Eine Microservice-Architektur zum konversationsbasierten Erkunden von Softwarevisualisierungen

Bachelorarbeit vorgelegt von Stefan Bieliauskas

Angefertigt für den schriftlichen Teil der Bachelorprüfung im
Internationalen Studiengang Medieninformatik
an der Hochschule Bremen,
Fachbereich Elektrotechnik und Informatik

Wintersemester 2016/2017

Erstprüfer:	Prof. Dr. Thorsten Teschke
Zweitprüfer:	Prof. Dr. Lars Braubach

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

LOC	Lines of Code
DLR	Deutsches Zentrum für Luft- und Raumfahrt
Sofia	Software "Framework" for Interaction with Software Architecture
NLP	Natrual language Proccesing
RCE	Remote Component Environment
SOA	Service Oriented Architecture
CI	Continuous Integration
EDA	event driven architecture
OSGi	Open Service Gateway initiative
HTTP	Hypertext Transfer Protocol
REST	Representational state transfer
SSL	Secure Socket Layer
WAMP	Web Application Message Protocol
URI	Unified Resource Identifier
RPC	Remote Procedure Call
ISC	Inter service communication
AMQP	Advanced Message Queuing Protocol
npm	Node package manager

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit fachlich und persönlich unterstützt und motiviert haben.

Besonders möchte ich mich bei Herrn Prof. Dr. Thorsten Teschke bedanken, der meine Bachelorarbeit betreut und begutachtet. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken. Zudem gilt mein Dank auch Herrn Prof. Dr. Lars Braubach, der mir als Zweitkorrektor unterstützend zur Seite stand.

Abschließend bedanken möchte ich mich bei meinen Eltern, die mir dieses Studium ermöglicht haben und auf deren Unterstützung ich immer zählen kann und konnte; zudem bei meinen Freunden, die mich während dieser Zeit unterstützten und begleiteten.

Stefan Bieliauskas

Abstract

Die nachfolgende Bachelorarbeit umfasst die Konzeption und prototypische Implementierung einer Microservice Architektur, zur Interaktion über ein konversationsbasiertes Userinterface [61]. Dabei wird am Beispiel der Userinteraktion mit einem Chat-Bot eine abstrakte Softwarevisualisierung zu einem OSGi Projekt gesteuert. Die Architektur nutzt das WAMP V2 Protokoll zur Kommunikation unter den einzelnen Services und zeigt Vorteile bei der Anbindung neuer Services auf. Durch die Implementierung dieser Architektur traten dabei auch Schwachstellen auf. Beispielsweise war für eine effektive Entwicklung eine komplexe lokale Einrichtung mit mehreren Abhängigkeiten notwendig. Die Nutzung des Microservice Pattern in Kombination mit dem WAMP V2 Protokoll ergab dabei eine hochflexible Lösung, bei denen die einzelnen Services eine geringe Komplexität aufwiesen.

1. Kapitel 1 – Einleitung

Diese Bachelorarbeit beschreibt eine Systemarchitektur, die es ermöglicht innerhalb eines Chats mit Softwarevisualisierungen zu interagieren.

1.1 Motivation

Die Komplexität von Softwareprojekten, gemessen anhand der Lines of Code (LOC) [1], ist in den vergangenen Jahren deutlich angestiegen [2]. Dies stellt Entwickler vor neue Herausforderungen, einerseits im Hinblick auf die Einarbeitungsphase in bestehende Projekte [3] andererseits auch durch den Wandel des Softwareentwicklungsprozesses hin zu verteilten Teams [4] [5]. Zu denen laut der wissenschaftlichen Mitarbeiterin Eirini Kalliamvakou, der University of Victoria, ebenfalls die wachsende Open-Source Community gezählt werden kann [5].

Die Problematik während der Einarbeitungsphase neuer Entwickler besteht häufig in der unzulänglichen Betrachtung von großen Projekten, in Form von Quellcode. Zum effektiven Verständnis der Zusammenhänge sind Visualisierungen auf einer höheren Abstraktionsebene notwendig [6].

Der Bereich der Softwarevisualisierung bietet mit verschiedenen Metaphern und Methoden die Möglichkeit die Zusammenhänge abstrakt darzustellen. Zu dieser Thematik entstanden im Deutschen Zentrum für Luft und Raumfahrt (DLR) bereits mehrere wissenschaftliche Arbeiten und Prototypen [7] [8].

Diese Arbeiten zeigen neue Ansätze für die Visualisierung von Source-Code, basieren jedoch noch auf einer manuellen Extraktion von Metadaten, wie Paket- und Klassennamen, des Source-Code. Entwickler haben demnach nicht die Möglichkeit immer auf aktuelle Visualisierungen zurück zu greifen [7].

Des Weiteren ist die Benutzerinteraktion nur prototypisch umgesetzt. Sodass sich hier ein weites Forschungsfeld zur Steigerung des Nutzens und der besseren Integration in den Arbeitsprozess des Entwicklers bietet.

1.2 Problemfeld

Die entstandenen Arbeiten zu den Themen "Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte" [7] sowie "Visualisierung von mit OSGi-Komponenten realisierten Softwarearchitekturen im 3-dimensionalem Raum mit Virtual Reality" [9] betrachten die Problematiken der Softwarevisualisierung von OSGi-Projekten. Das OSGi Framework ist eine Service Delivery Platform, die es ermöglicht Java Projekte als Komponenten (Bundles genannt) zu strukturieren und separat in der JVM bereit zu stellen [10]. Wie in Abschnitt 3.4 ausführlich erläutert wird.

In den Arbeiten von Herrn Marquardt [7] und Frau Brüggemann [8] geht es um konkrete Fragestellungen eines Benutzers wie beispielsweise:

- Wo wird Klasse xyz eigentlich alles benutzt?
- Gibt es Bundles, die keine Abhängigkeiten besitzen?
- Welche Abhängigkeiten sind zu beachten, wenn ich die Methode xyz ändere?

Diese Fragestellungen konnten bereits beantwortet werden [9], jedoch ist die Zugänglichkeit dieser Ergebnisse aktuell noch problematisch. Es gibt zurzeit kein System, welches die Ergebnisse der vergangenen Prototypen zusammenfasst und Entwicklern direkt und komfortabel bereitstellt. Wodurch die neuen Möglichkeiten der Visualisierung nur eingeschränkt nutzbar sind.

In diesem Problemfeld, mit komplexen Fragestellungen, bieten Conversational Interfaces einen neuen Ansatz [11]. Dieser ermöglicht es dem Nutzer, seine Fragen in natürlicher Sprache zu formulieren. Die entsprechenden Antworten werden, je nach Fragestellung passend z. B. als Bild oder Text präsentiert. Diese Art der Interaktion ist in diesem Kontext noch nicht erprobt und bedarf einer Evaluation.

1.3 Ziele und Aufgaben

Ziel dieser Arbeit ist es eine Microservice Architektur zu konzipieren und prototypisch zu implementieren. Der Prototyp ermöglicht es dem Nutzer mit Softwarevisualisierungen über ein Conversational Interface via Text oder Bild [12]

zu interagieren und Fragen wie in in Abschnitt 1.2 beschrieben zu beantworten.

Die Implementierung einzelner Komponenten steht hierbei nicht im Vordergrund. Beispielsweise wird nicht näher auf Natrual Language Proceccing (NLP) eingegangen.

Der Fokus liegt auf der Architektur für ein Conversational Interface. Hierbei wird der Frage nachgegangen, wie genau die Services untereinander kommunizieren können, um Eigenschaften wie Austauschbarkeit und Erweiterbarkeit bzw. allgemein gefasst Wartbarkeit / Übertragbarkeit nach ISO 25000:2014 [13] zu unterstützen.

1.4 Anwendungskontext

Diese Arbeit ist in Zusammenarbeit mit dem Deutschen Zentrum für Luft und Raumfahrt entstanden.

Das DLR ist eine der führenden Forschungseinrichtungen im Luft und Raumfahrtsektor. Mit mehr als 8.000 Mitarbeitern und einem Budget von mehr als 880 Mio. Euro pro Jahr ist das DLR eine der größten Forschungseinrichtungen in Deutschland [14].

Die Einrichtung für Simulations- und Softwaretechnik mit Sitz in Köln, Braunschweig und Berlin unterteilt sich in drei Abteilungen [15]:

- Intelligente und verteilte Systeme
- Software für Raumfahrtsysteme und interaktive Visualisierung
- High-Performance Computing

Die Abteilung Intelligente und verteilte Systeme forscht in den Bereichen der Visualisierung von Softwarearchitekturen [16] und entwickelt Software für den multidisziplinären Entwurf von Raum- und Luftfahrtzeugen [17]. Diese Software Remote Component Enviroment (RCE) genannt ist mithilfe des OSGi-Frameworks umgesetzt und ermöglicht es Ingenieuren den Entwurfsprozess von Luftgefährten zu einem großen Teil zu automatisieren.

Dabei ermöglicht RCE komplexe Analyse- und Optimierungsprobleme über mehrere Organisationen hinweg zu automatisieren [18]. RCE wird maßgeblich

von der Abteilung für Intelligente und verteilte Systeme des DLR vorangetrieben, ist quelloffen und erfährt überdies auch erste Anpassungen durch externe Entwickler [19].

1.5 Aufbau der Arbeit

Die Arbeit gliedert sich in sechs Kapitel auf. Nach der Einführung werden in Kapitel 2 – Anforderungsanalyse die Anforderungen, Zielgruppen und Qualitätsmerkmale genauer beschrieben, sowie der funktionale Umfang der Software festgelegt.

Anschließend werden in Kapitel 3 – Grundlagen die Technologien und Terminologien, die für diese Arbeit relevant sind, näher erörtert. Dies schließt beispielsweise die Erläuterung der im Anwendungskontext wesentliche Anwendung RCE, sowie das OSGi-Framework ein. Die Konzeption des Projekts wird in Kapitel 4 – Konzeption erläutert und anschließend dieses Konzept in Kapitel 5 – Umsetzung prototypisch implementiert. Die Ergebnisse und der Ausblick befinden sich in Kapitel 6 – Abschluss.

2. Kapitel 2 – Anforderungsanalyse

2.1 Aktueller Status und Problemstellung

Während der Weiterentwicklung des RCE Projekts werden die Mitarbeiter, die an der Entwicklung kontinuierlich beteiligt sind, immer wieder von studentischen Mitarbeitern unterstützt. Die Einarbeitung neuer Mitarbeiter, sowie das Verständnis der Struktur für externe Personen ist mit der fortscheitenden Komplexität des Projekts eine wachsende Herausforderung.

Auf Basis von RCE ist die Arbeit „Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte“ von Herrn Marquardt entstanden. Diese beschäftigt sich mit der Darstellung von komplexen Softwareprojekten.

Die Ergebnisse dieser Arbeit in Form einer Darstellung für Bundles und Klassen werden für diesen Prototyp in der vorhandenen Form integriert.

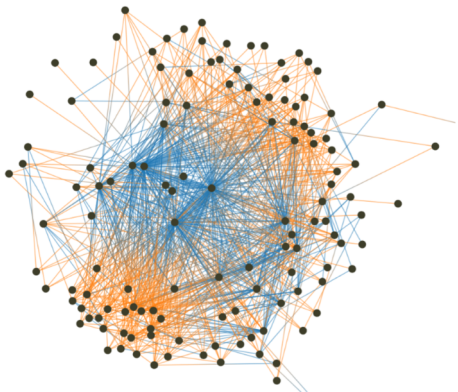

Abbildung 1 - Bundleabhängigkeiten

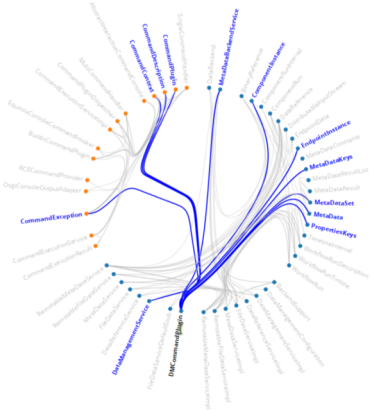

Abbildung 2 - Klassenabhängigkeiten

In Abbildung 1 erkennt man die Im- und Exports zwischen einzelnen OSGi Bundles. Diese Ansicht ermöglicht es Zusammenhänge zwischen einzelnen Bundles zu verdeutlichen. In Abbildung 2 ist dagegen dargestellt wo überall eine bestimmte Klasse importiert wird. Dies gestattet es eventuelle Auswirkungen einer Änderung an der Klasse zu bewerten und ggf. auch zu erkennen ob die Klasse eine zentrale Rolle im Projekt spielt [7].

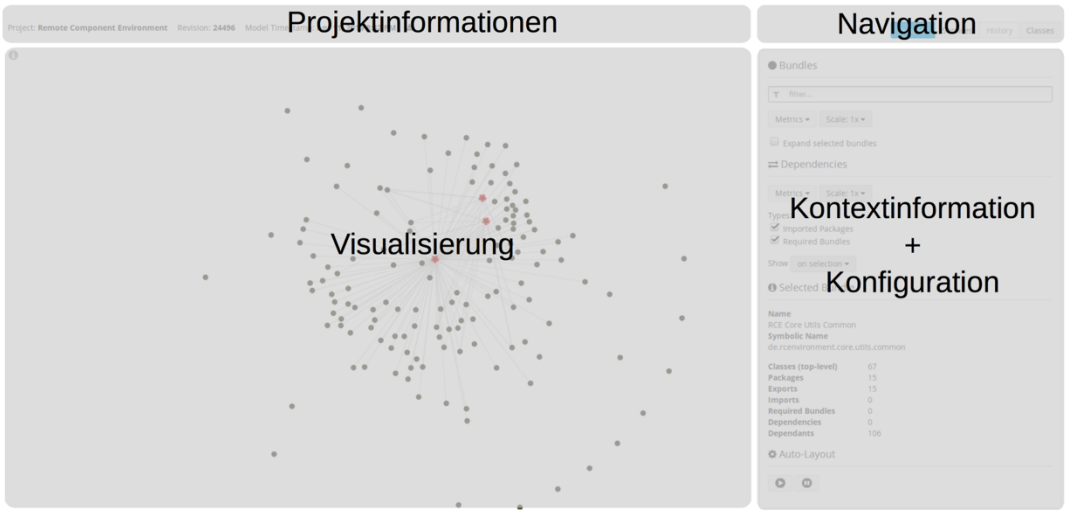

Abbildung 3 - Aktuelle Weboberfläche

Die in Abbildung 3 dargestellte Weboberfläche ermöglicht es mit den in Abbildung 2 und Abbildung 1 dargestellten Visualisierungen zu interagieren. Hierbei bietet das Tool verschiedene Möglichkeiten der Konfiguration wie beispielsweise:

- Anzeige Im- /Exports: Immer, bei Click, bei Mausüberfahrt
- Anzeige der LOC oder Anzeige des Verhältnis public/private Methoden pro Bundle

Das einschränken auf z. B. nur Bundles die eine Beziehung mit dem Feature “Login“ implizieren ist mit dieser Oberfläche nur eingeschränkt, über eine Suche nach dem Paketnamen, möglich.

Die nötigen Daten extrahiert ein Tool zur Analyse eines SVN Repositories, jedoch wird dieses Tool im Moment nur manuell konfiguriert und gestartet. Sodass nicht immer aktuelle Darstellungsdaten zum Projekt zur Verfügung stehen.

2.2 Zielgruppen

Softwarevisualisierungen bieten unterschiedlichen Zielgruppen verschiedene Informationen an. Hierbei müssen diese Zielgruppen mit ihren Schwerpunkten zunächst festgelegt werden.

Für den in 2.1 beschriebenen Problemkomplex kristallisieren sich mehrere Zielgruppen heraus, die der Tabelle 1 zu entnehmen sind.

Zielgruppe	Ziele
Entwickler – (Projektkenntnis)	- Auswirkungen bei Änderungen von Schnittstellen; Welche anderen Bundles sind betroffen?
Entwickler – (Einsteiger)	- Überblick über die Architektur - Wo ist das Feature xyz
Architekt	- Schwachstellen / Verletzung der Architekturrichtlinien feststellen

Tabelle 1 - Zielgruppen

Auch wenn die Ziele zwischen den beiden Entwicklergruppen überlappen können wird zwischen Entwicklern mit Erfahrungen im Projekt und neuen Mitarbeitern bzw. einem externen Entwickler, der ggf. einmalig etwas zur Entwicklung beisteuern möchte, unterschieden.

Die Unterteilung basiert auf der Annahme, dass ein Entwickler der neu in das Projekt einsteigt sich zunächst einmal einen Überblick über die Architektur verschafft. Dies geschieht in der Regel durch das Einholen von Informationen von Entwicklern mit Projektkenntnissen.

2.3 Qualitätskriterien

Im Bereich der Softwareentwicklung wurden über die Jahre verschiedene Qualitätskriterien erarbeitet, die dazu dienen bestimmte Eigenschaften eines Softwareprodukts festzulegen. Der ISO/IEC 9126 Standard stellt verschiedene Kriterien zur Verfügung, die durch ihre Bewertung in wichtig bzw. weniger wichtig Rahmenbedingungen festlegen, die für ein Softwareprojekt gelten sollen. Dieser Standard liegt dieser Arbeit zugrunde.

Der Fokus dieses Prototypens liegt stark auf der Modifizierbarkeit und Analysierbarkeit, welche beide dem Merkmal Änderbarkeit zugeordnet werden kön-

nen. Außerdem wird Wert auf die Benutzbarkeit, sowie die Übertragbarkeit gelegt. Worunter unter anderem Eigenschaften wie Änderbarkeit, Austauschbarkeit und einfache Installation fallen [20].

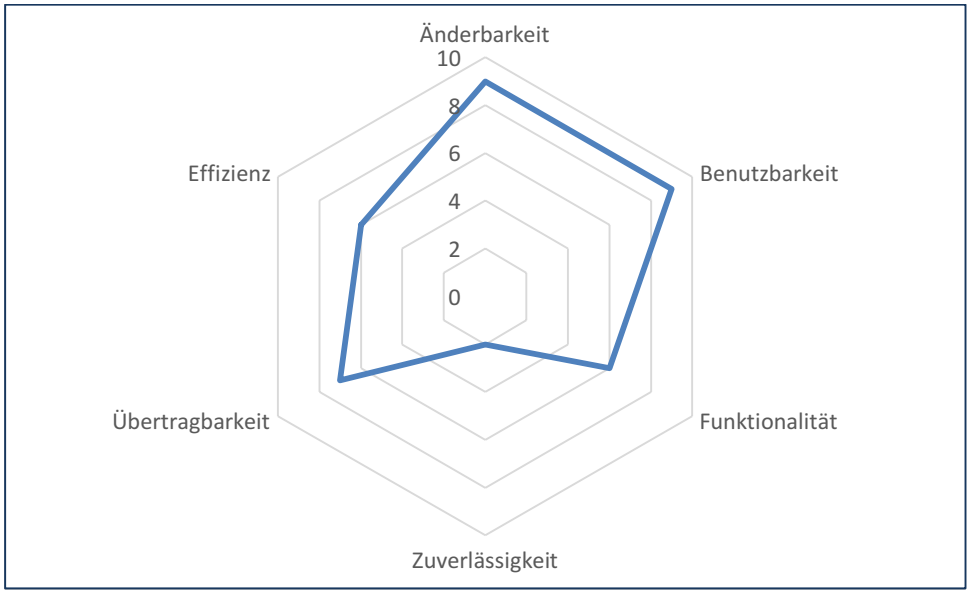

Abbildung 4 - Qualitätskriterien

Der Abbildung 4 ist die Bewertung der sechs Hauptmerkmale zu entnehmen. Hierbei gilt die Wertung von 10 als besonders wichtig, sowie 0 als weniger wichtig. Hierbei erkennt man, dass die Merkmale der Zuverlässigkeit, sowie der Effizienz nicht im Fokus dieses Prototypens stehen sollen.

2.4 Funktionale Anforderungen

Um den Rahmen der Software zu definieren sind neben den in 2.3 genannten Qualitätskriterien ebenso die funktionalen Anforderungen zu beschreiben. Weitere Anforderungen kristallisierten sich aus persönlichen Gesprächen mit den Projektverantwortlichen, sowie aus Fachgespräche mit einzelnen Entwicklern heraus.

Nachfolgend werden die funktionalen Anforderungen in Form von User-Stories beschrieben. User Stories beschreiben in der Fachsprache des Nutzers die Anforderung in einem oder mehreren kurzen Sätzen. Dabei werden Fragen wie wann, wer und was passieren soll beantwortet. Durch die Verwendung der Sprache des Nutzers ergibt sich eine gemeinsame Diskussionsgrundlage über die den Funktionsumfang einer Anforderung. Des Weiteren wird nicht beschrieben wie dies technisch umgesetzt werden soll, somit ergeben sich in der

Konzeption größere Freiheiten als bei einer technisch beschriebenen Anforderung [21].

	User-Story	Darstellung
#1	Als neuer Entwickler in einem Projekt möchte ich mir einen Überblick über die Zusammenhänge in dem Projekt verschaffen.	Bundles
#2	Als erfahrener Entwickler möchte ich erfahren wo eine bestimme Klasse überall verwendet wird.	Klassen
#3	Als erfahrener Entwickler möchte ich wissen in welchem Bundle eine bestimmte Klasse ist und wie dieses Bundle im Zusammenhang mit anderen Bundles steht.	Klassen
#4	Als neuer Entwickler möchte ich mit einem erfahrenen Entwickler über Klassen bzw. Zusammenhänge diskutieren und dabei eine visuelle Unterstützung erhalten.	Klassen / Bundles
#5	Als erfahrener Entwickler möchte ich wissen wo überall eine bestimmte Methode verwendet wird.	Klassen
#6	Als neuer Entwickler möchte ich wissen wo genau sich ein bestimmtes Feature befindet. Beispielsweise möchte ich wissen wo sich denn der Login Prozess befindet.	Bundles

Tabelle 2 - User-Stories

Die in Tabelle 2 aufgeführten User-Stories können grundsätzlich auf zwei verschiedene Visualisierungen zurückgeführt werden.

- Visualisierung von Bundles
- Visualisierung von Klassen

Die Visualisierungen von Bundles und Klassen können mit den in Abbildung 2 und Abbildung 1 gezeigten Ansätzen gelöst werden. Auffällig ist die Anforderung der nebenläufigen Unterstützung innerhalb einer Diskussion in Tabelle 2 - #4. Diese ist, im Gegensatz zu den anderen Anforderungen, mit den aktuellen Tools nur schwer darstellbar.

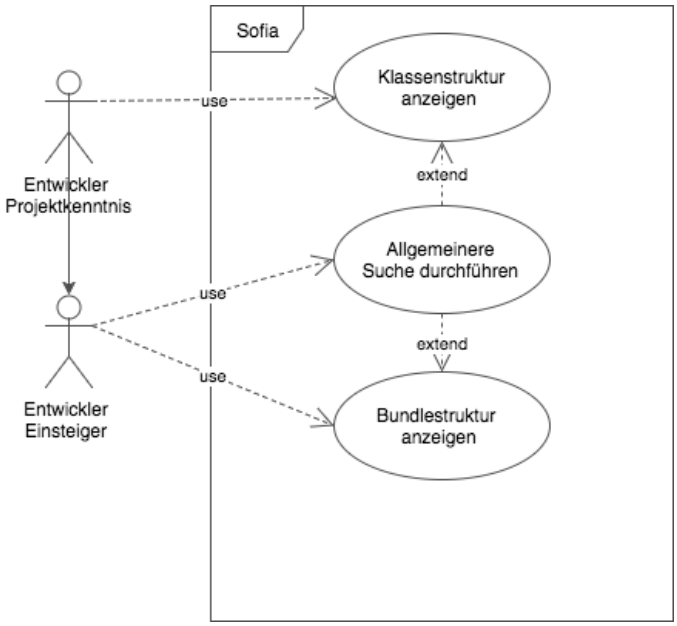

Abbildung 5 - Use Cases

Die in Tabelle 2 gezeigten Anforderungen lassen sich in das in Abbildung 5 gezeigte Use-Case Diagramm überführen. Basierend auf den geführten Fachgesprächen wird davon ausgegangen, dass ein User der Zielgruppe Entwickler (Einsteiger) eher allgemeine Fragen wie beispielsweise: „Suche etwas das mit Login zu tun hat“ stellt.

3. Kapitel 3 – Grundlagen

Das nachfolgende Kapitel bietet einen Überblick über Konzepte und Terminologien, die innerhalb dieser Arbeit relevant sind.

3.1 Microservices

Der Begriff Microservices taucht seit einigen Jahren immer häufiger auf und beschreibt laut Martin Fowler eine Spezialisierung im Bereich der Service Oriented Architecture (SOA) [22]. Dabei ist dieser nicht vereinheitlich oder standardisiert [23].

SOA wurde unter verschiedenen Gesichtspunkten ebenfalls unterschiedlich definiert. Allgemein steht SOA für eine neue Generation von IT-Systemen bei denen die Modularität im Vordergrund steht. Des Weiteren beschreibt SOA u.a. eine neue Art der Zusammenarbeit, bei der die Anforderungen des Nutzers im Mittelpunkt stehen [24] [23].

Eine Microservice Architektur definiert hingegen ein konkretes Softwareprodukt als eine Sammlung aus einzelnen Services, die wiederum jeweils über einfache Protokolle (engl. „dumb pipes“) kommunizieren. Hierbei ist es entscheidend, dass für jeden Service folgendes individuell bestimmt werden kann: Technologische Basis, Art der Bereitstellung des Service und individuelle Skalierungsverfahren [23]. Weiter können ebenfalls Eigenschaften wie das automatische Erstellen und Testen von Services, laut u.a. Fowler und Levis, zu den Charakteristiken einer Microservice Architektur gezählt werden [23] [25].

Dieses automatische Erstellen und Testen eines Service wird zumeist mittels Continuous Integration (CI) umgesetzt [23]. Dabei führen Entwickler mindestens einmal pro Tag ihren Entwicklungsstand zentral zusammen, wodurch das automatische Erstellen und Testen der Software gestartet wird. Dies dient zur frühzeitigen Fehlererkennung [26].

Im Vergleich zu einer monolithischen Anwendung, bei der die gesamte Anwendung in einem entwickelt, bereitgestellt und gestartet wird, sind Microservices individuell skalierbar.

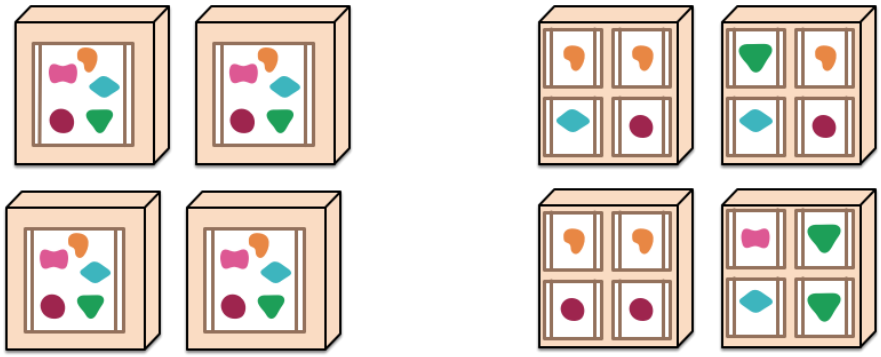

Abbildung 6 - Monolithisch vs. Microservices [23]

Auf der linken Seite von Abbildung 6 sieht man eine monolithische Anwendung, die eine Instanz pro Host, insgesamt vier, zur Verfügung stellt. Dabei bilden die farblich unterschiedlichen Symbole bestimmte Softwarekomponenten ab. Auf der rechten Seite ist zu erkennen, dass die Verteilung der einzelnen Softwarekomponenten nicht mehr an den Host gebunden ist. Die einzelnen Softwarekomponenten, auch Service genannt, können unabhängig voneinander aktualisiert bzw. skaliert werden [23] [27]. Welches das Ziel einer Microservice Architektur ist.

Durch die Verteilung von Anforderungen auf verschiede, voneinander getrennte Softwaresysteme ergeben sich mehrere Herausforderungen, die sich bei einer monolithischen Anwendung nicht in dem Maße ergeben würden. Darunter fallen Fragen bezüglich der Testbarkeit eines verteilten Systems, sowie der Fehlertoleranz beim Ausfall eines einzelnen Service [23]. Des Weiteren ergeben sich neue Möglichkeiten bei der Kommunikation zwischen den Services wie z.B. die in 3.1.1 beschriebene Architektur.

3.1.1 Event-Driven-Arichtecture

Der Begriff der Event-Driven-Architeture (EDA) wurde in den späten Achtziger Jahren im Bereich der Unternehmenssoftware geprägt. Dabei beschreibt dieser ein Pattern indem Nachrichten von Systemen produziert werden und andere Systeme diese konsumieren [28]. EDA wird häufig in Verbindung mit dem in 3.6.1 beschriebenen Publish-Subscribe Pattern verwendet [29]. Wie in 3.6.1 beschrieben wird, ermöglicht dies eine Nachricht an mehrere Teilnehmer zu verteilen.

Im Zusammenspiel mit u.a. dem Publish-Subscribe Pattern kann eine EDA implementiert werden. Durch die Entkopplung von Sender und Empfänger führt dies zu einer größeren Unabhängigkeit untereinander. Die Eigenschaft der Entkopplung verschiedener Softwarekomponenten nennt man auch loose-coupling [28].

3.1.2 Docker

Die Docker Plattform nutzt verschiedene Technologien der Virtualisierung um für den Betrieb von Software die entsprechende Laufzeitumgebung bereit zu stellen [30].

Hierbei werden die folgenden Prinzipien verfolgt:

- Plattformunabhängigkeit
- Wiederverwendbare Komponenten
- Versionierung

Der Vorteil dieser Technologie liegt darin, dass die entsprechende Umgebung, wie beispielsweise benötige Libraries oder Verzeichnisstrukturen, reproduzierbar und versioniert bereitgestellt werden [30].

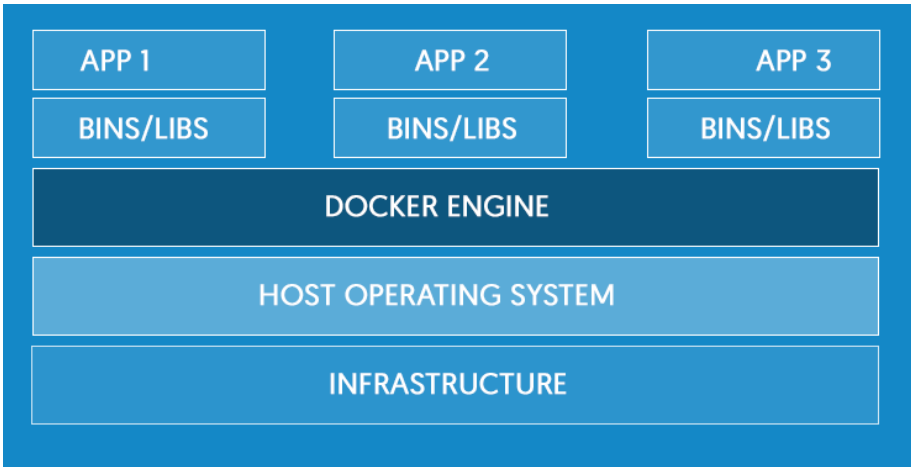

Abbildung 7 - Docker-Stack [31]

In Abbildung 7 ist die zugrundeliegende Struktur zu erkennen, in der einzelne Softwarekomponenten in sogenannten Containern getrennt voneinander ausgeführt werden. Trotz der Tatsache, dass diese drei Container auf demselben Host ausgeführt werden, ist keine separate Betriebssystemvirtualisierung notwendig, da durch die Container eine entsprechende Separation erreicht wird.

Die Docker Technologie ermöglicht somit das APP1 in dem Verzeichnis BINS/LIBS andere Libraries vorfindet als APP2 oder APP3 [31].

Docker wird häufig im Zusammenhang mit dem in 3.1 genannten Microservices Pattern verwendet um die unterschiedlichen Laufzeitumgebungen für die Services bereitzustellen [32].

3.2 Conversational-UI

Konversationsbasierte Userinterfaces zeichnen sich durch die Interaktion auf Basis von Sprach- oder Texteingabe des Nutzers aus. Dabei übernimmt ein Software-Agent oftmals einen aktiven Part in der Konversation [33].

Das Conversational-UI, unter anderem auch als Chatbot oder Bot bezeichnet, beschreibt die Interaktion eines Nutzers mit einem Software-Agent. Grundsätzlich gibt es mehrere Definitionen für Conversational-UIs. Der Begriff wird häufig im Zuge mit Begrifflichkeiten wie: Software Agent, Virtual Agent oder auch Intelligent Personal Agent genannt [34]. Der Begriff Agent wurde von Car Hetwitt als erstes beschrieben als: „A self-contained, interactive and concurrently-executing object, possessing internal state and communication capability.“ [35].

Die Herausforderung bei einem konversationsbasierten Userinterface besteht dabei zu einem in der hochkomplexe Prozess des Verstehens von Sätzen, sodass ein Software-Agent die Anfrage verarbeiten kann [36]. Zum anderen aus der Interaktion mit dem Nutzer. Dabei können Nutzer neue Eingaben an das Interface senden, bevor die bereits gesendete Anfrage verarbeitet wurde oder aber auch die bestehende Anfrage um weitere Informationen ergänzen [37].

Diese Herausforderungen haben direkten Einfluss auf die entsprechenden Anforderungen an ein System zur Verarbeitung von konversationsbasierten Anfragen.

3.3 Softwarevisualisierungen

Der Bereich der Softwarevisualisierung beschreibt ein Teilgebiet der Darstellung von Informationen mittels abstrakter Graphen. Dabei liegt der Schwer-

punkt auf der Darstellung von Zusammenhängen innerhalb eines Softwareprojekts. Ein Grund für eine solche Darstellung sind die hohen Kosten, die während der Wartung eines Softwareprodukts entstehen. Diese Kostensteigerung ist maßgeblich auf die Einarbeitungszeit in die zu wartende Software zurückzuführen [38].

Die Motive eines Softwareentwicklers, der sich durch entsprechende Visualisierungen beim Entwicklungsprozess unterstützen lässt, lassen sich wie folgt einteilen [39]:

- Code Verständnis
- Debugging
- Architektur Entscheidungen

Dabei tragen die in Abbildung 1 und Abbildung 2 gezeigten Visualisierungen zum Code Verständnis und eventuellen Architektur Entscheidungen bei [7].

3.4 OSGi Framework

Die Open Service Gateway initiative (OSGi) beschreibt ein modulares Framework für die Entwicklung von Javaprojekten. Dabei bietet dieses Framework verschiedene Konzepte an, um die Entwicklung komplexer Javaanwendungen zu vereinfachen. Diese Vereinfachung wird u.a. in Form von Bundle- und Service- Konzepten umgesetzt. Diese beschreiben Beziehungen zwischen Programmteilen [40].

Verwendung findet das OSGi Framework in vielen bekannten Open Source Anwendungen, unter anderem sind hier zwei Javaprojekte zu nennen Eclipse [41] und der GlassFish Server [42].

Service

Ein OSGi Service beschreibt eine Java Klasse, die ihre Methoden innerhalb des OSGi Framework anderen Bundles bereitstellt. Dabei wird empfohlen die Java Service Klasse durch die Implementierung eines definierten Interfaces dem OSGi Framework bereitzustellen [40].

Bundle

Ein Bundle ist die kleinste Einheit im OSGi Framework und bietet die Möglichkeit eine Programmkomponente separat zu installieren und zu starten bzw. zu stoppen. Dies ermöglicht es dem OSGi Framework ein Bundle während der Laufzeit zu aktualisieren oder aber zu deaktivieren, ohne das Hauptprogramm zu stoppen [40]. Das Einsatzgebiet dieser Technologie ist vielseitig, beispielsweise ist jedes Eclipse Plug-In ein einzelnes OSGi Bundle. Plug-Ins in Eclipse können einzeln deaktiviert und während der Laufzeit aktualisiert bzw. installiert werden [43].

3.5 HTTP / REST

Das Hyper Text Transport Protokoll (HTTP) wurde durch den RFC 2621 standardisiert und bildet die Grundlage zur Abfrage von Hyper Text Markup Dokumenten (HTML-Dokumenten) via Request-Response Pattern [44]. Die Technologie basiert auf dem TCP Protokoll und stellt somit sicher, dass gesendete Pakete beim Empfänger ankommen [45]. HTTP bietet durch die Erweiterung des RFC 5246 ebenfalls die Möglichkeit der verschlüsselten Verbindung mittels einer Secure Socket Layer (SSL) Verbindung [46].

HTTP wird mittlerweile nicht nur zur Auslieferung von HTML Dokumenten genutzt, sondern oftmals auch als Protokoll zur Kommunikation mit Services. Dabei findet der Austausch immer häufiger unter Verwendung des Represental State Transfer (REST) Pattern statt. Jenes wurde zuerst von Thomas Fielding im Jahre 2002 beschrieben und bietet die Möglichkeit Abfrage-, Lösch- und Erzeuge-Operationen nach einem definiertem Schema über HTTP durchzuführen.

3.6 Web Application Messaging Protocol

Im Gegensatz zum in 3.5 beschriebenem HTTP Protokoll beschreibt das Web Application Message Protokoll (WAMP) eine Technologie, die das Publish-Subscribe Pattern unterstützt. Dies ermöglicht die Nutzung verschiedener Transport Layer, wie beispielsweise TCP, ZeroMQ oder aber standartmäßig dem Websocket Protokoll. Websockets ermöglichen eine Zwei-Wege Kommunikation zwischen einem Server und einem Client. Sodass der Server aktiv

Daten an den Client senden kann [47]. Websockets sind durch den RFC6455 standardisiert und können im Gegensatz zu reinen TCP Verbindungen in allen gängigen Webbrowsern über eine Javascript API genutzt werden [48] [49]. Das WAMP-Protokoll ist ein offizielles Subprotokoll des Websockets Standards [50].

Innerhalb des WAMP Protokolls werden folgende Konzepte definiert [51]:

- Router – Zentrale Einheit zur Verarbeitung von Anfragen
- Komponente – Registriert sich beim Router z.B. Client- oder Serverkomponente

Der Router übernimmt dabei die zentrale Aufgabe des Annehmens und Verteilens von Anfragen. Dies gilt für das Publish-Subscribe Pattern sowie für das RPC Pattern. Dabei wird diese zentrale Einheit auch als Broker bezeichnet [52].

Eine Komponente steht dabei für ein Programm, dass eine Verbindung zum Router aufbaut und anschließend Anfragen via Remote Procedure Calls und Publish-Subscribe Pattern stellen kann.

3.6.1 Publish-Subscribe

Das Publish-Subscribe Pattern unterstützt die Möglichkeit eine Nachricht über ein definiertes Topic an mehrere Abnehmer zu verteilen, siehe hierzu Abbildung 8. Die Unabhängigkeit zwischen einem Publisher (Produziert Messages) und einem Subscriber (Empfängt Messages) wird dadurch erhöht [53].

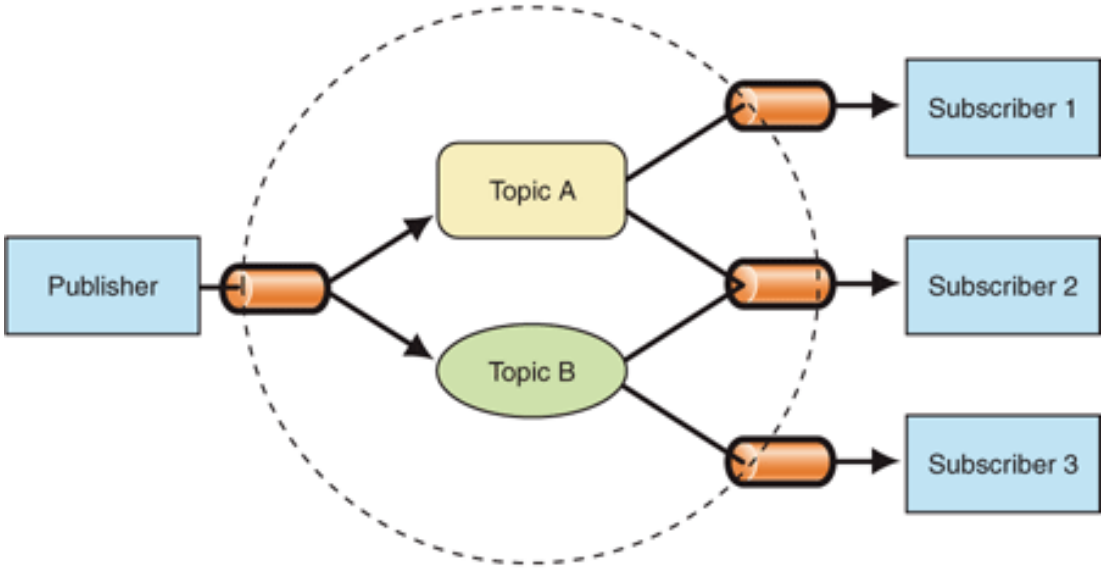

Abbildung 8 - Public-Subscribe Pattern [53]

Dabei wird im WAMP Protokoll dieser Mechanismus über eine zentrale Stelle abgewickelt, dem Router. Der Router übernimmt dabei die Verwaltung der Subscriber und auch die Zustellung von eingehenden Events eines Publishers an die Subscriber. Dabei werden die Topics im WAMP Protokoll über so genannte Uniform Ressource Identifier (URI) beschrieben, beispielsweise [51]:

- com.myapp.mytopic1
- com.myapp.myprocedure1
- com.myapp.myerror1

Jene URIs dienen zur Identifizierung von Topics oder auch Remote Procedure Call Methoden.

3.6.2 Remote Procedure Call

Das WAMP Protokoll unterstützt auch Remote Procedere Calls (RPC). Diese nutzen das Request Response Pattern, bei dem eine Anfrage an ein System gesendet wird und nach Bearbeitung zu einer Rückmeldung führt [54].

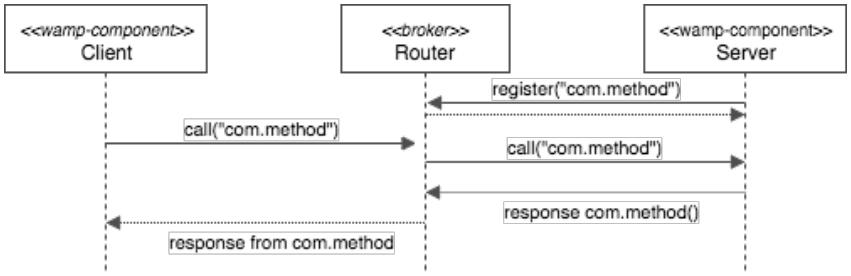

Abbildung 9 - RPC - WAMP

Abbildung 9 erläutert den Ablauf einer RPC Anfrage im WAMP Protokoll. Dabei ist zu erkennen, dass zunächst die Methode beim Router registriert werden muss, um anschließend von weiteren Komponenten verwendet werden zu können [51].

Dies unterstützt die Eigenschaft des loose-coupling zwischen der aufrufenden (in Abbildung 9 Client) und der anbietenden (in Abbildung 9 Server) Komponente, auch bei RPC Aufrufen [55].

4. Kapitel 4 – Konzeption

Die Softwareentwicklung hat in den vergangenen Jahren gezeigt, dass eine ausführliche Konzeption im späteren Projektverlauf Zeit und Kosten spart [56]. In diesem Kapitel wird eine Architektur beschrieben, welche die in Kapitel 2 genannten Anforderungen mittels Microservice Pattern erfüllt.

4.1 Architektur

Die steigende Komplexität von Softwaresystemen führte zu Methoden, die die Struktur einer Software grundlegend beschreiben. Die Softwarearchitektur kann dabei als eine Art “Bauplan“ für ein Softwaresystem verstanden werden [57].

Der Begriff der Systemarchitektur grenzt sich zum verwandten Begriff der Softwarearchitektur ab, indem bei der Systemarchitektur die Betrachtung von Software- und ggf. Hardwarekomponenten in den Vordergrund rückt. Bei der Systemarchitektur wird das Zusammenspiel ganzer Komponenten beschrieben. Ein Beispiel für eine klassische Systemarchitektur ist das in Abbildung 10 gezeigte 3-Schichten Modell [57]. Softwarearchitektur hingegen würde die in der Logik-Schicht enthaltene Software beschrieben.

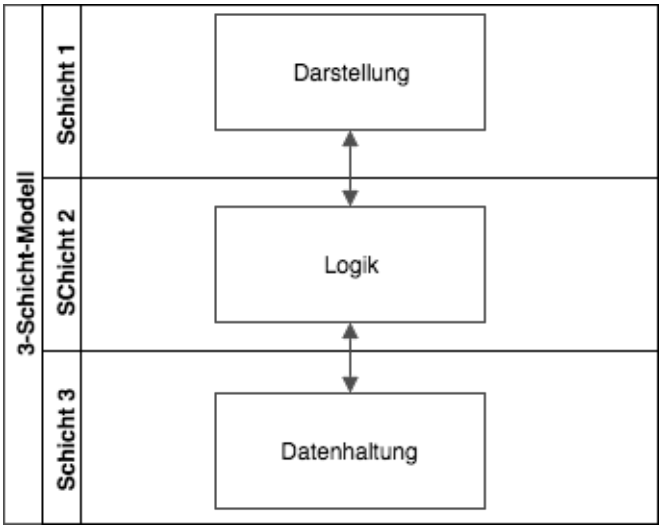

Abbildung 10 - 3-Schicht-Modell

In dieser Arbeit wird, wie wie bereits in Abschnitt 1.3 beschrieben, die Systemarchitektur mittels des Microservice Pattern näher betrachtet. Bei einer

Micoservice Architektur können die einzelnen Use-Cases auf entsprechende Services aufgeteilt werden. Diese Aufteilung ist bei einer neuen Anwendung sinnvoll, da noch keine zusammenhängenden Klassendiagramme existieren. Diese wären auch nicht förderlich für ein verteiltes System aus Services, die unabhängig voneinander entwickelt werden sollen. Bei der Transformation einer bestehenden Anwendung in eine Mircoservice Architektur kämen ggf. andere Ansätze zum tragen [58].

Alternativ würde ein monolithischer Ansatz in Frage kommen, der aufgrund der in 2.3 genannten Qualitätskriterien jedoch Nachteile bezüglich der Austauschbarkeit und Wartbarkeit mit sich brächte. Jenes bedingt sich durch die in Abschnitt 4.2 beschriebe Verwendung unterschiedlicher Programmiersprachen für verschiedene Use-Cases.

Im Use-Case Diagramm, beschrieben in Abbildung 5, sind die aus den Anforderungen extrahierten Use-Cases abgebildet. Dies behandelt jedoch noch nicht die entsprechenden technischen Prozesse für eine konversationsbasierte Interaktion. In Abbildung 11 wurde die Darstellung um die technischen Use-Cases erweitert, die bei der konversationsbasierten Interaktion mit einem Nutzer notwendig sind.

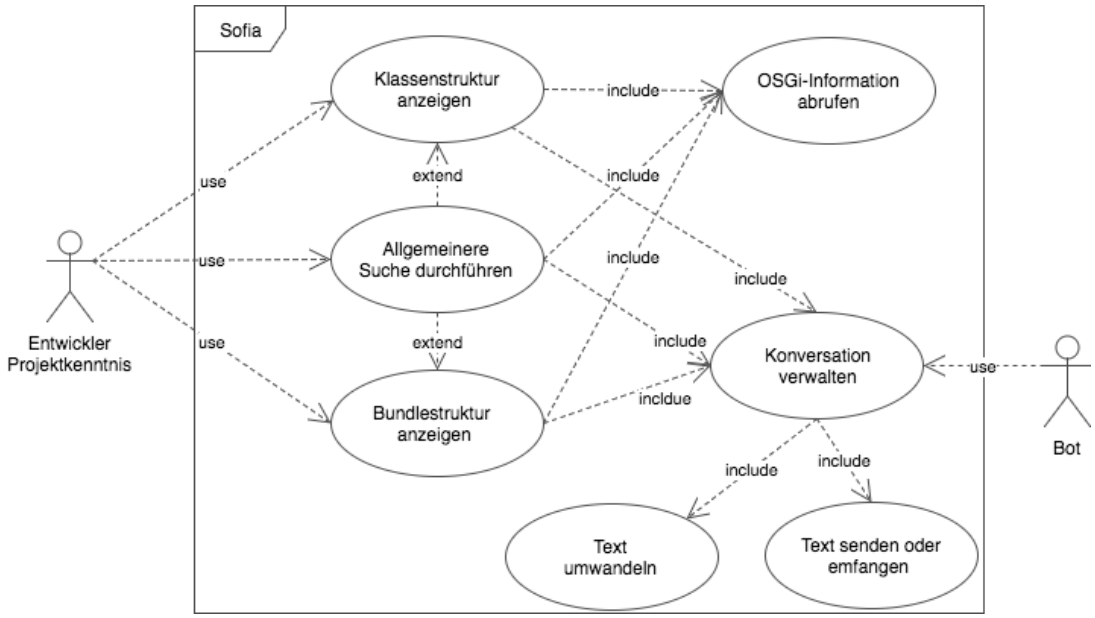

Abbildung 11 - Use-Cases – Technisch

Die in Abbildung 11 hinzugefügten Use-Cases stehen für spezielle Aufgaben:

- OSGI-Informationen abrufen: Nutzung der OSGIRestAPI
- Konversation verwalten: Nutzerfragen verstehen
- Text umwandeln: Transformation von Text in Events
- Text senden oder empfangen: Sendet / Empfängt eine Nachricht an / des Users

Die einzelnen Use-Cases werden im folgenden Abschnitt 4.2 den einzelnen Services zugeordnet.

4.2 Modularisierung

Eine der ersten Definitionen der Softwaremodularisierung beschreibt das Konzept wie folgt [59]:

- Strukturierung eines Programms in unabhängige Einheiten
- Die Definition von globalen Variablen
- Das Exportieren von Variablen und Prozeduren zur Nutzung in anderen Modulen.

Die Modularisierung des Systems wird nachfolgend anhand der Aufteilung von Use-Cases in Services durchgeführt. Ein Service definiert sich im Microservice Pattern über die folgenden Eigenschaften: Unabhängige Entwicklungs- Skalierungs- und Deploymentprozesse, sowie die Bereitstellung definierter Schnittstellen zur Kommunikation mit anderen Services [23].

Abbildung 12 zeigt eine Übersicht über die Aufteilung in die einzelnen Services und wird in den nachfolgenden Absätzen genauer erläutert.

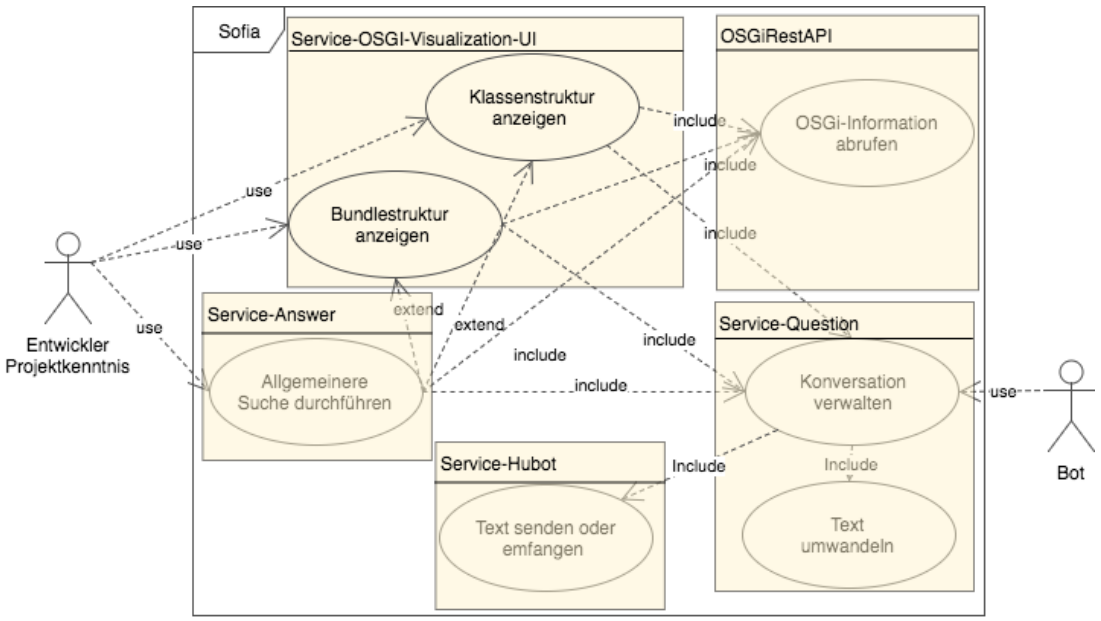

Abbildung 12 - Use-Cases - Service Zuordnung

4.2.1 Service-Question

Der Question Service übernimmt die Aufgaben des Use-Cases Konversation verwalten und Text umwandeln. Die konkrete Aufgabe umfasst dabei die Texteingaben zu verarbeiten und in komplexere Nachrichten zu übersetzten.

Die Tabelle 3 zeigt die Umwandlung von textuellen Eingaben des Nutzers in Nachrichten mit strukturierten Informationen.

Texteingabe	Nachrichten-Typ	Daten
Zeige mir etwas über Namespace gui an	Namespace	{„namespace“: „gui“}
Was für Abhängigkeiten hat die Klasse Login	Class	{„className“: „Login“}
Zeige mir bitte mehr Informationen zu diesen Bundles an	MoreInfo	{„target“: „bundles“}

Tabelle 3 - Zuordnung Fragen – Nachrichten

4.2.2 Service-OSGi-Visualization-UI

Der Service OSGi-Visualization-UI bildet den Funktionsumfang der Use-Cases Bundle anzeigen und Klasse anzeigen ab.

Dieser Service bietet dem Nutzer die Ansicht der Softwarevisualisierungen und setzt dabei die folgenden Nachrichtentypen in direkte Visualisierungen um:

- Type: Namespace → Bundles anzeigen, filtern nach dem Namespace
- Type: Class → Klassenstruktur der entsprechenden Klasse anzeigen

In dem in Abbildung 13 gezeigten UML 2.0 Komponenten Diagramm wird der Service schematisch dargestellt. Dabei ist hervorzuheben, dass die eigentliche Visualisierung, wie in 5.2.3 beschrieben, durch ein externes Modul bereitgestellt wird. Der Service übernimmt die Steuerung der Visualisierung, basierend auf den eingehenden Events.

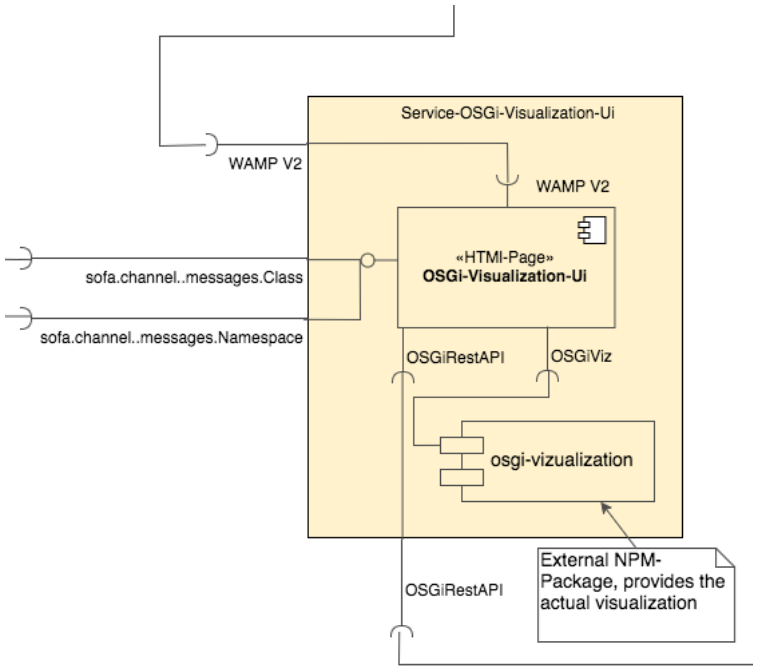

Abbildung 13 - Service-OSGi-Visualization

Dieser Service ist aus technischer Betrachtungsweise ein HTML-Dokument mit entsprechender Logik in Javascript. Dies bedingt sich aus der bereits be-

stehenden, externen OSGi-Visualization, die in Javascript und HTML implementiert wurde. Diese Einschränkung hat einen Einfluss auf die Anforderungen an die in 4.3 beschriebene Kommunikation zwischen den Services.

4.2.3 Service-Anwser

Der Anwser Service biete die Möglichkeit abstrakte Fragen zu stellen und übernimmt somit die Funktion des Use-Cases Allgemeine Suche durchführen.

Dabei soll dieser Service mit Fragen wie Beispielsweise „Zeige mir etwas zum Thema Login an“ umgehen können und entscheiden welche Visualisierung, über Klassen oder über Namespaces, angezeigt werden soll. Hierbei soll es ebenfalls die Möglichkeit geben vom Service aus Rückfragen an den Nutzer zu stellen. Beispielsweise um bei mehrfachem auffinden einer Klasse zu Fragen welche Klasse genau angezeigt werden soll.

4.2.4 Service-Hubot

Der Hubot Service dient der Kommunikation mit externen Chatdiensten wie beispielsweise IRC, Slack oder RocketChat. Das Hubot Framework bietet ein standardisiertes Interface zur Kommunikation mit Chatplattformen. Es ermöglicht über verschiedene Connectoren Chatplattformen anzusprechen [60].

Dabei ist die einzige Aufgabe dieses Service die Umwandlung von Textnachrichten in entsprechende Events und von Events in Textnachrichten an den Nutzer.

4.2.5 OSGiRestAPI

Die OSGiRestAPI ist eine externe API, die genutzt wird um Informationen über Bundles und Klassen eines OSGI-Projekts abzurufen. Dabei wird die Entwicklung dieser Software nicht in dieser Arbeit behandelt [61].

4.3 Kommunikation

Für die erfolgreiche Implementierung des Microservice Pattern müssen zwischen den einzelnen Services entsprechende Kommunikationsprotokolle definiert werden. Dabei gibt es eine Vielzahl von konzeptionellen und technischen Möglichkeiten. Diese Kommunikation wird auch Inter-Service-Communication (ISC) genannte [62].

Zu den gängigsten Paradigmen zählen die Verwendung des REST Pattern über HTTP, sogenannte REST-APIs. In diesem Fall kommunizieren Services häufig direkt ohne einen zentralen Broker miteinander [63].

Abbildung 14 zeigt die Anwendung des REST-Patterns auf die in 4.2 beschriebenen Services.

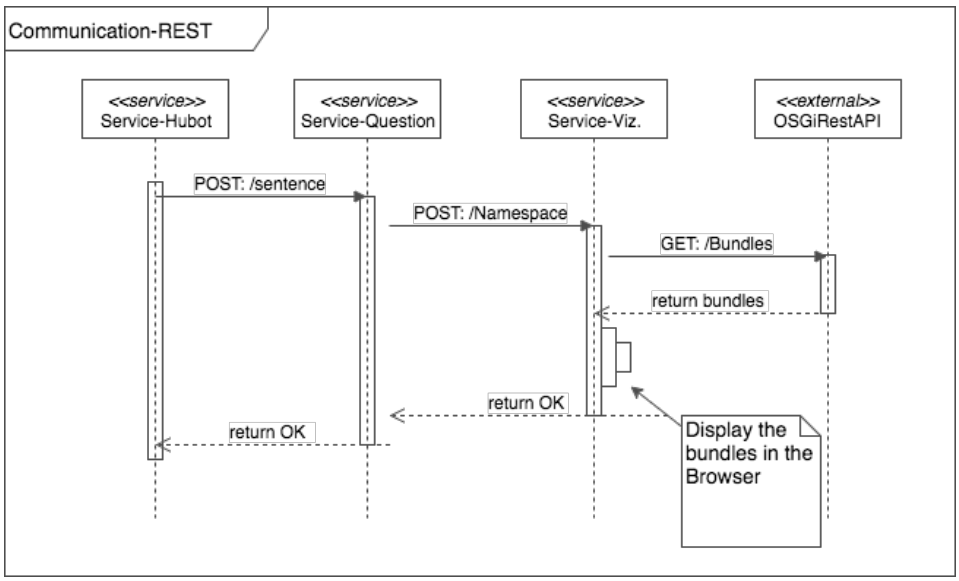

Abbildung 14 - Kommunikation–REST

Die in Abbildung 14 gezeigte REST Kommunikation hat auf der einen Seite den Vorteil, dass der Service-Hubot eine Statusrückmeldung erhält, um beispielsweise zu überprüfen ob das Anzeigen der Visualisierung erfolgreich war. Auf der anderen Seite kann es bei mehreren schnellen Eingaben von Benutzern zu ausstehenden Anfragen der Visualisierung kommen. Die Verwaltung dieser mehrfachen Anfragen muss jeder Service implementieren, jenes kann zu einer erhöhten Komplexität in jedem Service beitragen.

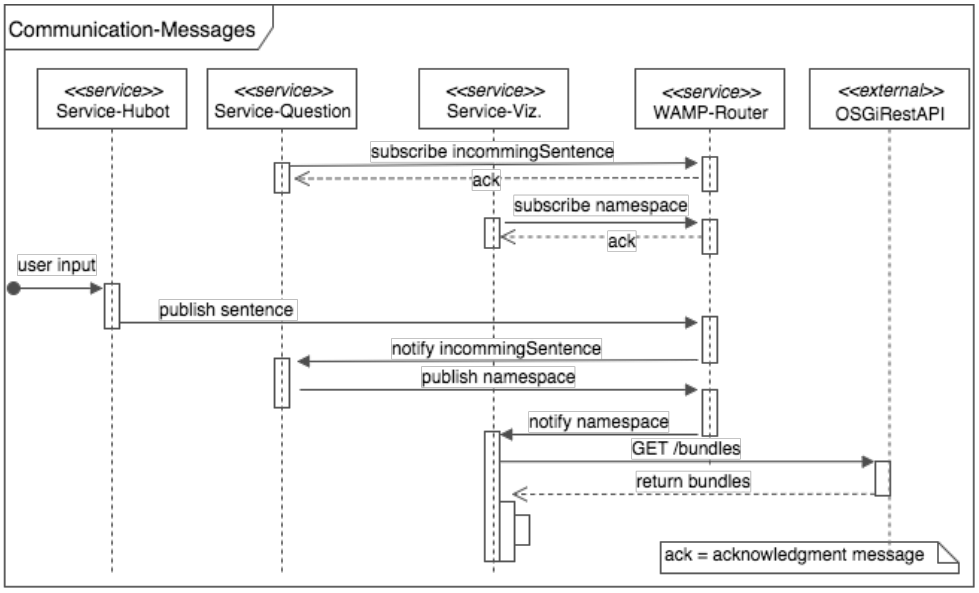

Abbildung 15 - Kommunikation–Messages

Bei der in Abbildung 15 gezeigten, auf einzelnen Nachrichten basierenden, Kommunikation ist zu erkennen, dass der Service-Hubot mit dem Senden der Nachricht (auch Message genannt) die Anfrage abschließt. Im Gegensatz zum Verhalten in Abbildung 14 agiert dieser unabhängiger von anderen Services, da er nicht direkt mit dem Service-Question kommuniziert. Diese Eigenschaft ist im Hinblick auf die in 2.3 genannten Qualitätskriterien wünschenswert. Aus diesem Grund wurde die in Abbildung 14 gezeigte Kommunikationsarchitektur gewählt.

Bei einer Event-Driven-Arichtecture ist die Spezifikation der Events, sowie der Event Daten ein Part der Konzeptionsphase [64]. Die Definition dieser ist der Abbildung 16 zu entnehmen.

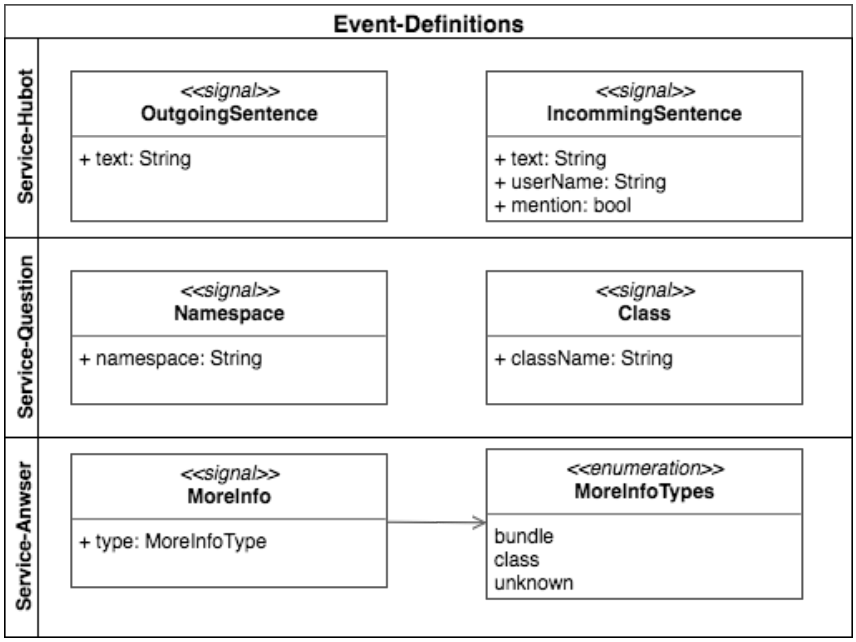

Abbildung 16 - Event-Definition

Die Definition der in Abbildung 16 gezeigten Events geht zum einen auf die in Tabelle 3 gelisteten Nachrichten-Typen zurück. Sowie zum anderen auf zwei Event-Typen zur Kommunikation mit dem User, IncomingSentence und OutgoingSentence.

Das in Abbildung 17 dargestellte UML Component Diagramm zeigt die einzelnen, in 4.2 genannten Services, mit ihrer internen Struktur sowie den geforderten bzw. angebotenen Schnittstellen.

Dabei werden die in Abbildung 16 gezeigten Event Definitionen unter einer bestimmten URI dargestellt und bieten somit eine Schnittstelle an. Eine URI dient zur Differenzierung zwischen verschieden Themen (Topics), siehe Abbildung 8.

Die URIs werden nach dem Schema: *sofia.channel.<ChannelId>.messages.<EventName>* bzw. für RPC Aufrufe *sofia.channel.<ChannelId>.rpc.<MethodName>* definiert. Die *<ChannelId>* dient zur eindeutigen Zuordnung einer Nachricht zu einem bestimmten Chatroom. In Abbildung 17 wird die *<ChannelId>* durch *..* als Platzhalter ersetzt.

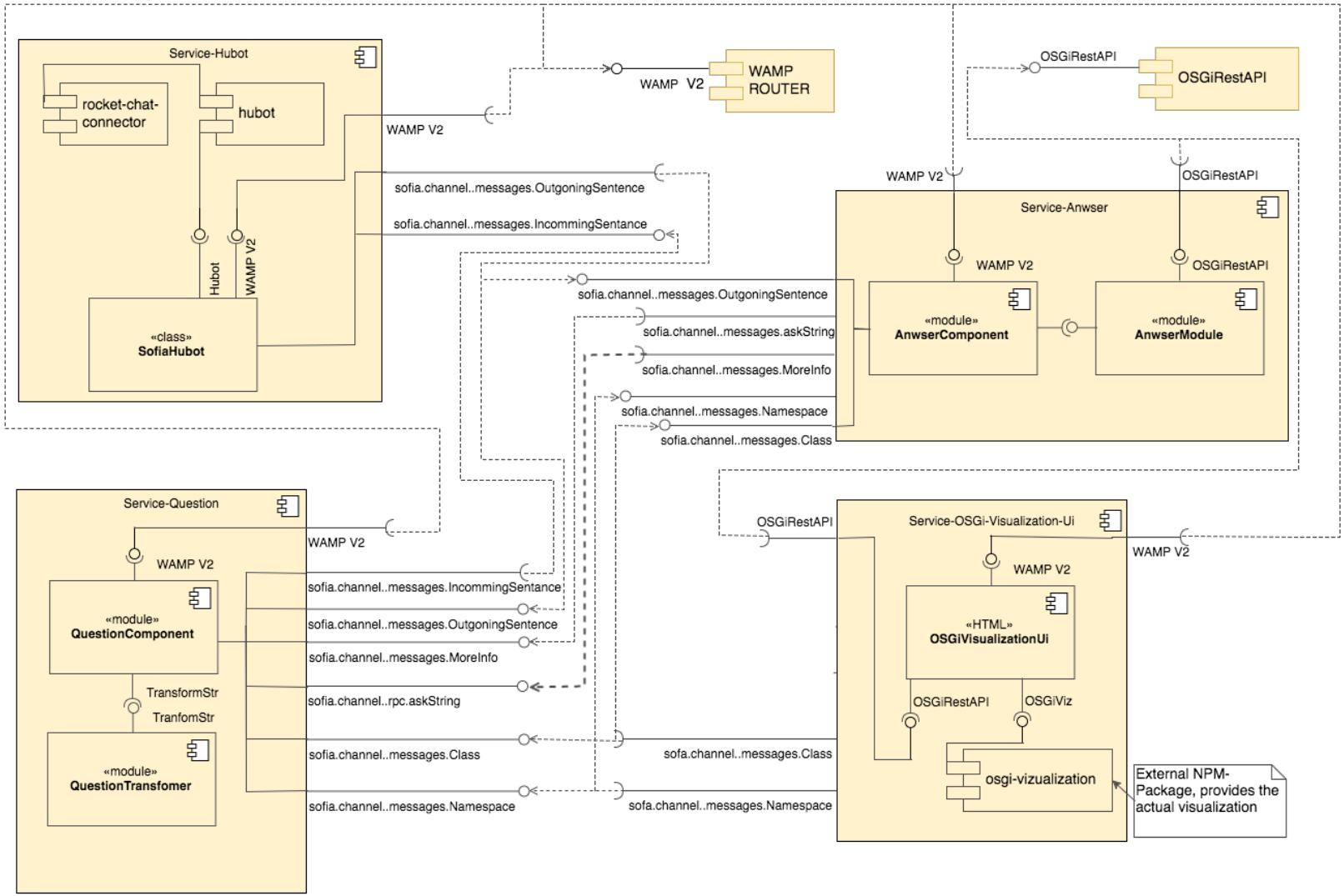

Abbildung 17 - System Übersicht

4.4 User-Interface

Das Userinterface setzt sich aus den in Abbildung 18 gezeigten Bereichen zusammen.

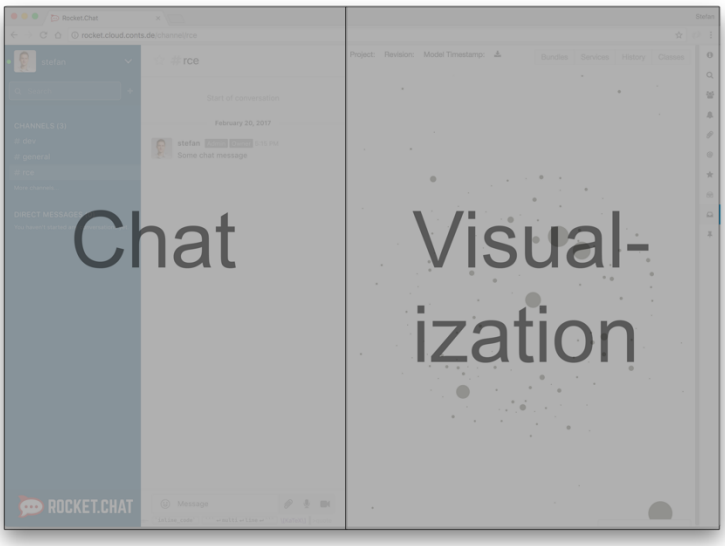

Abbildung 18 - Userinterface

Auf der linken Seite befindet sich dabei der Chatbereich. Welcher durch die quelloffene und communitygetriebene Slack Alternative RocketChat bereitgestellt wird. Die Software bietet die Möglichkeit per Gruppenchat miteinander zu kommunizieren und stellt für verschiedene Plattformen entsprechende Client Anwendungen bereit. Alle Anwendungen sind unter MIT Lizenz lizensiert [65].

Die rechte Seite wird von der Softwarevisualisierung bestimmt. Dabei wird hier der in 4.2.2 beschriebene OSGi-Visualization-UI Service angezeigt.

4.5 Testverfahren

Die Entwicklung hin zu einem verteilten System mittels Microservice Architektur bietet viele Vorteile, jedoch auch neue Herausforderungen. Die Aufteilung einer Software in unabhängige kleine Einheiten steigert die Komplexität von den Testverfahren und des Entwicklungsprozesses. Gerade aufgrund der unterschiedlichen technologischen Plattformen sowie der separaten Entwicklung der einzelnen Services ergeben sich komplexe Test- und Debug- Bedingungen [66].

Für diesen Prototypen werden beispielhaft Unit-Tests in den einzelnen Services implementiert. Die Einführung von serviceübergreifenden Integrationstests bietet weitreichende Möglichkeiten zur Steigerung der Softwarequalität, überschreitet jedoch den Umfangs dieser Bachelorarbeit.

5. Kapitel 5 – Implementierung

Die Implementierung ergibt sich aus der in Kapitel 2 beschriebenen Architektur und beschreibt die technische Umsetzung der einzelnen Services, sowie die Kommunikation untereinander.

5.1 Architektur

Für die in 4.1 beschriebene Architektur, insbesondere für die unter 4.3 aufgezeigte Kommunikation zwischen den einzelnen Services, ist zunächst die Entscheidung bezüglich des Kommunikationsprotokolls zu treffen, da die Implementierung der einzelnen Services hiervon abhängt.

Für die Kommunikation zwischen den einzelnen Services bieten sich verschiedene Protokolle an. Dabei ergaben sich aus dem in Kapitel 4 beschriebenen Konzept folgende technische Anforderungen:

- Remote Procedure Calls – Für Rückfragen eines Service beim Nutzer
- Publish-Subscribe - Zur Unterstützung des loose couplings zwischen Services

Für die Implementierung des Prototypen kommt das in 3.6 beschrieben WAMP V2 Protokoll zum Einsatz.

Eine weit verbreitete Alternative, die ebenfalls die beiden o.g. Features unterstützt, wäre das Advanced Messaging Queuing Protocol (AMQP) gewesen [67]. Der Broker RabbitMQ, der das AMQP Protokoll umsetzt, bietet umfangreiche Features und sehr gute Möglichkeiten der Skalierung [68], jedoch fehlt hierbei die standardmäßige Unterstützung von Websockets als Transportprotokoll [69]. Aufgrund der Tatsache, dass der in 4.2.2 beschriebe Service als HTML-Dokument ausgeliefert wird benötigt dieser Service im Idealfall eine direkte Anbindung via Websockets, um auf entsprechende Events zu reagieren. Des Weiteren ist die Durchführung von RPC Anfragen mithilfe von AMQP im Vergleich zu WAMP V2 von der Codestruktur her komplexer. Wie den Beispielen in Abbildung 24 und Abbildung 25 zu entnehmen ist.

5.2 Microservices

Nachfolgend wird für die einzelnen Services eine Beschreibung der technischen Umsetzung gegeben.

5.2.1 Service-Hubot

Der Service-Hubot stellt die Funktionalität, zur Verbindung zwischen dem WAMP V2 Protokoll und dem der Chat Software, bereit. Dabei werden Hubots wahlweise in Javascript oder einer Meta-Sprache für Javascript (CoffeeScript) geschrieben. In diesem Fall wurde für die Implementierung Javascript verwendet, da die Sprache CoffeeScript eine Nischensprache darstellt und daher für diese Arbeit nicht zielführend wäre [70].

Das neue Hubot Script veröffentlicht jede Nachricht unter der WAMP V2 URI *sofia.channel.<ChannelId>.messages.IncommingSentence*.

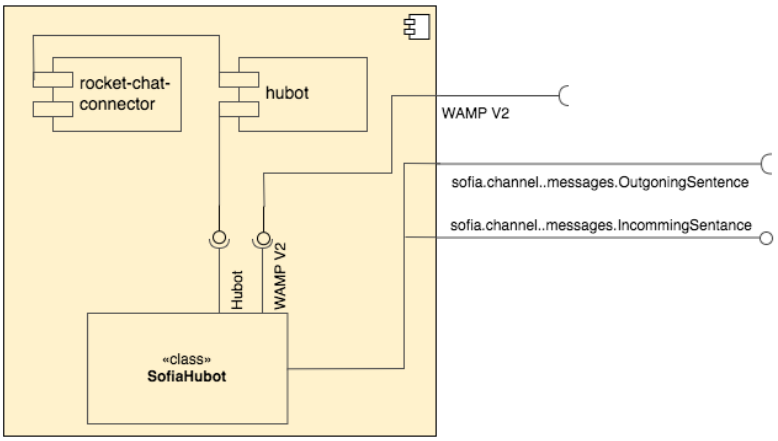

Abbildung 19 - Service-Hubot - Struktur

Dabei werden die externen Pakete hubot und hubot-rocketchat mithilfe des Paketverwaltungssystem node package mananger (npm) eingebunden. Die Hubot Bibliothek stellt dabei die Standardfunktionen zu Interaktion mit einem Chatraum zur Verfügung. Hubot-rocketchat stellt hingegen die Verbindung zur RocketChat Instanz her und sorgt für die entsprechende API Anbindung.

5.2.2 Service-Question

Die Aufgabe des Question Service besteht in der Umwandlung von Texteingaben in komplexere Events. Um die Eigenschaft der Erweiterbarkeit zu unterstützten wird für diesen Service die Programmiersprache Python gewählt.

Zurzeit basieren viele der aktuellen NLP Libraries, sowie auch Bibliotheken aus dem Bereich der künstlichen Intelligenz auf Python. Im Hinblick auf eine eventuelle zukünftige Verwendung dieser Technologien wurde Python als Implementierungssprache gewählt.

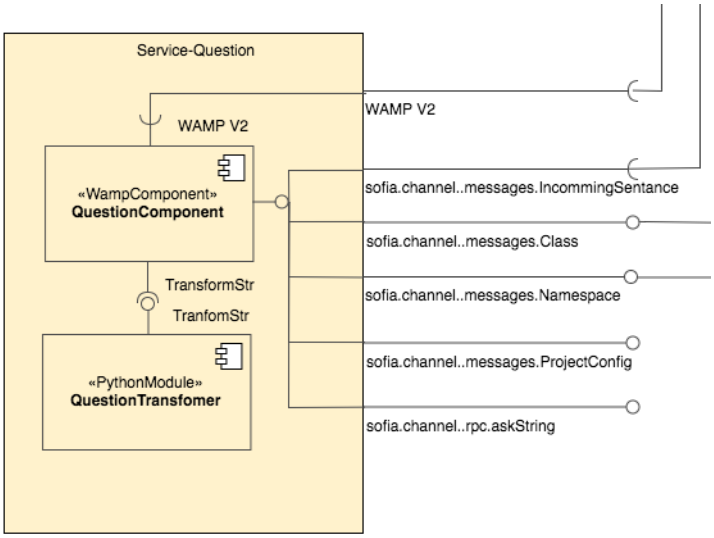

Abbildung 20 - Service-Question - Struktur

Durch Separierung der Publish-Subscribe Methoden (QuestionComponent) und der Transformation von Text in entsprechende Events wird die Testbarkeit des Moduls durch einfache Unittests erhöht, sowie die Austauschbarkeit dieser Funktionen gesteigert. Zur Kommunikation mit dem WAMP Router kommt dabei die Bibliothek Autobahn.ws zum Einsatz. Die das WAMP V2 Protokoll implementiert.

5.2.3 Service-OSGi-Visualization-UI

Hierbei handelt es sich um die Darstellungsebene die die Events *Namespace* und *Class* in entsprechende Visualisierungen umsetzt. Innerhalb der Umsetzung kam Javascript zum Einsatz.

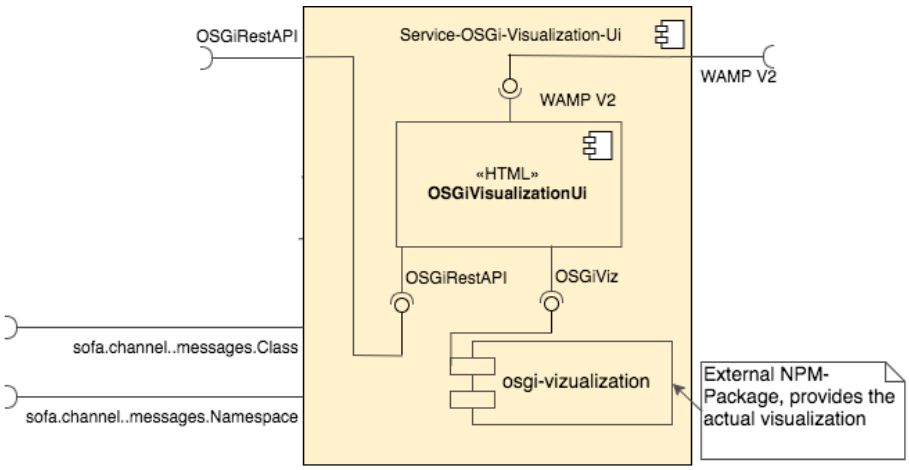

Abbildung 21 - Service-OSGi-Visualization-UI – Struktur

Dabei bindet dieser Service das Packet osgi-visualization mithilfe von npm ein. Die einzelnen Visualisierungen selbst befinden sich in der Drittkomponente osgi-visualization. Der Service OSGi-Visualization-UI steuert anhand der eingehenden Events *Namespace* und *Class* die entsprechende Visualisierung.

5.2.4 Service-Answer

Dieser Service wurde ebenfalls mittels Python umgesetzt, da die Konfiguration und Testumgebung bereits im Question-Service erprobt wurden. Dabei kommuniziert dieser Service mit dem Question-Service, sowie dem Hubot und der Visualisierung. Dies liegt an der Anforderung, dass der Service Rückfragen an den Nutzer stellen soll, um z.B. zu entscheiden welche Visualisierung angezeigt wird.

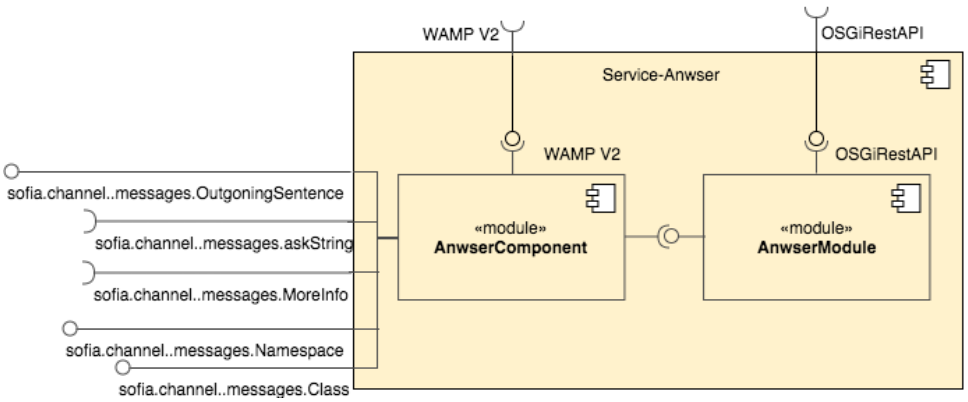

Abbildung 22 - Service-OSGi-Visualization-UI – Struktur

Hierbei hervorzuheben ist die Verwendung der RPC Methode askString. Dabei handelt es sich um eine Methode, die durch den Service-Question bereitgestellt wird und erst Rückmeldung gibt, wenn der Benutzer eine Eingabe getätigt hat. Eine Verwendung von einem Event, wie z.B. bei Namespace ist in diesem Fall nicht Sinnvoll, da bei einer Rückfrage an den Nutzer das Programm so lange warten soll bis die Eingabe des Nutzers abgeschlossen wurde. Siehe hierzu Abbildung 14, die Kommunikation via REST, ein ebenfalls wie RPC auf dem Request-Response Pattern basierendes Protokoll, und Abbildung 15 der Umsetzung der Kommunikation via Publish-Subscribe Pattern.

5.2.5 WAMP-Router

Zur Kommunikation untereinander dient der zentrale WAMP V2 Router. Das WAMP V2 Protokoll wird von mehreren Brokern unterstützt. Der Crossbar.io Router ist dabei der am weitesten verbreitete [71]. Er basiert auf Python 2.7 und bietet vielseitige Konfigurationsmöglichkeiten. Beispielsweise können sehr fein granulierte Autorisierungsmethoden genutzt werden [72].

In diesem Projekt kommt die Standardkonfiguration des Routers zum Einsatz. Sie ermöglicht eine anonyme Autorisierung auf dem sogenannten realm1. Ein Realm bildet in WAMP einen in sich geschlossen Bereich. Die Realms auf dem Router agieren vollkommen autark [51].

Der Crossbar.io Router wird direkt über das offizielle Docker Image mithilfe des folgenden Befehls gestartet:

```
docker run -it -p 8080:8080 crossbario/crossbar
```

Abbildung 23 - Crossbar Docker Start

5.3 Userinterface

Das Userinterface wird mithilfe einer modifizierten Version von RocketChat umgesetzt. Die quelloffene Chatplattform wird mithilfe des Meteor Frameworks entwickelt, welches u.a. hochabstrahierte Funktionen für Modularisierung und Datenverwaltung bereitstellt [73].

Die Erweiterung der bestehenden Software um einen weiteren Tab, in der sogenannten Tabbar, wurde mithilfe des Modulsystems von Meteor umgesetzt.

Dabei wird die bestehende Code-Basis nicht angepasst, sondern ein weiteres Meteor-Modul hinzugefügt.

Der RocketChat Source-Code wird während des Erstellens, innerhalb des Docker Containers, heruntergeladen und um das eigene Modul ergänzt. Anschließend wird ein neuer Build des Source-Code angestoßen und das Ergebnis im Container gespeichert. Es steht somit ein Docker-Container zur Verfügung, der alle Funktionen von RocketChat unterstützt, für eine Aktualisierung ausgelegt ist und um eine Funktion erweitert wurde.

5.4 Deployment und Testing

Aufgrund der verschiedenen Technologien und Programmiersprachen (Java, Javascript, Python, Drittanbieter Software) wird jeder Service als einzelner Docker-Container bereitgestellt. Dies ermöglicht es alle Services auf einem Host auszuführen ohne, dass dies zu Kompatibilitätsproblemen führt. Beispielsweise ist der Crossbar WAMP V2 Router nur mit Python 2.6 kompatibel. Der Question Service ist jedoch auf Python 3.6 ausgelegt. Ohne eine entsprechende reproduzierbare und virtualisierte Laufzeitumgebung könnte es zu Kompatibilitätsproblemen beider Services kommen.

Der Build-Prozess ist in jedem einzelnen Service in der Datei Dockerfile hinterlegt und wird bei jeder Änderung auf dem Remote Repository automatisch durch eine Continuous Integration Plattform ausgeführt.

Zudem führt sie die Unittests jedes Service aus und veröffentlicht nur bei erfolgreich abgeschlossenen Tests den Docker-Container.

6. Kapitel 6 – Abschluss

Das nachfolgende Kapitel bietet einen Überblick über die gewonnenen Erkenntnisse und gibt einen Ausblick zu möglichen weiterführenden Entwicklungen.

6.1 Ergebnis

Während der Konzeptions- sowie der Implementierungsphase ergaben sich mehrere Ergebnisse und Problematiken, die nachfolgend genauer ausgeführt werden.

Zunächst werden die in Abschnitt 2.3 angestrebten Qualitätskriterien betrachtet. Dabei erfolgte die Beurteilung der Komplexität und somit der Wartbarkeit und Austauschbarkeit, der einzelnen Module, anhand der Lines of Code.

Service	Technologie	LOC
Service-Question	Python	233
Service-Hubot	Javascript (node.js)	143
Service-OSGi-Visualization-UI	Javascript (Browser)	143
Service-Anwser	Python	

Tabelle 4 - LOC per Service

Dabei setzt sich die LOC Kennzahl aus den Zeilen, die zum Ausführen des Services benötigt werden sowie den Kommentarzeilen zusammen. Die entsprechenden Unit-Tests sind nicht mit eingeschlossen.

Um die Komplexität der entsprechenden Services möglichst gering zu halten wurde versucht die LOC niedrig zu halten. Dabei zeigte sich, dass die beiden in Python geschriebenen Services mehr LOC aufweisen. Die Erklärung hierfür liegt, wie in Kapitel 5 beschrieben, in den komplexeren Use-Cases dieser beiden Services. Somit besteht in diesem Fall keine Korrelation zwischen der höheren LOC Zahl und der Wahl der Programmiersprache.

Für die in Abschnitt 1.2 aufgezeigte Problemstellung bietet die Microservice Architektur, in Bezug auf die Erweiterbarkeit, eindeutige Vorteile. Darunter versteht sich u.a. die einfache Möglichkeit des Hinzufügens weitere Services, die

auf die definierten Events reagieren. Durch die Verwendung des technologisch unabhängigen Protokolls WAMP V2 können neue Services in anderen Programmiersprachen verfasst werden als die verwendeten.

Dies zeigt sich auch in der Anbindung der Chat-Software über ein Hubot-Script. Dadurch kann mit minimalem Aufwand das System auch beispielsweise über einen IRC Chat angesprochen werden. Trotz der beschrieben Vorteile sind auch typische Probleme einer Microservice Architektur aufgetreten [74] . Dabei fiel vor allem die aufwendige Entwicklungsumgebung ins Gewicht, die durch das Verbinden des zu entwickelnden Services mit den bereits bestehenden Services komplex einzurichten war. Eine mehrschichte testgetriebene Entwicklung und klare Schnittstellendefinition sind daher für einen Praxiseinsatz zu empfehlen.

6.2 Zusammenfassung

Die Microservice Architektur stellt eine weitreichende Umstellung des Softwareentwicklungsprozesses dar. Auch wenn die zugrundeliegenden Prinzipien wie Modularisierung und Separierung von Softwarekomponenten bereits seit Jahrzehnten bekannt sind, ergibt sich erst aus dem Zusammenspiel, sowie der konsequenten Anwendung dieser eine hochflexible Architekturlösung. Welche insbesondere für große Softwareteams interessant ist, da sie einen verteilten, weitgehend unabhängig voneinander stattfindenden Entwicklungsprozess ermöglicht.

Im Zusammenspiel mit den Herausforderungen eines konversationsbasierten Interfaces sowie der Einbindung unterschiedlichster Technologien wird die Komplexität einer Mircoservice Architektur in dieser Arbeit deutlich. Dabei zeigt sich, dass die Konzeption eines verteilten Systems die Vorteile der technischen Unabhängigkeit und klar definierten Schnittstellen untereinander aufzeigt. Jedoch offenbarten sich ebenfalls die Nachteile einer solch verteilten Lösung. Martin Fowler stellt dabei die erschwerte Konsistenz sowie die erhöhte Fehlerquelle durch Netzwerkabfragen in der Vordergrund [74]. Ebenfalls nicht zu vernachlässigen sind die in 5.4 verdeutlichten hohen Anforderungen an Entwicklungs- und Testumgebungen, um effektive Fehler- und Entwicklungsarbeit durchzuführen [75].

Die Erkundung von Softwarearchitekturen mittels Konversationen bietet eine spannende Möglichkeit der Interaktion. Die Anforderung war es hierbei eine möglichst hohe Flexibilität bezüglich der Austausch- und Erweiterbarkeit einzelner Services zu gewährleisten. Hierfür eignet sich das in Kapitel 4 beschriebene Konzept, welches sich in der niedrigen LOC von durchschnittlich 200 pro Service zeigt.

Das WAMP V2 Protokoll stellte dabei eine hohe Abstraktionsebene für die Kommunikation unter den Services bereit und trug damit maßgeblich zur niedrigen LOC pro Service bei. Dabei stellte das WAMP V2 Protokoll alle Funktionen für die Implementierung einer Event Driven Architecture, sowie auch für ein Request Response Pattern bereit. Wie in Abschnitt 5.1 beschrieben wäre dies durch Verwendung von beispielsweise AMQP nicht standardmäßig möglich gewesen.

Die Umsetzung der Event Driven Architecture ergibt ein System, dass sich durch die Eigenschaft des loose-coupling zwischen den einzelnen Services auszeichnet. Jedoch eignet sich dieser Ansatz nicht für alle Anwendungsfälle, wie der im Abschnitt 5.2.4 beschriebene Service darlegt. Dieses Beispiel zeigt, dass der der Microservice Ansatz hier den Vorteil der Anpassbarkeit an die Anforderungen bietet, da für jeden Service einzeln entschieden werden kann über welches Pattern (RPC vs. Publish-Subscribe) er Schnittstellen bereitstellt.

6.3 Ausblick

Während der Konzeption und Implementierung ergaben sich mehrere Fragestellungen bei denen eine tiefere Betrachtung sinnvoll erscheint. Jedoch aufgrund des Umfangs dieser Bachelorarbeit nicht weiter verfolgt werden konnten.

Im Zusammenhang mit dem Microservice Pattern ergeben sich Fragen bezüglich der lokalen Entwicklungs- und Testumgebung. Dabei zeigte sich dies bereits während der Implementierung und es ergaben sich konkrete Fragen, wie die Funktionalität zwischen den einzelnen Services zu testen wäre.

Ein weiterer potentieller Bereich ist die Weiterentwicklung von Microservices mit Event Driven Arichtecture. Dabei geht es um die Frage der Anpassung und

Abwärtskompatibilität von Schnittstellen bei Änderungen des Schemas. Aktuelle Forschungen und Vorträge hierzu zeigen eine starke Entwicklung hin zu Protokollen wie Protocoll Buffers und Avero, die Eigenschaften wie Abwärtskompatibilität standardmäßig unterstützen [76].

Ein interessanter Punkt, der sich während der Konzeption auftat, beschäftigte sich mit der Sinnhaftigkeit des Microservice Pattern für neue Softwareprodukte. Bisher wird in der Fachliteratur zumeist die Überführung von bestehender Software in eine Microservice Architektur betrachtet. Eine Entwicklung von neuen Systemen, unmittelbar mit dem Microservice Pattern, steht bislang kaum im Fokus.

Für das konzipierte System ergeben sich auch ganz konkrete Erweiterungen. Auf der einen Seite können weitere Funktionen mit komplexeren Fragen ergänzt werden, wie beispielsweise: „Zeige mir doch bitte die Zusammenhänge zwischen KlasseA und KlasseB“. Auf der anderen Seite verspricht die Anbindung von Spracheingabesystemen wie Amazon Alexa oder auch Google Home eine interessante Möglichkeit der Interaktion. Dabei könnte eine solche Interaktionen auch innerhalb von Virtual Reality Visualisierungen, wie in Kapitel 1 beschrieben, von Nutzen sein. Insgesamt bietet der Bereich der Interaktion mit Softwarevisualisierungen mithilfe eine Konversation (Sprache oder Text) ein weites und interessantes Forschungsfeld und bedarf einer vertieften Betrachtung.

Anhang

```
#!/usr/bin/env python
import pika
import uuid

class FibonacciRpcClient(object):
    def __init__(self):
        self.connection = pika.BlockingConnection(pika.Connection-
Parameters(
                host='localhost'))

        self.channel = self.connection.channel()

        result = self.channel.queue_declare(exclusive=True)
        self.callback_queue = result.method.queue

        self.channel.basic_consume(self.on_response, no_ack=True,
                                   queue=self.callback_queue)

    def on_response(self, ch, method, props, body):
        if self.corr_id == props.correlation_id:
            self.response = body

    def call(self, n):
        self.response = None
        self.corr_id = str(uuid.uuid4())
        self.channel.basic_publish(exchange='',
                                   routing_key='rpc_queue',
                                   properties=pika.BasicProperties(
                                         reply_to =
self.callback_queue,
                                         correlation_id =
self.corr_id,
                                         ),
                                   body=str(n))
        while self.response is None:
            self.connection.process_data_events()
        return int(self.response)

fibonacci_rpc = FibonacciRpcClient()

print(" [x] Requesting fib(30)")
response = fibonacci_rpc.call(30)
print(" [.] Got %r" % response)
```

Abbildung 24 - AMQP RPC Beispiel [77]

```
###############################################################################
# The MIT License (MIT)
#
# Copyright (c) Crossbar.io Technologies GmbH
#
# Permission is hereby granted, free of charge, to any person ob-
taining a copy
# of this software and associated documentation files (the "Soft-
ware"), to deal
```

```
# in the Software without restriction, including without limita-
tion the rights
# to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
# copies of the Software, and to permit persons to whom the Soft-
ware is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be
included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MER-
CHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN
# THE SOFTWARE.
#
###############################################################################
import asyncio
import datetime

from autobahn.asyncio.wamp import ApplicationSession, ApplicationRunner
class Component(ApplicationSession):
    """
    A simple time service application component.
    """
    async def onJoin(self, details):
        def utcnow():
            now = datetime.datetime.utcnow()
            return now.strftime("%Y-%m-%dT%H:%M:%SZ")
        await self.register(utcnow, u'com.timeservice.now')
if __name__ == '__main__':
    runner = ApplicationRunner(
        environ.get("AUTOBAHN_DEMO_ROUTER", u"ws://127.0.0.1:8080/ws"),
        u"crossbardemo",
    )
    runner.run(Component)
```

Abbildung 25 - WAMP V2 RPC Beispiel [78]

Literaturverzeichnis

[1] N. E. Fenton und S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS, 1997.

[2] A. Tomasini, „From a Product Vision to a running software,“ - - 2014. [Online]. Available: http://www.slideshare.net/tumma72/from-a-product-vision-to-a-running-software-and-back-again-and-agile-coach-story-from-andrea-tomasini. [Zugriff am 21 12 2016].

[3] Microsoft, A. Begel und B. Simon, „Struggles of new college graduates in their first software development job,“ Microsoft, - - 2008. [Online]. Available: https://www.microsoft.com/en-us/research/publication/struggles-of-new-college-graduates-in-their-first-software-development-job/. [Zugriff am 21 12 2016].

[4] J.-F. Schrape, „Open Source Softwareprojekte zwischen Passion und Kalkül,“ - 2 2015. [Online]. Available: http://www.uni-stuttgart.de/soz/oi/publikationen/soi_2015_2_Schrape_Open_Source_Softwareprojekte_zwischen_Passion_und_Kalkuel.pdf. [Zugriff am 21 12 2016].

[5] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer und D. M. German, „Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub,“ - - 2015. [Online]. Available: http://etc.leif.me/papers/Kalliamvakou2015.pdf. [Zugriff am 21 12 2016].

[6] A. M. V. u. A. E. A. von Mayrhauser, „Program understanding behaviour during enhancement of large-scale software,“ *Journal of Soft- ware Maintenance: Research and Practice,* Bd. 9, Nr. 7, p. 299–327, 1997.

[7] T. Marquardt, „Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte,“ 08 04 2016. [Online]. Available: http://elib.dlr.de/105575/. [Zugriff am 15 11 2016].

[8] M. Brüggemann, „Visualisierung von mit OSGi-Komponenten realisierten Softwarearchitekturen im 3-dimensionalem Raum mit Virtual Reality,“ 08 12 2016. [Online]. Available: http://elib.dlr.de/108828/.

[9] C. u. S. A. Teichmann, 16 12 2013. [Online]. Available: Analysis of Software-Engineering-Processes. [Zugriff am 15 11 2016].

[10] Venkat Amirisetty; Motorola; Erkki Rysa; Nokia Corporation; , - - -. [Online]. Available: http://download.oracle.com/otndocs/jcp/mom-1.0-fr-oth-JSpec/. [Zugriff am 21 12 2016].

[11] T. Štolfa, „The Future of Conversational UI Belongs to Hybrid Interfaces,“ 10 03 2016. [Online]. Available: https://medium.com/the-layer/the-future-of-conversational-ui-belongs-to-hybrid-interfaces-8a228de0bdb5#.83bj72so4. [Zugriff am 11 12 2016].

[12] V. Zue, „CONVERSATIONAL INTERFACES: ADVANCES AND CHALLENGES,“ (MIT) Massachusetts Institute of Technology, - - 2000. [Online]. Available: https://www.researchgate.net/profile/James_Glass/publication/2985743_Cnversational_interfaces_Advances_and_challenges/links/0c9605208e8c26d543000000.pdf. [Zugriff am 26 12 2016].

[13] ISO, - 03 2014. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764. [Zugriff am 26 12 2016].

[14] D. Z. f. L. u. R. e.V., „Das DLR im Überblick,“ 05 12 2016. [Online]. Available: http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10443/637_read-251/#/gallery/8570. [Zugriff am 11 12 2016].

[15] Deutsches Zentrum für Luft und Raumfahrt e.V., „DLR - Simulations- und Softwaretechnik Köln/Braunschweig/Berlin,“ 02 12 2016. [Online]. Available: http://dlr.de/sc. [Zugriff am 02 12 2016].

[16] Deutsches Zentrum für Luft und Raumfahrt e.V., „oftware für Raumfahrtsysteme und interaktive Visualisierung,“ DLR, Unbekannt Unbekannt Unbekannt. [Online]. Available: http://www.dlr.de/sc/desktopdefault.aspx/tabid-1200/1659_read-3101/. [Zugriff am 21 12 2016].

[17] Deutsches Zentrum für Luft und Raumfahrt e.V. , „RCE,“ Unbekannt Unbekannt Unbekannt. [Online]. Available: http://www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513/. [Zugriff am 21 12 2016].

[18] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), 08 12 2016. [Online]. Available: http://rcenvironment.de/. [Zugriff am 05 01 2017].

[19] H. A. (DLR), Interviewee, *Interview zum Umgang mit Softwarevisualisierungen.* [Interview]. 19 12 2016.

[20] I. J. 1. 7. 6, ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model, -: American National Standards Institute, 2007.

[M. Cohn, User Stories Applied: For Agile Software Development, Bd. 1,
2 Addison-Wesley, 2004.
1
]

[M. Fowler, „Software Development in the 21st Century," in *ThoughtWorks*
2 *XCONF 2014*.
2
]

[M. u. J. L. Fowler, „Microservices," 25 03 2014. [Online]. Available:
2 http://www.martinfowler.com/articles/microservices.html. [Zugriff am 15 11
3 2016].
]

[Ordanini, Andrea und P. Pasini, „Service co-production and value co-
2 creation: The case for a service-oriented architecture (SOA).," *European*
4 *Management Journal,* Bd. 26, Nr. 5, pp. 289-297, 2008.
]

[C. Richardson, „Pattern: Microservice Architecture," 2017. [Online].
2 Available: http://microservices.io/patterns/microservices.html. [Zugriff am
5 24 02 2017].
]

[M. Fowler, „Continuous Integration," 01 05 2006. [Online]. Available:
2 https://www.martinfowler.com/articles/continuousIntegration.html. [Zugriff
6 am 25 02 2017].
]

[J. Thönes, „Microservices," *IEEE Software,* Bd. 32, Nr. 1, pp. 116-116,
2 2015.
7
]

[28] M. Fowler, „Focusing on Events,“ 25 01 2006. [Online]. Available: https://martinfowler.com/eaaDev/EventNarrative.html. [Zugriff am 20 02 2017].

[29] H. a. Y. A. a. P. L. a. M. F. Taylor, Event-Driven Architecture: How SOA Enables the Real-Time Enterprise, Pearson Education, 2009.

[30] C. Boettiger, „An introduction to Docker for reproducible research, with examples from the R environment,“ *SIGOPS Oper. Syst. Rev.,* Bd. 49, Nr. 1, pp. 71--79, 01 2015.

[31] Docker Inc., „WHAT IS DOCKER?,“ 17 02 2017. [Online]. Available: https://www.docker.com/what-docker. [Zugriff am 17 02 2017].

[32] Docker Inc. , „Modern Application Architecture for the Enterprise,“ 21 01 2016. [Online]. Available: https://www.docker.com/sites/default/files/WP_Modern%20App%20Architecture%20for%20Enterprise%20-%20Jan%202016.pdf. [Zugriff am 17 02 2017].

[33] V. Zue, „CONVERSATIONAL INTERFACES: ADVANCES AND CHALLENGES,“ Massachusetts Institute of Technology (MIT), 1997.

[34] R. Kar und R. Haldar, „Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements,“ *CoRR,* Bd. 1611, Nr. 03799, 11 2016.

[35] C. Hewitt, in *Viewing control structures as patterns of passing messages. Artificial intelligence*, 1977, pp. 323-364.

[36] C. Matthews, An introduction to natural language processing through Prolog, Routledge, 2016.

[37] A. Zumbrunnen, „Technical and social challenges of conversational design,“ 13 06 2016. [Online]. Available: https://uxdesign.cc/my-website-is-now-conversational-here-is-what-i-learned-7e943cc6ace0#.k5jglevjq. [Zugriff am 25 02 2017].

[38] K. W. u. H. A. M. M.-A. D. Storey, „How Do Program Under- standing Tools A ect How Programmers Understand Programs,“ *Pro- ceedings of the Fourth Working Conference on Reverse Engineering ,* 1997.

[39] M. Petre., „Mental Imagery, Visualisation Tools and Team Work,“ *Second Program Visualization Workshop,* 2002.

[40] OSGi Alliance , „OSGi™ Alliance Announces the Immediate Availability of OSGi Service Platform Release 3,“ 2003. [Online]. Available: https://www.osgi.org/wp-content/uploads/OSGi_R3_release.pdf. [Zugriff am 12 01 2017].

[41] Wikipedia , „OSGi,“ [Online]. Available: https://en.wikipedia.org/wiki/OSGi#Projects_using_OSGi.

[42] Oracle, „Developing OSGi Application Bundles for GlassFish Server,“ 2011. [Online]. Available:

[2] https://docs.oracle.com/cd/E18930_01/html/821-2418/gkqff.html. [Zugriff am 25 02 2017].

[43] L. Vogel, S. Scholz und v. GmbH, „OSGi Modularity - Tutorial,“ 02 02 2008,2017. [Online]. Available: http://www.vogella.com/tutorials/OSGi/article.html#plug-in-or-bundles-as-software-component. [Zugriff am 14 02 2017].

[44] T. I. Society, „ Hypertext Transfer Protocol -- HTTP/1.1,“ ietf, 1999.

[45] D. I. PROGRAM, „ TRANSMISSION CONTROL PROTOCOL,“ Defense Advanced Research Projects Agency(DARPA), Arlington, 1981.

[46] T. Dierks, E. Rescorla und I. RTFM, „ The Transport Layer Security (TLS) Protocol Version 1.2,“ ietf, 2008.

[47] I. Fette, I. Google, A. Melnikov und I. Ltd., „The WebSocket Protocol,“ Internet Engineering Task Force (IETF), 2011.

[48] Internet Engineering Task Force (IETF), „rfc6455,“ 12 2011.

[49] MDN contributors, „WebSockets API,“ 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Zugriff am 22 02 2017].

[icann, 16 02 2017. [Online]. Available:
5 https://www.iana.org/assignments/websocket/websocket.xml. [Zugriff am
0 28 02 2017].
]

[T. Oberstein, A. Goedde und T. GmbH, „The Web Application Messaging
5 Protocol," ietf, 2015.
1
]

[G. Hohpe und B. Woolf, Enterprise Integration Patterns: Designing,
5 Building, and Deploying Messaging Solutions, Addison-Wesley
2 Professional, 2003.
]

[M. Corporation, „Publish/Subscribe," 16 02 2017. [Online]. Available:
5 https://msdn.microsoft.com/en-us/library/ff649664.aspx. [Zugriff am 16 02
3 2017].
]

[Oracle and/or its affiliates, „Request-Response," 2017. [Online]. Available:
5 https://docs.oracle.com/cd/E17904_01/doc.1111/e17363/chapter05.htm#
4 FPCON246. [Zugriff am 22 02 2017].
]

[T. GmbH, „http://wamp-proto.org/why/," 2014. [Online]. Available:
5 http://wamp-proto.org/why/. [Zugriff am 24 02 2017].
5
]

[P. S. THOMAS und S. THOMAS, „TEN GOLDEN RULES FOR COST
5 SAVING IN SOFTWARE," *International Journal of Research in*
6 *Engineering & Technology,* Bd. 2, Nr. 3, pp. 25-30, 2014.
]

[S. Dustdar, H. Gall und M. Hauswirth, Software-Architekturen für Verteilte
5 Systeme, Springer, 2003.

7
]

[E. Wolff, „How to Split Your System into Microservices,“ 10 11 2016.
5 [Online]. Available: https://www.slideshare.net/ewolff/how-to-split-your-
8 system-into-microservices. [Zugriff am 25 02 2017].
]

[M. Reiser und N. Wirth, Programming in Oberon: Steps Beyond Pascal
5 and Modula, Addison-Wesley Pub, 1992.
9
]

[G. Inc., „What is Hubot,“ 19 02 2017. [Online]. Available:
6 https://hubot.github.com/. [Zugriff am 19 02 2017].
0
]

[S. Bieliauskas, „Expose: Eine Microservice-Architektur zum
6 konversationsbasierten Erkunden von Soft- warevisualisierungen,“ 19 12
1 2016. [Online]. Available: https://github.com/B-Stefan/bachelor-
] thesis/tree/master/expose. [Zugriff am 19 02 2017].

[A. Kurkin, „Inter-service Communication,“ 20 02 2017. [Online]. Available:
6 http://howtocookmicroservices.com/communication/. [Zugriff am 20 02
2 2017].
]

[O. Gierke und E. Wolff, „Welchen Herausforderungen müssen sich
6 Entwickler im Zusammenhang mit Microservices stellen,“ in *Integration*
3 *von Microservices REST vs Messaeging* .
]

[B. M. Michelson, „Event-driven architecture overview,“ *Patricia Seybold*
6 *Group,* Bd. 2, 2006.
4
]

[Rocket.Chat Contributors, „Rocket.Chat,“ 20 02 2017. [Online]. Available:
6 https://github.com/RocketChat/Rocket.Chat. [Zugriff am 20 02 2017].
5
]

[Infosys ltd. , „An Insight into Microservices Testing Strategies,“ 2016.
6 [Online]. Available: https://www.infosys.com/it-services/validation-
6 solutions/white-papers/documents/microservices-testing-strategies.pdf.
]

[OASIS®, „AMQP (Advanced Message Queuing Protocol),“ 2017. [Online].
6 Available: https://www.amqp.org. [Zugriff am 20 02 2017].
7
]

[M. Rostanski, K. Grochla und A. Seman, „Evaluation of highly available
6 and fault-tolerant middleware clustered architectures using RabbitMQ,“ in
8 *Computer Science and Information Systems (FedCSIS)*.
]

[D. I. a. R. G. John Fallows, „Advanced Message Queuing Protocol
6 (AMQP) WebSocket Binding (WSB) Version 1.0,“ OASIS Committee,
9 2016.
]

[S. O'Grady, 20 07 2016. [Online]. Available:
7 http://redmonk.com/sograd y/2016/07/20/language-rankings-6-16/. [Zugriff
0 am 25 02 2017].
]

[sametmax, „Introduction to WAMP, a protocol enabling PUB/SUB and
7 RPC over Websocket,“ 2014.
1
]

[T. GmbH, „Crossbar.io Documentation,“ [Online]. Available:
7 http://crossbar.io/docs/. [Zugriff am 25 02 2017].

[2] Meteor Development Group Inc., „THE FASTEST WAY TO BUILD JAVASCRIPT APPS,“ 2017. [Online]. Available: https://www.meteor.com/. [Zugriff am 25 02 2017].

[3] M. Fowler, „microservice trade offs,“ 01 07 2015. [Online]. Available: https://martinfowler.com/articles/microservice-trade-offs.html. [Zugriff am 25 02 2017].

[4] T. Clemson, „Testing Strategies in a Microservice Architecture,“ 18 11 2014. [Online]. Available: https://martinfowler.com/articles/microservice-testing/. [Zugriff am 25 02 2017].

[5] A. a. M. S. K. Sumaray, A comparison of data serialization formats for optimal efficiency on a mobile platform, ACM, 2012.

[6] I. Pivotal Software, „Remote procedure call (RPC),“ 2017. [Online]. Available: https://www.rabbitmq.com/tutorials/tutorial-six-python.html.

[7] Crossbar.io Technologies GmbH, „Crossbar Timeservice example,“ 2017. [Online]. Available: https://github.com/crossbario/autobahn-python/blob/master/examples/asyncio/wamp/rpc/timeservice/backend.py. [Zugriff am 20 02 2017].

[8] B. Venners, „The Making of Python,“ 13 01 2013. [Online]. Available: http://www.artima.com/intv/pythonP.html. [Zugriff am 17 02 2017].

[9]

[D. Kuhlman, A Python Book: Beginning Python, Advanced Python, and
8 Python Exercises, Platypus Global Media , 2011.
0
]

[Google Inc. , „What is Google Cloud Pub/Sub?,“ 2017. [Online]. Available:
8 https://cloud.google.com/pubsub/docs/overview. [Zugriff am 20 02 2017].
1
]

Eidesstattliche Erklärung

„Ich versichere, dass ich die vorstehende Arbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe“

_____