Planung und Entwicklung einer Software zur automatisierten Schnittstellenspezifikation eines Servers mit Web Service Schnittstellen.

Dokumentation der betrieblichen Projektarbeit

Für die Abschlussprüfung zum Fachinformatiker - Anwendungsentwicklung

Prüfling
Fabian Fuchs
Fukuistraße 26
21423 Winsen

Ausbildungsbetrieb
BITMARCK Technik GmbH
Hammerbrookstraße 38
20097 Hamburg

Inhalt

Tabellenverzeichnis

Abbildungsverzeichnis

1. Projektbeschreibung

1.1 Einleitung

Dies ist die Dokumentation der betrieblichen Projektarbeit, die im Rahmen meiner Ausbildung zum Fachinformatiker in der Fachrichtung Anwendungsentwicklung bei der BITMARCK Technik GmbH (BMT) entstanden ist. Unterstrichene Begriffe werden im Glossar erklärt (Anhang A.1). Zudem habe ich auf externe Quellen zurückgegriffen, die im Literaturverzeichnis aufgeführt sind (siehe Anhang A.3).

Für das Erstellen dieser Dokumentation habe ich Microsoft Word 2013 verwendet.

Die Präsentation wird in Microsoft PowerPoint 2013 erstellt.

Die BITMARCK Technik GmbH ist eine der vier Tochtergesellschaften der BITMARCK Holding GmbH. Sie ist aus dem ausgegliederten IT-Bereich der DAK-Gesundheit (Deutsche Angestellten Krankenkasse) entstanden und stellt die technische Infrastruktur für umfangreiche IT-Leistungen ihrer Kunden der gesetzlichen Krankenversicherung zur Verfügung. Die vier weiteren Gesellschaften sind die BITMARCK Software GmbH, BITMARCK Beratung GmbH, BITMARCK Service GmbH und die BITMARCK Vertriebs- und Projekt GmbH. Die BITMARCK ist ein Full-Service IT-Dienstleister für gesetzliche Krankenversicherungen, entwickelt Softwareprodukte und bietet individuelle Beratungs- und Serviceleistungen an. Der größte Partner ist die DAK-Gesundheit, für welche die ganzen IT-Prozesse gesteuert werden.

In der Abteilung Zusatzprodukte bin ich einer von vielen Entwickler für das Projekt 21c|ng. 21c|ng ist eine neue Software für Krankenkassen um die Verwaltung der Versicherten zu vereinfachen. Aufgrund einiger Fusionen im Krankenkassenbereich, kommen häufig neue Versicherte aus Fremdsystemen hinzu und mit 21c|ng wird das Integrieren einfacher.

Innerhalb dieses Projektes arbeiten wir mit einem agilen Entwicklungskonzept. Bei diesem Entwicklungskonzept handelt es sich um Scrum, mit dessen Hilfe meine Kollegen und ich die anfallende Arbeit anhand der Kenntnisse planen und verteilen können. Ich bin mit einigen anderen Kollegen als *ESB* Entwickler für das Projekt tätig. Zu diesem Bereich zählt der Integration Server, hierbei handelt es sich um ein Produkt der SoftwareAg. Auf diesem Server kümmere ich mich um die Datenmappings für die *Requests* und *Responses* in Verbindung zum *ServicePoint*.

1.2 Projektbeschreibung

Der ServicePoint wird als eine Schnittstelle zu *Datenbanken*, als auch zum Großrechner genutzt und stellt Web-Services bereit, die von vielen anderen Programmen aufgerufen werden. Zu jeder dieser Schnittstellen gibt es zwei Dateien, eine *WSDL*- und eine *XSD*- Datei. Beides sind Dateien im *XML*- Format. Die WSDL- Datei enthält eine Auflistung der Operationen und deren Datentypen. In der XSD- Datei sind die Datentypen wiederum genauer beschrieben. Unter den ServicePointEntwicklern gibt es eine Einigung, dass in den XSD- Dateien eine Dokumentation zu den Datentypen gepflegt wird. Diese Einigung wurde in einer Definition of

Done, kurz *DoD*, festgehalten. Somit ist jeder Entwickler, dessen Zuständigkeitsbereich der die Entwicklung des ServicePoint ist, verpflichtet die Schritte innerhalb der DoD einzuhalten, wodurch eine bessere Codequalität erreicht wird. Ein weiterer Schritt in der DoD ist die Entwicklung von Testfällen für den produzierten Code.

Mein Projekt soll anhand beider Dateien, also der WSDL- und der referenzierten XSD- Datei, eine Doku in Form einer *HTML*- Datei erzeugen, die als Schnittstellenspezifikation dient. Es sollen also alle Operationen mit ihren Übergabeparametern und Rückgabewerten, sowie deren Eigenschaften aufgelistet werden. Das Programm soll mithilfe von *ANT* in einen automatischen Build- und Deploymentprozess integriert werden (siehe Abbildung 1).

Nach jedem erfolgreichen Durchlauf der ServicePoint-Tests soll dieses Programm aufgerufen werden, welches zu jeder WSDL- Datei eine eigene HTML- Datei erzeugt. Allerdings soll es auch möglich sein, dass bereits während der Entwicklung einer neuen Schnittstelle eine Dokumentation vorab erzeugt wird, um die Zeit zwischen Entwicklung und Einsatz möglichst gering zu halten. Daher soll es den Entwicklern möglich sein das Programm händisch zu starten.

1.3 Projektbegründung

Im Laufe der Zeit kommen immer mehr Schnittstellen im ServicePoint hinzu und für jede Ergänzung/Änderung an einer dieser Schnittstellen erfolgt eine Anpassung der Dokumentation. Da allerdings schon in den WSDL- und XSD- Dateien alle relevanten Informationen, wie benutzte Datentypen, Variablennamen und Kommentare vorhanden sind, handelt es sich bei dem händischen Dokumentieren um doppelte Arbeit. Diese Arbeit wiederum wird gerne vergessen und dadurch eine liegt dann eine fehlerhafte Dokumentation vor. Daher können die Arbeiten für diesen Schritt aus meiner Sicht und der des Fachbereichs, also den ServicePointEntwicklern, automatisiert werden. Somit müssen die Entwickler nach dem Abschluss dieses Projektes nur noch ihre Dokumentation innerhalb des Java-Codes und der XSD-Datei pflegen. Anhand ihrer Dokumentationen innerhalb der Dateien wird im Nachgang dann eine Schnittstellenspezifikation im Laufe des Build- und Deploymentprozesses erzeugt.

1.4 Projektziel

Ziel dieses Projekts ist es, die Dokumentationsarbeit der ServicePointEntwickler zu verringern und die Qualität der Dokumentation zu verbessern, indem für eine einheitliche Dokumentation gesorgt wird. Somit kann die Dokumentationszeit zu verringert werden.

1.5 Zielgruppe

Die Zielgruppe der Anwendung sind vorwiegend die ServicePointEntwickler der BMT, die in den unterschiedlichsten Abteilungen arbeiten und für die unterschiedlichsten Projekte tätig sind. Es profitieren allerdings auch die Entwickler der *Umsysteme* von diesem Programm, da die Dokumentationen der Schnittstellen des ServicePoints nun einheitlicher dargestellt werden und auch schon vor der Fertigstellung einer Schnittstelle die erste Schnittstellenspezifikation herausgegeben werden kann.

2. Analyse

2.1 Ist-Zustand

Bei der Entwicklung am ServicePoint gibt es für jede Schnittstelle eine WSDL-Datei, in der mehrere Operationen beschrieben sind. Jede der Operationen hat natürlich Übergabeparameter und einen Rückgabewert, die in der zugehörigen XSD-Datei beschrieben sind. Innerhalb einer WSDL können mehrere XSDs referenziert werden und jede XSD kann gleichzeitig auch andere XSDs referenzieren und importieren. Dadurch entsteht ein sehr komplexes Datenmodell, durch das ein Entwickler durchnavigieren muss um an die benötigten und relevanten Informationen zu kommen. Diese muss dann wiederum von Hand in ein bestehendes Dokument eintragen werden bzw. es muss ein neues Dokument anhand des festgelegten Schemas erstellt werden.

2.2 Soll-Konzept

In dem automatischen Build- und Deploymentprozess des ServicePoint soll ein weiterer Schritt ergänzt werden. In diesem Schritt soll anhand der vorhandenen WSDL- und XSD- Dateien eine Dokumentation erzeugt werden, die auf dem Server abgelegt wird und so anderen Entwicklern zur Verfügung steht.

Außerdem soll das Programm händisch mit dem lokalen Entwicklungsstand zu starten sein, um bereits während der Entwicklung eine Dokumentation an andere weitergeben zu können, da zwischen Entwicklung und Einbindung häufig nur ein kurzer Zeitraum liegt und eine vorläufige Dokumentation schon eine einfachere Entwicklung der Aufrufe erlaubt.

2.3 Kosten-Nutzen-Analyse

Die Kosten-Nutzen-Analyse ist eine betriebswirtschaftliche Analysemethode, die zur Gegenüberstellung von Kosten und Nutzen verwendet wird. Hierbei werden alle für das Projekt benötigte Kosten, wie *Personentage* (PT) und eventuelle Softwareanschaffungen, summiert und dem Nutzen, wie zum Beispiel Zeitersparnis, gegenübergestellt.

2.3.1 Kosten

Die Kosten, die in der BMT anfallen, werden in Personentagen (PT) angegeben. Ein PT entspricht hierbei einem Aufwand von 7,8 Stunden. Da es sich bei dem Projekt um ein internes Projekt handelt, wird der interne Verrechnungssatz verwendet. Dieser beträgt laut dem Bereich Kundenservice 520€ je PT. Somit berechnet sich der Stundensatz wie folgt:

$$\frac{520{,}00€/PT}{7{,}8\ Stunden/PT} = \frac{520{,}00€/\cancel{PT}}{7{,}8\ Stunden/\cancel{PT}} \cong 66{,}67\frac{€}{Stunde}$$

Die Projektzeit beträgt 70 Stunden. Zusätzlich zu meinem Aufwand werden noch weitere vier Arbeitsstunden in der BMT für den internen Test, die Freigabe und die Integration benötigt. Außerdem wurden im Laufe der Fachgespräche Kollegen benötigt, sodass hier nochmal für die zwei Kollegen jeweils zwei Stunden anfielen. Insgesamt ergeben sich dadurch folgende Kosten:

$$66{,}67\,\frac{€}{Stunde} * (70\ Stunden + 4\ Stunden + 2 * 2\ Stunden) = 5.200{,}00€$$

2.3.2 Nutzen

Der Nutzen meiner Entwicklung besteht aus einer Zeitersparnis der Entwickler. Für jede Stunde Entwicklung kommt eine halbe Stunde Dokumentation hinzu. Davon gehen knapp fünf Prozent in das Schreiben der externen Dokumentation ab, welches durch mein Projekt entfällt.

$$0{,}5\,\frac{h * Dokumentation}{h * Entwicklung} * \frac{5}{100} = 0{,}025\frac{h * Dokumentation}{h * Entwicklung}\ Ersparnis$$

Eine Analyse ergab, dass jeder Entwickler pro Tag knapp vier Stunden mit der Entwicklung für den ServicePoint verbringt. Somit ergibt sich bei den sechs ServicePointEntwicklern folgende Entwicklungszeit:

$$6\ Entwickler * 4\ \frac{h * Entwicklung}{Arbeitstag * Entwickler} = 24\ \frac{h * Entwicklung}{Arbeitstag}$$

Um die Zeitersparnis pro Tag zu berechnen, muss die täglich anfallende Entwicklungszeit aller Entwickler mit der Ersparnis pro Entwicklungsstunde multipliziert werden. Es ergibt sich folgende Rechnung:

$$24 * \frac{h * Entwicklung}{Arbeitstag} * 0{,}025\frac{h * Dokumentation\ Ersparnis}{h * Entwicklung} = 0{,}6h\ \frac{Ersparnis}{Arbeitstag}$$

Der jährliche Nutzen in Stunden pro Jahr ergibt sich durch die Multiplikation der täglichen Ersparnis mit der durchschnittlichen Anzahl von Arbeitstagen pro Monat und mit der Anzahl der Monate im Jahr.

$$0{,}6h\ \frac{Ersparnis}{Arbeitstag} * 20\ \frac{Arbeitstage}{Monat} * 12\ \frac{Monate}{Jahr} = 144\,h\ \frac{Ersparnis}{Jahr}$$

Da es sich hierbei jeweils um interne Mitarbeiter handelt, wird auch hier wieder der interne Verrechnungssatz genutzt. Der monetäre Nutzen entspricht der Multiplikation von dem internen Verrechnungssatz pro Stunde mit dem jährlichen Ersparnis. Somit ergibt sich folgende Rechnung:

$$144\ \frac{h}{Jahr} * 66{,}67\ \frac{€}{h} \cong 9.600{,}00\frac{€}{Jahr}$$

Es gibt auch noch einen weiteren Nutzen, der sich jedoch nicht monetär festhalten lässt bzw. nur schwer abbilden lässt, da er unregelmäßig auftreten kann. Hierbei handelt es sich einerseits um die Verbesserung der Aktualität der Dokumentation, andererseits wird auch die Qualität der Dokumentation steigen, da sich die Dokumentationsstile der Entwickler unterscheiden. Somit gab es bis jetzt Unterschiede in den einzelnen Schnittstellenspezifikationen, welche mit diesem Programm entfallen.

2.3.3 Gegenüberstellung von Kosten und Nutzen

Die folgende Tabelle zeigt eine Gegenüberstellung der berechneten Kosten und dem zu erwartenden monetären Nutzen. Da als Zeitpunkt der Produktivnahme meiner Entwicklung bisher nur festgelegt ist, dass dies im Januar 2016 erfolgen soll, kann hier kein genaueres Datum angegeben werden.

Für die nachstehende Veranschaulichung bin ich daher vom 01.01.2016 ausgegangen.

Die Kosten in der Veranschaulichung entsprechen den einmaligen Kosten für die Entwicklung im Jahre 2015 und den veranschlagten Kosten für die Inbetriebnahme der Erweiterung. Folgekosten sind bei diesem Projekt nicht zu erwarten, können aber immer noch in Form von Wartung oder Erweiterungen entstehen.

	2015	2016	2017
Kosten	5.200,00€	0,00€	0,00€
Nutzen	0,00€	9.600,00€	9.600,00€
Bilanz	**-5.200,00€**	**4.400,00€**	**14.000,00€**

Tabelle 1: Gegenüberstellung von Kosten und Nutzen

2.3.4 Amortisation

Die Amortisation beschreibt den Zeitraum, der benötigt wird, um mit der Einsparung der realisierten Anforderung die Summe aller Kosten zu decken.

$$\frac{Kosten}{Nutzen\ im\ Jahr} = Amortisation$$

$$\frac{5.200{,}00€}{9.600{,}00\ €/Jahr} \cong 0{,}54\ Jahre$$

Die Amortisationszeit bei gleichbleibendem Personal und Aufwand beträgt also knapp 0,54 Jahre. Somit dauert es ca. sieben Monate bis die Kosten des Projekts gedeckt sind.

3. Designphase

Bei dem Design dieses Projektes ist es erforderlich, auf das gewünschte Zielformat zu achten. Da es gewünscht war, dass die generierte Dokumentation der eines Javadoc-Generators entspricht, musste ein passendes Konzept dafür entwickelt werden, die Daten in einem passenden Format an einem Generator zu übergeben bzw. selbst zu erzeugen. Das Einlesen der Dateien wollte ich so einfach wie möglich gestalten, da eine geringere Komplexität der Wartbarkeit zugutekommt.

3.1 Entscheidungsprozess XML-Parser

Die Wahl für den XML-Parser fiel relativ schnell auf *JAXB*, weil es folgende Vorteile hat:

- Verfügbar ab Java Version 1.6
- Direktes Umwandeln der einzelnen *XML-Knoten* in Java Objekte möglich

3.2 Planung der Prozesse

Zusätzlich bestand ein Großteil der Arbeit in dieser Phase daraus, die Abläufe der Anwendung und die Integration in den bestehenden Build- und Deploymentprozess zu planen. So wurde in dieser Phase festgelegt, dass das Projekt mithilfe von ANT-Skripten in den bestehenden Build- und Deploymentprozess zu integrieren ist. Die Wahl fiel hier auf ANT-Skripte, da diese bereits in dem Build- und Deploymentprozess verwendet werden. Die ersten Entwürfe für den Ablauf des Prozesses wurden von mir händisch gezeichnet, danach aber mithilfe von dem Programm „yEd Graph Editor“ elektronisch abgebildet und abgespeichert (Abbildung 1). Der gesamte Programmablauf ist nach dem EVA-Prinzip - also Eingabe, Verarbeitung und Ausgabe - gegliedert.

Der ganze Prozess wird für jede einzelne WSDL- durchgeführt. Die Eingabe erfolgt durch das Einlesen der einzelnen WSDL- Datei und der zugehörigen XSD- Dateien. Beim Einlesen kann mithilfe von JAXB anhand der generierten Java-Klassen der Dateiinhalt in Java-Objekte umgewandelt. Diese Objekte wiederum werden in die Verarbeitung, die in diesem Programm aus zwei Schritten besteht, übergeben. Dort werden aus den Objekten die relevanten Informationen extrahiert und in Objekte von den selbst geschriebenen Klassen abgespeichert. Danach werden die ganzen Relationen zwischen den Dateien aufgelöst, sodass die Attribute in dem Objekt bis in ihre atomaren Größen aufgebrochen sind. Danach werden aus dem Objekt in einem letzten Schritt die Informationen aufbereitet und in dem benötigten Format an die Ausgabe übergeben. Dort werden die Informationen dann in einem handelsüblichen HTML- Format in einer Datei abgespeichert, die die einzelne WSDL- repräsentiert.

4. Realisierungsphase

4.1 Entwicklungsumgebung

Als Entwicklungsumgebung benutze ich die *IDE* *Eclipse* in der Version 3.8.4 (Juno). Innerhalb der IDE habe ich Zugriff auf diverse Plug-Ins, mit denen ich meine Code Qualität erhöhen kann.

Die Plug-Ins „PMD“ und „Find Bugs“ fallen in diese Kategorie. Beide sind in der Lage meinen Code zu analysieren und sie markieren Stellen, an denen Bugs auftreten können bzw. wo die Code Qualität lückenhaft ist. Dazu nutzt PMD diverse Regeln, die ich für mein Projekt individuell anpassen konnte. Zudem nutze ich für die Versionierung meines Projektes ein *File Repository*, welches ich mithilfe von *Tortoise SVN* erstellt habe.

Es ist mir dank des Plug-Ins „Subclipse“ direkt möglich ein *SVN-Commit* aus meiner IDE durchzuführen. Um die Änderungen in meiner Working Copy mit dem Stand im *Trunk* zu vergleichen, nutze ich das Tool *WinMerge*.

4.2 Vorbereitung

Bevor ich mit der Entwicklung beginnen konnte, musste ich erstmal ein SVN-Check Out auf das Repository für den ServicePoint machen, da hier die ganzen WSDL- und XSD- Dateien zu finden sind. Außerdem habe ich eine eigene WSDL- und dazugehörige XSD- Dateien anhand der vorhandenen Dateien erstellt (Resultat Abbildung 8). Damit bekam ich ein besseres Verständnis der ganzen Datenstruktur. Weiterhin erlaubte es mir, in die Testfälle

Dateien einzubeziehen, die unabhängig vom kompletten Entwicklungsprozess des ServicePoint sind. Somit müssen die Testfälle für diese Anwendung bei Änderungen am ServicePoint nicht erneut angepasst werden.

Zudem wollte ich eine saubere Trennung zwischen Produktivcode und Testcode gewährleisten, weswegen ich mich für zwei Projekte, SpDokumentierer und TestSpDokumentierer in Eclipse entschieden habe. In das Projekt TestSpDokumentierer wurde eine Abhängigkeit (Dependency) auf das Projekt SpDokumentierer hinzugefügt, um einen Zugriff auf die Klassen und deren Methoden zu erlauben. Darüber hinaus musste auch in den Build Path für das Testprojekt eine JUnit Library hinzugefügt werden um die ganze Testautomatisierung und umsetzung zu ermöglichen. In meinem Fall hat auch jede Testklasse das gleiche Package wie die zugehörige Klasse, um das Zugreifen auf Methoden zu ermöglichen, die als package visible deklariert sind.

4.3 Generieren der Java Objekte

Anhand der Basis-XSD- Dateien, die den Aufbau von WSDL- und XSD- Dateien beschreiben, konnte ich mithilfe von JAXB die grundlegenden Java-Klassen generieren. Es gibt mehrere Wege die Java Klassen zu erzeugen, hierfür muss lediglich mindestens Java 7 installiert sein. Der erste Weg ist mithilfe von Eclipse, so kann man im Package Explorer einen Rechtsklick auf die gewünschte Datei im XML-Format machen und dann „Generate JAXBClasses“ auswählen (Abbildung 2). Dadurch öffnet sich ein Fenster, in dem der Zielordner und das Zielpaket angegeben werden kann.

Der aus meiner Sicht einfachere Weg geschieht jedoch über die Kommandozeile (Abbildung 3). Man kann mit dem Befehlt „xjc“ aus XSD- bzw. WSDL- Dateien Java-Klassen erzeugen. Die installierte Java Version muss Java 1.7 oder höher sein. Der Befehl xjc benötigt noch weitere Parameter, hierfür werden die Ergänzungen -d für den Zielordner (meistens src) und -p für die package Deklaration benötigt. Die package Deklaration definiert das Unterverzeichnis, in dem die Klassen angelegt werden sollen. Dadurch kann der generierte Code von selbst entwickelten Code getrennt werden, indem für den Parameter -p „de.bmt.spdokumentierer.generated“ und „.xsd“ bzw. „.wsdl“ genutzt wird und dadurch eine saubere Trennung zwischen generierten Code für die Beschreibung einer WSDL- oder einer XSD- und meinem selbst erstellten Code entsteht.

Jede generierte Klasse wird mit den passenden Javadoc Kommentaren, XML-Annotationen und Methoden zum Zugriff auf die Daten innerhalb der XML-Dateien generiert. Die Javadoc-Kommentare enthalten hierbei Beispiele, was die Klasse in einer XSD- bzw. WSDL- Datei repräsentieren kann. Die XML-Annotationen ermöglichen ein Unmarshalling, also ein Aufbrechen einer XML- Datei in Java Objekte oder ein Marshalling ein Schreiben von Java Objekte in eine XML- Datei.

4.4 Testentwicklung

Da ich mich für Test Driven Developement (*TDD*) als Vorgehen in der Entwicklung entschieden habe, ist die Entwicklung der Testfälle der Kern meiner Entwicklungsarbeit. Grundsätzlich versuche ich für jede öffentliche Methode mindestens einen Testfall zu entwickeln. Der Testfall wird zuerst definiert und entwickelt und erst dann kann ich mit der Implementierung anfangen. Während der gesamten Entwicklung versuchte ich die Faustregel von Uncle Bob, bekannt unter dem Namen Robert Cecil Martin, einzuhalten. Diese besagt: „30 Sekunden

Testentwicklung - max. zehn Minuten Implementierung“, also kleine Testfälle zu schreiben aber dafür dann viele. Nachdem der Test erfolgreich ist, soll ein Refactoring der Implementierung des Testcodes und des Testfalls erfolgen. Somit entsteht ein iteratives Vorgehen für jeden Testfall, jede Klasse, jede Komponente. Für jeden Testfall werden die vier Schritte, das Erstellen des Testfalls, die Implementierung des Codes, das Refactoring des Codes und das Refactoring des Testfalls immer wieder wiederholt. Das erfolgt für jedes Modul, jede Komponente. Hierbei ist jede Ebene außer den Unit Tests nur ein Zusammenspiel der untergeordneten Ebene.

Dadurch entsteht abgesehen von der iterativen Entwicklung für jeden Testfall ein iteratives Vorgehen, in dem man sich von den kleinen Prozessen zu den größeren durcharbeitet.

4.5 Datenbeschaffung

Die Datenbeschaffung erfolgt bei mir über eine Utility-Klasse, in der ich die ganze Verwaltung der Verbindung des von Java bereitgestellten FileReaders zu der Datei kontrolliere. Hierzu besitzt sie mehrere öffentliche Methoden (siehe Outline-Auszug in Abbildung 4). Es gibt zwei Methoden mit deren Hilfe man den Dateiinhalt in das gewünschte Format mithilfe von JAXB umwandelt. Einerseits ist es die Methode readXsdFile, welche ein Java-Objekt vom Typ Schema zurückgibt, die andere Methode ist readWSDLFile, welche ein Java-Objekt vom Typ TDefinitions zurückgibt (Quellcodeauszug siehe Abbildung 5 im Anhang). Beide Objekte werden anhand von automatisch generierten Klassen instanziiert.

Da beide Methoden direkt auf das Dateisystem zugreifen, kann es hier leicht zu Fehlern bezüglich der Dateien kommen. Dazu zähle ich FileNotFoundException und IOException. Die FileNotFoundException beschreibt das Fehlen einer Datei, während eine IOException Fehler innerhalb des In- und Outputs beschreibt, wie das Abreißen eines Input Streams. Beide Fehler werden zwar von JAXB intern in einem Catch-Block abgefangen, jedoch werden die Fehler durchgereicht und somit muss mein Fehlerhandling in der Lage sein, die übergebenen Fehler abzufangen und nach Fehlerfall entsprechend zu reagieren.

4.6 Datenaufbereitung

Die Datenaufbereitung ist das Herzstück dieser Anwendung, da ein Großteil der Informationen schon direkt in dem generierten Java Objekt vorliegt, allerdings dort eine starke Verschachtelung vorherrscht, sodass erst eine Aufbereitung erfolgen muss. Zwischen WSDL- und XSD- Dateien gibt es wenige bis keine strukturellen Gemeinsamkeiten, weswegen für jedes Dateiformat eine eigene Klasse zum Formattieren und Aufbereiten genutzt wird.

Um den Zugriff auf die relevanten Informationen zu vereinfachen, werden die Daten aus den Informationen der generierten Objekte ausgelesen und in ein selbst geschriebenes Objekt abgespeichert. In den selbst generierten Objekten überschreibe ich auch die equals- und die hashCode-Methode um einen einfachen Vergleich auf die von mir gewünschten Attribute (meistens der Target Namespace, ein eindeutiger Namensbereich) zu ermöglichen.

In einer Schleife wird über alle WSDL- Dateien iteriert, die sich im Ordner befinden. Für jede WSDL-Datei werden zuerst die Beziehungen zu den anderen XSD- Dateien aufgelöst. Innerhalb der Auflösung dieser Beziehung zu der XSD- und dem Einlesen dieser werden die Relationen zu den anderen XSD- Dateien geklärt. Dies ermöglicht während des Einlesens ein Aufbrechen der Datentypen in ihre atomare Struktur. Somit ist es innerhalb der Dokumentation

direkt möglich zu sehen, aus welchen Datentypen der Parameter besteht und welche der Felder als Pflichtfelder deklariert sind.

Sobald die Relationen aufgelöst sind, kann der eigentliche Inhalt der WSDL- Datei eingelesen werden. Hierbei handelt es sich um die Operationen der Schnittstelle mit ihren definierten Parametern.

4.7 Datenausgabe

Die Datenausgabe erfolgt als ein HTML-Dokument. Das Resultat ist vergleichbar mit dem eines Javadoc-Generators. So wird für jede WSDL- Datei eine eigene HTML-Seite erzeugt. Die Seite besteht aus mehreren Teilen, eine Auflistung aller XSD- Importe, Auflistung aller Methoden mit deren Parametern und Rückgabewerten.

Das Ganze wird über mehrere Framesets und Frames realisiert. Auf der linken Bildschirmhälfte befindet sich ein weiteres Frameset, welches für die Navigation zuständig ist. In diesem Frameset ist der obere Teil für die Navigation zwischen den WSDL- Dateien zuständig und der untere Teil zeigt alle Methoden der jeweiligen Selektion an. Bei dem Klick auf „Alle WSDL-Dateien“ werden alle Operationen von jeder WSDL- Datei angezeigt. Nach dem Klick auf eine spezifische WSDL- Datei werden auch nur die Operationen dieser im unteren Fenster angezeigt. Beim Klick auf eine der Operationen wird im großen Inhaltsfenster in die Detailansicht für die WSDL- zu der jeweiligen Operation gesprungen.

Wurde noch keine Operation angeklickt oder es wurde in dem Navigationsbereich des Inhalts auf „Overview“ geklickt, so wird im Inhaltsfenster die Übersicht aller WSDL-Dateien angezeigt (Abbildung 6). Ein Klick auf eine der WSDL- Dateien sorgt für eine Anzeige der Detailansicht für diese WSDL- Datei (Abbildung 7). Zusätzlich wird in der Titelleiste des Browsers der Titel auf den Namen der WSDL- gesetzt und im Navigationsbereich des Inhaltes kann mit einem Klick auf „Overview“ wieder in die Übersicht über alle WSDL- Dateien gesprungen werden.

4.8 Fehlerhandling

Durch das Test Driven Developement wird natürlich ein Großteil der möglichen Fehler, bei sinnvollen Testfällen, bereits abgedeckt. Sollte es dennoch während der Laufzeit zu mir bisher unbeachteten Fehlern kommen, so werden alle entstehenden Exceptions in eine Logdatei mit ihrem kompletten Stacktrace aufgeführt, sodass ich in der Lage bin, die Fehler in künftigen Versionen zu beheben. Zusätzlich gibt es viele Stellen, an denen ich versuche Fehler, die durch die Umgebung entstehen, zu umgehen. Meine FileReaderUtils sind ein gutes Beispiel dafür. Diese statische Klasse steht in direkter Verbindung mit der Umgebung, und bestimmte Fälle in der Umgebung können zu Fehlern in der Klasse führen. Das größte Fehlerpotential entsteht beim Initialisieren des Readers, sollte ein falscher Pfad angegeben sein oder die Datei ist bereits durch Dritte geöffnet, so wird die Datei blockiert. Hierdurch kann der FileStreamReader nicht komplett initialisiert werden. Daher überprüfe ich vor jedem Aufruf ob die Verbindung überhaupt zustande gekommen ist. Sollte dies nicht der Fall gewesen sein, wird erneut versucht den Reader zu initialisieren und dann bei erneutem Fehler eine Logmeldung geschrieben.

5. Qualitätssicherung

Da es sich bei der Anwendung um eine Java-Anwendung handelt und ich mich für ein Test Driven Developement als Entwicklungsprozess entschieden habe, konnte ich bereits während der Entwicklung die gewünschte Funktionalität gewährleisten. Hierzu entstehen während der Entwicklung sogenannte Grey-Box-Tests. Ein Grey-Box-Test verbindet die Vorteile von Black-Box-Tests und White-Box-Tests. Sie werden vor dem Erstellen des produktiven Codes definiert und enthalten fachliche Anforderungen und Erwartungen. Somit muss anders als bei den Black-Box-Tests kein weiterer Entwickler für das Erstellen der Testfälle einbezogen werden. Weiterhin ist das Auslassen möglicher Fehlerfälle weniger wahrscheinlich als bei einem White-Box-Test, welche zwar von dem gleichen Entwickler wie im produktiven Code erzeugt werden, aber nach dem Erstellen des Codes erzeugt werden.

Da bei mir lokal auf dem PC auch der aktuelle ausgecheckte Stand aus dem SVN-Repository für den ServicePoint vorhanden war, konnte ich auch eine Integration in den Prozess gewährleisten. Die WSDL- und XSD- Daten des ServicePoints ermöglichten es mir weiterhin, mein Projekt auf eine unbekannte, komplexe WSDL- und dazugehörige Dateien zu testen. Da ich während der Entwicklung zum größten Teil mit meiner selbst erstellten WSDL- und dazugehörigen XSD- Dateien gearbeitet habe, waren mir diese Informationen bekannt und ich hatte einen relativ begrenzten Blick auf alles.

Wie in Kapitel 4.1 Entwicklungsumgebung erwähnt, nutzte ich auch PMD und Find Bugs um eine gute Codequalität sicherzustellen. Beide dieser Tools ermöglichen mir das Finden potentieller Fehler innerhalb meines Codes, da sie umfangreicher als die Compilerwarnungen von Eclipse sind.

6. Abschluss des Projekts

6.1 Übergabe an den Fachbereich

Mit Abschluss des Projekts wurde die Anwendung von mir dem Fachbereich, also allen ServicePointEntwicklern, in einer Präsentation vorgestellt und die vorhandenen Funktionen demonstriert. Um die Einführung in die Nutzung des Programms zu erleichtern, wurde von mir ein Benutzerhandbuch geschrieben. Zusätzlich erhoffe ich mir durch das Benutzerhandbuch weniger Rückfragen der ServicePointntwickler, da in diesem Dokument die richtige Benutzung der Anwendung aufgeführt ist und auch die von mir betrachteten Fehlermeldungen, die bei der Ausführung auftreten können und was deren Ursache sein kann.

6.2 Dokumentation des Projekts

Um ein einfacheres Weiterentwickeln zu ermöglichen habe ich aus meinem Quellcode mit dem Dokumentierwerkzeug Javadoc und den dafür benötigten Annotationen in den Kommentaren eine Codedokumentation erzeugt. Weiterhin sind auch noch zusätzliche Kommentare in den einzelnen Methoden vorhanden, die vom Javadoc-Generator zwar nicht mit einbezogen werden, aber bei der Weiterentwicklung für jeden Entwickler ersichtlich sind und so eine schnellere Einarbeitung in den Code ermöglichen. Dadurch soll eine einfachere Wartung des Codes für meine Kollegen und mich möglich sein.

6.3 Soll-Ist-Abgleich

Nach Abschluss der letzten Dokumentationsarbeiten, habe ich einen Soll-Ist-Abgleich erstellt, um dem Fachbereich eine Möglichkeit zu bieten, einen Vergleich zwischen den gewünschten und den umgesetzten Funktionen zu bieten. Weiterhin erfolgte ein Vergleich der ursprünglichen Planung aus dem Projektantrag mit der für die Umsetzung benötigten Zeit.

Die folgende Darstellung ist nur dazu da, um die im Projektantrag geschätzte Zeit und die real benötigte Zeit miteinander zu vergleichen. Die Reihenfolge der Punkte entspricht nicht der Reihenfolge der Umsetzung. Die in der Tabelle angegebenen Zeiten sind jeweils im Stundenformat.

	Soll	**Ist**
Analysephase	**13**	**11**
Ist-Zustand	2	5
Fachgespräche	4	2
Soll Konzept	5	3
Kosten-Nutzen Analyse	2	1
Designphase	**10**	**12**
Erzeugen Klassendiagramms	3	3
Abläufe der Anwendung	7	9
Entwicklungsphase	**35**	**35**
Schreiben von Tests	10	10
Erzeugen des Models	4	6
Erzeugen der Logik	17	19
Erzeugen der GUI	4	0
Projektübergabe	**12**	**12**
Dokumentation	10	10
Projektabgabe	2	2
Summe	**70**	**70**

Tabelle 2: Gegenüberstellung Soll-Aufwand und Ist-Aufwand

Innerhalb des Zeitraums für die betriebliche Projektarbeit gab es nur geringere Abweichungen von meiner ursprünglichen Planung. Folgende Änderungen haben sich während der Entwicklung ergeben: In der Analysephase hat die Ist-Analyse weitaus mehr Zeit in Anspruch genommen, da das Datenmodell sich als weitaus komplexer als ursprünglich geplant herausstellte. Jedoch konnte etwas Zeit in dem Fachgesprächen und Soll-Konzept eingespart werden, da sich hier schnell das gewünschte Endprodukt rauskristallisierte und es somit weniger Rückfragen an den Fachbereich gab.

Da der Wunsch vom Fachbereich geäußert wurde, das Programm in einem *CI*-Prozess zu integrieren, entfiel die Entwicklung für GUI-Aufgaben und ich konnte die extra Zeit in das Erzeugen des Datenmodells innerhalb meines Programmes und der Logik investieren. Insgesamt ist die benötigte Zeit jedoch bei den 70 ursprünglich geplanten Stunden geblieben.

6.4 Fazit

Da ich bis zum Anfang dieses Projektes nur mit dem Erzeugen der Aufrufe vom Integration Server zum ServicePoint zu tun hatte, war dieses Projekt eine gute Möglichkeit ein besseres Verständnis vom ServicePoint, mit dessen Datenkonzept und Datenverarbeitung zu bekommen.

Im Laufe dieses Projektes kamen durch meine bisher geringen Kenntnisse vom ServicePoint und dem Aufbau eines dynamischen Interpreters für die Dateien einige Herausforderungen auf. Diese galt es alle selbständig zu bewältigen und dadurch konnte ich meine Kenntnisse aus den verschiedensten Aspekten der Softwareentwicklung verbessern.

Alles in allem bin ich mit dem Ergebnis der Projektarbeit sehr zufrieden, da ich im Laufe der Entwicklung viele Kenntnisse erlangt habe. Jedoch bin ich schon immer eine selbstkritische Person gewesen und dadurch sehe ich auch nach wie vor viel Potential für Verbesserungen.

6.5 Ausblick

Wie gerade schon erwähnt, gibt es aus meiner Sicht noch viel Potential für Verbesserungen innerhalb der Anwendung. Einerseits ist es eine Optimierung der Performance, da an vielen Stellen ein Caching der häufiger genutzten XSD- Dateien sinnvoll wäre, ich dieses aber Aufgrund des Umfangs des Projektes noch nicht umsetzen konnte.

Außerdem sehe ich die Möglichkeit, durch dieses Projekt eine Dokumentationsabdeckung anzuzeigen. Somit können die ServicePointEntwickler sehen, an welchen Stellen noch Verbesserungspotential besteht.

Natürlich kann es im Laufe der Verwendung trotz Testautomatisierung und Testphase zum Auftreten von diversen Bugs kommen. In diesen Fällen muss natürlich Wartung erfolgen um den Fehler zu beseitigen.

Da mein Programm auch nicht nur an den ServicePoint gebunden ist sondern relativ dynamisch gehalten wurde, kann es nach geringen Anpassungen auch für andere Projekte genutzt werden, solange dort die Definition der Datentypen erfolgt und die Operationen, der Port und sonstige Informationen über die Schnittstelle in der WSDL- dokumentiert sind.

A. Anhang

A.1 Glossar

Begriff	Erklärung
ANT	= **A**nother **N**eat **T**ool Open Source Tool mit dessen Hilfe man automatisch aus vorhandenem Quellcode installierbare Softwarepakete erzeugen kann.
CI	= **C**ontinuos **I**ntegration Beschreibt den Prozess des kontinuirlichen Zusammenfügens mehrerer Komponenten einer Anwendung.
Datenbank	Eine Datenbank beschreibt eine Datensammlung an einem zentralen Ort mit einer geregelten Zugriffskontrolle.
DoD	=**D**efinition **o**f **D**one Eine Checkliste mit deren Hilfe die Vollständigkeit eines Vorgangs überprüft werden kann. Bei der Kontrolle ist die Einhaltung der DoD Pflicht.
ESB	= **E**nterprise **S**ervice **B**us Sammlung von Komponenten eines Programmes zur Datenbeschaffung und Datenaufbereitung.
Eclipse	Ein Open Source Programm, welches zur Entwicklung von diverser Software genutzt werden kann. Es kann durch unterschiedlichste Plugins in seiner Funktionalität erweitert werden.
File Repository	Ein Repository welches lokal, also nicht auf einem Server angelegt wurde. Daher lohnt es sich nur für Einzelanwender.
HTML	= **H**yper **T**ext **M**arkup **L**anguage Eine Sprache zur strukturierten Datenhaltung und Anzeige von Texten, Bildern und Hyperlinks.
IDE	= **I**ntegrated **D**evelopment **E**nvironment Sammlung von Anwendungsprogrammen innerhalb eines einzelnen Programmes um die Entwicklung von Software ohne viel hin und herwechseln zu ermöglichen.
Java	Objektorientierte Programmiersprache
Javadoc	Dokumentationswerkzeug, welches aus Java Quelltexten inkl Kommentaren, die eine bestimmte Syntax benötigen, eine HTML-Dokumentation erzeugt.

JAXB	= **J**ava **A**rchitecture for **X**ML **B**inding Integriert in Java ab Version 1.6. Es ermöglicht das einfache Mappen von Java-Objekten in Dateien im XML-Format. Gleichzeitig kann man auch XML-Dateien in Java-Klassen umwandeln bzw. in Instanzen dieser Klassen erzeugen.
JUnit	Framework, welches das Testen von Java-Programmen ermöglicht. Hierzu werden sogenannte Unit Tests erstellt, die auch automatisiert aufgerufen werden können.
Open Source	dt. Quellenoffen Bezeichnung für ein Programm dessen Quellcode und zur freien Verfügung steht.
Personentag (PT)	Menge der Arbeit, die eine Person an einem Arbeitstag vollbringt.
Repository	dt. Projektarchiv In einem Repository, werden alle eingecheckte Änderungen gespeichert. Dies ermöglicht es die Änderungen ganz einfach auf andere Clients/Entwicklungsstationen zu übertragen. Jede dieser Revisionen kann mit einem Kommentar versehen werden. Zusätzlich ist es möglich noch weitere Felder zu ergänzen.
Request	Anfrage an einen Server.
Response	Antwort des Servers, die als Reaktion auf ein Request geschickt wird.
Revision	Sammlung aller Änderungen die mit einem Commit in das SVN Repository kamen.
ServicePoint	Sammlung von WebService Schnittstellen zur Verbindung zu mehreren Datenbanken oder Großrechner (HOST)
SVN	Open Source Programm von Apache, zur Versionsverwaltung von Dateien. Alle Versionen werden in einem zentralen Repository gespeichert.
TDD	= **T**est **D**riven **D**evelopement Ein Entwicklungskonzept, in dem die Tests vor dem Code entwickelt werden. Jeder Test wird so klein wie möglich gehalten und dafür werden dann dementsprechend viele entwickelt. Es gibt mehrere Iterationen in diesem Prozess und es wird durch Änderungen am Code versucht die Funktionalität des Tests herzustellen.
Tortoise-SVN	Eine freie Software, welche einen Client für Subversion bereitstellt. Dies geschieht über eine

	Shell Extension, sodass direkt über den Windows Explorer die GUI über einen Rechtsklick aufgerufen werden kann und die gewünschte Operation ausgewählt werden kann.
Trunk	Hauptentwicklungszweig innerhalb eines SVN-Projektes.
Umsystem	Ein Fremdsystem welches in einer Beziehung zu einem anderen System steht, wird auch als Umsystem bezeichnet.
WinMerge	Bei WinMerge handelt es sich um ein Open Source Programm, welches das vergleichen von Ordnern oder einzelnen Dateien unter Windows vereinfacht.
WSDL	= **W**eb **S**ervices **D**escription **L**anguage Beschreibungssprache für Webservices zum Datenaustausch von Nachrichten auf Basis von XML Dateien. (W3C Standard)
XML	= E**x**tensible **M**arkup **L**anguage Sprache zur Anzeige von hierachisch strukturierten Daten im Textformat
XML-Knoten	Als Knoten bezeichnet man alle Teile der Baumstruktur einer XML-Datei. Hierbei kann jeder Knoten beliebig viele untergeordnete Knoten haben, so genannte Child-Nodes.
XSD	= **X**ML **S**chema **D**efinition Beschreibungssprache für die Definition von Objekten innerhalb einer Datei im XML-Format. Die Sprache unterstützt viele einfache Datentypen mit deren Hilfe komplexere Datentypen erzeugt werden können.

A.2 Abbildungsverzeichnis

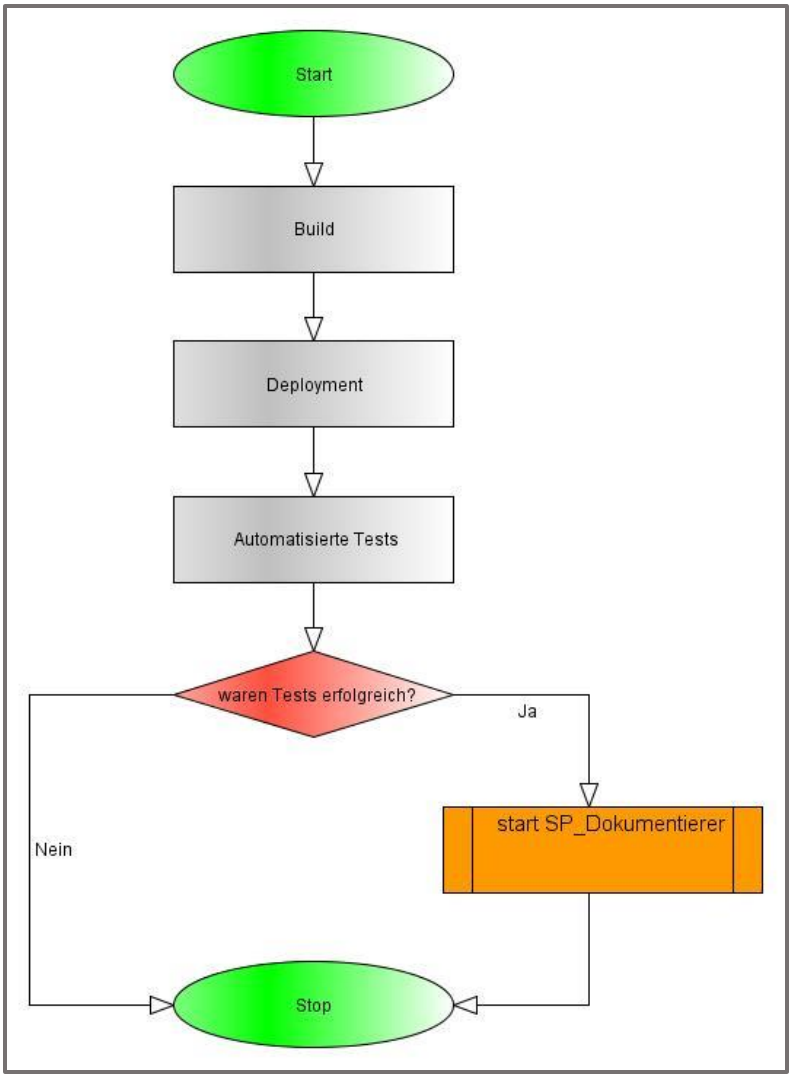


Abbildung 1 Programmablaufplan der Integration des Programmes in den bestehenden Build- und Deploymentprozess

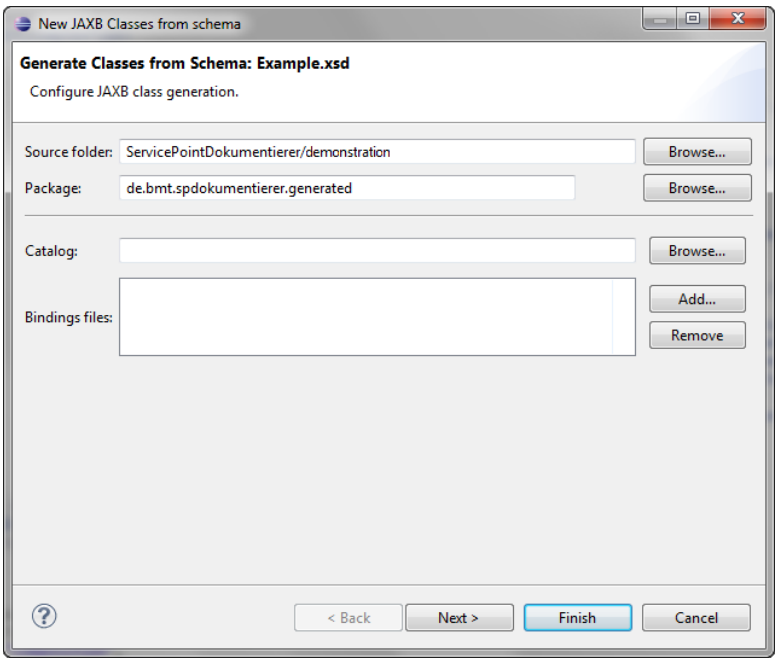
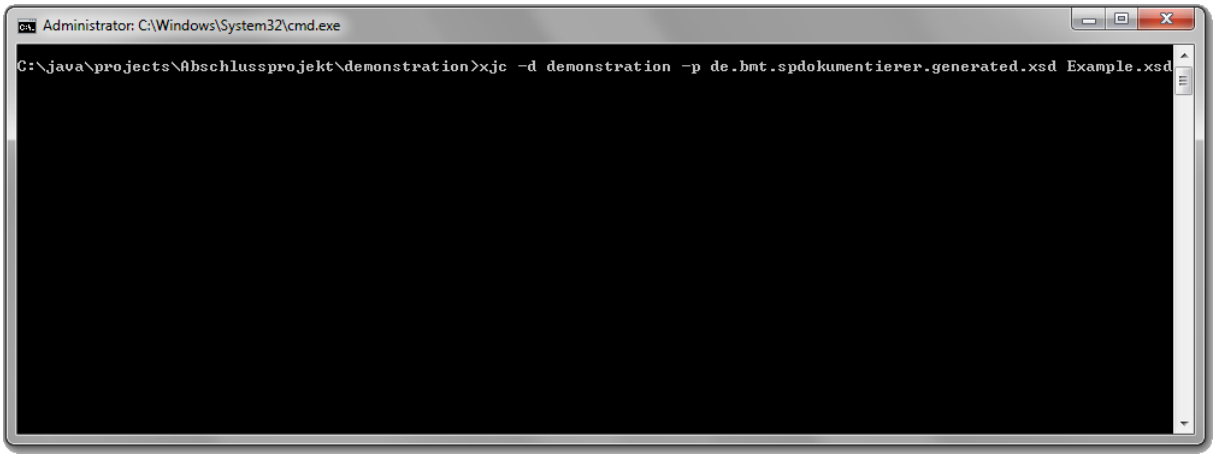


Abbildung 2
Generierung der Java Klassen mithilfe von JAXB innerhalb von Eclipse

Abbildung 3
Generierung der grundlegenden Java Klassen mithilfe von JAXB über das CMD-Fenster aufgerufen.



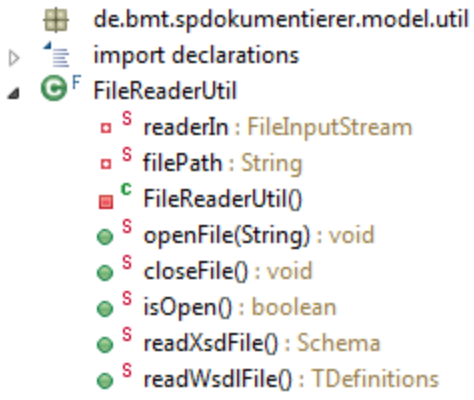


Abbildung 4
Auszug Klassenstruktur FileReaderUtils

```
	/**
	 *
	 * @return {@link Schema} XmlSchema
	 */
	public static Schema readXsdFile() {
		Schema result = null;
		// if no connection is set and the set filePath is not empty, it
		// tries to open the file
		if (!(isOpen() || StringUtils.isEmpty(filePath))) {
			openFile(filePath);
		}
		// validation that the fileReader was initialized
		if (isOpen()) {
			result = JAXB.unmarshal(readerIn, Schema.class);
		} else {
			LoggerUtil.logException("Fehler beim Lesen der Datei");
		}
		return result;
	}

	/**
	 *
	 * @return {@link TDefinitions} wsdlDefinition
	 */
	public static TDefinitions readWsdlFile() {
		TDefinitions result = null;
		// if no connection is set and the set filePath is not empty, it
		// tries to open the file
		if (!(isOpen() || StringUtils.isEmpty(filePath))) {
			openFile(filePath);
		}
		// validation that the fileReader was initialized
		if (isOpen()) {
			result = JAXB.unmarshal(readerIn, TDefinitions.class);
		} else {
			LoggerUtil.logException("Fehler beim Lesen der Datei");
		}
		return result;
	}
```

Abbildung 5
Auszug Quellcode aus den FileReaderUtils

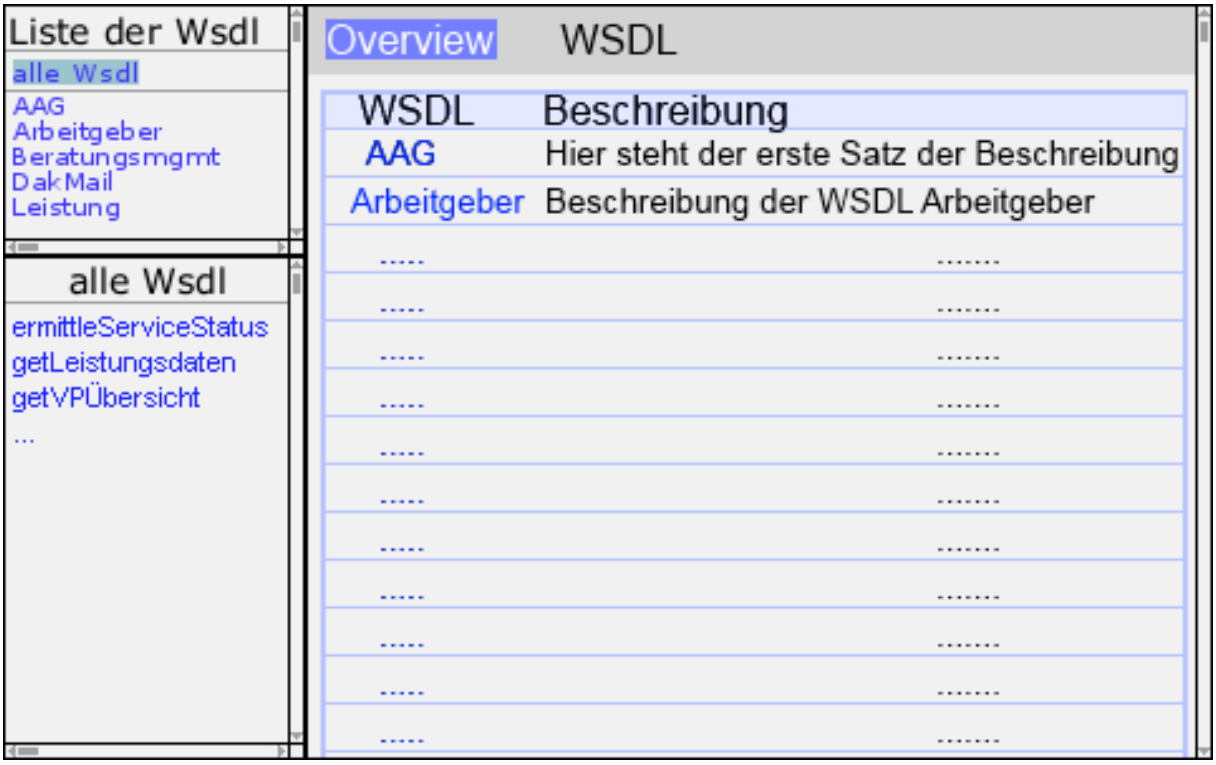


Abbildung 6
Konzept zu der Anzeige der Dokumentation in der Übersicht von allen WSDL- Dateien.

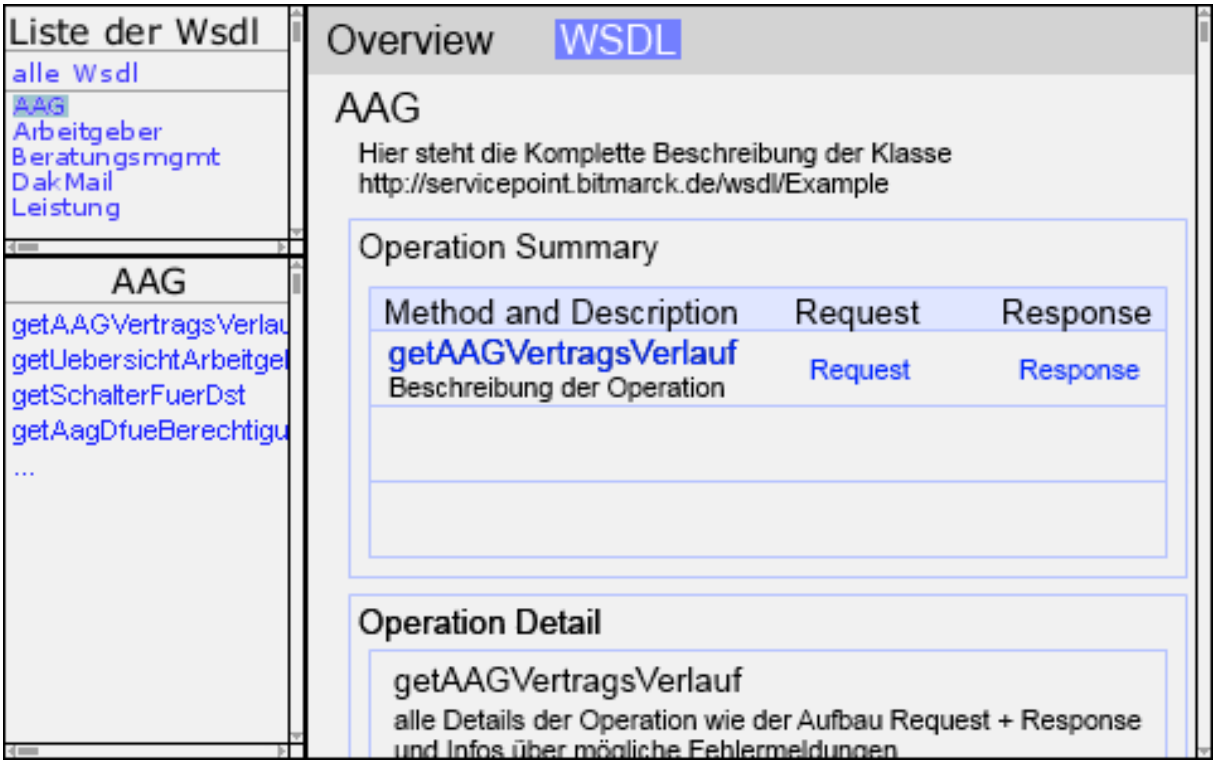


Abbildung 7
Konzept zu der Anzeige der Dokumentation bei Detailansicht einer WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Example"
        targetNamespace="http://servicepoint.bitmarck.de/wsdl/Example"
        xmlns:tns=http://servicepoint.bitmarck.de/wsdl/Example
        xmlns:extp="http://servicepoint.bitmarck.de/types/Example"
        xmlns:xsd1="http://servicepoint.bitmarck.de/types/common"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <wsdl:types>
                <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                        <xsd:import namespace="http://servicepoint.bitmarck.de/types/Example"
                                schemaLocation="../types/Example.xsd">
                        </xsd:import>
                        <xsd:import namespace="http://servicepoint.bitmarck.de/types/common"
                                schemaLocation="../types/common.xsd">
                        </xsd:import>
                </xsd:schema>
        </wsdl:types>

        <!-- helloWorldMessage -->
        <wsdl:message name="helloWorldRequest">
                <wsdl:part name="request" element="extp:helloWorldRequest" />
        </wsdl:message>
        <wsdl:message name="helloWorldResponse">
                <wsdl:part element="extp:helloWorldResponse" name="response" />
        </wsdl:message>

        <!-- Fault message -->
        <wsdl:message name="ExampleFault">
                <wsdl:part name="parameters" element="xsd1:Warnings"></wsdl:part>
        </wsdl:message>

        <wsdl:portType name="Example_PortType">
                <!-- helloWorld operation -->
                <wsdl:operation name="helloWorld">
                        <wsdl:input message="tns:helloWorldRequest"></wsdl:input>
                        <wsdl:output message="tns:helloWorldResponse"></wsdl:output>
                        <wsdl:fault name="fault" message="tns:ExampleFault"></wsdl:fault>
                </wsdl:operation>
        </wsdl:portType>

        <wsdl:binding name="ExampleSOAP" type="tns:Example_PortType">
                <soap:binding style="document"
                        transport="http://schemas.xmlsoap.org/soap/http" />
                <wsdl:operation name="helloWorld">
                        <soap:operation

        soapAction="http://servicepoint.bitmarck.de/wsdl/Example/helloWorld" />
                        <wsdl:input>
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                                <soap:body use="literal" />
                        </wsdl:output>
                        <wsdl:fault name="fault">
                                <soap:fault use="literal" name="fault" />
                        </wsdl:fault>
                </wsdl:operation>
        </wsdl:binding>

        <wsdl:service name="Example">
                <wsdl:port binding="tns:Example" name="Example_Port">
                        <soap:address
                        location="http://servicepoint.bitmarck.de/ServicePointWS/Example" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Abbildung 8
Aufbau der Example.wsdl

A.3 Literaturverzeichnis