Eine Web Service Schnittstelle für ein Web Service Entwickler-Portal

Bachelor-Arbeit

im Studiengang Informationstechnologien,
angefertigt im Arbeitsbereich Softwaresysteme
der Technischen Universität Hamburg-Harburg

von

Helge Sören Klimek
Matrikelnummer: 22806

Hamburg, Oktober 2005

Betreuer: Dipl. Inform. Rainer Marrone
Gutachter: Prof. Dr. Joachim W. Schmidt

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, daß ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde weder einer anderen öffentlichen oder privaten Institution vorgelegt noch veröffentlicht.

`Helge Sören Klimek`
Hamburg, den 24. Oktober 2005

Danksagung

Der Entstehung dieser Arbeit standen viele Menschen hilfreich zur Seite. Ich möchte Prof. Dr. Joachim W. Schmidt für seine guten Anregungen und fachliche Betreuung danken. Mein besonderer Dank gilt Dipl. Inform. Rainer Marrone für seine Unterstützung. Er hat sich für die Betreuung dieser Arbeit viel Zeit genommen.

Weiterer Dank gilt Dr. Hans-Werner Sehring, Sebastian Bossung, Birgit Guth, Jürgen Meincke und Werner Wendt, für fachliche Unterstützung, hilfreiche Ratschläge und Tips.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dataport ist der Dienstleister, der für die öffentlichen Verwaltungen der Freien- und Hansestadt Hamburg sowie für das Land Schleswig Holstein die Informations- und Kommunikationstechnik zur Verfügung stellt. Die Anstalt des öffentlichen Rechts ist am 01.01.2004 aus der Fusion der Datenzentrale Schleswig-Holstein mit dem Landesamt für Informationstechnik (Hamburg) und der IuK-Abteilung des Senatsamtes für Bezirksangelegenheiten (Hamburg) hervorgegangen.

Seither umfasst das Aufgabengebiet von Dataport unter anderem die folgenden Dienstleistungen: Lösungen für Fachaufgaben wie Anwendungsentwicklung für Länder und Kommunen, Rechenzentrumsdienste wie hochverfügbare Plattformen (z/OS, Unix, Windows) und das Systems-Management, Netzdienste wie Infrastrukturdienste und Netzwerkmanagement, Internetdienste welche Internetzugänge, Planung, Realisierung, Hosting, Firewall und E-Mailing umfassen und Telekommunikationsdienste wie Sprach- und Datendienste für Hamburg und Schleswig-Holstein.

Ein zentrales Betätigungsfeld ist dabei die Weiterentwicklung des GovernmentGateway sowie die Entwicklung der Fachverfahrenanwendungen. Das Gateway wurde als strategische E-Government-Komponente für die sichere Bereitstellung von Dienstleistungen im Internet entwickelt und ist als HamburgGateway seit 2003 in Betrieb. Nach der Fusion Anfang 2004 wurde es mit dem Namen GovernmentGateway in die Dataport Plattform integriert. Im Wesentlichen stellt das Gateway einen sicheren Zugang im Intra- und Internet zu den *Backend*- und Fachverfahrens-Anwendungen dar.

Fachverfahren werden sukzessive in das GovernmentGateway integriert und dafür mit Web Service Schnittstellen erweitert, die für die Kommunikation zwingend erforderlich sind. Nutzer der Fachverfahren sind Bürger, Unternehmen und Behörden, die über das GovernmentGateway oder direkt über die Web Service Schnittstellen die Verfahren benutzen können, um Informationen zu beziehen.

Im Zusammenhang mit der Softwareentwicklung für das Gateway gibt es eine Vielzahl von Entwicklern, die involviert sind: Basis-Architektur-Entwickler, Gateway-Fachverfahren-Entwickler und Web Service Entwickler. Zudem gibt es Kunden-Entwickler, die zum Beispiel von anderen Behörden oder Unternehmen stammen und Client-Anwendungen für Web Services schreiben, um diese zu nutzen.

Für die Entwicklung ist es notwendig, daß die oben genannten Entwicklergruppen Informationen austauschen. Dieses Wissen erstreckt sich dabei nicht nur auf eine Vielzahl von Dokumenten, wie Code-Beispiele, Schnittstellen-Beschreibungen oder Datenbank-Sicherungen, sondern auch auf Zusatzinformationen zu diesen Dokumenten. Dazu zählen unter anderem Informationen über die Abhängigkeiten zwischen den Dokumenten, Vorgaben, von wann bis wann ein Dokument Gültigkeit besitzt und sehr viel mehr. Dabei muss eine große Menge heterogener Informationen in Zusammenhang gebracht werden.

Da die Anzahl der Entwickler sowie die Anzahl der zu verwaltenden Dokumente stetig zunimmt, wird eine Plattform benötigt, mit der die Dokumente verwaltet und mit der Wissensträger ausfindig gemacht werden können. Diese Aufgabe soll durch das Entwickler-Portal realisiert werden. Insbesondere soll das Portal eine steigende Anzahl von Web Service Beschreibungen und Informationen von Dataport den beteiligten Entwicklern und Kunden zur Verfügung stellen. [11],[13]

1.1 Motivation für die Web Service Schnittstelle im Entwickler-Portal

Das GovernmentGateway unterstützt die beiden Applikationsinfrastrukturen *.net* und *Java*. Das liegt unterem Anderem daran, daß beide Architekturen in den Ländern Schleswig-Holstein und Hamburg bereits im Einsatz sind.

Aufgrund dieser Heterogenität spielen Web Services für Dataport eine erhebliche Rolle. Sie werden verwendet, um die Welten von *.net* und *Java* sowie *Legacy*-Applikationen zu verknüpfen.

Zur Kommunikation zwischen den Komponenten des Gateways und den Lösungsapplikationen gibt es zwei Schnittstellenarten: Bei protokollgebundenen Schnittstellen (Kommunikationsschnittstellen) wird die Kommunikation über ein meist an das *Internet Protocol* (IP) gebundenes, standardisiertes Netzwerkprotokoll (zum Beispiel HTTP, SOAP,...) abgewickelt. Es ist in der Regel systemunabhängig. Das *Application Programming Interface* (API) also die Programmierschnittstellen werden auf Ebene des Quellcodes eingebunden und sind überlichweise an die Programmiersprache gebunden.

Dataport favorisiert die standardisierten, protokollgebundenen Schnittstellen, da ihre Implementation nur einmal vorgenommen werden muss, um mit *.net* und *Java* zu funktionieren. Die Kommunikation aus dem Gateway mit Fachverfahren läuft generell nur über Web Services.

Das Entwickler-Portal soll ähnlich eines Fachverfahrens in das *Backend* von Dataport integriert und mit dem GovernmentGateway verbunden werden. Dadurch sollen vorhandene Gateway-Funktionalitäten wie zum Beispiel die Benutzerverwaltung und Sicherheitsmaßnahmen durchgehend verwendet werden.

Da es bisher innerhalb des Entwickler Portals keine Möglichkeit gibt Fremdsysteme einzubinden, entsteht hieraus die Notwendigkeit, eine Web Service Schnittstelle zu integrieren. Diese Bachelor-Arbeit befasst sich mit dieser Aufgabe. [13]

1.2 Struktur der Arbeit

Die Realisierung der Web Service Schnittstelle ist in dieser Arbeit in mehrere Teile untergliedert. In Kapitel 2 wird zunächst die Web Service Technologie eingeführt und erklärt. Neben ein paar Definitionen unterschiedlicher Hersteller wird das Simple Object Access Protocol (SOAP), die Web Service Description Language (WSDL) und das Universal Description, Discovery and Integration (UDDI) kurz vorgestellt und erklärt.

Im 3. Kapitel geht es um eine Analyse, die zum Integrieren der Schnittstelle erforderlich ist. Dazu wird zunächst das GovernmentGateway noch einmal genauer vorgestellt. Anschließend wird die Erweiterung vom infoAsset Broker zum Entwickler-Portal beschrieben. Es wird die Funktionalität der Schnittstelle und zum Schluß des Kapitels werden die Zielarchitektur und ihre Komponenten festgelegt.

In Kapitel 4 wird das Design erörtert. Das betrifft sowohl das Design des Web Service Interface sowie die Web Service Architektur. Es wird besprochen, wie die Technologien miteinander verbunden und integriert werden sollen. Zusätzlich werden Technologien aufgezeigt, die zwar eine nützliche Ergänzung sein könnten, jedoch nicht umgesetzt wurden.

Das 5. Kapitel handelt die Implementierung ab. Dabei geht es um eine rückwärtige Betrachtung der Implementation. Wo sind Probleme aufgetreten, welche Unterschiede zum geplanten Design gibt es?

Das letzte Kapitel ist der Ausblick, hier werden das Projekt im Bezug auf die Zukunft beleuchtet und kommende Herausforderungen skizziert.

2 Überblick Web Services

Web Services setzen sich zunehmend als Technologie zum Aufbau von verteilten, lose gekoppelten und Service orientierten Anwendungen durch. Große und komplexe Anwendungen können aus Web Services komponiert werden, die verteilt über ein Netzwerk erreichbar sind. Doch der größte Vorteil liegt in der Überwindung von Plattform- und Programmiersprachengrenzen. So ist es für die Benutzung eines Web Services belanglos, in welcher Sprache er implementiert wurde oder auf welchen System er läuft: Ein Web Service, der beispielsweise mit Java geschrieben wurde und unter Mac OS X läuft, kann problemlos mit einem in .NET geschriebenen Web Service auf einem Windows XP Rechner kommunizieren. Möglich wird dieses durch die Verwendung von offenen Standards, die von der Industrie und Unternehmen anerkannt und unterstützt werden.

Die Idee von verteilten, lose gekoppelten und Service orientierten Anwendungen ist nicht neu. Technologien wie der Common Object Request Broker (CORBA) [31], Microsofts Distributed Component Object Model (DCOM) [26] und Suns Remote Method Invocation (RMI) [37] für Java bieten diese Fähigkeiten schon seit langer Zeit. Sie alle leiden jedoch darunter, proprietäre Protokolle zum Austausch von Botschaften zu verwenden. Insbesondere für DCOM und RMI kommt hinzu, daß der Einsatzbereich durch die Beschränkung auf die Betriebssystemfamilie (Windows) beziehungsweise die Programmiersprache (Java) eng begrenzt ist.

Durch die Extensible Markup Language (XML) [42] lassen sich Daten Plattformunabhängig beschreiben. Web Services setzen auf XML basierte, standarisierte Sprachen und Protokolle wie WSDL [48], XML-Schema [44] und das Simple Object Access Protocol (SOAP) [45] zur Beschreibung von Operationen, Daten und Datentypen. Zum Nachrichtenaustausch werden offene und bewährte Protokolle wie beispielsweise das Hypertext Transfer Protocol (HTTP) [6, Seite 497 ff.] verwendet.

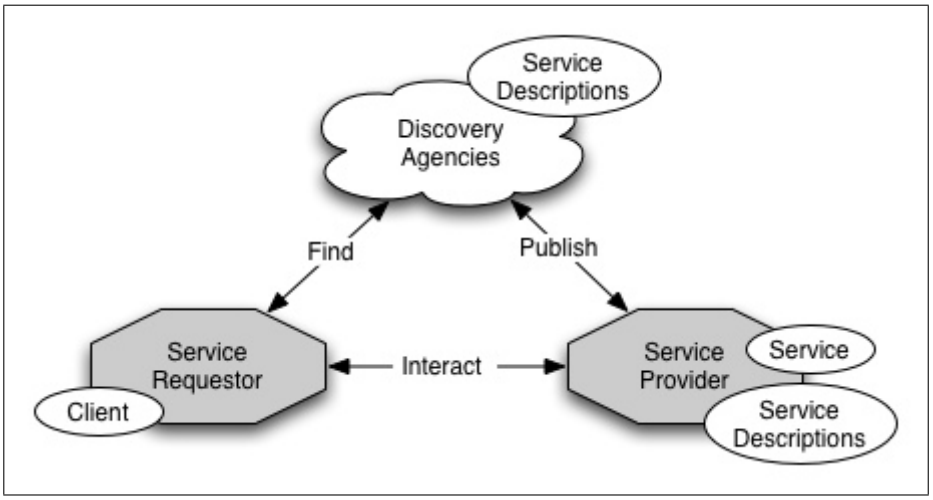

Abbildung 2.1: Web Service Dreieck (angelehnt an [33] auf Seite 163).

Bei Web Services gibt es Dienstanbieter und Dienstnutzer (siehe Abbildung 2.1). Zum Auffinden von Web Services gibt es noch zentrale Verzeichnisdienste. Bei ihnen können Web Services von

ihren Anbietern registriert werden und so Nutzern zum Beispiel über eine Suche zugänglich gemacht werden.

Im E-Business existieren verschiedene Geschäftsbeziehungen: Business to Business (B2B) und Business to Customer (B2C) sind gebräuchlich. Je nachdem was für eine Beziehung vorliegt, unterscheiden sich die direkten Nutzer. Bei B2C liegt meist eine Human to Application (H2A) Schnittstelle vor. Das heißt, daß ein Kunde direkt mit der Applikation interagiert, zum Beispiel über einen Webbrowser oder eine Client-Anwendung. Wenn es sich um B2B Beziehungen handelt, gibt es zwei mögliche Schnittstellen: Applikation to Applikation (A2A) und H2A. Bei A2A sind zwei Anwendungen miteinander verbunden. So könnte zum Beispiel die Material- und Lagersoftware eines Unternehmens mit der Bestellsoftware eines Zulieferers kommunizieren, um automatisch Material nachzubestellen wenn das Lager leer ist. Da die beiden Anwendungen auf absolut verschiedenartigen Computern mit unterschiedlicher Software laufen können, spielt hier Interoperabilität eine große Rolle. Web Services sind hier von großer Bedeutung.

2.1 Übersicht über Definitionen von Web Services

Softwareunternehmungen finden viele unterschiedliche Definitionen für Web Services, die sich im Kern ähneln:

> Web services are a type of service that can be shared by and used as components of distributed Web-based applications. – Bea [4]

> Web Services are an XML based technology that allow applications to communicate with each other, regardless of the environment, by exchanging messages in a standardized format (XML) via web interfaces (SOAP and WSDL APIs). – Mozilla [30]

> A Web Service is programmable application logic accessible using standard internet protocols. Web Services combine the best aspects of component-based development and the Web. Like components, Web Services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies which are accessed via proprietary protocols, Web Services are accessed via ubiquitous Web protocols (ex: HTTP) using universally-accepted data formats (ex: XML). – systinet [38]

> Web services is a technology that allows applications to communicate with each other in a platform- and programming language-independent manner. A Web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another Web service. A group of Web services interacting together in this manner defines a particular Web service application in a Service-Oriented Architecture (SOA). – IBM [21]

Bea, Mozilla und systinet beschränken sich auf die Beschreibung von Web Services. Im Kern beschreiben sie alle Web Services als Technologie zum Realisieren von verteilten Anwendungen. IBM geht einen Schritt weiter und führt den Begriff der Service orientierten Architekturen (SOA) gleich mit ein. IBMs Beschreibung lautet in etwa: Eine Gruppe von Web Services, die

untereinander interagieren, bilden eine Web Service Applikation in einer Service orientierten Architektur.

Meine Kurzdefinition sieht folgendermaßen aus:

> Web Services sind eine Technologie die es Anwendungen erlauben, miteinander über standard Internet Protokolle zu kommunizieren. Zur Beschreibung der Schnittstellen werden XML basierte Sprachen verwendet. Mit Web Services lassen sich verteilte, lose gekoppelte und Service orientierte Anwendungen aufbauen. Die einzelnen Web Services interagieren Plattform- und Programmiersprachen unabhängig. Wenn eine Anwendung aus Web Services aufgebaut wird, nutzt diese eine Service orientierte Architektur (SOA).

Oft wird von grobkörnigen (*coarse-grained*) Web Services gesprochen. So werden Web Services bezeichnet, die eine Menge von zusammengehörigen Funktionen ausführen. Den Gegensatz bilden feinkörnige (*fine-grained*) Web Services. Bei ihnen wird nur eine einzelne Funktion ausgeführt. Ein *coarse-grained* Web Service würde zum Beispiel bei einer Ticketbestellung die Abwicklung der gesamten Bestellung umfassen, ein *fine-grained* Web Service würde hingegen nur eine Operation des Bestellprozesses bearbeiten. [24, Seite 6 ff.], [25, Seite 49 ff.]

2.2 Architektur von Web Services

Web Services sind, was die Bindung an Transport- oder Message-Protokolle betrifft, sehr flexibel. Ich werde mich in dieser Arbeit auf die gebräuchlichste Kombination konzentrieren: HTTP mit SOAP. Es ist jedoch ausdrücklich möglich, Web Services auch mit anderen Protokollen zu verwenden.

2.2.1 Simple Object Access Protocol - SOAP

SOAP [45] ist ein auf XML [42] basierendes Messaging Protokoll, welches die Grundlage für Web Services bildet. Es bietet einen simplen und konsistenten Mechanismus zum Austausch von typsicheren XML Nachrichten zwischen Applikationen.

Im Wesentlichen bietet SOAP eine peer-to-peer Kommunikation. Eine SOAP-Message ist eine Ein-Weg Übertragung einer Nachricht von einem Sender zu einem Empfänger. Eine Applikation kann dabei sowohl als Sender und auch als Empfänger tätig werden.

Das Nachrichten-Format besteht aus drei wesentlichen Teilen. Ein Envelope Element kann ein Header Element beinhalten, muss aber auf jeden Fall ein Body Element besitzen.

Envelope Der Envelope repräsentiert die SOAP Nachricht und bildet die Container für `header` und `body`.

Header Der header wird verwendet, um dem Empfänger zusätzliche Verarbeitungs- oder Kontrollinformationen zu übermitteln. Hier können zum Beispiel Daten zur Authentifizierung, Transaktionen, Quality of Service, Dienst-Abrechnung, et cetera stehen. Das Header Element ist optional.

Body Das Body Element trägt die Nutzdaten der SOAP Nachricht, es muss das erste Subelement vom `Envelope` Element sein. Wenn ein `Header` Element vorhanden ist, muss das Body Element direkt nach ihm folgen.

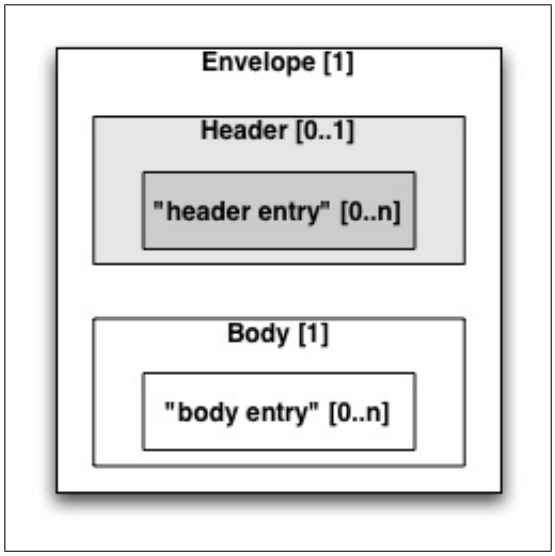

Abbildung 2.2: Aufbau einer SOAP-Nachricht (angelehnt an [33] auf Seite 77). In den eckigen Klammern stehen Kardinalitäten, in grau sind optionale Teile.

Zur Fehlerbehandlung kann im `Body` optional genau ein `Fault`-Element enthalten sein. Es trägt Felder für einen Fehler-Code, einen Fehler-Text und Details für die verarbeitende Anwendung.

SOAP bietet kein Typsystem und so können Elemente in einer SOAP Nachricht untypisiert sein. Dies kann zu Problemen führen, wenn der Empfänger einer typisierten Nachricht nicht die Typen überprüfen kann. Und selbst wenn der Sender Typinformationen mit überträgt, muss sicher gestellt werden, daß Sender und Empfänger dasselbe Verständnis über diese Typen haben. Zum Typisieren von `messages` kann das `type` Attribut verwendet werden. Dies ist allerdings keine zwingende Vorgabe.

Für die SOAP bodies gibt es zwei verschiedene Übertragungsarten:

Document Style (Auch: Dokumentorientiert) Der Inhalt des SOAP bodies ist ein frei wählbarer XML Inhalt und SOAP schränkt die Struktur der zu transportierenden XML Instanz nicht ein. Diese Übertragungsart wird häufig auch als *message-oriented style* bezeichnet. (Siehe Abbildung 2.3 auf der nächsten Seite)

RPC Style Diese Art bezeichnet einen entfernten Funktionenaufruf (*Remote Procedure Call* - RPC). Dabei wird eine Funktion mit bestimmten Parametern aufgerufen und es werden bestimmte Werte als Ergebnis zurückerwartet. Bei Verwendung des RPC Style gelten bestimmte Konventionen für die Struktur der XML Instanzen, die im SOAP body übertragen werden. (Siehe Abbildung 2.4 auf der nächsten Seite)

Losgelöst von der Übertragungsart können die übermittelten Botschaften typisiert oder nicht typisiert sein. Allgemein wird der Zusatz *literal* verwendet, um zu beschreiben, daß es sich um untypisierte Daten handelt. Für typisierte Nachrichten wird *encoded* verwendet. Daraus folgend ergeben sich vier verschiedene Kombinationen: *rpc/literal*, *rpc/encoded*, *document/literal* und *document/encoded*.

Mit SOAP Remote Procedure Calls können entfernte Methoden aufgerufen, ausgeführt und Ergebnisse zurück übertragen werden. Das Ganze ist möglich ohne die Implementierung der Funktion zu kennen. RPC gibt eine bestimmte Struktur der SOAP Nachricht vor, welche durch

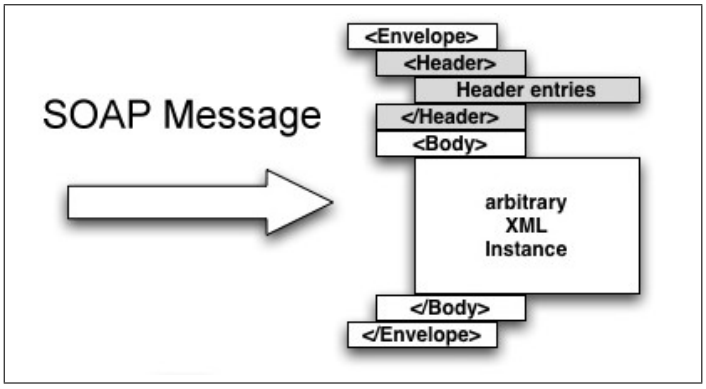

Abbildung 2.3: SOAP document style (angelehnt an [33] auf Seite 99). In grau sind optionale Teile.

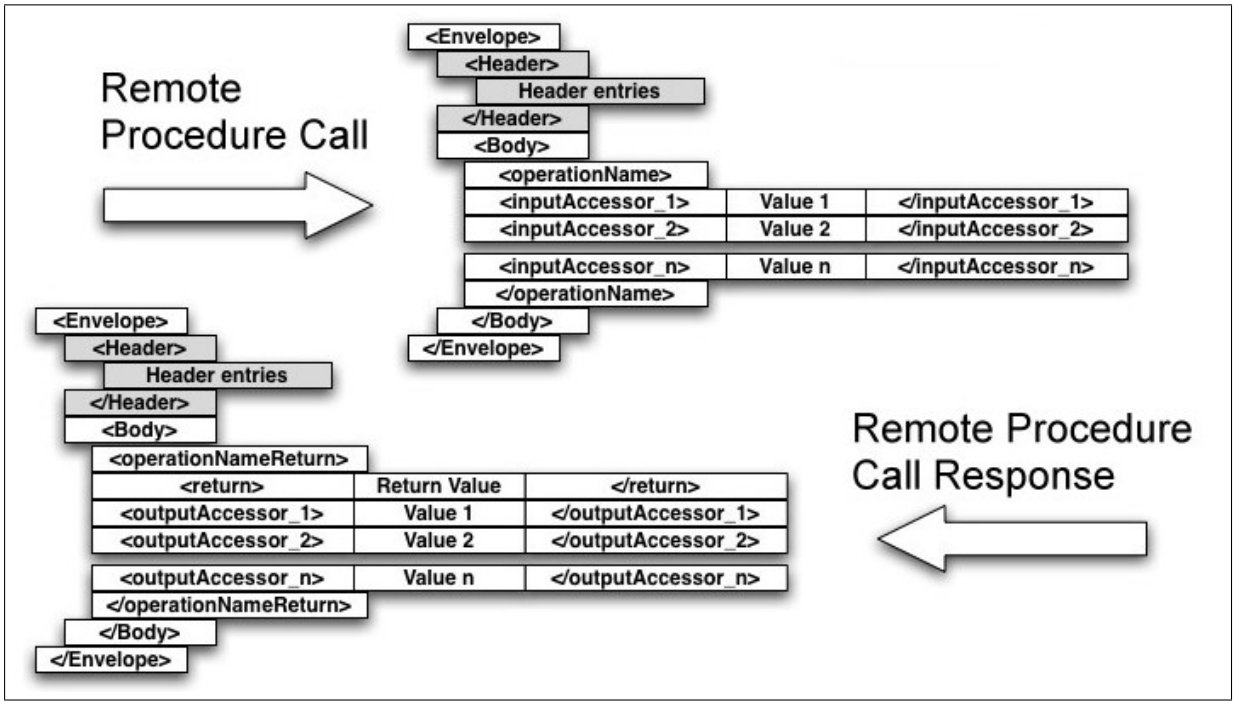

Abbildung 2.4: SOAP RPC style (angelehnt an [33] auf Seite 101). In grau sind optionale Teile.

die SOAP Runtime erzwungen wird. Die SOAP Struktur ist jedoch eng an das Interface der Implementierung gekoppelt. Wenn sich die Funktion ändert, muss die Struktur der SOAP Nachricht geändert werden.

Bei der Dokumentenorientierten Verarbeitung von Daten ist es so, daß alle Daten, die zur Bearbeitung eines Geschäftsprozesses notwendig sind, im Dokument gespeichert und übertragen werden. Die Anwendungsteile, die den Verarbeitungsprozess nachbilden, ändern nur die Dokument-Teile, die sie betreffen.

SOAP unterwirft das zu übertragene XML Dokument keinen Einschränkungen, wenn es dokumentorientiert übertragen wird. Die SOAP Runtime übergibt das ganze im body übertragene Dokument an die verarbeitende Applikation. Die gesamte Nachrichtenstruktur ist eher lose mit dem eigentlichen Dienst gekoppelt. Das liegt daran, daß die Anwendung die gesamte Verantwortung für die Verarbeitung des übertragenen Dokuments trägt.

Die SOAP Runtime bietet einen eigenen encoding Mechanismus zum Serialisieren von anwendungsspezifischen Datentypen. In heterogenen Anwendungsgebieten ist die Verwendung von SOAP encodings allerdings nicht ratsam, da es zu Interoperabilitätsproblemem kommen kann.

2.2.2 Web Service Description Language - WSDL

Web Services werden in der vom World Wide Web Consortium (W3C) [41] standarisierten Web Service Description Language (WSDL) [48] beschrieben. Dabei handelt es sich bei WSDL um ein XML Vokabular zum Beschreiben von Funktionen, wie diese kommuniziert werden und wo sie zugänglich sind. WSDL bietet strukturierte Mechanismen zum Beschreiben von Operationen, das Format in dem die Nachrichten ausgetauscht und welche Protokolle unterstützt werden sowie zum Beschreiben der Zugangspunkte zu einer Instanz eines Web Services.

Die Beschreibung eines Web Services in WSDL lässt sich inhaltlich in zwei grobe Blöcke unterteilen [33, Seite 104]:

Abstraktes Interface Enthält die Beschreibung des Interface Layouts mit den unterstützten Operationen, den Parametern für die Operationen und den abstrakten Datentypen. Es enthält jedoch keinerlei Angaben über konkrete Datenstrukturen, verwendete Netzwerk-Protokolle oder Netzwerkadressen.

Konkrete Implementation Die Implementierung bindet das abstrakte Interface an ein konkretes Netzwerkprotokoll mit konkreter Adresse und konkreten Datenstrukturen.

Ein abstraktes Interface kann an viele verschiedenartige, konkrete Implementationen gebunden werden. Auf diese Art kann ein Client die Implementation wählen, die am besten zu seinen technischen Ressourcen passt.

Die Beschreibung des abstrakten Interfaces besteht aus den folgenden Teilen:

portType Ein portType ist eine benannte Menge von `operation`s, die der Web Service unterstützt.

operation Operationen beschreiben abstrakt einen Funktionsaufruf. Sie können eine input, output oder fault `message` enthalten.

message Eine message ist eine abstrakte, typisierte Beschreibung der Daten die ausgetauscht werden. Messages bestehen aus `parts`.

part Jeder part ist mit einem Datentyp verbunden.

types Das types Element ist ein Container für Definitionen von Datentypen. WSDL ist hier nicht auf XML-Schema festgelegt, allerdings ist dieses System jenes, welches bevorzugt eingesetzt wird.

Der zentrale Punkt des Interfaces ist der `portType`. Dieser wird in der Service Implementation an das `binding` Element gebunden, welches dort das zentrale Element bildet.

Die Service Implementation in WSDL besteht aus folgenden Teilen:

binding Das binding legt ein konkretes Protokoll fest und spezifiziert ein Datenformat für einen `portType`.

port Der port beschreibt einen einzelnen, konkreten Endpunkt, zusammengesetzt aus einer Netzwerkadresse und einem `binding`.

service Ein service ist eine Menge von zusammengehörigen `ports`. Diese `ports` können sich den gleichen `portType` teilen und werden von unterschiedlichen `bindings` oder Netzwerkadressen verwendet. Im service stehen also eine Sammlung von zusammengehörigen Endpunkten.

WSDL kennt vier Übertragungsprimitiven. Diese werden über die Reihenfolge der Elemente innerhalb des `operation` Elements im `portType` definiert.

One-Way Der Endpunkt empfängt eine Nachricht

Request-Response Der Endpunkt empfängt eine Nachricht und antwortet entsprechend.

Solicit-Response Der Endpunkt sendet eine Nachricht und empfängt eine korrelierte Nachricht.

Notification Der Endpunkt sendet eine Nachricht.

Auch wenn request-response und solicit-response im WSDL-Dokument modelliert werden können, entscheidet das konkrete Binding wie die Nachrichten tatsächlich übertragen werden. So kann zum Beispiel eine request-response Nachricht als zwei einzelne anstelle von einer Übertragung realisiert sein.

Ein WSDL-Dokument kann als Interface-Vertrag verstanden werden. In ihm stehen die Konditionen mit denen ein Client mit dem Web Service kommunizieren kann. Er besteht aus zwei Teilen, einem abstrakten Service-Interface und bindings die ein konkretes Protokoll und Netzwerkadressen an den Dienst binden.

Um Web Services in einer dynamischen Umwelt zu finden, ist es unzureichend, nur eine technische Beschreibung des Services zu haben. Zusätzlich ist die Beschreibung in WSDL für den Menschen unverständlich und auch unzureichend, um die genaue Bedeutung eines Web Services zu erfahren. Ganz abgesehen davon, daß die Frage des Auffindens von Web Services bisher völlig ungeklärt geblieben ist. Im nächsten Teil werde ich deshalb den Registrierungsdienst UDDI erläutern.

2.2.3 Universal Description, Discovery and Integration - UDDI

UDDI [32] kann als (zentrale) Suchmaschine für Web Services verstanden werden. Es ist eine Verzeichnisplattform, auf der Benutzer nach Web Services suchen und sich über angebotene Web Services informieren können. Dazu gehören technische Aspekte wie auch allgemeine Informationen über den Anbieter und den Dienst. Nur ein Benutzer, der in der Lage ist, ein bestimmtes Web Services Interface zu erfüllen, ist auch in der Lage den Service zu nutzen. Natürlich wird ein Benutzer auch eine Vorauswahl treffen wollen, um eine Menge von Web Services zu finden, die seinen Anforderungen genügen. Anforderungen können neben der Funktionalität eines Dienstes zum Beispiel auch dessen geografische Nähe sein oder die Reputation eines Dienstanbieters.

In einem UDDI Verzeichnis können folgende Informationen gefunden werden:

- Informationen über die Organisationen oder die Unternehmen, die Web Services veröffentlichen.
- Beschreibungen der Web Services, die angeboten werden.
- Informationen zu den technischen Interfaces dieser Web Services.

Das UDDI Verzeichnis Daten-Modell besteht aus fünf Hauptdatentypen. Sie alle sind durch XML-Schema beschrieben und hierarchisch zueinander aufgebaut. Es werden zwei Arten von Informationen angeboten: Zum Einen geschäftsbezogene Daten und zum Anderen technische Informationen über Web Service bindings.

busniess entity Die Geschäftsdaten enthalten Informationen über den Dienstanbieter. Darunter sind Daten wie Name, Kontakt oder eine Kategoriesierung des Unternehmens. Die business entity ist das höchstrangigste Object in der Hierarchie.

business service Ein Dienstanbieter, identifiziert durch durch die `business entity`, kann eine Vielzahl von Web Services anbieten. Diese können zu Gruppen von zusammengehörigen Web Services zusammengefasst werden. Jede Aggregation ist beschrieben durch ein business service, welches allgemeine Informationen enthält. Unter anderem sind folgende Daten aufgenommen: Ein Name für die Aggregation und eine Beschreibung oder eine Kategorisierung der Dienste. Alle hier angegebenen Informationen sind nicht technischer Natur.

binding template Diese Datenstruktur nimmt Informationen über die Adresse eines Dienst Endpunktes auf. Ausserdem kann es Daten beinhalten zur Implementierung des Dienstes oder Hinweise, wo weitere Informationen gefunden werden können. Ein `business service` enthält binding templates. Eines für jeden Web Service.

tModel Das tModel oder technical Model enthält die Web Service Interface Informationen. Es wird festgehalten, an welche Spezifikationen und Konventionen sich ein Web Service hält oder nicht. Ein Client, der ohne diese Informationen versucht, mit dem Web Service zu kommunizieren, wird wahrscheinlich Probleme bekommen. Von daher sind diese Informationen wichtig.

publisher assertion Große Unternehmen, die aus vielen Tochterunternehmen bestehen, können möglicherweise nicht adäquat durch ein einziges `business entity` beschrieben werden. Die Informationen zum Beschreiben könnten zu unterschiedlich

und zu umfangreich sein. Das publisher assertion Datenobjekt enthält Informationen über die Beziehungen zwischen verschiedenen `business entities`.

Während WSDL sich einzig auf die Technischen Aspekte von Web Services konzentriert, bietet UDDI auch nicht technische Informationen an. UDDI dient Benutzern als Hilfe Web Services zu finden, die sie zuvor nicht kannten. Dabei bietet UDDI eine Kategorisierung von technischen und nicht technischen Eigenschaften von Web Services, sowie weiterführende Informationen. Es ist jedoch sehr unwahrscheinlich, daß alle benötigten Informationen ausreichend und vollständig vorhanden sind oder überhaupt im UDDI Verzeichnis abgebildet werden. Für Anwendungen bleibt ein dynamisches Auffinden und Ausführen von Web Service weiterhin unerreichbar. In Unternehmen oder einem anderen begrenzten Umfeld können die UDDI Dienste allerdings sehr hilfreich sein. Angelehnt an [33, Seite 151 ff.].

2.3 Vorzüge von Web Services

Für Web Services spricht eine Vielzahl von Argumenten. Die verwendeten Protokolle und Beschreibungssprachen sind offen, weit verbreitet und genießen eine große Akzeptanz.

Bis zu den Sicherheitsaspekten sind die folgenden Punkte an [24, Seite 6 ff.] und [25, Seite 49 ff.] angelehnt.

Wie bereits angesprochen, sind Web Services mit einem Dialekt der plattform unabhängigen Sprache XML beschrieben. Programmiersprachen-, Betriebssystem- und Plattformgrenzen werden durch diese Eigenschaft von XML transparent. Die Implementierungen der Web Services sind so einfach austauschbar und damit sehr flexibel.

Durch die Verwendung von Standardprotokollen wie SOAP und HTTP wird zusätzlich die lose Kopplung von Web Services unterstützt. Die Protokolle sind offen, wohlbekannt und die Implementierungen der Internet Protokolle auf den üblichen Betriebssystemen längst Standard. Für die Anderen sind viele (freie) Implementierungen verfügbar. Durch das Vorhandensein dieser Protokolle werden die Abhängigkeiten zu proprietären Systemen und deren Bibliotheken vermieden. Dies führt letztendlich zu einer einfachen Wiederverwendbarkeit der Web Services.

Aufgrund der losen Koppelung tendieren Web Service basierte Anwendungen dazu, gut zu skalieren. Das liegt an den geringeren Abhängigkeiten, verglichen mit eng gekoppelten Anwendungen. Die Web Services in Web Service orientierten Architekturen neigen dazu *coarse-grained*, Dokumentorientiert und asynchron zu sein. Ein asynchroner Service verarbeitet seine Daten ohne daß der Client gezwungen wird, auf den Service zu warten. Der Kommunikationsaufwand, den ein Client benötigt, um mit einem *coarse-grained*, asynchronen Web Service zu interagieren ist relativ gering. Dokumentenorientierte Web Services tragen positiv zur Skalierbarkeit bei, da hier die gesamten Daten in einem Dokument ausgetauscht werden und nicht viele granulare Daten übertragen werden müssen. Diese Verringerung des Protokolloverheads (Verhältnis zwischen Protokoll- und Nutzdaten) wirkt sich positiv auf die Netzwerklast aus.

Bei vielen verteilten Anwendungen gibt es Probleme, wenn diese über das Internet funktionieren sollen. Dies liegt hauptsächlich daran, daß Firewalls die meisten Protokolle und Ports bis auf wenige blockieren. Eine der wenigen Ausnahmen ist HTTP mit seinem Standardport 80.

Wenn als Transportschicht HTTP gewählt wird, ist es für Web Services einfach möglich, auch über die Grenzen von Firewalls hinweg zu kommunizieren. Der Port für HTTP ist wählbar, aber fest und der Standardport wohl bekannt. In der Firewall muss nur ein einziger Port freigegeben sein, da sich dieser nicht ändert.

Andere Systeme (zum Beispiel DCOM) verwenden dynamisch gewählte Ports. Hier wäre es in der Firewall nötig, viele Ports (*Port-Ranges*) freizugeben. Das ist ein Sicherheitsrisiko, welches nicht gern eingegangen wird.

Ein spezieller Vorzug von HTTP und SOAP gegenüber Microsofts DCOM ist, daß HTTP und SOAP einzig als Service Schicht für die Web Services fungieren. DCOM hingegen ist Microsofts Haupt-Protokoll für *inter-application* Kommunikation. Es wird nicht nur von Programmen verwendet, von denen man erwartet, daß sie als Server fungieren. DCOM wird zum Beispiel auch für eine Vielzahl von Desktop Kommunikationen verwendet. Eine Sicherheitslücke in diesem System kann nicht nur den eigentlichen Web Service gefährden, sondern die Sicherheit des gesamten Computers. [8]

3 Anforderungen an die Web Service Schnittstelle

In diesem Kapitel soll erarbeitet werden, was benötigt, wird um das Entwickler-Portal mit einer Web Service Schnittstelle zu erweitern. Welche Funktionen des Portals müssen auf den Web Service abgebildet werden und welche nicht? Dazu wird zunächst noch einmal der Broker vorgestellt. Danach werden die Erweiterungen und Modifikationen erklärt, die den Broker zum Entwickler-Portal machen. Schließlich muss eine Architektur gefunden werden, mit der sich die Anforderungen umsetzen lassen.

3.1 Vorstellung des GovernmentGateways

Das GovernmentGateway der Freien und Hansestadt Hamburg ist der zentrale Zugangspunkt zu Funktionen des E-Governments. Es dient verschiedenen Benutzergruppen als webbasierter Zugang zu den Fachverfahren. Je nach Benutzergruppe kann der Zugang unterschiedlich sein. Im Internet ist das GovernmentGateway über die Webseite `http://gateway.hamburg.de` zu erreichen. Nach einer Registrierung können Bürger bestimmte Fachverfahren, also Funktionen der Behörden, nutzen. Andere Funktionen benötigen eine höhere Sicherheitsstufe, für die sich Bürger zusätzlich noch bei einem Hamburger Kundenzentrum ausweisen müssen. Es gibt gebührenpflichtig und kostenlose Fachverfahren. Für Unternehmen läuft der Vorgang analog zu dem der Bürger. Fachverfahren, die nicht öffentlich sind, werden allerdings explizit freigegeben. Die Freigabe erfolgt für einen „Masteruser“. Ein Masteruser ist ein Benutzer, mit dem weitere Benutzer mit den selben Berechtigungen angelegt und verwaltet werden können. Die Unternehmen können so weitere Benutzer für das Fachverfahren mit dem Masteruser erstellen. Für andere Behörden ist das Vorgehen ähnlich wie für Firmen, nur daß diese für die Dienste nicht bezahlen müssen. Ohne Authentifizierung sind nur öffentliche Fachverfahren zugänglich (zum Beispiel die Abfrage von Wasserstandspegeln).

3.1.1 Die Architektur des GovernemntGateway

Viele der als E-Government angebotenen Dienste existieren bereits als Fachverfahrens-Anwendungen bei Dataport. Sie liegen allerdings auf unterschiedlichen Servern und sind unterschiedlich implementiert. Um diese Fachverfahren weiter zu verwenden, nutzt Dataport Web Services, um die Heterogenität zu überbrücken. Auf diese Art können die Fachverfahren unverändert bleiben und müssen nur mit einem Adapter für die Web Service Schnittstelle erweitert werden.

Das GovernmentGateway (siehe Abbildung 3.1 auf der nächsten Seite) ist in Schichten aufgebaut. Auf die Präsentations-Schicht (links) kann über das Internet mit unterschiedlichen Endgeräten zugegriffen werden. Benutzer sind Bürger, Unternehmen und andere Behörden, die das GovernmentGateway nutzen wollen. In der Demilitarisierten Zone (DMZ) [23, Seite 493 ff.] - A zwischen der ersten und der zweiten Firewall befinden sich die Web Server, welche für die Generierung und Ausgabe der Internetseiten, also der Präsentation, verantwortlich sind. Die Applikationsschicht und Datenbankschicht liegen in der DMZ - B, zwischen der zweiten und

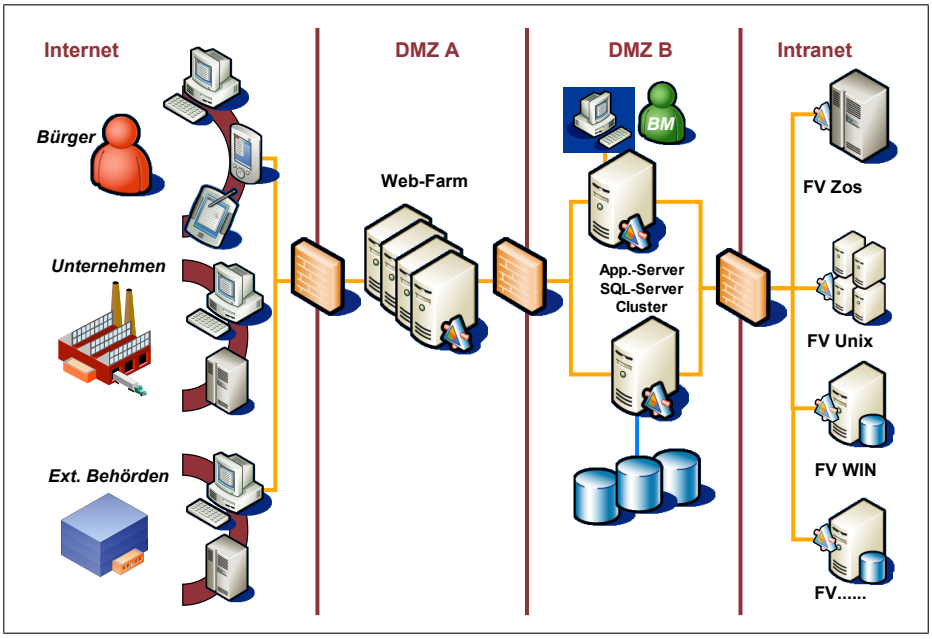

Abbildung 3.1: Architektur des GovernmentGateways [12]

der dritten Firewall. Hier liegt die Geschäftslogik für das Gateway, Schnittstellen, Fachverfahren und Basisfunktionen des Gateways, sowie der Datenbank-Cluster. Hinter der dritten Firewall liegen *backend* Fachverfahrens-Anwendungen.

Der Vorteil dieser Architektur ist, daß die Benutzer des GovernmentGateways alle Fachverfahren absolut transparent über eine Web Seite mit einem Design nutzen können. Dabei ist es ganz unabhängig, zu welchem Bereich der öffentlichen Verwaltung die Fachverfahren gehören.

3.2 infoAsset Broker – Entwickler-Portal

Der infoAsset Broker [17] ist eine Standardsoftware für ein Internet Portal für das Wissensmanagement in mittelständischen und großen Unternehmen sowie Behörden. Der Broker erschließt und vernetzt dazu strukturierte und unstrukturierte Informationen aus heterogenen Quellen und stellt sie Mitarbeitern und Teams über eine intuitive Web Oberfläche zur Verfügung.

Damit große Informationsbestände mit dem Broker zu beherrschen sind, müssen die Informationsobjekte thematisch kategorisiert werden. Mit dem Broker kann solch eine Taxonomie erstellt und gepflegt werden. Die Begriffe der Taxonomie, die als Konzept bezeichnet werden, sind in der Wissendslandkarte strukturiert grafisch dargestellt. Zusätzlich kann über die Konzepte in der Wissenslandkarte gezielt zu den Informationsobjekten hin navigiert werden.

Innerhalb des Brokers werden alle Informationsobjekte wie zum Beispiel Dokumente, Bilder, Personen, Gruppen oder Projekte einheitlich als *Assets* hinterlegt. Dadurch ist der Broker in der Lage, seine generischen Funktionen gleichsam auf alle Informationsobjekte anzuwenden. Der tatsächliche Typ eines Informationsobjekts ist in dem Falle unerheblich. Zu den Broker-Funktionen, die auf diese Art genutzt werden können, zählen unter anderem die Suche, die Verknüpfung von Informationsobjekten, ihre Klassifizierung und event-basierte Benachrichtigungen.

Durch die Verwendung von Broker Templates und Handlern wird eine klare Trennung von Darstellung, Inhalt und Ablauflogik vollzogen. Neben der Vereinheitlichung des Layouts eröffnet

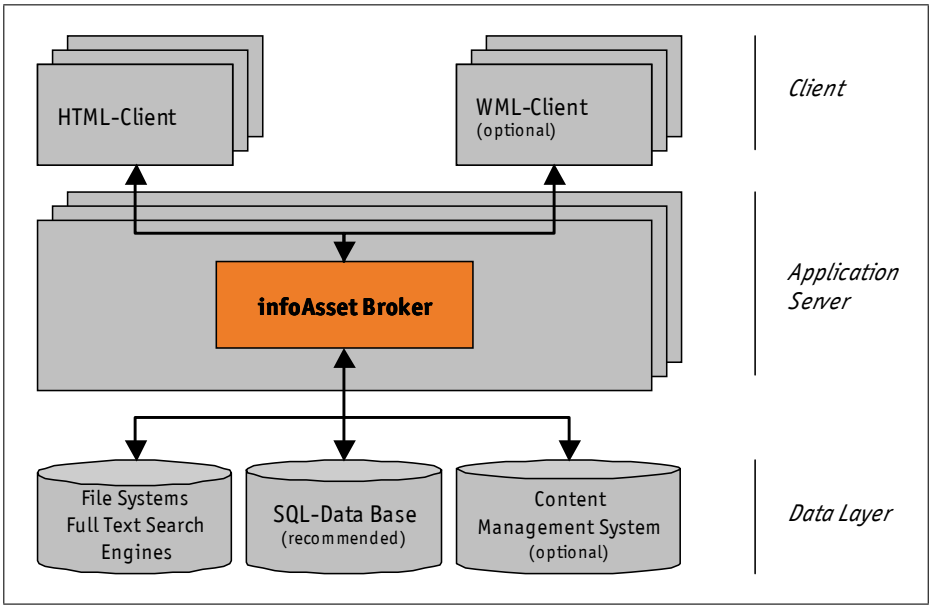

Abbildung 3.2: Architektur des infoAsset Brokers (angelehnt an [18, Folie 5])

sich so die Möglichkeit, das Layout kunden- und gerätespezifisch anzupassen.

Die Schnittstellen für den Broker sind über die Schichten der Architektur hinweg offen gelegt. Hierdurch kann das System in vielen Bereichen um Dienste, Werkzeuge und Content-Manager erweitert und so an verschiedene Anforderungen angepasst werden.

Der Broker ist vollständig in Java 2 geschrieben und als Client-Server Architektur realisiert. Die Kommunikation zwischen Broker und Client erfolgt ausschließlich über das HTTP-Protokoll. Dadurch kann der Broker auch über Firewalls hinweg einfach eingesetzt werden. [18]

3.2.1 Erweiterungen für das Entwickler-Portal

Für die Firma Dataport soll ein System zur Dokumentverwaltung und zum Dokumentmanagement realisiert werden. Dieses soll zunächst für Entwickler des GovernmentGateways und später auch für weitere Entwicklergruppen genutzt werden. Das Entwickler-Portal soll alle beteiligten Entwickler mit den für sie notwendigen Informationen versorgen und hierbei auch die Schutzwürdigkeit der Daten berücksichtigen: Nicht jeder darf alles sehen.

Der Zugriff auf das System soll über zwei Wege möglich sein: Zum Einen lesend durch das GovernmentGateway und zum Anderen lesend und schreibend durch eine Anmeldung am Web-Portal des Entwickler-Portals selbst. Beim Zugriff über das Gateway muss die Authentifizierung der Benutzer über das Gateway erfolgen. Hierbei müssen auch die vorhandenen Benutzerrollen übertragen und beachtet werden. Das Entwickler-Portal soll als *backend* Anwendung ähnlich den Fachverfahren aufgestellt werden. Da alle Fachverfahren über Web Service Schnittstellen mit dem Gateway kommunizieren, ist diese Schnittstelle für den Broker zwingend erforderlich, damit Daten aus dem Portal gelesen werden können, um sie im Gateway darzustellen.

Damit der Broker die Anforderungen von Dataport erfüllen kann, mussten unter anderem Dokument-Modell sowie das Berechtigungskonzept des Brokers angepasst werden.

Eine genaue Beschreibung zu den folgenden Teilen ist in der Bachelor Arbeit von Jun Zhang [14] zu finden.

3.2.1.1 Entwickler-Portal Analyse

Zu Beginn wurden mehrere Benutzergruppen identifiziert, die die Anwendung nutzen:

- Basis-Architektur-Entwickler, die Gateway-Fachverfahren-Entwicklern zum Beispiel Binär-Dateien und Dokumentationen zur Verfügung stellen.
- Gateway-Fachverfahren-Entwickler, die anderen Gateway-Fachverfahren-Entwicklern beispielsweise Know-How oder Templates anbieten.
- Standarisierer für Webservices, die Standards / Styles veröffentlichen. Nutzer sind Entwickler, die Webservices schreiben.
- Webservice-Entwickler, die Dokumentationen, Know-how und Code zu Webservices zur Verfügung stellen. Nutzer sind hier interne Entwickler, die Webservices selber erstellen sollen.
- Webservice-Provider, stellen Webservice-Schnittstellen zur Verfügung. Mit diesen können Entwickler Anwendungen schreiben, die Webservices nutzen. Ein Teil dieser Nutzergruppe sind Kunden-Entwickler von Firmen. Diese benutzen die Webservices von Behörden, um Daten zur weiteren Verarbeitung zu beziehen.

Diese Gruppen lassen sich generalisieren auf Dokument-Anbieter und Dokument-Bezieher. Ihre inhaltliche Trennung lässt sich durch Rollen modellieren. Dadurch bleibt die Struktur flach und hierdurch einfach erweiterbar und pflegbar.

Durch die Generalisierung reduziert sich die Anzahl der Use-Cases (siehe Abbildung 3.3) auf ein Anwendungsfall-Szenario. Dieser ist in der Anwendungsfall-Beschreibung erläutert. Dateien, Binaries, Beschreibungen etc. werden zunächst abstrakt als Dokumente bezeichnet.

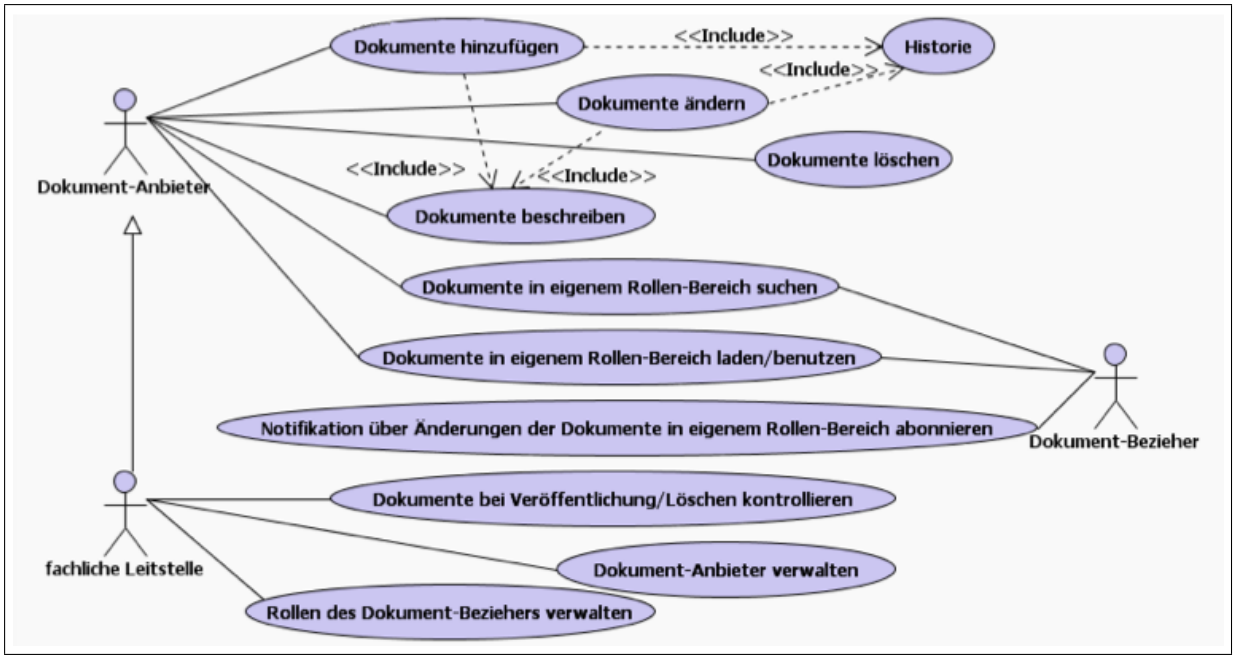

Abbildung 3.3: Use Cases des Entwickler-Portals

Beim Einpflegen neuer Dokumente muss der Benutzer über Rollen festlegen, welche Benutzer (Rolleninhaber) die Dokumente sehen und benutzen dürfen. Das Publizieren von Dokumenten im System muss von der fachlichen Leitstelle bestätigt werden.

Dokumente müssen mit Metadaten versehen werden. Dazu können sie mit Schlagworten versehen werden. Die Dokumente werden (soweit möglich) zur Volltextsuche indiziert und sie können mit anderen Dokumenten verknüpft werden. Mehrere Dokumente können dabei in Verzeichnissen zusammengefasst werden.

Ein Benutzer, der ein Dokument in das System eingepflegt hat, ist dessen Eigentümer. Er kann es später ändern oder neue Dokumente und Metadaten hinzufügen. Das Löschen ist nur mit Zustimmung der fachlichen Leitstelle möglich.

Beim Ändern oder Hinzufügen von Dokumenten bleiben die alten Dokumente als Historie erhalten. Die Metadaten werden übernommen. Alte Dokumente werden mit einem Bezeichner „Historisch“ gekennzeichnet und mit einer Versionsnummer nummeriert. Das aktuellste Dokument (wird mit dem Bezeichner „Aktuell“ angezeigt) wird immer zuerst angezeigt.

Benutzer können sich über Änderungen an Dokumenten benachrichtigen lassen. Sie erhalten dann, wenn die Dokumente geändert werden, eine Benachrichtigung per E-Mail. Dazu müssen die Benutzer im Notification-Dienst das Dokument abonnieren.

Dokument-Anbieter kommen über das Backend. Sie werden von der fachlichen Leitstelle in das System eingepflegt. Sie weist diesen Benutzern auch die Rollen zu. Dokument-Bezieher kommen über das Gateway. Sie bringen ihre Registrierung mit. Die fachliche Leitstelle weist den Dokument-Beziehern ihre Rollen zu.

Dokumente wurden zum Einen durch Attribute erweitert, die nötig geworden sind, zum Anderen wurde ein neuer Workflow zum Abbilden der Dokument Stati eingeführt.

3.2.1.2 Erweitertes Dokumentdatenmodell im Entwickler-Portal

Informationen werden im Broker in *Assets* gehalten. Das für das Projekt entscheidentste *Asset* ist das *Document Asset*. Document hat dabei eine generellere Bedeutung, es können Web Services darin abgelegt sein wie auch XML-Dokumente oder Binär-Dateien oder Word-Dokumente. Das *Document Asset* enthält bereits diverse Attribute, die zum Speichern von Daten verwendet werden (im Folgenden werden nur relevante Attribte wiedergegeben):

- Name, ein String, der frei gewählt werden kann (entsprechend den Benamungsregeln des Brokers) und teilweise zur Identification verwendet wird
- Title, ein String, der den Dokument-Titel enthält, falls einer vorhanden ist.
- DocumentKind, ein String, aus einer vordefinierten Domäne, der definiert, um was für einen Typ von Dokument es sich bei dem *Asset* handelt.
- DocumentStatus, ein String-Wert, der den Status des Dokuments repräsentiert.
- Comment, ein String für Notizen zum Dokument.
- Concept, dieses Attribut referenziert *Concept Assets*, welche einen Teil der Wissenslandkarte aufbauen.
- ExpiryDate, ein Datum, das festlegt, wann ein Dokument abläuft, falls der Wert gegeben ist.
- Directory, eine Referenz auf ein *Directory Asset*, in dem sich das Dokument befindet.
- Filename, ein String für den Namen der Datei, wenn eine Datei an das Dokument angehängt ist.

- LocalFilename, ein String, welcher den vollen relativen Pfad im Broker zu einer dem Dokument angehängten Datei enthält.
- Version, ein String, der die Versionsnummer des Dokuments enthält.
- ParentVersion, eine Referenz zu einem anderen *Document Asset*, welches der Vater zu diesem Dokument ist.
- SuccessorVersion, eine Referenz zu einem *Document Asset*, welches die Nachfolger-Version zu diesem Dokument ist.
- Creator, eine Referenz zu einem *Person Asset*, welches das Dokument erstellt hat.
- LastEdior, eine Referenz zu dem *Person Asset* der Person, die das Dokument zuletzt bearbeitet hat.
- LastEditedDate, ein Datum welches das Datum der letzten Änderung enthält.
- LockedBy, eine Referenz zu einem *Person Asset* welches das Dokument gerade bearbeitet und es gesperrt hat.

Für das Entwickler-Portal ist das *Document Asset* erweitert worden. Um zeitliche Abläufe abzubilden, gibt es weitere Datumsfelder:

- GueltigAb, ab wann ein Dokument gültig ist.
- GueltigBis, bis wann ein Dokument gültig ist.
- FreigegebenDate, Datum an welchem das Dokument von der Fachlichen Leitstelle freigegeben wurde.
- veroeffentlichtDate enthält das Datum, an welchem das Dokument veröffentlicht werden soll.

Um Vernetzungen und Verknüpfungen abzubilden, gibt es zwei Felder zum Referenzieren von anderen Dokumenten. Die Referenzen sind gerichtet und bidirektional.

- VerwendeteDokumente referenziert alle *Document Assets* die dieses Dokument verwendet.
- WirdVerwendetVonDokumenten referenziert alle *Document Assets* die dieses Dokument benutzen.

Wie bereits erwähnt sind die Verknüpfungen gerichtet. Das bedeutet, wenn ein Dokument A eine Verknüpfung „VerwendeteDokumente“zu Dokument B hat, so wird Dokument A bei Dokument B unter „WirdVerwendetVonDokumenten“auftauchen.

Es gibt Referenzen zu *Person Assets*, die besondere Wissensträger mit dem Dokument in Verbindung bringen:

- Autor ist der Ersteller eines Dokuments. Bei angehängten Dateien zum Beispiel der Ersteller des Dokument-Anhangs.
- Eigentuemer ist die Person, die das Dokument in das Portal einpflegt.
- AnsprechpartnerFachlich ist ein Wissensträger mit besonderem fachlichen Wissen (zum Beispiel zu Vorschriften oder Gesetzen eines Fachverfahrens).

- AnsprechpartnerTechnisch ist ein Wissensträger mit besonderem technischen Wissen (zum Beispiel zur programmiertechnischen Umsetzung eines Programmteils).

Dokumente können *Concept Assets* referenzieren, die benutzt werden, um ein Dokument zu kategorisieren. Aufgrund der Anforderungen ist es im Entwickler-Portal notwendig, ein Dokument mehr als nur eindimensional zu kategoriesieren. Zu dem existierenden *Concept* kommen noch drei Referenzen hinzu.

- Kategorie dient zum Kategoriesieren des Dokuments anhand von gegebenen Kategorien.
- FachlicherKontext enthält Verknüpfungen zu gegebenen fachlichen Kontext Konzepten in dessen Zusammenhang es einzuordnen ist.
- RechtlicheGrundlage das selbe wie oben, jedoch für rechtliche Grundlagen eines Fachverfahrens.

Ein weiteres wichtiges Attribut ist eine Referenz zu einem *Directory Asset*. Dieses Verzeichnis wird vom Entwickler-Portal dazu benutzt, um historische Versionen eines Dokuments dort abzulegen. Jedes Dokument erhält sein eigenes Historienverzeichnis, in dem alle seine Versionen abgelegt werden. Mit Hilfe der Referenz wird auf das Verzeichnis für die historischen Dokumente eines Dokuments verwiesen.

Weitere Attribute sind:

- Zielgruppe, referenziert *Group Asset*s, mit denen festgelegt werden kann, welche Benutzergruppen Zugang zu dem Dokument erhalten.
- Zweck, ein String zum Beschreiben einer Intention für das Dokument.
- DiffPreVersion, ein String, der zum Beschreiben der Unterschiede zur vorherigen Version verwendet werden soll.
- GrundFuerUngueltigkeit, ein String, in dem erklärt werden soll, warum ein Dokument nur bis zu einem gewissen Datum Gültigkeit hat.
- URL, ein String für eine URL zu einem Dokument.
- TestURL, ein String für eine Test-URL zu einem Dokument.
- Verfuegbarkeit, ein String mit dem die Verfügbarkeit des Dokuments beschrieben werden kann.
- HistoryStatus, ein boolean Wert, der beschreibt, ob das Dokument in der Historie liegt.

3.2.1.3 Erweitertes Dokumentzustandsmodell im Entwickler-Portal

Das State-Diagram (siehe Abbildung 3.4 auf der nächsten Seite) Beschreibt die Zustände, die ein Dokument während seiner Existenz annehmen kann. Wenn es angelegt wird, ist es zunächst „unveröffentlicht“. Während dieser Zeit kann der Eigentümer des Dokuments es ständig und frei ändern und auch löschen. Hat das Dokument schließlich den Grad erlangt, daß es veröffentlicht werden soll, so kann eine Veröffentlichung bei der Fachlichen Leitstelle beantragt werden. Damit geht das Dokument in den Zustand „veröffentlichung beantragt“ über.

Sollte der Antrag abgelehnt werden, gelangt das Dokument zurück in den Zustand „unverföffentlicht“, andernfalls wird es zu einem „wartenden“ oder „öffentlichen“ Dokument. Je nachdem, ob

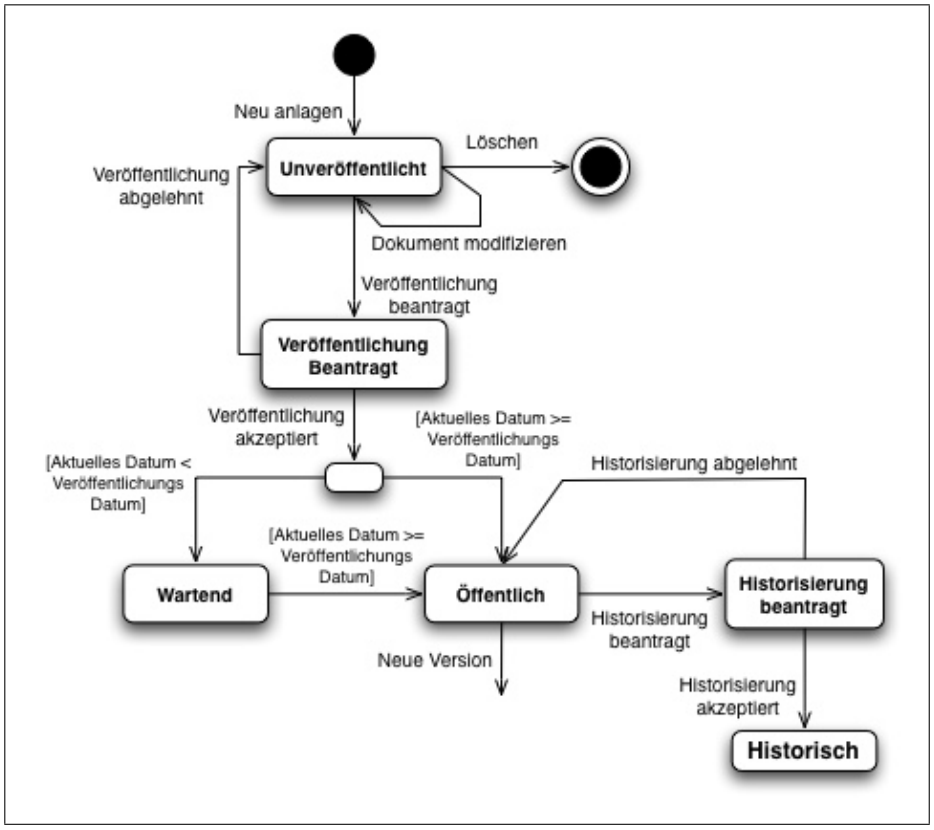

Abbildung 3.4: Auszug aus dem Status-Diagramm der Dokumente

das Veröffentlichungsdatum, welches in den Dokumentdaten eingegeben ist, bereits erreicht ist oder nicht. Wenn ein Dokument erst einmal „öffentlich“ ist, kann es nicht mehr gelöscht werden. Damit sollen Dokumentveränderungen lückenlos dokumentiert werden. Jetzt kann entweder eine neue Version des Dokuments erstellt oder es historisch gemacht werden.

Wenn es historisch gemacht werden soll, so muss dieses wieder bei der Fachlichen Leitstelle beantragt werden, damit geht das Dokument zu dem Zustand „Historisierung beantragt“ über. Wenn die Fachliche Leitstelle den Vorgang ablehnt, wird das Dokument auf „öffentlich“ zurückgesetzt, andernfalls bekommt es den Status „historisch“. Historische Dokumente werden in einem extra Historien-Verzeichnis gesammelt, damit sie nicht die Sicht auf aktuelle Dokumente behindern. Dort sind sie chronologisch nach Versionsnummer abgelegt. Für die Suche kann explizit angegeben werden, ob historische Daten mit durchsucht werden sollen.

Zu einem „öffentlichen“ Dokument kann eine neue Version erstellt werden. Dabei existieren zunächst die neue Version sowie die alte Version parallel. Die neue Version bleibt zunächst im Zustand „unveröffentlicht“, bis die Fachliche Leitstelle der Veröffentlichung zugestimmt hat. Jetzt hängen die Zustandsübergänge von der Zeit, genauer dem Veröffentlichungsdatum des neuen Dokuments, ab. Wenn das Veröffentlichungsdatum erreicht ist, wird die neue Version des Dokuments „öffentlich“ und die alte Version wird „historisch“. Bei einer Ablehnung des Antrags auf Veröffentlichung bleibt die neue Version des Dokuments im „unveröffentlicht“ Zustand.

Die Fachliche Leitstelle hat Sonderrechte, mit der sie Teile der Metadaten von Dokumenten ändern darf. Diese dienen dazu, das Dokument innerhalb des Entwickler-Portals an richtiger Stelle zu platzieren.

3.2.1.4 Erweitertes Berechtigungskonzept im Entwickler-Portal

Das Berechtigungskonzept des Brokers für Dokumente basiert auf Berechtigungsrollen für Dokument-Verzeichnisse. Dabei können für jedes Dokument-Verzeichnis die Rechte über Benutzer-Rollen festgelegt werden. Ein Benutzer, der eine bestimmte Rolle hat, kann dann eine bestimmte Tätigkeit in dem Dokument-Verzeichnis auszuführen. Mit dem Broker lassen sich vier verschiedene Tätigkeiten über Rollen autorisieren:

- **Lesen**, erlaubt das Lesen von Dokumenten innerhalb des Verzeichnisses.
- **Schreiben**, gestattet das Schreiben von Dokumenten sowie das Lesen von nicht öffentlichen Dokumenten.
- **Editieren** ist zum Bearbeiten von Meta-Daten des Verzeichnisses oder zum Erstellen oder Löschen von Unterverzeichnissen.
- **Neue Dokumente erstellen** erlaubt es, neue Dokumente anzulegen oder Dokumente zu ändern, die von dem Benutzer angelegt wurden.

Benutzergruppen werden im Broker benutzt, um Benutzer zu arrangieren. Es können verschiedene Benutzer in unterschiedliche Benutzergruppen eingepflegt werden. Sie übernehmen dabei die Rollen der Mitgliedschaft in der Benutzergruppe. Jede Benutzergruppe kann ein Dokument-Verzeichnis haben, in dem die Benutzer der Benutzergruppe anhand ihrer Rollen die entsprechenden Rechte bekommen. Auf diese Art können Benutzer Dokumente anlegen und diese modifizieren.

Für Dataport muss das Konzept erweitert werden, die Freigaben müssen auf Dokumentebene getroffen werden können, also für jedes Dokument einzeln. Dies ist eine Besonderheit, die das Entwickler-Portal stark vom ursprünglichen Broker abhebt. Es gibt zwei Gruppen von Benutzern, die auf Dokumente im Entwickler-Portal Zugriff haben. Zum Einen sind es Benutzer, die über das GovernmentGateway kommen und nur einen Lesezugriff auf bereits veröffentlichte Dokumente haben und zum Anderen sind es Benutzer, die über das Intranet direkt auf das Entwickler-Portal zugreifen. Diese Benutzer haben einen Lese- und Schreibzugriff.

Zusammen mit den Anwendungsfällen (siehe Abbildung 3.3 auf Seite 17) ergeben sich so drei Autorisationsszenarien:

- **Autorisierung über das Gruppenverzeichnis für Broker-Benutzer:** Broker-Benutzer, die in der Gruppe Entwickler sind, haben die Rollen zum Erstellen von Dokumenten und zum Schreiben sowie zum Ändern von Dokumenten, die sie selbst angelegt haben. Zusätzlich können sie veröffentlichte Dokumente lesen, die von anderen Mitgliedern der Entwickler Gruppe erstellt wurden.
- **Autorisierung über die Mitgliedschaft in einer Zielgruppen-Rolle für Broker-Benutzer:** Mit den Zielgruppen ist es möglich, für ein Dokument die Leser-Rollen einer bestimmten Gruppe von Benutzern zuzuordnen. Dazu gibt es für jede Zielgruppe eine Gruppe. Als Zielgruppe können im Broker Konzepte (*Concept Assets*) unterhalb des Zielgruppen-Konzepts ausgewählt werden.
- **Autorisierung über die Mitgliedschaft in einer Zielgruppen-Rolle für Gateway Benutzer:** Wenn das Gateway Daten für einen Benutzer bezieht, müssen seine Rollen mit übertragen und überprüft werden. Die Zielgruppen können von der Fachlichen Leitstelle verändert und es können neue hinzugefügt werden. Gatewaybenutzer haben nur Lesezugriff auf Dokumente, auf die sie zugreifen dürfen, daher haben sie nur eine Lese-Rolle

für die entsprechenden Verzeichnisse. Die Autorisierung funktioniert über die Namen der Zielgruppen-Konzepte im Broker, sie müssen mit den Rollennamen im GovernmentGateway übereinstimmen.

Die Idee hinter dem Konzept ist es, ein *Concept Asset* für die Autorisation zu verwenden. Nur ein Gateway-Benutzer, der vom Gateway dieselbe Rolle mitbringt, wie es das Konzept vorsieht, erhält die Lese-Berechtigung.

3.3 Funktionalität

Die Anforderungen an die Web Service Schnittstelle werden begrenzt durch Vorgaben von Dataport. Es müssen also nicht alle Broker Funktionen über die Web Service Schnittstelle abgebildet und zugänglich gemacht werden.

Der Broker [17] soll bei Dataport mit dem Government Gateway verbunden werden. Die Benutzer die über das Gateway kommen, sollen jedoch nur einen lesenden Zugriff auf die bestehenden Daten haben. Somit können Funktionen zum Verändern oder Erstellen von neuen Dokumenten von Anfang an ausgeschlossen werden.

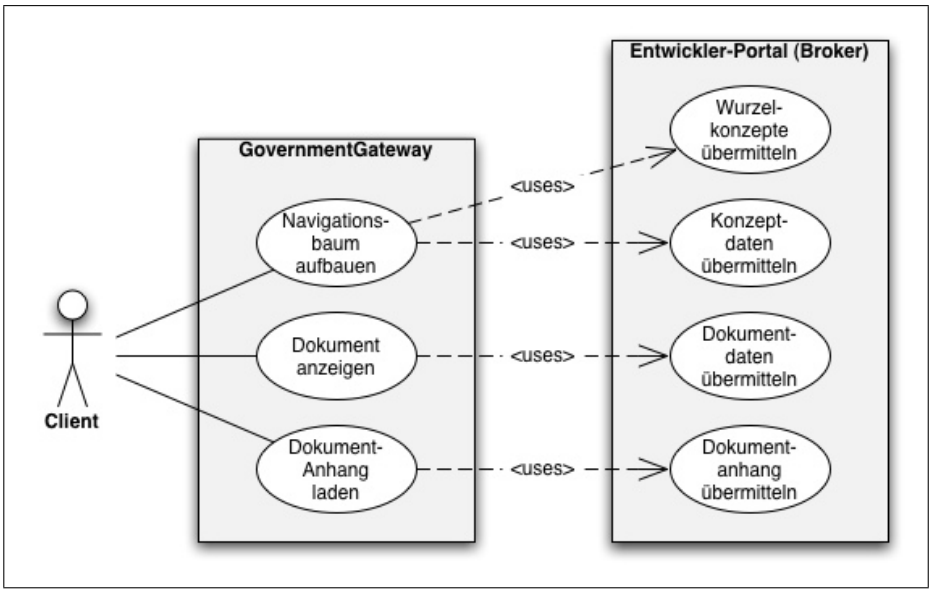

Abbildung 3.5: Use Case Diagramm: Aufbau des Navigationsbaums, Laden von Dokumenten und von Dokumentanhängen.

Explizit gefordert dagegen ist das Übertragen des Navigationsbaumes, um auch über das Gateway eine grafische Navigation zu ermöglichen. Zudem müssen Dokumente übertragen werden, sowie ihre angehängten Dateien.

Zum Übertragen des Konzeptbaums lassen sich verschiedene Modelle vorstellen. Die beiden Extremformen sind dabei den ganzen Baum auf einmal zu laden und als anderes Extrem das Nachladen von möglichst kleinen Teilbäumen. Die erste Lösung zielt darauf ab, sofort alle Daten in den Client zu transportieren und so Nachladen zu vermeiden. Die zweite Lösung überträgt nur benötigte Informationen und somit geringe Datenmengen.

Der Vorteil beim Laden des ganzen Baumes wäre eine Reduktion des Protokoll-Overheads (Verhältnis von Protokolldaten zu Nutzdaten), da alle Daten auf einmal übertragen würden.

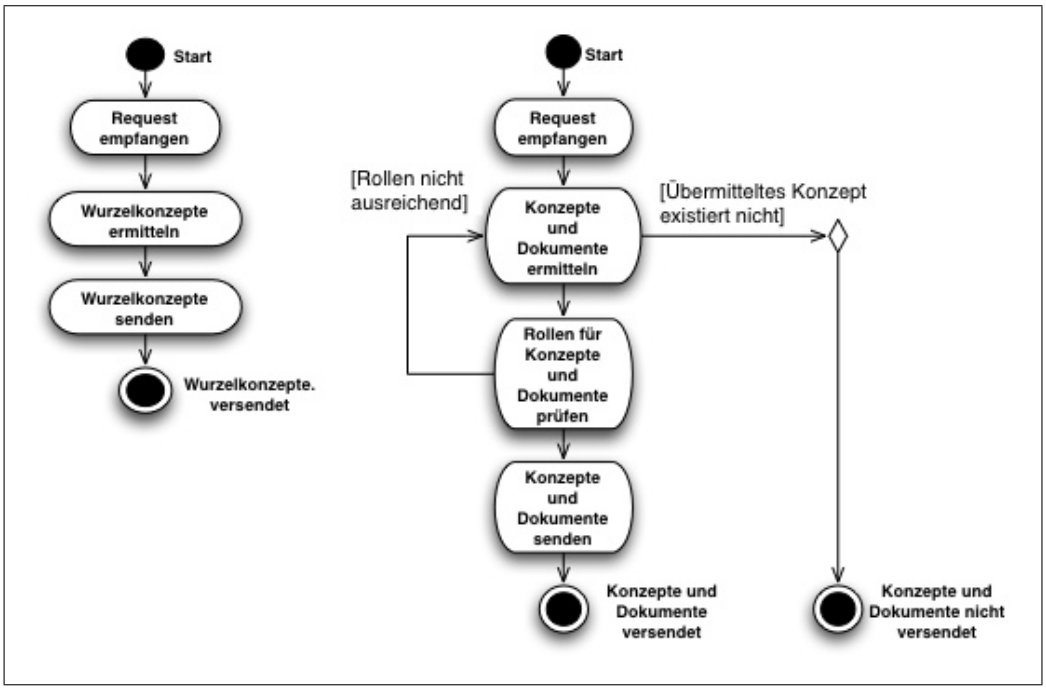

Abbildung 3.6: Ablauf Diagramm für die Use Cases: Laden von Wurzelkategorien und Laden von Kategorien und Dokumenten zu einer gegebenen Kategorie

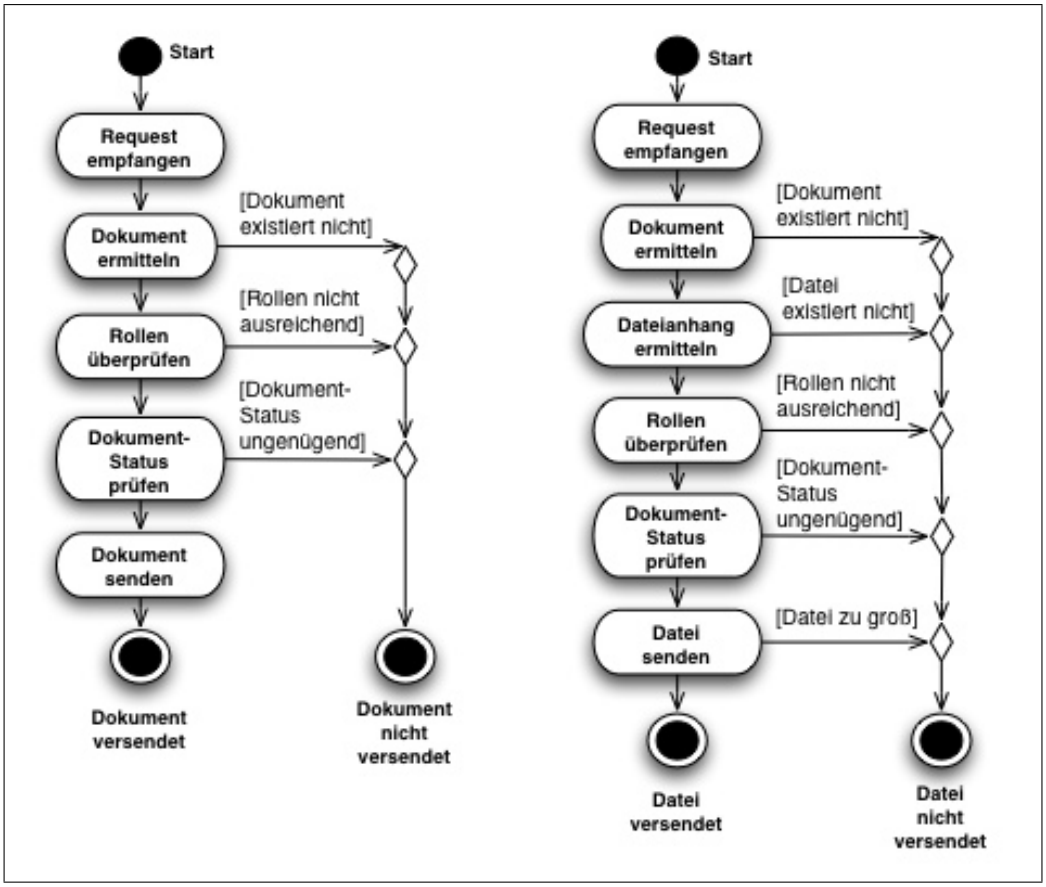

Abbildung 3.7: Ablauf Diagramm für die Use Cases: Laden von Dokumenten und von Dokumentanhängen.

Beim mehrfachen Nachladen von Teilbäumen muss jedes Mal ein neuer Request über das Netzwerk geschickt, verarbeitet und zurückgeschickt werden.

Dennoch scheint das Laden des ganzen Konzeptbaums unpraktikabel. Es kann keine Aussage darüber getroffen werden, wieviele Konzepte sich in der Datenbank des Brokers befinden. Da der Baum also möglicherweise sehr groß ist und aufgrund unterschiedlicher Berechtigungen jeweils für die einzelnen Benutzer geladen werden muss, ist diese Lösung nicht sinnvoll. Praktikabel hingegen erscheint es, nur Teile des Baumes zu laden. Man kann zum Beispiel beim Aufblättern eines Teilbaums die direkt unter dem Knoten liegenden Dokumente und Konzepte laden.

Im Broker kann dies relativ problemlos realisiert werden, da die Konzept-Objekte Informationen über ihre Sub-Konzepte und angehängte Dokumente (und vieles mehr) enthalten.

Das Laden von Konzepten direkt unter dem Wurzelknoten wird als Sonderfall betrachtet und als solcher getrennt behandelt. Er soll als Einstiegspunkt zum Aufbauen eines Konzept-Baums dienen.

Beim Laden von Konzepten unter einem anderen Konzept im Baum sowie beim Laden von Dokumenten oder Dokumentanhängen müssen die Rollen des Benutzers kontrolliert werden. Bei fehlender Berechtigung dürfen die Daten nicht ausgeliefert werden. Wenn eine fehlerhafte Konzept- oder Dokument-ID übermittelt wurde, können ebenfalls keine Daten gesendet werden. Beim Senden von Dokumentanhängen können darüber hinaus noch weitere Fehler entstehen, zum Beispiel wenn die Datei nicht vorhanden oder sie zu groß ist.

Für das Laden von Dokumenten und Dokumentanhängen gibt es noch eine zusätzliche Beschränkung, die auf Besonderheiten des Entwickler-Portals zurückzuführen ist. Für Dataport wurde das Dokument-Modell modifiziert. Die Ausgabe eines Dokuments, beziehungsweise eines Dokumentanhangs über das GovernmentGateway ist nur erlaubt, wenn das Dokument den Status „Öffentlich“ hat.

Die Abläufe werden in den beiden Abbildungen 3.6 und 3.7 auf der vorherigen Seite dargestellt.

3.4 Zielarchitektur

Der infoAsset Broker [17] benötigt einen Java kompatiblen Computer und ein Datenbanksystem. Unterstützt werden vom Broker eine ganze Reihe von Datenbankensystemen:

- Oracle
- IBM DB2
- MySQL
- Microsoft SQL-Server

Dataport wird für den Broker eine Microsoft SQL-Server Datenbank zur Verfügung stellen.

Das Java Runtime Enviroment (JRE) muss Version 2 oder neuer sein. Dataport installiert auf dem Server eine Java 5 Version für den Broker. Als unterliegende Betriebssysteme werden für den Dauerbetrieb folgende Systeme von infoAsset empfohlen:

- Sun Solaris
- HP-UX Unix
- Windows NT

- Windows 2000 Server
- Windows 2003 Server

Von Dataport wird ein Rechner mit dem Betriebsystem Windows XP Professional zur Verfügung gestellt.

Als Broker Client kann im Intranet ein üblicher JavaScript fähiger Web Browser zum Einsatz kommen. Beispiele hierfür sind (laut infoAsset [17]):

- Netscape (ab 6.1)
- Microsoft Internet Explorer (ab 5.0)
- Mozilla (ab 1.0)
- Opera (ab 6.0)

Werden Portalinformationen über die Web Service Schnittstelle im GovernmentGateway dargestellt, gelten die Anforderungen des GovernmentGateways. Das GovernmentGateway benötigt ebenfalls einen JavaScript fähigen Browser. Damit kann die obige Liste übernommen werden.

Der Broker selbst verfügt von Haus aus über keine Schnittstelle für fremde Softwaresysteme. Eine Web Service Schnittstelle ist nicht vorhanden. Üblicherweise wird der Broker durch das Hinzufügen neuer Handler erweitert. Die Schnittstellen für Handler sind offen. Auf diese Art können Handler selber geschrieben und anschließend in den Broker integriert werden. Dazu ist ein Ändern der `Handler.txt` Datei notwendig. Beim Neustarten des Brokers wird diese Datei ausgelesen und die entsprechenden Handler geladen. Das Installieren neuer Handler geht fast immer einher mit dem Hinzufügen neuer Templates. Templates sind für die Darstellung von Portal-Inhalt im Broker zuständig. Aus ihnen sind die Eingabemasken aufgebaut, die im Handler ausgelesen werden und sie sind die Schablonen die aus dem Handler mit Daten befüllt und als Portalseite an den Client zurückgeschickt werden.

Zur Integration einer Web Service Schnittstelle gibt es verschiedene grundlegende Ansätze, die im folgenden diskutiert werden.

3.4.1 Komponentenauswahl für die Realisierung der Web Service Schnittstelle

Um den infoAsset Broker [17] mit einer Web Service Schnittstelle zu erweitern, bieten sich zwei grundsätzliche Möglichkeiten an. Die erste Möglichkeit nutzt in wesentlichen Teilen Broker Bestandteile. Die zweite Lösung setzt auf externe Komponenten.

Wenn hauptsächlich Broker Bestandteile benutzt werden sollen, werden die Requests an den Server des Brokers geschickt. Dieser stellt fest, ob es sich um ein SOAP-Dokument handelt. Dies geschieht über den Mime-Type, der im HTTP-Request enthalten ist. Wenn ein SOAP Dokument erkannt wird, wird dessen SOAP-Anhang im HTTP Request Objekt des Brokers gespeichert und im Handler (in `doBusinessLogic`) über einen XML-Parser (zum Beispiel Apache Axis [2]) ausgewertet.

Zum Versenden des Responses kann die Template-Engine des Brokers verwendet werden. Im Handler wird dazu das entsprechende Template ausgewählt (`setTemplateName`) und mit Daten befüllt (`putSubstitutions`) und schließlich an den Requestor versendet.

Die Entwicklung von Web Services entspricht also bis auf das parsen von XML-Dokumenten genau dem Entwickeln von anderen Broker Handlern, das schließt auch die Registrierung der Handler beim Broker ein.

Als zusätzliche Software-Komponente ist einzig ein XML-Parser notwendig, der Zugriff auf die übergebenen Daten aus dem SOAP-Dokument gibt. Ob ein schneller SAX-Parser verwendet oder zunächst ein Document Object Model (DOM) -Baum aufgebaut wird, bleibt dem Entwickler überlassen.

Der Nachteil an der Version ist allerdings, daß die Entwicklung der Templates fehlerträchtig ist. Es ist nicht möglich, die Templates mit einem Programm und der WSDL-Datei zu generieren, sie müssen von Hand geschrieben werden. Insbesondere bei komplexen oder großen Web Services ist dies sehr fehlerträchtig.

Durch die fehlende Tool-Unterstützung muss selbst bei kleinen Änderungen in der WSDL-Beschreibung immer manuell das Template angepasst werden. Zusätzlich müssen Handler und Template zueinander stimmig sein. Wenn im Template ein Platzhalter nicht eingetragen ist, der im Handler gefüllt werden soll, werden zwangsläufig unvollständige oder fehlerhafte Dokumente übertragen.

In den Broker müsste zusätzlich eine Funktion integriert werden, die eine hinterlegte WSDL-Datei überträgt, wenn ein `?WSDL` Parameter in der URL übergeben wurde.

Die zweite Möglichkeit setzt auf Jetty [28] und Apache Axis [2].

Jetty ist ein in Java geschriebener HTTP Server und Servlet Container. Er wurde unter der Beratung von Mort Bay Consulting [29] geschrieben und ist unter der Apache 2.0 Lizenz veröffentlicht. Jetty wurde seit 1995 stetig optimiert und hat sich zu einem kleinen und effizienten Web Server entwickelt. Er findet in kommerziellen und Open Source Produkten eine weite Verbreitung und ist zum Beispiel in Produkten von BEA, Cisco Systems, Hewlett-Packard, IBM und systinet eingebunden. Jetty ist genau wie Axis Open Source Software.

Apache Axis ist im Prinzip die Weiterentwicklung von Apache SOAP 3.0, jedoch wurde es von Grund auf neu designed und geschrieben. Die Hauptintention war eine flexiblere, modularere und schnellere SOAP-Implementation zu entwickeln. So bietet Axis gegenüber Apache SOAP Unterstützung für WSDL, *Remote Procedure Call*s (RPC), unterstützt *document* und *literal* und ist schneller als sein Vorgänger. Axis hält sich an die Richtlinien des World Wide Web Consortium (W3C) für SOAP 1.1, 1.2 und für WSDL 1.1. Es wird bereits in zahlreichen Produkten verwendet, darunter Produkte von den Firmen Apple, Bea, Borland, IBM und Macromedia.

Axis wurde designed, um auf Application Servern installiert zu werden. Jetty ist so ein Server und kann Axis ausführen. Damit Jetty die Web Services für den Broker bearbeiten kann, muss eine Möglichkeit geschaffen werden, auf die Services vom Broker zugreifen zu können.

Um von Jetty Zugriff auf die Broker Services zu bekommen, ist es zum Beispiel denkbar, die Services als Singleton [16] öffentlich zu machen. Wenn Jetty im Broker gestartet wird, sind die Services nach einem Aufruf des Singletons verfügbar.

Axis kommt mit einem Generator für Java Klassen (`wsdl2java`), der aus einer WSDL-Beschreibung die *skeletons* für Server oder die *stubs* für Clients generieren kann. Durch die Tool-Unterstützung ist es hier mit geringem Aufwand möglich, Veränderungen in der Web Service Beschreibungung umzusetzen. Die generierten Klassen sind orthogonal aufgebaut und lassen sich einfach und intuitiv verwenden. Um das Auslesen von XML-Strukturen oder das Aufbauen der selbigen muss man sich nicht kümmern. Fehler, die sich einschleichen, wenn man Tag-Namen, Namespaces oder ähnliches, ein- oder übergeben muss, werden umgangen.

Wegen dieser Vorteile wird die Web Service Schnittstelle für das Entwickler-Portal in dieser Arbeit mit Jetty und Apache Axis aufgebaut und mit dem Entwickler-Portal verknüpft.

4 Design einer Web Service Schnittstelle im infoAsset Broker Information Portal

Dieses Kapitel behandelt das Design der Web Service Schnittstelle. Es wird festgelegt, wie die Implementierung stattfinden soll. Dazu gehört auch, daß die Verbindung zwischen Entwickler-Portal, Jetty und Axis beschrieben wird.

4.1 Aufbau der Web Service Schnittstelle

Für den Web Service sind im vorherigen Kapitel bereits vier Funktionen identifiziert worden. Diese vier Funktionen sind zum Aufbau des Konzeptbaums, zum Laden eines Dokuments und zum Laden des Dateianhangs eines Dokuments notwendig. Der Aufbau des Konzeptbaums erfolgt über zwei Funktionen. Initial werden die Wurzelkonzepte geladen, anschließend werden über eine andere Funktion jeweils die Kindelemente zu einem Konzept geladen.

Requests

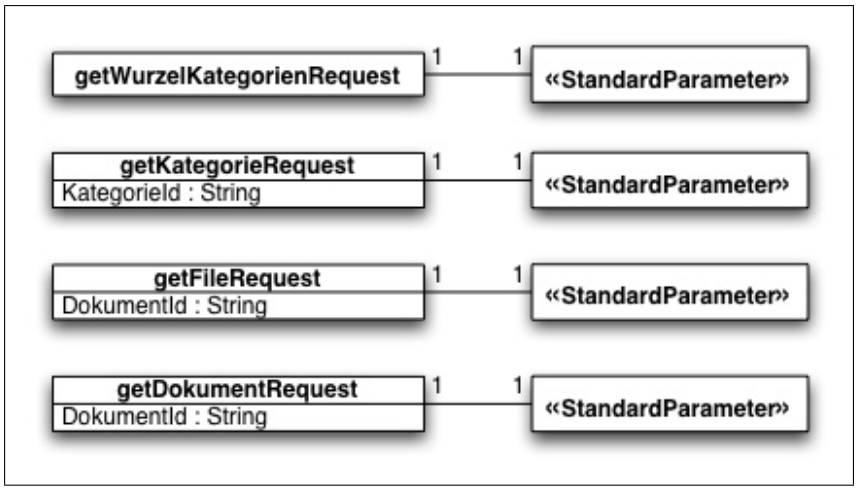

Abbildung 4.1: Initiales Klassendiagramm der Requests

Zur Überprüfung der Berechtigung benötigt der Broker die Rollen aus dem GovernmentGateway, die ein Benutzer hat. Damit später auch zum Beispiel ein Abo-Dienst über die Web Service Schnittstelle verwendet werden kann, bietet es sich an, die E-Mail Adresse des Benutzers mit zu übertragen.

In jedem Request, der den Broker erreicht, werden diese Daten benötigt. Im Nachfolgenden wird die Verbindung aus einer Menge von Rollen, sowie der E-Mail Adresse des Benutzers als `StandardParameter` bezeichnet. Die E-Mail Adresse wie auch die einzelnen Rollen werden als Strings (`xsd:String`) typisiert. Ein Benutzer kann 0 . . . n Rollen haben.

Zum Laden eines Dokuments oder zum Laden eines Dokumentanhangs (also einer Datei) ist neben dem `Standardparameter` nur eine Dokument-Id notwendig. Die Dokument-Id ist ein

Brokerweit eindeutiges Identifikationsmerkmal für Dokumente, mit der ein Dokument aus dem Bestand herausgesucht werden kann. Die Dokument-Id wird als String (`xsd:String`) beschrieben.

Zum Aufbau des Konzeptbaums gibt es zwei Methoden. Die erste bezieht die Kindelemente des Wurzelknotens. Die andere Methode dient zum Nachladen der Kindelemente eines gegebenen Konzepts. Jedes Konzept wird mit einer Concept-Id eindeutig beschrieben. Zum Beziehen der Kinder des Wurzelknotens ist neben dem `StandardParameter` kein weiterer Parameter notwendig.

Wenn zu einem gegebenen Konzept die Kindelemente geladen werden sollen, ist es notwendig, die Concept-Id des aktuellen Konzepts zu übermitteln. Zusätzlich zum `StandardParameter` muss also noch ein String (`xsd:String`) mit der Concept-Id mitgeschickt werden.

Responses

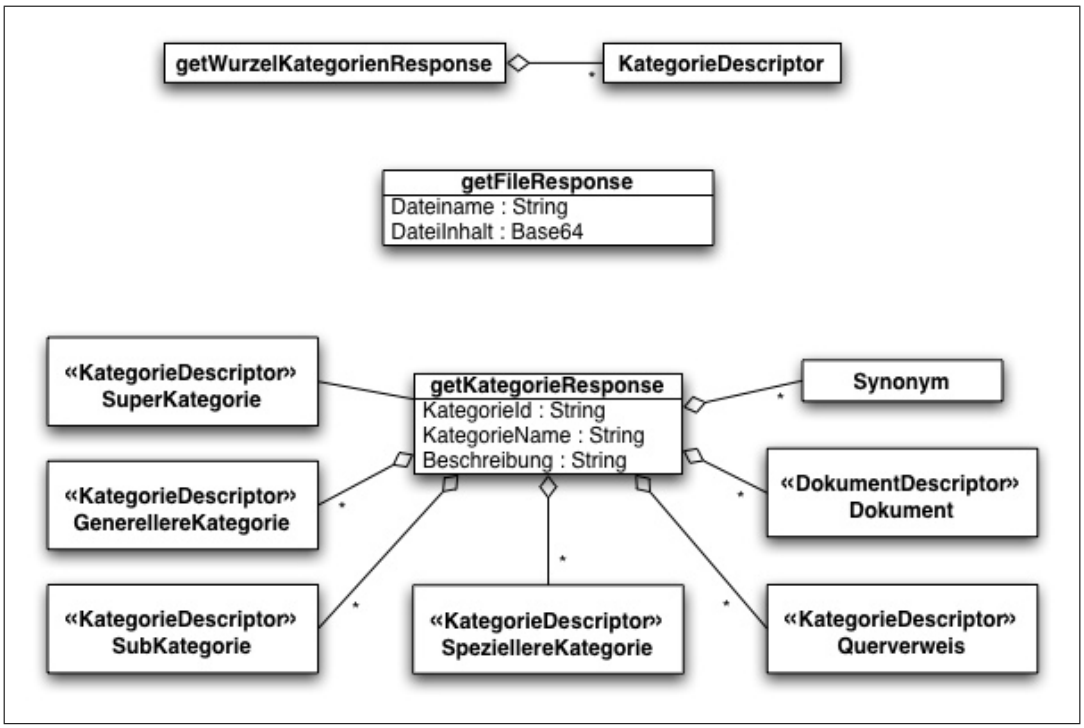

Abbildung 4.2: Initiales Klassendiagramm der Responses (1/2)

Zum Laden des Dateianhangs ist es notwendig, den Dateiinhalt für den Transport zu codieren. Dies wird mit einem `xsd:base64` Encoding gemacht. Der Dateiname muss ebenfalls mit in der Response Nachricht mitgeschickt werden, um dem Dateiinhalt auf der Clientseite seinen ursprünglichen Dateinamen zuordnen zu können.

Ein Request nach Konzepten unter dem Wurzelknoten ist der erste Request, der gestellt wird, um einen neuen Konzept-Baum aufzubauen. Der Broker stellt hier eine Menge von Konzepten zur Verfügung, die direkt unter dem Wurzelknoten hängen. Ein Konzept besteht immer aus einem Namen und seinem Schlüssel, der Concept-Id. Beide werden als String in einem `KategorieDescriptor` zusammengefasst. Die Bezeichnung Kategorie stammt aus der Festlegung, die Kategorien eines Dokuments über Konzepte abzubilden. Der Response für einen Request nach Wurzelkategorien muss also eine Liste von `KategorieDescriptor`en sein.

Wenn ein Request zu den Kindelementen eines Konzepts beantwortet werden soll, müssen mehr Daten übertragen werden. Es müssen die Konzept-Id und der Name zurück übermittelt werden.

Zusätzlich sollte die Beschreibung zu dem Konzept mit übertragen werden. Diese drei Informationen werden als String abgebildet. Im Broker wird zudem eine Synonymliste gepflegt. Die Synonyme werden als String gespeichert und so in die Web Service Schnittstelle übernommen. Jedes Konzept unterhalb des Wurzelknotens hat genau ein Superkonzept. Dieses wird mit einer Konzept-Id und einem Konzeptnamen beschrieben. Beide Daten werden durch Strings abgebildet. Die Id dient zum eindeutigen Identifizieren des Konzepts. Da diese Daten immer zusammengehören, werden sie wie andere Konzepte mit einem `KategorieDescriptor` beschrieben. In Navigations-Applet des Brokers werden noch weitere Daten abgebildet: Generellere Konzepte, Subkonzepte, speziellere Konzepte und Querverweise. Alle diese Listen enthalten Konzepte die mit dem `KategorieDescriptor` adäquat beschrieben werden. Unter einem Konzept im Baum können auch Dokumente hängen. Um Dokumente zu beschreiben reichen ähnlich wie bei Konzepten, eine Dokument-Id und ein Name. Die Dokument-Id identifiziert ein Dokument eindeutig. Die beiden Daten können wie bei Konzepten als String modelliert werden und in einem Descriptor zusammengefasst werden. Dieser wird folgend als `DokumentDescriptor` bezeichnet.

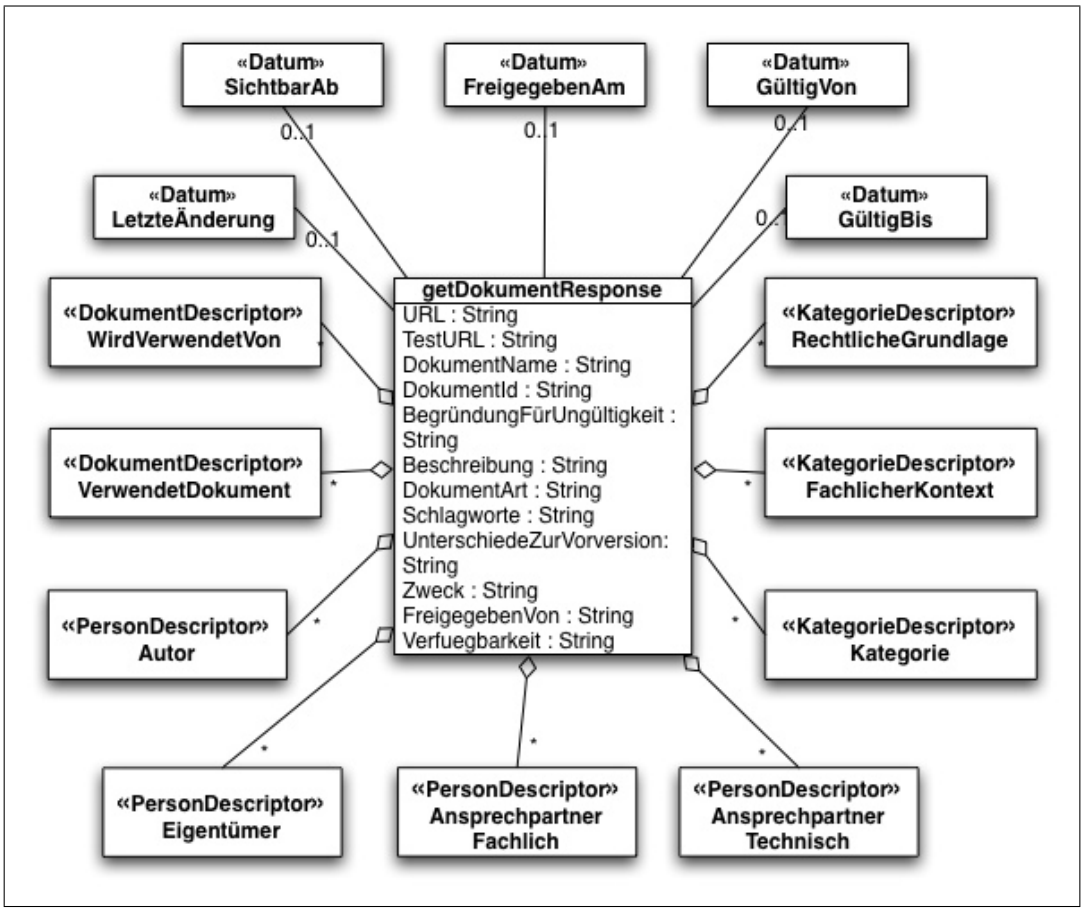

Abbildung 4.3: Initiales Klassendiagramm der Responses (2/2)

Die umfangreichsten Daten werden beim Abrufen eines Dokuments übermittelt. Der Broker hält eine große Menge von Daten zu einem Dokument. Die wichtigsten sollen über den Web Service im Response übermittelt werden. Neben den Datenfeldern, die der Broker ohnehin schon führt, sind durch die Anpassung hin zum Entwickler-Portal noch neue dazugekommen. Die Felder gehen auf Anforderungen von Dataport an das Entwickler-Portal zurück. Eine genaue Bedeutung der Datenfelder ist in der Bachelor Thesis von Jun Zhang [14] zu finden.

Dokumente enthalten eine Menge von Daten, die als Strings abgebildet werden. Dazu zählen

folgende:

- Begründung für Ungültigkeit
- Beschreibung
- Dokument-Id
- Dokumentart
- Dokumentname
- Schlagworte
- TestURL
- Unterschiede zur Vorversion
- URL
- Verfügbarkeit
- Zweck

Zum Beispiel für die zeitliche Steuerung von Dokument-Zuständen werden einige Datumsfelder in den Dokumenten gesichert. Diese Typisierung sollte für den Fall der späteren Weiterverarbeitung beibehalten werden, deshalb sollten die Datumsfelder nicht als Zeichenkette sondern als Datum übermittelt werden. Die relevanten Datumsfelder sind die folgenden:

- Freigegeben Am
- Gültig Von
- Gültig Bis
- Letzte Änderung

Innerhalb des Entwickler-Portals ist es möglich, daß Dokumente voneinander abhängig sind. Diese Abhängigkeiten werden mit dem `DokumentDescriptor` abgebildet. Auf diese Art kann auch über die Web Service Schnittstelle über voneinander abhängige Dokumente navigiert werden. Folgende Datenfelder müssen dazu mit dem `DokumentDescriptor` abgebildet werden:

- Verwendet Dokument
- Wird verwendet von

Ähnlich den Dokumenten können auch Personen in Beziehung zu Dokumenten stehen. Damit eine Person über die Schnittstelle erreichbar ist, muss diese in einem `PersonenDescriptor` beschrieben werden. Dieser enthält den Vor- und Nachnamen der Person, sowie seine Person-Id aus dem Broker. Die Person-Id ist eindeutig und kann zum Beziehen der Personendaten verwendet werden. Felder die den `PersonDescriptor` verwenden sind:

- Ansprechpartner fachlich
- Ansprechpartner technisch

- Autor
- Eigentümer

Da Dokumente an Konzepte gebunden werden und Kategorien über Konzepte abgebildet werden, müssen auch diese Abhängigkeiten abgebildet werden. Dazu dient der bereits beschriebene `KategorieDescriptor`. Dieser wird für die folgenden Attributen verwendet:

- Fachlicher Kontext
- Kategorie
- Rechtliche Grundlage

Diese Attribute bilden ein Dokument aus dem Entwickler-Portal in die Web Service Schnittstelle ab.

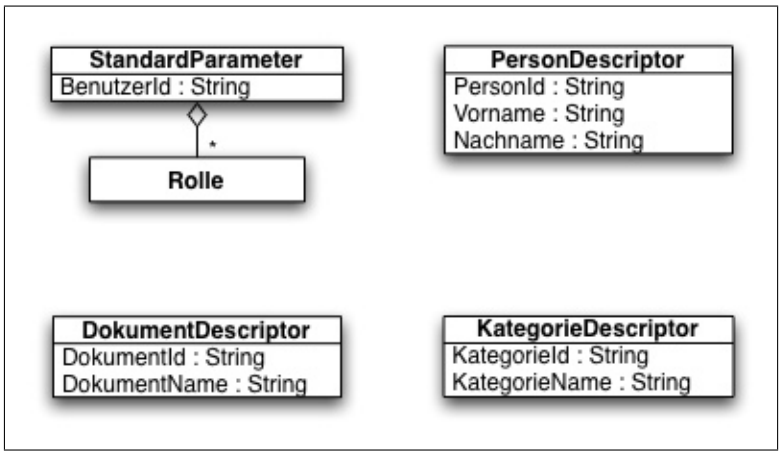

Abbildung 4.4: Initiales Klassendiagramm der Wrapperklassen

4.2 Web Service Architektur

Die Entscheidung für Jetty mit Apache Axis und dem Broker wurde bereits in Kapitel 3.4.1 ausfürlich behandelt.

Zur Realisierung einer Web Service Schnittstelle muss der Broker angepasst werden. Ziel der Anpassung ist das Freilegen der Broker Services. Jetty muss im Broker gestartet werden. Diese beiden Modifikationen betreffen den Broker direkt. Wenn diese Änderungen einmal vorgenommen sind, kann ein beliebiger Web Service für den Broker geschrieben werden.

Für Web Services gibt es diverse Erweiterungen, die verschiedene andere Probleme im Zusammenhang mit Web Services adressieren:

Prozess Modellierung Zur Beschreibung von Geschäftsprozessen gibt es die Spracherweiterungen *Business Process Execution Language for Web Services* (BPEL4WS) [5] die von unter anderem Microsoft [27] und IBM [20] unterstützt wird und die *Business Process Modelling Language* (BPML) die unter anderem von SAP [35] und SUN [36] getragen wird.

Chroeographie Für die Beschreibung von Choreographien können die Beschreibungssprachen *Web Service Choreography Description Language* (WSCDL) [50] und *ebXML Business Process Specification Schema* (BPSS) [40] verwendet werden.

Sicherheit Für Web Service Sicherheit gibt es verschiedene Spezifikationen. Dazu zählen WS-Security, WS-SecureConversation, WS-Trust und WS-Federation.

Transaktionen Um Transaktionen, zu realisieren stellt Microsoft zwei Spezifikationen zur Verfügung. Die Eine zielt auf atomare Transaktionen, die Andere auf Langzeit-Transaktionen ab. Für atomare Transaktionen wird ein *2-Phase Commit Protocol* (2PC) realisiert, welches den *ACID (Atomicity, Concurrency, Isolation, Durability)* Anforderungen genügt. Microsofts Spezifikation hierfür ist WS-AtomicTransaction, für Langzeittransaktionen ist sie WS-BusinessActivity sowie WS-Coordination.

Die Liste ist mit den oben genannten Erweiterungen ist längst nicht vollständig. Während sich für Web Services Standards etabliert haben, ist dies bei den Erweiterungen noch nicht der Fall. Viele der Standards für Erweiterungen sind noch im Entstehen und es wird noch dauern, bis sich diese in der Praxis und Industrieübergreifend durchgesetzt haben.

Für die Realisierung der Web Services für das Entwickler-Portal werden keine Web Service Erweiterungen benötigt, da weder ganze Geschäftsprozesse modelliert werden müssen, noch Transaktionen benötigt werden. Die Web Services werden in einer speziell geschützten Umgebung bei Dataport eingesetzt und wie die anderen bereits laufenden Web Services nicht zusätzlich abgesichert.

4.2.1 Representational State Transfer - REST

Mit *Representational State Transfer* (REST) wird ein Architekturstil bezeichnet der auf Roy Fielding und seine Dissertation [3] aus dem Jahr 2000 zurückgeht. REST basiert auf Prinzipien die bereits im *World Wide Web* (WWW) eingesetzt werden. Es handelt sich um kein Produkt und um keinen Standard. Die REST Prinzipien sind nicht für Web Services gedacht gewesen, sie können aber auch auf Web Services angewandt werden.

Nach REST ist jedes Objekt einer Anwendung eine Resource, die nach außen über einen *Uniform Resource Locator* (URL) adressierbar und veränderbar ist. Eine Anfrage nach solch einem Objekt wird über einen HTTP GET-Request formuliert. Wie die Repräsentation der angefragten Resource aussieht, ist nicht spezifiziert, eine Resource kann mehrere Repräsentationen haben. Über Verknüpfungen in zurückgelieferten Daten können andere Resourcen aufgerufen werden. Auf diese Art findet der *State Transfer* statt, von dem REST seinen Namen hat. Für den Fall, daß zum Beispiel ein XML-Dokument zurückgeliefert wurde, kann über XLINK [43] zu einer anderen Resource navigiert werden.

Als Einstiegspunkt kann eine einzige URL dienen, von der aus auf viele andere URLs und damit Resourcen verzweigt wird. Auf diese Art ließe sich die Idee der aus HTML-Dokumenten *(HTML - Hypertext Markup Language)* bekannten Hyperlinks auf Web Services übertragen.

Um Web Services nach den REST Prinzipien umzusetzen, reicht es nicht nur, die GET-Methode von HTTP zu verwenden. Für REST wurde die Semantik des HTTP Protokolls übernommen. Die zentrale Rolle spielen die HTTP-Methoden:

GET Mit GET wird eine Repräsentation einer Resource abgefragt. Requests sollten frei von Seiteneffekten sein. Das bedeutet, GET Requests können mehrfach gesendet werden, ohne eine Veränderung an der Resource auszulösen.

POST POST erlaubt es, einer Resource etwas hinzuzufügen. POST ist nicht frei von Seiteneffekten: Der Aufruf der POST Methode an einer Resource führt zu ihrer Veränderung.

PUT Eine neue Resource kann mit PUT erstellt oder der Inhalt einer bestehenden Resource ersetzt werden.

DELETE Bestehende Resourcen können mit DELETE gelöscht werden.

Die HTTP-Methoden stellen die Verben dar, die auf die Hauptwörter, also die Resourcen, angewandt werden. Jede REST Resource besitzt über die oben genannten HTTP-Methoden eine generische Schnittstelle mit der sich die meisten Anwendungsfälle realisieren lassen.

In einer REST Nachricht müssen alle Informationen enthalten sein, um diese zu bearbeiten. Sie sollen also selbstbeschreibend sein. Es bedarf keines Wissen über voherige oder folgende Nachrichten. Der Server kennt keinen Status und der Client verwaltet seinen Status selbst. Der Client entscheidet über die Reihenfolge, in der er verschiedene Methoden auf einem Server aufruft.

Vorteile von REST

In REST ist jede Resource direkt über ihre URL adressierbar. So kann auf jede Resource über eine generische Schnittstelle zugegriffen und sie so verändert werden. Diese generische Schnittstelle ermöglicht eine sehr lose Koppelung der Anwendungsteile. Möchte ein Server zum Beispiel mehr Daten zu einer Resource übermitteln als bisher, muss dazu nicht die Schnittstelle geändert werden.

Proxies und Caches können mit REST verwendet werden. Ein Server kann seine Antwort als Cache fähig oder nicht Cache fähig markieren. Durch die Verfolgung von Links werden die Grenzen zwischen Anwendungen und Server transparent gemacht. Da REST *stateless* ist, muss kein Status zwischen den Servern propagiert werden. All diese Punkte führen zu einer sehr guten Skalierbarkeit von REST Anwendungen.

Für Firewalls wird keine spezielle Technologie benötigt, um REST Requests zu loggen oder zu überwachen. Es können problemlos einzelne HTTP-Methoden gesperrt und freigegeben werden. Resourcen auf die zugegriffen werden kann, können über URL-Filter begrenzt werden. REST funktioniert dabei trotzdem über Firewalls hinweg, da es nur den HTTP-Port verwendet, der meist frei gegeben ist. Dadurch, daß die REST URLs sehr aussagekräftig sind, lässt sich über das Logging der Requests eine gute Überwachung erzielen. Im Gegensatz dazu können sich bei SOAP hinter einer URL eine Vielzahl von verschiedenen Operationsaufrufen verbergen. Diese werden von einer Firewall in der Regel nicht erfasst. Um dies zu ändern müssten die gesamten SOAP-Bodies analysiert werden.

REST nutzt die bisher vorhandenen HTTP Authentifizierungsmechanismen.

Nachteile von REST

Das Serialisieren und Deserialisieren von und nach zum Beispiel XML wird in REST nicht behandelt. Es existiert kein gemeinsamer Standard zum Definieren und Beschreiben der Austauschformate zwischen Client und Server.

Die verschiedenen URLs zum Zugriff auf die Resourcen müssen oft dynamisch erzeugt werden. Dies führt zu einem erhöhten Aufwand auf dem Sever.

HTTP PUT und GET sind weit verbreitet und implementiert und haben im *World Wide Web* ihre Effizienz bewiesen. Die Unterstützung für die Verben PUT und DELETE ist weniger verbreitet, ob sie ebenso gut skalieren wie GET und POST ist noch offen. Bei einer nachträglichen Erweiterung muss auf eine ausreichende Skalierbarkeit geachtet werden.

Es ist aufwendig, atomare Transaktionen mit REST zu implementieren.

REST Zusammenfassung

Merkmale einer REST Anwendung:

- Der Client ist aktiv und ruft vom passiven Server eine Repräsentation einer Resource ab, beziehungsweise modifiziert die Resource.
- Resourcen besitzen eine ihnen zugeordnete URL, mit der sie adressiert werden.
- Jede Repräsentation kann auf weitere Resourcen verweisen.
- Der Server verfolgt keinen Clientstatus. Jede Anfrage an den Server beinhaltet alle Informationen, die zum Bearbeiten der Anfrage notwendig sind.
- Proxies und Caches werden unterstützt. Der Server kann seine Antwort als Cache fähig oder nicht Cache fähig kennzeichnen.

REST ist ein Architekturstil für große Anwendungen mit unbekannter Benutzeranzahl. Allerdings werden Probleme wie das Serialisieren von und nach Austauschformaten nicht behandelt; die Lösung dieses Problems bleibt dem Entwickler überlassen.

Durch eine Kombination von REST und SOAP können Web Service Architekturen gebaut werden, die die Flexibilität und Skalierbarkeit des *World Wide Web*s nutzen und trotzdem die Benutzung der Web Service Technologien wie SOAP und WSDL ermöglichen. Ein Schritt in diese Richtung ist die Aufnahme der *Web Method Specification Features* in SOAP 1.2 [46] sowie die Beschreibung der Verwendung von GET, wenn es sich um die Ausführung von Web Service Methoden ohne Veränderungen oder Seiteneffekte handelt, die zudem das Cachingverhalten nicht stören.

Bei Dataport werden derzeit standard SOAP basierte Web Services eingesetzt, die nicht den Prinzipien von REST folgen. Da auf eine Beschreibung der Web Services in WSDL nicht verzichtet werden kann – sie ist Grundlage zur Generierung der Stubs und Skeletons – und Dataport eine formalisierte Kommunikation zum Beispiel über XML-Schema und WSDL fordert, können die REST Prinzipien nicht umgesetzt werden.

4.2.2 Semantische Web Services durch Ontologien

Während WSDL Web Services syntaktisch ausführlich beschreibt, kommt eine semantische Beschreibung der Web Services zu kurz. Die Namen der Messages, Types und so fort geben in WSDL-Dateien allenfalls dem Menschen einen Aufschluss darüber, was sich hinter dem Namen verbergen könnte. Dem Computer bleibt der Sinn des Web Services jedoch ganz verborgen.

Verzeichnisdienste wie UDDI [32] helfen hier nicht weiter. Denn auch die Daten, die in UDDI aufgenommen werden, sind für den Computer nicht in ausreichendem Maße auswertbar. Ihm bleibt weiter die Möglichkeit verschlossen, einen passenden Web Serice für eine gegebene Aufgabe

zu finden und auszuführen. Selbst für Menschen ist es schwer, in großen UDDI-Beständen einen passenden Web Service mit passender Schnittstelle zu finden. Zu ungenau sind die bisherigen Suchverfahren, die sich einzig auf die Beschreibungen stützen.

Der Grund dafür ist, daß die Daten, die wir produzieren, typischerweise unzureichend beschrieben werden. Zur Beschreibung werden Daten benutzt, die sich schlecht oder gar nicht in Bezug zu anderen Informationen setzen lassen. Üblicherweise kann man Informationen jedoch nur in einem Kontext zu anderen Informationen deuten.

Der Computer benötigt formalisierte, semantisch reiche Metadaten, um Informationen sinnvoll in Beziehung setzen zu können. Genau dieses Problem greifen Ontologien auf. Ontologien dienen als Basis zur semantischen Beschreibung, durch eine geteilte Konzeptualisierung für einen Bereich. [19]

Semantische Informationen in Web Standards integrieren

Die folgende Beschreibung gibt einen Einblick wie Standard Web Services Technologien erweitert werden können, um eine semantische Suche zu unterstützen. Der Text orientiert sich dabei an [1].

Die Beschreibung der Semantik mit in die Web Service Beschreibung aufzunehmen, wird als Schlüssel für das Web Service Discovery Problem verstanden. Ontologien [19] werden als Basis zur Beschreibung von semantischen Informationen angesehen. Sie bieten eine Konzeptualisierung für einen abgegrenzten Bereich, die zwischen den Benutzern geteilt wird. Durch ein Verknüpfen von Ontolgie-Konzepten mit Konzepten einer Resource können semantische Bedeutungen der Resource beschrieben werden.

Um eine semantische Suche zu ermöglichen, muss eine Suche realisiert werden, die es erlaubt, semantische Konzepte mit einzubeziehen. Anhand der angegebenen semantischen Informationen kann ein Suchsystem die übergebenen Konzepte mit denen aus den Resourcen abgleichen.

Web Service Beschreibungen (WSDL) sind erweiterbar und lassen sich mit neuen Elementen versehen, die zum Beispiel zum Aufnehmen von semantischen Daten verwendet werden können. Der Vorteil dieser Lösung ist, daß die semantischen Daten direkt mit der Beschreibung der Schnittstelle verbunden sind. Eine Anwendung, die mit den semantischen Daten nicht umgehen kann, wird sich jedoch an ihnen nicht stören. Auf diese Art bleibt die Lösung rückwärtskompatibel.

Beim Mapping von Ontologie-Konzepten auf Konzepte der Web Services, sollten mehr als nur die reine Funktionalität einer Funktion beschrieben werden. Es ist sinnvoll, Vorbedingungen und Effekte einer Funktion ebenfalls zu beschreiben. Dies ermöglicht nicht nur das Auffinden von Web Services mit einer passenden Funktion, sondern auch Web Services zu finden, deren Funktionen zu vorgegebenen Parametern passen, oder die bestimmte Seitenbedingungen erfüllen. Beispielsweise können zwei Bestellfunktionen dieselben Parameter haben, jedoch die eine veranlasst das Senden von einer Bestellbestätigung per Post, während die andere E-Mails verschickt.

Vorbedinungen können logische Vorgaben sein, welche zum Beispiel wahr sein müssen, bevor eine Operation ausgeführt werden darf. Effekte sind Veränderungen, die durch das Ausführen der Operation stattfinden. Die Effekte und Vorbedingungen können als Kindelemente des `operation` Elements in die WSDL Beschreibung integriert werden. Ganz generell können die ontologischen Konzepte über Erweiterungselemente in WSDL integriert werden, wie dies auch in der WSDL Spezifikation 1.2 [48] vorgeschlagen wird.

Damit weiter vorhandene Web Services Standards verwendet werden können, wird vorgeschlagen UDDI zu erweitern, daß es die Ontologie-Beschreibungen aus dem WSDL-Dokument lesen kann. Auch diese Erweiterung ist rückwärtskompatibel, lässt aber das Erweitern um Ontologien zu.

Anerkannte Spezifikationen und Taxonomien können in UDDI als *tModels* registriert werden. Um die semantischen Informationen in UDDI abzubilden, können vier *tModels* erstellt und registriert werden. Das erste enthält dabei die Konzepte, die die Funktionalität repräsentieren. Das zweite und dritte *tModel* enthalten die Konzepte für input und output. Das vierte *tModel* nimmt die Gruppierung jeder Operation mit seinen inputs und outputs auf. Die *tModel*s können mit dem *overviewURL* Feld mit entsprechenden Ontologien verknüpft werden. So können all diese *tModel*s auch mit einer einzigen übergreifenden und übergeordneten Ontologie verknüpft werden.

Seit UDDI Version 3 [39] können *keyedReferenceGroups* erstellt werden. Damit können Kategorisierungen von tModels gruppiert werden. Jede *keyed reference* hat einen *keyValue* welcher ein ontologisches Konzept repräsentiert und einen *tModelKey*, welcher die Ontologie selbst darstellt. So kann jede Operation mit ihren inputs, outputs, Vorbedingungen und Effekten mit Hilfe in eine *keyedReferenceGroup* gruppiert werden.

Damit ein Benutzer einen passenden Web Service finden kann, erstellt er ein *Template* seines gesuchten Web Services. Darin verwendet er analog zum Beschreibungsvorgang von Web Services die ontologischen Konzepte für inputs, outputs, Vorbedingungen und Effekte.

Nun kann ein Suchalgorithmus als erstes die Web Services heraussuchen, deren Operationen zu dem gesuchten Service passen. Dies wird anhand von ontologischen Konzepten geprüft, die auf die Operationen gemapped sind. Dann wird die Ergebnismenge anhand ihrer semantischen Ähnlichkeit sortiert. Dazu werden input und output Konzepte des Templates mit denen der Web Services Operationen verglichen. Eine optionale dritte Phase kann nun noch die semantische Ähnlichkeit von Vorbedingungen und Effekten überprufen und in das Ergebnis mit einbeziehen.

Einen anderen Ansatz verfolgt [9], wo Web Services über *Service Profile*, *Process Model* und *Service Grounding* mithilfe von DAML-S (DAML - Darpa Agent Markup Language [10]) Ontologien beschreiben werden. Die Beschreibung ist dabei ähnlich dem aktuellen OWL [47], welches eine weitere Möglichkeit zum Beschreiben von Semantischen Web Services bietet. OWL ist eine Überarbeitung der DAML+OIL Web Ontology Sprache. Dabei wurde aus dem Design und der Anwendung von DAML+OIL gelernt und das Wissen als Verbesserung in OWL eingebracht. OWL ist dabei noch aussagelkräftiger als der obige Ansatz, da er unter anderem auch die semantische Komposition von Web Services zur Erlangung eines Ziels zulässt.

Einen Kontrast zu den Ontoloie basierten Ansätzen stellt [7] dar, hier werden die Objekt-Beziehungen zu ihrer Anwendung durch konzeptuelle Modellierung beschrieben. Als Grundlage dienen hier Assets und nicht Ontologien. Das vorgestellte System ist dabei offen und dynamisch gestaltet. Benutzer sind in der Lage, die Beschreibungen der Objekte ihren persönlichen Sichtweisen anzupassen. Das System reagiert auf die Änderungen ohne das Programmierkenntnisse notwendig sind. Ein unabhängig existierendes Domänen-Modell hilft bei der Aufgabe des Auffindens der richtigen Operationen und dabei diese konsistent zu implementieren. Auf diese Art und Weise, sind die Benutzer nicht einer globalen Sichtweise, wie bei Ontologien, unterworfen.

Die Realisierung von semantischen Web Services war keine Vorgabe von Dataport, deswegen wurden sie nicht umgesetzt.

4.2.3 Jetty mit Apache Axis

Axis kommt in Form eines Web-Archives, das in einem Servlet 2.2 konformen Web-Container wie zum Beispiel Jetty installiert werden kann. Im Archiv sind drei Servlets enthalten: Eines für die Laufzeit, eines zum Verfolgen von SOAP-Aufrufen und eines für die Administration.

Dabei ist Axis mudular in Schichten aufgebaut, so daß Dienste aus höheren Schichten auf die Dienste der unter ihnen liegenden Schichten aufbauen. Die Module gliedern Axis in Teilsysteme, welche verschiedene Aufgaben erledigen. Beispiele für Teilsysteme sind der Transport von Nachrichten, die Administration und der Aufruf von Serviceimplementierungen.

Die Verarbeitung von Request- und Response-Nachrichten wird von Handlern erledigt. Die Handler können zur Verarbeitung zu Handler-Ketten zusammengefügt werden. Dabei hat eine Handler-Kette die selbe Schnittstelle wie ein einzelner Handler. Die Verkettung von Handlern kann über eine Konfigurationsdatei angepasst werden, so daß keine Änderung an Axis selbst notwendig ist, um auf die Verarbeitung von Nachrichten Einfluß zu nehmen. Zusätzlich kann durch eigene Handler die Bearbeitung von Nachrichten erweitert oder verändert werden.

Um Anwendungen mit Axis zu schreiben, kann zwischen zwei alternativen Vorgehensweisen gewählt werden: Entweder das Axis Archiv wird mit eigenem Code erweitert oder Bestandteile von Axis können in ein eigenes Web-Archiv übernommen werden. [2]

In dieser Arbeit wird das Web Archiv von Axis mit eigenem Code erweitert. Das Entwickeln eigener Handler oder Anpassen von Handler-Ketten ist für die Realisierung der Web Service Schnittstelle nicht notwendig, da nur einfache Requests und Responses verschickt und verarbeitet werden und Axis dieses bereits leistet. Zudem soll die Ablauflogik hauptsächlich durch den Broker, beziehungsweise seine Web Services, gesteuert werden.

Jetty wird unmodifiziert benutzt. Er soll als Server und Servlet-Container aus dem Broker heraus gestartet werden und das Axis Servlet ausführen. Um Axis mit Jetty zusammen zu fügen genügt es, aus dem Axis-Archiv den Ordner `axis`, der in `webapps` liegt, in den Web-Applikations Ordner von Jetty zu kopieren. Die Position des Web-Applikation Ordners kann frei gewählt werden.

Mit dem Apache Admin Tool (`org.apache.axis.utils.Admin`) können die Web Services *deployed* werden, nachdem sie implementiert wurden. Es erzeugt eine globale Konfigurationsdatei aus einer *Web Service Deployment Descriptor* (WSDD) Datei. Hierzu muss der Server mit Axis allerdings laufen. Dieser Schritt muss nur einmal ausgeführt werden, um die Konfigurationsdatei aus der `deploy.wsdd` Datei zu erstellen.

4.3 Integration in das Entwickler-Portal

Der Broker ist relativ monolithisch aufgebaut. In der Verarbeitung von Requests greifen die Handler auf die Services, die mit den Business Objekten verbunden sind zu, um die Daten zu verarbeiten. Eine Schnittstelle für fremde Anwendungen ist dabei nicht vorgesehen. In der verwendeten Broker-Version sind die Services einzig im Broker verfügbar.

Damit fremde Anwendungen überhaupt die Broker Services sinnvoll nutzen können, ist eine saubere Verteilung der Business-Logik zwischen Handlern und Broker Services notwendig. Liegt zuviel Logik in den Handlern, ist eine Anbindung von Fremdanwendungen schwieriger, da hier viel Business-Logik neu geschrieben werden müsste. Ein Zugriff auf die Handler ist Architekturbedingt sehr ungünstig.

Unglücklicherweise gibt es keine Schnittstelle für die Anbindung von Fremdsoftware. Wären die Services beispielsweise über ein Singleton [16] von Anfang an öffentlich zugänglich, könnte Jetty als zusätzliche externe Anwendung gestartet werden. Das in Jetty laufende Axis könnte mit seinen Web Services über das Singleton auf die Services des Brokers zugreifen und darüber die Business-Objekte des Brokers nutzen. Hierfür wäre keine weitere Verbindung zwischen den beiden Softwareprodukten notwendig. Jetty liefe in der selben Virtuellen Java Maschine wie der Broker und über das Singleton könnten beide Anwendungen miteinander kommunizieren.

Da die Services nicht öffentlich zugänglich sind, müssen die Brokerquellen modifiziert werden, damit Jetty einen Zugriff auf die Business-Objekte des Brokers bekommen kann. Der Vorteil einer Modifikation des Brokers liegt darin, daß Probleme durch Synchronisation im Wesentlichen ausgeschlossen werden können.

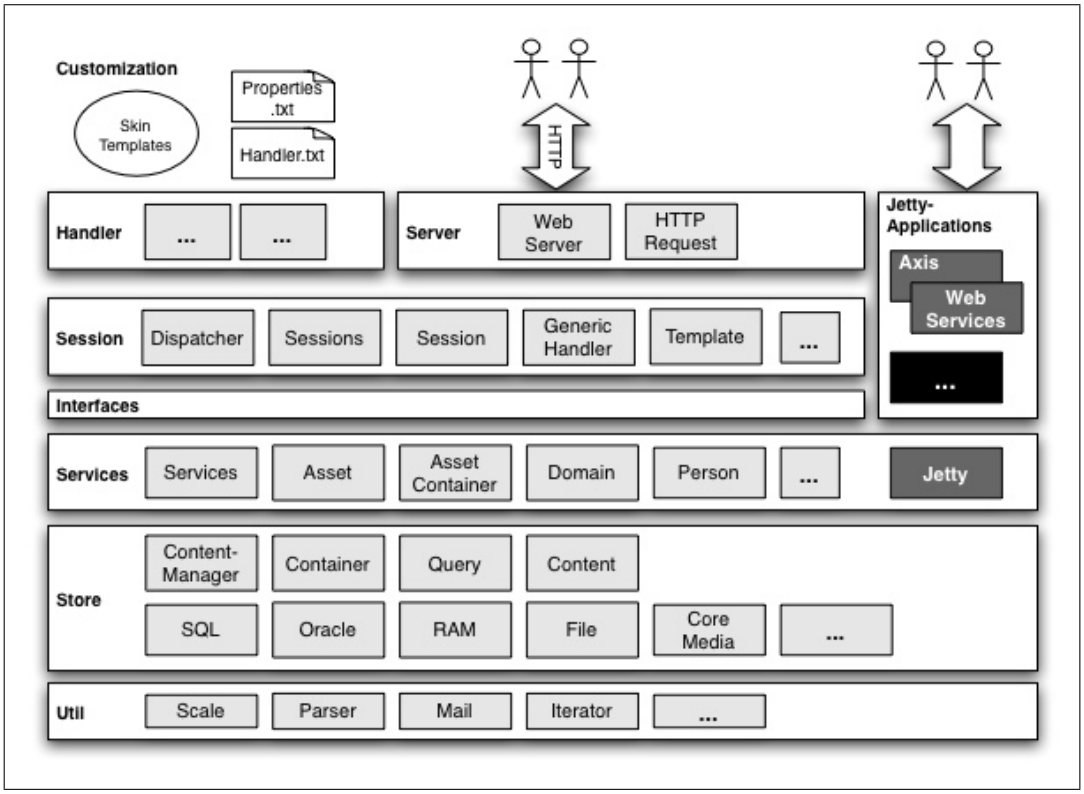

Abbildung 4.5: Erweiterte Architektur des Brokers (neue Bestandteile in dunkelgrau und schwarz) angelehnt an [18, Folie 6]

Ziel der Modifikation am Broker ist es, Jetty in die Service Schicht zu integrieren. Die Architektur über Jetty bricht mit der Broker-Architektur und ermöglicht mit einem Schlag die Nutzung der *Java Enterprise Edition* (J2EE) Technologien (siehe Abbildung 4.5). Mit der erweiterten Architektur können nicht nur die Web Services in Axis genutzt werden, sondern auch Servelts zum Darstellen von Inhalt oder *JSP (Java Server Pages) Standard Tag Library* (JSTL).

Jetty kann nicht überall im Broker-Code gestartet werden. Für die Integration ist es notwendig, daß die Broker-Services bereits laufen. Andererseits muss Jetty früh genug gestartet werden, so daß der Integrationsaufwand möglichst gering bleibt.

Die Broker Server, also der FTP-Server und der Broker eigene Web Server werden in der Klasse `Broker.java` in der Methode `postInit()` gestartet. Zu diesem Zeitpunkt sind die Services des Brokers bereits erreichbar, so daß hier auch Jetty gestartet werden kann.

Die Services werden vorher gestaretet. Sie können zum Beispiel in `AbstractExtension` abgegriffen und zum Initialisieren eines Singletons verwendet werden.

5 Realisierung der Web Service Schnittstelle

Damit ein Web Service überhaupt implementiert werden kann, muss zunächst eine Web Service Beschreibung in WSDL erstellt werden.

Auch wenn WSDL-Dateien in XML beschrieben werden und damit grundsätzlich von Menschen gelesen und verstanden werden können, sind sie doch oft zu umfassend, als daß eine Erstellung von Hand Sinn machen würde. Zur Unterstützung bei der Modellierung wird deshalb Eclipse mit den *Web Tools Platform* (WTP) [15] Erweiterungen verwendet.

Wenn der Web Service durch eine WSDL-Datei beschrieben ist, kann diese als Basis für die Generation von Java-Klassen mit Hilfe des Axis `wsdl2java` Code-Generators benutzt werden. Sobald die Java-Klassen vorhanden sind, kann die Implementation der Web Service Methoden beginnen.

Die Anwendung `java2wsdl` generiert neben den Java-Klassen auch zwei *Web Service Deployment Descriptor* (WSDD) Dateien, `deploy.wsdd` und `undeploy.wsdd`, die zum An- beziehungsweise Abmelden eines Web Services bei Jetty verwenden werden können.

5.1 WSDL - Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://broker.tuhh.de/09-2005/DocumentRetrieval/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://broker.tuhh.de/09-2005/DocumentRetrieval/"
  xmlns:intf="http://broker.tuhh.de/09-2005/DocumentRetrieval/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">


  <wsdl:types>
    <schema elementFormDefault="qualified"
      targetNamespace="http://broker.tuhh.de/09-2005/DocumentRetrieval/"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="StandardParameterType">
        <sequence>
          <element name="benutzerId" type="xsd:string" />
          <element maxOccurs="unbounded" name="rolle"
            type="xsd:string" />
        </sequence>
      </complexType>
      <complexType name="KategorieDescriptorType">
        <sequence>
          <element name="kategorieName" type="xsd:string" />
          <element name="kategorieId" type="xsd:string" />
        </sequence>
      </complexType>
      <complexType name="KategorieListeType">
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0"
            name="kategorieDescriptor" type="impl:KategorieDescriptorType" />
        </sequence>
      </complexType>
      <complexType name="WurzelKategorienRequestType">
```

```
        <sequence>
          <element name="standardParameters"
            type="impl:StandardParameterType" />
        </sequence>
      </complexType>
      <element name="getWurzelKategorieRequest"
        type="impl:WurzelKategorienRequestType" />
      <complexType name="WurzelKategorieListeType">
        <sequence>
          <element name="kategorieListe"
            type="impl:KategorieListeType" />
        </sequence>
      </complexType>
      <element name="getWurzelKategorieResponse"
        type="impl:WurzelKategorieListeType" />
      <element name="getKategorieResponse"
        type="impl:KategorieDatenType" />
    </schema>
  </wsdl:types>

  <wsdl:message name="getWurzelKategorienRequest">
    <wsdl:part element="impl:getWurzelKategorieRequest"
      name="getWurzelKategorieRequest" />
  </wsdl:message>
  <wsdl:message name="getWurzelKategorienResponse">
    <wsdl:part element="impl:getWurzelKategorieResponse"
      name="getWurzelKategorieResponse" />
  </wsdl:message>

  <wsdl:portType name="Documents">
    <wsdl:operation name="getWurzelKategorien"
      parameterOrder="getWurzelKategorieRequest">
      <wsdl:input message="impl:getWurzelKategorienRequest"
        name="getWurzelKategorienRequest" />
      <wsdl:output message="impl:getWurzelKategorienResponse"
        name="getWurzelKategorienResponse" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="DocumentsSOAPSoapBinding"
    type="impl:Documents">
    <wsdlsoap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getWurzelKategorien">
      <wsdlsoap:operation
        soapAction="http://localhost:8070/axis/services/Documents/getWurzelKategorien" />
      <wsdl:input name="getWurzelKategorienRequest">
        <wsdlsoap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="getWurzelKategorienResponse">
        <wsdlsoap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="Documents">
    <wsdl:port binding="impl:DocumentsSOAPSoapBinding"
      name="DocumentsSOAP">
      <wsdlsoap:address
        location="http://127.0.0.1:8070/axis/services/DocumentsSOAP" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

(Das obige Beispiel ist gekürzt. Es enthält nur die Web Service Methoden um die Wurzelkate-

gorien zu beziehen. Das vollständige WSDL-Dokument befindet sich im Anhang.)

5.2 Modifikationen am Broker

de.infoasset.broker.services.core.BrokerServices

Die folgende kleine Klasse stellt das Singleton dar, welches im Broker gestartet und in der SOAP-Implementation verwendet wird.

```
0  package de.infoasset.broker.services.core;

   import de.infoasset.broker.interfaces.core.*;

   public class BrokerServices {
5     private static Services services;

      protected BrokerServices(final Services services) {
         this.services =services;
      }
10
      public static Services getBrokerServices(){
         return services;
      }
   }
```

de.infoasset.broker.services.core.AbstractExtension

Hier wird eine Instanz der obigen Klasse erstellt und mit den Services beladen.

```
0     public void configure(Preferences preferences) {
         this.services = (IMPServices) preferences.getReference
         (Services.class.getName());
         new BrokerServices(services);   // SERVICES FUER JETTY
         ...
```

Nachdem durch die obigen Erweiterungen am Broker die Services grundsätzlich für jede Klasse des Brokers über das Singleton zugänglich sind, kann Jetty gestartet werden.

de.infoasset.broker.server.Broker (I)

Die folgende kleine Methode überprüft zunächst, ob eine Umgebungsvariable mit dem Namen `JETTY_WEBAPPS` vorhanden ist und liest ihren Wert. Der so übergebene Wert ist der Pfad zu dem Web-Applikationen Ordner für Jetty. Anschließend wird überprüft, ob das Verzeichnis vorhanden ist. Wenn alle Überprüfungen positiv ausgefallen sind, wird Jetty mit dem übergebenen Parameter gestartet.

```
0      void startJetty() {
           try {
               String jetpath = System.getenv("JETTY_WEBAPPS");
               if (jetpath == null || jetpath.equals("")) {
                   services.getLog().info
5                  ("jetty NOT STARTED: JETTY_WEBAPPS was not set or empty.");
                   return;
               } else {
                   File f = new File(jetpath);
                       if (f.isDirectory() && f.exists()) {
10                     Server server = new Server();
                       SocketListener listener = new SocketListener();
                       listener.setPort(8070);
                       server.addListener(listener);
                       server.addWebApplications(jetpath);
15                     server.start();
                       services.getLog().info("jetty STARTED.");
```

```
                    return;
                } else {
                    services.getLog().info
                    ("jetty NOT STARTED: JETTY_WEBAPPS contains an illegal path.");
                    return;
                }
            }
        }  catch (Exception ex) {
            services.getLog().info("jetty ERROR:" +ex.toString());
            ex.printStackTrace();
        }
    }
```

de.infoasset.broker.server.Broker (II)

In der Methode `postInit()` wird die obige `startJetty()` - Methode schließlich aufgerufen und startet den Server. Jetty ist hier zwischen den *File Transfer Protocol* (FTP) [6, Seite 473 ff.] Server und den Broker Web Server (im Code-Ausschnitt nicht mehr zu sehen) eingeschoben worden.

```
   public void postInit() {
      broker = this;
      log = Logger.getLogger("Broker");
      this.sessions.initUsers();
      this.sessions.setOnlineChecker();
      log.info("Services version " + services.getVersion()
         + " successfully loaded.");
      ftpServer.setAllowAnonymous(false);
      ftpServer.start();
      services.getLog().info("ftpServer started");
      startJetty();         // JETTY STARTEN
      ...
```

6 Zusammenfassung

In dieser Arbeit wurde gezeigt, wie das Entwickler-Portal mit einer Web Service Schnittstelle erweitert werden kann. Dazu wurde zunächst das Government Gateway vorgestellt und danach der Broker mit den Erweiterungen, die ihn zum Entwickler-Portal machen. Es wurden die Web Service Technologien SOAP, WSDL und UDDI aufgeführt und erläutert. Für die Integration in das Entwickler-Portal wurden Komponenten für die Schnittstelle ausgewählt und schließlich zur Integration verwendet.

Das GovernmentGateway ist die E-Government-Plattform von Dataport. Über sie können Fachverfahren von Dataport Bürgern, Unternehmen und anderen Behörden zur Verfügung gestellt werden. Das Entwickler-Portal wird in diesem Zusammenhang Informationen für die Software-Entwicklungen von Dataport den Entwicklern zur Verfügung stellen. In ihm können einzelne Dokumente gezielt bestimmten Benutzergruppen zur Verfügung gestellt werden. Dokumente können untereinander verknüpft werden, um so die Abhängigkeiten von bestimmten Softwareteilen zu modellieren.

Das Entwickler-Portal soll als *backend* Anwendung ähnlich eines Fachverfahrens aufgestellt und in das GovernmentGateway integriert werden. Da Fachverfahren über Web Services mit dem Gateway kommunizieren, ist eine Web Service Schnittstelle für das Entwickler-Portal eine Vorgabe von Dataport.

Web Services sind plattform unabhängige Software Schnittstellen die, die konkrete Implementation eines Dienstes verdecken. Sie bedienen sich offenen Standardtechnologien wie XML-Schema, XML und SOAP für den Zugriff auf die Services und zur Beschreibung der Schnittstelle.

Da der Broker über keine Schnittstellen für Fremdanwendungen verfügt, muss ein eigener Weg für die Erweiterung gefunden werden. Hierzu ist es notwendig einen Zugriff auf die Broker-Services zu bekommen, da mit ihnen ein Zugriff auf die Business-Objekte des Brokers möglich ist. Um den Broker mit der Web Service Schnittstelle zu erweitern, wird der in Java geschriebene Web Server und Servlet-Container Jetty in den Broker integriert. Er kann zusammen mit Apache Axis als Web-Applikation Web Services zur Verfügung stellen. Axis liefert zudem ein Toolkit, mit dem aus WSDL-Beschreibungen Java Klassen generiert werden können.

Durch die Verwendung von Jetty können nicht nur Web Services mit dem Broker realisiert werden, zusätzlich können eine ganze Reihe anderer Java 2 Enterprise Edition (J2EE) Technologien verwendet werden, dazu zählen auch Servlets und JSP (Java Server Pages) Standard Tag Library (JSTL).

Die Implementation der SOAP-Methoden mit Axis ging gut und intuitiv. Es traten jedoch zwei Probleme im Zusammenhang mit dem Axis Java Klassen Generator auf, die im folgenden erläutert werden.

Axis 1.2.1 Probleme mit Groß- und Kleinschreibung

Es stellte sich heraus, daß Axis 1.2.1 (erschienen am 15. Juni 2005) ein Problem mit Requests hat, dessen XML-Elemente einen Namen mit großem Anfangsbuchstaben haben. Beim Analysieren von Web Service Methoden-Namen verliert Axis die Groß- und Kleinschreibung des ersten Buchstaben. Wird später eine Methode auf dem Server angefragt, kommt es zu einem Fehler,

wenn der erste Buchstabe des Methoden-Namens groß war. Axis hat den ersten Buchstaben fälschlicherweise klein dargestellt und ist nun nicht mehr in der Lage, die richtige Methode zu finden.

Das Problem kann in der Version auf zwei Arten umgangen werden:

Ändern der WSDL Durch das Ändern der Element-Namen in der WSDL-Beschreibung kann verhindert werden, daß der Fehler in Axis aktiv wird. Dazu ist es notwendig, alle Element-Namen so zu ändern, daß sie mit einem kleinen Buchstaben beginnen.

Modifikation von Axis Axis ist eine offene Software. Der Quellcode ist in den Archiven von Axis jeweils enthalten. Damit ist es möglich, Axis selber bis zu einem offiziellen Bugfix oder neuem Release zu *patchen*.

Je nach Einsatzgebiet und Umgebung des Web Services ist die eine oder die andere Lösung vorzuziehen. Ist zum Beispiel schon die WSDL-Beschreibung an viele Kunden verteilt, ist es möglicherweise besser, Axis von Hand zu *patchen*. Wenn hingegen der Kundenkreis klein ist und keine Veränderung an Axis vorgenommen werden soll, dann mus die WSDL-Datei angepasst werden.

Bis zum Abschluß dieser Arbeit gab es kein offizielles Release oder Bugfix, welches diesen Fehler behebt.

Axis 1.2.1 Probleme mit nicht gesetztem `elementFormDefault` Attribut

Während der Kommunikation mit einem *.net* Client trat ein Problem auf, welches sich auf das Fehlen eines `elementFormDefault="qualified"` Attributes im `Schema` Element des ursprünglichen WSDL-Dokuments zurückführen ließ.

Mit `elementFormDefault="qualified"` wird festgesetzt, daß jede zu dem Schema konforme XML-Instanz die Elemente aus dem Schema mit ihrem Namespace qualifizieren muss.

Scheinbar nimmt Axis `wsdl2java` als Standardwert `unqualified` an, wenn gar kein `elementFormDefault` im Schema-Element der WSDL-Datei eingetragen ist. Fälschlicherweise trägt es dann jedoch in dem über den Server abrufbaren WSDL-Dokument ein `qualified` in das Attribut ein. Der *.net* Client ließt das `qualified` und Behandelt die Ein- und Ausgaben entsprechenderweise. Hier tritt dann richtigerweise ein Fehler auf, wenn XML-Instanzen empfangen werden, die tatsächlich `unqualified` sind.

Die Kommunikation läuft reibungslos, wenn vor dem Generieren der Klassen einmal das WSDL-Dokument auf Vorhandensein des `elementFormDefault="qualified"` Attributs überprüft wird und dieses notfalls berichtigt wird.

6.1 Ausblick

Mit dem Erweitern des Entwickler-Portals um Jetty und Axis wurden die Möglichkeiten des Brokers stark erweitert. Mit einem Schlag können JSTL, Servelts und Web Services zusammen mit dem Broker verwendet werden. Wie Web Services verwendet werden können, wurde in dieser Arbeit gezeigt. Es können auf diese Art auch weitere Web Services implementiert und so weitere Broker-Dienste angeboten werden.

Mit Jetty als Middleware eröffnet sich die Möglichkeit, die J2EE-Technologien für beziehungsweise mit den Broker Services zu nutzen. Es erscheint daher lohnenswert, über den weitergehenden

Nutzen der erweiterten Architektur nachzudenken. Jetty gilt als ein äußerst leistungsfähiger und robuster Web Server, der sicherlich auch mehr Aufgaben als die ausschließliche Web Service Verarbeitung übernehmen könnte.

Kritischer ist die Übertragung von Dokument-Anhängen als Binäranhang in Web Services zu betrachten und auch die vermeintliche „Plattform-Unabhängigkeit“ von Web Services.

Interoperabilität von Web Services

Während die Web Service Technologie als absolut plattformunabhängig konzipiert ist, sieht die Realität mit den Toolkits und Anwendungen ganz anders aus. Dabei sind es unterschiedliche Annahmen und Fehler in der Software, die die Interoperabilität stören.

Die während dieser Arbeit aufgetretenen Interoperabilitätsprobleme ließen sich alle auf ein Fehlverhalten von Axis zurückführen. Die Fehler in der lose gekoppelten Umgebung zu finden, ist nicht einfach. Meist bleibt es dem Entwickler verborgen, was in den Tiefen der Toolkits tatsächlich geschieht, selbst wenn im Falle wie mit Axis der Quellcode mitgeliefert wird. Es bleibt einem Entwickler also nichts anderes übrig, als sich eines Proxies für die Überprüfung der Richtigkeit der Kommunikation zu bedienen.

Diese mühsame Fehlersuche kann viel Zeit beanspruchen. Und wenn der Fehler gefunden ist, kann er mitunter nicht einmal ausgeräumt werden, da er tief im Inneren einer fremden Komponente verborgen ist. Es bleibt also nur die Möglichkeit eines Workarounds. Auf diese Art passt sich der Entwickler wieder den Vorgaben einiger Hersteller an.

Es bleibt zu hoffen, daß Probleme in der Interoperabilität in der Zukunft ausgeräumt und die Fehler in der Software zügiger als bisher verbessert werden. Im Falle von Axis sind jetzt vier Monate vergangen, wenige Tage nach dem Release der Version 1.2 wurde der Fehler schon bekannt. Behoben ist er bisher nicht. Um eine echte Plattformunabhängigkeit herzustellen, reicht es nicht, Spezifikationen aufzustellen, es muss auch ein gemeinsames Verständnis darüber vorhanden sein.

Versionierung von Web Services

Im Entwickler-Portal wird jedes einmal veröffentlichte Dokument aufgehoben und sobald es durch eine neuere Version ersetzt wird in die Historie verschoben. Eingetragene Benutzer sollen sich über Veränderungen an Dokumenten, zum Beispiel einer neueren Version, über Notifications benachrichtigen lassen können. Zusätzlich können Dokumenten von vornherein Gültigkeitszeiträume zugewiesen werden. Auf diese Art sollen Web Service Entwickler und Nutzer gleichermaßen ihre Anwendungen rechtzeitig an Veränderungen anpassen können.

Der Web Service aus dieser Arbeit, für den Austausch von Dokumenten und Teilbäumen der Wissenslandkarte, ist über seine URL, die ein Datum enthält, versioniert.

Ein Problem heutiger Web Service Definitionen ist, daß nicht festgelegt ist, wie die Evolution, beziehunsweise die Versionierung, einzelner Web Services vonstatten gehen soll. Allein im World Wide Web Consortium [41] gibt es drei Gruppen [49] die sich mit dem Thema beschäftigen.

Aus der fehlenden Spezifizierung für Web Service Versionen können große Probleme entstehen. In Web Service Orientierten Umgebungen stehen die Web Services, auf der die eigene Anwendung basiert, mitunter unter der Administration Dritter. Diese können Veränderungen am Web Service Interface vornehmen, wodurch die eigene Anwendung nicht mehr funktioniert. Es ist auch möglich, daß sich nicht das Web Service Interface ändert, sondern die Implementation, so daß Daten anders verarbeitet werden. Durch die Veränderung der Implementation könnten Seiteneffekte auftreten, die von der eigenen Anwendung nicht erwünscht sind, aber aufgrund des selben Interfaces auch nicht erkannt werden können. Mit zunehmender Zahl von Web Service Nutzern wird es zudem schwieriger einen Web Service so zu überarbeiten, daß alle Nutzer den Veränderungen zustimmen können.

Die Arbeit [34] beschäftigt sich weitergehend mit der Problematik der Versionierung von Web Services. Die vorgeschlagenen Ansätze decken die beiden oben angesprochenen Problemszenarien ab. Die Implementationen werden über Hashwerte abgesichert. Versionsnummern werden in Headerfelder von SOAP-Nachrichten aufgenommen. Für die Umsetzung ist allerdings die Anpassung der Infrastruktur, unter anderem der Web Server, notwendig.

Web Services mit Anhängen und Netzwerklast

Zur Zeit werden die Datei-Anhänge von Dokumenten über die Web Services als Base64 [22] kodierte Nachrichten an das GovernmentGateway übermittelt. Um eine Datei so zu übertragen, müssen die Binärinhalte für den Transport umkodiert werden. Auf der Empfängerseite muss der Dateiinhalt dann wieder zurück in das ursprüngliche Format gewandelt werden. Aufgrund des Marshallings und Unmarshallings wird Prozesszeit verbraucht. Durch den Transfer in Base64 über SOAP wird das Netzwerk stark belastet, da der gesamte Binärinhalt der Datei so umkodiert werden muss, daß nur Buchstaben, Zahlen, + und / verwendet werden, um den Binärinhalt darzustellen. Durch das Marshalling vergrößert sich also die tatsächlich über die Leitung übermittelte Nachricht, da nur ein Bruchteil der möglichen Symbole zum Übertragen der Nachricht verwendet werden kann (circa 64 von 256 Symbolen, [22]). Dies kann, insbesondere wenn viele oder große Dateien übermittelt werden, zu einer erheblichen Netzwerklast führen.

Für den Transport von Binärdateien gibt es überdies bereits viel effizientere und bewährtere Protokolle wie zum Beispiel das *File Transfer Protocol* (FTP) [6, Seite 473 ff.]. Es wäre beispielsweise denkbar, einen FTP-Dienst einzurichten, der Zugriff auf die Broker-Dateien erhält und über ein *Geheimnis* überprüfen kann, ob ein Benutzer berechtigterweise auf eine Datei zugreift. Dieses *Geheimnis* wird vom Broker erstellt, wenn ein Benutzer auf einen Dateianhang zugreifen will. Zusätzlich gibt er einen Link an den Benutzer, mit dem er nach der Datei bei dem Dienst anfragen kann. Falls das *Geheimnis* authentisch ist, wird der Zugriff gewährt, andernfalls nicht. Nach einmaliger Benutzung wird das *Geheimnis* zerstört und kann nicht wieder verwendet werden. Das Mitteilen der notwendigen Daten kann wie bisher über Web Services von statten gehen, einzig der Dateitransfer läuft über das effizientere FTP-Protokoll.

Web Services sind nicht dazu gedacht, Dateien zu übermitteln. Das Übermitteln von Dateien in einer Base64 kodierten Nachricht stellt eher eine Erweiterung für Web Services dar. Für den täglichen Gebrauch in einer produktiven Umgebung scheinen sie jedoch zu uneffektiv.

Literaturverzeichnis

[1] Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth, John Miller, *Adding Semantics to Web Services Standards*, Large Scale Distributed Information Systems (LSDIS) Lab Department of Computer Science, University of Georgia Athens, GA 30602

[2] Apache Foundation, Axis: URL: `http://ws.apache.org/axis/` *Zuletzt abgerufen am: 11.08.2005.*

[3] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation, University Of California, Irvine, 2000

[4] Bea: URL: `http://e-docs.bea.com/wls/docs61/webServices/overview.html#1023892` *Zuletzt abgerufen am: 17.08.2005.*

[5] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, *Business Process Execution Language for Web Services*, Version 1.1, May 5th, 2003 URL: `ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf` *Zuletzt abgerufen am: 11.08.2005.*

[6] Douglas E. Comer, *Computernetzwerke und Internets*, Pearson Studium, 2002, ISBN 3-8273-7023-X

[7] Sebastian Bossung, Hans-Werner Sehring, Joachim W. Schmidt, *Conceptual Content Management for Enterprise Web Services*, Software Technology and Systems Institute (STS), Hamburg University of Science and Technology (TUHH)

[8] Bruce Schneier, *Crypto-Gram Newsletter*, June 15th, 2000, URL: `http://www.schneier.com/crypto-gram-0006.html` *Zuletzt abgerufen am: 22.08.2005.*

[9] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew McDermott, David Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, Katia Sycara, *DAML-S: Web Service Description for the Semantic Web*

[10] DARPA Agent Markup Language Homepage (DAML) URL: `http://www.daml.org` *Zuletzt abgerufen am: 09.09.2005.*

[11] Dataport: Wer sind wir URL: `http://www.dataport.de` *Zuletzt abgerufen am: 09.09.2005.*

[12] Dataport, GovernmentGateway Präsentation, Folie 2

[13] Dataport, Dataport-Plattform, *Motivation, Zielsetzungsumfang und Architekturskizze der E-Government-Plattform*, April 2004

[14] Jun Zhang, *Design and Implementation of a Web Service Development Portal - The case study DATAPORT*, Bachelor Thesis, Hamburg University of Science and Technology, Software System Institute, 2005

[15] Eclipse: Web Tools Platform Project : URL: `http://www.eclipse.org/webtools/index.html` *Zuletzt abgerufen am: 02.09.2005.*

[16] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Entwurfsmuster*, Addison Wesley Verlag, 2001, ISBN 3-8273-1862-9

[17] infoAsset AG, infoAsset Broker: URL: `http://www.infoasset.de/contents/products/index.htm` *Zuletzt abgerufen am: 11.08.2005.*

[18] infoAsset AG, *Entwicklerschulung infoAsset Broker 2.0*, PowerPoint-Präsentation 040616-MaWe-infoAsset-Entwicklerschulung.ppt

[19] Heiner Stuckenschmidt, Frank van Harmelen, *Information Sharing on the Semantic Web*, Springer Verlag, 2005, ISBN 3-540-20594-2

[20] International Business Machines (IBM), URL: `http://www.ibm.com/` *Zuletzt abgerufen am: 19.08.2005.*

[21] International Business Machines (IBM), New to SOA and Web services URL: `http://www-128.ibm.com/developerworks/webservices/newto/websvc.html#1` *Zuletzt abgerufen am: 17.08.2005.*

[22] Internet Engineering Task Force (IETF), Request for Comments (RFC) 2045, *Multipurpose Internet Mail Extensions (MIME)*, Part One: Format of Internet Message Bodies, URL: `http://www.ietf.org/rfc/rfc2045.txt` *Zuletzt abgerufen am: 10.10.2005.*

[23] Matt Bishop, *Introduction to Computer Security*, Addison Wesley Verlag, 2005, ISBN 0-321-24744-2

[24] David Chappell, Tyler Jewell, *Java Web Services*, O'Reilly, 2002, ISBN 0-596-00269-6

[25] Eric Armstrong, Stephanie Bodoff, Debbie Carson, *The Java Web Services Tutorial*, Addison Wesley Verlag, 2002, ISBN 0-201-76811-9

[26] Microsoft MSDN, DCOM Architecture URL: `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp` *Zuletzt abgerufen am: 19.08.2005.*

[27] Microsoft URL: `http://www.microsoft.com/` *Zuletzt abgerufen am: 19.08.2005.*

[28] MortBay Consulting, Jetty: URL: `http://jetty.mortbay.org/jetty/` *Zuletzt abgerufen am: 11.08.2005.*

[29] MortBay Consulting: URL: `http://www.mortbay.com/mortbay/mbindex.html` *Zuletzt abgerufen am: 11.08.2005.*

[30] Mozilla: URL: `http://www.mozilla.org/projects/webservices/` *Zuletzt abgerufen am: 17.08.2005.*

[31] Object Management Group, Catalog of OMG CORBA/IIOP Specifications URL: `http://www.omg.org/technology/documents/corba_spec_catalog.htm` *Zuletzt abgerufen am: 19.08.2005.*

[32] Organization for the Advancement of Structured Information Standards: Universal Description, Discovery and Integration, URL: `http://uddi.org` *Zuletzt abgerufen am: 22.08.2005.*

[33] Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, *Perspectives on Web Services*, Springer Verlag, 2003, ISBN 3-540-00914-0

[34] Robert Steele, Takahiro Tsubono, *Reserving Immutable Services through Web Service Implementation Versioning*, University of Technology, Sydney, Australia

[35] Systems Applications Products in Data Processing (SAP) AG, URL: `http://www.sap.com/` *Zuletzt abgerufen am: 19.08.2005.*

[36] Sun Microsystems, URL: `http://www.sun.com/` *Zuletzt abgerufen am: 19.08.2005.*

[37] Sun Microsystems, Java Remote Method Invocation (Java RMI) URL: `http://java.sun.com/products/jdk/rmi/index.jsp` *Zuletzt abgerufen am: 19.08.2005.*

[38] systinet: *Web Services: A Practical Introduction*, A Systinet White Paper, 2003, Seite 5

[39] UDDI Spec Technical Committee Specification, July 19th, 2002, URL: `http://uddi.org/pubs/uddi-v3.00-published-20020719.htm` *Zuletzt abgerufen am: 22.08.2005.*

[40] UN/CEFACT and OASIS: *ebXML Business Process Specification Schema*, Version 1.01, URL: `http://www.ebxml.org/specs/ebBPSS.pdf` *Zuletzt abgerufen am: 02.09.2005.*

[41] World Wide Web Consortium URL: `http://www.w3.org/` *Zuletzt abgerufen am: 19.08.2005.*

[42] World Wide Web Consortium, Extensible Markup Language (XML) URL: `http://www.w3.org/XML/` *Zuletzt abgerufen am: 19.08.2005.*

[43] Steve DeRose, Eve Maler, David Orchard, World Wide Web Consortium, *XML Linking Language (XLink)*, Version 1.0, June 27th, 2001, URL: `http://www.w3.org/XML/Linking` *Zuletzt abgerufen am: 10.10.2005.*

[44] World Wide Web Consortium, *XML Schema*, URL: `http://www.w3.org/XML/Schema` *Zuletzt abgerufen am: 19.08.2005.*

[45] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, World Wide Web Consortium, *Simple Object Access Protocol (SOAP) 1.1*, May 8th, 2000, URL: `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/` *Zuletzt abgerufen am: 06.09.2005.*

[46] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, World Wide Web Consortium, *SOAP Version 1.2 Part 2: Adjuncts*, June 24th, 2003, URL: `http://www.w3.org/TR/2003/REC-soap12-part2-20030624/` *Zuletzt abgerufen am: 06.09.2005.*

[47] Deborah L. McGuinness, Frank van Harmelen, World Wide Web Consortium, *OWL Web Ontology Language Overview*, February, 10th 2004, URL: `http://www.w3.org/TR/owl-features/` *Zuletzt abgerufen am: 06.09.2005.*

[48] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, Sanjiva Weerawarana, World Wide Web Consortium, *Web Services Description Language (WSDL) 1.2*, URL: `http://www.w3.org/TR/2003/WD-wsdl12-20030611/` *Zuletzt abgerufen am: 19.08.2005.*

[49] David Booth, Canyang Kevin Liu, World Wide Web Consortium, *Web Services Description Language (WSDL)*, Version 2.0 Part 0: Primer, 5.2 Web Service Versioning, URL: `http://www.w3.org/TR/wsdl20-primer/#adv-versioning` *Zuletzt abgerufen am: 06.09.2005.*

[50] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, World Wide Web Consortium, *Web Services Choreography Description Language*, Version 1.0, December 19th, 2004, URL: `http://www.w3.org/TR/ws-cdl-10/` *Zuletzt abgerufen am: 06.09.2005.*

Anhang

Documents.wsdl - Implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://broker.tuhh.de/09-2005/DocumentRetrieval/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Documents"
   targetNamespace="http://broker.tuhh.de/09-2005/DocumentRetrieval/">
   <wsdl:types>
      <xsd:schema elementFormDefault="qualified" targetNamespace="http://broker.tuhh.de
          /09-2005/DocumentRetrieval/" xmlns:tns="http://broker.tuhh.de/09-2005/
          DocumentRetrieval/">
         <xsd:complexType name="dokumentDatenType">
            <xsd:sequence>
               <xsd:element name="DokumentName" type="xsd:string"
                  minOccurs="1" maxOccurs="1">
               </xsd:element>
               <xsd:element name="DokumentId" type="xsd:string"
                  minOccurs="1" maxOccurs="1">
               </xsd:element>
               <xsd:element name="GueltigVon" type="xsd:date"
                  minOccurs="0" maxOccurs="1">
               </xsd:element>
               <xsd:element name="GueltigBis" type="xsd:date" maxOccurs="1" minOccurs
                   ="0"></xsd:element>
               <xsd:element name="BegruendungFuerUnGueltigkeit"
                  type="xsd:string">
               </xsd:element>
               <xsd:element name="Beschreibung"
                  type="xsd:string">
               </xsd:element>
               <xsd:element name="Dokumentart" type="xsd:string"></xsd:element>
               <xsd:element name="KategorienListe" type="tns:kategorieListeType">
                  </xsd:element>
               <xsd:element name="Schlagworte" type="xsd:string">
               </xsd:element>
               <xsd:element name="UnterschiedeZurVorversion"
                  type="xsd:string">
               </xsd:element>
               <xsd:element name="Zweck" type="xsd:string"></xsd:element>
               <xsd:element name="URL" type="xsd:string"></xsd:element>
               <xsd:element name="TestUrl" type="xsd:string"></xsd:element>
               <xsd:element name="Verfuegbarkeit"
                  type="xsd:string">
               </xsd:element>
               <xsd:element name="WirdVerwendetVonListe" type="tns:dokumentListenType">
                  </xsd:element>
               <xsd:element name="FreigegebenVon" type="xsd:string">
                  </xsd:element>
               <xsd:element name="FreigegebenAm" type="xsd:date" maxOccurs="1" minOccurs
                   ="0"></xsd:element>
               <xsd:element name="SichtbarAb" type="xsd:date" maxOccurs="1" minOccurs
                   ="0"></xsd:element>
               <xsd:element name="LetzteAenderung"
                  type="xsd:date" maxOccurs="1" minOccurs="0">
               </xsd:element>
               <xsd:element name="AutorenListe" type="tns:personListeType">
                  </xsd:element>
               <xsd:element name="EigentuemerListe" type="tns:personListeType">
```

```
            </xsd:element>
         <xsd:element name="AnsprechpartnerFachlichListe" type="tns:personListeType
             ">
            </xsd:element>
         <xsd:element name="AnsprechpartnerTechnischListe" type="tns:
             personListeType">
            </xsd:element>
         <xsd:element name="FachlicheKontexteListe" type="tns:kategorieListeType">
            </xsd:element>
         <xsd:element name="RechtlicheGrundlagenListe" type="tns:kategorieListeType
             ">
            </xsd:element>
         <xsd:element name="VerwendeteDokumenteListe" type="tns:dokumentListenType
             ">
            </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="dokumentDescriptorType">
      <xsd:sequence>
         <xsd:element name="DokumentName"
            type="xsd:string">
         </xsd:element>
         <xsd:element name="DokumentId" type="xsd:string"></xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="dokumentListenType">
      <xsd:sequence>
         <xsd:element name="DokumentDescriptor"
            type="tns:dokumentDescriptorType" minOccurs="0"
            maxOccurs="unbounded">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="dokumentList"
      type="tns:dokumentListenType">
   </xsd:element>
   <xsd:complexType name="standardParameterType">
      <xsd:sequence>
         <xsd:element name="BenutzerId" type="xsd:string"
            minOccurs="1" maxOccurs="1">
         </xsd:element>
         <xsd:element name="Rolle" type="xsd:string"
            minOccurs="1" maxOccurs="unbounded">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="standardParameter"
      type="tns:standardParameterType">
   </xsd:element>
   <xsd:element name="getDokumentResponse"
      type="tns:dokumentDatenType">
   </xsd:element>
   <xsd:element name="getKategorieResponse"
      type="tns:kategorieDatenType">
   </xsd:element>
   <xsd:element name="getDokumentRequest" type="tns:dokumentRequestType">
      </xsd:element>
   <xsd:element name="getWurzelKategorieRequest" type="tns:
       wurzelKategorienRequestType">
      </xsd:element>
   <xsd:element name="getKategorieRequest" type="tns:kategorieRequestType">
      </xsd:element>
   <xsd:element name="getWurzelKategorieResponse"
      type="tns:wurzelKategorieListeType">
   </xsd:element>
   <xsd:complexType name="kategorieDatenType">
      <xsd:sequence>
         <xsd:element name="KategorieId" type="xsd:string"
            minOccurs="1">
         </xsd:element>
```

```
            <xsd:element name="KategorieName" type="xsd:string"
               minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="Beschreibung" type="xsd:string"
               minOccurs="1" maxOccurs="1">
            </xsd:element>
            <xsd:element name="SynonymListe" type="tns:synonymListeType">
               </xsd:element>
            <xsd:element name="SuperKategorie"
               type="tns:kategorieDescriptorType">
            </xsd:element>
            <xsd:element name="GenerellereKategorienListe" type="tns:
                kategorieListeType">
               </xsd:element>
            <xsd:element name="SubKategorienListe" type="tns:kategorieListeType">
               </xsd:element>
            <xsd:element name="SpeziellereKategorienListe" type="tns:
                kategorieListeType">
               </xsd:element>
            <xsd:element name="QuerverweiseKategorienListe" type="tns:
                kategorieListeType">
               </xsd:element>
            <xsd:element name="DokumentListe"
               type="tns:dokumentListenType">
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="kategorieDescriptorType">
         <xsd:sequence>
            <xsd:element name="KategorieName"
               type="xsd:string">
            </xsd:element>
            <xsd:element name="KategorieId" type="xsd:string"></xsd:element>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="wurzelKategorieListeType">
         <xsd:sequence>
            <xsd:element name="KategorieListe" type="tns:kategorieListeType">
               </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="personDescriptorType">
         <xsd:sequence>
            <xsd:element name="Vorname" type="xsd:string"></xsd:element>
            <xsd:element name="Nachname" type="xsd:string"></xsd:element>
            <xsd:element name="PersonId" type="xsd:string"></xsd:element>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:element name="getFileResponse" type="tns:fileResponseType">
         </xsd:element>
      <xsd:element name="getFileRequest" type="tns:dokumentRequestType">
         </xsd:element>
      <xsd:complexType name="kategorieListeType">
         <xsd:sequence>
            <xsd:element name="KategorieDescriptor" type="tns:kategorieDescriptorType"
                maxOccurs="unbounded" minOccurs="0"></xsd:element>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="personListeType">
         <xsd:sequence>
            <xsd:element name="PersonDescriptor"
               type="tns:personDescriptorType" maxOccurs="unbounded" minOccurs="0">
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="kategorieRequestType">
```

```
            <xsd:sequence>
               <xsd:element name="StandardParameters" type="tns:standardParameterType"></
                   xsd:element>
               <xsd:element name="KategorieId" type="xsd:string"></xsd:element>
            </xsd:sequence>
         </xsd:complexType>

         <xsd:complexType name="fileResponseType">
            <xsd:sequence>
               <xsd:element name="Dateiname" type="xsd:string"></xsd:element>

               <xsd:element name="DateiInhalt" type="xsd:base64Binary"></xsd:element></
                   xsd:sequence>
         </xsd:complexType>

         <xsd:complexType name="dokumentRequestType">
            <xsd:sequence>
               <xsd:element name="StandardParameters" type="tns:standardParameterType"></
                   xsd:element>
               <xsd:element name="DokumentId" type="xsd:string"></xsd:element>
            </xsd:sequence>
         </xsd:complexType>

         <xsd:complexType name="synonymListeType">
            <xsd:sequence>
               <xsd:element name="Synonym" type="xsd:string" maxOccurs="unbounded"
                   minOccurs="0"></xsd:element>
            </xsd:sequence>
         </xsd:complexType>

         <xsd:complexType name="wurzelKategorienRequestType">
            <xsd:sequence>
               <xsd:element name="StandardParameters" type="tns:standardParameterType"></
                   xsd:element>
            </xsd:sequence>
         </xsd:complexType>

      </xsd:schema>
   </wsdl:types>

   <wsdl:message name="getDokumentResponse">
      <wsdl:part name="getDokumentResponse"
         element="tns:getDokumentResponse">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getDokumentRequest">
      <wsdl:part name="getDokumentRequest"
         element="tns:getDokumentRequest">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getWurzelKategorienResponse">
      <wsdl:part name="getWurzelKategorienResponse"
         element="tns:getWurzelKategorieResponse">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getWurzelKategorienRequest">
      <wsdl:part name="getWurzelKategorieRequest"
         element="tns:getWurzelKategorieRequest">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getKategorieResponse">
      <wsdl:part name="getKategorieResponse"
         element="tns:getKategorieResponse">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getKategorieRequest">
      <wsdl:part name="getKategorieRequest"
         element="tns:getKategorieRequest">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getFileResponse">
```

```
      <wsdl:part name="getFileResponse"
         element="tns:getFileResponse">
      </wsdl:part>
   </wsdl:message>
   <wsdl:message name="getFileRequest">
      <wsdl:part name="getFileRequest" element="tns:getFileRequest"></wsdl:part>
   </wsdl:message>
   <wsdl:portType name="Documents">

      <wsdl:operation name="getDokument">
         <wsdl:input message="tns:getDokumentRequest"></wsdl:input>
         <wsdl:output message="tns:getDokumentResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getWurzelKategorien">
         <wsdl:input message="tns:getWurzelKategorienRequest"></wsdl:input>
         <wsdl:output message="tns:getWurzelKategorienResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getKategorie">
         <wsdl:input message="tns:getKategorieRequest"></wsdl:input>
         <wsdl:output message="tns:getKategorieResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getFile">
         <wsdl:input message="tns:getFileRequest"></wsdl:input>
         <wsdl:output message="tns:getFileResponse"></wsdl:output>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="DocumentsSOAP" type="tns:Documents">
      <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="getDokument">
         <soap:operation
            soapAction="http://localhost:8070/axis/services/Documents/getDokument" />
         <wsdl:input>
            <soap:body use="literal" />
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getWurzelKategorien">
         <soap:operation
            soapAction="http://localhost:8070/axis/services/Documents/getWurzelKategorien
                " />
         <wsdl:input>
            <soap:body use="literal" />
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getKategorie">
         <soap:operation
            soapAction="http://localhost:8070/axis/services/Documents/getKategorie" />
         <wsdl:input>
            <soap:body use="literal" />
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getFile">
         <soap:operation
            soapAction="http://localhost:8070/axis/services/Documents/getFile" />
         <wsdl:input>
            <soap:body use="literal" />
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
```

```
   <wsdl:service name="Documents">
      <wsdl:port binding="tns:DocumentsSOAP" name="DocumentsSOAP">
         <soap:address
            location="http://localhost:8070/axis/services/DocumentsSOAP" />
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```