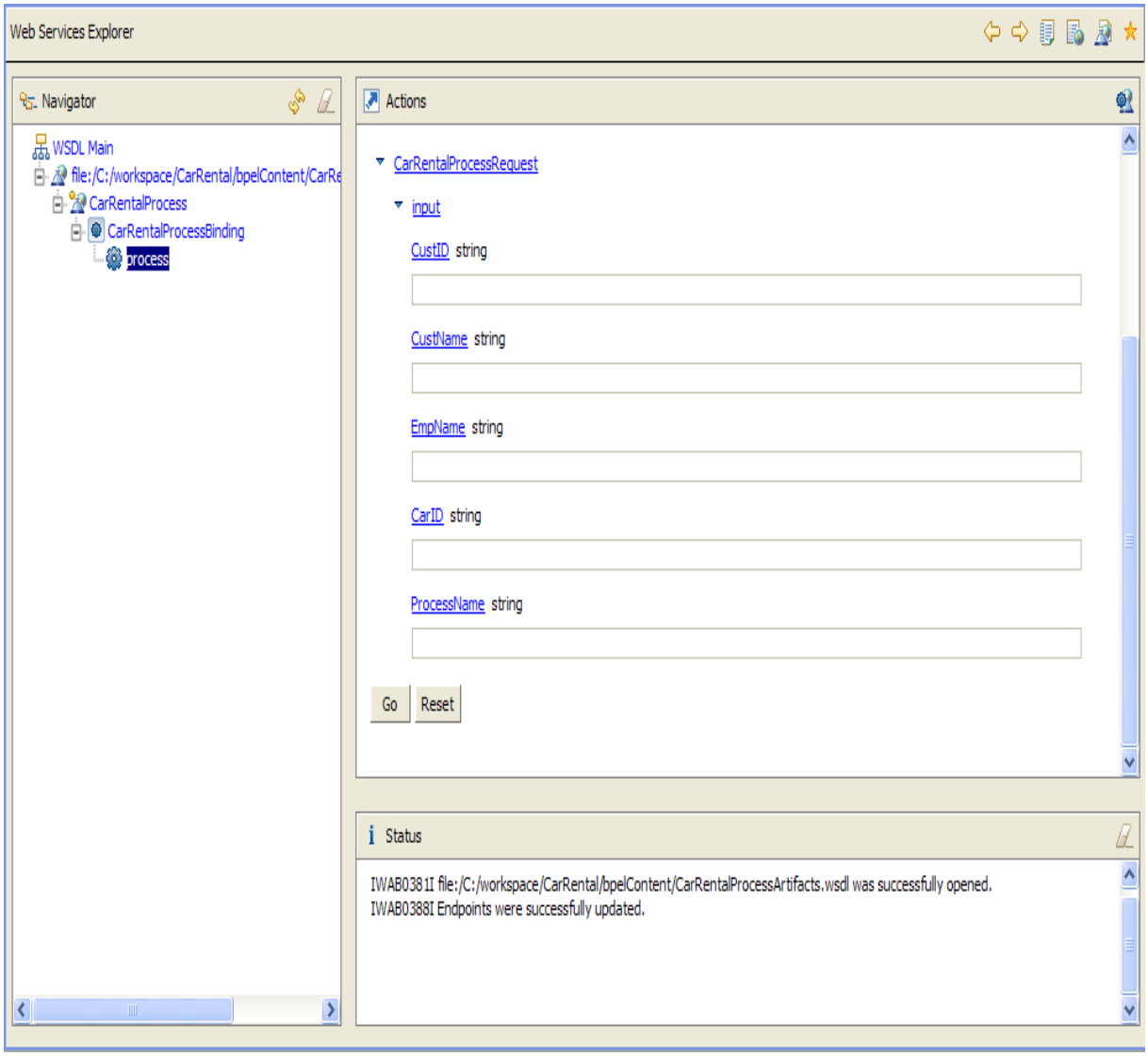


Abbildung 38 : Eingabe für den Testen

Wir geben hier „CustID = 1, CustName = Niedermann, EmpName=liu, CarID=1, processName =CarRenatlProcess“ ein.

Anhand von der voraus in der Oracle Datenbank gespeicherten Daten soll die Resultat „Succeed to sign a Contract!“ sein. Die Informationen für die angewendeten Regeln soll in der Tabelle „RuleLogTable“ gespeichert werden.

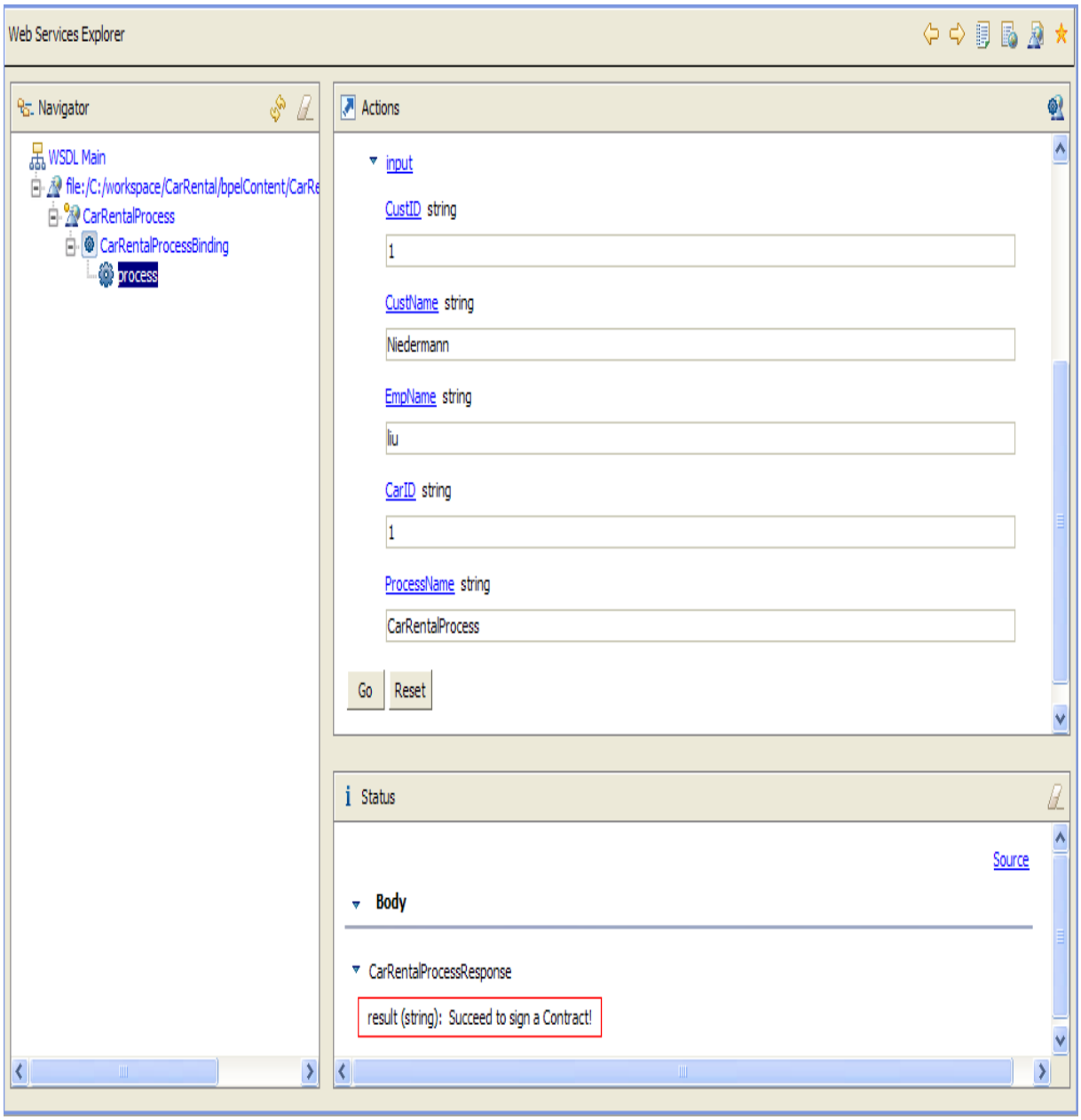


Abbildung 39 : Resultat mit „Succeed to sign a Contract !“

Die Informationen in der Tabelle „RuleLogTable“ sind :

EDIT	RULEID	TIMESTART	PROCESSNAME	RULEMASSAGE	RULESTATUS	RULENAME
	182	2011/08/27:11:12:07	CarRentalProcess	-	OK	Rules
	181	2011/08/27:11:12:03	CarRentalProcess	-	OK	RuleCustomer
					row(s) 1 - 2 of 2	

Abbildung 40 : Informationen in der Tabelle „RuleLogTable“ für zwei aufgerufenen Regeln

Wobei werden zwei Regeln aufgerufen.

Bei der Eingabe „CustID = 2, CustName = Niedermann, EmpName=liu, CarID=1 processName =CarRenatlProcess “ soll die Resultat „ Contract canceled“ sein.

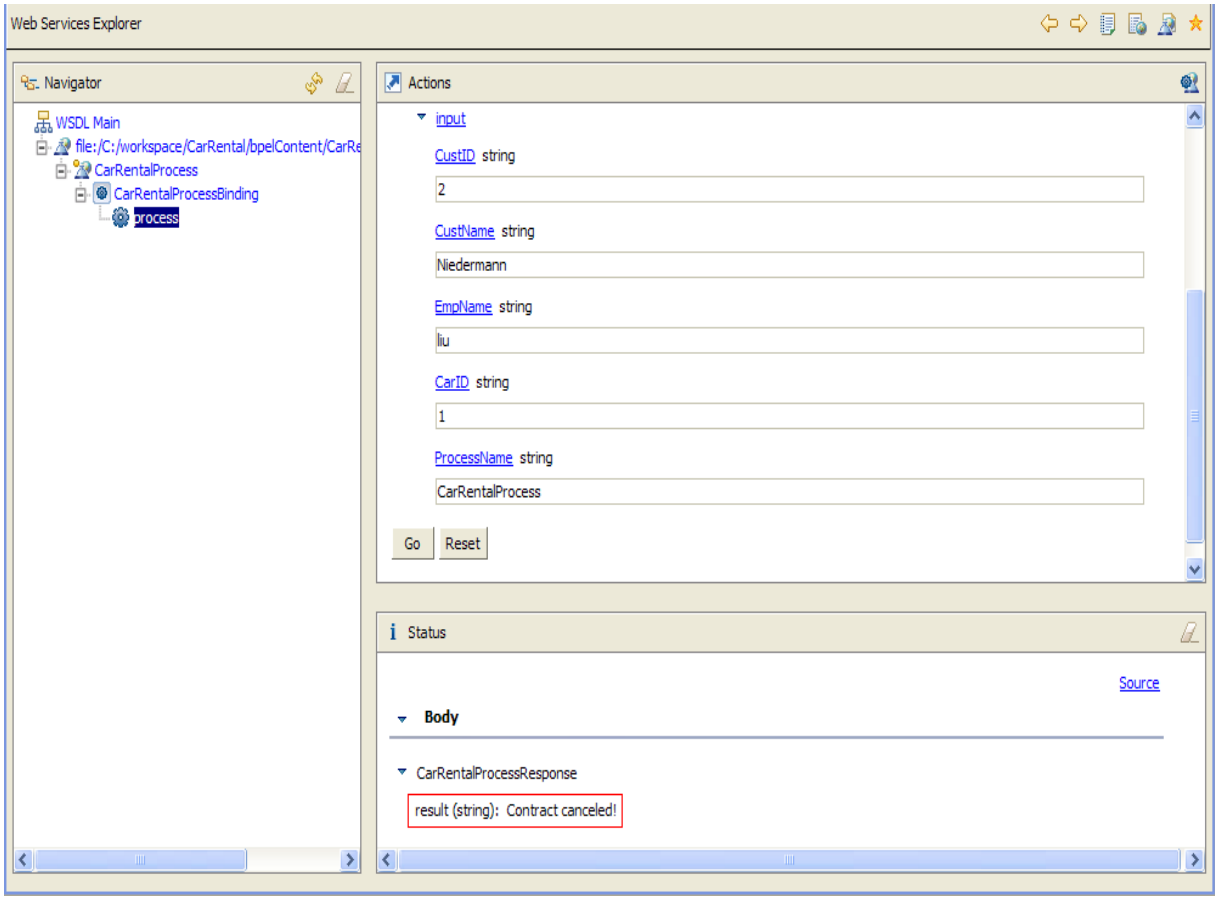


Abbildung 41 : Resultat mit „Contract canceled“

Nur die Regel „RuleCustomer“ wird hier aufgerufen und die entsprechende Informationen werden in der Tabelle „RuleLogTable“ gespeichert.

EDIT	RULEID	TIMESTART	PROCESSNAME	RULEMASSAGE	RULESTATUS	RULENAME
	182	2011/08/27:11:12:07	CarRentalProcess	-	OK	Rules
	181	2011/08/27:11:12:03	CarRentalProcess	-	OK	RuleCustomer
	183	2011/08/27:11:24:04	CarRentalProcess	-	OK	RuleCustomer

row(s) 1 - 3 of 3

Abbildung 42 : Informationen in der Tabelle „RuleLogTable“ für die einzige aufgerufenen Regel

6. Evaluation

In diesem Kapitel wird der Extended Business Rule Manager ausgewertet. Dazu nehmen wir zuerst 6 Regeln als Beispiele an.

- Regel 1 und 2 bezieht sich nur auf die Prozessdaten (also z.B. Wenn „CustID==1“, soll die Resultat „true“ sein, sonst „false“ oder Wenn „CustName==Niedermann“, soll die Resultat „true“ sein, sonst „false“).

- Regel 3 und 4 bezieht sich nicht nur auf die Prozessdaten (Diese Prozessdaten werden nur zur Findung des Wertes von der operativen Daten verwendet), sondern auch die operativen Daten (also z.B. Wenn „Rental==0“, soll die Resultat „true“ sein, sonst „false“ oder Wenn „Age >20“, soll die Resultat „true“ sein, sonst „false“).

- Regel 5 und 6 bezieht sich sowohl auf die Prozessdaten als auch die operativen Daten (also z.B. Wenn „Rental == 0“ und „CustID==1“, soll die Resultat „true“ sein, sonst „false“ oder Wenn „Age>20“ und „CustName==Niedermann“, soll die Resultat „true“ sein, sonst „false“).

Dann setzen wir die 6 Regeln sowohl als normalen Business Rules ohne den Extended Business Rule Manager als auch als extended Business Rules mit dem Extended Business Rule Manager in den Geschäftsprozess ein:

Regel	"Normalen Business Rules"	"Extended Business Rules"
1	√	√
2	√	√
3	x	√
4	x	√
5	x	√
6	x	√

Tabelle 1 : Die Auswertung mit 3 Regeln

Als normalen Business Rules können nur Regel 1 und 2 beschrieben und auch geprüft werden, dagegen können alle 6 Regeln als extended Business Rules beschrieben und auch geprüft werden. Also der extenden Business Rule Manager funktioniert mit allen 6 Regeln.

7. Zusammenfassung und Ausblick

In diesem Kapitel wird nun zusammengefasst, was in dieser Arbeit gemacht wurde. Darüber hinaus wird ein Ausblick für zukünftige Arbeiten gegeben.

In dieser Diplomarbeit wurden der Entwurf, die Implementierung und der Testen des extended Business Rule Managers betrachtet. Dazu wurde zunächst eine grafische Benutzeroberfläche zur Definition Business Regeln bereitgestellt. Dann kamen Web Services als Regel-Prüfer zum Einsatz. Diese Regel-Prüfer müssen nicht nur die erstellten Regeln prüfen, sondern auch sich mit einer Datenbank zur Findung der Werte der operativen Daten verbinden. Schließlich wurde die erstellten Web Services in einem Geschäftsprozess getestet. Darüber hinaus wurden alle benötigen Technologien auch vorgestellt.

Ausblick

In dieser Diplomarbeit wurde jede Regel in einer XML-Datei gespeichert. Also Jede Regel entspricht einer XML-Datei. Das ist nicht günstig für den Entwurf des Business Rule Managers. Denn wenn in einem Geschäftsprozess mehr als eine Regel geprüft werden, müssen auch mehr als ein Web Service eingerichtet werden. Eine Lösung dafür ist die Integration aller Regeln in einer XML-Datei. Dann wird jede Regel nach einem Algorithmus sortiert und nur von einem Web Service aufgeruft.

Darüber hinaus wurde die Aktion der Regeln nur einfach realisiert. Also wenn die Bedingungen erfüllt werden, wird nur „true“ ausgegeben, sonst nur „false“. Diese ist für die Anwendung in der Geschäftsprozesse sehr eingeschränkt. Deswegen muss die Aktion komplexer implementiert werden (z.B. Nicht nur true/false, sondern weitere Aktionen auslösen). Eine Idee ist der Entwurf einer externen Komponete (Orchestration von Web Services).

Literaturverzeichnis

[DBR00] Defining Business Rules ~ What Are They Really? the Business Rules Group formerly, known as the GUIDE Business Rules Project, Final Report Revision1.3July,2000.URL http://www.businessrulesgroup.org/first_paper/br01c0.htm. (Zitiert auf den Seiten 4,5 und 6)

[EML] Extensible Markup Language URL http://de.wikipedia.org/wiki/Xml

[DOM] Document Object Model
URL http://en.wikipedia.org/wiki/Document_Object_Model

[SAX] Simple API for XML
URL http://en.wikipedia.org/wiki/Simple_API_for_XML

[EH] XML Pull Parsing, Elliotte Rusty Harold, Software Development 2002 East
URL http://www.ibiblio.org/xml/slides/sd2002east/xmlpull/XML_Pull_Parsing.html

[XS] XML Schema URL http://de.wikipedia.org/wiki/XML_Schema

[BV08] Serviceorientierte Architekturen – SOA B. Vikum 5.Juli 2008(Zitiert auf den Seiten 6 und 7)

[SOAP] Simple Object Access Protocol URL http://de.wikipedia.org/wiki/SOAP

[WSBPEL] Web Services Business Process Execution Language Version 2.0 OASIS Standard, Diane Jordan,John Evdemon,Alexandre Alves,Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera,Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin,Vinkesh Mehta, Satish Thatte,Danny van der Rijn,Prasad Yendluri, Alex Yiu, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[SOA] Service orientierte Architekturen
URL http://itmkb.campus02.at/index.php/Service_orientierte_Architekturen

[NRM] Deep Business Optimization: A Platform for Automated Process Optimization, F. Niedermann, S. Radeschütz, B. Mitschang (Zitiert auf den Seiten 2 und 3)

[NRM] Business Process Optimization using Formalized Optimization Patterns
F. Niedermann, S. Radesch and B. Mitschang (Zitiert auf den Seiten 2 und 3)

[CU] Java ist auch eine Insel, C. Ullenboom, Programmieren für die Java 2-Plattform in der Version 5

[2DPJDG07] Oracle® Database Express Edition, 2 Day Plus Java Developer Guide 10g February 2007

[BM06] Hibernate , Prof. Dr. B. Müller 28. August 2006 (Zitiert auf den Seiten 1,2 und 13)

[HE] Hibernate (Framework) URL http://de.wikipedia.org/wiki/Hibernate_(Framework)

[SH08] First Hibernate example, S. Hennebrueder, February, 9th 2008 (Zitiert auf den Seiten 2)

[ODE10] Apache ODE – User Guide. Apache Software Foundation, 2010.

URL http://ode.apache.org/user-guide.html (Zitiert auf Seite 35)

[ODE] Apache ODE
URL http://ode.apache.org/architectural-overview.data/OdeArchitecture.png

[TALT] Tomcat als lokaler Testserver

URL http://www.jsptutorial.org/content/tomcat#innerLink2

[MT] Eclipsing Axis!, MARC TEUFEL (Zitiert auf den Seiten 2)

Alle Links wurden zum letzten mal am 16. Oktober 2011 abgerufen.

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen verwendet zu haben.

_____

(Jian Liu)