Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 62

Instanz-Management für unifizierte Service-Kompositionen

Stefan Fürst

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Frank Leymann
Betreuer/in:	Dipl.-Inf. Katharina Görlach
Beginn am:	21. Mai 2013
Beendet am:	8. November 2013
CR-Nummer:	C.2.4, D.2.11, H.4.1, I.1.2, I.7.2, J.0

Kurzfassung

Service-Kompositionen können durch Sprachen wie BPEL oder ConDec spezifiziert werden. Für die Ausführung der durch diese Sprachen beschriebenen Prozesse werden verschiedene Engines verwendet. Die Engine ist bei der Ausführung eines Prozesses dafür zuständig, die Web Services des Prozesses aufzurufen. Die Web Services befinden sich in einer verteilten Ablaufumgebung an verschiedenen Standorten, wodurch beim Aufruf des Web Services Verzögerungen und Kosten durch Datentransfers entstehen. Die Verzögerungen und Kosten können minimiert werden, indem die Engine nahe den verwendeten Web Services betrieben wird. Um die Zahl der verschiedenen Engines innerhalb einer Ablaufumgebung zu reduzieren, wurde ein Ansatz erforscht, bei dem Prozessspezifikationen von verschiedenen Sprachen zu formalen Grammatiken transformiert werden können. Die Funktionalität der Engine wird dabei durch einen endlichen Automaten realisiert. Für diesen bereits vorhandenen Ansatz wird eine Komponente benötigt, die Instanzen des Automaten erzeugt und deren Betrieb in einer verteilten Ablaufumgebung ermöglicht.
Im Rahmen dieser Bachelorarbeit wird die Realisierung einer Komponente vorgestellt, die Instanzen eines Automaten in einer Cloud Umgebung erstellt und verwaltet. Dabei wird ein Konzept zur Platzierung von Automaten-Instanzen in einer Region innerhalb der Amazon Cloud Umgebung präsentiert. Die Region wird dabei so ausgewählt, dass die die Zugriffskosten auf die Web Services minimiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1. Einleitung

Ein Mittel zur Automatisierung von Geschäftsprozessen ist die Verwendung von service-orientierten Architekturen (SOA). Bei SOA handelt es sich um ein Architekturmuster, bei dem verschiedene Funktionalitäten eines Systems gekapselt als Services angeboten werden. Zur Ausführung einer Aufgabe werden dabei mehrere Services miteinander koordiniert. Zur Implementierung eines Services werden häufig Web Services verwendet. Diese bieten eine Funktion über das Netzwerk an und werden über eine verteilte Ablaufumgebung bereitgestellt. Geschäftsprozesse beschreiben mehrere Abläufe, die sich aus einzelnen Aufgaben zusammensetzen. Die Aufgaben werden durch Web Services implementiert. Zur Umsetzung eines Geschäftsprozesses werden dementsprechend Kompositionen von Web Services verwendet. Um einen Prozess zu spezifizieren, werden Sprachen wie die Business Process Execution Language (BPEL) [JJ] oder ConDec [PA06] verwendet. Zur Ausführung eines Prozesses wird eine *Engine* benötigt. Diese navigiert über den Prozess und führt die in der Beschreibung des Prozesses spezifizierten Aktivitäten aus. Eine Aktivität kann dort zum Beispiel der Aufruf eines Web Services sein.

Für verschiedene Sprachen zur Beschreibung von Service-Kompositionen werden unterschiedliche Engines benötigt. Falls mehrere Prozesse vorliegen, die in unterschiedlichen Sprachen spezifiziert sind, müssen innerhalb einer Ablaufumgebung mehrere Engines gleichzeitig betrieben werden. Um dieses Problem zu beheben, wird ein Ansatz zur Unifizierung von Service-Kompositionen erforscht [GLC13]. Dieser verwendet formale Grammatiken zur Beschreibung von Service-Kompositionen und endliche Automaten, um diese auszuführen. Dabei sollen Beschreibungen von Service-Kompositionen, die in BPEL oder ConDec verfasst sind, in formale Grammatiken übersetzt werden. Dadurch wird es möglich, mit einem endlichen Automaten die Beschreibungen von Service-Kompositionen aus verschiedenen Sprachen auszuführen. Die formalen Grammatiken dienen hierbei zur Spezifikation von Prozessen. Der endliche Automat stellt die Komponente dar, die für die Navigation über einen Prozess verantwortlich ist. Der endliche Automat wird zudem um eine Komponente zum Aufruf von Web Services erweitert. Da sich die verwendeten Web Services über mehrere Regionen in der Welt verteilt befinden können, entstehen durch den Kontakt mit den verwendeten Web Services unter Umständen große Latenzen, welche die Ausführung verzögern. Zudem ist es möglich, dass die Web Services große Datenmengen benötigen, die über die Distanz zwischen Automat und Web Service transferiert werden müssen. Daher ist es sinnvoll, die ausführende Instanz des Automaten nahe den benötigten Web Services zu platzieren, um die Distanzen, und damit auch die entstehenden Latenzen, zu minimieren.

Das Ziel dieser Arbeit ist die Entwicklung und Implementierung einer Instanz-Management Komponente, welche für die Instanziierung von endlichen Automaten in einer verteilten Ablaufumgebung (Cloud) verantwortlich ist. Diese soll, in Abhängigkeit der verwendeten

Web Services einer Service-Komposition, ermitteln, in welcher Region die Instanz des ausführenden Automaten erzeugt werden soll. Dadurch soll eine möglichst effiziente Ausführung der Komposition ermöglicht werden. Eine Region ist dabei ein Ort, an dem die Cloud Instanz einer Anwendung in einem Rechenzentrum betrieben werden kann. Der bereits vorhandene Automat soll zudem in einer generischen Form als Web Service angeboten werden. Eine Instanz dieses Web Services soll also beliebig viele formale Grammatiken ausführen können. Außerdem soll auf die Skalierbarkeit der Lösung geachtet werden. Das heißt, es sollen entsprechend der Anzahl an Anfragen (formalen Grammatiken) neue Instanzen des Web Services in der Cloud erzeugt werden. Hierzu soll die Instanz-Management Komponente erkennen, wann eine Instanz nicht die benötigten Ressourcen für eine schnelle Verarbeitung der Anfragen besitzt. Umgekehrt sollen keine unnötigen Instanzen des Automaten existieren, um möglichst sparsam mit den verfügbaren Ressourcen umzugehen.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen und Forschungsstand In diesem Kapitel werden einige Grundbegriffe erklärt. Außerdem wird auf die Funktionsweise der bereits vorhandenen Komponenten und die Rolle der Instanz-Management Komponente im Gesamtsystem eingegangen. Zusätzlich dazu wird ein bereits vorhandener Ansatz zur Verteilung von Anwendungen auf verschiedene Regionen vorgestellt.

Kapitel 3 – Anforderungen und Ablaufumgebung In den Anforderungen werden die funktionalen und nichtfunktionalen Zieleigenschaften der Instanz-Management Komponente festgelegt. Außerdem werden verschiedene Anbieter von Cloud Services evaluiert.

Kapitel 4 – Entwurf Im Entwurf wird die Software konzeptioniert, deren Funktionen sich aus den Anforderungen ergeben. Hierbei wird nicht auf konkrete Technologien eingegangen, sondern es erfolgt eine abstrakte Beschreibung der Umsetzung der Funktionalitäten.

Kapitel 5 – Implementierung Im Implementierungskapitel werden zunächst die Technologien erläutert, die bei der Umsetzung der Software verwendet wurden. Danach wird die Architektur des Systems vorgestellt und es wird auf Implementierungsdetails wichtiger Komponenten eingegangen.

Kapitel 6 – Diskussion Das Ergebnis der Arbeit wird hier in Form eines Beispieles vorgestellt. Außerdem wird untersucht, inwiefern die Anforderungen erfüllt wurden und es findet eine Diskussion des Ergebnisses statt.

Kapitel 7 – Zusammenfassung und Ausblick Abschließend werden die Arbeit und deren Erkenntnisse zusammengefasst. Zusätzlich wird ein Ausblick gegeben, welche Herausforderungen noch bewältigt werden müssen.

2. Grundlagen und Forschungsstand

In diesem Kapitel werden zunächst in Abschnitt 2.1 einige Grundbegriffe der Arbeit erläutert. Danach wird in Abschnitt 2.2 die Funktion der bereits vorhandenen Komponenten und deren Interaktion mit der Instanz-Management Komponente erklärt. Zuletzt wird in Abschnitt 2.3 eine thematisch verwandte Arbeit vorgestellt, die sich mit der effizienten Verteilung von Instanzen auf verschiedene Regionen befasst.

2.1. Grundbegriffe

Zunächst sollen in diesem Abschnitt einige Begriffe erklärt werden, die eine zentrale Rolle in der Arbeit spielen.

2.1.1. Cloud Computing und Cloud Regionen

Als Ablaufumgebung der zu entwickelnden Instanz-Management Komponente soll eine Cloud verwendet werden. Das National Institute of Standards and Technology (NIST), das dem U.S. Department of Commerce untersteht, definiert Cloud Computing wie folgt (Übersetzung vom Bundesamt für Informationssicherheit):

„*Cloud Computing ist ein Modell, das es erlaubt, bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.*“ [MG11] [clo]

NIST nennt dabei folgende relevanten Charakteristiken [MG11]:

- Die angebotenen Ressourcen werden geteilt von mehreren Akteuren verwendet (*resource pooling*).
- Der Zugriff auf die Ressourcen erfolgt auf Anforderung und ohne menschliche Interaktion mit dem Anbieter des Services (*on-demand self-service*).
- Der Zugriff auf die Ressourcen ist mit Standardmechanismen über das Netzwerk möglich. Somit werden heterogene Klienten unterstützt (*broad network access*).
- Kapazitäten können elastisch erworben und freigegeben werden. Dies kann automatisch abhängig von der Nutzungsintensität geschehen. Aus Sicht des Anwenders scheinen die Ressourcen unendlich groß zu sein (*rapid elasticity*).

- Die Nutzung der Ressourcen kann überwacht und gesteuert werden (*measured service*).

Eine Cloud bietet folglich die Möglichkeit, dynamisch und mit minimalem Aufwand auf einen geteilten Pool von Ressourcen zuzugreifen. Bei den Ressourcen kann es sich zum Beispiel um Rechenkapazität oder Speicher handeln. Die Ressourcen einer Cloud werden dabei stets als Service angeboten. Ein wichtiger Begriff im Zusammenhang mit Cloud Computing ist die *Cloud Region*. Anbieter von Cloud Services (zum Beispiel Amazon Web Services) verwenden zur Bereitstellung der Cloud Infrastruktur häufig Rechenzentren in verschiedenen geographischen Regionen. Eine geographische Region, in der ein Rechenzentrum zur Verfügung steht, wird in diesem Zusammenhang als Cloud Region beziehungsweise als Region bezeichnet. Bei der Verwendung von Cloud Angeboten kann die Region oft explizit ausgewählt werden.

2.1.2. Web Service und Service-Komposition

Cloud Computing basiert auf der Nutzung von Services, das heißt eine serviceorientierte Architektur ist eine Voraussetzung einer Cloud. Zur Umsetzung von Services werden Web Services verwendet. Das World Wide Web Consortium (W3C) definiert einen Web Service wie folgt:

„*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*“ [HB]

Web Services erlauben also Interaktion zwischen verschiedenen Systemen über das Netzwerk. Mit SOAP wird ein XML basiertes Nachrichtenformat verwendet, wodurch auf einen Web Service unabhängig von Plattform und Programmiersprache zugegriffen werden kann. Die Beschreibung der Operationen, die ein Web Service anbietet, erfolgt über die Web Service Definition Language (WSDL), die ebenfalls auf XML basiert [DJMZ05, S.26-27].

Bei einer Service-Komposition wird die Funktionalität von mehreren Web Services zu einem neuen Service kombiniert [ACHM04, S.245]. Service-Kompositionen werden verwendet, um die Komplexität von Aufgaben zu reduzieren. Dabei können komplexe Web Services inkrementell aus Web Services mit niedrigerem Abstraktionslevel aufgebaut werden [ACHM04, S.247]. Prinzipiell können Service-Kompositionen nach Orchestrierungen und Choreographien unterschieden werden. Bei einer Orchestrierung wird der Ablauf eines Prozesses und damit die Aufrufe der Web Services zentral gesteuert, während bei einer Choreographie die Web Services miteinander kooperieren [DJMZ05, S202-203]. Im Folgenden wird im Zusammenhang von Service-Kompositionen immer von Orchestrierungen ausgegangen.

2.1.3. Formale Grammatiken

Für die Ausführung einer Service-Komposition muss eine Beschreibung vorliegen, in der die verschiedenen Abläufe spezifiziert werden. In der vorliegenden Arbeit werden hierfür formale Grammatiken verwendet.

Bei einer formalen Grammatik handelt es sich um ein Konstrukt aus der theoretischen Informatik, welches dazu dient formale Sprachen zu beschreiben [Scho8, S.3]. Bei einer formalen Sprache handelt es sich um eine Kombination von Elementen eines definierten Alphabets [Scho8, S.3]. Eine Grammatik beschreibt dabei Nicht-Terminale, Terminale, Produktionsregeln und ein Startsymbol [Scho8, S.5]. Mit der Hilfe von Produktionsregeln können Symbolen durch andere Symbole substituiert werden. Symbole sind Terminale oder Nicht-Terminale. In dem Zusammenhang von Produktionsregeln wird oft von der linken und der rechten Seite einer Regel gesprochen. Die linke Seite beinhaltet dabei die Symbole, aus denen abgeleitet wird und die rechte beinhaltet die Symbole, die das Ergebnis der Regel darstellen. Die Produktionsregeln einer Grammatik werden so lange angewendet, bis sich keine weitere Regel mehr anwenden lässt. Das Startsymbol gibt an, mit welcher Regel begonnen werden muss. Dabei muss das Startsymbol aus der Menge der Nicht-Terminale stammen.

2.2. Das Gesamtsystem des Automaten

Der Ansatz zur Spezifikation und Ausführung von Service-Kompositititionen, der dieser Arbeit zugrunde liegt, lässt sich folgendermaßen zusammenfassen:
Die Beschreibung aller nötigen Artefakte einer Service-Komposition erfolgt durch eine formale Grammatik. Diese kann in einem Unifizierungsprozess, beispielsweise aus einem BPEL-Prozessmodell erstellt werden. Die formalen Grammatiken werden durch einen endlichen Automaten ausgeführt. Dieser ist zum Beispiel auch dafür verantwortlich, die Aufrufe von Web Services zu initiieren. Um eine effiziente Ausführung des Automaten zu gewährleisten, soll dieser möglichst nahe an den verwendeten Web Services platziert werden. Hierfür ist die Instanz-Management Komponente zuständig.

In diesem Abschnitt soll die Funktionsweise des Automaten, der Aufbau der Grammatiken und das Zusammenspiel dieser Komponente mit der Instanz-Management Komponente erklärt werden. Auf das Unifizierungsverfahren, also die Umwandlung der Spezifikationen von Service-Kompositionen verschiedener Sprachen in eine formale Grammatik, wird in dieser Arbeit nicht eingegangen, da dieses unabhängig von der Ausführung des Automaten und der Instanz-Management Komponente ist. Eine Beschreibung des Unifizierungsverfahrens findet sich in [Gör13].

2.2.1. Service Grammatiken

Die formalen Grammatiken, die zur Spezifikation einer Service Komposition verwendet werden, werden auch als Service Grammatiken bezeichnet. Sie stellen die Grundlage für

die Ausführung eines Automaten dar, da sie die Regeln enthalten, nach denen sich der Automat verhält. In einer Service Grammatik werden Terminale, Nicht-Terminale und die Produktionsregeln bekannt gegeben, wie in Abschnitt 2.1.3 beschrieben. Es gibt jedoch gewisse Unterschiede zu gewöhnlichen formalen Grammatiken [GLC13].

Der erste Unterschied besteht darin, dass Nicht-Terminale um einen Typ erweitert werden. Dieser dient dazu, den Aufruf eines Web Services über ein Nicht-Terminal zu ermöglichen. Der Typ eines Nicht-Terminals wird in der Grammatik spezifiziert. Dabei werden die zum Aufruf eines Web Services nötigen Informationen angegeben. Dies sind zum Beispiel die Adresse des Web Services und die Ein- und Ausgabeparameter.

Außerdem unterscheiden sich die Produktionsregeln von denen einer gewöhnlichen formalen Grammatik. Dort sind nichtdeterministische Produktionsregeln nicht erlaubt, das heißt es sind nicht mehrere Regeln möglich, die sich auf dieselbe Folge von Symbolen beziehen. Bei den hier verwendeten Service Grammatiken hingegen ist Nichtdeterminismus in bestimmten Situationen erlaubt. Dies ist genau dann der Fall, wenn der Nichtdeterminismus zur Laufzeit des Automaten aufgelöst wird. Dies bedeutet konkret, dass der Automat Informationen zur Laufzeit erhalten muss, die bestimmen, welche Regel ausgeführt wird. Hierfür werden Eingabe-Nicht-Terminale definiert. Diese sind mit Information verknüpft, die der Automat zur Laufzeit liefert. Dementsprechend dürfen mehrere Produktionsregeln, die auf der linken Seite dieselben Symbole verwenden nur genau dann existieren, wenn sie als Ergebnis ein Eingabe-Nicht-Terminal liefern. [GLC13]

Ein Auszug einer Service Grammatik befindet sich in Anhang A. Hier ist jeweils ein Beispiel für ein Nicht-Terminal, einen Nicht-Terminal Typen, ein Terminal und eine Produktionsregel angegeben.

2.2.2. Die Funktionsweise des Automaten

Die formalen Grammatiken werden von einem endlichen Automaten ausgeführt. Die Funktionsweise des Automaten und seinen zugehörigen Komponenten wird in Abb. 2.1 dargestellt. Der Automat liest ein Nicht-Terminal ein und wendet auf dieses eine Produktionsregel an. Dabei kann das Nicht-Terminal durch ein weiteres Nicht-Terminal oder ein Terminal substituiert werden. Es wird mit dem Startsymbol der Grammatik begonnen und es wird so lange fortgefahren, bis sich keine Regeln mehr anwenden lassen.

Ein Nicht-Terminal besitzt dabei einen Typ (siehe 2.2.1), durch den ein Nicht-Terminal mit dem Aufruf eines Web Services assoziiert sein kann. In diesem Fall gibt der Typ des Nicht-Terminals die Adresse eines Web Services an. Der Aufruf des Web Services wird durch den Automaten initiiert und erfolgt durch die *Service Invocation* Komponente.

Bei der Service Invocation müssen gewöhnlich Daten in Form von Parametern übermittelt werden. Die Daten werden per Referenz verwaltet. Hierbei dient das *Reference Resolution System* (RRS) zur Speicherung der Daten. Das RRS ist ein Web Service, der unter Angabe einer Referenz die entsprechenden Daten zurückliefert. Die Referenzen für die Service Invocation

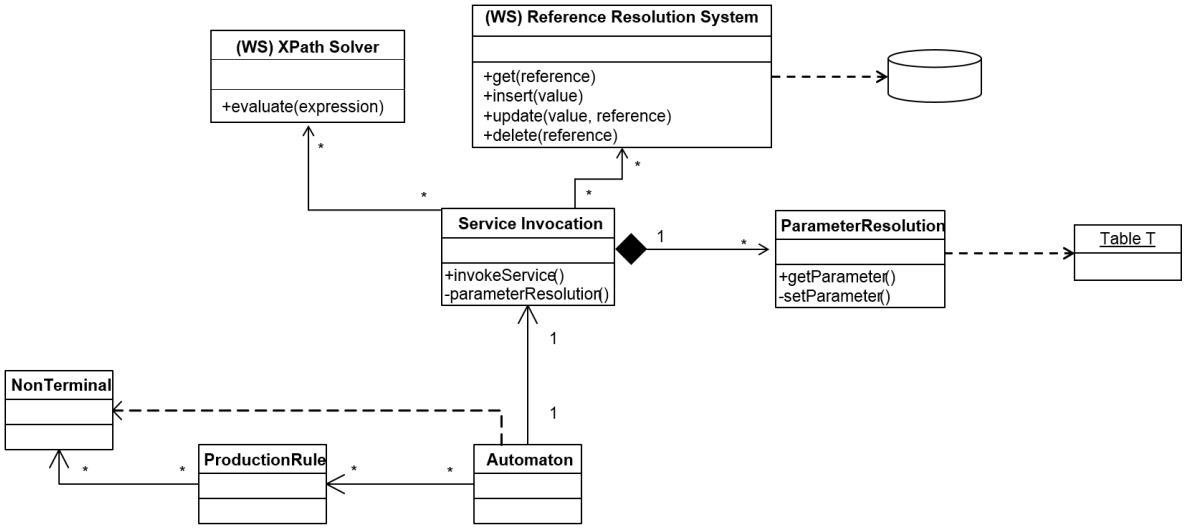

Abbildung 2.1.: Die einzelnen Komponenten des Automaten in einem UML Klassendiagramm.

werden durch die Nicht-Terminale zur Verfügung gestellt. Innerhalb einer Kompositionsinstanz werden die Referenzen durch die *ParameterResolution* Komponente verwaltet.

Schließlich kann der tatsächliche Aufruf des gewünschten Web Services erfolgen (zum Beispiel in Abb. 2.1 ein Web Service zur Auflösung von XPath Ausdrücken). Das Ergebnis des Service Aufrufs wird im RRS gespeichert.

2.2.3. Die Funktion der Instanz-Management Komponente im Gesamtsystem

Der Automat soll von der Instanz-Management Komponente verwaltet werden. Dies wird in Abb. 2.2 dargestellt. Die Instanz-Management Komponente verwaltet die Instanzen des Automaten und der Komponenten, die für die Ausführung des Automaten erforderlich sind (in Abb. 2.2 als *Engine Core Components* bezeichnet). Dazu gehört der Automat selbst (*Queued Automaton*), der die Produktionsregeln der Grammatik (*Grammar*) anwendet. Bei der Ausführung der Grammatik ist es nötig, die Nicht-Terminal Typen aufzulösen und entsprechende Service Aufrufe durchzuführen. Die wird in Abb. 2.2 durch *Non-Terminal Type Resolution* und *Service Invocation* dargestellt. Die Grammatik ist unabhängig vom Automaten selbst. Sie kann beliebig ausgetauscht oder ausgeführt werden.

Das RRS, die Datenbank des RRS, die Evaluierung von XPath Ausdrücken und sonstige Web Services, die während der Ausführung aufgerufen werden(*Engine Context Components*), stehen in keiner direkten Beziehung zu der Instanz-Management Komponente. Sie werden lediglich während der Service Invocation verwendet, wie im vorigen Abschnitt beschrieben. Jedoch muss zur Ausführung einer formalen Grammatik zwingend ein RRS verfügbar sein.

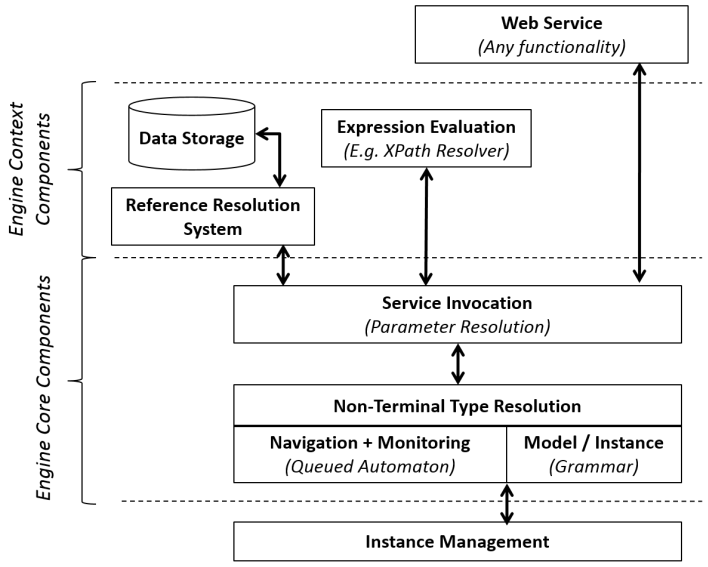

Abbildung 2.2.: Die Instanz-Management Komponente im Gesamtsystem.

2.3. Forschungsstand

Die Instanz-Management Komponente hat nicht nur die Aufgabe, Instanzen eines Automaten zu erzeugen, sondern diese auch sinnvoll auf verschiedene Regionen zu verteilen. Im Folgenden wird ein bereits erforschter Ansatz vorgestellt, der sich mit der Verteilung von Web Services auf verschiedene Regionen befasst. Dieser dient als Grundlage für den später vorgestellten Algorithmus zur Verteilung von Instanzen.

In [GL12] wird der Begriff *Closeness* eingeführt, um zu bewerten, in welcher Region die Instanz eines Web Services erstellt werden soll. Dabei sollen Verzögerungen und Kosten durch Datentransfers minimiert werden. Closeness besteht sowohl aus einem statischen als auch einem dynamischen Aspekt. Der statische Teil verwendet ein Kontext Modell, das definiert, welche Abhängigkeiten Web Services zu anderen Web Services besitzen. Die möglichen Kontext Typen müssen dabei manuell definiert werden und teilen sich in verschiedene Klassen auf. Diese sind häufig abhängig von der verwendeten Infrastruktur. In einer verteilten Umgebung könnten zum Beispiel Ressourcen in dieselbe Klasse eingeordnet werden, wenn sich sich auf demselben Server befinden. Eine andere Klasse wäre es, wenn sich die verwandten Ressourcen im gleichen Rechen-Cluster befinden. Der dynamische Teil der Closeness wird als *Betweenness* bezeichnet. Zwischen jedem Paar voneinander abhängiger Web Services gibt es einen Betweenness Wert, der ebenfalls von der Infrastruktur abhängig ist. Der Betweenness Wert berechnet sich zum Beispiel daraus, wie viele Daten zwei Web Services austauschen und wie groß die Wahrscheinlichkeit ist, dass die beiden Web Services Kontakt zueinander aufnehmen. Wenn beispielsweise ein neuer Web Service X bereitgestellt werden soll und dieser die Web Services S1 und S2 verwendet, wird der Beetweenness Wert zwischen X und S1 und X und S2 berechnet. Angenommen zwischen X und S1 wird mit

100% Wahrscheinlichkeit eine Datenmenge von 5 GB ausgetauscht und zwischen X und S2 wird mit 50% Wahrscheinlichkeit eine Datenmenge von 6 GB ausgetauscht. In diesem Fall ist der Betweenness Wert zwischen X und S1 höher, da hier im Mittel mehr Daten ausgetauscht werden. Die Betweenness Werte müssen stets dynamisch errechnet werden, um beispielsweise das Auftauchen neuer Web Services verarbeiten zu können.

Außerdem wird in [GL12] ein Ansatz zur dynamischen Verteilung von Instanzen vorgeschlagen. Hierzu werden ebenfalls die bereits erwähnten Kontext Klassen verwendet. Dabei wird zunächst nach wenig ausgelasteten Instanzen mit einem niedrigen Kontext gesucht, das heißt nach einer möglichst nahen Instanz mit niedriger CPU Last. Wird eine solche nicht gefunden, wird eine neue Instanz erzeugt. Bei der Erzeugung von neuen Instanzen werden die Kontext Klassen und die Betweenness miteinbezogen. Dabei werden die verwendeten Web Services als Orientierungspunkte verwendet und mithilfe der Kontext Klassen werden verschiedene Alternativen berechnet, die als Ziel zur Platzierung des neuen Services dienen können. Die Entscheidung wird getroffen, indem für jede Alternative die Betweenness zu den anderen Web Services berechnet und dann verglichen wird.

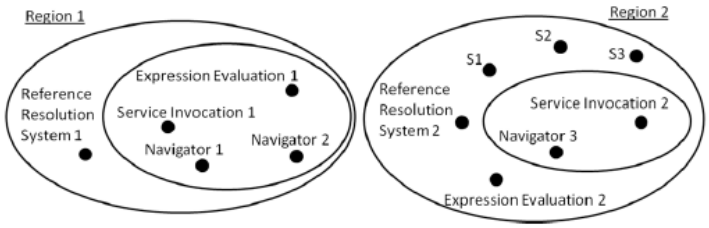

Abbildung 2.3.: Beispiel für die Verteilung von Instanzen anhand Betweenness und Kontext Klassen. Die Punkte stellen Instanzen von verschiedenen Web Services dar. [GL12]

In Abbildung 2.3 wird ein Beispiel für die Berechnung anhand verschiedener Web Services gezeigt. Das Kontext Modell gibt vor, dass Navigator und Service Invocation sich sehr nahe beieinander befinden müssen (Kontext Klasse 1). Navigator und und das Reference Resolution System müssen sich ebenfalls nahe beieinander befinden (Kontext Klasse 2). Die sonstigen Web Services (S1, S2, S3) dürfen an einem beliebigen Standort instantiiert sein (Kontext Klasse 3). Sowohl Region 1 als auch Region 2 aus Abbildung 2.3 erfüllen diese Vorgabe. Der Betweenness Wert entscheidet folglich, in welcher Region ein neuer Service X bereitgestellt wird. Da sich S1, S2 und S3 in Region 2 befindet, werden die Betweenness Werte zwischen X und S1, S2, S3 in die Berechnung miteinbezogen. In Region 1 ist dies nicht der Fall. Region 2 erhält damit insgesamt einen höheren Betweenness Wert als Region 1. Damit wird Region 2 für die Bereitstellung von X gewählt.

3. Anforderungen und Ablaufumgebung

Nachdem die Grundlagen nun vorgestellt worden sind und die grobe Funktion der Instanz-Management Komponente in das Gesamtsystem eingeordnet wurde, werden in diesem Kapitel die Zieleigenschaften in Form von Anforderungen ausformuliert. Die Anforderungen ergeben sich dabei aus der Aufgabenstellung. Hierbei werden in Abschnitt 3.1 die funktionalen und in Abschnitt 3.2 die nichtfunktionalen Anforderungen erklärt. Die Anforderungen sind nummeriert.

Eine generelle Vorgabe dieser Arbeit ist die Verwendung einer verteilten Ablaufumgebung (*Cloud*) zur Ausführung der Instanz-Management Komponente und des Automaten. Aus diesem Grund erfolgt in Abschnitt 3.3 die Auswahl der späteren Ablaufumgebung für die Instanz-Management Komponente. Hierfür werden zunächst mehrere Vergleichskriterien definiert, die Anforderungscharakter besitzen. Anschließend werden die Anbieter bezüglich dieser Anforderungen evaluiert. Eine weitere Vorgabe ist die Verwendung der Programmiersprache Java für die Umsetzung. Dies wird bei der Auswahl der Ablaufumgebung bereits berücksichtigt.

Im Folgenden wird außerdem unter einer Anfrage immer eine formale Grammatik verstanden, die durch einen Automaten verarbeitet werden soll. Es ist außerdem möglich, dass ein Automat bei der Verarbeitung einer Grammatik zusätzliche Nachrichten vom Client erhält. Zur Abgrenzung wird eine solche Nachricht als *Message* bezeichnet.

3.1. Funktionale Anforderungen

A1 - Regionen Berechnung Die Instanz-Management Komponente soll eine Funktion bieten, um die Region zu errechnen, in der eine Grammatik durch einen Automaten verarbeitet werden soll. Dies soll auf Basis der in der Grammatik angegebenen Web Services geschehen.

A2 - Verwaltung von verschiedenen Instanzen eines Automaten Wenn eine Anfrage für eine bestimmte Region vorliegt, soll die Instanz-Management Komponente diese Anfrage an einen Automaten in dieser Region weiterleiten. Existiert in einer Region kein Automat, muss dort eine neue Instanz erstellt werden. Außerdem muss die Instanz-Management Komponente dazu in der Lage sein, bereits vorhandene Instanzen des Automaten und alle zugehörigen Ressourcen zu löschen. Die Instanz-Management Komponente muss stets Zugriff auf alle existierenden Instanzen besitzen. Die Instanz-Management Komponente soll außerdem dazu fähig sein, Automaten in verschiedenen Regionen zu verwalten.

A3 - Ausführung des Automaten möglich Der Automat soll in einer Art und Weise in der Ablaufumgebung bereitgestellt werden, dass eine fehlerfreie Ausführung des Automaten möglich ist. Dabei sollen dem Automaten alle benötigten Komponenten zur Verfügung gestellt werden. Dies beinhaltet insbesondere auch die Möglichkeit, den Automaten mit Messages zu versorgen (*Message Correlation*).

A4 - Überblick über Web Services Um eine Berechnung der Zielregion einer Grammatik durchführen zu können, soll die Instanz-Management Komponente die verfügbaren Web Services in der verwendeten Ablaufumgebung kennen. Die Adresse und die Region der Web Services muss also bekannt sein. Befindet sich ein Web Service nicht in der verwendeten Ablaufumgebung, muss er der Instanz-Management Komponente nicht bekannt sein.

A5 - Instanzverwaltung des Reference Resolution Systems (RRS) Eine unverzichtbare Komponente des Gesamtsystems ist das Reference Resolution System (siehe Abschnitt 2.2.2). Dieses wird als Web Service angeboten. Das Erzeugen und Löschen der Instanzen des RRS soll ebenfalls die Instanz-Management Komponente übernehmen. Hierbei soll in jeder Region, in der sich derzeit mindestens eine Instanz eines Automaten befindet, eine Instanz des RRS platziert werden. Befindet sich in einer Region keine Automateninstanz, soll dort auch kein RRS zur Verfügung stehen.

A6 - Generischer Automat Die Anwendung in der Ablaufumgebung, die für die Ausführung von Grammatiken verantwortlich ist, soll nicht an einen festen Automaten gekoppelt sein. Stattdessen soll diese mehrere beliebige Automaten enthalten, die verschiedene Grammatiken ausführen können.

3.2. Nichtfunktionale Anforderungen

A7 - Garantierte Verarbeitung einer Anfrage Es soll sichergestellt werden, dass eine Anfrage, das heißt eine Grammatik, die beim Instanz-Manager eingeht, auf jeden Fall verarbeitet wird. Dies soll unabhängig von Fehlern sein, die durch die Instanz-Management Komponente oder durch die Umgebung des Automaten verursacht werden. In diesem Fall darf die Anfrage durch einen Fehler nicht verloren gehen. Wird ein Fehler durch eine syntaktisch inkorrekte Grammatik ausgelöst, muss diese nicht verarbeitet werden.

A8 - Optimale Instanzenverteilung Bei der Berechnung der Zielregion soll die Region gewählt werden, die bei der Ausführung der Grammatik die geringsten Kosten aller verfügbaren Regionen verursacht. Dabei sollen insbesondere die Verzögerungen durch den Aufruf von Web Services berücksichtigt werden.

A9 - Effizienter Umgang mit Ressourcen Da in Cloud Umgebungen üblicherweise genau die Ressourcen bezahlt werden, die verwendet wurden (Pay-Per-Use), ist ein sparsamer Umgang mit den Cloud Services sinnvoll. Hierbei sollen insbesondere keine unnötigen Rechenkapazitäten oder nicht benötigter Speicherplatz verwendet werden. Stattdessen

soll möglichst effizient mit bereits vorhanden Ressourcen umgegangen werden. Umgekehrt bedeutet dies, dass nicht oder wenig verwendete Instanzen sowie nicht mehr benötigte Dateien gelöscht werden sollen.

A10 - Skalierbarkeit Um große Mengen an Anfragen verarbeiten zu können, ohne stark steigende Wartezeiten für das Verarbeiten einer Anfrage zu erhalten, soll die Infrastruktur mit der Zahl der Anfragen skalieren. Dies bezieht sich insbesondere auf die Anzahl der Rechnerinstanzen in einer Region, die dem System zur Verfügung stehen.

3.3. Evaluation verschiedener Cloud Anbieter

Die spätere Implementierung soll in einer verteilten Ablaufumgebung, das heißt einer Cloud, lauffähig sein. Hierfür wird die Cloud Umgebung des Unternehmens Amazon gewählt. In diesem Abschnitt wird diese Auswahl ausführlich begründet, da es sich bei der Cloud Umgebung um ein zentrales Werkzeug dieser Arbeit handelt. Hierzu werden verschiedene Cloud Anbieter evaluiert. Bei der Entscheidung wurden *Windows Azure*[Mich], *Google Cloud* [Gooe] und *Amazon Web Services* (AWS)[Amab] berücksichtigt.

Die Ergebnisse werden in Tabelle 3.1 zusammengefasst. Die dort angewandten Kriterien werden in Abschnitt 3.3.1 erklärt. Eine ausführlichere Untersuchung der Angebote sowie das Ergebnis finden sich in den darauffolgenden Abschnitten.

3.3.1. Bewertungskriterien

Für die Bewertung der Anbieter wurden mehrere Kriterien ausgewählt, die für die Entwicklung der Instanz-Management Komponente relevant sind. Diese stellen gleichzeitig die Anforderungen an die Cloud Angebote dar, die erfüllt werden müssen. Diese Kriterien werden hier erklärt.

Regionen Das Kriterium Regionen gibt an, ob die Region, in der eine Anwendung erstellt wird, explizit ausgewählt werden kann. Diese Funktion ist unerlässlich für die Instanz-Management Komponente, da diese einen Automaten abhängig von den verwendeten Web Services in einer Region platzieren muss.

PaaS Eine wichtige Funktion einer Cloud Umgebung ist die Verfügbarkeit eines *Platform as a Service* (PaaS) Angebots. Hierbei soll es möglich sein, eine Anwendung allein durch das Bereitstellen eines entsprechenden Containers zu starten, während der Service sich um die Bereitstellung der Infrastruktur kümmert. Dieser Punkt ist relevant, da er die Bereitstellung einer Anwendung vereinfacht.

Java API Da als Programmiersprache Java gewählt wird, ist der Umfang der Java API wichtig. Es wird folglich untersucht, welche Möglichkeiten es gibt, die Cloud Umgebung durch die Java API zu modifizieren.

	Regionen	PaaS	Java API	Konfigurierbarkeit	sonstige Services	Gratis Angebote
AWS	Ja	Ja	Umfangreich	Umfangreich	Viele	Ja
Azure	Ja	Ja	Eingeschränkt	Umfangreich	Viele	Teilweise
Google	Ja	Ja	Umfangreich	Eingeschränkt	Wenige	Ja

Tabelle 3.1.: Verschiedene Anbieter von Cloud Services im Vergleich.

Konfigurierbarkeit Ein weiteres Merkmal, das bewertet wird, ist die generelle Konfigurierbarkeit der Cloud Umgebung, zum Beispiel welche Rechenleistung zur Verfügung steht, oder wie genau sich Komponenten, wie beispielsweise ein Load-Balancer, konfigurieren lassen.

Sonstige Services Bei diesem Punkt wird untersucht, welche weiteren Services zur Verfügung gestellt werden, die nützlich sind, jedoch nicht zwingend für die Umsetzung der Instanz-Management benötigt werden. Darunter fallen Services zur automatischen Skalierung einer Anwendung, Warteschlangen für Nachrichten (Queue) und Services, die Anfragen gleichmäßig auf verschiedene Rechnerinstanzen einer Anwendung verteilen (Load-Balancing).

Gratis Angebote Da für die Arbeit keine Gelder zur Verfügung gestellt werden, ist ein weiterer wichtiger Aspekt, ob und wieweit sich die Angebote gratis verwenden lassen. Außerdem soll untersucht werden ob Studienprojekte durch den jeweiligen Anbieter unterstützt werden.

3.3.2. Untersuchung der Anbieter

Die verschiedenen Anbieter werden nun nach den eben erläuterten Anforderungen bewertet. Die Ergebnisse hiervon befinden sich in Tabelle 3.1.

Amazon Web Services

Bei den Angeboten von Amazon Web Services lässt sich eine Region explizit auswählen. Hierbei stehen in Amerika, Europa und Asien jeweils mehrere Subregionen zur Auswahl bereit. Als Platform as a Service Angebot steht *Elastic Beanstalk* zur Verfügung [Amap]. Hierbei muss lediglich ein Container, der die gewünschte Anwendung enthält, zur Verfügung gestellt werden. Dieser Vorgang lässt sich auch über die Java API durchführen [Amad]. Die Java API bietet sämtliche Konfigurationsmöglichkeiten, die auch über das Web Interface zur Verfügung stehen. Die Möglichkeiten zur Konfiguration sind insgesamt sehr detailliert. Es stehen verschiedene Hardwarekonfigurationen für die Recheninstanzen zur Verfügung [Amap]. Das Betriebssystem sowie die sonstige Software auf den Instanzen lässt sich beliebig konfigurieren. Des Weiteren steht ein Service für Queues [Aman], sowie ein Load-Balancer [Amae]

und automatische Skalierung [Amao] zur Verfügung. Auch diese Komponenten lassen sich frei konfigurieren [Amac]. Sämtliche, für diese Arbeit relevanten Services, stehen außerdem zeitlich begrenzt gratis zur Verfügung [Amag]. Zudem unterstützt Amazon Studenten mit eigenen Projekten mit Gutscheinen für die Nutzung von AWS [Amah].

Windows Azure

Windows Azure lässt beim Start von verschiedenen Services eine Auswahl der Region zu. Verfügbare Regionen sind Nord-Amerika, Europa, Ozeanien und Asien [Micj]. Diese lassen sich teilweise noch weiter aufschlüsseln. Zudem biete Windows Azure ein Platform as a Service Angebot (*Azure Cloud Services*). Um eine Java Anwendung bereitzustellen, müssen hierbei zunächst mehrere Konfigurationsdateien erstellt werden, die auf dem Server die benötigte Java Umgebung erzeugen [Micc]. Hierbei werden Services wie automatische Skalierung selbstständig bereitgestellt [Micg]. Die Java API hat nur einen eingeschränkten Funktionsumfang und bietet keinen Zugriff auf alle Möglichkeiten. Beispielsweise ist es nicht möglich eine Anwendung über die Java API bereitzustellen. Hierfür muss teilweise eine REST-API und ein Kommandozeilen Interface zur Hilfe genommen werden [Mica]. Die einzelnen Komponenten bieten detaillierte Konfigurationsmöglichkeiten. Auch sonstige Services wie Queues [Micd] und Komponenten für automatische Skalierung, sowohl vertikal als auch horizontal, und Load-Balancing stehen zur Verfügung [Mice]. Die Möglichkeiten, Windows Azure gratis zu benutzen, sind jedoch eingeschränkt. Insbesondere virtuelle Rechenkapazitäten lassen sich nur gegen Bezahlung verwenden [Micf]. Dafür erhält der Anwender bei Registrierung einen Gutschein, der die Nutzung der Services für eine gewisse Zeit ermöglicht [Micb]. Auch bei Windows Azure gibt es Programme um Studierende durch Gratis Nutzungskontingente zu unterstützen [Mici].

Google Cloud

Auch bei den Services der Google Cloud lässt sich die Region explizit auswählen. Hier stehen jedoch nur die Regionen Nord-Amerika und Europa zur Verfügung [Gooa]. Mit *Google App Engine* (GAE) steht ein Platform as a Service Angebot bereit, das es erlaubt, eine Anwendung nur durch den Upload eines Containers zu starten[Goob]. GAE erlaubt jedoch keinen Zugriff auf die verwendeten Ressourcen wie die Rechnerinstanzen. Wenn dies gewünscht ist, muss stattdessen *Google Compute Engine* verwendet werden. Hierbei kann virtuelle Rechenkapazität gemietet werden, ein Anwendungsserver und andere Komponenten müssen jedoch durch den Anwender eingerichtet werden [Goof]. Die Java API bietet Zugriff auf die meisten Funktionen der Google Cloud Services [Gooh]. Die Konfigurationsoptionen bei Google App Engine sind jedoch beschränkt. Die unterliegenden Ressourcen lassen sich hier nicht gezielt konfigurieren. Zum Beispiel kann bei GAE nicht explizit ausgewählt werden, welche Hardwarekonfiguration eine verwendete Rechnerinstanz besitzen soll. Außerdem stehen bei den Google Cloud Services Load-Balancing und Auto-Skalierung zur Verfügung [Goob]. Hierbei lässt sich jedoch nicht steuern, wie viele Rechnerinstanzen verwendet werden sollen und ob generell skaliert werden soll. Diese Funktionen stehen nur für Compute Engine zur

Verfügung. Einen Queue Service bietet Google nicht. Google App Engine lässt sich in zeitlich begrenztem Rahmen kostenlos nutzen, jedoch steht im kostenlosen Angebot nur eine nach oben begrenzte Skalierbarkeit zur Verfügung [Good]. Für Compute Engine gibt es keine Angebote zur Gratis-Nutzung [Goog]. Angebote für Studierende gibt es nicht [Gooc].

3.3.3. Ergebnis

Aus den untersuchten Cloud Services wird Amazon Web Services als beste Lösung für diese Arbeit betrachtet, da dort sämtliche gestellten Anforderungen erfüllt werden. Insbesondere wichtig sind die umfangreichen Konfigurationsmöglichkeiten und die Java API, die Zugriff auf sämtliche Funktionen von Amazon Web Services direkt aus der Java Umgebung erlaubt, ohne auf andere Werkzeuge zurückgreifen zu müssen. Außerdem steht ein PaaS Angebot (Elastic Beanstalk) zur Verfügung, welches die Bereitstellung einer Anwendung vereinfacht und dennoch Zugriff auf die unterliegenden Ressourcen bietet.

Gegen die Verwendung von Windows Azure spricht vor allem die wenig umfangreiche Java API. Außerdem ist das Bereitstellen einer Java Anwendung im Vergleich zu den anderen Angeboten wenig komfortabel, da bei Azure zunächst mehrere Konfigurationsdateien erstellt werden müssen. Bei den Angeboten von Google und Amazon ist nur der Container mit der Anwendung nötig, um diese bereitzustellen. Google lässt bei App Engine jedoch keinen Zugriff auf die verwendeten Ressourcen und auf deren Konfigurationsmöglichkeiten zu. Dies spricht gegen die Verwendung der Google Services.

4. Entwurf

In diesem Kapitel wird der Entwurf vorgestellt, der als Grundlage für die Umsetzung der Anforderungen dient. Dabei wird vor allem auf das generelle Konzept der Instanz-Management Komponente eingegangen und es erfolgt eine abstrakte Beschreibung, wie die Funktionalitäten umgesetzt werden. Es werden keine konkreten Technologien beschrieben, die zur Umsetzung verwendet werden.

Grundlegend sollen die Funktionalitäten durch zwei verschiedene Anwendungen realisiert werden: Durch eine zentrale Instanz-Management Komponente und eine Umgebung, in der ein Automat in der Cloud betrieben werden kann. Die Entwicklung einer Umgebung für den Automaten ist nötig, da der Automat nicht autonom in der Cloud betrieben werden kann. Es soll für den Automaten also eine Anwendung entwickelt werden, die in der Cloud lauffähig ist und einen oder mehrere Automaten betreiben kann. Diese Anwendung wird als *Umgebung des Automaten* oder als *Automaten-Umgebung* bezeichnet. Als Automat wird wie bisher die Komponente bezeichnet, die für die Ausführung einer Grammatik zuständig ist. Das Verhältnis von Instanz-Management Komponente zu Automaten-Umgebung wird in Abbildung 4.1 dargestellt. Die Instanz-Management Komponente besitzt Zugriff auf beliebig viele Instanzen von Automaten-Umgebungen. Die Instanzen der Automaten-Umgebungen sind auf verschiedene Regionen verteilt, während die Instanz-Management Komponente in einer bestimmten Region lokalisiert ist. Eine Automaten-Umgebung innerhalb einer Region kann dort aber auf mehreren Rechnern betrieben werden. Zudem können innerhalb einer Automaten-Umgebung mehrere Automaten betrieben werden.

Abbildung 4.1.: Das Verhältnis der Instanz-Management Komponente zu den Automaten-Umgebungen.

Die Aufteilung in zwei verschiedene Anwendungen ist sinnvoll, da es in einer verteilten Ablaufumgebung genau eine Komponente geben soll, die Anfragen verwaltet. Zudem ist ein mehrfaches Vorkommen der Instanz-Management Komponente nicht sinnvoll, da diese zur Verteilung der Anfragen auf verschiedene Instanzen einen Überblick über sämtliche existierende Instanzen des Automaten benötigt. Bei einer Verteilung der Instanz-Management Komponente über mehrere Regionen müsste hierfür eine Synchronisierung zwischen den

verschiedenen Instanz-Management Komponenten stattfinden, damit diese aktuelles Wissen über alle existierenden Automaten-Umgebungen besitzen. Durch die Verwendung einer zentralen Komponente kann auf einen solchen Mechanismus verzichtet werden.

Eine genaue Aufteilung der Aufgabenbereiche auf die beiden verschiedenen Anwendungen erfolgt in Abschnitt 4.1. Daraufhin wird in Abschnitt 4.2 die Instanz-Management Komponente konzeptioniert. Die Kozeptionierung der Umgebung des Automaten erfolgt in Abschnitt 4.3.

4.1. Aufgabenverteilung

Die meisten der geforderten Funktionalitäten aus Kapitel 3 werden durch die Instanz-Management Komponente umgesetzt. Einige Anforderungen werden jedoch durch die Automaten-Umgebung erfüllt.

Für Anforderung A1 muss die Berechnung einer Zielregion umgesetzt werden. Dies fällt der Instanz-Management Komponente zu, da die Region berechnet werden muss, bevor die Grammatik an einen Automaten der Zielregion geschickt werden kann. Anforderung A2 gibt die Verwaltung der verschiedenen Instanzen des Automaten vor. Um dies zu realisieren, besitzt die Instanz-Management Komponente eine Übersicht über alle Automaten-Umgebungen und über die Rechner, die diese verwenden. Die Automaten-Umgebung besitzt dagegen Informationen über die Automaten selbst, die innerhalb der Umgebung betrieben werden. Anforderung A3 beschreibt eine Funktion, um den Automaten in einer verteilten Ablaufumgebung ausführbar zu machen. Dies soll von der Instanz-Management Komponente und der Umgebung des Automaten sichergestellt werden. Dabei ist es insbesondere die Aufgabe der Automaten-Umgebung, Automaten innerhalb der Umgebung zur Verarbeitung von Anfragen bereitzustellen. Laut Anforderung A4 muss ein Überblick über die in den Grammatiken verwendeten Web Services hergestellt werden. Dies ist Aufgabe der Instanz-Management Komponente, da dies zur Berechnung der Zielregion des Automaten erforderlich ist. Die Verwaltung des RRS (Anforderung A5) fällt ebenfalls in den Aufgabenbereich der Instanz-Management Komponente. Der Automat ruft das RRS lediglich auf wie einen beliebigen anderen Web Service. In A6 wird Generalität des Automaten gefordert. Diese muss durch die Umgebung des Automaten sichergestellt werden.

Die nichtfunktionalen Anforderungen werden ebenfalls den verschiedenen Komponenten zugewiesen. Die Garantie, dass eine Anfrage verarbeitet wird (A7), muss dabei von beiden Komponenten gewährleistet werden. Beide Komponenten sollen bei der Verarbeitung sicherstellen, dass eine Anfrage im Fehlerfall nicht verloren geht. In A8 wird Optimalität für die Verteilung der Instanzen gefordert, das heißt eine Minimierung der Kosten während der Ausführung einer Grammatik. Dies soll bei der Berechnung der Region durch die Instanz-Management Komponente sichergestellt werden. Die Anforderungen A9 und A10 beziehen sich auf Effizienz und Skalierbarkeit und sollen sowohl durch die Instanz-Management Komponente als auch durch die Automaten-Umgebung erfüllt werden. Die Umgebung des Automaten soll dafür zuständig sein, dem Automaten ausreichend Rechenleistung zur Verfügung zu stellen. Diese soll bei Bedarf erhöht oder verringert werden. Dies kann zum Beispiel

durch die Verwendung von mehreren Rechnern geschehen. Die Instanz-Management Komponente hingegen soll bestimmen, ob in einer Region überhaupt eine Automaten-Umgebung existiert.

4.2. Instanz-Management Komponente

Nachdem die Verteilung der Aufgaben im vorigen Abschnitt vorgestellt wurde, wird nun zunächst die Funktionalität der zentralen Instanz-Management Komponente konzipiert. Hierbei gibt es keine Sortierung nach den Anforderungen mehr; stattdessen werden ähnliche Aufgaben zusammen behandelt.

4.2.1. Anfragen Verarbeitung

Der Einstiegspunkt für die Instanz-Management Komponente soll die Entgegennahme und Verarbeitung von Anfragen darstellen. Dabei ist es insbesondere wichtig, dass eine Anfrage nicht durch einen unerwarteten Fehler verloren geht (vgl. Anforderung A7). Die Anfragen sollen deshalb in einer Warteschlange gespeichert werden. Diese soll persistent sein und keine Anfragen verlieren, auch im Falle, dass die Instanz-Management Komponente nicht verfügbar ist. Die Speicherung der Anfragen wird in Abbildung 4.2 dargestellt. Die Entgegennahme von Anfragen soll über eine Schnittstelle geschehen, die dem Nutzer zur Verfügung steht. Diese soll nur die Funktion haben, Anfragen entgegen zu nehmen und sie danach in der Warteschlange zu speichern. Bei der Verarbeitung der Anfrage wird diese zunächst aus der Warteschlange entfernt. Im Fehlerfall soll sie immer zurück in die Warteschlange gelegt werden.

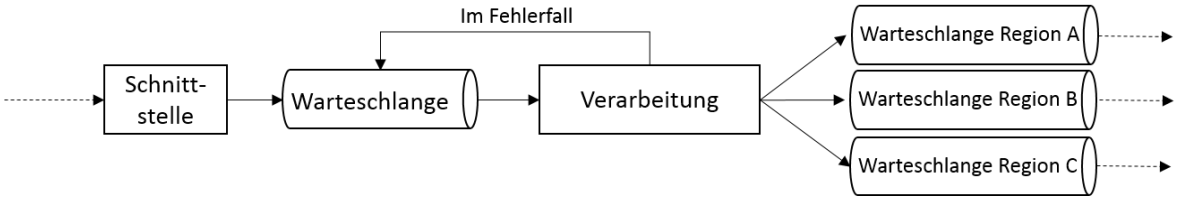

Abbildung 4.2.: Die Speicherung einer Anfrage während der Verarbeitung. Die Pfeile stellen das Senden der Anfrage von einer Komponente zur anderen dar. Gestrichelte Pfeile stehen für das Senden zu einer anderen Anwendung.

Während der Verarbeitung einer Anfrage soll die Berechnung der Region (siehe Abschnitt 4.2.2) erfolgen und eine Instanz einer Automaten-Umgebung aus der Zielregion angefordert werden (siehe Abschnitt 4.2.4). Außerdem soll die Anfrage letztendlich zu der entsprechenden Automaten-Umgebung gesendet werden. Bei letztgenanntem Schritt ist es möglich, dass die Automaten-Umgebung zu diesem Zeitpunk keine Anfragen entgegen nehmen kann. Eine Automaten-Umgebung kann zum Beispiel keine Anfragen entgegen nehmen,

wenn der Rechner, auf dem diese betrieben werden soll, noch nicht bereit ist. Die Anfrage soll in diesem Fall in eine weitere Warteschlange eingefügt und versandt werden, sobald die Automaten-Umgebung bereit ist. Dabei wird für jede Region eine eigene Warteschlange verwendet, wie in Abbildung 4.2 dargestellt. Die Automaten-Umgebung liefert selbst eine Rückmeldung darüber, wenn sie bereit ist. Der Status der Automaten-Umgebung muss also nicht abgefragt werden. In der Wartezeit sollen weitere Anfragen verarbeitet werden können. Die Anfrage wird der Automaten-Umgebung über eine Schnittstelle übergeben, die diese hierfür zur Verfügung stellt. Die Automaten-Umgebung ist für die weitere Verarbeitung der Anfrage zuständig.

4.2.2. Berechnung der Region

Der erste Schritt bei der Verarbeitung einer Anfrage stellt die Berechnung der Region dar, in der die Anfrage verarbeitet werden soll (*Zielregion*). Eine Anfrage ist hier eine Grammatik, die ausgeführt werden soll. Die Berechnung ist Aufgabe der Instanz-Management Komponente. Hierbei wird gefordert, dass die Berechnung ein optimales Ergebnis liefert. Das heißt die Grammatik soll bei der Ausführung minimale Kosten verursachen (vgl. Anforderung A8).

Faktoren, die hier bei der Berechnung berücksichtigt werden sollen, sind die Entfernungen zu den verwendeten Web Services und die Häufigkeit, mit der diese während der Ausführung einer Grammatik verwendet werden. Dies zielt vor allem auf eine Minimierung der Latenz bei der Kommunikation mit den Web Services ab. Hierbei wird die Distanz zwischen den Regionen als Metrik verwendet. Die Distanzwerte sind dabei statisch. Die Bewertung einer Region ergibt sich durch eine Analyse der in der Grammatik verwendeten Web Services in Kombination mit den Distanzen. Bei dem Verfahren werden Regionen bevorzugt, die möglichst viele Web Services enthalten. Falls eine Region alle Web Services enthält, wird diese immer zur Zielregion erklärt, falls es nicht noch andere Regionen gibt, die dieselbe Eigenschaft erfüllen. Ansonsten wird die Region als Zielregion gewählt, die die minimale Latenz bei der Kommunikation mit den Web Services anderer Regionen bietet. Der Algorithmus, der verwendet werden soll, um die Zielregion zu berechnen, wird in Alg. 4.1 als Pseudocode dargestellt.

Zunächst wird die in der Anfrage enthaltene Grammatik untersucht. Dies geschieht in den Zeilen 2 - 3 des Algorithmus. Hierbei werden die Web Services ermittelt, die bei der Ausführung der Grammatik benötigt werden. Außerdem wird mit Hilfe eines Web Service Indexes (siehe Abbschnitt 4.2.3) untersucht, welche Web Services von diesen verfügbar sind. Dabei werden nur Web Services berücksichtigt, die sich in derselben Ablaufumgebung befinden, wie die Instanz-Management Komponente und die Automaten-Umgebung (*Closed World*, vgl. [GL12]). Externe Web Services werden bei der Berechnung der Region ignoriert, da über deren Distanz zu den verwendeten Regionen keine zuverlässige Aussage getroffen werden kann.

Für den Fall, dass keiner der verwendeten Web Services in der Closed World verfügbar ist, kann die Berechnung abgebrochen und stattdessen eine zufällige Region verwendet werden. Dies geschieht in den Zeilen 4 - 6. Danach wird in Zeile 9 - 22 für jede Region,

Algorithmus 4.1 Algorithmus zur Berechnung der Zielregion.

```
    procedure ComputeRegion(grammar)
        requiredWS ← getWs(grammar)
        availableWs ← getAvailableWs(requiredWs)
        if availableWs.isEmpty( ) then
 5:         return randomRegion
        end if
        finalRegion ← null
        bestDistance ← ∞
        for all r ∈ Regions do
10:         ranking ← 0
            wsNotInR ← getUnavailbleWs(r, requiredWs)
            for all ws ∈ wsNotInR do
                n ← getNumberOfAccesses(ws, grammar)
                closestRegion ← getClosestRegion(ws)
15:             distance ← getDistance(closestRegion)
                ranking ← ranking + n ∗ distance
            end for
            if ranking ≤ bestDistance then
                bestDistance ← ranking
20:             finalRegion ← r
            end if
        end for
        return finalRegion
    end procedure
```

die in der verteilten Ablaufumgebung (Amazon Web Services) zur Verfügung steht, eine Bewertung erstellt. Die Bewertung ergibt sich dabei aus der Distanz zu den Web Services, die sich nicht in der Region befinden und der Häufigkeit der Verwendung dieser Web Services. Es wird folglich für jeden Web Service, der sich *nicht* in der zu bewertenden Region befindet, die Distanz zwischen der zu bewertenden Region und der Region, die den Web Service beinhaltet und die geringste Distanz zur bewertenden Region besitzt, mit der Anzahl der Web Service Aufrufe multipliziert und auf den bisherigen Bewertungs-Wert addiert. Hierfür wird in Zeile 11 zunächst eine Liste der Web Services erstellt, die nicht in der zu bewertenden Region verfügbar sind. Danach wird in den Zeilen 13 - 16 von Algorithmus 4.1 schrittweise die Bewertung berechnet. Beispielsweise soll die Bewertung für die Region Amerika erstellt werden. Es werden die Web Services A und B benötigt. Web Service A ist in der Region Amerika verfügbar und erhöht die Bewertung daher nicht. Web Service B ist nicht in Amerika verfügbar. In Zeile 13 wird die Anzahl der Zugriffe auf B ermittelt. Hierbei wird als Ergebnis beispielsweise 5 geliefert. In Zeile 14 wird Europa als nächste Region ermittelt, die Web Service B enthält. Die Distanz zwischen Europa und Amerika beträgt 1000. Folglich wird in Zeile 16 die Bewertung auf 5000 erhöht. Die Anzahl der Web Service Aufrufe kann dabei durch eine Analyse der Produktionsregeln der Grammatik bestimmt werden,

indem gezählt wird, wie oft das Nicht-Terminal eines Web Services auf der rechten Seite der Produktionsregeln vorkommt. Hierbei handelt es sich jedoch nur um eine Schätzung, da Regeln mehrfach ausgeführt werden können. Wie oft eine Regel tatsächlich verwendet wird, kann erst während der Ausführung des Automaten bestimmt werden. Wenn die Berechnung für jeden nicht verfügbaren Web Service durchgeführt wurde, wird die Bewertung der Region mit der bisher günstigsten Region verglichen. Ist die Bewertung niedriger, wird die eben bewertete Region zur neuen Zielregion erklärt, wie es in den Zeilen 18 - 21 dargestellt wird. Wenn beispielsweise die bisherige Zielregion Asien mit einer Bewertung von 10000 war, dann wird nun Amerika mit der Bewertung von 5000 zur neuen Zielregion erklärt.

4.2.3. Web Service Index

Bei der Berechnung der Region ist es erforderlich, die Ablaufumgebung nach bekannten Web Services zu durchsuchen, die für die Ausführung einer Service-Komposition verwendet werden. Dabei soll ein Index dieser Web Services erstellt werden.

Der Web Service Index soll zu dem Zeitpunkt, an dem die Berechnung der Region stattfindet, Informationen über alle bekannten und verfügbaren Web Services besitzen. Bei den Informationen, die gespeichert werden sollen, handelt es sich um den Namen und die Operationen des Web Services, die Region des Web Services und die URL, die den Zugriff auf den Web Service ermöglicht. Um diese Informationen zusammenzustellen, muss die Ablaufumgebung nach diesen Web Services durchsucht werden. Die Erstellung des Indexes soll zudem laufend geschehen und nicht erst beginnen, wenn die Informationen zu den Web Services bei der Berechnung der Region angefordert werden. Dabei wird nach allen Web Services gesucht, die während des Betriebs der Instanz-Management Komponente bereits in einer Grammatik verwendet wurden. Ein Web Service wird dabei in die Suche mit aufgenommen, wenn er zum ersten Mal in einer Grammatik auftaucht.

In den Grammatiken werden Web Services und deren Operationen eindeutig über den Namen identifiziert. Bei der Erstellung des Indexes wird gleich vorgegangen. Wenn nach einem Web Service aus einer Grammatik gesucht wird, werden folglich zwei Eigenschaften überprüft: Zum einen, ob der Web Service denselben Namen besitzt wie in der Grammatik und zum anderen, ob dessen Operationen gleich benannt sind wie in der Grammatik vorgegeben. Es wird für die Ablaufumgebung der Instanz-Management Komponente also angenommen, dass die Kombination aus Web Service Namen und dem Namen einer Operation des Web Services eindeutig ist.

4.2.4. Verwaltung der Instanzen des Automaten

Wenn die Berechnung einer Region abgeschlossen ist, muss die Instanz einer Automaten-Umgebung zur Verfügung gestellt werden. Die Bereitstellung und Verwaltung von Instanzen der Automaten-Umgebung ist eine zentrale Funktion der Instanz-Management Komponente. Dabei soll sie stets Zugriff auf sämtliche existierende Instanzen der Automaten-Umgebung

besitzen. Es müssen folglich für jede Automaten-Umgebung bestimmte Verwaltungsinformationen gespeichert werden. Darunter fallen Informationen, die den Zugriff auf die Automaten-Umgebung ermöglichen, wie die URL, und Status-Informationen der Automaten-Umgebung, zum Beispiel ob diese Anfragen entgegen nehmen kann oder nicht. Die Verwaltungsinformationen müssen angelegt werden, wenn eine neue Automaten-Umgebung erzeugt wird und können zusammen mit der Automaten-Umgebung gelöscht werden. Eine Automaten-Umgebung kann gelöscht werden, wenn die Verarbeitung der Grammatiken in dieser Region abgeschlossen ist. Dies wird der Instanz-Management Komponente durch eine Nachricht der Automaten-Umgebung signalisiert. Für Nachrichten der Automaten-Umgebung soll eine extra Schnittstelle bereit gestellt werden. Wenn die Instanz-Management Komponente eine Lösch-Nachricht erhält, soll sie alle Cloud-Ressourcen der Automaten-Umgebung freigeben. Dazu gehören zum Beispiel die Rechner, auf der die Automaten-Umgebung betrieben wird.

In einer Region soll immer maximal eine Instanz einer Automaten-Umgebung erstellt werden. Dementsprechend existieren für eine Region auch die Verwaltungsinformationen nur einmal, auch wenn die dortige Automaten-Umgebung auf mehreren Rechnern betrieben wird. Wenn ein Automat aus einer bestimmten Region angefordert wird, werden die Verwaltungsinformationen der Automaten-Umgebung dieser Region zurückgeliefert. Die Verwaltungsinformationen sollen die nötigen Informationen bieten, um zum Beispiel eine Anfrage zu der Automaten-Umgebung zu senden. Wenn ein Automat in einer Region angefordert wird, in der noch keine Automaten-Umgebung existiert, wird diese zu diesem Zeitpunkt erzeugt. Wenn die Automaten-Umgebung in einer Region auf mehreren Rechnern betrieben wird, soll sich ein Load-Balancer darum kümmern, Anfragen auf diese zu verteilen. In den Verwaltungsinformationen der Automaten-Umgebung werden dabei die Daten gespeichert, die für den Zugriff auf den Load-Balancer erforderlich sind. Bei der Message Correlation hingegen sollen die Messages nicht an den Load-Balancer geschickt werden. Die Messages müssen explizit an den Rechner geschickt werden, auf dem eine Anfrage verarbeitet wird. Hierfür müssen auch die Zugriffsinformationen zu den Rechnern gespeichert werden, auf denen die Automaten-Umgebung betrieben wird.

Es ist möglich, dass die Instanz-Management Komponente fehlerhaft terminiert und dabei nicht die existierenden Automaten-Umgebungen löscht. In diesem Fall sollen die Automaten-Umgebungen bei einem Neustart der Instanz-Management Komponente verwendet werden können, obwohl diese keine Verwaltungsinformationen über die Automaten-Umgebungen besitzt. Um diese Automaten-Umgebungen verwenden zu können, soll bei der Erstellung einer Automaten-Umgebung in einer Region überprüft werden, ob in dieser Region bereits eine Automaten-Umgebung existiert.

4.2.5. Message Correlation

Nachdem eine Anfrage bei der Automaten-Umgebung eingegangen ist, ist es möglich, dass der Client Messages schickt, die sich auf diese Anfrage beziehen. Diese schickt der Client an die Schnittstelle der Instanz-Management Komponente und es ist deren Aufgabe, diese genau

an die Automaten-Umgebung weiterzuleiten, die diese Anfrage verarbeitet (Anforderung A3).

Hierfür soll der Client eine ID erhalten, wenn er eine Anfrage in Form einer Grammatik sendet. Diese identifiziert die Anfrage eindeutig. Unter Angabe dieser ID soll es möglich sein, Messages zu dem Rechner zu senden, auf dem die Grammatik bearbeitet wird. Da es vorkommen kann, dass eine Anfrage noch nicht bearbeitet wurde, wenn eine Message zu dieser Anfrage eingeht, muss es eine Möglichkeit geben, die Message solange in einer Warteschlange zu speichern. Sobald bekannt wird, dass eine Anfrage von einem bestimmten Rechner bearbeitet wird, sollen die betroffenen Messages in der Warteschlange an diesen Rechner gesendet werden. Der Rechner, auf dem die Anfrage verarbeitet wird, kann erst bestimmt werden, wenn die Anfrage dort eingegangen ist. Sobald dies geschieht, sendet die Automaten-Umgebung der Instanz-Management Komponente eine Nachricht, in der angegeben ist, auf welchem Rechner die Anfrage verarbeitet wird. Eine ID mit dem zugehörigen Ziel soll aus dem System gelöscht werden, sobald die Anfrage von einem Automaten ausgeführt wurde.

4.2.6. Reference Resolution System - Instanzverwaltung

Eine weitere Funktion, die durch die Instanz-Management Komponente erfüllt werden soll, ist die Verwaltung des Reference Resolution System (Anforderung A5). Die Verwaltung der RRS Instanzen wird in ähnlicher Weise wie die Verwaltung der Instanzen der Automaten-Umgebung konzipiert.

Auch hier soll die Instanz-Management Komponente Verwaltungsinformation der RRS Instanzen speichern. Dabei soll die Instanz-Management Komponente Informationen über alle Instanzen des RRS besitzen. Die Instanz-Management Komponente soll außerdem dafür verantwortlich sein, neue Instanzen des RRS zu erzeugen oder nicht mehr benötigte Instanzen zu löschen. In einer Region soll eine RRS Instanz existieren, wenn dort Automaten betrieben werden, das heißt wenn dort eine Automaten-Umgebung vorhanden ist. Hierbei ist für jede Region genau eine Instanz des RRS vorgesehen. Dementsprechend soll in einer Region eine neue Instanz des RRS erzeugt werden, wenn dort eine Automaten-Umgebung erzeugt wird. Die Instanz des RRS soll gelöscht werden, wenn in der Region keine Automaten mehr existieren, das heißt wenn die Automaten-Umgebung gelöscht wird. Im Falle eines Absturzes der Instanz-Management Komponente, soll ein bereits vorhandenes RRS in einer Region wiederverwendet werden können, wie bei der Verwaltung der Automaten-Umgebungen.

Da es sich bei dem RRS um eine externe Komponente handelt, kann nicht beurteilt werden, ob das RRS auch beim Betrieb auf mehreren Rechnern korrekt funktioniert. Hierbei müsste sichergestellt werden, dass es zu keinen Inkonsistenzen in der Datenbank des RRS kommt. Daher ist ein Betrieb des RRS auf mehreren Rechnern nicht vorgesehen.

4.3. Die Umgebung des Automaten

Im folgenden Abschnitt wird die Umgebung des Automaten konzipiert. Diese dient dazu, den Betrieb von Automaten in der Cloud zu ermöglichen. Dabei soll die Umgebung des Automaten die Anwendung in der Cloud darstellen. Ein Überblick der Automaten-Umgebung wird in Abbildung 4.3 gezeigt. Die Automaten-Umgebung kann über eine Schnittstelle Anfragen und Messages der Instanz-Management Komponente entgegen nehmen. In Abb. 4.3 wird die Schnittstelle gestrichelt dargestellt. Von der Schnittstelle aus werden Anfragen und Messages zu Warteschlangen weitergeleitet. Innerhalb der Automaten-Umgebung können mehrere Automaten betrieben werden, die Anfragen verarbeiten. Die Anfragen und Nachrichten aus den Warteschlangen werden den Automaten zugewiesen. Die Automaten benötigen bei der Verarbeitung einer Anfrage Zugriff auf eine Service Invocation Komponente. Die verschiedenen Automaten werden dabei wie in Abbildung 4.3 dargestellt durch die Automaten-Umgebung verwaltet.

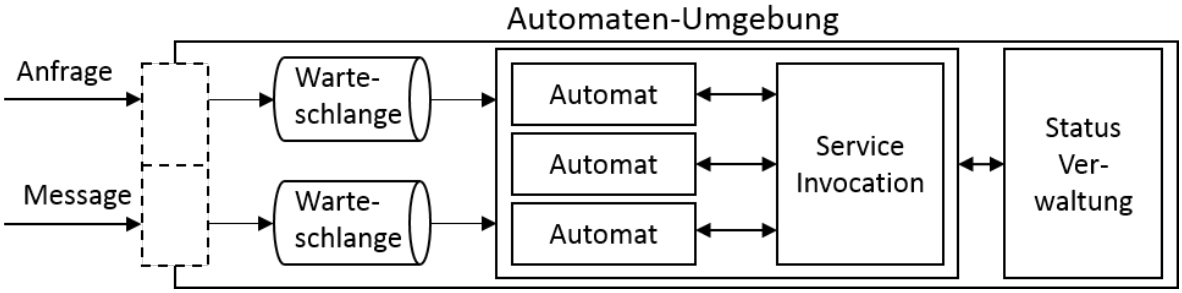

Abbildung 4.3.: Die Umgebung des Automaten im Überblick. Die gestrichelte Komponente stellt die Schnittstelle zur Automaten-Umgebung dar.

4.3.1. Schnittstelle der Automaten-Umgebung

Um Anfragen entgegen nehmen zu können, muss die Automaten-Umgebung eine Schnittstelle besitzen. Diese wird in Abb. 4.3 durch die gestrichelten Linien dargestellt. Sie dient der Instanz-Management Komponente dazu, der Automaten-Umgebung Anfragen in Form von Grammatiken zu übergeben. Diese soll gleichzeitig auch dazu verwendet werden, um Messages entgegen zu nehmen (Message Correlation).

Die Schnittstelle soll keine Funktion zur Verarbeitung von Anfragen umsetzen. Sie soll diese lediglich Anfragen entgegen nehmen und in einer persistenten Warteschlange speichern, damit diese von anderen Bestandteilen der Umgebung abgerufen werden können. Für Messages gilt dasselbe (siehe Abbildung 4.3.

4.3.2. Integration des Automaten in die Umgebung

Zur Verarbeitung von Anfragen muss der Automat in die Umgebung integriert werden. Dabei soll ein gleichzeitiger Betrieb von mehreren Automaten möglich sein, wie in Abbildung 4.3 dargestellt. Ein Automat soll durch die Übergabe einer Grammatik gestartet werden können. Dies soll mit beliebig vielen verschiedenen Grammatiken möglich sein. Eventuell müssen dem Automaten während der Ausführung Messages zugeführt werden.

Die Anfragen sollen aus der Warteschlange bezogen werden, in der die Anfragen über das Interface abgelegt werden. Messages sollen zwischengespeichert werden, bis sie benötigt werden. Hierzu dient eine Warteschlange, wie in Abbildung 4.3 dargestellt. Zur Zuordnung der Messages zu einer Anfrage wird dieselbe ID verwendet, wie bei der Instanz-Management Komponente. Die benötigten Funktionen zur Zuordnung der Message sollen dabei durch die Umgebung des Automaten zur Verfügung gestellt werden. Sobald die Ausführung einer Anfrage beendet ist, soll außerdem die Instanz-Management Komponente über das Interface der Instanz-Management Komponente benachrichtigt werden. Dies ist nötig, damit diese die ID der Anfrage aus ihrem Speicher entfernen kann. Eine Zustellung von Messages bezüglich dieser ID ist zu diesem Zeitpunkt nicht mehr nötig, da die Anfrage vollständig ausgeführt wurde.

Für den Betrieb eines Automaten ist zudem eine zusätzliche Service Invocation Komponente erforderlich (siehe 2.2). Diese soll dem Automaten in derselben Umgebung bereit gestellt werden.

4.3.3. Statusverwaltung

Für die Verwaltung der Umgebung und die Interaktion mit der Instanz-Management Komponente muss die Automaten-Umgebung gewisse Statusinformationen speichern. Dazu gehört zum Beispiel die URL der Instanz-Management Komponente, die zur Kommunikation benötigt wird. Außerdem soll hierbei der Zustand aller laufenden Automaten gespeichert werden. Der Statusverwaltung der Automaten-Umgebung soll stets bekannt sein, wie viele Automaten gerade betrieben werden und welche Anfrage diese verarbeiten. Dieser Zusammenhang wird auch in Abbildung 4.3 dargestellt. Für den Fall, dass keine Anfragen bearbeitet werden, soll veranlasst werden, dass die Automaten-Umgebung entfernt wird, um Ressourcen zu sparen (vgl. Anforderung A10). Hierfür soll die Automaten-Umgebung eine Nachricht an die entsprechende Schnittstelle der Instanz-Management Komponente senden.

Außerdem soll über die Statusverwaltung der Automaten-Umgebung bestimmt werden, wann die Automaten-Umgebung in der Lage ist, Anfragen entgegen zu nehmen und wann nicht. Bevor eine Automaten-Umgebung bereit ist, müssen zum Beispiel die Warteschlangen bereitgestellt werden. Der Bereitschaftsstatus soll mit der Instanz-Management Komponente kommuniziert werden.

4.3.4. Skalierung

Die Automaten-Umgebung hat die Aufgabe, die verwendeten Ressourcen effizient zu nutzen. Außerdem sollen die zugrunde liegenden Ressourcen mit der Anzahl der Anfragen skalieren. Um die bereits verwendeten Ressourcen optimal auszunutzen, sollen mehrere Automaten gleichzeitig betrieben werden können. Außerdem soll erkannt werden, wenn die Ressourcen einer Automaten-Umgebung nicht mehr ausreichen, um alle Anfragen zu verarbeiten. Hierzu soll die Verwendung der genutzten Ressourcen überwacht werden. Wenn diese ausgelastet sind, sollen zusätzliche Rechner für die Automaten-Umgebung verwendet werden (*horizontale Skalierung*). Für den Fall, dass sich die Anzahl der Anfragen reduziert, sollen entsprechend weniger Rechner verwendet werden. Wenn zusätzliche Rechner verwendet oder bestehende freigegeben werden, muss dies der Instanz-Management Komponente mitgeteilt werden. Außerdem soll sichergestellt werden, dass verschiedene Rechner gleichmäßig ausgelastet werden, das heißt die Anfragen müssen ausgewogen auf diese verteilt werden.

5. Implementierung

Nachdem im Entwurf die konzeptionelle Umsetzung der Anforderungen allgemein erklärt wurde, befasst sich dieses Kapitel mit der Implementierung der im Entwurf beschriebenen Funktionalitäten. Hierfür werden in Abschnitt 5.1 zunächst die verwendeten Funktionen von Amazon Web Services erklärt. Daraufhin wird in Abschnitt 5.2 die Architektur der einzelnen Komponenten und des gesamten Systems erläutert. Zuletzt wird in den Abschnitten 5.3 und 5.4 auf ausgewählte Implementierungsdetails der Instanz-Management Komponente und der Automaten-Umgebung eingegangen.

5.1. Verwendete Amazon Web Services

Zur Implementierung der Instanz-Management Komponente werden verschiedene Cloud Angebote von Amazon Web Services [Amab] verwendet. Die Entscheidung für die AWS wird in Abschnitt 3.3 begründet. Im Folgenden werden die Web Services erklärt, die in der weiteren Implementierung verwendet werden, um das Verständnis in den darauf folgenden Abschnitten zu erleichtern.

5.1.1. Elastic Compute Cloud

Bei *Elastic Compute Cloud* (EC2) handelt es sich um einen Web Service, der Rechenkapazität aus der AWS Cloud zur Verfügung stellt [Amam]. Es wird damit ermöglicht, virtuelle Rechner zu mieten. Diese werden als EC2-Instanzen bezeichnet. Dabei kann festgelegt werden, wie viel Rechenleistung, Speicher und Netzwerkleistung einer EC2-Instanz zur Verfügung gestellt werden soll. Zudem können Sicherheitsbestimmungen, wie externe Zugriffsrechte auf die Instanz oder auch das gewünschte Betriebssystem, festgelegt werden.

5.1.2. CloudWatch

CloudWatch kann dazu verwendet werden, um die Leistungsdaten von verschiedenen Amazon Web Services zu überwachen [Amal]. Hierbei stehen für verschiedene Services Metriken zur Verfügung, nach denen deren Leistungsfähigkeit beurteilt wird. Dabei können Alarme definiert werden, die ausgelöst werden, sobald selbst definierte Schwellwerte passiert werden. Wenn eine EC2-Instanz überwacht wird, kann beispielsweise die CPU-Last oder die Anzahl der Festplatten Zugriffe überwacht werden [Amak]. In diesem Beispiel könnte nun festgelegt

werden, dass ein Alarm ausgelöst wird, sobald die durchschnittliche CPU-Auslastung der Instanz über einen Zeitraum von fünf Minuten über 70 Prozent liegt.

5.1.3. Auto Scaling

Bei *Auto Scaling* handelt es sich um einen Web Service, der dazu dient, automatisch EC2-Instanzen zu erzeugen und zu löschen [Amao]. Dies geschieht abhängig von einer Metrik, die selbst bestimmt werden kann. Zum Beispiel kann festgelegt werden, dass ab einer bestimmten CPU-Last einer EC2-Instanz automatisch eine neue EC2-Instanz erzeugt wird oder dass eine EC2-Instanz gelöscht wird, sobald eine definierte Schwelle unterschritten wird. Zudem kann bestimmt werden, wie oft die EC2-Instanzen zum Beispiel auf CPU Last überprüft werden und es können generelle Grenzen über die Anzahl der EC2-Instanzen definiert werden. Die Überwachung der Leistungsdaten erfolgt durch CloudWatch. Das heißt, sobald CloudWatch einen Alarm auslöst, erzeugt Auto Scaling eine neue EC2-Instanz. Außerdem überprüft Auto Scaling regelmäßig die Gesundheit der zugeordneten EC2-Instanzen und erzeugt automatisch neue, falls eine EC2-Instanz nicht funktionsfähig ist. Die EC2-Instanzen werden in sogenannten Auto Scaling Gruppen verwaltet. Wenn über Auto Scaling eine neue EC2-Instanz erzeugt wird, gehört diese zur selben Auto Scaling Gruppe wie die EC2-Instanz, die durch die neue EC2-Instanz unterstützt werden soll.

5.1.4. Elastic Load Balancing

Elastic Load Balancing ist ein Service, der Load-Balancing für EC2-Instanzen implementiert [Amae]. Hierbei werden die Anfragen auf die EC2-Instanzen verteilt, die dem Load-Balancer zugeteilt sind. Der Load-Balancer achtet dabei auch darauf, dass er Anfragen nur an funktionsfähige EC2-Instanzen weiterleitet. Die Rechenkapazitäten des Load-Balancers werden automatisch angepasst.Bei vielen eingehenden Anfragen werden diese also erhöht, beziehungsweise bei wenigen Anfragen gesenkt.

5.1.5. Elastic Beanstalk

Bei Elastic Beanstalk handelt es sich um einen Web Service, der die Bereitstellung von Anwendungen in der Cloud soweit wie möglich automatisieren soll [Amap]. Dabei ist es nötig, einen Container, der die Anwendung enthält, hochzuladen und Elastic Beanstalk stellt die nötige Infrastruktur für die Anwendung bereit. Die Infrastruktur einer Elastic Beanstalk Anwendung wird in Abbildung 5.1 dargestellt. Eine Anwendung wird dabei auf einer oder mehreren EC2-Instanzen betrieben. Die EC2-Instanzen werden in der Abbildung durch orange Rechtecke visualisiert. Diese werden in einer Auto-Scaling Gruppe verwaltet, wie in der Abbildung durch die blau gestrichelte Umrandung dargestellt. Der Zustand der EC2-Instanzen wird durch CloudWatch überwacht. Überschreitet eine der zu überwachenden Ressourcen einen bestimmten Wert, wird ein Alarm ausgelöst. Dieser wird von der Auto-Scaling Gruppe dazu verwendet, um neue EC2-Instanzen zu erzeugen oder die Anzahl der

Instanzen zu reduzieren. Die Anfragen an die Anwendung können von einem beliebigen Ort außerhalb oder innerhalb der Cloud gesendet werden. Der Client, der Anfragen sendet, wird in der Abbildung grau dargestellt. Gewöhnlich werden die Anfragen dabei an einen Load-Balancer gesendet, der diese auf die EC2-Instanzen der Auto-Scaling Gruppe verteilt. Der Load-Balancer ist in der Abbildung blau markiert. In manchen Fällen ist es jedoch sinnvoll eine EC2-Instanz direkt anzusprechen. In diesem Fall kann die Anfrage auch gesendet werden, ohne den Load-Balancer zu verwenden. Obwohl Elastic Beanstalk die Bereitstellung der Infrastruktur übernimmt, ist es möglich, auf die einzelnen Komponenten, wie die EC2-Instanzen oder den Load-Balancer, zuzugreifen und diese individuell zu konfigurieren.

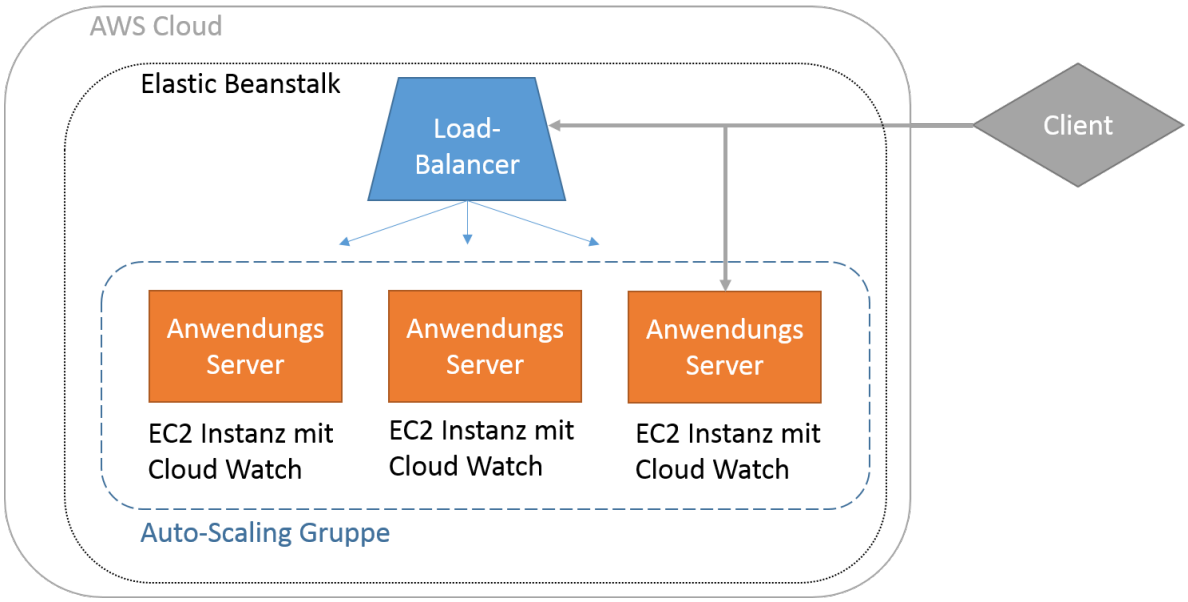

Abbildung 5.1.: Die Elastic Beanstalk Infrastruktur im Überblick.

5.1.6. Simple Storage Service

Bei *Simple Storage Service* (S3) handelt es sich um eine Cloud Lösung zur Speicherung von Daten [Amaf]. Die Organisation der Daten erfolgt dabei über sogenannte *Buckets*. Ein Bucket dient als Wurzelverzeichnis, in dem Dateien abgelegt und Ordner erstellt werden können. Für den Bucket selbst lassen sich zum Beispiel Zugriffsbestimmungen für alle untergeordnete Dateien oder ein Lebenszyklus für die Dateien festlegen.

5.1.7. Simple Queue Service

Simple Queue Service (SQS) stellt eine Nachrichten Warteschlange zur Verfügung [Aman]. In dieser können reine Textnachrichten gespeichert werden, die maximal eine Größe von 256 KB besitzen dürfen. Die Warteschlange erlaubt paralleles Lesen und Schreiben, gibt jedoch keine

Garantie über die Reihenfolge, in der die Nachrichten bezogen werden. Die Warteschlange lässt sich als Puffer und zur Kommunikation zwischen verschiedenen Anwendungen einsetzen.

5.1.8. Regionen

Bei dem Start eines Services wird die Auswahl einer Region erlaubt. Bei den verfügbaren Regionen handelt es sich um Gebiete, in denen sich ein Amazon Rechenzentrum befindet. Wenn ein Service einer bestimmten Region zugeordnet wird, dann wird er in dem dortigen Rechenzentrum betrieben. Für Zugriffe, die aus der räumlichen Nähe der Region kommen, ergeben sich so zum Beispiel automatisch Vorteile wie bessere Latenzen. Es macht Sinn, die Region zu verwenden, aus deren Nähe die meisten Zugriffe zu erwarten sind. Zudem profitieren auch unterschiedliche Amazon Web Services davon, wenn sie in derselben Region betrieben werden. Zum Beispiel ist der Datenverkehr zwischen EC2-Instanzen, die in derselben Region liegen, kostenlos [Amai].

Die verfügbaren Regionen sind [Amaj]:

- Asien-Pazifik
  - Singapur
  - Sydney
  - Tokyo
- Europa
  - Irland
- Nord-Amerika
  - Kalifornien
  - Oregon
  - Virginia
- Süd-Amerika
  - Sao Paulo

Zudem existiert die Region *GovCloud*. Diese kann jedoch ausschließlich durch die Regierung der Vereinigten Staaten verwendet werden und wird in dieser Arbeit daher ignoriert.

5.2. Architektur

Die Grundlagen für die Implementierung sind nun bekannt. In diesem Abschnitt wird darauf aufbauend die Architektur des Gesamtsystems sowie der einzelnen Komponenten im Detail erklärt.

5.2.1. Schichtenarchitektur

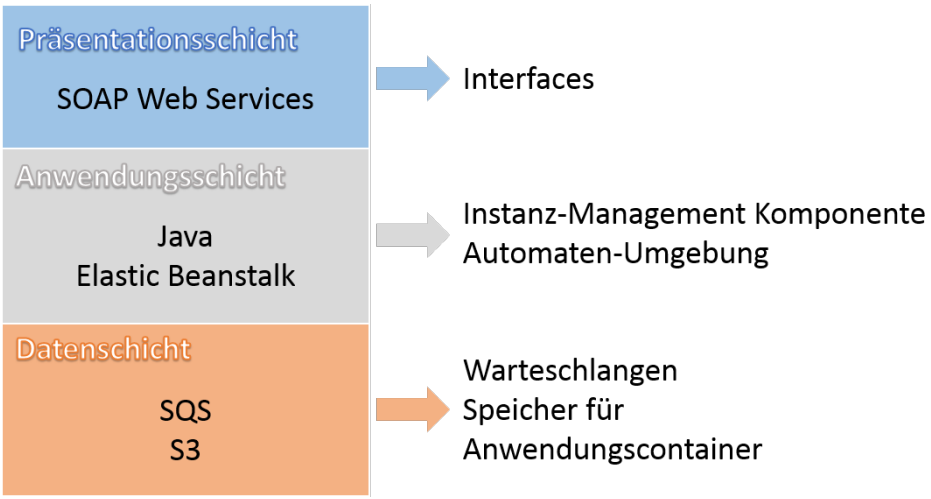

Abbildung 5.2.: Die entwickelten Anwendungen im Schichtenmodell.

Der generelle Aufbau der Instanz-Management Komponente und der Automaten-Umgebung lässt sich in eine 3-Schichten Architektur einordnen, wie in Abbildung 5.2 dargestellt. Keine der Anwendungen verfügt über eine grafische Benutzeroberfläche. Sowohl die Instanz-Management Komponente als auch die Automaten-Umgebung kommunizieren ausschließlich über verschiedene SOAP Web Services, die hier als die Präsentationsschicht bezeichnet werden. Die Anwendung kann somit von einem beliebig konzipierten Client verwendet werden, solange dieser SOAP unterstützt. Die Anwendungsschicht beinhaltet die Komponenten der Instanz-Management Komponente und der Automaten-Umgebung. Die Anwendungen sind dabei in Java implementiert und werden durch Elastic Beanstalk bereitgestellt. Die Infrastruktur einer Elastic Beanstalk Anwendung wird in Abschnitt 5.1.5 erklärt. Die EC2-Instanzen sind dabei konfiguriert, wie in Abbildung 5.3 dargestellt. In dieser Abbildung werden die Ressourcen gezeigt, auf denen die Anwendungen ausgeführt werden. Dabei gibt es mehrere Ebenen, die aufeinander aufsetzen. Diese werden in der Abbildung durch die blauen Rechtecke dargestellt. Die Pfeile symbolisieren die Hierarchie zwischen den Ebenen. Auf der untersten Ebene befinden sich die EC2-Instanzen. Eine EC2-Instanz wird als virtueller Server verwendet, auf dem das Betriebssystem ausgeführt wird. Das Betriebssystem wird in der Abbildung als Ebene über den EC2-Instanzen dargestellt. Es basiert auf einem 64 Bit Linux. Bei dem Linux OS handelt es sich um ein von Amazon zur Verfügung

gestelltes Linux Image, welches für den Betrieb auf EC2-Instanzen optimiert ist [Amaa]. Das Linux OS wird zum Betrieb eines Apache Tomcat 7 Webserver verwendet. Dieser dient zur Ausführung der Instanz-Management Komponente und der Automaten-Umgebung. Bei beiden Anwendungen handelt es sich um Java Web Anwendungen. Sie werden auf der obersten Ebene von Abbildung 5.3 dargestellt.

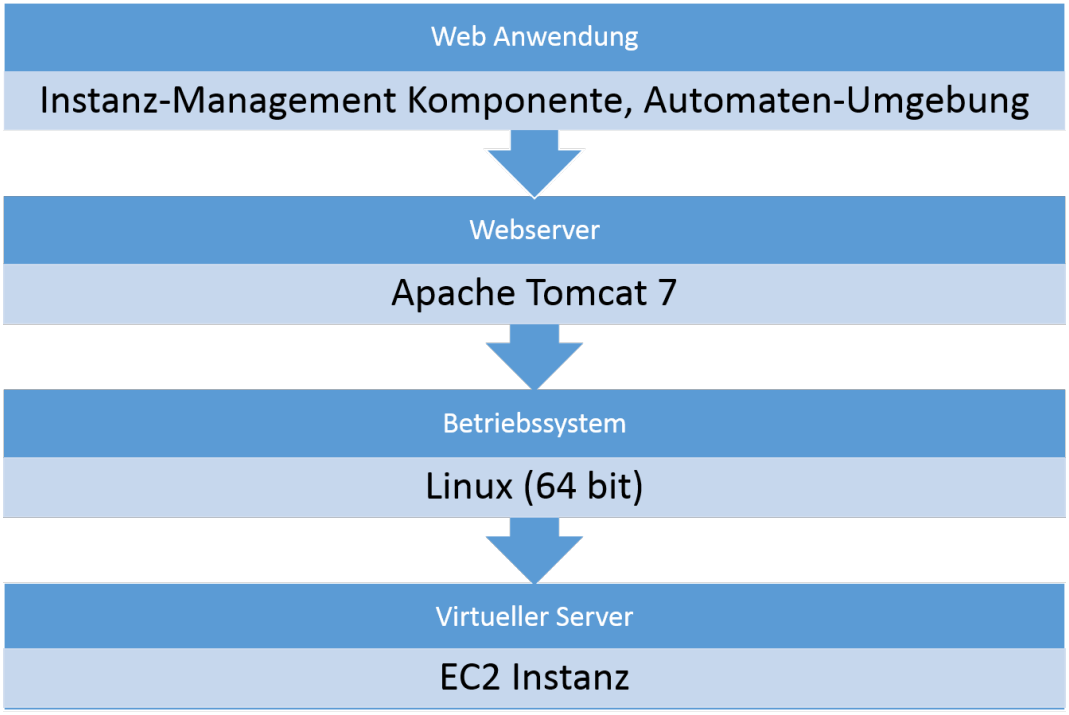

Abbildung 5.3.: Die unterliegenden Ressourcen der Anwendungen.

Für die Speicherung von Daten werden ausschließlich Amazon Web Services verwendet. Hierbei wird eine Warteschleife aus SQS verwendet, um Anfragen persistent zu speichern und weiterzugeben. Außerdem wird S3 benötigt, um zum Beispiel Anwendungscontainer zu speichern. Eine weitere persistente Speicherung von Informationen ist nicht erforderlich. Eine Datenbank oder Ähnliches wird daher nicht benötigt.

5.2.2. Aufbau und Zusammenspiel der Anwendungen

Der Aufbau der einzelnen Anwendungen wird an dieser Stelle erläutert. Hierbei soll lediglich ein Überblick über die Bestandteile der Anwendungen und deren Kommunikation geschaffen werden, ohne dabei auf die Funktionen der einzelnen Bestandteile einzugehen, da diese in den Kapiteln 5.3 und 5.4 beschrieben sind.

Die Anwendungen und ihre Komponenten werden in Abbildung 5.4 dargestellt. Die Funktionalität der Instanz-Management Komponente wird auf sechs Komponenten aufgeteilt, die in

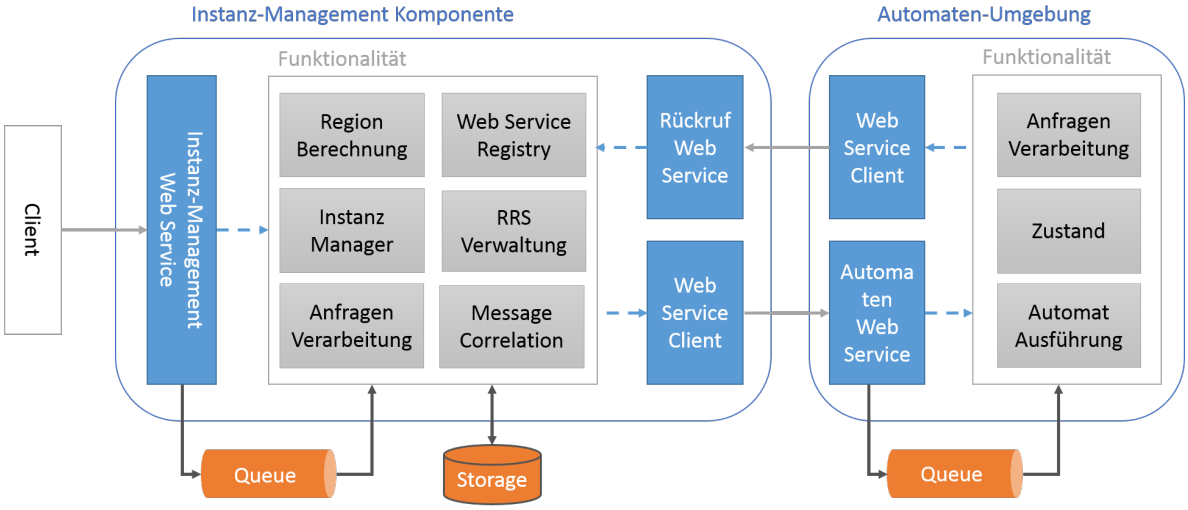

Abbildung 5.4.: Der Aufbau der einzelnen Komponenten und ihre Beziehungen. Bei den grauen Pfeilen handelt es sich um den Aufruf eines Web Services, wobei der Pfeil in Richtung des verwendeten Web Services zeigt. Schwarze Pfeile stehen für den Austausch von Daten. Blau gestrichelte Pfeile symbolisieren Funktionsaufrufe zwischen den Teilen der Anwendung. Die Färbung der Komponenten orientiert sich an der Grafik zur Schichtenarchitektur(Abb. 5.2). Blau gefärbte Rechtecke sind Komponenten der Präsentationsschicht. Graue Rechtecke sind Komponenten der Anwendungsschicht. Orange Elemente stehen für die Datenschicht.

Abb. 5.4 grau markiert sind. Hier wird insbesondere auf die Komponente *Instanz Manager* hingewiesen. Diese ist für die Verwaltung der Instanzen der Automaten-Umgebung zuständig und darf nicht mit der Anwendung Instanz-Management Komponente verwechselt werden. Zudem bietet die Instanz-Management Komponente zwei verschiedene Web Services und einen Web Service Client an. Diese Komponenten sind blau gefärbt. Der *Instanz-Management Web Service* dient zur Kommunikation mit dem Client, der in der Abbildung links dargestellt ist. Der *Rückruf Web Service* stellt eine Schnittstelle für die Automaten-Umgebung dar. Der Web Service Client der Instanz-Management Komponente hat die Aufgabe, den Kontakt mit dem *Automaten Web Service* aufzunehmen. Die Automaten-Umgebung wird in Abb. 5.4 gleich unterteilt wie die Instanz-Management Komponente. Dabei muss beachtet werden, dass mehrere Automaten-Umgebungen existieren können. Die Automaten-Umgebung teilt die Kernfunktionalität auf drei verschiedene Komponenten auf, die in Abb. 5.4 grau markiert sind. Zudem ist bei der Automaten-Umgebung ein Web Service und ein Web Service Client enthalten. Diese Komponenten sind blau gekennzeichnet. Der *Automaten Web Service* dient bei der Kommunikation mit der Instanz-Management Komponente dazu, um Anfragen an die Automaten-Umgebung zu übermitteln. Der *Web Service Client* der Automaten-Umgebung wird dazu verwendet, um Rückrufe zur Instanz-Management Komponente durchzuführen.

Die Web Services dienen zudem dazu, die grau markierten Komponenten der Anwendungen zu aktivieren. Die Web Service Clients der jeweiligen Anwendungen werden umgekehrt durch diese Komponenten aktiviert, wodurch die Kommunikation mit der jeweils anderen Anwendung ermöglicht wird. Diese Beziehungen werden durch die blau gestrichelten Pfeile in der Abbildung dargestellt.

Außerdem kommunizieren beide Anwendungen mit Web Services von Amazon, die zur Speicherung von Daten verwendet werden. Diese sind in Abb. 5.4 orange dargestellt. SQS wird hierbei als Warteschlange und persistenter Speicher für Anfragen verwendet. Durch SQS werden außerdem Anfragen zwischen den Web Services und den anderen Komponenten weitergeleitet. Der Speicherdienst S3 wird nur durch die Instanz-Management Komponente verwendet. Der Zugriff auf die Services von Amazon erfolgt hierbei ausschließlich durch die grau markierten Komponenten, die die Anwendungslogik implementieren.

5.3. Implementierung der Instanz-Management Komponente

Im vorigen Abschnitt wurde die Architektur des Systems vorgestellt. Im Folgenden Abschnitt wird die Implementierung der Komponenten der Instanz-Management Komponente erklärt. Eine Übersicht der Komponenten wird in Abbildung 5.4 dargestellt. In Abbildung 5.5 wird die Interaktion der verschiedenen Komponenten der Instanz-Management Komponente gezeigt.

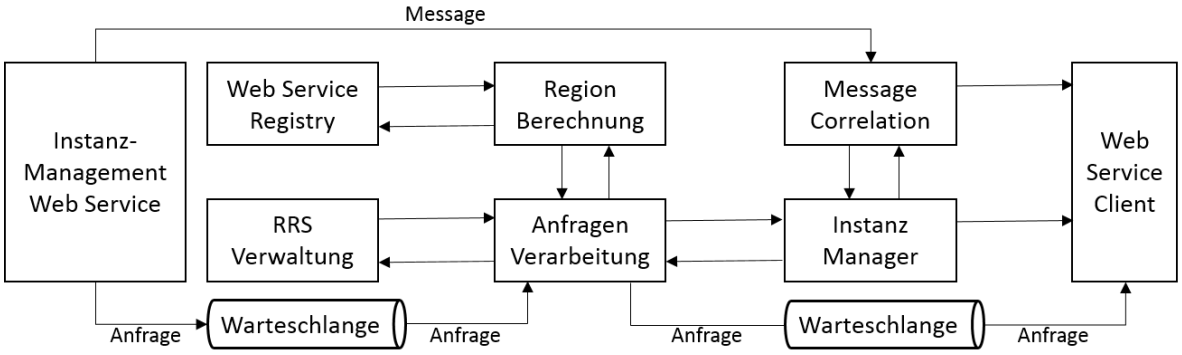

Abbildung 5.5.: Die Interaktion der Bestandteile der Instanz-Management Komponente. Auf den *Rückruf Web Service* wurde der Übersicht halber verzichtet. Er kommuniziert mit den Komponenten *RRS Verwaltung*, *Anfragen Verarbeitung*, *Instanz Manager* und *Message Correlation*.

Der *Instanz-Management Web Service* empfängt Anfragen und sendet diese an eine Warteschlange. Die Komponente *Anfragen Verarbeitung* entfernt Anfragen aus der Warteschlange und bearbeitet sie. Dabei verwendet sie zur Berechnung der Zielregion die Komponente *Region Berechnung*. Diese verwendet dabei die Komponente *Web Service Registry*. Diese stellt einen Index der Web Services aus der Amazon Cloud zur Verfügung. Nach der Berechnung

der Region fordert *Anfragen Verarbeitung* von *RRS Verwaltung* eine Instanz des Reference Resoultion Systems (RRS) an und von *Instanz Manager* eine Instanz einer Automaten-Umgebung. Die Anfrage wird dann in einer Warteschlange gespeichert und durch den *Web Service Client* zur Automaten-Umgebung geschickt. Falls der *Instanz-Management Web Service* eine Message erhält, übergibt er diese der *Message Correlation* Komponente. Diese fordert das Ziel der Message von *Instanz-Manager* an und übergibt sie dann dem *Web Service Client*.

5.3.1. Web Services und Web Service Client

Die Instanz-Management Komponente bietet zwei verschiedene Web Services an. Der *Instanz-Management Web Service* aus Abb. 5.4 dient zur Kommunikation mit dem Client. Er bietet zwei Operationen. Die Operation `sendRequest(..)` wird dazu verwendet, der Instanz-Management Komponente Anfragen zu übermitteln. Die Operation `sendMessage(..)` dient zur Übermittlung von Messages an die Instanz-Management Komponente. Die Operationen von *Rückruf Web Service* können von den Automaten-Umgebungen für Rückmeldungen an die Instanz-Management Komponente verwendet werden. Die Instanz-Management Komponente besitzt außerdem einen Web Service Client, der zur Kontaktaufnahme mit dem *Automaten Web Service* der Automaten-Umgebung dient. Hierbei wird eine Verbindung zu dem *Automaten Web Service* der Automaten-Umgebung aufgebaut und danach die gewünschte Operation aufgerufen. Der Web Service Client bietet einen Aufruf für jede Operation des *Automaten Web Services*.

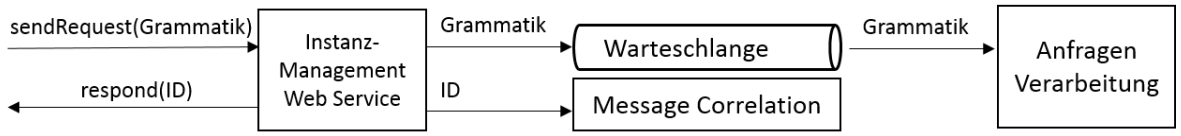

Abbildung 5.6.: Das Verhalten der Operation sendRequest des *Instanz-Management Web Services*. Die Pfeile stehen für den Austausch von Daten. Die Rechtecke symbolisieren die beteiligten Komponenten.

Die Erzeugung des Codes für die Web Services und der Clients erfolgt mit dem Framework ApacheCXF[1]. Hierzu muss eine Basisklasse angegeben werden, deren Methoden als Web Service Operationen angeboten werden sollen. Der *Instanz-Management Web Service* ist in der Klasse `ManagementWS` implementiert. Das Verhalten der Operation `sendRequest(..)` wird in Abbildung 5.6 dargestellt. Die Operation benötigt als Parameter eine Grammatik und liefert eine Anfragen-ID zurück, die bei dieser Operation generiert wird. Gleichzeitig wird die Anfragen-ID durch die *Message Correlation* Komponente gespeichert. Die Grammatik wird lediglich zu einer Warteschlange weitergeleitet. Die Warteschlange wird durch Amazon SQS zur Verfügung gestellt. Zum Zugriff auf die Warteschlange wird die SQS-API verwendet. Die Komponente *Anfragen Verarbeitung* entfernt die Anfrage zur Verarbeitung aus der

[1]http://cxf.apache.org/

Warteschlange. Die Methode `sendMessage(..)` nimmt als Parameter eine Nachricht und eine ID entgegen. Diese werden direkt an die Message Correlation Komponente weitergeleitet. Der *Rückruf Web Service* ist in der Klasse `AutomatonCallback` implementiert. Die Operationen von `AutomatonCallback` werden schrittweise in den Abbildungen 5.8, 5.9und 5.10 dargestellt und in den folgenden Kapiteln erklärt.

5.3.2. Anfragen Verarbeitung

Die Komponente *Anfragen Verarbeitung* verarbeitet eingehende Anfragen. Bei der Verarbeitung sind vor allem die Klassen `RequestHandler` und `RequestHandlerManager` relevant. `RequestHandlerManager` verwaltet einzelne Instanzen der Klasse `RequestHandler`. Von `RequestHandlerManager` existiert immer nur eine Instanz (*Singleton-Pattern*). Sowohl `RequestHandlerManager` als auch `RequestHandler` leiten von *Thread*[2] ab und können somit als Thread ausgeführt werden. Der `RequestHandlerManager`-Thread läuft dabei so lange, bis er von anderer Stelle gestoppt wird.

Der `RequestHandlerManager` Thread erzeugt nicht beliebig viele `RequestHandler` Threads, sondern limitiert deren Anzahl über die Anzahl der verfügbaren CPU-Kerne. Wenn eine Anfrage zu bearbeiten ist, wird zuerst überprüft, ob es `RequestHandler` Threads gibt, die keine Aufgabe besitzen. Wenn dies der Fall ist, wird die Anfrage einem solchen Thread zugewiesen. Ansonsten wird überprüft, ob ein neuer Thread erzeugt werden kann, der die Aufgabe dann übernimmt. Ist dies nicht der Fall, muss gewartet werden, bis ein existierender Thread seine Aufgabe abschließt und wieder bereit steht.

Ein `RequestHandler`-Thread lässt die Zielregion einer Anfrage berechnen und fordert dort die Instanz einer Automaten-Umgebung von der Komponente *Instanz Manager* an. Wenn die Instanz der Automaten-Umgebung Anfragen entgegen nehmen kann, wird die Anfrage sofort über den *Web Service Client* dorthin gesendet. Falls die Instanz noch nicht bereit ist oder in der Region noch kein RRS existiert, wartet der `RequestHandler` Thread nicht. Stattdessen wird über SQS eine Warteschlange erstellt, in der alle Anfragen gespeichert werden, die in dieser Region ausgeführt werden sollen, aber auf die Automaten-Umgebung warten müssen. Der `RequestHandler` Thread ist somit in der Wartezeit nicht blockiert, sondern kann neue Anfragen bearbeiten. Sobald das RRS und die Automaten-Umgebung dieser Region sich bereit melden werden die Anfragen in der Warterschlange an die Automaten-Umgebung der Region gesendet. Das RRS gilt dabei als bereit, sobald eine Instanz des RRS durch die Komponente *RRS Verwaltung* zur Verfügung gestellt wird. Die Automaten-Umgebung meldet sich durch die Verwendung der Operation `automatonReady(..)` des *Rückruf Web Services* der Instanz-Management Komponente bereit. Wenn während einem beliebigen Verarbeitungsschritt eine *Exception*[3] geworfen wird, wird die Anfrage zurück zur Warteschlange für Anfragen gesendet, damit diese nicht verloren geht. Dort bleibt sie, bis sie vom `RequestHandlerManager` Thread erneut einem `RequestHandler` zugewiesen

[2] java.lang.Thread
[3] java.lang.Exception

wird. Ist die Bearbeitung einer Anfrage abgeschlossen, terminiert der entsprechende Thread nicht, sondern meldet der Instanz von `RequestHandlerManager`, dass er keine Anfrage zu verarbeiten hat und wartet, bis er eine neue Anfrage von der `RequestHandlerManager` Instanz erhält.

5.3.3. Regionen Berechnung

Die Berechnung der Zielregion wird durch die verschiedenen `RequestHandler` Threads, deren Abläufe bereits erklärt wurden, initiiert. Die Implementierung der Berechnung hält sich dabei an Algorithmus 4.1, wobei hier einige Besonderheiten beachtet werden müssen.

Am Anfang der Berechnung muss ein Index der Web Services erstellt werden, die in Elastic Beanstalk verfügbar sind. Dies geschieht mittels der Komponente *Web Service Registry*, deren Implementierung in Abschnitt 5.3.4 vorgestellt wird. Der *Web Service Registry* Komponente werden hierfür die Namen der Web Services und der Operationen mitgeteilt, die in einer Grammatik verwendet werden. Um die Namen der Web Services zu ermitteln, muss zunächst die XML-Grammatik geparst werden. Dies geschieht in der Klasse `XMLGrammar`. Beim Parsen der Web Service Namen und Operationen wird nach den Typen der Nicht-Terminale gesucht, die für Web Services stehen (siehe 2.2.1). In Algorithmus 4.1 müssen zudem die Web Service Aufrufe gezählt werden. Hierfür wird ebenfalls die Grammatik analysiert, wobei die Klasse `XMLGrammar` verwendet wird.

Für die Ermittlung der Distanzen zwischen den Regionen in Algorithmus 4.1 wird die Klasse `RegionMap` verwendet. In dieser sind die Distanzen der Regionen zueinander hart kodiert. Für die Distanzen wurden die ungefähren Luftliniendistanzen der Rechenzentren zueinander verwendet.

Sobald die Zielregion ermittelt ist, wird für jeden Web Service, der in der Amazon Cloud verfügbar ist, die URL in die Grammatik eingefügt. Somit wird bei der Ausführung der Grammatik der Web Service in der Ablaufumgebung über die eingefügte URL erreicht. Hierbei werden die Web Services verwendet, die sich in der Zielregion befinden oder der Zielregion am nächsten sind. Das Einfügen der Web Service-URLs in die Grammatik geschieht über die Klasse `XMLGrammar`. Für Web Services, die nicht in der Amazon Cloud gehostet sind, wird keine Änderung der Adresse vorgenommen.

5.3.4. Web Service Registry

Die *Web Service Registry* ist ein wichtiger Bestandteil für die Berechnung der Zielregion einer Grammatik, da sie einen Web Service Index implementiert. Die wichtigste Funktionalität ist dabei auf die Klassen `RegistryManager` und `Crawler` aufgeteilt. Die Klasse `RegistryManager` speichert Informationen zu den Web Services, die bei der Erstellung des Indexes berücksichtigt werden sollen. Gespeichert werden der Name des Web Services und die Namen seiner Operationen, da diese benötigt werden, um einen Web Service zu identifizieren. Zusätzlich dazu wird der Name des Web Service Ports gespeichert, da dieser für den Zugriff auf die

WSDL-Datei eines Web Services notwendig ist. Den verschiedenen Web Services werden die IDs der Elastic Beanstalk Anwendungen zugeordnet, die diesen Web Service anbieten. Die Information über die zu suchenden Web Services müssen dabei von einer anderen Komponente hinzugefügt werden. Dies geschieht in der Regel durch die Komponente *Region Berechnung*.

Die Erstellung des Web Service Indexes geschieht über die Klasse `Crawler`, die von *Thread* ableitet. Eine Instanz von `Crawler` wird über den Konstruktor der Klasse immer an eine bestimmte Region gebunden. Bei der Suche wird vorgegangen wie folgt: Zunächst wird über die Elastic Beanstalk API eine Liste aller der Instanz-Management Komponente verfügbaren Anwendungen in einer Elastic Beanstalk Region abgefragt. Nun wird für jede der Anwendungen überprüft, ob sie einen Web Service enthält. Dabei wird zunächst mit Hilfe des Portnamens getestet, ob die Anwendung eine WSDL-Datei enthält. Wenn dies nicht der Fall ist, bietet die Anwendung den entsprechenden Web Service nicht an. Wenn die Anwendung aber eine WSDL-Datei enthält, wird in dieser überprüft, ob der Name des Services und seine Operationen dem gesuchten Web Service entsprechen. Wenn dies der Fall ist, wird die Elastic Beanstalk ID dieser Anwendung in der Klasse *RegistryManager* gespeichert und dem Web Service zugeordnet. Dieser Vorgang wird bei allen Elastic Beanstalk Anwendungen für jeden zu suchenden Web Service durchgeführt. Bei einer Aktualisierung des Indexes werden bereits bekannte Anwendungen, die einen Web Service anbieten, überprüft. Dabei wird getestet, ob der Web Service noch verfügbar ist. Bei der Überprüfung wird zunächst über die Elastic Beanstalk API abgefragt, ob die Anwendung und ihre Infrastruktur noch existiert. Im nächsten Schritt wird die WSDL-Datei der Anwendung überprüft. Dabei wird untersucht, ob der Web Service Namen und die Namen der Operationen noch dieselben sind. Wenn die Anwendung nicht mehr existiert oder die WSDL-Datei sich verändert hat, wird sie aus dem Index gelöscht.

`RegistryManager` bietet eine blockierende und eine nicht-blockierende Aktualisierung des Web Service Indexes an. Die nicht-blockierende Version kann verwendet werden, um initial die existierenden Web Services in die Registry aufzunehmen. In diesem Fall können die `Crawler` Threads im Hintergrund den Index erstellen. Die blockierende Variante ist für die Berechnung der Region erforderlich, da es hier zwingend notwendig ist, einen aktuellen Index zu besitzen. Bei der blockierenden Methode wird gewartet, bis der Index jeder Region aktualisiert wurde.

5.3.5. Instanz-Manager

Wenn bei der Verarbeitung einer Anfrage oder in einer anderen Situation eine Instanz einer Automaten-Umgebung angefordert wird, fällt dies in den Aufgabenbereich der Komponente *Instanz-Manager*. Die Verwaltung der Instanzen geschieht dabei hauptsächlich über die Klassen `InstanceManagement` und `InstanceRepresentation`. Bei `InstanceManagement` handelt es sich um die zentrale Klasse, die Informationen über alle Instanzen der Automaten-Umgebung besitzt. Die Verwaltungsinformationen der Automaten-Umgebung werden in einer Instanz der Klasse `InstanceRepresentation` gespeichert. Diese bietet Zugriff auf die relevanten Eigenschaften der Elastic Beanstalk Anwendung, wie die URL und den Status

der Anwendung. Es kann nur eine Instanz der Klasse `InstanceManagement` existieren. Auf diese kann in der Anwendung global zugegriffen werden. Sie verwaltet alle Instanzen von `InstanceRepresentation`. `InstanceManagement` steuert den Zugriff auf die Instanzen von `InstanceRepresentation` und lässt nur eine dieser Instanzen pro Region zu. Wenn eine Instanz von `InstanceRepresentation` aus einer bestimmten Region von `InstanceManagement` angefordert wird, wird entweder eine neue Instanz von `InstanceRepresentation` erzeugt oder eine bestehende zurückgeliefert.

Bei der Erzeugung einer Instanz von `InstanceRepresentation` wird die Instanz einer Automaten-Umgebung in Elastic Beanstalk erzeugt. Hierfür sind mehrere Schritte notwendig, die in Abbildung 5.7 visualisiert werden. In sämtlichen Schritten wird die Java API verwendet, um Kontakt zu AWS aufzunehmen. Eine zentrale Rolle spielt hierbei die Klasse `AWSElasticBeanstalkClient`, die in der Java API enthalten ist. Über diese werden sämtliche Aufrufe an die Elastic Beanstalk AWS Schnittstelle übermittelt.

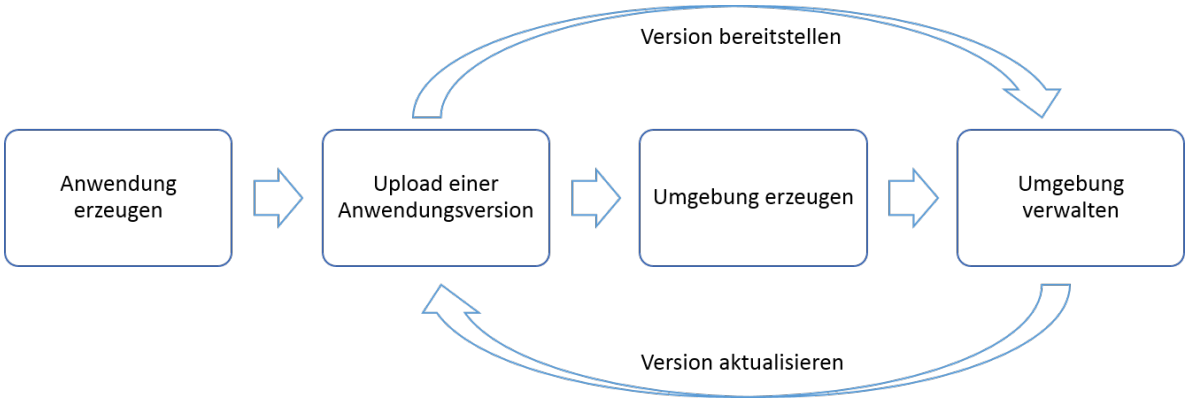

Abbildung 5.7.: Die Schritte zur Erzeugung einer Elastic Beanstalk Anwendung.

Um eine Anwendung auf Elastic Beanstalk bereitzustellen, wird dort zunächst eine Anwendung erzeugt, die noch keine Funktion bietet. Hierbei wird lediglich ein Name für die Anwendung angegeben. Im nächsten Schritt wird eine Implementierung der Anwendung in einem Container auf Amazon S3 hochgeladen. Die Implementierung wird als *Anwendungsversion* bezeichnet. Die Anwendungsversion wird dabei als *Web Application Archive* (WAR-Datei) bereitgestellt. Eine WAR-Datei enthält eine Anwendung, die nach der Java-Servlet-Spezifikation verpackt ist. Schließlich wird eine sogenannte *Umgebung* erzeugt. Hierbei wird festgelegt, welche Anwendung in dieser Umgebung betrieben werden soll und welche Anwendungsversion verwendet werden soll. Beim Erzeugen der Umgebung wird die Infrakstruktur für die Anwendung bereitgestellt. Darunter fallen EC2-Instanzen, ein Load-Balancer, eie Auto Scaling Gruppe und CloudWatch Alarme. Bei der Erstellung der Umgebung kann die Infrakstur manuell konfiguriert werden. An dieser Stelle wird ein Betriebssystem (Linux) und ein Anwendungsserver (Tomcat) für die EC2-Instanzen festgelegt. Für sämtliche andere Optionen werden die Standardeinstellungen verwendet. Nachdem die Umgebung erzeugt wurde und die Infrastruktur bereit steht, kann sie beliebig modifi-

ziert werden. Zum Beispiel kann wie in Abbildung 5.7 die Anwendungsversion aktualisiert werden.

Die Klasse `InstanceRepresentation` bietet nicht nur Zugriff auf die Elastic Beanstalk Anwendung, sondern auch auf die EC2-Instanzen, die verwendet werden, um die Anwendung zu betreiben. Dies ist insbesondere für die *Message Correlation* Komponente erforderlich, da hierfür gezielt EC2-Instanzen adressiert werden müssen. Um Zugriff auf die EC2-Instanzen zu erhalten, muss die EC2-API verwendet werden. Mithilfe der EC2-Api wird eine Liste aller EC2-Instanzen der Region abgefragt. Die IDs dieser EC2-Instanzen werden mit den IDs der EC2-Instanzen verglichen, die der Elastic Beanstalk Umgebung zugeordnet sind. So kann festgestellt werden, welche der EC2-Instanzen zum Betrieb der Elastic Beanstalk Anwendung verwendet werden. Die Informationen, die zum Zugriff auf die EC2-Instanzen verwendet werden, werden in der Klasse `InstanceRepresentation` gespeichert.

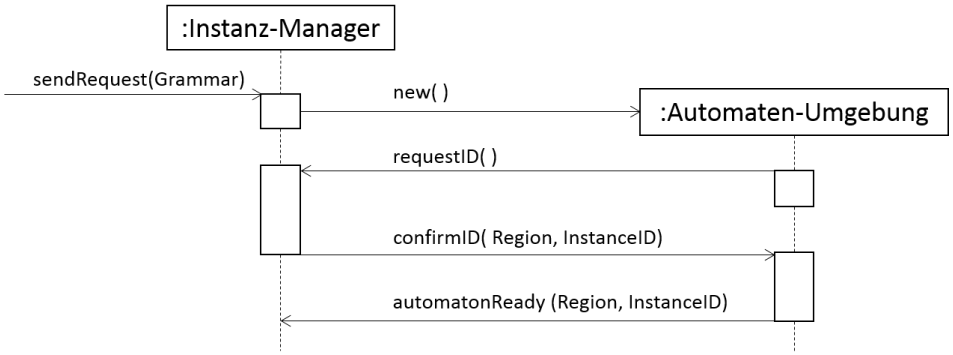

Abbildung 5.8.: Die Nachrichten zur Bestätigung einer Automaten-Umgebung in einem UML-Sequenzdiagramm.

Über `InstanceRepresentation` kann außerdem abgefragt werden, ob die entsprechende Instanz der Automaten-Umgebung bereit ist, Anfragen zu übernehmen, oder nicht. Eine Automaten-Umgebung ist nicht immer bereit, da es eine gewisse Zeit erfordert, die Ressourcen für die Umgebung bereitzustellen, nachdem der Befehl zur Erstellung der Umgebung abgesendet wurde. In dieser Zeit ist es dementsprechend noch nicht möglich, eine Anfrage an die Anwendung zu schicken. In `InstanceRepresentation` wird gespeichert, ob eine Automaten-Umgebung bereit ist Anfragen entgegen zu nehmen oder nicht. Wenn die Automaten-Umgebung bereit ist, muss dies für die einzelnen EC2-Instanzen, auf denen die Automaten-Umgebung betrieben wird, bestätigt werden. Um eine EC2-Instanz zu bestätigen, werden mehrere Schritte durchgeführt. Diese werden in Abbildung 5.8 dargestellt. Hier wird zunächst eine Grammatik durch `sendRequest(..)` übermittelt. Daraufhin wird eine neue Automaten-Umgebung erzeugt. Diese beginnt den Bestätigungsvorgang durch den Aufruf der Operation `requestID()` des *Rückruf Web Services*. Über die Klasse `InstanceManagement` wird nun von sämtlichen existierenden `InstanceRepresentation` Instanzen eine Liste der unbestätigten EC2-Instanzen angefordert. Bei diesen Instanzen wird die Web Service Operation `confirmID(..)` des *Automaten Web Service* der Automaten-Umgebung aufgerufen. Dabei werden die Region und die ID der jeweiligen EC2-Instanz als Parameter übergeben.

Eine EC2-Instanz wird jedoch erst als bestätigt gewertet, wenn ein Aufruf der Operation `automatonReady(..)` des *Rückruf Web Services* durch die Automaten-Umgebung erfolgt. Dabei werden erneut die Region und die ID der EC2-Instanz angegeben. Dieser Vorgang wird für alle unbestätigten EC2-Instanzen durchgeführt.

Die Klasse `InstanceManagement` sorgt zusätzlich für die Löschung von Automaten-Umgebungen. Für eine Löschung müssen Nachrichten zwischen Instanz-Management Komponente und Automaten-Umgebung ausgetauscht werden. Die Nachrichten werden in Abbildung 5.9 dargestellt. Eine Löschung wird durch den Aufruf der Operation `deleteAutomaton(..)` des *Rückruf Web Services* durch die Automaten-Umgebung ausgelöst. Die Automaten-Umgebung in einer Region kann jedoch auf mehreren EC2-Instanzen betrieben werden. Wenn eine dieser EC2-Instanzen durch die Automaten-Umgebung entfernt wird, verwendet diese immer die Operation `deleteAutomaton(..)`. Daher muss vor der Löschung der Automaten-Umgebung überprüft werden, ob nur noch eine EC2-Instanz in der Umgebung betrieben wird. Ansonsten wäre es möglich, dass die gesamte Automaten-Umgebung gelöscht wird, obwohl diese noch auf anderen EC2-Instanzen betrieben werden soll. Falls die Operation `deleteAutomaton(..)` durch die letzte existieren EC2-Instanz einer Automaten-Umgebung erfolgt, wird die Automaten-Umgebung gelöscht, wie in Abbildung 5.9 durch `delete()` dargestellt. Dabei sind dieselben Schritte nötig, wie zum Bereitstellen einer Elastic Beanstalk Anwendung. Auch hier müssen über die API die Umgebung, die Anwendung und die Anwendungsversion separat gelöscht werden. Das Löschen einzelner EC2-Instanzen ist nicht Aufgabe der Instanz-Management Komponente, sondern der Automaten-Umgebung.

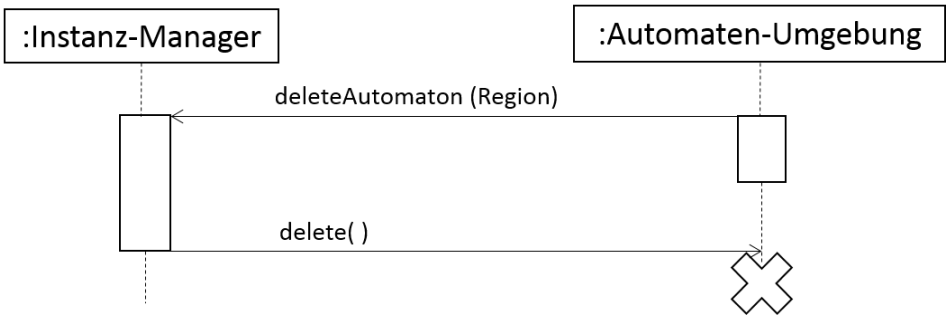

Abbildung 5.9.: Die Nachrichten zur Löschung einer Automaten-Umgebung in einem UML-Sequenzdiagramm.

5.3.6. RRS Verwaltung

Eine Bedingung, die für die Verarbeitung einer Grammatik erfüllt sein muss, ist das Vorhandensein des RRS in der Region der Automaten-Umgebung. Das RRS muss hierbei bereitgestellt werden, bevor eine Grammatik ausgeführt wird. Die Verwaltung der Instanzen des RRS ist analog zu der Verwaltung der Instanzen der Automaten-Umgebung implementiert. Der einzige Unterschied besteht darin, dass keine Skalierung vorgesehen ist (vgl. Abschnitt 4.2.6). In der Implementierung der RRS Verwaltung ergeben sich daher nur zwei nennenswerte

Unterschiede. Zum einen werden EC2-Instanzen nicht extra gesucht oder gespeichert. Der Zugriff erfolgt daher vollständig über die Elastic Beanstalk API. Zum anderen liefert das RRS selbst keine Rückmeldung darüber, wann es bereit ist. Der Zustand des RRS wird daher aktiv in einem Zeitintervall von 60 Sekunden über die Elastic Beanstalk API abgefragt. Sobald festgestellt wird, dass die RRS Instanz bereit ist, wird die Komponente *Anfragen Verarbeitung* benachrichtigt. Diese kann nun wartende Anfragen an die Automaten-Umgebung senden.

5.3.7. Message Correlation

Für die *Message Correlation* Komponente sind die Klassen `MessageCorrelation` und `IdTable` relevant. `MessageCorrelation` ist dabei für die reine Weiterleitung der Messages zuständig. In `IdTable` wird gespeichert, welche Anfrage von welcher Automaten-Umgebung verarbeitet wird. Dabei wird die ID der Anfrage der URL der Automaten-Umgebung zugeordnet. Hierbei wird immer die URL der EC2-Instanz gespeichert, nicht die URL des Load-Balancers der Umgebung, da dieser die Message zu einer beliebigen Automaten-Umgebung innerhalb der Auto-Scaling Gruppe schickt.

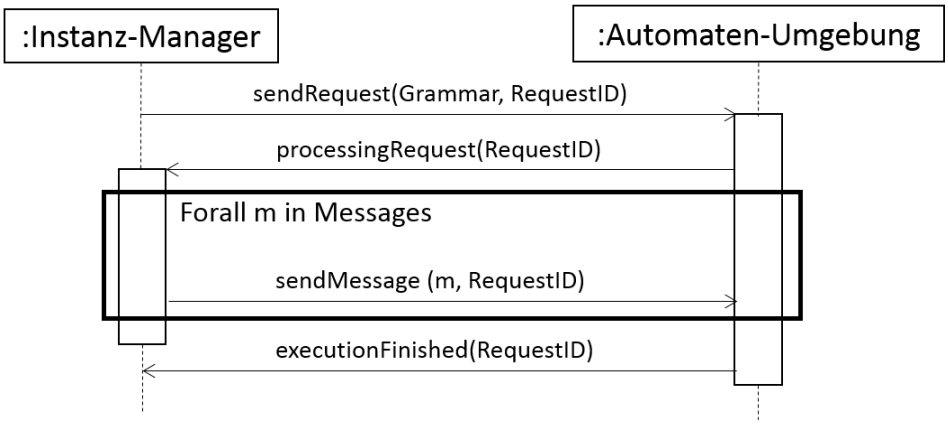

Abbildung 5.10.: Die Nachrichten zur Verarbeitung einer Anfrage in einem UML-Sequenzdiagramm.

`IdTable` verwendet eine Tabelle zur Zuordnung einer Anfragen-ID zu der Automaten-Umgebung, die diese bearbeitet. Ein Eintrag in der Tabelle wird beim Eingang einer Anfrage angelegt. Zu dieser Zeit ist die EC2-Instanz, auf der die Anfrage bearbeitet wird, jedoch noch nicht bekannt. Diese wird erst dann bekannt, wenn die Anfrage dorthin geschickt wurde. Der Eintrag in der Tabelle ist deshalb noch nicht vollständig. Um den Eintrag in der Tabelle zu vervollständigen, ist die Message Correlation Komponente auf eine Rückmeldung der Automaten-Umgebung angewiesen. Die hierfür verwendeten Nachrichten werden in Abbildung 5.10 gezeigt. In der Abbildung wird zunächst über `sendRequest(..)` eine Anfrage an die Automaten-Umgebung geschickt. Die Automaten-Umgebung teilt mit, durch welche EC2-Instanz die Anfrage verarbeitet wird. Dies geschieht über den *Rückruf Web Service* durch die Operation `processingRequest(..)`. Wenn die Instanz-Management Komponente diese

Nachricht erhalten hat, kann sie die Messages an die entsprechende EC2-Instanz schicken. Sobald die Automaten-Umgebung über `executionFinished(..)` des *Rückruf Web Service* signalisiert, dass die Anfrage verarbeitet wurde, kann der Tabelleneintrag dieser Anfrage gelöscht werden.

Der Vorgänge zur Weiterleitung einer Message werden in Abbildung 5.11 dargestellt. Die Weiterleitung erfolgt ebenfalls über die Klasse `MessageCorrelation`. Wenn eine Message bei der Web Service Implementierung `ManagementWS` eingeht, wird die Message über `forwardMessage(..)` zur Klasse `MessageCorrelation` weitergeleitet. In der Klasse `MessageCorrelation` wird mithilfe der Klasse `IdTable` überprüft, ob die Automaten-Umgebung, die die zugehörige Anfrage bearbeitet, bereits bekannt ist. Dies geschieht durch die Methode `getMessageTarget(..)`. Wenn das Ziel der Nachricht bekannt ist, wird die Message direkt dorthin gesendet. Der Fall, wenn das Ziel der Nachricht nicht bekannt ist, wird in Abb. 5.11 in der linken Bildhälfte dargestellt. Hier liefert `getMessageTarget(..)` keine URL zurück. Die Message wird in diesem Fall so lange in einer Warteschlange gespeichert, bis die Automaten-Umgebung bekannt ist. Das weitere Vorgehen wird in der rechten Bildhälfte von Abbildung 5.11 dargestellt. Durch die Klasse `AutomatenCallback` wird `MessageCorrelation` mitgeteilt, dass die Anfrage verarbeitet wird. Dies geschieht durch die Methode `processingRequest(..)`. Gleichzeitig wird in `IDTable` das Ziel der Messages gesetzt, was durch die Methode `addMessageTarget(..)` geschieht. Danach entfernt `MessageCorrelation` die Messages aus der Warteschlange und schickt sie zum Ziel.

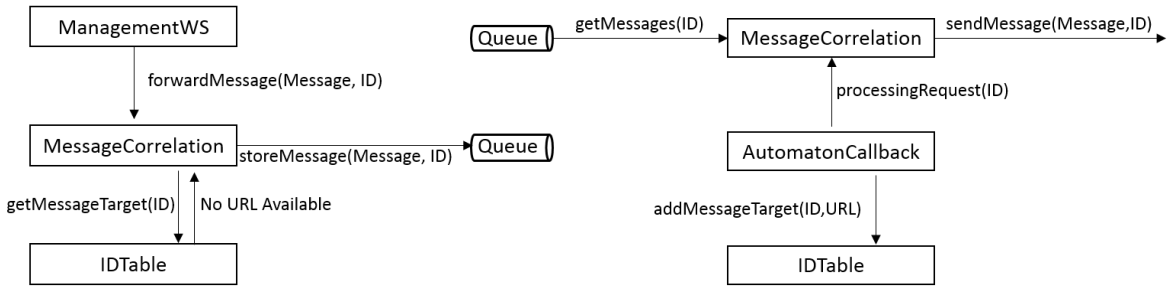

Abbildung 5.11.: Das linke Bild stellt das Vorgehen dar, wenn das Ziel einer Message noch nicht bekannt ist. Das rechte Bild stellt dar, wie die Message weitergeleitet wird.

5.4. Implementierung der Automaten-Umgebung

Nachdem die Implementierung des Instanz-Managers im vorigen Abschnitt vorgestellt wurde, wird in diesem Abschnitt auf die Implementierung der Automaten-Umgebung eingegangen. Hierbei werden einige Vorgänge nicht mehr, oder weniger genau, erklärt, da diese ähnlich implementiert sind, wie bei der Instanz-Management Komponente. Auf die Implementierung des Web Service Clients (siehe 5.3.1) und die Komponente *Automat Ausführung* (siehe Abb. 5.4) wird deshalb nicht mehr eingegangen. Bei *Automat Ausführung*

handelt es sich um den eigentlichen Automaten und zugehörige Komponenten. Diese wurden nicht im Rahmen dieser Arbeit entwickelt, weshalb die Implementierung hier nicht erläutert wird. Die Komponente wird lediglich als JAR-Datei in das Projekt der Automaten-Umgebung eingebunden, was den Zugriff auf deren Funktionen ermöglicht.

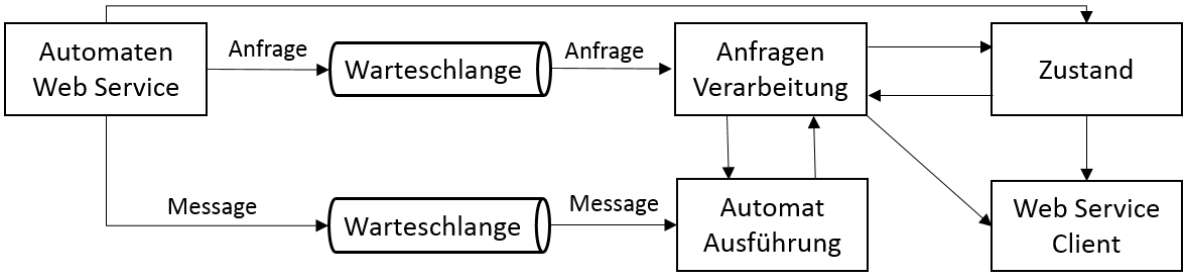

Abbildung 5.12.: Die Interaktion der Bestandteile der Automaten-Umgebung.

Eine Übersicht über die Komponenten der Automaten-Umgebung wird in Abb. 5.12 dargestellt. Der *Automaten Web Service* nimmt Anfragen und Messages entgegen und speichert sie in einer Warteschlange. Außerdem kommuniziert sie mit der Komponente *Zustand*. Die Komponente *Anfragen Verarbeitung* entfernt Anfragen aus der Warteschlange und steuert die Komponente *Automaten Ausführung*. *Automaten Ausführung* führt einen oder mehrere Automaten aus und gibt entsprechendes Feedback an *Anfragen Verarbeitung*. Dabei entnimmt *Automaten Ausführung* die Messages aus der Warteschlange. *Anfragen Verarbeitung* gibt zudem Informationen über den Verarbeitungsstatus von Anfragen an die *Zustand* Komponente weiter. Sowohl *Zustand* als auch *Anfragen Verarbeitung* geben über den *Web Service Client* Rückmeldungen an die Instanz-Management Komponente. Die genauen Abläufe werden in den folgenden Abschnitten erklärt.

5.4.1. Automaten Web Service

Der *Automaten Web Service* dient, gleich wie die Web Services der Instanz-Management Komponente, zur Übermittlung von Anfragen. Der *Automaten Web Service* besitzt drei verschiedene Operationen. Die Operation `sendRequest(..)` arbeitet identisch wie die gleich benannte Operation des *Instanz-Management Web Services*, nur wird an dieser Stelle keine ID erzeugt. Die Operation `sendMessage(..)` leitet eine Message im Gegensatz zur Instanz-Management Komponente direkt an eine Warteschlange weiter. Zusätzlich gibt es noch die Operation `confirm(..)`, deren Funktion in Abschnitt 5.4.2 erklärt wird.

Auch die Web Services der Automaten-Umgebung wurden mit Apache CXF erzeugt.

5.4.2. Zustand

Ein wichtiger Teil zur Verwaltung der Automaten-Umgebung stellt die Komponente *Zustand* dar. Hierbei ist vor allem die Klasse `Status` zu erwähnen. Durch diese wird festgelegt, wann

die Automaten-Umgebung bereit ist, Anfragen entgegen zu nehmen und wann nicht. Die Zustände der Automaten-Umgebung werden in Abbildung 5.13 dargestellt. Blaue Rechtecke stehen in der Abbildung für Zustände. Blaue Pfeile symbolisieren Zustandsübergänge. Die Automaten-Umgebung besitzt mehrere Zustände, in denen sie bereit (in der Abbildung grün umrandet) oder nicht bereit ist (in der Abbildung rot umrandet).

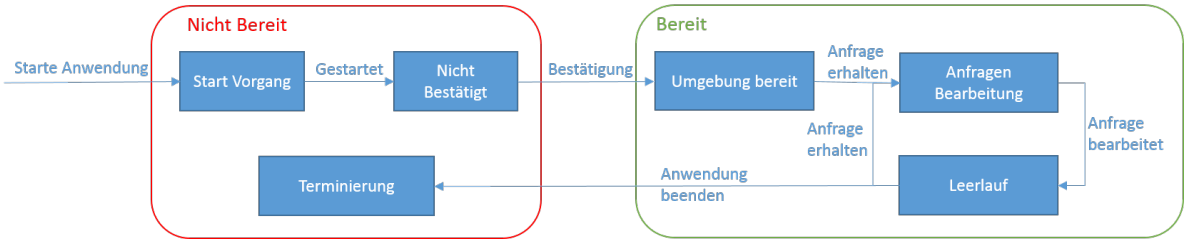

Abbildung 5.13.: Die Zustände der Automaten-Umgebung im Überblick.

Zu Beginn ist die Anwendung nicht bereit, da sie zunächst gestartet werden muss. Nach dem Startvorgang muss die Umgebung zunächst bestätigt werden (vgl. Abb. 5.13). Die Automaten-Umgebung gilt als bestätigt, wenn in `Status` die Region der Automaten-Umgebung und die ID der EC2-Instanz, auf der die Automaten-Umgebung ausgeführt wird, gespeichert ist. Beide Informationen sind zu Beginn nicht verfügbar und werden der Automaten-Umgebung über die Operation `confirm(..)` des *Automaten Web Service* durch die Instanz-Management Komponente mitgeteilt (siehe Abb. 5.8). Die Instanz-Management Komponente verwendet die Operation `confirm(..)` erst, wenn die Automaten-Umgebung die Operation `requestID(..)` des *Rückruf Web Services* der Instanz-Management Komponente verwendet. Die Verwendung von `requestID(..)` durch die Automaten-Umgebung muss dabei zuerst erfolgen, da die Instanz-Management bis zu diesem Zeitpunkt keine Information darüber verfügt, ob die Infrastruktur der Automaten-Umgebung bereit steht. Sobald die Operation `confirm(..)` durch die Instanz-Management Komponente verwendet wird, wird die Operation `automatonReady(..)` der Instanz-Management Komponente aufgerufen und die Automaten-Umgebung kann nun Anfragen bearbeiten (siehe Abb. 5.8). Der Aufruf von `automatonReady(..)` ist erforderlich, um der Instanz-Management Komponente mitzuteilen, dass die Automaten Umgebung nun bereit ist. Die Automaten Umgebung befindet sich nun im Zustand *Umgebung bereit* aus Abbildung 5.13. In `Status` wird außerdem gespeichert, ob die Automaten-Umgebung gerade eine Anfrage bearbeitet oder sich im Leerlauf befindet. Dies wird durch in Abbildung 5.13 durch die Zustände *Anfragen Bearbeitung* und *Leerlauf* ausgedrückt. Sobald die Automaten-Umgebung in den Zustand *Leerlauf* übergeht, wird ein Timer von 15 Minuten gestartet. Zu dem Zeitpunkt, an dem dieser Timer abläuft, wird die Operation `deleteAutomaton(..)` des *Rückruf Web Services* aufgerufen (siehe Abb. 5.9). Falls während der 15 minütigen Wartezeit eine neue Anfrage eingeht, wird der Timer abgebrochen. Sobald die Anwendung gelöscht wird, das heißt in den Zustand *Terminierung* übergeht, gilt sie als *nicht bereit* und kann keine Anfragen mehr entgegen nehmen.

5.4.3. Anfragen Verarbeitung

Die Verarbeitung von Anfragen erfolgt ähnlich wie beim Instanz-Manager und wird durch den *Automaten Web Service* angestoßen, wenn eine Anfrage eingeht. Dabei verhalten sich die Klassen `AutomatonThreadManager` und `AutomatonThread` gleich zueinander wie `RequestHandlerManager` und `RequestHandler`.

Jede Instanz von `AutomatonThread` führt einen Automaten aus. Die Instanzen von `AutomatonThread` erhalten die Grammatiken hierfür von der einzigen existierenden Instanz von `AutomatonThreadManager`. Die Instanz von `AutomatonThreadManager` bezieht die Anfragen wiederum aus der SQS Warteschlange, in der die Anfragen durch den *Automaten Web Service* gespeichert werden. Wenn der Automat einer `AutomatonThread` Instanz die Ausführung einer Grammatik beendet hat, teil der `AutomatonThread` dies der Instanz von `AutomatonThreadManager` mit. Diese benachrichtigt `Status` darüber, wenn sich alle Threads im Leerlauf befinden, wodurch dort der Lösch-Timer gestartet wird. Außerdem werden vor und nach der Ausführung eines Automaten die jeweiligen Operationen `processingRequest(..)` und `executionFinished(..)` des *Rückruf Web Services* ausgeführt (vgl. Abb. 5.10).

In der Komponente *Anfragen Verarbeitung* ist zudem die Funktion implementiert, Messages an bestimmte Instanzen von `AutomatonThread` weiterzuleiten. Hierbei wird ähnlich vorgegangen, wie bei der *Message Correlation* Komponente der Instanz-Management Komponente. Es wird lokal eine Tabelle geführt, die speichert, welcher Thread welche Anfrage bearbeitet. Die Zuordnung erfolgt dabei über die eindeutige ID des Threads, die durch das Betriebssystem vergeben wird. Um die Messages bis zur Verwendung zu speichern, besitzt jeder Thread eine eigene Warteschlangen auf SQS. Der Zugriff auf die Warteschlange erfolgt über die SQS API.

5.4.4. Skalierung

Um die Automaten-Umgebung skalierbar zu gestalten, werden über Elastic Beanstalk die Services Load-Balancing, Auto Scaling und CloudWatch verwendet. Die Skalierung erfolgt ausschließlich über diese Services. Sie werden automatisch beim Start der Elastic Beanstalk Anwendung verwendet. Durch den Auto Scaling Service werden dabei auf verschiedenen EC2-Instanzen jeweils eine komplette Automaten-Umgebung erstellt. Der Auto-Scaling Service ist dabei so konfiguriert, dass eine neue EC2-Instanz gestartet wird, sobald auf allen bestehenden EC2-Instanzen über einen Zeitraum von drei Minuten eine durchschnittliche CPU Last von über 80% vorliegt. Dabei werden maximal fünf EC2-Instanzen pro Automaten-Umgebung erstellt. Eine EC2-Instanz wird gelöscht, wenn bei dieser über einen Zeitraum von drei Minuten eine durchschnittliche CPU Last von unter 50% vorliegt. Die Anfragen, die von der Instanz-Management Komponente an die Automaten-Umgebung geschickt werden, werden durch den Load-Balancer auf die EC2-Instanzen, auf denen die Automaten-Umgebungen betrieben wird, verteilt. Der Load-Balancer schickt Anfragen dabei bevorzugt an EC2-Instanzen mit niedriger CPU Last. Dies bringt das Problem mit sich, dass Anfragen auch an eine EC2-Instanz geschickt werden können, obwohl diese noch nicht bereit ist,

das heißt noch nicht von der Instanz-Management Komponente bestätigt wurde. Ohne die Region und ID der EC2-Instanz zu kennen, können die Anfragen jedoch nicht verarbeitet werden. Sie werden aus diesem Grund so lange lokal zwischengespeichert, bis diese Informationen vorhanden sind. Sobald die Region bekannt ist, werden die Nachrichten zur SQS Warteschlange geschickt, die von allen Automaten-Umgebungen der Region verwendet wird, um Anfragen zu speichern. Ohne die Region zu kennen, ist auch dies nicht möglich, da bei SQS immer eine Region explizit angegeben werden muss.

6. Diskussion

In diesem Kapitel soll das Ergebnis der Arbeit diskutiert werden. Hierzu wird in Abschnitt 6.1 die beispielhafte Verarbeitung einer Anfrage durch den die Instanz-Management Komponente schrittweise vorgestellt, um die Funktionen der Instanz-Management Komponente zusammenzufassen. Auf dieser Basis wird in 6.2 untersucht, ob die Ziele der Arbeit und die Anforderungen aus Kapitel 3 erfüllt wurden. Abschließend werden in 6.3 eventuelle Schwächen und Verbesserungsmöglichkeiten diskutiert.

6.1. Beispielhafte Verarbeitung einer Anfrage

In diesem Abschnitt wird die beispielhafte Verarbeitung einer Anfrage vorgestellt, um die Funktionsweise der Instanz-Management Komponente und der Automaten-Umgebung zusammenzufassen. Der Ablauf wird außerdem in einem UML-Aktivitätsdiagramm dargestellt (siehe Abb. 6.1).

Die Verarbeitung einer Anfrage wird stets durch den Empfang einer Anfrage durch die Instanz-Management Komponente initiiert. Bei der Anfrage handelt es sich um eine formale Grammatik in XML Form. Die Anfrage wird durch die Instanz-Management Komponente entgegengenommen und es wird eine eindeutige ID für diese Anfrage generiert. Die ID wird gespeichert und zum Client zurückgesendet. Die Anfrage selbst wird in die Warteschlange für Anfragen eingeordnet.

Nun beginnt die Verarbeitung der Anfrage. Der erste Schritt dabei ist die Berechnung der Zielregion, in der die Grammatik ausgeführt werden soll. Dieser Schritt wird in der Abbildung durch den Zustand *Region Berechnen* dargestellt. Danach wird eine Instanz einer Automaten-Umgebung in der Zielregion angefordert. An dieser Stelle gibt es zwei Möglichkeiten, die in der Abbildung durch die Verzweigung *Aut.-Umg. verfügbar?* dargestellt werden. Bei der ersten Möglichkeit ist in dieser Region bereits eine Instanz der Automaten-Umgebung vorhanden, die bereit ist Anfragen entgegenzunehmen. In diesem Fall wird die Anfrage direkt an die Automaten-Umgebung geschickt. Ist keine Instanz der Automaten-Umgebung vorhanden, wird in der Zielregion eine neue erstellt. Die Anfrage wird in einer Warteschlange gespeichert, bis die neue Instanz der Automaten-Umgebung bereit dazu ist, Anfragen entgegenzunehmen. Bevor die Automaten-Umgebung Anfragen entgegen nehmen kann, muss diese zunächst durch die Instanz-Management Komponente bestätigt werden (Zustand *Bestätigung* in der Abbildung).

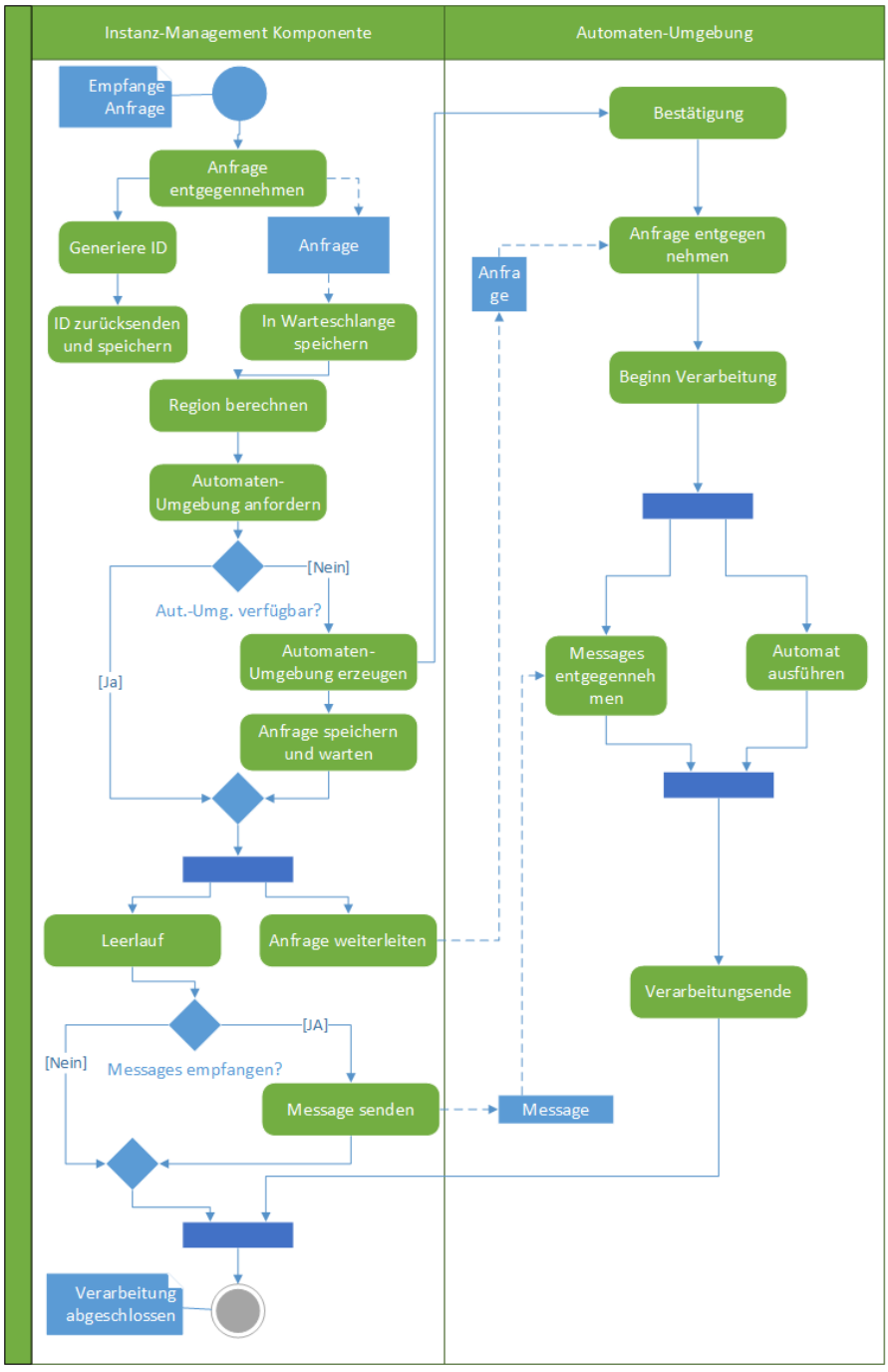

Abbildung 6.1.: Verarbeitung einer Anfrage in einem UML Aktivitätsdiagramm.

Die weiteren Aktionen von Instanz-Management Komponente und Automaten-Umgebung finden parallel statt. Dies wird durch den blauen Parallelisierungsbalken bei der Instanz-Management Komponente dargestellt. Die Instanz-Management Komponente leitet die An-

frage zur Automaten-Umgebung weiter, die diese entgegen nimmt. Die Instanz-Management Komponente befindet sich nun zunächst im Leerlauf. Die Automaten-Umgebung beginnt mit der Verarbeitung der Grammatik. Darüber wird auch die Instanz-Management Komponente benachrichtigt. Falls diese Messages empfangen hat (Verzweigungsknoten *Messages empfangen?* in der Abbildung) sendet sie diese an die Automaten-Umgebung. In der Automaten-Umgebung wird der Automat ausgeführt und parallel dazu werden Messages entgegen genommen. Die Parallelität wird in der Abbildung durch den Parallelisierungsbalken vor *Messages entgegennehmen* und *Automat ausführen* dargestellt. Nach dem Ende dieser parallelen Aktionen in der Automaten-Umgebung ist dort die Verarbeitung beendet. Es wird eine Rückmeldung an die Instanz-Management Komponente gegeben, womit die parallele Aktivität von Instanz-Management Komponente und Automaten-Umgebung beendet ist. Dies wird durch den Synchronisierungsbalken bei der Instanz-Management Komponente dargestellt. Damit ist die Verarbeitung der Anfrage abgeschlossen.

6.2. Untersuchung des Ergebnisses

Anhand des Beispielablaufes aus dem vorigen Abschnitt wird nun untersucht, inwiefern die Ziele der Arbeit erreicht wurden. Die generellen Ziele der Arbeit aus der Einleitung wurden erfüllt. Die Instanz-Management Komponente ist dazu in der Lage, verschiedene Instanzen eines Automaten zu verwalten und auf unterschiedliche Regionen der Amazon Cloud zu verteilen. Sie kann zudem Instanzen eines Automaten entfernen. Die Auswahl der Region geschieht dabei, wie gefordert, über die in der Grammatik verwendeten Web Services. Zudem wird der Automat in einer generischen Form bereitgestellt, wodurch dieser prinzipiell beliebig viele verschiedene Grammatiken verarbeiten kann. Die Skalierbarkeit des Automaten wird über die Automaten-Umgebung sichergestellt. Die Skalierung selbst wird dabei vollständig von der Amazon Infrastruktur übernommen. Diese kümmert sich auch darum, überflüssige Rechenkapazitäten wieder freizugeben. Die Automaten-Umgebung selbst wird von der Instanz-Management Komponente entfernt, sobald die Automaten-Umgebung nicht mehr benötigt wird.

Auch den Anforderungen, die in Kapitel 3 formuliert wurden, wird durch die vorliegende Implementierung nachgekommen. Die Anforderungen A1 (Regionen Berechnung), A2 (Verwaltung von verschiedenen Instanzen eines Automaten), A6 (Generischer Automat), A9 (Effizienter Umgang mit Ressourcen) und A10 (Skalierbarkeit) sind bereits durch die allgemeinen Ziele abgedeckt. Anforderung A3 (Ausführung des Automaten möglich) wird durch die *Message Correlation* Komponente der Instanz-Management Komponente und durch die Automaten-Umgebung erfüllt. Hierbei kann ein Automat gestartet werden und während der Ausführung kann dieser mit Messages versorgt werden. Durch die *Web Service Registry* Komponente wird Anforderung A4 (Überblick über Web Services) erfüllt, da diese Informationen zu sämtlichen Web Services speichert. Die geforderte Verwaltung der RRS-Instanzen aus Anforderung A5 wird durch die Komponente *RRS Verwaltung* implementiert. Anforderung A7 (Garantierte Verarbeitung einer Anfrage) wird durch die Speicherung der Anfragen in einer persistenten Warteschlange erfüllt. Zudem werden die Anfragen im Fehlerfall wieder dort gespeichert. In Anforderung A8 wird eine optimale Verteilung der Automaten auf die

verschiedenen Regionen gefordert. Die Anforderung ist bezüglich der verwendeten Metrik erfüllt. Für jede Region wird eine Bewertung erstellt, in der die Kosten der Web Service Aufrufe bewertetet werden. Die Kosten werden durch die Auswahl der Region mit der besten Bewertung minimiert.

6.3. Alternative Umsetzungsmöglichkeiten

Wie im vorigen Abschnitt gezeigt, wurden die Ziele und Anforderungen der Arbeit durch die Implementierung erfüllt. An dieser Stelle werden alternative Umsetzungsmöglichkeiten für gewisse Probleme diskutiert, die Nachteile der vorliegenden Implementierung beheben würden, jedoch andere Probleme mit sich bringen würden.

Ein Problem stellt die lange Zeitspanne dar, die für die Bereitstellung einer Elastic Beanstalk Anwendung benötigt wird. Wenn eine neue Automaten-Umgebung erzeugt wird, vergeht häufig eine Zeit von 5 - 10 Minuten, bis die Anwendung tatsächlich bereitsteht. Die genaue Dauer unterscheidet sich bei jedem Bereitstellungsvorgang. Für die Verarbeitung einer Anfrage bedeutet dies eine ebenso lange Wartezeit. Eine solche Verzögerung bei der Bearbeitung von Anfragen ist nicht wünschenswert. Um dies zu vermeiden, könnte in jeder Region standardmäßig eine Automaten-Umgebung betrieben werden, die nie gelöscht wird. Hierbei würden unter Umständen aber lange Leerlauf-Phasen für eine Automaten-Umgebung entstehen, wenn die Automaten-Umgebung einer Region eigentlich nicht benötigt wird. Dies wäre ein Widerspruch zu der Anforderung, dass möglichst viele Ressourcen gespart werden sollen. Letztendlich wurde in der Implementierung die erste Lösung gewählt, da über die Dauer der Verarbeitungszeit in den Anforderungen keine Vorgaben gemacht wurden.

Ein weiterer Punkt, der diskutiert werden kann, ist das Vorgehen bei der Berechnung der Region. In der verwendeten Implementierung wird die Zugriffshäufigkeit auf Web Services analysiert. Außerdem wird die Latenz durch die Strecke bewertet, die bei einem Web Service Aufruf zurückgelegt wird. Ein Aspekt der dabei nicht berücksichtigt wird, ist die Datenmenge, die die Web Services transferieren. Diese ist nicht ohne Weiteres ermittelbar. Die Latenz könnte ebenfalls effizienter bestimmt werden, als über die reine Entfernung der Rechenzentren, in denen die Web Services zur Verfügung stehen. Die Latenz ist unter anderem abhängig von der tatsächlichen Leitungslänge und der Qualität der Leitung sowie der Stationen, über die ein Paket geschickt wird. Um hierbei eine zuverlässige Abschätzung zu treffen, müssten die Latenzen zwischen den einzelnen Rechenzentren regelmäßig über einen längeren Zeitraum gemessen werden. Eine einzelne Messung einer Latenz zu verwenden, wäre nicht sinnvoll, da diese nur eine Momentaufnahme darstellt und stark variieren kann. Die absolute Distanz zweier Rechenzentren hat jedoch immer einen Einfluss auf die Latenz, auch wenn sie nicht die einzige Einflussgröße ist.

Die Skalierung der Automaten-Umgebung könnte ebenfalls anders umgesetzt werden. Würden die Komponenten wie Load-Balancing und Auto Scaling in der Instanz-Management Komponente implementiert werden, hätte man hierauf besseren Zugriff. Des Weiteren benötigt es weniger Zeit, eine Elastic Beanstalk Anwendung ohne diese Komponenten zu starten, da hierbei nur eine EC2-Instanz gestartet und konfiguriert werden muss. Damit könnte auch

das zuerst beschriebene Problem entschärft werden. Jedoch erfordert es erheblichen Aufwand, diese Komponenten selbst mit der Zuverlässigkeit und Qualität zu implementieren, die durch AWS gewährleistet wird. Daher wurde die Skalierung der Anwendungen der Amazon Infrastruktur überlassen.

7. Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit abschließend zusammengefasst werden. Außerdem wird ein Ausblick gegeben, welche Aufgaben in der Zukunft noch zu bewältigen sind.

7.1. Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde eine Instanz-Management Komponente für unifizierte Service-Kompositionen entwickelt, die in der Lage ist, die Instanzen eines endlichen Automaten zur Ausführung von Service-Kompositionen in der Amazon Cloud zu verwalten und diese sinnvoll über verschiedene Cloud-Regionen zu verteilen. Zudem wurde eine Umgebung für den Automaten entwickelt, die dessen Betrieb in der Cloud erlaubt.

Zu Beginn der Arbeit wurden zunächst mehrere Anbieter von Cloud Services evaluiert, um daraus das Angebot zu wählen, welches am besten zur Erfüllung der gestellten Anforderungen geeignet ist. Hierbei wurden die Anbieter bezüglich mehrerer Kriterien verglichen, was zur Entscheidung für das Cloud Angebot Amazon Web Services geführt hat.

Danach wurde anhand der Anforderungen ein Konzept für die Instanz-Management Komponente erstellt. Hierbei wurde festgestellt, dass die geforderten Funktionalitäten sich nur schwer in einer einzigen zentralen Komponente realisieren lassen. Stattdessen wurde beschlossen, bestimmte Funktionalitäten zum Automaten auszulagern. Der Automat und die zusätzlich benötigten Komponenten wurden in einer Anwendung zusammengefasst, die als Automaten-Umgebung bezeichnet wurde. Diese verwaltet den Zustand des Automaten und versorgt den Automaten mit Anfragen. Die Instanz-Management Komponente kümmert sich dagegen hauptsächlich um die Verwaltung der verschiedenen Automaten-Umgebungen. Eine weitere Aufgabe der Instanz-Management Komponente ist die Verteilung der Automaten-Umgebungen auf verschiedene Regionen, wofür ein Algorithmus entwickelt wurde. Dieser berücksichtigt die Web-Services einer Kompositionsgrammatik und bezieht die Verwendungshäufigkeit der Web Services sowie die Entfernung eines Web Services zu einer Region mit ein.

Anschließend wurden Implementierungsdetails beider Anwendungen und ihrer Komponenten erläutert. Dabei wurde ein besonderer Fokus auf das Zusammenspiel der beiden Anwendungen und auf die Integration der Anwendungen in die Amazon Cloud gelegt. Abschließend wurde in einer Evaluation festgestellt, dass die Anforderungen der Arbeit erfüllt wurden, jedoch wurden Design-Entscheidungen angesprochen, die noch weitere Optimierungen benötigen.

7.2. Ausblick

In Abschnitt 6.3 wurden bereits einige Funktionen der Anwendungen angesprochen, die mit einem gewissen Aufwand erweitert werden können.

Für die Optimierung der Instanz Verteilung auf verschiedene Regionen könnten in einem längeren Betrieb Daten über die Verwendung der Web Services gesammelt werden. Hierbei könnten beispielsweise die Datenmengen analysiert werden, die ein Web Service bei einem Aufruf durchschnittlich sendet. Dadurch könnten Web Services, die große Datenmengen benötigen, stärker gewichtet werden, als solche, die auf geringe Datenmengen zugreifen. Zudem könnten über einen längeren Zeitraum Daten über die gemessenen Latenzen zwischen den Regionen gesammelt werden. Diese könnten anstatt der reinen Entfernung als Metrik für die Entfernung der Regionen zueinander verwendet werden. Zudem könnten die Daten dynamisch aktualisiert werden, wenn die Latenzen zur Laufzeit stetig überwacht und analysiert werden würden.

Auch die Skalierungsfunktion bietet noch Raum für Optimierungen. Die Amazon Services zur Skalierung bieten grundsätzlich viele Möglichkeiten zur Konfiguration, zum Beispiel, ab welchen Schwellen neue Instanzen erzeugt werden und wie viele Instanzen prinzipiell verwendet werden sollen. Hierbei könnte evaluiert werden, welche Einstellungen für die Skalierung einen optimalen Kompromiss aus Leistung und Einsparung von Ressourcen bieten. Hierzu müssten verschiedene Konfigurationen in Testszenarien überprüft werden. Zudem wurde das Zeitintervall, nach der eine Automaten-Umgebung im Falle von Inaktivität vollständig gelöscht wird, fest auf eine Zeit von 15 Minuten gesetzt. An dieser Stelle könnte untersucht werden, ob dieses Zeitintervall eine sinnvolle Länge besitzt. Ein weiterer Ansatz wäre eine dynamische Berechnung dieser Zeitspanne, wobei hierfür zunächst Grundlagen geschaffen werden müssten, auf denen die Berechnung basieren sollte.

A. Anhang

```
</grammar>

        <nonTerminals>

                <nonTerminal>
                        <name> D1 </name>
                        <type> RRS </type>
                        <input>
                                <value> 2 </value>
                        </input>
                        <output>
                                <reference> X </reference>
                        </output>
                </nonTerminal>

                ...

        </nonTerminals>

        <nonTerminalTypes>

                <nonTerminalType name="RRS">
                        <wsa:EndpointReference>
                                <wsa:Address>
                                        http://95.208.155.213:8081/RRS/services/RRSPort?wsdl
                                </wsa:Address>
                        </wsa:EndpointReference>
                        <operation>insert</operation>
                 <wsdl:binding>RRSServiceSoapBinding</wsdl:binding>
                 <wsdl:portType>RRS</wsdl:portType>
                 <wsdl:port>RRSPort</wsdl:port>
                 <wsdl:service>RRSService</wsdl:service>
                 <namespace>http://default\_package/</namespace>
                </nonTerminalType>

                ...

        </nonTerminalTypes>

        <terminals>

                <terminal>
                        <name> s1 </name>
                </terminal>

                ...
```

```
        </terminals>



        <rules>

                <rule>
                        <LHS>
                                <nonTerminalRef> S </nonTerminalRef>
                        </LHS>
                        <RHS>
                                <nonTerminalRef> D1 </nonTerminalRef>
                                <nonTerminalRef> D2 </nonTerminalRef>
                                <nonTerminalRef> D3 </nonTerminalRef>
                                <nonTerminalRef> X </nonTerminalRef>
                        </RHS>
                </rule>

                ...

        </rules>

        <start>
                <nonTerminalRef> S </nonTerminalRef>
        </start>

</grammar>
```

Literaturverzeichnis

[ACHM04] G. Alonso, F. Casati, K. Harumi, V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 2004. ISBN: 978-3-642-07888-0. (Zitiert auf Seite 12)

[Amaa] Amazon. Amazon Linux AMIs. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonLinuxAMIBasics.html. [Online; abgerufen am 18.09.2013]. (Zitiert auf Seite 42)

[Amab] Amazon. Amazon Web Services Documentation. http://aws.amazon.com/documentation/. [Online; abgerufen am 10.09.2013]. (Zitiert auf den Seiten 21 und 37)

[Amac] Amazon. Architectural Overview. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.concepts.architecture.html. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Amad] Amazon. AWS SDK for Java API Reference. http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/index.html. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 22)

[Amae] Amazon. How Elastic Load Balancing Works. http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/SvcIntro_HowELBWorks.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf den Seiten 22 und 38)

[Amaf] Amazon. Introduction to Amazon S3. http://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 39)

[Amag] Amazon. Kostenloses Nutzungskontingent für AWS. http://aws.amazon.com/de/free/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Amah] Amazon. Kostenloses Nutzungskontingent für AWS. http://aws.amazon.com/de/grants/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Amai] Amazon. Pricing. http://aws.amazon.com/ec2/#pricing. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 40)

[Amaj] Amazon. Produkte und Services nach Region. http://aws.amazon.com/de/about-aws/globalinfrastructure/regional-product-services/. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 40)

[Amak] Amazon. Supported AWS Services. http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/supported_services.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 37)

[Amal] Amazon. What Is Amazon CloudWatch. http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsCloudWatch.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 37)

[Amam] Amazon. What is Amazon EC2? http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf Seite 37)

[Aman] Amazon. What is Amazon Simple Queue Service? http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/Welcome.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf den Seiten 22 und 39)

[Amao] Amazon. What is Auto Scaling? http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/WhatIsAutoScaling.html. [Online; abgerufen am 19.09.2013]. (Zitiert auf den Seiten 23 und 38)

[Amap] Amazon. What Is AWS Elastic Beanstalk and Why Do I Need It? http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html. [Online; abgerufen am 10.09.2013]. (Zitiert auf den Seiten 22 und 38)

[clo] Bundesamt für Sicherheit in der Informationstechnik. Cloud Computing Grundlagen. https://www.bsi.bund.de/DE/Themen/CloudComputing/Grundlagen/Grundlagen_node.html. [Online; abgerufen am 19.10.2013]. (Zitiert auf Seite 11)

[DJMZ05] W. Dostal, M. Jeckle, I. Melzer, B. Zengler. *Service-orientierte Architekturen mit Web Services*. Elsevier, 2005. ISBN: 978-3-8274-1457-1. (Zitiert auf Seite 12)

[GL12] K. Görlach, F. Leymann. Dynamic Service Provisioning for the Cloud. In *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*, SCC '12, S. 555–561. IEEE Computer Society, Washington, DC, USA, 2012. doi: 10.1109/SCC.2012.30. URL http://dx.doi.org/10.1109/SCC.2012.30. (Zitiert auf den Seiten 16, 17 und 28)

[GLC13] K. Görlach, F. Leymann, V. Claus. Unified Execution of Service Compositions. *IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013)*, Kauai, Hawai, December 16-18, 2013. (Zitiert auf den Seiten 9 und 14)

[Gooa] Google. Available Regions & Zones. https://developers.google.com/compute/docs/zones#available. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Goob] Google. Google App Engine. https://cloud.google.com/products/app-engine. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Gooc] Google. Google App Engine Billing FAQ. https://developers.google.com/appengine/kb/billing#discount. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 24)

[Good] Google. Google App Engine Pricing. https://cloud.google.com/pricing/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 24)

[Gooe] Google. Google Cloud Platform. https://cloud.google.com/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 21)

[Goof] Google. Google Compute Engine. https://cloud.google.com/products/compute-engine. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Goog] Google. Google Compute Engine Pricing. https://cloud.google.com/pricing/compute-engine. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 24)

[Gooh] Google. Java Service APIs. https://developers.google.com/appengine/docs/java/apis. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Gör13] K. Görlach. A Generic Transformation of Existing Service Composition Models to a Unified Model. Technischer Bericht Informatik 2013/01, Universität Stuttgart, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, 2013. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2013-01&engl=0. (Zitiert auf Seite 13)

[HB] H. Haas, A. Brown. Web Services Glossary. http://www.w3.org/TR/ws-gloss/. W3C.[Online; abgerufen am 21.09.2013]. (Zitiert auf Seite 12)

[JJ] D. Jordan, J. Jordan. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. OASIS.[Online; abgerufen am 23.09.2013]. (Zitiert auf Seite 9)

[MG11] P. Mell, T. Grance. The NIST definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011. (Zitiert auf Seite 11)

[Mica] Microsoft. API and Schema References for Windows Azure. http://msdn.microsoft.com/en-us/library/windowsazure/ff800682.aspx. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Micb] Microsoft. Free Trial. https://www.windowsazure.com/de-de/pricing/free-trial/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Micc] Microsoft. How to Create and Deploy a Cloud Service. http://www.windowsazure.com/en-us/manage/services/cloud-services/how-to-create-and-deploy-a-cloud-service/. [Online; abgerufen am 17.10.2013]. (Zitiert auf Seite 23)

[Micd] Microsoft. Messaging. http://www.windowsazure.com/de-de/services/messaging/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Mice] Microsoft. Virtual Machines. http://www.windowsazure.com/en-us/documentation/services/virtual-machines/?fb=de-de. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Micf] Microsoft. Virtuelle Computer – Preisdetails. http://www.windowsazure.com/de-de/pricing/details/virtual-machines/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Micg] Microsoft. What is a cloud service? https://www.windowsazure.com/en-us/manage/services/cloud-services/what-is-a-cloud-service/?fb=de-de. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Mich] Microsoft. Windows Azure Documentation. http://www.windowsazure.com/en-us/documentation/?fb=de-de. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 21)

[Mici] Microsoft. Windows Azure in Education. http://www.windowsazure.com/en-us/community/education/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[Micj] Microsoft. Windows Azure Trust Center. https://www.windowsazure.com/en-us/support/trust-center/privacy/. [Online; abgerufen am 10.09.2013]. (Zitiert auf Seite 23)

[PA06] M. Pesic, W. Aalst. A Declarative Approach for Flexible Business Processes Management. In J. Eder, S. Dustdar, Herausgeber, *Business Process Management Workshops*, Band 4103 von *Lecture Notes in Computer Science*, S. 169–180. Springer Berlin Heidelberg, 2006. doi:10.1007/11837862_18. (Zitiert auf Seite 9)

[Sch08] U. Schoening. *Theoretische Informatik - kurz gefasst*. Spektrum, 2008. ISBN: 978-3827418241. (Zitiert auf Seite 13)

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift