Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3124

Ein Verzeichnisdienst für Services und Ressourcen

Hao Jin

Studiengang: Informatik

Prüfer: Jun.-Prof. Dr.-Ing. Dimka Karastoyanova

Betreuer: Dipl.-Inf. Mirko Sonntag

begonnen am: 30. September 2010

beendet am: 01. April 2011

CR-Klassifikation: H.3.3, H.3.5, D.2.2

Inhaltverzeichnis

1 Einleitung

SOA ist ein besonderer Architekturstil, der lose Kopplung und dynamisches Binding von Services behandelt. Web Service ist eine Implementierung von SOA. Es ist eine Software Anwendung, die von anderen Software Anwendungen verwendet werden kann. Die funktionalen Eigenschaften von Service werden durch WSDL beschrieben, WS-Policy spezifiziert die nicht-funktionalen Eigenschaften von Service.

1.1 Motivation

Um die Web Services verwenden zu können, müssen die Service vorher registriert werden, dann können die Services mit ihren Eigenschaften gesucht, gefunden und ausgewählt werden. UDDI ist dafür zuständig, aber in der Praxis wird kommt es vor, dass UDDI zu schwergewichtig und zu komplex ist. Es ist deswegen erwünscht, ein neues leichtgewichtiges Service anzubieten, das die Registrierung und das Suchen von Web Services ermöglichen kann. Außerdem sind die Datenbanken ebenfalls Resources und werden für die Ausführung von wissenschaftlichen Anwendungen benötigt. Um die Datenbanken einzusetzen, ist es auch nötig, die Datenbanken zu registrieren, zu suchen, zu finden und auswählen zu können.

1.2 Aufgabenstellung

Die Aufgabe dieser Arbeit ist ein Konzept und die prototypische Implementierung eines leichtgewichtigen Verzeichnisdient für Services und Resources zu entwickeln. Der Verzeichnisdienst wird in der Arbeit auch „Registerservice“ genannt. Das Registerservice ist für die Registrierung, das Suchen und die verwaltung von Services, Datenbanken und Anbieter zuständig. Für die Registrierung und das Suchen von Services werden sowohl die funktionalen Eigenschaften (WSDL) als auch die nicht-funktionale Eigenschaften (WS-Policy) berücksichtigt. Für die Registrierung und das Suchen von Datenbanken wird ebenfalls die Anforderung (Policy) für Nutzung der Datenbank beachtet.

1.3 Gliederung

Der weitere Aufbau der Arbeit wird in diesem Abschnitt beschrieben. In Kapitel 2 werden Grundlagen von Web Services und das Sofware Fragmento vorgestellt, die zum Verständnis der Arbeit dienen sollen. Im Kapitel 3 wird das Konzept der Arbeit vorgestellt. In diesem Kapitel wird beschrieben, welche Funktionalitäten und wie die Funktionalitäten vom Service angeboten wird. Im Kapitel 4 wird spezifiziert, wie das im Kapitel 3 vorgestellte Konzept realisiert wird. Schließlich wird im Kapitel 5 diese Arbeit zusammengefasst.

2 Grundlagen

2.1 SOA

SOA ist die Abkürzung für Service-Oriented Architecture. Es ein besonderer Architekturstil, der Lose Kopplung und dynamisches Binding von Services behandelt [1]. Die Prinzipien können durch ein Dreieck beschrieben werden.

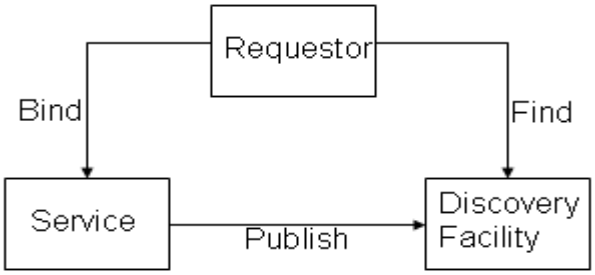

Abbildung 1: SOA Dreieck

Ein Service-Anbieter kann ein Service publizieren, die von Anderen benutzt werden können. Ein Service Nuzter verwendet das Discovery Facility, um die Services zu suchen. Der Nutzer wählt ein gewünschtes Service aus und bindet das Service.

2.2 Web Service

Web Service ist eine Implementierung von SOA, ein Web Service ist eine Software-Anwendung, die von anderen Software-Anwendugen verwendet werden kann. Eine genauere Definition wird in [3] gegeben:

"A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."

Ein Web Service wird durch Web Service Description Language (WSDL) und Web Service Policy beschrieben, WSDL spezifiziert die funktionalen Eigenschaften, die Policy beschreibt die nicht-funktionalen Eigenschaften von einem Web Service. SOAP definiert das

Message-Format und das Message-Verarbeitungsmodel und wird von Web Service zur Kommunikation verwendet. UDDI ist zuständig für die Registrierung und das Suchen für Web Services.

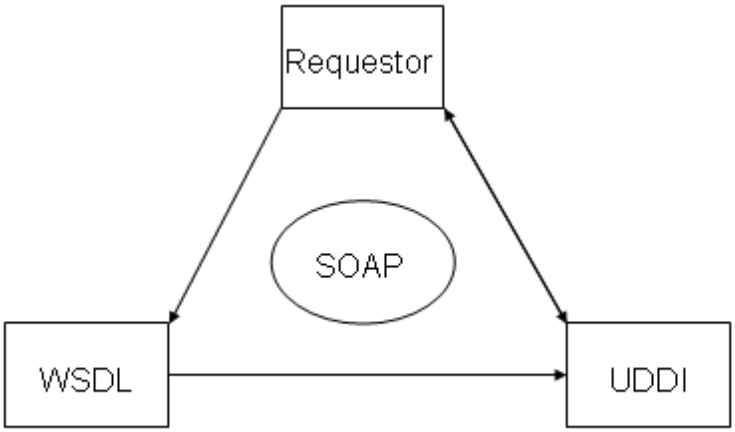

Abbildung 2: Web Service Dreieck

2.3 WSDL

WSDL ist eine sehr wichtige Grundlage für Web Service-Anwendung, WSDL ist eine Abkürzung für Web Service Description Language, durch WSDL können Web Service auf standardisiertem Weg beschrieben werden [2]. Ein WSDL-Dokument besteht aus zwei Teilen: einem wiederverwendbaren abstrakten Teil und einem konkretem Teil. Der abstrakte Teil von WSDL beschreibt das operationale Verhalten von Web Service durch das Auflisten der hereinkommenden und hinausegehenden Messages von Services. Der konkrete Teil von WSDL beschreibt, wo und wie auf ein Web Service zugegriffen werden kann [1].

2.3.1 WSDL 1.1

Definitionen

Types: Definitionen der nötigen Daten Typen

Message: Abstrakte Definition von ausgetauschten Daten

Operation: Abstrakte Aktionen, die durch das Service unterstützt werden

Port Type: Eine Menge von Operationen , die durch einen Endpoint unterstützt werden

Binding: Concrete Protocol und Daten Format, die verwendet werden, um einen Port Type zu implementieren.

Port: Einzelnes individuelles Endpoint, das durch eine Netzwerkadresse identifiziert ist, unterstützt ein bestimmtes Binding.

Service : Eine Menge von bezogenen Endpoints

Struktur

WSDL1.1 Dokument enthält normalerweise zwei Gruppen von Definitionen:
ein abstrakten Teil, der spezifiziert, was das Service anbietet und drei Elemente enthält: <types>, <message> und <portType>. Ein konkreter Teil, der spezifiziert, wie und wo auf ein Service zugegriffen werden kann und zusätzlich noch die zwei Elemente <binding> und <service > enthält (gezeigt in Abbildung 3).

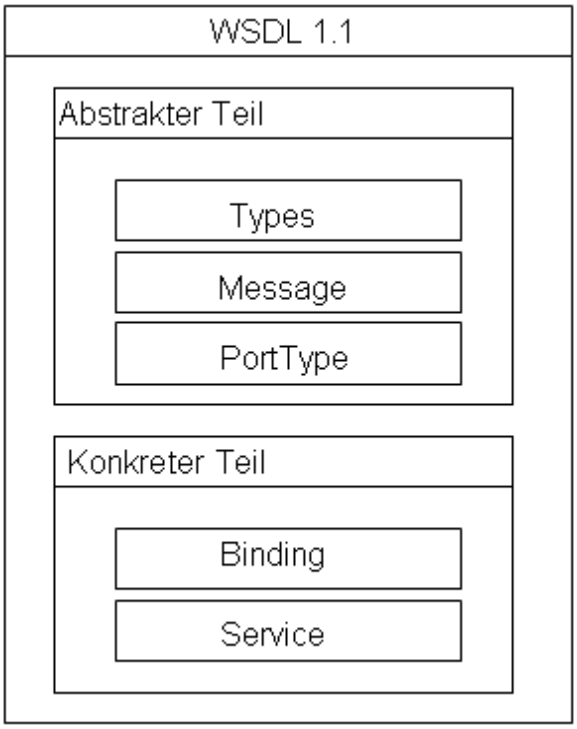

Abbildung 3: WSDL 1.1 Struktur

Definitionen

Das <definitions> Element ist das Wurzelelement eines WSDL-Dokumentes. In dem werden ein einziges Targetnamespace und alle nötigen Namespaces definiert.

Das folgende Beispiel zeigt wie die Namespaces im <definitions> Element definiert werden.

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.jin.ustutt.da/RegisterService/"
xmlns:da="http://www.jin.ustutt.da/RegisterService/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="RegisterService" targetNamespace="http://www.jin.ustutt.da/RegisterService/">
</wsdl:definitions>
```

Listing 1: WSDL Element definitions

Types

Die Struktur von <types> Element

```
<types>
  <xsd:schema.../>*
</types>
```

Listing 2: WSDL Element types [4]

Mit Hilfe vom XML-Schema können die Datentypen im <types> Element definiert werden. Die Datentypen werden beim Definieren der Messages referenziert. Die definitionen der Datentypen mit XML Schema werden entweder direkt im <types> Element umschlossen oder in einer externen Datei gespeichert. In diesem Fall wird die Importanweisung im <types> Element verwendet, um die Definitionen der Datentypen zu importieren. [2]
Im folgendem Beispiel sind ein komplexer Datentyp „authenticateProviderType“ und zwei weitere Elemente, „authenticateProviderRequestMessage“ und „authenticateProviderResponseMessage“ im <types> Element definiert.

```
<wsdl:types>
  <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetnamespace="http://www.jin.ustutt.da/RegisterService/">
    <xsd:complexType name="authenticateProviderType">
      <xsd:sequence>
        <xsd:element name="email" type="xsd:string" />
        <xsd:element name="password" type="xsd:string" />
```

```
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="authenticateProviderRequestMessage">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="authenticateProvider"
          type="tns:authenticateProviderType" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="authenticateProviderResponseMessage">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="providerId" type="xsd:long" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
```

Listing 3: Beispiel von WSDL Element types

Messages

Die Struktur vom <message> Element:

```
<message name="nmtoken" >*
  <part name = "nmtoken" elment="qname"? type="qname"? />*
</message>
```

Listing 4: WSDL Element message [4]

Das <Message> Element spezifiziert die Messages, die beim Aufruf einer Opertion von einem Web Service ausgetauscht werden. Eine Message besteht aus einem oder mehreren Parts. Parts sind typisiert und spezifiziert alle Parameter von einem RPC. Messsage ist abstrakt, das konkrete Format wird durch das Binding beschrieben. [6]

Im folgenden Beispiel werden zwei <message> Elemente „authenticateProviderRequest“ und „authenticateProviderResponse“ definiert

```
<wsdl:message name="authenticateProviderRequest">
   <wsdl:part name="parameter"
element="tns:authenticateProviderRequestMessage" />
</wsdl:message>
<wsdl:message name="authenticateProviderResponse">
   <wsdl:part name="parameter"
   element="tns:authenticateProviderResponseMessage" />
</wsdl:message>
```

Listing 5: Beispiel vom WSDL Element message

PortTypes

Die Struktur vom <portType> Element

```
<portType name="nmtoken">
   <operation name="nmtoken" parameterOrder="nmtokens“?>*
      <input name="nmtoken"? message="qname"/>?
      <output name="nmtoken"? message="qname"/>?
      <fault name="nmtoken" message="qname"/>*
   </operation>
</portType>
```

Listing 6: WSDL Element portType [4]

Das <portType> Element spezifiziert das Interface eines Web Service und ist eine Menge von abstrakten Operationen und Messages. Vier Arten von Operationen werden unterstützt [1]:

- **One-Way**, ein Message wird zum Service geschickt. Das Service erzeugt kein Response Message.
- **Request-Response**, ein Message wird zum Service geschickt. Das Service erzeugt eine Respons-Message.
- **Solicit-Response**, ein Service schickt zuerst eine Message und erhält eine Response-Message.
- **Notification**, ein Service schickt eine Message und erhält keine Response-Message.

Im folgenden Beispiel wird eine Request-Response Operation „authenticateProvider“ mit Input Message „tns:authenticateProviderRequest“ und Output Message „tns:authenticateProviderResponse“ im <portType> Element spezifiziert.

```
<wsdl:portType name="RegisterService">
   <wsdl:operation name="authenticateProvider">
      <wsdl:input message="tns:authenticateProviderRequest" />
      <wsdl:output message="tns:authenticateProviderResponse" />
   </wsdl:operation>
</wsdl:portType>
```

Listing 7: Beispiel vom WSDL Element portType

Bindings

Die Struktur vom <binding> Element:

```
<binding name="nmtoken" type="qname">
   <-- extensibility element  ->*
   <operation name="nmtoken">*
      <-- extensibility element  -->*
      <input name="nmtoken"?>?
         <-- extensibility element -->*
      </input>
      <output name="nmtoken"?>?
         <-- extensibility element -->*
      </output>
      <fault name="nmtoken">*
         <-- extensibility element-->*
      </fault>
   </operation>
</binding>
```

Listing 8: WSDL Element binding [4]

Das <binding> Element gehört zum konkreten Teil eines WSDL-Dokumentes, das spezifiziert auf Service zugegriffen werden kann. WSDL hat selber keine Standardmethode um Messages darzustellen, WSDL verwendet die Erweiterbarkeit um zu spezifizieren, wie die Messsages durch die Nutzung von SOAP, HTTP, MIME etc. ausgetauscht werden [1].

SOAP Binding

Das SOAP Binding von WSDL ist eine Erweiterung vom WSDL, es beschreibt, wie eine SOAP-Message über ein Netzwerk übertragen wird. Ein SOAP Message kann über verschiedene Transportprotokolle wie HTTP, JMS, usw. übertragen werden. Eine Binding-Spezifikation ist für einen einzelnen Schritt zwischen den Konten gültig, nicht aber für den ganzen Message Path [1]. Meistens wird das HTTP-Binding von SOAP Anwendungen verwendet.

Die SOAP Web-Methode wird vom HTTP-Binding genutzt, damit es die Anwendungen ermöglicht, eine Web Methode auszuwählen (GET oder POST).

Die Struktur sieht demnach wie folgt aus:

```
<soap:binding
transport="uri"?
style="rpc | document"?>
```

Listing 9: Extensibility Element soap binding [6]

Das verwendete Transportprotokoll kann HTTP, SMTP, FTP oder JMS usw. sein und wird durch entsprechende URI dargestellt.

Es werden zwei Styles im SOAP Binding definiert:

- Dokument Style, alle Parts vom <message> Element werden als Kinder des <Body> Elements in SOAP Envelope eingefügt.
- RPC Style, alle Parts vom <message> Element werden in einem äußeren Element eingepackt und das einzelne eingepackte Element wird als einziges Kind von <body> Element in SOAP Envelope eingefügt.

SOAP Operation

```
<soap:operation
soapAction="uri"?
style="rpc | document"?>
```

Listing 10: Extensibility Element soap binding [6]

Das <soap:operation> Element definiert ein Action-URI, das die Operation eindeutig identifizieren kann. Im Element werden zwei Styles von Operationen definiert:

- Dokument Style, die Messages sind Dokumente
- RPC Style, die Messages sind Parameter

SOAP Body

```
<soap:body
parts="nmtokens"?
use="literal | encoded"?
encodingStyle="uri-list"?
namespace="uri"?>
```

Listing 11: Extensibility Element soap body [6]

Das <soap: body> Element spezifiziert wie die Message-Parts in SOAP Body Element auftauchen. Die Parts einer Message können entweder abstrakte Typ-Definitionen oder konkrete Schemadefintionen sein.

SOAP Header

```
<soap:header
message="qname"
part="nmtoken"
use="literal | encoded"?
encodingStyle="uri-list"?
namespace="uri"?>*
```

Listing 12: Extensibility Element soap header [6]

Das <soap:header> Element spezifiziert, welcher Part von welcher Message als Header im SOAP Envelope vom entsprechenden SOAP Request enthalten ist [6].

Im folgenden Beispiel ist ein <wsdl:binding> Element mit dem Name „RegisterServiceSOAP“ definiert. Das Binding verwendet das Port Type „tns:RegisterService“ wie auch das HTTP als Transport. Alle Parts von <message> Elementen, die von der Operation „authenticateProvider“ verwendet werden, werden als Kinder des <body> Elements in SOAP Envelope eingefügt.

```
<wsdl:binding name="RegisterServiceSOAP" type="tns:RegisterService">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="authenticateProvider">
      <soap:operation soapAction=
      "http://www.jin.ustutt.da/RegisterService/authenticateProvider" />
      <wsdl:input>
         <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal" />
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

Listing 13: Beispiel WSDL Element binding

Service

<service> ist das letzte Element der WSDL-Beschreibung. Das Element gehört zum konkreten Teil des WSDL-Dokumentes. Es hat folgende Struktur:

```
<wsdl:service name="nmtoken">
   <wsdl:port name="nmtoken" binding="qname">*
      <soap:address location="uri"/>
   </wsdl:port>
</wsdl:service>
```

Listing 14: WSDL Element service

Das Element <service> spezifiziert, wo ein Service durch das Kindelement <port> gefunden werden kann. Ein <service> Element kann beliebig viele <port> Elemente enthalten. Ein <port> Element beschreibt, wo ein Port Type durch ein gegebenes Binding angeboten wird. Das Attribut „binding“ spezifiziert, welches Binding vom <port> Element eingesetzt wird. Das Kindelement <soap:address> vom <port> beschreibt die Adresse vom Port. Das folgende Beispiel zeigt, dass ein <service> Element mit dem Namen „RegisterService“ definiert wird.

```
<wsdl:service name="RegisterService">
   <wsdl:port name="RegisterServiceSOAPPort"
   binding="tns:RegisterServiceSOAP" >
      <soap:address
      location="http://localhost:8080/axis2/services/RegisterService/"/>
   </wsdl:port>
</wsdl:service>
```

Listing 15: Beispiel WSDL Element service

2.3.2 WSDL2.0

Struktur

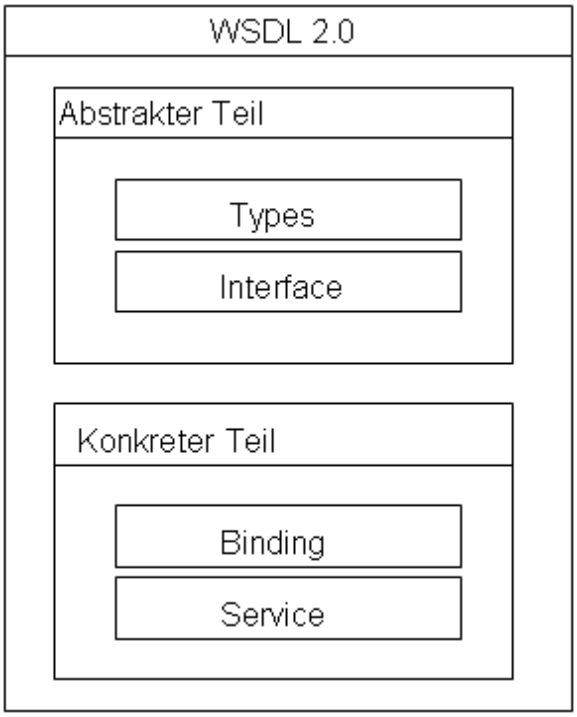

Abbildung 4: WSDL 2.0 Struktur

WSDL2.0 Dokument hat das Wurzelelement <description>, in dem ein abstrakter Teil und ein konkreter Teil definiert werden (gezeigt in Abbildung 4). Der abstrakte Teil enthält die Elemente <types> und <interface>, die Elemente <binding> und <service> gehören zum konkreten Teil.

Im vergleich mit WSDL1.1 ist das Wurzelelement nicht mehr <defintions> in WSDL2.0, sondern <description>, das Element <portType> wird durch <interface> ersetzt. Der größte Unterschied ist, dass das Element <message> in WSDL2.0 eliminiert wird. Die Messages werden in <interface> definiert. Bei der Defintion der Datentyen stehen außer XML Schema noch RelaxNG und DTDs zur Verfügung. Außerdem wird das Konzept der Vererbung eingeführt.Ein Interface kann von einem oder mehreren bereits definierten Interfaces abgeleitet werden [2].

2.4 SOAP

SOAP ist eine Message-Architektur und ein Message-Verarbeitungsmodel. Eine SOAP-Message ist die Grundeinheit der Kommunikation zwischen SOAP Konten [1].

SOAP Message-Struktur

Eine SOAP-Message besteht aus, wie in Abbildung 5 gezeigt, einem Envelope Element. Das Element besteht aus einem Body-Element und einem optionalen Header-Element, während im Body-Element die eigentlichen Nutzdaten stehen, können im Header Element die Meta-Informationen, beispielsweise zum Routing, zur Verschlüsselung oder zur Transaktionsidentifizierung, untergebracht werden.

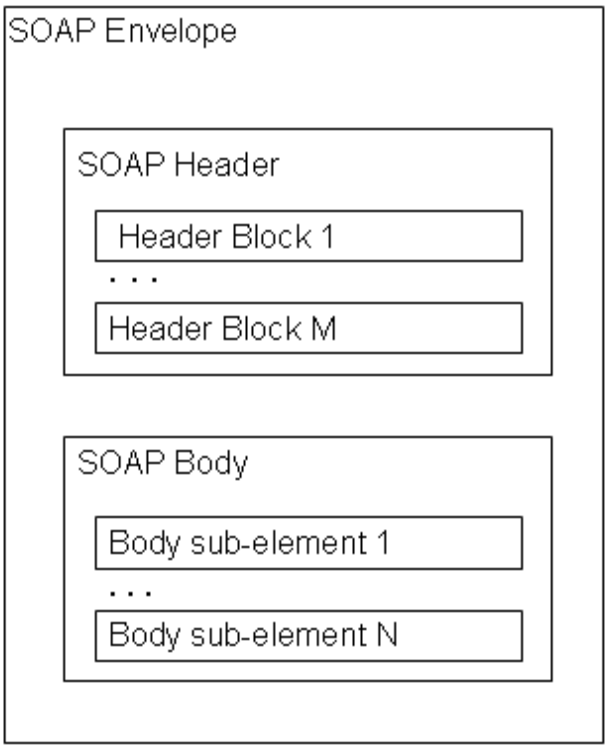

Abbildung 5: SOAP Message Struktur [1]

Das folgende Beispiel zeigt eine SOAP Requst-Message und eine Response-Message an:

Das Request-Message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:reg="http://www.jin.ustutt.da/RegisterService/">
```

```
<env:Header/>
<env:Body>
<reg:authenticateProviderRequestMessage>
<reg:authenticateProvider>
<email>abc@hotmal.com</email>
<password>password1234</password>
</reg:authenticateProvider>
</reg:authenticateProviderRequestMessage>
</env:Body>
</env:Envelope>
```

Listing 16: Beispiel der SOAP Request Message

Die Response Message :

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:reg="http://www.jin.ustutt.da/RegisterService/">
<env:Header/>
<env:Body>
<reg:authenticateProviderResponseMessage>
<providerId>5</providerId>
</reg:authenticateProviderResponseMessage>
</env:Body>
</env:Envelope>
```

Listing 17: Beispiel der SOAP Response Message

Definitionen

Sender: Knoten, das eine Message sendet.

Receiver: Knoten, das eine Message empfängt.

Message path: Menge von Knoten, über die eine einzelne Message passt, inklusiv dem initial Sender, null oder mehrere Intermediaries und ein Ultimate-Receiver.

Initial sender: Knoten, der eine Message erzeugt und welcher der Ausgangspunkt von einem Message Path ist.

Intermediary: Knoten, der sowohl ein Sender als auch ein Receiver ist.

Ultimate-Receiver: Receiver ist der Zielort einer Message und zuständig für die Verarbeitung des Inhalts vom Body and allen Header-Blöcken, die auf dem Receiver gezielt sind. Ultimate-Receiver kann nicht ein Intermediary für die gleiche SOAP Message sein.

Verarbeitungsmodell von SOAP

Das Verarbeitungsmodel spezifiziert, wie ein Knoten eine SOAP-Message verarbeitet, falls es die SOAP-Message empfängt. Wenn ein Initial-Sender eine SOAP Message zu einem Ultimate-Receiver schickt, können noch mehrere Knoten, auch Intermediaries genannt, zwischen den beiden Knoten vorkommen, deswegen ist es notwendig festzustellen, wie die Knoten die Message verarbeiten soll. Normalerweise ist das Body-Element von einer SOAP-Message auf dem Ultimate-Receiver gezielt. Die Intermediaries und der Ultimate-Receiver müssen oder können die Header-Blöcke verarbeiten, dies ist abhängig vom „role“-Attribut der Header-Blöcke.

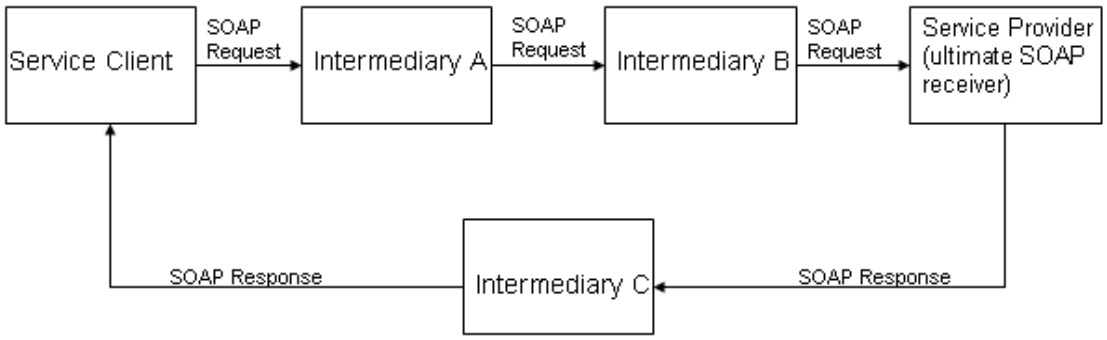

Abbildung 6: Verarbeitungspfad einer SOAP Message mit Intermediaries [2]

2.5 WS-Policy

WS-Policy [14] beschreibt die nicht-funktionalen Eigenschaften von Web Services und ermöglicht dem Serviceanbieter die Richtlinien bezüglich Sicherheit, Qualität und Version seines Services in Form von XML-Daten für den Servicenutzer bereitzustellen. Diese Policies werden dann an entsprechenden Stellen in die WSDL-Datei des Services eingefügt.

Umgekehrt kann auch ein Servicenutzer seine Anforderung an einem Service in Form von Policies formulieren. Die Intersection der Policies vom Serviceanbieter und vom Servicenutzer dient zum Wahl eines Web-Services.

2.5.1 Definitionen

Eine Policy ist eine potenziell leere Menge von Policy Alternatives.
Eine Policy Alternative ist eine potenziell leere Menge von Policy Assertions.
Eine Policy Assertion ist eine Anforderung, eine Fähigkeit oder eine Eigenschaft des Verhaltens.
Ein Policy Assertion Type repräsentiert eine Klasse von Assertions und impliziert ein Schema für die Assertions.
Ein Policy Subject ist eine Entität (z.B, Endpoint, Message, Resource, Interaction), mit der eine Policy assoziiert werden kann.
Ein Policy Scope ist eine Menge von Policy Subjects, auf die eine Policy zutreffen kann.
Ein Policy Attachment ist ein Mechanismus für die Assoziierung einer Policy mit einem oder mehreren Policy Scopes.
Eine effektive Policy ist eine Kombination von relevanten Policies für ein gegebenes Policy Subject

2.5.2 Policy Model

Assertion

Eine Policy Assertion spezifiziert ein Verhalten, das eine Anforderung (oder Fähigkeit) von einem Policy Subject ist. Assertions geben Domain-spezifische Semantik, wie Sicherheit und Transaktionen an und werden verlangt in separaten, Domain-spezifischen Spezifikationen zu definieren. Ein Typ der Assertions ist stark abhängig von der Autoren-definierten Domain und nur durch den QName vom Wurzelelement der Assertion identifiziert[14].

Alternative

Eine Policy Alternative ist eine potenziell leere Menge von Policy Assertions. Eine Alternative mit leerer Assertion gibt kein Verhalten an, eine Alternative mit einer oder mehreren Assertions zeigt nur das von diesen Assertions implizierte Verhalten an. Eine Policy Alternative kann mehrere Assertions von einem Typ sein, die Assertions innerhalb einer Alternative ist nicht geordnet.

Policy

Eine Policy ist eine potenziell leere Menge von Policy Alternatives. Eine Policy mit leeren Alternatives wird als eine leere Policy betrachtet. Eine Policy mit einer oder mehreren Policy Alternatives bietet die Möglichkeiten an, die Anforderungen auszuwählen, die durch die Policy Alternatives in der Policy spezifiziert sind. Die Alternatives innerhalb einer Policy sind nicht geordnet.

2.5.3 Operator

WS-Policy spezifiziert drei Operatoren: **Policy**, **All** und **ExactlyOne**. Alle die, die direkt in einem All Operator stehenden Assertions und Operatoren müssen erfüllt sein. Der ExactlyOne-Operator verlangt, dass es genau eine der direkt im Operator stehenden Assertions und Operatoren gelten muss. Das folgende Beispiel betrifft eine Policy mit vier Alternatives. Die erste Alternative schließt zwei Assertions um, die zweite enthält auch zwei Assertions, die dritte und vierte haben jeweils drei Assertions.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:All> <A/> <D/> </wsp:All>
    <wsp:All> <B/> <C/> </wsp:All>
    <wsp:All> <B/> <C/> <D/> </wsp:All>
    <wsp:All> <A/> <B/> <D/> </wsp:All>
  </wsp:ExactlyOne>
<wsp:Policy>
```

Listing 18: Beispiel vom WS-Policy

2.5.4 Normalform

Um die interoperability der Policies einfach zu implementieren, wird eine Normalform definiert, die vor allem die Voraussetzung für den Intersection-Algorithmus darstellt. Eine Policy in Normalform ist wie folgt definiert:

```
<wsp:Policy ... >
   <wsp:ExactlyOne>
      (<wsp:All>
         ( <Assertion ...> ... </Assertion> )*
      </wsp:All>)*
   </wsp:ExactlyOne>
</wsp:Policy ... >
```

Listing 19: Normalform vom WS-Policy [14]

Das folgende ist ein Beispiel einer Policy der Normalform:

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsp:All> <A/> <B/> </wsp:All>
      <wsp:All> <C/> </wsp:All>
      <wsp:All> <D/> <E/> </wsp:All>
   </wsp:ExactlyOne>
<wsp:Policy>
```

Listing 20: Beispiel einer Policy in der Normalform

2.5.5 Kompaktform

In einer Policy der Kompaktform ist es erlaubt, dass eine Assertion optional sein kann.

```
<Assertion(wsp:Optional="xs:boolean”)?...> … </Assertion>
```

Listing 21: Optionale Assertion

Ist der Wert „True“, ist die Assertion äquivalent zu:

```
<wsp:ExactlyOne>
   <wsp:All> <Assertion ...> ... <Assertion/> </wsp:All>
```

```
   <wsp:All/>
</wsp:ExactlyOne>
```

Listing 22: Die äquivalente Form der optionalen Assertion mit Optional=true

Ist der Wert „False“, ist die Assertion äquivalent zu:

```
<wsp:ExactlyOne>
   <wsp:All> <Assertion ...> ... <Assertion/> </wsp:All>
</wsp:ExactlyOne>
```

Listing 23: Die äquivalente Form der optionalen Assertion mit Optional=false

Außerdem ist eine beliebige Reihenfolge von Operatoren möglich. Die Operatoren können rekursiv verschachtelt werden. Im Vergleich mit der Normalform ist die Flexibilität der Vorteil der Kompaktform.

Das folgende Beispiel ist ein Beispiel einer Policy der Kompaktform:

```
<wsp:Policy>
   <wsp:All>
   <wsp:ExactlyOne> <C/> <B/> </wsp:ExactlyOne>
   </wsp:All>
   <wsp:All>
      <wsp:Policy>
         <wsp:ExactlyOne> <A/> <C/> </wsp:ExactlyOne>
      </wsp:Policy>
   </wsp:All>
</wsp:Policy>
```

Listing 24: Beispiel einer Policy im Kompaktform

äquivalent zu der Normalform:

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsp: All> <A/> <B/> </wsp: All>
      <wsp: All> <A/> <C/> </wsp: All>
      <wsp: All> <B/> <C/> </wsp: All>
      <wsp: All> <C/> </wsp: All>
```

```
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 25: Beispiel der äquivalenten Policy in der Normalform

2.5.6 Intersection der Policies

Die Policy Intersection wird verwendet um die Kompatibilität von zwei Policies festzustellen und ist eine Funktion, die zwei Policies nimmt und eine Policy liefert.
Um die Intersection von zwei Policies auszuführen, müssen die beiden Policies in der Normalform sein. Zwei Policies sind kompatibel, wenn eine Alternative in einer Policy mit einer Alternative in einer anderenPolicy kompatibel ist. Das Ergebnis der Intersection ist eine Policy mit allen kompatiblen Alternatives.
Zwei Policy Alternatives sind kompatibel, wenn jede Assertion in einer Alternative mit einer Assertion in einer anderen Alternative ist.
Zwei Policy Assertions sind kompatibel, wenn sie den gleichen Typ haben. Wenn eine Assertion eine nested Policy Expression enthält, muss die andere Assertion auch eine nested Policy Expression enthalten und die Alternative in der nested Policy Expression von einer Assertion muss mit der Alternative in der nested Policy von einer anderen Assertion kompatibel sein.
Im Folgenden sind zwei Policies dargestellt. Die Policy 1 enthält zwei Alternatives A1 und A2, die Policy 2 enthält zwei Alternatives A3 und A4.

```
<wsp:Policy
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypoliy/200702"
xmlns:wsp="http://www.w3.org/ns/ws-policy" >
<! -- Policy P1 -->
  <wsp:ExactlyOne>
    <wsp:All> <! -- Alternative A1 -->
      <sp:SignedElements>
        <sp:XPath>/S:Envelope/S:Body</sp:XPath>
      </sp:SignedElements>
```

```
      <sp:EncryptedElements>
         <sp:XPath>/S:Envelope/S:Body</sp:XPath>
      </sp:EncryptedElements>
    </wsp:All>
    <wsp:All> <! -- Alternative A2 -->
      <sp:SignedParts>
         <sp:Body />
         <sp:Header
         Namespace="http://www.w3.org/2005/08/addressing" />
      </sp:SignedParts>
      <sp:EncryptedParts>
         <sp:Body />
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

```
<wsp:Policy
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypoliy/200702"
xmlns:wsp="http://www.w3.org/ns/ws-policy" >
<! -- Policy P2 -->
  <wsp:ExactlyOne>
    <wsp:All> <! -- Alternative A3 -->
      <sp:SignedParts />
      <sp:EncryptedParts>
         <sp:Body />
      </sp:EncryptedParts>
    </wsp:All>
    <wsp:All> <! -- Alternative A4 -->
      <sp:SignedElements>
```

```
          <sp:XPath>/S:Envelope/S:Body</sp:XPath>
       </sp:SignedElements>
     </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 26: Beispiel von zwei Policies [14]

Die Alternative A2 in Policy 1 ist kompatibel mit A3 in Policy 2. Das Ergebnis der Intersection der Policies ist die folgende Policy mit einer einzigen Alternative, die alle Assertion in A2 und A3 enthält.

```
<wsp:Policy
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypoliy/200702"
xmlns:wsp="http://www.w3.org/ns/ws-policy" >
<! -- Intersection of P1 and P2 -->
   <wsp:ExactlyOne>
     <wsp:All>
       <sp:SignedParts >
          <sp:Body />
          <sp:Header
          Namespace=http://schemas.xmlsoap.org/ws/2004/08/addressing />
       </sp:SignedParts>
       <sp:EncryptedParts>
          <sp:Body />
       </sp:EncryptedParts>
       <sp:SignedParts />
       <sp:EncryptedParts>
          <sp:Body />
       </sp:EncryptedParts>
     </wsp:All>
  </wsp:ExactlyOne>
```

```
</wsp:Policy>
```

Listing 27: Beispiel vom Intersection von zwei Policies [14]

2.5.7 Policy attachment

WS-Policy Attachment spezifiziert die Policy Assoziierung mit Subjects, es gibt zwei Mechanismen:

- **XML Element Attachment**, Policy Assertions werden definiert oder assoziiert als die Bestandteile der Defintion von einem Subject
- **External Policy Attachment**, Policy Assertions werden unabhängig definiert und durch externes Binding assoziiert.

Merge von Policies

Wenn mehrere Policies mit einem Subject assoziiert werden, wird eine effektive Policy durch Merge Operationen berechnet. Eine Merge Operation von zwei Policies wird wie folgt definiert:

- Die beiden Policies werden in die Normalform umgewandelt.
- Das <wsp: Policy> Element jeder Policy wird durch <wsp: All > Element ersetzt, dann werden beide Kinder eines neuen <wsp: Policy> Elementes.
- Die neue Policy wird in die Normalform umgewandelt.
- Die neue Policy in der Normalform wird als merged Policy für den Subject

Zum Beispiel sind die folgene zwei Policys gegeben:

```
<wsp:Policy> <!–– Policy1 -->
  <wsp:All>
    <wsp:ExactlyOne> <C/> <B/> </wsp:ExactlyOne>
  </wsp:All>
</wsp:Policy>
<wsp:Policy> <!-- Policy2 -->
  <wsp:ExactlyOne> <A/> <C/> </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 28: Beispiel von zwei Policies für Merge

Die beiden Policies werden zuerst in die Normalform umgewandelt:

```
<wsp:Policy> <!—- Policy1 in Normalform -->
  <wsp:ExactlyOne >
    <wsp:All> <B/> </wsp:All>
    <wsp:All> <C/> </wsp:All>
  </wsp:ExactlyOne >
</wsp:Policy>
<wsp:Policy> <!-- Policy2 in Normalform -->
  <wsp:ExactlyOne>
    <wsp:All> <A/> </wsp:All>
    <wsp:All> <C/> </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 29: Beispiel von zwei Policies in der Normalform für Merge

Die beiden <wsp: Policy> Elementen werden jeweils durch das <wsp:All> Element ersetzt und ein neues <wsp:Policy> Element wird als das Wurzelelement hinzugefügt.

```
<wsp:Policy> <!—-neu hinzugefügtes <wsp:Policy> Wurzelelement -->
  <wsp:All> <!—- <wsp:Policy> durch <wsp:All> ersetzt -->
    <wsp:ExactlyOne >
      <wsp:All> <B/> </wsp:All>
      <wsp:All> <C/> </wsp:All>
    </wsp:ExactlyOne >
  </wsp:All>
  <wsp:All> <!-- <wsp:Policy> durch <wsp:All> ersetzt -->
    <wsp:ExactlyOne>
      <wsp:All> <A/> </wsp:All>
      <wsp:All> <C/> </wsp:All>
    </wsp:ExactlyOne>
```

```
  </wsp:All>
</wsp:Policy>
```

Listing 30: Beispiel vom merged Policy

Die merged Policy:

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp: All> <A/> <B/> </wsp: All>
    <wsp: All> <A/> <C/> </wsp: All>
    <wsp: All> <B/> <C/> </wsp: All>
    <wsp: All> <C/> </wsp: All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 31: Beispiel vom merged Policy in der Normalform

Policy Assoziierung mit WDSL Elementen

XML Element Attachment

Die <wsp:PolicyReference> Elemente können als Kindelemente von WSDL-Elementen direkt angehängt werden, um eine Policy mit einem WSDL-Element zu assoziieren. Das folgende Beispiel zeigt an, dass drei Policies „RmPolicy“, „X509EndpointPolicy“ und „SecureMessagePolicy“ in einer WSDL-Dokument definiert sind. Die Policies „RmPolicy“ und „X509EndpointPolicy“ sind mit dem Binding "StockQuoteSoapBinding" assoziiert, die Policy „SecureMessagePolicy“ ist mit den Input- und Output- Messages der Operation „GetLastTradePrice“ assoziiert.

```
<wsdl:definitions name="StockQuote"
      targetNamespace="http://www.example.com/stock/binding"
      xmlns:tns="http://www.example.com/stock/binding"
      xmlns:fab="http://www.example.com/stock"
      xmlns:rmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
      xmlns:sp="http://docs.oasis-open.org/ws-sx/
```

```
    ws-securitypolicy/200702"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-utility-1.0.xsd" >
<wsp:Policy wsu:Id="RmPolicy" >
 <rmp:RMAssertion>
   <wsp:Policy/>
  </rmp:RMAssertion>
</wsp:Policy>
<wsp:Policy wsu:Id="X509EndpointPolicy" >
   <sp:AsymmetricBinding>
     <wsp:Policy>
          <!-- Details omitted for readability -->
       <sp:IncludeTimestamp />
       <sp:OnlySignEntireHeadersAndBody />
     </wsp:Policy>
  </sp:AsymmetricBinding>
 </wsp:Policy>
 <wsp:Policy wsu:Id="SecureMessagePolicy" >
   <sp:SignedParts>
     <sp:Body />
   </sp:SignedParts>
   <sp:EncryptedParts>
     <sp:Body />
   </sp:EncryptedParts>
  </wsp:Policy>
  <wsdl:import namespace="http://www.example.com/stock"
        location="http://www.example.com/stock/stock.wsdl" />
```

```
    <wsdl:binding name="StockQuoteSoapBinding" type="fab:Quote" >
      <wsoap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http" />
      <wsp:PolicyReference URI="#RmPolicy" wsdl:required="true" />
      <wsp:PolicyReference URI="#X509EndpointPolicy"
      wsdl:required="true" />
      <wsdl:operation name="GetLastTradePrice" >
        <soap:operation
           soapAction="http://www.example.com/stock/Quote/
           GetLastTradePriceRequest" />
          <wsdl:input>
            <soap:body use="literal" />
            <wsp:PolicyReference URI="#SecureMessagePolicy"
                                 wsdl:required="true" />
          </wsdl:input>
          <wsdl:output>
            <soap:body use="literal" />
            <wsp:PolicyReference URI="#SecureMessagePolicy"
                               wsdl:required="true" />
          </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

Listing 32: Beispiel vom Policy Assoziierung mit WSDL [15]

External Policy Attachment

Die Policies mit einem Subject können unabhängig vom WSDL-Dokument durch das Element <wsp:PolicyAttachment> definiert werden. Das <wsp:PolicyAttachment> Element wird durch das folgende Pseudo-Schema definiert:

```
<wsp:PolicyAttachment ... >
```

```
  <wsp:AppliesTo>
     <x:DomainExpression/> +
  </wsp:AppliesTo>
  (<wsp:Policy>...</wsp:Policy> |
  <wsp:PolicyReference>...</wsp:PolicyReference> ) +
  <wsse:Security>...</wsse:Security> ?
  ...
</wsp:PolicyAttachment>
```

Listing 33: External Policy Attachment [15]

Im WSDL gibt es vier Policy Subjects:

- **Service Policy Subject**, eine mit einem Service assoziierte Policy, betrifft nur das Element wsdl:service.
- **Endpoint Policy Subject**, eine mit einem Endpoint assoziierte Policy, betrifft die Elemente:

  wsdl:port

  wsdl:binding

  wsdl:portType
- **Operation Policy Subject**, eine mit einer Operation assoziierte Policy,betrifft die Elemente:

  wsdl:portType/operation

  wsdl:binding/operation
- **Message Policy Subject**, eine mit einem Message assoziierte Policy, betrifft die Elemente:

  wsdl:message

  wsdl:binding/opeation/input

  wsdl:binding/opeation/output

  wsdl:binding/opeation/fault

  wsdl:portType/opeation/input

  wsdl:portType/opeation/output

wsdl:portType/opeation/fault

2.6 Web Services Resource Framework

Web Services Resource Framework ist eine Menge von OASIS publizierte Spezifikationen für Web Services [17]. Es spezifiziert einen Mechanismus, um die Beziehung zwischen Web Services und deren Status zu beschreiben. Es besteht aus den Spezifikationen WS-Resource, WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup und WS-BaseFaults. In der Arbeit werden nur die ersten drei Spezifikationen behandelt.

2.6.1 WS-Resource

WS-Resource beschreibt die Beziehung zwischen einem Web Service und einem Resource in Web Service Framework und definiert wie WS-Resources referenziert werden [17].

Definitionen

Ein Resource ist eine logische Entität, die folgende Eigenschaften besitzt:

- Es muss identifizierbar sein.
- Es muss eine Menge von null oder mehrere Eigenschaften haben, die durch XMLInfoset darstellbar sind.
- Es kann einen Lebenszyklus haben.

Ein **WS-Resource** ist die Kompositition von einem Web Service und einem durch das Web Serivce zugreifbarem Resource. Es ist detailiert wie folgt definiert:

- Ein WS-Resource wird durch einen Endpoint Reference referenziert(ERP). Ein ERP kann genau ein WS-Resource referenzieren.
- Die Eigenschaften des WS-Resources müssen mit einem XML Infoset dargestellt werden. Es muss den Zugriff der Eigenschaften durch die Message Exchange unterstützen, die in der WS-ResourceProperties Spezifikation definiert sind.

- Ein WS-Resource kann die in WS-ResourceLifetime Spezifikation definierte Message Exchange unterstützen.

2.6.2 WS-Resource Properties

Ein **Resource Property** ist die Information, die als ein Teil vom Statusmodell von einem WS-Resouce definiert ist. Es kann ein Teil von einer Information über den Status, Meta-Daten, Verwaltbarkeit vom Resource reflektieren.
Ein **Resource Properties Dokument** stellt eine logische Komposition von Resource Property Elementen dar. Es enthält alle Eigenschaften von einem WS-Resource und wird mit einem WSDL1.1 Port Type assoziiert [18]

```
<wsdl:defintions …>
   . . .
   <wsdl:portType . . .
   wsrf-rp:ResourceProperties=”xsd:QName”? . . .>
      . . .
   </wsdl:portType>
   . . .
</wsdl:defintions>
```

Listing 34: Die Assoziierung vom Resource Properties Dokument

Ein **Resource Property Element** ist eine XML Darstellung einer Resource Eigenschaft und erscheint als Kindelement des Wurzelelements eines Resource properties Dokumentes. Es muss eine XML GED(global element defintion) sein und identifiziert durch den Qname.
Ein **Resource Property Value** ist der Wert, der mit einer Eigenschaft eines Resource assoziiert wird.
Im folgenden Beispiel [18] wird ein WSDL Dokument angezeigt, in dem der portType GenericDiskDrive und die mit der PortType assoziierten Resource Properties Dokument definiert werden.

```
<wsdl:definitions ... xmlns:tns="http://example.com/diskDrive" ...>
```

```
…
  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/diskDrive" ... >
    <!-- Resource property element declarations -->
      <xsd:element name="NumberOfBlocks" type="xsd:integer"/>
      <xsd:element name="BlockSize" type="xsd:integer" />
      <xsd:element name="Manufacturer" type="xsd:string" />
      <xsd:element name="StorageCapability" type="xsd:string" />
    <!-- Resource properties document declaration -->
      <xsd:element name="GenericDiskDriveProperties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="tns:NumberOfBlocks"/>
            <xsd:element ref="tns:BlockSize" />
            <xsd:element ref="tns:Manufacturer" />
            <xsd:any minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="tns:StorageCapability"
            minOccurs="0" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
<!-- Association of resource properties document to a portType -->
  <wsdl:portType name="GenericDiskDrive"
  wsrf-rp:ResourceProperties="tns:GenericDiskDriveProperties" >
    <operation name="start" …/>
    <operation name="stop" …/>
  </wsdl:portType>
</wsdl:definitions>
```

Listing 35:Beispiel von der Assoziierung von Resource Properties Dokument [18]

Das folgende Besipiel stellt die Request-Message dar, die verwendet wird, um drei Resource Property Elemente abzuholen.

```
<wsrf-rp:GetMultipleResourceProperties
xmlns:tns="http://example.com/diskdrive" . . .>
   <wsrf-rp:ResourceProperty>
      tns:NumberOfBlocks
   </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty>
      tns:BlockSize
   </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty>
      tns:StorageCapability
   </wsrf-rp:ResourceProperty>
</wsrf-rp:GetMultipleResourceProperties>
```

Listing 36: Beispiel vom GetMultipleResourceProperties Request Message [18]

Das folgende stellt die entsprechende Response-Message dar

```
<wsrf-rp:GetMultipleResourcePropertiesResponse
xmlns:ns1="http://example.com/diskdrive"
xmlns:ns2="http://example.com/capabilities" ...>
   <ns1:NumberOfBlocks>22</ns1:NumberOfBlocks>
   <ns1:BlockSize>1024</ns1:BlockSize>
   <ns1:StorageCapability>
      <ns2:NoSinglePointOfFailure> true </ns2:NoSinglePointOfFailure>
   </ns1:StorageCapability>
   <ns1:StorageCapability>
      <ns2:DataRedundancyMax>42</ns2:DataRedundancyMax>
   </ns1:StorageCapability>
```

```
</wsrf-rp:GetMultipleResourcePropertiesResponse>
```

Listing 37: Beispiel vom GetMultipleResourceProperties Response Message [18]

Operationen

WS-ResourceProperties definiert eine Menge von Operationen (Message Exchanges). Durch die Operationen kann der Status von Resources abgefragt oder geändert werden.

Die Operation **GetResourcePropertyDocument** wird verwendet um die Werte aller Resource Properties abzuholen, die mit dem WS-Resource assoziiert werden. Die Request Message der Operation hat die folgende Form[18]:

```
<wsrf-rp:GetResourcePropertyDocument />
```

Die wsa:Action muss das URI enthalten:

```
http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest
```

Die Response Message der Operation hat die Form:

```
<wsrf-rp:GetResourcePropertyDocumentResponse>
   {any}
</wsrf-rp:GetResourcePropertyDocumentResponse>
```

Die wsa:Action muss das URI enthalten:

```
http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse
```

Im folgenden Beisipiel wird gezeigt wie die Request Message und die Response Message der Operation aussehen.

Angenomen ist das folgende ein Resource Properties Dokument für ein WS-Resource, das durch den PortType GenericDiskDrive definiert wird.

```
<tns:GenericDiskDriveProperties xmlns:tns="http://example.com/diskDrive"
xmlns:cap="http://example.com/capabilities">
   <tns:NumberOfBlocks>22</tns:NumberOfBlocks>
```

```
   <tns:BlockSize>1024</tns:BlockSize>
   <tns:Manufacturer>DrivesRUs</tns:Manufacturer>
   <tns:StorageCapability>
      <cap:NoSinglePointOfFailure>true</cap:NoSinglePointOfFailure>
   </tns:StorageCapability>
   <tns:StorageCapability>
      <cap:DataRedundancyMax>42</cap:DataRedundancyMax>
   </tns:StorageCapability>
</tns:GenericDiskDriveProperties>
```

Listing 38: Beispiel eines Resource Properties Dokumentes [18]

Das folgende ist die Request Message der Operation:

```
<soap:Envelope . . .>
   <soap:Header>
      <wsa:Action>
         http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocumen
         t/GetResourcePropertyDocumentRequest
      </wsa:Action>
. . .
   </soap:Header>
   <soap:Body>
      <wsrf-rp:GetResourcePropertyDocument/>
   </soap:Body>
</soap:Envelope>
```

Listing 39: Beispiel einer getResourcePropertyDokument Request Message [18]

Das folgende Beispiel betrifft die Response Message der Operation:

```
<soap:Envelope …>
   <soap:Header>
      <wsa:Action>
```

```
        http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocumen
        t/GetResourcePropertyDocumentResponse
      </wsa:Action>
. . .
   </soap:Header>
   <soap:Body>
      <wsrf-rp:GetResourcePropertyDocumentResponse
      xmlns:tns="http://example.com/diskDrive"
      xmlns:cap="http://example.com/capabilities">
         <tns:GenericDiskDriveProperties>
            <tns:NumberOfBlocks>22</tns:NumberOfBlocks>
            <tns:BlockSize>1024</tns:BlockSize>
            <tns:Manufacturer>DrivesRUs</tns:Manufacturer>
            <tns:StorageCapability>
               <cap:NoSinglePointOfFailure>true</cap:NoSinglePointOfFailur
               e>
            </tns:StorageCapability>
            <tns:StorageCapability>
               <cap:DataRedundancyMax>42</cap:DataRedundancyMax>
            </tns:StorageCapability>
         </tns:GenericDiskDriveProperties>
      </wsrf-rp:GetResourcePropertyDocumentResponse>
   </soap:Body>
</soap:Envelope>
```

Listing 40: Beispiel einer getResourcePropertyDokument Response Message [18]

In WS-ResourceProperties werden die anderen Operationen auf ähnliche Weise definiert:

- Die Operation **GetResourceProperty** wird verwendet um einen einzelnen Resource Property von einem WS-Resource abzuholen.

- Die Operation **GetMultipleResourceProperties** wird eingesetzt um mehrere Resource Properties von einem WS-Resource abzuholen.
- Die Operation **QueryResourceProperties** ermöglicht das Resource Properties Dokument von einem WS-Resource durch einen Anfrage Ausdruck wie Xpath, abzufragen.
- Die Operation **PutResourcePropertyDocument** wird verwendet, um die Werte der Properties eines WS-Resources völlstandig durch ein total neues Resource Property Dokument zu ersetzen.
- Die Operation **SetResourceProperties** ermöglicht drei verschiedene Typen von Änderung des Resource Properties Dokumentes:

  Insert: Ein neues Resource Property Element kann im Resource Properties Dokument eingefügt werden.

  Update: Ein oder mehrere vorkommende Resource Property Elemente können geändert werden.

  Delete: Ein oder mehrere vorkommende Resource Property Elemente können gelöscht werden.
- Die Operation **InsertResourceProperties** ermöglicht einen oder mehrere Element Wert(e) einer Resource Property im Resource Properties Dokument von einem WS-Resource einzufügen.
- Die Operation **DeleteResourceProperties** wird eingesetzt, um alle Werte einer Resource Property zu löschen.
- Die Operation **UpdateResourceProperties** ist zuständig für die Änderung von einem oder mehrere Werte einer Property.

2.6.3 WS-Resource Lifetime

In der WS-ResourceLifetime Spezifikation wird beschrieben wie ein Resource zerstört und die Lebensdauer eines WS-Resource überwacht werden kann. Um ein Resource zu zerstören, werden zwei Methoden definiert: **Immediate Destruction** und **Scheduled Destruction**.

Immediate Destruction

Ein Resource wird unmittelbar zerstört, es wird durch die Nutzung der Message Exchange realisiert, die DestroyRequest Message hat das folgende Format [19]:

```
<wsrf-r1:Destroy />
```

Die wsa: Action muss das URI enthalten:

```
http://docs.oasis-open.org/wsrf/rlw-2/ImmediateResourceTermination/Dest
royRequest
```

Falls das WS-Resource die DestroyRequest Message erhält, schickt es entweder eine DestroyResponse Message wenn das Resource erfolgreich zerstört wird oder es schickt eine Fault Message zurück. Die DestroyResponse Message hat das folgende Format [19]:

```
<wsrf-r1:DestroyResponse />
```

Das folgende Beispiel beschreibt eine DestroyRequest Message:

```
<soap:Envelope . . .>
  <soap:Header>
     . . .
     <wsa:Action>
        http://docs.oasis-open.org/wsrf/rlw-2/ImmediateResourceTerminati
        on/DestroyRequest
     </wsa:Action>
     . . .
  </soap:Header>
  <soap:Body>
     <wsrf-rl:Destroy/>
  </soap:Body>
</soap:Envelope>
```

Listing 41: Beispiel einer DestroyRequest Message [19]

Das folgende Beispiel beschreibt eine DestroyResponse Message:

```
<soap:Envelope . . .>
  <soap:Header>
    . . .
    <wsa:Action>
      http://docs.oasis-open.org/wsrf/rlw-2/ImmediateResourceTerminati
      on/DestroyResponse
    </wsa:Action>
    . . .
  </soap:Header>
  <soap:Body>
    <wsrf-rl:DestroyResponse />
  </soap:Body>
</soap:Envelope>
```

Listing 42: Beispiel einer DestroyReponse Message [19]

Scheduled Destruction

Die Störung des Resource ist zeitbasiert und kann von einem Client gesteuert werden, indem das Client einen Zeitpunkt für die Terminierung des Resource anlegt. Ist die Zeit abgelaufen, wird das Resource automatisch zerstört. Um Scheduled Desstruction zu unterstützen, muss das Resource Properties Dokument die Resource Property Elemente **CurrentTime** und **TermitationTime** enthalten und die Operation **SetTerminationTime** unterstützen.
Das Element **CurrentTime** bietet die aktuelle Zeit an, die dem WS-Resource bekannt ist und hat das folgende Format [19]:

```
<wsrf-rl:CurrentTime>xsd:dateTime</wsrf-rl:CurrentTime>
```

Das Resource Property Element wsrf-r1:CurrentTime muss genau ein Mal definiert werden und darf durch die Operation **SetResourceProperties** nicht geändert werden.
Das Element **TerminationTime** zeigt die aktuelle Terminierungszeit vom WS-Resource an. Es hat das folgende Format [19]:

```
<wsrf-rl:TerminationTime xsi:nil=”xsd:boolean”?>
  xsd:dateTime
</wsrf- rl:TerminationTime>
```

Das Resource Property Element wsrf-r1:TerminationTime muss genau ein Mal definiert werden und darf durch die Operation **SetResourceProperties** nicht geändert werden. Wenn das Resource Property Element das Attribut xsi:nil mit den Wert „True“ enthält, dann gibt es keine Scheduled Destruction Zeit.

Die Operation **SetTerminationTime** ermöglicht die Änderung der geplanten Terminierungszeit. Das folgende ist ein Pseudo-Schema von der Request Message der Operation:

```
<wsrf-rl:SetTerminationTime>
  (<wsrf-rl:RequestedTerminationTime xsi:nil=”xsd:boolean”?>
    xsd:dateTime
  </wsrf-rl:RequestedTerminationTime>)
| (<wsrf-rl:RequestedLifetimeDuration>
     xsd:duration
  </wsrf-rl:RequestedLifetimeDuration>)
</wsrf-rl:SetTerminationTime>
```

Listing 43: Beispiel einer SetTerminationTime Request Message [19]

Die Operation bietet zwei Möglichkeiten an, um den Zerstörungszeitpunkt zu bestimmen:
Der Zerstörungszeitpunkt wird durch das Element wsrf-rl: RequestedTerminationTime direkt angegeben.
Die verbleibende Lebensdauer wird durch das Element wsrf-rl: RequestedLifetimeDuration angegeben.
Wenn das WS-Resource die SetTerminationTime Request annimmt, muss die Terminierungszeit aktualisiert werden und eine Response Message wird zurückgeschickt.
Das folgende zeigt das Format der Response Message:

```
<wsrf-rl:SetTerminationTimeResponse>
  <wsrf-rl:NewTerminationTime xsi:nil=”xsd:boolean”?>
```

```
      xsd:dateTime
    </wsrf-rl:NewTerminationTime>
    <wsrf-rl:CurrentTime>
      xsd:dateTime
    </wsrf-rl:CurrentTime>
<wsrf-rl:SetTerminationTimeResponse>
```

Listing 44: Beispiel einer SetTerminationTime Response Message [19]

2.7 XMLO_Fragmento

Das Fragmento bedeutet ein fragmentorientierten Prozess Artefakt-Repository und verwaltet Prozesse und Prozess-Fragmente für den Einsatz im Bereich vom Compliance. Das Fragmento ist zuständig für das Speichern, das Zugreifen und die Versionsverwaltung aller Artefakte, die mit einem Prozess relevant sind. Das Fragmento ist auf einem Repository aufgebaut, das durch das Projekt Master entwickelt wurde [11].
Das Master-Repository unterstützt das Modell für die Versionierung der Artefakte. Wenn ein neues Artefakt erstellt wird, erzeugt das Repository ein neues Versioned Objekt(gezeigt in Abbildung 7). Dieses Objekt ist der Container für die verschiedenen Versionen des Artefakts (gezeigt in Abbildung 8). Das Versioned Objekt ist auch ein Versionhistory Objekt, das den Zugriff auf die Root-Version (die erste Version) und die Basis Version (die neueste Version) ermöglicht. Eine Version eines Artefakts ist durch einen "Version Descriptor" Objekt dargestellt. Es speichert das Datum der Erstellung, Metadaten und einem Verweis auf das XML-Dokument des Artefakts.

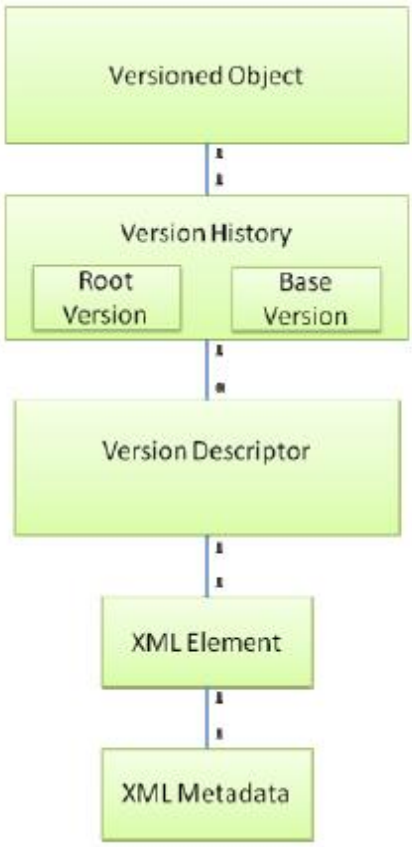

Abbildung 7: Modell der Versionsverwaltung [11]

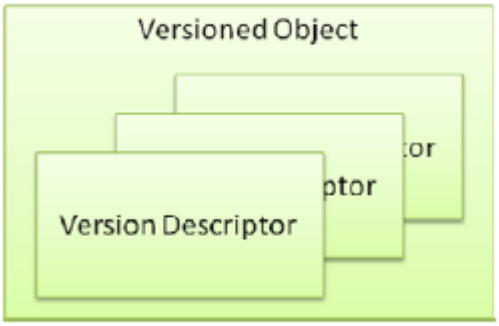

Abbildung 8: Versionen eines Artefakts [11]

Das Master-Repository kann ebenfalls neue Relationen zwischen Version Deskriptoren erstellen (In Abbildung 9). Diese Funktion kann zum Erstellen eines Bündels von Artefakten verwendet werden. Es unterstützt auch die Erstellung einer Annotation von einem Fragment und einer textuellen Annotation. Es ist wichtig zu beachten, dass die Relationen zwischen den Deskriptoren Version erstellt werden und nicht zwischen Versioned Objekten.

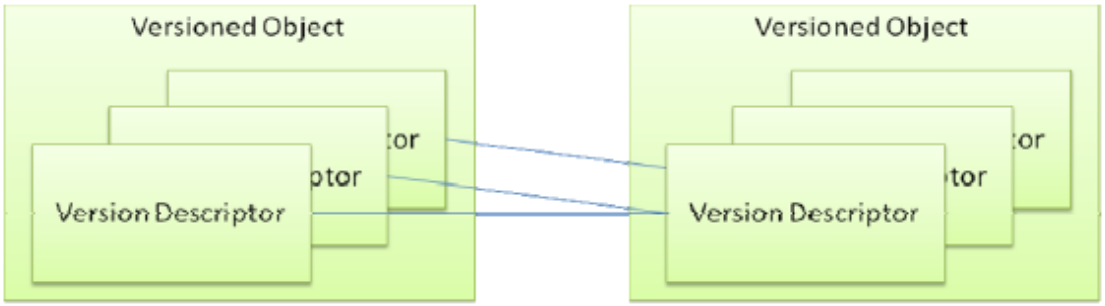

Abbildung 9: Relationen zwischen Artefakten [11]

Das Fragmento unterteilt die XML basierten Artefakte auf sechs verschieden Typen: Prozess oder Prozess Fragment Model, WSDL-Artefakt, Deployment Deskriptor, Modeller Information, Transformationsregel und Annotation.

Wenn ein Artefakt angelegt wird, ordnet das Repository dem Artefakt eine eindeutige ID zu. Die Relationen zwischen Artefakten können durch IDs erstellt werden.

Die Verwaltung von Relationen gehört auch zu der Aufgabe des Repositorys. Die Abbildung 10 zeigt das konzeptionelle Modell der verschiedenen Arten von Artefakten und ihrer Relation.

Im Fragmento Repository besteht ein Artefakt aus den folgenden Teilen:

- Einer Id, der eindeutig das Artefakt identifiziert.
- Metadaten, die zur Beschreibung des Artefakts dienen.
- Einer XML Dokument, die das Inhalt von Artefakt ist.
- Einem Typ, der den Typ vom Artefakt wie „WSDL“, „Process“, usw. beschreibt.
- Relation(en) zu anderen Artefakten.

Im Fragmento Repository besteht eine Relation aus den folgenden Teilen:

- Einem Quell, das ein Artefakt ist.
- Einem Ziel, das ein anderes Artefakt ist.
- Einem Typ, der den Typ von der Relation beschreibt.
- Einer weiteren Beschreibung

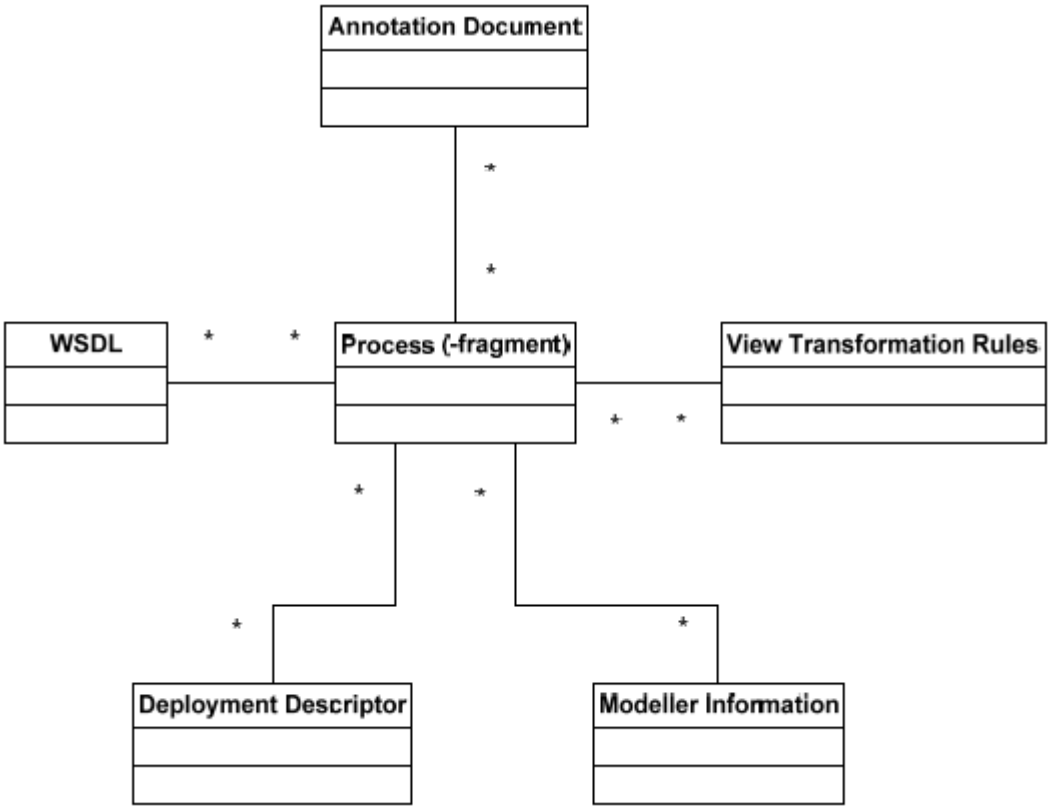

Abbildung 10: Konzeptionelles Model für Fragmento [11]

Das Fragmento Service bietet die folgenden Funktionalitäten an [11]:

1. **createArtefact**: Die Operation wird verwendet um ein neues Artefakt zu erstellen.

2. **retrieveArtefact**: Die Operation wird verwendet um eine bestimmte Version eines Artefakts abzurufen, ohne Durchführung eines Check Outs.

3. **retrieveArtefactBundle**: Die Operation liefert ein Artefakt und alle Artefakte, die mit dem Artefakt in Relation stehen.

4. **retrieveArtefactHistory**: Die Operation liefert eine Liste von Ids von Version Deskriptor, die die Versionshistorie eines Artefakts repräsentieren.

5. **checkOutArtefact**: Die Operation setzt eine Sperre für das angeforderte Artefakt und gibt das Artefakt und einen für Check In angeforderte Sperren-Id zurück.

6. **checkInArtefact**: Die Operation erstellt eine neue Version eines Artefakts. Für die Zulassung muss auch die entsprechende Sperren-Id gegeben werden. Basierend auf den Parameter keepRelations. Die Relationen des Artefakts gelten auch für die neue Version. Die Operation gibt die Id der neuen Version zurück.

7. **browseArtefacts**: Die Operation implementiert die Suchfunktion. Basierend auf den Eingabeparameter gibt diese Operation eine Liste der Version Deskriptoren aus, die der Abfrage entsprechen. Als Eingabeparameter werden akzeptiert: Artefakt-Typ, Zeitintervall des Erstellens, Suchbegriff für die Beschreibung oder das Inhalt des Dokumentes
8. **retrieveArtefactLatestVersion**: Die Operation liefert die neueste Version eines Artefakts. Um alte Revisionen abzurufen, wird die Operation retrieveArtefact eingesetzt.
9. **browseLocks**: Die Operation liefert eine Liste aller gesperrten Artefakte.
10. **releaseLocks**: Die Operation kann eine Sperre wieder freigeben.
11. **createRelation**: Die Operation ermöglicht die Erstellung einer Relation von einem Artefakte zu einem anderen.
12. **retrieveRelation**: Die Operation liefert die Details einer Relation.
13. **browseRelations**: Die Operation bietet Suchfunktionen für die Relation an. Die gültige Eingabeparameter sind eine Quelle (ID eines Version Deskriptors), ein Ziel, ein Typ oder ein Zeitintervall für das Erstellen.
14. **updateRelation**: Die Operation ermöglicht es eine vorhandene Relation zu ändern.
15. **deleteRelation**: Die Operation löscht vollständig eine Relation.

3 Konzepte

In der Arbeit wird ein Web Service, das Registerservice entwickelt. Das Registerservice bietet die Funktionalitäten an, um Web Services, Datenbanken und Anbieter zu registrieren, zu suchen und zu verwalten. Bevor die Funktionalitäten vom Registerservice vorgestellt werden, wird zuerst die Relation zwischen Web Services, Datenbanken und Anbietern kurz erklärt. Die Relation wird in Abbildung 11 gezeigt:

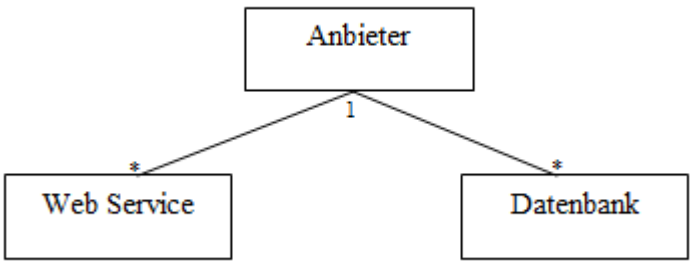

Abbildung 11: Relationen zwischen Web Services, Datenbanken und Anbietern

- Ein Anbieter kann beliebig viele Web Services bzw. Datenbanken anbieten
- Ein Web Service bzw. eine Datenbank gehört nur zu einem Anbieter.

3.1 Anbieter

Bevor ein Web Service bzw. eine Datenbank registriert wird, muss zuerst der Anbieter des Services bzw. der Datenbank registriert werden. Die Gründe dafür sind:

- Die Informationen über die Anbieter sind erwünscht von der Nutzerseite.
- Es ist von der Anbieterseite auch erwünscht, die schon registrierten Web Services und Datenbanken zu verwalten. Es ist klar, dass Operationen wie Ändern, Löschen nur von dem entsprechenden Anbietern ausgeführt werden dürfen. Es ist deswegen notwendig, einen Anbieter zu authentifizieren, ob es der richtige Anbieter des Services bzw. der Datenbank ist, bevor er ein Service bzw. eine Datenbank ändert bzw. löscht.

Die Informationen eines Anbieters müssen enthalten:

- Eine Id, die einen Anbieter eindeutig identifizieren kann, wird bei der Registrierung eines Anbieters automatisch generiert und darf nicht geändert werden.
- Einen Namen des Anbieters
- Eine Emailadresse, die einen Anbieter eindeutig identifizieren und zusammen mit dem Passwort zur Authentifizierung des Anbieters dienen kann.
- Ein Passwort, das zusammen mit der Emailadresse zur Authentifizierung dienen kann.
- Ein Registrierungsdatum, die bei der Registrierung automatisch erstellt wird und nicht geändert werden darf.

Außerdem können die Informationen eines Anbieters optional enthalten:

- Eine Adresse: Strasse, Stadt, Postleitzahl, Land
- Eine Web Site
- Eine Telefonnummer

3.2 Web Service

3.2.1 WS-Bündel

Die Voraussetzung für die Registrierung von Web Services ist die Registrierung des Anbieters für den Service.
Bei der Registrierung von Web Services muss der Anbieter ein WS- Bündel von XML Dokumenten liefern, die mit den Web Services relevant sind. Nach der Registrierung wird das WS-Bündel als ein Artefakt-Bündel gespeichert. Im Artefakte-Bündel gibt es für jedes Dokument vom WS-Bündel ein entsprechendes Artefakt. Wie ein WS-Bündel aussieht, wird in Abbildung 12 gezeigt:

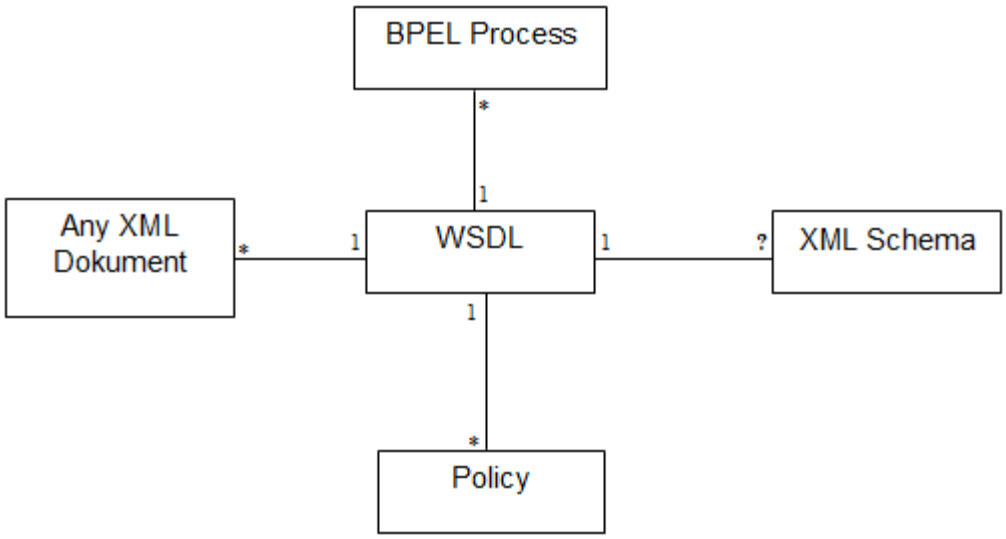

Abbildung 12: WS-Bündel

Wie im Kapitel 2 spezifiziert, beschreibt WSDL die funktionalen Eigenschaften von Web Service. Das WSDL Dokument spielt eine zentrale Rolle bei der Registrierung von Web Services. EinWS-Bündel besteht aus:

- Ein WSDL-Dokument, das die funktionalen Eigenschaften von Services spezifiziert.
- Beliebig viele BPEL-Process Dokumente, die zum Aufruf von Services dienen.
- Beliebig viele Policy-Dokumente, die die nicht funktionalen Eigenschaften von Services beschreiben.
- Ein optionales XML-Schema Dokument für das Resource Properties Dokument.
- Beliebig viele andere XML Dokumente, die relevant von Web Services sind.

Ein Artefakt besteht aus:

- Eine Id, die ein Artefakt eindeutig identifizieren kann.
- Ein Typ von drei Typen: "WSDL", "Process" und "Anntotation".
- Eine Beschreibung. Die Artefakte werden in drei Typen unterteilt, außer WSDL-Artefakte und BPEL-Process Artefakte haben alle anderen Artefakte den Typ "Annotation". Die Beschreibung unterteilt den Typ "Annotation" in mehrere Kind-Typen.
- Ein Inhalt

- Relationen. Eine Relation stammt entweder von einem WSDL-Artefakt oder von einer Datenbank zu einem Non-WSDL-Artefakt.

Wenn ein Artefakt-Bündel für ein WS-Bündel angelegt wird, sind der Typ, die Beschreibung und die Relationen des Artefakts schon festgelegt. Die drei Eigenschaften eines Artefakts dürfen nicht geändert werden solange das Artefakt noch in Artefakt-Bündel steht. Die Drei Eigenschaften für ein einzelnes Artefakt werden im Folgenden spezifiziert:

- Ein WSDL-Artefakt hat den Typ "WSDL", die Beschreibung "WSDL" und die Relationen, die jeweils vom WSDL-Artefakt zum jedem anderen Artefakt im Artefakt-Bündel sind.
- Ein XML-Schema-Artefakt hat den Typ "Annotation", die Beschreibung "Schema" und eine Relation mit WSDL-Artefakt.
- Ein BPEL-Process-Artefakt hat den Typ "Process", die Beschreibung "Process" und eine Relation mit WSDL-Artefakt.
- Ein Policy-Artefakt hat den Typ "Annotation", die Beschreibung "Policy" und eine Relation mit WSDL-Artefakt.
- Ein anderes Artefakt hat den Typ "Annotation", die Beschreibung "Other" und eine Relation mit WSDL Artefakt.

3.2.2 Service Metadaten

Für jeden Service, das durch ein WSDL-Artefakt beschrieben wird, gibt es Service Metadaten. Durch die Service Metadaten wird der Zusammenhang (gezeigt in Abbildung 13) zwischen Services, WSDL-Artefakten und Anbietern angelegt. Für einen Service kann es dadurch einfach festgestellt werden, welches WSDL-Artefakt das Service beschreibt und welchem Anbieter der Service und das WSDL-Artefakt gehören.

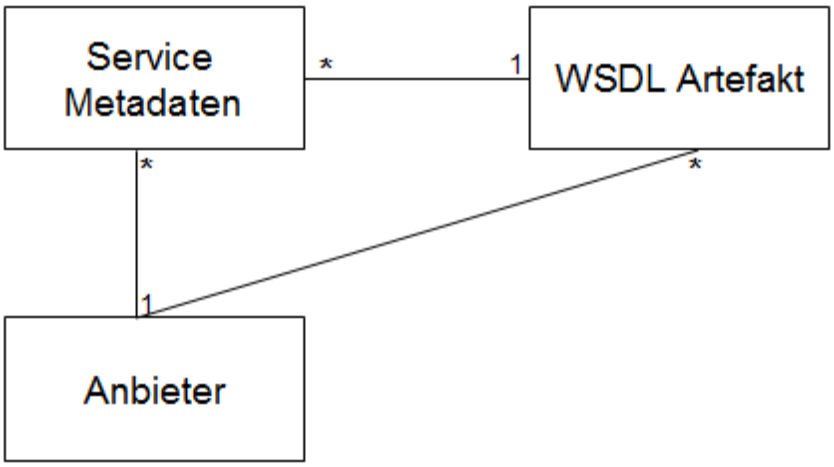

Abbildung 13: Der Zusammenhang zwischen Service, WSDL-Artefakt und Anbieter

Service Metadaten bestehen aus:

- Eine Id, die einen Service eindeutig identifiziert
- Ein Service Name, der mit dem entsprechenden Service Namen im WSDL-Artefakt identisch ist, darf nicht geändert werden, solange der entsprechende Service Name im WSDL-Artefakt ungeändert bleibt.
- Eine Service Beschreibung. Bei der Registrierung eines Services wird ein Service Name gesetzt, kann aber später vom Anbieter geändert werden.
- Eine Service Adresse, die mit der entsprechenden Service Adresse im WSDL-Artefakt identisch ist, darf nicht geändert werden, solange sie im WSDL-Artefakt ungeändert bleibt.
- Ein Service Targetnamespace, das mit dem Targetnamespace im WSDL-Artefakt identisch ist, darf nicht geändert werden, solange es im WSDL-Artefakt ungeändert bleibt.
- Ein Registrierungsdatum, der bei der Registrierung von einem Service automatisch generiert wird, darf nicht geändert werden.
- Eine WSDL-Artefakt Id spezifiziert welches WSDL-Artefakt das Service beschreibt.
- Eine WSDL Dokument Adresse, die spezifiziert, wo das WSDL Dokument von einem Anbieter bereitzustellen ist.
- Eine Anbieter Id spezifiziert von wem der Service angeboten wird.

- Eine Bewertungsanzahl. Ein Nutzer kann einen Service bewerten. Jedes mal wenn ein Service bewertet wird, erhöht sich die Anzahl um 1. Der Anfangswert ist 0.
- Eine Service Note, die eine durchschnittliche Note aller von Nutzern gegebenen Bewertungsnoten wiedergibt. Die Note hat den Wertbereich zwischen 0 und 10 und den Anfangswert 0.

3.3 Datenbanken

Eine Datenbank ist ein Resource, mit der beliebig viele Policies assoziiert werden können und kann beliebig viele Datenbanknutzer haben (wie in Abbildung 14 gezeigt). Die Policies spezifizieren die Anforderungen des Anbieters für die Nutzung der Datenbank

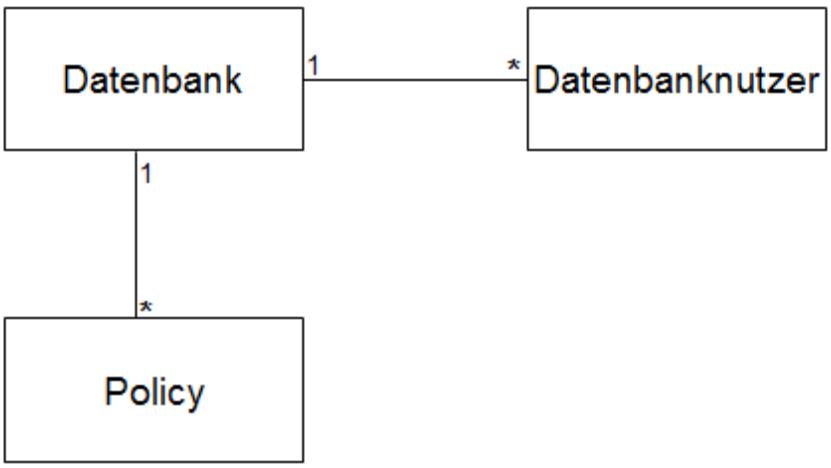

Abbildung 14: Der Zusammenhang zwischen Policy, Datenbank und Datenbanknutzer

Die Informationen einer Datenbank müssen enthalten:

- Eine ID, die eine Datenbank eindeutig identifizieren kann
- Ein Datenbank Name, der vom Anbieter geliefert wird.
- Eine ID des Anbieters, die den Anbieter der Datenbank identifizieren kann.
- Einen Datenbank-Treiber
- Eine Datenbank-Adresse, die spezifiziert, wo die Datenbank gefunden werden kann.
- Eine Beschreibung der Datenbank.

- Ein Registrierungsdatum, das bei der Registrierung der Datenbank automatisch generiert wird und nicht geändert werden darf.
- Eine Bewertungsanzahl. Jeder Nutzer kann eine Datenbank bewerten. Jedes mal wenn eine Datenbank bewertet wird, erhöht sich die Anzahl um 1. Der Anfangswert ist 0.
- Eine Datenbank Note, die eine durchschnittliche Note aller von Nutzern gegebenen Bewertungsnoten wiedergibt. Die Note hat den Wertbereich zwischen 0 und 10 und den Anfangswert 0.

Datenbanknutzer

Die Informationen eines Datenbanknutzers müssen enthalten:

- Eine ID, die einen Datenbanknutzer eindeutig identifizieren kann.
- Ein Name
- Ein Passwort
- Ein Erstellungsdatum, das bei der Registrierung automatisch generiert wird.

Außerdem können die Informationen eines Anbieters optional enthalten:

- Eine Emailadresse
- Eine Adresse: Strasse, Stadt, Postleitzahl, Land
- Eine Web Site
- Eine Telefonnummer

Bevor eine Datenbank registriert wird, muss der Anbieter der Datenbank vorher registriert werden. Die Policies, die mit einer Datenbank assoziiert sind, spezifizieren die Anforderungen des Anbieters für die Nutzung der Datenbank. Bei der Registrierung einer Datenbank mit Policies werden die Informationen von der Datenbank gespeichert und die Policy-Artefakte für gegebene Policy Dokumente angelegt, wichtig ist, für jedes Policy-Artefakt jeweils eine Relation von der Datenbank zum Policy-Artefakt zu erstellen. Durch die Relationen kann man einfach feststellen, welche Policies zu welchen Datenbanken gehören. Ein neues Policy Dokument kann ebenfalls nach der Registrierung einer Datenbank hinzugefügt werden. Die mit einer Datenbank in Relationen stehenden Policy-Artefakte

können ebenfalls geändert bzw. entfernt werden. Die Datenbanknutzer einer Datenbank können nach der Registrierung der Datenbank separat angelegt werden.

3.4 Die Operationen

Für die Registrierung, das Suchen und die Verwaltung von Anbietern, Services und Datenbanken werden die folgenden vier Typen von Operationen vom Registerservice angeboten:

Operationen für Anbieter

- **registerProvider**, die Operation ist für die Registrierung von einem Anbieter zuständig
- **updateProvider**, die Operation ermöglicht die Änderung von Informationen eines registrierten Anbieters
- **retrieveProvider**, die Operation ermöglicht es Informationen eines Anbieters zu liefern.
- **getPassword**, die Operation wird eingesetzt um ein Passwort zurückzubekommen, wenn ein Anbieter sein Passwort vergessen hat.
- **deleteProvider**, die Operation ist für die Entfernung eines Anbieter zuständig.

Operationen für Artefakte

- **addNewArtefact**, die Operation ermöglicht ein neues Non-WSDL-Artefakt zu einem WSDL-Artefakt bzw. zu einer Datenbank hinzuzufügen.
- **updateArtefact**, die Operation wird verwendet, um ein Non-WSDL-Artefakt zu ändern.
- **browseArtfacts**, die Operation bietet verschiedene Möglichkeiten an Artefakte zu suchen
- **deleteArtefact**, die Operation wird eingesetzt, um ein Non-WSDL-Artefakt zu löschen.
- **RetrieveArtefact**, die Operation nimmt eine Artefakt ID ein und liefert den Typ, die Beschreibung und den Inhalt vom Artefakt.

- **RetrieveArtefactBundle**, die Operation nimmt eine ID eines WSDL-Artefakts bzw. eine ID einer Datenbank ein und gibt den Typ, die Beschreibung und den Inhalt von allen Artefakten im Artefakt-Bündel bzw. von allen Policy-Artefakten aus, die mit der Datenbank in Relation stehen.

Operationen für Datenbanken

- **registerDatabase**, die Operation ist für Registrierung einer Datenbank zuständig.
- **updateDatabase**, die Operation ermöglicht die Änderung von Informationen einer Datenbank
- **browseDatabase**, die Operation bietet verschiedene Möglichkeiten an, um Datenbanken zu suchen.
- **valuateDatabase**, die Operation wird verwendet, um eine Datenbank zu bewerten.
- **deleteDatabase**, die Operation ermöglicht es eine Datenbank zu löschen.
- **addNewDatabaseUser**, die Operation ist für die Anlegung eines Datenbanknutzers zuständig.
- **updateDatabaseUser**, die Operation ermöglicht die Informationen eines Datenbanknutzers zu ändern
- **browseDatabaseUser**, die Operation bietet dem Datenbank Anbieter verschiedene Möglichkeiten an, um Datenbanknutzer zu suchen
- **deleteDatabaseUser**, die Operation wird verwendet, um einen Datenbanknutzer zu löschen.

Operationen für Web Services

- **registerWebServices**, die Operation ist für die Registrierung von Web Services zuständig.
- **updateWebServices**, die Operation ermöglicht es Informationen von Web Services zu ändern.
- **BrowseWebServices**, die Operation bietet verschiedene Möglichkeiten an, um Services zu suchen.
- **RetrieveWebService**, die Operation liefert alle Informationen eines Services zurück

- **valuateWebServcie**, die Operation ermöglicht es, ein Service von Nutzern bewerten zu lassen.
- **deleteWebServcies**, die Operation wird verwendet, um Services zu löschen.

Wie die einzelnen Operationen funktionieren, wird in folgenden detailliert beschrieben.

3.4.1 Operationen für Anbieter

registerProvider

Eingabe:

Mandatory: providername, email1, email2, passoword1, password2

Optional: street, city, zipcode, country, telephone, website

Ausgabe: providerId

Beschreibung:

Bei der Registrierung eines Anbieters muss der Anbieter die Informationen über den providername, email1, email2, password1 und password2 angeben. Diese fünf Felder sind für die Registrierung von Anbietern notwendig. Der Wert von email1 wird mit dem Wert von email2 verglichen. Sind die beiden Werte identisch, wird dann der Wert von password1 mit dem Wert von password2 verglichen. Sind die beiden Werte identisch, wird kontrolliert ob die Werte der optionalen Felder zusätzlich noch vorhanden sind. Die Emailadresse und das Passwort eines Anbieters dienen zusammen der Authentifizierung vom Anbieter. Wenn die Registrierung fehlerfrei ausgeführt wird, werden alle Informationen auf der Datenbank gespeichert. Eine ID wird für den Anbieter automatisch generiert und zurückgegeben. Ein Email wird ebenfalls an der gegebenen Emailadresse zugeschickt, um die Registrierung zu bestätigen. Die Email enthält die ID Nummer vom Anbieter, die eindeutig den Anbieter identifiziert.

retrieveProvider

Eingabe:

Mandatory: providerId

Optional:

Ausgabe: Informationen vom Provider

Beschreibung:

Es ist möglich, dass ein Nutzer die allgemeinen Informationen von einem anderen Anbieter anfordert. Für diese Operation soll der Nutzer ein providerId liefern. Durch die gegebene ProviderId, wird zuerst überprüft, ob der Provider existiert. Ist dies der Fall, werden alle Informationen vom Provider außer dem Passwort zurückgegeben.

getPassword

Eingabe:

Mandatory: email

Optional:

Ausgabe: providerId

Beschreibung:

Es kommt sehr häufig vor, dass ein Anbieter das Passwort vergisst, das bei der Registrierung in der Datenbank gespeichert wurde. Das Web Service bietet auch die Funktionalität an um das vergessene Passwort zurück zu bekommen. Beim Abfragen des Passwortes braucht der Anbieter nur die Emailadresse, die bei der Registrierung angegeben wurde anzugeben. Es wird kontrolliert, ob die Emailadresse vorhanden ist. Ist dies der Fall, wird ein Anbieter mit der Emailadresse gefunden. Eine Email wird mit dem Passwort vom Anbieter an die Emailadresse zugeschickt und die ID des Anbieters wird zurückgegeben.

authenticateProvider

Eingabe:

Mandatory: email, password

Optional:

Ausgabe: providerId

Beschreibung:

Der Anbieter muss sich selber authentifizieren um eigene Services und Datenbanken zu verwalten. Bei der Authentifizierung muss der Anbieter eine Emailadresse und ein Passwort eingeben. Es wird ein Anbieter durch die gegebene Emailadresse gesucht. Wird ein Provider

gefunden, wird das gegebene Passwort mit dem Passwort des gefundenen Providers verglichen. Falls die beiden Werte identisch sind, dann ist der Anbieter authentifiziert. Die ID Nummer vom Anbieter wird zurückgegeben.

updateProvider

Eingabe:

Mandatory: email, password

Optional: name, street, city, zipcode, country, telephone, website, email1, emal2, password1, password2

Ausgabe: providerId

Beschreibung:

Vor der Änderung muss der Anbieter eine Emailadresse und ein Passwort angeben, um sich zu authentifizieren. Falls der Anbieter authentifiziert ist, wird kontrolliert, welche optionale Parameter vom Anbieter gegeben sind. Die Werte der vorkommenden optionalen Parameter werden jeweils mit dem entsprechenden im Datenbank gespeicherten Wert verglichen, wenn die beiden Werte nicht identisch sind, wird der in der Datenbank gespeicherte Wert durch den gegebenen Wert ersetzt. Die Änderung der Emailadresse bzw. dem Passwort wird besonders behandelt, der Anbieter muss für die Änderung der Emailadresse bzw. des Passwortes zwei mal den gleichen Wert eingeben um Tippfehler zu vermeiden. Falls die Informationen geändert werden, wird die ID vom Anbieter zurückgegeben.

deleteProvider

Eingabe:

Mandatory: email, password

Optional:

Ausgabe: providerId

Beschreibung:

Vor dem Löschen des Anbieters muss der Anbieter die Emailadresse und das Passwort angeben um sich zu authentifizieren. Ist der Anbieter authentifiziert, wird kontrolliert ob es noch web Services bzw. Datenbanken vom Anbieter vorhanden sind. Ist dies der Fall, darf der

Anbieter nicht gelöscht werden. Sonst wird der Anbieter gelöscht und die ID vom Anbieter zurückgegeben.

3.4.2 Operationen für Artefakte

addNewArtefact

Eingabe:

Mandatory: email, password, wsdlId | databaseId, description, type, Inhalt des Artefakts

Optional:

Ausgabe: artefaktId, relationId

Beschreibung:

Nach der Registrierung von einem Web Service bzw. einer Datenbank kann der Anbieter ein neues XML Dokument im WS-Bündel bzw. ein neues Policy Dokument in der Datenbank hinzufügen. Der Typ des hinzufügenden Artefakts darf nicht „WSDL“ sein, weil in einem Bündel nur genau ein WSDL-Artefakt vorhanden sein darf.

Beim Hinzufügen eines Artefakts zum WSDL-Artefakt muss der Anbieter eine Emailadresse, ein Passwort, die ID des WSDL-Artefakts, die Beschreibung, den Typ und den Inhalt des neuen Artefakts eingeben. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Die ID des WSDL-Artefakts dient der Auswahl des WSDL-Artefakts. Ist der Anbieter authentifiziert, wird das WSDL-Artefakt durch die gegebene ID gesucht. Fall das WSDL-Artefakt existiert, wird irgendein Web Service durch die ID des WSDL-Artefakts gesucht. Die Anbieter ID eines gefundenen Web Services wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das WSDL-Artefakt zum Anbieter. Dann wird das neue Artefakt angelegt. Eine Relation wird vom WSDL-Artefakt zum neuen Artefakt erstellt. Falls das neue Artefakt erfolgreich hinzugefügt wird, werden die ID des Artefakts dem Anbieter zurückgegeben.

Beim Hinzufügen eines Artfakts zu einer Datenbank ist der Vorgang ähnlich wie beim Hinzufügen eines Artefakts zum WSDL-Artefakt.

deleteArtefact

Eingabe:

Mandatory: email, password, artefaktId

Optional:

Ausgabe: artefaktId

Beschreibung:

Der Typ des zu löschenden Artefakts darf nicht „WSDL“ sein, denn das Entfernen eines WSDL-Artefakts bedeutet das Entfernen des ganzen WS-Bündels. Für das Entfernen eines WSDL-Artefakts steht die Operation **deleteWebServices** zur Verfügung. Beim Löschen eines Artfakts muss der Anbieter Emailadresse, ein Passwort und die ID eines Artefakts eingeben. Die Emailadresse und das Passwort dienen zusammen der Authentifizierung des Anbieters. Die ID des Artefakts dient zum Auswahl des Artefakts. Ist der Anbieter authentifiziert wird das Artefakt durch die gegebene ID gesucht. Falls das Artefakt existiert und kein WSDL-Artefakt ist, wird zuerst durch die Artefakt ID das bezogene WSDL-Artefakt gesucht.
Falls gefunden:
dann wird irgendein Web Service durch die ID des gefundenen WSDL-Artefakts gesucht. Die Anbieter ID eines gefundenen Web Services wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das Artefakt zum Anbieter. Dann wird die Relation vom WSDL-Artefakt zum Artefakt gesucht und gelöscht. Anschließend kann das Artefakt gelöscht werden und die ID des Artefakts wird dem Anbieter zurückgegeben.
Falls nicht gefunden:
wird durch die Artefakt ID die bezogene Datenbank gesucht. Die Anbieter ID der gefundenen Datenbank wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das Artefakt zum Anbieter. Dann wird die Relation aus der Datenbank zum Artefakt gesucht und gelöscht. Anschließend kann das Artefakt gelöscht werden und die ID vom Artefakt wird dem Anbieter zurückgegeben.

updateArtefact

Eingabe:

Mandatory: email, password, artefaktId, desription, Inhalt des Artefakts

Optional:

Ausgabe: artefaktId

Beschreibung:

Der Type des zu ändernden Artefakts darf nicht „WSDL“ sein, denn die Änderung eines WSDL-Artefakts kann zu Änderungen des Web Serivces führen. Dafür ist die Operation **updateWebServices** zuständig. Bei der Änderung eines Artefakts muss der Anbieter eine Emailadresse, ein Passwort, die ID von einem Artefakt, die neue Beschreibung und das neue Inhalt eingeben. Die Emailadresse und das Passwort dienen zusammen der Authentifizierung des Anbieters. Die Id des Artefakts dient zum Auswahl des Artefakts. Ist der Anbieter authentifiziert, wird das Artefakt durch die gegebene ID gesucht. Fall das Artefakt existiert und keine WSDL-Artefakt ist, wird zuerst durch die Artefakt ID das entsprechende WSDL-Artefakt gesucht.

Wenn gefunden:

dann wird irgendein Web Service durch die ID des gefundenen WSDL-Artefakts gesucht. Die Anbieter ID eines gefundenen Web Services wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das Artefakt zum Anbieter. Dann können die alten Informationen durch neue ersetzt werden. Nach der Aktualisierung wird die ID des Artefakts dem Anbieter zurückgegeben.

Wenn nicht gefunden:

wird durch die Artefakt ID die entsprechende Datenbank gesucht. Die Anbieter ID der gefundenen Datenbank wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das Artefakt zum Anbieter. Dann können die alten Informationen durch neue ersetzt werden. Nach der Aktualisierung wird die ID des Artefakts dem Anbieter zurückgegeben.

browseArtefacts

Eingabe:

Mandatory: suchMethode

Optional:

Ausgabe: Eine Menge von Artefakt-Metadaten

Beschreibung:

Für das Suchen von Artefakten ist die Authentifizierng des Anbieters nicht nötig, jeder Nutzer kann diese Funktion verwenden, es stehen fünf Suchmethoden zur Verfügung und zwar durch einen Typ von Artefakten, eine Beschreibung von Artefakten, einen Inhalt von Artefakten, einen Zeitinterval des Anlegens von Artefakten und den Typ mit einem Zeitraum des Anlegens von Artefakten. Als Ergebnis wird eine List von Artefakten Meta-Informationen zurückgegeben, die die ID, die Beschreibung und den Typ von Artefakten enthalten.

retrieveArtefact

Eingabe:

Mandatory: artefaktId

Optional:

Ausgabe: artefaktType, description, Inhalt des Artefakts

Beschreibung:

Bei der Abholung eines Artefakts muss man eine Artefakt ID eingeben. Das Artefakt wird durch die ID gesucht, falls gefunden, werden der Typ, die Beschreibung und der Inhalt vom Artefakt zurückgegeben.

retrieveArtefactBundle

Eingabe:

Mandatory: wsdlId | databaseId

Optional:

Ausgabe: Eine Menge von Artefakt-Metadaten

Beschreibung:

Bei der Abholung eines Artefaktbündels muss man eine ID eingeben. Die ID kann entweder eine WSDL-ArtefaktId oder eine DatenbankId sein.

Falls die ID eine WSDL-ArtefaktId ist:

wird das WSDL-Artefakt durch die ID gesucht, falls das WSDL-Artefakt existiert, werden alle Artefakte gesucht, die jeweils eine Relation mit dem WSDL-Artefakt haben. Für jedes gefundene Artefakt inklusive WSDL-Artefakt, wird der Typ, die Beschreibung und der Inhalt des Artefakts zurückgegeben

Falls die ID eine DatenbankId ist:

wird die Datenbank durch die ID gesucht, falls sie existiert, werden alle Artefakte gesucht, die jeweils eine Relation mit der Datenbank haben. Für jedes gefundene Artefakt ,wird der Typ, die Beschreibung und der Inhalt des Artefakts zurückgegeben

3.4.3 Operationen für Datenbanken

registerDatabase

Eingabe:

Mandatory: email, password, databaseName, databaseDriver, databaseAddress

Optional: description, Policy Dokumente

Ausgabe: databaseId

Beschreibung:

Bevor der Anbieter eine Datenbank registriert, muss der Anbieter sicherstellen, ob er sich schon registriert hat. Wenn Nein, muss der Anbieter sich zuerst registrieren, sonst muss der Anbieter eine Emailadresse und ein Passwort angeben, um sich zu authentifizieren. Ist der Anbieter authentifiziert, dann kann die Datenbank registriert werden. Bei der Registrierung der Datenbank muss der Anbieter Informationen über den Namen, dem Treiber und die Adresse von der Datenbank angeben. Diese drei Informationen sind für die Registrierung der Datenbank erforderlich. Die Datenbanknutzer können nur nach der Registrierung einer Datenbank separat angelegt werden. Für eine Datenbank wird eine in der Datenbank eindeutige ID automatisch generiert. Optional kann der Anbieter noch Policy Dokumente angeben, die mit der Datenbank assoziiert sind. Für jedes gegebene Policy Dokument wird jeweils ein Policy Artefakt angelegt und eine Relation wird von der Datenbank zum Artefakt

erstellt. Ist die Registrierung erfolgreich durchgeführt, wird die ID der Datenbank und die IDs der Artefakte zurückgegeben.

updateDatabase

Eingabe:

Mandatory: email, password, databaseId

Optional: databaseName, databaseDriver, databaseAddress, description

Ausgabe: databaseId

Beschreibung:

Nach der Registrierung einer Datenbank darf der Anbieter jeder Zeit die Informationen über die Datenbank ändern. Aber die Operation unterstützt nicht funktion die mit der Datenbank assoziierten Policy Artefakte zu ändern. Das Registerservice bietet eine andere Operation **updateArtefakt** an, die für die Änderung eines Non-WSDL-Artefakts zuständig ist. Bei der Änderung muss der Anbieter eine Emailadresse, ein Passwort, eine ID einer Datenbank und neue Informationen über die Datenbank eingeben. Die Emailadresse und das Passwort dienen zusammen der Authentifizierung des Anbieters. Database ID dient zur Auswahl der Datenbank. Ist Der Anbieter authentifiziert, wird eine Datenbank durch die gegebene Datenbank ID gesucht. Wurde die Datenbank gefunden, wird die ID des Anbieters von der gefundenen Datenbank mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Dann darf der Anbieter erst die Informationen über die Datenbank ändern, weil der Anbieter nur die Informationen eigener Datenbank ändern darf. Es wird dann kontrolliert, welche optionale Parameter vom Anbieter gegeben sind. Die Werte der vorkommenden optionalen Parameter werden jeweils mit dem entsprechenden im Datenbank gespeicherten Wert verglichen, wenn die beiden Werte nicht identisch sind, wird der in der Datenbank gespeicherte Wert durch den gegebenen Wert ersetzt. Wenn die Änderung fehlerfrei ausgeführt wird, wird die ID der Datenbank zurückgegeben.

addNewDatabaseUser

Eingabe:

Mandatory: email, password, databaseId, userName, userPassword

Optional: street, city, zipcode, country, telephone, website, userEmail

Ausgabe: userId

Beschreibung:

Beim Anlegen des Nutzers von einer Datenbank, muss der Anbieter eine Emailadresse, ein Passwort, eine ID einer Datenbank und die Informationen über den neuen Nutzer eingeben. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Die Datenbank ID dient zur Auswahl der Datenbank. Ist der Anbieter authentifiziert, wird eine Datenbank durch die gegebene Datenbank ID gesucht. Wird die Datenbank gefunden, wird die ID des Anbieters von der gefundenen Datenbank mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Dann darf der Anbieter einen neuen Datenbanknutzer anlegen, weil der Anbieter nur Nutzer für eigene Datenbanken anlegen darf. Der Anbieter muss noch Informationen von userName, userPassword anbieten. Es wird kontrolliert, ob noch Werte der optionalen Felder vorhanden sind. Falls der Nutzer erfolgreich angelegt ist, wird eine ID für den Nutzer automatisch generiert und dem Anbieter zurückgegeben.

updateDatabaseUser

Eingabe:

Mandatory: email, password, databaseId, userId

Optional: street, city, zipcode, country, telephone, website, userPassword, userEmail

Ausgabe: userId

Beschreibung:

Bei der Änderung des Nutzers von einer Datenbank, muss der Anbieter eine Emailadresse, ein Passwort, eine ID einer Datenbank, eine ID des Nutzers und neue Informationen vom Nutzer eingeben. Die Emailadresse und das Passwort dienen zusammen der Authentifizierung des Anbieters. Die Datenbank ID dient zur Auswahl der Datenbank. Nutzer ID dient zur Auswahl des Nutzers. Ist der Anbieter authentifiziert, wird eine Datenbank durch die gegebene Datenbank ID gesucht. Falls die Datenbank gefunden wird, wird die ID des Anbieters von der gefundenen Datenbank mit der durch die Authentifizierung zurückgegebene ID des Anbieters

verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Dann wird der Nutzer durch die gegebene Nutzer ID gesucht. Falls der Nutzer existiert und die Datenbank ID vom gefundenen Nutzer mit gegebener Datenbank ID identisch ist, darf der Anbieter die Informationen über den Datenbank Nutzer aktualisieren. Es wird dann kontrolliert, welche optionale Parameter vom Anbieter gegeben sind. Die Werte der vorkommenden optionalen Parameter werden jeweils mit dem entsprechenden in Datenbank gespeicherten Wert verglichen. Wenn die beiden Werte nicht identisch sind, wird der in der Datenbank gespeicherte Wert durch den gegebenen Wert ersetzt. Falls die Informationen geändert werden, wird die ID des Nutzers dem Anbieter zurückgegeben.

deleteDatabaseUser

Eingabe:

Mandatory: email, password, databaseId, userId

Optional:

Ausgabe: userId

Beschreibung:

Beim Löschen eines Nutzers von einer Datenbank, muss der Anbieter eine Emailadresse, ein Passwort, eine ID von einer Datenbank und eine ID des Nutzers eingeben. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Die Datenbank ID dient zur Auswahl der Datenbank. Die Nutzer IDdient zur Auswahl des Nutzers. Ist der Anbieter authentifiziert, wird eine Datenbank durch die gegebene Datenbank ID gesucht. Falls die Datenbank gefunden ist, wird die ID des Anbieters von der ausgewählten Datenbank mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Dann wird der Nutzer durch die gegebene Nutzer ID gesucht. Falls der Nutzer existiert und die Datenbank ID Nummer vom gefundenen Nutzer mit der gegebenen Datenbank ID identisch ist, darf der Anbieter den Nutzer löschen. Fall der Nutzer fehlerfrei gelöscht wird, wird die ID des Nutzers dem Anbieter zurückgegeben.

browseDatabaseUser

Eingabe:

Mandatory: email, password, databaseId, suchwert einer Suchmethode

Optional:

Ausgabe: eine Menge von Datenbank Nuzter

Beschreibung:

Beim Suchen der Nutzer von einer Datenbank, muss der Anbieter eine Emailadresse, ein Passwort, eine Id von einer Datenbank und eine Suchmethode eingeben. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Die Datenbank ID dient zur Auswahl der Datenbank. Ist der Anbieter authentifiziert, wird die Datenbank durch die gegebene Datenbank ID gesucht. Falls die Datenbank existiert, wird die ID des Anbieters von der gefundenen Datenbank mit der durch die Authentifizierung zurückgegebene ID des Anbieters verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Nur der Anbieter der Datenbank darf die Nutzer der Datenbank suchen. Der Anbieter kann eine von vier Suchmethoden auswählen:

findByUserName, der Suchwert wird mit den gespeicherten Namen aller Nutzer der Datenbank verglichen.

findByEmail, der Suchwert wird mit den gespeicherten Emails der aller Nutzer der Datenbank verglichen.

findByAll, der Suchwert wird mit alle gespeichterten Informationen aller Nutzer der Datenbank verglichen

findAll, braucht keinen Suchwert, alle Nutzer einer Datenbank werden gelistet und alle Informationen über den Nutzer dem Anbieter zurückgegeben.

deleteDatabase

Eingabe:

Mandatory: email, password, databaseId

Optional:

Ausgabe: databaseId, eine Menge von ArtefaktIds

Beschreibung:

Beim Löschen einer Datenbank, muss der Anbieter eine Emailadresse, ein Passwort und eine ID von einer Datenbank eingeben. Die Emailadresse und das Passwort dienen zusammen der Authentifizierung des Anbieters. Die Datenbank ID dient zur Auswahl der Datenbank. Ist der Anbieter authentifiziert, wird eine Datenbank durch die gegebene Datenbank ID gesucht. Falls die Datenbank existiert, wird die ID des Anbieters von der gefundenen Datenbank mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Falls die beiden IDs identisch sind, gehört die Datenbank zum Anbieter. Dann wird kontrolliert, ob die Nutzer der Datenbank noch vorhanden sind. Die Datenbanknutzer werden nämlich durch die Datenbank ID gesucht. Falls noch welche existieren, werden zuerst alle Nutzer der Datenbank gelöscht, dann wird weiter kontrolliert, ob noch Artefakte vorhanden sind, die Relationen mit der Datenbank haben. Ist dies der Fall, werden zuerst die Relationen gelöscht, dann die Artefakte. Danach kann die Datenbank gelöscht werden. Falls kein Nutzer der Datenbank existiert, kann die Datenbank direkt gelöscht werden. Ist die Datenbank erfolgreich gelöscht, dann wird die Datenbank ID dem Anbieter zurückgegeben.

valuateDatabase

Eingabe:

Mandatory: databaseId, Bewertungsnote

Optional:

Ausgabe: databaseId, Bewertungsanzahl, Note der Datenbank

Beschreibung:

Bei der Bewertung einer Datenbank, ist die Authentifizierung nicht nötig. Jeder kann eine Datenbank einfach bewerten. Für die Bewertung sind eine ID einer Datenbank und die Bewertungsnote erforderlich. Es wird die Datenbank zuerst durch die Datenbank ID gesucht, falls gefunden, erhöht sich die Bewertungsanzahl der gefundenen Datenbank um 1, die Note der Datenbank wird auf (Bewertungsnote + Note der Datenbank)/Bewertungsanzahl gesetzt. Falls die Bewertung der Datenbank fehlerfrei ausgeführt wird, wird die ID, die Bewertungsanzahl und die Note der Datenbank zurückgegeben.

browseDatabase

Eingabe:

Mandatory: Suchwert einer Methode

Optional:

Ausgabe: Eine Menge von Datenbanken

Beschreibung:

Für das Suchen der Datenbanken kann eine von sechs Suchmethoden ausgewählt werden:

findByProviderName, der Suchwert wird zuerst mit dem gespeicherten Namen aller Anbieter verglichen, es können mehrere Anbieter mit gleichem Namen gefunden werden. Dann werden die Datenbanken durch die IDs des gefundenen Providers gesucht

findByAll, der Suchwert wird mit allen gespeicherten Informationen aller Datenbanken verglichen

findByRating, der Suchwert muss eine Zahl zwischen Null und Zehn sein. Alle Datenbanken, dessen Noten größer gleich die gegebene Zahl sind, werden gesucht.

findbyDatabaseName, der Suchwert wird mit dem gespeicherten Datenbank Namen aller Datenbanken verglichen

findByPolicy, der Suchwert ist eine Policy. Die Suchmethode wird folgend detailliert beschrieben:

Es wird zuerst kontrolliert, ob die gegebene Policy im Nomalform ist. Ist dies nicht der Fall, wird die Policy normalisiert.

Nur die Datenbanken werden ausgewählt, mit denen ein oder mehrere Policy-Artefakte assoziiert sind.

Für jede ausgewählte Datenbank wird kontrolliert, ob die mit der Datenbank assoziierten Policies in der Normalform sind. Ist dies nicht der Fall, werden die Policies normalisiert.

Dann werden die Policies gemerged. Die durch Merge entstehende Policy wird mit der gegebenen normalisierten Policy verglichen, wenn die beiden Policies kompatibel sind, ist die Datenbank qualifiziert.

Falls welche gefunden werden, werden alle Informationen von den Datenbanken zurückgegeben.

3.4.4 Operationen für web Services

RegisterWebServices

Eingabe:

Mandatory: email, password, wsdlUri, WS-Bündel

Optional:

Ausgabe: eine Menge von ArtefaktIds, eine Menge von ServiceIds

Beschreibung:

Bei der Registrierung von Web Service muss der Anbieter eine Emailadresse, ein Passwort, ein WS-Bündel und die Adresse des WSDL Dokumentes liefern. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Wie schon beschrieben enthält das WS-Bündel genau ein WSDL Dokument und mehrere andere XML Dokumente. Das WSDL Dokument enthält alle Informationen über einen bzw. mehrere Web Services wie Service Name, Service Endpoint, Service Bindings, Service Operationen usw. Die Adresse des WSDL Dokumentes spezifiziert wo das WSDL Dokument vom Anbieter bereitgestellt wird. Ist der Anbieter authentifiziert, wird zuerst ein WSDL-Artefakt für das WSDL Dokument angelegt. Die WSDL-Artefakt ID wird automatisch generiert. Falls es noch weitere XML Dokumente im WS-Bündel vorkommen, wird für die Dokumente jeweils ein Artefakt mit einer ID erzeugt. Für jedes Non-WSDL-Artefakt wird eine Relation vom WSDL-Artefakt zum Artefakt angelegt. Dann wird das WSDL-Artefakt geparst, die Informationen über Service Name, Targetnamespace, Service Endpoint extrahiert und eine ID für jeweils einen Web Service automatisch generiert. Diese Informationen sind für die Erstellung vom Service Metadaten notwendig. Falls die Registrierung fehlerfrei ausgeführt wird, werden die IDs von Services und zusammen mit allen Artefakt IDs dem Anbieter zurückgegeben.

updateWebServices

Eingabe:

Mandatory: email, password, (wsdlId, wsdldokument) | (serviceId, [wsdluri], [serviceDesription])

Optional:

Ausgabe: Eine Menge von serviceIds

Beschreibung:

Bei der Änderung von Web Services, darf der Anbieter nur die Informationen der WSDL Adresse und der Service Description von Service Metadaten oder dem Inhalt des WSDL-Artefakts ändern. Für die Änderung der Non-WSDL- Artefakte steht die Operation **updateArtefakt** zur Verfügung.

Der Anbieter gibt eine Emailadresse und ein Passwort ein, um sich zu authentifizieren. Ist der Anbieter authentifiziert, wird kontrolliert, was geändert werden soll, ist es der Inhalt von WSDL Dokument:

dann wird das WSDL-Artefakt durch die gegebene ID gesucht. Fall das WSDL-Artefakt existiert, sucht irgendein Web Service durch die ID des WSDL-Artefakts. Die Anbieter ID eines gefundenen Web Services wird mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das WSDL-Artefakt zum Anbieter. Dann werden alle Service Metadaten, die vom WSDL-Artefakt abhängig sind, gelöscht. Dann kann das alte WSDL Dokument durch das neue ersetzt werden und das geänderte WSDL-Artefakt wird wie bei der Registrierung der Web Service neu geparst. Die Matadaten Information für Services wird neu extrahiert und wieder gespeichert. Die neuen Service IDs werden dem Anbieter zurückgegeben.

Handelt es sichdabei Metadaten von einem Service zu ändern:

wird der Service durch eine gegebene serviceID gesucht. Fall sie existiert, wird die Anbieter ID des gefundenen Services mit der durch die Authentifizierung zurückgegebenen ID des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört der Service zum Anbieter. Dann können die alten Informationen durch die neuen entsprechend geändert und gespeichert werden.

deleteWebServices

Eingabe:

Mandatory: email, password wsdlId

Optional:

Ausgabe: eine Menge von ArtefaktIds und eine Menge von serviceID

Beschreibung:

Der Anbieter darf nicht nur ein einzelnes Web Service, sondern muss alle Web Services, die in einer WSDL Datei definiert sind löschen. Beim Löschen der Web Services muss der Anbieter eine Emailadresse, ein Passwort und die ID Nummer des WSDL-Artefakts eingeben. Die Emailadresse und das Passwort dienen zusammen zur Authentifizierung des Anbieters. Die ID Nummer des WSDL-Artefakts dient zur Auswahl des WSDL-Artefakts. Ist der Anbieter authentifiziert, wird das WSDL-Artefakt durch die gegebene ID Nummer gesucht. Fall das WSDL-Artefakt existiert, sucht irgendein Web Service durch die ID des WSDL-Artefakts. Die Anbieter ID eines gefundenen Web Services wird mit der durch die Authentifizierung zurückgegebenen ID Nummer des Anbieters verglichen. Wenn die beiden IDs identisch sind, gehört das WSDL-Artefakt zum Anbieter. Dann werden alle Web Services durch die WSDL ID gesucht. Alle gefundenen Web Services werden einfach gelöscht. Es werden auch alle Artefakte gesucht, die jeweils eine Relation mit dem WSDL-Artefakt haben. Falls gefunden, werden zuerst alle Relationen zwischen dem WSDL-Artefakt und den gefundenen Artefakten gelöscht und dann die Artefakte. Anschließend kann das WSDL-Artefakt gelöscht werden. Werden alle Web Services und alle Artefakte erfolgreich gelöscht, werden die IDs von gelöschten Web Services und Artefakten dem Anbieter zurückgegeben.

valuateWebService

Eingabe:

Mandatory: serviceId, Bewertungsnote

Optional:

Ausgabe: serviceId

Beschreibung:

Bei der Bewertung eines Web Services, ist eine Authentifizierung nicht nötig, jeder kann einen Web Service einfach bewerten. Für die Bewertung, sind die ID eines Web Services und die Bewertungsnote erforderlich. Das Web Service wird zuerst durch die Web Service ID gesucht, falls gefunden, erhöht sich die Bewertungsanzahl von der gefundenen Web Service um 1, die Note des Web Services wird auf (Bewertungsnote + Note der

Datenbank)/Bewertungsanzahl gesetzt. Falls die Bewertung des Web Services fehlerfrei ausgeführt wird, wird die ID, die Bewertungsanzahl und die Note vom Web Service zurückgegeben.

browseWebServices

Eingabe: Suchwert einer Suchmethode

Mandatory:

Optional:

Ausgabe: eine Menge von Services

Beschreibung:

Beim Suchen von Web Services kann eine von vier Suchmethoden ausgewählt werden:

findByAll, die Information wird in WSDL Artefakt durchgesucht, wenn gefunden, dann werden alle relevanten Web Services mit der ID, dem Service Namen, dem Anbieter Namen, der Beschreibung, Erstellungsdatum, der Bewertungsanzahl und der Bewertungsnote zurückgegeben.

findByServiceName, der gegebene Name wird von Service Metadaten durchgesucht. Falls gefunden, werden alle gefundenen Web Services zurückgeliefert.

findByRating, der Suchwert muss eine Zahl zwischen Null und Zehn sein. Alle Web Services, dessen Noten größer gleich die gegebene Zahl sind, werden gesucht.

findByPolicy Das suchen durch Service ID, eine Operation und eine gegebene Policy. Der Ausgangspunkt der Methode ist, dass ein Servicenutzer eine Operation von einem Service aufrufen will, die gegebene Policy beschreibt die Anforderung an den Service vom Servicenutzer. Wie diese Methode funktioniert, wird im folgenden beschrieben.

Zuerst ist die Service ID und eine Operation bekannt, dadurch können das entsprechende WSDL-Artefakt und alle Artefakte, die jeweils eine Relation mit dem WSDL-Artefakt hat, gefunden werden.

Die Policy ist entweder als Kindelement von WSDL definiert oder in einem externen Artefakt gespeichert. Für beide Fälle, kann festgestellt werden, mit welchen WSDL-Elementen die Policies assoziiert sind. Dann wird eine Policy, der zu dem gefundenen Service gehört für jeweils einen Endpoint wie folgend berechnet:

Berechnen der effektiven Policy für das Service Policy Subject, die einen Merge von allen Policies ist und die mit dem entsprechenden wsdl:service Element assoziiert.
Berechnen der effektiven Policy für das Endpoint Policy Subject, die einen Merge von den mit dem vom Service verwendeten wsdl:port Element assoziierten Policies, den mit dem vom Port genutzten wsdl:binding Element assoziierten Policies und den mit dem vom Binding eingesezten wsdl:portType assoziierten Policies.
Berechnen der effektiven Policy für das Operation Policy Subject, die einen Merge von den mit dem wsdl11:portType/wsdl11:operation Element assoziierten Policies und den mit dem wsdl11:binding/wsdl11:operation Element assoziierten Policies.
Die effektive Policy für Messages wird ebenfalls berechnet.
Die gegebene Policy in der Normalform wird mit der durch Merge aller vier effektiven Policies entstandenen Policy verglichen. Falls beide Policies kompatibel sind, dann wird der Service mit entsprechenden Bindings und Operationen zurückgegeben.

retrieveWebService

Eingabe:

Mandatory: serviceId

Optional:

Ausgabe: detaillierte Informationen über das Service

Beschreibung:

Durch die eingegebene Web Service ID kann zuerst das WSDL-Artefakt gefunden werden. Das gefundene WSDL-Artefakt wird geparst und die Informationen über die Bindings, die Operationen und von den Operationen verwendeten Messages werden aus dem WSDL-Artefakt extrahiert und zurückgegeben.

4 Implementierungen

Das Registerservice ist auf Fragmento aufgebaut. Die meisten Operationen vom Fragmento, bezogen auf Artefakte sind sehr wichtig für das Registerservice. Einige Operationen von Fragmento werden geändert um sich an das Registerservice anzupassen und einige werden vom Registerservice direkt als Bestandsteile übernommen.

4.1 Repository Datenbank

Die von Fragmento verwendete Datenbank „Repository“ enthält 11 Tabellen, wie in Abbildung 5 gezeigt. Um das Registerservice zu implementieren, wird die Datenbank Repository um 4 Tabellen erweitert (gezeigt in Abbildung 6):

- Die Tabelle Provider, die die Informationen der registrierten Anbieter der Web Services bzw. der Datenbanken speichert. Die Tabelle enthält 11 Felder: Uid, Providername, Street, Zipcode, City, Country, Telephone, Website, Email, Password, Created.

  Die Felder der Tabelle providername, email und password dürfen nicht leer sein, das Feld Uid ist Keyfeld der Tabelle. Wenn ein neuer Anbieter angelegt wird, wird ein Wert vom Feld automatisch generiert und dem Anbieter zugeordnet. Das Uid identifiziert den Anbieter eindeutig und entspricht die ID des Anbieters.
- Die Tabelle Servicemetadata, die die Meta-Informationen der registrierten Web Services speichert. Die Tabelle enthält 10 Felder: Uid, Provider_Uid, Servicename, WSDLUri, WSDLUid, Description, Serviceendpoint, Targetnamespace, Nummberofvote, Userrating.
- Die Tabelle Database, die die Informationen der registrierten Datenbanken speichert. Die Tabelle enthält 9 Felder: Uid, Dbname, Dbdriver, Description, DbUri, Numberofvote, Userrating, Created, Provider_Uid.
- Die Tabelle Dbuser, die die Informationen der Nutzer der registrierten Datenbanken speichert. Die Tabelle enthält 12 Felder: Uid, Username, Street, Zipcode, City, Country, Telephone, Website, Email, Password, Created, Database_Uid.

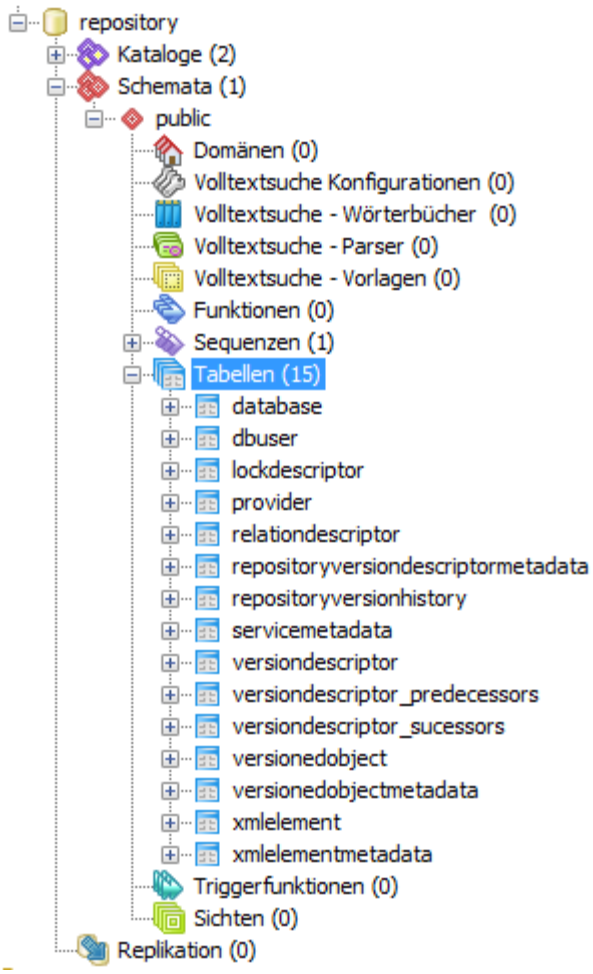

Abbildung 15: Die Datenbank Repository

4.2 Die Klassen und Methoden

Die Klasse **Provider** definiert eine Menge von Datentypen, die die Felder der Tabelle provider der Datenbank entsprechen und die getter und setter Methoden, die für die Operationen an die definierten Daten ermöglichen. Die Klasse spielt für das Anlegen der Tabelle eine zentrale Rolle.

Das Interface **GenericDao**, die eine Menge von Methoden definiert, die für die Operationen an den Tabellen der Datenbank zuständig sind:

- Die Methode **persist**(entitiy) ermöglicht einen neuen Datensatz in der entsprechenden Tabelle der Datenbank zu speichern.

- Die Methode **find**(primaryKey) ermöglicht einen Datensatz durch einen Primary Key in einer Tabelle der Datenbank zu suchen.
- Die Methode **findAll**() ermöglicht alle Datensätze einer Tabelle der Datenbank zu listen.

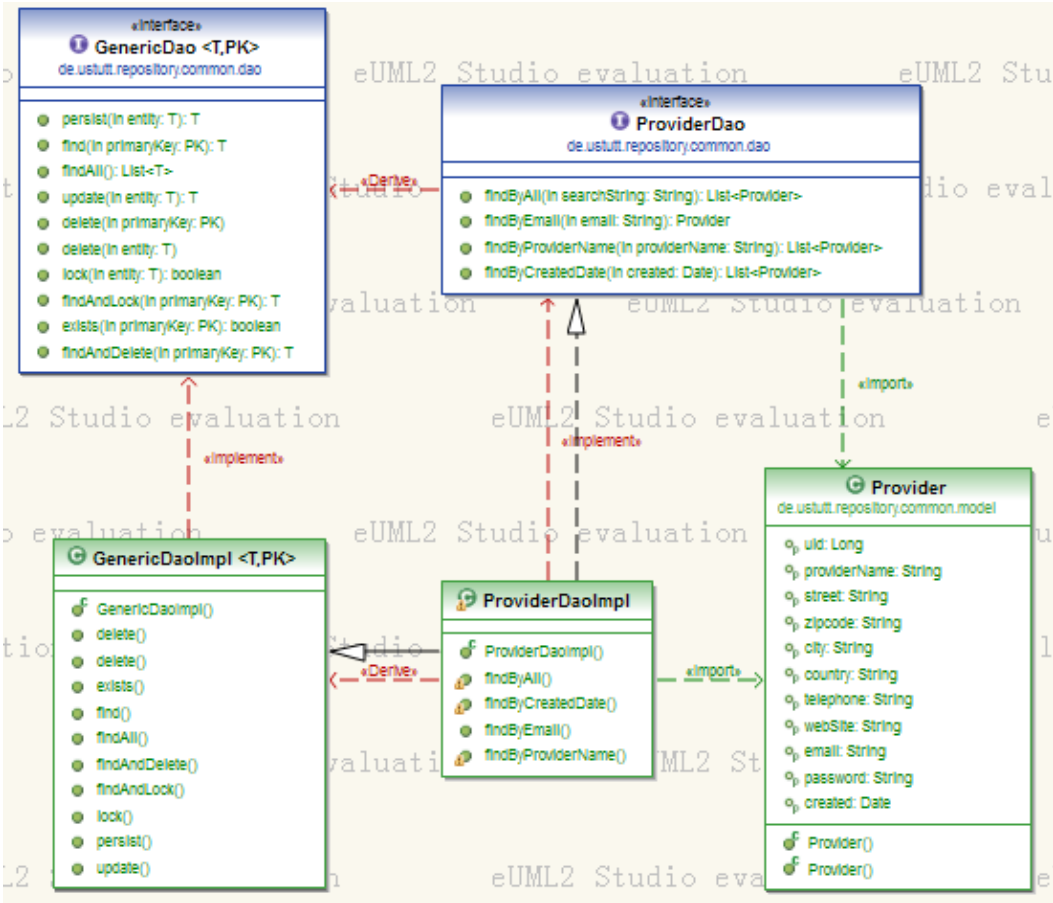

Abbildung 16: Klassendiagramm

- Die Methode **update**(entity) ist zuständig für die Änderung eines in einer Tabelle der Datenbank gespeicherten Datensatzes.
- Die Methode **delete**(primaryKey) bietet die Möglichkeit an, um einen Datensatz einer Tabelle der Datenbank durch den Primary Key zu löschen.
- Die Methode **delete**(entity) ermöglicht einen gegebenen Datensatz einer Tabelle der Datenbank zu löschen.
- Die Methode **lock**(entity)
- Die Methode **findAndLock**(primaryKey)
- Die Methode **exists**(primaryKey) kontrolliert, ob ein Datensatz mit dem Primary Key in einer Tabelle der Datenbank existiert.

- Die Methode **findAndDelete**(primaryKey) ermöglicht einen Datensatz in einer Tabelle der Datenbank durch PrimaryKey zu suchen. Wenn gefunden, dann wird der Datensatz gelöscht.

Die Klasse **GenericDaoImpl** implementiert alle Methoden, die im Interface **GenericDao** definiert sind.

Das Interface **ProviderDao** ist vom Interface GenericDao vererbt. Alle die im **GenericDao** definierten Methoden sind deswegen auch im ProviderDao definiert. Außerdem werden noch vier weitere Methoden definiert:

- Die Methode **findByAll**(searchString: String) ermöglicht das Suchen der Datensätze der Tabelle provider der Datenbank durch eine beliebige Eingabe.
- Die Methode **findByEmail**(email: String) ist zuständig für das Suchen von einen Datensatz in der Tabelle provider der Datenbank durch eine gegebene Emailadresse.
- Die Methode **findByProviderName**(providerName: String) bietet die Möglichkeit an, die Datensätze in der Tabelle provider durch einen gegebenen Provider Name zu suchen.
- Die Methode **findByCreateDate**(created: Date) ermöglicht die Datensätze der Tabelle provider der Datenbank durch das Anlegungsdatum zu suchen.

Die Klasse **ProviderDaoImpl** ist von der Klasse **GenericDao** vererbt und implementiert alle Methoden, die im Interface ProviderDao definiert sind.

Die Klasse **ServiceMetadata** definiert eine Menge von Daten, die die Felder der Tabelle provider der Datenbank entsprechen und die getter und setter Methoden, die für die Operationen an die definierten Daten ermöglichen.

Das Interface **ServiceMetadataDao** ist vom Interface **GenericDao** vererbt. Außer die vom **GenericDao** definierten Methoden, sind noch sieben Methoden vom **ServiceMetadataDao** definiert:

- Die Methode **findByAll**(searchString: String) ermöglicht das Suchen der Datensätze der Tabelle **servicemetadata** der Datenbank durch eine beliebige Eingabe.
- Die Methode **findByServiceName**(serviceName: String) ermöglicht das Suchen der Datensätze der Tabelle **servicemetadata** der Datenbank durch einen gegebenen Service Name.

- Die Methode **findByServiceEndpoint**(serviceEndpoint: String) bietet die Möglichkeit an, ein Service durch gegebenen Service Endpoint zu suchen.
- Die Methode **findByCreatedDate**(created: Date) ist genau wie die Methode für **ProviderDao**.

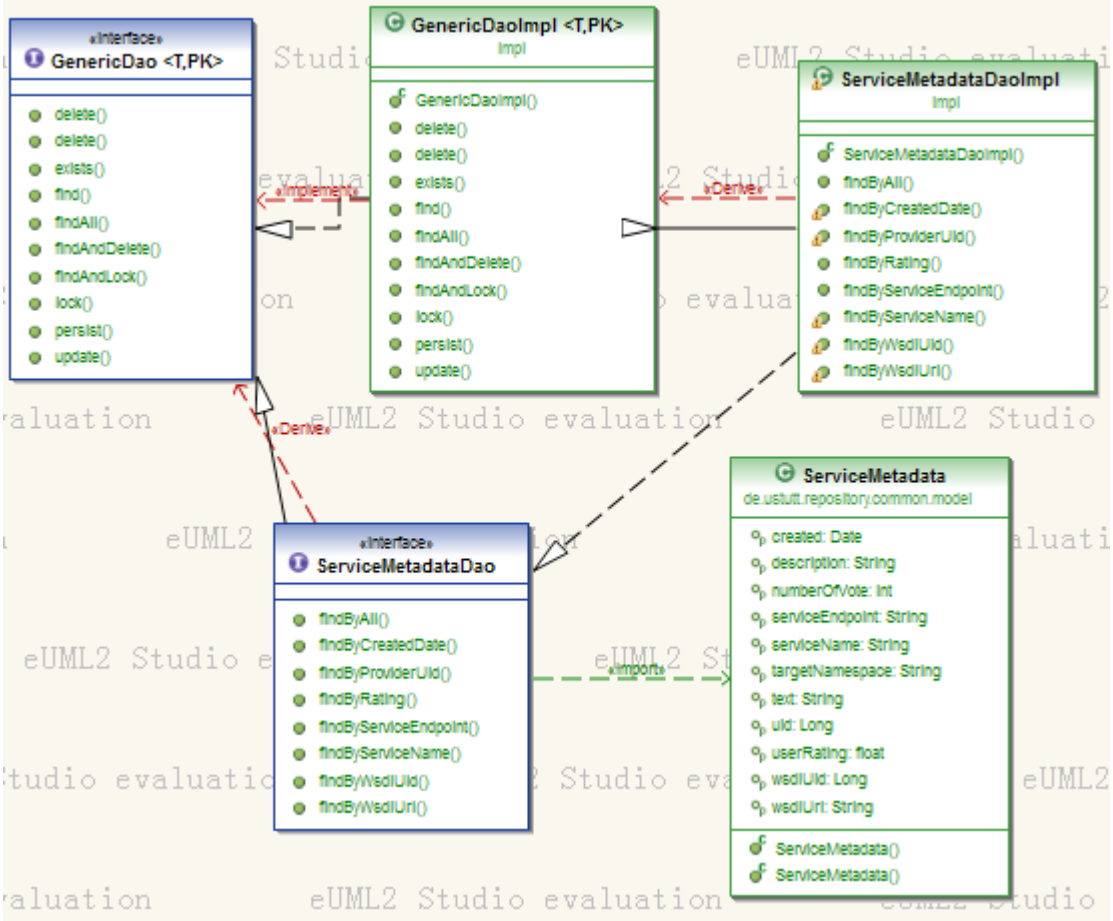

Abbildung 17: Klassendiagramm

- Die Methode **findByWsdlUri**(wsdlUri: String) ermöglicht das Suchen durch ein gegebenes WSDL-URI.
- Die Methode **findByWsdlUid**(wsdlUid: Long) ist zuständig für das Suchen durch eine gegebene WSDL Uid Nummer.
- Die Methode **findbyProviderUid**(wsdlUid: Long) ermöglicht die Datensätze in der Tabelle **servicemetadata** der Datenbank durch eine gegebene Provider ID Nummer zu suchen.

Die Klasse **ServiceMetadataDaoImpl** ist vom **GenericDaoImpl** vererbt und implementiert alle im **ServiceMetadataDao** definierten Methoden.

Die Klasse **Database** definiert eine Menge von Daten, die die Felder der Tabelle database der Datenbank entsprechen und die getter und setter Methoden, die für die Operationen an die definierten Daten ermöglichen.

Das Interface **DatabaseDao** ist vom Interface **GenericDao** vererbt. Alle in **GenericeDao** definierten Methoden sind ebenfalls vom **DatabaseDao** definiert, außerdem sind noch fünf Methoden vom **DatabaseDao** definiert:

- Die Methode **findByAll(**searchString: String**)** ist genau wie die Methode für **ProviderDao**.
- Die Methode **findByDbName(**dbName: String**)** ermöglicht die Datensätze in der Tabelle database der Datenbank durch den gegebenen Datenbank Name zu suchen.
- Die Methode **findByDbUri(**dbUri: String**)** bietet die Möglichkeit an, einen Datensatz von der Tabelle database durch gegebenes Datenbank URI zu suchen.

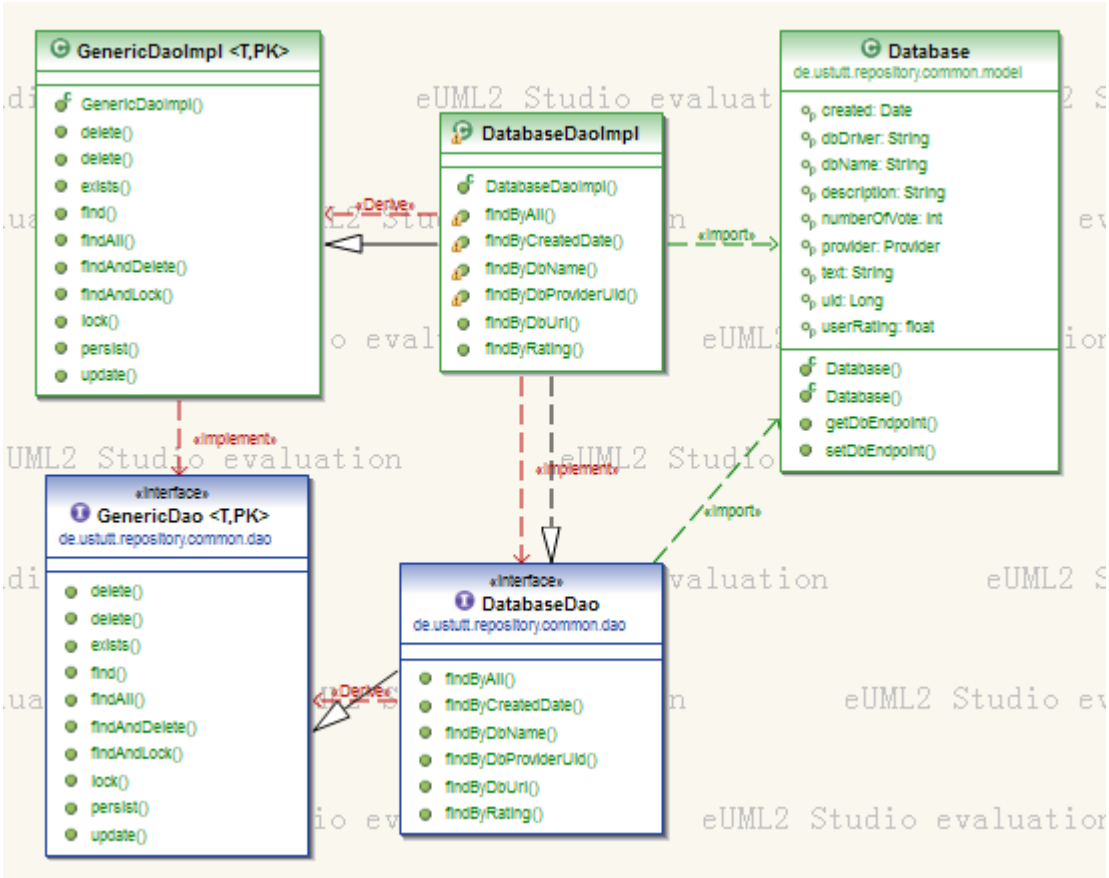

Abbildung 18: Klassendiagramm DatabaseDaoImpl

- Die Methode **findByDbProviderUid(**dbProviderUid: Long**)** ist zuständig für das Suchen von Datensätze durch eine gegebene Provider ID Nummer.

- Die Methode **findByCreatedDate(**created: Date**)** ist genau wie die Methode für **ProviderDao**.

Die Klasse **DatabaseDaoImpl** ist vom **GenericDaoImpl** vererbt und implementiert alle im **DatabaseDao** definierten Methoden.

Die Klasse **DbUser** definiert eine Menge von Daten, die die Felder der Tabelle dbuser der Datenbank entsprechen und die getter und setter Methoden, die für die Operationen an die definierten Daten ermöglichen.

Die Klasse **DbUserDao** ist vom Interface **GenericDao** vererbt. Alle im **GenericDao** definierten Methoden sind deswegen auch im **ProviderDao** definiert. Außerdem werden noch vier weitere Methoden definiert:

- Die Methode **findByAll(**searchString: String**)** ist genau wie die Methode für **ProviderDao**.
- Die Methode **findByUserName(**userName: String**)** ermöglicht das Suchen der Datensätze durch einen gegebenen User Namen.
- Die Methode **findByDatabaseUid(**databaseUid: Long**)** ist zuständig für das Suchen der Datensätze durch eine gegebene Database ID Nummer.
- Die Methode **findByCreatedDate(**created: Date**)** ist genau wie die Methode für **ProviderDao**.

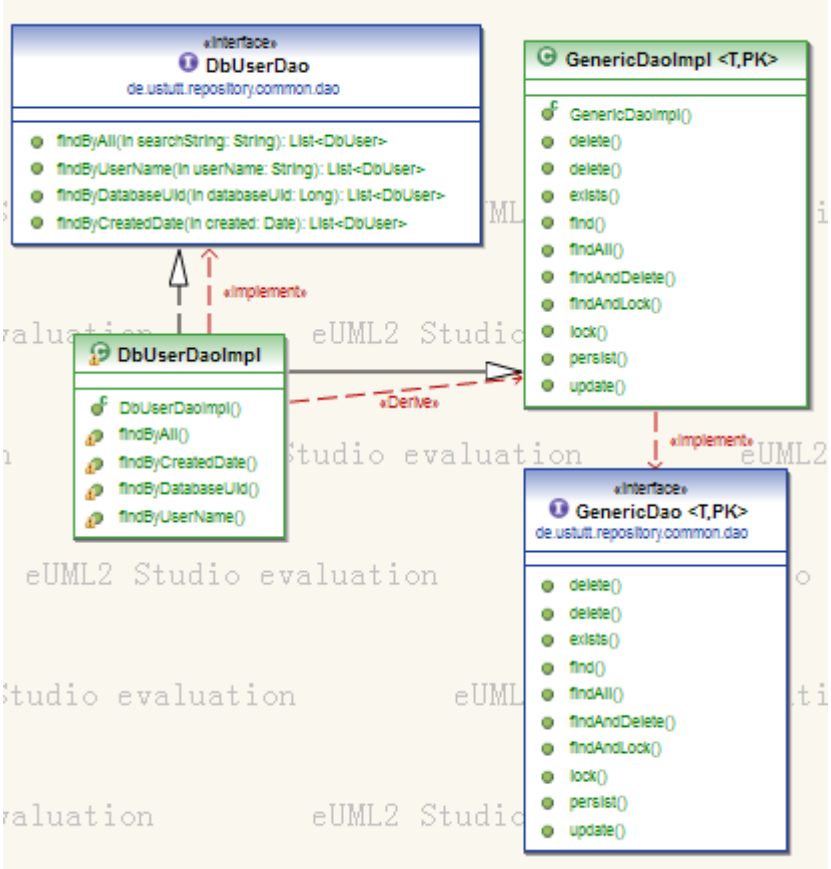

Abbildung 19: Klassendiagramm DbUserDaoImpl

Die Klasse **DbUserDaoImpl** ist vom **GenericDaoImpl** vererbt und implementiert alle im **DbUserDao** definierten Methoden.

Die Klasse **VersionDesriptor** definiert die Struktur eines Artefakts, das Interface **VersionDescriptorDao** definiert die möglichen Operationen und wird von der Klasse **VersionDescriptorDaoImpl** implementiert.

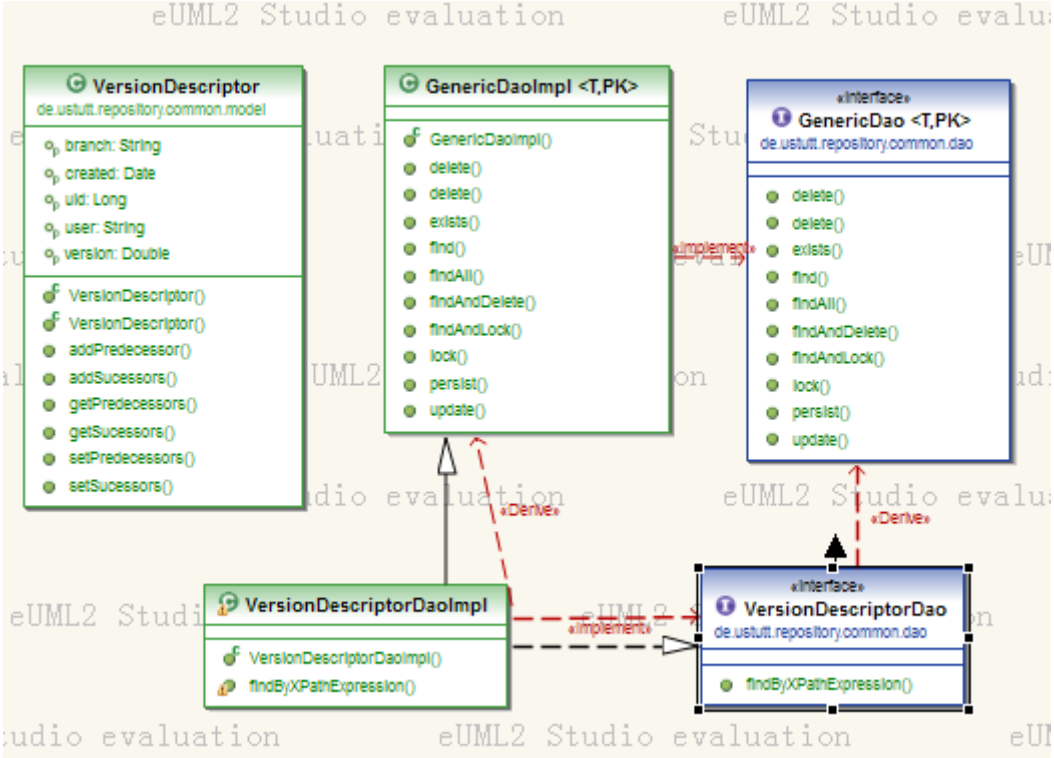

Abbildung 20: Klassendiagramm VersionDescriptorDaoImpl

Die für die Implementierung des RegisterSerive verwendeten Methoden:

Die Methode **find(**primarykey**)**

Die Methode **delete(**primaryKey**)**

Die Methode **update(**entity**)**

Die Methode **persist(**entitiy**)**

Die Klasse **RelationDescriptor** von Fragmento definiert das Relation Model. Das Interface **RelationManager** definiert eine Menge Operationen basierend auf das Relation Model. Die Klasse **RelationManagerImpl** ist die Implementierung von Interface. Das Folgende ist ein Klassendiagramm.

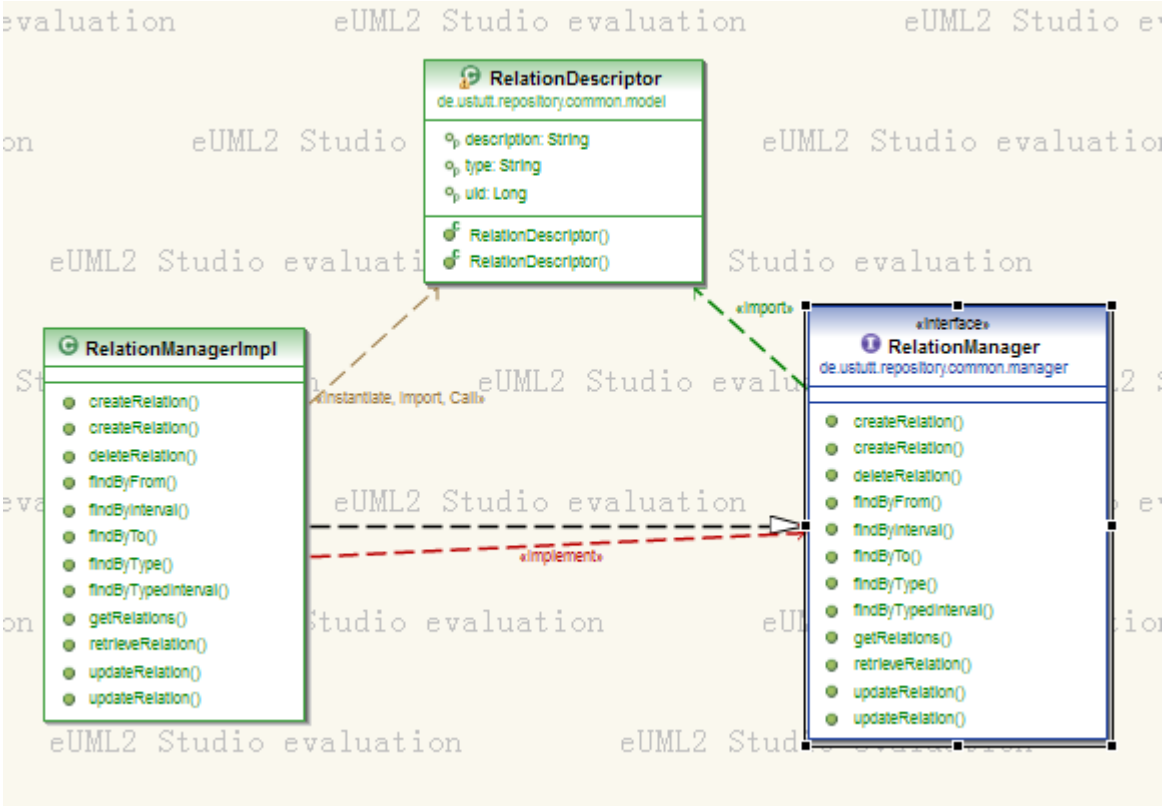

Abbildung 21: Klassendiagramm von RelationManagerImpl

Die für die Implementierung des Registerservice verwendeten Methoden:

Die Methode **findByFrom(**from**)**

Die Methode **findByTo(**to**)**

Die Methode **deleteRelation(**primaryKey**)**

Die Methode createRelation**(**from, to, description**)**

Die Klassen **ServiceItem**, **OperationItem** und **ParameterItem** definieren die Struktur von einem Service, die Klasse **WsdlParser** ist zuständig für das Parsen von WSDL Artefakten.

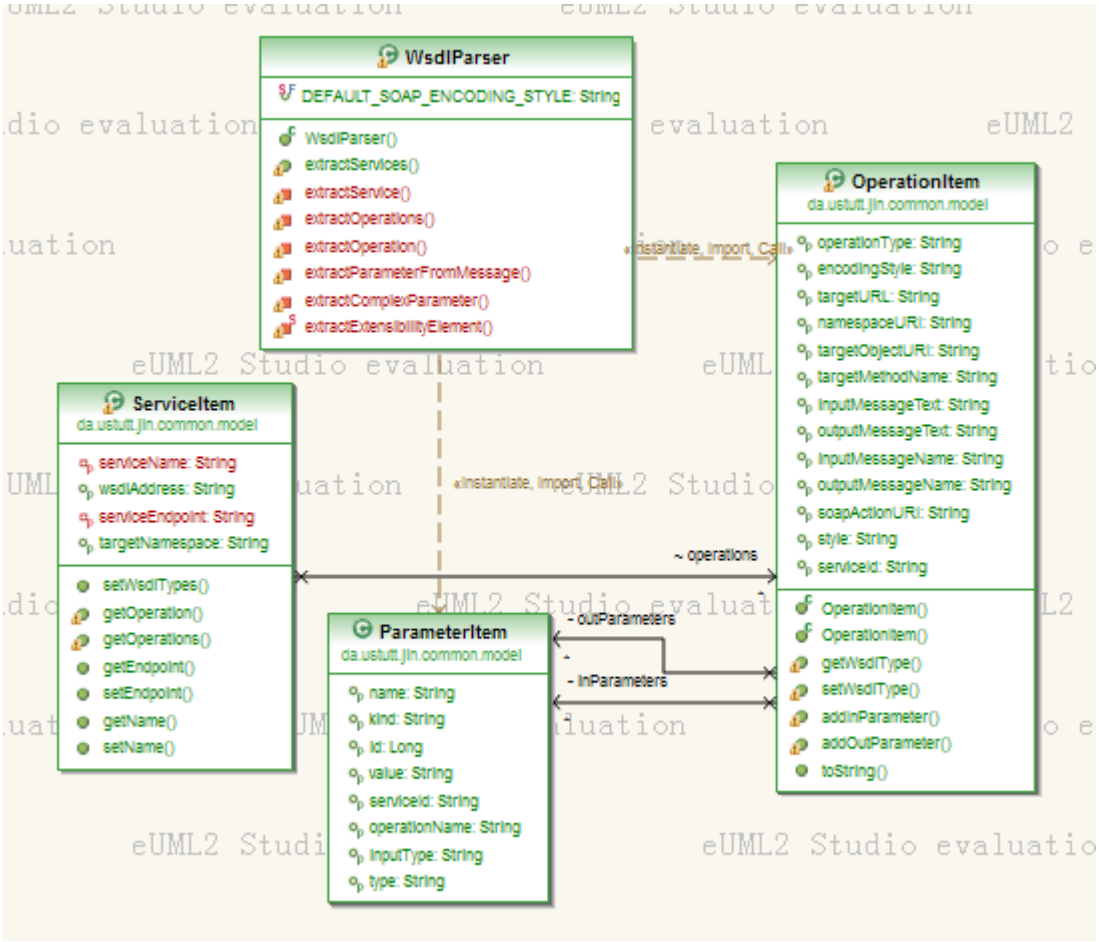

Abbildung 22: Klassendiagramm WsdlParser

Die Methode **extractServices()**, nimmt ein WSDL Dokument vom Typ Document als Eingabe, extrahiert alle **javax.wsdl.Servcice** Elemente vom WSDL Dokument, ruft die Methode **extractService()** auf, um die Objekte einzelnd zu verarbeiten und gibt dann eine Liste von **ServiceItem** Objekten aus.

Die Methode **extractService()**, die Methode nimmt ein **javax.wsdl.Service** Element und ein leeres **ServiceItem** Objekt als Eingabe, extrahiert alle **javax.wsdl.Binding** Elemente, ruft die Methode **extractOperations()** auf, um die Elemente einzelnd zu verarbeiten und ruft die Methode **extractExtensibilityElement()** auf, um die Adresse von Service zu finden. Dann setzt das **ServiceItem** Objekt auf die extrahierten Informationen und gibt das Objekt aus.

Die Methode **extractOperations()**, nimmt ein Binding Element als Eingabe, extrahiert das SOAPBinding Element und alle **javax.wsdl.BindingOperation** Elemente, und ruft die

Methode **extractOperation()** auf, um die Elemente einzelnd zu verarbeiten und gibt dann eine Liste von **OperationItem** Elementen aus.

Die Methode **extractOperation()**, nimmt ein **BindingOperation** Element und ein leeres OperationItem Objekt als Eingabe, extrahiert **SOAPOperation** Elemente und alle Message Elemente, ruft die Methode **extractParameterFromMesasage()** auf, um die Elemente einzelnd zu arbeiten, Dann setzt das **OperationItem** Objekt auf die extrahierten Informationen und gibt das Objekt aus.

Die Methode **extractParameterFromMessage()** ruft die Methode **extractComplexParameter()** auf, um alle primitiven Parameter zu extrahieren.

4.3 Realierung des Registerservice

Für die Impementierung des Registerservice wird zuerst ein WSDL Dokument „registerSerivce.wsdl“ erstellt. Im Dokument werden alle beschriebenen Operationen in einem PortType definiert, der Port Type wird nur von SOAP Binding verwendet.

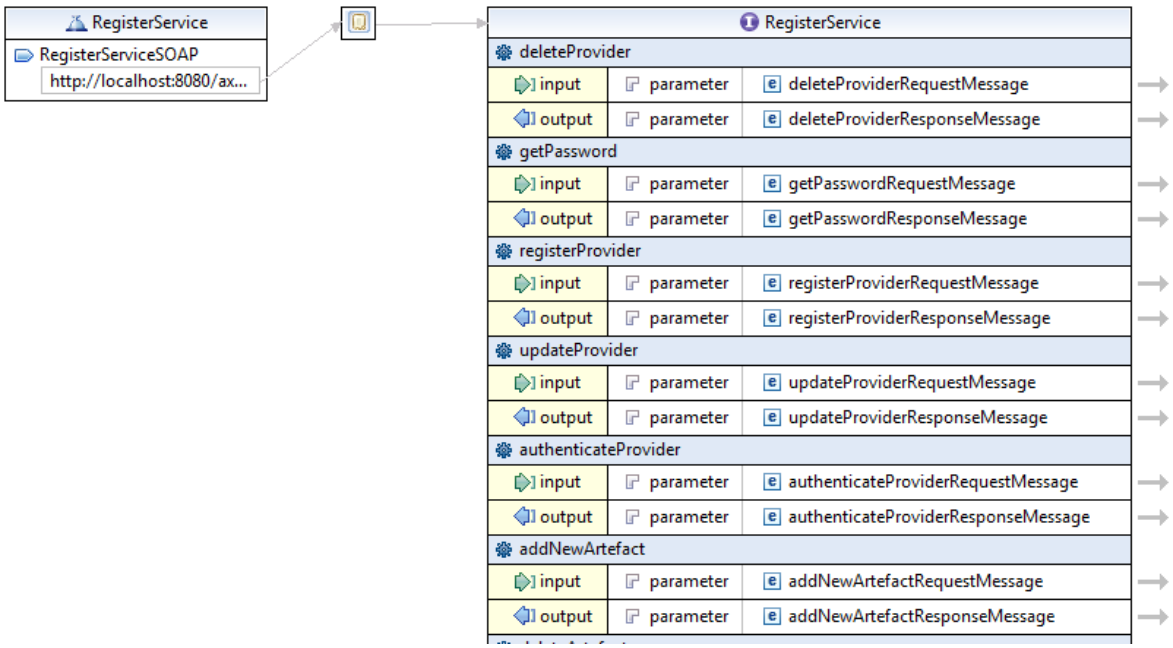

Abbildung 23: Ein Schnitt von WSDL Dokument vom Registerservice

Apach Axis2 bietet ein Werkzeug für die Code Generierung „wsdl2java“, um von WSDL Dokumente direkt in Java Codes zu generieren. Durch das Werkzeug wird eine Reihe von Klassen, einen Deployment Discriptor „service.xml“ und ein Registerservice.wsdl Dokument erzeugt. Die meisten von den generierten Klassen repräsentieren die Datentypen und Elementen, die in RegisterService.wsdl definiert wurden. Alle Klassen besitzen Eigenschaften und entsprechende „get“ und „set“ Methoden nach der Typdefinition im XML Schema vom WSDL Dokument und implementieren das Interface **ADBBean**. Die Klassen ermöglichen die Implementierung der Funktionalitäten vom Registerservice durch normale Java-Programmierung. Die SOAP Request Message enthaltenen Daten können durch entsprechende get-Methode ausgelesen werden und die zurückegegebenen Werte können durch entsprechende set-Methode in SOAP Response Message gesetzt [2] werden. Zum Beispiel ist die Klasse **RegisterProviderRequestMessage**(gezeigt in Abbildung 24) eine solche Klasse:

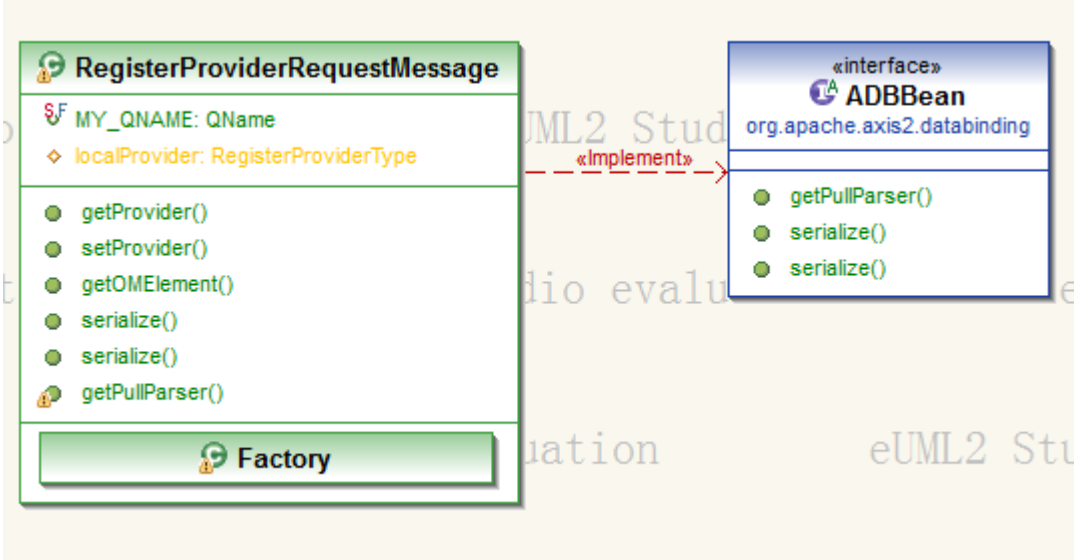

Abbildung 24: Klassendiagramm vom RegisterProviderRequestMessage

Die generierte Klasse **RegisterServiceMessageReceiverInOut**, ermöglicht, eingehende Message in Aufrufe an die richtige Service-Methode umzusetzen[2], die Klasse spielt keine große Rolle Bei der Entwicklung.

Das Interface **RegisterServiceSkeletonInterface** definiert die Schnittstelle für alle Methoden, die im WSDL Dokument definierten Operationen entsprechen. Die Klasse **RegisterServiceSkeleton** implementiert das Interface **RegisterServiceSkeletonInterface**.

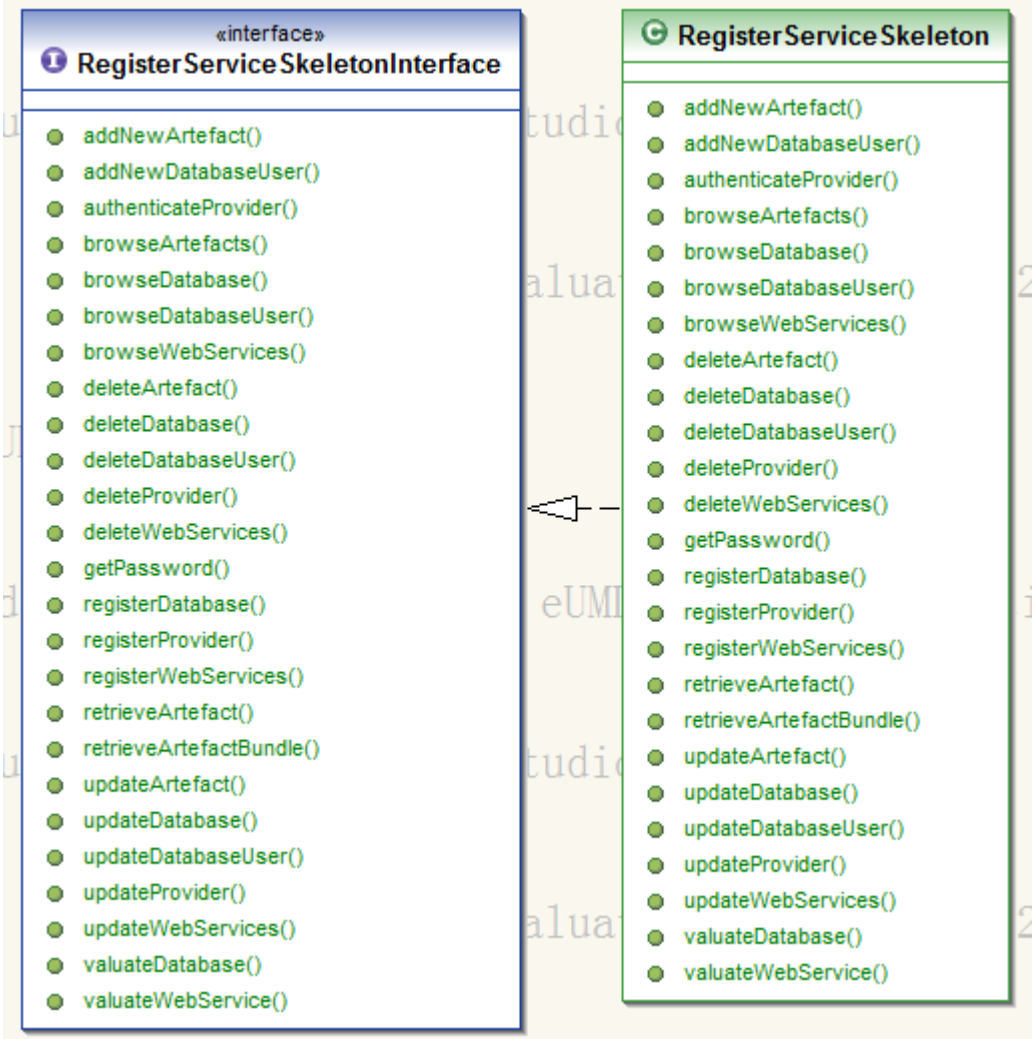

Abbildung 25: Klassendiagramm von RegisterServiceSkeleton

Die Klasse **RegisterServiceImpl,** wie in Abbildung 26 gezeigt, ist von der Klasse **RegisterServiceSkeleton** abgeleitet und überschreibt deren Methode. Außerdem enthält die Klasse noch einige weitere Methoden, die zu der Implementierung der in der Klasse **RegisterServiceSkeleton** definierten Methoden dienen. Die Klasse ist zuständig für die Implementierung vom Registerservice. Eigentlich kann die Implementierung vom Registerservice einfach direkt in die Klasse **RegisterServiceSkeleton** eingefügt werden, aber das ist nicht erwünscht, weil die Klasse im Laufe der Entwicklung neu generiert werden könnten (wenn eine neue Operation in WSDL Dokument definiert wird), das bedeutet, dass die neu generierte **RegisterServiceSkeleton** Klasse die alte ersetzt und irgendwie die Codes der alten Klasse in die neue hinein kopiert werden muss, das macht es aber sehr umständlich.

Wenn die Implementierung in einer abgeleiteten Klasse steht, kann die Klasse **RegisterServiceSkeleton** beliebig oft gelöscht werden.

Abbildung 26: Klassendiagramm RegisterServiceImpl

4.3.1 Implementierung der Operationen für Anbieter

registerProvider

Die Methode **registerProvider()** ist für die Implementierung der Operation **registerProvide**r zuständig.

Die Request Message hat den Messagetyp **RegisterProviderRequestMessage** und muss name, email1, emal2, password1, password2 enthalten, optional können street, city, zipcode, country, telephone, website, in der Message verpackt werden. Durch die get-Methode der Klasse, können die Daten vom SOAP Request Message ausgelesen werden.
Die Response Message hat den Messagetyp **RegisterProviderResponseMessage**. Durch die set-Methode kann die zürückgegebene providerID in SOAP Response Message gesetzt werden.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **persist()** von **ProviderDao** wird aufgerufen, um die Informationen in Datenbank zu speichern
2. Der Konstruktor **SendEmail()** der Klasse **SendEmail**, ermöglicht das Verschicken einer Email um die Registrierung zu bestätigen.

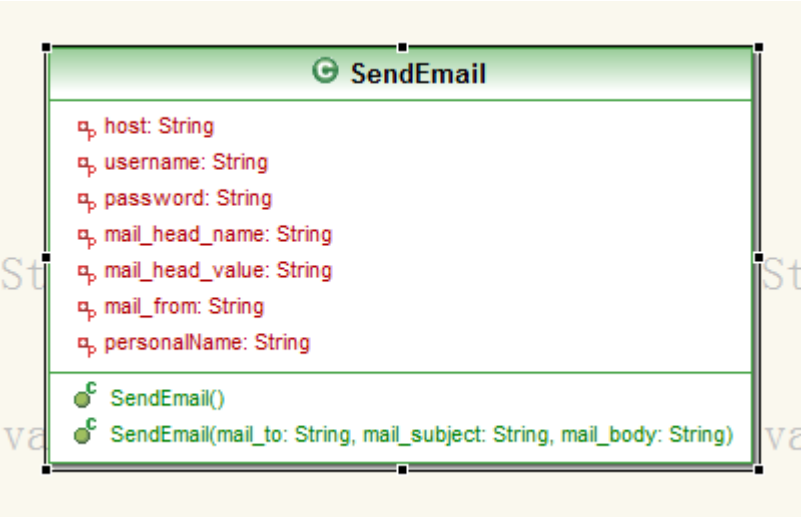

Abbildung 27: Klassendiagramm von SendEmail

authenticateProvider

Die Methode **authenticateProvider ()** ist für die Implementierung der Operation **authenticateProvider** zuständig.

Die Request Message hat den Messagetyp **AuthenticateProviderRequestMessage**, durch die getEmail() und die getPassword() Methoden der Klasse, ist es möglich, die Email und das Passwort von SOAP Request Message auszulesen.

Die Response Message hat den Messagetyp **AuthenticateProviderResponseMessage**, durch die Methode setPoviderId() der Klasse die zürückgegebene ProviderId in SOAP Response Message gesetzt wird.
Die Methode **findByEmail()** von **ProviderDao** wird eingesetzt, um ein Anbieter durch die ausgelesene Emailadresse in der Tabelle „provider“ zu suchen
Wird ein Provider gefunden, wird das gegebene Passwort mit dem Passwort des gefundenen Providers verglichen. Falls die beiden Werte identisch sind, dann ist der Anbieter authentifiziert. Die providerId wird zurück gegeben.

retrieveProvider

Die Methode **retrieveProvider()** ist für die Implementierung der Operation **retrieveProvider** zuständig.
Die Request Message hat den Messagetyp **RetrieveProviderRequestMessage** und muss providerId enthalten
Die Response Message hat den Messagetyp **RetrieveProviderResponseMessage**
Die von der Methode aufgerufenen Methoden:
Die Methode **find()** von **ProviderDao** wird aufgerufen, um diese Methode zu implementieren.

getPassword

Die Methode **getPassword()** ist für die Implementierung der Operation **getPassword** zuständig.
Die Request Message hat den Messagetyp **getPasswordRequestMessage** und muss email enthalten
Die Response Message hat den Messagetyp **getPasswordResponseMessage**
Die von der Methode aufgerufenen Methoden:
Die Methode **findByEmail()** von **Providerdao** wird aufgerufen, um Passwort zu holen.
Der Konstruktor **SendEmail()** der Klasse **SendEmail**, ermöglicht das Verschicken einer Email mit dem gefundenen Passwort.

.

updateProvider

Die Methode **updateProvider ()** ist für die Implementierung der Operation **updateProvider** zuständig.

Die Request Message hat den Messagetyp **updateProviderRequestMessage** und muss email und password enthalten, optional können name, street, city, zipcode, country, telephone, website, email1, emal2, password1, password2 in der Message verpackt werden.

Die Response Message hat den Messagetyp **updateProviderResponseMessage**

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticate()** wird aufgerufen, um zu überprüfen, ob der Anbieter existiert. Ist der Anbieter authentifiziert, wird die providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **find()** von **ProviderDao** wird aufgerufen, um den Datensatz des Anbieters abzuholen.
3. Die Methode **update()** von **ProviderDao** wird aufgerufen, um die Änderungen in Datenbank zu speichern.

deleteProvider

Die Methode **deleteProvider ()** ist für die Implementierung der Operation **deleteProvider** zuständig.

Die Request Message hat den Messagetyp **DeleteProviderRequestMessage** und muss email und password enthalten

Die Response Message hat den Messagetyp **DeleteProviderResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticate()** wird aufgerufen, um zu überprüfen, ob der Anbieter existiert. Ist der Anbieter authentifiziert, wird die providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **findByDbProviderUid()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob es noch registrierte Datenbanken vom Anbieter gibt, wenn Ja, gibt sie "null" aus.

3. Die Methode **findByProviderUid()** von **ServiceMetadataDao** wird aufgerufen, um zu überprüfen, ob es noch registrierte Web Services von der Anbieter gibt, wenn Ja, gibt sie "null" aus.
4. Die Methode **delete()** von **ProviderDao** aufgerufen, um den Datensatz des Anbieters zu löschen.

4.3.2 Implementierung der Operationen für Artefakte

addNewArtefact

Die Methode **addNewArtefact()** ist für die Implementierung der Operation **addNewArtefact** zuständig.

Die Request Message hat den Messagetyp **AddNewArtefact RequestMessage** und muss email, password, fromId, xmlDokument enthalten

Die Response Message hat den Messagetyp **AddNewArtefact ResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen. Ist der Anbieter authentifiziert, wird ProviderId geliefert, sonst gibt sie "null" aus.
2. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um zu überprüfen, ob das Artefakt existiert. Ist dies der Fall:
   i. Es wird überprüft, ob die providerId des gefundenen Artefakts und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.
   ii. Dann wird es kontrolliert, ob das Artefakt WSDL-Artefakt ist. Ist dies nicht der Fall, gibt sie "null" aus.
   iii. Die Methode **createArtefakt()** wird aufgerufen, um ein Artefakt anzulegen.
   iv. Die Methode **createRelation()** von **RelationManager** wird aufgerufen, um eine Relation von der Datenbank bzw. dem WSDL-Artefakt zum neuen angelegten Artefakt anzulegen
3. Sonst wird die Methode **find()** von **DatabaseDao** aufgerufen, um zu überprüfen, ob eine Datenbank existiert. Ist dies der Fall:

i. Es wird überprüft, ob die providerId der gefundenen Datenbank und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie “null” aus.

ii. Die Methode **createArtefakt()** wird aufgerufen, um ein Artefakt anzulegen.

iii. Die Methode **createRelation()** von **RelationManager**wird aufgerufen, um eine Relation von der Datenbank zum neuen angelegten Artefakt anzulegen**.**

4. Sonst gibt sie “null” aus.

deleteArtefact

Die Methode **deleteArtefact()** ist für die Implementierung der Operation **deleteArtefact** zuständig.

Die Request Message hat den Messagetyp **DeleteArtefact RequestMessage** und muss email, password, artefaktId enthalten

Die Response Message hat den Messagetyp **DeleteArtefact ResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. die Methode **Authenticate()** wird aufgerufen, um den Anbieter zu authentifizieren. Ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find()** von **VersionDesriptorDao** wird aufgerufen, um zu überprüfen, ob das Artefakt existiert. Ist dies nicht der Fall, gibt sie “null” aus.
3. Es wird kontrolliert, ob das gefundene Artefakt vom Typ WSDL ist. Ist dies der Fall, gibt sie “null” aus.
4. Die Methode **findbyTo()** von **RelationManger** wird aufgerufen, um das FromId zu finden. Die FromId ist entweder eine WSDL-ArtefaktId oder eine DatabaseId
5. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um zu überprüfen, ob ein WSDL-Artefakt durch die fromId gefunden werden kann, ist dies der Fall:
   i. Es wird überprüft, ob die providerId des gefundenen WSDL-Artefakts und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie “null” aus.

ii. Die Methode **deleteRelation()** von **RelationManager** wird aufgerufen, um die Relation vomWSDL-Artefakt zum Artefakt zu löschen.

iii. die Methode **delete()** von VersionDescriptorDao wird aufgerufen, um das Artefakt zu löschen.

6. Sonst:

i. Die Methode **find()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die providerId der gefundenen Datenbank und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus

ii. Die Methode **deleteRelation()** von **RelationManager** wird aufgerufen, um die Relation von der Datenbank zum Artefakt zu löschen.

iii. die Methode **delete()** von **VersionDescriptorDao** wird aufgerufen, um das Artefakt zu löschen.

updateArtefact

Die Methode **updateArtefact()** ist für die Implementierung der Operation **updateArtefact** zuständig.

Die Request Message hat den Messagetyp **UpdateArtefact RequestMessage** und muss email, password, fromId, enthalten

Die Response Message hat den Messagetyp **UpdateArtefact ResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **Authenticate()** wird aufgerufen, um den Anbieter zu authentifizieren. Ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **find()** von **VersionDesriptorDao** wird aufgerufen, um zu überprüfen, ob das Artefakt existiert. Ist dies nicht der Fall, gibt sie "null" aus.
3. Es wird kontrolliert, ob das gefundene Artefakt vom Typ WSDL ist. Ist dies der Fall, gibt sie "null" aus.
4. Die Methode **findbyTo()** von **RelationManager** wird aufgerufen, um die FromId zu finden. Die FromId ist entweder eine WSDL-ArtefaktId oder eine DatabaseId

5. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um zu überprüfen, ob ein WSDL-Artefakt durch die fromId gefunden werden kann, ist dies der Fall:
   i. Es wird überprüft, ob die providerId des gefundenen WSDL-Artefakts und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.
   ii. Die Methode **getXMLElement()** wird aufgerufen, um ein **XMLElement** Objekt von der Request Message abzuholen.
   iii. Die Methode **setXmlElement()** von **VersionDescriptor** wird aufgerufen, um den alten Inhalt durch den neuen zu ersetzen.
   iv. Die Methode **update()** von **VersionDescriptorDao** wird aufgerufen, um die Änderung in der Datenbank zu speichern.
6. Sonst ist die fromId eine ID einer Datenbank:
   i. Es wird überprüft, ob die providerId der gefundenen Datenbank und die durch die Authentifizierung gelieferte providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.
   ii. Die Methode **getXMLElement()** wird aufgerufen, um ein **XMLElement** Objekt von der Request Message abzuholen.
   iii. Die Methode **setXmlElement()** von der Klasse **VersionDescriptor** wird aufgerufen, um den alten Inhalt durch den neuen zu ersetzen.
   iv. Die Methode **update()** von **VersionDescriptorDao** wird aufgerufen, um die Änderung in der Datenbank zu speichern.

browseArtefacts

Die Methode **browseArtefacts()** ist für die Implementierung der Operation **browseArtefacts** zuständig.

Die Methode wird direkt von Fragmento übernommen.

retrieveArtefact

Die Methode **retrieveArtefact()** ist für die Implementierung der Operation **retrieveArtefact** zuständig.

Die Methode wird direkt von Fragmento übernommen.

retrieveArtefactBundle

Die Methode **retrieveArtefactBundle()** ist für die Implementierung der Operation **retrieveArtefactBundle** zuständig.

Die Request Message hat den Messagetyp **RetrieveArtefactRequestMessage** und muss eine WSDL-ArtefaktId oder eine databaseId enthalten.

Die Response Message hat den Messagetyp **RetrieveArtefactResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Es wird kontrolliert welches Artefakt-Bündel geholt werden soll, das Artefakt-Bündel von einer Datenbank oder von Web Services. Ist es ein Datenbank Artefakt-Bündel:
   i. Die Methode **find()** von DatabaseDao wird aufgerufen, um zu überprüfen, ob die Datenbank existiert. Ist dies der Fall, wird der Datensatz geholt, sonst gibt sie "null" aus
   ii. Die Methode **findByFrom()** von **RelationManager** wird aufgerufen, um die Artefakte zu finden, die jeweils mit der Datenbank in Relation stehen.
   iii. Für jedes gefundene Artefakt wird die Methode **retrieveArtefact()** aufgerufen, um die Artefakte zu holen.
2. Sonst:
   i. Die Methode **find()** von **ServiceMetadataDao** wird aufgerufen, um zu überprüfen, ob das WSDL-Artefakt existiert. Ist dies nicht der Fall, gibt sie "null" aus.
   ii. Die Methode **findByFrom()** von **RelationManager** wir aufgerufen, um die Artefakte zu finden, die jeweils mit dem WSDL-Artefakt in Relation stehen.
   iii. Für jedes Artefakt inklusiv dem WSDL Artefakt wird die Methode **retrieveArtefact()** aufgerufen, um die Artefakte zu holen.

4.3.3 Implementierung der Operationen für Datenbanken

registerDatabase

Die Methode **registerDatabase()** ist für die Implementierung der Operation **registerDatabase** zuständig.

Die Request Message hat den Messagetyp **RegisterDatabaseRequestMessage** und muss email, password, databaseName, databaseDriver, databaseUri enthalten. Optional können Policy Dokumente in der Message verpackt werden.

Die Response Message hat den Messagetyp **RegisterDatabaseResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticate()** wird aufgerufen, um zu überprüfen, ob der Anbieter existiert. Ist dies der Fall, wird eine providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **findByDbUri()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank schon registriert ist. Ist dies der Fall, gibt sie "null" aus.
3. Die Methode **persist()** von **DatabaseDao** wird aufgerufen, um die Informationen der Datenbank in der Datenbank Repository zu speichern.
4. Für jedes Policy Dokument wird die Methode **createArtefact()** aufgerufen, um ein Policy Artefakt anzulegen, wenn es noch Policy Dokumente im Request Message gibt.
5. Für jedes angelegte Policy Artefakt wird die Methode **createRelation()** von **RelationManager** aufgerufen, um eine Relation von der Datenbank zum Policy Artefakt zu erzeugen.

updateDatabase

Die Methode **updateDatabase ()** ist für die Implementierung der Operation **updateDatabase** zuständig.

Die Request Message hat den Messagetyp **UpdateDatabaseRequestMessage** und muss email, password, databaseId enthalten. Optional können databaseName, databaseDriver, databaseAddress, description in der Message verpackt werden.

Die Response Message hat den Messagetyp **UpdateDatabaseResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird ProviderId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find**() von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie “null” aus.
3. es wird überprüft, ob die ID des Anbieters von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebene providerId identisch sind. Wenn nein, gibt sie “null” aus.
4. Die Methode **update()** von **DatabaseDao** wird aufgerufen, um die Änderung des Datensatz in der Datenbank Repository zu speichern

addNewDatabaseUser

Die Methode **addNewDatebseUser()** ist für die Implementierung der Operation **addNewDatabaseUser** zuständig.

Die Request Message hat den Messagetyp **AddNewDatabaseUserRequestMessage** und muss email, password, databaseId, userName, userPassword enthalten. Optional können street, city, zipcode, country, telephone, website, userEmail in der Message verpackt werden.

Die Response Message hat den Messagetyp **AddNewDatabaseUserResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie “null” aus.
3. es wird überprüft, ob die providerId von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebene providerId identisch sind. Wenn nein, gibt sie “null” aus.
4. Die Methode **persist()** der Klasse **DbUserDao** wird aufgerufen, um das Datenbanknutzer Objekt in der Datenbank Repository zu speichern

updateDatabaseUser

Die Methode **updateDatabaseUser()** ist für die Implementierung der Operation **updateDatabaseUser** zuständig.

Die Request Message hat den Messagetyp **UpdateDatabaseUserRequestMessage** und muss email, password, databaseId, userId enthalten. Optional können street, city, zipcode, country, telephone, website, userEmail in der Message verpackt werden.

Die Response Message hat den Messagetyp **UpdateDatabaseUserResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **find()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie "null" aus.
3. Es wird überprüft, ob die providerId von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebene providerId identisch sind. Wenn Nein, gibt sie "null" aus.
4. Die Methode **find**() von **DbUserDao** wird aufgerufen, um zu überprüfen, ob den Datenbanknutzer existiert, wenn Nein, gibt sie "null" aus.
5. Es wird Kontrolliert, ob die databaseId des gefundenen Datenbanknutzers und die von der Request Message gelieferten databaseId identisch sind. Wenn nein, gibt sie "null" aus.
6. Die Methode **update()** von **DbUserDao** wird aufgerufen, um die Änderung des Datensatzes in der Datenbank Repository zu speichern

deleteDatabaseUser

Die Methode **deleteDatabaseUser()** ist für die Implementierung der Operation **deleteDatabaseUser** zuständig.

Das RequestMessage hat den Messagetyp **DeleteDatabaseUserRequestMessage** und muss email, password, databaseId, userId enthalten.

Die Response Message hat den Messagetyp **DeleteDatabaseUserResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird ProviderId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie “null” aus.
3. Es wird überprüft, ob die providerId von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebene providerId identisch sind. Wenn nein, gibt sie “null” aus.
4. Die Methode **find()** von **DbUserDao** wird aufgerufen, um zu überprüfen, ob der Datenbanknutzer existiert, wenn Nein, gibt sie “null” aus.
5. Es wird Kontrolliert, ob die databaseId des gefundenen Datenbanknutzers und die von der Request Message gelieferten databaseId identisch sind. Wenn Nein, gibt sie “null” aus.
6. Die Methode **delete()** von **DbUserDao** wird aufgerufen um den Datenbanknutzer zu löschen.

browseDatabaseUser

Die Methode **browseDatabaseUser()** ist für die Implementierung der Operation **browseDatabaseUser** zuständig.

Das RequestMessage hat den Messagetyp **browseDatabaseUserRequestMessage** und muss email, password, databaseId, Suchwert einer Suchmethode enthalten.

Die Response Message hat den Messagetyp **browseDatabaseUserResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird die providerId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find**() von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie “null” aus.
3. es wird überprüft, ob die ID des Anbieters von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebenen providerId identisch sind. Wenn nein, gibt sie “null” aus.
4. Es wird kontrolliert, welche Suchmethode verwendet wird. Ist es **findByUserName**:

   i. die Methode **findByUserName()** von **DbUserDAO** wird aufgerufen, um Datenbanknutzer durch den gegebenen Suchwert zu finden.
5. Sonst, ist es **findByEmail**:
   i. die Methode **findByEmail()** von **DbUserDAO** wird aufgerufen, um Datenbanknutzer durch den gegebenen Suchwert zu finden.
6. Sonst, ist es **findByAll**:
   i. die Methode **findByAll()** von **DbUserDAO** wird aufgerufen, um Datenbanknutzer durch den gegebenen Suchwert zu finden.
7. Sonst:
   i. die Methode **findAll()** von **DbUserDAO** wird aufgerufen, um alle Datenbanknutzer der Datenbank zu finden.

deleteDatabase

Die Methode **deleteDatabase()** ist für die Implementierung der Operation **deleteDatabase** zuständig.

Die Request Message hat den Messagetyp **DeleteDatabaseRequestMessage** und muss email, password, databaseId enthalten.

Die Response Message hat den Messagetyp **DeleteDatabaseResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen, ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie “null” aus.
2. Die Methode **find**() von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert, wenn nein, gibt sie “null” aus.
3. es wird überprüft, ob die ID des Anbieters von der gefundenen Datenbank und die durch die Authentifizierung zurückgegebenen providerId identisch sind. Wenn nein, gibt sie “null” aus.
4. die Methode **findByFrom**() von **RelationManager** wird aufgerufen, um zu überprüfen ob es noch Policy Artefakte gibt, die jeweils mit der Datenbank in Relationen stehen.

5. Für jedes gefundene Policy Artefakt wird die Methode **deleteRelation()** von **RelationManager** aufgerufen, um die Relation von der Datenbank zum Artefakt zu löschen. Dann wird die Methode **deleteArtefact()** aufgerufen, um das Artefakt löschen.
6. Die Methode **delete()** von **DatabaseDao** wird aufgerufen, um die Datenbank zu löschen.

valuateDatabase

Die Methode **valuateDatabase()** ist für die Implementierung der Operation **valuateDatabase** zuständig.

Die Request Message hat den Messagetyp **ValuateDatabaseRequestMessage** und muss databaseId und Bewertungsnote enthalten.

Die Response Message hat den Messagetyp **ValuateDatabaseResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **find()** von **DatabaseDao** wird aufgerufen, um zu überprüfen, ob die Datenbank existiert. Ist dies der Fall, wird der Datensatz abgeholt, sonst gibt sie "null" aus.
2. Die Bewertungsanzahl des Datensatzes erhöht sich um 1, die Note des Datensatzes wird auf (Bewertungsnote + Note der Datenbank)/Bewertungsanzahl gesetzt
3. Die Methode **update()** der Klasse **DatabaseDao** wird aufgerufen, um die Änderung des Datensatzes in der Datenbank Repository zu speichern

browseDatabase

Die Methode **browseDatabase()** ist für die Implementierung der Operation **browseDatabase** zuständig.

Die Request Message hat den Messagetyp **BrowseDatabaseRequestMessage** und muss einen Suchwert einer Suchmethode enthalten.

Die Response Message hat den Messagetyp **BrowseDatabaseResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Es wird kontrolliert, welche Suchmethode verwendet wird. Ist es **findByProviderName**:

i. Die Methode **findByProviderName()** von **ProviderDao** wird aufgerufen, um durch den gegebenen Suchwert Anbieter zu finden.

ii. Für jeden gefundenen Anbieter wird die Methode **findByDbProviderUid()** von **DatabaseDao** aufgerufen, um Datenbanken durch providerId zu finden.

2. Sonst, ist es **findByEmail**:
   i. die Methode **findByEmail ()** von **DatabaseDao** wird aufgerufen, um Datenbanken durch den gegebenen Suchwert zu finden.
3. Sonst, ist es **findByAll**:
   i. die Methode **findByAll()** von **DatabaseDAO** wird aufgerufen, um Datenbanken durch den gegebenen Suchwert zu finden.
4. Sonst, ist es **findByRating**
   i. die Methode **findByRating()** von **DatabaseDAO** wird aufgerufen, um alle Datenbanken zu finden, dessen Note größer gleich des gegebenen Wertes ist.

Die Suchmethode **findByPolicy** wird in der Arbeit nicht implementiert. Apache Neethi Framework unterstützt WS-Policy und die Operationen Normalisierung, Intersection und Merge. Durch die Operationen kann einfach überprüft werden, ob die Policy der Service Seite und die Policy der Client Seite potenziell kompatibel sind. Das Problem, wie man es überprüft, ob zum Beispiel beide Policies echt kompatibel sind, wird in dieser Arbeit nicht gelöst.

4.3.4 Implementierung der Operationen für Web Services

registerWebServices

Die Methode **registerWebServices()** ist für die Implementierung der Operation **registerWebServices** zuständig.

Die Request Message hat den Messagetyp **registerWebServices RequestMessage** und muss email, password, wsdlUri, WS-Bündel enthalten.

Die Response Message hat den Messagetyp **registerWebServicesResponseMessage**

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticate()** wird aufgerufen, um zu überprüfen, ob der Anbieter existiert. Ist dies der Fall, wird eine providerId geliefert, sonst gibt sie “null” aus.

2. Für das WSDL Dokument des WS-Bündels wird die Methode **createArtefact()**, um ein WSDL-Artefakt anzulegen.
3. Für jedes andere Dokument des WS-Bündels wird die Methode **createArtefact()** aufgerufen, um ein Artefakt zu erstellen, dann wird die Methode **createRelation()** von **RelationManager** aufgerufen, um die Relation vom WSDL-Artefakt zum Artefakt anzulegen.
4. Die Methode **extractServices()** von **WsdlParser** wird aufgerufen, um das WSDL-Artefakt zu parsen und Informationen von Services zu extrahieren.
5. Für jeden extrahierten Service wird ein Objekt der Klasse **ServiceMetadata** mit entsprechenden Informationen instanziiert und die Methode **persist()** von **ServiceMetadataDao** wird aufgerufen, um das Objekt in Datenbank Repository zu speichern.

updateWebServices

Die Methode **updateWebServices()** ist für die Implementierung der Operation **updateWebServices** zuständig.

Die Request Message hat den Messagetyp **updateWebServicesRequestMessage** und muss email, password, (wsdlId, wsdlDokument) | (serviceId, [wsdlUri], [serviceDescription]) enthalten.

Die Response Message hat den Messagetyp **updateWebServicesResponseMessage**

Die von der Methode aufgerufenen Methoden:

1. Es wird durch die Request Message kontrolliert, was geändert werden soll, das WSDL-Artefakt oder Metadaten von Services. Falls es WSDL-Artefakt ist:
   i. Die Methode **authenticateProvider()** wird aufgerufen. Ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie “null” aus.
   ii. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um zu überprüfen, ob das WSDL-Artefakt existiert. Ist dies nicht der Fall, gibt sie “null” aus.
   iii. Die Methode **findByWsdlUid()** von **ServiceMetadataDao** wird aufgerufen, um alle Datensätze der Service Metadata durch wsdlId zu finden

iv. es wird überprüft, ob die providerId von einem gefundenen Datensatz und die durch die Authentifizierung zurückgegebene providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.

v. Die Methode **delete()** vom **ServiceMetadataDao** wird aufgerufen, um alle gefundenen Datensätze zu löschen.

vi. Die Methode **getXMLElement()** wird aufgerufen, um ein **XMLElement** Objekt von der Request Message abzuholen.

vii. Die Methode **setXmlElement()** der Klasse **VersionDescriptor** wird aufgerufen, um den alten Inhalt durch den neuen zu ersetzen.

viii. Die Methode **update()** von **VersionDescriptorDao** wird aufgerufen, um die Änderung in der Datenbank zu speichern.

ix. Die Methode **extractServices()** von der Klasse **WsdlParser** wird aufgerufen, um das geänderte WSDL-Artefakt neu zu parsen und Informationen von Services zu extrahieren.

x. Für jeden extrahierten Service wird ein Objekt der Klasse **ServiceMetadata** mit entsprechenden Informationen instanziiert. Die Methode **persist()** von **ServiceMetadataDao** wird aufgerufen, um das Objekt in Datenbank Repository zu speichern.

2. Sonst:

i. Die Methode **authenticateProvider()** wird aufgerufen. Ist der Anbieter authentifiziert, wird providerId geliefert, sonst gibt sie "null" aus.

ii. Die Methode **find()** von **ServiceMetadataDao** wird aufgerufen, um zu überprüfen, ob Service existiert. Ist dies nicht der Fall, gibt sie "null" aus.

iii. Es wird überprüft, ob die Id des Anbieters vom gefundenen Service und die durch die Authentifizierung zurückgegebenen providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.

iv. Die Methode **update()** von **ServiceMetadataDao** wird aufgerufen, um die Änderung des Datensatzes in der Datenbank Repository zu speichern

deleteWebServices

Die Methode **deleteWebServices()** ist für die Implementierung der Operation **deleteWebServices** zuständig.

Die Request Message hat den Messagetyp **DeleteWebServices RequestMessage** und muss email, password, wsdlId enthalten.

Die Response Message hat den Messagetyp **DeleteWebServicesResponseMessage**

Die von der Methode aufgerufenen Methoden:

1. Die Methode **authenticateProvider()** wird aufgerufen. Ist der Anbieter authentifiziert, wird eine providerId geliefert, sonst gibt sie "null" aus.
2. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um zu überprüfen, ob das WSDL-Artefakt existiert. Ist dies nicht der Fall, gibt sie "null" aus.
3. Die Methode **findByWsdlUid()** von **ServiceMetadataDao** wird aufgerufen, um alle Datensätze der Service Metadata durch wsdlId zu finden
4. Es wird überprüft, ob die providerId von einem gefundenen Datensatz und die durch die Authentifizierung zurückgegebenen providerId identisch sind. Ist dies nicht der Fall, gibt sie "null" aus.
5. Die Methode **delete()** von **ServiceMetadataDao** wird aufgerufen, um alle gefundenen Datensätze zu löschen.
6. Die Methode **findByFrom()** von **RelationManager** wird aufgerufen, um alle Artefakte zu finden, die jeweils mit dem WSDL-Artefakt in Relation stehen.
7. Die Methode **delete()** vom **VersionDescriptorDao** wird aufgerufen, um alle gefundenen Artefakte und das WSDL-Artefakt zu löschen

valuateWebService

Die Methode **valuateDatabase()** ist für die Implementierung der Operation **valuateDatabase** zuständig.

Die Request Message hat den Messagetyp **ValuateWebServiceRequestMessage** und muss serviceId, Bewertungsnote enthalten.

Die Response Message hat den Messagetyp **ValuateWebServiceResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **find()** von **ServiceMetadataDao** wird aufgerufen, um zu überprüfen, ob der Service existiert. Ist dies der Fall, wird der Datensatz abgeholt, sonst gibt sie "null" aus.
2. Die Bewertungsanzahl des Datensatzes erhöht sich um 1, die Note des Datensatzes wird auf (Bewertungsnote + Note des Service) / Bewertungsanzahl gesetzt.
3. Die Methode **update()** der Klasse **ServiceMetadataDao** wird aufgerufen, um die Änderung des Datensatzes in der Datenbank Repository zu speichern

browseWebServices

Die Methode **browseWebServices()** ist für die Implementierung der Operation **browseWebServices** zuständig.

Die Request Message hat den Messagetyp **BrowseWebServicesRequestMessage** und muss einen Suchwert einer Suchmethode enthalten.

Die Response Message hat den Messagetyp **BrowseWebServicesResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Es wird kontrolliert, welche Suchmethode verwendet wird. Ist es **findByServiceName**:
   i. Die Methode **findByServiceName()** von **ServiceMetadataDao** wird aufgerufen, um Service Metadaten durch den gegebenen Suchwert zu finden.
2. Sonst, ist es **findByAll**:
   i. die Methode **findByAll()** von **ServiceMetadataDAO** wird aufgerufen, um Datenbanken durch den gegebenen Suchwert zu finden.
3. Sonst, ist es **findByRating**
   i. die Methode **findByRating()** von **ServiceMetadataDAO** wird aufgerufen, um alle Services durch den gegebenen Suchwert zu finden, dessen Noten größer gleich der gegebene Wert ist.

Die Suchmethode **findByPolicy** wird in der Arbeit nicht implementiert. Apache Neethi Framework unterstützt WS-Policy und die Operationen Normalisierung, Intersection und Merge. Durch die Operationen kann einfach überprüft werden, ob die Policy der Service Seite

und die Policy der Client Seite potenziell kompatibel sind. Das Problem, wie man es überprüft, ob zum Beispiel beide Policies echt kompatibel sind, wird in dieser Arbeit nicht gelöst.

retrieveWebService

Die Request Message hat den Messagetyp **RetrieveWebServiceRequestMessage** und muss eine serviceId enthalten.

Die Response Message hat den Messagetyp **RetrieveWebServiceResponseMessage**.

Die von der Methode aufgerufenen Methoden:

1. Die Methode **find()** von **ServiceMetadataDao** wird aufgerufen, um zu überprüfen, ob das Service existiert. Ist dies der Fall, wird der Datensatz geholt, sonst gibt sie "null" aus.
2. Die Methode **find()** von **VersionDescriptorDao** wird aufgerufen, um das WSDL-Artefakt zu holen, das das Service beschreibt.
3. Die Methode **extractServices()** der Klasse **WsdlParser** wird aufgerufen, um das WSDL-Artefakt zu parsen und Informationen von Services zu extrahieren.
4. Alle Informationen vom Service werden geholt, dessen Name mit dem Service Namen vom gefundenen Datensatz identisch sind.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept entwickelt um die Registrierung und das Suchen von Web Services und Datenbanken zu ermöglichen.

Zunächst wurden im Kapitel 2 die technischen Grundlagen zum Verständnis der Arbeit vorgestellt. Es wurden zuerst das Dreieck von SOA und die Definition vom Web Service kurz erläutert, dann wurde der Begriff WSDL ausführlich erklärt. WSDL beschreibt nämlich die funktionalen Eigenschaften von Web Service. Ein WSDL Dokument besteht aus zwei Teilen: einem wiederverwendbaren abstrakten Teil und einem konkreten Teil. Der abstrakte Teil von WSDL beschreibt welche Funktionalitäten ein Web Service anbietet. Der abstakte Teil enthält drei Elemente <types>, <message> und <portType>. Der konkrete Teil der WSDL beschreibt, wie und wo auf ein Web Service zugegriffen werden kann und enthält zwei Elemente <binding> und <service>.

SOAP ist eine Message Architektur und ein Message Verarbeitungsmodel. Eine SOAP-Message besteht aus einem Envelope Element, dieses Element wiederum besteht aus einem Body-Element und einen optionalen Header-Element. Im Body Element stehen die eigentlichen Nutzdaten, im Header Element können die Meta-Informationen, beispielsweise zum Routing, zur Verschlüsselung oder zur Transaktionsidentifizierung untergebracht werden. Das SOAP Message Verarbeitungsmodel spezifiziert wie ein Knoten ein SOAP Message verarbeitet, falls es die SOAP Message empfängt.

Die WS-Policy beschreibt die nicht-funktionalen Eigenschaften von Web Service. Eine Policy ist entweder in der Normalform oder in der Kompaktform. Die Normalform von Policies ist wichtig für die Operationen Intersection und Merge. Die Operation Intersection überprüft, ob zwei Policies potenziell kompatibel sind. Wenn mehrere Policies mit einem Policy Subject assoziiert sind, kann eine effektive Policy durch die Operation Merge berechnet werden.

Web Services Resource Framework (WSRF) spezifiziert einen Mechanismus, um die Beziehung zwischen Web Services und deren Status zu beschreiben.

Am Ende dieses Kapitels wurde noch das Software Fragmento vorgestellt. Fragmento ist zuständig für das Speichern, das Zugreifen und die Versionsverwaltung aller Artefakte, die mit einem Prozess relevant sind.

In Kapitel 3 wurde das Konzept der Entwicklung vom Registerservice detailliert spezifiziert. Das Registerservice ermöglicht es Web Services und Datenbanken zu registrieren, zu suchen und zu verwalten. Vor der Registrierung von einem Service bzw. einer Datenbank muss der Anbieter registriert sein. Ein Anbieter ist zuständig für die Verwaltung eigener Services und Datenbanken. Bei der Registrierung von Web Services soll der Anbieter ein WS-Bündel liefern. Ein WS-Bündel enthält genau ein WSDL-Dokument, beliebig viele BPEL Prozess Dokumente, beliebig viele Policy Dokumente und ein optionales XML Schema Dokument. Für das WS-Bündel wird entsprechend ein Artefakt-Bündel angelegt. Für jedes XML Dokument im WS-Bündel gibt es ein entsprechendes Artefakt im Artefakt-Bündel. Für jedes Non-WSDL-Artefakt im Artefakt-Bündel wird eine Relation vom WSDL-Artefakt zum Artefakt angelegt. Die Relationen ermöglichen es durch ein WSDL-Artefakt, alle anderen Artefakte im Artefakt-Bündel zu finden. Dann wird das WSDL-Artefakt geparst, um die Metadaten für die Services, die durch das WSDL Artefakt beschrieben wurden zu extrahieren und zusammen mit von dem Provider gelieferten Informationen in der Datenbank zu speichern. Die Metadaten ermöglichen es einfach festzustellen, wem die Artefakten gehören. Für die Registrierung einer Datenbank soll der Anbieter genügende Informationen, wie Name, Adresse, Treiber usw. liefern. Außerdem kann ein Anbieter zusätzlich noch das Policy Dokument mitgeben, die die Anforderung vom Anbieter für die Nutzung der Datenbank spezifiziert. Das Policy Dokument wird ebenfalls als Artefakt in die Datenbank Repository gespeichert. Für jedes Artefakt gibt es eine Relation von der Datenbank und zum Artefakt. Die Datenbanknutzer können erst nach der Registrierung vom Anbieter separat angelegt werden.

Nach der Registrierung können die Datenbanken, Services und Artefakte von jedem gesucht und nur vom Besitzer verwaltet werden. Das Registerservice bietet eine Reihe von Operationen an, um sie entsprechend zu operieren.

Im Kapitel 4 wurde die Implementierung vom Konzept behandelt, es wurden nur die wichtigen Klassen und Methoden vorgestellt. Die Klasse **RegisterServiceImp** ist die Implementierung vom Service und wurde zusammen mit den Methoden der Klasse detailliert spezifiziert.

Bei der Implementierung der Suchmethode **findByPolicy** der Operation **browseWebservices**

und der Operation **browsedatabase**, kommt das Problem auf wie man es überprüft, ob zum Beispiel beide Policies echt kompatibel sind oder nicht. Diese Frage wird in dieser Arbeit nicht gelöst.

Literaturverzeichnis

[1] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, 2005.

[2] Thilo Frotscher, Marc Teufel, Dapeng Wang. *Java Web Service mit Apache Axis2.* 2007 entwickler.press.

[3] Daniel Austin, Abbie Barbir, Christopher Ferris, Sharad Garg. *Web Services Architecture Requirements,* W3C Working Group Note 11 February 2004. http://www.w3.org/TR/wsa-reqs.

[4] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, *Web Services Description Language (WSDL) 1.1,* W3C Note 15 March 2001. http://www.w3.org/TR/wsdl.

[5] Prof. Dr. Frank Leymann, Institute of Architecture of Application Systems, Universität Stuttgart. Die Vorlesung Web Services, Kapitel 09 SOAP, WS2009/10.

[6] Prof. Dr. Frank Leymann, Institute of Architecture of Application Systems, Universität Stuttgart. Die Vorlesung Web Services, Kapitel 11 WSDL, WS2009/10.

[7] James Clark, Steve DeRose. *XML Path Language (XPath) Version 1.0 W3C,* Recommendation 16 November 1999. http://www.w3.org/TR/xpath.

[8] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh. *XPointer Framework*, W3C Recommendation 25 March 2003. http://www.w3.org/TR/xptr-framework/.

[9] David Orchard, Asir S Vedamuthu, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, Ümit Yalçinalp. *WSDL 1.1 Element Identifiers*, W3C Working Group Note 20 July 2007. http://www.w3.org/TR/wsdl11elementidentifiers.

[10] Nilo Mitra, Yves Lafon. *SOAP Version 1.2 Part 0: Primer (Second Edition)*,W3C Recommendation 27 April 2007. http://www.w3.org/TR/soap12-part0/.

[11] *Fragmento Dokumentation.* Lizenz: Apache 2 License. http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/downloads/Fragmento-documentation.pdf.

[12] Alexander Wiese. Diplomarbeit Nr. 2664, *Konzeption und Implementierung von WS-Policy- und WSRF-Erweiterungen für einen Open Source Enterprise Service Bus.* 23. Februar 2008.

[13] Zhilei Ma. Diplomarbeit Nr. 2405, *WS-Policy Editor - Ein Werkzeug für Editieren, Normalisierung, Verschmelzen und Scheiden von Web Service Policies*. 27. März 2006.

[14] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, Ümit Yalçinalp. W*eb Services Policy 1.5 – Framework* , W3C Recommendation 04 September 2007. http://www.w3.org/TR/ws-policy/.

[15] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, Ümit Yalçinalp. *Web Services Policy 1.5 – Attachment*, W3C Recommendation 04 September 2007. http://www.w3.org/TR/ws-policy-attach/.

[16] Giovanni Della-Libera, Martin Gudgin, Phillip Hallam-Baker, Maryann Hondo, Hans Granqvist, Chris Kaler, Hiroshi Maruyama, Michael McIntosh, Anthony Nadalin, Nataraj Nagaratnam, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Doug Walter, Riaz Zolfonoon. *Web Services Security Policy Language (WS-SecurityPolicy).* http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf.

[17] Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedhukin. *Web Services Resource 1.2 (WS-Resource)*, OASIS Standard, 1 April 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, 2006.

[18] Steve Graham, Jem Treadwell. *Web Services Resource Properties 1.2 (WS-ResourceProperties)*, OASIS Standard, 1 April 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[19] Latha Srinivasan, Tim Banks. *Web Services Resource Lifetime 1.2 (WS-ResourceLifetime)*, OASIS Standard, 1 April 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.

[20].Tom Maguire, David Snelling, Tim Banks. *Web Services Service Group 1.2 (WS-ServiceGroup)*, OASIS Standard, 1 April 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf.

[21]Lily Liu, Sam Meder. *Web Services Base Faults 1.2 (WS-BaseFaults)*, OASIS Standard, April 1 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.

[22]Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers. *UDDI Version 3.0.2*, UDDI Spec Technical Committee Draft, Dated 20041019. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

[22] Prof. Dr. Frank Leymann, Institute of Architecture of Application Systems, Universität Stuttgart. Die Vorlesung Web Services, Kapitel 11 Policies For Web Services, WS2009/10.

Listingsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BPEL . . . . . Business Process Execution Language
EPR . . . . . Endpoint Reference
GED . . . . . Global Element Definition
HTTP . . . . . Hypertext Transfer Protocol
JMS . . . . . Java Message Service
ME . . . . . Message Exchange
MEP . . . . . Message Exchange Pattern
SOA . . . . . Serviceorientierte Architektur
SOAP . . . . . .Simple Object Accesss Protocol
UDDI . . . . . Universal Description, Discovery and Integration
URI . . . . . Uniform Resource Identifier
WSDL . . . . . Web Services Description Language
WSRF . . . . . Web Services Resource Framework
XML . . . . . Extensible Markup Language

Namespaces

präfix	Namespace
soap	http://schemas.xmlsoap.org/soap/envelope/
wsdl	http://schemas.xmlsoap.org/wsdl/
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-secext-1.0.xsd
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy
sp	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702
wsa	http://www.w3.org/2005/08/addressing
wsrm	http://schemas.xmlsoap.org/ws/2005/02/rm/policy
wsrf-rp	http://docs.oasis-open.org/wsrf/rp-2
wsrf-rl	http://docs.oasis-open.org/wsrf/rl-2
xsd	http://www.w3.org/2000/10/XMLSchema

Erklärung

Hiermit versichere ich, diese Arbeit selbständigverfasst und nur die angegebenen Quellen benutzt zu haben.

Hao Jin