Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Ferdinand Tatzig

Entwicklung einer Webservice-basierten Programmierschnittstelle für ein Media-Asset-Management-System

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ferdinand Tatzig

Entwicklung einer Webservice-basierten Programmierschnittstelle für ein Media-Asset-Management-System

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informatik
Studienrichtung Softwaretechnik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. sc. pol. Wolfgang Gerken
Zweitgutachter : Prof. Dr. Olaf Zukunft

Abgegeben am 22. Februar 2008

Ferdinand Tatzig

Thema der Diplomarbeit

Entwicklung einer Webservice-basierten Programmierschnittstelle für ein Media-Asset-Management-System

Stichworte

document-literal, jadis.net, Java, MAMS, Media-Asset-Management-System, PHP, php/Java bridge, Programmierschnittstelle, Schnittstelle, SOAP, SSL, Webservice, WSDL, XML

Kurzzusammenfassung

Die zweitwerk software engineering GmbH entwickelt und vertreibt das Media-Asset-Management-System jadis.net. Bei jadis.net handelt es sich um eine Anwendung, mit der digitale Mediendaten wie zum Beispiel Bilder, Filme, Musikdateien oder Texte verwaltet werden können. jadis.net ist eine in der Programmiersprache PHP entwickelte serverseitige Software, die über einen Webbrowser bedient wird.

Mittlerweile besteht bei Benutzern der Bedarf, Teile der Funktionalität von jadis.net in ihren eigenen Anwendungen zu nutzen. Daher wurde die Entwicklung einer Webservice-basierten Programmierschnittstelle beschlossen. Die Operationen der Programmierschnittstelle sollen dabei in einem WSDL-Dokument beschrieben werden, sodass entsprechende Werkzeuge den Code für die Client-Anwendungen automatisch generieren können. Die Operationen werden dann über ein Netzwerk eingeleitet, wobei der Nachrichtenaustausch über SOAP erfolgt.

Diese Diplomarbeit beschreibt den Entwicklungsprozess der Programmierschnittstelle von der Anforderungsanalyse, über die Auswahl einer geeigneten Architektur bis hin zum Erstellen des WSDL-Dokumentes und der Implementierung der notwendigen Klassen.

Ferdinand Tatzig

Title of Diploma Thesis

Developing a Web Service Based Application Programming Interface for a Media Asset Management System

Keywords

API, application programming interface, authentication, document-literal, jadis.net, Java, MAMS, media asset management system, PHP, php/Java bridge, SOAP, SSL, Web service, WSDL, XML

Abstract

zweitwerk software engineering GmbH (Ltd.) develops and sells the media asset management system jadis.net. jadis.net is an application which allows users to manage digital media data such as pictures, films, music files or texts. It is a serverside based software which can be used with a web browser. The PHP programming language was chosen for its implementation.

The current need of users to make part of the jadis.net functionality available to their own applications has led to the decision to develop a Web service based application programming interface (API). The operations of this API are to be defined in a WSDL document such that adequate tools can automatically generate the code needed by the client applications. The operations will be remotely invoked via SOAP messages.

This diploma thesis deals with the development process of the API. It describes the requirements analysis, the choice of an appropriate architecture and the implementation of the API. The last issue includes the definitions of the WSDL document as well as the actual source code.

Danksagung

Ich möchte mich für die Erstellung der Diplomarbeit bei den Mitarbeiterinnen und Mitarbeitern der zweitwerk software engineering GmbH und meinen Eltern für ihre Unterstützung bedanken.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Aufgabenstellung

Die zweitwerk sofware engineering GmbH[1] entwickelt und vertreibt das Media-Asset-Management-System (MAMS) jadis.net. Hierbei handelt es sich um eine über einen Webbrowser bedienbare Verwaltungssoftware für Mediendokumente wie zum Beispiel Bilder, Filme und Animationen. Die Software wird in der Regel von den Marketingabteilungen verschiedener Unternehmen sowie von Buchverlagen, Hochschulen und Hochschulbibliotheken verwendet. Für Hochschulen und ihre Bibliotheken existiert eine spezielle Version. jadis.net ist eine serverseitige Anwendung, die über PHP-Skripte auf eine Datenbank zugreift.

Mittlerweile besteht ein Bedarf für Anwendungen, die über Netzwerkverbindungen Zugriff auf die Daten von jadis.net haben. Kunden im Besitz von jadis.net-Lizenzen möchten zum Beispiel Anwendungen entwickeln oder erweitern, welche die Daten von jadis.net verarbeiten.

Hierfür muss jadis.net um eine Programmierschnittstelle erweitert werden, die die Daten zugänglich macht. Die Funktionen beziehungsweise Methoden dieser Schnittstelle müssen dabei entfernt aufrufbar sein. Eine als Webservice implementierte Schnittstelle würde dieser Anforderung genügen.

Die zweitwerk software engineering GmbH hat somit beschlossen, eine Programmierschnittstelle entwickeln zu lassen, die als Webservice den Zugriff auf die Daten von jadis.net gewährt. Die Operationen dieses Webservice sollen in einem WSDL-Dokument veröffentlicht werden. Die Kommunikation soll über den Austausch von SOAP-Nachrichten erfolgen. Die Entwicklung der Schnittstelle ist gleichzeitig das Thema dieser Diplomarbeit.

Die Programmierschnittstelle wird zunächst für die Niedersächsische Staats- und Universitätsbibliothek Göttingen entwickelt und betrifft daher die Hochschulversion von jadis.net. Für die erste Version wird nur Lesezugriff auf die Daten von jadis.net gefordert. Es muss aber bereits hierbei sichergestellt werden, dass die Schnittstelle die Benutzerauthentifizierung von jadis.net unterstützt. Somit wird sie der Tatsache Rechnung tragen können, dass Benutzer unterschiedliche Rechte an den Daten besitzen.

[1] http://www.zweitwerk.com/

1.2. Zielsetzung

Diese Diplomarbeit behandelt die Erweiterung einer bestehenden Anwendung um einen Webservice. Der Webservice soll Nutzern der Anwendung gestatten, eigene Programme zu entwickeln, welche die Daten der Anwendung verarbeiten. Die Anwendung bekommt somit eine Programmierschnittstelle.

Von besonderem Interesse sind zwei Punkte. Der erste betrifft die Architektur, die der Programmierschnittstelle zugrunde liegen soll. Hierfür werden vier Ansätze vorgestellt. Sie kommen aufgrund der Architektur der zu erweiternden Anwendung oder bestimmter Programmiervorkenntnisse infrage. Ein wichtiges Kriterium bei der Entscheidung für eine Architektur ist die Bewertung der zur Verfügung stehenden Programmbibliotheken.

Der andere Punkt betrifft die Gebrauchstauglichkeit der Programmierschnittstelle. Da sie als Webservice gedacht ist, werden Methodenaufrufe komplexer sein als bei lokal eingebundenen Programmbiliotheken. Jeder Aufruf muss nämlich in eine SOAP-Nachricht verpackt, jede Antwort aus einer SOAP-Nachricht entpackt werden. Zum Versand und Empfang dieser Nachrichten werden darüber hinaus Netzwerkverbindungen initiiert. Hierfür sollten Mechanismen zur Verfügung stehen, die das Berarbeiten der SOAP-Nachrichten dem Benutzer abnehmen.

1.3. Kapitelüberblick

Das Kapitel „Analyse“ gibt im ersten Abschnitt eine Einführung in jadis.net. Im zweiten Abschnitt behandelt es die Leistungsmerkmale, die die Programmierschnittstelle erfüllen muss.

Die vier zur Auswahl stehenden Architekturen werden im Kapitel „Wahl der Architektur“ vorgestellt und bewertet. Schließlich wird eine Entscheidung zugunsten einer Architektur gefällt.

Die Implementierung der Schnittstelle wird im gleichnamigen Kapitel dokumentiert. Hierzu zählen sowohl das WSDL-Dokument als auch der Programmcode. Das Kapitel schließt mit einem clientseitigen Modultest und einer Webanwendung, die einen typischen Anwendungsfall demonstriert.

Das Fazit bewertet die fertige erste Version der Programmierschnittstelle und den Prozess ihrer Entwicklung. Ein besonderes Augenmerk liegt dabei auf der als Zielsetzung formulierten Beantwortung der Frage nach ihrer Gebrauchstauglichkeit. Darüber hinaus wird ein Ausblick auf die weitere Entwicklung der Schnittstelle sowie den Einsatz von Webservices gewagt.

1.4. Voraussetzungen und Begriffserklärungen

Diese Diplomarbeit setzt grundlegende Kenntnisse über objektorientierte Programmierung, Webservices und XML voraus.

Der in der Diplomarbeit häufig verwendete Begriff „Methode“ trifft auf einige Unterprogramme im jadis.net-Code nicht zu, da dieser nicht vollständig objektorientiert ist. Für ein Unterprogramm, das weder über ein Objekt noch über eine Klasse aufgerufen wird, wird die Bezeichnung „Funktion“ verwendet. Im Allgemeinen werden Unterprogramme im jadis.net-Code als „Methoden“ bezeichnet, um das Lesen durch Ausdrücke wie „Methoden beziehungsweise Funktionen“ nicht zu erschweren. In einem ausschließlich auf Webservices bezogenen Zusammenhang, wie z. B. bei einem WSDL-Dokument, kommt der Begriff „Operation“ zum Tragen.

2. Analyse

Dieses Kapitel stellt zuerst die Software jadis.net vor. Dann geht es auf die Anforderungen an die Programmierschnittstelle ein. Dabei werden die wichtigsten Aspekte von Webservices in Bezug auf die Schnittstelle erläutert. Zum Schluss werden die künftigen Operationen der Schnittstelle vorgestellt.

2.1. Einführung in jadis.net

jadis.net ist ein Media-Asset-Management-System (MAMS) zum Verwalten von Mediendaten.

Ein Benutzer von jadis.net legt, je nach Themenbereich, Mappen mit Objekten an. Die Objekte enthalten sogenannte *Assets*. Hierbei handelt es sich um Bilder, Filme, Musikdateien, Textbausteine oder andere digitalisierte Mediendaten. Der Benutzer kann den Objekten Metadaten mit zusätzlichen Informationen beifügen, um sie für spätere Recherchezwecke zu verwenden.

Es besteht die Möglichkeit, Bilder in ein anderes Format zu konvertieren. Eine Volltextsuche für Textdokumente sowie Funktionen zum Im- und Exportieren von Metadaten im XML-Format sind ebenfalls vorhanden.

Eine Benutzerverwaltung ordnet jedem Benutzer eine Gruppe zu. Anhand der Gruppe werden einem Benutzer administrative Rechte wie das Anlegen von neuen Benutzern übertragen. Inhaltliche Rechte, das heißt Rechte innerhalb der Systematik von jadis.net, werden über sogenannte *Ressorts*, die Geschäftsbereichen ähneln, geregelt. Der Benutzer kann dabei durchaus Mitglied mehrerer Ressorts sein.

jadis.net ist eine serverseitige Anwendung, die über einen Webbrowser bedient wird. Auf dem Webserver greifen PHP-Skripte auf eine Datenbank zu.

Die Anmeldeprozedur verläuft bei jadis.net in zwei Schritten. Zuerst meldet sich der Benutzer mit Name und Passwort am System an. Danach wir ihm eine Auswahl an Ressorts präsentiert. Die Ressorts sind all jene, bei denen der Benutzer registriert ist. Nachdem er sich für

ein Ressort entschieden hat, kann der Benutzer mit jadis.net arbeiten. Er kann sich jederzeit abmelden, worauf die Anmeldemaske aufs Neue erscheint.

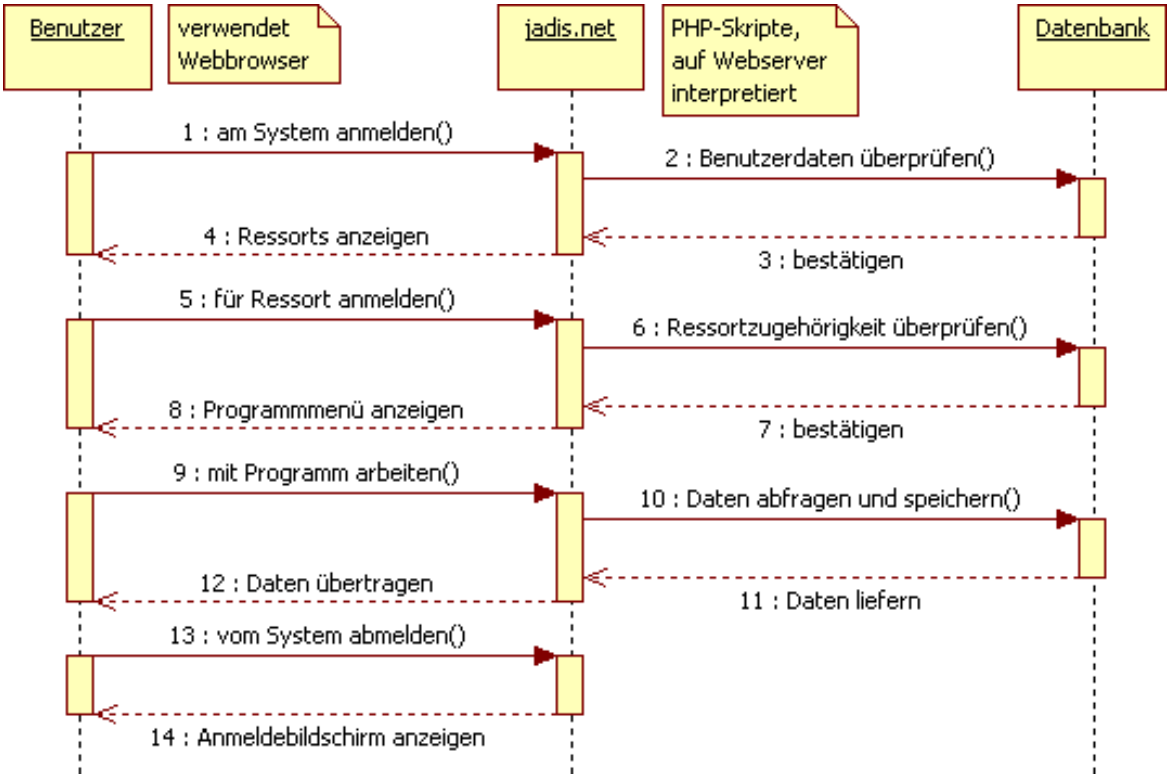


Abbildung 2.1.: Übersicht von jadis.net

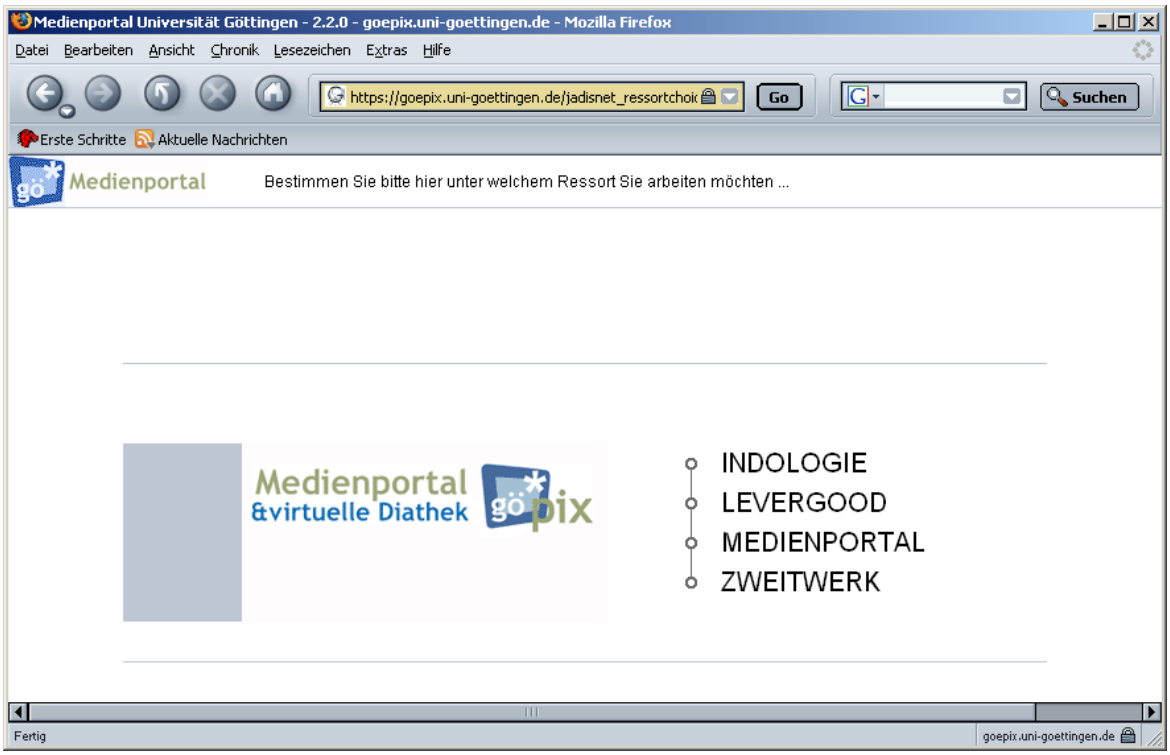


Abbildung 2.2.: Ressortauswahl bei jadis.net

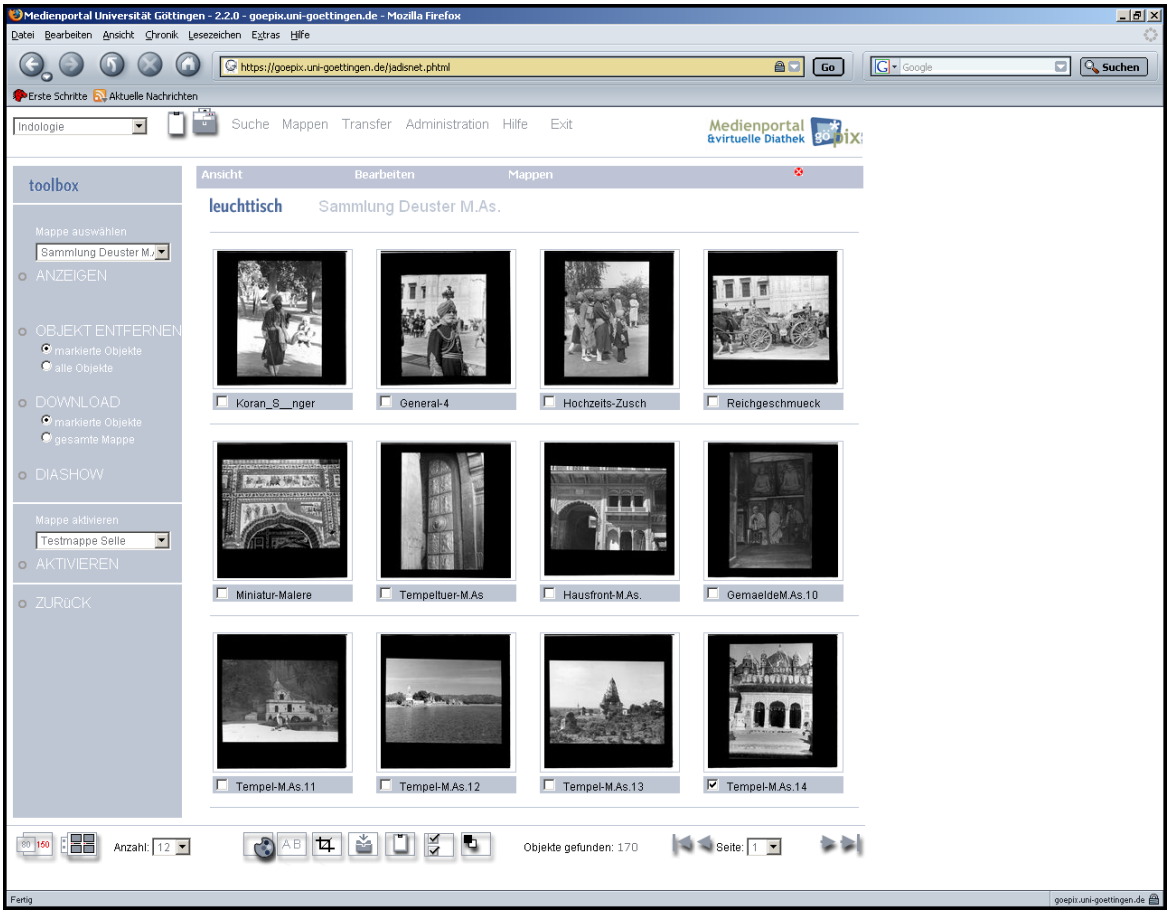


Abbildung 2.3.: Medienobjekte einer Mappe bei jadis.net

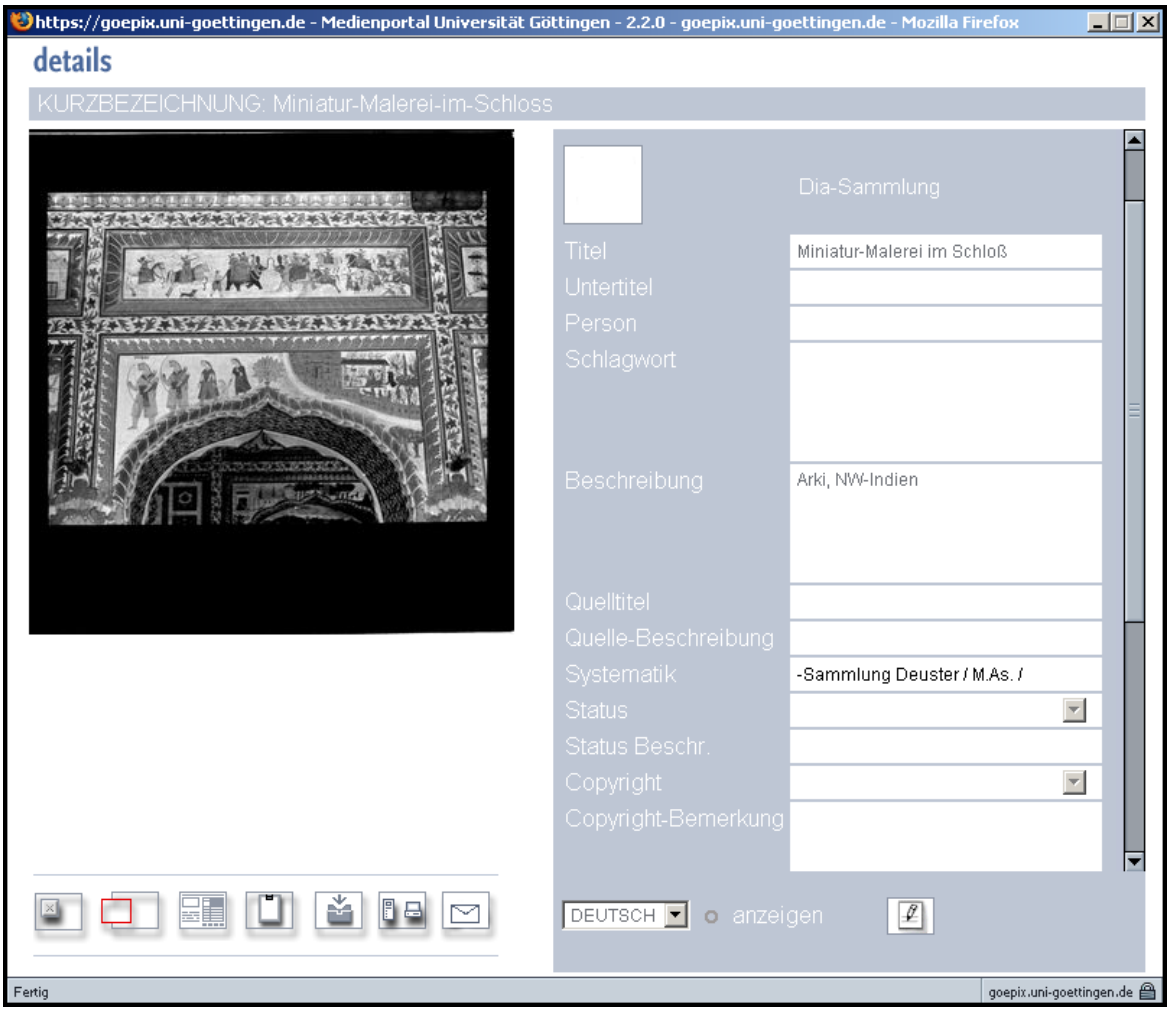


Abbildung 2.4.: Asset-Informationen bei jadis.net

2.2. Die Programmierschnittstelle als Webservice

Webservice-Definition Die Definition des W3C (World Wide Web Consortiums) für einen Webservice soll als Grundlage für das Verständnis der Programmierschnittstelle als Webservice dienen:

> Ein Webservice ist ein Softwaresystem, das entworfen wurde, um die Zusammenarbeit zwischen Rechnern über ein Netzwerk zu unterstützen. Es hat eine Schnittstelle, die in einem automatisch verarbeitbaren Format (insbesondere WSDL) beschrieben ist. Andere Systeme arbeiten, dieser Beschreibung folgend, mit dem Webservice zusammen. Sie tun dies mittels SOAP-Nachrichten, die typischerweise über HTTP mit einer XML-Serialisierung in Verbindung mit anderen Web-Standards übertragen werden.[1]

Die Programmierschnittstelle stellt Benutzern einen Teil der Funktionalität von jadis.net für ihre eigenen Anwendungen zur Verfügung. Die Beschreibung der Schnittstelle befindet sich in einem WSDL-Dokument. Nach dieser Beschreibung richten sich die Anwendungen, um die Operationen der Schnittstelle nutzen zu können. Die Operationen werden eingeleitet, indem die Anwendungen als Clients über SOAP mit der Schnittstelle Nachrichten austauschen. Die Nachrichten enthalten sowohl die Anfragen und eventuell benötigte zusätzliche Informationen als auch die Antworten, welche das Ergebnis oder die gewünschten Daten liefern, in serialisierter Form als Inhalt von XML-Elementen. Als Übertragungsprotokoll für die Nachrichten dient HTTP.

Aufbau einer SOAP-Nachricht Eine SOAP-Nachricht ist eine XML-Struktur, bestehend aus einem Envelope, welcher ein oder mehrere optionale Header- und ein zwingend vorgeschriebenes Body-Element enthält. Der Body ist für die Übertragungen der Nutzdaten zuständig, während der Header zusätzliche Informationen für die Verarbeitung der Nachricht transportiert.[2] Im Header kann angegeben werden, ob ihn der Nachrichtenempfänger verarbeiten muss.[3]

Falls ein Fehler auftritt und eine Ausnahmebedingung definiert ist, verschickt der Sender einen SOAP-Fault.[4]

---

[1] Vgl. (W3C04a) W3C (World Wide Web Consortium): *Web Services Glossary, Eintrag „Web service“* (Übers. d. Verf.). http://www.w3.org/TR/ws-gloss/

[2] Vgl. (WCL+05) Weerawarana, Sanjiva u. a.: *Web Services Platform Architecture*. S. 65

[3] Vgl. (WCL+05) S. 71 f.

[4] Vgl. (WCL+05) S. 65

2.2.1. Anforderungen an die Programmierschnittstelle

Die Programmierschnittstelle soll Benutzern ermöglichen, eigene Anwendungen so auszustatten, dass sie standortunabhähngig auf die Daten von jadis.net zugreifen können. Deshalb hat die zweitwerk software engineering GmbH entschieden, die Schnittstelle als Webservice entwickeln zu lassen.

Darüber hinaus werden einige zusätzliche Anforderungen gestellt, auf die im Folgenden näher eingegangen wird.

2.2.1.1. Veröffentlichung als WSDL-Dokument

Die Operationen der Programmierschnittstelle sollen in einem WSDL-Dokument veröffentlicht werden. Das Format des Dokumentes wird der WSDL-Spezifikation 1.1[5] entsprechen, da diese Version zurzeit noch weit verbreitet ist und in der Literatur häufig für Beispielanwendungen verwendet wird.[6]

Für die Gebrauchstauglichkeit ist es wichtig, dass zwischen Schnittstelle und Anwendungen, die auf sie zugreifen, keine Probleme beim Nachrichtenaustausch aufgrund inkompatibler Typen entstehen. Um eine maximale Interoperabilität zu ermöglichen, soll das WSDL-Dokument einem weit verbreiteten Standard entsprechen. Deshalb wird es konform zum WS-I-Anforderungsprofil[7] sein. Diese Entscheidung hat folgende Konsequenzen:

- Als Nachrichtenprotokoll wird SOAP verwendet.[8]
- Das Parameterformat `encoded` wird nicht verwendet. Stattdessen hat das Attribut `use` den Wert `literal` in sämtlichen `wsdlsoap:body`-, `wsdlsoap:fault`-, `wsdlsoap:header`- und `wsdlsoap:headerfault`-Elementen.[9]
- Der Empfehlung für die Deklaration von Arrays in der WSDL-Spezifikation 1.1 unter Abschnitt 2.2[10] wird nicht gefolgt. Das WSDL-Dokument verwendet weder den Typ `soapenc:Array` noch die Attribute `wsdl:arrayType` oder `soapenc:arrayType`.[11]

---

[5] Vgl. (W3C01) W3C (World Wide Web Consortium): *Web Services Description Language (WSDL) 1.1*. http://www.w3c.org/TR/wsdl

[6] Vgl. z. B. (Ric06) Richards, Robert: *Pro PHP XML and Web Services*. S. 674 und (WCL+05) S. 107

[7] Vgl. (WS-06) WS-I (Web Services Interoperability Organizazion): *WS-I Basic Profile Version 1.1*. http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[8] Vgl. (WS-06) R2401 (Empfehlung 2401, Übers. d. Verf.). http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[9] Vgl. (WS-06) R2706. http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[10] Vgl. (W3C01) Abschn. 2.2. http://www.w3c.org/TR/wsdl

[11] Vgl. (WS-06) R2110f., R2113. http://www.ws-i.org/Profiles/BasicProfile-1.1.html

Da weder WS-I (Web Services Interoperability Organization) noch das W3C zwischenstaatlich anerkannte Organisationen sind, können sie strenggenommen keine Standards festlegen. Daher sprechen sie nur *Empfehlungen* aus, die aber bei der Entwicklung der Programmierschnittstelle befolgt werden sollen.

Stil des WSDL-Dokumentes Das WSDL-Dokument wird den Stil `document-literal` haben. Dieser Stil führt zwar zu einer etwas komplizierteren Struktur des Dokumentes, erlaubt aber ein einfaches Validieren einer SOAP-Nachricht. Der Grund hierfür ist, dass sämtliche Elemente des Nachrichten-Bodys in einem XML-Schema definiert und nicht Teil von WSDL-Definitionen sind.[12]

In Anbetracht der Tatsache, dass das WSDL-Dokument nur ein einziges Mal erstellt, jede SOAP-Nachricht aber validiert wird, ist die kompliziertere Struktur des WSDL-Dokumentes in Kauf zu nehmen.

2.2.1.2. Bereitstellung eines Authentifizierungsmechanismus

Da jadis.net verlangt, dass sich die Benutzer zuerst am System und danach für ein Ressort anmelden, muss auch die Programmierschnittstelle diese Anmeldeprozedur bereitstellen. Das bedeutet, dass eine Operation einen Fehler zurückgeben müssen, sofern die zweifache Anmeldung nicht erfolgt ist.

2.2.1.3. Einfache Installation beim Kunden

Der Aufwand, die Programmierschnittstelle bei einem Kunden zu installieren, sollte sich in Grenzen halten. Dies betrifft die Benutzung von Programmbibliotheken und Werkzeugen, die möglicherweise im Zusammenhang mit der Schnittstelle benötigt werden.

2.2.1.4. Einfache Integration in die bisherige jadis.net-Umgebung

An der jadis.net-Umgebung, auf deren Daten die Programmierschnittstelle zugreifen wird, sollten so wenige Änderungen wie möglich vorgenommen werden.

[12] Vgl. (But03) BUTEK, Russell: *Which Style of WSDL should I use?*. http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

2.2.1.5. Sicherheit

SOAP-Nachrichten bestehen aus XML-Elementen, deren Inhalt zunächst einmal im Klartext übertragen wird. Dies ist besonders im Hinblick auf die Anmeldung problematisch, da hier ein Passwort übermittelt wird.

Zum Verschlüsseln von XML-Elementen hat das W3C als Lösungsansatz das Verfahren XML Encryption[13] festgelegt. Hierbei wird der Inhalt von XML-Elementen verschlüsselt. Dabei kann der verschlüsselte Inhalt auch weitere XML-Elemente enthalten.

Zumindest für die erste Version der Programmierschnittstelle ist es jedoch noch nicht vorgesehen, einzelne Elemente von SOAP-Nachrichten verschlüsselt zu übertragen. Außerdem entstünde hierbei durch das Ver- und Entschlüsseln ein zusätzlicher Overhead bei der Bearbeitung der SOAP-Nachrichten.

Eine dem Benutzer zur Verfügung stehende Möglichkeit, SOAP-Nachrichten ohne Klartext zu übertragen, ist das verschlüsselte Versenden der Nachrichten über eine SSL-Verbindung[14]. Bei SSL tritt die Verschlüsselung unterhalb der Anwendungsschicht auf. Es werden also nicht die Bodys einzelner SOAP-Nachrichten oder bestimmte Elemente verschlüsselt, sondern der gesamte Datenverkehr. Für Webservices kann dies ein Problem darstellen.[15] Folgendes Szenario ist damit nämlich ausgeschlossen: Eine SOAP-Nachricht wird vom Sender zum entgültigen Empfänger über eine Zwischenstation versandt. Die Zwischenstation darf den Inhalt bestimmter Nachrichtenelemente nicht lesen, muss aber andere lesen können, deren Inhalt wiederum nicht für den Endempfänger bestimmt ist. Bei einer SSL-Kommunikation ist aber die gesamte Nachricht ohne Hinweis auf Anfang und Ende eines bestimmten Elementes verschlüsselt. Eine teilweise Entschlüsselung einer SOAP-Nachricht ist somit nicht möglich.

Da die Programmierschnittstelle keine Verschlüsselung einzelner Elemente für unterschiedliche Stationen beim Nachrichtenversand vorsieht, bietet SSL jedoch einen ausreichenden Verschlüsselungsmechanismus.

Als digitales Signaturverfahren hat das W3C XML-Signature[16] definiert. Das Verfahren soll im Falle von SOAP-Nachrichten sicherstellen, dass sich der Sender einer Nachricht mit einer digitalen Signatur ausweisen kann.

---

[13] Vgl. (W3C02a) W3C (World Wide Web Consortium): *XML Encryption Syntax and Processing*. http://www.w3c.org/TR/xmlenc-core/

[14] Vgl. (Tan03) Tanenbaum, Andrew S[tuart]: *Computer Networks*. S. 813–816

[15] (JZ02) Jeckle, Mario ; Zengler, Barbara: *SOAP – aber sicher*. http://www.jeckle.de/secureSOAP.html

[16] Vgl. (W3C02b) W3C (World Wide Web Consortium): *XML-Signature Syntax and Processing*. http://www.w3c.org/TR/xmldsig-core/

Aufgrund der Art und Weise, wie bei einer PHP-Anwendung Sitzungen wiederaufgenommen werden,[17] wird bei der Programmierschnittstelle ein anderes Verfahren angewandt. Im Kapitel „Wahl der Architektur“ wird näher darauf eingegangen.

2.2.2. Operationen der Programmierschnittstelle

Die erste Version der Programmierschnittstelle soll zunächst nur Operationen anbieten, die Lesezugriff auf die Daten von jadis.net gewähren. Die Operationen werden benötigt, um einem Benutzer Zugriff auf die Daten seiner Assets zu gewähren.

2.2.2.1. createSession

`createSession` leitet eine Benutzeranmeldung am jadis.net-System ein und eröffnet damit eine Sitzung. Die Operation informiert nach vollzogener Anmeldung über die Ressorts, für die der Benutzer registriert ist.

2.2.2.2. endSession

`endSession` beendet die Sitzung. Der Benutzer muss sich danach für den Aufruf von Schnittstellenoperationen erneut anmelden.

2.2.2.3. selectDepartment

Mit dieser Operation wählt der Benutzer ein Ressort aus, für das er registriert ist. Alle folgenden Operationen stehen ihm nur zur Verfügung, nachdem er sich für ein Ressort entschieden hat.

2.2.2.4. selectPortfolioList

`selectPortfolioList` übermittelt eine Liste aller Mappen, bei denen der Benutzer mindestens Leserechte besitzt.

In Mappen werden Medienobjekte zusammengefasst.

[17] Vgl. (PHP08) PHP GROUP, The: *PHP Manual, Function Reference, Session Handling Functions, Introduction*. http://www.php.net/manual/en/ref.session.php

2.2.2.5. selectPortfolioObjects

Diese Operation liefert die Medienobjekte einer bestimmten Mappe, für die der Benutzer mindestens Leserechte hat.

Medienobjekte enthalten ein oder mehrere Assets. Sie liefern außerdem Metadaten wie beispielshalber Titel, Beschreibung und Schlagwörter über die Assets.

2.2.2.6. selectAssets

`selectAssets` liefert die Assets eines bestimmten Medienobjektes.

Assets sind zum Beispiel Bilder, Filme, Musikdateien oder Texte. Jedem Asset ist mindestens ein Vorschaubild zugeordnet.

2.2.2.7. selectCatalogNodes

`selectCatalogNodes` übermittelt dem Benutzer die Katalogknoten in der Systematik von jadis.net, mit dem ein Medienobjekt verknüpft ist.

2.2.2.8. selectPreview

Die Operation `selectPreview` liefert das Vorschaubild eines bestimmten Assets in einer von vier Größen. Das Benutzer erhält dabei die Binärdaten des gesamten Bildes.

3. Wahl der Architektur

Dieses Kapitel stellt vier Architekturansätze vor, die für die Realisierung der Programmierschnittstelle infrage kommen. Sie unterscheiden sich hinsichtlich der Kommunikation mit der jadis.net-Geschäftslogik und der Auswahl der Programmiersprache. Die Wahl der Programmiersprache richtet sich nach dem Architekturansatz und den bei der zweitwerk software engineering GmbH verwendeten Programmiersprachen für Internet-basierte Anwendungen. Dies sind PHP und Java.

Der erste Ansatz sieht eine direkte Kommunikation zwischen Schnittstelle und der jadis.net-Geschäftslogik vor. Die Schnittstelle greift dabei als Modul von jadis.net über PHP-Methodenaufrufe auf die Geschäftslogik zu.

Die drei übrigen Ansätze geben einer indirekten Kommunikation zwischen der Programmierschnittstelle und der jadis.net-Geschäftslogik den Vorzug. Hier sind Schnittstelle und jadis.net auf unterschiedlichen Rechnern installiert. Die Schnittstelle greift deshalb über ein Netzwerkprotokoll auf die Methoden der jadis.net-Geschäftslogik zu.

Jeder Architekturansatz wird im Folgenden einem Test hinsichtlich seiner Realisierbarkeit und Praktikabilität unterzogen. Ein Test-Webservice soll erstens die Anmeldung eines Benutzers bei jadis.net ermöglichen und eine Sitzung einleiten. Zweitens soll der Benutzer in der Lage sein, ein Ressort auszuwählen, für das er registriert ist. Die dafür benötigten Operationen werden in einem WSDL-Dokument veröffentlicht.

Während des Tests werden Authentifizierungsdaten zwischen den beteiligten Rechnern ausgetauscht und ausgewertet. Dadurch gibt der Test einen Hinweis auf die Komplexität, die bei der Implementierung der jeweiligen Schnittstellenarchitektur auftreten wird.

Der Test ist repräsentativ, weil jeder jadis.net-Benutzer sich erstens anmelden und zweitens ein Ressort auswählen muss, um die Operation nutzen zu können, welche ihm Zugriff auf die Mappen,- Medienobjekt- und Asset-Daten gewähren. Der Test kann somit direkt in die Programmierschnittstelle integriert werden. Zudem werden die Elemente im XML-Schema des WSDL-Dokumentes, die für die Testoperationen verwendet werden, dieselbe Struktur wie die Elemente aller anderen Operationen haben. Dasselbe gilt für die Struktur der im WSDL-Dokument definierten Nachrichten.

Literatur Einen allgemeinen Einblick in das Thema Webservices bietet das Buch *Web Services Platform Architecture* von Sanjiva Weerawarana u. a. (WCL+05). Für die Entwicklung von Webservices mit PHP empfiehlt sich das Buch *Pro PHP XML and Web Services* von Robert Richards (Ric06). Das Thema Java und Webservices wird in den Büchern *Java Web Services mit Apache Axis2* von Thilo Frotscher, Marc Teufel und Dapeng Wang (FTW07) und *Building Web Services with Java* von Steve Graham u. a. (GDS+04) behandelt. Letzteres geht ausführlich auf viele Aspekte von Webservices ein und ist weniger ein Handbuch für die Java-Programmierung von Webservices.

3.1. Test

3.1.1. Testoperationen

Der Webservice bietet die Operationen `createSession` und `selectDepartment` an. `createSession` führt eine Benutzeranmeldung durch und leitet eine Sitzung ein. `selectDepartment` wählt ein Ressort aus, für das der angemeldete Benutzer registriert ist.

`createSession` übermittelt eine Sequenz, welche die Zeichenketten `username` und `password` als Elemente enthält. `username` und `password` repräsentieren den Namen und das Passwort eines Benutzers. `createSession` gibt nach dem Aufruf durch einen Test-Client ein `SessionObject`-Element zurück. Dieses enthält eine Sequenz, bestehend aus der Zeichenkette `authenticationData` und einer Sequenz von `Department`-Elementen. `authenticationData` enthält das Authentifizierungsdatum für alle folgenden Aufrufe. Die `Department`-Elemente stellen die Ressorts dar, für die der Benutzer registriert ist. Ein `Department`-Element enthält eine Sequenz, die den Ganzzahlwert `departmentId` und die Zeichenkette `name` besitzt. Sie entsprechen der Identifizierungsnummer und dem Namen eines Ressorts.

Ein ungültiger Benutzername oder ein falsches Passwort lösen beim Aufruf von `createSession` eine Exception aus und führen zum Abbruch der Operation. Die Exception wird dem Client als SOAP-Fault übermittelt.

`selectDepartment` überträgt als Element den Ganzzahlwert `departmentId`. `departmentId` hat den Wert des gleichnamigen Elementes eines zuvor übermittelten `Department`-Elementes. Im Header der SOAP-Nachricht, mit der `selectDepartment` aufgerufen wird, befindet sich das Element `authenticate`. Dieses enthält den Wert des `authenticationData`-Elementes, welches beim Aufruf von `createSession` übertragen wurde. Ein fehlender oder falscher Wert von `authenticate` wird mit einer Exception und dem Abbruch der Operation quittiert. Dem Client wird ein SOAP-Fault übermittelt. Dasselbe

gilt für den Fall, dass die IP-Adresse des Test-Clients nicht mehr derjenigen beim Aufruf von `createSession` entspricht. Die Berücksichtigung der IP-Adresse bringt eine zusätzliche Sicherheit, da ein abgefangenes Authentifizierungsdatum dann nicht ohne Weiteres für das unautorisierte Einleiten einer Operation missbraucht werden kann. Selbstverständlich gilt dies nicht für den Fall, dass mehrere Benutzer über einen Proxy-Server diesselbe IP-Adresse nach außen zum Internet haben.

`selectDepartment` gibt im Erfolgsfall den booleschen Wert `true` zurück. Im Falle des Scheiterns wird der boolesche Wert `false` an den Test-Client übermittelt.

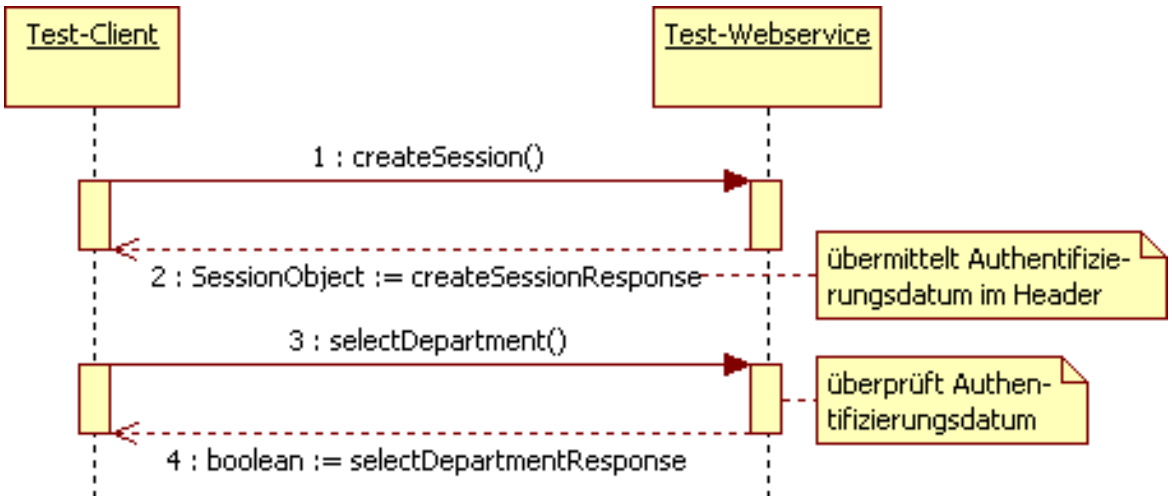


Abbildung 3.1.: Ablauf des Tests

Der Test ist bestanden, wenn folgende vier Abläufe zu den entsprechenden Ergebnissen führen:

1. Der Test-Client ruft die Operation `createSession` beim Test-Webservice auf und übermittelt dabei gültige Werte für Name und Passwort. Daraufhin erhält er das Authentifizierungsdatum und die Ressorts, für die der Benutzer angemeldet ist.

   Mit der Operation `selectDepartment` übermittelt er dann die Identifizierungsnummer eines der Ressorts. Er erhält als Antwort den booleschen Wert `true`.

2. Der Ablauf verhält sich analog zum ersten Ablauf. Dieses Mal wird beim Aufruf von `selectDepartment` jedoch eine Identifizierungsnummer übermittelt, die auf kein Ressort verweist, für das der Benutzer angemeldet ist. Der Test-Client erhält daraufhin den Wert `false`.

3. Der Test-Client ruft `createSession` mit ungültigem Benutzernamen oder Passwort auf. Als Konsequenz wird ihm ein SOAP-Fault übermittelt.

4. Der Test-Client meldet sich erfolgreich mit `createSession` an. Er übermittelt beim Aufruf von `selectDepartment` jedoch ein ungültiges Authentifizierungsdatum. Dies führt zum Erhalt eines SOAP-Faults.

3.1.2. Test-Client

Der Test-Client ist in der Programmiersprache PHP implementiert. Bei ihm läuft der Test folgendermaßen ab: Zu Beginn ruft das Startskript des Clients die Webservice-Operation `createSession` auf. Dafür ist beim Client die Klasse `CreateSessionParameters` mit den Instanzvariablen `$userId` und `$password` vorhanden. Ein `CreateSessionParameters`-Objekt repräsentiert dabei die Sequenz der Webservice-Operation `createSession`. Die Instanzvariablen des Objektes entsprechen den gleichnamigen Elementen dieser Sequenz. Der Name der Klasse `CreateSession` könnte auch ein anderer sein; die Bezeichnungen der Instanzvariablen müssen aber grundsätzlich mit den Sequenzelementen übereinstimmen.

Der Client instanziiert ein Objekt der von PHP Version 5 bereitgestellten Klasse `SoapClient`[1]. Ein Parameter beim Konstruktoraufruf ist dabei der URI des WSDL-Dokumentes. Durch die Informationen im WSDL-Dokument ist das `SoapClient`-Objekt in der Lage, Methodenaufrufe anzunehmen, die im WSDL-Dokument definierten Operationen entsprechen. Nach der Instanziierung ruft der Client die Methode `createSession(CreateSessionParameters $params)` des `SoapClient`-Objektes auf. Der Parameter ist das zuvor instanziierte `CreateSessionParameters`-Objekt. Das `SoapClient`-Objekt veranlasst in diesem und allen weiteren Fällen die Serialisierung der PHP-Objekte in XML-Elemente. Dasselbe gilt für die Deserialisierung der XML-Elemente in PHP-Objekte bei den Antwortnachrichten.

Nachdem der Client das `SessionObject`-Element durch den Aufruf von `createSession(CreateSessionParameters $params)` erhalten hat, wählt er eines der sich darin befindenden `Department`-Elemente als Ressort aus. Der Wert des dazugehörigen `departmentId`-Elementes dient ihm als Parameter für den Aufruf der `SoapClient`-Methode `selectDepartment(integer $departmentId)`. Das `authenticationData`-Element des `SessionObject`-Elementes wird von der `SoapClient`-Methode `__setSoapHeaders(array $soapHeaders)`[2] verarbeitet. Diese Methode wird vor dem Aufruf von `selectDepartment(integer $departmentId)` aufgerufen.

---

[1] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions*. http://www.php.net/manual/en/ref.soap.php

[2] Vgl. (Ric06) RICHARDS, Robert: *Pro PHP XML and Web Services*. S. 718f.

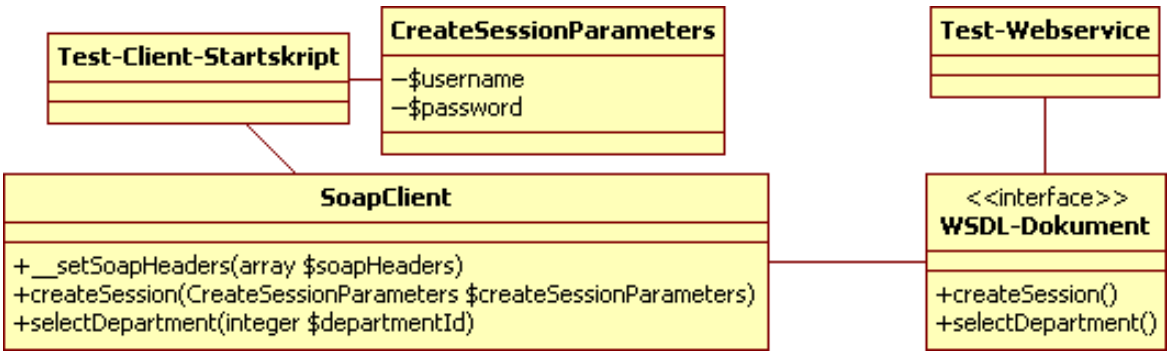


Abbildung 3.2.: Test-Client

3.2. Direkte Kommunikation mit jadis.net

3.2.1. Programmierschnittstelle als Modul von jadis.net

3.2.1.1. Modell

Die Entwicklung der Programmierschnittstelle als Modul von jadis.net bedeutet, dass jadis.net um einen Webservice erweitert wird. Dieses Webservice-Modul benutzt die öffentlichen Methoden der jadis.net-Geschäftslogik, um an die von einem Client verlangten Daten zu kommen.

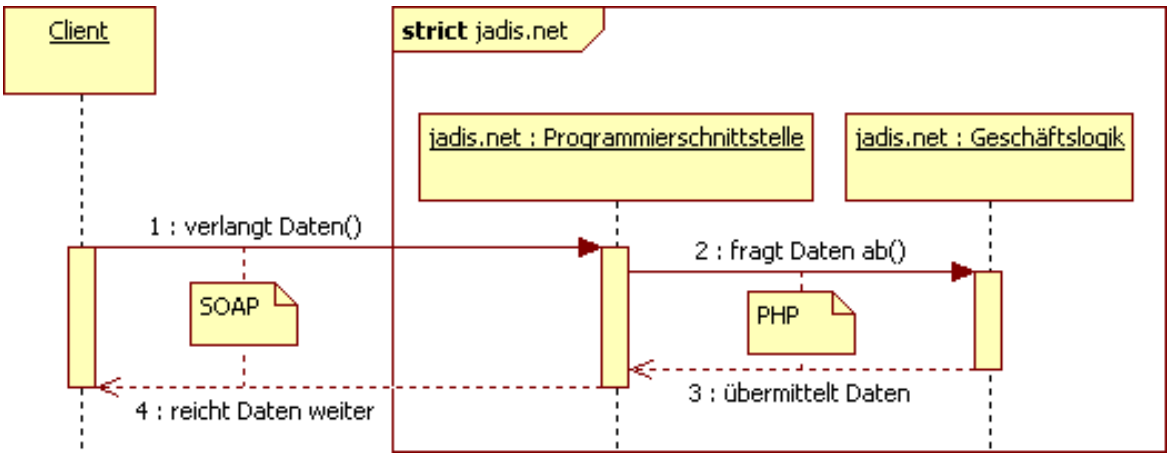


Abbildung 3.3.: Programmierschnittstelle als Modul von jadis.net

Authentifizierungsmechanismus Die Authentifizierung beim Aufruf von Operationen der Programmierschnittstelle läuft folgendermaßen ab: Zuerst ruft ein Client die Schnittstellen-

operation zur Sitzungseröffnung auf und übermittelt als Parameter Name und Passwort des Benutzers. Die Schnittstelle ruft daraufhin die entsprechende Methode der jadis.net-Geschäftslogik auf, die die Gültigkeit von Benutzername und Passwort überprüft. Im Erfolgsfall erzeugt die Schnittstelle ein Authentifizierungsdatum und übermittelt es an den Client. Mit diesem Datum weist sich der Client bei jeder weiteren Operation bei der Programmierschnittstelle aus. Die Schnittstelle überprüft dann die eingehende Nachricht auf die Gültigkeit des Authentifizierungsdatums und bearbeitet im Erfolgsfall die Anfrage.

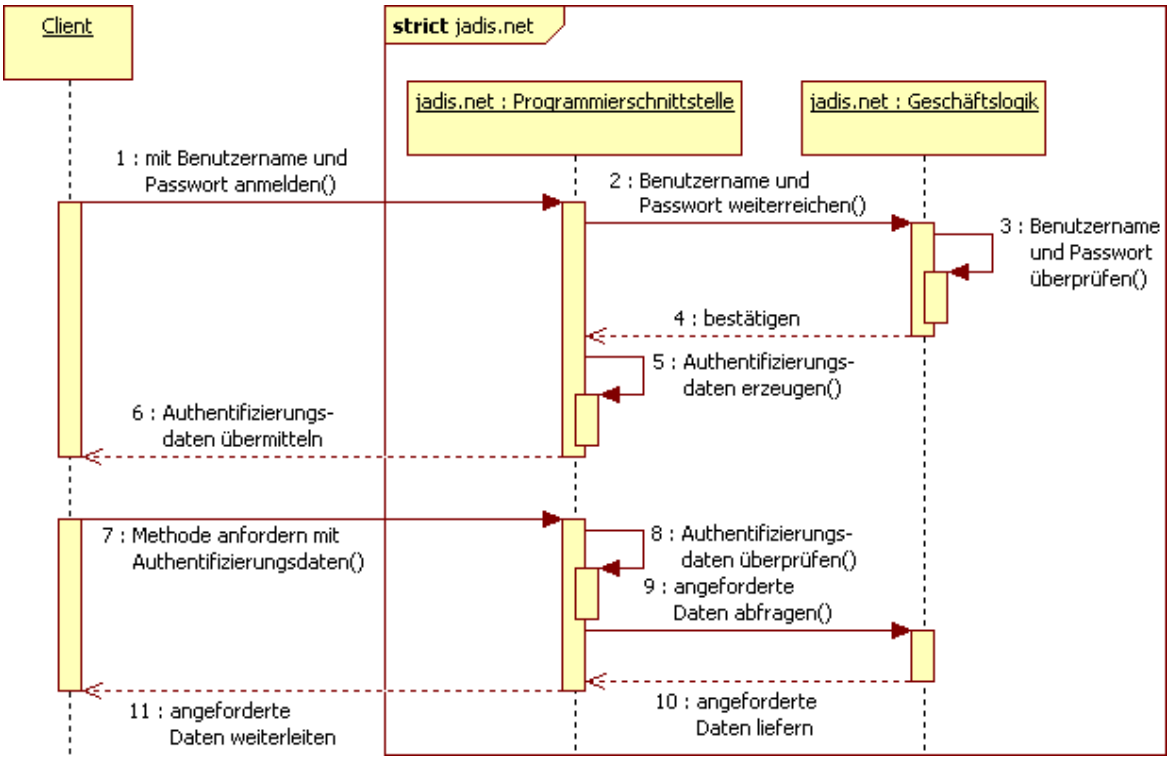


Abbildung 3.4.: Authentifizierungsmechanismus bei direkter Kommunikation mit jadis.net

3.2.1.2. Benötigte Software

Da die zweitwerk software engineering GmbH ihre PHP-Installationen nach und nach von Version 4 auf Version 5 umstellen will, wird für diesen Ansatz PHP in der Version 5[3] benötigt. Hierbei ist zu beachten, dass die Installation mit dem Schalter `--enable-soap`[4] durchgeführt wird. Damit werden die für die Webservice-Entwicklung benötigten Klassen mitgeliefert.

[3] Vgl. (PHP08) *Downloads*. http://www.php.net/downloads.php

[4] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions*. http://www.php.net/manual/en/ref.soap.php

Des Weiteren muss ein Webserver vorhanden sein, der in der Lage ist, PHP-Skripte zu interpretieren. Die zweitwerk software engineering GmbH wird nach der Umstellung auf PHP 5 den Apache HTTP-Server in der Version 2.2[5] verwenden. Daher wird dieser Webserver für die folgende Untersuchung verwendet.

3.2.1.3. Test-Implementierung

Der Webservice ist ein PHP-Skript, das die Methoden der in PHP 5 eingeführten Klasse `SoapServer`[6] nutzt. Dieses Objekt veranlasst die Deserialisierung von XML-Elementen in PHP-Objekte und die Serialisierung von PHP-Objekten in XML-Elemente.

Nach der Instanziierung eines `SoapServer`-Objektes wird mit der Methode `setClass(string $class_name [ , mixed $args [ , mixed ... ]])`[7] die Klasse `SoapHandler` eingebunden. Dort sind die Methoden `createSession(CreateSessionParameters $params)`, `authenticate(string $authData)` und `selectDepartment(integer $departmentId)` implementiert. `createSession(CreateSessionParameters $params)` und `selectDepartment(integer $departmentId)` entsprechen dabei den im WSDL-Dokument festgelegten Operationen `createSession` und `selectDepartment`. Damit auf die Methoden zugegriffen werden kann, sind sie mit dem Zugriffsmodifizierer `public` gekennzeichnet.

Die Methode `handle([string $soap_request])`[8] des `SoapServer`-Objektes bearbeitet jede eingehende SOAP-Nachricht.

Eine `createSession`-Operation führt zum Aufruf der `SoapHandler`-Methode `createSession(CreateSessionParameters $params)`. Diese leitet eine Benutzeranmeldung an die Funktion `verifyUser(string $usr, string $pwd)` der jadis.net-Geschäftslogik weiter. Dort wird die Anmeldung am System vollzogen, und die Sitzung wird eingeleitet.

Während der Anmeldung legt die jadis.net-Geschäftslogik eine Datei an, die solange existiert, wie der Benutzer angemeldet ist. Diese Sitzungsdatei enthält Informationen über Anmeldestatus, Registrierungen und Rechte des Benutzers. Jede Interaktion des Benutzers

---

[5] Vgl. (Apa08) APACHE SOFTWARE FOUNDATION, The: *HTTP Server Project. Download*. http://httpd.apache.org/download.cgi

[6] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions*. http://www.php.net/manual/en/ref.soap.php

[7] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions, SoapServer->setClass()*. http://www.php.net/manual/en/function.soap-soapserver-setclass.php

[8] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions, SoapServer->handle()*. http://www.php.net/manual/en/function.soap-soapserver-handle.php

hat eine neue Abarbeitung eines jadis.net-Skriptes zufolge. Das entsprechende Skript kann dann auf die Sitzungsdatei zugreifen und feststellen, ob sich der Benutzer zuvor angemeldet hat. Jede Sitzungsdatei hat einen eigenen Namen, der eine eindeutige Beziehung zwischen Benutzer und Anmeldung herstellt.

Der Name der Sitzungsdatei wird aufgrund seiner Eindeutigkeit als Authentifizierungsdatum für den Nachrichtenaustausch verwendet. Außerdem wird in der Datei die IP-Adresse des Clients vermerkt. Der Webservice erzeugt nun ein Objekt der vorher implementierten Klasse `SessionObject` und weist das Authentifizierungsdatum der Instanzvariable `$authData` als Wert zu.

Danach ruft der Webservice die Klassenmethode `getDepartments(integer $userId)` der Klasse `SqlStatements` auf. Der Parameter `$userId` stellt die Identifizierungsnummer des Benutzers am jadis.net-System dar. Diese Nummer wurde nach dem erfolgreichen Aufruf der Funktion `verifyUser(string $usr, string $pwd)` von der jadis.net-Geschäftslogik in die Sitzungsdatei geschrieben und steht somit allen weiteren Arbeitsschritten zur Verfügung.

Im Gegensatz zur Funktion `verifyUser(string $usr, string $pwd)` stellt die jadis.net-Geschäftslogik die Klasse `SqlStatements` und somit auch ihre Klassenmethode `getDepartments(string $userId)` nicht zur Verfügung. Die Klasse `SqlStatements` muss daher zusätzlich implementiert werden. Sie enthält dann sämtliche Klassenmethoden, die die Ergebnismengen von Anfragen an die jadis.net-Datenbank liefern. Die Verbindung zur Datenbank wird durch Methoden der jadis.net-Geschäftslogik eingegangen. Die Klassenmethoden müssen die entsprechenden SQL-Anfragen jedoch selbst zusammensetzen, wobei auch hierfür teilweise Methoden der jadis.net-Geschäftslogik zur Unterstützung bereit stehen.

`getDepartments(string $userId)` liefert ein Array, dessen Elemente assoziative Arrays sind. Ein assoziatives Array entspricht einer Map in der Programmiersprache Java. Jedes assoziative Array besitzt zwei Schlüssel, die sich auf ein Ressort beziehen, für das der Benutzer registriert ist. Der eine Schlüssel verweist auf die Identifizierungsnummer des Ressorts, der andere auf den Ressortnamen. Der Webservice instanziiert für jedes dieser Ressorts ein Objekt der zuvor implementierten Klasse `Department`. Die Identifizierungsnummer und der Ressortname werden den Instanzvariablen `$departmentId` und `$name` des jeweiligen `Department`-Objektes zugewiesen. Die `Department`-Objekte werden daraufhin in einem Array abgelegt. Das vorher erzeugte `SessionObject`-Objekt erhält das Array als Wert für die Instanzvariable `$department`.

Als letzten Arbeitsschritt einer `createSession`-Operation sendet der Webservice das `SessionObject`-Objekt an den Client.

Eine `selectDepartment`-Operation leitet den Aufruf der `SoapHandler`-Methode `selectDepartment(integer $departmentId)` ein. Das `SoapServer`-Objekt veranlasst jedoch vorher die Abarbeitung der `SoapHandler`-Methode `authenticate(string $authData)`. Diese Methode überprüft, ob das in der SOAP-Nachricht übermittelte Authentifizierungsdatum dem Namen der für den Benutzer angelegten Sitzungsdatei gleicht und die in der Datei vermerkte IP-Adresse mit der aktuellen IP-Adresse des Benutzer-Clients übereinstimmt. Trifft dies zu, wird die bei jeder Webservice-Operation auf den booleschen Wert `FALSE` gesetzte Instanzvariable `$hasAuthenticationSucceeded` auf `TRUE` gesetzt. Andernfalls wird eine Exception mit einer Nachricht geworfen und als SOAP-Fault an den Client weitergereicht. Die Operation `selectDepartment` wird danach abgebrochen.

Hat die Instanzvariable `$hasAuthenticationSucceeded` den Wert `TRUE`, ruft die Methode `selectDepartment(integer $departmentId)` die Klassenmethode `getDepartments(integer $userId)` der Klasse `SqlStatements` auf. Diese gibt ein Array sämtlicher Identifizierungsnummern von Ressorts zurück, bei denen eine Registrierung des Benutzers vorliegt. Entspricht eine dieser Identifizierungsnummern dem vom Client übermittelten Parameter `$departmentId`, sendet der Webservice den booleschen Wert `TRUE` an den Client. Findet sich keine Übereinstimmung wird `FALSE` übermittelt.

Falls die Instanzvariable `$hasAuthenticationSucceeded` den Wert `FALSE` hat, wirft `selectDepartment(integer $departmentId)` eine Exception. Diese mit einer entsprechenden Nachricht als SOAP-Fault an den Client versandt.

3.2.1.4. Bewertung

Der Test wurde erfolgreich abgeschlossen, wobei der größte Aufwand für das Erstellen des WSDL-Dokumentes entstand.

Mit der Klasse `SoapServer` erleichtert PHP 5 die Implementierung eines Webservice. Die automatische Serialisierung und Deserialisierung der Daten sowie das automatische Versenden der SOAP-Nachrichten erlauben Programmierern, sich hauptsächlich den Webservice-Methoden zu widmen.

Hingegen existiert noch kein Werkzeug für PHP 5, das Server-Code aus einem WSDL-Dokument generieren könnte. Dies bedeutete jedoch kein Hindernis bei der Implementierung des Test-Webservice.

Die Anfertigung des Test-Webservice erforderte, diesen um Funktionalität zu erweitern, die eigentlich Teil der Geschäftslogik ist. Eine Beschränkung auf bisherige Methoden der jadis.net-Geschäftslogik war nicht möglich. Das liegt vor allem daran, dass dort keine Methode vorhanden ist, die genau die SQL-Anfrage an die Datenbank stellt, welche für den Erhalt der Ressort-Daten notwendig ist. Eine Erweiterung der Programmierschnittstelle um

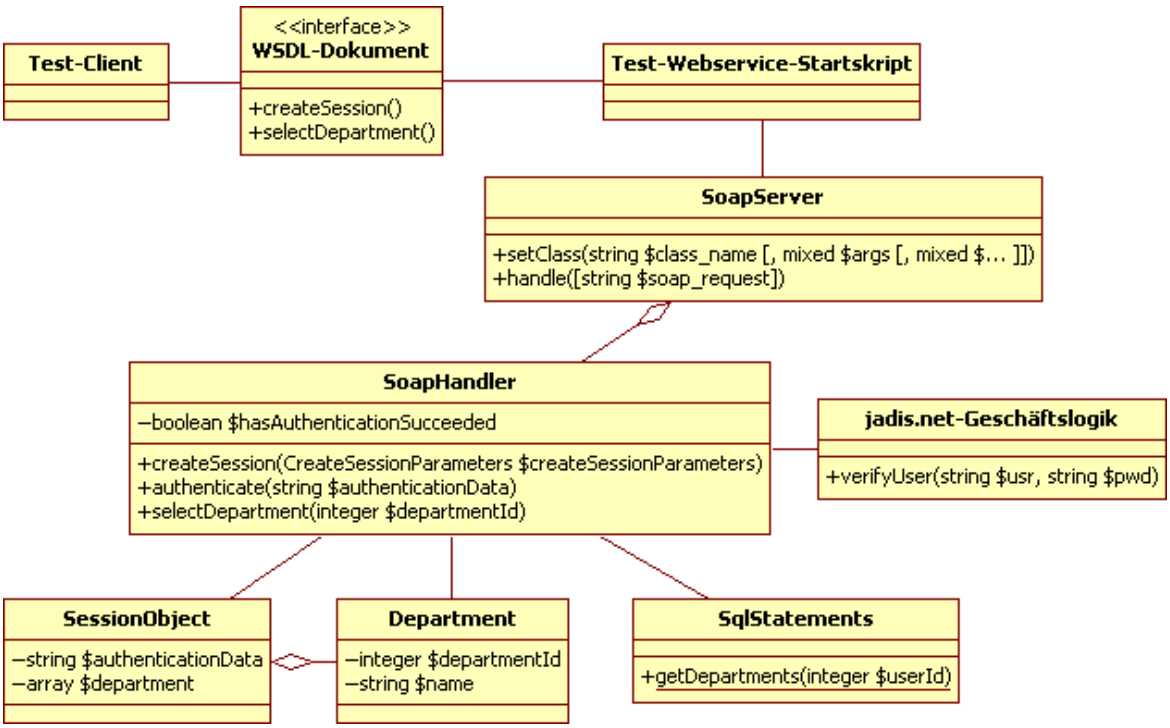


Abbildung 3.5.: Test-Webservice als Modul von jadis.net

eine eigene Geschäftslogik ist mithin nicht zu vermeiden. Dabei ist zu beachten, dass die Kopplung zwischen der Programmierschnittstelle und der jadis.net-Geschäftslogik so lose wie möglich ausfällt.[9] Andererseits sollte die Schnittstelle so weit wie möglich auf die öffentlichen Methoden der jadis.net-Geschäftslogik zugreifen. Ansonsten könnte eine zusätzliche Komplexität entstehen. Diese erwiese sich insbesondere im Hinblick auf Änderungen am Datenmodell als nachteilig, da Anpassungen dann sowohl an der jadis-net.Geschäftslogik als auch an der Programmierschnittstelle vorgenommen werden müssten.

3.3. Indirekte Kommunikation mit jadis.net

Bei der indirekten Kommunikation greift die Programmierschnittstelle über ein Netzwerkprotokoll auf die Geschäftslogik von jadis.net zu. Hierfür werden drei Architekturansätze untersucht. Im ersten kommuniziert die Schnittstelle über das XML-basierte Netzwerkprotokoll php/Java bridge mit der jadis.net-Geschäftslogik. Bei den beiden anderen Ansätzen werden SOAP-Nachrichten verschickt. Hier ist die Schnittstelle sowohl Webservice als auch

[9] Vgl. (Bal98) BALZERT, Helmut: *Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. S. 474

Webservice-Client. Letzterer ist sie im Hinblick auf die Kommunikation mit der jadis.net-Geschäftslogik.

Authentifizierungsmechanismus Die Authentifizierung läuft sowohl zwischen Webservice-Client und Programmierschnittstelle als auch zwischen Programmierschnittstelle und der Geschäftslogik von jadis.net ab. Die zusätzliche Authentifizierung ist notwendig, da der Betrieb von Schnittstelle und jadis.net auf zwei unterschiedlichen Rechnern erfolgt. Nach dem Versuch eines Clients, mit der Schnittstelle eine Sitzung zu eröffnen, übermittelt die Schnittstelle die empfangenen Werte für Benutzername und Passwort an die jadis.net-Geschäftslogik. Dort wird ihre Gültigkeit überprüft. Im Erfolgsfall erzeugt und übermittelt die Geschäftslogik ein Authentifizierungsdatum, mit dem sich die Programmierschnittstelle bei weiteren Operationen ausweisen muss. Danach erzeugt die Schnittstelle ein Authentifizierungsdatum und überträgt es an den Client. Bei jeder weiteren Operation liefert der Client dieses Authentifizierungsdatum mit. Die Schnittstelle überprüft die Daten und startet im Erfolgsfall die entsprechende Operation der jadis.net-Geschäftslogik. Dabei übermittelt sie ihrerseits das ihr zuvor von der Geschäftslogik übertragene Authentifizierungsdatum. Dieses Datum wird von der Geschäftslogik ausgewertet. Ist es korrekt, erhält die Schnittstelle die angeforderten Daten, die sie an den Client weiterleitet.

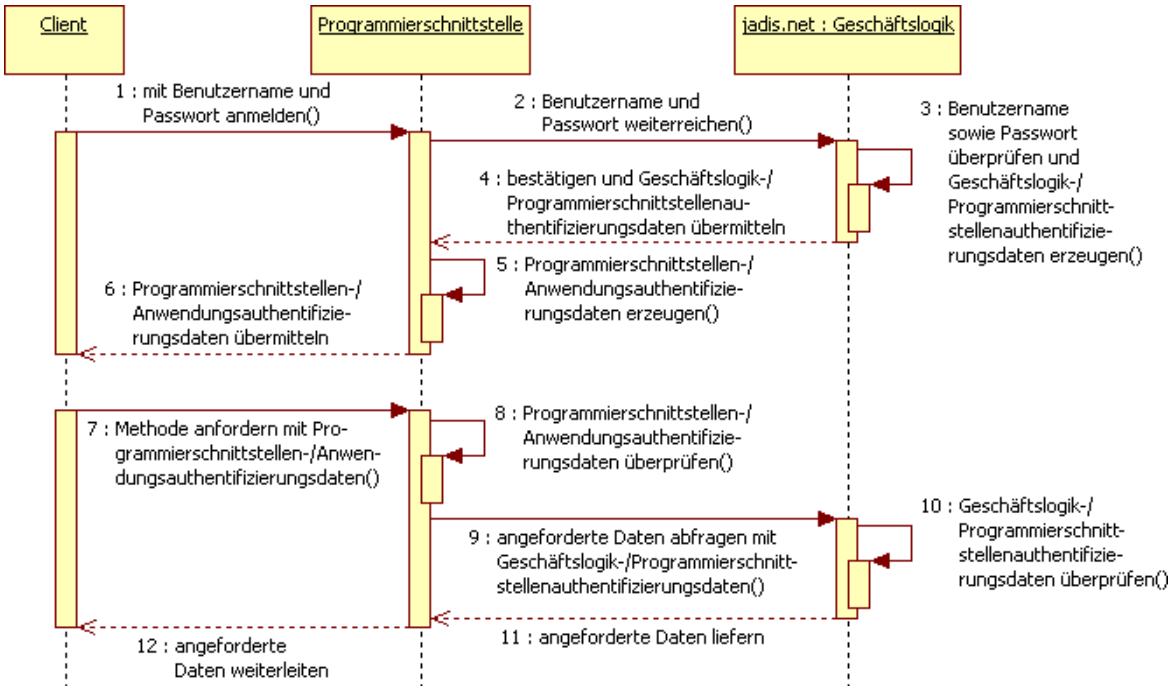


Abbildung 3.6.: Authentifizierungsmechanismus bei indirekter Kommunikation mit jadis.net

3.3.1. Programmierschnittstelle als Client mit php/Java bridge

3.3.1.1. Modell

Die Programmierschnittstelle ist in Java implementiert und wird von der Webservices-Engine Axis2 als Webservice verfügbar gemacht.

Die Webservices-Engine Axis2 findet bei diesem Modell Berücksichtigung, da sie gegenüber Axis Version 1.4 einige Vorteile bietet. Neben besserer Performance[10] und einer flexibleren Kommunikationsinfrastruktur[11] bietet sie Code-Generatoren als Plug-ins für die Entwicklungsumgebungen Eclipse und IntelliJ IDEA an[12]. Das entscheidende Kriterium für den Einsatz von Axis2 ist aber die bessere Unterstützung aktueller Webservice-Standards.[13]

Axis2 läuft als Servlet im Servlet-Container Apache Tomcat. Die Programmierschnittstelle ist dabei in das Verzeichnis von Axis2 für Webservices eingebunden. Wenn eine Anwendung eine Operation der Programmierschnittstelle einleitet, liefert die Schnittstelle die gewünschten Daten, indem sie auf die entsprechenden Methoden der jadis.net-Geschäftslogik zugreift. Das XML-basierte Netzwerkprotokoll php/Java bridge ermöglicht, dass die in Java entwickelte Programmierschnittstelle die PHP-Methoden der Geschäftslogik aufrufen kann.

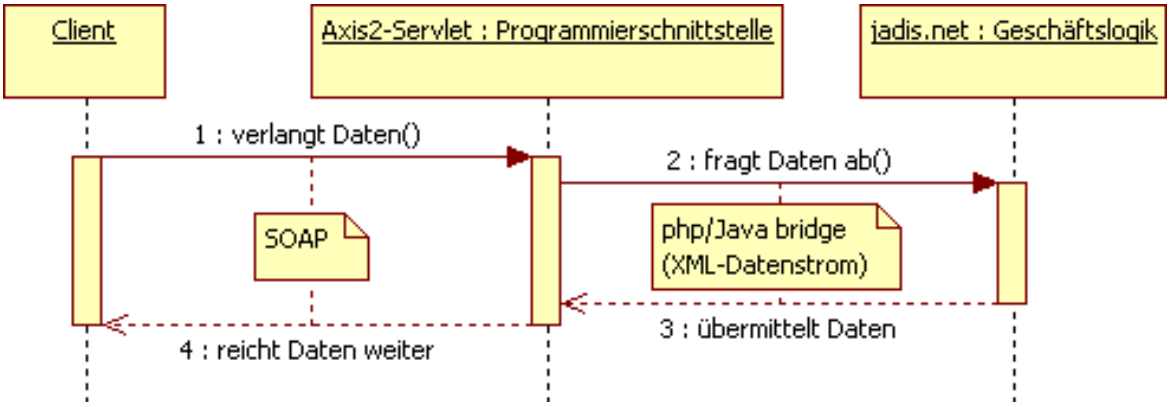


Abbildung 3.7.: Programmierschnittstelle als Client mit php/Java bridge

[10] Vgl. (FTW07) FROTSCHER, Thilo ; TEUFEL, Marc ; DAPENG, Wang: *Java Web Services mit Apache Axis2*. S. 24
[11] Vgl. (FTW07) S. 24
[12] Vgl. (Apa07b) APACHE SOFTWARE FOUNDATION, The: *Apache Axis2 Tools* http://ws.apache.org/axis2/tools/index.html
[13] Vgl. (FTW07) S. 24

3.3.1.2. Benötigte Software

Die Programmierschnittstelle setzt bei diesem Ansatz ein jadis.net-System voraus, das über ein Netzwerk erreichbar ist. Die Schnittstelle selbst ist auf einem anderen Rechner installiert.

Für die Entwicklung der Schnittstelle als Java-Client von jadis.net liefert das Java Standard Edition Development Kit in der Version 1.6[14] von Sun Microsystems Laufzeitumgebung, Programmierschnittstelle, Compiler und Debugger. Als Servlet-Container dient Apache Tomcat in der Version 6[15].

Da sich es bei der Schnittstelle um einen Java-Webservice handelt, wird die Webservices-Engine Axis2 benötigt. Sie wird in der Version 1.3[16] als Webarchiv zum Einsatz kommen.

Für die Kommunikation zwischen der Schnittstelle und den PHP-Skripten von jadis.net wird das XML-basierte Netzwerkprotokoll php/Java bridge in der Version 4[17] verwendet. Dies ist erforderlich, weil das Java Standard Edition Development Kit keine Schnittstelle anbietet, die die Ausführung PHP-Codes zuließe.

3.3.1.3. Test-Implementierung

Zur Einarbeitung in die Verwendung von php/Java bridge werden die Beispiele implementiert, die im Installationsdokument `INSTALL.J2SE` von php/Java bridge behandelt werden. Hierfür wird zunächst keine Servlet-Engine benötigt.

Die im Dokument `INSTALL.J2SE` behandelten Beispiele lassen sich nicht ausführen. Änderungen am Code und Modifizierungen der Konfiguration bleiben ebenso ergebnislos wie die Installation von php/Java bridge auf einem anderen Betriebssystem. Die Beispiele lassen sich weder unter Microsoft Windows XP Professional noch unter Debian/GNU Linux Version 4.0 ausführen.

[14] Vgl. (Sun08) SUN MICROSYSTEMS: *Sun Developer Network (SDN). Java SE Downloads*. http://java.sun.com/javase/downloads/?intcmp=1281

[15] Vgl. (Apa07a) APACHE SOFTWARE FOUNDATION, The: *Apache Tomcat. Tomcat 6 Downloads*. http://tomcat.apache.org/download-60.cgi

[16] Vgl. (Apa07b) *Axis2. Axis2 1.3 Release*. http://ws.apache.org/axis2/download/1_3/download.cgi

[17] Vgl. (BKD+07) BÖEKEMEIER, Jost u. a.: *php/Java bridge, Files*. http://sourceforge.net/project/showfiles.php?group_id=117793

3.3.1.4. Bewertung

Der Versuch einen Test-Client in Java zu implementieren, der das Netzwerkprotokoll php/Java bridge nutzt, schlug fehl. Es gelang nicht einmal, die Beispielprogramme zum Laufen zu bringen. Die Informationen auf der Website des php/Java-bridge-Projektes legen zudem nahe, dass das Protokoll größtenteils zum Aufrufen von Java-Code in PHP-Skripten verwendet wird.[18]

Aufgrund der nur knapp bemessenen Zeit für die Entwicklung der Programmierschnittstelle wird daher auf weitere Nachforschungen verzichtet.

Ein weiteres Argument, diesen Ansatz nicht weiter zu verfolgen, ist die Tatsache, dass die getestete vierte php/Java-bridge-Version derzeit von der fünften abgelöst wird. Es ist erstens nicht bekannt, wie lange die vierte Version noch gepflegt wird. Zweitens ist unklar, ob Anwendungen, die die vierte Version nutzen, noch reibungslos mit der fünften funktionieren.

Es ist jedoch nicht ausgeschlossen, dass dieser Ansatz grundsätzlich zum Erfolg führen würde. Problematisch ist allerdings, dass keine einheitliche Dokumentation von php/Java bridge vorhanden ist. Besonders das für die Programmierschnittstelle relevante Aufrufen von PHP-Methoden in Java-Anwendungen ist spärlich dokumentiert. Erklärungen erscheinen hauptsächlich in Installationsdateien, und eine verständliche Erläuterung der internen Funktionsweise von php/Java bridge ist nicht vorhanden. Immerhin erscheint ein regelmäßiger Newsletter.

3.3.2. Programmierschnittstelle als Webservice-Client in PHP implementiert

3.3.2.1. Modell

Dieser Ansatz sieht eine Implementierung der Programmierschnittstelle in PHP vor. Die Schnittstelle nutzt als Client die in einem WSDL-Dokument veröffentlichten Operationen eines jadis.net-Webservice. Der Nachrichtenaustausch geschieht dabei über SOAP.

3.3.2.2. Benötigte Software

Die Software für die Serverumgebung ist die gleiche wie bei der Implementierung der Schnittstelle als Modul von jadis.net (s. S. 20 f.). Es werden allerdings zwei Webserver benötigt. Auf

[18] Vgl. z. B. (BKD+07) http://php-java-bridge.sourceforge.net/pjb/examples.php

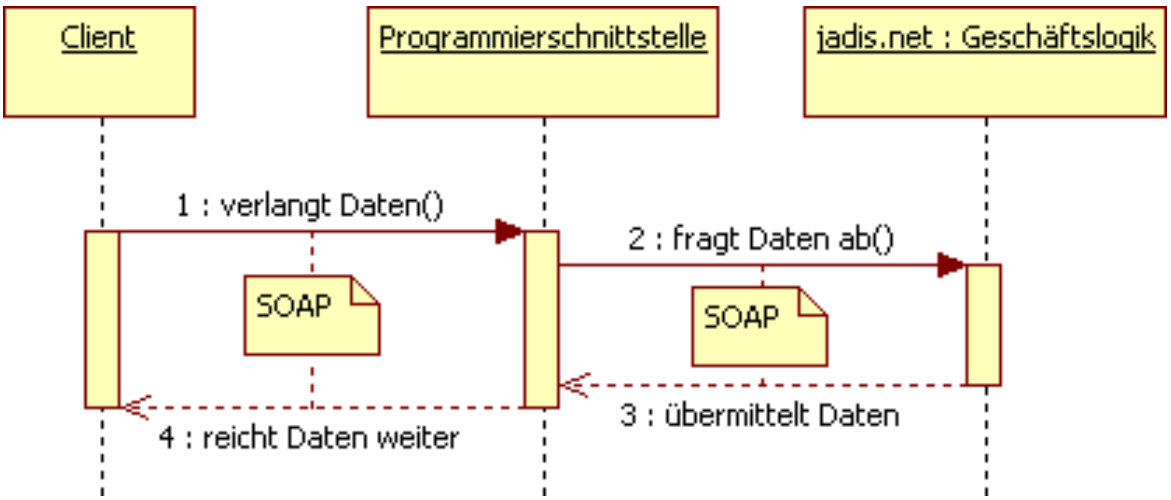


Abbildung 3.8.: Programmierschnittstelle als Webservice-Client

dem einen läuft das nun als Webservice ansprechbare jadis.net-System, auf dem anderen die Programmierschnittstelle.

3.3.2.3. Test-Implementierung

Als Webservice-Erweiterung für jadis.net wird der Test-Webservice genommen, der beim Modulansatz implementiert wurde (s. S. 21–24).

Das WSDL-Dokument des Test-Webservice, der Anfragen an den Test-Webservice für den Modulansatz weiterleitet, enthält fast denselben Inhalt wie das WSDL-Dokument des Modulansatz-Test-Webservice. Lediglich der URI muss so angepasst werden, dass er auf den neuen Test-Webservice verweist. Das Startskript des neuen Test-Webservice unterscheidet sich von dem des Modulansatz-Test-Webservice nur insofern, als dem Konstruktor der Klasse `SoapServer` der URI des neuen WSDL-Dokumentes übergeben wird.

Der Code der Klasse `SoapHandler` weicht hingegen deutlich von demjenigen des Modulansatz-Test-Webservice ab. `SoapHandler` besitzt ein `SoapClient`-Objekt als Instanzvariable. Das `SoapClient`-Objekt ermöglicht den Zugriff auf die Operationen des Modulansatz-Test-Webservice. Die Methoden `createSession(CreateSessionParameters $params)` und `selectDepartment(integer $departmentId)` rufen die gleichnamigen Methoden des `SoapClient`-Objektes auf. Die empfangenen Parameter werden dabei weitergeleitet. Die erhaltenen Rückgabewerte werden dann an den Test-Client übermittelt. Parameter und Rückgabewerte können ohne zusätzlichen Programmieraufwand weitergeleitet werden. Es

müssen keine ihrer Objektstruktur entsprechenden Klassen implementiert werden. Die Programmiersprache PHP bildet nämlich Klassen auf assoziative Arrays und umgekehrt ab.[19] In diesem Fall repräsentieren die Schlüssel eines assoziativen Arrays die Instanzvariablen, und die Werte des assoziativen Arrays repräsentieren die Werte der Instanzvariablen.

Die Authentifizierung des Test-Clients läuft entsprechend der Authentifizierung beim Test des Modulansatzes (s. S. 22 f.) ab.

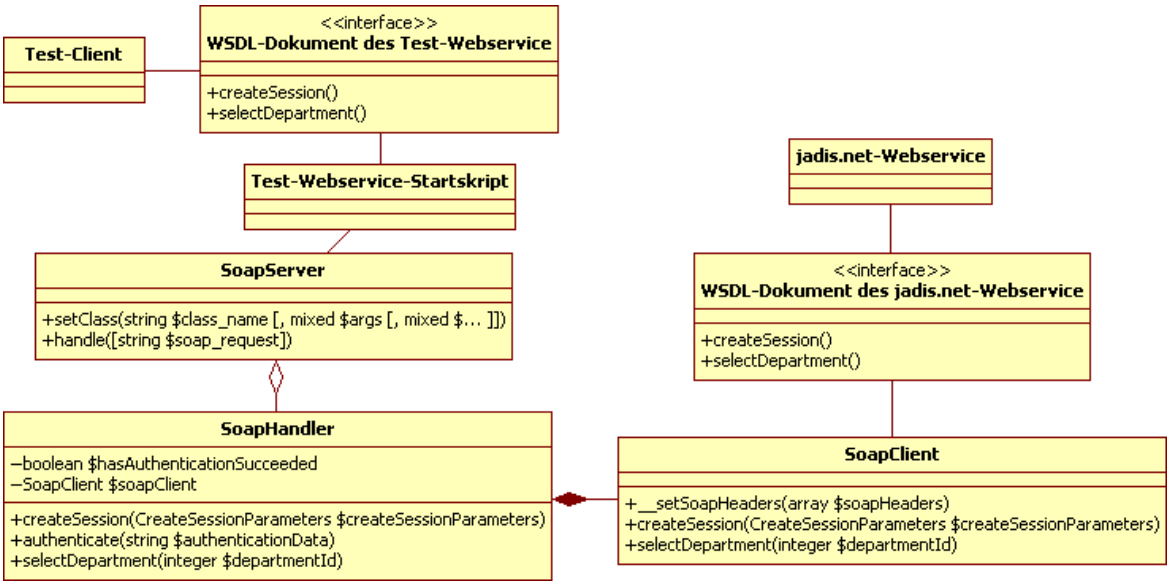


Abbildung 3.9.: Test-Webservice als Webservice-Client in PHP implementiert

3.3.2.4. Bewertung

Die Testanwendung konnte erfolgreich implementiert werden.

Da der Testwebservice das WSDL-Dokument fast vollständig aus dem WSDL-Dokument des Modulansatz-Test-Webservice ableiten konnte, enstand hierbei fast gar kein Aufwand. Die Implementierung der Klasse `SoapHandler` gestaltete sich ebenfalls unkompliziert. Hierfür mussten lediglich die bereits implementierten Operationen des Modulansatz-Test-Webservice aufgerufen werden.

[19] Vgl. (Ric06) S. 717 f.

3.3.3. Programmierschnittstelle als Webservice-Client in Java implementiert

3.3.3.1. Modell

Wie beim voherigen Ansatz (s. S. 28) wird auch hier die Programmierschnittstelle als Webservice-Client eines jadis.net-Webservice implementiert. Bei diesem Ansatz erfolgt die Entwicklung jedoch in der Programmiersprache Java.

3.3.3.2. Benötigte Software

Auch hier werden für das als Webservice erreichbare jadis.net-System und die Programmierschnittstelle wieder zwei Rechner verwendet.

Eine in Java realisierte Schnittstelle als Client eines jadis.net-Webservice erfordert fast die gleiche Umgebung wie die eines Java-Clients von jadis.net über php/Java bridge (s. S. 27). Der einzige Unterschied besteht im Verzicht auf php/Java bridge, da Schnittstelle und jadis.net über SOAP-Nachrichten miteinander kommunizieren.

3.3.3.3. Test-Implementierung

Als Webservice-Erweiterung für jadis.net dient wieder der Test-Webservice des Modulansatzes (s. S. 21–24).

Das von Axis2 mitgelieferte Programm WSDL2Java generiert mittels der entsprechenden WSDL-Dokumente die server- und clientseitigen Klassen. Dabei handelt es sich nicht nur um solche Klassen, die die Zusammensetzung der SOAP-Nachrichten und die Kommunikation mit den anderen Rechnern übernehmen. Für Serialisierung und Deserialisierung zuständige Klassen wie zum Beispiel `SessionObject` werden ebenfalls erzeugt. Allerdings existiert eine Klasse, die sowohl server- als auch clientseitig verwendet wird, dann mit demselben Code in doppelter Ausführung. Deswegen ist darauf zu achten, dass Server- und Client-Klassen in unterschiedlichen Paketen zusammengefasst werden. Der Paketname wird mit dem Schalter `-p`[20] beim Aufruf von WSDL2Java festgelegt.

Die Klasse `JadisNetWebServiceSkeleton` bietet Methodenrümpfe für die Webservice-Operationen `createSession` und `selectDepartment` an. Diese Rümpfe werden so editiert, dass sie über ein Objekt der generierten Client-Klasse

[20] Vgl. (Apa07b) *Apache Axis2 Advanced User's Guide*. http://ws.apache.org/axis2/1_3/adv-userguide.html

`JadisNetWebServiceStub` die Operationen des Modulansatz-Test-Webservice ausführen können. Die Rückgabewerte der Operationen werden dann zu den Rückgabewerten der `JadisNetWebServiceSkeleton`-Methoden verarbeitet. Hierzu müssen Tiefenkopien sämtlicher Werte gemacht werden.

Ein Beispiel für ein Objekt, das eine Tiefenkopie erfordert, ist das `SessionObject`-Objekt. Es enthält neben der Zeichenkette `authenticationData` ein Array von `Department`-Objekten, die wiederum die Zeichenketten `departmentId` und `name` besitzen. Zuerst müssen die Werte der Instanzvariablen `departmentId` und `name` sämtlicher `Department`-Objekte den Instanzvariablen neuer, für den Test-Client bestimmter, `Department`-Objekte zugewiesen werden. Dann wird ein für den Test-Client bestimmtes `SessionObject`-Objekt erzeugt. Über die Methode `addDepartment(Department param)` werden dem Array des neuen `SessionObject`-Objektes die neuen `Department`-Objekte zugewiesen.

Obwohl die im vorigen Absatz erwähnten Klassen das Interface `org.apache.axis2.databinding.ADBBean` implementieren, funktioniert kein Casting auf das Interface.

Zur Authentifizierung des Test-Clients wird das vom Modulansatz-Test-Webservice übermittelte Authentifizierungsdatum durch einen Zufallswert ersetzt. Der Zufallswert wird dem Test-Client zurückgegeben und dient diesem als Authentifizierungsdatum beim Test-Webservice.

Der Zufallswert ist außerdem der Schlüssel für eine Map, die die Beziehungen zwischen den Authentifizierungen enthält. Die Werte der Map sind `ClientIpAddressAndServerAuthData`-Objekte. Jedes dieser Objekte besitzt als Instanzvariablen die IP-Adresse des Test-Clients und das Authentifizierungsdatum für den Modulansatz-Test-Webservice. Wenn der Test-Client ein korrektes Authentifizierungsdatum übermittelt hat, liefert die Map das Datum für die Authentifizierung beim Modulansatz-Test-Webservice und die IP-Adresse des Test-Clients. Nun kann die Sitzung mit dem Modulansatz-Test-Webservice wieder aufgenommen werden. Die IP-Adresse des Test-Clients soll sicherstellen, dass das Authentifizierungsdatum für den Test-Client noch vom selben Rechner stammt. Wenn der Test-Client kein oder ein falsches Authentifizierungsdatum übertragen hat, kann kein entsprechender Schlüssel für die Map gefunden werden. Somit schlägt die Authentifizierung fehl, und die Anfrage wird nicht an den Modulansatz-Test-Webservice weitergeleitet.

Die Map ist eine Klassenvariable der Klasse `JadisNetWebServiceSkeleton`, weil für jede eingehende und ausgehende SOAP-Nachricht ein neues `JadisNetWebServiceSkeleton`-Objekt instanziiert wird. Die Map wird lediglich beim ersten Aufruf des `JadisNetWebServiceSkeleton`-Konstruktors instanziiert und ist für andere Klassen nicht sichtbar.

Die Klasse `JadisNetWebServiceSkeleton` hat zudem die Methode `init(MessageContext requestMessageContext)` implementiert. Diese Methode wird bei jeder Test-Client-Nachricht von der Methode `invokeBusinessLogic(MessageContext msgContext, MessageContext newMsgContext)` der Klasse `JadisNetWebServiceMessageReceiverInOut` aufgerufen. Für den Methodenaufruf muss die generierte Klasse `JadisNetWebServiceMessageReceiverInOut` editiert werden.

Das `MessageContext`-Objekt ist der aktuellen ein- oder ausgehenden Nachricht zugeordnet.[21] Dem Test-Webservice liefert es unter anderem den Header einer eingegangenen Nachricht. Dort befindet sich das Authentifizierungsdatum des Test-Clients. Zudem informiert das `MessageContext`-Objekt über die IP-Adresse des Senders.

3.3.3.4. Bewertung

Der Testverlauf zeigte, dass es prinzipiell möglich ist, einen PHP-Client die Operationen eines in Java implementierten Webservice verwenden zu lassen, der wiederum die Operationen eines PHP-Webservice nutzt. Die entsprechenden WSDL-Dokumente reichen aus, um Methoden der stark typisierten Programmiersprache Java mit der untypisierten Skriptsprache PHP aufzurufen und umgekehrt.

Der Vorteil der Webservices-Engine Axis2 ist unter anderem das Vorhandensein eines Werkzeuges, das sowohl server- als auch clientseitigen Java-Code generiert. Der serverseitige Code ist jedoch nicht besonders flexibel. So musste zum Beispiel die eigentlich vollständig generierte Klasse `JadisNetWebServiceMessageReceiverInOut` um eine Zeile Programmcode erweitert werden.

Axis2 liegt jedoch eine sehr flexible Architektur zugrunde. Um diese zu nutzen und nicht auf generierten Code angewiesen zu sein, ist allerdings ein hoher Einarbeitungsaufwand notwendig. Hierbei kann die spärliche Onlinedokumentation der Axis2-Programmierschnittstelle ein Hindernis sein.[22]

[21] Vgl. (FTW07) S. 247 f.
[22] Vgl. (Apa07b) *Overview (Axis2 API)*. http://ws.apache.org/axis2/1_3/api/index.html

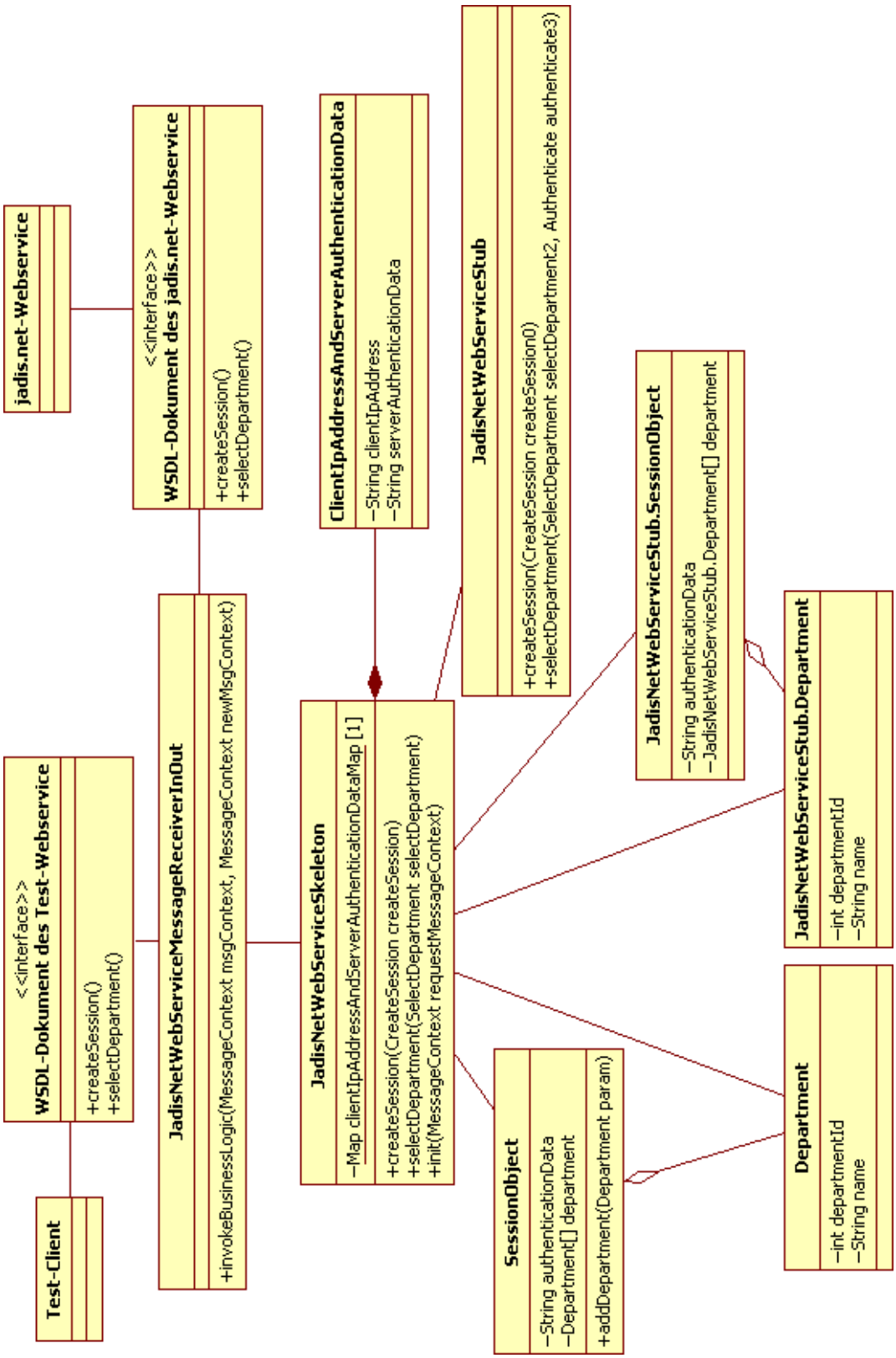


Abbildung 3.10.: Test-Webservice als Webservice-Client in Java implementiert

3.4. Entscheidung zugunsten einer Architektur

Nach einer Vorstellung der verschiedenen Architekturansätze bei der zweitwerk software engineering GmbH fiel der Entschluss, den Modulansatz zu implementieren.

Der Hauptgrund für die Entscheidung ist, dass diesem Ansatz die einfachste Architektur zugrunde liegt. Bei der Niedersächsischen Staats- und Universitätsbibliothek Göttingen muss bei diesem Ansatz neben der Installation der Programmierschnittstelle lediglich eine Umstellung der dort vorhandenen Serverumgebung vollzogen werden.

Dies ist aber bei näherem Betrachten keine triviale Aufgabe. So muss ein nicht geringer Teil der bestehenden jadis.net-Geschäftslogik umprogrammiert werden. Ansonsten wäre eine Anbindung des Schnittstellen-Moduls nicht möglich.

Der Grund für die notwendige Anpassung ist die Entstehungsgeschichte von jadis.net. Teile des heute noch verwendeten Codes stammen aus dem Jahre 2000. Zu diesem Zeitpunkt bot die Programmiersprache PHP noch nicht die Möglichkeit, einen konsequent objektorientierten Programmiersansatz zu verfolgen. Erst seit PHP 5 ist es zum Beispiel möglich, Objekte zu kapseln.[23]

Erschwerend kommt hinzu, dass die jadis.net-Geschäftslogik Code enthält, der eigentlich die Bildschirmausgabe betrifft. So trat bei der Test-Implementierung zum Beispiel der Fall auf, dass eine missglückte Anmeldung nicht den erwarteten booleschen Wert `FALSE` zurückgab, sondern nur eine Fehlermeldung auf dem Bildschirm vorsah. Eine erfolgreiche Anmeldung hätte hingegen den Wert `TRUE` geliefert.

Für die beiden Architekturansätze, die eine indirekte Kommunikation mit jadis.net vorsehen und deren Tests erfolgreich verliefen, spricht die Erweiterbarkeit. Es ist beispielshalber denkbar, neben der Universitätsversion von jadis.net auch die Standardversion um ein Programmierschnittstellen-Modul zu erweitern. Ein Client könnte dann über einen zwischengeschalteten Webservice Operationen nutzen, die entweder die eine oder die andere jadis.net-Version anböte. Dabei spielte es wäre es bdeitungslos, wenn der Client keine Informationen über den letztlichen Lieferaten der Daten erhielte.

Die Wahl der Programmiersprache für die Entwicklung eines zwischengeschalteten Webservice ist in Bezug auf jadis.net auch eine Geschmacksfrage.

[23] Vgl. (PHP08) *PHP Manual, Language References, Classes and Objects (PHP 5), Visibility* `http://www.php.net/manual/en/language.oop5.visibility.php`

Die folgende Tabelle listet die Bewertung der Testergebnisse in der Reihenfolge ihrer Relevanz auf. Das wichtigste Kriterium steht dabei an oberster Stelle.

	Direkte Kommunikation mit jadis.net	**Indirekte Kommunikation mit jadis.net**		
		über php/Java bridge	*über SOAP (PHP)*	*über SOAP (Java)*
Test erfolgreich verlaufen	ja	nein	ja	ja
einfache Installation	$\oplus$	$\ominus$	$\circ$	$\circ$
Code-Generierung möglich	nein	nein	nein	ja
einfache Implementierung	$\oplus$	nicht getestet	$\oplus$	$\circ$
Flexibilität	$\circ$	nicht getestet	$\oplus$	$\oplus$
Dokumentation	$\oplus$	$\ominus\ominus$	$\oplus$	$\circ$

Tabelle 3.1.: Bewertung der Architekturansätze

4. Implementierung

Dieses Kapitel behandelt die Implementierung der Programmierschnittstelle als Modul von jadis.net.

Als Vorgehensmodell für die Entwicklung dient der Contract-First-Ansatz[1]. Dieser Ansatz verlangt nach einer Definition der Schnittstelle in Form eines WSDL-Dokumentes *vor* ihrer Implementierung. Das WSDL-Dokument wird dabei dem WS-I-Anforderungsprofil[2] entsprechen.

Der Contract-First-Ansatz zwingt Programmierschnittstelle und Clients, beim Erstellen der SOAP-Nachrichten ausschließlich solche Datentypen zu verwenden, die einem XML-Schema-Datentyp des WSDL-Dokumentes entsprechen. Inkompatibilitäten der Datentypen beim Nachrichtenversand sind somit ausgeschlossen, und die Interoperabilität ist gewährleistet. Die Programmiersprache PHP ist zwar untypisiert; es ist aber nützlich sich den Contract-First-Ansatz im Hinblick auf spätere Webservice-Projekte anzueignen.

Wegen des Contract-First-Ansatzes wird zuerst das WSDL-Dokument vorgestellt. Danach folgt die Behandlung des Programmierschnittstellen-Codes.

Der letzte Teil des Kapitels beschäftigt sich mit einem clientseitigen Modultest und einer Webanwendung, die die Operationen der Schnittstelle nutzt.

4.1. WSDL-Dokument

Das WSDL-Dokument besteht aus folgenden fünf Abschnitten:

1. Der erste Abschnitt definiert ein XML-Schema mit Elementen und ihren Datentypen.[3]

   Es ist auch möglich, Schemas aus anderen Dateien im WSDL-Dokument zu referenzieren.

2. Hier werden aus den Elementen Nachrichten zusammengesetzt.[4]

[1] Vgl. (FTW07) S. 58–61
[2] Vgl. (WS-06) http://www.ws-i.org/Profiles/BasicProfile-1.1.html
[3] Vgl. (W3C01) Abschn. 2.2. http://www.w3.org/TR/wsdl
[4] Vgl. (W3C01) Abschn. 2.3. http://www.w3.org/TR/wsdl

3. In diesem Abschnitt fasst das Port-Typ-Element zueinander gehörende Nachrichten zu Operationen zusammen.[5]
4. Das Bindungs-Element legt in diesem Abschnitt das konkrete Übertragungsprotokoll und Format für die Operationen und Nachrichten des Port-Typs fest.[6]
5. Der letzte Abschnitt beschreibt den Service. Dem Service-Element wird ein Name zugewiesen und ein Port-Element übergeben.[7] Das Port-Element legt die Adresse, unter der die Programmierschnittstelle erreichbar ist, für die im vorigen Abschnitt definierte Bindung fest.[8]

Anzumerken wäre, dass ein WSDL-Dokument laut Spezifikation durchaus mehrere Port-Typ-, Bindungs-, Service- und Port-Elemente enthalten kann.[9] Die Programmierschnittstelle benötigt aber nur je eines dieser Elemente.

Im Folgenden wird anhand der Operationen `createSession` und `selectDepartment` der Aufbau des WSDL-Dokumentes für die Programmierschnittstelle beispielhaft erläutert.

4.1.1. Typen und Elemente

4.1.1.1. Typen

```
<xsd:complexType name="SessionObject">
  <xsd:sequence>
    <xsd:element
        maxOccurs="1"
        minOccurs="1"
        name="authenticationData"
        type="xsd:string" />
    <xsd:element
        maxOccurs="unbounded"
        minOccurs="0"
        name="department"
        type="tns:Department" />
  </xsd:sequence>
</xsd:complexType>

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

[5] Vgl. (W3C01) Abschn. 2.4. http://www.w3.org/TR/wsdl
[6] Vgl. (W3C01) Abschn. 2.5. http://www.w3.org/TR/wsdl
[7] Vgl. (W3C01) Abschn. 2.7. http://www.w3.org/TR/wsdl
[8] Vgl. (W3C01) Abschn. 2.6. http://www.w3.org/TR/wsdl
[9] Vgl. (W3C01) Abschn. 2.1. http://www.w3.org/TR/wsdl

```
17    <xsd:sequence>
        <xsd:element
19          maxOccurs="1"
            minOccurs="1"
21          name="departmentId"
            type="xsd:integer" />
23      <xsd:element
            maxOccurs="1"
25          minOccurs="0"
            name="name"
27          type="xsd:string" />
      </xsd:sequence>
29  </xsd:complexType>
```

In den Zeilen 1 bis 14 wird der komplexe Datentyp `SessionObject` definiert. Er enthält eine Sequenz als Element.

Eine Sequenz schreibt vor, dass die in ihr enthaltenen Elemente in exakt der Reihenfolge auftreten, die ihre Definition im XML-Schema vorgibt. Die Sequenz enthält genau ein Zeichenketten-Element `authenticationData` und eine unbestimmte Anzahl an `department`-Elementen, deren Typ in den Zeilen 16 bis 29 definiert wird. Die Attribute `maxOccurs` und `minOccurs` geben dabei die maximale beziehungsweise minimale Anzahl des Auftretens der Sequenzelemente an.

Bei der Implementierung der Programmierschnittstelle wird der komplexe Datentyp `SessionObject` später auf die gleichnamige Klasse abgebildet. Die Elemente `authenticationData` und `department` werden dann die Instanzvariablen der Klasse sein. Da das `department`-Element kein oder beliebig viele Male auftreten kann, wird die entsprechende Instanzvariable auf ein Array verweisen.

Die Zeilen 16 bis 29 beschreiben den komplexen Datentyp `Department`. Dieser enthält eine Sequenz, bestehend aus zwei Elementen. Das Ganzzahl-Element `departmentId` muss dabei genau einmal auftreten, während das Zeichenketten-Element `name` höchstens einmal auftreten darf.

`Department` wird später auf die Klasse desselben Namens mit den Instanzvariablen `departmentId` und `name` abgebildet.

Aus der Definition der komplexen Datentypen `SessionObject` und `Department` lässt sich ableiten, dass bei der Implementierung `SessionObject` und `Department` in einer *Has-a*-Relation zueinander stehen. Ein `SessionObject` hat dabei eine beliebige Anzahl an `Department`-Objekten.

Da die Elemente Begriffe der jadis.net-Terminlogie darstellen, lässt sich folgende Tabelle aufstellen. Die Bezeichnung in einer Spalte hat dabei ihre jeweiligen Entsprechungen in derselben Zeile.

jadis.net-Bezeichnung	**XML-Datentyp oder -Element**	**PHP-Code**
–	`SessionObject`	Klasse `SessionObject`
–	`authenticationData`	`$authenticationData` (Instanzvariable von `SessionObject`)
Ressort	`Department`	Klasse `Department` (als Array Instanzvariable von `SessionObject`)
Identifizierungsnummer eines Ressorts	`departmentId`	`$departmentId` (Instanzvariable von `Department`)
Name eines Ressorts	{codename	`$name` (Instanzvariable von `Department`

Tabelle 4.1.: Beziehungen zwischen jadis.net-Begriffen, XML-Datentypen und PHP-Code

4.1.1.2. Elemente

```
<xsd:element name="createSession">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element
          maxOccurs="1"
          minOccurs="1"
          name="username"
          type="xsd:string" />
      <xsd:element
          maxOccurs="1"
          minOccurs="1"
          name="password"
          type="xsd:string" />
    </xsd:sequence>
```

```
15    </xsd:complexType>
  </xsd:element>
17
  <xsd:element name="createSessionResponse" type="tns:SessionObject" />
19
  <xsd:element name="createSessionExceptionElement">
21    <xsd:complexType>
        <xsd:sequence>
23        <xsd:element
              maxOccurs="1"
25            minOccurs="1"
              name="message"
27            type="xsd:string" />
        </xsd:sequence>
29    </xsd:complexType>
  </xsd:element>
31
  <xsd:element name="authenticate" type="xsd:string" />
33
  <xsd:element name="authenticateExceptionElement">
35    <xsd:complexType>
        <xsd:sequence>
37        <xsd:element
              maxOccurs="1"
39            minOccurs="1"
              name="message"
41            type="xsd:string" />
        </xsd:sequence>
43    </xsd:complexType>
  </xsd:element>
45
  <xsd:element name="selectDepartment" type="xsd:integer" />
47
  <xsd:element name="selectDepartmentResponse" type="xsd:boolean" />
49
  <xsd:element name="selectDepartmentExceptionElement">
51    <xsd:complexType>
        <xsd:sequence>
53        <xsd:element
              maxOccurs="1"
55            minOccurs="1"
              name="message"
57            type="xsd:string" />
        </xsd:sequence>
```

```
59    </xsd:complexType>
</xsd:element>
```

Die Zeilen 1 bis 16 definieren das Element `createSession`. Es enthält eine Sequenz mit genau einem Zeichenketten-Element `username` und genau einem Zeichenketten-Element `password`.

Die 18. Zeile legt das Element `createSessionResponse` fest. Dieses Element ist vom Typ `tns:SessionObject`, der eigens für die Programmierschnittstelle definiert wurde (s. S. 39). Der vor dem Doppelpunkt stehende Ausdruck bei `tns:SessionObject` bezeichnet den Namensraum des Datentyps.

Die Zeilen 20 bis 30 widmen sich dem Element `createSessionExceptionElement`. Dieses Element ist ein komplexer Datentyp und enthält eine Sequenz. Die Sequenz enthält das Zeichenketten-Element `message`, welches genau einmal auftritt.

Die 32. Zeile definiert das Zeichenketten-Element `authenticate`.

Die Zeilen 34 bis 44 bestimmen das Element `authenticateExceptionElement`, welches die gleiche Struktur wie `createSessionExceptionElement` hat.

Die 46. Zeile definiert das Ganzzahl-Element `selectDepartment` und die 48. Zeile das boolesche Element `selectDepartmentResponse`.

Die Zeilen 50 bis 60 legen das Element `selectDepartmentExceptionElement` fest. Seine Struktur ist die gleiche wie bei `createSessionExceptionElement` und `authenticateExceptionElement`.

Die Namen der Elemente suggerieren bereits, welchen Nachrichten sie im nächsten Abschnitt zugeordnet werden.

4.1.2. Nachrichten

```
<wsdl:message name="CreateSessionRequestMessage">
2   <wsdl:part name="parameters" element="tns:createSession" />
</wsdl:message>
4
<wsdl:message name="CreateSessionResponseMessage">
6   <wsdl:part name="result" element="tns:createSessionResponse" />
</wsdl:message>
8
<wsdl:message name="CreateSessionException">
10  <wsdl:part name="noSession" element="tns:createSessionExceptionElement"
        />
</wsdl:message>
```

```
12
<wsdl:message name="AuthenticateMessage">
14   <wsdl:part name="requestHeader" element="tns:authenticate" />
</wsdl:message>
16
<wsdl:message name="AuthenticateException">
18   <wsdl:part
         name="failedAuthentication"
20       element="tns:authenticateExceptionElement" />
</wsdl:message>
22
<wsdl:message name="SelectDepartmentRequestMessage">
24   <wsdl:part name="parameters" element="tns:selectDepartment" />
</wsdl:message>
26
<wsdl:message name="SelectDepartmentResponseMessage">
28   <wsdl:part name="result" element="tns:selectDepartmentResponse" />
</wsdl:message>
30
<wsdl:message name="SelectDepartmentException">
32   <wsdl:part
         name="failedDepartmentSelection"
34       element="tns:selectDepartmentExceptionElement" />
</wsdl:message>
```

Hier werden die im Abschnitt Elemente definierten Elemente den `wsdl:part`-Elementen von Nachrichten zugeordnet.

Das WS-I-Profil schreibt für den in diesem WSDL-Dokument verwendet Stil `document-literal` vor, dass eine Nachricht höchstens ein `wsdl:part`-Element haben darf.[10] Der Grund ist, dass bei mehreren `wsdl:part`-Elementen der Name der Operation in der SOAP-Nachricht nicht vorhanden wäre. Stattdessen werden nun mehrere Parameter in einem Element zusammengefasst, welches dann den Namen der Operation haben kann.[11]

4.1.3. Port-Typ

```
1 <wsdl:portType name="JadisNet">

3   <wsdl:operation name="createSession">
```

[10] Vgl. (WS-06) R2201. http://www.ws-i.org/Profiles/BasicProfile-1.1.html
[11] Vgl. (But03) http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

```
    <wsdl:input message="tns:CreateSessionRequestMessage" />
    <wsdl:output message="tns:CreateSessionResponseMessage" />
    <wsdl:fault name="noSession" message="tns:CreateSessionException" />
  </wsdl:operation>

  <wsdl:operation name="selectDepartment">
    <wsdl:input message="tns:SelectDepartmentRequestMessage" />
    <wsdl:output message="tns:SelectDepartmentResponseMessage" />
    <wsdl:fault
        name="failedAuthentication"
        message="tns:AuthenticateException" />
    <wsdl:fault
        name="failedDepartmentSelection"
        message="tns:SelectDepartmentException" />
  </wsdl:operation>

</wsdl:portType>
```

Die Programmierschnittstelle verfährt nach dem Prinzip *request-response*. Auf eine Anfragenachricht sendet sie eine Antwortnachricht.[12].

Die Operation `createSession` hat `tns:CreateSessionRequestMessage` als Anfrage-, `tns:CreateSessionResponseMessage` als Antwort- und `tns:CreateSessionException` als Fehlernachricht.

`selectDepartment` fast analog dazu `tns:SelectDepartmentRequestMessage` als Anfrage-, `tns:SelectDepartmentResponseMessage` als Antwort- und `tns:SelectDepartmentException` als Fehlernachricht zusammen. Außerdem wird `selectDepartment` mit `tns:AuthenticateException` eine weitere Fehlernachricht zugewiesen.

Mit dem Port-Typ ist die abstrakte Beschreibung der Programmierschnittstelle abgeschlossen. Die innere Struktur der Operationen ist hiermit festgelegt. Eine Aussage über das Übertragungsprotokoll und das Nachrichtenformat wurde jedoch noch nicht gemacht. Hierfür ist die Bindung zuständig.

4.1.4. Bindung

```
<wsdl:binding name="JadisNetBinding" type="tns:JadisNet">

  <wsdlsoap:binding
      style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
```

[12] Vgl. (W3C01) Abschn. 2.4. http://www.w3.org/TR/wsdl

```
6
    <wsdl:operation name="createSession">
8     <wsdlsoap:operation soapAction="createSesssion" style="document" />
      <wsdl:input>
10      <wsdlsoap:body parts="parameters" use="literal" />
      </wsdl:input>
12    <wsdl:output>
        <wsdlsoap:body parts="result" use="literal" />
14    </wsdl:output>
      <wsdl:fault name="noSession">
16      <wsdlsoap:fault name="noSession" use="literal" />
      </wsdl:fault>
18  </wsdl:operation>

20  <wsdl:operation name="selectDepartment">
      <wsdlsoap:operation soapAction="selectDepartment" style="document" />
22    <wsdl:input>
        <wsdlsoap:header
24          message="tns:AuthenticateMessage"
            part="requestHeader"
26          use="literal" />
        <wsdlsoap:body parts="parameters" use="literal" />
28    </wsdl:input>
      <wsdl:output>
30      <wsdlsoap:body parts="result" use="literal" />
      </wsdl:output>
32    <wsdl:fault name="failedAuthentication">
        <wsdlsoap:fault name="failedAuthentication" use="literal" />
34    </wsdl:fault>
      <wsdl:fault name="failedDepartmentSelection">
36      <wsdlsoap:fault name="failedDepartmentSelection" use="literal" />
      </wsdl:fault>
38  </wsdl:operation>

40 </wsdl:binding>
```

Die Bindung legt in den Zeilen 3ff. das Übertragungsprotokoll fest. Bei der Programmierschnittstelle werden SOAP-Nachrichten über HTTP versandt.

Die Zeilen 8 und 20f. geben an, dass die jeweiligen Operationen den Stil `document` benutzen.

Das Nachrichtenformat wird durch das Element `wsdlsoap:body` beziehungsweise `wsdlsoap:fault` bestimmt. Das WS-I-Profil schreibt hierfür das Format `literal` vor.[13]

Für die Anfragenachricht der Operation `selectDepartment` sehen die Zeilen 23 bis 26 einen Header vor. Der Header übermittelt die Nachricht `tns:AuthenticateMessage`. In einer Anfragenachricht übermittelt ein Client hiermit das Authentifizierungsdatum, mit dem er sich bei der Programmierschnittstelle ausweist.

4.1.5. Service

```
<wsdl:service name="JadisNetWebService">
2   <wsdl:port name="JadisNetPort" binding="tns:JadisNetBinding">
      <wsdlsoap:address location="<path to webservice>" />
4   </wsdl:port>
</wsdl:service>
```

Das Element `wsdl:service` gibt der Programmierschnittstelle einen Namen. Es enthält ein `wsdl:port`-Element, dem die zuvor definierte Bindung zugewiesen wird. Das `wsdl:port`-Element seinerseits umfasst ein `wsdlsoap:address`-Element. Dieses Element legt die Adresse fest, unter der die Programmierschnittstelle erreichbar ist.

4.2. Webservice

Für jede im WSDL-Dokument beschriebene Operation ist in der Klasse `SoapHandler` eine gleichnamige Methode implementiert. Diese wird beim Einleiten einer Operation durch eine Anfragenachricht aufgerufen.

Wiederverwendung von Code Der Quellcode des Test-Webservice aus dem Modularansatz (s. S. 21–24) dient als Grundlage für die Implementierung der Programmierschnittstelle. Tatsächlich kann der Test-Webservice fast komplett übernommen werden, um das Startskript sowie die Methoden `createSession(CreateSessionParameters $params)`, `authenticate(AuthenticateParameters $authParams)` und `selectDepartment(integer $departmentId)` zu implementieren.

4.2.1. Klassenmodell

Das Klassendiagramm zeigt, dass die Klasse `SoapHandler` die zentrale Rolle im Modell der Programmierschnittstelle einnimmt.

[13] Vgl. (WS-06) R2706. http://www.ws-i.org/Profiles/BasicProfile-1.1.html

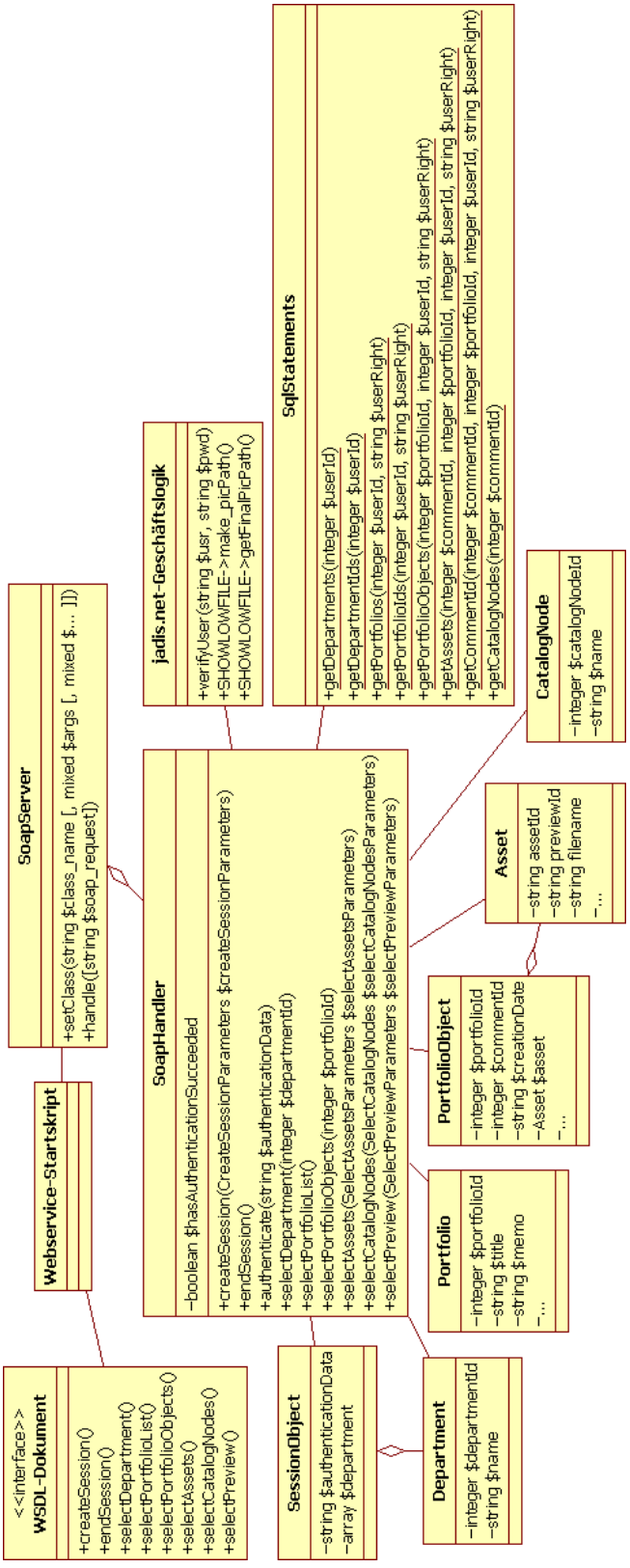


Abbildung 4.1.: Klassenmodell der Programmierschittstelle

4.2.2. Klasse SoapHandler

Die Klasse `SoapHandler` enthält `$hasAuthenticationSucceeded` als einzige Instanzvariable. Sie ist booleschen Typs und macht eine Aussage, ob der Benutzerclient authentifiziert ist und somit eine Methode ausgeführt werden kann. `$hasAuthenticationSucceeded` wird im Konstruktor der Wert `FALSE` zugewiesen. Bei jedem Aufruf der Methode `setClass(string $class_name [, mixed $args [, mixed ... ]])`[14] eines `SoapServer`-Objektes im Startskript wird eine neue Instanz der Klasse `SoapHandler` angelegt. Die Methode `authenticate(AuthenticateParameters $authParams)` wird aufgerufen, wenn das Authentifizierungsdatum des Benutzerclients im Header einer Nachricht übermittelt worden ist. Die Methode setzt dann den Wert von `$hasAuthenticationsSucceeded` auf `TRUE`, wenn ein korrektes Authentifizierungsdatum vorliegt. Mit anderen Worten bedeutet dies, dass für jede Operation außer der Sitzungseinleitung der Wert von `$hasAuthenticationSucceeded FALSE` ist. Die der Operation entsprechende Methode kann nur ausgeführt werden, wenn `authenticate(AuthenticateParameters $params)` vorher den Wert auf `True` gesetzt hat.

Falls bei einer Methode eine Exception auftritt, wird grundsätzlich ein SOAP-Fault an den Client übermittelt, der eine entsprechende Fehlernachricht enthält.

Die Tabelle auf der folgenden Seite listet die öffentlichen Methoden der Klasse `SoapHandler` und ihre entsprechenden Aktionen am jadis.net-System auf.

[14] Vgl. (PHP08) *PHP Manual, Function Reference, SOAP Functions, SoapServer->setClass()*. http://www.php.net/manual/en/function.soap-soapserver-setclass.php

Klasse `soapHandler`	**jadis.net-Aktion**
`createSession( CreateSessionParameters $params)`	Anmeldung beim jadis.net-System
`endSession()`	Abmeldung vom jadis.net-System
`authenticate( AuthenticateParameters $params)`	keine entsprechende Aktion; wird für die Auswertung des Authentifierungsdatums im Header einer SOAP-Nachricht benötigt
`selectDepartment(integer $departmentId)`	Auswahl eines Ressorts
`selectPortfolioList()`	Anfordern der Daten aller Mappen, für die der Benutzer Leserechte hat
`selectPortfolioObjects(integer $portfolioId)`	Anfordern der Daten aller Medienobjekte einer bestimmten Mappe, für die der Benutzer Leserechte hat
`selectAssets( SelectAssetsParameters $params)`	Anfordern der Daten aller Assets eines bestimmten Mappenobjektes
`selectCatalogNodes( SelectCatalogNodesParameters $params)`	Anfordern aller Katalogknoten, auf die der Benutzer Zugriff hat
`selectPreview( SelectPreviewParameters $params)`	Anfordern eines bestimmten Vorschaubildes, das zu einem bestimmten Asset gehört

Tabelle 4.2.: Beziehungen zwischen SoapHandler-Methoden und jadis.net-Aktionen

4.2.2.1. createSession(CreateSessionParameters $params)

Die Methode `createSession(CreateSessionParameters $params)` leitet mit dem Namen des Benutzers und seinem Passwort die Anmeldung am jadis.net-System an und initiiert somit eine Sitzung. Sie legt außerdem eine Sitzungsdatei an, die unter anderem die jadis.net-Identifizierungsnummer enthält. `createSession(CreateSessionParameters $params)` ruft dafür die PHP-Funktion `session_start()`[15] auf.

Die Anmeldung geschieht durch den Aufruf der Funktion `verifyUser(string $usr, string $pwd)` der jadis.net-Geschäftslogik. Ist die Anmeldung erfolgreich verlaufen, gibt die Funktion den booleschen Wert `TRUE` zurück. Danach wird ein Objekt der Klasse `SessionObject` angelegt. Der Name der Sitzungsdatei wird dabei der Instanzvariablen `$authenticationData` als künftiges Identifizierungsdatum des Benutzerclients zugewiesen. Darüber hinaus liefert die Klassenmethode `getDepartments(integer $userId)` der Klasse `SqlStatements` Identifizierungsnummer und Name aller Ressorts, für die der Benutzer registriert ist. Diese Tupel werden den Instanzvariablen `$departmentId` und `$name` je eines neuen `Department`-Objektes zugewiesen. Die `Department`-Objekte bilden dann die Elemente eines Arrays, das der Instanzvariablen `$department` des `SessionObject`-Objektes zugeordnet wird. Das `SessionObject`-Objekt wird schließlich an den Benutzerclient übermittelt.

Im Falle des Scheiterns der Anmeldung wird eine Exception geworfen. Die Exception wird dann als Fehlernachricht an den Client des Benutzers übermittelt. Des Weiteren löscht die PHP-Funktion `session_destroy()`[16] die Sitzungsdatei.

Parameterübergabe Für die Benutzeranmeldung werden der Benutzername und ein Passwort benötigt. Die Anfragenachricht `CreateSessionRequestMessage` der Operation `createSession` besitzt jedoch nur ein `wsdl:part`-Element (s. S. 42). Das `element`-Attribut des `wsdl:part`-Elementes ist aber das vorher definierte Element `tns:createSession`. Dieses Element enthält eine Sequenz mit den Elementen, die Name und Passwort des Nutzers repräsentieren (s. S. 42).

Der Parameter von `createSession(CreateSessionParameters $params)` ist somit ein Objekt der Klasse `CreateSessionParameters` mit den Instanzvariablen `$username` und `$password`.

[15] Vgl. (PHP08) *PHP Manual, Function Reference, Session Handling Functions, session_start* http://www.php.net/manual/en/function.session-start.php

[16] Vgl. (PHP08) *PHP Manual, Function Reference, Session Handling Functions, session_destroy* http://www.php.net/manual/en/function.session-destroy.php

Die Klasse `CreateSessionParameters` muss allerdings nicht implementiert werden. Der Parameter wird einfach als Instanz der vordefinierten PHP-Klasse `stdClass`[17] interpretiert. Die Instanzvariablen `$username` und `$password` werden ihr dabei automatisch zugeordnet.

Um zu verhindern, dass Parameter ohne Wert[18] übergeben werden, wird zu Beginn von `createSession(CreateSessionParameters $params)` überprüft, ob der Parameter einen Wert enthält. Darüber hinaus wird getestet, ob der Typ des Parameters demjenigen entspricht, der im WSDL-Dokument vorgeschrieben ist. Wenn einer der beiden Fälle nicht zutrifft, wird eine Exception geworfen. Diese Maßnahme gilt für sämtliche Methoden der Schnittstelle.

4.2.2.2. endSession()

`endSession` beendet die Sitzung, indem die PHP-Funktion `session_destroy()`[19] aufgerufen wird. Die Funktion löscht die Sitzungsdatei und gibt im Erfolgsfall `TRUE` zurück. Dieser Rückgabewert wird auch an den Benutzerclient übermittelt.

4.2.2.3. authenticate(AuthenticateData $authData)

Diese Methode wird aufgerufen, wenn im Header der Anfragenachricht das Authentifizierungsdatum übertragen worden ist. `authenticate(AuthenticateData $authData)` überprüft erstens, ob eine Sitzungsdatei existiert, die den Namen des übergebenen Parameters hat. Zweitens wird die akutelle IP-Adresse des Clients mit der in der Sitzungsdatei vermerkten IP-Adresse verglichen.

Fallen beide Tests positiv aus, wird die Instanzvariable `$hasAuthenticationSucceeded` auf `TRUE` gesetzt. Die Operation wird damit fortgesetzt, dass die dazugehörende Methode den Wert von `$hasAuthenticationSucceeded` überprüft.

Bei fehlendem oder falschem Authentifizierungsdatum wird eine Exception geworfen und eine Fehlernachricht per `SoapFault` übermittelt. Dasselbe gilt bei einer unbekannten IP-Adresse. Die Operation wird im Anschluss abgebrochen.

---

[17] Vgl. (PHP08) *PHP Manual, Appendices, List of Reserved Words, Predefined Classes*. http://www.php.net/manual/en/reserved.classes.php

[18] Vgl. (PHP08) *PHP Manual, Language Reference, Types, NULL* http://www.php.net/manual/en/language.types.null.php

[19] Vgl. (PHP08) *PHP Manual, Function Reference, Session Handling Functions, session_destroy* http://www.php.net/manual/en/function.session-destroy.php

4.2.2.4. selectDepartment(integer $departmentId)

`selectDepartment(integer $departmentId)` wählt für den Benutzer ein Ressort aus, das die als Parameter übergebene Identifizierungsnummer hat. Hierfür untersucht die Methode unter Zuhilfenahme der `SqlStatements`-Klassenmethode `getDepartments(integer $userId)`, ob der Benutzer für das gewählte Ressort registriert ist. Ist das der Fall, wird die Identifizierungsnummer des Ressorts in der Sitzungsdatei vermerkt. Sämtliche im Folgenden beschriebenen Methoden überprüfen mit dieser Information, ob eine Ressortauswahl vorliegt. Tut sie es nicht, wird eine Exception geworfen und eine Fehlernachricht übertragen.

`selectDepartment(integer $departmentId)` gibt `TRUE` zurück, wenn die Ressortauswahl funktioniert hat, andernfalls `FALSE`.

4.2.2.5. selectPortfolioList()

Die Methode `selectPortfolioList()` übermittelt eine Liste der Mappen, für die der Benutzer mindestens Leserechte besitzt. Die `SqlStatements`-Klassenmethode `getPortfolios(integer $userId, string $userRight)` liefert hierfür die Ergebnismenge.

Die Daten der Ergebnismenge werden den Instanzvariablen von `Portfolio`-Objekten zugewiesen. Ein Array von `Portfolio`-Objekten bildet dann den Rückgabewert von `selectPortfolioList()`.

Damit bei der späteren Serialisierung keine Probleme auftreten, werden jene Zeichenketten, die Datumswerte enthalten, in den entsprechenden Änderungsmethoden mit einem regulären Ausdruck umgewandelt. Dies ist notwendig, weil PHP den Datenbanktyp `date` nicht automatisch in das entsprechenden Format für den XML-Typen `date` umwandelt.

4.2.2.6. selectPortfolioObjects(integer $portfolioId)

Diese Methode liefert alle Medienobjekte einer Mappe, für die der Benutzer mindestens ein Leserecht besitzt. Sie tut dies unter Zuhilfenahme der `SqlStatements`-Klassenmethode `getPortfolioIds(integer $portfolioId, integer $userId, string $userRight)`.

Die Daten der Ergebnismenge werden auf den Instanzvariablen von `PortfolioObject`-Objekten abgebildet. Eine Außnahme bilden die Asset-Daten eines Mappenobjektes. Diese werden den Instanzvariablen eines `Asset`-Objektes zugeordnet. Das Asset-Objekt ist

dann eine Instanzvariable des entsprechenden `PortfolioObject`-Objektes. Ein Array von `PortfolioObject`-Objekten wird schließlich zurückgegeben.

Für Datumswerte gilt dieselbe Behandlung wie bei der Methode `selectPortfolioList()` (s. S. 52).

4.2.2.7. selectAssets(SelectAssetsParameters $params)

`selectAssets(SelectAssetsParameters $params)` übermittelt die Daten aller Assets eines bestimmten Medienobjektes. Die `SqlStatements`-Klassenmethode `getAssets(integer $commentId, integer $portfolioId, integer $userId, string $userRight)` liefert die entsprechenden Daten.

Die Asset-Daten werden den Instanzvariablen von `Asset`-Objekten zugewiesen. Diese Objekte werden als Elemente eines Arrays an den Benutzerclient übermittelt.

4.2.2.8. selectCatalogNodes(SelectCatalogNodesParameters $params)

Diese Methode gibt die Katalogknoten in der Systematik zurück, mit der ein Medienobjekt verknüpft ist. Die `SqlStatements`-Klassenmethode `getCatalogNodes(integer $commentId)` liefert die Ergebnismenge.

Die Daten der Ergebnismenge werden auf `CatalogNode`-Objekte abgebildet. Die `CatalogNode`-Objekte werden dann als Elemente eines Arrays zurückgegeben.

4.2.2.9. selectPreview(SelectPreviewParameters $params)

Mit dieser Methode wird das Vorschaubild eines Assets an den Benutzerclient übertragen.

Das `SelectPreviewParameters`-Objekt enthält unter anderem die Identifizierungsnummer des Vorschaubildes sowie sein Größenformat. Diese Werte werden an den Konstruktor der Klasse `SHOHLOWFILE` übergeben. Die Klasse `SHOWLOWFILE` ist Teil der jadis.net-Geschäftslogik und hilft beim Aufsuchen von Vorschaubildern im jadis.net-System. Die `SHOWLOWFILE`-Methoden `make_picPath()` und `getFinalPicPath()` liefern das gewünschte Bild.

Das Bild wird mit der PHP-Funktion `base64_encode(string $data)` [20] in das binäre Datenformat Base64 kodiert und kann so in der Antwortnachricht übertragen werden.[21] Das Clientprogramm des Benutzers muss die Bilddaten nach der Übertragung wieder decodieren.

4.3. Tests

4.3.1. Modultest

Der Modultest überprüft die Operationen der Programmierschnittstelle clientseitig. Testmethoden leiten die gleichnamigen Operationen der Schnittstelle ein. Innerhalb einer Testmethode werden die übertragenen Daten Vergleichen mit ihren Erwartungswerten unterzogen. Dabei wird auch das Verhalten von Exceptions überprüft.

Die überprüften Daten entstammen der Datenbank eines jadis.net-Testsystems. Die Datenbank ist während der Entwicklung der Schnittstelle nicht verändert worden, sodass keine Fehler wegen neuer oder aktualisierter Datensätze auftreten können. Die Reihenfolge der die Daten enthaltenen Elemente ist sichergestellt, die sie außnahmslos in Sequenzen übermittelt werden. Deshalb können keine Fehler auftreten, indem der erwartete Wert für ein Datum dem erwarteten Wert eines anderen zugeordnet würde und umgekehrt.

Die Operationen werden von einer in Java implementierten Test-Suite aufgerufen. Die Klassen der Test-Suite wurden mit dem Axis2-Werkzeug WSDL2Java automatisch erstellt. Lediglich jene Klassen, die ausschließlich für die Tests zuständig sind, wurden nachträglich implementiert.

4.3.1.1. Benötigte Software

Neben den üblichen Programmbibliotheken (s. S. 27) wird für den Modultest die Bibliothek JUnit[22] benötigt.

[20] Vgl. (PHP08) *PHP Manual, Function Reference, URL Functions, base64_encode* http://www.php.net/manual/en/function.base64-encode.php

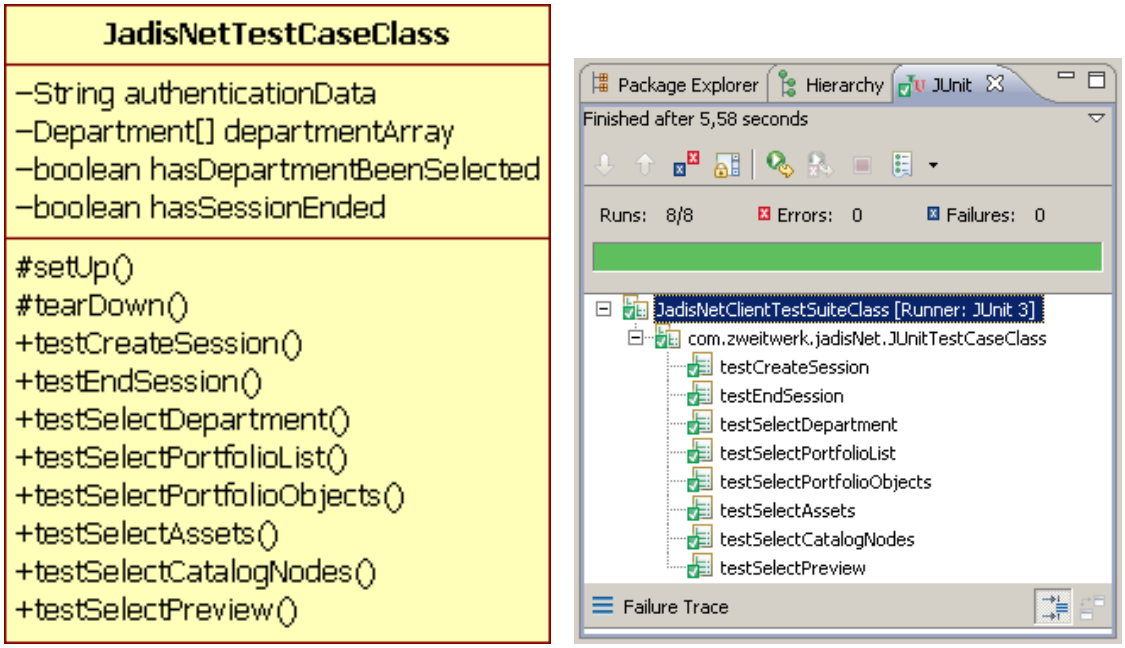
[21] Vgl. (W3C04b) W3C (WORLD WIDE WEB CONSORTIUM): *XML Schema Part 1: Structures Second Edition*. Abschn. 3.14.7. http://www.w3.org/TR/xmlschema-1/

[22] Vgl. (JUn02) JUNIT.ORG: *JUnit, Files*. http://sourceforge.net/project/showfiles.php?group_id=15278&package_id=12472

4.3.1.2. Testfall-Klasse

Die Klasse `JadisNetTestCaseClass` enthält die den Schnittstellenoperationen zugeordneten Testmethoden. Da die Testmethoden in keiner festgelegten Reihenfolge ausgeführt werden, können die Operationen `createSession`, `selectDepartment` und `endSession` dort nicht ausgeführt werden. Ansonsten wäre nicht sichergestellt, dass diese Operationen an erster, zweiter und letzter Stelle ausgeführt werden. Daher werden `createSession` sowie `selectDepartment` in der Methode `setUp()` und `endSession` in der Methode `tearDown()` ausgeführt. Diese Methoden werden vor beziehungsweise nach den Testmethoden augerufen. Die Ergebnisse der drei besagten Operationen werden Instanzvariablen zugewiesen, die dann in den entsprechenden Testmethoden mit den Erwartungswerten verglichen werden. Die Instanzvariable für die Operation `endSession` wird in der Testmethode auf den booleschen Wert `false` überprüft, da eine Sitzung während eines Testfalles selbstverständlich nicht beendet werden kann.

4.3.1.3. Ergebnis



(a) Testfall-Klasse (b) Ergebnisse des Modultests

Abbildung 4.2.: Testfall-Klasse und Ergebnisse des Modultests

Der Test findet auch nach mehrmaligem Durchlauf keine Fehler.

4.3.2. In PHP implementierte Webanwendung

Die Webanwendung soll einen typischen Anwendungsfall demonstrieren: Ein Benutzer lässt sich über einen Webbrowser das Vorschaubild eines Assets anzeigen. Hierfür sind folgende Schritte notwendig:

1. Zuerst meldet sich der Benutzer am jadis.net-System an.

   Eine fehlgeschlagene Anmeldung führt dazu, dass eine Fehlermeldung erscheint. Die Fehlermeldung ist innerhalb des `createSessionExceptionElement`-Elementes (s. S. 42) an die Webanwendung übertragen worden. Das Klicken auf den „zurück“-Link führt zur Anmeldeseite zurück.

2. Nach erfolgreicher Anmeldung wählt der Benutzer aus einer Liste von Ressorts, für die er registiert ist, eines aus.

3. Dann werden ihm die Namen aller Mappen in einer Auswahlliste angezeigt. Eine dieser Mappen wählt er aus.

4. Im nächsten Schritt erscheint eine Liste aller Medienobjekte der Mappe. Auch hiervon wählt er eines aus.

5. Eine weitere Liste lässt den Benutzer ein dem Mappenobjekt zugehöriges Asset auswählen.

6. Die letzte Liste enthält die Dateinamen aller Vorschaubilder des gewählten Assets. Der Benutzer wählt ein Bild aus und gibt dabei eines von vier Größenformaten an.

7. Nun erscheint das Vorschaubild auf seinem Browser. Der Benutzer verlässt die Anwendung, indem er auf den „Ende“-Knopf unter dem Bild klickt.

8. Zum Schluss wird dem Benutzer die Abmeldung bestätigt.

4.3.2.1. Benötigte Software

Die Webanwendung verlangt nach einem Apache HTTP-Server in der Version 2.2[23] und einer SOAP-fähigen PHP-5-Installation[24].

[23] Vgl. (Apa08) APACHE SOFTWARE FOUNDATION, The: *HTTP Server Project. Download*. http://httpd.apache.org/download.cgi

[24] Vgl. (PHP08) *Downloads*. http://www.php.net/downloads.php

Abbildung 4.3.: Zugriff auf Programmierschnittstelle von Webanwendung

4.3.2.2. Skripte und Klassen

Das Startskript lässt auf dem Webbrowser den Anmeldebildschirm erscheinen. Für jede Aktion wird ein neues HTML-Skript verwendet. Werte wie zum Beispiel die Identifizierungsnummer des ausgewählten Ressorts werden von einem Skript zum nächsten per HTTP-`POST`-Befehl übertragen. Das empfangende Skript kann die Werte aus der superglobalen PHP-Variable `$_POST`[25] auslesen und zum Einleiten einer weiteren Schnittstellenoperation nutzen.

Operationen der Programmierschnittstelle werden in Funktionen gesonderter PHP-Skripte eingeleitet. Diese Skripte erzeugen die benötigten `SoapClient`- und `SoapHeader`-Objekte. Sie implementieren wenn nötig auch die Klassen, auf denen die Parameter für die Anfragenachrichten abgebildet werden. Das zu Beginn der Sitzung von der Programmierschnittstelle zugeteilte Authentifizierungsdatum wird in der Sitzungsdatei der Webanwendung gespeichert.

Die Funktionen geben Teilmengen der von der Programmierschnittstelle übermittelten Daten zurück. Die meisten Daten werden für die Webanwendung nämlich nicht benötigt.

Das Vorschaubild wird nach seiner Übertragung im temporären Verzeichnis des Webservers gespeichert. Bei der Abmeldung wird es wieder gelöscht.

[25] Vgl. (PHP08) *PHP Manual, Appendices, List of Reserved Words, Predefined Variables*. `http://www.php.net/manual/en/reserved.variables.php`

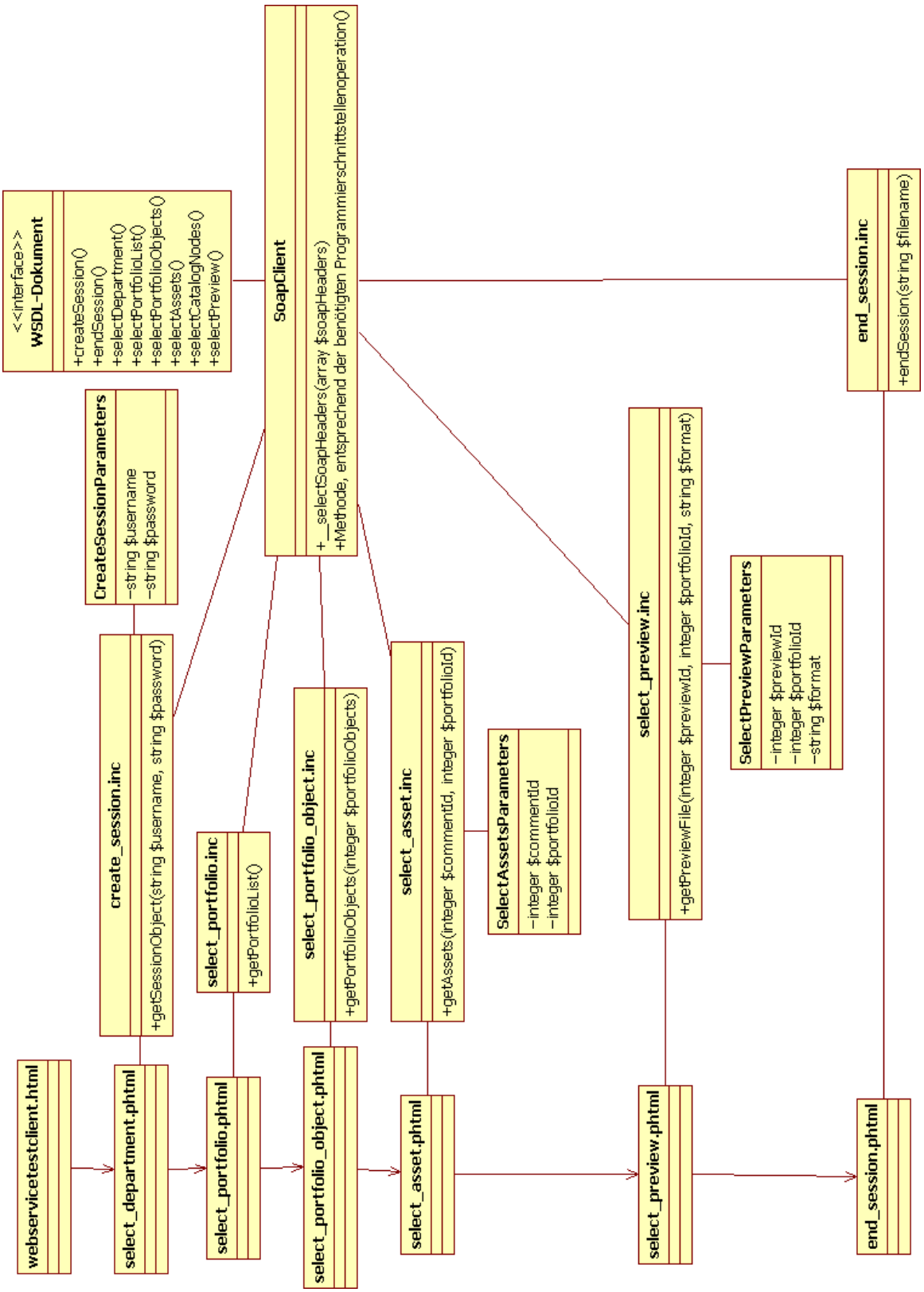
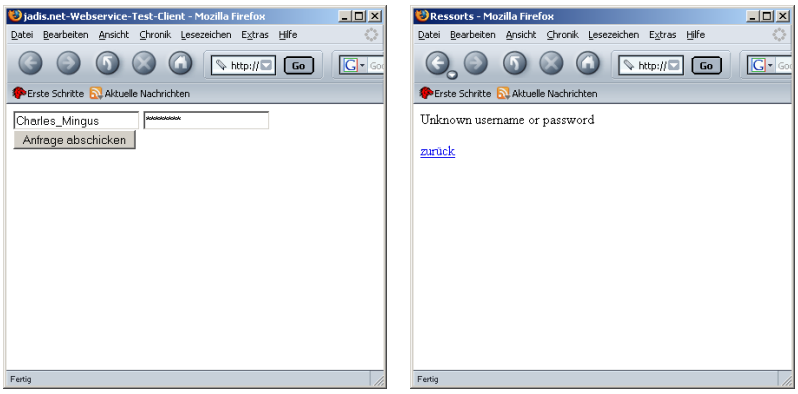


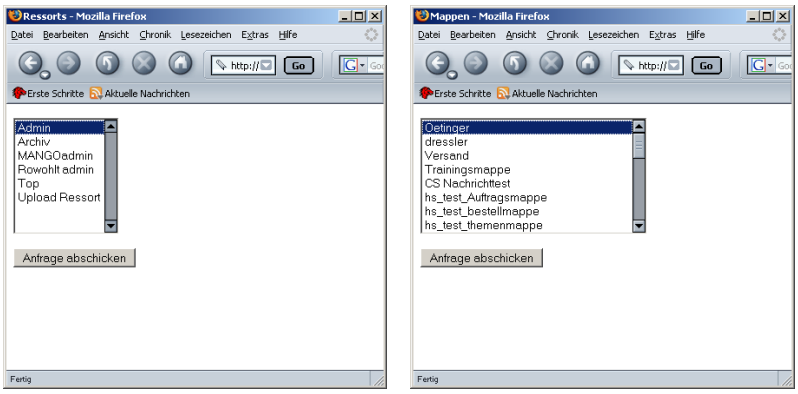
Abbildung 4.4.: Klassenmodell der Webanwendung

4.3.2.3. Die Webanwendung in Aktion



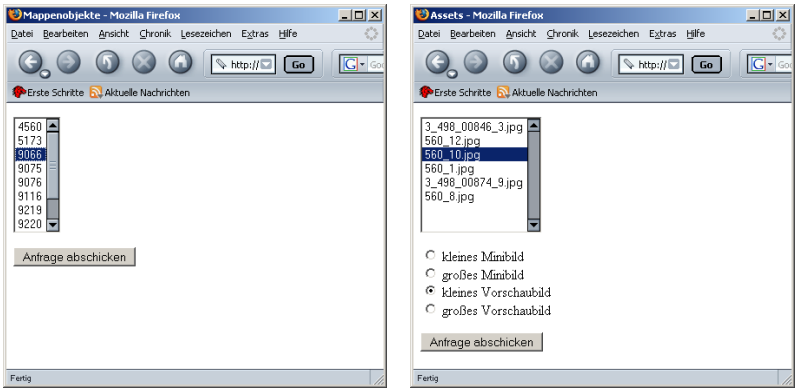
(a) Anmeldung (b) Fehlgeschlagene Anmeldung

Abbildung 4.5.: Anmeldung bei der Webanwendung



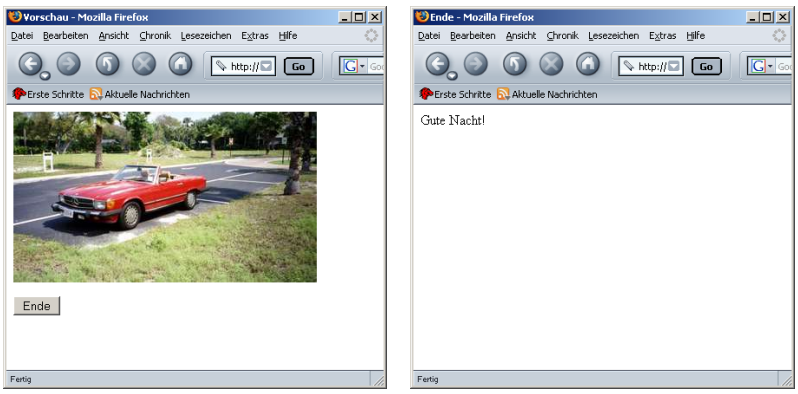
(a) Ressortauswahl (b) Mappenauswahl

Abbildung 4.6.: Mappen- und Ressortauswahl bei der Webanwendung



(a) Medienobjektauswahl (b) Assetauswahl

Abbildung 4.7.: Medienobjekt- und Assetauswahl bei der Webanwendung



(a) Vorschau (b) Ende

Abbildung 4.8.: Vorschau und Ende bei der Webanwendung

5. Fazit

Im ersten Teil dieses Kapitels wird die fertige Programmierschnittstelle im Hinblick der an sie im Kapitel „Analyse“ gestellten Anforderungen bewertet. Danach folgt eine allgemeine Kritik des Entwicklungsprozesses. Das Kapitel wird dann mit einem Ausblick auf die weitere Nutzung der Schnittstelle und die Zukunft von Webservices beschlossen.

5.1. Bewertung der Programmierschnittstelle

Die erste Version der Programmierschnittstelle gewährt den Zugriff auf bestimmte Daten der Universitätsversion von jadis.net. Hierbei handelt es sich ausschließlich um lesenden Zugriff. Dieser Anforderung wird die Schnittstelle vollständig gerecht.

Darüber hinaus bestehen zusätzliche Forderungen an die Schnittstelle:

Veröffentlichung als WSDL-Dokument Ein WSDL-Dokument beschreibt die Operationen der Programmierschnittstelle. Die Form der Beschreibung folgt einem De-facto-Standard, der das automatische Verarbeiten ihrer Informationen erleichtert. Das WSDL-Dokument lässt zum Beispiel das Axis2-Werkzeug WSDL2Java server- oder clientseitigen Java-Code generieren. Dieser Code kann dann für das Versenden oder Empfangen der SOAP-Nachrichten verwendet werden.

Für die Gebrauchstauglichkeit der Schnittstelle ist das WSDL-Dokument unverzichtbar. Es beschreibt sämtliche Operationen, die die Schnittstelle zur Verfügung stellt und definiert das Verfahren für den Nachrichtenaustausch. Somit ist sichergestellt, dass ein Benutzer nichts über die interne Struktur der Schnittstelle wissen muss. Sogar wenn sie selbst auf einen weiteren Webservice zugreifen müsste, wäre dies für den Benutzer vollkommen unbedeutend. Die entsprechenden Werkzeuge zur Code-Generierung vorausgesetzt, muss der Benutzer nicht einmal persönlich wissen, unter welcher Adresse die Programmierschnittstelle erreichbar ist. Manche Werkzeuge sind auch in der Lage, die Annotationen im WSDL-Dokument als Kommentar für die generierten Methoden und Klassen zu übernehmen. Mit diesen Voraussetzungen ausgestattet ist der Benutzer in der Lage, sich ausschließlich auf die Verarbeitung

der empfangenen Daten zu konzentrieren, fast so als ob er eine Programmierschnittstelle benützte, die ihm eine lokale Programmbibliothek bietet. Die einzige Einschränkung ist die Tatsache, dass er aufgrund des im WSDL-Dokument verwendeten Stils `document-literal`, zu übergebende Parameter den Instanzvariablen einer Klasse zuweisen muss. Er muss sich also darüber im Klaren sein, dass jede Methode, die eine Anfragenachricht verschickt, maximal einen Parameter übergeben bekommt.

Sind keine Code-Generierungswerkzeuge vorhanden, kann es allerdings mühselig werden, die für die Serialisierung notwendigen Klassen zu implementieren. Dies ist jedoch vor allem ein Problem des Zeitaufwandes und weniger der Komplexität.

Bereitstellung eines Authentifizierungsmechanismus Die Programmierschnittstelle nutzt die Art und Weise, wie die Programmiersprache PHP Sitzungen bei Webanwendungen wiederaufnimmt, für ihren Authentifizierungsmechanismus. Das beim Nachrichtenaustausch verwendete Authentifizierungsdatum entspricht dabei dem Namen der Sitzungsdatei, die bei einer neuen Verbindung von jadis.net angelegt wird. Dieser Ansatz hat sich als robust erwiesen. In sämtlichen Tests, bei denen das Authentifizierungsdatum manipuliert worden war, wurde eine Exception geworfen und ein SOAP-Fault versandt.

Das Authentifizierungsdatum wird nicht im Body, sondern im Header einer SOAP-Nachricht übermittelt. Somit besteht eine Trennung zwischen Authentifizierungs- und Nutzdaten bei der Nachrichtenübertragung.

Einfache Installtion beim Kunden Die Installation der Programmierschnittstelle beim Kunden verlangt die dortige Umstellung von PHP Version 4 auf Version 5. Ist diese mit den richtigen Grundeinstellungen vollzogen, verläuft die Installation der Schnittstelle problemlos. Es wird kein weiterer Server benötigt.

Einfache Integration in die bisherige jadis.net-Umgebung Die Integration in das vorhandene jadis.net-System wäre ohne Veränderungen an der jadis.net-Geschäftslogik nicht möglich gewesen. Ursachen waren einerseits älterer PHP-Code, der keiner strikten Trennnung von Frontend und Geschäftslogik folgt. Allerdings wird dieser Code zurzeit ohnehin überarbeitet.

Andererseits macht die Bereitstellung der Programmierschnittstelle eine Umstellung von PHP 4 auf PHP 5 nötig. Diese Umstellung ist zwar auch ohne die Entwicklung der Schnittstelle geplant, stellt aber in der Praxis ein nicht-triviales Problem dar. Die zweitwerk software engineering GmbH hat daher jeweils eine auf PHP 4 und eine auf PHP 5 aufbauende Version

ihrer Testsysteme in Betrieb. Die Programmierschnittstelle ist nur auf letzterer Version verfügbar. Eine Bereitstellung der Schnittstelle mittels der älteren PHP-Webservice-Frameworks NuSOAP [1] und PEAR:SOAP [2] war ausdrücklich nicht geplant.

Sicherheit Das Thema Sicherheit betrifft in erster Linie den autorisierten Zugriff auf die Daten. Dieser ist mit dem verwendeten Authentifizierungsmechanismus größtenteils gewährleistet. Das unbefugte Eindringen in eine bestehende Sitzung über ein abgefangenes Authentifizierungsdatum *und* eine gefälschte IP-Adresse ist jedoch nicht ausgeschlossen. Sofern der Missbrauch direkt im Internet geschieht, kann der Angreifer aber zumindest die Antwortnachricht nicht erhalten, da diese an die gefälschte IP-Adresse gesendet wird. Nur wenn er über denselben Proxy-Server wie der befugte Benutzer mit der Programmierschnittstelle kommuniziert, kann er Nachrichten auch entgegennehmen. Hier muss das Problem im Netz, das sich hinter dem Proxy-Server befindet, gelöst werden.

Benutzer, die Wert darauf legen, dass die SOAP-Nachrichten nicht im Klartext versandt werden, müssen eine SSL-Verbindung zur Programmierschnittstelle aufbauen. Da keine weiteren Stationen zwischen Benutzer und Programmierschnittstelle vorhanden sind, welche nur spezielle Elemente der SOAP-Nachrichten auswerten dürfen, ist dieses Verfahren ausreichend.

5.2. Kritik des Entwicklungsprozesses

Im Anfangsstadium der Entwicklung der Programmierschnittstelle lag das Hauptaugenmerk auf dem Verstehen der Struktur des jadis.net-Codes. Dies war notwendig, um die Methoden zu finden, welche die Benutzeranmeldung am jadis.net-System vollziehen und die zu übertragenden Daten liefern. Diese Aufgabe erwies sich erwartungsgemäß als schwierig. Das lag vor allem an der schieren Masse Dateien, in denen der Code enthalten ist. Ein kleines Problem stellte auch die Tatsache dar, das einige Methodennamen nicht die Funktionalität boten, die ihr Name suggerierte. Das hängt meiner Vermutung nach damit zusammen, dass jadis.net seit dem Jahr 2000 kontinuierlich weiterentwickelt und überarbeitet worden ist und sinnvollere Namen für manche Methoden schon von älteren Methoden besetzt waren. In diesem Zusammenhang möchte ich auch für einen einheitlichen Styleguide in Firmen plädieren.

[1] Vgl. (AN05) AYALA, Dietrich ; NICHOL, Scott: *NuSOAP – SOAP Toolkit for PHP*. http://sourceforge.net/projects/nusoap/

[2] Vgl. (PHP08) *PEAR, Packages, SOAP*. http://pear.php.net/package/SOAP

Verständlicherweise ist der gesamte jadis.net-Code zurzeit auf die Interpretation eines PHP-4-Interpreters ausgelegt. Dies ist im Hinblick auf die Defizite dieser PHP-Version problematisch. Es ist sicherlich nicht böswillig zu behaupten, dass PHP 4 nicht den Anforderungen gerecht wird, die an große Projekte gestellt werden. Als Beispiel sei hier nur die mangelnde Kapselung von Objekten genannt. Insofern ist es zu begrüßen, dass der Umstieg auf PHP 5 nun vollzogen wird.

Aus Sicht der objektorientierten Lehre ist es nicht wünschenswert, objektorientierten mit prozeduralem Code zu mischen. Ein Abstrahieren für Modellierungszwecke fällt dadurch schwer. Es wäre hilfreich gewesen, wenn der Programmierschnittstelle eine wohldefinierte jadis.net-Schnittstelle zur Verfügung gestanden hätte. Diese hätte man dann nach Bedarf erweitern können, ohne die Struktur von jadis.net zu verändern. Es muss hierbei abermals erwähnt werden, dass PHP erst nach und nach einen objektorientierten Aufsatz bekommen hat. Hier ähnelt die Situation in gewisser Hinsicht derjenigen der Programmiersprache C++. Ein C++-Compiler kann in vielen Fällen auch reinen C-Code verarbeiten.

An zweiter Stelle des Entwicklungsprozesses stand das Vertrautmachen mit dem Aufbau eines WSDL-Dokumentes. Da es für PHP kein erprobtes Werzeug für das Erzeugen eines WSDL-Dokumentes aus PHP-Code bietet, lag es nahe, den Contract-First-Ansatz zu befolgen. Dieser schreibt vor, das WSDL-Dokument vor dem Code eines Webservice anzufertigen. Rückblickend beanspruchte das Anfertigen des WSDL-Dokumentes neben dem Verständnis des jadis.net-Codes den höchsten Aufwand bei der Entwicklung der Programmierschnittstelle. Hierbei war besonders das Buch *Pro PHP XML and Web Services* (Ric06) von Richard Roberts eine große Hilfe. Zudem enthält es auch leicht nachvollziehbare Beispiele für die Entwicklung von Webservices und Webservice-Clients. Allerdings folgt das Buch in seinem Beispiel-WSDL-Dokument nicht ausschließlich den Empfehlungen des WS-I-Profils für die Interoperabilität, obwohl es selbst empfiehlt, das Profil zu befolgen. Das Buch verwendet den Typ `soapenc:Array`, den das WS-I-Profil ausschließt. Da ich das erste Test-WSDL-Dokument nach der im Buch-Beispiel vorgegebenen Struktur verfasste, musste ich festellen, dass WSDL2Java aus dem Dokument keinen clientseitigen Java-Code generierte.

Ich bin mir mittlerweile nicht mehr sicher, ob es die richtige Entscheidung war, im WSDL-Dokument den Stil `document-literal` zu verwenden. Ich tat dies, weil er dem WS-I-Profil entspricht und ich annahm, dass er häufig Anwendung findet. Beim Erstellen eines Clients zu Demonstrationszwecken stolperte ich aber zunächst über die Tatsache, dass mehrere Parameter einer Methode in *einem* Element der entsprechenden Webservice-Operation zusammengefasst werden und sie somit clientseitig den Instanzvariablen einer neuen Klasse zugewiesen werden müssen. In diesem Zusammehang war es praktisch, dass WSDL2Java die entsprechenden Java-Klassen automatisch erzeugte. Zudem wies die Code-Vervollständigung, die viele Programmierentwicklungsumgebungen anbieten, in diesem Fall darauf hin, das einer Methode ein einzelnes Objekt übergeben werden musste.

Für spätere Projekte werde ich jedoch den Stil `rpc-literal` in einem WSDL-Dokument testweise verwenden.

An dritter Stelle stand die Auswahl der Architektur für die Programmierschnittstelle. Nachdem der Modulansatz-Test-Webservice fertiggestellt worden war, war es einfach einen Webservice als Client des Modulansatz-Test-Webservice zu implementieren. Lediglich der Versuch, einem Test-Webservice Zugriff auf die jadis.net-Geschäftslogik über das Protokoll php/Java bridge zu gewähren, war ermüdend und schlug fehl.

Die schließliche Implementierung des Programmierschnittstellen-Codes war sowohl auf Seiten des Servers als auch des Clients unkompliziert. Die Klassen `SoapServer`, `SoapClient` und `SoapHeader` boten jede Funktionalität, die notwendig war, um die Auswertung der SOAP-Nachrichten nicht von Hand zu machen. Ich will jedoch nicht ausschließen, dass sie bei umfangreicheren SOAP-Nachrichten zu wenig flexibel sind.

5.3. Ausblick

Die Programmierschnittstelle wurde und wird zwar zurzeit nur für einen Kunden entwickelt; es steckt aber der Wunsch dahinter, die für die Systematik von jadis.net benötigte Funktionalität grundsätzlich öffentlich zu machen. Das bedeutet, dass Anwender ihre eigenen Frontends für jadis.net entwickeln können. Somit hat die Programmierschnittstelle auch einen Demonstrationszweck.

Im nächsten Schritt werden der Programmierschnittstelle Operationen hinzugefügt, die Schreibzugriffe auf die Mediendaten erlauben. Damit wird den Benutzern die Gelegenheit gegeben, Anwendungen zu implementieren, die eine tatsächliche Verwaltung der Daten gestatten.

Die Idee, die Schnittstelle als Webservice-Client eines jadis.net-Webservice zu konzipieren, nimmt einen älteren Ansatz bei der zweitwerk software engineering GmbH wieder auf. Dieser Ansatz sah zwar keine Architektur mit Webservices vor; ein zwischen Client und den verschiedenen jadis.net-Versionen operierender Vermittler war aber auch bereits hier vorgesehen. Insofern traf der indirekte Architekturansatz durchaus auf das Wohlwollen der Verantwortlichen bei der zweitwerk software engineering GmbH. Er wurde jedoch aufgrund seiner höheren Komplexität nicht ausgewählt. Es müssten mehrere Server aufgesetzt werden, was wiederum erheblichen Aufwand bedeutete. Die Staats- und Universitätsbibliothek Göttingen ist immerhin auf den laufenden Betrieb von jadis.net angewiesen. Dadurch, dass dieser Architeturansatz jedoch prinzipiell möglich ist, wird er eventuell in der Zukunft verwirklicht.

Ob sich Webservices im Allgemeinen langfristig etablieren können, hängt vor allem davon ab, dass die Hersteller einen einheitlichen Standard befolgen. Die Erfahrung zeigt, dass es keine

Rolle spielt, ob es sich dabei um eine tatsächliche Norm oder lediglich um einen De-facto-Standard handelt. Entscheidend wird sein, dass am Markt gleich stark auftretende Firmen nicht versuchen, gegensätzliche Standards zu etablieren, um sich somit einen Vorteil zu verschaffen. Das würde nämlich dazu führen, dass sich entweder kein Standard durchsetzte oder eine Firma einen Standard gemäß ihren Produkten definierte.

A. Quellcode

In den folgenden Abschnitten sind die für das Textverständnis hilfreichen Quellcodes aufgelistet. Auf den Abdruck des WSDL-Dokumentes und des PHP-Quellcodes für die Programmierschnittstelle wird verzichtet, da dies zu umfangreich wäre. Wichtige Teile sowohl des WSDL-Dokumentes als auch des Quellcodes sind außerdem bereits in den Test-Webservices enthalten. Der Code des Modultests und der Code der Beispiel-Webanwendung für die Programmierschnittstelle sind ebenfalls nicht abgedruckt. Sämtliche Quellcodes stehen jedoch auf der CD-ROM, die zusammen mit dieser Diplomarbeit erscheint, zur Verfügung.

A.1. Test-Webservices für die Architekturwahl

Hier sind das WSDL-Dokument und die PHP- und Java-Dateien für die Test-Webservices zum Wahl der Architektur abgedruckt. Eigentlich müsste es sich dabei um drei WSDL-Dokumente und noch mehr PHP-Dateien handeln. Da diese aber fast gleichen Inhalts sind, wird darauf verzichtet, sämtliche Dateien abzudrucken. Stattdessen stehen im Code Kommentare, die auf Unterschiede hinweisen. Dateien, die auf die jadis.net-Geschäftslogik zugreifen, und automatisch generierte Java-Klassen sind nicht abgedruckt. Abschließend steht der Quellcode eines PHP-Clients, der auf die Test-Webservices zugreift.

A.1.1. WSDL-Dokument

Das folgende WSDL-Dokument gilt für alle drei Test-Webservices.

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

3 <!-- the namespace must be set to "jadisnetwebservice" -->
  <!-- instead of "jadisNet" for the Java Test web service -->
5
  <wsdl:definitions
7 targetNamespace="urn:jadisNet.zweitwerk.com"
    xmlns:tns="urn:jadisNet.zweitwerk.com"
```

```
9     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
11    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

13  <!-- types and elements -->

15  <wsdl:types>

17    <!-- the namespace must be set to "jadisnetwebservice" -->
      <!-- instead of "jadisNet" for the Java Test web service -->
19
      <xsd:schema
21        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="urn:jadisNet.zweitwerk.com">
23
        <!-- types -->
25
        <xsd:complexType name="SessionObject">
27        <xsd:sequence>
            <xsd:element
29              maxOccurs="1"
                minOccurs="1"
31              name="authenticationData"
                type="xsd:string">
33          </xsd:element>
            <xsd:element
35              maxOccurs="unbounded"
                minOccurs="0"
37              name="department"
                type="tns:Department">
39          </xsd:element>
          </xsd:sequence>
41      </xsd:complexType>

43      <xsd:complexType name="Department">
          <xsd:sequence>
45          <xsd:element
                maxOccurs="1"
47              minOccurs="1"
                name="departmentId"
49              type="xsd:integer">
            </xsd:element>
51          <xsd:element
                maxOccurs="1"
```

```
53              minOccurs="0"
                name="name"
55              type="xsd:string">
            </xsd:element>
57        </xsd:sequence>
      </xsd:complexType>
59
      <!-- elements -->
61
      <xsd:element name="createSession">
63        <xsd:complexType>
            <xsd:sequence>
65            <xsd:element
                  maxOccurs="1"
67                minOccurs="1"
                  name="username"
69                type="xsd:string">
              </xsd:element>
71            <xsd:element
                  maxOccurs="1"
73                minOccurs="1"
                  name="password"
75                type="xsd:string">
              </xsd:element>
77          </xsd:sequence>
          </xsd:complexType>
79      </xsd:element>

81      <xsd:element
          name="createSessionResponse"
83        type="tns:SessionObject" />

85      <xsd:element name="createSessionExceptionElement">
          <xsd:complexType>
87          <xsd:sequence>
              <xsd:element
89                maxOccurs="1"
                  minOccurs="1"
91                name="message"
                  type="xsd:string">
93            </xsd:element>
            </xsd:sequence>
95        </xsd:complexType>
      </xsd:element>
```

```
97
        <xsd:element name="authenticate" type="xsd:string" />
99
        <xsd:element name="authenticateExceptionElement">
101       <xsd:complexType>
            <xsd:sequence>
103           <xsd:element
                  maxOccurs="1"
105               minOccurs="1"
                  name="message"
107               type="xsd:string">
              </xsd:element>
109         </xsd:sequence>
          </xsd:complexType>
111     </xsd:element>

113     <xsd:element name="selectDepartment" type="xsd:integer" />

115     <xsd:element name="selectDepartmentResponse" type="xsd:boolean" />

117     <xsd:element name="selectDepartmentExceptionElement">
          <xsd:complexType>
119         <xsd:sequence>
              <xsd:element
121               maxOccurs="1"
                  minOccurs="1"
123               name="message"
                  type="xsd:string">
125           </xsd:element>
            </xsd:sequence>
127       </xsd:complexType>
        </xsd:element>
129
    </xsd:schema>
131
  </wsdl:types>
133
  <!-- messages -->
135
  <wsdl:message name="CreateSessionRequestMessage">
137   <wsdl:part name="parameters" element="tns:createSession" />
  </wsdl:message>
139
  <wsdl:message name="CreateSessionResponseMessage">
```

```
141     <wsdl:part name="result" element="tns:createSessionResponse" />
      </wsdl:message>
143
      <wsdl:message name="CreateSessionException">
145     <wsdl:part name="noSession" element="
          tns:createSessionExceptionElement" />
      </wsdl:message>
147
      <wsdl:message name="AuthenticateMessage">
149     <wsdl:part name="requestHeader" element="tns:authenticate" />
      </wsdl:message>
151
      <wsdl:message name="AuthenticateException">
153     <wsdl:part
          name="failedAuthentication"
155       element="tns:authenticateExceptionElement" />
      </wsdl:message>
157
      <wsdl:message name="SelectDepartmentRequestMessage">
159     <wsdl:part name="parameters" element="tns:selectDepartment" />
      </wsdl:message>
161
      <wsdl:message name="SelectDepartmentResponseMessage">
163     <wsdl:part name="result" element="tns:selectDepartmentResponse" />
      </wsdl:message>
165
      <wsdl:message name="SelectDepartmentException">
167     <wsdl:part
          name="failedDepartmentSelection"
169       element="tns:selectDepartmentExceptionElement" />
      </wsdl:message>
171
      <!-- port -->
173
      <wsdl:portType name="JadisNet">
175
        <wsdl:operation name="createSession">
177       <wsdl:input message="tns:CreateSessionRequestMessage" />
          <wsdl:output message="tns:CreateSessionResponseMessage" />
179       <wsdl:fault
            name="noSession"
181         message="tns:CreateSessionException" />
        </wsdl:operation>
183
```

```
    <wsdl:operation name="selectDepartment">
185   <wsdl:input message="tns:SelectDepartmentRequestMessage" />
      <wsdl:output message="tns:SelectDepartmentResponseMessage" />
187   <wsdl:fault
          name="failedAuthentication"
189       message="tns:AuthenticateException" />
      <wsdl:fault
191       name="failedDepartmentSelection"
          message="tns:SelectDepartmentException" />
193 </wsdl:operation>

195 </wsdl:portType>

197 <!-- bindings -->

199 <wsdl:binding name="JadisNetBinding" type="tns:JadisNet">

201   <wsdlsoap:binding
        style="document"
203     transport="http://schemas.xmlsoap.org/soap/http" />

205   <wsdl:operation name="createSession">
        <wsdlsoap:operation soapAction="createSesssion" style="document" />
207     <wsdl:input>
          <wsdlsoap:body parts="parameters" use="literal" />
209     </wsdl:input>
        <wsdl:output>
211       <wsdlsoap:body parts="result" use="literal" />
        </wsdl:output>
213     <wsdl:fault name="noSession">
          <wsdlsoap:fault name="noSession" use="literal" />
215     </wsdl:fault>
    </wsdl:operation>
217
    <wsdl:operation name="selectDepartment">
219     <wsdlsoap:operation
          soapAction="selectDepartment"
221       style="document" />
        <wsdl:input>
223       <wsdlsoap:header
            message="tns:AuthenticateMessage"
225         part="requestHeader"
            use="literal" />
227       <wsdlsoap:body parts="parameters" use="literal" />
```

```
      </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body parts="result" use="literal" />
      </wsdl:output>
      <wsdl:fault name="failedAuthentication">
        <wsdlsoap:fault name="failedAuthentication" use="literal" />
      </wsdl:fault>
      <wsdl:fault name="failedDepartmentSelection">
        <wsdlsoap:fault name="failedDepartmentSelection" use="literal" />
      </wsdl:fault>
    </wsdl:operation>

  </wsdl:binding>

  <!-- service -->

  <wsdl:service name="JadisNetWebService">
    <wsdl:port name="JadisNetPort" binding="tns:JadisNetBinding">
      <wsdlsoap:address location="http://<path to webservice>" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

A.1.2. Programmierschnittstelle als Modul von jadis.net

A.1.2.1. Webservice-Startskript

Das folgende Skript ist der Einstiegspunkt des Webservice. Es wird auch für den Test-Webservice als PHP-Client eines jadis.net-Webservice (s. S. 80) verwendet.

```
<?php

/*************************************************************/
/*                                                           */
/* script which registers the class that contains the        */
/* implementation of the methods of the test web service with */
/* the PHP SOAP server object; the script serves also as the */
/* entry point for the execution process of a web service method */
/*                                                           */
/*************************************************************/
```

```
12 // the file which will be included depends if the web service is a
   // module of jadis.net or a PHP client of a jadis.net web service
14 require_once('soap_handler_for_test_module_of_jadis_net');

16 // require_once(
   //      'soap_handler_for_test_web_service_of_jadis_net_web_service.inc');
18
   define(
20          'URL_OF_WSDL_DOCUMENT',
            'http://<path to WSDL document>/TestWebService.wsdl');
22
   $server = new SoapServer(
24          URL_OF_WSDL_DOCUMENT,
            array('soap_version'=>SOAP_1_2, 'encoding'=>$html_charset));
26 $server->setClass('SoapHandler');
   $server->handle();
28
   ?>
```

A.1.2.2. Implementierung der Webservice-Methoden

```
1  <?php

3  require_once('jn_api_constants.inc');
   require_once('jn_api_department.inc');
5  require_once('jn_api_session_object.inc');
   require_once('jn_api_sql_statements.inc');
7  require_once('jn_login.inc');

9  /**
    * class which contains the implementation of the test web services's
11  * methods; it is called by the test web service entry script
    */
13 class SoapHandler {

15   /**
      * indicates whether or not correct authentication data have
17    * been submitted in the header of the SOAP request message
      *
19    * @var bool
      */
21   private $hasAuthenticationSucceeded;
```

```
23  public function __construct() {
      $this->hasAuthenticationSucceeded = FALSE;
25  }

27  public function createSession($params) {
      if (!(isset($params->username)
29        && is_string($params->username)
          && $params->username != '')) {
31
        // try to destroy session in case
33      // server has started it automatically
        session_destroy();
35      if (!(isset($params->password)
            && is_string($params->password)
37          && $params->password != '')) {
          $details = array('message'=>WebServiceConstants::
              MISSING_USERNAME_AND_PASSWORD_DETAILS_MESSAGE_VALUE);
39        throw new SoapFault(WebServiceConstants::NO_SESSION_FAULT_CODE,
              WebServiceConstants::
                  MISSING_USERNAME_AND_PASSWORD_FAULT_STRING,
41            $_SERVER['SERVER_NAME'],
              $details,
43            WebServiceConstants::NO_SESSION_FAULT_NAME);
        }
45      $details = array('message'=>WebServiceConstants::
            MISSING_USERNAME_DETAILS_MESSAGE_VALUE);
        throw new SoapFault(WebServiceConstants::NO_SESSION_FAULT_CODE,
47          WebServiceConstants::MISSING_USERNAME_FAULT_STRING,
            $_SERVER['SERVER_NAME'],
49          $details,
            WebServiceConstants::NO_SESSION_FAULT_NAME);
51    }
      if (!(isset($params->password)
53        && is_string($params->password)
          && $params->password != '')) {
55
        // try to destroy session in case
57      // server has started it automatically
        session_destroy();
59      $details = array('message'=>WebServiceConstants::
            MISSING_PASSWORD_DETAILS_MESSAGE_VALUE);
        throw new SoapFault(
61          WebServiceConstants::NO_SESSION_FAULT_CODE,
            WebServiceConstants::MISSING_PASSWORD_FAULT_STRING,
```

```
63              $_SERVER['SERVER_NAME'],
                $details,
65              WebServiceConstants::NO_SESSION_FAULT_NAME);
        }
67      session_start();

69      // if the global variable $api_access were not set to
        // true, the function verifyUser(string $usr, string $pwd)
71      // would call the printf(string $format [, mixed $args
        // [, mixed $... ]]) function and terminate
73      // without returning a boolean value
        global $api_access;
75      $api_access = TRUE;
        if (!verifyUser($params->username, $params->password)) {
77        session_destroy();
          $details = array('message'=>WebServiceConstants::
              UNKNOWN_USERNAME_OR_PASSWORD_DETAILS_MESSAGE_VALUE);
79        throw new SoapFault(
              WebServiceConstants::NO_SESSION_FAULT_CODE,
81            WebServiceConstants::UNKNOWN_USERNAME_OR_PASSWORD_FAULT_STRING,
              $_SERVER['SERVER_NAME'],
83            $details,
              WebServiceConstants::NO_SESSION_FAULT_NAME);
85      }

87      // write IP address of client into session
        $_SESSION['IpAddress'] = $_SERVER['REMOTE_ADDR'];
89      $sessionObject = new SessionObject();
        $sessionObject->setAuthenticationData(session_id());
91      $resultSet = SqlStatements::getDepartments($_SESSION['UsrId']);
        foreach ($resultSet as $key=>$value) {
93        $department = new Department();
          $department->setDepartmentId($value['ressortid']);
95        $department->setName($value['ressortstr']);
          $sessionObject->addDepartment($department);
97      }
        return $sessionObject;
99    }

101   /**
       * attempts to authenticate the user in order to process his request;
103    * this method is invoked by the header of each request message
       *
105    * @param the authentication data ID as a string value
```

```
     */
107  public function authenticate($authParams) {

109    // check if authentication data have been submitted
       if (!(isset($authParams)
111        && is_string($authParams)
           && $authParams != '')) {
113      $details = array('message'=>WebServiceConstants::
             MISSING_AUTHENTICATION_DATA_DETAILS_MESSAGE_VALUE);
         throw new SoapFault(
115          WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
             WebServiceConstants::MISSING_AUTHENTICATION_DATA_FAULT_STRING,
117          $_SERVER['SERVER_NAME'],
             $details,
119          WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
       }
121    session_id($authParams);
       session_start();
123
       // check if session exists; if variable "IpAddress" is
125    // not set, the existing session has not been resumed;
       // thus invalid authentication data have been submitted
127    if (!isset($_SESSION['IpAddress'])) {
         session_destroy();
129      $this->
             throwFailedAuthenticationBecauseInvalidAuthenticationDataException
             ();
       }
131
       // check if IP address of client corresponds with session
133    if (strcmp($_SESSION['IpAddress'], $_SERVER['REMOTE_ADDR']) != 0) {
         $this->
             throwFailedAuthenticationBecauseInvalidAuthenticationDataException
             ();
135    }
       $this->hasAuthenticationSucceeded = TRUE;
137  }

139  public function selectDepartment($departmentId) {
       if (!$this->hasAuthenticationSucceeded) {
141      $this->throwFailedAuthenticationException();
       }
143    if (!(isset($departmentId) && is_integer($departmentId))) {
```

```
        $details = array('message'=>WebServiceConstants::
            NO_DEPARTMENT_ID_SUBMITTED_DETAILS_MESSAGE_VALUE);
145     throw new SoapFault(
            WebServiceConstants::FAILED_DEPARTMENT_SELECTION_FAULT_CODE,
147         WebServiceConstants::NO_DEPARTMENT_ID_SUBMITTED_FAULT_STRING,
            $SERVER['SERVER_NAME'],
149         $details,
            WebServiceConstants::FAILED_DEPARTMENT_SELECTION_FAULT_NAME);
151   }
      $isUserMemberOfDepartment = FALSE;
153   $resultSet = SqlStatements::getDepartmentIds($_SESSION['UsrId']);

155   // check if user is member of selected department
      for ($i = 0; $i < count($resultSet); $i++) {
157     if ($departmentId == $resultSet[$i]['ressortid']) {
          $isUserMemberOfDepartment = TRUE;
159       break;
        }
161   }
      if ($isUserMemberOfDepartment) {
163
        // write ID of selected department into
165     // session if user is member of department
        $_SESSION['DepartmentId'] = $departmentId;
167   }
      return $isUserMemberOfDepartment;
169 }

171 private function throwFailedAuthenticationException() {
      $details = array('message'=>WebServiceConstants::
          NO_VALID_AUTHENTICATION_DATA_SUBMITTED_DETAILS_MESSAGE_VALUE);
173   throw new SoapFault(
          WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
175       WebServiceConstants::
              NO_VALID_AUTHENTICATION_DATA_SUBMITTED_FAULT_STRING,
          $SERVER['SERVER_NAME'],
177       $details,
          WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
179 }

181 private function
      throwFailedAuthenticationBecauseInvalidAuthenticationDataException()
       {
```

```
        $details = array('message'=>WebServiceConstants::
            INVALID_AUTHENTICATION_DATA_DETAILS_MESSAGE_VALUE);
183     throw new SoapFault(
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
185         WebServiceConstants::INVALID_AUTHENTICATION_DATA_FAULT_STRING,
            $_SERVER['SERVER_NAME'],
187         $details,
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
189   }
}
191
?>
```

A.1.3. Programmierschnittstelle als PHP-Client eines jadis.net-Webservice

A.1.3.1. Webservice-Startskript

Der PHP-Test-Client eines jadis.net-Webservice verwendet das gleiche Startskript wie das Testmodul von jadis.net (s. S. 74).

A.1.3.2. Implementierung der Webservice-Methoden

```
<?php
2
require_once('jn_api_constants.inc');
4 require_once('jn_api_department.inc');
require_once('jn_api_session_object.inc');
6 require_once('jn_api_sql_statements.inc');
require_once('jn_login.inc');
8
define('ISO_CHARACTER_SET', 'iso-8859-1');
10
define(
12   'URL_OF_WSDL_DOCUMENT',
   'http://<path to WSDL document>/TestWebService.wsdl');
14
/**
16  * class which contains the implementation of the test web services's
 * methods; it is called by the test web service entry script
18  */
class SoapHandler {
```

```
20
    /**
22   * indicates whether or not correct authentication data have
     * been submitted in the header of the SOAP request message
24   *
     * @var bool
26   */
    private $hasAuthenticationSucceeded;
28
    /**
30   * @var SoapClient;
     */
32  private $soapClient;

34  public function __construct() {
      $this->hasAuthenticationSucceeded = FALSE;
36    $this->soapClient = new SoapClient(
        URL_OF_WSDL_DOCUMENT,
38      array(
          'soap_version'=>SOAP_1_2,
40        'encoding'=>ISO_CHARACTER_SET));
    }
42
    public function createSession($params) {
44    session_start();
      try {
46      $createSessionResponse =
          $this->soapClient->createSession(
48          $params);
        $_SESSION['AuthenticationData'] =
50        $createSessionResponse->authenticationData;

52      // write IP address of client into session
        $_SESSION['IpAddress'] = $_SERVER['REMOTE_ADDR'];
54      $createSessionResponse->authenticationData =
          session_id();
56      return $createSessionResponse;
      } catch (SoapFault $e) {
58      session_destroy();
        throw $e;
60    }
    }
62
    /**
```

```
64      * attempts to authenticate the user in order to process his request;
        * this method is invoked by the header of each request message
66      *
        * @param the authentication data ID as a string value
68      */
    public function authenticate($authParams) {
70
      // check if authentication data have been submitted
72    if (!(isset($authParams)
          && is_string($authParams)
74        && $authParams != '')) {
        $details = array('message'=>WebServiceConstants::
            MISSING_AUTHENTICATION_DATA_DETAILS_MESSAGE_VALUE);
76      throw new SoapFault(
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
78          WebServiceConstants::MISSING_AUTHENTICATION_DATA_FAULT_STRING,
            $_SERVER['SERVER_NAME'],
80          $details,
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
82    }
      session_id($authParams);
84    session_start();

86    // check if session exists; if variable "IpAddress" is
      // not set, the existing session has not been resumed;
88    // thus invalid authentication data have been submitted
      if (!isset($_SESSION['IpAddress'])) {
90      session_destroy();
        $this->
            throwFailedAuthenticationBecauseInvalidAuthenticationDataException
            ();
92    }

94    // check if IP address of client corresponds with session
      if (strcmp($_SESSION['IpAddress'], $_SERVER['REMOTE_ADDR']) != 0) {
96      $this->
            throwFailedAuthenticationBecauseInvalidAuthenticationDataException
            ();
      }
98    $this->hasAuthenticationSucceeded = TRUE;
    }
100
    public function selectDepartment($departmentId) {
102   if (!$this->hasAuthenticationSucceeded) {
```

```
        $this->throwFailedAuthenticationException();
104     }
        $header = new SoapHeader(
106         'urn:jadisNet.zweitwerk.com',
            'authenticate',
108         $_SESSION['AuthenticationData']);
        $this->soapClient->__setSoapHeaders(array($header));
110     try {
          return $this->soapClient->selectDepartment(
112           $departmentId);
        } catch (SoapFault $e) {
114       throw $e;
        }
116   }

118   private function throwFailedAuthenticationException() {
        $details = array('message'=>WebServiceConstants::
          NO_VALID_AUTHENTICATION_DATA_SUBMITTED_DETAILS_MESSAGE_VALUE);
120     throw new SoapFault(
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
122         WebServiceConstants::
              NO_VALID_AUTHENTICATION_DATA_SUBMITTED_FAULT_STRING,
            $SERVER['SERVER_NAME'],
124         $details,
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
126   }

128   private function
        throwFailedAuthenticationBecauseInvalidAuthenticationDataException()
        {
        $details = array('message'=>WebServiceConstants::
          INVALID_AUTHENTICATION_DATA_DETAILS_MESSAGE_VALUE);
130     throw new SoapFault(
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_CODE,
132         WebServiceConstants::INVALID_AUTHENTICATION_DATA_FAULT_STRING,
            $_SERVER['SERVER_NAME'],
134         $details,
            WebServiceConstants::FAILED_AUTHENTICATION_FAULT_NAME);
136   }
  }
138
  ?>
```

A.1.4. Programmierschnittstelle als Java-Client eines jadis.net-Webservice

A.1.4.1. Java-Skeleton-Quellcode

```
1  /**
    * JadisNetWebServiceSkeleton.java
3   *
    * This file was auto-generated from WSDL
5   * by the Apache Axis2 version: 1.3  Built on :
    * Aug 10, 2007 (04:45:47 LKT)
7   */
  package com.zweitwerk.jadisnetwebservice;
9
  import com.zweitwerk.jadisNet.JadisNetWebServiceStub;
11
  import java.rmi.RemoteException;
13 import java.util.HashMap;
  import java.util.Map;
15 import java.util.Random;

17 import org.apache.axis2.AxisFault;
  import org.apache.axis2.context.MessageContext;
19
  /**
21  * JadisNetWebServiceSkeleton java skeleton for the axisService
    */
23 public class JadisNetWebServiceSkeleton {

25   public static final String AUTH_EXCEPTION_MESSAGE = "No valid
       authentication data submitted";

27   private static Map<String, ClientIpAddressAndServerAuthData>
       clientIpAddressAndServerAuthDataMap = null;

29   private MessageContext requestMessageContext;

31   public JadisNetWebServiceSkeleton() {
       if (JadisNetWebServiceSkeleton.clientIpAddressAndServerAuthDataMap ==
           null) {
33       JadisNetWebServiceSkeleton.clientIpAddressAndServerAuthDataMap =
           new HashMap<String, ClientIpAddressAndServerAuthData>();
       }
```

```
35      }

37      /**
         * Auto generated method signature
39       *
         * @param createSession
41       */
        public com.zweitwerk.jadisnetwebservice.CreateSessionResponse
            createSession(
43           com.zweitwerk.jadisnetwebservice.CreateSession createSession)
             throws AxisFault, com.zweitwerk.jadisNet.CreateSessionException,
45           com.zweitwerk.jadisnetwebservice.CreateSessionException,
             RemoteException {
47          JadisNetWebServiceStub stub = new JadisNetWebServiceStub();
            JadisNetWebServiceStub.CreateSession serverCreateSession = new
                JadisNetWebServiceStub.CreateSession();
49          serverCreateSession.setUsername(createSession.getUsername());
            serverCreateSession.setPassword(createSession.getPassword());
51          JadisNetWebServiceStub.CreateSessionResponse
                serverCreateSessionResponse = stub
                 .createSession(serverCreateSession);
53          JadisNetWebServiceStub.SessionObject serverSessionObject =
                serverCreateSessionResponse
                 .getCreateSessionResponse();
55          Random randomNumberGenerator = new Random();
            String clientAuthData = Long
57               .toHexString(randomNumberGenerator.nextLong());
            ClientIpAddressAndServerAuthData clientIpAddressAndServerAuthData =
                new ClientIpAddressAndServerAuthData();
59          clientIpAddressAndServerAuthData
                 .setClientIpAddress(requestMessageContext.getProperty(
61                   MessageContext.REMOTE_ADDR).toString());
            clientIpAddressAndServerAuthData
63               .setServerAuthData(serverSessionObject
                     .getAuthData());
65          JadisNetWebServiceSkeleton.clientIpAddressAndServerAuthDataMap
                 .put(clientAuthData,
67                   clientIpAddressAndServerAuthData);
            com.zweitwerk.jadisnetwebservice.SessionObject clientSessionObject =
                new com.zweitwerk.jadisnetwebservice.SessionObject();
69          clientSessionObject.setAuthData(clientAuthData);
            JadisNetWebServiceStub.Department serverDepartmentArray[] =
                serverSessionObject
71               .getDepartment();
```

```
        for (JadisNetWebServiceStub.Department e : serverDepartmentArray) {
73          com.zweitwerk.jadisnetwebservice.Department clientDepartment = new
                com.zweitwerk.jadisnetwebservice.Department();
            clientDepartment.setDepartmentId(e.getDepartmentId());
75          clientDepartment.setName(e.getName());
            clientSessionObject.addDepartment(clientDepartment);
77      }
        com.zweitwerk.jadisnetwebservice.CreateSessionResponse
            clientCreateSessionResponse = new com.zweitwerk.jadisnetwebservice
            .CreateSessionResponse();
79      clientCreateSessionResponse
            .setCreateSessionResponse(clientSessionObject);
81      return clientCreateSessionResponse;
    }
83
    /**
85   * Auto generated method signature
     *
87   * @param selectDepartment
     */
89  public com.zweitwerk.jadisnetwebservice.SelectDepartmentResponse
        selectDepartment(
        com.zweitwerk.jadisnetwebservice.SelectDepartment selectDepartment)
91      throws AxisFault, com.zweitwerk.jadisNet.AuthenticateException,
        com.zweitwerk.jadisnetwebservice.AuthenticateException,
93      com.zweitwerk.jadisNet.SelectDepartmentException,
        com.zweitwerk.jadisnetwebservice.SelectDepartmentException,
95      RemoteException {
      ClientIpAddressAndServerAuthData clientIpAddressAndServerAuthData =
          JadisNetWebServiceSkeleton.clientIpAddressAndServerAuthDataMap
97          .get(requestMessageContext.getEnvelope().getHeader()
                .getFirstElement().getText());
99    if (clientIpAddressAndServerAuthData != null
          && clientIpAddressAndServerAuthData
101             .getClientIpAddress().compareTo(
                    requestMessageContext.getProperty(
103                     MessageContext.REMOTE_ADDR).toString()) == 0) {
        JadisNetWebServiceStub stub = new JadisNetWebServiceStub();
105     JadisNetWebServiceStub.SelectDepartment serverSelectDepartment =
            new JadisNetWebServiceStub.SelectDepartment();
        serverSelectDepartment.setSelectDepartment(selectDepartment
107         .getSelectDepartment());
        JadisNetWebServiceStub.Authenticate serverAuthenticate = new
            JadisNetWebServiceStub.Authenticate();
```

```
109         serverAuthenticate
                .setAuthenticate(clientIpAddressAndServerAuthData
111                 .getServerAuthData());
            JadisNetWebServiceStub.SelectDepartmentResponse
                serverSelectDepartmentResponse = stub
113             .selectDepartment(serverSelectDepartment,
                    serverAuthenticate);
115         boolean hasDepartmentBeenSelected = serverSelectDepartmentResponse
                .getSelectDepartmentResponse();
117         com.zweitwerk.jadisnetwebservice.SelectDepartmentResponse
                clientSelectDepartmentResponse = new com.zweitwerk.
                jadisnetwebservice.SelectDepartmentResponse();
            clientSelectDepartmentResponse
119             .setSelectDepartmentResponse(hasDepartmentBeenSelected);
            return clientSelectDepartmentResponse;
121       } else {
            com.zweitwerk.jadisnetwebservice.AuthenticateExceptionElement
                authenticateExceptionElement = new com.zweitwerk.
                jadisnetwebservice.AuthenticateExceptionElement();
123         authenticateExceptionElement
                .setMessage(JadisNetWebServiceSkeleton.AUTH_EXCEPTION_MESSAGE);
125         com.zweitwerk.jadisnetwebservice.AuthenticateException
                authenticateException = new com.zweitwerk.jadisnetwebservice.
                AuthenticateException();
            authenticateException.setFaultMessage(authenticateExceptionElement)
                ;
127         throw authenticateException;
          }
129     }

131     public void init(MessageContext requestMessageContext) {
          this.requestMessageContext = requestMessageContext;
133     }

135     class ClientIpAddressAndServerAuthData {

137       private String clientIpAddress;

139       private String serverAuthData;

141       private ClientIpAddressAndServerAuthData() {
            clientIpAddress = "";
143         serverAuthData = "";
          }
```

```
145
    private String getClientIpAddress() {
147   return clientIpAddress;
    }
149
    private void setClientIpAddress(String clientIpAddress) {
151   this.clientIpAddress = clientIpAddress;
    }
153
    private String getServerAuthData() {
155   return serverAuthData;
    }
157
    private void setServerAuthData(String serverAuthData) {
159   this.serverAuthData = serverAuthData;
    }
161 }
}
```

A.1.5. Client der Test-Webservices

```
<?php
2
define('ISO_CHARACTER_SET', 'iso-8859-1');
4
define(
6       'URL_OF_WSDL_DOCUMENT',
        'http://<path to WSDL document>/TestWebService.wsdl');
8
class CreateSessionParameters {
10    private $username;
      private $password;
12    public function __construct($initialUsername, $initialPassword) {
        $this->username = $initialUsername;
14      $this->password = $initialPassword;
      }
16  }

18  $soapClient = new SoapClient(
        URL_OF_WSDL_DOCUMENT,
20      array('soap_version'=>SOAP_1_2, 'encoding'=>ISO_CHARACTER_SET));
    $params = new CreateSessionParameters(
22      'John_Coltrane',
        'A_Love_Supreme');
```

```
24 echo '<html>';
   echo '<body>';
26 try {
       $createSessionResponse = $soapClient->createSession($params);
28     echo 'Authentifizierungsdatum: ';
       echo $createSessionResponse->authenticationData;
30     echo '<br>';
       echo '<br>';
32     $departmentId = 0;
       if (is_array($createSessionResponse->department)) {
34         $department = $createSessionResponse->department[0];
           $departmentId = $department->departmentId;
36         $i = 0;
           foreach ($createSessionResponse->department as $key=>$value) {
38             echo 'Ressort ';
               echo $value->name;
40             echo '<br>';
               echo 'Ressort-ID: ';
42             echo $value->departmentId;
               if ($i++ < count($createSessionResponse->department) - 1) {
44                 echo '<br>';
                   echo '<br>';
46             }
           }
48     } else {
           $department = $createSessionResponse->department;
50         $departmentId = $department->departmentId;
           echo 'Ressort ';
52         echo $department->name;
           echo '<br>';
54         echo 'Ressort-ID: ';
           echo $department->departmentId;
56     }
       echo '<br>';
58     echo '<br>';

60     // the namespace must be set to "jadisnetwebservice"
       // instead of "jadisNet" for the Java Test web service
62     $header = new SoapHeader(
                   'urn:jadisNet.zweitwerk.com',
64                 'authenticate',
                   $createSessionResponse->authenticationData);
66     $soapClient->__setSoapHeaders(array($header));
```

```
    $selectDepartmentResponse = $soapClient->selectDepartment(
        $departmentId);
68  echo 'Ressort ausgewaehlt: ';
    if ($selectDepartmentResponse) {
70      echo 'true';
    } else {
72      echo 'false';
    }
74 } catch (SoapFault $e) {
    echo $e->getMessage();
76 }
echo '</body>';
78 echo '</html>';

80 ?>
```

Literaturverzeichnis

AN05 AYALA, Dietrich ; NICHOL, Scott: *NuSOAP – SOAP Toolkit for PHP*. Version: Juli 2005. http://sourceforge.net/projects/nusoap/, Abruf: 22. Februar 2008

Apa07a APACHE SOFTWARE FOUNDATION, The: *Apache Tomcat*. Version: März 2007. http://tomcat.apache.org/index.html, Abruf: 22. Februar 2008

Apa07b APACHE SOFTWARE FOUNDATION, The: *Axis2*. Version: Juli 2007. http://ws.apache.org/axis2/index.html, Abruf: 22. Februar 2008

Apa08 APACHE SOFTWARE FOUNDATION, The: *Apache HTTP Server Project*. Version: Januar 2008. http://httpd.apache.org/, Abruf: 22. Februar 2008

Bal98 BALZERT, Helmut: *Lehrbuch der Software-Technik*. Bd. 2: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg u. Berlin : Spektrum Akademischer Verlag, 1998

BKD+07 BÖEKEMEIER, Jost ; KOERBER, Jon ; DENGLER, Martin T. ; JAYAWARDANA, Nandika ; LONDENBERG, Kai ; MACHADO, André F. ; RUBY, Sam ; SAMARANAYAKE, Sanka: *php/Java bridge*. Version: Januar 2007. http://php-java-bridge.sourceforge.net/pjb/index.php, Abruf: 22. Februar 2008

But03 BUTEK, Russell: *Which Style of WSDL should I use ?* Version: Oktober 2003. http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/, Abruf: 22. Februar 2008. – geändert im Mai 2005

FTW07 FROTSCHER, Thilo ; TEUFEL, Marc ; WANG, Dapeng: *Java Web Services mit Apache Axis2*. [Frankfurt am Main] : entwickler.press, 2007

GDS+04 GRAHAM, Steve ; DAVIS, Doug ; SIMEONOV, Simeon ; DANIELS, Glen ; BRITTENHAM, Peter ; NAKAMURA, Yuichi ; FREMANTLE, Paul ; KÖNIG, Dieter ; ZENTNER, Claudia: *Building Web Services with Java. Making Sense of XML, SOAP, WSDL, and UDDI*. 2. Aufl. Indianapolis, IN : Sams Publishing, 2004. – Nachdruck v. 2005

JUn02 JUNIT.ORG: *JUnit*. Version: September 2002. http://www.junit.org/home

JZ02 JECKLE, Mario ; ZENGLER, Barbara: *SOAP – aber sicher*. Version: Januar/Februar 2002. http://www.jeckle.de/secureSOAP.html, Abruf: 22. Februar 2008. – Erschienen in der Ausgabe 1 (Januar/Februar 2002) der Zeitschrift OBJEKTSpektrum

PHP08 PHP GROUP, The: *PHP*. Version: Januar 2008. http://www.php.net/, Abruf: 22. Februar 2008

Ric06 RICHARDS, Robert: *Pro PHP XML and Web Services*. Berkeley, CA : Apress, 2006

Sun08 SUN MICROSYSTEMS: *Java.sun.com*. Version: 2008. http://java.sun.com/, Abruf: 22. Februar 2008

Tan03 TANENBAUM, Andrew S.: *Computer Networks*. Upper Saddle River, NJ : Prentice Hall / PTR, 2003

W3C01 W3C (WORLD WIDE WEB CONSORTIUM): *Web Services Description Language (WSDL) 1.1*. Version: März 2001. http://www.w3c.org/TR/wsdl, Abruf: 22. Februar 2008. Verfasst v. Erik Christensen u. a.

W3C02a W3C (WORLD WIDE WEB CONSORTIUM): *XML Encryption Syntax and Processing*. Version: Dezember 2002. http://www.w3c.org/TR/xmlenc-core/, Abruf: 22. Februar 2008. Verfasst v. Donald Eastlake und Joseph Reagle

W3C02b W3C (WORLD WIDE WEB CONSORTIUM): *XML-Signature Syntax and Processing*. Version: Februar 2002. http://www.w3c.org/TR/xmldsig-core/, Abruf: 22. Februar 2008. Verfasst v. Donald Eastlake, Joseph Reagle und David Solo

W3C04a W3C (WORLD WIDE WEB CONSORTIUM): *Web Services Glossary*. Version: Februar 2004. http://www.w3c.org/TR/ws-gloss, Abruf: 22. Februar 2008. Verfasst v. Hugo Haas u. Allen Brown (bis Juni 2002

W3C04b W3C (WORLD WIDE WEB CONSORTIUM): *XML Schema Part 1: Structures Second Edition*. Version: Oktober 2004. http://www.w3.org/TR/xmlschema-1/, Abruf: 22. Februar 2008. Verfasst v. Henry S. Thompson u. a.

WCL+05 WEERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Upper Saddle River, NJ u. a.: Prentice Hall / PTR, 2005

WS-06 WS-I (WEB SERVICES INTEROPERABILITY ORGANIZATION): *WS-I Basic Profile Version 1.1*. Version: April 2006. http://www.ws-i.org/Profiles/BasicProfile-1.1.html, Abruf: 22. Februar 2008. Hg. v. Keith Ballinger u. a.

Glossar

GNU: *GNU is not UNIX* ⟨engl.⟩ (Projekt, das mit dem Ziel gegründet wurde, ein vollständig freies Betriebssystem zu entwickeln)

HTTP: *Hypertext Transfer Protocol* ⟨engl.⟩ (Protokoll der Anwendungsschicht zur Datenübermittlung in Intranets und im *World Wide Web*)

IP: *Internet Protocol* ⟨engl.⟩ (Netzwerkprotokoll der Vermittlungsschicht)

MAMS: *media asset management system* ⟨engl.⟩ (Verwaltungssoftware für Mediendokumente)

PHP: *PHP: Hypertext Preprocessor* ⟨engl.⟩ (Programmiersprache, die hauptsächlich zur Erstellung von Skripten für dynamische Webseiten verwendet wird)

RPC: remote procedure call ⟨engl.⟩ (Funktionsaufruf bei entfernten Rechnern über ein Netzwerk)

SOAP: früher *Simple Object Access Protocol*, seit Version 1.2 keine spezielle Bedeutung ⟨engl.⟩ (Protokoll für XML-basierte Nachrichten bei Webservices; baut zumeist auf HTTP auf)

SQL: *Structured Query Language* ⟨engl.⟩ (deklarative Sprache für die Abfrage und Manipulation von Daten relationaler Datenbanken)

SSL: *Secure Sockets Layer* ⟨engl.⟩ (ein hybrides Verschlüsselungsprotokoll für Datenübertragungen im Internet)

URI: *Uniform Resource Identifier* ⟨engl.⟩ (eine Zeichenfolge zur Identifizierung einer physischen oder abstrakten Ressource)

W3C: *World Wide Web Consortium* ⟨engl.⟩ (Gremium zur Standardisierung der das World Wide Web bestimmenden Techniken; es gibt Empfehlungen ab, bei deren Entwicklung es sich verpflichtet hat, ausschließlich Technologien zu verwenden, die frei von Patentgebühren sind)

WS-I: *Web Services Interoperability Organization* ⟨engl.⟩ (ein Zusammenschluss von Unternehmen der Informationstechnologiebranche, der sich für die Interoperabilität von Webservice-Spezifikationen einsetzt)

WSDL: *Web Services Description Language* ⟨engl.⟩ (eine XML-basierte Sprache zur Beschreibung von Webservices)

XML: *Extensible Markup Language* ⟨engl.⟩ (eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten)

Index