technische universität
dortmund

Bachelorarbeit

# Konzeption und Implementierung einer Schnittstelle zur Sicherheitsanalyse mit CARiSMA im Kontext von Serviceorientierten Architekturen

**Andreas Beckmann**
**16. Januar 2014**

Gutachter: Prof. Dr. Jan Jürjens
Dipl.-Inform. Dipl.-Math. Sebastian Pape

Prof. Dr. Jan Jürjens Lehrstuhl 14 Software Engineering
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund
http://www-jj.cs.uni-dortmund.de/secse

Andreas Beckmann
andreas.beckmann@udo.edu
Matrikelnummer: 133677
Studiengang: Bachelor Angewandte Informatik

Bachelorarbeit
Thema: Konzeption und Implementierung einer Schnittstelle zur Sicherheitsanalyse mit CARiSMA im Kontext von Serviceorientierten Architekturen

Eingereicht: 16. Januar 2014

Betreuer: Christian Wessel, Sebastian Pape

Prof. Dr. Jan Jürjens Lehrstuhl 14 Software Engineering
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund

# Ehrenwörtliche Erklärung

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

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dortmund, den 16. Januar 2014

Andreas Beckmann

# Kurzfassung

Im Rahmen guten Softwareengineerings stellen Modellierungssprachen wie UML eine unverzichtbare Hilfe dar. Sie erlauben es, Informationen über die verschiedensten Aspekte einer Software- oder Systemlösung schnell und verständlich zwischen Kommunikationspartnern auszutauschen. Durch die Entwicklung von UML-Erweiterungen wie UMLsec ist es möglich, bereits bei der Modellierung von System- und Softwarelösungen Sicherheitsanforderungen festzulegen. Diese Sicherheitsanforderungen können im Folgenden durch Sicherheitsanalysewerkzeuge wie CARiSMA auf das Einhalten von gewünschten Sicherheitseigenschaften untersucht werden. Sicherheitsanalysewerkzeuge wie CARiSMA weisen jedoch das Problem auf, dass sie nicht in automatisierte Prozesse integriert werden können, da sie eine manuelle Eingabe des Nutzers benötigen. Im Rahmen dieser Arbeit wird dieser Mangel für das Sicherheitsanalysewerkzeug CARiSMA behoben und somit eine Möglichkeit zur Integration in automatisierte Prozesse geschaffen. Genauer wird die Konzeption und Implementierung einer Schnittstelle zu CARiSMA im Kontext von serviceorientierten Architekturen durchgeführt und beschrieben. Hierzu wird CARiSMA aus der Entwicklungsumgebung Eclipse herausgelöst und in Form eines OSGi-Bundles als eigenständiger über SOAP-Anfragen erreichbarer Webservice zur Verfügung gestellt.

# Abstract

The Unified Modeling Language is an indispensable tool in the context of good software engineering. It provides a way to communicate various aspects of a software concept to other parties in an understandable and compact format. Due to the development of UML extensions like UMLsec it is possible to define security requirements inside an UML model. Using a model checker like CARiSMA allows to check those requirements against a set of desired security properties. However, model checkers such as CARiSMA have the shortcoming that they cannot be integrated into automated processes, since they require manual input by the user. This thesis will address these shortcomings for the model checker CARiSMA, thus creating a possibility to integrate it into automated processes. More specifically, the thesis will cover the design and implementation of the integration of CARiSMA as a webservice into a service-oriented architecture.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** xiii

**Tabellenverzeichnis** xv

**Quellcodeverzeichnis** xvii

**1 Einleitung** 1
1.1 Motivation und Hintergrund . . . 1
1.2 Vorarbeiten/Related Work . . . 2
1.3 Aufbau der Arbeit . . . 3

**2 Aufgabenstellung** 5
2.1 Konzeption . . . 5
2.2 Implementierung . . . 6
2.3 Ziele . . . 6
2.3.1 Muss-Ziele . . . 6
2.3.2 Soll-Ziele . . . 7
2.3.3 Kann-Ziele . . . 8

**3 Grundlagen** 11
3.1 Unified Modeling Language . . . 11
3.2 UMLsec . . . 11
3.3 CARiSMA . . . 11
3.3.1 Check . . . 12
3.3.2 Check-Registry . . . 12
3.3.3 Model-Type-Registry . . . 12
3.4 Serviceorientierte Architekturen . . . 13
3.4.1 Webservices . . . 13
3.4.2 SOAP . . . 13
3.4.3 WSDL . . . 14
3.4.4 REST . . . 15
3.4.5 Vergleich von SOAP und REST . . . 16
3.5 OSGi . . . 16
3.5.1 Eclipse Equinox . . . 17
3.5.2 Distributed OSGi . . . 17
3.5.3 Declerative OSGi Services . . . 18
3.6 SQLite-Datenbank . . . 18

3.7 Verwendete Werkzeuge . . . 18
3.7.1 SoapUI . . . 18
3.7.2 RESTClient . . . 18

**4 Konzept 19**
4.1 Anwendungsfalldiagramm . . . 19
4.2 Aktivitätsdiagramme . . . 29
4.2.1 Run Analysis . . . 29
4.2.2 Get Available Checks . . . 29
4.2.3 Authenticate User . . . 29
4.2.4 Create User . . . 30
4.2.5 Read User . . . 32
4.2.6 Update User . . . 32
4.2.7 Delete User . . . 32
4.2.8 Create File . . . 33
4.2.9 Read File . . . 33
4.2.10 Update File . . . 35
4.2.11 Delete File . . . 36
4.3 Kommunikation . . . 37
4.3.1 WSDL-Datei . . . 37
4.3.2 REST-Anfragen . . . 39
4.4 Klassendiagramm . . . 40
4.4.1 CarismaWS . . . 40
4.4.2 Gateway . . . 42
4.4.3 RestInterface . . . 42
4.4.4 RestImpl . . . 42
4.4.5 Controller . . . 42
4.4.6 WSConnector . . . 42
4.4.7 DatabaseInterface . . . 42
4.4.8 DatabaseHelper . . . 42
4.5 Architektur . . . 42
4.5.1 Komponente Kommunikationsschnittstelle . . . 43
4.5.2 Komponente Controller . . . 43
4.5.3 Komponente Datenbank . . . 44
4.5.4 Komponente CARiSMA . . . 44
4.6 Sicherheit . . . 44
4.6.1 Abhören . . . 44
4.6.2 Integrität . . . 44
4.6.3 Authentifizierung . . . 44
4.6.4 Wiedereinspielattacke . . . 45
4.6.5 Auslesen der Datenbank . . . 45
4.6.6 Zugriff auf fremde Dateien . . . 45

**5 Implementierung** **47**
5.1 Terminal . . . . . . . . . . 47
5.1.1 Starten der Equinox-Instanz . . . . . . . . . . 47
5.1.2 Kommandozeilenaufrufe . . . . . . . . . . 47
5.1.3 Kommunikation mit CARiSMA . . . . . . . . . . 48
5.1.4 Starten CARiSMAs . . . . . . . . . . 48
5.2 Webservice . . . . . . . . . . 48
5.2.1 Kommunikationsschnittstellen . . . . . . . . . . 48
5.2.2 Controller . . . . . . . . . . 50
5.2.3 Datenbank . . . . . . . . . . 50
5.2.4 CARiSMA . . . . . . . . . . 50
5.3 SVN-Hook . . . . . . . . . . 50
5.4 Nachinstallieren . . . . . . . . . . 51
5.5 Implementierung der Sicherheitsanforderungen . . . . . . . . . . 51

**6 Validierung** **53**
6.1 Tests . . . . . . . . . . 53
6.1.1 Teststrategie . . . . . . . . . . 53
6.1.2 Komponenten-Tests . . . . . . . . . . 53
6.1.3 Systemtest . . . . . . . . . . 54
6.2 Qualitätssicherung . . . . . . . . . . 55
6.2.1 Überprüfung durch Checkstyle . . . . . . . . . . 55
6.2.2 Dokumentation . . . . . . . . . . 55

**7 Fazit & Ausblick** **57**
7.1 Fazit . . . . . . . . . . 57
7.1.1 Evaluation der Ziele . . . . . . . . . . 57
7.2 Ausblick . . . . . . . . . . 60

**A Weitere Informationen** **61**

**Literaturverzeichnis** **93**

# Abbildungsverzeichnis

3.1 Elemente einer WSDL-Datei [FZ09, Abbildung 3.4] . . . . . . . . . . . 15
3.2 Struktur der OSGi-Plattform [All13] . . . . . . . . . . . . . . . . . . . 17

4.1 Anwendungsfalldiagramm . . . . . . . . . . . . . . . . . . . . . . . . . 20
4.2 Aktivitätsdiagramm des Anwendungsfalles „Run Analysis“ . . . . . . 30
4.3 Aktivitätsdiagramm des Anwendungsfalles „Get Available Checks“ . . 31
4.4 Aktivitätsdiagramm des Anwendungsfalles „Authenticate User“ . . . . 31
4.5 Aktivitätsdiagramm des Anwendungsfalles „Create User“ . . . . . . . 32
4.6 Aktivitätsdiagramm des Anwendungsfalles „Read User“ . . . . . . . . 33
4.7 Aktivitätsdiagramm des Anwendungsfalles „Update User“ . . . . . . . 33
4.8 Aktivitätsdiagramm des Anwendungsfalles „Delete User“ . . . . . . . 34
4.9 Aktivitätsdiagramm des Anwendungsfalles „Create File“ . . . . . . . . 34
4.10 Aktivitätsdiagramm des Anwendungsfalles „Read File“ . . . . . . . . 35
4.11 Aktivitätsdiagramm des Anwendungsfalles „Update File“ . . . . . . . 36
4.12 Aktivitätsdiagramm des Anwendungsfalles „Delete File“ . . . . . . . . 37
4.13 Klassendiagramm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.14 Architektur der Komponenten . . . . . . . . . . . . . . . . . . . . . . 43

# Tabellenverzeichnis

4.1 Beschreibung des Anwendungsfalles „Run Analysis“ . . . . . . . . . . . 21
4.2 Beschreibung des Anwendungsfalles „Get Available Checks“ . . . . . . 22
4.3 Beschreibung des Anwendungsfalles „Authenticate User“ . . . . . . . 22
4.4 Beschreibung des Anwendungsfalles „Create User“ . . . . . . . . . . . 23
4.5 Beschreibung des Anwendungsfalles „Read User“ . . . . . . . . . . . . 24
4.6 Beschreibung des Anwendungsfalles „Update User“ . . . . . . . . . . 24
4.7 Beschreibung des Anwendungsfalles „Delete User“ . . . . . . . . . . . 25
4.8 Beschreibung des Anwendungsfalles „Create File“ . . . . . . . . . . . 26
4.9 Beschreibung des Anwendungsfalles „Read File“ . . . . . . . . . . . . 26
4.10 Beschreibung des Anwendungsfalles „Update File“ . . . . . . . . . . . 27
4.11 Beschreibung des Anwendungsfalles „Delete File“ . . . . . . . . . . . 28

6.1 Testüberdeckungsgrade der Klassen . . . . . . . . . . . . . . . . . . . . 54
6.2 Testüberdeckungsgrade der Bundles . . . . . . . . . . . . . . . . . . . . 55

# Quellcodeverzeichnis

A.1 Quellcode der Controller-Klasse . . . . . . . . . . . . . . . . . . . . . . . 61
A.2 Quellcode der Gateway-Klasse . . . . . . . . . . . . . . . . . . . . . . . . 70
A.3 Quellcode des RestInterface-Interfaces . . . . . . . . . . . . . . . . . . . . 86
A.4 Quellcode des DatabaseInterface-Interfaces . . . . . . . . . . . . . . . . . . 89

# 1 Einleitung

Im Folgenden wird die Arbeit motiviert und eine Reihe von wichtigen Hintergrundinformationen erläutert. Sie wird zu Vorarbeiten in Bezug gesetzt und verwandte Arbeiten werden betrachtet. Im Anschluss wird der Aufbau der Arbeit dargestellt.

## 1.1 Motivation und Hintergrund

Viele moderne Softwareprojekte weisen eine deutlich gestiegene Komplexität auf. Dies führt dazu, dass ein unorganisierter und undokumentierter Versuch, ein solches Softwareprojekt zu realisieren, einem deutlich höheren Risiko ausgesetzt ist, zu scheitern, als ein wohlorganisiertes und dokumentiertes Projekt.
Als ein unverzichtbares Werkzeug für den gesamten Softwareentwicklungsprozess hat sich hier die Unified Modeling Language (UML) [BRJ06], eine abstrakte Modellierungssprache, welche beispielsweise bei einer modellbasierten Softwareentwicklung genutzt wird, herausgestellt. Mit Hilfe der UML lassen sich Informationen schnell zwischen allen Projektbeteiligten austauschen, sowie wichtige architektonische Rahmenbedingungen festlegen. Auch erlaubt sie wichtige Abläufe und Strukturen in frühen Entwicklungsphasen festzulegen und im Anschluss deren Einhaltung zu überprüfen.
So ist es durch die Entwicklung der UML-Erweiterung UMLsec [Jür05] möglich, Sicherheitseigenschaften in UML-Modelle zu integrieren und gegen bestehende Sicherheitsanforderungen zu überprüfen. Dies ist besonders bei der Betrachtung von stetig wachsenden Kosten, die Ausfälle oder die Kompromittierung eines Systemes nach sich ziehen können, wie sie zum Beispiel in der Studie „*Market Price Effects of Data Security Breaches.*“ [MRW11] vorgestellt werden, sehr interessant.
Da die Komplexität von so erstellten Modellen eine manuelle Überprüfung gegen Sicherheitsanforderungen höchst aufwendig gestalten würde, wurde das Programm UMLsec-Tool entwickelt, mit welchem es möglich war, UML-Modelle gegen Sicherheitsanforderungen zu überprüfen.
Dieses Werkzeug wurde im Folgenden reimplementiert und wird nun unter dem Namen CARiSMA (CompliAnce/Risk/Security-Model-Analyzer) [WW13] in Kooperation von der Technischen Universität Dortmund und dem Fraunhofer-Institut für Software- und Systemtechnik ISST weiterentwickelt. CARiSMA wird derzeit als Eclipse-Plugin bereitgestellt und enthält 25 Checks, Sicherheitsanalyseverfahren von Modellen bezüglich verschiedener Sicherheitsanforderungen. Das Ausführen eines oder mehrerer Checks auf einem Modell wird Analyse genannt. Solche Analysen spielen im modernen Softwareentwicklungsprozess eine immer wichtigere Rolle und sind teilweise bereits durch Auftraggeber oder Zertifizierungsstellen vorgeschrieben.

Um den Entwicklungsprozess sicherheitskritischer Lösungen weiter zu verbessern und Fehler in möglichst frühen Entwicklungsphasen aufzudecken, wäre eine automatisierte Sicherheitsanalyse im Rahmen von geregelten Entwicklungsprozessen wünschenswert. Diese Integration wird jedoch durch die aktuelle Architektur CARiSMAs behindert. So ist es derzeit ausschließlich möglich, Sicherheitsanalysen mit CARiSMA manuell an einem Computer mit einer lokalen CARiSMA-Installation durchzuführen. Dies bedeutet, dass ein Endanwender eine kompatible Eclipse-Instanz installiert haben muss, welche alle für CARiSMA und die jeweils gewünschten Analysen benötigten Abhängigkeiten enthält. Diese Abhängigkeiten können dazu führen, dass der Einsatz modernster Entwicklungshilfen nicht ohne weiteren Arbeitsaufwand möglich ist, da eventuelle Abhängigkeiten hier zu Problemen führen können. Auch ist die Implementierung nur mit großem Aufwand zu anderen Entwicklungsumgebungen zu portieren, da ein großer Teil der Architektur neu entwickelt werden müsste. Dies schließt die Verwendung für Personen, welche nicht Eclipse nutzen möchten, generell aus.
Die Bereitstellung einer Schnittstelle in Form eines Webservices [FZ09, Kapitel 3] kann hier dazu genutzt werden, diese Schwächen zu beseitigen. Die Implementierung einer neuen Benutzerschnittstelle, wie sie von einer anderen Entwicklungsumgebung benötigt werden würde, würde sich auf die Integration einer neuen grafischen Oberfläche mit einer Anbindung zu einem Webservice begrenzen. Im Rahmen dieser Bachelorarbeit wurde deshalb eine Schnittstelle zu CARiSMA im Kontext von serviceorientierten Architekturen, wie sie von Finger und Zeppenfeld [FZ09, Kapitel 2] beschrieben werden, konzeptioniert und implementiert.

## 1.2 Vorarbeiten/Related Work

Es gibt neben dem bereits erwähnten CARiSMA noch andere Ansätze, Sicherheitsaspekte bereits auf Modell-Ebene zu analysieren, das sogenannte *Model Checking*. Zu nennen wären besonders SecureUML und CORAS.
SecureUML, welches von Torsten Lodderstedt, David Basin und Jürgen Doser entwickelt wird, ist eine Modellierungssprache mit dem Ansatz, Sicherheitsanforderungen direkt in den Spezifikationen zu ermöglichen [LBD02]. Es ist verschiedene Analysesoftware, wie SecureMOVA [Bas+09], verfügbar, welche SecureUML-Modelle auf deren Integrität untersuchen. Diese und vergleichbare Lösungen verwenden jedoch, wie CARiSMA auch, den Ansatz einer lokalen Installation.
CORAS, entwickelt 2001 bis 2003 von der SINTEF Group[1], ermöglicht eine UML-basierte Risikoanalyse [Bra+07]. Das CORAS-Tool [LSS11], welches den Anwender bei der Erstellung und Analyse von CORAS-Modellen unterstützt, setzt jedoch erneut auf ein manuelles Ausführen lokaler Installationen.
Es fehlt bisherigen Lösungen eine Möglichkeit der Anbindung in moderne Geschäftsprozesse ohne die Notwendigkeit des manuellen Eingreifens eines Nutzers.
Die Vorteile der Integration von Softwarelösungen als Webservices in serviceorientierten Architekturen, wie der Interoperabilität und dem beschleunigtem Vertrieb aktualisierter Versionen, wurden bereits in „*An overview of standards and related*

[1] Größte unabhängige skandinavische Forschungsorganisation

*technology in web services*“ [TP02] erläutert. Den Bedarf der Absicherung, bedingt durch sonst unverschlüsselte und wiederholbare Anfragen, wurde bereits in „*Web services: problems and future directions*“ [Wan+04] vorgestellt, Lösungsmöglichkeiten, wie die Verwendung von verschlüsselten Kommunikationswegen oder der Implementierung von WS-Security, in „Practical subversion“ [RB06, Kapitel 9] und „*Taking steps to secure Web services*“ [Gee03] gezeigt. Die Integration eines Webservices mit Hilfe von OSGi wird in „*Think Large, Act Small: An Approach to Web Services for Embedded Systems Based on the OSGi Framework*“ [Roe+10] beispielhaft für einen PKW-Ortungsdienst durchgeführt.

## 1.3 Aufbau der Arbeit

In Kapitel 2 wird die Aufgabe der Arbeit genauer dargestellt und die im Rahmen der Vorbereitungen festgelegten Ziele werden vorgestellt. Im Anschluss werden in Kapitel 3 die Grundlagen, welche zum Verständnis der Arbeit benötigt werden, erläutert. Teil dieser Grundlagen sind die verwendeten Technologien und Protokolle. Das für die Umsetzung entwickelte Konzept wird in Kapitel 4 dargestellt und erläutert. Es wird darauf eingegangen, warum welche Entscheidungen getroffen wurden. Die mit dem Konzept durchgeführte Implementierung wird in Kapitel 5 erläutert. Die Validierung der so erstellten Softwarelösung findet in Kapitel 6 statt. Zum Abschluss wird in Kapitel 7 ein Fazit gezogen und mit einem Ausblick ein Ausgangspunkt für Folgearbeiten gegeben.

# 2 Aufgabenstellung

Die in der Aufgabenstellung bereits erwähnten Technologien und Fachausdrücke werden in Kapitel 3 genauer erläutert.
Es soll eine Schnittstelle zur Sicherheitsanalyse mit CARiSMA im Kontext von serviceorientierten Architekturen konzeptioniert und implementiert werden. Dadurch soll die Verfügbarkeit CARiSMAs erhöht werden und eine Möglichkeit geschaffen werden, CARiSMA außerhalb einer Eclipse-Instanz zu bedienen und in automatisierte Entwicklungsprozesse zu integrieren.
Diese Aufgabe setzt sich hierbei aus drei Teilschritten zusammen:

1. CARiSMA und Eclipse trennen, sodass CARiSMA ohne eine Eclipse-Instanz lauffähig ist,

2. CARiSMA als Webservice bereitstellen und

3. den Webservice in einen Entwicklungsprozess integrieren.

An die Umsetzung der Arbeit stellen sich folgende besondere Anforderungen:

- Die Lösung muss sehr gut erweiterbar sein. Das Hinzufügen neuer Checks sollte keinen nennenswerten Arbeitsaufwand darstellen.

- Es sollen möglichst wenige Änderungen an der grundlegenden CARiSMA-Architektur nötig sein.

- Die Lösung muss in den Grundlagen sicher sein. So darf es nicht durch einfache Angriffe, wie beispielsweise eine Wiedereinspielattacke, möglich sein, sich sicherheitskritische Informationen anzueignen.

## 2.1 Konzeption

Mit Hilfe des entwickelten Konzeptes soll es möglich sein, die spätere Implementierung durchzuführen. Auch soll das Konzept als Dokumentation dienen und somit Dritten die Möglichkeit bieten, die grundlegende Funktionsweise der entwickelten Lösung nachzuvollziehen.
An die Konzeption stellt sich die Anforderung, dass

- alle eventuellen Änderungen an bestehenden Architekturen dargestellt werden,

- die Architektur und die Arbeitsweise der späteren Lösung klar ersichtlich ist,

- die Modelle konform mit gültigen Standards in der Softwareentwicklung erstellt werden.

Ein Bestandteil des Konzeptes setzt sich aus einer Beschreibung in Form von Prosatext und gegebenenfalls für das Verständnis hilfreichen Modellen zusammen.

## 2.2 Implementierung

Die Implementierung der Softwarelösung soll basierend auf dem Konzept erfolgen. Es soll sich hierbei an Richtlinien für gut geschriebenen Quellcode gehalten werden, wie sie in „*Clean Code*“ [Mar09] vorgestellt werden. Der Quellcode soll sinnvoll kommentiert und mit lesbaren Methoden- sowie Variablennamen versehen sein. Jeglicher Quellcode soll am Ende durch die Stufen Komponenten-, Integrations-, und Systemtest gelaufen sein und jede Teststufe erfolgreich durchlaufen.

## 2.3 Ziele

Im Folgenden werden die in der Vorbereitungsphase definierten Ziele, welche es zu erreichen gilt, vorgestellt. Diese lassen sich in die Kategorien Muss-, Soll- und Kann-Ziele unterteilen, wobei Muss-Ziele die höchste Priorität besitzen und Kann-Ziele die niedrigste. Ob ein Ziel erreicht wurde, lässt sich Kapitel 7 entnehmen.

### 2.3.1 Muss-Ziele

a) Softwarekonzept entwickeln
Im Sinne guter softwaretechnischer Verfahrensweisen muss im Rahmen dieser Arbeit ein Softwarekonzept entwickelt werden, welches das weitere Vorgehen in der Implementierung vorab erkennbar und nachvollziehbar darstellt. Es sollen besonders die Themengebiete Schnittstellen zu und Änderungen an der Architektur vorgestellt werden. Am Ende müssen folgende Modelle realisiert sein: Modell der SOAP-Anfragen, Klassendiagramm, welches die Änderungen an der Architektur verdeutlicht, Aktivitätsdiagramme, aus denen alle möglichen Aktivitäten erkennbar sind (Ablauf einer Anfrage, Ablauf von Analysen, Ablauf von Antworten). Des Weiteren muss eine Fließtextbeschreibung der Funktionen der unterschiedlichen Komponenten verfasst werden.

b) CARiSMA in einem OSGi-Container ausführen
Für die Umsetzung als serviceorientierter Dienst ist es notwendig, CARiSMA ohne grafische Oberfläche ausführen zu können. Um dies zu ermöglichen, wird CARiSMA im OSGi-Container Equinox [Wüt+08], einer Implementierung des OSGi-Frameworks [MVA10], ausgeführt. Hierbei wird so vorgegangen, dass zuerst die Abhängigkeiten und anschließend CARiSMA selbst in den Container eingebunden werden. Ein solches Vorgehen ist möglich, da alle Eclipse-Plugins auf OSGi-Bundles basieren und somit als solche ausgeführt werden können. Dieses Ziel ist erreicht, wenn CARiSMA innerhalb eines OSGi-Containers Anfragen entgegennehmen kann. Um dies vorführen zu können, wird ein Kommandozeilenaufruf implementiert, welcher exemplarisch einen Dummy-Check ausführen kann.

c) SOAP-Anfragen ermöglichen
Das Bundle soll SOAP-Anfragen verarbeiten können. Um dies zu erreichen, wird zusätzlich zu Equinox noch das Apache cxf-dosgi-Framework [Fou13b] eingesetzt. Dieses erweitert das Bundle um die Möglichkeit, von einem entfernten Dienst oder Nutzer als Webservice angesprochen werden zu können. Das Ziel gilt als erfüllt, wenn das Bundle eine, dem im Softwarekonzept vorgestellten Modell entsprechende, Anfrage empfangen und verarbeiten kann.

d) SingleOclCheck als Webservice anbieten
Das grundlegende Ziel dieser Arbeit, die Ausführbarkeit eines Checks via eines Webservices zu implementieren, wird exemplarisch für den SingleOclCheck, einen der in CARiSMA empfohlenen Checks, umgesetzt. Dieser benötigt mehrere Parameter und die Übergabe eines Modells. Um dies zu erreichen, müssen 2.3.1 b) und c) bereits implementiert sein. Hierzu wird neben CARiSMA auch der SingleOclCheck als OSGi-Bundle integriert. Über das laufende CARiSMA muss es möglich sein, über eine Netzwerkverbindung auf den Check zuzugreifen, ihm seine Parameter zu übergeben und die Ausgabe verarbeiten zu können.

e) Beleg der Funktionalität der Model-Type-Registry
Innerhalb CARiSMAs werden die unterschiedlichen Model-Types (derzeit UML und BPMN) mit Hilfe der Model-Type-Registry verwaltet. In dieser Aufgabe geht es darum, zu zeigen, dass auch Checks mit BPMN-Modellen möglich sind, was eine funktionierende Model-Type-Registry zeigen würde. Um dies zu erreichen, soll es möglich sein, in einer laufenden Instanz des Bundles den SingleOclCheck und den BPMN-Dummy-Check auszuführen.

### 2.3.2 Soll-Ziele

a) Analysen anbieten
Eine Analyse setzt sich innerhalb CARiSMAs aus mehreren Checks zusammen, welche das Modell auf unterschiedliche Eigenschaften untersuchen und Ergebnisse ausgeben. Hier geht es darum, zu zeigen, dass man solche auch mit Hilfe des Webservices erstellen und durchführen kann. Folglich soll zum Erreichen des Zieles eine beispielhafte Analyse bestehend aus mindestens zwei Checks durchführbar sein.

b) Automatisches Nachinstallieren neuer Checks
Das Ziel hier ist es, dass das Bundle einen neuen Check von der Update-Seite [WW13] herunterlädt, einbindet und über den Webservice erreichbar macht.
Ausnahmen wären hier Fälle, in denen Abhängigkeiten durch den neuen Check eingeführt werden. Ein solcher Fall soll nicht hier realisiert werden, sondern ist Teil des Kann-Zieles *Automatischen Nachinstallieren mit Fremdquellen*.
Hierzu soll eine regelmäßige Überprüfung der Update-Seite implementiert werden, welche neuere Versionen herunterlädt. Die Checks werden in einen beste-

henden Ordner entpackt und bei einem Neustart des OSGi-Containers automatisch geladen. Über die Check-Registry sollen diese dann auch automatisch angesprochen werden.

c) Erfüllung der CIA-Ziele
Um Sicherheitsaspekte zu beachten, sollen hier erste Absicherungen vorgenommen werden. Hierzu soll WS-Security [RB06, Kapitel 9], eine Spezifikation zur Berücksichtigung von Sicherheitsaspekten, implementiert werden, wodurch die SOAP-Anfragen signiert und verschlüsselt übertragen werden können. Die Verfügbarkeit soll durch eine, anhand von Testfällen überprüfte, Ausnahmebehandlung erreicht werden, welche die Wahrscheinlichkeit für Systemausfälle durch fehlerhafte Anfragen senkt.

d) REST-Anfragen ermöglichen
Neben SOAP-Anfragen gehören REST-Anfragen zu den Standards der Kommunikationsmethoden für Webservices [RR07] und sollen deshalb auch verarbeitet werden können. Hierzu ist eine neue Kommunikationsstrategie zu wählen, da REST mit Hilfe von eindeutigen Adressen (Uniform Resource Identificator (URI)) arbeitet. Auch kann die Zustandslosigkeit von REST-Anfragen einen bedeutenden Nachteil bei der Verwendung in diesem Kontext darstellen, da es laut dem Paradigma nicht erlaubt wäre, Informationen zwischen verschiedenen Aufrufen zu speichern (beispielsweise Ergebnisse von langlaufenden Analysen). Durchgeführt wird die Anbindung durch REST-Anfragen exemplarisch anhand des SingleOclCheckers, um die konzeptionelle Durchführung zu zeigen und einen Startpunkt für Folgearbeiten zu liefern.

e) Anwendungsbeispiel entwickeln
Um einen möglichen Anwendungsfall zu demonstrieren, wird ein SVN-Hook auf der Serverseite eines Repositories implementiert. Dieser soll bei einem Commit einer neuen Version eines sicherheitskritischen Modells dieses automatisch durch den Webservice prüfen lassen. Bei Nichtbestehen der ausgewählten Sicherheitsanforderungen wird dieser Commit abgelehnt. Dieses Vorgehen ist analog zu der Best-Practice, keinen fehlerhaften Code hochzuladen [RB06].

### 2.3.3 Kann-Ziele

a) So viele Checks wie möglich anbieten
Hier ist das Ziel, so viele der derzeitig (Stand August 2013) über die Update-Seite verfügbaren Checks anzubinden wie es möglich ist. Ein Erreichen des Zieles kann durch einen direkten Funktionsumfangvergleich mit dem Eclipse-Plugin gezeigt werden.

b) Automatische Nachinstallieren mit Fremdquellen
Hierbei handelt es sich um eine Erweiterung des Zieles *Automatisches Nachinstallieren neuer Checks*. Hier soll die Möglichkeit geschaffen werden, dass auch neue Abhängigkeiten mitinstalliert werden können. Hierbei gilt es zu klären, wie man eventuell neue Endnutzer-Lizenzvereinbarungen zugänglich machen

kann und wie man nach Quellen der neuen Abhängigkeiten suchen kann. In einer ersten Instanz sollen nur Abhängigkeiten installiert werden können, welche sich innerhalb der Plugin-Verwaltung von Eclipse befinden. Abhängigkeiten, welche von Drittthersteller-Seiten zu installieren sind, sind im Anschluss zu ermöglichen.

c) Evaluation anhand eines Live-Beispiels
Als Live-Beispiel kann eine automatische Validierung von Modellen, welche der E-Learning-Plattform Moodle übergeben werden, entwickelt werden. Das hier zu betrachtende Szenario wäre, dass im Rahmen einer Übungsaufgabe ein UML-Diagramm erstellt werden soll, welches definierte Sicherheitsanforderungen erfüllen muss. Lädt nun ein Teilnehmer seine Lösung für die Aufgabe hoch, wird diese Lösung an den CARiSMA-Webservice weitergeleitet, dort analysiert und der Teilnehmer erhält eine sofortige Rückmeldung, ohne dass ein Übungsgruppenleiter die Aufgabe manuell überprüfen muss.

# 3 Grundlagen

Im Folgenden werden die für das Verständnis der Arbeit benötigten Grundlagen erläutert. Die Grundlagen sollen soweit erklärt werden, dass die Arbeit mit ihnen verstanden werden kann. Für ein tiefergreifendes Verständnis empfiehlt sich die Lektüre der angegebenen Quellen.

## 3.1 Unified Modeling Language

Die Unified Modeling Language (UML) ist eine abstrakte Modellierungssprache, entwickelt von der Object Management Group (OMG) [BRJ06]. Mit Hilfe der UML lassen sich Spezifikationen, Konzepte und Dokumentationen von Software und Komponenten festhalten und auch grafisch visualisieren. Diese Informationen können im Folgenden zur Kommunikation zwischen Projektbeteiligten genutzt werden. Jedes UML-Modell baut auf einem UML-Metamodell auf, welches erweitert werden kann. Durch diese Architektur ermöglicht es die UML Erweiterungen, sogenannte Profile, zu entwickeln. Ein Profil kann hierbei aus Stereotypen, Constraints und Tags bestehen.

## 3.2 UMLsec

Die UML-Erweiterung UMLsec [Jür05] ermöglicht es, Sicherheitseigenschaften in UML-Modelle zu integrieren. Diese UML-Modelle können mit Hilfe der Sicherheitseigenschaften gegen bestehende Sicherheitsanforderungen getestet werden. Hierdurch ist es möglich, Sicherheitslücken in frühen Phasen der Entwicklung ausfindig zu machen.

## 3.3 CARiSMA

CARiSMA (*CompliAnce/Risk/Security-Model-Analyzer*) ist ein Sicherheitsanalysewerkzeug, welches, wie bereits erwähnt, in Kooperation von der Technischen Universität Dortmund und dem Fraunhofer-Institut für Software- und Systemtechnik ISST entwickelt wird. Es entstammt der Reimplementierung des UMLSec-Tools und wird derzeit als Eclipse-Plugin bereitgestellt. Mit Hilfe von CARiSMA lassen sich Modelle durch Analysen auf Sicherheitsanforderungen untersuchen. Hierbei stellt eine Analyse das Ausführen von einem oder mehreren Checks (vgl. Kapitel 3.3.1) auf einem gegebenen Modell dar. Diese Checks werden von einer Check-Registry (vgl. Kapitel 3.3.2) verwaltet. Ob ein gegebenes Modell von CARiSMA analysiert werden kann und ob ein Check diesen Modelltypen unterstützt, wird über die Model-Type-

Registry (vgl. Kapitel 3.3.3) verwaltet. Ergebnisse von Analysen werden als Text gespeichert und eventuell erstellte oder veränderte Dateien referenziert.

### 3.3.1 Check

Ein Check stellt eine Erweiterung CARiSMAs dar und wird mit dem Interface *CarismaCheck* implementiert. Dieses Interface stellt sicher, dass alle zum Ausführen des Checks durch CARiSMA benötigten Methodenaufrufe verfügbar sind. Innerhalb der Manifest-Datei, einer Beschreibungsdatei, welche jeder Check enthalten muss, wird genau definiert, welche Checkparameter diesem Check übergeben werden müssen, wie der eindeutige Identifikator des Checks lautet und für welche Modelltypen der Check verfügbar ist. Soll ein Check ausgeführt werden, so müssen ihm alle als nicht optional gekennzeichneten Parameter übergeben werden. Sind alle Parameter korrekt übergeben, wird der Check gegen das Modell ausgeführt und das Ergebnis als Text an CARiSMA zurückgegeben. Innerhalb des Checks erstellte oder veränderte Dateien werden ebenfalls als Referenz zurückgegeben.

#### Checkparameter

Für die Ausführung eines Checks werden verschiedene vom Check festgelegte Parameter benötigt. Diese von der Klasse CheckParameter erbenden Parameter stellen die typischen primitiven Datentypen, Strings, sowie die Parametertypen Input- und OutputFileParameter zur Verfügung. Input- und OutputFileParameter sind hierbei Verweise auf Dateien, welche zur Ausführung benötigt werden. Jeder CheckParameter verfügt über eine Beschreibung, sowie eine eindeutige Kennung, welche von CARiSMA benötigt wird, um die Parameter dem Check zu übergeben. Auch muss definiert sein, ob ein CheckParameter optional ist.

### 3.3.2 Check-Registry

Um Checks dynamisch einbinden zu können, verwaltet CARiSMA die verschiedenen Checks in einer sogenannten Check-Registry. Diese Registry durchsucht bei der Initialisierung die innerhalb ihrer OSGi-Umgebung (vgl. Kapitel 3.5) laufenden Prozesse. Wird ein Check mit der für CARiSMA-Checks typischen Namenskonvention de.carisma.check gefunden, wird dieser der Check-Registry hinzugefügt. Die Check-Registry enthält alle über den Check abrufbaren Informationen. Ist ein Check nicht in der Check-Registry vorhanden, so kann dieser nicht von der CARiSMA-Instanz ausgeführt werden.

### 3.3.3 Model-Type-Registry

Um verschiedene Modelltypen unterstützen zu können, verwaltet die Model-Type-Registry die verschiedenen Modelltypen. Hierbei stellt ein Modeltyp eine Implementierung des ModelLoader-Interfaces dar. Diese Implementierung ermöglicht das Laden von Modellen dieses Typs und die Aufbereitung, sodass Checks, welche mit diesem Modelltyp arbeiten, die Modelle weiterverarbeiten können.

## 3.4 Serviceorientierte Architekturen

Serviceorientierte Architekturen werden, wie in „SOA und WebServices“ [FZ09] beschrieben, in verschiedenen Fachpublikation abweichend definiert. Die in dieser Arbeit verwendetet Definition ist die, welche auch in „SOA und WebServices“ [FZ09] zum Einsatz kommt. Sie soll im Folgenden weiter beschrieben werden.
Unter einer serviceorientierten Architektur versteht man das Konzept, Software so zu gestalten, dass sie in möglichst verschiedenen Kontexten unter Verwendung verschiedener verteilter Dienste (Services) verwendet und lose gekoppelt zu einer großen Endanwendung aggregiert (orchestriert) werden kann. Dabei ist nicht vorgegeben, wie diese Verteilung erfolgen muss. So ist es möglich, dass eine solche serviceorientierte Architektur im Rahmen eines internen Netzwerkes, innerhalb eines Programmes oder offen über das Internet angeboten wird. Ein großer Vorteil der serviceorientierten Architektur ist, dass die unterschiedlichen Dienste in verschiedenen Programmiersprachen auf verschiedenen Systemen laufen können. Dies ermöglicht es, für jede Aufgabe die bestmöglich geeignete Technologie zu verwenden, um so eine für den Anwendungsfall optimale Lösung bieten zu können. Eine serviceorientierte Architektur weist für jede Komponente folgende, in ihren jeweiligen Kapiteln detaillierter beschriebene, Bestandteile auf. Man hat einen Dienst, welcher die Anwendungslogik enthält und eine Kommunikationsschnittstelle bereit hält, sowie ein Kommunikationsprotokoll (beispielsweise SOAP (vgl. Kapitel 3.4.2) oder REST (vgl. Kapitel 3.4.4)), welches für die Kommunikation zwischen den Webservices und zur Kommunikation mit einem Frontend genutzt werden kann. Im Kontext von Webservices mit SOAP wird auch eine Beschreibungsdatei (wie eine WSDL-Datei (vgl. Kapitel 3.4.3)), welche die Schnittstellen zu dem Dienst beschreibt, genutzt. In Kapitel 3.4.5 werden die Kommunikationsarten SOAP und REST kurz verglichen.

### 3.4.1 Webservices

Webservices stellen den Grundbaustein einer serviceorientierten Architektur dar. Sie bestehen aus einer Schnittstelle, welche von außen über Nachrichten angesprochen werden kann und einer Anwendungslogik. Die Anwendungslogik sollte hier im Idealfall eine einzelne Funktionalität bereit stellen, welche dann mit anderen Funktionalitäten im Verbund eine größere Anwendung zusammenstellt. Als Beispiel könnte hier das Überprüfen von Personendaten eines Antragsstellers innerhalb einer Kreditverwaltungsanwendung darstellen. Im Rahmen von Webservices unterscheidet man zwischen dem Webservice-Konsumenten (*Service-Client*) und dem Anbieter ( *Service-Provider*).

### 3.4.2 SOAP

SOAP[1] ist ein Netzwerkkommunikationsprotokoll, welches im Zusammenhang mit serviceorientierten Architekturen verwendet wird. Es wurde im Juni 2003 als Empfehlung von der World Wide Web Consortium (W3C) anerkannt und wird von ei-

[1] Früher Akronym für *Simple Online Access Protocol*, heute verwendet als Eigenname.

ner Arbeitsgruppe des W3C weiterentwickelt. SOAP baut auf Transportprotokollen der Transport- und Anwendungsschicht des TCP/IP-Referenzmodelles auf und wird meist mit dem Hyper Text Transfer Protocol (HTTP) genutzt. Theoretisch wäre es jedoch auch möglich, SOAP-Nachrichten aufbauend auf dem Simple Mail Transfer Protocol (SMTP) oder ähnlichen Protokollen zu versenden. SOAP-Nachrichten nutzen XML-Syntax zur Datenverarbeitung.
Eine SOAP-Nachricht enthält ein sogenanntes Envelope-Wurzel-Element. Dieses definiert in Form von Attributen die Verweise zu verwendeten Namensräumen. Des Weiteren enthält es die Kinderelemente Body und ein optionales Header-Element. Das Body-Element beinhaltet die Anfrageparameter, wie die auszuführende Operation auf Webservice-Seite, welche Parameter dieser Funktion übergeben werden sollen und weitere eventuell für das Ausführen der Anfrage benötigten Informationen. Innerhalb des Header-Elementes können Zusatzinformationen wie Nutzerdaten, Anfrageidentifikatoren oder öffentliche Schlüssel übertragen werden.

### 3.4.3 WSDL

Die WebServices-Description-Language (WSDL) ist eine in XML geschriebene Beschreibungssprache, um Netzwerkdienste wie Webservices in einem XML-Dokument zu beschreiben. Sie wurde im Juni 2007 in die Empfehlungen des W3C aufgenommen. Diese Datei enthält alle Informationen, die benötigt werden, um den Webservice anzusprechen. Im Detail setzt sich eine WSDL-Datei gemäß der in Abbildung 3.1 zu sehenden Struktur zusammen aus den folgenden Elementen:

- Types
  Dieses Element ist vorgesehen, um die Datentypen, welche im Rahmen der Kommunikation verwendet werden, zu definieren. Hierbei ist es möglich, komplexe Datentypen, welche sich aus elementaren Datentypen zusammensetzen, zu beschreiben.

- Message
  Hier werden die zur Kommunikation zwischen Service-Client und Service-Provider genutzten Nachrichtenformate abstrakt definiert.

- PortType
  Hier werden die vom Webservice nach außen zur Verfügung gestellten Methoden aufgelistet und es wird bekannt gemacht, welche Eingabeparameter eine Methode zur fehlerfreien Ausführung benötigt. Wenn eine Methode Rückgabewerte liefert, ist hier auch die Antwortnachricht, welcher der Provider an den Client sendet definiert.

- Binding
  Mit Hilfe der Bindungen wird das Protokoll, wie beispielsweise HTTP, für einen bestimmten Schnittstellentypen definiert. Auch wird festgelegt welche Nachrichtenformate (wie SOAP) möglich sein sollen.

- Service & Port
  Das Service-Element, welches auch das Port-Element enthält, wird genutzt, um Informationen wie die Adresse und die verwendeten Ports zu veröffentlichen.

Eine WSDL-Datei ermöglicht zwei verschiedene Strategien, einen Webservice zu erstellen: Contract-First und Code-First.

- Contract-First
  Bei diesem Ansatz erstellt man am Anfang des Entwicklungsprozesses eine WSDL-Datei. Diese wird im Anschluss mit Hilfe von Werkzeugen wie WSDL2Java [Fou13a] in Quellcode übersetzt. Dieser Quellcode kann dann als Bibliothek genutzt werden, um den Rest des Webservice zu implementieren.

- Code-First
  Hierbei entwickelt man zuerst die Logik und die Schnittstelle in der gewünschten Programmiersprache und erstellt aus dieser Anwendungslogik eine WSDL-Datei.

Bei Beginn der Konzeption eines Webservices gilt es also auch festzulegen, welche dieser Strategien zum Einsatz kommen soll.

### 3.4.4 REST

Das Programmierparadigma Representational State Transfer entstand im Rahmen einer Dissertation [Fie00] und ist eine Alternative zu SOAP als Kommunikationsoption für Webservices. In der Struktur verzichtet REST auf Bestandteile wie eine WSDL-Datei und ein eigenes, auf bereits verfügbaren Protokollen aufsetzendes, Protokoll (wie SOAP). Es wird das Hyper Text Transfer Protocol direkt genutzt. Bei einem Einsatz von REST wird jeder Ressource (was hier ein Datensatz oder eine Operation sein kann) ein eindeutiger Identifikator (Uniform Resource Locator, URL) zugewiesen. Mit Hilfe der URL ist die Ressource von außen erreichbar. Sie kann somit über HTTP angesprochen werden, wobei eventuell benötigte Parameter direkt

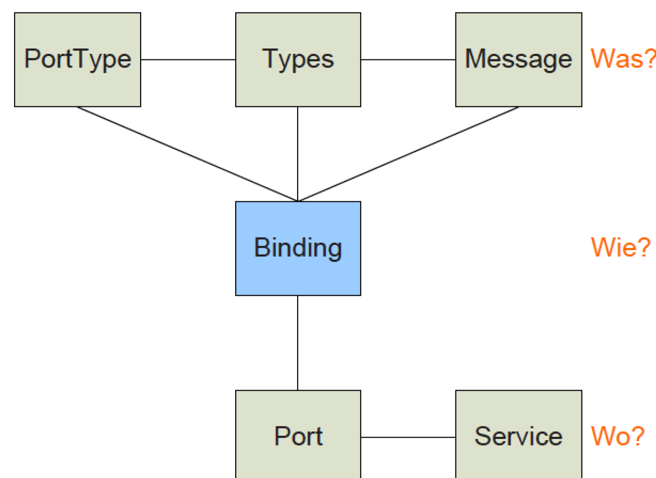


**Abbildung 3.1:** Elemente einer WSDL-Datei [FZ09, Abbildung 3.4]

innerhalb der URL oder innerhalb des Bodys der Anfrage mitübertragen werden können. Mit Hilfe der unterschiedlichen HTTP-Aufrufe GET, POST, PUT, DELETE ist es auch möglich auf einer Ressource verschiedene Operationen auszuführen. So könnte ein Aufruf per GET eine Liste mit Elementen zurückgeben, mit POST ein neues Element der Liste hinzugefügt werden, welches mit PUT aktualisiert und via DELETE gelöscht wird. Auch ist es möglich, durch das geforderte Ausgabeformat zu wählen, in welcher Repräsentation die Anfrageergebnisse ausgeliefert werden sollen. Es besteht somit die Möglichkeit, die Dokumentation in einem bestimmten Format zu hinterlegen. Zur Verschlüsselung nutzt REST den HTTPS-Standard.

### 3.4.5 Vergleich von SOAP und REST

Im Vergleich zu SOAP sieht REST keine Beschreibung der Dienste mit Hilfe einer standardisierten Datei vor. Dies erschwert es, Anwendungen für einen Webservice, der REST nutzt, zu implementieren, da es nicht sicher ist, dass man alle benötigten Informationen besitzt. Mit Hilfe der WSDL-Datei, wie sie bei SOAP zum Einsatz kommt, können Anbindungen von Webservices durch Programme generiert werden, was eine Implementierung von Service-Clients für SOAP sehr beschleunigt. Zustandsinformationen über mehrere Anfragen hinweg zu speichern ist ebenfalls nicht bei REST vorgesehen, da REST als zustandslos konzeptioniert ist.

## 3.5 OSGi

Durch die OSGi-Alliance[2] wird eine dynamische, plattformunabhängige Ausführungsplattform für Java-Programme spezifiziert und eine (nicht für den Produktiveinsatz vorgesehene) Referenzimplementierung bereitgestellt. Es wurde durch den Java Community Process als offizielles Java-Komponentenmodell anerkannt. Wie in Abbildung 3.2 zu sehen, baut die OSGi-Plattform (bunt dargestellt) auf der Java Virtual Machine (JVM) und dem Betriebssystem (Native Operating System) auf und erweitert diese mit Hilfe ihrer Ebenen um Funktionalitäten, welche eine verbesserte Modularisierung ermöglichen. Diese Modularisierung wird durch ein Komponentenmodell erreicht. Dies bedeutet, dass einzelne Teile einer Anwendung in Form von Komponenten (Services/Bundles) bereitgestellt werden. Auf diese Komponenten können (ähnlich einer serviceorientierten Architektur) andere Komponenten zugreifen und so die Gesamtheit der Komponenten zu einer großen Endanwendung aggregiert werden. Die für diese Arbeit relevanten OSGi-Ebenen sollen hier kurz vorgestellt werden:

- Services Layer
  Bei der Services-Ebene handelt es sich um eine stark in der Life-Cycle-Ebene verankerte Ebene, welche das dynamische Laden von Services innerhalb von Bundles verwaltet. Diese Services werden mit Hilfe der von ihnen bereitgestellten Service Reference innerhalb einer zentralen Service Registry verwaltet. Ein Bundle kann mit Hilfe von Services eine Implementierung eines Interfaces in

[2] Früher Open Services Gateway initiative, heute verwendet als Eigenname.

Form eines Services fordern, ohne genauere Spezifikationen anzugeben. Im Detail wird dies in Kapitel 3.5.3 erklärt.

- Modules Layer
  Die Modules-Ebene ist mit der Verwaltung der verschiedenen Bundles beauftragt. Hierbei werden Bundles unter anderem auf ihre Lauffähigkeit hin überprüft und analysiert welche weiteren Bundles diese als Abhängigkeit benötigen. Außerdem dient die Modules-Ebene als Klassenlader.

- *Bundles*
  *Bundles* stellen die Möglichkeit dar, lauffähige Java-Applikationen in einer OSGi-Umgebung zur Verfügung zu stellen. Sie werden als Java-Archive-Dateien (JAR) exportiert, welche in sich ebenfalls wieder JAR-Dateien enthalten können, die als Ressourcen oder Klassen zugänglich gemacht werden können. Genauer bestehen sie aus den benötigten Ressourcen, einer Manifest-Datei, welche alle Abhängigkeiten und eine Liste der durch das *Bundle* zur Verfügung gestellten Bibliotheken enthält und einem optionalem Dokumentationsordner.

### 3.5.1 Eclipse Equinox

Eclipse Equinox ist eine quelloffene Implementierung der OSGi-Spezifikation durch die Eclipse Foundation [MVA10]. Diese Implementierung wird auch als Plattform für die von der Eclipse Foundation entwickelte Entwicklungsumgebung Eclipse genutzt. Dies ermöglicht es, Eclipse-Plugins innerhalb der Eclipse Equinox Laufzeitumgebung als *Bundles* auszuführen. Hierzu müssen alle für das Ausführen des Plugins benötigten Abhängigkeiten installiert und gestartet sein.

### 3.5.2 Distributed OSGi

Bei Distributed OSGi handelt es sich um ein Subprojekt innerhalb der Apache CXF-Arbeitsgruppe, welches die Distribution-Provider-Komponente aus der Compendium Spezifikation implementiert. Es verfolgt das Ziel, es zu erleichtern, Bundles als Webservices zur Verfügung zu stellen. Hierfür wird die Remote-Services-Funktionalität mit Hilfe von SOAP und einer WSDL-Datei genutzt. Hierbei wird

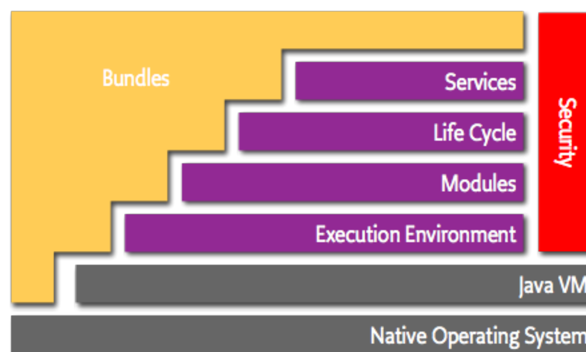


**Abbildung 3.2:** Struktur der OSGi-Plattform [All13]

eine Schnittstelle bereitgestellt, mit welcher man den Webservice konfigurieren kann. Auch wird ein Jetty-Server gestartet, welcher diese Webservices von außen verfügbar macht.

### 3.5.3 Declerative OSGi Services

Bei den Declerative OSGi Services handelt es sich um eine Möglichkeit, Services innerhalb eines OSGi-Frameworks zu definieren. Mit Hilfe von Services kann man Funktionalitäten in Bundles dynamisch integrieren. Die Voraussetzung ist, dass der Service das Interface implementiert, welches innerhalb des Bundles benötigt wird. Wird dieses Bundle ausgeführt, wird die Service-Registry nach einem passenden Eintrag durchsucht. Wird einer gefunden, wird dieser zurückgegeben. Das Bundle wird dann die Funktionalität des Services nutzen.

## 3.6 SQLite-Datenbank

SQLite ist eine eingebettete relationale Datenbank, welche im Jahr 2000 erschienen ist und seitdem besonders, bedingt durch die hohe Portierbarkeit und Effizienz, im mobilen Bereich zum Einsatz kommt [Owe06]. Die Stärke des Systems ist, dass keine dedizierte Verwaltungssoftware benötigt wird, wie es bei Alternativen wie MySQL [DuB13] der Fall ist, welche einen eigenständigen Serverprozess benötigen. Dennoch erlaubt sie einen Großteil der SQL-Befehle, wie sie in dem 1992 von der ISO verabschiedeten Standard SQL-92 [ISO92] festgelegt wurden, auszuführen. Es fehlen die Befehle zur Rechteverwaltung auf einer Datenbank. Der gesamte Inhalt einer Datenbank wird innerhalb einer Datei gespeichert und verwaltet. Bibliotheken zum Arbeiten mit SQLite sind in den meisten der gängigen Programmiersprachen verfügbar.

## 3.7 Verwendete Werkzeuge

Im Rahmen dieser Arbeit werden verschiedene Werkzeuge benutzt, welche für die Realisierung essentiell sind. Diese werden hier vorgestellt.

### 3.7.1 SoapUI

Zum Testen der SOAP-Aufrufe wurde das von SMARTBEAR entwickelte Werkzeug SoapUI [Sof13] in der kostenlosen Version verwendet. Dieses ermöglicht es, mit einer gegebenen WSDL-Datei Testaufrufe zu generieren. Diese Testaufrufe können im Anschluss mit den gewünschten Werten und unter Angabe weiterer Informationen, wie bestimmten Header-Informationen, an den Server gesendet werden. Auch wurden hiermit die in Kapitel 6.1 beschriebenen Systemtests durchgeführt.

### 3.7.2 RESTClient

Für das Testen der REST-Aufrufe wurde die Erweiterung RESTClient des Webbrowsers Firefox [Fou13c] verwendet. Diese stellt eine grafische Oberfläche zur Verfügung, mit welcher sich einfach REST-Aufrufe erstellen lassen.

# 4 Konzept

Hier wird das Konzept der Arbeit vorgestellt. Die Vorstellung des Konzepts soll das weitere Vorgehen in der Implementierung vorab erkennbar machen, um damit das Verständnis dieses Vorgehens erleichtern. Zuerst werden in Kapitel 4.1 die Anwendungsfälle identifiziert und erläutert. Deren Abläufe werden in Kapitel 4.2 mit Hilfe von Aktivitätsdiagrammen dargestellt. Kapitel 4.3 befasst sich mit den Kommunikationskonzepten. In Kapitel 4.4 wird das Klassendiagramm und die darin enthaltenen Klassen vorgestellt. Die Architektur der Softwarelösung wird in Kapitel 4.5 festgelegt. Die konzeptionellen Sicherheitsvorkehrungen werden in Kapitel 4.6 vorgestellt. Da die Entwicklungsrichtlinie CARiSMAs englischsprachigen Quellcode fordert und die gegebenen Modelle als Vorstufe des Quellcodes verstanden werden, wurde sich dazu entschieden, die Modelle ebenfalls englischsprachig zu verfassen.

## 4.1 Anwendungsfalldiagramm

Bei der Erstellung des Konzeptes haben sich die Anwendungsfälle, wie sie in Abbildung 4.1 zu sehen sind, als sinnvoll erwiesen. Sie sollen im Folgenden kurz vorgestellt und im Anschluss in Form von Tabellen detailliert beschrieben werden.
Ein Nutzer muss in der Lage sein, eine Analyse auf einem Modell auszuführen. Dies stellt die grundlegende Funktion des Webservices dar (Run Analysis, vgl. Tabelle 4.1). Des Weiteren muss eine Möglichkeit bestehen, sich die verfügbaren Checks mit den für ihre Ausführung benötigten Informationen anzeigen zu lassen (Get Available Checks, vgl. Tabelle 4.2). Um Missbrauch zu vermeiden, muss es möglich sein, dass sich Nutzer authentifizieren (Authenticate User vgl. Tabelle 4.3). Dies erzeugt den Bedarf Nutzer anlegen zu können (Create User, vgl. Tabelle 4.4). Zur Verwaltung von Nutzern ist es auch nötig, diese löschen (Delete User, vgl. Tabelle 4.7), verändern (Update User, vgl. Tabelle 4.6) und auch die dem Nutzer zugehörigen Informationen anzeigen zu können (Read User, vgl. Tabelle 4.5). Um Analysen auf Modellen ausführen und Eingabe- und Ausgabeparameter in Form von Dateien realisieren zu können ist eine Verwaltung von Dateien mit den Funktionen Anlegen (Create File, vgl. Tabelle 4.8), Anzeigen (Read File, vgl. Tabelle 4.9), Verändern (Update File, vgl. Tabelle 4.10) und Löschen (Delete File, vgl. Tabelle 4.11) notwendig. Um die wiederholte Beschreibung gleicher Abläufe zu vermeiden, wurde der Anwendungsfall Authenticate User in der Beschreibung anderer Anwendungsfälle als Include eingebunden.

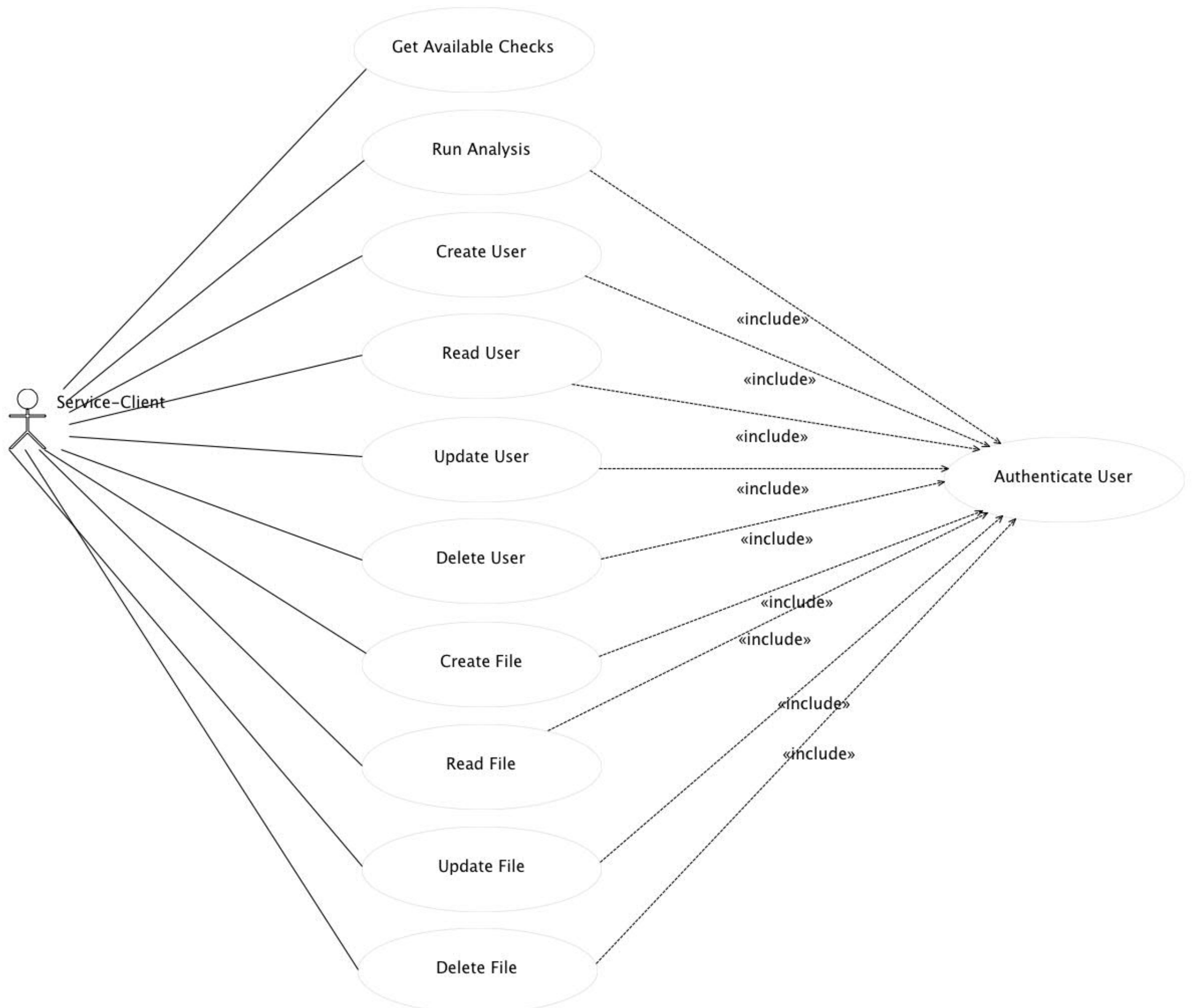


**Abbildung 4.1:** Anwendungsfalldiagramm

| Name | Run Analysis |
|---|---|
| Kurzbeschreibung | Ausführen einer Analyse auf einem Modell |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Run-Analysis-Anfrage an den Webservice. |
| Ergebnis(se) | Analyseergebnisse |
| Eingehende Daten | Liste der gewünschten Checks mit allen benötigten Parametern, Identifikationsnummer des Modells |
| Vorbedingungen | Das zu überprüfende Modell ist hochgeladen und die geforderten Checks sind installiert und aktiv. |
| Nachbedingungen | Die Ergebnisse der Checks wurden in eine Datei geschrieben, welche innerhalb der Datenbank hinterlegt wurde. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Überprüfung der eingehenden Daten.<br>3. Ausführen der Analyse:<br>Die Analyse wird in einer CARiSMA-Instanz ausgeführt und die Ergebnisse werden in dem Dateisystem gespeichert und der Datenbank hinzugefügt.<br>4. Rückgabe der Antwort:<br>Der Service-Client erhält eine Antwort mit Referenzen zu den mit der Analyse verknüpften Dateien und eine Bestätigung, ob die Ausführung erfolgreich war. |

**Tabelle 4.1:** Beschreibung des Anwendungsfalles „Run Analysis“

| Name | Get Available Checks |
|---|---|
| Kurzbeschreibung | Auflisten der Informationen, welche über die Checks vorliegen |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Get-Available-Checks-Anfrage an den Webservice. |
| Ergebnis(se) | Liste mit verfügbaren Checks und den Informationen |
| Eingehende Daten | Es werden keine eingehenden Daten benötigt. |
| Vorbedingungen | Der Webservice ist verfügbar. |
| Nachbedingungen | Liste der Checks wurde an den Anfragenden übermittelt. |
| Essenzielle Schritte | 1. Auflisten verfügbarer Checks und Informationen:<br>CARiSMA gibt eine Liste mit den verfügbaren Checks zurück, welche aufgearbeitet wird.<br>2. Rückgabe der Antwort:<br>Die Liste wird in ein übertragbares Format umgewandelt und an den Service-Client zurückgegeben. |

**Tabelle 4.2:** Beschreibung des Anwendungsfalles „Get Available Checks“

| Name | Authenticate User |
|---|---|
| Kurzbeschreibung | Authentifizierung des Service-Client |
| Akteure | Service-Client |
| Auslöser | Der Webservice erhält einen Aufruf zu einer Operation, welche nur authentifiziert zur Verfügung stehen soll. |
| Ergebnis(se) | Authentifizierung |
| Eingehende Daten | Nutzername und Passwort des Service-Clients |
| Vorbedingungen | Service-Client ist der Datenbank bekannt. |
| Nachbedingungen | Die Authentizität des Nutzers wurde nachgewiesen. |
| Essenzielle Schritte | 1. Prüfe Datenbank:<br>Der Nutzername muss sich in der Datenbank befinden.<br>2. Prüfe Passwort:<br>Es wird eine Operation zur Überprüfung des Passwortes aufgerufen.<br>3. Status des Service-Clients ändern:<br>Die Authentifizierung des Nutzers wurde erfolgreich durchgeführt. |

**Tabelle 4.3:** Beschreibung des Anwendungsfalles „Authenticate User“

| Name | Create User |
|---|---|
| Kurzbeschreibung | Erstellen eines neuen Nutzers |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Create-User-Anfrage an den Webservice. |
| Ergebnis(se) | Nutzer erstellt |
| Eingehende Daten | Nutzername und Passwort des neuen Nutzers |
| Vorbedingungen | Der Nutzername ist noch nicht benutzt. |
| Nachbedingungen | Der Nutzer wurde erfolgreich angelegt. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Prüfe Nutzername:<br>Der Nutzername muss eindeutig sein und darf nicht bereits in der Datenbank vorhanden sein.<br>3. Lege neuen Nutzer an:<br>Es wird ein neuer Datenbankeintrag in der Nutzerdatenbank angelegt.<br>4. Rückgabe der Antwort:<br>Es wird eine Antwort an den Service-Client gesendet, welche angibt, dass das Anlegen erfolgreich war. |

**Tabelle 4.4:** Beschreibung des Anwendungsfalles „Create User“

| Name | Read User |
|---|---|
| Kurzbeschreibung | Auflisten aller Informationen über den Service-Client |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Read-User-Anfrage an den Webservice. |
| Ergebnis(se) | Informationen dargestellt |
| Eingehende Daten | Nutzernamedes Nutzers |
| Vorbedingungen | Der Nutzer ist dem System bekannt. |
| Nachbedingungen | - |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Hole Informationen aus Datenbank:<br>Aus der Datenbank müssen die dem Nutzer zugehörigen Dateien ausgelesen werden.<br>3. Rückgabe der Antwort:<br>Die Liste mit den Informationen wird weiterverarbeitet und an den Service-Client zurückgegeben. |

**Tabelle 4.5:** Beschreibung des Anwendungsfalles „Read User“

| Name | Update User |
|---|---|
| Kurzbeschreibung | Ändern des Passwortes des Nutzers |
| Akteure | Service Client |
| Auslöser | Der Service-Client schickt eine Update-User-Anfrage an den Webservice. |
| Ergebnis(se) | Passwort geändert |
| Eingehende Daten | Nutzername des Nutzers, neues Passwort |
| Vorbedingungen | Der Nutzer ist dem System bekannt. |
| Nachbedingungen | Das Passwort wurde erfolgreich aktualisiert. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Ändere Passwort:<br>Der Datenbankeintrag des Nutzers wird geändert.<br>3. Rückgabe der Antwort:<br>Ob das Passwort erfolgreich geändert wurde, wird an den Service-Client gesendet. |

**Tabelle 4.6:** Beschreibung des Anwendungsfalles „Update User“

| Name | Delete User |
|---|---|
| Kurzbeschreibung | Löschen eines Nutzers |
| Akteure | Service Client |
| Auslöser | Der Service-Client schickt eine Delete-User-Anfrage an den Webservice. |
| Ergebnis(se) | Nutzer gelöscht |
| Eingehende Daten | Nutzername des zu löschenden Nutzers |
| Vorbedingungen | Der Nutzer ist dem System bekannt. |
| Nachbedingungen | Die Informationen und Dateien des Nutzers wurden gelöscht. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Hole Informationen:<br>Alle mit den Nutzer verknüpften Dateien müssen aufgelistet werden.<br>3. Lösche alle Dateien:<br>Die zuvor aufgelisteten Dateien werden aus dem System gelöscht.<br>4. Lösche Nutzer:<br>Der Datenbankeintrag des Nutzers wird entfernt.<br>5. Rückgabe der Antwort:<br>Information, ob der Nutzer erfolgreich gelöscht wurde, wird an den Service-Client gesendet. |

**Tabelle 4.7:** Beschreibung des Anwendungsfalles „Delete User“

| Name | Create File |
|---|---|
| Kurzbeschreibung | Erstellen einer Datei |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Create-File-Anfrage an den Webservice. |
| Ergebnis(se) | Datei |
| Eingehende Daten | Nutzerdaten, Datei, Informationen zur Datei |
| Vorbedingungen | Der Nutzer muss dem System bekannt sein. |
| Nachbedingungen | Die Datei wurde erfolgreich angelegt. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Schreibe Datei ins Dateisystem:<br>Die übergebene Datei wird in das Dateisystem geschrieben und ein Datenbankeintrag angelegt.<br>3. Rückgabe der Antwort:<br>Eine Referenz der Datei wird an den Service-Client gesendet. |

**Tabelle 4.8:** Beschreibung des Anwendungsfalles „Create File“

| Name | Read File |
|---|---|
| Kurzbeschreibung | Herunterladen einer Datei |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Read-File-Anfrage an den Webservice. |
| Ergebnis(se) | Dateiinhalt |
| Eingehende Daten | Eindeutiger Identifikator der Datei |
| Vorbedingungen | Die Datei ist bei dem Service-Provider vorhanden. |
| Nachbedingungen | - |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Prüfe, ob Datei vorhanden ist.<br>3. Prüfe, ob Nutzer berechtigt ist Datei zu lesen.<br>4. Rückgabe der Antwort:<br>Dateiname und Inhalt der Datei werden an den Service-Client gesendet. |

**Tabelle 4.9:** Beschreibung des Anwendungsfalles „Read File“

| Name | Update File |
|---|---|
| Kurzbeschreibung | Aktualisieren einer bereits vorhandenen Datei |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Update-File-Anfrage an den Webservice. |
| Ergebnis(se) | Geänderte Datei |
| Eingehende Daten | Identifikator der Datei, neuer Inhalt der Datei |
| Vorbedingungen | Die zu aktualisierende Datei liegt auf Provider-Seite vor. |
| Nachbedingungen | Die Datei wurde aktualisiert. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Prüfe, ob Datei existiert.<br>3. Prüfe, ob Nutzer Zugriffsrecht auf Datei besitzt.<br>4. Schreibe neuen Inhalt in die Datei.<br>5. Rückgabe der Antwort:<br>Eine Referenz zu der Datei wird an den Service-Client gesendet. |

**Tabelle 4.10:** Beschreibung des Anwendungsfalles „Update File“

| Name | Delete File |
|---|---|
| Kurzbeschreibung | Löschen einer vorhandenen Datei |
| Akteure | Service-Client |
| Auslöser | Der Service-Client schickt eine Delete-File-Anfrage an den Webservice. |
| Ergebnis(se) | Datei gelöscht |
| Eingehende Daten | Identifikationsnummer der Datei |
| Vorbedingungen | Datei ist vorhanden. |
| Nachbedingungen | Datei wurde entfernt. |
| Essenzielle Schritte | 1. Include: Authenticate User<br>2. Prüfe, ob die Datei vorhanden ist.<br>3. Prüfe, ob Nutzer Zugriffsrecht auf Datei besitzt.<br>4. Lösche Datei:<br>Die Datei wird aus dem Dateisystem und den zugehörigen Datenbankeintrag gelöscht.<br>5. Rückgabe der Antwort:<br>Ein Antwort die angibt, ob die Datei gelöscht wurde wird an den Service-Client gesendet. |

**Tabelle 4.11:** Beschreibung des Anwendungsfalles „Delete File“

## 4.2 Aktivitätsdiagramme

Aus den Anwendungsfällen, welche in Kapitel 4.1 aus den Anforderungen extrahiert wurden, werden hier die Aktivitätsdiagramme abgeleitet. Diese dienen dazu, die Abläufe der unterschiedlichen Anwendungsfälle weiter zu definieren. Jedes Diagramm wird außerdem in Prosatext erläutert.

### 4.2.1 Run Analysis

Das Aktivitätsdiagramm des Anwendungsfalles „Run Analysis“ ist in Abbildung 4.2 zu sehen. Zu Beginn der Ausführung wird der Nutzer authentifiziert (beschrieben in Abschnitt 4.2.3). Wurde dies erfolgreich abgeschlossen, wird überprüft, ob das zu analysierende Modell auf der Provider-Seite vorhanden ist und ob der Nutzer die benötigten Zugriffsrechte besitzt. Ist dies nicht der Fall, wird eine Fehlermeldung an den Service-Client zurückgegeben und die Ausführung abgeschlossen. Wurde das Modell erfolgreich geladen, wird durch die in der Anfrage übertragene Liste der geforderten Checks iteriert bis alle Checks zu einer Analyse hinzugefügt wurden. Hierbei wird überprüft, ob der hinzuzufügende Check verfügbar ist. Steht der Check nicht zur Verfügung, wird die Ausführung abgebrochen und eine Fehlernachricht an den Service-Client gesendet. Innerhalb dieser Schleife, werden alle für den Check übergebenen CheckParameter überprüft. Wird ein übertragener Checkparameter nicht vom Check gefordert, so wird die Ausführung mit einer Fehlernachricht beendet. Sonst wird der Checkparameter der Analyse übergeben. Sind alle Checks der Analyse hinzugefügt, wird diese ausgeführt. Das Ergebnis der Analyse wird in einer Datei gespeichert. Ein Verweis zu dieser Datei, Verweise zu möglichen weiteren Ausgabedateien und ein Wahrheitswert, ob die Analyse erfolgreich ausgeführt wurde, werden an den Service-Client zurückgegeben.

### 4.2.2 Get Available Checks

Zu Beginn wird eine Liste mit allen durch CARiSMA verfügbaren Checks abgerufen. Diese Liste wird in ein für die Antwort passendes Format umgewandelt und an den Service-Client zurückgegeben. Das Aktivitätsdiagramm dieses Anwendungsfalles ist in Abbildung 4.3 zu sehen.

### 4.2.3 Authenticate User

Zuerst wird überprüft, ob der übergebene Nutzername existiert. Ist dies nicht der Fall, wird die weitere Ausführung des Programms beendet und dem Service-Client eine Fehlermeldung gesendet. Ist der Nutzername vorhanden, wird das diesem zugeordnete Passwort mit dem übertragenen verglichen. Sind diese nicht identisch, wird wieder die weitere Ausführung des Programmes beendet und eine Fehlermeldung an den Service-Client gesendet. Ist das übertragene identisch mit dem gespeicherten, wird der Nutzer authentifiziert und das Unterprogramm wird beendet. In diesem Fall gilt der Anwendungsfall als erfolgreich ausgeführt. Das entsprechende Modell ist in Abbildung 4.4 zu sehen.

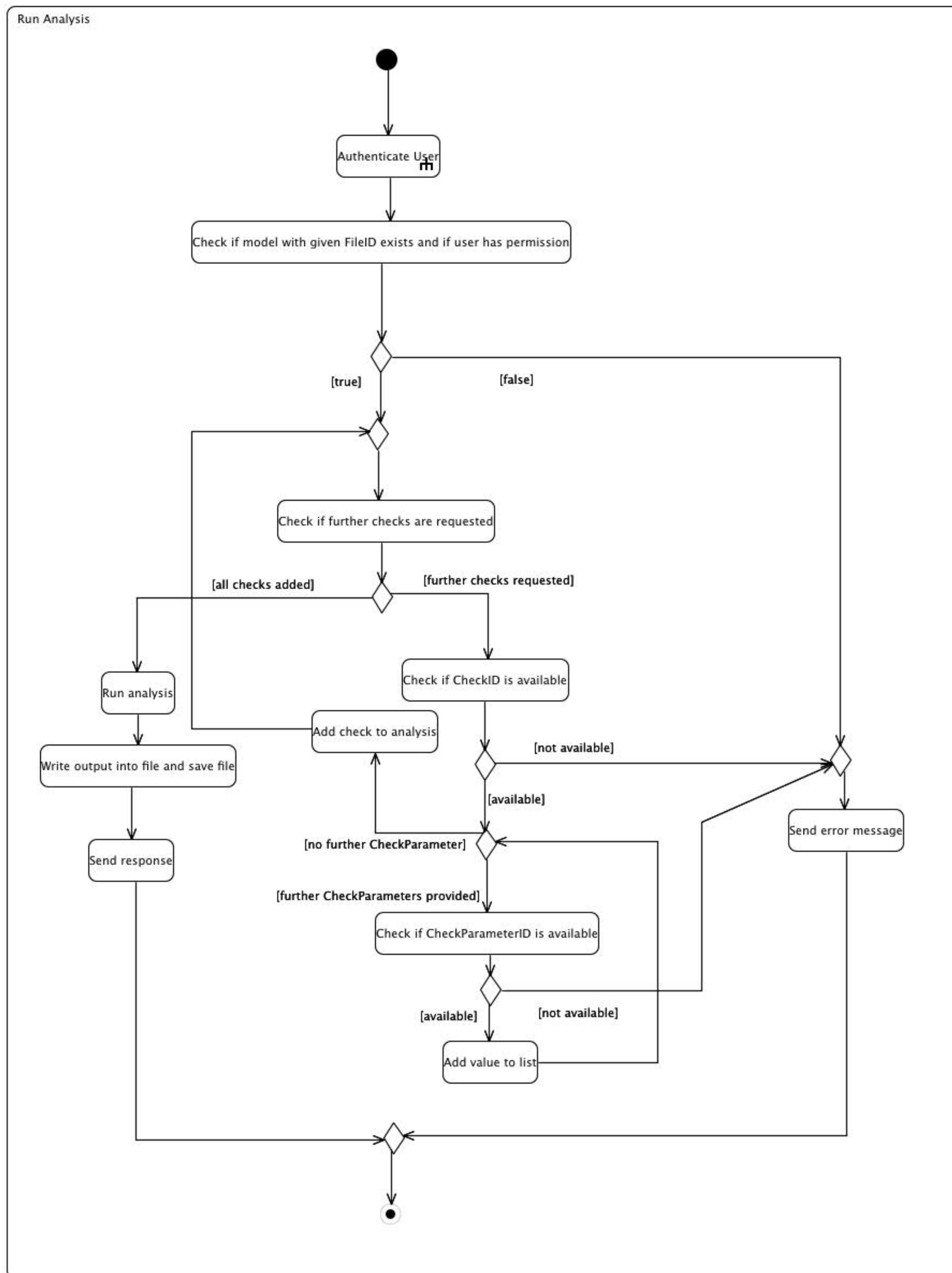


**Abbildung 4.2:** Aktivitätsdiagramm des Anwendungsfalles „Run Analysis“

### 4.2.4 Create User

Soll ein neuer Nutzer angelegt werden, wird zuerst der Service-Client authentifiziert. Ist der Service-Client erfolgreich authentifiziert, wird überprüft, ob der anzulegende Nutzername bereits verwendet wird. Ist dies der Fall, wird eine Fehlermeldung an den Service-Client gesendet und das Programm beendet. Wenn der Nutzername verfügbar ist, wird ein Datenbankeintrag für den Nutzer angelegt. Zum Abschluss

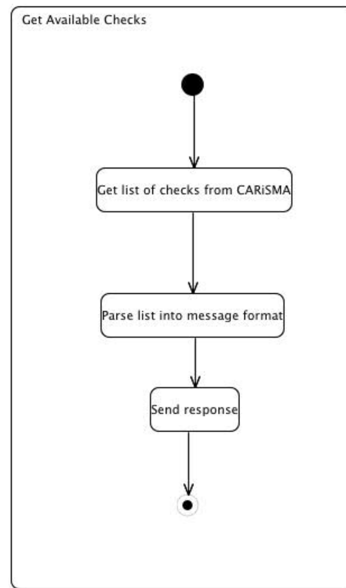


**Abbildung 4.3:** Aktivitätsdiagramm des Anwendungsfalles „Get Available Checks“

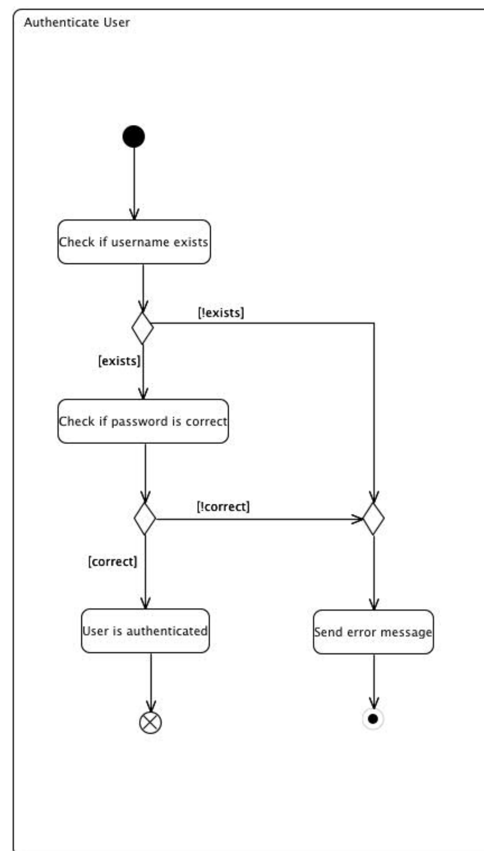


**Abbildung 4.4:** Aktivitätsdiagramm des Anwendungsfalles „Authenticate User“

wird der Service-Client über das erfolgreiche Ausführen der Operation informiert. Das Aktivitätsdiagramm dieses Anwendungsfalles ist in Abbildung 4.5 zu sehen.

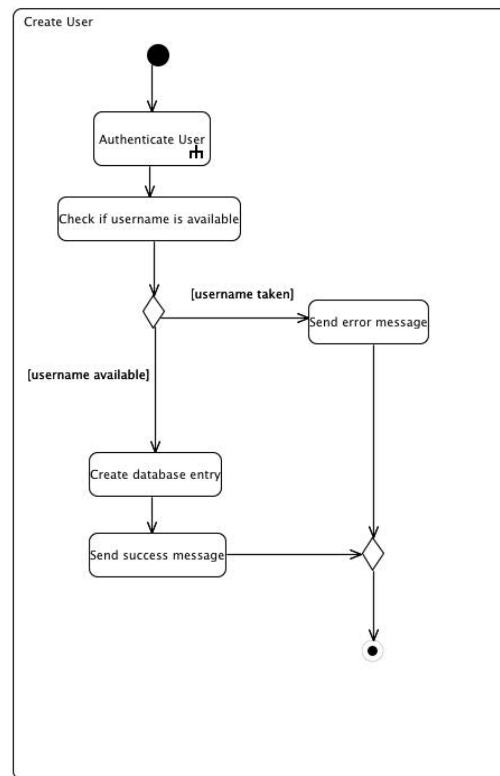


**Abbildung 4.5:** Aktivitätsdiagramm des Anwendungsfalles „Create User“

### 4.2.5 Read User

Um alle Informationen über einen Nutzer abzurufen, muss sich dieser zuerst authentifizieren. Ist dies erfolgreich geschehen, wird eine Liste mit Referenzen zu allen dem Nutzer zuzuordnenden Dateien erstellt. Diese Liste wird im Anschluss so weiterverarbeitet, dass sie dem Service-Client zurückgesendet werden kann und diesem zurückgesendet. Das entsprechende Modell ist in Abbildung 4.6 zu sehen.

### 4.2.6 Update User

Hat sich der Service-Client erfolgreich authentifiziert wird das gespeicherte Passwort durch das neue Passwort ersetzt, und eine Erfolgsmeldung an den Service-Client gesendet (vgl. Abbildung 4.7).

### 4.2.7 Delete User

Hat sich der zu löschende Nutzer erfolgreich authentifiziert wird eine Liste mit allen ihm zugeordneten Dateien erstellt. Diese Liste wird durchiteriert und die Dateien gelöscht. Sind alle Dateien gelöscht, wird der Nutzer aus der Datenbank gelöscht und eine Erfolgsnachricht an den Service-Client übertragen. Das Aktivitätsdiagramm dieses Anwendungsfalles ist in Abbildung 4.8 zu sehen.

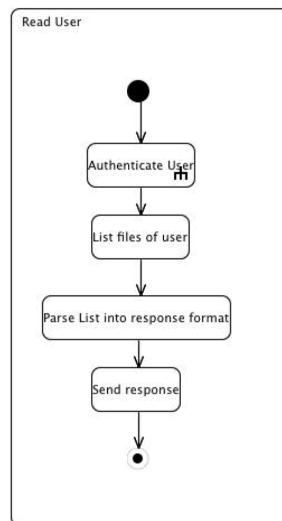


**Abbildung 4.6:** Aktivitätsdiagramm des Anwendungsfalles „Read User“

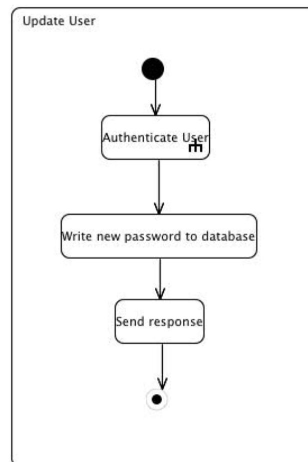


**Abbildung 4.7:** Aktivitätsdiagramm des Anwendungsfalles „Update User“

### 4.2.8 Create File

Um eine Datei anzulegen, muss sich der Service-Client erfolgreich authentifizieren. Ist dies geschehen, wird der übertragene Inhalt der Datei in eine Datei innerhalb des Dateisystems des Service-Providers gespeichert. Ein Verweis auf diese Datei wird mit einer eindeutigen Identifikationsnummer innerhalb eines Datenbankeintrages gespeichert. Eine Referenz zu der Datei wird dem Service-Client übertragen. Das Diagramm dieser Aktivität ist in Abbildung 4.9 zu sehen.

### 4.2.9 Read File

Um eine Datei auslesen zu können, muss sich der Service-Client authentifizieren. Ist dies erfolgreich geschehen, wird überprüft, ob die übergebene Identifikationsnummer einer Datei zugeordnet werden kann. Ist dies nicht möglich, wird eine Fehlermeldung

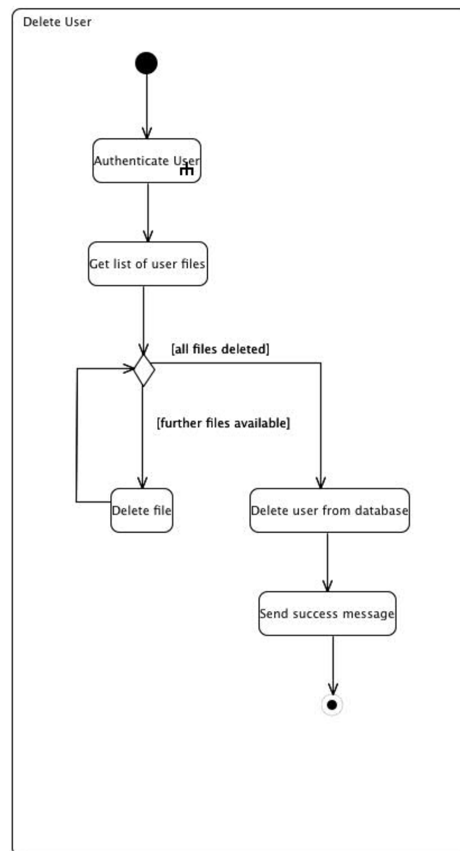


**Abbildung 4.8:** Aktivitätsdiagramm des Anwendungsfalles „Delete User“

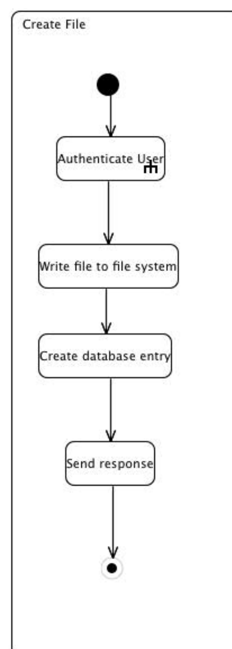


**Abbildung 4.9:** Aktivitätsdiagramm des Anwendungsfalles „Create File“

an den Service-Client übertragen und die weitere Ausführung des Programmes beendet. Wurde die Identifikationsnummer einer Datei zugeordnet, wird überprüft, ob der Service-Client die Berechtigung besitzt, diese Datei zu lesen. Besitzt er die Berechtigung nicht, wird ihm eine Fehlermeldung übertragen und die Ausführung beendet. Hat der Service-Client die Berechtigung, wird der Inhalt der Datei ausgelesen und dieser mit dem Dateinamen an den Service-Client übertragen. Das entsprechende Diagramm ist in Abbildung 4.10 zu sehen.

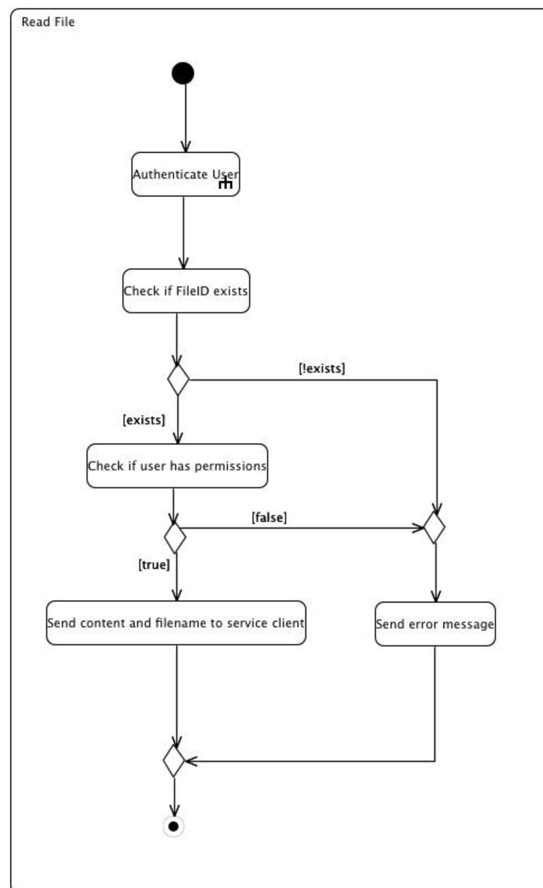


**Abbildung 4.10:** Aktivitätsdiagramm des Anwendungsfalles „Read File“

### 4.2.10 Update File

Um eine Datei aktualisieren zu können, muss sich der Service-Client zuerst authentifizieren. Ist dies erfolgreich geschehen, wird überprüft, ob die übergebene Identifikationsnummer einer Datei zugeordnet werden kann. Ist dies nicht möglich, wird eine Fehlermeldung an den Service-Client übertragen und die weitere Ausführung des Programmes beendet. Wurde die Identifikationsnummer einer Datei zugeordnet, wird überprüft, ob der Service-Client die Berechtigung besitzt, diese Datei zu bearbeiten. Besitzt er die Berechtigung nicht, wird ihm eine Fehlermeldung übertragen und die Ausführung beendet. Hat der Service-Client die Berechtigung, wird der neue Inhalt in die Datei geschrieben und eine Antwortnachricht mit einer Refe-

renz zu der Datei gesendet. Das Aktivitätsdiagramm dieses Anwendungsfalles ist in Abbildung 4.11 zu sehen.

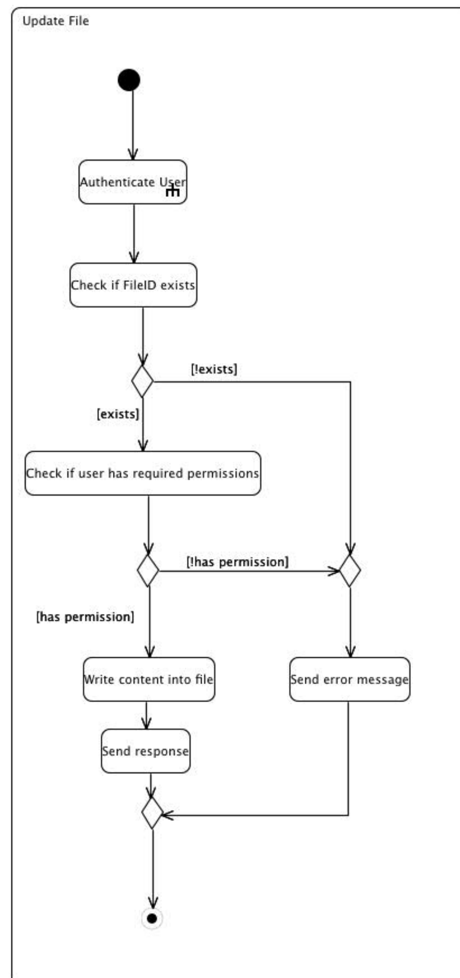


**Abbildung 4.11:** Aktivitätsdiagramm des Anwendungsfalles „Update File“

## 4.2.11 Delete File

Um eine Datei löschen zu können, muss sich der Service-Client zuerst authentifizieren. Ist dies erfolgreich geschehen, wird überprüft, ob die übergebene Identifikationsnummer einer Datei zugeordnet werden kann. Ist dies nicht möglich, wird eine Fehlermeldung an den Service-Client übertragen und die weitere Ausführung des Programmes beendet. Wurde die Identifikationsnummer einer Datei zugeordnet, wird überprüft, ob der Service-Client die Berechtigung besitzt, diese Datei zu löschen. Besitzt er die Berechtigung nicht, wird ihm eine Fehlermeldung übertragen und die Ausführung beendet. Hat der Service-Client die Berechtigung, wird die Datei gelöscht und eine Erfolgsnachricht an den Service-Client versendet. Das entsprechende Diagramm ist in Abbildung 4.12 zu sehen.

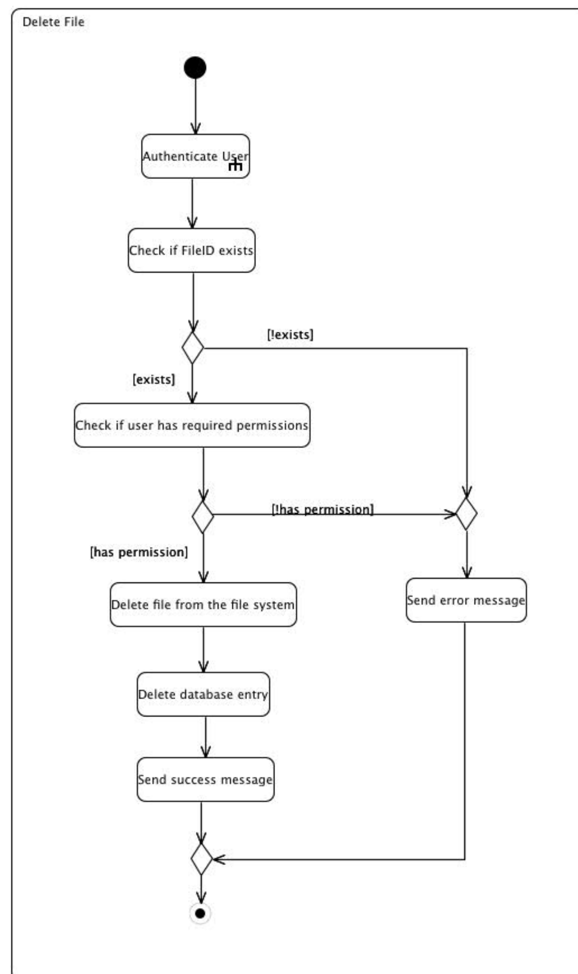


**Abbildung 4.12:** Aktivitätsdiagramm des Anwendungsfalles „Delete File“

## 4.3 Kommunikation

Die in dieser Arbeit verwendeten Kommunikationsarten sind durch ihre verschiedenen Ansätze in ihrem Aufbau sehr unterschiedlich. Im Folgenden soll die WSDL-Datei, aus welcher sich die SOAP-Aufrufe ableiten, vorgestellt werden. Im Anschluss das Format der REST-Aufrufe.

### 4.3.1 WSDL-Datei

In dieser Arbeit wurde sich für das in Kapitel 3.4.3 erwähnte Contract-First Vorgehen entschieden, dies bedeutet, dass vor der Implementierung des Quellcodes, eine WSDL-Datei erstellt wurde. Hierzu wurde zu jedem Anwendungsfall (mit Ausnahme des Authenticate-User-Anwendungsfalles), eine Operation definiert. Bei der Erstellung der Nachrichten und der in diesen verwendeten Typen wurde sich auch an die in den Anwendungsfällen benötigten eingehenden Daten gehalten. Bei der Erstellung der zu CARiSMA analogen Datentypen wie beispielsweise CheckParameter,

wurde sich an der Struktur der in CARiSMA genutzten Datentypen gehalten. Bei der Benennung der Nachrichten wurde das Muster: Operationsname für eingehende Nachrichten und Operationsname mit angehängtem *Response* für Antworten genutzt. Beispielsweise ist bei der Operation runAnalysis *runAnalysis* die eingehende Nachricht und *runAnalysisResponse* die Antwort.
Die Operationen sollen hier im Detail vorgestellt werden. Kursiv in Klammern finden sich die jeweiligen Typenbezeichner und Variablennamen wie sie in der WSDL-Datei zu finden sind. Bezeichner von Datentypen beginnen hierbei mit einem Großbuchstaben, Variablennamen mit einen Kleinbuchstaben.

- **runAnalysis**
  runAnalysis bekommt als Eingabe die Identifikationsnummer des zu analysierenden Modelles (*fileID)* und eine Liste der durchzuführenden Checks (*check)*. Hierbei gehört zu jedem Check, die eindeutige Identifikationskennung, sowie Listen mit den jeweiligen benötigten Check Parametern (*checkParameter, integerCheckParameter, stringCheckParameter...*). Jede Art von Check Parametern ist hier durch einen eigenen Datentypen wiedergegeben.
  Als Antwort wird eine Referenz (*FileReference*) auf die durch die Analyse erstellte Ergebnisdatei (*resultFileRef*), sowie eine Liste mit weiteren Referenzen zu in der Analyse erstellten Ausgabedateien (*outputFileRef*) und ein Wahrheitswert, welcher angibt, ob die Analyse erfolgreich ausgeführt wurde (*succesful*), zurückgegeben.

- **getAvailableChecks**
  Die Operation benötigt keine Eingabeparameter. Zurück wird eine Liste mit allen verfügbaren Checks (*check*) gegeben. Zur Vereinheitlichung wird hierzu derselbe Datentyp genutzt, wie bei den Checks in der Operation runAnalysis (*Check*).

- **createUser**
  Als Eingabeparameter werden Nutzername (*username*) und Passwort (*password*) des anzulegenden Nutzers als String übertragen. Zurück wird ein Wahrheitswert gesendet, welcher angibt ob der Nutzer erfolgreich angelegt wurde (*out*).

- **readUser**
  readUser benötigt als Eingabeparameter den eindeutigen Nutzernamen des Nutzers. Zurück kommt eine Liste mit Referenzen zu den Dateien(*fileRef*), welche zu dem Nutzer gehören.

- **updateUser**
  Übergeben wird hier der Nutzername (*usernam*) und das neue Passwort (*password*) als String. Als Antwort wird ein Wahrheitswert gesendet, welcher angibt, ob das Passwort erfolgreich geändert wurde (*out*).

- **deleteUser**
  Die Anfrage enthält dien Nutzernamen des zu löschenden Nutzers als String

(*username*). Als Antwort wird ein Wahrheitswert gesendet, welcher angibt, ob der Nutzer erfolgreich gelöscht wurde (*successfull*).

- **createFile**
  In der Anfrage ist der Dateiname (*filename*) der zu erstellenden Datei und der Inhalt (*content*) als String enthalten. Die Antwort enthält eine Referenz (*FileReference*) zu der Datei (*fileRef*).

- **readFile**
  Mit der Anfrage wird die Identifikationsnummer der Datei als String übertragen (*fileID*). In der Antwort befindet sich eine Referenz (*FileReference*) zu der Datei (*fileRef*).

- **updateFile**
  Der Operation wird die Identifikationsnummer (*fileID*) der zu aktualisierenden Datei mit dem aktualisierten Inhalt (*content*) gesendet. Als Antwort wird eine Referenz zu der Datei (*fileRef*) gesendet.

- **deleteFile**
  Die Anfrage enthält die Identifikationsnummer der Datei (*fileID*). In der Antwort, wird ein Wahrheitswert übertragen der angibt, ob das Löschen der Datei erfolgreich war (*out*).

### 4.3.2 REST-Anfragen

Durch die REST-Anfragen, gilt es dieselben Operationen bereitzustellen, wie bei SOAP-Aufrufen. Die zu übertragenen Informationen sind dieselben wie bei den SOAP-Aufrufen und werden in Form von Form-Parametern an den Webservice übertragen. Antworten werden in Form von JSON-Objekten als Text übertragen. Im Detail sollen die Operationen im Folgenden dargestellt werden.

- runAnalysis
  Um das Ausführen einer Analyse durch die REST-Schnittstelle zu veranlassen, wird ein POST-Aufruf auf der URI *Webadresse/runAnalysis* mit den jeweiligen Informationen durchgeführt.

- getAvailableChecks
  Die Informationen über die verfügbaren Checks können über einen GET-Aufruf auf der URI *Webadresse/getAvailableChecks* abgerufen werden.

- user
  Ein neuer Nutzer wird über einen POST-Aufruf auf der URI *Webadresse/user* angelegt. Diese Operation entspricht dem createUser-SOAP-Aufruf.
  Informationen über bestehende Nutzer, analog zur readUser SOAP-Operation, werden durch Aufrufen der jeweiligen URI *Webadresse/user/{username}* abgerufen. Der {username} entspricht hier dem eindeutigen Benutzernamen eines Nutzers.

Analog zur updateUser-SOAP-Operation, kann man das Passwort eines Nutzers durch einen PUT-Aufruf auf der URI *Webadresse/user/{username}* ändern.
Löschen kann man einen Nutzer durch einen DELETE-Aufruf auf der jeweiligen Ressource *Webadresse/user/{username}*.

- file
  Die Aufrufe auf der file-Resource erfolgen analog zur den Aufrufen auf der user-Ressource. Hierbei wird die Webadresse auf *Webadresse/file* bzw. *Webadesse/file/{fileID}* geändert. Die {fileID} entspricht hierbei der eindeutigen Identifikationsnummer der Datei. Dementsprechend wird die createFile-SOAP-Operation durch einen POST-Aufruf auf der file-Ressource durchgeführt, die readFile-SOAP-Operation durch einen GET-Aufruf auf der file/{fileID}-Ressource , die updateFile-SOAP-Operation durch einen PUT-Aufruf auf der file/{fileID}-Ressource und die deleteFile-SOAP-Operation durch einen DELETE-Aufruf auf der file/{fileID}-Ressource.

## 4.4 Klassendiagramm

Das Klassendiagramm ist in Abblidung 4.13 zu sehen. Durch die Implementierung benötigte Activator-Klassen werden hier nicht dargestellt, da sie nicht zur eigentlichen Funktionalität beitragen.
Das Interface CarismaWS stellt das aus der WSDL generierte Interface zur Kommunikation per SOAP dar und Gateway die Implementierung dieses Interfaces. Analog handelt es sich bei RestInterface um das annotierte Interface, welches die Kommunikation per REST anbietet und bei der Klasse RestImpl um die Implementierung dieses Interfaces. Die jeweiligen Implementierungen dienen dazu, die Anfragen der Kommunikationsarten entgegenzunehmen, aufzubereiten und im Anschluss an den Controller weiterzugeben. Die Funktionalität des Controllers besteht darin, dass er die Anfragen durch die Kommunikationsschnittstellen entgegennimmt und verarbeitet. Hierzu ruft der Controller die jeweils benötigten Funktionalitäten innerhalb von Carisma und der Implementierung des DatabaseInterface auf. Die Carisma-Klasse stellt hier die unveränderte Carisma-Klasse aus der Carisma-Bibliothek dar. Der WSConnector stellt die Implementierung des UIConnectors aus der CARiSMA-Bibliothek dar. Er dient der Protokollierung der CARiSMA-Funktionalitäten. Die Klasse DatabaseHelper stellt die Implementierung des DatabaseInterfaces dar, auf welcher der Controller arbeiten wird um Daten persistent zu speichern. Hierzu verwendet diese Implementierung das Interface Connection mit welchem der Zugriff auf die Datenbank ermöglicht wird.
Die Klassen sollen im Folgenden detaillierter erklärt werden.

### 4.4.1 CarismaWS

Die Klasse CarimsaWS stellt das mit Hilfe von WSDL2Java generierte Interface dar. Jede Operation der WSDL-Datei generiert hier eine Methode, welche mit Java

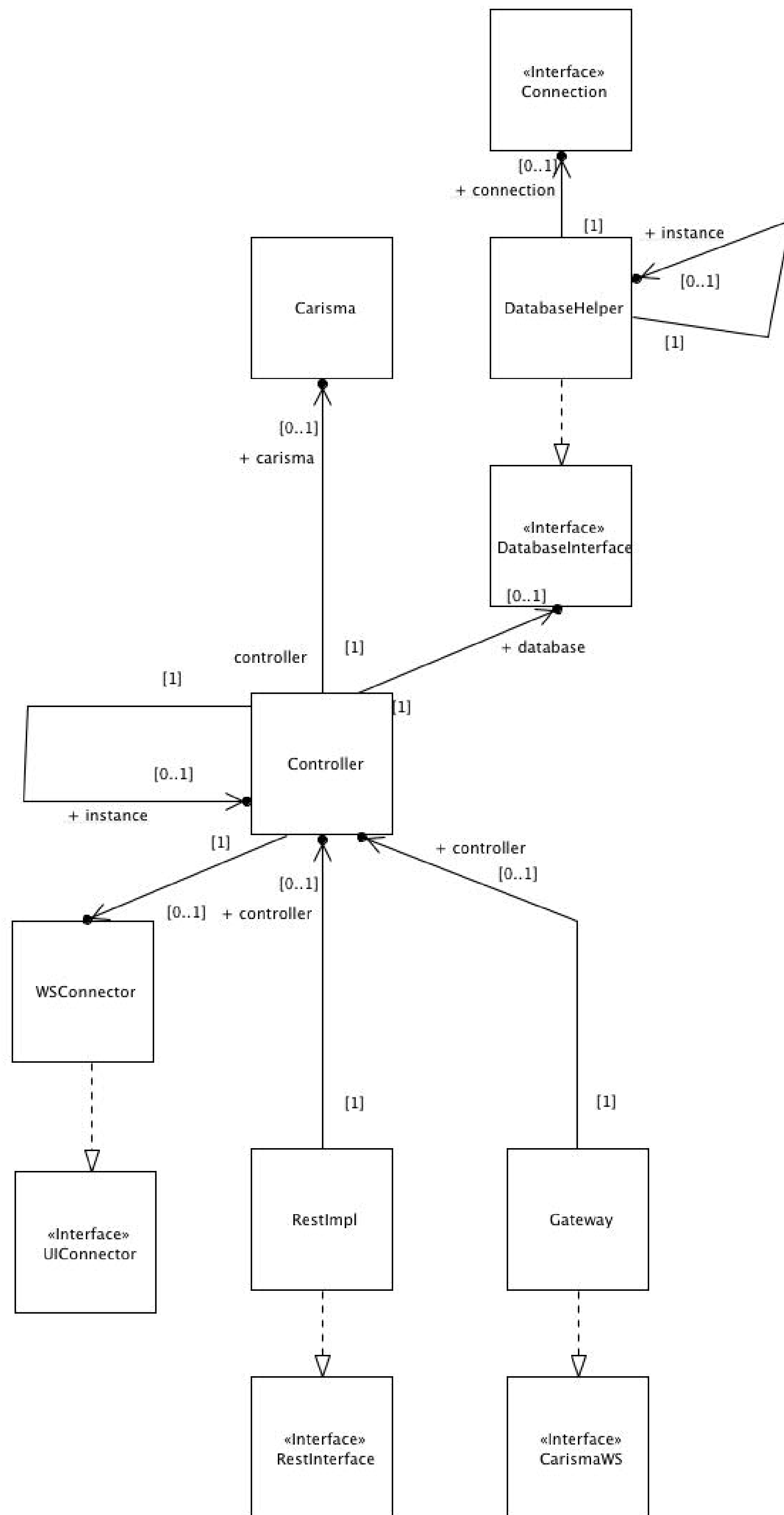


**Abbildung 4.13:** Klassendiagramm

Objekten, die den in der WSDL definierten Datentypen entsprechen, aufgerufen werden und diese zurückgeben.

### 4.4.2 Gateway

Die Klasse Gateway implementiert das CarismaWS-Interface. Die Methoden haben in der Regel drei essentielle Schritte. Zuerst werden die Informationen in ein für den Controller handhabbares Format verarbeitet. Mit diesen wird der Controller aufgerufen, welcher das Ergebnis zurückgibt. Als letzten Schritt wird die Rückgabe in ein Objekt der SOAP-Schnittstelle umgewandelt und zurückgegeben.

### 4.4.3 RestInterface

Analog zu CarimsaWS beinhaltet das RestInterface annotierte Methoden für jede Anfrage, die per REST-Aufruf zur Verfügung stehen soll.41

### 4.4.4 RestImpl

Die Klasse RestImpl implementiert das RestInterface und arbeitet mit denselben essentiellen Schritten wie das Gateway, lediglich basierend auf der REST-Kommunikation.

### 4.4.5 Controller

Der Controller enthält für jeden Anwendungsfall eine Methode, die diesen Anwendungsfall bereitstellt. Diese Methoden sollen von den Kommunikationsschnittstellen aufgerufen werden. Innerhalb der Methoden werden die jeweils benötigten Komponenten, Carisma oder Datenbank (vgl. Kapitel 4.5), angesprochen und die Ergebnisse weiterverarbeitet.

### 4.4.6 WSConnector

Der WSConnector implementiert das UIConnector-Interface. Diese Implementierung enthält Protokollierungsfunktionalitäten.

### 4.4.7 DatabaseInterface

Gegen dieses Interface wird der Controller entwickelt, was die zuvor beschriebene Portabilität der Datenbankkomponente ermöglicht. Es enthält Methodenkörper für jede auf der Datenbank benötigte Anfrage.

### 4.4.8 DatabaseHelper

Diese Klasse implementiert das DatabaseInterface für eine SQLite-Datenbank.

## 4.5 Architektur

Hier wird die Architektur der Softwarelösung vorgestellt und die oben beschriebenen Klassen ihren jeweiligen Komponenten zugeordnet. Die Software soll wie in

Abbildung 4.14 zu sehen ist, hauptsächlich aus vier voneinander separaten Komponenten bestehen. Einer Kommunikationsschnittstelle, welche Kommunikationsprotokolle implementiert, einem Controller-Bundle, welches die Kommunikation der verschiedenen Komponenten realisiert, einem Datenbank-Bundle, welches die benötigten Daten persistent speichert und einem CARiSMA-Bundle, welches die Logik von CARiSMA bereitstellt. Eine solche Architektur ermöglicht es, die Kommunikationsschnittstelle und die Datenbank einfach auszutauschen und so für den Anwendungsfall optimierte Softwarelösungen bereitzustellen.

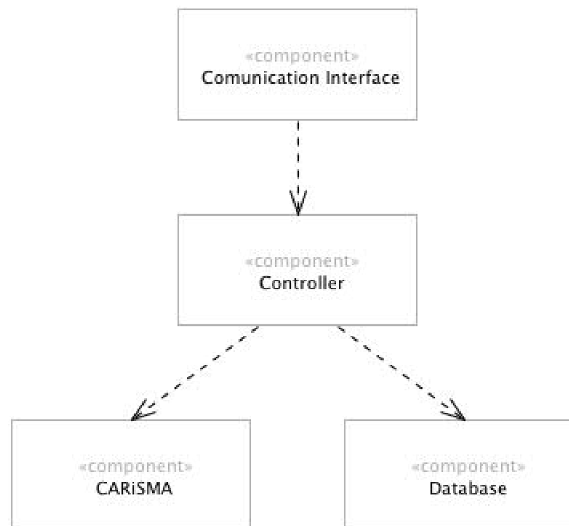


**Abbildung 4.14:** Architektur der Komponenten

### 4.5.1 Komponente Kommunikationsschnittstelle

Diese Komponente dient der Realisierung der Kommunikationsschnittstelle, welche von außen angesprochen werden kann. Die Implementierung der SOAP-Schnittstelle und der REST-Schnittstelle stellen Instanzen dieser Komponente dar. Dementsprechend besteht die SOAP-Instanz dieser Komponente aus einem Bundle, welches das CarismaWS Bundle anbietet und einem Bundle, welches dieses Bundle implementiert. Diese Aufteilung des Interfaces und der Implementierung basiert auf der OSGi-Konvention, Interfaces und Implementierungen zu trennen, um so Interfaces verbessert wiederverwenden zu können. Die REST-Instanz besteht analog aus einem Bundle, welches die RestInterface-Logik anbietet und einem Bundle, dass die RestGatway-Implementierung enthält.

### 4.5.2 Komponente Controller

Die Controller Komponente wird durch ein Bundle realisiert, welches die Controller- und WSConnector-Klasse anbietet. Dieses Bundle wird zur Laufzeit die verfügbare Datenbank-Implementierung zugewiesen bekommen.

### 4.5.3 Komponente Datenbank

Um in der Lage zu sein, verschiedene Datenbanksysteme einfach einbinden zu können, wurde hier mit deklarativen OSGi Services gearbeitet. Hierzu implementiert ein Bundle das DatabaseInterface, gegen welches auch der Controller entwickelt wurde. Ein weiteres Bundle enthält die Implementierung in Form der DatabaseHelper-Klasse.

### 4.5.4 Komponente CARiSMA

Hier wird eine CARiSMA-Instanz unverändert von der Eclipse-Plugin Version genutzt.

## 4.6 Sicherheit

Im Rahmen des Zieles *Erfüllung der CIA-Ziele* (Kapitel 2.3.2 c)) zu erbringende Sicherheitsvorkehrungen werden hier konzeptionell anhand von möglichen Angriffsszenarios vorgestellt. Es wird davon ausgegangen, dass der zur Verfügung stehende Rechner, auf welchem der Dienst angeboten wird, abgesichert ist und sich keine Schadsoftware auf diesem befindet.

### 4.6.1 Abhören

Da das Abhören von Anfragen nicht vermeidbar ist, gilt es die Kommunikation zu verschlüsseln, um so Angreifern die Möglichkeit zu nehmen, Informationen aus mitgelesenen Anfragen zu erhalten. Hierzu wird eine asynchrone Verschlüsselungstechnik eingesetzt. Die Nutzerseite verschlüsselt die Anfrage mit dem öffentlichen Schlüssel des Servers, welche im Anschluss nur mit Hilfe des privaten Schlüssels des Servers entschlüsselt werden kann. Die Authentizität dieses Schlüssels kann mit Hilfe von Zertifikaten im Rahmen einer Public-Key-Infrastruktur überprüft werden. Um Antworten entsprechend absichern zu können, muss dem Server der öffentliche Schlüssel des Nutzers bekannt sein. Dieser wird im Rahmen einer authentifizierten Nutzerfunktion über einen gesicherten Kanal an den Webservice übertragen. Alle folgenden Antworten des Servers zu dem Nutzer werden dann mit dem bekannten Schlüssel verschlüsselt. Durch dieses Vorgehen ist es Angreifern nicht mehr möglich, Anfragen mitzulesen und daraus Informationen zu erhalten.

### 4.6.2 Integrität

Die Manipulation von Antworten kann genutzt werden, um beispielsweise durch eine Analyse aufgedeckte Sicherheitsprobleme zu verwischen. Damit dies nicht geschieht, müssen Antworten von der Serverseite mit dem privaten Schlüssel signiert werden, sodass bei Misstrauen die Integrität des Inhaltes überprüft werden kann.

### 4.6.3 Authentifizierung

Damit sich Nutzer authentifizieren können, kommt eine einfache Kombination aus Nutzername und Passwort zum Einsatz. Diese Kombination wird auf Serverseite in

einer Datenbank abgespeichert, wobei das Passwort gesondert verschlüsselt werden muss. Diese Nutzerinformationen werden bei jeder Anfrage des Nutzers mitgesendet, wodurch keine Verwaltung von Sitzungsschlüsseln notwendig wird. Stimmt die Kombination aus Nutzername und Passwort nicht überein, wird die weitere Verarbeitung der Anfrage abgebrochen und eine Fehlermeldung an den Nutzer zurückgeliefert.

### 4.6.4 Wiedereinspielattacke

Um zu vermeiden, dass mögliche Angreifer in der Lage sind, durch das Wiedereinspielen von aufgenommenen Anfragen an Informationen zu gelangen, soll eine eindeutige Anfrage-Identifikationsnummer verwendet werden. Diese Anfrage-Identifikationsnummer besteht aus einer zufällig generierten Zeichenkette. Diese Zeichenkette wird auf der Serverseite in den Datenbankeintrag des Nutzers geschrieben und eine folgende Anfrage nur durchgeführt, wenn die übertragene Zeichenkette mit der in der Datenbank hinterlegten übereinstimmt. Wurde ein Aufruf mit der gespeicherten Zeichenkette ausgeführt, wird eine neue Zeichenkette für den Nutzer generiert und die alte gelöscht. Die neue Zeichenkette kann der Nutzer mit Hilfe eines Aufrufs von der Serverseite erfragen. Für diesen Aufruf wird keine eindeutige Identifikationsnummer benötigt.

### 4.6.5 Auslesen der Datenbank

Sollte es einem Angreifer möglich sein, die Datenbank des Webservices auszulesen, wären im Klartext gespeicherte Passwörter kompromittiert. Um dies zu vermeiden gilt es nicht die Passwörter im Klartext, sondern einen auf Basis des Passworts berechneten Hashwert zu speichern.
Da ein so abgespeichertes Passwort durch sogenannte „Rainbow Tables“, Tabellen, welche Hashwerte zu bekannten Zeichenketten enthalten, beschleunigt zurückverfolgt werden könnte, gilt es die Passwörter mit einem Salt-Wert zu bearbeiten. Bei der Bearbeitung wird dem Passwort vor dem Berechnen des Hash-Wertes dieser Salt-Wert angehängt. Der Salt-Wert wird im Klartext in der Datenbank abgelegt und beim Anlegen eines Nutzer Eintrags zufällig generiert. Durch dieses Vorgehen werden Hash-Werte aus „Rainbow Tables“ entwertet, da diesen der Salt-Wert fehlt. Muss ein Nutzer authentifiziert werden, wird der Salt-Wert aus der Datenbank ausgelesen, der Hash-Wert des übertragenen Passworts mit dem angehängten Salt-Wert berechnet und im Anschluss die beiden Hash-Werte verglichen. Sind diese gleich, ist das übertragene Passwort mit dem gespeicherten identisch.

### 4.6.6 Zugriff auf fremde Dateien

Ohne Absicherungen könnte es Angreifern mit einem gültigen Nutzernamen und Passwort möglich sein, die Dateien anderer auszulesen oder zu verändern. Um dies zu vermeiden, wird ein simples Rechtesystem eingeführt, bei welchem nur der Ersteller einer Datei Zugriff auf diese hat. Hierfür wird in der Datenbank, welche die Dateien verwaltet, die eindeutige Nutzeridentifikationsnummer zu dem Dateieintrag gespeichert. Fordert nun ein Nutzer Zugriff auf eine Datei, wird dessen Nutzeridentifikationsnummer gegen die hinterlegte abgeglichen. Sind beide Werte gleich, so

erhält der Nutzer den benötigten Zugriff. Sind sie unterschiedlich, wird der Zugriff verweigert.

# 5 Implementierung

Hier werden im Einzelnen die dem Bereich Implementierung zugeschriebenen Arbeitsschritte erklärt. Im Detail soll auf die genaue Umsetzung, aufgetretene Schwierigkeiten und eventuelle Verbesserungsmöglichkeiten eingegangen werden. Der vollständige Quellcode der Arbeit ist auf der beiliegenden CD hinterlegt. Quellcode, der für die Arbeit von besonderem Interesse ist, wird zusätzlich innerhalb des Anhangs abgedruckt. Mit Ausnahme der Implementierung des SVN-Hooks (vgl. Kapitel 5.3) wurde die Implementierung in der Programmiersprache Java, Version 1.6, unter Verwendung der Entwicklungsumgebung Eclipse Kepler durchgeführt.

## 5.1 Terminal

Im Folgenden soll die Implementierung der Terminal-Anwendung beschrieben werden. Zuerst wird gezeigt, wie es ermöglicht wurde, Equinox zu starten und die benötigten Bundles zu installieren und zu starten. Im Anschluss soll gezeigt werden, wie die Kommandozeilenaufrufe implementiert wurden und wie die Kommunikation von CARiSMA zu der Kommandozeile realisiert wurde.

### 5.1.1 Starten der Equinox-Instanz

Um CARiSMA innerhalb der Terminal-Anwendung ausführen zu können, musste zuerst eine Möglichkeit gefunden werden, eine Equinox-Instanz mit Hilfe von Java-Quellcode zu starten. Hierzu wurde die Equinox-Archiv-Datei innerhalb der Anwendung abgelegt und von dort über die Framework-Factory mit den gewünschten Konfigurationen gestartet.
Die nächste Schwierigkeit ist es, Bundles innerhalb der gestarteten Equinox-Instanz zu installieren und dann zu starten. Zur Installation wird beim Start der Anwendung der Dateipfad zu den Bundles abgefragt. Der hierbei angegebene Ordner wird nach JAR-Dateien durchsucht und die gefundenen werden installiert. Die Installation erfolgt hier nach dem Starten der Instanz mit dem abrufbaren Context-Objekt, welches bei dem Start der Equinox-Instanz erstellt wird, durch das Aufrufen der Methode installBundle(). Das hierbei zurückgegebene Bundle-Objekt wird innerhalb einer Liste gespeichert. Zum Starten der Bundles wird nun diese Liste an Bundles durch iteriert und durch den Aufruf bundle.startBundle() das jeweilige Bundle gestartet.

### 5.1.2 Kommandozeilenaufrufe

Um Kommandozeilenaufrufe anzubieten, wurde das CARiSMATerminal-Bundle implementiert. Zur Bereitstellung der Kommandos wurde das CommandProvider-Interface

durch die Klasse MyCommandProvider implementiert. Diese definiert eigene Kommandos zu implementieren, welche dann wie reguläre Equinox-Kommandozeilenaufrufe zur Verfügung stehen und bei einem Aufruf der Hilfe auch erklärt werden können. Das Aufrufen eines solchen Kommandos ruft die entsprechenden Methoden innerhalb des AnalysisController auf. Die AnalysisController-Methoden bereiten die übergebenen Parameter auf, rufen benötigte CARiSMA-Methoden auf und geben das Ergebnis auf dem Terminal aus.

### 5.1.3 Kommunikation mit CARiSMA

Um in der Lage zu sein, der Anwendung Parameter zu übergeben, wurde das UIConnector-Interface in der Klasse UIConnectorImpl implementiert. Diese Implementierung fragt bei Bedarf auf der Kommandozeile nach dem Wert des jeweils gefragten Check-Parameters. Dieser Parameter wird im Anschluss in das jeweilige Format konvertiert, dem Check-Parameter hinzugefügt und der Check-Parameter zurückgegeben. Diese Funktionalität wird als Teil des CARiSMATerminal-Bundles bereitgestellt.

### 5.1.4 Starten CARiSMAs

Um CARiSMA innerhalb des OSGI-Containers starten zu können, wurde das Plugin auf die benötigten Abhängigkeiten untersucht. Diese Abhängigkeiten wurden mit dem Plugin in den Installationsordner kopiert und somit beim Starten der Anwendung installiert. Zum Starten CARiSMAs waren keine Änderungen nötig.

## 5.2 Webservice

Die folgende Beschreibung der Implementierung des Webservices baut sich den Komponenten des Webservices entsprechend auf. Zuerst sollen die Implementierungen der Kommunikationsschnittstellen in Kapitel 5.2.1 für SOAP (vgl. Kapitel 5.2.1) und REST (vgl. Kapitel 5.2.1) vorgestellt werden. Die Implementierung der Controller-Komponente wird in Kapitel 5.2.2 beschrieben. Zum Abschluss soll die Implementierung der Datenbank-Komponente in Kapitel 5.2.3 beschrieben werden.
Im Rahmen der Implementierung des Webservices wurde der in Kapitel 5.1.1 beschriebene TerminalLauncher zum Starten und Ausführen der Equinox-Umgebung, welche den Webservice bereitstellt, wiederverwendet. Es muss lediglich der Ordner mit den Webservice-Bundles als Installationsordner übergeben werden. Im Rahmen dieser Implementierung wurden Sicherheitsmaßnahmen nicht berücksichtigt, da diese zu einem späteren Zeitpunkt implementiert werden.

### 5.2.1 Kommunikationsschnittstellen

Die Implementierungen der Kommunikationsschnittstellen ähneln sich im generellen Aufbau insofern, dass jede Schnittstelle in einem Bundle ein Interface anbietet und in einem anderen Bundle das Interface implementiert. Im Folgenden sollen die jeweiligen Implementierungen detailliert beschrieben werden.

### SOAP

Als Erstes wurde aus der WSDL-Datei der Java-Quellcode generiert und in einem Bundle zur Verfügung gestellt. Dieser Quellcode enthält das CarismaWS-Interface, sowie die in der WSDL-Datei definierten Datentypen als Java-Klassen. Der generierte Quellcode wird im Rahmen des CARiSMAWSInterface-Bundles bereitgestellt. Diese Art der Bereitstellung ermöglicht es, die Implementierung der SOAP-Schnittstelle schnell auszutauschen.
Als Nächstes wurde die Klasse Gateway (vgl. Listing A.2) implementiert. Bei der Implementierung der Kommunikation per SOAP kam es zu dem Problem, dass die WSDL-Datei nicht innerhalb des Programmes übernommen wird, sondern eine neue WSDL-Datei aus dem Quellcode, wie es bei einem Code-First Vorgehen der Fall wäre, aus dem Quellcode generiert wird. Dies bedeutet, dass aus der im Konzept definierten WSDL-Datei (vgl. Kapitel 4.3) Quellcode generiert wird, aus welchem eine WSDL-Datei generiert wird. Aus Mangel an Zeit und da dieser Fehler keine Funktionseinschränkung darstellt, wurde dieser Fehler nicht behoben. Die Klasse Gateway wird innerhalb des CARiSMASoapGateway-Bundles bereitgestellt. Durch den Activator dieses Bundles wird der Webservice konfiguriert und dem Jetty-Server bekannt gemacht.
Bei der Implementierung der SOAP-Schnittstelle fiel auf, dass ein Contract-First-Vorgehen hier nicht optimal war. So wäre es möglich gewesen, bei einem Code-First-Vorgehen die bestehenden CARiSMA-Objekte besser zu nutzen, sodass man das zusätzliche Parsen der SOAP-Objekte auf CARiSMA-Objekte hätte vermeiden können.

### REST

Bei der Implementierung der REST-Schnittstelle wurde die JAX-RS Bibliothek [Fou13a] genutzt, welche es erlaubt Webservices durch Annotationen in Java-Interfaces zu definieren. Die REST-Implementierung erfolgte analog zu der SOAP Implementierung, wobei das Interface RestInterface, zu sehen in Listing A.3, manuell annotiert werden musste, um die entsprechenden REST-Operationen implementieren zu können. Als Erstes wurde mit der @Path-Annotation festgelegt, unter welchem Endpunkt der Webservice zu erreichen ist. Mit den Annotationen @GET, @POST, @PUT, @DELETE wurde festgelegt, mit welcher HTTP-Methode die Ressource aufgerufen werden muss, um eine Methode auszuführen. Die @Path-Annotation an den Methoden sagt aus, unter welchem Pfad diese zu erreichen ist. Ist in geschweiften Klammern wie bei den Datei-Operationen ein Variablenname angegeben, bedeutet dies, dass sich hier zur Laufzeit definierte Ressourcen befinden. Dies ermöglicht es, dynamisch Ressourcen wie Dateien anhand ihrer Identifikationsnummer zu adressieren. Durch die @Produces-Annotation wird angegeben, welches Format die Antwort haben wird (in dieser Implementierung immer Klartext). Die Annotationen an den Variablen in den Methoden geben an, wie die Variablen übergeben werden. @PathParam bedeutet hier, dass die Variable aus dem Ressourcenpfad stamt, mit @QueryParam annotierte Variablen werden an den Ressourcenpfad angehängt übergeben.
Das Interface wurde in der Klasse RestImpl implementiert.

Das Interface wird als CARiSMARestInterface-Bundle, die Implementierung als CARiSMARestGateway-Bundle bereitgestellt. Der Activator des CARiSMARestGateway-Bundle definiert, wie der Activator des CARiSMASoapGateway-Bundles, die einzelnen Einstellungen des Webservices und macht diesen ebenfalls dem Jetty-Server bekannt.

### 5.2.2 Controller

Die Controller-Komponente setzt sich, wie im Konzept vorgestellt, aus den Klassen WSConnector und Controller zusammen.
Die Klasse Controller wurde als Singleton implementiert, wodurch zu jedem Zeitpunkt mit derselben Controller-Instanz gearbeitet wird. Die Implementierung der Controller-Klasse erwies sich als unproblematisch und konnte mit Hilfe des Konzeptes (vgl. Kapitel 4) durchgeführt werden. Die Controller-Klasse ist in Listing A.1 zu sehen.
Die Implementierung des UIConnector-Interfaces in Form des WSConnectors konnte ebenso leicht implementiert werden. Die Protokollierung erfolgt direkt auf der Kommandozeile.
Die Controller-Komponente wird in Form des CARiSMAController-Bundle bereitgestellt.

### 5.2.3 Datenbank

Als Erstes wurde das DatabaseInterface, wie es in Listing A.4 zu sehen ist, erstellt. Es wurde nach den aus dem Konzept abzuleitenden Anforderungen erstellt und wird als DatabaseInterface-Bundle bereitgestellt.
Bei der Implementierung der Klasse DatabaseHelper wurde zuerst die Initialisierung der Datenbank realisiert. Dies umfasste das erstellen der Datenbank, eine Verbindung zu dieser aufzubauen und das Anlegen der benötigten Tabellen. Nachdem dieser Schritt erfolgreich implementiert wurde, wurde das DatabaseInterface und somit die vom Controller benötigten Methoden implementiert. Die Klasse DatabaseHelper wird im Rahmen des SQLiteDatabase-Bundles bereitgestellt.

### 5.2.4 CARiSMA

Um CARiSMA im Kontext des Webservices nutzen zu können, war eine Änderung innerhalb des Quellcodes notwendig. So wurde ein neuer Konstruktor mit angepassten Parametern in der Klasse Analysis angelegt. Dies wäre vermeidbar gewesen, jedoch wäre der hierfür benötigte Aufwand nicht verhältnismäßig gewesen.

## 5.3 SVN-Hook

Bei der Realisierung des SVN-Hooks wurden die Skriptsprachen bash und php verwendet.
Als Erstes wurde das preCommitHook-Bash-Skript, welches vor dem Schreiben von Änderungen in einer SVN-Versionsverwaltung aufgerufen wird, so verändert, dass es ein weiteres bash-Skript aufruft. Dieses bash-Skript ruft ein php-Skript auf, wel-

ches in der Lage ist, mit dem Webservice zu kommunizieren. Um dieses php-Skript zu erstellen, wurde das von Knut Urdalen und John Lindal entwickelte Werkzeug wsdl2php genutzt, welches analog zu wsdl2java php-Quellcode aus einer WSDL-Datei generiert. Mit Hilfe des generierten Quellcodes war es möglich, dass das php-Skript UML-Dateien an den Webservice sendet, einen Dummy-Check auf diesen Modellen ausführt und zum Abschluss die Ergebnisse herunterlädt.
Die heruntergeladenen Ergebnisse werden im Folgenden weiterverarbeitet und überprüft, ob die festgestellte Menge an Elementen innerhalb der UML-Datei einen festgelegten Schwellenwert überschreitet. Wird dieser Schwellenwert überschritten, wird der Commitversuch abgebrochen und dem Nutzer eine Fehlermeldung zurückgegeben.

## 5.4 Nachinstallieren

Um Checks nachinstallieren oder aktualisieren zu können, musste zuerst eine Möglichkeit gefunden werden, die neuen Checks herunterzuladen. Diese Möglichkeit wurde bei einer genauen Analyse der Struktur der Update-Seite [WW13] gefunden. Auf dieser kann man sich unter der Webadresse `http://vm4a003.itmc.tu-dortmund.de/carisma/updatesite/plugins/` alle von der Update-Seite angebotenen Plugins anzeigen lassen. Diese Seite wird mit Hilfe des SAXParsers weiterverarbeitet, sodass man eine Liste mit den genauen Bezeichnern der Plugins erhält. Diese Liste wird mit der Liste der bereits installierten Plugins abgeglichen. Sollte bei dem Abgleichen festgestellt werden, dass eine neue Version eines bereits installierten Plugins oder ein bisher komplett unbekanntes Plugin verfügbar ist, wird die Webadresse zu diesem aus der Internetseite extrahiert und die Equinox-Instanz mit dem Befehl installBundle(Webadresse des Plugins) aufgerufen. Dieser Befehl installiert das unter der Webadresse vorliegende Plugin. Es muss im Anschluss die CARiSMA Check- und Model-Type-Registry aktualisiert werden, sodass die neuen Checks als verfügbar angezeigt werden. Die Implementierung wird als CARiSMAUpdate-Bundle bereitgestellt und kann in eine einfache Webservice-Equinox-Instanz installiert werden und neben dem Webservice laufen.
Als Problem erwies sich, dass bei den Plugins, welche von der Internetseite heruntergeladen wurden, der Bezeichner „de.umlsec.tool“ genutzt wurde, welcher nicht mit der im Rahmen der Arbeit verwendeten Version aus der Versionsverwaltung kompatibel ist. Somit besteht die theoretische Möglichkeit, Updates mit diesem Bundle durchzuführen, jedoch müssen hierzu die Plugins auf der Update-Seite aktualisiert werden.

## 5.5 Implementierung der Sicherheitsanforderungen

Um die im Konzept vorgestellten Sicherheitsvorkehrungen betreffend Abhören, Integrität, Authentifizierung und Wiedereinspielattacken (vgl. Kapitel 4.6) zu realisieren, wäre es notwendig, sogenannte Interceptoren zu implementieren. Diese Interceptoren würden vor der Verarbeitung des Anfrage-Bodys ausgeführt und erlauben es Operationen auf dem Head der Anfrage auszuführen. Es hätte folgende Interceptoren realisiert werden müssen:

- ein Interceptor, der die Anfragen entschlüsselt, um diese weiterverarbeiten zu können,
- einen der die im Header übertragenen Nutzerdaten abgleicht und
- einen weiteren, der die eindeutige Anfrage-Identifikationsnummer (Nonce) überprüft.

Analog zu diesen Interceptoren, welche eingehende Nachrichten bearbeiten können, gibt es solche auch für ausgehende Nachrichten. Hier wären zwei Interceptoren nötig:

- einen der die Antworten signiert und
- einen der die Antworten verschlüsselt.

Die Implementierung dieser Interceptoren war im Rahmen der Bearbeitung dieser Abschlussarbeit nicht möglich, da die verwendete Distributed-OSGi-Version das Hinzufügen von Interceptoren nur durch Umwege erlaubt [Ber13]. Dadurch, dass sich erst am Ende der Bearbeitungszeit dieser Arbeit ein Versionswechsel als Problemlösung abzeichnete, war ein Wechsel der Versionen und somit die Behebung des Problems zeitlich nicht mehr möglich.
Der Zugriff auf fremde Dateien und das Hashen der Passwörter wurde wie beschrieben umgesetzt.

# 6 Validierung

In diesem Kapitel erfolgt die Validierung der in Kapitel 5 vorgestellten Implementierung. Hierzu wurden gemäß guter Softwareengineering-Praxis zum einen Tests (vgl. Kapitel 6.1) durchgeführt, zum anderen wurde der produzierte Quellcode auf seine Qualität untersucht (vgl. Kapitel 6.2).

## 6.1 Tests

Im Folgenden wird die angewandte Teststrategie beschrieben und die verschiedenen durchgeführten Tests werden erläutert. Die Teststrategie wird im Detail in Kapitel 6.1.1 erklärt. In Kapitel 6.1.2 werden die Komponenten-Tests dargestellt. Folgend werden in Kapitel 6.1.3 die durchgeführten Systemtests beschrieben. Es werden das jeweilige Testvorgehen und bei der Durchführung aufgetretene Probleme vorgestellt. Auch werden durch die Tests aufgefallene Schwächen aufgezeigt. Der im Rahmen der Tests erstellte Quellcode ist auf der CD zu dieser Arbeit hinterlegt. Auf Integrationstests wurde aus Zeitgründen verzichtet. Die in den Tests verwendeten Modelle stammen aus der CARiSMA-Versionsverwaltung.

### 6.1.1 Teststrategie

Als Teststrategie wurde eine testgetriebene Entwicklung angestrebt, bei welcher Testfälle bereits vor der Implementierung festgelegt werden. Das Vorgehen wurde jedoch bedingt durch die Arbeitsweise bei der Entwicklung nicht komplett durchgeführt und durch ein alternatives Vorgehen ersetzt. Im weiteren Verlauf der Bearbeitung wurde ein White-Box-Testvorgehen genutzt. Hierbei werden Testfälle mit Hilfe von Wissen über die Implementierung des Systems angefertigt. Bei diesem Vorgehen wurde besonderer Wert auf eine hohe Abdeckung der Testkriterien Anweisungsüberdeckung und Zweigüberdeckung gelegt.

### 6.1.2 Komponenten-Tests

Die im Rahmen dieser Arbeit durchgeführten Komponenten-Tests wurden mit Hilfe des Frameworks JUnit und der Bibliothek PowerMock durchgeführt. JUnit wurde hier zur Automatisierung der Testfälle und deren Auswertung genutzt. PowerMock ermöglichte das Erstellen von Mock-Objekten, um ein von anderen Komponenten isoliertes Testen zu ermöglichen. Der Einsatz der bekannteren Bibliothek EasyMock war hier nicht ausreichend, da es nötig war, auch statische Methoden und Konstruktoren zu mocken. Um in der Lage zu sein, auch von außen nicht sichtbare Klassen eines Bundles testen zu können, ohne jedoch selber den Build-Path verwalten zu

müssen, wurde zu jedem Bundle ein Fragment-Bundle erstellt. Die mit den verwendeten Testfällen erreichten Überdeckungsgrade sind in Tabelle 6.1 zu sehen. Um die Überdeckungsgrade automatisiert bei der Ausführung der Tests berechnen zu lassen, wurde das Werkzeug EclEmma [HJM13], eine Implementierung des Emma-Werkzeugs als Eclipse-Erweiterung, verwendet.
Im Rahmen des Versuches die Überdeckungen weiter zu verbessern erwiesen sich die verschiedenen Ausnahmebehandlungen als problematisch. Diese konnten nicht ausreichend und verlässlich mit Hilfe der Mock-Objekte ausgeführt werden.

| Klassenname | Anweisungsüberdeckung | Zweigüberdeckung |
|---|---|---|
| Gateway | 99,7% | 94,2% |
| RestImpl | 98,6% | 96,2% |
| Controller | 97,4% | 90,5% |
| WSConnector | 100% | 100% |
| DatabaseHelper | 89,0% | 97,9% |

**Tabelle 6.1:** Testüberdeckungsgrade der Klassen

Der Komponenten-Test zeigte, dass ein Ungleichgewicht in der Komplexität der Methoden existiert, was zu einer Analyse der komplexeren Methoden führte. Die erhöhte Komplexität beschränkte sich hierbei hauptsächlich auf die Methoden, die die Ausführung des „Run Analysis“-Anwendungsfalles betreffen. Eine Reduzierung der Komplexität konnte nicht erreicht werden.

### 6.1.3 Systemtest

Beim Systemtest wurde davon ausgegangen, dass die Umwandlung der SOAP-Nachrichten durch die Apache-Bibliotheken korrekt durchgeführt wird. Auch wurde sich auf die Überprüfung des Systems mit der SOAP-Schnittstelle fokussiert, da die REST-Schnittstelle nur ein Nebenziel geringerer Priorität ist.
Die Systemtests wurden analog zum Vorgehen bei den Komponenten-Tests innerhalb eines Fragmentes des SoapGatewayBundles implementiert. Die Testmethoden rufen, ohne einen Mock zu benutzen, die zu testende Gateway-Methode auf und vergleichen das Ist-Ergebnis mit dem gegebenen Soll-Ergebnis. Für das Ausführen der Testfälle ist es erforderlich, dass die in den Testfällen genutzten Checks und deren Abhängigkeiten von dem Testsystem gestartet werden.
Bei der Implementierung wurde eine möglichst hohe Anweisungs- und Pfadüberdeckung angestrebt. Mit der Eclipse-Erweiterung EclEmma wurden diese wieder automatisch berechnet. Die erreichten Werte sind in Tabelle 6.2 zu sehen. Dass diese nicht den in den Komponenten-Tests erreichten Werten entsprechen liegt daran, dass bereits Teile von Funktionalitäten in den Klassen vorhanden sind, die noch nicht von den angebotenen Funktionalitäten genutzt werden. Auch sind verlässliche Tests der Ausnahmebehandlung schwer zu implementieren. Im Übrigen sind manche Pfade durch bereits durchgeführte Fehlerbehandlungen innerhalb der aufrufenden Komponenten nicht bei der Betrachtung des gesamten Systems zu erreichen.

| Klassenname | Anweisungsüberdeckung | Zweigüberdeckung |
|---|---|---|
| CARiSMASOAPGateway | 99,5% | 88,7% |
| Controller | 92,8% | 66,7% |
| SQLiteDatabase | 72,2% | 68,0% |

**Tabelle 6.2:** Testüberdeckungsgrade der Bundles

## 6.2 Qualitätssicherung

Im Sinne guten Softwareengineerings ist es ebenfalls wichtig, gut lesbaren und qualifiziert kommentierten Quellcode zu erstellen. In diesem Kapitel wird beschrieben, wie der in dieser Arbeit erstellte Quellcode auf diese Eigenschaften untersucht wurde.

### 6.2.1 Überprüfung durch Checkstyle

Der produzierte Quellcode wurde mit Hilfe der Eclipse-Erweiterung Checkstyle überprüft. Als Einstellung wurde die Konfiguration des CARiSMA-Entwicklungsteam genutzt, welche innerhalb der CARiSMA-Versionsverwaltung hinterlegt ist. Der Quellcode wurde insoweit bearbeitet, dass keine Warnungen bei einem Test mit Checkstyle auftraten.

### 6.2.2 Dokumentation

Der Quellcode wurde mit Hilfe des Dokumentationswerkzeugs Javadoc dokumentiert. Zusätzlich wurden Quellcodepassagen, welche sich hierzu anboten, mit weiteren Kommentaren versehen. Mit dem Werkzeug Google Code Pro Analytics wurde ein Verhältnis von Kommentaren zu Zeilen Quellcode von ungefähr 15% bestimmt.

# 7 Fazit & Ausblick

Zum Abschluss wird in Kapitel 7.1 ein Fazit der Arbeit gezogen und in Kapitel 7.2 ein Ausblick auf mögliche weitere Arbeiten gegeben.

## 7.1 Fazit

Es ist gelungen, das zu Grunde liegende Ziel dieser Arbeit zu erreichen. Eine Schnittstelle zur Sicherheitsanalyse mit CARiSMA im Kontext von serviceorientierten Architekturen wurde konzeptioniert und implementiert. Es ist somit möglich, CARiSMA in automatisierte Prozesse zu integrieren. Im Detail sollen in Kapitel 7.1.1 die im Rahmen der Vorarbeit festgelegten Ziele (vgl. Kapitel 2.3) darauf überprüft werden, ob sie erreicht wurden.

### 7.1.1 Evaluation der Ziele

**Softwarekonzept entwickeln**

Für das Erreichen des Muss-Zieles *Softwarekonzept entwickeln* (vgl. Kapitel 2.3.1 a)) galt es, ein Konzept zu entwickeln, mit welchem man die Implementierung vorab erkennen, durchführen und nachvollziehen konnte. Dieses Konzept befindet sich in Kapitel 4.
Genauer waren ein Modell der SOAP-Anfragen, ein Klassendiagramm, welches die Änderungen an der Architektur verdeutlicht, Aktivitätsdiagramme, aus denen alle Aktivitäten hervorgehen und eine Fließtextbeschreibung der Funktionen der Komponenten zu erstellen. Das Modell der SOAP-Anfragen wird mit Hilfe der WSDL-Datei in Kapitel 4.3.1 beschrieben. Das Klassendiagramm wird in Kapitel 4.4 dargestellt und erläutert. Da mit Ausnahme der Implementierung eines eigenen UIConnectors keine Änderungen an der Architektur nötig waren, wurden solche auch nicht genauer dargestellt. Da eine Unterteilung der Aktivitätsdiagramme, wie sie in der Erläuterung des Ziels beschrieben wurde, sich nicht als sinnvoll erwies, wurde sich für die Diagramme, wie sie in Kapitel 4.2 dargestellt und beschrieben werden, entschieden. Des Weiteren wurde zur Vervollständigung des Konzepts ein Anwendungsfalldiagramm (vgl. Kapitel 4.1) erstellt und beschrieben.

**CARiSMA in einem OSGi-Container ausführen**

Um zu zeigen, dass CARiSMA außerhalb der Eclipse-Entwicklungsumgebung ausführbar ist, sollte im Rahmen des Muss-Zieles *CARiSMA in einem OSGi-Container ausführen* (vgl. Kapitel 2.3.1 b)) ein Kommandozeilenaufruf implementiert werden,

welcher exemplarisch einen Dummy-Check ausführen kann. Die erfolgreiche Implementierung dieses Kommandozeilenaufrufes und wie es ermöglicht wurde CARiSMA außerhalb der Eclipse-Entwicklungsumgebung auszuführen wird in Kapitel 5.1 dargestellt. Eine Ausführung des Dummy-Checks ist über Kommandozeilenaufrufe möglich.

### SOAP-Anfragen ermöglichen

SOAP-Anfragen verarbeiten zu können, sollte durch die Bearbeitung des Muss-Zieles *SOAP-Anfragen ermöglichen* (vgl. Kapitel 2.3.1 c)) erreicht werden. Die Umsetzung dieses Zieles ist in Kapitel 5.2 beschrieben. Dass die Möglichkeit, SOAP-Anfragen zu verarbeiten, geschaffen wurde, kann mit Testaufrufen, zu finden auf der beigelegten CD, überprüft werden.

### SingleOclCheck als Webservice anbieten

Für das Muss-Ziel *SingleOclCheck als Webservice anbieten* (vgl. Kapitel 2.3.1 d)) sollte gezeigt werden, dass es möglich ist, einen CARiSMA-Check als Webservice anzubieten. Die Implementierung des Webservices ist in Kapitel 5.2 beschrieben. Dass es möglich ist, den SingleOCLCheck auszuführen, kann mit Testaufrufen auf der CD nachvollzogen werden.

### Beleg der Funktionalität der Model-Type-Registry

Um zu zeigen, dass es möglich ist, verschiedene Modelltypen mit dem Webservice zu verarbeiten, sollte im Rahmen des Muss-Zieles *Beleg der Funktionalität der Model-Type-Registry* (vgl. Kapitel 2.3.1 e)) die Ausführung des BPMN2-Dummy-Checks und des SingleOCLChecks in einer laufenen Instanz ermöglicht werden. Dies war durch die Konzeption der Softwarelösung ohne Probleme möglich. Es mussten lediglich die entsprechenden Abhängigkeiten installiert werden. Dies kann mit Testaufrufen auf der beigelegten CD überprüft werden.

### Analysen anbieten

Das Soll-Ziel *Analysen anbieten* (vgl. Kapitel 2.3.2 a)) gilt als erfolgreich abgeschlossen, wenn es ermöglicht wurde, Analysen aus mehreren Checks anzubieten. Das Konzept des Webservices (vgl. Kapitel 4.2.1) ist darauf ausgelegt, eine Liste von Checks ausführen zu können. Da die Implementierung, vorgestellt in Kapitel 5.2, sich an dieses Konzept gehalten hat, ist das Ausführen von Analysen möglich. Überprüft werden kann dies mit Testaufrufen auf der beigelegten CD.

### Automatisches Nachinstallieren neuer Checks

Es sollte mit dem Soll-Ziel *Automatisches Nachinstallieren neuer Checks* (vgl. Kapitel 2.3.2 b)) eine Möglichkeit geschaffen werden, Checks automatisiert von der CARiSMA-Updateseite herunterzuladen und zu installieren. Die Implementierung eines Bundles, welches diese Funktion bereitstellt, ist in Kapitel 5.4 beschrieben.

### Erfüllung der CIA-Ziele

Da die Sicherheit der Software in diesem Kontext einen hohen Stellenwert hat, sollte für das Soll-Ziel *Erfüllung der CIA-Ziele* (vgl. Kapitel 2.3.2 c)) der erstelle Webservice abgesichert werden. Das Sicherheitskonzept wird in Kapitel 4.6 erläutert und eine teilweise Implementierung dieses Konzepts in Kapitel 5.5 beschrieben. Aufgrund des dort beschriebenen Fehlers war es nicht möglich, alle Sicherheitsziele zu implementieren. Dennoch wird ein mögliches Vorgehen bei einer Implementierung erklärt, sodass eine auf dieser Arbeit aufbauende Arbeit einen möglichst detailliert beschriebenen Ansatzpunkt hat, die Erfüllung der Sicherheitsanforderungen fertig zu stellen.

### REST-Anfragen ermöglichen

Ob es möglich ist, den Webservice über ein alternatives Kommunikationsprotokoll anzusprechen, sollte im Soll-Ziel *REST-Anfragen ermöglichen* (vgl. Kapitel 2.3.2 d)) überprüft werden. Das Konzept einer auf REST basierenden Kommunikation wurde in Kapitel 4.3.2 entworfen. Die Implementierung dieses Konzeptes wird in Kapitel 5.2.1 beschrieben. Es wurde bei der Bearbeitung mehr als das geforderte Ziel erreicht und eine vollständige Alternative zu der SOAP-Schnittstelle geschaffen. Durch Testaufrufe auf der CD ist das Erreichen des Ziels überprüfbar.

### Anwendungsbeispiel entwickeln

Um eine Möglichkeit aufzuzeigen, wie man den erstellen Webservice benutzen kann, sollte im Rahmen des Soll-Ziels *Anwendungsbeispiel entwickeln* (vgl. Kapitel 2.3.2 e)) ein SVN-Hook implementiert werden, der ein hochgeladenes Modell automatisiert auf Sicherheitsanforderungen überprüft und gegebenenfalls den Commit ablehnt. Die Implementierung eines solchen SVN-Hooks wird in Kapitel 5.3 beschrieben. Der erstellte Quellcode liegt auf der CD vor.

### So viele Checks wie möglich anbieten

Um den Webservice mit einer möglichst hohen Funktionalität anbieten zu können, sollte für das Kann-Ziel *So viele Checks wie möglich anbieten* (vgl. Kapitel 2.3.3 a)) eine möglichst hohe Anzahl an Checks dem Webservice bekannt gemacht werden. Durch die in der Implementierung genutzte Architektur ist es möglich, den Webservice mit jedem Check, welcher von der aktuellen CARiSMA-Entwickler-Version unterstützt wird, zu nutzen. Es ist lediglich notwendig, alle benötigten Abhängigkeiten innerhalb des OSGi-Containers zu installieren.

### Automatisches Nachinstalleren mit Fremdquellen

Aufgrund von Zeitmangel war es nicht möglich das Kann-Ziel *Automatisches Nachinstallieren mit Fremdquellen* (vgl. Kapitel 2.3.3 b)) zu bearbeiten.

### Evaluation anhand eines Live-Beispiels

Die *Evaluation anhand eines Live-Beispiels* (vgl. Kapitel 2.3.3 c)), ein weiteres Kann-Ziel, war ebenfalls zeitlich nicht durchführbar.

## 7.2 Ausblick

Aufbauend auf dieser Arbeit sind Arbeiten betreffend die Sicherheit der Softwarelösung denkbar. Hier wäre das Ziel, das Sicherheitskonzept, vorgestellt in Kapitel 4.6 und Kapitel 5.5, weiter auszuarbeiten und zu implementieren. Auch wären die anderen nicht erreichten Ziele *Automatische Nachinstallieren mit Fremdquellen* (vgl. Kapitel 2.3.3 b)) und *Evaluation anhand eines Live-Beispiels* (vgl. Kapitel 2.3.3 c)) Ansatzpunkte für Folgearbeiten.
Das Absichern der REST-Schnittstelle könnte ebenfalls ein interessanter Anknüpfungspunkt sein. Ein weiterer möglicher Ansatzpunkt wäre eine Verbesserung der Integration in Geschäftsprozesse. Derzeit müssen die von CARiSMA angelegten Testergebnisse auf die für den automatisierten Geschäftsprozess relevanten Ergebnisse geparsed werden. Würde man auf der Seite CARiSMAs die Möglichkeit schaffen, die Ergebnisse aus dem Report separiert auslesen zu können und diese gezielt zurückzugeben, würde man hier die Integrierbarkeit CARiSMAs in Geschäftsprozesse deutlich verbessern.

# A Weitere Informationen

Innerhalb des Anhangs sind für das Verständnis der Arbeit hilfreiche Quelltexte abgedruckt. Sie dienen unter anderem dazu, das Zusammenspiel der Webservice-Komponenten zu verdeutlichen. So ist es mit diesen Klassen möglich, das Verfahren der Verarbeitung einer eingehenden SOAP-Nachricht im Quelltext nachzuverfolgen.

**Listing A.1:** Quellcode der Controller-Klasse

```
package carismacontroller;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import carisma.core.Carisma;
import carisma.core.analysis.Analysis;
import carisma.core.analysis.result.AnalysisResult;
import carisma.core.checks.CheckDescriptor;
import carisma.core.checks.CheckRegistry;
import carisma.core.models.ModelType;
import carisma.core.models.ModelTypeRegistry;
import dbinterface.DatabaseInterface;

/**
 *
 * @author andreasbeckmann
 *
 */
public class Controller {

    /**
     * The Instance of the Controller
     */
    private static Controller instance;
    /**
```

```
 * The Carisma Object
 */
private Carisma carisma;
/**
 * The DatabaseInterface.
 */
private DatabaseInterface database;

/**
 * Returns the current Controller instance.
 *
 * @return a Controller Instance
 * @author andreasbeckmann
 */
public static Controller getInstance() {
   if (instance == null) {
      instance = new Controller();
   }
   return instance;
}

/**
 * Returns a List of all available Checks.
 *
 * @return A List of available Checks
 * @author andreasbeckmann
 */
public final List<CheckDescriptor> getChecks() {
   carisma = Carisma.getInstance();
   CheckRegistry checkRegistry = carisma.getCheckRegistry
      ();
   if (checkRegistry == null) {
      return null;
   }
   List<CheckDescriptor> checks = checkRegistry.
      getRegisteredChecks();
   return checks;
}

/**
 * Creates a new User if the username is available.
 *
 * @param username
 *             The username of the User to be created
 * @param password
```

```
 *              The password of the User to be created
 * @return a boolean if the User was created successfully
 * @author andreasbeckmann
 */
public final boolean createUser(final String username,
   final String password) {
   if (database == null) {
      return false;
   }
   return database.insertUser(username, password);
}

/**
 * Deletes a User.
 *
 * @param username
 *              The Username of the User to be deleted
 * @return A Boolean if the User was deletet successfully
 * @author andreasbeckmann
 */
public final boolean deleteUser(final String username) {
   return database.deleteUser(username);
}

/**
 * Returns the File corresponding the ID in the Backend.
 *
 * @param fid
 *              The fileId of the File to be read.
 * @return The File corresponding to the fileID or null
 *    if an error occured.
 * @author andreasbeckmann
 */
public final File readFile(final int fid) {
   String filepath = database.getFilePath(fid, database.
      getUID("test"));
   // write AnalysisResultFile into response
   if (filepath != null) {
      File file = new File(filepath);
      if (file.isFile()) {
         // return response to Gateway
         return file;
      }
   }
   return null;
```

```
}

/**
 * Runs a Analysis.
 *
 * @param analysis
 *            The Analysis to be run.
 * @return The FileID of the created result File. -1 if
 *   an error occured.
 * @author andreasbeckmann
 */
public final int runAnalysis(final Analysis analysis) {
   System.out.println("Starte␣Analyse");
   if (analysis == null) {
      return -1;
   }
   String report;
   carisma = Carisma.getInstance();
   if (carisma == null) {
      return -1;
   }
   WSConnector connector = new WSConnector();
   carisma.runAnalysis(analysis, connector);
   List<AnalysisResult> analysisResults = carisma.
      getAnalysisResults();
   // Iterate over all available AnalysisResults to find
      the one for this
   // Analysis
   System.out.println(String.valueOf(analysisResults.size
      ()));
   for (AnalysisResult analysisResult : analysisResults)
      {
      if (analysisResult.getAnalysis().getName()
            .equals(analysis.getName())) {
         report = analysisResult.getReport();
         File reportFile = writeStringToFile(
               analysis.getName() + ".txt", report);
         int fileID = database.insertFile(reportFile,
               database.getUID("test"));
         return fileID;
      }
   }

   // return fid (if fid -1 access Exception)
   return -1;
```

```
}

/**
 * Writes a String into a File.
 *
 * @param name
 *            the name of the File
 * @param content
 *            the content of the file
 * @return the created File
 * @author andreasbeckmann
 */
private File writeStringToFile(final String name, final
  String content) {
  if (name == null || content == null) {
    return null;
  }
  Date date = new Date();
  String resultFilePath = String.valueOf(date.getTime())
    ;
  File resultFilePathFile = new File(resultFilePath);
  if (!(resultFilePathFile.exists())) {
    if (!resultFilePathFile.mkdirs()) {
      return null;
    }
  }
  String fileName = name;
  File resultFile = new File(resultFilePath + "/" +
    fileName);
  try {
    resultFile.createNewFile();
  } catch (IOException e) {
    e.printStackTrace();
  }
  try {
    PrintWriter out = new PrintWriter(resultFile);
    out.write(content);
    out.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
  return resultFile;
}
```

```
/**
 * Creates a new File containing the content.
 *
 * @param filename
 *            The filename of the File to be created.
 * @param content
 *            The content of the File to be created.
 * @return The fileID of the new File.
 * @author andreasbeckmann
 */
public final int createFileWithContent(final String
   filename,
      final String content) {
   File file = writeStringToFile(filename, content);
   if (file != null) {
      int fileID = database.insertFile(file, database.
         getUID("test"));
      return fileID;
   }
   return -1;
}

/**
 * Creates a File without any content.
 *
 * @param filename
 *            The filename of the File to be created.
 * @return The fileID of the new File.
 * @author andreasbeckmann
 */
public final int createFileWithoutContent(final String
   filename) {
   Date date = new Date();
   String parentPath = String.valueOf(date.getTime());
   File parentFile = new File(parentPath);
   parentFile.mkdirs();

   File file = new File(parentFile.getAbsolutePath() + "/
      " + filename);
   try {
      file.createNewFile();
   } catch (IOException e) {
      e.printStackTrace();
   }
```

```
    int fid = database.insertFile(file, database.getUID("
        test"));
    return fid;
}

// Method will be used by DS to set the quote service
/**
 * @author andreasbeckmann
 * @param service
 *             The implementation of the Database
     Interface supposed to be
 *             used.
 */
public final synchronized void setDatabase(final
    DatabaseInterface service) {
    System.out.println("Database␣was␣set");
    Controller controller = getInstance();
    controller.database = service;
}

/**
 * Used by Declerative Services to unset the Database if
     eg. the Bundle
 * containing the Database Implementation is stopped.
 *
 * @param service
 *             The Implementation of the DatabaseInterface
      to be unset.
 * @author andreasbeckmann
 */
public final synchronized void unsetDatabase(final
    DatabaseInterface service) {
    if (this.database == service) {
        this.database = null;
    }
}

/**
 * Returns the name of the file linked to the FileID.
 *
 * @param fileID
 *             The FileID of the File.
 * @return The Filename.
 * @author andreasbeckmann
 */
```

```
public final String getFilename(final int fileID) {
   String filepath = database.getFilePath(fileID,
      database.getUID("test"));
   if (filepath != null) {
      File file = new File(filepath);
      return file.getName();
   }
   return null;
}

/**
 * Returns the File linked to the fileID.
 * @param modellFileID
 *             The FileID of the File requested.
 * @return The File linked to the fileID.
 * @author andreasbeckmann
 */
public final File getFile(final int modellFileID) {
   String filepath = database.getFilePath(modellFileID,
         database.getUID("test"));
   if (filepath != null) {
      File file = new File(filepath);
      return file;
   }
   return null;
}

/**
 * Deletes the File linked to the fileID.
 * @param fileID
 *             the File ID of the file to be deleted
 * @return If the File was deleted successfully
 * @author andreasbeckmann
 */
public final boolean deleteFile(final int fileID) {
   File file = getFile(Integer.valueOf(fileID));
   if (file != null) {
      if (file.delete()) {
         return database.deleteFile(fileID);
      }
   }
   return false;
}
```

```
/**
 * Returns the {@link Modeltype} for the given file
    extension.
 * @param extension
 *            The fileExtension
 * @return the Modeltype corresponding to the extension
 * @author andreasbeckmann
 */
public final ModelType getTypeForFileExtension(final
   String extension) {
   carisma = Carisma.getInstance();
   ModelTypeRegistry registry = carisma.
      getModelTypeRegistry();
   ModelType modeltype = registry.getTypeForExtension(
      extension);
   return modeltype;

}

/**
 * Updates a File with the new content.
 * @param fileID
 *            The fileID of the File to be updated.
 * @param content
 *            If the File was updated successfully.
 * @return If the File was updated successfully
 * @author andreasbeckmann
 */
public final boolean updateFile(final String fileID,
   final String content) {
   File file = readFile(Integer.valueOf(fileID));
   if (file != null) {
      String filename = file.getName();
      String path = file.getAbsolutePath();
      file.delete();
      File newFile = writeStringToFile(filename, content)
         ;
      if (newFile != null) {
         return true;
      }

   }
   return false;
}
```

```
    /**
     * Updates a Users password.
     * @param username
     *            The userID of the User to be updated.
     * @param password
     *            The new Password.
     * @return If the User was updated successfully.
     * @author andreasbeckmann
     */
    public final boolean updateUser(final String username,
        final String password) {
        return database.updateUser(database.getUID(username),
            password);
    }

    /**
     * Returns a List with all fileIDs linked to a User.
     * @param username the Name of the USer
     * @return a ArrayList of Integers
     */
    public ArrayList<Integer> getUserFiles(String username) {
        return database.getUserFiles(database.getUID(username)
            );
    }
}
```

**Listing A.2:** Quellcode der Gateway-Klasse

```
package carismasoapgateway;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.xml.soap.SOAPException;
import javax.xml.ws.Holder;

import carisma.core.analysis.Analysis;
import carisma.core.analysis.CheckReference;
import carisma.core.checks.CheckDescriptor;
import carisma.core.checks.CheckParameterDescriptor;
import carisma.core.checks.ParameterType;
import carisma.core.models.ModelType;
```

```java
import carismacontroller.Controller;
import carismaws.AnalysisResponse;
import carismaws.BooleanParameter;
import carismaws.CarismaWS;
import carismaws.Check;
import carismaws.CheckList;
import carismaws.FileReference;
import carismaws.FloatParameter;
import carismaws.FolderParameter;
import carismaws.InputFileParameter;
import carismaws.IntegerParameter;
import carismaws.OutputFileParameter;
import carismaws.ReadUserResp;
import carismaws.StringParameter;

public class Gateway implements CarismaWS {

    Controller controller;

    /**
     * Called to save a File in the backend.
     *
     * @param file
     *            The {@link carismaws.File} Object to be
     *  saved
     */
    @Override
    public FileReference createFile(carismaws.File file) {

        FileReference fileReference = new FileReference();
        if (file == null) {
            fileReference.setFileID("-1");
            return fileReference;
        }
        controller = Controller.getInstance();
        int fileID = controller.createFileWithContent(file.
            getFilename(),
                file.getContent());
        fileReference.setFileID(String.valueOf(fileID));
        fileReference.setFilename(file.getFilename());
        return fileReference;
    }

    /**
```

```
     * Creates a new User with the given Information if non
       other User with the
     * same username exists.
     *
     * @param username
     *             the username of the User to be created
     * @param password
     *             the password of the User to be created
     * @return if the creation was succesfull
     * @author andreasbeckmann
     */
    @Override
    public boolean createUser(String username, String
        password) {
        controller = Controller.getInstance();
        boolean success = controller.createUser(username,
            password);
        return success;
    }

    /**
     * Deletes the file with the fileID.
     *
     * @param fileID
     *             the unique ID of the File to be deleted
     * @return if the deletion was succesfull
     */
    @Override
    public boolean deleteFile(String fileID) {
        try {
            controller = Controller.getInstance();
            return controller.deleteFile(Integer.valueOf(fileID
                ));
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Deletes the user with the according Username
     *
     * @param username
     *             The username of the User to be deleted.
     * @author andreasbeckmann
     */
```

```
@Override
public boolean deleteUser(String username) {
    controller = Controller.getInstance();
    boolean success = controller.deleteUser(username);
    return success;
}

/**
 * Called when a SOAP Message containing a
     getAvailableChecks Requests
 * arrives.
 *
 * @return a {@link CheckList} Object representing the
     Response Message
 * @author andreasbeckmann
 */
@Override
public CheckList getAvailableChecks() {
    controller = Controller.getInstance();
    CheckList checkListResponse = new CheckList();
    List<CheckDescriptor> checks = controller.getChecks();
    // For every registerd Check parse the Information
        into a list
    for (CheckDescriptor checkDesc : checks) {
        Check check = new Check();
        check.setCheckID(checkDesc.getCheckDescriptorId());
        check.setEnabled(true);
        // For every CheckParameterDescriptor check which
            Parametertype it
        // is and add an accordign Descriptor to the
            Response
        for (CheckParameterDescriptor checkParDesc :
            checkDesc
                .getParameters()) {
            ParameterType type = checkParDesc.getType();
            carismaws.CheckParameterDescriptor
                parameterDescriptor =
                parseCheckParameterDescriptor(checkParDesc);
            if (type == ParameterType.STRING) {
                carismaws.StringParameter parameter =
                    parseStringParameter(parameterDescriptor);
                check.getStringParamter().add(parameter);
            } else if (type == ParameterType.INTEGER) {
```

```
                carismaws.IntegerParameter parameter =
                    parseIntegerParameter(parameterDescriptor)
                    ;
                check.getIntegerParameter().add(parameter);
            } else if (type == ParameterType.FLOAT) {
                carismaws.FloatParameter parameter =
                    parseFloatParameter(parameterDescriptor);
                check.getFloatParameter().add(parameter);
            } else if (type == ParameterType.BOOLEAN) {
                carismaws.BooleanParameter parameter =
                    parseBooleanParameter(parameterDescriptor)
                    ;
                check.getBooleanParameter().add(parameter);
            } else if (type == ParameterType.INPUTFILE) {
                carismaws.InputFileParameter parameter =
                    parseInputFileParameter(
                    parameterDescriptor);
                check.getInputFileParamter().add(parameter);
            } else if (type == ParameterType.OUTPUTFILE) {
                carismaws.OutputFileParameter parameter =
                    parseOutputFileParameter(
                    parameterDescriptor);
                check.getOutputFileParameter().add(parameter)
                    ;
            } else if (type == ParameterType.FOLDER) {
                carismaws.FolderParameter parameter =
                    parseFolderParameter(parameterDescriptor);
                check.getFolderParameter().add(parameter);
            } else {
                return null;
            }

        }
        checkListResponse.getCheck().add(check);
    }

    return checkListResponse;
}

/**
 * Called to get the Content of a File saved on the
 *   server.
 *
 * @param fileID
 *            The unique file ID of the File
```

```
 * @return The {@link carismaws.File} Object of the File
    requested
 * @author andreasbeckmann
 */
@Override
public carismaws.File readFile(String fileID) {
   try {
      controller = Controller.getInstance();
      File file = controller.getFile(Integer.valueOf(
         fileID));
      if (file != null) {
         carismaws.File result = new carismaws.File();
         result.setFilename(file.getName());
         BufferedReader br = null;
         if (file.isFile()) {
            // Write Content into the response
            br = new BufferedReader(new FileReader(file))
               ;
            StringBuilder sb = new StringBuilder();
            String line = br.readLine();
            while (line != null) {
               sb.append(line);
               sb.append('\n');
               line = br.readLine();
            }
            String content = sb.toString();
            result.setContent(content);
            br.close();
            return result;
         }
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}

/**
 * Returns the available Informations concerning a User.
 *
 * @param username
 *            The username of the User
 * @return The username of the User and a List with {
    @link FileReference}
 *         Objects referencing the Users Files.
```

```
 */
@Override
public ReadUserResp readUser(String username) {
    controller = Controller.getInstance();
    ReadUserResp response = new ReadUserResp();
    response.setUsername(username);
    ArrayList<Integer> fileIDs = controller.getUserFiles(
        username);
    // For every file registered to the User create a
        FileReference and add
    // it to the list in the response
    if (fileIDs != null) {
        for (int fileID : fileIDs) {
            FileReference fileRef = new FileReference();
            fileRef.setFileID(String.valueOf(fileID));
            fileRef.setFilename(controller.getFilename(
                fileID));
            response.getFileRef().add(fileRef);
        }
    }
    return response;

}

/**
 * Called when a SOAP Message containing a call to
    runAnalysis arrives
 *
 * @param check
 *              a List of the Checks to be performed {@link
      Check}
 * @param modellFileID
 *              the FileID of the modell to be checked
 * @return a {@link AnalysisResponse} which makes Up the
    AnalysisResponse
 *          Message
 * @author andreasbeckmann
 */
@Override
public AnalysisResponse runAnalysis(List<Check> check,
    String modellFileID) {
    controller = Controller.getInstance();
    AnalysisResponse analysisResponse = new
        AnalysisResponse();
    if (check == null | modellFileID == null) {
```

```
        FileReference fileRef = new FileReference();
        fileRef.setFileID("-1");
        analysisResponse.setResultFileRef(fileRef);
        return analysisResponse;
    }
    // String filepath = databaseHelper.getFilePath(
        modellFileID, 0);
    File modelFile = controller.getFile(Integer.valueOf(
        modellFileID));
    if (modelFile == null || !modelFile.isFile()) {
        FileReference fileRef = new FileReference();
        fileRef.setFileID("-1");
        analysisResponse.setResultFileRef(fileRef);
        return analysisResponse;
    }
    String modelType = "";
    String extension = "";

    int i = modelFile.getName().lastIndexOf('.');
    if (i > 0) {
        extension = modelFile.getName().substring(i + 1);
    }
    ModelType type = controller.getTypeForFileExtension(
        extension);
    // create a Analysis with the Information in the
        Request
    Analysis analysis = new Analysis(String.valueOf(new
        Date().getTime()),
            type.getName(), modelFile);

    // For every Check defined in the Message parse the
        parameters and
    // create a Carisma.Check
    for (Check in : check) {
        CheckReference checkRef = new CheckReference(in.
            getCheckID(), true);
        for (StringParameter parameter : in.
            getStringParamter()) {
            carismaws.CheckParameterDescriptor checkParDesc
                = parameter
                    .getCheckParameterDescriptor();
            CheckParameterDescriptor
                checkParameterDescriptor = new
                CheckParameterDescriptor(
```

```
            checkParDesc.getId(), checkParDesc.getName
                (),
            checkParDesc.getDescription(),
                ParameterType.STRING,
            false, checkParDesc.getDefaultValue());
    carisma.core.analysis.StringParameter e = new
        carisma.core.analysis.StringParameter(
            checkParameterDescriptor);
    e.setValue(parameter.getValue());
    e.setQueryOnDemand(false);
    checkRef.getParameters().add(e);
}
// for every IntegerParameter in the Request create
    a
// Carisma.IntegerParameter and add it to the
    Carisma.CheckReference
for (IntegerParameter parameter : in.
    getIntegerParameter()) {
    carismaws.CheckParameterDescriptor checkParDesc
        = parameter
            .getCheckParameterDescriptor();
    CheckParameterDescriptor
        checkParameterDescriptor = new
        CheckParameterDescriptor(
            checkParDesc.getId(), checkParDesc.getName
                (),
            checkParDesc.getDescription(),
                ParameterType.INTEGER,
            false, checkParDesc.getDefaultValue());
    carisma.core.analysis.IntegerParameter e = new
        carisma.core.analysis.IntegerParameter(
            checkParameterDescriptor);
    e.setValue(parameter.getValue());
    e.setQueryOnDemand(false);
    checkRef.getParameters().add(e);
}
// for every FloatParameter in the Request create a
// Carisma.FloatParameter and add it to the Carisma
    .CheckReference
for (FloatParameter parameter : in.
    getFloatParameter()) {
    carismaws.CheckParameterDescriptor checkParDesc
        = parameter
            .getCheckParameterDescriptor();
```

```
    CheckParameterDescriptor
        checkParameterDescriptor = new
        CheckParameterDescriptor(
            checkParDesc.getId(), checkParDesc.getName
                (),
            checkParDesc.getDescription(),
                ParameterType.FLOAT,
            false, checkParDesc.getDefaultValue());
    carisma.core.analysis.FloatParameter e = new
        carisma.core.analysis.FloatParameter(
            checkParameterDescriptor);
    e.setValue((Float) parameter.getValue());
    e.setQueryOnDemand(false);
    checkRef.getParameters().add(e);

}
// for every BooleanParameter in the Request create
    a
// Carisma.BooleanParameter and add it to the
    Carisma.CheckReference
for (BooleanParameter parameter : in.
    getBooleanParameter()) {
    carismaws.CheckParameterDescriptor checkParDesc
        = parameter
            .getCheckParameterDescriptor();
    CheckParameterDescriptor
        checkParameterDescriptor = new
        CheckParameterDescriptor(
            checkParDesc.getId(), checkParDesc.getName
                (),
            checkParDesc.getDescription(),
                ParameterType.BOOLEAN,
            false, checkParDesc.getDefaultValue());
    carisma.core.analysis.BooleanParameter e = new
        carisma.core.analysis.BooleanParameter(
            checkParameterDescriptor);
    e.setValue(parameter.isValue());
    e.setQueryOnDemand(false);
    checkRef.getParameters().add(e);

}
// for every FolderParameter in the Request create
    a
// Carisma.FolderParameter and add it to the
    Carisma.CheckReference
```

```
for (FolderParameter parameter : in.
   getFolderParameter()) {
   carismaws.CheckParameterDescriptor checkParDesc
      = parameter
         .getCheckParameterDescriptor();
   CheckParameterDescriptor
      checkParameterDescriptor = new
      CheckParameterDescriptor(
         checkParDesc.getId(), checkParDesc.getName
            (),
         checkParDesc.getDescription(),
            ParameterType.FOLDER,
         false, checkParDesc.getDefaultValue());
   carisma.core.analysis.FolderParameter e = new
      carisma.core.analysis.FolderParameter(
         checkParameterDescriptor);
   Date now = new Date();
   File folder = new File(String.valueOf(now.
      getTime()) + "/"
         + parameter.getValue());
   folder.mkdirs();
   e.setValue(folder);
   e.setQueryOnDemand(false);
   checkRef.getParameters().add(e);

}
// for every ImputFileParameter in the Request
   create a
// Carisma.InputFileParameter and add it to the
// Carisma.CheckReference
for (InputFileParameter parameter : in.
   getInputFileParamter()) {
   carismaws.CheckParameterDescriptor checkParDesc
      = parameter
         .getCheckParameterDescriptor();
   CheckParameterDescriptor
      checkParameterDescriptor = new
      CheckParameterDescriptor(
         checkParDesc.getId(), checkParDesc.getName
            (),
         checkParDesc.getDescription(),
            ParameterType.INPUTFILE,
         false, checkParDesc.getDefaultValue());
   carisma.core.analysis.InputFileParameter e = new
       carisma.core.analysis.InputFileParameter(
```

```
            checkParameterDescriptor);
        File inputFile = controller.getFile(Integer.
            valueOf(parameter
                .getValue()));
        if (!inputFile.isDirectory()) {
            e.setValue(inputFile);
            e.setQueryOnDemand(false);
            checkRef.getParameters().add(e);
        }
    }
    // for every OutputFileParameter in the Request
        create a
    // Carisma.OutputFileParameter and add it to the
    // Carisma.CheckReference
    for (OutputFileParameter parameter : in.
        getOutputFileParameter()) {
        carismaws.CheckParameterDescriptor checkParDesc
            = parameter
                .getCheckParameterDescriptor();
        CheckParameterDescriptor
            checkParameterDescriptor = new
            CheckParameterDescriptor(
                checkParDesc.getId(), checkParDesc.getName
                    (),
                checkParDesc.getDescription(),
                ParameterType.OUTPUTFILE, false,
                checkParDesc.getDefaultValue());
        carisma.core.analysis.OutputFileParameter e =
            new carisma.core.analysis.OutputFileParameter
            (
                checkParameterDescriptor);
        int fid = controller.createFileWithoutContent(
            parameter
                .getValue());
        File outputFile = controller.getFile(fid); //
            create a new File
        e.setValue(outputFile);
        e.setQueryOnDemand(false);
        checkRef.getParameters().add(e);
        FileReference fileRef = new FileReference();
        fileRef.setFileID(String.valueOf(fid));
        fileRef.setFilename(parameter.getValue());
        analysisResponse.getOutputFileRef().add(fileRef)
            ;
    }
```

```
        analysis.getChecks().add(checkRef); // add the
           parsed
                                         // Carisma.
                                            CheckReference to the
                                         // List of Checks

    }
    int fileID = controller.runAnalysis(analysis);

    if (fileID != -1) {
        analysisResponse.setSuccesful(true);
        FileReference resultFileRef = new FileReference();
        resultFileRef.setFileID(String.valueOf(fileID));
        resultFileRef.setFilename(controller.getFilename(
           fileID));
        analysisResponse.setResultFileRef(resultFileRef);
    } else {
        analysisResponse.setSuccesful(false);
    }
    return analysisResponse;

}

/**
 * Called to Update a Users Password.
 *
 * @param username
 *            The Users username
 * @param password
 *            The Users new password
 * @return if the Password was changed successfully
 * @author andreasbeckmann
 */
@Override
public boolean updateUser(String username, String
    password) {
    controller = Controller.getInstance();
    return controller.updateUser(username, password);
}

/**
 * @param parameterDescriptor
 * @return
 * @author andreasbeckmann
 */
```

```
private BooleanParameter parseBooleanParameter(
      carismaws.CheckParameterDescriptor
         parameterDescriptor) {
   carismaws.BooleanParameter parameter = new carismaws.
      BooleanParameter();
   parameter.setCheckParameterDescriptor(
      parameterDescriptor);
   parameter.setQueryOnDemand(false);
   return parameter;
}

/**
 *
 * @param checkParDesc
 * @return
 * @author andreasbeckmann
 */
private carismaws.CheckParameterDescriptor
   parseCheckParameterDescriptor(
      CheckParameterDescriptor checkParDesc) {
   carismaws.CheckParameterDescriptor parameterDescriptor
       = new carismaws.CheckParameterDescriptor();
   parameterDescriptor.setDefaultValue(checkParDesc.
      getDefaultValue());
   parameterDescriptor.setDescription(checkParDesc.
      getDescription());
   parameterDescriptor.setId(checkParDesc.getID());
   parameterDescriptor.setName(checkParDesc.getName());
   parameterDescriptor.setOptional(checkParDesc.
      isOptional());
   carismaws.ParameterType parameterType = new carismaws.
      ParameterType();
   parameterDescriptor.setType(parameterType);
   return parameterDescriptor;
}

/**
 *
 * @param parameterDescriptor
 * @return
 * @author andreasbeckmann
 */
private FloatParameter parseFloatParameter(
      carismaws.CheckParameterDescriptor
         parameterDescriptor) {
```

```
        carismaws.FloatParameter parameter = new carismaws.
            FloatParameter();
        parameter.setCheckParameterDescriptor(
            parameterDescriptor);
        parameter.setQueryOnDemand(false);
        return parameter;
    }

    /**
     *
     * @param parameterDescriptor
     * @return
     * @author andreasbeckmann
     */
    private FolderParameter parseFolderParameter(
            carismaws.CheckParameterDescriptor
                parameterDescriptor) {
        FolderParameter parameter = new FolderParameter();
        parameter.setCheckParameterDescriptor(
            parameterDescriptor);
        parameter.setQueryOnDemand(false);
        return parameter;
    }

    /**
     *
     * @param parameterDescriptor
     * @return
     * @author andreasbeckmann
     */
    private InputFileParameter parseInputFileParameter(
            carismaws.CheckParameterDescriptor
                parameterDescriptor) {
        carismaws.InputFileParameter parameter = new carismaws
            .InputFileParameter();
        parameter.setCheckParameterDescriptor(
            parameterDescriptor);
        parameter.setQueryOnDemand(false);
        return parameter;
    }

    /**
     *
     * @param parameterDescriptor
     * @return
```

```
 * @author andreasbeckmann
 */
private IntegerParameter parseIntegerParameter(
      carismaws.CheckParameterDescriptor
         parameterDescriptor) {
   carismaws.IntegerParameter parameter = new carismaws.
      IntegerParameter();
   parameter.setCheckParameterDescriptor(
      parameterDescriptor);
   parameter.setQueryOnDemand(false);
   return parameter;
}

/**
 *
 * @param parameterDescriptor
 * @return
 * @author andreasbeckmann
 */
private OutputFileParameter parseOutputFileParameter(
      carismaws.CheckParameterDescriptor
         parameterDescriptor) {
   OutputFileParameter parameter = new carismaws.
      OutputFileParameter();
   parameter.setCheckParameterDescriptor(
      parameterDescriptor);
   parameter.setQueryOnDemand(false);
   return parameter;
}

/**
 *
 * @param parameterDescriptor
 * @return
 * @author andreasbeckmann
 */
private StringParameter parseStringParameter(
      carismaws.CheckParameterDescriptor
         parameterDescriptor) {
   carismaws.StringParameter parameter = new carismaws.
      StringParameter();
   parameter.setCheckParameterDescriptor(
      parameterDescriptor);
   parameter.setQueryOnDemand(false);
   return parameter;
```

```
    }

    /**
     * Called to Update the content of an existing File.
     *
     * @param fileID
     *            The unique ID of the File to be updated
     * @param file
     *            a {@link carismaws.File} Object containing
     *     the new Content.
     */
    @Override
    public FileReference updateFile(String fileID, carismaws.
       File file) {
       Integer fileIDInt = Integer.valueOf(fileID);
       controller = Controller.getInstance();
       controller.updateFile(fileID, file.getContent());
       FileReference fileRef = new FileReference();
       fileRef.setFileID(fileID);
       fileRef.setFilename(controller.getFilename(fileIDInt))
          ;
       return fileRef;
    }

}
```

**Listing A.3:** Quellcode des RestInterface-Interfaces

```
package carismarest;

import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;

/**
 *
 * @author andreasbeckmann
 *
 */
@Path("carisma")
```

```
public interface RestInterface {

    /**
     * @author andreasbeckmann
     * @return The Available Checks in a String formatted as
         JsonObject.
     */
    @GET
    @Path("getAvailableChecks/")
    @Produces(MediaType.TEXT_PLAIN)
    String getAvailableChecks();

    //CRUD File Operations

    //Create File
    /**
     *
     * @param filename The Name of the new File.
     * @param content The Content of the new File.
     * @return A String containing a Reference to the File
         formatted as JsonObject.
     */
    @POST
    @Path("file")
    @Produces(MediaType.TEXT_PLAIN)
    String createFile(@QueryParam("filename") String filename
        , @QueryParam("content")String content);

    // Read File
    /**
     *
     * @param fileID The fileID of the requested File
     * @return The content and username of the File as String
         .
     */
    @GET
    @Path("file/{fileID}")
    @Produces(MediaType.TEXT_PLAIN)
    String readFile(@PathParam("fileID") int fileID);

    //Update File
    /**
     *
     * @param fileID The ID of the File to be updated.
     * @param content The new Content
```

```
 * @return The result.
 */
@PUT
@Path("file/{fileID}")
@Produces(MediaType.TEXT_PLAIN)
String updateFile(@PathParam("fileID") int fileID,
   @QueryParam("content") String content);

//Delete File
/**
 *
 * @param fileID the FileID of the File to be deleted.
 * @return If the File was deleted successfully. As
     String.
 */
@DELETE
@Path("file/{fileID}")
@Produces(MediaType.TEXT_PLAIN)
String deleteFile(@PathParam("fileID") int fileID);

//CRUD User Operations

//Create User
/**
 *
 * @param username The Username of the user to be created
     .
 * @param password The password of the user to be created
     .
 * @return If the User was created successfull as String.
 */
@POST
@Path("user")
@Produces(MediaType.TEXT_PLAIN)
String createUser(@QueryParam("username") String username
   , @QueryParam("password") String password);


/**
 *
 * @param username The name of the User to be created.
 * @return the Username and references to all of the
     Users Files.
 */
@GET
```

```
@Path("user/{username}")
@Produces(MediaType.TEXT_PLAIN)
String readUser(@PathParam("username") String userID);
//Delete User
/**
 *
 * @param username The name of the User to be deleted.
 * @return If the User was created successfull as String.
 */
@DELETE
@Path("user/{username}")
@Produces(MediaType.TEXT_PLAIN)
String deleteUser(@PathParam("username") String username)
   ;

/**
 *
 * @param username The name of the User to be updated.
 * @param password The new Password.
 * @return If the User was updated successfully.
 */
@PUT
@Path("user/{username}")
@Produces(MediaType.TEXT_PLAIN)
String updateUser(@PathParam("userID") String username,
   @QueryParam("password") String password);

//runAnalysis

/**
 *
 * @param analysis A Analysis Formated as JsonObject as
 *   String.
 * @return A Reference to the Resultfile.
 */
@POST
@Path("runAnalysis")
@Produces(MediaType.TEXT_PLAIN)
String runAnalysis(@QueryParam("analysis") String
   analysis);



}
```

**Listing A.4:** Quellcode des DatabaseInterface-Interfaces

```
package dbinterface;

import java.io.File;
import java.util.ArrayList;
/**
 *
 * @author andreasbeckmann
 *
 */
public interface DatabaseInterface {


    /**
     * Called to insert a new User in the Database. Checks if
        a User with the
     * same name already exists. If not the new entry gets
       inserted.
     *
     * @param username
     *            The name of the User to be created
     * @param password
     *            The Password of the User to be created
     * @return if the creation was succesful
     * @author andreasbeckmann
     */
    boolean insertUser(String username, String password);

    /**
     * Checks if a given username and password match.
     *
     * @param username
     *            the username of the User to check
     * @param password
     *            the password of the User in question
     * @return if the username an password match
     * @author andreasbeckmann
     */
    boolean checkUser(String username, String password);

    /**
     * If a user with the given name exists, it gets deleted.
     *
     * @param username
     *            the name of the User to be deleted
     * @return if a User was deleted
```

```
     * @author andreasbeckmann
     */
    boolean deleteUser(String username);

    /**
     * Gets the User ID of the user.
     *
     * @param username
     *            the name of the user
     * @return the ID if a User was found, else -1
     * @author andreasbeckmann
     */
    int getUID(String username);

    /**
     * Inserts the Path to the given File into the Files
       Table.
     *
     * @param file
     *            the File Object which Path needs to be
       stored
     * @param uid
     *            the uid of the user to be connected to the
       file
     * @return the fileID (-1 if an error occured)
     * @author andreasbeckmann
     */
    int insertFile(File file, int uid);

    /**
     * Returns the filePath of the File corresponding to the
       fileId.
     *
     * @author andreasbeckmann
     * @param fid
     *            the unique File ID of the File
     * @param uid
     *            the User ID
     * @return the Absolute Filepath
     */
    String getFilePath(int fid, int uid);

//  /**
//   * Drop's the Users Table
//   *
```

```
//   * @author andreasbeckmann
//   * @return if the Table was droped succesfully
//   */
// public abstract boolean dropUsersTable();

    /**
     * Prints all avaialable Filepaths of the Files Table in
        the commandLine.
     *
     * @author andreasbeckmann
     */
    void printFiles();

    /**
     *
     * @param fileID The fileID of the File to be deleted.
     * @return If the File was deleted successfully.
     */
    boolean deleteFile(int fileID);

    /**
     *
     * @param uID The Users userID
     * @return the Username.
     */
    String getUserByID(Integer uID);

    /**
     *
     * @param userID the userid of the User to be updated.
     * @param password The new Password.
     * @return If the User was updated successfully.
     */
    boolean updateUser(int userID, String password);

    ArrayList<Integer> getUserFiles(int userID);

}
```

# Literatur


[All13] OSGi Alliance. Online, Zugriff. Dez. 2013. URL: `osgi.org`.

[Bas+09] D. Basin u. a. „Automated analysis of security-design models“. In: *Information and Software Technology* 51.5 (2009), S. 815–831.

[Ber13] S. Beryozkin. Online, Zugriff. Juni 2013. URL: `https://issues.apache.org/jira/browse/DOSGI-166`.

[Bra+07] F. den Braber u. a. „Model-based security analysis in seven steps—a guided tour to the CORAS method“. In: *BT Technology Journal* 25.1 (2007), S. 101–117.

[BRJ06] G. Booch, J. Rumbaugh und I. Jacobson. *Das UML-Benutzerhandbuch: aktuell zur Version 2.0*. Programmer's choice. Addison-Wesley, 2006.

[DuB13] P. DuBois. *MySQL*. Addison-Wesley, 2013.

[Fie00] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. AAI9980887. Diss. 2000.

[Fou13a] Apache Software Foundation. Online, Zugriff. Dez. 2013. URL: `apache.org`.

[Fou13b] Apache Software Foundation. *Apache CXF project*. Online, Zugriff. Dez. 2013. URL: `http://cxf.apache.org/distributed-osgi.html`.

[Fou13c] Mozilla Foundation. Online, Zugriff. Dez. 2013. URL: `mozilla.org/de/firefox`.

[FZ09] P. Finger und K. Zeppenfeld. *SOA und WebServices*. Informatik im Fokus. Springer London, Limited, 2009.

[Gee03] D. Geer. „Taking steps to secure Web services“. In: *Computer* 36.10 (2003), S. 14–16.

[HJM13] M. R. Hoffmann, B. Janiczak und E. Mandrikov. Online, Zugriff. Jan. 2013. URL: `http://http://www.eclemma.org`.

[ISO92] ISO. *ISO/IEC 9075:1992: Title: Information technology — Database languages — SQL*. pub-ISO, 1992.

[Jür05] J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.

[LBD02] T. Lodderstedt, D. Basin und J. Doser. „SecureUML: A UML-Based Modeling Language for Model-Driven Security“. In: *The unified modeling language: model engineering, concepts, and tools; 5th international*. Bd. 2460. Springer, 2002, S. 426–441.

[LSS11] M. S. Lund, B. Solhaug und K. Stølen. „The CORAS Tool“. In: *Model-Driven Risk Analysis*. Springer, 2011, S. 339–346.

[Mar09] R. C. Martin. *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall, 2009.

[MRW11] Edward A. Morse, Vasant Raval und John R. Wingender. „Market Price Effects of Data Security Breaches.“ In: *Information Security Journal: A Global Perspective* 20.6 (2011), S. 263 –273.

[MVA10] J. McAffer, P. VanderLei und S. Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley, 2010.

[Owe06] Mike Owens. *The definitive guide to SQLite*. Apress, 2006.

[RB06] G. Rooney und D. Berlin. *Practical subversion*. Apress, 2006.

[Roe+10] R. Roelofsen u. a. „Think Large, Act Small: An Approach to Web Services for Embedded Systems Based on the OSGi Framework“. In: *Exploring Services Science*. Springer, 2010, S. 239–253.

[RR07] L. Richardson und S. Ruby. *Web Services mit REST*. O'Reilly Germany, 2007.

[Sof13] SmartBear Software. Online, Zugriff. Dez. 2013. URL: `smartbear.com`.

[TP02] A. Tsalgatidou und T. Pilioura. „An overview of standards and related technology in web services“. In: *Distributed and Parallel Databases* 12.2-3 (2002), S. 135–162.

[Wan+04] H. Wang u. a. „Web services: problems and future directions“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.3 (2004), S. 309–320.

[WW13] S. Wenzel und D. Warzecha. Online, Zugriff. Dez. 2013. URL: `http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php`.

[Wüt+08] G. Wütherich u. a. *Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox*. dpunkt, 2008.