Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 151

Service Registry für die bedarfsabhängige Bereitstellung von Diensten

Alexander Blehm

Studiengang:	B.Sc. Softwaretechnik
Prüferin:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuerin:	Dipl.-Inf. Karolina Vukojevic-Haupt
begonnen am:	27.05.2014
beendet am:	07.11.2014
CR-Nummer:	H.3.5

Zusammenfassung

In einer geschäftlichen Umgebung werden Services ständig benötigt, daher muss man sich dort keine Gedanken um die Einsparung von Ressourcen machen. Anders ist es in einer wissenschaftlichen Umgebung. Hier werden Services seltener und unregelmäßiger genutzt. Es ist daher naheliegender Services nur dann bereitzustellen, wenn sie auch wirklich gebraucht werden. Sogenannte "On-Demand"-Services haben den Vorteil, dass sie bei Bedarf provisioniert und nach der Benutzung de-provisioniert werden können. Eine Service Registry erleichtert die Suche nach Services, indem sie Services an einer zentralen Stelle verwaltet. Sie macht Vorgänge wie Service Discovery (das Suchen nach funktional passenden Services) und Dynamic Binding (das automatische Zuweisen eines Aufrufes an einen passenden Service) überhaupt möglich. Es gibt keine Service Registry, die On-Demand Services unterstützt. Um eine Lösung für diese Problemstellung zu entwickeln, werden in dieser Arbeit folgende Ansätze diskutiert: das Anpassen existierender Lösungen oder die Entwicklung einer eigenen Lösung für eine Service Registry mit Unterstützung von On-Demand-Services. Das Resultat ist eine Service Registry mit einer Grundfunktionalität zum Registrieren von üblichen, provisionierten Services und einer Erweiterung für nicht-provisionierte On-Demand-Services.

Abstract

Services are used regularly and continuously in a business environment, therefore you do not have to worry about saving resources. This is different in a scientific environment, where services are used rarely and at irregular times. It is more suitable to provide services whenever they are needed. The advantage of so called "on-demand" services is the possibility of provisioning services whenever they are needed and deprovisioning them after usage. A service registry makes the search for services easier, by managing services in a central location. It makes operations, like service discovery (the search for services that fit functionally) and dynamic binding (the automatic assignment of a call to a service) possible. There exists no such service registry for on-demand services. To develop a solution for this problem, the following approaches are discussed in this thesis: adjusting existing solutions or developing a new solution for a service registry with support for on-demand services. The result is a service registry with the core functionality for typical, continuously provisioned services and an extension for not provisioned on-demand services.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit baut auf der Arbeit "Service Selection for On-demand Provisioned Services" auf mit dem Fokus auf die Service Registry [1]. Dort wird eine Architektur für das On-Demand-Provisionieren von Services vorgestellt und die Service Registry ist ein Teil davon. Die Aufgabe einer Service Registry in dieser Architektur ist es, Services zu verwalten, die von Service-Anbietern (Service Provider) registriert werden. Service-Konsumenten (Service Consumer) können die Service Registry dann durchsuchen, um für ihre Anforderungen einen passenden Service zu finden. Für die Suche bietet die Service Registry die Funktion der *Service Discovery* und *Service Selection* an. Service Discovery ist eine Funktion der Service Registry, mit der man als Service Konsument eine Menge von Services angeboten bekommt, die der funktionalen Anforderung des Service-Aufrufes entsprechen. Eine weitere Funktion der Service Registry ist die Service Selection, in der anhand von nicht-funktionalen Anforderungen geprüft wird, welcher Service aus dieser Menge für die Anforderung des Konsumenten geeignet ist. Beide Eigenschaften (funktionale und nicht-funktionale Anforderungen) muss die Service Registry verwalten können.

In der Arbeit "Service Selection for On-demand Provisioned Services" werden auch neue Typen von Services eingeführt. Üblich sind Services, die ständig installiert sind, eine feste Endpunktadresse haben und immer auf Anfragen hören (provisionierte Services). Solche Services können über Drittanbieter in der Service Registry angeboten werden und unterscheiden sich von On-demand-Services. On-demand-Services sind Services, die noch nicht provisioniert sind. Für solche Services wird in der Service Registry ein Service Angebot mit sogenanntem Service Package zur Verfügung gestellt. Das Service Package enthält alle Daten zur Provisionierung des Services. Ist ein solcher On-demand-Service provisioniert, spricht man von einer *Service-Instanz*. Ähnlich, wie ein provisionierter Service, haben die Service Instanzen eine eigene Endpunktadresse und hören auf Anfragen eines Service-Konsumenten, bis sie nicht mehr gebraucht werden. Nachdem die Nutzung einer Instanz abgeschlossen ist, wird sie wieder de-provisioniert. Folglich muss in der Service Registry sowohl das Angebot des ursprünglichen, nicht-provisionierten Services als auch die provisionierten Service-Instanzen verwaltet werden können.

Für das Registrieren von Services existieren bereits viele verschiedene Lösungen. Diese Service Registry-Lösungen bauen oft auf firmeneigenen "Service Orientierten Architekturen" (SOA) auf und haben ihre eigene Interpretation von einer idealen Service Registry, was dazu führt, dass die Lösungen sehr unterschiedlich ausfallen und auch zum Teil Terminologien verschieden interpretieren. Dies erschwert es, eine Übersicht über diese Lösungen zu behalten. Es bietet jedoch keine dieser Lösungen eine Service Registry, die On-demand-Services verwalten kann. Die Realisierung einer solchen Service Registry bildet den Hauptteil dieser Arbeit.

1.1 Motivation

Provisionierte Services werden in einer geschäftlichen Umgebung regelmäßig und ständig gebraucht. Es macht Sinn, dass diese Services immer laufen und ihre Funktionalität bereitstellen. In einer wissenschaftlichen Umgebung werden Services seltener und unregelmäßiger genutzt. Der Rechenaufwand eines einzigen Services kann aber hoch sein und begrenzte Rechenkapazität aufbrauchen.

Die Arbeit "Service Selection for On-demand Provisioned Services" erklärt, wie Services im Laufe eines Workflows benutzt werden. Der Workflow ist ein Plan, der eine bestimmte Aufgabe schrittweise löst und in einigen seiner Schritte auch Services aufruft und diese rechnen lässt. Die Gesamtzeit der Ausführung eines solchen Plans kann Wochen betragen. Wenn man davon ausgeht, dass während dieser Zeit nur provisionierte Services eingesetzt werden, dann müssten diese Services zur gesamten Ausführungszeit vorhanden sein. Jedoch werden diese Services, die sonst niemand anders benötigt, vielleicht nur wenige Stunden der Gesamtzeit beansprucht und es wird schnell klar, dass hier wegen provisionierten Services Rechenressourcen zu lange blockiert sind. Um Ressourcen einzusparen und freizugeben, wenn diese nicht benötigt werden, können On-demand-Services genutzt werden. Diese werden provisioniert, wenn sie gebraucht werden und de-provisioniert, wenn sie nicht mehr gebraucht werden.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist es die Service Registry, die in der zuvor eingeführt Architektur beschrieben wurde, zu realisieren. In der Abbildung 1 wird das Vorgehen in der Arbeit verdeutlicht.

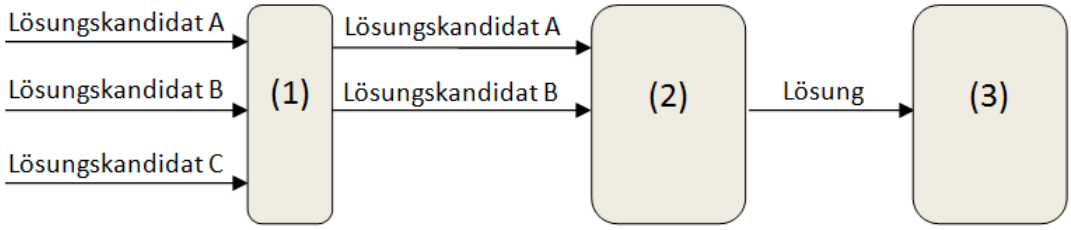

Abbildung 1: Vorgehen in der Arbeit

Es soll zuerst geprüft werden, ob es Lösungskandidaten gibt, die man möglicherweise für On-demand-Services erweitern könnte. Diese Lösungskandidaten werden im ersten, kleinen Schritt (1) gefiltert. Der Schritt ist kleiner abgebildet, da in dieser Arbeit darauf geachtet wurde, ob ein Lösungskandidat Open-Source ist oder nicht. Anschließend muss analysiert werden, welche Vor- und Nachteile ein existierender und anpassbarer Lösungskandidat mit sich bringt. Diese Analyse wird im größeren, zweiten Schritt (2) durchgeführt. Hier muss die Open-Source-Lösung und die Dokumentation des Lösungskandidats sehr genau betrachtet werden und jeder positive oder negative Aspekt festgehalten werden. Danach wird entschieden, ob eine existierender Lösungskandidat angepasst wird oder ob eine neue Lösung implementiert werden muss. Die Entscheidung ist die gewählte Lösung zwischen den Schritten (2) und (3), die im dritten Schritt (3) implementiert wird. Das Resultat ist

schließlich eine Service Registry, die zusammengefasst den folgenden Anforderungen gerecht werden muss:

- Das Verwalten von provisionierten Services
- Das Verwalten von nicht-provisionierten On-Demand-Services
- Das Verwalten von provisionierten Service-Instanzen jedes On-Demand-Services
- Das Ermöglichen einer Service Discovery für funktionale Anforderungen an einen Service
- Das Ermöglichen einer Service Selection für nicht-funktionale Anforderungen an einen Service

1.3 Aufbau der Arbeit

In Kapitel 2 werden einige Grundlagenbegriffe eingeführt, die im Laufe der Arbeit öfter vorkommen und für das Verständnis wichtig sind. Kapitel 3 gibt eine Übersicht über eine Architektur, welche die Forschungsgrundlage für die Service Registry ist. Dort soll klar werden, warum es die Service Registry überhaupt gibt und wie sie in ihrem Umfeld einzuordnen ist. Kapitel 4 analysiert bestehende Service Registry-Lösungen und prüft dabei, ob sie vorteilhaft für die Anforderungen der Service Registry angepasst werden können. Nach der Analyse wird auch entschieden, ob eine bestehende Lösung für die Implementierung angepasst wird, oder eine eigene Implementierung die Service Registry realisieren wird. Danach wird in Kapitel 5 die konzeptionelle Ausarbeitung der Service Registry vorgestellt und in Kapitel 6 folgt schließlich die eigentliche Beschreibung zur Implementierung. In Kapitel 7 werden die Kerninhalte der Arbeit zusammengefasst und in Kapitel 8 wird ein Ausblick für die Service Registry gegeben. Kapitel 9, 10 und 11 listen jeweils ein Abbildungs-, Tabellen- und Literaturverzeichnis auf.

2 Grundlagen

SOA bringt ein breites Spektrum an Begriffen mit sich. Um im Kontext dieser Arbeit mit klaren Begriffen arbeiten zu können, werden in diesem Kapitel Grundbegriffe eingeführt, erweitert und gegebenenfalls voneinander abgegrenzt. Viele Begriffe werden bewusst in der englischen Form in deutschen Sätzen verwendet, da der Gebrauch so allgemein üblich ist.

2.1 Services

Ein Service ist ein Softwareprogramm, das auf bestimmte Anfragen bestimmte Antworten generiert. Er hat eine bestimmte Funktionalität und ist damit ein Dienstleister. Ein Service hat eine einheitliche Schnittstelle, während die innere Implementierung des Services "versteckt" ist. Dadurch ist ein Service ein Element mit einer losen Kopplung. In unserem Kontext unterscheiden wir zwischen provisionierten Services und nicht provisionierten Services. Provisionierte Services sind Services, die immer aufgerufen werden können und eine feste Endpunktadresse (oder nur Endpunkt) haben. Nicht provisionierte Services bestehen aus einem Service Package von dem mehrere Service-Instanzen provisioniert werden können. Diese können dann wie ein provisionierter Service Anfragen entgegennehmen. Bei nicht-provisionierten Services wird außerdem unterschieden zwischen nicht provisionierten "Shared Services" ("geteilter Service") und nicht provisionierten "Dedicated Services" ("dedizierter Service"). Hierauf wird im Kapitel 5.1 noch genauer eingegangen.

2.2 Service Orientierte Architektur

Eine "Service Orientierte Architektur" (SOA) ist ein Architekturmuster, das auf Services basiert. Die SOA ist oft an komplexen Prozessen orientiert. Man kann sich hier Geschäftsprozesse vorstellen, wie z.B. das Bestellen eines Produktes bei einem Online-Händler. Wenn ein Kunde ein Produkt bestellt, wird die Bestellung an den Händler weitergegeben. Der Händler muss nun beispielsweise folgende Schritte ausführen:

1. Prüfen, ob das Produkt noch verfügbar ist
2. Produkt versenden
3. Rechnung für den Kauf ausstellen

Jeder dieser kleineren drei Schritte könnte von einem Service bearbeitet werden. Ein Service könnte in einer Datenbank prüfen, ob das Produkt noch vorhanden ist. Ein weiterer Service würde bei der Versandabteilung des Online-Händlers das Produkt zum Versand freigeben und ein dritter Service könnte den Rechnungspreis berechnen.

2.3 Binding

Das Binding beschreibt das technische Protokoll zur Kommunikation mit einem Service. Es ist eine Kombination aus einer Kodierungsart, wie beispielsweise SOAP [1], und einem Protokoll, wie beispielsweise HTTP [2]. In einer WSDL Datei beschreibt das Binding zusätzlich das Format, in dem Eingabe- und Ausgabeparameter von Operationen verschickt werden.

2.4 WSDL

WSDL steht für Web Service Description Language und ist eine einheitliche Beschreibungssprache für die Schnittstellen von Services. Sie wird auf Basis von XML beschrieben und hat den Vorteil, dass sie eine Kommunikation erlaubt, die unabhängig von Programmiersprache oder Protokoll der Kommunikationspartner ist. Die Abbildung 2 zeigt etwa, wie ein WSDL-Dokument aufgebaut ist.

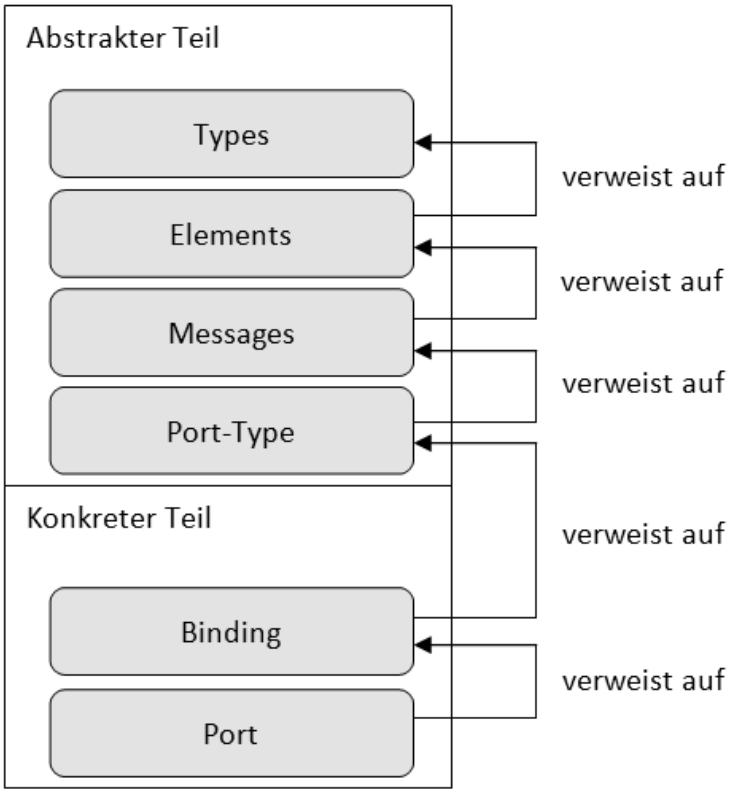

Abbildung 2: Aufbau einer WSDL

Eine WSDL-Datei ist schematisch in zwei Teile unterteilt: den abstrakten Beschreibungsteil und den konkreten Beschreibungsteil. Im abstrakten Teil werden zuerst Datentypen definiert, die entweder aus einzelnen primitiven Datentypen bestehen, oder zu komplexeren, zusammengesetzten Datentypen definiert werden können. Anschließend werden im abstrakten Teil Elemente beschrieben, die auf diese Datentypen verweisen. Im nächsten Schritt werden Nachrichten (Messages) definiert, die wiederum auf Elemente verweisen. Operationen im Port-Type verweisen dann auf Nachrichten. Diese Nachrichten können die Eingabe- und Ausgabeparameter der Funktionsaufrufe (oder Operationen) eines Services beschreiben.

[1] Simple Object Access Protocol, http://de.wikipedia.org/wiki/SOAP
[2] Hypertext Transfer Protocol, http://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Im konkreten Teil einer WSDL wird dann festgelegt, welches Binding benutzt wird. Das Binding verweist wiederum auf den Port-Type und muss für jede abstrakte Operation des Port-Types die konkreten Formate der Eingabe- und Ausgabeparameter bestimmen. Schließlich wird noch ein Port definiert. Der Port beinhaltet die Endpunktadresse und verweist auf das Binding [2].

2.5 Service Registry

Eine Service Registry ist ein Programm, dass relevante Daten zum Ansprechen von Services verwaltet. Es dient dazu, den Überblick über viele Services zu behalten und schnell bestimmte Services zu finden. Da die Funktion eines Services sich oft auf kleine Teilaufgaben beschränkt, werden für größere Prozesse viele Services kombiniert, um komplexere Aufgaben zu lösen. Eine Service Registry sollte daher Services auf irgendeine Weise klassifizieren können. Der Ansatz von UDDI (siehe Kapitel 2.6) sieht mehrere Möglichkeiten vor, einen Service zu klassifizieren: zum einen mit eindeutigen, industriellen Schlüsseln, zum anderen mit den optionalen tModels, die mit Hilfe einer WSDL-Datei eine Schnittstelle des Services beschreiben können. Gerade der zweite Ansatz mit der WSDL Schnittstellenbeschreibung spiegelt sich in vielen Service Registry-Lösungen wieder. In unserem Kontext wird die Service Registry diesen Ansatz mit einem "Service Interface" umsetzen. Dazu wird im Kapitel 5.1 genaueres beschrieben.

2.6 UDDI

UDDI steht für Universal Description, Discovery and Integration. Es ist eine Spezifikation, die eine Struktur für ein Verzeichnis beschreibt, in dem Services registriert werden können. UDDI beschreibt also effektiv eine Möglichkeit, Daten in einer Art Service Registry abzubilden. Die Abbildung 3 bildet die Grundelemente von UDDI ab.

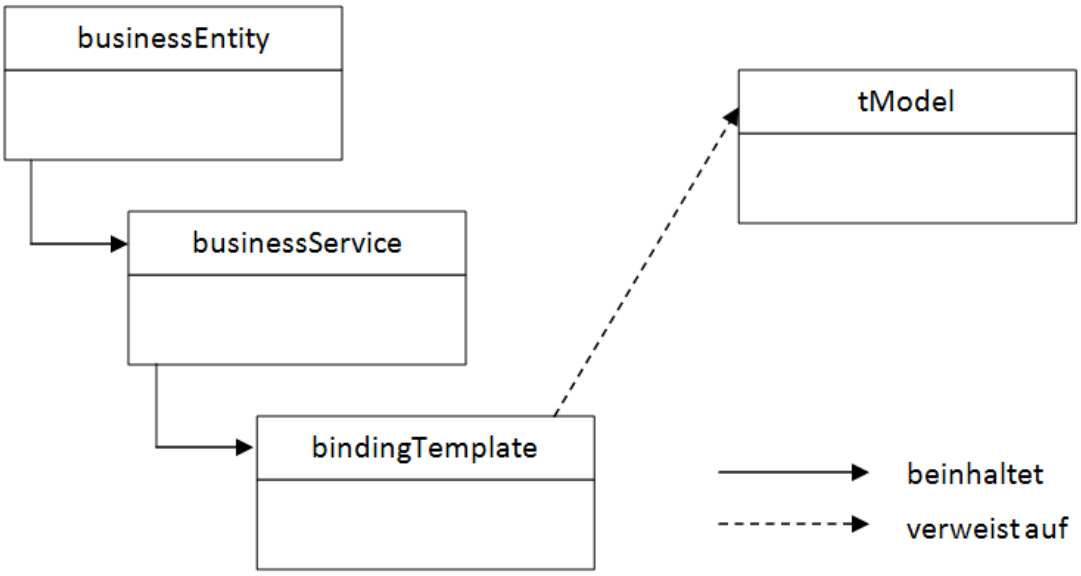

Abbildung 3: Grundelemente von UDDI

Das Wurzelelement befindet sich in der Abbildung oben links und heißt "businessEntity". In der businessEntity gibt es Informationen, die speziell für eine Firma ausgelegt sind, die einen oder mehrere Services bereitstellt. Darunter kann man Kontaktinformationen,

industrielle Kategorien und einen Schlüssel angeben, der die Firma eindeutig identifiziert. Die businessEntity beinhaltet mehrere "businessService"-Elemente. Ein businessService stellt einen Service dar. In diesem Element kann man eine textuelle Beschreibung für einen Service hinterlegen, mehrere Bindings für den Service bestimmen und den Service kategorisieren (mit den gleichen industriellen Kategorien, wie bei der businessEntity). Der businessService beinhaltet ein oder mehrere "bindingTemplate"-Elemente. Im bindingTemplate wird konkret beschrieben, über welches Protokoll und über welche Endpunktadresse der Service ansprechbar ist. Das bindingTemplate verweist auf ein tModel. Das tModel ist ein optionales Element, dass Schnittstellenbeschreibungen für einen Service enthält. Im tModel lässt sich ein Link zu einer WSDL-Datei angeben und auch hier kann das tModel (und damit die Schnittstelle) kategorisiert werden.

2.7 Policy

Eine Policy ist ganz abstrakt eine Regel, die auf etwas angewendet werden kann. In SOA werden Policies auf Services angewendet, um für Services nicht-funktionale Anforderungen zu formulieren oder Sicherheitsregeln hinzuzufügen. Eine Policy kann zum Beispiel die Anzahl der Aufrufe an einen Service für einen Service-Konsumenten täglich auf eine feste Zahl beschränken oder beschreiben, dass ein Service besonders schnell oder besonders sicher ist.

3 Bisherige Arbeiten

Es wird an dieser Stelle etwas genauer auf die Architektur und das Umfeld eingegangen, in der sich die Service Registry befindet. Eine komplette Beschreibung der Architektur findet sich in der Arbeit "Service Selection for On-demand Provisioned Services" [1]. Die Abbildung 4 veranschaulicht eine vereinfachte Nachbildung der Architektur.

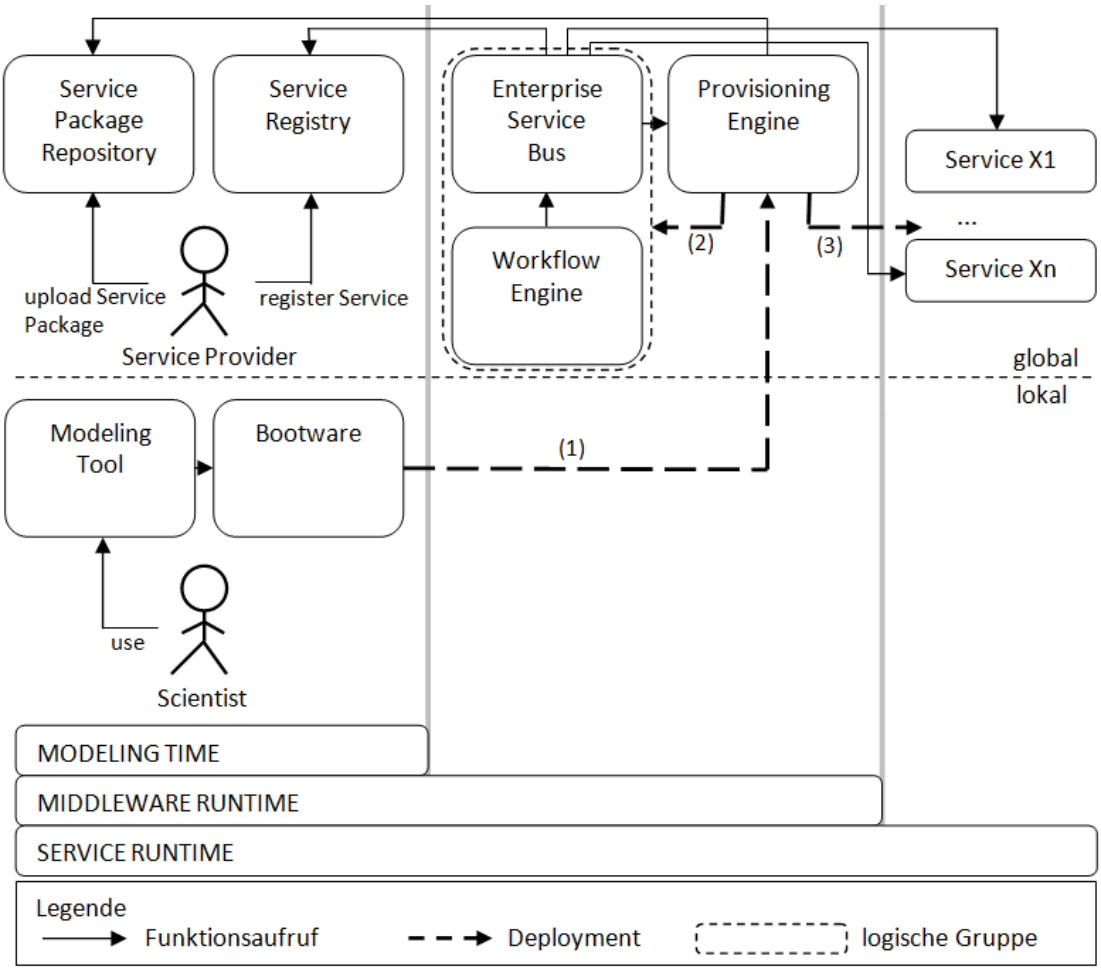

Abbildung 4: Architektur und Umfeld der Service Registry [1]

Im Folgenden werden in diesem Kapitel Referenzen auf Elemente in der Architektur kursiv geschrieben. Die Architektur ist in drei Phasen aufgeteilt. Diese sind im unteren Teil in der Abbildung zu sehen. Es gibt die *MODELING TIME* (Model-Phase), die *MIDDLEWARE RUNTIME* (Middleware-Phase) und die *SERVICE RUNTIME* (Service-Phase). Die Abbildung ist in eine lokale Seite und eine globale Seite aufgeteilt. Die Elemente auf lokaler Seite beschreiben jeden Wissenschaftler (Scientist), der auf seinem lokalen Computer arbeitet. Die Elemente in der globalen Seite befinden sich in einer Cloud[3]. Die globale Seite ist immer lauffähig und jeder kann auf ihre Elemente zugreifen.

In der Model-Phase werden in der Architektur auf lokaler Seite das *Modeling Tool* und die *Bootware* vom *Wissenschaftler* (Scientist) genutzt. Auf globaler Seite werden die *Service Package Repository* und die *Service Registry* ausgeführt. Ein Service Provider registriert Services in der *Service Registry* und lädt für On-demand-Services entsprechende Service

[3] Datenspeicherung in einer Cloud, http://de.wikipedia.org/wiki/Cloud_Computing

Packages in die *Service Package Repository* hoch. Die Service Packages sind Installationspakete, die alle Daten zum Provisionieren eines nicht-provisionierten On-demand-Services enthalten. Wichtig ist an dieser Stelle, dass das Registrieren eines Services in der *Service Registry* nur die Informationen zum Service speichert, wie eine Endpunktadresse, eine WSDL-Datei für die Schnittstelle und einige mehr. Die Besonderheit der *Service Registry* in dieser Architektur ist, dass sie für On-demand-Services einen Verweis auf ein Service Package im *Service Package Repository* speichert. Die *Service Package Repository* dagegen beinhaltet die eigentlichen Service Packages.

Auf lokaler Seite definiert der *Wissenschaftler* zunächst einen Workflow. Ein Workflow ist ein Plan, der schrittweise Operationen ausführt und an bestimmten Stellen Services aufruft. Dieser Workflow wird im *Modeling Tool* erstellt. Mit Hilfe der *Bootware* wird dann die *Provisioning Engine* aufgesetzt und es wird das Service Package weitergegeben, das die Workflow Middleware aufsetzen soll (1). Die *Provisioning Engine* ist eine komplexere Komponente, die in der Lage ist jede Art von Service zu Provisionieren und es wird an dieser Stelle nicht genauer auf sie eingegangen. Am Ende der Model-Phase provisioniert die *Provisioning Engine* die Workflow Middleware (2). Diese Workflow Middleware besteht aus dem *Enterprise Service Bus (ESB)* und der *Workflow Engine* und ist mit einem gestrichelten Kasten in der Abbildung markiert.

In der Middleware-Phase wird der Plan, der zuvor in der Model-Phase vom Wissenschaftler definiert wurde, auf der *Workflow Engine* ausgeführt. Der *ESB* bekommt Service-Aufrufe von der *Workflow Engine*, um die Schritte im Plan auszuführen. Für provisionierte Services leitet der *ESB* den Aufruf direkt an die Endpunktadresse des Services weiter. Für nicht-provisionierte Services muss der *ESB* zuerst mit der *Service Registry* und der *Provisioning Engine* kommunizieren. Im ersten Schritt bekommt der *ESB* von der *Service Registry* alle Informationen, die er zum provisionieren des nicht-provisionierten Services braucht, wie zum Beispiel den Verweis auf das Service Package. Dann ruft der *ESB* die *Provisioning Engine* auf, damit die *Provisioning Engine* den Service provisioniert (3). Dies ist in der Abbildung mit den Service-Instanzen *Service X1* bis *Service Xn* dargestellt. Das Provsionieren der On-demand-Services leitet die Service-Phase ein. Nachdem die Services provisioniert wurden, werden die Aufrufe an die Service-Instanzen weitergeleitet.

In der Service-Phase führen die Service-Instanzen die gewünschte Funktionalität nach dem Service-Aufruf aus. Sobald ein Service fertig ist mit der Ausführung, gibt er eine Antwort an den *ESB* zurück. Der *ESB* leitet die Antwort direkt an die *Workflow Engine* zurück. Wenn die Services mit der Ausführung fertig sind, wird die Unterscheidung zwischen nicht-provisionierten "Dedicated Services" und nicht-provisionierten "Shared Services" wichtig. Die nicht-provisionierten Dedicated Services sind nur für einen Aufruf vorgesehen. Das bedeutet, dass der *ESB* für die Dedicated Service-Instanz nach Bearbeitung des Service-Aufrufs sofort die *Provisioning Engine* aufruft, um die Instanz zu de-provisionieren. Anders ist es bei Shared Service-Instanzen. Hier muss der *ESB* zuerst prüfen, ob die Instanz noch Service-Aufrufe verarbeitet. Erst wenn der Service keine Service-Aufrufe mehr verarbeitet, kann dieser analog de-provisioniert werden. Wenn die *Workflow Engine* mit der Ausführung des Plans fertig ist, wird durch die *Bootware* das De-provisionieren der Workflow-Middleware initiiert. Zuletzt de-provisioniert die *Bootware* auch die *Provisioning Engine*.

4 Analyse bestehender Lösungen

Im Folgenden werden bestehende Service Registry-Lösungen betrachtet, die möglicherweise für nicht-provisionierte Services angepasst werden könnten. Für die bereits existierenden Lösungen wird vor allem auf Kriterien geachtet, die in der Tabelle 1 zusammengefasst sind. Anforderungen wie das Registrieren von provisionierten Services und nicht-provisionierten Services entfallen in der Kriterientabelle, da davon ausgegangen wird, dass einfache, provisionierte Services registriert werden können und nicht-provisionierte Services nicht. Weitere, in der Tabelle 1 nicht aufgeführte Eigenschaften der bestehenden Service Registry-Lösungen werden ebenfalls betrachtet, sofern sie als Besonderheit auffallen.

Kriterium	Beschreibung des Kriteriums
Service Discovery	Dies ist die Möglichkeit technische Schnittstellen zu definieren, um funktionale Anforderungen an einen Service zu stellen.
Service Selection	Dies ist die Möglichkeit Policies oder nicht-funktionale Anforderungen an einen Service zu stellen.
User Guide	Ein User Guide ist eine Dokumentation zur Benutzung der bestehenden Lösung. Durch sie lässt sich der praktische Nutzen der Lösung besser verstehen.
GUI	Das "Graphical User Interface", also die bestehende grafische Benutzeroberfläche sollte vorteilhaft größtenteils wiederverwendbar sein, damit man dort bei einer Anpassung nicht zu viel Arbeit investieren muss.
Development Guide	Ein Development Guide ist eine Dokumentation, die mindestens einen Einstieg in den existierenden Open-Source-Code gibt und die Struktur des Codes erklärt.
Aktive Community	Eine "aktive Community", die zeigt, dass das Projekt noch aktiv ist und bei der man sich bei Problemstellungen an Menschen aus der aktiven Community wenden kann.

Tabelle 1: Kriterien bei der Analyse

4.1 Apache jUDDI

Das Programm jUDDI[4] ist eine Open-Source Java-Implementierung der UDDI-Spezifikation von OASIS [3]. Alle Funktionen, die in der UDDI Spezifikation beschrieben sind, werden von jUDDI sowohl programmatisch als auch über die grafische Oberfläche angeboten. Die Lizenz für die Software ist die "Apache Software License, Version 2.0" [5].

Da jUDDI eine UDDI-Implementierung ist, stellt sich zuerst die Frage, in wie fern UDDI bereits für die Anforderungen an die Service Registry aus Kapitel 1.2 geeignet ist. In Kapitel 2.6 wurde bereits genannt, dass UDDI businessServices in den Grundelementen beschreibt, was Services mit einer Endpunktadresse sind. Damit wäre die Anforderung für provisionierte Services erfüllt. Nicht-provisionierte Services sieht die UDDI Spezifikation nicht vor, daher wird voraussichtlich auch jUDDI um nicht-provisionierte Services erweitert werden müssen. UDDI bietet eine Möglichkeit, Schnittstellen mit tModels für Services zu

[4] Java UDDI, http://juddi.apache.org/
[5] Open-Source-Lizenz von Apache, http://www.apache.org/licenses/LICENSE-2.0.html

beschreiben und diese zu kategorisieren. Die Struktur eines tModels wird in Abbildung 5 dargestellt.

```
<tModel tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">
    <name>Interface-Name</name>
    <description>Interface for Web Service</description>
    <overviewDoc>
        <description>The service's WSDL document</description>
        <overviewURL useType="text">
            <!-- WSDL ADDRESS -->
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference keyName="Category of Interface"
            keyValue="514191"
            tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"/>
    </categoryBag>
</tModel>
```

Abbildung 5: tModel für funktionale Anforderungen

Für ein tModel kann also eine WSDL-Datei zur Beschreibung der Schnittstelle referenziert werden (im Tag "overviewURL"). Des weiteren lässt sich im "categoryBag" eine eindeutige Kategorie für das Service-Interface beschreiben. Dies würde eine Service-Discovery für funktionale Anforderungen an einen Webservice ermöglichen. Mit tModels lassen sich in UDDI auch Policies beschreiben. Ein Beispiel dafür ist in Abbildung 6 dargestellt.

```
<tModel tModelKey="uddi:uddi.org:specification:v3_policy">
    <name>example-org:policy</name>
    <description>Policy Description service specification</description>
    <overviewDoc>
        <overviewURL useType="text">
            http://example.org/pubs/policy.htm#policyDesc
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference keyName="example-org:types:specification"
            keyValue="specification"
            tModelKey="uddi:example.org:categorization:types"/>
    </categoryBag>
</tModel>
```

Abbildung 6: tModel für nicht-funktionale Anforderungen

Statt einer WSDL für das Tag "overviewURL" wird ein Policy-Dokument angegeben. Bei der Kategorisierung müsste dann eine eindeutige nicht-funktionale Anforderung eingetragen werden. Eine genauere Beschreibung der Policies in UDDI findet sich in der UDDI-Spezifikation [3]. Damit wäre mit UDDI auch eine Service Selection für nicht-funktionale Anforderungen möglich. Da UDDI nicht-provisionierte Services nicht vorsieht und nach genauerer Betrachtung jUDDI nicht-provisionierte Services auch nicht unterstützt, muss jUDDI erweitert werden. Im Folgenden wird also anhand der Kriterien analysiert, ob sich jUDDI vorteilhaft anpassen lässt.

In der Dokumentation von jUDDI finden sich ein User-Guide und einige Demo-Videos, die die Benutzung von jUDDI erklären. Das erfüllt das Kriterium des User-Guides und hilft, das bestehende System zu verstehen. Jedoch hat jUDDI bei der Implementierung der grafischen Benutzeroberfläche deutliche Schwächen. Es ist ohne die Dokumentation nicht intuitiv klar, wie die grafische Benutzeroberfläche für Services zu benutzen ist. Die Usability der grafischen Oberfläche ist ebenfalls niedrig. Hier wurden einfache Usability-Praktiken, wie das Highlighting eines Menüs, in dem man sich befindet, nicht implementiert. Das bedeutet, dass Zeit in die Verbesserung oder ein neues Design der grafischen Oberfläche investiert werden muss und jUDDI keine fertige grafische Oberfläche mit sich bringt. Deshalb ist das Kriterium für die "GUI" nicht erfüllt. Auf der Homepage von jUDDI ist bei der Dokumentation ebenfalls einen Development Guide zu finden, jedoch erklärt dieser lediglich, wie man externe Clients entwickeln kann, um die UDDI Schnittstellen zu benutzen, die bereits existieren. Der Development Guide gibt keinen Code-Einstieg und beschreibt nicht die existierende Architektur des Systems und des Codes. Damit ist das Kriterium für den Development Guide ebenfalls nicht erfüllt. Apache jUDDI hat kein eigenes Forum. Fragen zu jUDDI finden sich vereinzelt auf verschiedenen anderen Foren, aber es scheint keine zentrale Stelle dafür zu geben. Es wird deutlich, dass die Community sich nicht aktiv mit jUDDI beschäftigt. Ein weiterer Indikator dafür ist eine Art Wiki-Lexikon für jUDDI, das nur aus kaputten Links besteht. Damit ist das Kriterium für eine aktive Community ebenfalls nicht erfüllt.

Als Besonderheit ergänzend zu den definierten Kriterien ist bei der Analyse von jUDDI noch aufgefallen, dass es einen Blog gibt, das von einem jUDDI Entwickler geführt wird. Dort findet sich ein Architekturbild der Komponenten von jUDDI. Es gibt auf der jUDDI Homepage auch die Möglichkeit den Code anzusehen, ohne ihn zuvor selber bauen zu müssen [6].

Zusammengefasst wird in der Tabelle 2 gezeigt, dass nicht alle Kriterien zur Arbeit mit jUDDI erfüllt sind. Das Ergebnis der Analyse zeigt mit der Oberfläche und dem Mangel an Entwicklerunterstützung, dass eine Arbeit mit jUDDI sehr zeitkritisch sein könnte. Im Rahmen dieser Bachelorarbeit wird jUDDI daher nicht als existierende Lösung angepasst.

Kriterium	**erfüllt**	**Begründung**
Service Discovery	ja	Dies ist mit tModels möglich.
Service Selection	ja	Dies ist mit tModels möglich.
User Guide	ja	Verständlicher User Guide und Demo-Videos machen die Benutzung von jUDDI klar.
GUI	nein	Die GUI ist nicht ohne Dokumentation benutzbar und hat eine niedrige Usability.
Development Guide	nein	Kein Development Guide für den Code an sich, nur für externe Clients.
Aktive Community	nein	Es gibt keine aktive Community und kein jUDDI Forum.

Tabelle 2: Ergebnisse der Analyse von jUDDI

[6] eine externe Referenz für den jUDDI Code, http://juddi.apache.org/xref/

4.2 WSO2 Governance Registry

Die WSO2 Governance Registry (oder kurz WSO2 GREG) ist eine Open-Source-Lösung von WSO2, die unter anderem eine Service Registry beinhaltet [ref-WSO2-GREG-hp]. Anders als bei jUDDI ist WSO2 GREG keine UDDI-Implementierung. Sie ist jedoch genauso wie jUDDI unter der Apache License 2.0 verfügbar. WSO2 GREG interpretiert Inhalte, wie z.B. Services, auf eine eigene Weise mit Artefakten. Die Artefakte in WSO2 GREG sind durch XML-Strukturen beschrieben. Die Dokumentation beschreibt, wie durch bestimmte XML-Schnipsel ein Artefakt um Textfelder, oder Dropdown-Felder erweitert werden kann. Bearbeitet man ein Artefakt mit der korrekten XML-Syntax, so generiert sich auch eine entsprechende HTML Oberfläche dazu [7].

In WSO2 GREG gibt es die Möglichkeit, Links zu WSDL-Dateien für Services anzugeben, es ist aber nicht vorgesehen, eine Schnittstellenbeschreibung für mehrere Services zu verwenden. Es ist in WSO2 GREG zwar auch möglich ein Artefakt für WSDL-Dateien separat anzugeben, jedoch können diese nicht für ein Service-Artefakt genutzt werden. Damit ist die Service Discovery vorerst nicht möglich und es müssten hier Anpassungen im Code vorgenommen werden. Auch die Service Selection für nicht-funktionale Anforderungen ist in WSO2 nicht möglich. Ähnlich wie bei WSDL-Artefakten können separat Policy-Artefakte angelegt, aber nicht für Service-Artefakte benutzt werden.

Auf der Homepage von WSO2 GREG gibt es einen Link zur Dokumentation. Dort finden sich teilweise sehr ausführliche Beschreibungen zur Benutzung von WSO2 GREG. Damit ist das Kriterium für einen User Guide erfüllt. Die bestehende Oberfläche von WSO2 GREG ist sehr intuitiv bedienbar. Es ist sofort klar, wo ein Service angelegt werden kann und es ist verständlich welche Daten für den Service angelegt werden können und wozu. Damit ist das Kriterium für die GUI erfüllt, denn die Oberfläche lässt sich so wiederverwenden und es müssten eventuell nur kleine Anpassungen gemacht werden. Beschreibungen für Entwickler finden sich in der Dokumentation von WSO2 GREG keine. Es gibt nur eine kleine Anleitung, die beschreibt, wie man den Code der Anwendung bauen kann. Jedoch scheitert der "Build"-Prozess nach dieser Anleitung. Folglich ist das Kriterium für einen Development Guide nicht erfüllt. Support für Entwickler ist bei WSO2 GREG nur schwer zu erreichen. Das offizielle Forum zu WSO2 GREG ist seit 2012 nicht mehr benutzt worden. Für weitere Fragen wird man bei WSO2 an Stackoverflow[8] verwiesen. Eine Frage zum Code-Einstieg und dem fehlerhaften Build wurde aber schließlich auch auf Stackoverflow nicht beantwortet. Damit ist auch das Kriterium für die aktive Community nicht erfüllt in WSO2 GREG.

Die Ergebnisse der Analyse werden in der Tabelle 3 zusammengefasst. Auch hier sind nicht alle Kriterien erfüllt. Die Architektur von WSO2 GREG ist nicht klar und damit ist auch hier die Arbeit im Rahmen der Bachelorarbeit zu zeitkritisch. WSO2 GREG wird daher ebenfalls nicht als bestehende Lösung für die Service Registry angepasst.

[7] Configurable Governance Artifacts sind generische Artefakte in WSO2 GREG, https://docs.wso2.com/pages/viewpage.action?pageId=22185121

[8] Stackoverflow ist eine Frageplattform für Entwickler, http://stackoverflow.com/

Kriterium	erfüllt	Begründung
Service Discovery	nein	Man kann keine Schnittstellenbeschreibung mehrmals für Services verwenden.
Service Selection	nein	Man kann keine Policies für Services verwenden.
User Guide	ja	Verständlicher User Guide und Demo-Videos machen die Benutzung von jUDDI klar.
GUI	ja	Die GUI ist intuitiv bedienbar und ist größtenteils wiederverwendbar.
Development Guide	nein	Kein Development Guide für den Code, Dokumentation ist ein reiner User-Guide.
Aktive Community	nein	Offizielles Forum wird nicht mehr benutzt, Stackoverflow hat sich nicht als gute Alternative herausgestellt.

Tabelle 3: Ergebnisse der Analyse von WSO2 GREG

4.3 Oracle Service Registry

Die Oracle Service Registry, eine nicht Open-Source-Lösung, wurde ebenfalls betrachtet. Jedoch wurde hier schnell klar, dass sie die Service Registry-Anforderungen dieser Arbeit nicht erfüllt. Sie hat keine Unterstützung für nicht-provisionierte Services. Da die Oracle Service Registry nicht Open-Source ist und die Entwicklerhandbücher auch keine passende Konfigurationsmöglichkeit für die Anforderungen aus Kapitel 1.2 beschrieben haben wird die Oracle Service Registry auch nicht als mögliche Lösung betrachtet.

5 Service Registry

Die Service Registry für On-Demand-Services ist für ein bestimmtes Umfeld vorgesehen, in der es unter anderem zusammen mit einem Enterprise Service Bus (ESB) kommuniziert. Aus dem Datenmodell in Kapitel 5.1 und den Anforderungen vom ESB ergibt sich eine Schnittstelle, die die Service Registry nach außen hin anbieten muss. Die Methoden werden in einem Use-Case-Diagramm in Kapitel 5.2 vorgestellt und in Kapitel 6.3 später konkreter definiert.

5.1 Datenmodell

Das Datenmodell der Service Registry ist in Abbildung 7 mit Hilfe eines Entity-Relationship-Models abgebildet.

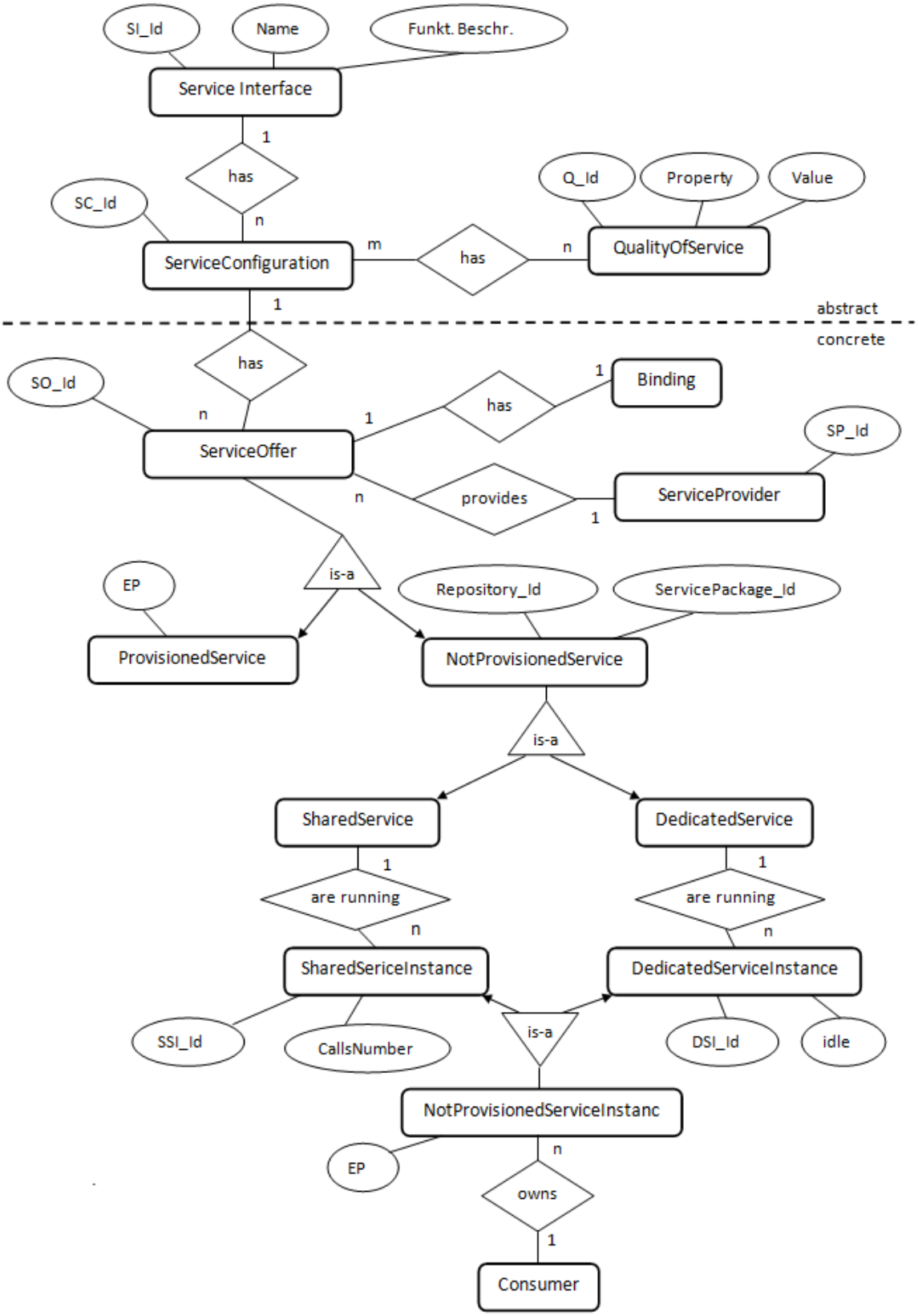

Abbildung 7: Datenmodell der Service Registry

Das Datenmodell teilt die Entitäten in abstrakte und konkrete Entitäten auf. Die abstrakten Entitäten sollen funktionale und nicht-funktionale Eigenschaften von Services definieren. Diese sollen Services kategorisieren und von mehreren Services benutzt werden, welche die gleichen funktionalen und nicht-funktionalen Eigenschaften haben. Der konkrete Teil beschreibt schließlich die eigentlichen Services. Im Folgenden wird jede Entität des Datenmodells genauer erläutert:

"Service Interface"

Das Service Interface soll die Schnittstelle zu einem Service definieren und damit eine funktionale Anforderung beschreiben. Dies wird für Services in der Service Registry eine Service Discovery möglich machen. Es hat einen eindeutigen PortType-Namen (Name) der eine eindeutige Schnittstelle eines Webservices beschreibt. Die funktionale Beschreibung besteht aus einer Adresse zu einer WSDL-Datei. Für diese WSDL ist vorgesehen, dass sie zumindest den abstrakten Teil definieren muss. Sie muss nicht zwingend den konkreten Teil (mit Binding Information und Endpunktadresse) definieren, da dies nicht relevant für eine Schnittstellenbeschreibung ist. Optional soll die funktionale Schnittstelle auch mit einer kleinen textuellen Beschreibung ergänzt werden können.

"QualityOfService"

Ein QualityOfService soll nicht-funktionale Anforderungen beschreiben können und damit eine Service Selection für Services in der Service Registry ermöglichen. Es werden vorerst Paare von nicht-funktionalen Eigenschaften (Property) mit Werten (Values) gefüllt. Die Ausarbeitung eines vollständigen Models für die nicht-funktionalen Anforderungen ist nicht Teil dieser Arbeit.

"ServiceConfiguration"

Eine ServiceConfiguration stellt Kombinationen funktionaler (Service Interface) und nicht-funktionaler (QualityOfService) Eigenschaften dar. Hiermit soll für Services eine passende Kombination abgebildet werden können. Jede ServiceConfiguration hat immer genau eine funktionale Beschreibung (ServiceInterface), kann aber mehrere nicht-funktionale Beschreibungen (QualityOfService) haben. Umgekehrt können Service Interfaces für mehrere ServiceConfigurations benutzt werden. Ebenso können einzelne QualityOfService für mehrere ServiceConfigurations benutzt werden.

"ServiceOffer"

Hier wird ein Service als "Angebot" (ServiceOffer) dargestellt, das immer von einem bestimmten Service Anbieter (ServiceProvider) kommt. Es wird bewusst Angebot genannt, um zu verdeutlichen, dass es sich hier nur um Informationen eines Services handelt und nicht um den Service selbst. Für jedes Angebot muss genau ein Binding festgelegt werden. Falls der gleiche Service mit einem anderen Binding registriert werden muss, so muss dafür ein neues ServiceOffer mit diesem Binding erstellt werden. Das Angebot steht für einen Service, der eine bestimmte ServiceConfiguration implementiert. Die gleiche ServiceConfiguration kann von mehreren Angeboten genutzt werden. Ein Angebot

beschreibt alle allgemeinen Eigenschaften eines konkreten Service-Angebots, wie einen Namen, eine vollständige WSDL-Datei, etc.

"Binding"

Das Binding beschreibt das technische Kommunikationsprotokoll, mit dem der Service ansprechbar ist. Es soll aus einer Kombination von Codiertyp und Netzwerkprotokoll bestehen. Ein Beispiel für so eine Kombination ist SOAP / HTTP. Jedes Binding kann für genau ein Service-Angebot genutzt werden.

"ServiceProvider"

Der Service-Anbieter (ServiceProvider) kann mehrere Angebote von Services anbieten. Er sollte eindeutig identifizierbar sein, um zu unterscheiden welches Angebot von welchem Anbieter stammt. Ein Service-Anbieter sollte auch nur in der Lage sein, seine eigenen Angebote zu bearbeiten, und diese auch wieder aus der Service Registry zu löschen.

"ProvisionedService"

Ein ProvisionedService beschreibt einen provisionierten Service. So ein Service wird in der Regel von Drittanbietern registriert. Er besitzt einen Endpunkt, der immer auf Anfragen hört, und erbt alle Eigenschaften des allgemeinen Service Angebots (ServiceOffer).

"NotProvisionedService"

Ein NotProvisionedService fasst alle Eigenschaften nicht-provisionierter Service-Angebote zusammen. In dem Fall kann hier eindeutig bestimmt werden, welches Service Package benutzt wird (ServicePackage_Id), um den Service zu installieren und welcher Typ von Paket das ist (Repository_Id). Es ist eine Zwischenabstraktion und erbt alle Eigenschaften des Service-Angebots (ServiceOffer).

"SharedService"

Ein SharedService ist ein konkretes Service-Angebot. Aus einem Shared-Service-Angebot lassen sich Instanzen von Services installieren, die Anfragen eines gleichen Benutzers durch die gleiche Instanz bearbeiten lassen können. Es erbt alle Eigenschaften eines nicht-provisionierten Service-Angebots (NotProvisionedService) und hat eine Liste von SharedServiceInstance. Dies sind die Service-Instanzen, die entstehen, wenn der Service provisioniert wird.

"DedicatedService"

Aus einem Dedicated-Service-Angebot lassen sich Instanzen von Services installieren, die jeweils nur eine Anfrage eines Benutzers bearbeiten können. Für jede weitere Anfrage des gleichen Benutzers muss eine neue Instanz dieses Angebots installiert werden. Es erbt alle Eigenschaften eines nicht-provisionierten Services-Angebots und hat eine Liste von DedicatedServiceInstance. Dies sind die Service-Instanzen, die entstehen, wenn der Service provisioniert wird.

"NotProvisionedServiceInstance"

NotProvisionedServiceInstance ist eine Instanz von nicht-provisionieren Services. Folglich muss sie eine Endpunktadresse haben, über die sie ansprechbar ist. Die NotProvisionedServiceInstance hat diesen Endpunkt, weil sie im Folgenden für die SharedServiceInstance und DedicatedServiceInstance verwendet wird.

"SharedServiceInstance"

Eine SharedServiceInstance ist eine Service-Instanz eines Shared-Service-Angebots. Sie erbt die Endpunktadresse einer allgemeinen "NotProvisionedServiceInstance" und ist dafür vorgesehen mehrere Anfragen eines Benutzers gleichzeitig verarbeiten zu können. Diese Anfragen (Calls) sollen in einer Zählvariable (Callsnumber) gezählt werden. Fällt die Callsnumber auf 0 zurück nachdem alle Anfragen abgearbeitet wurden, so kann die Instanz wieder de-provisioniert werden.

"DedicatedServiceInstance"

Eine DedicatedServiceInstance ist eine Instanz eines Dedicated-Service-Angebots. Sie erbt die Endpunktadresse einer allgemeinen "NotProvisionedServiceInstance" und kann immer nur eine Anfrage des gleichen Benutzers bearbeiten. Es gibt hier also nur den Zustand der Bearbeitung einer Anfrage oder den Zustand, in dem die Instanz keine Anfrage mehr bearbeitet und "abwesend" (idle) ist. Nachdem die eine Anfrage bearbeitet ist, für welche die Dedicated-Service-Instanz vorgesehen war, kann die Instanz wieder de-provisioniert werden.

"Consumer"

Der Consumer (Konsument oder Nutzer) benutzt nicht-provisionierte Service-Instanzen. Er kann mehrere Instanzen benutzen und muss bei den Instanzen als "Consumer" eingetragen werden, damit Anfragen korrekt umgeleitet werden. Dies ist deshalb wichtig, weil für Anfragen vom Consumer unterschieden werden muss, ob sie an ein Shared-Service-Angebot oder ein Dedicated-Service-Angebot gerichtet sind. Falls sie an ein Shared-Service-Angebot gerichtet sind, muss zuerst geprüft werden, ob es nicht schon eine provisionierte Shared-Service-Instanz für den Benutzer gibt. Falls es sie gibt, muss die Anfrage an diese Instanz umgeleitet werden und die Callsnumber der Instanz erhöht werden. Falls es sie nicht gibt, muss eine neue Shared-Service-Instanz provisioniert werden. Bei einer Anfrage an eine Dedicated-Service-Instanz wird einfach ohne weitere Prüfung eine neue Instanz provisioniert.

5.2 Use-Cases

In der Abbildung 8 werden die Use-Cases der Service Registry vorgestellt, die sich durch die zwei möglichen Benutzer der Service Registry ergeben: den menschlichen Benutzer und den Enterprise Service Bus (ESB).

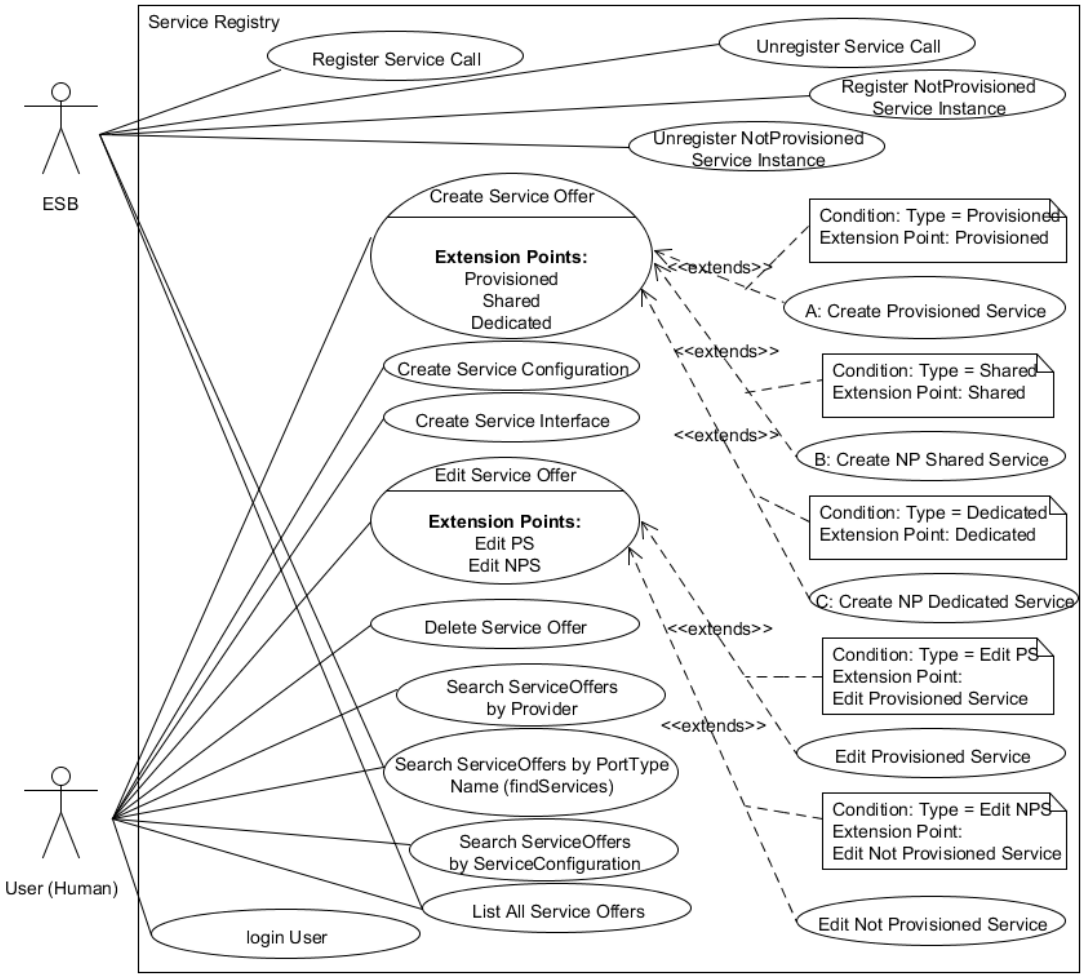

Abbildung 8: Use-Cases der Service Registry

Die folgenden Kapitel beschreiben Methoden, welche die Service Registry zur Verfügung stellen muss. Es gibt Methoden, die entweder ausschließlich von einem menschlichen Benutzer, ausschließlich vom Enterprise Service Bus (ESB) oder von beiden genutzt werden sollen.

5.2.1 Use-Cases vom Benutzer

Create Service Interface

Der Benutzer erstellt ein Service Interface, das später beim Erstellen von Service-Configurations angegeben werden kann. Das Service Interface soll eine Service Discovery ermöglichen und beschreibt funktionale Eigenschaften eines Services. Eine Service Configuration muss beim Erstellen immer ein bereits existierendes Service Interface angeben.

Create Service Configuration

Der Benutzer erstellt durch Angabe eines existierenden Service-Interfaces und zusätzlicher Angabe von Qualitätspaaren eine Service-Configuration. Es ist keine separate Methode für das Erstellen von nicht-funktionalen Eigenschaften (QualityOfService) eines Services vorgesehen, da sie implizit in der Create Service Configuration Methode erstellt werden. Die nicht-funktionalen Eigenschaften eines Services ermöglichen eine Service Selection. Die Service-Configuration soll helfen, Service-Angebote in der Service Registry zu kategorisieren.

Create Service Offer

Der Use-Case Create Service Offer ist die abstrakte Beschreibung zur Erstellung eines Service-Angebots. Der Use-Case muss in einem der drei folgenden Fälle konkretisiert werden:

1. Create Provisioned Service
2. Create NP Shared Service
3. Create NP Dedicated Service

Alle diese Use-Cases müssen beim Erstellen eine zuvor erstellte Service-Configuration referenzieren, damit klar ist, mit welcher Schnittstelle der Service anzusprechen ist.

Create Provisioned Service

Der Benutzer erstellt ein konkretes Service-Angebot für einen provisionierten Service. Dieser Use-Case ist für Drittanbieter von Services vorgesehen, die ihren Service extern betreiben und hier lediglich die Schnittstelle und eine Endpunktadresse für den Service angeben.

Create NP Shared Service

Der Benutzer erstellt ein konkretes Service-Angebot für einen nicht-provisionierten Shared-Service. Dieses Angebot muss angeben, mit welchem Service Package es provisioniert werden kann. Aus diesem Angebot sollen Shared-Service-Instanzen erstellt werden können.

Create NP Dedicated Service

Der Benutzer erstellt ein konkretes Service-Angebot für einen nicht-provisionierten Dedicated-Service. Dieses Angebot muss angeben, mit welchem Service Package es provisioniert werden kann. Aus diesem Angebot sollen Dedicated-Service-Instanzen erstellt werden können.

Edit Service Offer

Der Benutzer kann ein Service-Angebot bearbeiten. Dies ist ein abstrakter Use-Case, der in einem der folgenden Use-Cases konkretisiert werden muss:

1. Edit Provisioned Service
2. Edit Not Provisioned Service

Wichtig ist, dass der Benutzer das Service-Angebot nur bearbeiten kann, wenn er selbst als Service-Anbieter dieses Angebot in der Service Registry registriert hat.

Edit Provisioned Service

Der Benutzer bearbeitet ein existierendes Service-Angebot eines provisionierten Services, das er selbst registriert hat. Dieser Use-Case sollte nur vom Service-Anbieter des Service-Angebots genutzt werden können.

Edit Not Provisioned Service

Der Benutzer bearbeitet ein existierendes Service-Angebot eines nicht-provisionierten Services, das er selbst registriert hat. Dieser Use-Case sollte nur vom Service-Anbieter des Service-Angebots genutzt werden können.

Delete Service Offer

Der Benutzer löscht ein Service-Angebot aus der Service Registry. Dies kann der Benutzer nur für Service-Angebote anwenden, die er selber als Service-Anbieter in der Service Registry registriert hat.

Register User

"Register User" registriert einen Benutzer im System der Service Registry. Dieser Benutzer kann sich dann im System einloggen. Als registrierter Benutzer kann das System den Benutzer beim Registrieren von Service-Angeboten als Service-Anbieter vermerken. Der Benutzer kann im System nun auch als Service-Consumer für nicht-provisionierte Service-Instanzen vom ESB vermerkt eingetragen werden.

Login User

Dieser Use-Case Authentifiziert den Benutzer im System. Das System erkennt den Benutzer und kann ihn als Service-Anbieter beim Registrieren von Service-Angeboten vermerken.

Search Service Offers by Provider

Der Benutzer sucht in der Service Registry nach Service-Offers, die von einem bestimmten Service-Anbieter registriert wurden. Der Name des Anbieters muss genau den gesuchten Namen haben.

Search Service Offers by ServiceConfiguration

Der Benutzer sucht in der Service Registry nach Service Offers, die eine bestimmte Service Configuration implementieren. Diese Service-Configuration muss genau den gesuchten Namen haben.

5.2.2 Use-Cases vom ESB

Register Not Provisioned Service Instance

Dieser Use-Case registriert eine nicht-provisionierte Instanz eines nicht-provisionierten Service-Angebots. Diese Instanz kann entweder die eines Shared-Service-Offers oder eines Dedicated-Service-Offers sein. Durch diese Registrierung ist auch klar, dass die Anzahl der Aufrufe an die jeweilige nicht-provisionierte Instanz genau 1 ist.

Unregister Not Provisioned Service Instance

Dieser Use-Case löscht eine nicht-provisionierte Service-Instanz aus der Registry, wenn sie nicht mehr gebraucht wird. Das ist dann der Fall, wenn keine Aufrufe mehr von der Service-Instanz bearbeitet werden. Der ESB entscheidet, wann dieser Use-Case aufzurufen ist.

Register Service Call

Dieser Use-Case registriert einen Aufruf eines Service-Consumers für eine nicht-provisionierte Shared-Service-Instanz. Folglich wird die Zahl der Aufrufe um 1 inkrementiert. Dieser Use-Case ist nicht für Dedicated-Service-Instanzen vorgesehen, da für diese Instanzen nur 2 Zustände möglich sind: der Zustand mit einem Aufruf oder der Zustand mit keinem Aufruf. Dieser Use-Case kann erst auf eine Shared-Service-Instanz angewendet werden, nachdem sie mit "Register Not Provisioned Service Instance" provisioniert wurde.

Unregister Service Call

Dieser Use-Case dekrementiert die Zahl der Aufrufe eines Service-Consumers für eine nicht-provisionierte Shared-Service-Instanz um 1. Dieser Use-Case ist ebenfalls nur für Shared-Service-Instanzen gedacht und nicht für Dedicated-Service-Instanzen vorgesehen. Falls eine Shared-Service-Instanz nur noch einen Aufruf verarbeitet, darf dieser Use-Case nicht aufgerufen werden. Bei genau einem übrigen Aufruf muss "Unregister Not Provisioned Service Instance" benutzt werden.

5.2.3 Use-Cases von beiden Aktoren

Search Service Offers by PortType-Name (findServices)

Dieser Use-Case sucht in der Service Registry nach Service Offers, die Service Configurations implementieren, welche wiederum ein bestimmtes Service-Interface haben. Dieses Service-Interface muss genau den gesuchten PortType-Namen haben.

List all Service Offers

"List all Service Offers" gibt alle Service-Angebote in der Service Registry zurück ohne weitere Einschränkungen.

6 Implementierung

Für die Service Registry-Implementierung wird keine existierende Service Registry-Lösung angepasst. Diese Entscheidung wurde nach der Analyse existierender Lösungen in Kapitel 4 getroffen. Stattdessen wird eine neue Software implementiert, die genau auf die konzeptionelle Anforderungsbeschreibung in Kapitel 5 zugeschnitten ist. Die Implementierung trägt den Namen "Service-Registry-Service" (kurz SRS) und realisiert die Registry selbst als Webservice. Die folgenden Kapitel beinhalten eine detaillierte Beschreibung der Implementierung des Service-Registry-Service.

6.1 Verwendete Technologien

Java

Java[9] ist eine objektorientierte, imperative Programmiersprache die sehr stark verbreitet ist. Daher wird hier nicht weiter auf Java eingegangen.

JavaScript

JavaScript[10] Ist nicht zu verwechseln mit der Programmiersprache Java. JavaScript ist eine klassenlose Skriptsprache zur vereinfachten Implementierung von interaktiven Webapplikationen.

HttpServlets

Hiermit sind Java-Servlets gemeint aus dem Standard "javax package[11]" [4]. Diese Servlets kann man in einem Servlet-Container definieren und benutzen. HttpServlets bilden Schnittstellen und Methoden, deren Inhalte man mit einer eigenen Implementierung füllen kann. Anfragen in Form von HttpRequest-Objekten können dort verarbeiten werden und Antworten in Form von HttpResponse-Objekten gebaut werden.

Axis2

Axis2 ist die zweite Version von Apache Axis. Es ist eine SOAP-Engine, mit der es möglich ist unter anderem durch Data-Binding XML-Inhalte in Objekte von Java umzuwandeln. Folglich nimmt Axis2 bei der Entwicklung eines Web-Services die Arbeit des manuellen Parsens von XML-Nachrichten und der anschließenden Umwandlung in Java-Objekte ab.

[9] Java als Programmiersprache, http://de.wikipedia.org/wiki/Java_(Programmiersprache)
[10] JavaScript als Skriptsprache, http://de.wikipedia.org/wiki/JavaScript
[11] Organisieren von Java Klassen mit "packages", http://en.wikipedia.org/wiki/Java_package

Jetty

Jetty[12] ist eine leichtgewichtige Servlet-Engine, in der man seine eigene Webapplikation mit eigenen Servlets in Java definieren kann. Es ist ein Projekt, das im Umfeld einer Eclipse Community entwickelt wird. In der Implementierung wird die Embedded-Jetty Version benutzt, bei der man in einem Java-Projekt seinen eigenen Webserver definieren kann.

Tomcat

Apache Tomcat ist ein leichtgewichtiger Webserver, der die Spezifikation von Java-Servlets implementiert. Es erlaubt es dadurch, unter anderem, Webanwendungen in Form von Web-Application-Archives (WAR) zu deployen. Tomcat bietet eine alternative Möglichkeit, die SOAP-Schnittstelle der Implementierung zu nutzen, da sich das Java-Projekt der Implementierung ebenso als WAR exportieren und auf Tomcat deployen lässt.

SOAP

Das Simple Object Access Protocol (SOAP) ist ein Netzwerkprotokoll zum Austausch von Daten zwischen Systemen. Es ermöglicht in unserer Implementierung die Kommunikation zwischen dem ESB und dem SRS auf Basis von SOAP-Envelopes.

MySQL

MySQL ist ein weit verbreitetes, relationales Datenbankverwaltungssystem. In MySQL lassen sich mehrere Datenbanken definieren, die relational sind. MySQL wird in der ersten Version des SRS genutzt, kann aber auch durch jede andere relationale Datenbank ersetzt werden.

Hibernate

Hibernate ist ein Object-Relational-Mapping-Framework (ORM-Framework) für Java. Mit Hibernate können Objekte in einer relationalen Datenbank persistent gemacht werden. Dafür muss die Datenbankstruktur vom Entwickler nicht selbst gebaut werden und es müssen auch keine eigenen dynamischen SQL-Befehle geschrieben werden.

[12] das Eclipse-Projekt von Jetty, http://www.eclipse.org/jetty/

6.2 Architektur

In der Abbildung 9 sieht man ein Architekturbild, das eine Übersicht über die Komponenten des Service-Registry-Services darstellt.

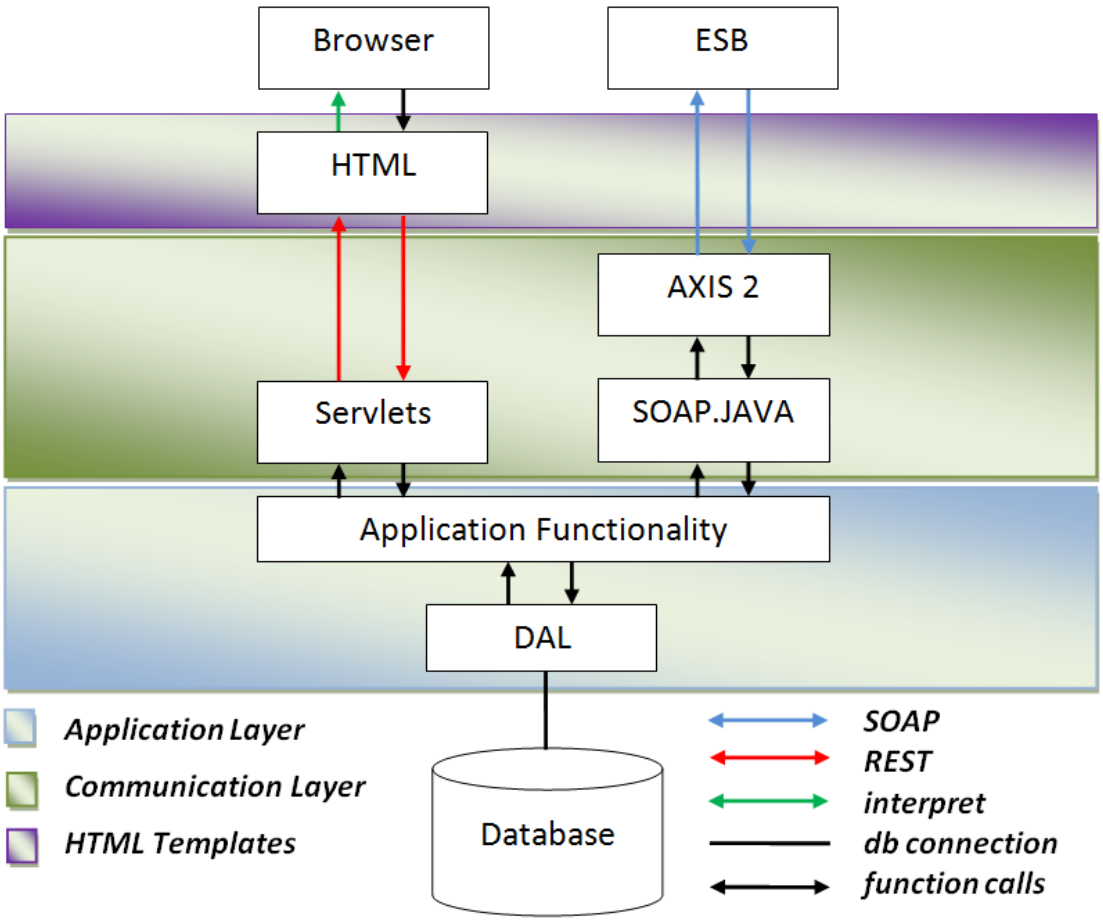

Abbildung 9: Die Architektur der Service Registry

Man betrachte das Architekturbild oben mittig beginnend bei den Möglichkeiten, wie der Service-Registry-Service angesprochen werden kann. Es gibt den ESB, der über das SOAP Protokoll Anfragen an den SRS stellt, und es gibt den Browser, der über HTTP Anfragen an den SRS stellt.

Über SOAP (rechts in der Abbildung) nimmt der SRS im Communication Layer Anfragen mit Hilfe von Axis2 generierten Java-Klassen entgegen. Es wurde dafür zuerst eine WSDL-Datei als Schnittstelle definiert, die bestimmte Methoden des SRS anbietet. Es gibt Methoden, die nur für den ESB relevant sind. Es gibt in der SOAP Schnittstelle aber auch Methoden, mit denen man jede andere Funktion der SRS-Applikationslogik nutzen kann, wie das Registrieren eines Service-Angebots. Das hat den Vorteil, dass man auch einen externen Webservice-Client für den SRS schreiben könnte. In der AXIS2-Komponente wird jede Anfrage in Java-Code formatiert. Es werden mit Hilfe von Axis2 Java-Objekte automatisch aus der WSDL-Schnittstellenbeschreibung generiert, die dann alle in einer Klasse zur Implementierung eines Mappings benutzt werden. Diese Klasse bildet die Komponente SOAP.JAVA. Dort lässt sich ein Mapping implementieren, das aus Axis2 Java-Objekten die

relevanten Daten für eine Anfrage ausliest und diese dann in Methoden für die Applikationslogik (Application Functionality) einsetzt. Sobald die Methoden der Applikationslogik ausgeführt sind, geben sie eine Antwort an die Mapping-Klasse von SOAP.JAVA zurück. Diese Antwort muss nun in ein Rückgabeobjekt von Axis2 eingesetzt werden und gibt dann eine SOAP-Antwort an den Aufrufenden zurück.

Für die REST Schnittstelle (links in der Abbildung) wurde zuerst eine Website implementiert, die das Benutzen des SRS verständlicher machen soll. Es ist eine grafische Oberfläche, die aus HTML-Templates besteht und von einem Browser interpretiert werden kann. Der Benutzer kann an bestimmte Stellen auf der Website navigieren und dort dann beispielsweise HTML-Formulare zum Eintragen eines Service-Angebots ausfüllen und abschicken. Diese Formulare werden dann über HTTP-Requests an Servlets im Java-Code weitergeleitet (siehe Komponente "Servlets"). Ähnlich wie beim Mapping der Axis2 Java-Objekte werden hier Inhalte aus HTTP-Request-Objekten ausgelesen und in entsprechende Methoden der Applikationslogik eingesetzt. Sobald die Methoden der Applikationslogik ausgeführt sind, geben sie eine Antwort an das Servlet zurück. Im Servlet muss aus dieser Antwort ein HTTP-Response-Objekt gebaut werden, das eine Antwort an den Aufrufenden zurückgibt.

Schließlich sitzt im Application Layer die Applikationslogik, die eng mit dem Data-Access-Layer (DAL) Objekte in der Datenbank persistent speichert oder Informationen aus der Datenbank ausliest. Es werden hier Data-Access-Objects benutzt, die jeweils für einzelne Modelklassen zuständig sind. Beispielsweise gibt es ein Data-Access-Object für den Benutzer (UserDAO.java). In der UserDAO.java werden dann die Methoden "registerUser()" und "loginUser()" angeboten zum Registrieren oder Authentifizieren eines Benutzers im System.

6.3 Schnittstellen

Hier wird konkret beschrieben, wie die Methoden der SOAP-Schnittstelle zum SRS benutzt werden können. Es wurde zuerst eine WSDL-Datei für die Schnittstelle definiert und daraus ein Axis2 Java-Skelett generiert, in der die Funktionalität dann ausimplementiert wurde. In dieser Arbeit wird generell beschrieben, welche Methoden über die SOAP-Schnittstelle verfügbar sind und welche Funktion sie implementieren. Für eine detaillierte Beschreibung der Parameter, der Felder die nicht leer sein dürfen, der generierten Antworten, sowie deren Bedeutung, wird an dieser Stelle auf das Benutzerhandbuch verwiesen.

initialize

Diese Methode Initialisiert das System, indem es einen Standard-Benutzer anlegt für den SRS. Der Standard-Benutzer kann zuvor in einer Konfigurationsdatei definiert werden. Diese Methode sollte vor allem bei der ersten Benutzung aufgerufen werden, oder nach einem Wechsel auf eine neue, leere Datenbank. Diese Methode ist nicht zwingend notwendig, da das System beim Jetty-Serverstart automatisch initialisiert wird und einen Benutzer anlegt, sofern er nicht vorhanden ist.

getAllServices

Diese Methode gibt alle Service-Angebote zurück, ohne Einschränkungen. Dies ist lediglich eine Hilfsmethode und kann vor allem bei vielen Service-Angeboten sehr zeitintensiv sein. Auch die Antwort kann bei vielen Angeboten unübersichtlich erscheinen.

searchByProvider

Diese Methode gibt alle Service-Angebote zurück, die von einem bestimmten Service-Anbieter (Provider) im SRS registriert wurden. Der Name des Service-Providers muss genau mit dem Namen eines Service-Providers in der Datenbank übereinstimmen.

searchByServiceConfiguration

Diese Methode gibt alle Service-Angebote zurück, die eine bestimmte Service-Configuration implementieren. Hier wird auch nach dem Namen der Service-Configuration gesucht. Der Name ist für jede Service-Configuration eindeutig und muss mit dem Namen einer Service-Configuration in der Datenbank genau übereinstimmen.

findService

"findService" hieß früher "searchByQualifiedPortTypeName" und gibt alle Service-Angebote zurück, die ein bestimmtes Service-Interface implementieren. Das Service-Inteface ist eindeutig durch den PortType-Namen identifizierbar und beschreibt die technische Schnittstelle eines Services. Der Name muss genau mit dem PortType-Namen des Service-Interfaces in der Datenbank übereinstimmen.

registerNPSInstance

Diese Methode registriert eine Service-Instanz eines nicht-provisionierten Service-Angebots. Hier kann sowohl eine Shared-Service-Instanz, als auch eine Dedicated-Service-Instanz registriert werden, durch die Angabe des Servie-Angebots, von welchem die Instanz erstellt wird. In dieser Methode muss auch der Benutzer angegeben werden, da die Unterscheidung zwischen Shared-Service-Instanz und Dedicated-Service-Instanz wichtig ist. Bei einem Aufruf für ein Dedicated-Service-Angebot wird einfach eine Dedicated-Service-Instanz erstellt, ohne weitere Prüfung und egal von welchem Benutzer. Bei einem Aufruf für ein Shared-Service-Angebot muss zuerst geprüft werden, ob es von diesem Angebot schon eine Shared-Service-Instanz des Benutzers, der im Aufruf angegeben wurde, gibt. Wenn ja, dann wird der Aufruf an diese Instanz weitergeleitet und lediglich "registerServiceCall" aufgerufen, um die Zahl des Aufrufs für die Shared-Service-Instanz um 1 zu inkrementieren. Falls es noch keine solche Shared-Service-Instanz gibt, wird eine neue für den jeweiligen Benutzer provisioniert.

unregisterNPSInstance

Diese Methode löscht eine Registrierung einer Service-Instanz eines nicht-provisionierten Service-Angebots. Diese Methode ist nur für Dedicated-Service-Instanzen, oder Shared-Service-Instanzen gedacht, deren aktuelle Zahl der Aufrufe genau 1 ist. Falls die Zahl höher

ist (was nur für Shared-Service-Instanzen möglich ist), darf die nicht-provisionierte Service-Instanz noch nicht aus der Registry gelöscht werden. Dafür sollte dann stattdessen die Anzahl der Aufrufe mit "unregisterServiceCall" um 1 dekrementiert werden.

registerServiceCall

Diese Methode inkrementiert die Zahl der Aufrufe eines Shared-Service-Instanz um 1. Diese Methode ist nur für Shared-Service-Instanzen vorgesehen, die bereits mindestens einen Aufruf verarbeiten, da dort die Zahl der Aufrufe höher als 1 sein kann.

unregisterServiceCall

Diese Methode dekrementiert die Zahl der Aufrufe einer Shared-Service-Instanz um 1. Diese Methode ist nur für Shared-Service-Instanzen vorgesehen, deren Aufrufzahl echt höher als 1 ist. Falls die Anzahl der Aufrufe genau 1 ist, sollte die Methode "unregisterNPSInstance" benutzt werden, um die Service-Instanz aus dem SRS zu löschen.

registerUser

Die Methode "registerUser" registriert einen Benutzer im System. Dieser Benutzer kann sich dann mit Name und Passwort im System authentifizieren mit der Methode "loginUser". Das Registrieren des Benutzers bringt zum einen die Möglichkeit, diesen Benutzer beim Anlegen von Service-Angeboten als Service-Anbieter einzutragen, und zum anderen kann der ESB den Benutzer für Aufrufe an nicht-provisionierte Service-Angebote mit angeben.

loginUser

Die Methode "loginUser" authentifiziert einen Benutzer im System mit Hilfe eines Authentifizierungs-Token. Dieses Token wird als Antwort zurückgegeben und repräsentiert den eingeloggten Benutzer, da der Name und das Passwort des Benutzers eingegeben werden müssen, um das Token zu bekommen. Es wird bei vielen anderen Methoden verwendet, in denen ein Service-Provider bekannt sein muss, wie beispielsweise das Registrieren eines Service-Angebots. Das Token hat den Vorteil, dass nur ein String übergeben werden muss und es nur zeitlich begrenzt gültig ist.

createProvisionedService

Diese Methode registriert ein Service-Angebot für einen provisionierten Service. Dieser Service wird in der Regel von Drittanbietern registriert für einen Service, der extern betrieben wird. Es muss hier also die ServiceConfiguration und ein Endpunkt angegeben werden, über den der Service erreichbar ist.

createNotProvisionedSharedService

Diese Methode registriert ein Service-Angebot für einen nicht-provisionierten Shared-Service. Es muss ein Service Package angegeben werden, dass ein Installationspaket aus der Service Package Repository referenziert und ein RepositoryType angibt. Aus diesem Service-Angebot können Shared-Service-Instanzen provisioniert werden, die mehrere Aufrufe des gleichen Benutzers gleichzeitig abarbeiten können.

createNotProvisionedDedicatedService

Diese Methode registriert ein Service-Angebot für einen nicht-provisionierten Dedicated-Service. Es muss ein Service Package angegeben werden, dass ein Installationspaket aus der Service Package Repository referenziert und ein RepositoryType angibt. Aus diesem Service-Angebot können Dedicated-Service-Instanzen provisioniert werden, die jeweils nur einen Aufruf eines Benutzers zur gleichen Zeit abarbeiten können.

editProvisionedService

"editProvisionedService" bearbeitet ein Service-Angebot für einen provisionierten Service. Hier können fast alle Daten geändert werden, die zuvor beim Anlegen dieses Angebots angegeben wurden. Der Service-Anbieter ist dafür verantwortlich, dass die Daten des Services auch mit den geänderten Daten im SRS übereinstimmen. Diese Methode kann nur vom Service-Anbieter genutzt werden, der dieses Service-Angebot auch registriert hat.

editNotProvisionedSharedService

Diese Methode bearbeitet ein Service-Angebot für einen nicht-provisionierten Shared-Service. Es können fast alle Daten geändert werden, die zuvor beim Anlegen dieses Angebots angegeben wurden. Der Service-Typ (Shared- oder Dedicated-Service) kann nicht geändert werden. Für eine Änderung des Service-Typs muss das alte Service-Angebot gelöscht werden und ein neues Service-Angebot registriert werden. Der Service-Anbieter ist dafür verantwortlich, dass die Daten des Services auch mit den geänderten Daten im SRS übereinstimmen. Diese Methode kann nur vom Service-Anbieter genutzt werden, der dieses Service-Angebot auch registriert hat.

editNotProvisionedDedicatedService

Diese Methode bearbeitet ein Service-Angebot für einen nicht-provisionierten Dedicated-Service. Es können fast alle Daten geändert werden, die zuvor beim Anlegen dieses Angebots angegeben wurden. Der Service-Typ (Shared- oder Dedicated-Service) kann nicht geändert werden. Für eine Änderung des Service-Typs muss das alte Service-Angebot gelöscht werden und ein neues Service-Angebot registriert werden. Der Service-Anbieter ist dafür verantwortlich, dass die Daten des Services auch mit den geänderten Daten im SRS übereinstimmen. Diese Methode kann nur vom Service-Anbieter genutzt werden, der dieses Service-Angebot auch registriert hat.

deleteService

"deleteService" löscht ein Service-Angebot aus dem SRS. Diese Methode ist mit Vorsicht zu verwenden. Das Löschen eines Service-Angebots für einen provisionierten Service löscht einfach das Angebot. Das Löschen eines Service-Angebots für nicht-provisionierte Services löscht auch alle eingetragenen Instanzen aus dem SRS. Das bedeutet nicht, dass die Instanzen automatisch de-provisioniert werden, sie wurden lediglich aus dem SRS gelöscht, weil sie ohne ihr Service-Angebot nicht mehr bearbeitet werden können.

createServiceInterface

Diese Methode rrstellt ein Service Interface, das eine Schnittstellenbeschreibung für einen Service ist. Mit dieser Schnittstellenbeschreibung wird eine funktionale Anforderung für Services definiert. Service Interfaces können später mit nicht-funktionalen Eigenschaften in Service-Configurations kombiniert werden. Service Interfaces haben einen eindeutigen PortType-Namen, der eine Service Discovery ermöglicht.

deleteServiceInterface

Diese Methode löscht ein ServiceInterface aus dem SRS. Ohne ein Service-Interface wird auch jede Service-Configuration unbrauchbar, die dieses Service Interface referenziert. Damit wird auch jede Service-Configuration gelöscht, die das Service Interface referenziert hat. Service-Angebote die eine solche Service-Configuration referenziert haben, haben dann vorerst keinen Eintrag mehr für eine Service-Configuration. Es kann eine neue Service-Configuration durch die Bearbeitungsmethoden editProvisionedService, editNotProvisionedSharedService und editNotProvisionedDedicatedService eingetragen werden.

createServiceConfiguration

Diese Methode erstellt eine Service-Configuration, die funktionale Eigenschaften mit Service Interfaces und nicht-funktionale Eigenschaften eines Services kombiniert. In dieser Methode werden die nicht-funktionalen Eigenschaften als Map aus String-String-Paaren mitgegeben. Eine Service-Configuration kann dann von Service-Angeboten zur Kategorisierung genutzt werden.

createServiceConfigurationWithInterface

Diese Methode erstellt sowohl ein Service Interface, als auch eine Service-Configuration. Diese Methode kombiniert das Hintereinanderausführen von "createServiceInterface" und "createServiceConfiguration" mit dem Unterschied, dass wenn etwas beim Erstellen der Service-Configuration schiefläuft auch kein Service-Interface erstellt wird. So gesehen ist die Methode atomar.

deleteServiceConfiguration

Diese Methode löscht eine Service-Configuration aus dem SRS. Service-Interfaces, die von der Service-Configuration referenziert wurden bleiben in der Datenbank, da sie unabhängig von den Service-Configurations sind. Service-Angebote die eine solche Service-Configuration referenziert haben, haben dann vorerst keinen Eintrag mehr für eine Service-Configuration. Es kann eine neue Service-Configuration durch die Bearbeitungsmethoden editProvisionedService, editNotProvisionedSharedService und editNotProvisionedDedicatedService eingetragen werden.

6.4 Grafische Benutzeroberfläche

In diesem Kapitel wird die konzeptionelle Arbeit an der grafischen Benutzeroberfläche für die Darstellung im Browser erläutert. Für konkretere Darstellungen und die Benutzung der Oberfläche wird auf das Benutzerhandbuch verwiesen. Für Beispiele bei der Entwicklung der grafischen Benutzeroberfläche wird auf das Entwicklerhandbuch verwiesen.

In der Abbildung 10 wird zunächst der konzeptionelle Aufbau der grafischen Benutzeroberfläche dargestellt.

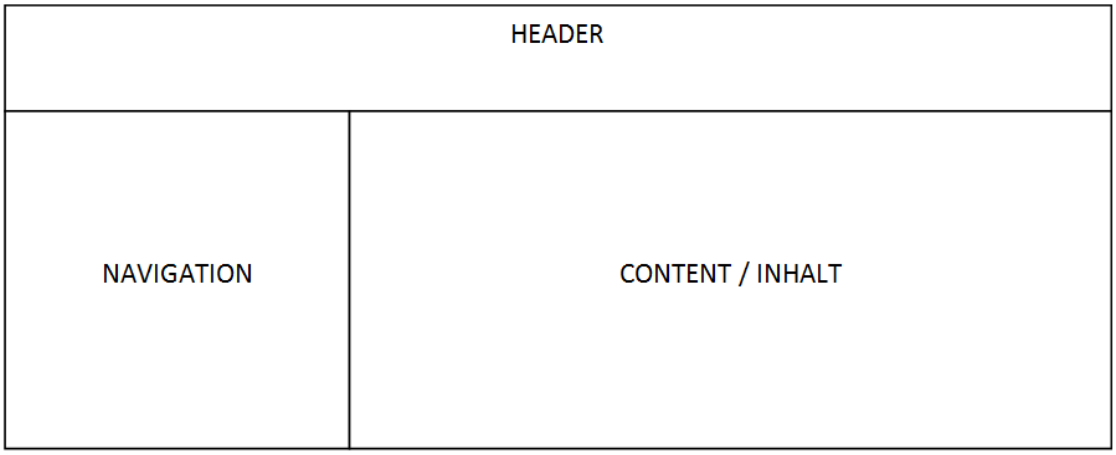

Abbildung 10: Konzeptioneller Aufbau der grafischen Benutzeroberfläche

Oben befindet sich der Header, der sich über die gesamte Breite des SRS streckt. Er ist größtenteils statisch und hat nur wenige Elemente, die sich bei der Navigation durch die Oberfläche verändern. Auf der linken Seite befindet sich die Navigation, in der es ein Seitenmenü geben wird, das Buttons für die einzelnen Menüpunkte anbietet. Der große Bereich unten rechts für den Inhalt passt sich mit jedem Untermenü dynamisch an.

In den folgenden Abbildungen werden die großen Abschnitte der Website mit einem Namen in Großbuchstaben versehen. In den Abschnitten sind Elemente mit spitzen Klammern versehen, wenn dort ein Button ist oder ein Text steht, der etwas beschreibt. Elemente, die sich bei der Navigation auf der Oberfläche dynamisch ändern, sind zusätzlich mit einem gestrichelten Kasten umrandet. In der Abbildung 11 wird erklärt, wie der Header aufgebaut ist.

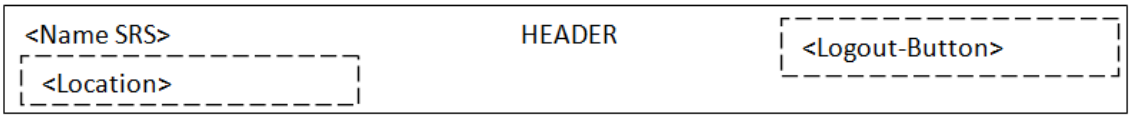

Abbildung 11: Aufbau des Headers

Im Header soll oben links der Name des Systems fest stehen. Unten links wird die "Location", also der Ort in der man sich auf der Website befindet, angezeigt. Dieses Element ändert sich dynamisch beim Navigieren in jedes Untermenü der grafischen Benutzeroberfläche. Oben rechts wird ein Logout-Button angezeigt, sobald ein Benutzer

sich im System eingeloggt hat. Grundsätzlich muss sich jeder Benutzer auf der Oberfläche anmelden, ob er nur nach Services suchen will, oder ob er ein Service-Angebot im SRS registrieren will. Die Login-Ansicht und die Ansicht zum Registrieren neuer Benutzer sind die einzigen, die sich vom restlichen Aufbau des Systems unterscheiden. Die Login-Ansicht wird in Abbildung 12 dargestellt.

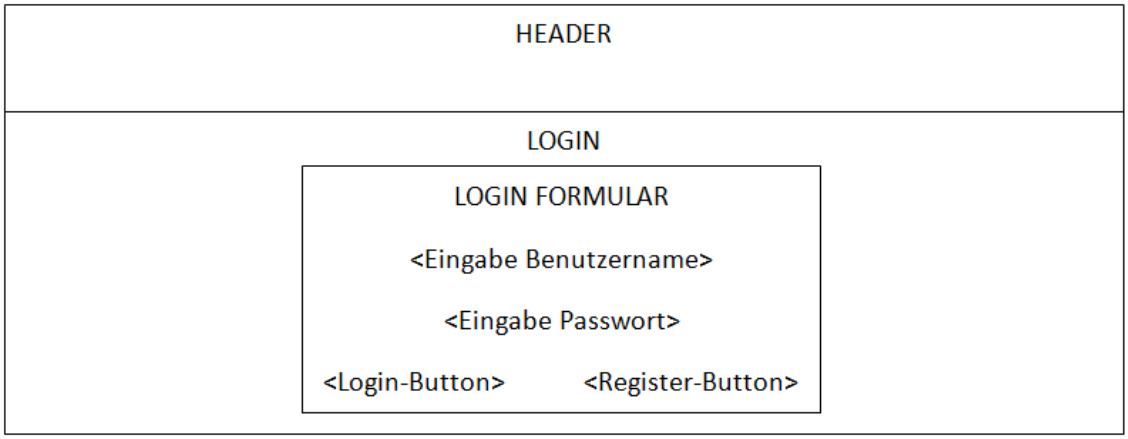

Abbildung 12: Die Login-Ansicht

Man bemerkt, dass im Abschnitt LOGIN ein weiterer Abschnitt mit LOGIN FORMULAR eingebaut ist. Die Login-Ansicht hat nicht die ursprünglich beschriebene Trennung von Navigation und Inhalt, die für alle untergeordneten Ansichten gilt. Stattdessen ist das LOGIN FORMULAR zentriert im Bild. Der LOGIN-Abschnitt ist nur ein Hintergrund, während das LOGIN FORMULAR die Eingabemaske für einen Benutzernamen und ein Passwort hat. Darin befindet sich unten links der Login-Button und unten rechts der Verweis auf die Registrierungsseite für einen neuen Benutzer. Die Ansicht zum Registrieren erstreckt sich ähnlich über die komplette Breite und wird hier nicht genauer beschrieben. Hat ein Benutzer sich mit der Login-Maske erfolgreich eingeloggt, so erscheint er in der initialen Ansicht der Oberfläche. Diese ist, wie in der vorher beschriebenen Abbildung 10 aufgebaut, wobei im Inhalt ein kleiner Beschreibungstext den Benutzer auffordert, ein Untermenü aus der Navigation auszuwählen. Der Aufbau der Navigation ist in der Abbildung 13 dargestellt.

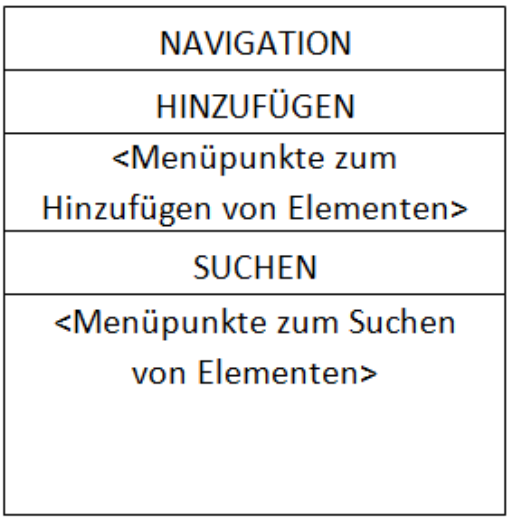

Abbildung 13: Aufbau der Navigation

Die Navigation hat Menüpunkte in Form von Buttons. Diese Menüpunkte sind gruppiert durch Zwischenüberschriften. Durch Klick auf jedes Menü ändert sich der große Abschnitt mit dem Inhalt der Website aus Abbildung 10. Oben befindet sich eine Überschrift (HINZUFÜGEN) für alle Menüpunkte, die eine Ansicht zum Hinzufügen von Elementen im SRS öffnen. Das sind dann beispielsweise Menüpunkte zum Hinzufügen eines Service Interfaces, einer Service-Configuration oder einem Service-Angebot. Darunter befindet sich eine Überschrift (SUCHEN) für alle Menüpunkte, die eine Ansicht zum Suchen von Elementen im SRS öffnen. Hier können alle Elemente, die zuvor hinzugefügt wurden, im System gesucht und eingesehen werden.

7 Zusammenfassung

Diese Arbeit baut auf der Architektur der bisherigen Arbeiten aus Kapitel 3 auf. Das Ziel der Arbeit ist es, aus den formulierten Anforderungen aus Kapitel 1.2 und dem Konzept aus Kapitel 5 eine Service Registry zu implementieren. Am IAAS[13] wurde ein Datenmodell für die Service Registry erarbeitet, das zu Beginn der Arbeit schon vorhanden war. Anhand dieses Datenmodells und den Anforderungen wurden bestehende Service Registry-Lösungen betrachtet und einige davon genauer analysiert. Eine Vorbedingung für die Analyse war, dass die bestehende Service Registry Lösung Open-Source ist, da keine der betrachteten Service Registry-Lösungen nicht-provisionierte Services unterstützt haben. Die Analyse musste prüfen, ob bestehende Lösungen vorteilhaft für die Anforderungen angepasst werden konnten. Für das Datenmodell der Service Registry mussten nebenher Änderungen vorgenommen werden. Das Ergebnis der Analyse fasst alle Vor- und Nachteile der bestehenden Lösungen zusammen. Es hat jedoch ergeben, dass bestehende Lösungen nur mit beträchtlichem Aufwand angepasst werden könnten. Folglich ist eine eigene Implementierung das Ergebnis dieser Arbeit. Die Implementierung wurde durch einen eingebauten Jetty-Webserver in einem Java-Projekt realisiert, der zwei Möglichkeiten zur Kommunikation anbietet: eine SOAP Schnittstelle, die primär durch den Enterprise Service Bus genutzt wird und eine grafische Oberfläche, auf der die Arbeit mit dem SRS für menschliche Benutzer vereinfacht werden soll.

8 Ausblick

Die Ausarbeitung und Implementierung des Service Registry Service für On-Demand-Services ist zwar für den Rahmen der Anforderungen in dieser Arbeit abgeschlossen. Es existiert jedoch noch Future Work, das den SRS erweitern kann. Diese Erweiterungen hängen mit dem Fortschritt der Forschungsarbeit für On-Demand-Services zusammen. Beispielsweise ist für diese Arbeit noch nicht klar gewesen, wie man das Provisionieren und De-provisionieren für On-Demand-Services optimieren kann. Der aktuelle Stand ist, dass Services einfach de-provisioniert werden, wenn es keine Aufrufe mehr für sie zum Bearbeiten gibt, obwohl man diese Services in einem Workflow eventuell nach kurzer Zeit wieder gebrauchen könnte.

In dieser Arbeit ist die Beschreibung der funktionalen Eigenschaften für Services im Service Interface genauer erarbeitet, als die Beschreibung für nicht-funktionale Eigenschaften. Die "Qualities-of-Services" bestehen momentan nur aus einfachen Key-Value-Paaren. Eine genauere Erarbeitung für die nicht-funktionalen Eigenschaften wäre für den SRS ebenfalls denkbar.

[13] Institut für Architektur von Anwendungssystemen, http://www.iaas.uni-stuttgart.de/

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Literatur

[1] Karolina Vukojevic-Haupt; Florian Haupt; Dimka Karastoyanova; Frank Leymann: Service Selection for On-demand Provisioned Services: Proceedings of the 18th IEEE International EDOC Conference (EDOC 2014) 2014.

[2] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana: Web Services Description Language (WSDL) 1.1.
URL: http://www.w3.org/TR/wsdl.

[3] OASIS: UDDI Spec TC. UDDI Version 3.0.2.
URL: http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[4] Sun Microsystems: JavaTM 2 Platform Enterprise Edition, v 1.4 API Specification.
URL: http://docs.oracle.com/javaee/1.4/api/overview-summary.html.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, _____

Alexander Blehm _____