Institut für Architektur von Anwendungssystemen

Universitätsstraße 38

70569 Stuttgart

Deutschland

Bachelorarbeit Nr. 32

Management von Cloud Applikationen in OpenTOSCA

– Cloud Application Management in OpenTOSCA –

von Christian Endres

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. F. Leymann

Betreuer/in: M.Sc. Wirt.-Inf. Alexander Nowak

Beginn am: 01.11.2012

Beendet am: 03.05.2013

CR-Nummer: J.0, K.6.2

Kurzfassung

Management von Cloud Applikationen in OpenTOSCA

– Cloud Application Management in OpenTOSCA –

von Christian Endres

„Topology and Orchestration Specification for Cloud Applications“ – kurz TOSCA – bietet eine Sprache, um Service-Komponenten und ihre Beziehungen untereinander zu beschreiben. Weiterhin können Prozesse wie das Erstellen oder Modifizieren eines Services beschrieben werden. Installation und Management der Cloud Applikationen sind dabei portabel und interoperabel beschrieben. Dies ermöglicht es, die Cloud Applikationen bei verschiedenen Cloud Anbietern zu betreiben und das Management der Anwendungen zu automatisieren. (1)

OpenTOSCA (2) ist eine an der Universität Stuttgart entwickelte Laufzeitumgebung für TOSCA, deren erste, rudimentäre Version im Herbst 2012 fertiggestellt wurde. Diese erste Open-Source-Referenzimplementierung wird nun in verschiedenen Bereichen erweitert und weiterentwickelt.

Diese Bachelorarbeit beschreibt, wie in OpenTOSCA Instanzen von Cloud Applikationen verwaltet und durch Managementpläne manipuliert werden. Weiterhin sieht der praktische Teil der Bachelorarbeit die Implementierung dieser Funktionalität in OpenTOSCA und einer grafischen Oberfläche vor, die diese Funktionalität dem Nutzer verfügbar macht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Systemadministratoren wenden viel Zeit auf, um Applikationen auf eigenen Server zu installieren und zu warten. Die Installation und Konfiguration geschieht manuell. Dasselbe gilt für die Wartung und Änderung der Applikationen, sowie deren Deinstallation. Häufig wird dies durch Skripte erleichtert, die wiederkehrende Tätigkeiten teilweise oder vollständig automatisieren.

Aktuell besteht der Trend, Hardwarekosten einzusparen und die eigene Infrastruktur in die Cloud auszulagern. „Topology and Orchestration Specification for Cloud Applications“ (1) (nachfolgend TOSCA genannt) verfolgt diesen Trend und beschreibt unter anderem die Möglichkeit, Automatisierungen vorzunehmen.

Daher entstand das Open-Source-Projekt OpenTOSCA (2), welches TOSCA verstehen und „ausführbar“ machen soll. TOSCA definiert für die Automatisierung Managementpläne, die Workflows beschreiben. Das Projekt OpenTOSCA realisiert dies durch „OASIS Web Services Business Process Execution Language”-Pläne (nachfolgend BPEL-Pläne genannt). Diese müssen jedoch durch Programme dritter Anbieter manuell gestartet werden. Auch erkennt OpenTOSCA nicht, welche Ergebnisse diese Managementpläne erzeugen.

1.2 Problemstellung

Das Datenformat in der TOSCA Spezifikation vorgestellte „Cloud Service Archive“ (1) (nachfolgend CSAR genannt), welches die Rohdaten einer Cloud Applikation beinhaltet. Die Installation geschieht durch Managementpläne, die TOSCA definiert oder referenziert. OpenTOSCA installiert BPEL-Pläne dafür auf einen „WSO2 Business Process Server“ (3) (nachfolgend BPS genannt). Jedoch ist keine Automatisierung implementiert, um Managementpläne mit ihren benötigten Eingabeparametern zu starten und deren Ergebnis auszuwerten. Um eine CSAR zu installieren beziehungsweise zu instanziieren, muss man einen entsprechenden Managementplan mithilfe Programme dritter Anbieter ausführen.

Die grafische Oberfläche (nachfolgend alte GUI genannt) bietet die Möglichkeit eine CSAR in OpenTOSCA zu laden. Die Verarbeitung einer CSAR in OpenTOSCA beschränkt sich größtenteils darauf, die enthaltenen TOSCA-Daten teilweise zu analysieren, Implementation Artifacts (1) auf einem Apache Tomcat (4) und BPEL-Pläne auf einem BPS zu installieren.

Daher sollen Konzepte entwickelt werden, welche die Ausführung von BPEL-Plänen und die Verwaltung der produzierten Ergebnisse beschreiben. Anschließend sollen diese Konzepte in OpenTOSCA integriert werden.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, die Ausführung und Überwachung der durch TOSCA definierten Managementpläne in die Laufzeitumgebung von OpenTOSCA zu integrieren und über eine GUI anzubieten.

Hierzu sollen Konzepte über das Aufrufen Managementplänen erarbeitet werden. Dieses Ziel spaltet sich thematisch jedoch in zwei Bachelorarbeiten, die zeitlich überlappend bearbeitet werden. Diese Bachelorarbeit beschäftigt sich damit, wie Managementpläne aktiviert und deren Ergebnisse verwaltet werden können. Weiterhin soll diese Funktionalität durch eine GUI intuitiv ausführbar gemacht werden.

Ausgenommen davon ist jedoch die Anbindung einer Software wie dem BPS, die die Managementpläne tatsächlich ausführt. Da diese Funktionalität für den Erfolg des praktischen Teils der Bachelorarbeit benötigt wird, soll diese Funktionalität simuliert werden.

OpenTOSCA soll die Verwaltung der Instanzen von Cloud Applikationen und deren Lebenszyklus beherrschen. Weiterhin soll die Ausführung und die Ergebnisse von Managementplänen grafisch dargestellt werden. Eine Historie soll einen Überblick über ausgeführte Managementpläne bereitstellen. Dabei sollen sowohl die Eingabeparameter, als auch das Ergebnis grafisch dargestellt werden. Der praktische Teil der Bachelorarbeit sieht die Implementierung der Funktionalität in OpenTOSCA und einer Weboberfläche vor.

1.4 Über dieses Dokument und den Rahmenbedingungen

Diese Bachelorarbeit beschäftigt sich mit dem Standard TOSCA. TOSCA befindet sich noch in Entwicklung und liegt in keiner finalen Fassung vor. OpenTOSCA ist eine

Referenzimplementierung dieses Standards und befindet sich entsprechend auch in der Entwicklungsphase, die diese Bachelorarbeit überdauert. OpenTOSCA ist in Java programmiert und verwendet BPEL-Pläne zur Verwaltung der Applikationen.

Diese Themenfelder sind in der Informatik angesiedelt. Die Sprache der Informatik ist Englisch. Daher ergibt sich für ein in der deutschen Sprache verfasstes Dokument die Schwierigkeit, Fachbegriffe zu verwenden. Für viele englische Begriffe gibt es keine oder nur unzureichende Übersetzungen. Auch sind in diesem Dokument viele Begriffe aus dem TOSCA-Universum und Namen von Klassen, Komponenten und Produkten verwendet. Um vor allem den Lesefluss nicht zu unterbrechen, oder zwanghafte Wortdefinitionen oder Umschreibungen zu vermeiden, werden solche Begriffe und Namen bewusst wie deutsche Wörter verwendet.

An dieser Stelle sei auf das Glossar mit der Kapitelnummer 9.1 hingewiesen. Im Glossar sind alle neu eingeführten und die wichtigsten der verwendeten TOSCA-Begriffe aufgeführt und erklärt.

1.5 Struktur

Der theoretische Teil der Bachelorarbeit wird durch dieses Dokument abgedeckt. Hier sind die einzelnen Punkte der Aufgabenstellung analysiert, Konzepte definiert und die Implementierung abstrakt beschrieben.

Der praktische Teil der Bachelorarbeit besteht aus der tatsächlichen Implementierung der Funktionalität in OpenTOSCA und einer Weboberfläche. Daher besteht das Ergebnis der Bachelorarbeit aus diesem Dokument, sowie einer Version von OpenTOSCA, in der die Implementierung der Funktionalität integriert ist.

2 TOSCA Grundlagen

TOSCA beschreibt Cloud Applikationen, die in einer beliebigen Cloud durch Managementpläne instanziiert und verwaltet werden können. Sobald durch TOSCA beschriebene, reale Objekte generiert oder viel mehr installiert werden, sind dies Instanzen.

2.1 Templates und Instanzen

Für OpenTOSCA ist das Verhalten des Inhalts einer CSAR durch die enthaltenen TOSCA-Daten definiert. Eine Instanz einer CSAR ist demnach eine Instanz der spezifischen TOSCA-Daten.

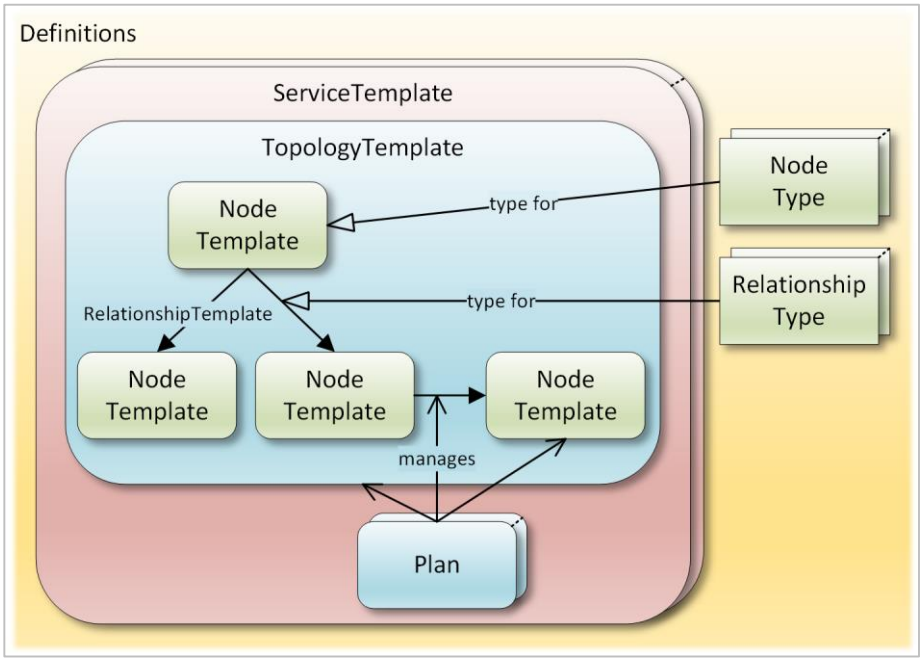

Abbildung 1: Grafische Darstellung von Teilen von TOSCA

TOSCA beschreibt Node Templates, Relationship Templates, Topology Templates und Service Templates, welche Vorlagen für konkrete Service-Instanzen sind. Die Ziele der Bachelorarbeit legen den Fokus auf die Sichtweise des Nutzers, der mithilfe von OpenTOSCA einen oder mehrere Services installieren und verwalten möchte. Daher sind für die Instanz eines Service Templates die Boundary Definitions von Bedeutung.

Die Boundary Definitions definieren unter anderem, wie ein Nutzer mit einer Service Template-Instanz über Managementpläne interagieren kann. Hierzu steht in der Spezifikation: „BoundaryDefinitions: This OPTIONAL element specifies the properties the Service Template exposes beyond its boundaries, i.e. properties that can be observed from outside the Service Template. […]“ (1). Neben spezifischen Eigenschaften sind in den Boundary Definitions Interfaces definiert: „Interfaces: This OPTIONAL element specifies the interfaces with operations that can be invoked on complete service instances created from the Service Template.” (1) Innerhalb eines Interfaces kann für eine Operation ein Plan definiert werden. Ein Nutzer interagiert mit einer Instanz nur über einen Managementplan, wenn dieser Managementplan durch ein Interface-Element „öffentlich sichtbar“ deklariert ist.

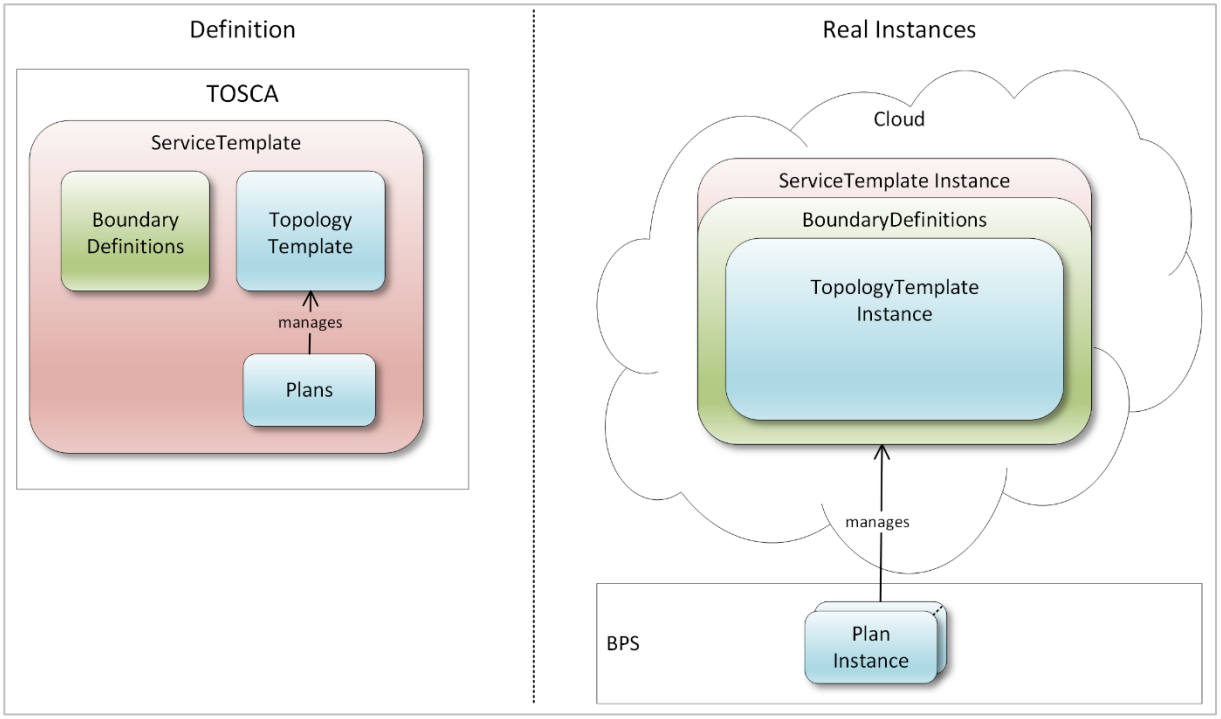

Abbildung 2: Gegenüberstellung von TOSCA und Instanzen in der realen Welt

Für den Nutzer ist hauptsächlich interessant, welchen Service (auch die Komposition aus mehreren untergeordneten Services) eine CSAR oder vielmehr eine Instanz einer CSAR bereitstellt. Die feingranulare Topologie ist hingegen der Level, auf dem sich ein Entwickler bewegt. Daher werden aus der Nutzersicht und damit für diese Bachelorarbeit instanziierte Cloud Applikation der CSAR und damit die Instanzen einer CSAR betrachtet (nachfolgend CSAR-Instanz genannt). Dies ist keine Einschränkung von TOSCA, da eine CSAR-Instanz mehrere Instanzen einer oder mehrere Service Templates umfassen kann.

2.2 Managementpläne

TOSCA definiert Managementpläne als Metametamodell, um Managementoperationen einer CSAR-Instanz zu beschreiben. Managementpläne können durch verschiedene Workflow-Sprachen realisiert werden. Ein Managementplan ist durch das TOSCA-Element Plan definiert, welches einem Service Template zugeordnet ist. Hierzu steht in der TOSCA Spezifikation: „Plans: This element specifies the operational behavior of the service. […] “ (1). OpenTOSCA akzeptiert BPEL als Metamodell und installiert die in TOSCA angegebenen oder referenzierten BPEL-Pläne auf einen BPS.

BPEL-Pläne sind demnach die Modelle für konkrete Prozessinstanzen. Es ergibt sich eine getrennte Betrachtung der Managementpläne und BPEL-Plänen, da BPEL-Pläne ein konkretere Beschreibung von operationalem Verhalten sind, als Managementpläne.

Wie auch die Templates in TOSCA sind Managementpläne eine Beschreibung, aus der beliebig viele Instanzen generiert werden können. Aufgrund der Vielzahl und der Möglichkeit, mehrere Managementplaninstanzen parallel auszuführen, muss jede Managementplaninstanz für OpenTOSCA eindeutig identifizierbar sein. Dies gilt ganz besonders während die Managementplaninstanz lebendig ist und OpenTOSCA auf eine Antwort wartet.

Aufgrund der geforderten Historie über die Ausführung von einzelnen Managementplänen ergibt sich zusätzlich die Anforderung, dass eine Managementplaninstanz über das Ende ihres Lebenszyklus hinaus eindeutig identifizierbar bleibt.

2.3 Zusammenhang zwischen Managementplänen und CSAR-Instanzen

TOSCA definiert explizit zwei Typen, die einen Managementplan charakterisieren können. Beim Typ „build“ (nachfolgend wird durch den Begriff Build-Plan Installationsverhalten assoziiert) erstellt eine Managementplaninstanz eine Service-Instanz eines Service Templates: „[…] plans used to initially create a new instance of a service from a Service Template.“ (1).

Beim Typ „termination“ (nachfolgend wird durch den Begriff Termination-Plan Löschverhalten assoziiert) werden Service-Instanzen deinstalliert: „plans used to terminate the existence of a service instance.” (1).

Die TOSCA Spezifikation bezeichnet alle anderen Managementpläne als „modification plans in general“ (1).

Demnach muss eine vollständige CSAR mindestens einen Build-Plan und Termination-Plan beinhalten, die durch die Boundary Definitions dem Nutzer zur Interaktion angeboten werden.

Managementpläne sind im Service Template-Element von TOSCA definiert. Eine Prozessinstanz eines BPEL-Plans ist demnach direkt zugehörig zu der Service-Instanz des Service Templates, das den Managementplan definiert, welches den BPEL-Plan referenziert oder beinhaltet.

Wie in Kapitel 2.1 beschrieben, werden in dieser Bachelorarbeit komplette CSAR-Instanzen betrachtet, die durch die Boundary Definitions dem Nutzer angeboten werden. Daher muss ein Build-Plan eine funktionsfähige Komposition der möglichen Service-Instanzen installieren, die eine CSAR ermöglicht. Die im TOSCA-Namen enthaltene Orchestrierung kann der Nutzer durch Managementpläne auslösen, die weder Build- noch Termination-Pläne sind (aber solche nutzen und aktivieren können). Analog zu Build-Plänen müssen Termination-Pläne, die der Nutzer direkt aktivieren kann, die komplette CSAR-Instanz löschen. Daher muss ein Termination-Plan jede mögliche Komposition aus Services löschen können.

3 Ist- und Soll-Zustand

Die Bachelorarbeit ist Teil der Weiterentwicklung von OpenTOSCA. OpenTOSCA liegt in einem Entwicklungsstand vor, der durch ein Studienprojekt und Hiwi-Tätigkeiten erarbeitet wurde und weiterhin wird. Dieses Kapitel zeigt einen Abriss darüber, welche Funktionalität und Struktur in OpenTOSCA zu Beginn der Bachelorarbeit vorhanden ist und welche Ziele durch die Bachelorarbeit verfolgt werden.

3.1 Überblick über OpenTOSCA

Dieses Kapitel beschreibt den Entwicklungsstand auf dem die Bachelorarbeit aufsetzt.

3.1.1 Architektur

Die Architektur von OpenTOSCA ist an das Model-View-Control Pattern angelehnt. Die ContainerAPI entspricht dabei der View. Die OpenToscaControl übernimmt die Funktion der Control. Die Core, IAEngine, PlanEngine und ToscaEngine leisten die Geschäftslogik. Das Model ist im nachfolgenden Bild nicht explizit aufgeführt. OpenTOSCA nutzt OSGI und jede der abgebildeten Komponenten bietet einen oder mehrere Services an.

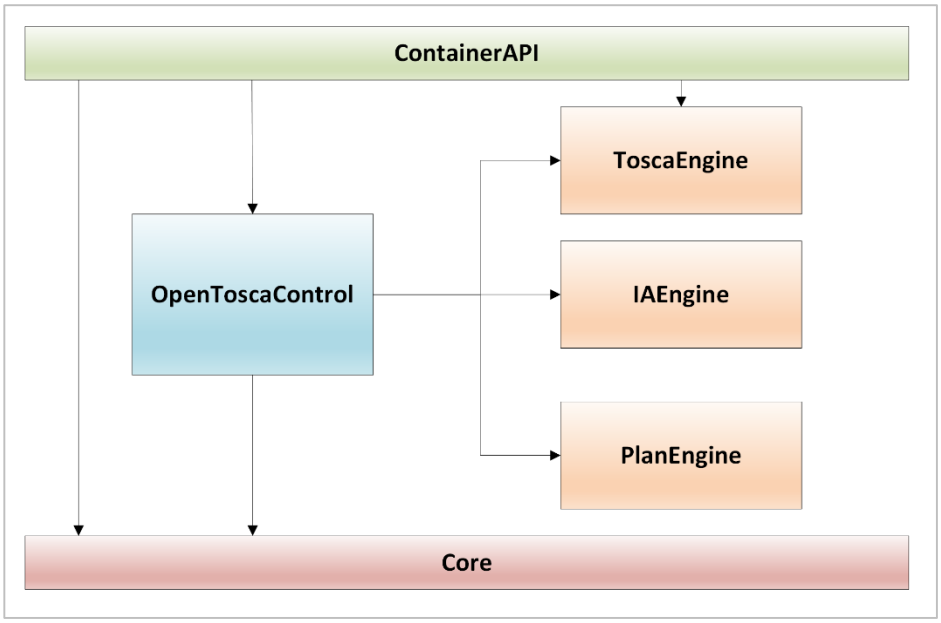

Abbildung 3: Grobarchitektur von OpenTOSCA

Die Komponenten und Zusammenhänge sind in den folgenden Kapiteln näher beschrieben.

3.1.2 ContainerAPI

Die ContainerAPI übernimmt in OpenTOSCA den Part der View. Hier impliziert der englische Begriff View unter Umständen zu viel, denn eine grafische Darstellung enthält die ContainerAPI nicht. Die ContainerAPI ist eine REST Schnittstelle, die mit Jersey in der Version 1.11 (5) implementiert ist. Hier werden die Funktionalität von OpenTOSCA und die Daten aus verschiedenen Komponenten nach außen kommuniziert.

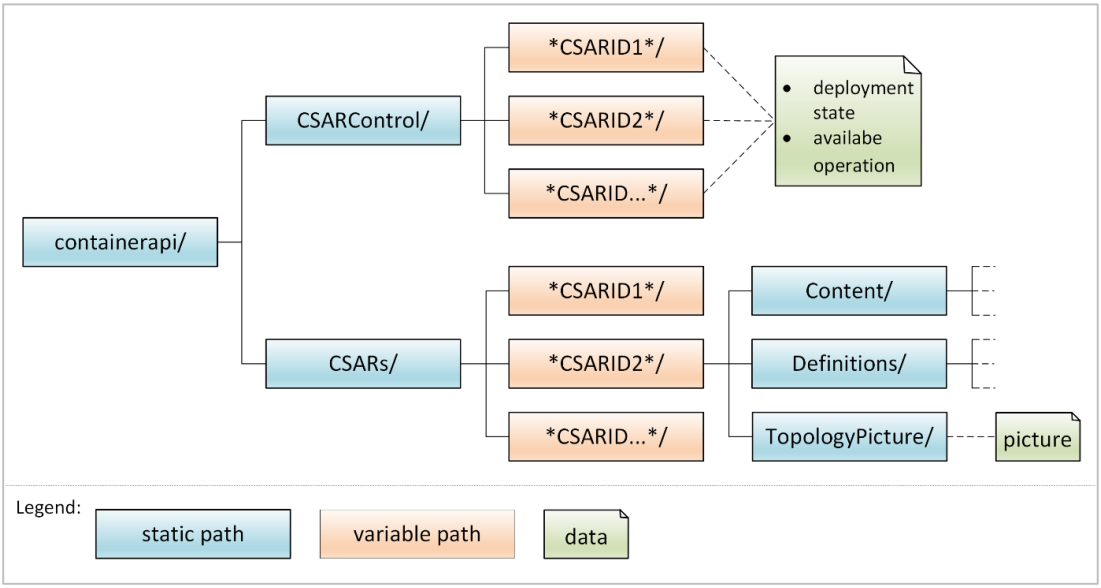

Abbildung 4: Ist-Zustand ContainerAPI

Obiges Bild zeigt vereinfacht, wie die Baumstruktur der ContainerAPI aufgebaut ist.

3.1.3 Core

Die Core entpackt ein geladenes CSAR-Archiv, verarbeitet dessen Metadaten und speichert die enthaltenen Dateien ab. Durch die Bachelorarbeit werden keine Änderungen an der Core vorgenommen. Daher entfällt eine nähere Betrachtung.

3.1.4 ToscaEngine

Die ToscaEngine analysiert die TOSCA-Dokumente, die in einer CSAR enthalten sind. Dies umfasst das Auffinden von importierten Dateien, das Nachvollziehen von Referenzen innerhalb von TOSCA und eine rudimentäre Überprüfung auf Korrektheit der TOSCA-Daten. Die Referenzen und referenzierten Daten werden abgespeichert, damit ein schnelleres Finden der Daten möglich ist. Weiterhin enthält die ToscaEngine einen Service zum serialisieren und deserialisieren von TOSCA-Dokumenten im XML-Format. Die durch die ToscaEngine

unterstützte Version von TOSCA ist die csprd01 vom 29.11.2012. Jedoch werden TOSCA-Daten nur soweit ausgewertet, wie es der Entwicklungsstand von OpenTOSCA benötigt.

3.1.5 Internes TOSAC-Modell

TOSCA-Daten werden in einem internen JAXB-Datenmodell gehalten. Die Klassen sind mit dem Programm XJC von JAXB (6) aus der XML Schema (7) von TOSCA generiert. Die durch das Datenmodell unterstützte Version von TOSCA ist die csprd01 vom 29.11.2012. Das Datenmodell sollte nur von der ToscaEngine verwendet werden, um die anderen Komponenten nicht in Abhängigkeit des Datenmodells zu bringen. In der Vergangenheit implementierte Komponenten nutzen jedoch noch das Modell. Die Abkopplung des Datenmodells durchzuführen, ist nicht Teil der Bachelorarbeit. Dennoch soll diese Richtlinie für die Entwicklung neuer Komponenten berücksichtigt werden.

3.1.6 IAEngine

Die IAEngine ist eine Komponente, welche die in den TOSCA-Dokumenten referenzierte Implementation Artifacts auf einen Tomcat installiert. Wie die Core wird diese Komponente durch die Bachelorarbeit nicht weiterentwickelt. Daher entfällt eine nähere Betrachtung.

3.1.7 PlanEngine

Die PlanEngine ist eine Komponente, welche die in den TOSCA-Dokumenten referenzierten BPEL-Pläne auf einem BPS installiert. Ziel dessen ist, dass die BPEL-Pläne als Web Service verwendbar sind. Wie die Core und die IAEngine wird diese Komponente durch die Bachelorarbeit nicht weiterentwickelt. Daher entfällt eine nähere Betrachtung.

3.1.8 GUI

Die GUI besteht aus zwei logischen Teilen. Zum einen die grafische Darstellung, bestehend aus einer „JavaServer Pages“ (nachfolgend JSP genannt) mit eingebundenem JavaScript-Code und CSS-Formatierungen. Der zweite Teil umfasst den Java-Code, der mit OpenTOSCA kommuniziert und die Daten für die Webseite zur Verfügung stellt.

3.1.8.1 GRAFISCHE OBERFLÄCHE

Die grafische Oberfläche hat zwei funktionale Ansichten. In beiden kann man CSARs in OpenTOSCA laden. Die erste Ansicht im Reiter Overview zeigt zudem eine Liste, in der man

Abbildung 5: Alte GUI in der Ansicht Overview

durch das Verzeichnis der entpackten CSAR navigieren kann und ein Bild der Topologie der CSAR. Wie im obigen Bild zu sehen ist, arbeitet diese GUI nicht fehlerfrei, da Teile der Ansicht „Deploy CSAR“ (die beiden Fortschrittsbalken, die beiden Buttons und das Dropdown-Feld links daneben) angezeigt werden.

Die zweite Ansicht eröffnet dem Nutzer die Möglichkeit, die drei Arbeitsschritte zum Verarbeiten einer CSAR manuell zu steuern. Diese einzelnen Schritte umfassen jeweils die Funktionalität der drei Engines: Analysieren der TOSCA-Daten in der CSAR, Installieren der Implementation Artifacts und Installieren der BPEL-Pläne.

Abbildung 6: Alte GUI in der Ansicht „Deploy CSAR“

Die dritte Ansicht About hat informellen Charakter: Sie zeigt die Namen und Email-Adressen der Entwickler an. Doch auch diese Ansicht zeigt dieselben Bestandteile der Ansicht „Deploy CSAR“ wie die Ansicht Overview, die eigentlich ausgeblendet werden müssten.

3.1.8.2 JAVA-IMPLEMENTIERUNG

Für den Datenaustausch zwischen der GUI und OpenTOSCA nutzt die GUI das Jersey-Paket „com.sun.jersey.api.client“, welches die Kommunikation mit der REST-Schnittstelle von OpenTOSCA ermöglicht. Diese Klasse der GUI nennt sich ContainerClient. Damit die gewünschten Daten auch in der grafischen Oberfläche abrufbar sind, enthält die GUI Servlets, die vom Browser des Nutzers angesprochen werden können.

Aufgrund der Implementierung müssen OpenTOSCA und der Webserver auf demselben System laufen: Um eine CSAR in OpenTOSCA zu laden, wird in der GUI die CSAR-Datei

ausgewählt, welche als Java InputStream in einem Servlet auf die lokale Festplatte geschrieben wird. Der ContainerClient gibt den lokalen Pfad an OpenTOSCA. OpenTOSCA sucht schlussendlich genau an dieser Stelle auf seinem eigenen System nach dieser Datei.

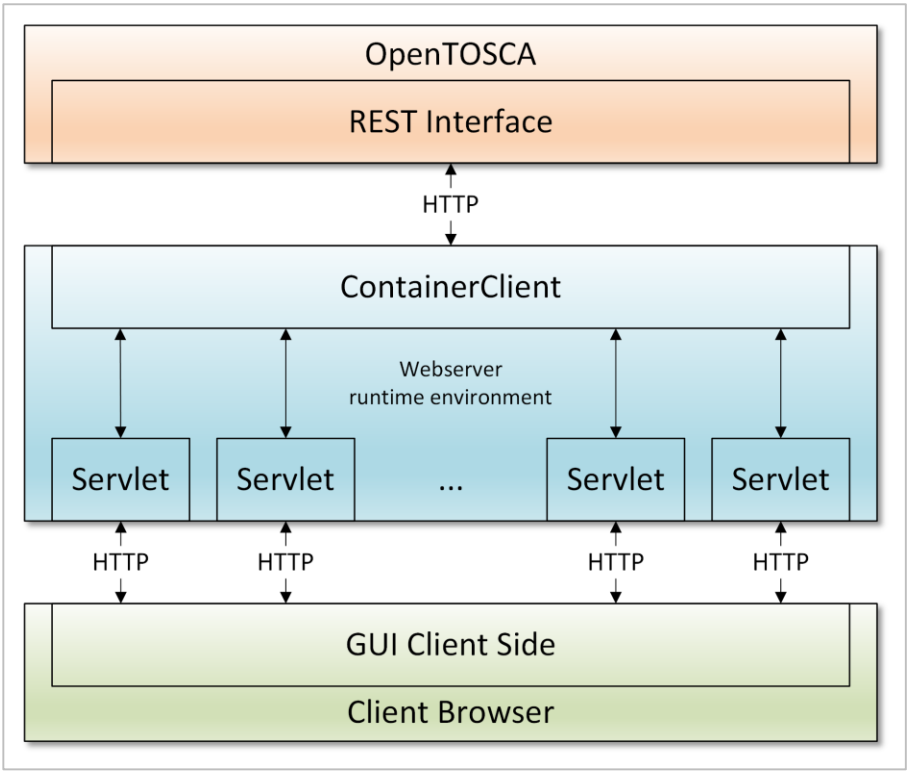

Abbildung 7: Kommunikation zwischen Client, alter GUI und OpenTOSCA

3.2 Anforderungsanalyse

Dieses Kapitel beschreibt abstrakt, welche Funktionalität für OpenTOSCA sich aus der Zielsetzung dieser Arbeit ableiten lassen. Die Gliederung der Unterkapitel entspricht dabei den einzelnen Arbeitspaketen der Implementierungsphase.

3.2.1 Verarbeitung der TOSCA-Daten

TOSCA ist ein sich in Entwicklung befindender Standard und OpenTOSCA ein TOSCA verstehender Prototyp. Daher versteht und verwendet OpenTOSCA TOSCA auch nur soweit, wie es die implementierte Funktionalität erfordert. Dementsprechend muss die ToscaEngine in OpenTOSCA weiterentwickelt werden, um in TOSCA-Dokumenten definierte Managementpläne zu verarbeiten.

3.2.2 Managementpläne aktivieren und Ergebnisse empfangen

OpenTOSCA verwendet BPEL-Pläne und einen BPS, um Managementpläne als Webservice ausführbar zu machen. Der nächste logische Schritt ist, die BPEL-Pläne durch OpenTOSCA zu aktivieren und das Ergebnis einer BPEL Prozessinstanz in OpenTOSCA zu empfangen. Diese Anforderung ist jedoch Teil einer separat vergeben Bachelorarbeit, die circa drei Monate nach dieser Bachelorarbeit gestartet ist. Diese Bachelorarbeit betrachtet, wie durch eine generische Schnittstelle zentral in OpenTOSCA Managementfunktionen von Service-Komponenten angesprochen werden können.

Nichtsdestotrotz wird diese Funktionalität benötigt, um die Ziele dieser Bachelorarbeit zu erreichen. Aufgrund der unterschiedlichen Startzeiten und der zwingenden Trennung zwischen den beiden Bachelorarbeiten, muss ein Mock-up (engl.: Attrappe, hier die Simulation einer noch nicht implementierten Funktionalität) entwickelt werden, der in einem BPS installierte BPEL-Pläne aktivieren und die entsprechenden Ergebnisse empfangen kann. Die Kommunikation soll über das SOAP-Protokoll realisiert werden. Die Mock-up-Komponente trägt in dieser Bachelorarbeit den Namen Servicebus.

3.2.3 Schreiben und Lesen von BPEL-Nachrichten

Um einen BPEL-Plan durch den Servicebus zu aktivieren, wird eine Request-Nachricht vom Servicebus an die Webservice-Adresse des BPEL-Plans geschickt. Entsprechend wird vom Servicebus die Response-Nachricht empfangen. Durch die Kommunikation über das SOAP-Protokoll wird eine Komponente benötigt, die SOAP Envelopes schreiben und verstehen kann. Dabei muss der Request mit den durch TOSCA definierten und durch den Nutzer eingegebenen Input-Parameter bestückt werden und der Response entsprechend der durch TOSCA definierten Output-Parameter verarbeitet werden.

3.2.4 Correlation zwischen Request und Response von BPEL-Plänen

Request- und Response-Nachrichten sind nicht zwingend zueinander zuordenbar. Daher muss ein Konzept entwickelt und implementiert werden, wie zweifelsfrei der Response einer Prozessinstanz dem vorhergegangenen Request zugeordnet werden kann. Weiterhin muss die einzelne Correlation unter allen anderen Correlations eindeutig sein, um sie in einer History zu speichern.

3.2.5 Verwaltung von CSAR-Instanzen

Build-Pläne erstellen CSAR-Instanzen, sowie CSAR-Instanzen durch einen Termination-Plan gelöscht werden. Daher benötigt OpenTOSCA eine Verwaltung der erzeugten CSAR-Instanzen.

3.2.6 History über Prozessinstanzen

OpenTOSCA soll nicht nur BPEL-Pläne aktivieren und deren Ergebnis verstehen können, sondern sich dies auch merken. Dabei soll OpenTOSCA jede Planinstanz – den Request und den entsprechenden Response – mit der CSAR und der CSAR-Instanz verbinden.

3.2.7 Erweiterung der REST-Schnittstelle

Die ContainerAPI soll die in den Kapiteln 3.2.1 bis 3.2.6 beschriebenen Funktionen und Daten zur Verfügung stellen. Weiterhin muss die ContainerAPI auch die Funktionalität bereitstellen, die durch die Erweiterung der GUI entstehen.

3.2.8 GUI

Da die alte GUI die neuen Ziele nicht abbildet, soll sie entweder erweitert oder ersetzt werden. Im Detail sollen folgende Funktionalitäten zusätzlich durch die GUI abgedeckt sein:

1. Verwaltung der Instanzen einer CSAR darstellen.
2. Anzeigen der ausführbaren Managementpläne einer CSAR-Instanz.
3. Aufrufen von Managementplänen durch die GUI.
4. Einfügen der benötigten Parameter eines Managementplans innerhalb der GUI.
5. Anzeigen von bereits ausgeführten Managementplänen.
6. Unterschiedliche Darstellung für noch aktive Prozessinstanzen und abgeschlossene Prozessinstanzen.
7. Darstellung des Fortschritts einer Prozessinstanz.
8. Darstellung der Input- und Output-Parameter einer abgeschlossenen Prozessinstanz.

4 Konzept

Dieses Kapitel widmet sich den Ideen und Gedanken zu den vorher beschriebenen Problemstellungen.

4.1 Modell der Managementpläne

Wie in Kapitel 3.1.5 beschrieben, existiert in OpenTOSCA ein internes Modell für die TOSCA-Daten. Dies soll jedoch nur von der ToscaEngine verwendet werden. Dadurch ergeben sich zwei mögliche grundsätzliche Vorgehensweisen: Entweder das vorhandene Datenmodell nutzen, oder ein neues erstellen.

Die ToscaEngine könnte die Daten traversieren und alle benötigten Informationen zur Verfügung stellen. Dies hätte zur Folge, dass das Interface des ToscaEngine-Services eine get-Methode für jede einzelne Information hat, wie es teilweise schon implementiert ist. Daher würde das Interface der ToscaEngine deutlich unübersichtlicher. Der Fokus der ToscaEngine liegt auch nicht auf der Traversierung von gespeicherten TOSCA-Daten, sondern auf der Verarbeitung der Daten. Daher würde sich hier anbieten den Service der ToscaEngine umzubauen, um die Funktionalitäten klar zu trennen. Es ist jedoch nicht Ziel der Bachelorarbeit, eine Datenhaltungsschicht für TOSCA-Daten zu erarbeiten. Weiterhin löst dies nicht den Bedarf der History, die in Kapitel 4.6 analysiert wird.

Die Alternative wäre ein zweites Datenmodell, welches die verarbeiteten Daten repräsentiert. Nachteil dessen ist, dass Daten – wie zum Beispiel den Namen und Typ eines Parameters eines Managementplans – redundant gespeichert sind. OpenTOSCA liest die TOSCA-Daten einer CSAR nur einmal und daher werden die Daten auch zu keinem Zeitpunkt aktualisiert. Daher und durch die geringe Datenmenge hat dieser Nachteil wenig Gewicht.

Dem gegenüber steht die Möglichkeit, das Modell nicht äquivalent zu TOSCA aufzubauen. Dies hätte den Vorteil, dass nur die für die Verarbeitung benötigten Daten gespeichert werden. Informationen über Managementpläne, die ein Nutzer nutzen kann, stehen in den Boundary Definitions. Die Informationen über die Parameter des Managementplans sind jedoch in einem Plan-Element zu finden. Andererseits sind Informationen über Policies in den Boundary

Definitions oder der tatsächliche Managementplan im PlanModel-Element des Plan-Elements nicht von Interesse. Durch eine Konsolidierung der Daten werden nur die benötigten Informationen in das neue Datenmodell gespeichert. Die Konsolidierung kann in der ToscaEngine während oder nach dem Auflösen der Referenzen innerhalb der TOSCA-Dokumente implementiert werden.

Ein weiterer Vorteil des zweiten Datenmodells wäre, dass es von mehreren Komponenten verwendet werden kann. OpenTOSCA soll beispielsweise die ausgeführten Managementpläne mit Eingabewerten und Ergebnissen speichern. Um dies zu erreichen, kann das Datenmodell zu den Parametern deren Wert abbilden.

Aufgrund der genannten Vor- und Nachteile fällt daher die Wahl auf das zweite Datenmodell. Das Datenmodell beschreibt Managementpläne, die öffentlich, über die Grenze der CSAR hinaus einsehbar, sind. Daher trägt es den Namen PublicPlan beziehungsweise PublicPlans.

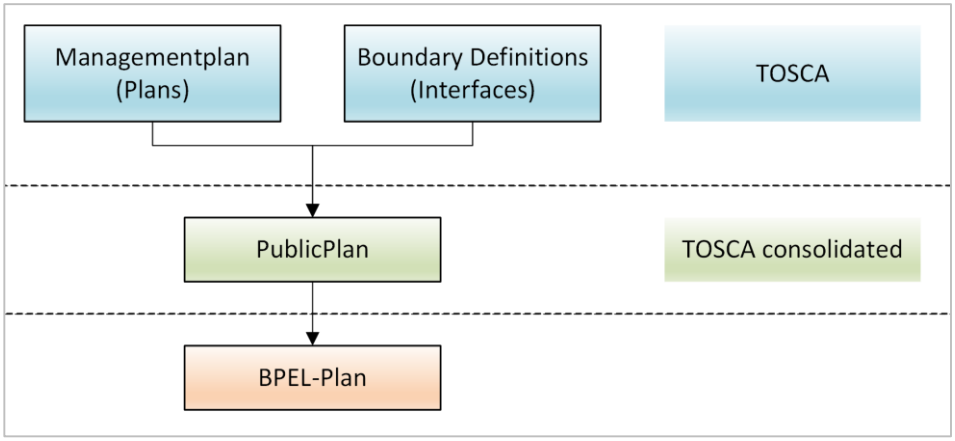

Abbildung 8: Zusammenhang von Managementplan und PublicPlan

PublicPlans sollen als XML Schema definiert sein. Dadurch kann ein Java-Datenmodell mit JAXB und XJC generiert werden. Die einfache Serialisierung zwischen Java-Objekten und XML-Dokument bietet viele Möglichkeiten in Hinblick auf den Datenaustausch zwischen OpenTOSCA und der GUI.

4.2 Erweiterung der Architektur von OpenTOSCA

Die bisherige Architektur bildet die Funktionalität von OpenTOSCA zu Beginn der Bachelorarbeit ab. Da aus den Zielen komplett neue Funktionalitäten resultieren, wird sich die Architektur von OpenTOSCA ändern.

Die Geschäftslogik implementierende Komponenten enden in OpenTOSCA mit dem englischen Begriff Engine. Daher wird die Komponente, welche die Aktivierung von Managementplänen auslöst, den Namen PlanInvocationEngine tragen. Weiterhin ist die Aufgabe der PlanInvocationEngine, Nachrichten von Managementplänen zu verarbeiten. Es liegt nahe, dass diese Komponente auch die korrekte Correlation zwischen Request und Response gewährleistet und damit die aktiven Managementpläne verwaltet.

Die Nachrichten werden vom schon bekannten Servicebus weiterverarbeitet, um daraus Prozessinstanzen zu aktivieren. Der Servicebus wird durch eine andere Bachelorarbeit ersetzt, daher wird der Name der Komponente nicht durch diese Bachelorarbeit geprägt und nur vorläufig in der Architektur übernommen.

Weiterhin wird eine Komponente benötigt, die Instanzen von CSARs verwaltet. Daher wird sie den Namen CSARInstanceManagementService tragen. Da ein Managementplan immer zugehörig zu einer CSAR-Instanz ist, wird die History auch von der Komponente CSARInstanceManagementService angeboten.

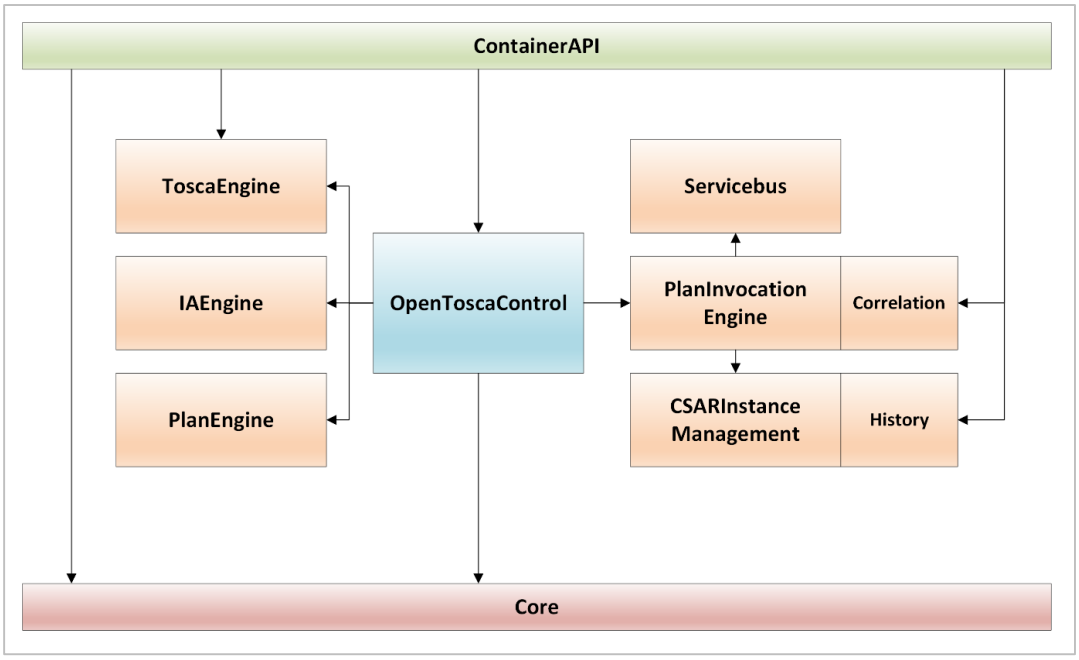

Abbildung 9: Zielarchitektur

4.3 PlanInvocationEngine und Servicebus

OpenTOSCA soll mit dem Zusammenspiel von PlanInvocationEngine und Servicebus Managementpläne ausführen.

4.3.1 Kommunikation per SOAP

SOAP ist das Protokoll, über das OpenTOSCA mit den BPEL-Webservices kommuniziert. Die PlanInvocationEngine hat die Aufgabe, mit den Nutzereingaben einen Managementplan zu aktivieren. Der Servicebus bietet die Funktionalität, Managementfunktionen von Service-

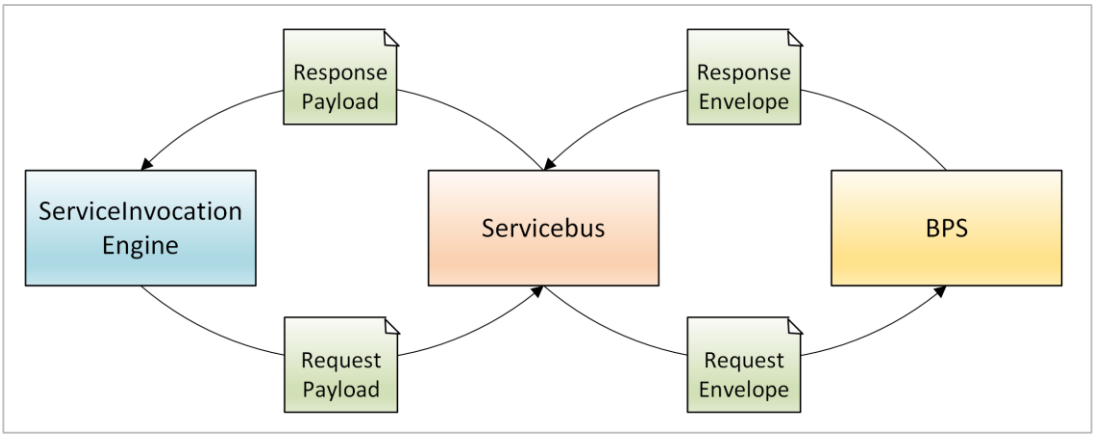

Abbildung 10: Kommunikation über SOAP

Komponenten aufzurufen. Daher ist die Aufteilung naheliegend, die PlanInvocationEngine den SOAP Body des SOAP Envelopes schreiben zu lassen. Während die PlanInvocationEngine sich um den Payload (engl.: die Nutzlast, in diesem Kontext die eigentlich zu übertragenden Daten im SOAP Envelope) kümmert, steuert der Servicebus das Wissen um beispielsweise das Routing oder der Verschlüsselung des Payloads bei. Michael Zimmermann – der Absolventen der Bachelorarbeit, die den Servicebus ersetzt – hat in Gesprächen diese Aufteilung akzeptiert.

Da die PlanInvocationEngine den SOAP Body schreibt, wird sie auch den Inhalt dessen generieren. Dazu muss der PlanInvocationEngine die Nutzereingaben übergeben werden, die den Inhalt der Requests des BPEL-Plans darstellt. Daraus wird die PlanInvocationEngine die in einer WSDL oder XML Schema definierten Nachricht generieren und in den SOAP Body kopieren. OpenTOSCA soll bestimmte Input-Parameter verstecken. Eine CorrelationID beispielsweise soll der Nutzer nicht selbst vergeben, sondern durch OpenTOSCA generiert werden. Asynchrone BPEL-Pläne müssen wissen, an welche Adresse – die sogenannte Callback-Adresse – die Response-Nachricht geschickt wird. Auch dies soll durch die Generierung einer Request-Nachricht von OpenTOSCA vor dem Nutzer versteckt werden. Das Wissen über die Callback-Adresse wird vom Servicebus bereitgestellt.

4.3.2 Kommunikation zwischen PlanInvocationEngine und Servicebus

PlanInvocationEngine und Servicebus sind deklarative Services, die miteinander agieren. Dies birgt die Problematik, dass beide die jeweils andere Komponente oder viel mehr deren Java-

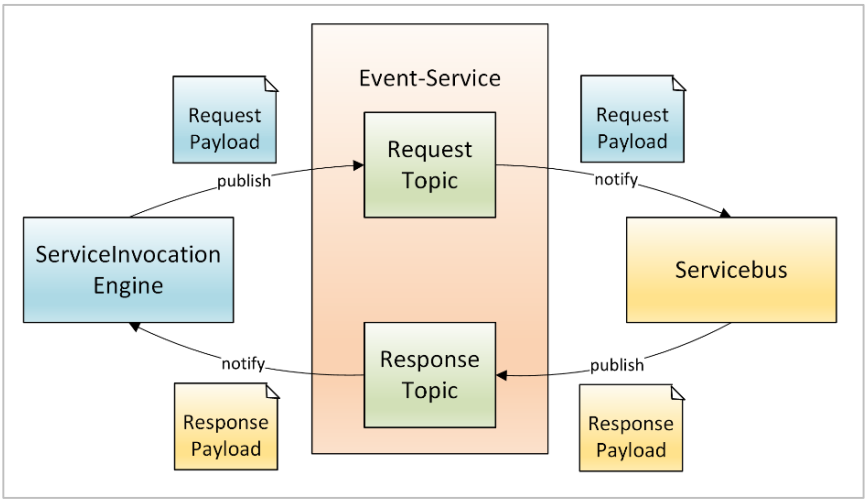

Abbildung 11: Kommunikation per OSGI Events

Package als OSGI-Abhängigkeit angeben. Dies Resultat wäre ein Zyklus der Abhängigkeiten, was OSGI nicht erlaubt. Weiterhin sollte die PlanInvocationEngine nicht die Funktionalität des Servicebusses als normalen Methodenaufruf auslösen. Dadurch würde die PlanInvocationEngine auf die Beendigung der Methode des Servicebusses warten und damit blockieren. Sollte der Servicebus beispielsweise auf die Antwort einer synchronen Prozessinstanz warten, würde die PlanInvocationEngine genauso während dieser Zeit blockieren.

Um eine asynchrone Kommunikation zwischen der PlanInvocationEngine und dem Servicebus zu erreichen, soll eine Kommunikation per Events mit dem „publish-subscribe“-Pattern aufgebaut werden. So kann die PlanInvocationEngine einen Request-Body auf einer Request-Liste im Event-Service veröffentlichen. Der Servicebus abonniert diese Liste und wird vom Event-Service benachrichtigt, sobald ein neuer Eintrag vorliegt. Dadurch erhält der Servicebus den Request-Body, ohne die PlanInvocationEngine zu kennen. Analog soll die Kommunikation für die Responses aufgebaut sein, damit PlanInvocationEngine und Servicebus voneinander entkoppelt sind.

4.4 Correlation von BPEL-Plänen

Sobald der Servicebus einen Request-Payload empfangen und an die PlanInvocationEngine weitergegeben hat, muss diese herausfinden, zu welchem Request der Response gehört. BPEL bietet den sogenannten „correlation set“-Mechanismus. Diese dienen der korrekten Ansteuerung von Prozessinstanzen: „[…] messages sent to such processes need to be delivered not only to the correct destination port, but also to the correct *instance* of the business process that provides the port. Messages which create a new business process instance, are a special case, as described […]” (8). Um Correlation Sets eines BPEL-Plans zu erkennen, müsste der BPEL-Plan analysiert werden. Dies sollte vermieden werden. Des Weiteren ist nicht gewährleistet, dass jeder BPEL-Autor Correlation Sets tatsächlich und korrekt einsetzt. Auch den Autoren vorzugeben, wie sie die Correlation Sets verwenden müssen und sie damit einzuschränken, ist keine genügende Lösung.

Daher muss der Autor eines BPEL-Planes eine Correlation einbinden, die dediziert durch OpenTOSCA verarbeitet wird. Dieses spezifische Element im Input-Payload muss in jedem Fall auch in den Output-Payload kopiert werden – im Fehlerfall durch einen „fault handler“. Dadurch kann die in der Response-Nachricht enthaltene Correlation einer Request-Nachricht zugeordnet werden. Die Generierung einer CorrelationID ist Aufgabe der PlanInvocationEngine. Der Aufbau der CorrelationID soll zweigeteilt sein. Der erste Teil ist eine Repräsentation der Zeit, zu der die CorrelationID generiert wurde. Da dies die Eindeutigkeit einer CorrelationID noch nicht gewährleistet, wird der zweite Teil der CorrelationID für jede CorrelationID hochgezählt. So können quasi gleichzeitig generierte CorrelationIDs voneinander unterschieden werden.

4.5 Verwaltung von CSAR-Instanzen

Eine in OpenTOSCA geladene CSAR bietet dem Nutzer Managementpläne an. Ein Build-Plan erstellt dabei eine Instanz des Services. Daher entsteht der Bedarf, dass OpenTOSCA die Instanzen einer CSAR kennt, solange die Instanz durch OpenTOSCA erstellt oder verändert wird. Dabei existieren zwei Szenarien, die für die Verwaltung der CSAR-Instanzen von Bedeutung sind.

Ein Build-Plan erstellt eine Instanz einer CSAR – jedoch benötigt die Prozessinstanz entsprechend Zeit. Die GUI muss darstellen, dass OpenTOSCA die Erstellung einer CSAR-Instanz gestartet hat. Damit die entsprechenden Daten für die GUI vorhanden sind, muss intern OpenTOSCA eine neue CSAR-Instanz erstellen. Die Abbildung 12 zeigt vereinfacht die Ablaufsequenz.

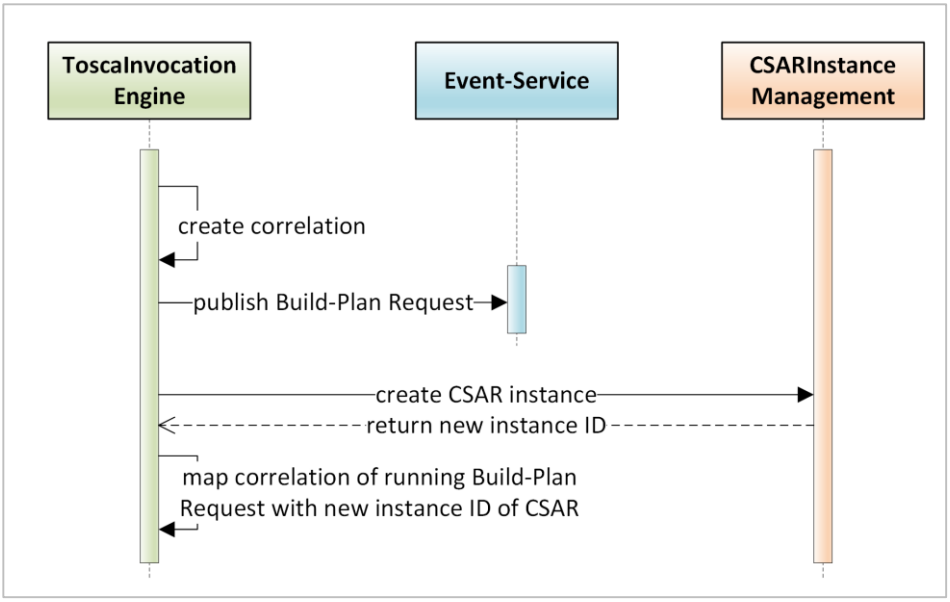

Abbildung 12: Sequenzdiagramm über den Start eines Build-Plans

Analog zum Build-Plan löscht ein Termination-Plan eine CSAR-Instanz. Es muss jedoch unterschieden werden, ob der Termination-Plan auch erfolgreich war. Schlägt der Termination-Plan fehl, lebt die CSAR-Instanz noch und muss dementsprechend OpenTOSCA weiterhin bekannt sein.

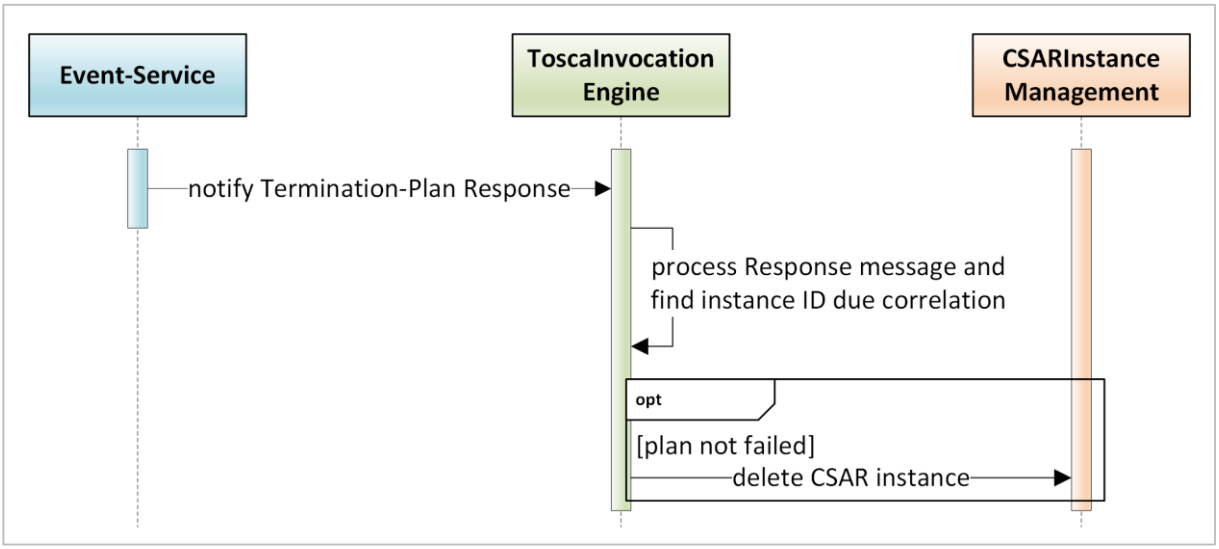

Abbildung 13: Sequenzdiagramm über die Verarbeitung eines Responses durch einen Termination-Plan

Daraus resultiert, dass ein Build-Plan eine komplette CSAR-Instanz erstellt und ein Termination-Plan eine komplette CSAR-Instanz deinstalliert und löscht. Das Szenario eines Managementplans, der beispielsweise die Skalierung eines Teils der Gesamttopologie der CSAR beeinflusst, ist damit jedoch nicht abgedeckt. TOSCA selbst definiert keine explizite Möglichkeit, dieses Verhalten eines Managementplans zu beschreiben. Ein PublicPlan, der dem Nutzer angeboten wird und weder einen Service instanziiert noch löscht, darf daher weder ein Build- noch ein Termination-Plan sein. Vielmehr nutzt dieser PublicPlan einen oder mehrere weitere Build- oder Termination-Pläne, die jedoch nicht über die Boundary Definitions dem Nutzer angeboten werden. Durch diese Entwickler-Richtlinie ist gewährleistet, dass der Lebenszyklus einer CSAR-Instanz klar definiert bleibt.

4.6 History

Sobald OpenTOSCA die Ergebnisse einer Instanz eines Managementplans verarbeitet hat, soll dies durch eine History für den Nutzer nachvollziehbar sein. Dies umfasst die Informationen über den Managementplan selbst, die durch den Nutzer eingegebenen Parameter und die durch die Instanz des Managementplans ausgegebenen Parameter.

Nach der Bearbeitung der Response-Nachricht durch die PlanInvocationEngine ist sowohl die Correlation selbst als auch die Nachricht für die PlanInvocationEngine nicht mehr von Interesse. Daher „vergisst“ sie beides.

Bevor die PlanInvocationEngine die Correlation löscht, muss sie sowohl die Correlation als auch die verarbeiteten Informationen an die History weitergeben. In der History müssen diese Informationen mit der CSAR-Instanz verknüpft werden, für die der Managementplan ausgeführt wurde.

Sobald in der History ein erfolgreicher Build-Plan gespeichert ist, kann die CSAR-Instanz als erfolgreich instanziiert angesehen werden. Sollte ein Termination-Plan scheitern, darf die CSAR-Instanz nicht gelöscht werden und der gescheiterte Termination-Plan muss in die History gespeichert werden.

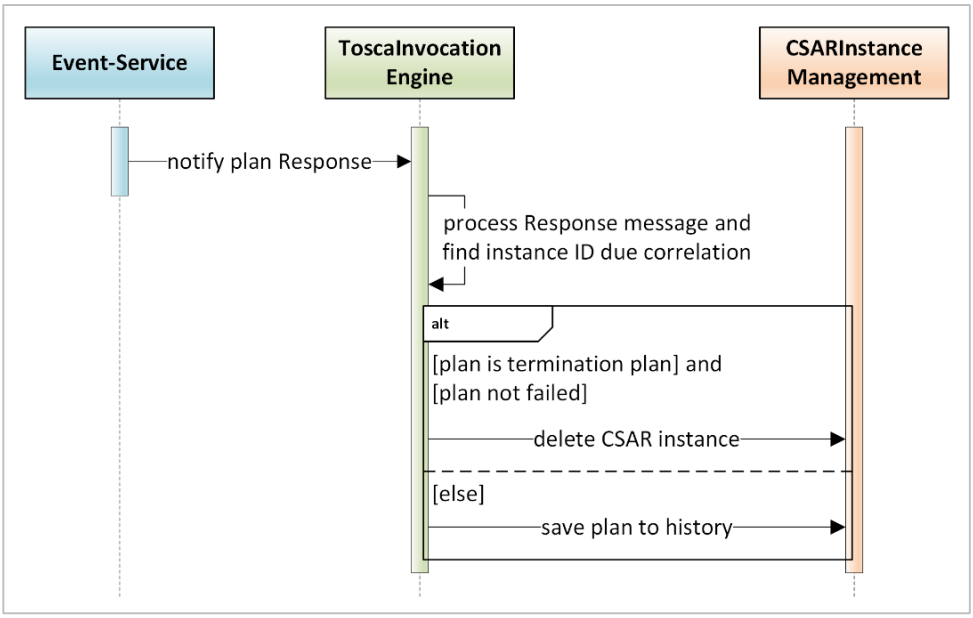

Abbildung 14: Sequenzdiagramm über die Speicherung in die History

4.7 Erweiterung der REST-Schnittstelle ContainerAPI

Die Funktionalität von OpenTOSCA wird durch die REST Schnittstelle ContainerAPI angeboten. Daher muss für die neue Funktionalität die im Kapitel 3.1.2 beschriebene ContainerAPI erweitert werden.

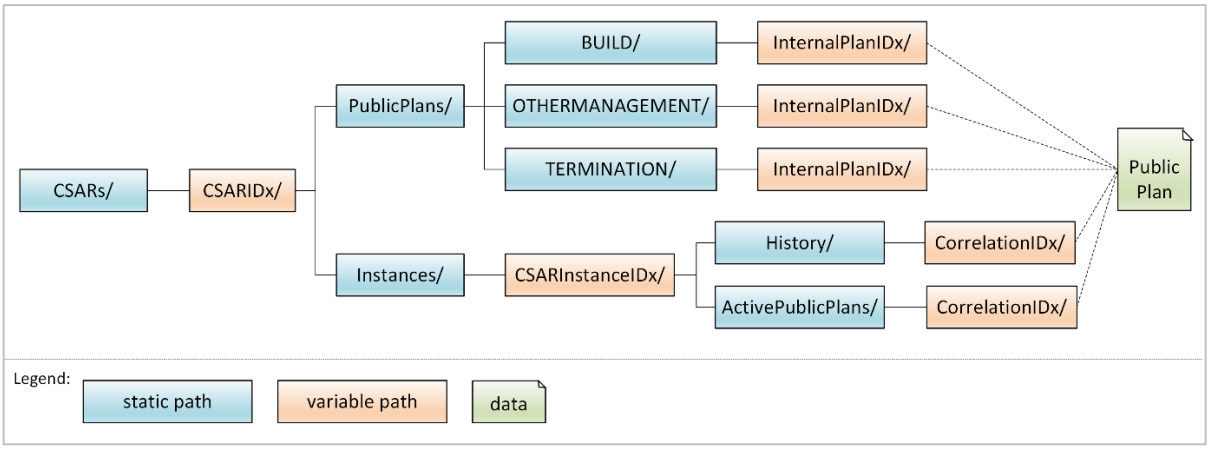

Abbildung 15: Konzept der Struktur-Erweiterung der ContainerAPI

Die Darstellung von Managementplänen soll durch das neue Datenmodell PublicPlans realisiert werden. Um einen gezielteren Zugriff auf einen Managementplan zu ermöglichen, soll die ContainerAPI Managementpläne anhand deren Typs anbieten. Um die Identifizierung eines Managementplans zu erleichtern soll eine numerische ID eingeführt werden.

CSAR-Instanzen, die durch einen PublicPlan instanziiert wurden oder gerade werden, sollen der CSAR selbst zugeordnet sein. Eine CSAR-Instanz soll über eine numerische ID identifizierbar sein. Weiterhin sollen sowohl die History, als auch die Informationen über aktuell laufende Prozessinstanzen direkt abhängig von der CSAR-Instanz sein.

Zusätzlich zur Darstellung von den gerade beschriebenen Informationen muss die ContainerAPI die Ausführung von PublicPlans anbieten. Jersey implementiert das CRUD-Pattern (englisches Akronym für create, read, update, delete). Daher sollte die entsprechende Operation auf der CSAR-Instanz (CSARInstanceID im obigen Bild) erfolgen.

4.8 Visuelle Darstellung

Die Konzepte bisher haben vorwiegend neue Funktionalität in OpenTOSCA selbst behandelt. Diese Konzepte sind mit dem Gedanken im Hinterkopf entstanden, die Funktionen auch einem menschlichen Nutzer intuitiv zu präsentieren. Demnach soll die alte GUI entsprechend weiter- oder neuentwickelt werden, um den vollen Funktionsumfang von OpenTOSCA visuell zu präsentieren.

4.8.1 Aufteilung der Website

Bisher sind die Funktionen der GUI in den Overview- und den „Deploy CSAR“-Bereich aufgeteilt. Die Overview zeigt Inhalte der CSAR wie das Bild der Topologie und die Verzeichnisstruktur in der CSAR sowie die darin enthaltenen Dateien. Die Ansicht „Deploy CSAR“ wiederum ermöglicht die Ansteuerung der einzelnen Verarbeitungsschritte einer CSAR: TOSCA analysieren und verarbeiten, Implementation Artifacts installieren und BPEL-Pläne installieren. Die Aufteilung in Informationen über und Verarbeiten der CSAR wird nicht verändert. Dennoch scheint der Inhalt der „Deploy CSAR“ Ansicht fehl am Platz. Ein Nutzer lädt eine CSAR in OpenTOSCA, um aus ihr Instanzen in der Cloud zu erzeugen. Die Verarbeitung der von der CSAR enthaltenen Daten erscheint dagegen als selbstverständlich. Daher sollen die drei Verarbeitungsschritte automatisch nach dem Laden einer CSAR ausgeführt werden.

Der frei werdende Platz kann dann für die Darstellung der noch zu implementierenden Funktionalität genutzt werden.

4.8.2 CSAR-Instanzen und Managementpläne in der GUI

Um die Managementfunktionalität in der GUI zu präsentieren, muss die GUI teilweise geändert werden. Folgendes Bild zeigt eine Konzeptzeichnung der GUI in der neuen Ansicht „Manage“.

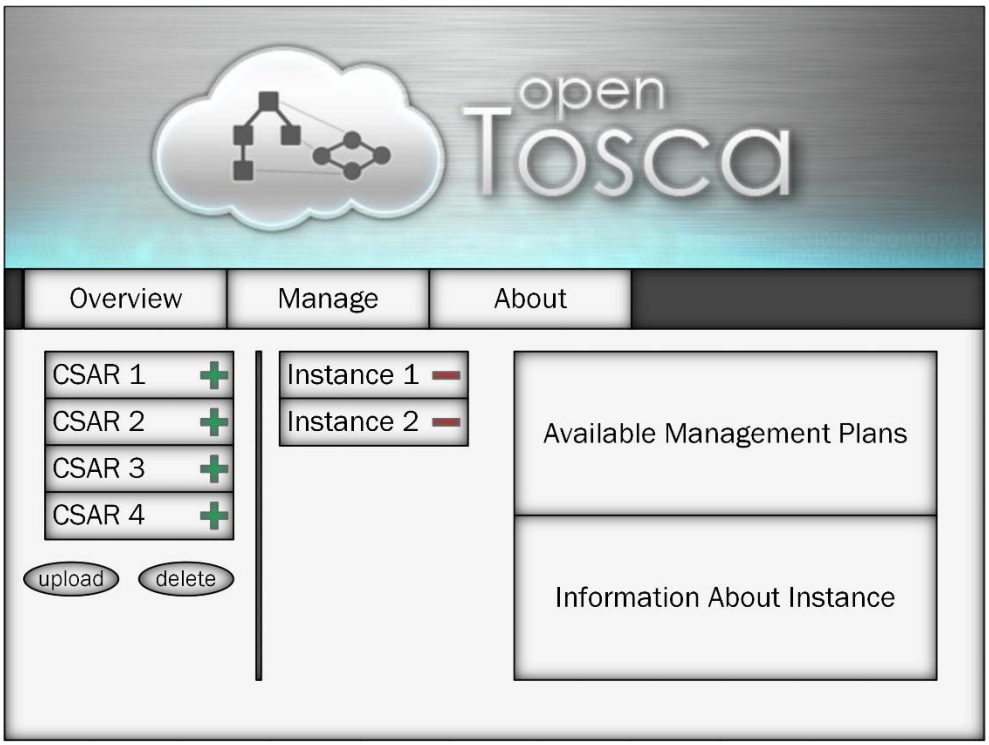

Abbildung 16: Konzept der GUI-Ansicht Manage

Die stilistischen Elemente sollen erhalten bleiben. Banner und Menüleiste der Ansichten bleiben gleich. Einzig die Ansicht „Deploy CSAR“ wird in Anlehnung an den Managementaspekt der Managementpläne in Manage umbenannt. Der Inhalt wird in der unteren Hälfte der Website dargestellt. Auf der linken Seite ist wie gehabt die Auflistung der in OpenTOSCA geladenen CSARs zu finden. Die Mitte listet die CSAR-Instanzen auf, die aus einer CSAR erzeugt wurden oder gerade erzeugt werden. Die rechte Seite zeigt die Managementfunktionalität einer in der Mitte markierten CSAR-Instanz an. Dies teilt sich grob in die Managementpläne der CSAR und in Informationen über die Instanz selbst. Die Informationen über die CSAR-Instanz beinhaltet Details über die Ausführung des Build-Plans, aktuell laufende Managementpläne und beendete Managementpläne in der History.

Eine CSAR-Instanz kann durch einen Klick auf das grüne Plus-Zeichen einer CSAR instanziiert werden. Durch den Klick soll ein Dialog aufgerufen werden, der die Input-Parameter des Build-Plans abruft. Eine CSAR-Instanz soll durch den Klick auf das rote Minus-Zeichen einer CSAR-Instanz gelöscht werden. Auch hier soll von einem Dialog die Input-Parameter des Termination-Plans abgefragt werden. Das Auslösen eines anderen Managementplans soll analog implementiert sein.

4.8.3 Darstellung des Fortschritts einer Prozessinstanz

Für die Darstellung des Fortschritts einer Prozessinstanz ist ein einfacher „Lade-Balken“ gefordert. Dieser soll den Fortschritt prozentual darstellen, in dem der ausgefüllte Balken die abgearbeiteten Aktivitäten zur Gesamtzahl der Aktivitäten in Relation setzt.

Die Laufzeitumgebung, in der die Prozessinstanz ausgeführt wird, müsste Informationen bereitstellen, durch die man den Prozessfortschritt nachvollziehen kann. Dadurch müsste OpenTOSCA näher an den BPS angebunden werden, um entsprechende Informationen abrufen zu können. Dies zu implementieren weicht jedoch stark von den eigentlichen Zielen dieser Bachelorarbeit ab.

Aufgrund des Sachverhalts wurde im Gespräch mit dem Betreuer festgelegt, dass die GUI nicht einen Fortschritt einer Prozessinstanz anzeigen soll. Jedoch soll weiterhin ersichtlich sein, dass eine Prozessinstanz aktiv ist und OpenTOSCA auf deren Ergebnisse wartet.

5 Implementierung

Diese Kapitel beschreibt, wie die Konzepte in OpenTOSCA implementiert sind. Die Struktur gliedert sich in zwei Teile. Zuerst sind die neuen Funktionen in OpenTOSCA selbst beschrieben. Danach ist dargelegt, wie die hinzugekommene Funktionalität durch die GUI angeboten ist.

5.1 OpenTOSCA

Dieses Unterkapitel beschreibt die neue Funktionalität von OpenTOSCA.

5.1.1 Einschränkungen bezüglich der Granularität und Umfang

Da der Prototyp OpenTOSCA weiterentwickelt wird, sind marginale Änderungen an Komponenten, die nicht Teil der Bachelorarbeit sind, nicht beschrieben – beispielsweise Korrekturen von „bugs“. Auch fehlen Informationen über die vormalige Implementierung, die nicht verändert wurde. Dadurch genügen vor allem die Diagramme nicht dem Anspruch, Komponenten oder die Kommunikation zwischen Komponenten vollständig zu dokumentieren.

Komponenten in OpenTOSCA kommunizieren miteinander über deklarative Services, die durch OSGI definiert sind. Diese Services werden von Schnittstellen beschrieben und werden gegen ihre Schnittstelle gebunden, um sie zu nutzen. Da dieses Vorgehen jedes Sequenzdiagramm und die Beschreibungen der Implementierung an Umfang deutlich vergrößern würde, wird OSGI nur erwähnt, wenn neue OSGI-Funktionalität genutzt wird.

5.1.2 TOSCA verarbeiten und konsolidieren

TOSCA wird in OpenTOSCA von der ToscaEngine verarbeitet. Nachdem die Referenzen in den TOSCA-Dokumenten aufgelöst wurde, konsolidiert die ToscaEngine durch die Klassen DefinitionsConsolidation und ExportedInterfacesConsolidition des Pakets „org.opentosca.toscaengine.service.impl.consolidition“ die nötigen Informationen. Die Informationen sind aus den TOSCA- und den referenzierten WSDL-Dokumenten in der CSAR zusammengetragen und werden in JAXB-annotierte Klassen der XML Schema unter 9.2 gespeichert. Aufbau, Zweck und Zusammenhänge sind im Kapitel 5.1.3 beschrieben.

5.1.3 PublicPlans, CorrelationID und Callback-Adresse

Ein PublicPlan ist eine Datenstruktur, die intern für OpenTOSCA Managementpläne beschreibt und durch Konsolidierung von TOSCA mi Daten befüllt wird. PublicPlans werden in OpenTOSCA verwendet, um durch die Boundary Definitions für den Nutzer sichtbar definierte Managementpläne zu beschreiben. Die ContainerAPI erzeugt von PublicPlans XML-Dokumente, welche beispielsweise von der GUI abgefragt werden. Auch akzeptiert die ContainerAPI XML-Dokumente nach dem PublicPlans XML-Schema, um über diese Managementpläne zu interagieren. Das folgende XML-Dokument ist ein durch OpenTOSCA und der GUI verarbeitetes Beispiel der CSAR „TestLifeCycle.csar“:

```
1  <PublicPlan CSARID="TestLifeCycle.csar"
2       InternalInstanceInternalID="0"
3       InternalPlanID="0"
4       PlanID="ns3:TestLifeCycleDemoBUILDPlan"
5       PlanType="http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
6       PlanLanguage="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
7       InterfaceName="TestBuildPlan"
8       OperationName="process"
9       InputMessageID="ns2:TestBuildPlanRequest"
10      hasFailed="false"
11      isActive="false"
12      xmlns:ns2="org.opentosca/test/planinvocation/buildplan"
13      xmlns:ns3="org.opentosca/test">
14
15      <InputParameter required="true" type="string" name="input">
16         test build</InputParameter>
17      <InputParameter required="true" type="correlation" name="CorrelationID">
18         1366735510933-0</InputParameter>
19      <InputParameter required="true" type="callback" name="CallbackAddress">
20         127.0.0.1</InputParameter>
21
22      <OutputParameter required="true" type="string" name="result">
23         test build</OutputParameter>
24      <OutputParameter required="true" type="correlation" name="CorrelationID">
25         1366735510933-0</OutputParameter>
26
27 </PublicPlan>
```

Das Beispiel zeigt die XML-Repräsentation eines PublicPlans, das in der History durch die PlanInvocationEngine abgespeichert wurde, nachdem der Build-Plan von OpenTOSCA aktiviert und dessen Ergebnis ausgewertet wurde. Folgende Tabelle erklärt die einzelnen Attribute und Elemente.

Attribut oder Element	**Erklärung**
CSARID	Die Textrepräsentation der internen ID einer CSAR in OpenTOSCA.
InternalInstanceInternalID	Die in OpenTOSCA interne ID einer CSAR-Instanz. Das zweite Internal resultiert aus der Benennung der HashMaps, in denen die Informationen gespeichert werden. Die ID entspricht der CSARInstanceID in der Abbildung 15 und Abbildung 17.
InternalPlanID	Die in OpenTOSCA interne ID des PublicPlans. Die ID entspricht der InternalPlanID in der Abbildung 15 und Abbildung 17.
PlanID	Dies ist der QName, durch den das Plan-Element in TOSCA identifiziert werden kann.
PlanType	Die URI aus TOSCA, die den Typ des Plans definiert. Zwei mögliche URIs gibt die TOSCA Spezifikation vor, um Build- und Termination-Pläne zu identifizieren.
PlanLanguage	Die URI aus TOSCA, die das Modell identifiziert, von dem der Plan ein Objekt ist.
InterfaceName	Eine URI oder NCName, mit dem TOSCA ein Interface in den Boundary Definitions benennt. Durch den InterfaceName wird zusätzlich während der Konsolidierung der PortType beziehungsweise das Interface in der WSDL identifiziert.
OperationName	Ein NCName, mit dem TOSCA eine Operation eines Interfaces in den Boundary Definitions benennt. Durch den OperationName wird zusätzlich während der Konsolidierung die Operation des durch den InterfaceName identifizierten PortTypes beziehungsweise Interfaces in der WSDL identifiziert.
InputMessageID	Der QName des XML Schema-Elements, das den SOAP Body-Inhalt definiert. Der QName wird während der Konsolidierung anhand des InterfaceName und OperationName identifiziert.
hasFailed	Zeigt an, ob der Response eines Managementplans einen SOAP Fault beinhaltet hat. Ist dieser Boolean wahr, so ist der SOAP Fault als Output-Parameter im PublicPlan gespeichert. Ist der Boolean mit falsch belegt, hat die Prozessinstanz keinen Fehler verursacht und war erfolgreich.

isActive	Dieser Boolean zeigt an, ob der PublicPlan eine aktive Prozessinstanz repräsentiert. Ist der Boolean mit wahr belegt, hat OpenTOSCA noch keinen Response für die CorrelationID verarbeitet.
InputParameter	Das Element ist vom Plan-Element in TOSCA übernommen. Zusätzlich kann der Wert des Elements mit Text belegt werden, um die Nutzereingabe zu speichern.
OutputParameter	Das Element ist vom Plan-Element in TOSCA übernommen. Zusätzlich kann der Wert des Elements mit Text belegt werden, um die Nutzereingabe zu speichern.

Tabelle 1: PublicPlan Aufbau und Erklärung

Ein PublicPlan beschreibt einen Managementplan komplett. Input- und Output-Parameter enthalten Informationen, die für die Correlation und Kommunikation notwendig sind. Die CorrelationID muss sowohl als Input- als auch als Output-Parameter enthalten sein. Für asynchrone BPEL-Pläne wird die Adresse des Callback-Mechanismus benötigt, welcher dann zumindest als Input-Parameter enthalten ist. Für diese Parameter soll dem Nutzer kein Eingabefeld angeboten werden. Die CorrelationID für die Nachrichten wird für jede Request-Nachricht generiert. Die Callback-Adresse wird zwar nicht generiert, dennoch genauso ohne Zutun des Nutzers eingefügt. Dies sind doch Anforderungen an die GUI und die PlanInvocationEngine.

5.1.4 ContainerAPI und OpenToscaControl

Die ContainerAPI nimmt durch Jersey Befehle für OpenTOSCA entgegen und gibt sie an die OpenToscaControl weiter. Die OpenToscaControl löst die entsprechende Geschäftslogik aus. Damit die ContainerAPI in verschiedenen Komponenten gespeicherte Informationen anbieten kann, ist die ContainerAPI an die deklarativ angebotenen OSGI-Services der einzelnen Komponenten angebunden.

Entsprechend dem Konzept und der Vorgehensweise der vorhergehenden Implementierung ist die ContainerAPI erweitert worden. Um wildes Blättern zu vermeiden, nachfolgend nochmals die ContainerAPI als vereinfachte Baumstruktur aus dem Konzept-Kapitel. Jedes Rechteck entspricht einer Ressource, welche beispielsweise in der URL-Zeile eines Browsers angesteuert werden kann.

Der Ast-Abschnitt PublicPlans bildet die verfügbaren Managementpläne einer CSAR ab. Diese sind nach ihrem Typ geordnet. So gibt die ContainerAPI für den Pfad „…/PublicPlans/BUILD/0“ den PublicPlan des ersten Build-Plans in serialisierter XML-Form zurück.

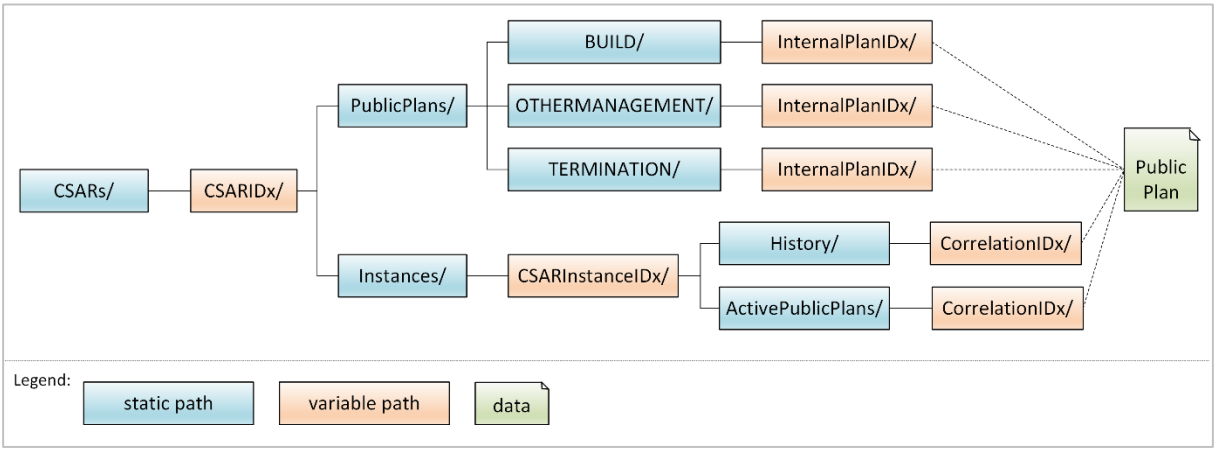

Abbildung 17: ContainerAPI Baumstruktur

Parallel dazu stellt der Ast-Abschnitt Instances die Möglichkeit bereit, sowohl Informationen über CSAR-Instanzen abzurufen, als auch Managementpläne auszuführen. Die CSARInstanceID ist die String-Repräsentation eines internen Identifikationsobjekts einer CSAR-Instanz. Diese ist durch die CSARID und einer intern vergebenen, numerischen und von der CSARID abhängigen ID aufgebaut. Da die CSARID in der URL schon angegeben ist, nutzt die ContainerAPI hinter Instances den internen numerischen Wert.

Die CorrelationID dient der Identifikation einer Prozessinstanz, welche unter „.../ActivePublicPlans“ gerade ausgeführt wird und analog unter „…/History“ abgeschlossen ist. Wie die Informationen über aktive und abgeschlossene Prozessinstanzen gespeichert werden, ist in den Kapiteln 5.1.9 und 5.1.10 beschrieben.

Jeder Pfadabschnitt gibt ähnlich einem Verzeichnis entweder die enthaltenen Teilpfade der nächst unteren Ebenen oder in den Baumblättern die entsprechenden Objekte zurück. Daher bekommt der Client für eine InternalPlanID oder CorrelationID einen PublicPlan zurück.

Um Managementfunktionalität einer CSAR-Instanz auszulösen, kann an die CSARInstanceID ein PublicPlan in XML-Form geschickt werden. Soll beispielsweise ein Termination-Plan mit der internen Plan-ID 0 für eine CSAR-Instanz mit der ID 0 ausgeführt werden, kann an die Adresse „…/Instances/0“ der Termination-Plan geschickt werden. Dieser muss dem PublicPlan der

Adresse „…/PublicPlans/TERMINATION/0“ entsprechen und die Input-Parameter des Managementplans enthalten.

Um dem REST-Konzept zu entsprechen, kann an eine CSARInstanceID die entsprechende Put- und Post-Anfrage gesendet werden. Jersey unterstützt in der verwendetet Version 1.11 nur Parameter für Delete-Anfragen, die die Ressource identifizieren. Wenn XML-Dokument an eine mit „@DELETE“ und „@CONSUMES(MediaType.APPLICATION_XML)“ annotierte Methode geschickt wird, wird diese Methode nicht ausgeführt. Daher wird ein Termination-Plan über eine Post-Anfrage aktiviert.

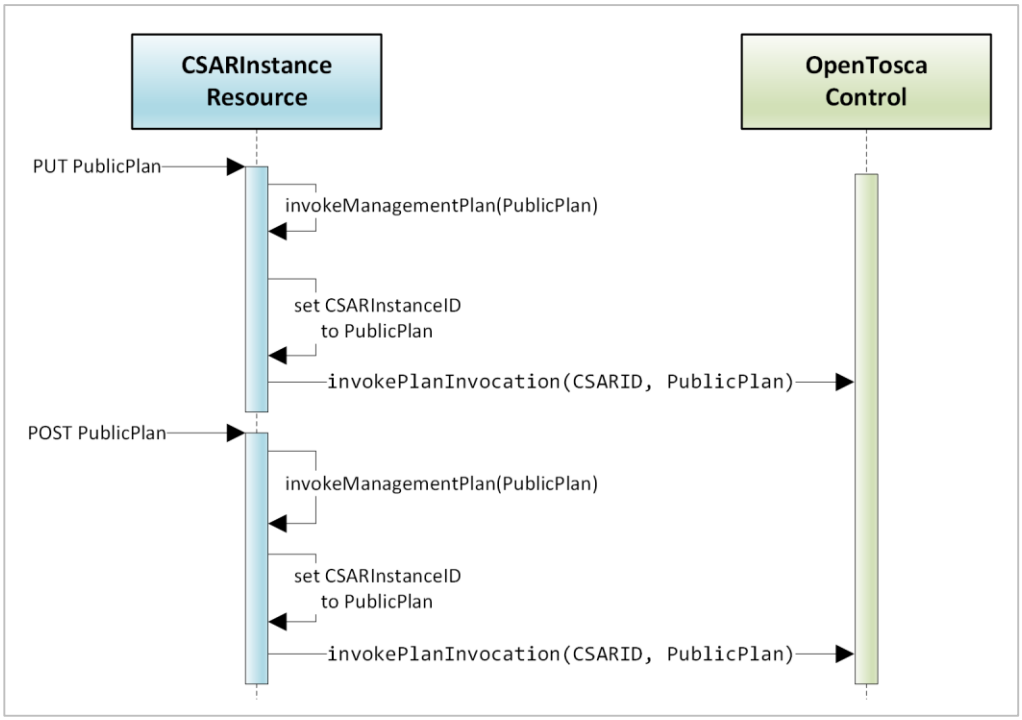

Abbildung 18: Sequenz über die Aktivierung von Managementplänen über die ContainerAPI

Dem übermittelten PublicPlan wird die Information der CSARInstanceID hinzugefügt. Danach wird abhängig der CSARID der PublicPlan an die OpenToscaControl weitergeben, damit diese die Planaktivierung übernimmt.

5.1.5 Ausführung eines BPEL-Prozesses durch die PlanInvocationEngine

Nachdem die ContainerAPI einen PublicPlan samt Input-Parameter der OpenToscaControl übermittelt hat, gibt die OpenToscaControl die Informationen weiter an die

PlanInvocationEngine. Die PlanInvocationEngine aktiviert den durch den PublicPlan beschriebenen Managementplan durch den Servicebus.

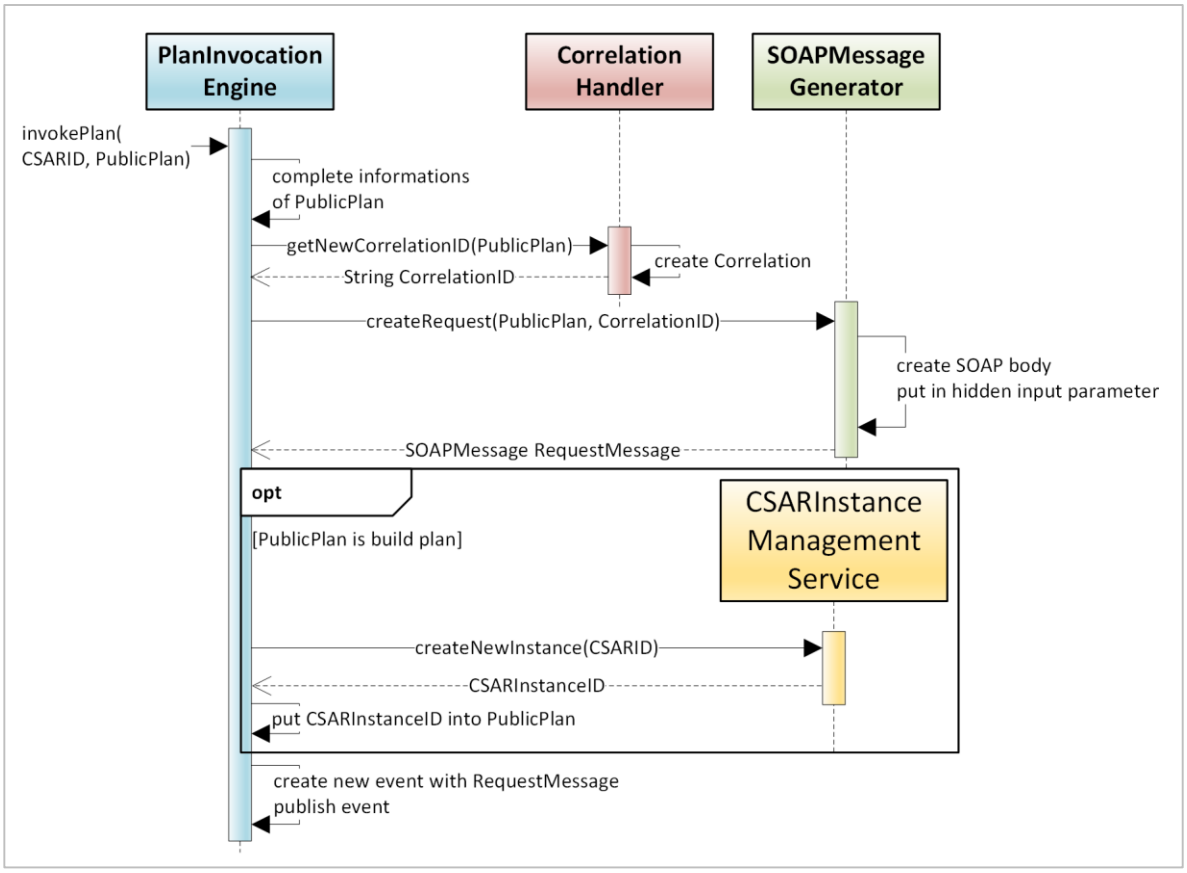

Abbildung 19: Aktivieren eines PublicPlans durch die PlanInvocationEngine

Die PlanInvocationEngine bekommt mit einer CSARID einen PublicPlan übergeben. Der PublicPlan enthält nur die vom Nutzer eingegebenen Informationen aus der GUI, da einzelne Felder wie die CorrelationID versteckt sind. Die PlanInvocationEngine ergänzt den PublicPlan mit den fehlenden, in Kapitel 5.1.3 beschriebenen Informationen aus der ToscaEngine. Dies ist im obigen Bild zugunsten der Übersichtlichkeit nur angedeutet. Der CorrelationHandler erzeugt eine neue CorrelationID, was in Kapitel 5.1.9 näher beschrieben ist.

Danach wird der SOAPMessageGenerator mit der Erstellung der SOAP Message und vor allem des SOAP Bodys mit dem Payload aus dem PublicPlan beauftragt. Als Datenformat für SOAP-Bestandteile sind die Klassen aus dem Paket „javax.xml.soap“ verwendet. Der SOAPMessageGenerator erfährt die Callback-Adresse vom Servicebus. Dazu horcht der SOAPMessageGenerator ähnlich wie in Kapitel 5.1.7 beschrieben auf die Liste „org_opentosca_plans/callbackaddress“ und speichert die Callback-Adresse intern bei jedem

Event ab. Danach gibt der SOAPMessageGenerator die SOAP Message mit dem enthaltenen SOAP Body an die PlanInvocationEngine zurück.

Ein Request an einen Build-Plan erzeugt eine neue CSAR-Instanz. Daher wird in diesem Fall der CSARInstanceManagementService mit der internen Erstellung einer neuen CSAR-Instanz beauftragt, welche auch in den PublicPlan gespeichert wird.

Schlussendlich erstellt die PlanInvocationEngine einen neuen Event und veröffentlicht ihn – wie in Kapitel 5.1.7 beschrieben – auf der Liste für Requests. Darin enthalten ist die SOAP Message und der PublicPlan, in dem die CSAR, die ID des auszuführenden Plans und die CSAR-Instanz-ID gespeichert sind.

5.1.6 Verarbeitung des Responses eines BPEL-Prozesses

Sobald eine Prozessinstanz den Response zurück geschickt hat, nimmt der Servicebus sie entgegen. Dieser schickt den SOAP Body wie in Kapitel 5.1.7 beschrieben an die PlanInvocationEngine, damit diese den Inhalt des Responses verarbeiten kann.

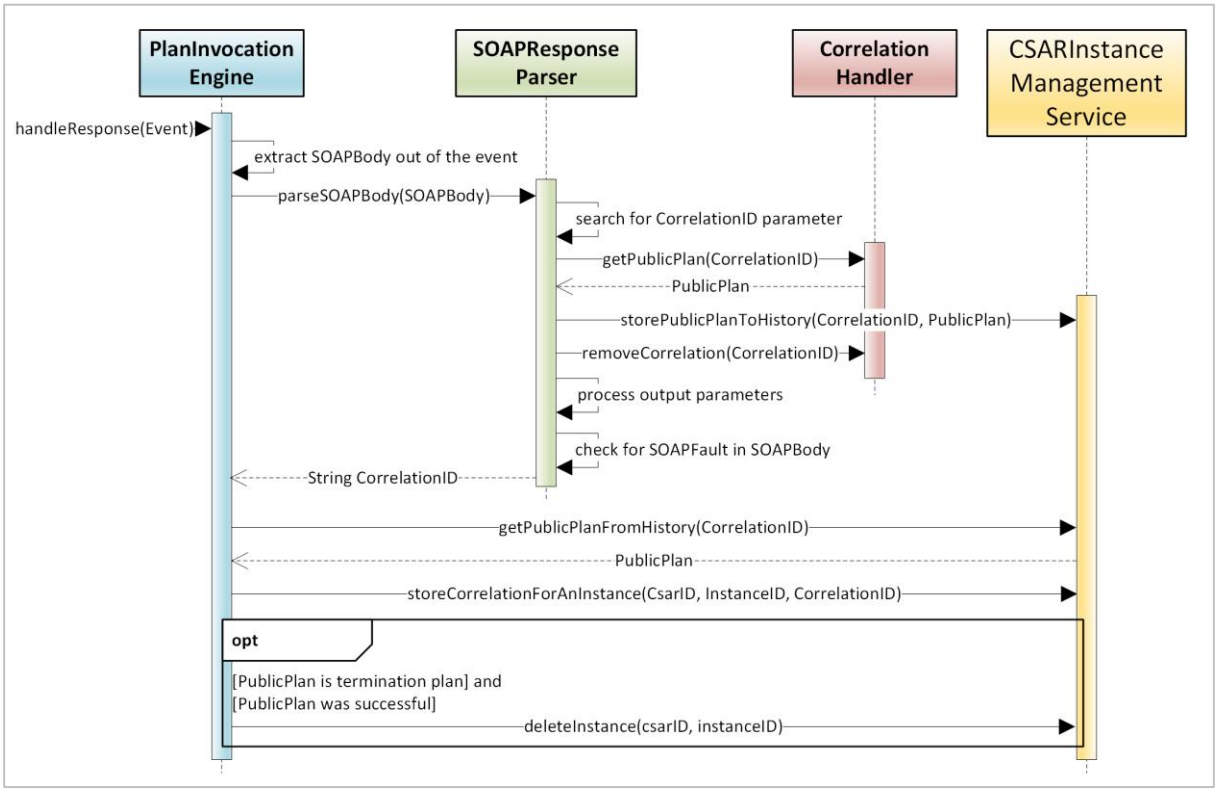

Abbildung 20: Verarbeiten eines Responses durch die PlanInvocationEngine

Sobald die PlanInvocationEngine einen Response per Event erhalten hat, übergibt die PlanInvocationEngine den SOAP Body an den SOAPResponseParser.

Der SOAPResponseParser sucht zuerst nach einer CorrelationID, damit der Response auch zu einem Request und PublicPlan zuordenbar ist. Hat der Response genau eine CorrelationID, wird der CorrelationHandler nach dem zugehörigen PublicPlan gefragt. In Kapitel 5.1.9 wird beschrieben, wie OpenTOSCA mit aktiven PublicPlans und dementsprechend deren Correlations umgeht. Nachdem der PublicPlan mit den Input-Parametern dem SOAPResponseParser bekannt ist, wird der PublicPlan in die History des CSARInstanceManagementServices gespeichert und die CorrelationID im CorrelationHandler gelöscht. Wie die History verwaltet wird, ist in Kapitel 5.1.10 beschrieben.

Damit die History auch die Output-Parameter des PublicPlans mit den entsprechenden Werten kennt, wird der Response ausgewertet und die Informationen in das Objekt des PublicPlans der History gespeichert. Schlussendlich wertet der SOAPResponseParser einen möglicherweise vorhandenen SOAP Fault aus und speichert die Informationen in den Output des PublicPlan-Objekt in der History.

Nach der Verarbeitung des Responses erhält die PlanInvocationEngine die CorrelationID vom SOAPResponseParser zurück. Die CorrelationID wird dazu verwendet, in der History den PublicPlan und die CorrelationID mit der entsprechenden CSARID und CSARInstanceID zu verknüpfen. Sollte der PublicPlan ein erfolgreicher Termination-Plan sein, wird die CSAR-Instanz durch den CSARInstanceManagementService gelöscht.

5.1.7 Event-Kommunikation zwischen PlanInvocationEngine und Servicebus

Die Kommunikation durch Events der beiden Services PlanInvocationEngine und Servicebus ist durch native OSGI-Funktionalität des Pakets „org.osgi.service.event“ (9) implementiert.

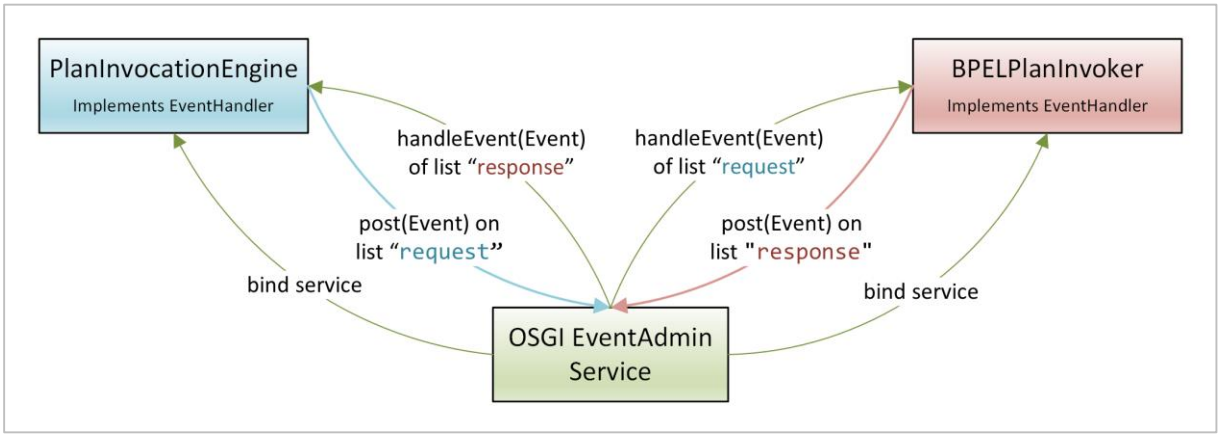

Abbildung 21: Austausch von Request- und Response-Events

Um Komponenten über OSGI Events kommunizieren zu lassen, muss das entsprechende Bundle der Laufzeitumgebung hinzugefügt werden. In OpenTOSCA läuft die Version 1.2.1 des Pakets „org.eclipse.equinox.event“ der Eclipse Foundation, welches in der Equinox SDK 3.7.1 (10) zu finden ist.

Um einen Event auf eine Liste zu veröffentlichen, muss die entsprechende Komponente den EventAdmin-Service binden. Das Interface des EventAdmin-Services bietet die Methode „post(Event event)“ an, welcher der zu veröffentlichende Event übergeben werden muss. Das Event-Objekt enthält dabei eine Referenz der Liste, auf welcher der Event veröffentlicht werden soll. Für den Austausch von Requests wird die Liste „org_opentosca_plans/requests“ und für Responses die Liste „org_opentosca_plans/responses“ genutzt.

Der veröffentlichte Event wird vom EventAdmin-Service an alle Komponenten übergeben, die die Liste abonniert haben. Dazu muss beispielsweise der Servicebus das Interface EventHandler implementieren und die Methode „handleEvent(Event event)“ überschreiben. Zusätzlich dazu muss die implementierende Klasse den Service EventHandler anbieten und diesem Service die Eigenschaft „event.topics“ hinzufügen. „event.topics“ definiert die Listen, die abonniert sind. Die Klasse BPELPlanInvoker des Bundles „org.opentosca.servicebus.mockup“ implementiert den Service EventHandler und abonniert die List „org_opentosca_plans/requests“. Analog dazu abonniert die Klasse PlanInvocationEngine die Liste „org_opentosca_plans/responses“.

Während die Klasse PlanInvocationEngine einen Request selbst veröffentlicht, veröffentlicht die Klasse Connection im Paket „org.opentosca.servicebus.mockup“ für eine empfangene Nachricht den Response. Der Servicebus ist im Kapitel 5.1.8 beschrieben.

5.1.8 Mock-up Servicebus

Der Servicebus schickt SOAP Inhalte an die entsprechende „Web Service“-Adresse und aktiviert damit den entsprechenden Managementplan.

Hierzu horcht die Klasse BPELPlanInvoker auf einen neuen Event auf der Liste „org_opentosca_plans/responses“. Durch den EventAdmin-Service ausgelöst, extrahiert der BPELPlanInvoker die SOAP Message aus dem Event und sucht nach der entsprechenden „Web

Service“-Adresse. Dies ist statisch über if-Konstrukte implementiert. Mit einem Objekt der Klasse Connection wird die SOAP Message verschickt.

Die Klasse Connection implementiert das Interface Runnable und löst die Kommunikation per SOAP aus. Die verwendete SOAPConnection ist eine Klasse aus dem Paket „javax.xml.soap“ und ermöglicht die synchrone Kommunikation per SOAP mit einem Web Service. Sobald ein Response eintrifft, veröffentlicht die Klasse Connection den Response durch den im BPELPlanInvoker statisch gebundenen EventAdmin-Service.

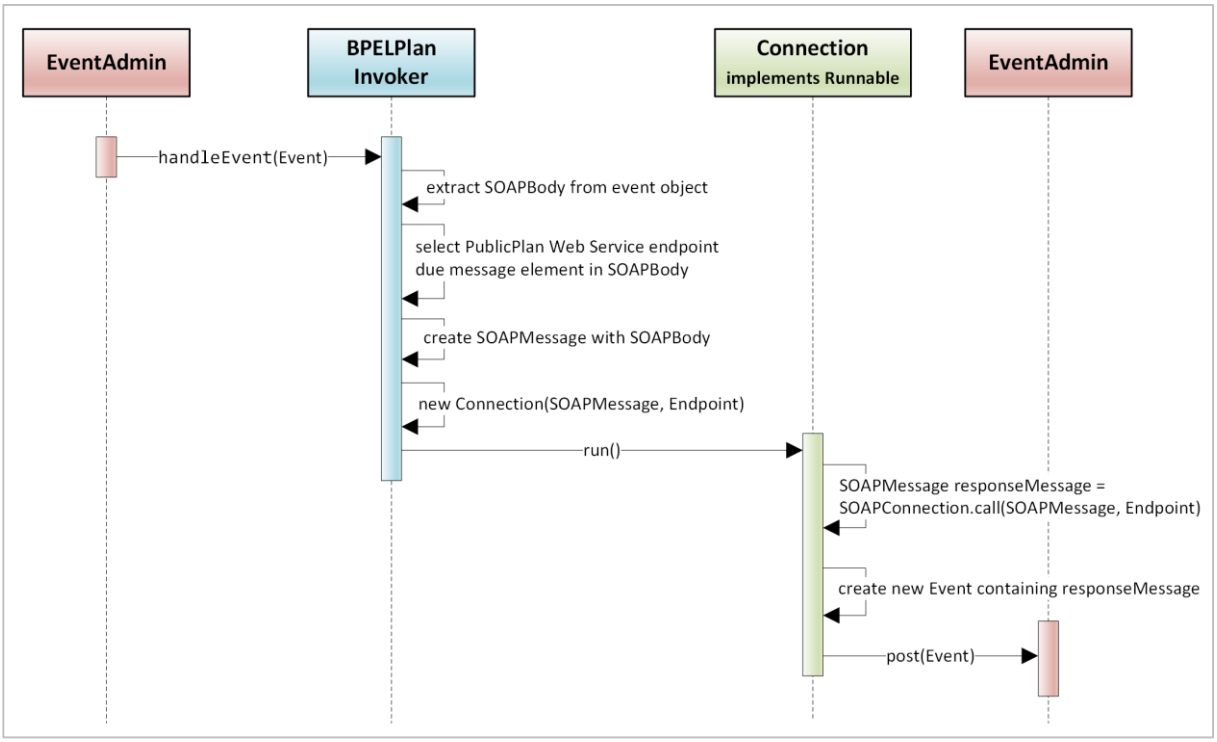

Abbildung 22: Funktionsweise des Servicebusses

Zusätzlich zur Aktivierung von Managementplänen, veröffentlicht der Servicebus die aktuelle Callback-Adresse auf der Event-Liste „org_opentosca_plans/callbackaddress“. Dementsprechend sendet der Servicebus die Callback-Adresse bei Programmstart und wenn sie sich ändert an die Liste.

5.1.9 Verwalten von aktiven Prozessinstanzen und deren Correlation

Dieses Kapitel beschreibt den in Kapitel 5.1.5 und 5.1.6 angesprochenen CorrelationHandler. Der CorrelationHandler ist im Paket „org.opentosca.planinvocationengine.service.impl.correlation“ angesiedelt und hat zwei Aufgaben. Einmal erzeugt er eine neue

CorrelationID für einen PublicPlan um eine Prozessinstanz eindeutig identifizierbar zu machen. Diese Methode ist „synchronized“, damit keine Effekte durch Nebenläufigkeit entstehen.

Die erzeugte CorrelationID besteht aus zwei durch ein Minuszeichen getrennte Teile. Der erste Teil ist die Systemzeit, die durch den Befehl „System.currentTimeMillis()“ durch Java bereitgestellt wird. Java ermöglicht es auch eine Zeitmessung mit Nanosekunden durchzuführen. Dies bildet aber weder die Systemzeit ab, noch ist der Wert zwingenderweise positiv: „values may be negative“ (11) und ist für die Messung einer Zeitdauer geeignet.

Der zweite Teil ist eine positive Zahl, beginnend bei null. Die Zahl wird für jede Millisekunde um eins hochgezählt, wenn eine weitere CorrelationID in dieser Millisekunde erzeugt werden soll.

Die zweite Aufgabe des CorrelationHandlers besteht darin, die Correlations von aktiven Managementplänen zu verwalten. Durch geschachtelte HashMaps wird zur CSARID die CorrelationID zu dem entsprechenden PublicPlan gespeichert. Die Information über die CSAR-Instanz ist im PublicPlan selbst gespeichert. Verarbeitet die PlanInvocationEngine einen Response, veranlasst sie den CorrelationHandler dazu, die CorrelationID zu löschen. Die Informationen über noch offene Correlations ist der ContainerAPI zugänglich, um diese Informationen zum Beispiel der GUI anzubieten.

5.1.10 Verwalten von CSAR-Instanzen und der History

In den Kapitel 5.1.5 und 5.1.6 wurde nicht nur der CorrelationHandler erwähnt, sondern auch der CSARInstanceManagementService. Aufgabe dieses Services ist es, CSAR-Instanzen zu verwalten und die History bereitzustellen.

Dazu verwendet der CSARInstanceManagementService zwei Datenstrukturen. Die erste hält die Informationen über die CSAR-Instanzen. Dazu wird über HashMaps die CSARID zur CSARInstanceID in Verbindung gesetzt. Die CSARInstanceID ist wiederum mit einer Liste von CorrelationIDs verknüpft. Die zweite Datenstruktur kümmert sich nur um die Speicherung von PublicPlans. Dazu wird durch eine HashMap eine CorrelationID auf den entsprechenden PublicPlan abgebildet. Dieser PublicPlan ist durch die PlanInvocationEngine mit allen Informationen befüllt und enthält die Informationen des Requests und des Responses.

Der CSARInstanceManagementService bietet als Service die gespeicherten Informationen intern an. Dies wird von der ContainerAPI genutzt, um beispielsweise die Informationen an die GUI weiterzugeben.

5.2 GUI

OpenTOSCA ist ein Programm, das komplett über eine REST-Schnittstelle gesteuert wird. Eine grafische Repräsentation der Funktionalität ist nicht nativ eingebunden und vorgesehen. Dieses Kapitel zeigt die Implementierung einer GUI, die auf einem Webserver betrieben werden kann.

Die alte GUI ist während des vorhergegangen Studienprojekts entstanden, um die damalige Funktionalität von OpenTOSCA anzubieten. Um eine dynamische GUI zu erhalten, wurde jQuery (12) eingebunden. Ein Framework, das mehr als nur JavaScript- und AJAX-Unterstützung bietet, wurde nicht verwendet. Im Hinblick auf eine Weiterentwicklung ist diese Entscheidung ungünstig, da es viele gute Frameworks gibt, die die Entwicklung sehr erleichtern. Daher ist die GUI neu und mit der Unterstützung von Frameworks implementiert, die nicht nur allein den Einsatz von JavaScript erleichtern.

Abbildung 23: Alte GUI in der Ansicht „Deploy CSAR“

5.2.1 Struts 2

Ziel der Neuimplementierung ist es, eine Webapplikation zu erstellen, die leicht zu erweitern ist. Struts 2 ist ein freies, kostenloses OpenSource-Framework, das für das Erstellen von Java-Webanwendungen gedacht ist. (13) Es bietet Unterstützung für viele gängigen Webtechnologien, welche in den „Key Technologies“ mit den üblichen Verdächtigen tituliert. (14) Auch kann man Struts 2 mit Plug-Ins erweitern, um die Funktionalität zu erweitern.

5.2.2 Aufbau der GUI

Das Design der alten GUI ist komplett übernommen. Einzig die einzelne JSP ist in mehrere Einzelne aufgeteilt, um die Übersichtlichkeit und Erweiterbarkeit zu fördern. Programmatisch ist der Aufbau der GUI mit Tiles implementiert. Tiles ist in Form des „struts2-tiles-plugin“ (15) in der Version 2.3.8 eingebunden. Neben dem Header, Footer und der Menüleiste für die drei Ansichten ist der Inhaltsteil wie folgt aufgeteilt:

Eine JSP stellt die linke Spalte dar, die die in OpenTOSCA geladenen CSARs anzeigt. Die JSP des Overview-Bereichs mit dem Bild der Topologie und der Möglichkeit, sich die Dateistruktur innerhalb der CSAR anzuschauen, ist um die Information erweitert, ob die CSAR in OpenTOSCA korrekt verarbeitet wurde. Dazu wird der interne Status einer CSAR abgefragt, mit dem OpenTOSCA darstellt, ob bei der CSAR sowohl die Verarbeitung der TOSCA-Informationen und die Installation der Implementation Artifacts und BPEL-Pläne erfolgreich waren. Die About-Ansicht hat sich dagegen nicht geändert.

Abbildung 24: GUI Management-Ansicht mit laufendem Build-Plan für eine CSAR-Instanz

In der Liste der geladenen CSARs kann auf das grüne Plus-Zeichen geklickt werden, um einen Dialog aufzurufen, der die Input-Parameter für den Build-Plan abfragt. Analog dazu wird über einen Klick auf das rote Minus-Zeichen in der Liste der CSAR-Instanzen der Termination-Plan aufgerufen. Folgendes Bild zeigt eine erfolgreich instanziierte CSAR und eine Instanz, deren

Build-Plan noch nicht fertig ist. Eine Prozessinstanz eines Build-Plans wird durch ein sich drehendes Rädchen vor der Instanz-Beschriftung in der CSAR-Instanz-Liste angezeigt.

Der Bereich „Available Management Plans“ listet die verfügbaren PublicPlans auf. Durch einen

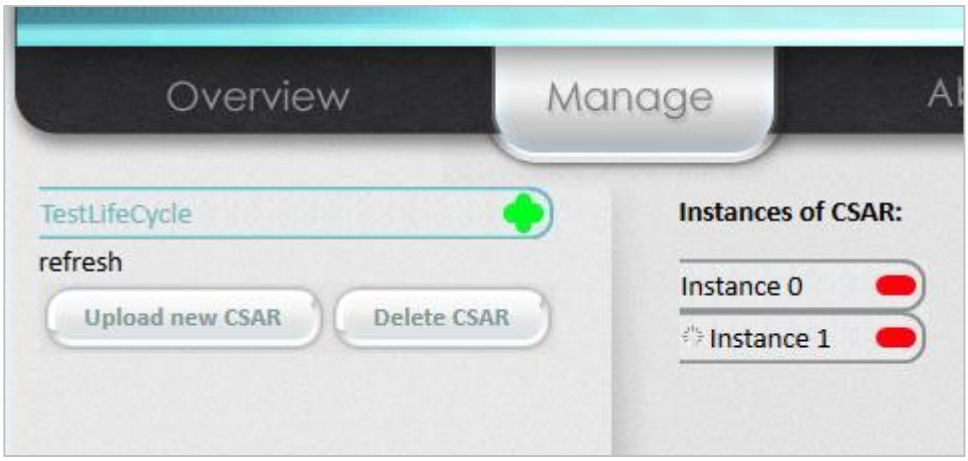

Abbildung 25: GUI Nahaufnahme einer fertigen CSAR-Instanz und einer mit laufendem Build-Plan

Klick darauf wird ein Dialog aufgerufen, der die Input-Parameter abfragt. Der Bereich „Information About Instance“ zeigt drei Informationen über die selektierte CSAR-Instanz an. Zuerst werden Informationen über die Installation der Instanz angezeigt. Nachfolgend werden zuerst die aktiven Prozessinstanzen und danach die abgeschlossenen Prozessinstanzen angezeigt.

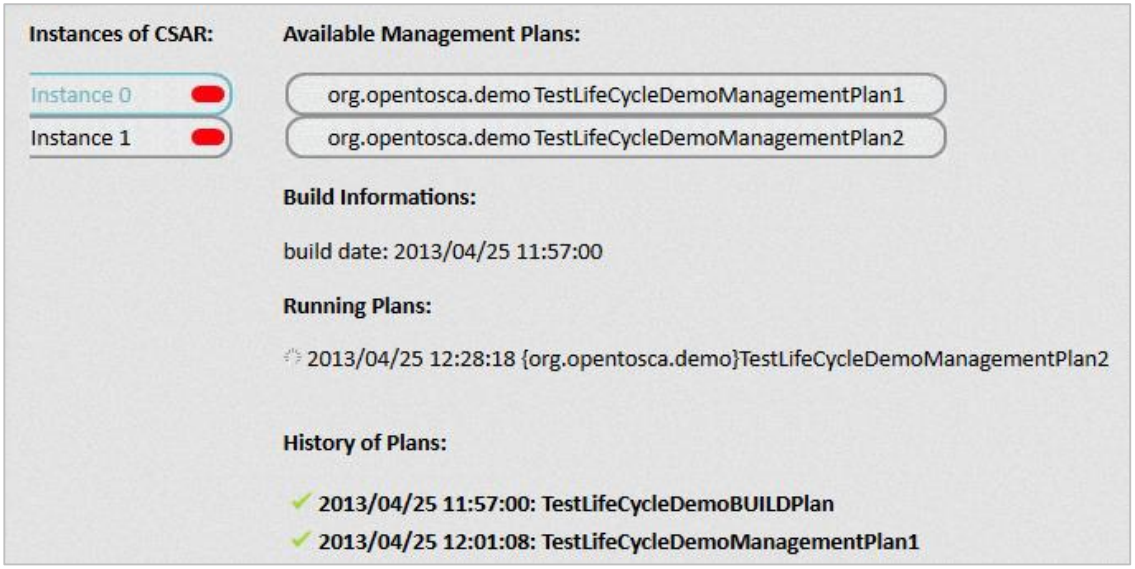

Abbildung 26: Informationen einer CSAR-Instanz inder GUI

PublicPlans in der History haben vor ihrem Namen entweder einen grünen Haken, der anzeigt, dass der Managementplan ohne Fehler zu Ende ausgeführt wurde. Ein rotes Kreuz symbolisiert einen Fehler.

Durch einen Klick auf einen PublicPlan-Eintrag der History können Informationen über den PublicPlan eingeblendet werden. Ein Klick auf den Knopf „invoke again“ ruft den Dialog für die Input-Parameter auf, in dem die Felder entsprechend der alten Ausführung ausgefüllt sind.

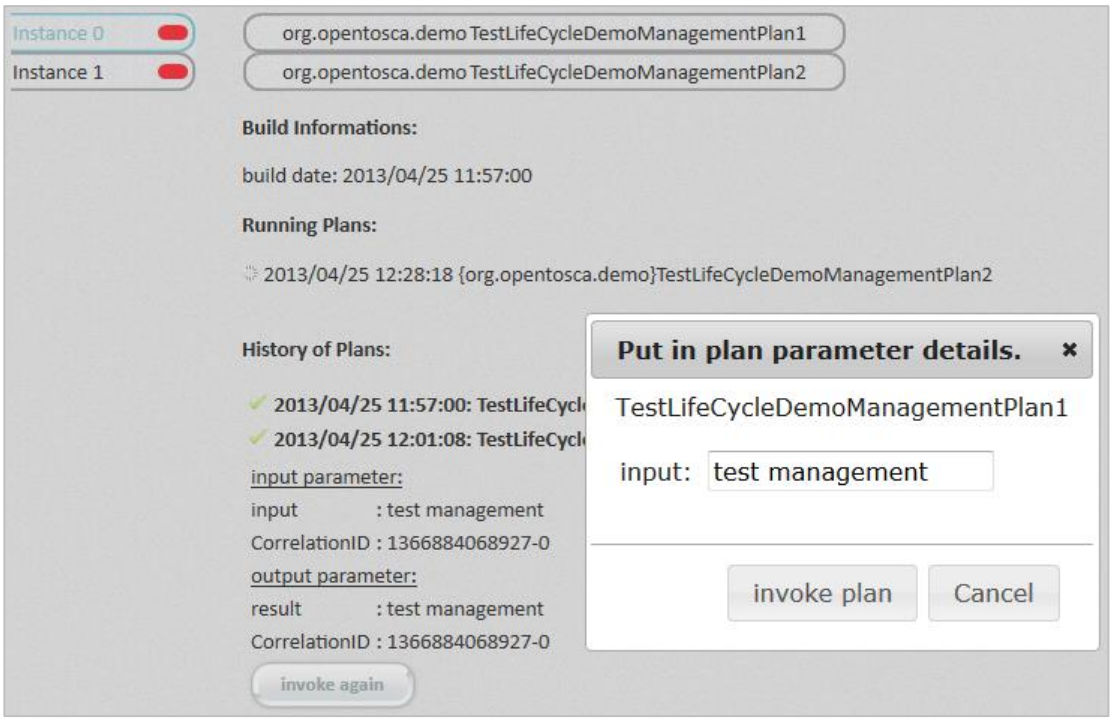

Abbildung 27: Erneutes Ausführen eines Managementplans in der GUI

5.2.3 Actions und Interceptors

Die JSPs selbst können nicht mit OpenTOSCA kommunizieren. In der alten GUI wurden Anfragen an OpenTOSCA per AJAX an Servlets geschickt. Diese verwendeten die Klasse ContainerClient, um die Anfrage an OpenTOSCA weiterzuleiten und gaben das Ergebnis an die AJAX-Anfrage zurück.

Das Prinzip mit Struts 2 ist ähnlich, denn in der aktuellen Implementierung werden über AJAX Anfragen an den Webserver geschickt, die wiederum an OpenTOSCA weitergeleitet werden. Jedoch kommen die Vorteile von Struts 2 zu tragen. Für Struts 2 implementiert man keine Servlets, sondern Actions. Diese können bequem über AJAX Anfragen angesteuert werden. Die sogenannten Interceptors übernehmen beispielsweise die Verarbeitung von den übergebenen Parameter oder das Formatieren der Rückgabe. Interceptors kann man zwar

überladen, für die aktuelle Implementierung wurden die Standard-Interceptors nicht geändert.

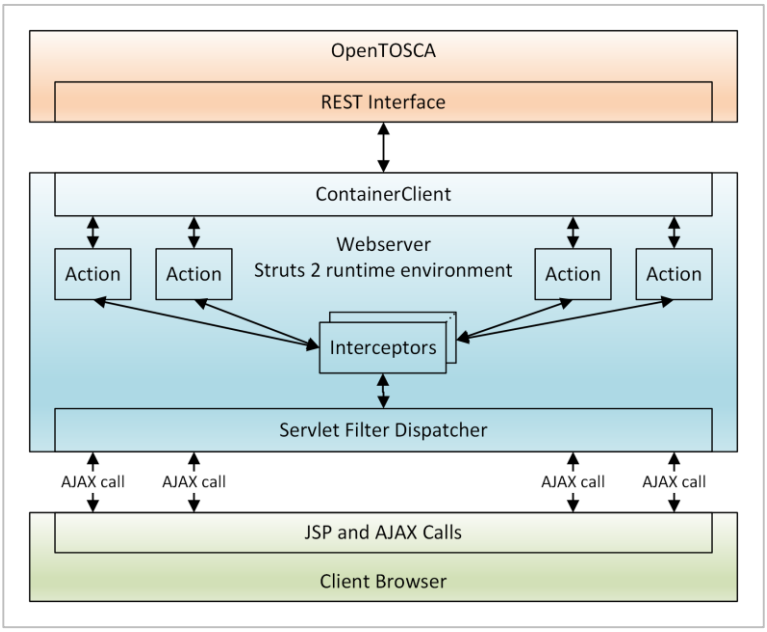

Abbildung 28: Funktionsweise der GUI in Zusammenspiel mit Struts 2

Nachfolgend werden in der Tabelle die implementierten Actions erklärt. Viele Rückgabewerte sind im JSON-Format, für dessen Verarbeitung das „struts2-json-plugin“ in der Version 2.3.8 eingebunden. (16) Für den Datentransfer der CSAR ist Jersey Multipart in der Version 1.11 eingebunden.

Action Name	Beschreibung des Zwecks
index	Diese Action baut über Tiles das Template auf und liefert die GUI an den Client.
uploadCSAR	Diese Action wurde aus einem Servlet der alten GUI übernommen. Die alte GUI schickte den lokalen Pfad einer CSAR an OpenTOSCA. Die Implementierung sowohl im ContainerClient, als auch in OpenTOSCA wurde dahingehend geändert, dass das durch Struts 2 bereitgestellte File-Objekt der CSAR als „Multipart-Form-Data“ an OpenTOSCA geschickt wird. Dadurch können Webserver und OpenTOSCA auf unterschiedlichen Computer laufen. Nach der Ausführung der Action wird eine „redirectAction“ ausgeführt, welche den Client an die index-Action weiterleitet.
deleteCSAR	Diese Action wurde aus einem Servlet der alten GUI übernommen und löst die Löschung einer CSAR in Open-TOSCA aus. Hierzu wurden keine Anpassungen vorgenommen.
getBuildPlanFromHistory	Diese Action bezieht den ersten Build-Plan aus OpenTOSCA. Zusätzlich wird aus der CorrelationID der Zeitpunkt der Ausführung berechnet. Die Daten werden im JSON-Format zurückgegeben.

getCSARContent	Diese Action wurde aus einem Servlet der alten GUI übernommen. Die Funktionsweise wurde etwas angepasst. Die Daten werden im JSON-Format zurückgegeben.
getCSARUploadInformations	Diese Action wurde aus der alten GUI übernommen und erfragt bei OpenTOSCA den Verarbeitungsstatus einer CSAR. Der Status zeigt an, ob die TOSCA-Daten erfolgreich verarbeitet wurden, die Implementation Artifacts installiert und die BPEL-Pläne installiert sind. Der letzte erfolgreich durchgeführte Arbeitsschritt wird als Text im JSON-Format zurückgegeben.
getHistory	Diese Action erfragt bei OpenTOSCA die PublicPlans, die für eine bestimmte CSAR-Instanz gespeichert sind. Die Public-Plans werden im JSON-Format zurückgegeben.
getRunningPlansInformations	Diese Action erfragt bei OpenTOSCA die PublicPlans, die gerade für eine bestimmte CSAR-Instanz ausgeführt werden. Die PlanIDs der PublicPlans werden im JSON-Format zurückgegeben.
getSpecificPlanFromHistory	Diese Action erfragt bei OpenTOSCA einen PublicPlan für eine CorrelationID. Der PublicPlan wird im JSON-Format zurückgegeben.
postBUILDInvocation	Diese Action nimmt die Input-Parameter für einen PublicPlan entgegen und speichert sie in einen PublicPlan. Weiterhin werden in den PublicPlan Informationen gespeichert, die für die Identifizierung des Build-Plans notwendig sind. Über den ContainerClient wird dieser PublicPlan an OpenTOSCA geschickt, um eine neue CSAR-Instanz zu instanziieren. Das Rückgabeformat ist JSON.
postMANAGEMENTInvocation, postTERMINATIONInvocation	Diese Actions nehmen die Input-Parameter für einen PublicPlan entgegen und speichern sie in einen PublicPlan. Weiterhin werden in den PublicPlan Informationen gespeichert, die für die Identifikation des Managementplans notwendig sind. Über den ContainerClient wird dieser PublicPlan an OpenTOSCA geschickt, um die Ausführung des entsprechenden Managementplans zu instanziieren. Das Rückgabeformat ist JSON.

Tabelle 2: Struts 2 Actions

5.2.4 AJAX durch jQuery

Um ein dynamisches Verhalten der GUI zu ermöglichen, ist jQuery in der Implementierung eingebunden. Für Struts 2 gibt es direkt ein Plug-In, das im Struts 2-Stil auch eigene Elemente über eine „tag library“ anbietet. Die eingebundene Version ist 3.5.1. (17)

5.2.5 Kommunikation zwischen dem Client und dem Webapplikation

Eine Action greift auf die statische Instanz der Klasse ContainerClient zu, um mit OpenTOSCA zu kommunizieren. Der ContainerClient ist intern eine Instanz eines „com.sun.jersey.api.client.Client“ und war in der alten GUI schon in dieser Form vorhanden.

Entsprechend der neuen Funktionalität und den neuen Actions ist der ContainerClient erweitert worden.

Die ContainerAPI unterstützt mehrere Datenformen für die Kommunikation: „text/plain“, „text/xml“, „application/xml“, „image/*“ und „application/octet-stream“. Um im Baum der REST-Schnittstelle CSARs und CSAR-Instanzen, sowie PublicPlans zu traversieren, schickt die ContainerAPI „text/xml“ zurück. Um aus den JAXB-annotierten Klassen der PublicPlans XML-Dokumente zwischen dem ContainerClient und der ContainerAPI auszutauschen, wird „application/xml“ verwendet. Im Format „application/octet-stream“ schickt der ContainerClient der ContainerAPI die Daten einer CSAR-Datei. Die Pfade zu den einzelnen Ressourcen entsprechen der in Kapitel 5.1.4 beschriebenen Baumstruktur der ContainerAPI.

6 Schlusswort

6.1 Zusammenfassung

OpenTOSCA bietet als erste OpenSource-Referenzimplementierung die Möglichkeit durch TOSCA beschriebene CSARs zu verarbeiten und in die Cloud zu installieren.

Diese Bachelorarbeit beschreibt Konzepte, wie mit OpenTOSCA CSARs durch Managementpläne installiert und geändert werden können. Weiterhin wird der Managementaspekt betrachtet, der die Verwaltung der resultierenden Instanzen aus CSARs und Managementplänen umfasst. Um dies in OpenTOSCA zu implementieren, sind die nötigen Erweiterungen konzeptionell beschrieben.

Das Kapitel Implementierung zeigt auf, wie OpenTOSCA und die GUI im Zuge des praktischen Teils dieser Bachelorarbeit erweitert wurde. OpenTOSCA ist um mehrere Komponenten und OSGI-Services reicher, die Instanzen von CSARs und Managementplänen instanziieren, verwalten und auch wieder löschen. Um dies auch einem menschlichen Nutzer möglichst komfortable zur Verfügung zu stellen, ist die GUI zu großen Teilen ersetzt worden.

6.2 Ausblick

Diese Bachelorarbeit erarbeitet die Grundbausteine für das Management von Cloud Applikationen: Erstellen, Ändern und Löschen. Dahingehend wurde der Prototyp OpenTOSCA erweitert. Das Ergebnis ist weiterhin ein Prototyp und bietet daher viel Potential zur Weiterentwicklung.

OpenTOSCA ermöglicht eine sehr einfache automatisierte Verwaltung von CSAR-Instanzen. Eine Cloud Applikation kann eine Topologie aus mehreren virtuellen Computern darstellen. Bisher wird eine Instanz einer CSAR jedoch noch nicht in ihre logische Bestandteile aufgeteilt, die sich beispielsweise durch die einzelnen beinhalteten Service Templates definieren. Das Management eines Teils einer Topologie geschieht momentan nur implizit, in dem der korrekte Managementplan ausgewählt wird. Informationen über die Webadresse der Website

und des Datenbankservers einer imaginären Webshop-CSAR können aktuell höchsten aus der Response-Nachricht ausgelesen werden. Eine Aufbereitung dessen für den Nutzer fehlt noch.

OpenTOSCA kennt CSAR-Instanzen nur als abstraktes Konstrukt und nicht als Gegenstand der realen Welt. Eine Anbindung an einen Cloud Anbieter ist noch nicht vorgesehen. Wenn beispielsweise eine CSAR-Instanz manuell gelöscht wurde, sollte diese auch in OpenTOSCA nicht mehr aufgeführt werden. Auf die Verwaltung der CSAR-Instanzen kann weder manuell Einfluss genommen werden, noch werden Informationen automatisiert aus der Cloud abgerufen.

Die Bachelorarbeit betrachtet konzeptionell Managementpläne generisch, da TOSCA verschiedenen Managementplan-Sprachen erlaubt. OpenTOSCA unterstützt jedoch nur BPEL. PublicPlans sehen die in TOSCA definierten Informationen über mögliche Managementpläne vor. Kontextsensitive Informationen werden dem Nutzer nicht angeboten: Der Nutzer muss anhand der ID eines Managementplans wissen, was dieser schlussendlich bewirkt. Der Nutzer kann auch keinen Einfluss auf Managementpläne nehmen. Das Abbrechen eines Managementplans und vor allem das Wiederherstellen eines vorherigen Zustands werden dem Nutzer nicht angeboten.

Die Boundary Definitions beinhalten auch nicht nur Managementpläne, die der Nutzer ausführen kann. Die Operation eines Interface kann auch auf Operationen von Node Templates und Relationship Templates zeigen. TOSCA sieht vor, Cloud Applikationen auch durch diese Operation-Elemente zu managen.

7 Literaturverzeichnis

1. **OASIS.** TOSCA Spezifikation. *Topology and Orchestration Specification for Cloud Applications Version 1.0.* [Online] [Zitat vom: 17. 04 2013.] http://docs.oasis-open.org/tosca/TOSCA/v1.0/csprd01/TOSCA-v1.0-csprd01.html.

2. **OpenTOSCA.** OpenTOSCA - Open Source Laufzeitumgebung für TOSCA. [Online] [Zitat vom: 24. 04 2013.] http://www.iaas.uni-stuttgart.de/OpenTOSCA/.

3. **WSO2.** WSO2 Business Process Server. [Online] [Zitat vom: 24. 04 2013.] http://wso2.com/products/business-process-server/.

4. **Apache Software Foundation.** Apache Tomcat. [Online] [Zitat vom: 24. 04 2013.] http://tomcat.apache.org/.

5. **Oracle.** Jersey. [Online] [Zitat vom: 24. 04 2013.] https://jersey.java.net/.

6. —. JAXB Reference Implementation. [Online] [Zitat vom: 24. 04 2013.] http://jaxb.java.net/.

7. **OASIS.** TOSCA Specification Public Review Draft 01 XML Schema. [Online] [Zitat vom: 14. 01 2013.] http://docs.oasis-open.org/tosca/TOSCA/v1.0/csprd01/schemas/TOSCA-v1.0.xsd.

8. —. BPEL-Spezifikation. *Web Services Business Process Execution Language Version 2.0.* [Online] [Zitat vom: 20. 04 2013.] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738499.

9. **OSGI Alliance.** OSGI Events. *Package org.osgi.service.event.* [Online] [Zitat vom: 23. 04 2013.] http://www.osgi.org/javadoc/r4v41/org/osgi/service/event/package-summary.html.

10. **Eclipse Foundation.** Equinox SDK 3.7.1. *Equinox Release Build: 3.7.1.* [Online] [Zitat vom: 23. 04 2013.] http://download.eclipse.org/equinox/drops/R-3.7.1-201109091335/index.php.

11. **Oracle.** API Beschreibung der Klasse System. *java.lang Class System.* [Online] [Zitat vom: 23. 04 2013.] http://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime%28%29.

12. **jQuery Foundation.** jQuery. *jQuery.* [Online] [Zitat vom: 23. 04 2013.] http://jquery.com/.

13. **Apache Software Foundation.** Struts 2 Website. [Online] [Zitat vom: 03. 04 2013.]

14. —. Key Technologies Primer. [Online] [Zitat vom: 23. 04 2013.] http://struts.apache.org/primer.html.

15. —. Tiles Plugin. [Online] [Zitat vom: 24. 04 2013.] http://struts.apache.org/release/2.1.x/docs/tiles-plugin.html.

16. —. JSON Plugin. [Online] [Zitat vom: 24. 04 2013.] http://struts.apache.org/release/2.1.x/docs/json-plugin.html.

17. **"johgep@gmail.com", owner.** struts2-jquery. [Online] [Zitat vom: 24. 04 2013.] http://code.google.com/p/struts2-jquery/.

18. **www.freeiconsweb.com.** Loading Animated Gif. [Online] [Zitat vom: 15. 04 2013.] http://www.freeiconsweb.com/Free-Downloads.asp?id=585.

19. —. SweetiePlus Icons. [Online] [Zitat vom: 15. 04 2013.] http://www.freeiconsweb.com/Free-Downloads.asp?id=1894.

20. **Breitenbücher, Uwe.** IAAS | Uwe Breitenbücher. [Online] [Zitat vom: 24. 04 2013.] http://www.iaas.uni-stuttgart.de/institut/mitarbeiter/breitenbuecher/index.php.

21. **Deininger, Marcus, et al.** *Studien-Arbeiten.* Zürich : vdf Hochschulverlag, 2002. ISBN 3-7281-2815-5.

22. **Rechenberg, Peter.** *Technisches Schreiben.* München : Carl Hanser Verlag, 2006. ISBN 3-446-40695-6.

8 Nachweis über Rechte dritter Personen

Weiterhin sind für die Implementierung Programme und Ressourcen dritter Personen verwendet worden. Um auch dies zu dokumentieren, listet das Kapitel **Fehler! Verweisquelle onnte nicht gefunden werden.** im Anhang die Programme und Bilder auf.

8.1.1 Grafiken

Die folgenden drei Grafiken sind in der GUI eingebunden.

Die Grafik  ist von der Seite (18) bezogen. Die Grafik ist ohne Einschränkung durch Lizenzbestimmungen veröffentlicht worden und darf laut der „Read-Me“-Datei völlig frei verwendet werden: „You are free to use these icons in any way you wish.“.

Die Grafiken  und  sind von der Seite (19) bezogen. Beide Grafiken sind unter der Lizenz „Creative Commons Attribution-ShareAlike 3.0 Unported“ veröffentlicht worden.

Die Grafik für das Anzeigen der Management-Ansicht in der GUI wurde von Uwe Breitenbücher (20) erstellt und unter die selbe Lizenz gestellt, unter der OpenTOSCA entwickelt wird: Apache License, Version 2.0.

8.1.2 Programme

Die Programme in folgender Liste wurden für die Erstellung dieses Dokuments, Bearbeitung von Grafiken und Programmierung oder durch Einbindung in die Implementierung (soweit nicht schon im Text angegeben) verwendet:

- Eclipse in den Versionen Helios und Juno
- Apache Maven und das Eclipse Plug-In m2e
- Paint.NET v3.5.10 (Lizenz: Attribution-NonCommercial 3.0 United States)
- Microsoft Office 2013

9 Anhang

9.1 Glossar

Boundary Definitions: „This OPTIONAL element specifies the properties the Service Template exposes beyond its boundaries, i.e. properties that can be observed from outside the Service Template. […]“ (1).

BPEL-Plan: Ein Managementplan, der in der Sprache BPEL notiert ist.

BPELPlanInvoker: Dies ist eine Klasse der OpenTOSCA-Komponente Servicebus. Der BPELPlanInvoker empfängt einen SOAP Body als OSGI Event und initiiert die Aktivierung eines BPEL-Plans.

BPS: „WSO2 Business Process Server“ (3)

Build-Plan: Ein Managementplan, dessen Attribut „planType“ den Wert „http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan“ enthält. Durch diesen Begriff wird ein Managementplan mit Installationsverhalten assoziiert. „[…] plans used to initially create a new instance of a service from a Service Template.“ (1).

Cloud Applikation: Eine oder mehrere Programme, die in Komposition zueinander von OpenTOSCA als ein Service in eine beliebige Cloud installiert und dort gemanagt werden soll. Ein Beispiel wäre eine Website, ein Webserver, ein Datenbankserver und das zugrundeliegende Betriebssystem.

ContainerAPI: Die ContainerAPI ist die Komponente in OpenTOSCA, über die man OpenTOSCA von außen ansprechen kann. Diese Komponente entspricht der View des MVC-Patterns.

ContainerClient: Diese Klasse ist ein Client der ContainerAPI und verhilft der GUI dazu, mit OpenTOSCA kommunizieren zu können.

CorrelationHandler: Diese Klasse der PlanInvocationEngine generiert zum einen die CorrelationID und merkt sich zum anderen CorrelationIDs von PublicPlans, für die noch keine Response-Nachricht verarbeitet wurde.

CorrelationID: Diese spezifische ID wird vom CorrelationHandler generiert, wenn die PlanInvocationEngine eine Request-Nachricht für einen PublicPlan erstellen lässt. Die CorrelationID dient dazu, eine spätere Response-Nachricht der vorhergehenden Request-Nachricht zuzuordnen.

CSAR: „Cloud Service Archive“ (1)

CSARInstanceManagementService: Diese Komponente verwaltet die CSAR-Instanzen und die dazugehörigen Informationen. Ausgenommen hiervon sind aktivierte PublicPlans, die noch keine Response-Nachricht zurückgeschickt haben. Diese Datenobjekte werden vom CorrelationHandler gespeichert.

CSAR-Instanz: Sobald eine CSAR durch OpenTOSCA verarbeitet und ein Build-Plan für diese CSAR erfolgreich ausgeführt wurde, ist der daraus resultierende Service in der Cloud eine CSAR-Instanz.

DefinitionsConsolidation: Die Klasse der ToscaEngine, die nach dem Auflösen der Referenzen in TOSCA die Konsolidierung von TOSCA-Daten initiiert. Bisher wird nur die Klasse ExportedInterfacesConsolidition ausgeführt.

EventAdmin: Ein OSGI-Service, der von Klassen in OpenTOSCA gebunden wird, um OSGI-Events auf den Event-Listen zu veröffentlichen.

EventHandler: Ein durch OSGI definiertes Interface, das Klassen in OpenTOSCA implementieren, um Events auf abonnierten Event-Listen vom EventAdmin zu erhalten.

ExportedInterfacesConsolidition: Die Klasse, welche die DefinitionsConsolidation aufruft, um Managementpläne in Verbindung mit den Boundary Definitions zu konsolidieren. Das Resultat nennt sich PublicPlan.

GUI: Die englische Abkürzung für „graphical user interface“. Hier müssen zwei verschiedene GUIs voneinander getrennt betrachtet werden: Die alte GUI zum Anfang der Bachelorarbeit wurde während der Bachelorarbeit von der neuen GUI ersetzt.

History: Eine geforderte Funktionalität, die ausgeführte Managementpläne mit ihren Ein- und Ausgabewerten speichert.

Management: Dieser Begriff umfasst in dieser Bachelorarbeit Tätigkeiten, die eine CSAR-Instanz verändern. Dies umschließt den kompletten Lebenszyklus der CSAR-Instanz, also auch die Installation der CSAR-Instanz.

Managementplan: Ein Managementplan wird von TOSCA durch das Plan-Element in einem Service Template definiert.

Mock-up: Der englische Begriff für Attrappe umschreibt hier eine Komponente, die das Interface einer anderen, noch nicht entwickelten Komponente anbietet und die Funktionsweise der noch nicht vorhandenen Komponente simuliert.

Nutzer: Der Nutzer in diesem Dokument ist eine imaginäre, menschliche Person, welche die Funktionalität von OpenTOSCA nutzt.

OpenToscaControl: Die Komponente in OpenTOSCA, welche die Control-Funktionalität des MVC-Patterns implementiert.

PlanInvocationEngine: Die Komponente in OpenTOSCA, welche aus den Nutzereingaben die Aktivierung eines Managementplans initiiert.

PublicPlan: Das abstrakte Konstrukt enthält konsolidierte Daten aus einem Managementplan und den Boundary Definitions.

Servicebus: Ein Mock-up einer Komponente, welche eine Schnittstelle für die Kommunikation zwischen OpenTOSCA und dem BPS anbietet.

Termination-Plan:	Ein Managementplan, dessen Attribut „planType“ den Wert „http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan“ enthält. Durch diesen Begriff wird ein Managementplan mit Löschverhalten assoziiert. „[…] plans used to terminate the existence of a service instance.“ (1).
ToscaEngine:	Die Komponente in OpenTOSCA, die TOSCA-Dokumente verarbeitet.

Tabelle 3: Glossar

9.2 XML Schema PublicPlans

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.opentosca.org/PublicPlansOfCSAR"
     xmlns:tns="http://www.opentosca.org/PublicPlansOfCSAR"
     elementFormDefault="qualified">

     <xs:element name="PublicPlans">
        <xs:complexType>
           <xs:sequence>
              <xs:element ref="tns:PublicPlan" minOccurs="0"
                 maxOccurs="unbounded" />
           </xs:sequence>
        </xs:complexType>
     </xs:element>

     <xs:element name="PublicPlan">
        <xs:complexType>
           <xs:sequence>
              <xs:element name="InputParameter" type="tns:Parameter"
                 minOccurs="1" maxOccurs="unbounded" />
              <xs:element name="OutputParameter" type="tns:Parameter"
                 minOccurs="1" maxOccurs="unbounded" />
           </xs:sequence>

           <xs:attribute name="CSARID" type="xs:string" use="required" />
           <xs:attribute name="PlanType" type="xs:anyURI" use="required" />
           <xs:attribute name="InternalPlanID" type="xs:int" use="required" />
           <xs:attribute name="PlanID" type="xs:QName" use="required" />
           <xs:attribute name="InternalInstanceInternalID" type="xs:int"
              use="required" />
           <xs:attribute name="InterfaceName" type="xs:string" use="required" />
           <xs:attribute name="OperationName" type="xs:string" use="required" />
           <xs:attribute name="InputMessageID" type="xs:QName" use="required" />
           <xs:attribute name="OutputMessageID" type="xs:QName" use="required" />
           <xs:attribute name="PlanLanguage" type="xs:anyURI" use="required" />
           <xs:attribute name="isActive" type="xs:boolean" use="required" />
           <xs:attribute name="hasFailed" type="xs:boolean" use="required" />
        </xs:complexType>
     </xs:element>

     <xs:complexType name="Parameter">
        <xs:simpleContent>
           <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string" use="required" />
              <xs:attribute name="type" type="xs:string" use="required" />
              <xs:attribute name="required" type="xs:boolean" use="required" />
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>

</xs:schema>
```

9.3 XML Schema OpenTOSCAElements

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3       targetNamespace="http://www.opentosca.org/Correlation"
4       elementFormDefault="qualified">
5
6       <xs:element name="CorrelationID" type="xs:string" />
7       <xs:element name="CallbackAddress" type="xs:anyURI" />
8
9  </xs:schema>
```

10 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.

Stuttgart, den 03.05.2013

Ort, Datum Christian Endres