Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 283

Monitoring Frontend für OpenTOSCA

Sebastian Bartenbach

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Dr. h. c. Leymann
Betreuer/in:	Dipl.-Inf. Florian Haupt
Beginn am:	10.11.2015
Beendet am:	11.05.2016
CR-Nummer:	C.2.4, H.3.5, H.5.2, H.5.4

Kurzfassung

Die *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [10] ist eine Sprache zur portablen und interoperablen Beschreibung von Cloud-Anwendungen. Diese Beschreibung umfasst deren Topologie, Verwaltung und Deployment. Sie ermöglicht es, Anwendungen automatisiert bei verschiedenen Cloud-Anbietern zu betreiben und zu managen.

OpenTOSCA [8] ist eine Open-Source Implementierung eines TOSCA Containers, die in den letzten Jahren an der Universität Stuttgart entwickelt wurde. Sie bietet eine Laufzeitumgebung für TOSCA-Anwendungen. Die Provisionierungs- und Verwaltungsprozesse, die von einem OpenTOSCA-Container angestoßen werden, sind oftmals komplex und laufen über einen längeren Zeitraum. In dieser Zeit werden Informationen über den Ablauf und Zustand dieser Prozesse generiert, die dem Nutzer allerdings aktuell nicht direkt angezeigt werden.

Während der Provisionierung einer Anwendung durch OpenTOSCA werden Ereignisdaten generiert, in einem Monitoring System abgelegt und über eine REST API [5] zur Verfügung gestellt. Diese Bachelorarbeit untersucht die Erweiterung des Monitorings um eine grafische Nutzerschnittstelle, die diese Informationen für den Nutzer zugänglich macht. Die Besonderheit besteht hierbei darin, dass das Monitoring Backend unterschiedliche Typen von Events verwaltet. Diese unterliegen einer Typhierarchie, wobei sich alle Events von einem Basistyp ableiten, der erweitert werden kann. Die Nutzerschnittstelle soll diese Besonderheit ebenfalls sinnvoll unterstützen.

Der praktische Teil der Arbeit befasst sich dabei mit der Implementierung zweier Benutzeroberflächen, sowohl für den Basistyp dieser Events, als auch eine speziell auf OpenTOSCA zugeschnittene Oberfläche. Eine prototypische Implementierung der OpenTOSCA GUI existiert bereits und kann als Ausgangspunkt für diese Arbeit verwendet werden.

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Ziel der Bachelorarbeit . . . 7
1.2 Aufbau der Bachelorarbeit . . . 7

2 Grundlagen **9**
2.1 REST . . . 9
2.2 TOSCA . . . 12
2.2.1 Topology . . . 12
2.2.2 Orchestration . . . 12
2.2.3 OpenTOSCA . . . 13

3 Ist- und Soll-Zustand **15**
3.1 Anforderungen . . . 15
3.1.1 Anforderungen an das generische Frontend . . . 15
3.1.2 Anforderungen an das OpenTOSCA Frontend . . . 16
3.2 Bisheriger Prototyp . . . 18
3.3 Analyse . . . 18
3.3.1 Generisches Frontend . . . 18
3.3.2 OpenTOSCA Frontend . . . 18

4 Entwurf **21**
4.1 Generisches Frontend . . . 21
4.2 OpenTOSCA Frontend . . . 22

5 Implementierung **23**
5.1 Frameworks . . . 23
5.1.1 jQuery . . . 23
5.1.2 Bootstrap . . . 23
5.2 Generisches Frontend . . . 23
5.2.1 Darstellung . . . 24
5.2.2 Technische Umsetzung . . . 25
5.3 OpenTOSCA Frontend . . . 28
5.3.1 Darstellung . . . 28
5.3.2 Technische Umsetzung . . . 32

6 Zusammenfassung und Ausblick **37**

Tabellenverzeichnis

1 Pflichtfelder für jeden Eventtyp 25

Abbildungsverzeichnis

2.1 Ableitung der REST Architektur [5] 10
5.1 Generisches Frontend mit einem „basic“ Event 24
5.2 Generisches Frontend mit gemischten Eventtypen 26
5.3 Architektur des generischen Frontends 27
5.4 OpenTOSCA Frontend mit zwei *Node Instances* 30
5.5 Überlappend gestapelte Node Instances 31
5.6 Node Instance mit darüber gelegtem *Properties* Pop-up 32

Listings

1 JSON Datenstruktur für generische Events 28
2 Initialisierung des DataTables Plugins 29
3 JSON Datenstruktur für OpenTOSCA Events 35
4 JSON Datenstruktur für eine Node Instance in OpenTOSCA . . 35

1 Einleitung

Dieses Kapitel beschreibt die Ziele dieser Arbeit und bietet eine kurze Übersicht ihres Aufbaus.

1.1 Ziel der Bachelorarbeit

Das Ziel der Arbeit ist es, die bisherige Monitoring-GUI für OpenTOSCA-Events zu ersetzen und zu erweitern. Dafür soll eine neue Implementierung mit erweiterter Funktionalität umgesetzt werden. Diese Implementierung soll übersichtlichen und wartbaren Code haben und nur wenige, begründete externer Tools oder Frameworks verwenden. Zusätzlich soll außerdem eine Variante der GUI realisiert werden, die generische Events in vereinfachter Form darstellen kann.

Das Monitoring dieser Events spaltet sich in zwei Komponenten. Ein bestehendes Backend, welches Events entgegen nimmt, persistent speichert und mittels einer REST-API zur Verfügung stellt.

Diese Arbeit widmet sich der zweiten Komponente in Form der Abfrage dieser Events und der sinnvollen Aufbereitung und Darstellung in Form einer Benutzeroberfläche.

1.2 Aufbau der Bachelorarbeit

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden die wichtigsten Grundlagen erklärt, unter anderem zu OpenTOSCA und REST.

Kapitel 3 – Ist- und Soll-Zustand: Hier werden der bisherige Stand der ehemaligen Monitoring-GUI erläutert sowie die Anforderungen an die Neuimplementierung und Erweiterung aufgezählt.

Kapitel 4 – Entwurf: Behandelt die Entwurfsentscheidungen und grundlegenden Überlegungen zur Entwicklung der neuen Benutzeroberfläche

Kapitel 5 – Implementierung: Erklärt die detaillierte Umsetzung der Implementierung der Monitoring-GUI für OpenTOSCA als auch der Oberfläche für generische Events.

2 Grundlagen

Dieses Kapitel beschreibt die relevanten Grundlagen, die für das Verständnis dieser Arbeit wichtig sind. Zunächst wird REST vorgestellt, da die Kommunikation mit dem Monitoring Backend über eine REST-API stattfindet. Darauf folgt eine Einführung in OpenTOSCA, da der zweite Teil des zu implementierenden Frontends eine Benutzeroberfläche speziell für OpenTOSCA Events ist.

2.1 REST

Representational State Transfer (kurz *REST*) ist ein Architekturstil für verteilte Hypermedia Systeme, der im Jahre 2000 von Roy Fielding in seiner Dissertation [5] eingeführt wurde. In dieser Arbeit analysiert er unterschiedliche Architekturstile hinsichtlich gewisser Eigenschaften, unter anderem ihrer Erweiterbarkeit, Einfachheit und Skalierbarkeit. Von diesen netzwerk-basierten Architekturen leitet er REST als einen hybriden Stil ab.

Um REST zu definieren, beschreibt Fielding zuerst einen *Null-Stil* und fügt diesem iterativ neue Einschränkungen hinzu, siehe Abbildung 2.1. Der Null-Stil selbst enthält keine Einschränkungen und dient als Ausgangspunkt, durch die Erweiterung der Einschränkungen gelangt Fielding letztendlich zu einer vollständigen Ableitung und somit der Definition von REST.

Die erste Beschränkung des Null-Stils findet durch die Einschränkung von REST auf den *Client-Server-Architekturstil* statt. Die Trennung in Client und Server, und somit deren Zuständigkeiten, ist dessen Kernprinzip. Dies erhöht die Skalierbarkeit und Portabilität, da der Client leichter auf ein anderes Zielsystem portiert werden kann, wohingegen der Server einfacher gestaltet werden kann und somit skalierbarer ist. Zusätzlich ermöglicht dies die voneinander unabhängige Entwicklung der Client- und Serverkomponenten.

Die zweite Einschränkung, die REST auferlegt wird, ist die *Statuslosigkeit* sämtlicher Kommunikation. Dies erhöht nach Fieldings Ansicht die Skalierbarkeit, Sichtbarkeit und Zuverlässigkeit. Da der Server keinen Status speichern muss, vereinfacht das zum einen dessen Implementierung, und zum anderen lässt sich problemlos ein skalierbares, verteiltes System umsetzen, bei dem ankommende Anfragen von unterschiedlichen Servern beantwortet werden können. Die Zuverlässigkeit wird erhöht, da Ausfälle zum Teil leichter „behoben" werden können, da es nicht passieren kann, dass ein gespeicherter Zustand verloren geht. Ein defekter Server kann beispielsweise durch ein *Fallback* ersetzt werden, ohne dass eine vorhergehende Synchronisierung innerhalb dieses Systems stattfinden muss. Die Sichtbarkeit wird erhöht, da ein Client sämtliche Informationen die nötig sind, damit der Server die Anfrage verstehen und korrekt bearbeiten kann, in vollem Umfang in seiner Anfrage mitschicken muss.

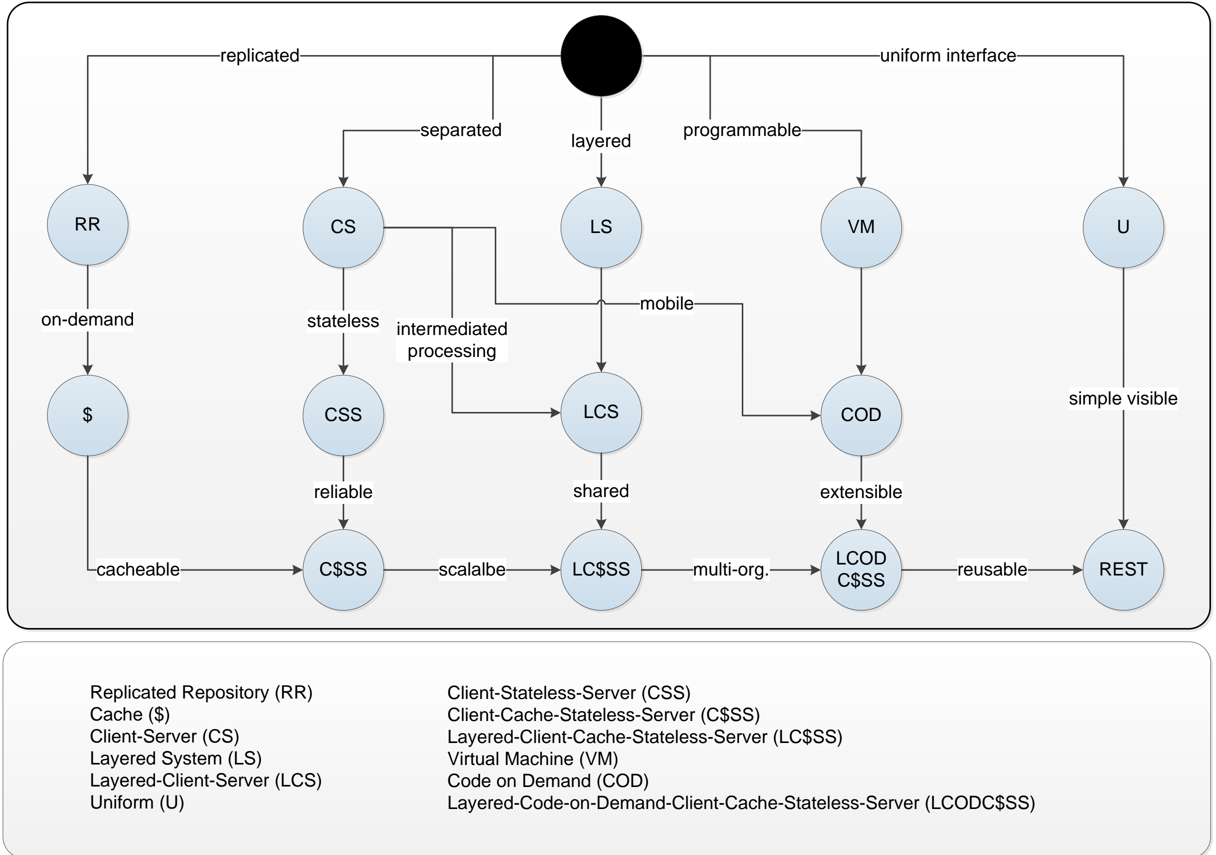

Abbildung 2.1: Ableitung der REST Architektur [5]

Diese Einschränkung birgt jedoch auch einige Nachteile. Zum einen verliert der Server die Kontrolle über die Konsistenz des Verhaltens der gesamten Anwendung. Diese Verantwortung obliegt somit ausschließlich dem Client, der dafür zuständig ist den Zustand zu halten. Zum anderen verringert sich der Durchsatz innerhalb des Netzwerks, da insbesondere sukzessive Anfragen einen *Overhead* erzeugen, weil gleiche Informationen mehrfach übermittelt werden müssen.
Um einen Teil dieser Nachteile zu verringern, wird als dritte Einschränkung ein *Caching* ermöglicht. Die Daten einer Antwort vom Server können damit als *cachable* oder *non-cachable* markiert werden. Sofern die Antwort auf eine Anfrage bereits im Cache eines Clients gespeichert ist, kann dieser verwendet werden, statt eine erneute Anfrage zu stellen. Dies entlastet sowohl den Server als auch das Netzwerk, dessen Durchsatz somit erhöht werden kann. Ferner kann sich der deutlich schnellere Zugriff auf den Client-Cache im Gegensatz zu der Verzögerung einer Serveranfrage positiv auf die Nutzererfahrung auswirken.

Durch Caching entsteht allerdings der Nachteil, dass möglicherweise alte Daten im Cache gehalten werden, wodurch sich signifikante Unterschiede im

Vergleich zur Antwort einer aktuellen Serveranfrage ergeben könnten.

Die vierte Einschränkung ist eine Weiterentwicklung der Client-Server-Einschränkung. Es handelt sich dabei um das *Layered System*, welches zur Unterteilung einer Architektur in hierarchische Schichten verpflichtet. Es beschränkt die Kommunikation aller Komponenten auf die direkt anliegenden Schichten. Dies kann insbesondere in Verbindung mit Caching zu einer erhöhten Performanz und somit besserer Skalierbarkeit führen, da eine zwischenliegende Komponente Anfragen bei weiteren Schichten vermeiden kann.

Die fünfte Einschränkung ist die der *einheitlichen Schnittstelle*. Sie ist die deutlichste Abhebung von anderen netzwerk-basierten Architekturstilen und laut Fielding das zentrale Feature von REST. Sie besagt, dass alle Komponenten eine einheitliche Schnittstelle sowohl verwenden als auch anbieten müssen. Durch diese Standardisierung soll die komplette Architektur eines Systems stark vereinfacht werden können. Der Nachteil hierbei ist, dass dies eine Verschlechterung der Effizienz nach sich ziehen kann, da ein standardisierter Austausch suboptimal im Vergleich zu einer auf die jeweilige Applikation zugeschnittene Kommunikation sein kann. Dies trifft jedoch nicht auf das Einsatzgebiet und den Transfer von großen Hypermedia Daten zu, wofür REST laut Fielding entworfen wurde.

Mit der einheitlichen Schnittstelle gehen weitere Einschränkungen einher, namentlich die *Identifikation von Ressourcen*, die *Manipulation von Ressourcen durch Repräsentationen*, *selbst-beschreibende Nachrichten* sowie *Hypermedia as the Engine of Application State* (HATEOAS). Auf diese vier Einschränkungen wird nun im Detail eingegangen.

Eine *Ressource* ist in REST eine abstrakte Bezeichnung für jegliche Information, die benannt werden kann. Jeder Ressource muss nun eine einmalige Bezeichnung zugeordnet werden, um der *Identifikation von Ressourcen* zu entsprechen und mittels dieses Ressourcenbezeichners eindeutig auf sie referenzieren zu können. Um dies zu gewährleisten können beispielsweise *Uniform Resource Identifier* [9] verwendet werden.

Eine Ressource kann verschiedene Repräsentationen besitzen, um die selben Informationen bspw. für mehrere Anwendungen auf unterschiedliche Art zur Verfügung stellen zu können. Die Beschreibung des jeweiligen Datenformats der Repräsentationen geschieht hierbei durch MIME Media Types [7]. Eine Repräsentation zeigt den aktuellen Stand der zugehörigen Ressource auf dem Server. Um die Ressource zu manipulieren, kann ein Client die von ihm verwendete Repräsentation verändern und zurück an den Server schicken. Dies ergibt die Einschränkung der *Manipulation von Ressourcen durch Repräsentationen*.

Repräsentationen können Referenzen auf weiterführende Ressourcen beinhalten. Diese Referenzen und Verknüpfungen beschreiben *HATEOAS*

und ermöglichen einem Client auch ohne detailliertes Wissen, ihnen zu folgen.
Die *selbst-beschreibenden Nachrichten* sind eine direkte Folge der zuvor beschriebenen Statuslosigkeit, die erfordert, dass in jeder Anfrage alle nötigen Informationen mitgeschickt werden.

Als sechste und letzte Einschränkung ergänzt Fieldings noch das optionale *Code-on-demand*. Dies soll es einem Client ermöglichen, zur Laufzeit Code zu empfangen und ausführen zu können, um die eigene Funktionalität zu erweitern.

Diese Einschränkungen führen nach Fielding insgesamt zu einer vollständigen Definition von REST.

2.2 TOSCA

Die Topology and Orchestration Specification for Cloud Applications (*TOSCA*) [10] beschreibt den strukturellen Aufbau von Cloud Applikationen. Diese hat vier Hauptziele: die Automatisierung des Deployments und Managements von Anwendungen, die Portabilität und die Interoperabilität zwischen unterschiedlichen Cloud-Anbietern, sowie ein vom Hersteller unabhängiges Ökosystem.

TOSCA beschäftigt sich sowohl mit der *Topology* (und der damit verbundenen *Service Structure*) als auch mit der *Orchestration*, die das Deployment und Management einer Topology umfasst.
Im Folgenden wird näher auf die Topology und Orchestration von TOSCA eingegangen, sowie abschließend OpenTOSCA kurz erläutert.

2.2.1 Topology

Eine Topology besteht aus abstrakten *Node Templates* die einen konkreten *Node Type* besitzen, sowie aus *Relationship Templates* mit zugehörigem *Relationship Type*. Sowohl Node Types als auch Relationship Types können darüber hinaus noch *Properties*, die diese näher beschreiben, besitzen. Einem Node Type können außerdem *Operations*, die von ihm ausgeführt werden können, zugeordnet sein. Die eigentliche Funktionalität dieser Operationen wird von (potentiell mehreren) *Artifacts* zur Verfügung gestellt und implementiert.

2.2.2 Orchestration

TOSCA benennt zwei Arten der Orchestration, *declarative* und die *imperative* Orchestration.

Die **declarative Orchestration** stellt die Frage, was genau realisiert werden soll. Dabei interpretiert die Laufzeitumgebung die gegebene Topology und

führt das Deployment durch. Die dafür nötige Logik ist hierbei vollständig im TOSCA-Container implementiert. Ein Beispiel wäre die Anweisung, einen Tomcat Server aufzusetzen. Die Auflösung und Erfüllung der dafür nötigen Abhängigkeiten, die zu diesem Server führen, obliegt der Laufzeitumgebung.

Die **imperative Orchestration** beschreibt den Weg, wie etwas realisiert werden soll. Dabei werden explizit alle einzelnen Schritte genau beschrieben, die notwendig sind, um das gewünschte Ziel zu erreichen. Das vorherige Beispiel würde hier aus einem *Management Plan* bestehen, der die Anweisungen enthält, eine virtuelle Maschine aufzusetzen, dort einem Tomcat Server zu installieren und diesen daraufhin zu starten.

2.2.3 OpenTOSCA

OpenTOSCA [8] ist eine Open Source Implementierung eines TOSCA Containers, die in den letzten Jahren an der Universität Stuttgart entwickelt wurde. Sie bietet eine Laufzeitumgebung für TOSCA-Anwendungen.

Neben dem beschriebenen OpenTOSCA Container werden noch einige Tools zur Verfügung gestellt. Dazu gehört ein „Self-Service UI“ für den Endnutzer (*Vinothek* genannt) und ein Admin UI, beide benutzen die API des Containers um mit ihm zu interagieren. Ferner existiert die *Winery*, welche die Erstellung und Modellierung von TOSCA Anwendungen sowie deren Topologie und Management-Plänen erlaubt. Sie dient der Arbeit mit *Cloud Service Archives* (CSAR) und kann diese im- und exportieren.

3 Ist- und Soll-Zustand

Dieses Kapitel erläutert die Anforderungen an die Implementierung im Rahmen dieser Bachelorarbeit. Ferner beschreibt es den bisherigen Stand des existierenden Prototypen der OpenTOSCA Monitoring-GUI und analysiert die notwendigen Schritte um die zuvor definierten Anforderungen zu erfüllen.

3.1 Anforderungen

Das Ziel ist ein Monitoring System, welches über eine REST API auf ein bestehendes Backend zugreift, um verschiedene Arten von Eventtypen abzufragen und darzustellen. Hierfür sollen zwei grafische Nutzerschnittstellen realisiert werden. Eine dient der Übersicht über allgemeine Events, die andere soll OpenTOSCA Events mit Informationen über Provisionierung- und Verwaltungsprozesse grafisch darstellen.

Für diese beiden Schnittstellen existieren eine Reihe von Anforderungen, die im Folgenden näher beschrieben werden.

3.1.1 Anforderungen an das generische Frontend

Nachfolgend werden die Anforderungen an das generische Frontend aufgezählt und jeweils kurz beschrieben.

A1 - Nutzung der REST API Die Interaktion mit dem Backend soll durch dessen REST API geschehen. Insbesondere werden darüber die gewünschten Events abgefragt oder alle gespeicherten Events gelöscht.

A2 - Events löschen Es soll möglich sein, alle Events zu löschen. Dies betrifft sowohl die persistente Speicherung im Backend als auch die aktuelle Anzeige im Frontend.

A3 - Dynamische Updates Events sollen dynamisch nachgeladen werden und so neu hinzugekommene Events in der Benutzeroberfläche angezeigt werden.

A4 - Performantes Nachladen Die zuvor beschriebenen dynamischen Updates sollen möglichst effizient sein. Insbesondere sollte sich die Anfrage nach Events beim Backend explizit auf jene beschränken, die dem Frontend bisher noch nicht bekannt sind.

A5 - Kontrolle über Datenabfrage Die Abfrage von Events soll vom Nutzer kontrollierbar sein. Daher soll die Möglichkeit bestehen, die initiale Abfrage sowie das dynamische Nachladen zu starten oder anzuhalten. Das Intervall zwischen den Updates soll frei wählbar sein.

A6 - Tabellenlayout Die Darstellung der Events soll in tabellarischer Form geschehen.

A7 - Daten sortieren Die Tabelle soll beliebig nach den Feldern der angezeigten Events (*id*, *source*, *datetime* und *message*) sortiert werden können.

A8 - Daten filtern Die Tabelle soll nach einigen Feldern gefiltert werden können. Hierfür bieten sich insbesondere das Feld *source* und *message* an.

A9 - Auswahl der Eventquelle Es soll möglich sein, aus einer Liste aller bekannten Quellen auszuwählen und nur Events anzuzeigen, die der gewählten *Source* entsprechen.

A10 - Teilmenge der Daten Aufgrund der potentiell großen Anzahl an Events soll auch ohne Filter die aktuelle Ansicht auf eine maximale Anzahl an Events beschränkbar sein, um die Seite nicht zu überladen.

3.1.2 Anforderungen an das OpenTOSCA Frontend

Im Folgenden werden die Anforderungen an das OpenTOSCA Frontend aufgezählt und jeweils kurz beschrieben. Aufgrund einer funktionalen Überschneidung treffen einige Anforderungen an das generische Frontend auch auf das OpenTOSCA Frontend zu. Dies betrifft insbesondere die Anforderungen **A1** bis **A5**, die deshalb hier nicht wiederholt aufgeführt werden.

Das grundlegende Design soll sich am Aussehen des bisherigen Prototypen orientieren. Einige spezifischeren Details diesbezüglich finden sich in den nachfolgenden Anforderungen. Ferner sollen alle funktionalen Eigenschaften des Prototypen im neuen Frontend ebenfalls vorhanden sein.

A11 - Verwaltung von Node Instances Sobald ein neues Event verarbeitet wird, dessen zugehörige *Node Instance* dem Frontend noch nicht bekannt ist, sollen die OpenTOSCA Informationen zu dieser Instance beim Backend erfragt werden. Ferner muss sie in der Benutzeroberfläche hinzugefügt und angezeigt werden.

A12 - Gruppierung nach Node Instances Alle Events sollen durch die zugehörige Node Instance gruppiert werden. Jede Node Instance soll grafisch klar getrennt sein, bspw. durch einen eigenen „Kasten“ pro Instance.

A13 - Freie Anordnung der Node Instances Die Kästen der Node Instances sollen sich per Drag'n'Drop vom Nutzer frei verschieben und positionieren lassen.

A14 - OpenTOSCA Informationen Alle OpenTOSCA Informationen zu einer Node Instance, die das Backend zur Verfügung stellt, sollen dieser Instance zugeordnet sein und innerhalb ihres Kastens angezeigt werden.

A15 - Grafische Darstellung Sofern möglich sollten Informationen grafisch sinnvoll und benutzerfreundlich präsentiert werden. Insbesondere soll der *Node Type* durch ein passendes Icon und der *State* einer Node Instance farblich dargestellt werden.

A16 - Events in Tabellenform Die zu einer Node Instance gehörigen Events sollen in tabellarischer Form oder in Form einer Liste innerhalb des Kastens der Instance angezeigt werden.

A17 - Events einklappen Die Tabelle oder Liste der Events soll standardmäßig sichtbar sein, sich jedoch zur besseren Übersicht ein- und ausklappen lassen.

A18 - Properties anzeigen Die *Properties* einer Node Instance sind ein längerer XML-String und sollen leicht zugänglich einsehbar sein, bspw. durch ein Pop-up.

A19 - Formatierung der Properties Das XML der Properties soll mittels Syntax-Highlighting und korrekter Einrückung übersichtlich und lesbar dargestellt werden.

A20 - Statusänderungen erkennen Neben den dynamischen Updates der Events (siehe Anforderung **A3**) soll der *State* aller bekannten Node Instances periodisch überprüft werden. Das Intervall soll hierbei vom Benutzer festlegbar sein, kann jedoch an das Intervall des Updates neuer Events gebunden sein. Bei einer Änderung muss die Anzeige (textuell und grafisch) in der Benutzeroberfläche entsprechend angepasst werden.

A21 - Filterfunktionen Es soll möglich sein, nach allen dem Frontend bekannten *Service Instances* sowie *Node Types* zu filtern und nur diejenigen Node Instances anzuzeigen, die beiden Filtern entsprechen.

3.2 Bisheriger Prototyp

Der bisherige Prototyp der OpenTOSCA-GUI erfüllt bereits einen Teil dieser Anforderungen. Auf die nicht oder nur unvollständig erfüllten Anforderungen wird später in der Analyse eingegangen.

Die folgenden Anforderungen sind vollständig erfüllt:

- **A11** - Verwaltung von Node Instances
- **A12** - Gruppierung nach Node Instances
- **A14** - OpenTOSCA Informationen
- **A15** - Grafische Darstellung
- **A16** - Events in Tabellenform
- **A17** - Events einklappen
- **A18** - Properties anzeigen
- **A20** - Statusänderungen erkennen

3.3 Analyse

Im Folgenden wird analysiert, welche der zuvor beschriebenen Anforderungen noch nicht erfüllt wurden, und die im Rahmen dieser Arbeit umgesetzt werden sollen.

3.3.1 Generisches Frontend

Da bisher noch kein Prototyp einer Benutzerschnittstelle für generische Events existiert, ist die erstmalige Umsetzung aller zuvor genannten Anforderungen an dieses Frontend (**A1** bis **A10**) erforderlich.

Hierfür wird entsprechend unabhängig vom bisherigen Prototypen oder der Neuimplementierung des OpenTOSCA Frontends ein eigenständiges Frontend zur Erfassung und Darstellung allgemeiner Events implementiert.

3.3.2 OpenTOSCA Frontend

Aufgrund der Existenz des bisherigen Prototypen ist es möglich, sich an der Umsetzung der bereits von ihm erfüllten Anforderungen zu orientieren. Da jedoch kein *Refactoring* des bisherigen Codes durchgeführt und der Prototyp auch nicht erweitert wird, muss die Neuimplementierung des Frontends alle bisher erfüllten Anforderungen vollständig umsetzen. Zusätzlich sind einige Funktionen im Prototypen nicht oder nur teilweise vorhanden.

Die folgenden Anforderungen implementierte der Prototyp nur teilweise:

- **A13** - Freie Anordnung der Node Instances: Drag'n'Drop der Node Instances ist möglich, jedoch nicht klar repräsentiert (das Logo des Node Types einer Instance muss geklickt werden). Ferner lassen sich Instances nicht problemlos überlappen, da die Reihenfolge (Sichtbarkeit) nicht veränderlich ist.
- **A19** - Formatierung der Properties: Der XML-String ist teilweise eingerückt. Es existiert kein Syntax-Highlighting.

Die folgende Anforderung wurde nicht umgesetzt:

- **A21** - Filterfunktionen

Zuallererst sollte das neue Frontend daher die Funktionalität des bisherigen Prototypen nachbilden um die jeweiligen Anforderungen ebenfalls erfüllen zu können. Darüber hinaus sind Verbesserungen in den beiden genannten Bereichen nötig, um die Anforderung vollständig und korrekt umzusetzen.

Abschließend soll die Erweiterung der Implementierung um die Filterfunktionen aus **A21** stattfinden.

4 Entwurf

Dieses Kapitel behandelt den Entwurf der Monitoring Frontends. Es geht dabei auf die Entwurfsentscheidungen und Überlegungen zur Umsetzung der beiden Benutzeroberflächen ein.

Da bereits eine prototypische Implementierung der OpenTOSCA Monitoring GUI existierte, die zwar nicht direkt erweitert, sich bei der Neuimplementierung jedoch stark an ihr orientiert werden sollte, konnten viele Entwurfsentscheidungen bereits davon abgeleitet werden.

Weder das Design noch die Funktionalität dieser GUI sind allerdings gut für das generische Frontend geeignet. Deshalb ist davon auszugehen, dass beide Frontends trotz einer ähnlichen Interaktion mit dem Backend, und der Abfrage von Events mittels einer REST-API, nur im geringen Maße Überschneidungen in ihrer Implementierung haben werden.

Daher werden das generische Frontend und das OpenTOSCA Frontend von vorne herein als separate Projekte behandelt, die sich nur wenige, gemeinsam verwendete Bibliotheken teilen. Im Folgenden wird näher auf beide Frontends eingegangen.

4.1 Generisches Frontend

Das generische Frontend soll „basic“ Events darstellen. Da dies der grundlegende Eventtyp ist, von dem alle spezialisierten oder erweiterten Typen abgeleitet werden, entspricht dies der Anzeige aller im Backend gespeicherten Events.

Da einerseits keine Kenntnis über die Art des jeweiligen Events zur Verfügung steht, als auch Events aller Typen gleichzeitig dargestellt werden sollen, muss die generische Oberfläche entsprechend sehr einfach gehalten werden. Eine umfangreiche Aufbereitung, bspw. mit einer sinnvollen Gruppierungen von Events, der Anzeige von Zusatzinformationen oder weitere domänenspezifische Anpassungen sind nicht möglich.

Somit verbleibt die Darstellung aller verfügbaren Events als eine Aufzählung, bei der jeder Eintrag die „Rohdaten“ des jeweiligen Events enthält, die den Informationen aller Felder dieses Events entspricht. Hierfür bietet sich eine tabellarische Darstellung an, in der jede Zeile einem Event entspricht, und die Spalten der Tabelle die Felder des Basic-Typs darstellen. Dies liefert eine schnelle Übersicht aller Events und ihrer Grundinformationen, wie sie vom Backend geliefert werden.

Da zusätzliche Felder, die spezialisierte Typen einführen können, ohne das benötigte Wissen über deren Bedeutung nicht interpretiert werden können, würde die Anzeige dieser Felder in einem generischen Frontend wenig Sinn

ergeben. Es würde ferner das Design unnötig komplex machen, da eine variable Anzahl an anzuzeigenden Feldern pro Eventtyp umgesetzt werden müsste.

Um die gesamten Events des Monitoring Systems im Überblick behalten zu können, bietet sich ein automatischer Abruf neuer Events an, um den Nutzer über alle Aktivitäten informieren und diese von ihm überwachen lassen zu können. Ferner sollte es dem Nutzer ermöglicht werden, die Anzahl an Events auf eine für ihn interessante Teilmenge einzuschränken, indem verschiedene Filtermethoden eingesetzt werden können, sowie einige oder alle Felder durchsucht und sortiert werden können.

4.2 OpenTOSCA Frontend

Das OpenTOSCA Frontend orientiert sich sehr stark an dem bisherigen Prototypen. Dieser hat sich in der Praxis bewährt und weist funktional nur wenige Mängel auf, weshalb hierbei keine besonderen oder umfassenden Entwurfsentscheidungen getroffen werden müssen, die vom Entwurf des Prototypen abweichen.

Da der Prototyp jedoch historisch gewachsen ist, von unterschiedlichen Personen ohne einheitlichen Stil entwickelt wurde, und viel *Bloat* in Form von Drittanbieter-Bibliotheken oder Assets mit sich bringt, sollte er nicht direkt erweitert werden. Ein *Refactoring* und die darauf folgende Weiterentwicklung des Prototypen wäre eine mögliche Option.

Aufgrund des umfangreichen und teilweise komplexen Codes im Verhältnis zu dem überschaubaren Umfang, den die realisierte Funktionalität bietet, liegt eine komplette Neuimplementierung jedoch näher. Diese soll eine neue Codebasis schaffen, sparsam mit der Verwendung von Drittanbieter-Bibliotheken umgehen und dabei die bisher bestehenden funktionalen Mängel ausbessert.

Das Design der Neuimplementierung soll eine Nachbildung des Prototypen sein. Darauf hin soll die Wiederherstellung des bisherigen Funktionsumfangs folgen. Zu diesem Zeitpunkt kann der Prototyp nun bereits durch die wartbare, „entschlackte“ und code-technisch übersichtlichere Implementierung ersetzt werden. Schlussendlich sollte die Erweiterung um die in Kapitel 3 verlangten Anforderungen erfolgen.

Bei der gesamten Entwicklung ist darauf zu achten, die zukünftige Erweiterbarkeit der Implementierung bestmöglich zu gewährleisten. Ferner soll eine „sparsame“ und begründete Verwendung von Bibliotheken und Tools ebenfalls ein Hauptaugenmerk sein.

5 Implementierung

Dieses Kapitel beschreibt die konkrete Implementierung der Monitoring-Frontends. Es unterteilt sich in das einfach gehaltene, generische Frontend sowie das OpenTOSCA-Frontend.

Beide Varianten sind als Web-Interface umgesetzt, die eingesetzten Technologien sind hierbei die in der Webentwicklung üblichen Auszeichnungssprachen HTML [6] und CSS [2], die Programmiersprache JavaScript [4] sowie einige Frameworks für diese Sprachen.

Den einzelnen Benutzeroberflächen ist eine einfache HTML-Übersichtsseite vorgeschaltet, die weiterführende Links beinhaltet. Dort kann der Nutzer auswählen, welche GUI verwendet werden soll.

5.1 Frameworks

Beide Frontends benutzen sowohl jQuery als auch Bootstrap, sowie weitere JavaScript-Bibliotheken. Im Folgenden wird ein kurzer Überblick über jQuery und Bootstrap gegeben, soweit dies für diese Arbeit relevant ist.

5.1.1 jQuery

jQuery [11] ist eine weit verbreitete, umfangreiche JavaScript Bibliothek, die insbesondere die Arbeit mit dynamisch generierten Inhalten, Events und die Kommunikation mit einem Server erleichtert. Sie ist damit sehr gut für den Anwendungszweck beider Frontends geeignet und wird extensiv genutzt. Der JavaScript-Code für das OpenTOSCA-Frontend wird beispielsweise direkt als „Plugin“ in jQuery verwendet, und auch die später näher erläuterten Bibliotheken „DataTables„ und „jQuery UI“ erweitern oder benötigen jQuery als Framework.

5.1.2 Bootstrap

Bootstrap [1] ist ein CSS-Framework und Template-System, das ein einfaches und elegantes Design vieler HTML-Elemente und oft verwendeter Muster bereitstellt. Es wird in kleinen Teilen für die Tabelle des generischen Frontends und für das Control Panel des OpenTOSCA-Frontends verwendet. Es ermöglicht ein einheitliches Design und erspart den Aufwand, diese Elemente händisch gestalten und anpassen zu müssen.

5.2 Generisches Frontend

Dieses Unterkapitel beschreibt die Umsetzung des generischen Frontends, das sämtliche Events anzeigt.

5.2.1 Darstellung

Das generische Frontend verwendet eine tabellarische Darstellung ohne weitere Aufbereitung der Daten. Dies liefert eine schnelle Übersicht der Rohdaten, wie sie vom Backend geliefert werden, und erfüllt außerdem die in Kapitel 3 festgelegte Anforderung `A6`.

Basic Event GUI

Start or stop refreshing all 5 seconds. Delete all events

Show 25 entries Search:

ID	Source ID	Date	Message
1	basicEventSource	Thu Jan 01 1970 01:00:00 GMT+0100 (CET)	this is an example for an event message
ID	Search Source ID basicEventSource	Date	Search Message

Showing 1 to 1 of 1 entries (filtered from 74 total entries) Previous 1 Next

Abbildung 5.1: Generisches Frontend mit einem „basic“ Event

In Abbildung 5.1 ist diese Übersicht mit nur einem „basic“ Event als Beispiel zu sehen. Die Anforderungen A1 bis A3 sowie A5 wurden durch die obere Leiste erfüllt, in der die wichtigsten Funktionen zu finden sind. Mittels „Start“ wird begonnen, die Events via REST vom Backend in einem vom Benutzer definierten Intervall zu erfragen und neue Events in die Tabelle einzufügen. Ein Klick auf „Stop“ beendet die automatische Abfrage und belässt die Tabelle im aktuellen Zustand, ohne weiter auf neue Events zu reagieren.

Insbesondere zu Testzwecken lässt sich auch das Löschen aller im Backend persistierten Events anstoßen, indem die Funktion „Delete all events“ ausgewählt wird (Anforderung A2). Hierbei ist darauf zu achten, dass die Aktion nicht rückgängig gemacht werden kann. Bei Bestätigung des Löschvorgangs durch das Backend reagiert die Benutzeroberfläche ebenfalls durch das Entfernen aller Events in der Tabelle, damit keine veralteten bzw. gelöschten Events mehr angezeigt werden.

Die Tabelle repräsentiert alle Felder, die für jeden Eventtyp zwingen vorgeschrieben sind. Mögliche Zusatzfelder eines speziellen Typs werden jedoch nicht berücksichtigt und entsprechend auch nicht dargestellt. Die notwendigen Felder sind in Tabelle 1 beschrieben.

Die Tabelle ermöglicht das Suchen und Filtern für einige dieser Felder. Sortiert werden kann auf- und absteigend nach allen Feldern, womit die Anforderung A7 erfüllt wird. Oberhalb der Tabelle befindet sich eine Suchfunktion, die ebenfalls alle Felder durchsucht. Die Anforderung A8 wurde umgesetzt, in dem zusätzlich unterhalb der Tabelle noch speziell nur

Feld	Beschreibung
id	Vom Backend automatisch vergebene ID
source	Die „Quelle“ des Events (auch *Source ID*)
datetime	Zeitstempel (bspw. an dem das Event erzeugt wurde)
message	Textnachricht des Events (möglicherweise leer)

Tabelle 1: Pflichtfelder für jeden Eventtyp

„Source ID“ oder „Message“ durchsucht werden kann. Da davon auszugehen ist, dass es nur eine geringe Zahl an Produzenten von Events gibt, besitzt die „Source ID“ Spalte ferner noch eine Drop-down Liste, die alle Source IDs aufzählt und per Klick nur Events der gewählten Quelle anzeigt (Anforderung A9).

Anforderung A10 wurde umgesetzt, indem standardmäßig nur eine gewisse Anzahl an Events gleichzeitig angezeigt werden (hier 25). Weitere Ergebnisse werden durch eine Pagination in „Seiten“ der Tabelle aufgeteilt, die rechts unten dargestellt und zwischen ihnen gewechselt werden kann. Links oben kann ferner die Anzahl der Events pro Seite festgelegt werden, es ist ebenfalls möglich, alle Events anzuzeigen (und die Pagination somit effektiv außer Kraft zu setzen).

Die Anzahl der Ergebnisse nach dem Anwenden eines Filters, sowie die aktuelle Teilansicht (aufgrund der Aufteilung auf Seiten) wird abschließend links unten in einem Statustext zusammengefasst.

In Abbildung 5.2 ist abschließend noch eine Übersicht mit gemischten Eventtypen und Quellen (Source IDs) zu sehen, die Source IDs und die Anzeige des Datums wurden für dieses Beispiel jedoch gekürzt. Neben drei „basic“ Events aus unterschiedlichen Quellen werden hier ebenfalls OpenTOSCA Events angezeigt, die jedoch im Gegensatz zum speziellen OpenTOSCA-Frontend nicht aufbereitet wurden.

5.2.2 Technische Umsetzung

Das generische Frontend besteht aus einer einzelnen HTML-Datei, die ihrerseits nur das „Control Panel„ (Start/stop/delete) und den Header und Footer der Tabelle definiert. Sie bindet außerdem einige weitere Styles und Scripts ein. Siehe Abbildung 5.3 für eine Übersicht der Architektur.
DataTables [3] ist ein jQuery Plugin, das die gesamte Verwaltung von Tabellenstrukturen und interaktiven Tabellen ermöglicht oder stark erleichtert. Aufgrund der Wahl, die Events des generischen Frontends in Tabellenform zu repräsentieren, bot sich das DataTables-Plugin für diesen Zweck an.

Show All entries Search:

ID	Source ID	Date	Message
1	basicEventSource	Thu Jan 01 1970	this is an example for an event message
2	hauptfn	Thu Jan 01 1970	test message
3	hauptfn	Thu Jan 01 1970	test message 2
953	nodeInstances/20	Wed May 04 2016	start install
954	nodeInstances/18	Wed May 04 2016	start creating VM
955	nodeInstances/18	Wed May 04 2016	create folder for l4t_c5f8828d-45cb-40bf-8888-2ac2
956	nodeInstances/18	Wed May 04 2016	copy VMDK /VMRepo/Ubuntu Server 12.04 (64Bit).vmdk
957	nodeInstances/18	Wed May 04 2016	configure VM
958	nodeInstances/18	Wed May 04 2016	power on
959	nodeInstances/18	Wed May 04 2016	determine IP address
960	nodeInstances/20	Wed May 04 2016	install finished

Abbildung 5.2: Generisches Frontend mit gemischten Eventtypen

Datenstruktur

Die vom Backend gelieferte JSON-Datenstruktur bei der Abfrage von Events ist in Listing 1 anhand von zwei beispielhaften Events zu sehen. Die Struktur ist ein Array von Objekten, in der jedes Objekt ein Event repräsentiert. Ein Objekt besteht wie auch in Tabelle 1 näher beschrieben aus vier Feldern: `id`, `message`, `source` und `datetime`. Sofern keine zur Anfrage passenden Events existieren, liefert das Backend ein leeres Array zurück.

Initialisierung

DataTables ermöglicht eine einfache Initialisierung auf jeder bestehenden Tabelle und stellt direkt mehrere Sortier-, Filter- und Suchmethoden sowie die dafür nötigen Eingabemasken zur Verfügung. Dabei muss darauf geachtet werden, die verwendete Datenstruktur und das Mapping auf die Spalten der Tabelle korrekt anzugeben.

Eine vereinfachte Übersicht der Initialisierung für das generische Frontend ist in Listing 2 zu sehen. DataTables erwartet standardmäßig einen JSON-Datensatz mit dem Feld `data`, welches die eigentliche Datenstruktur enthält. Dies ist jedoch hier nicht der Fall, wie in Listing 1 zu sehen ist. Um dem Plugin zu vermitteln, dass es sich bei den verwendeten Daten um ein *flat array* handelt, musste der `dataSrc` Option ein Leerstring übergeben werden (Zeile 4).

Die restlichen (für diese Übersicht gekürzten) Optionen sind unter anderem

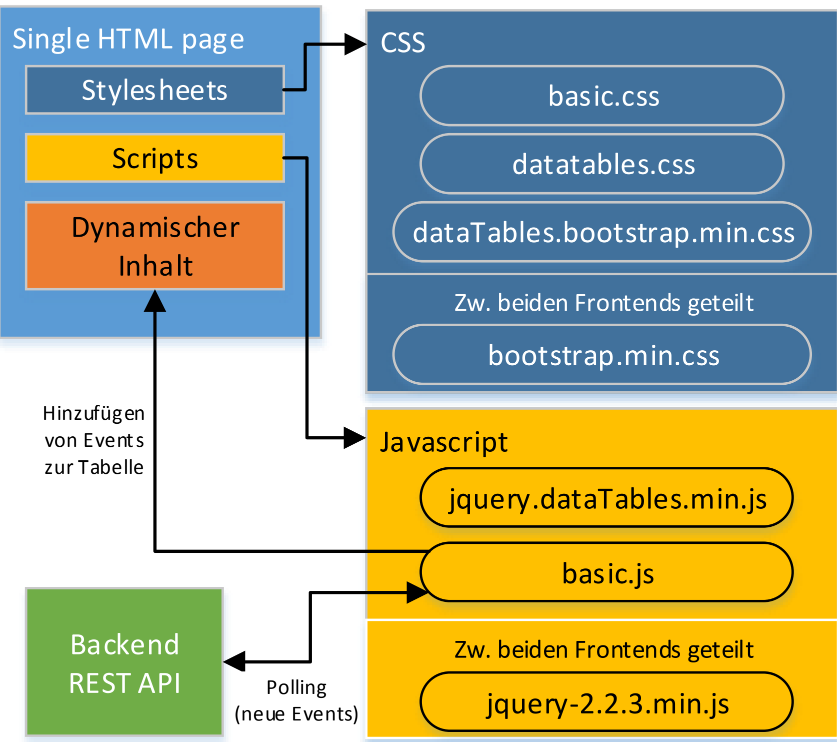

Abbildung 5.3: Architektur des generischen Frontends

eine URL, von der die initialen Daten für die Tabelle abgerufen werden (Zeile 3), ein Mapping zwischen den Feldern aus dem erhaltenen Datensatz und den Spalten der Tabelle (Zeile 7-14), als auch eine abschließende Funktion die aufgerufen wird, sobald der Aufbau der Tabelle durch das Plugin abgeschlossen wurde (Zeile 16).

Nach erfolgreicher Initialisierung der Tabelle wird sich die höchste bekannte ID eines Events gespeichert, um in weiteren Aufrufen nur neuere Events beim Backend zu erfragen.
Ferner werden die Funktionen `addSearchSelect()` und `addSearchInput()` aufgerufen, die Auswahl- und Suchfelder unterhalb der Tabelle für ausgewählte Spalten erzeugen. Die entsprechenden Spalten können definiert werden, indem in der Grundstruktur der `index.html` die CSS-Klasse `selectable` für ein Auswahlfeld und die Klasse `searchable` für ein Freitextfeld gesetzt werden. Im generischen Frontend wurde für die Spalte *Message* eine Textsuche verwendet, für die *Source ID* hingegen sowohl eine Textsuche als auch ein Auswahlfeld, da die Anzahl an verschiedenen Eventquellen in der Praxis beschränkt ist.

Abschließend wird ein Polling im vom Nutzer festgelegten Intervall gestartet.

```
[
  {
    "id": 1,
    "message": "Some event happened!",
    "source": "Generic Event Source",
    "datetime": 1462372421337
  },
  {
    "id": 2,
    "message": "setFile ./sixcms_tmp/tmpscript.sh",
    "source": "nodeInstances/20",
    "datetime": 1462372450000
  }
]
```


Listing 1 JSON Datenstruktur für generische Events

Workflow

Die Anforderung A4 wurde umgesetzt, indem das Polling mittels eines GET Requests alle Events beim Backend erfragt, die eine höhere ID als die zuletzt bekannte ID haben. Diese Anfrage geht an die Ressource `/monitoring/eventstorages/basic/eventlist?afterEventID={id}`.

Bei Erfolg wird vom Server eine Response wie in Listing 1 erhalten. Sofern seit der letzten Anfrage keine neue Events erzeugt wurden, ist das erhaltene JSON-Array leer und an dieser Stelle wird abgebrochen. Anderenfalls wird die neue höchste Event ID gespeichert und im Anschluss über das Array iteriert.

DataTables bietet eine umfangreiche API, die unter anderem das Hinzufügen neuer Daten durch den Aufruf `datatable.row.add(rowData)` ermöglicht, mittels `datatable.draw()` kann die Anzeige der Tabelle aktualisiert werden, um die neu eingepflegten Daten sichtbar zu machen.

Abschließend werden alle Auswahlfelder durch `addSearchSelect()` aktualisiert, sofern bspw. neue *Event Sources* hinzugekommen sind.

5.3 OpenTOSCA Frontend

Dieses Unterkapitel beschreibt die Umsetzung des OpenTOSCA Frontends, das domänenspezifisches Wissen über OpenTOSCA-Events verwendet um diese sinnvoll zu gliedern und darzustellen, sowie zusätzliche Informationen zu ihnen zu beziehen.

5.3.1 Darstellung

Das Layout des OpenTOSCA Frontends orientiert sich sehr stark an der bisherigen Implementierung der Monitoring GUIs.

```
DataTable({
  "ajax": {
    url: "/monitoring/eventstorages/basic/eventlist",
    dataSrc: "" // Die JSON Daten sind ein flat array
  },

  "columns": [
      // Die Felder eines Events werden in der Reihenfolge
      // der Spalten der Tabellen zugeordnet
      { data: "id" }, // Erste Spalte
      { data: "source" },
      { data: "datetime" },
      { data: "message" } // Letzte Spalte
  ],

  "initComplete": function () { onTableInitComplete(); }
});
```

Listing 2 Initialisierung des DataTables Plugins

Allgemein

Jedes Event hat genau eine `Source`, die in diesem Fall einer *Node Instance* entspricht. Im Gegensatz zu generischen Events, deren Sources sehr verschieden und insbesondere unbekannt sind, stehen zu jeder Node Instance allerdings weiterführende Informationen zur Verfügung, die abgefragt und entsprechend dargestellt werden können. Daher bietet sich eine Gruppierung der Events nach ihrer jeweiligen Node Instance an, wobei dort die gesammelten Informationen zusammengeführt werden können. Abbildung 5.4 zeigt das OpenTOSCA Frontend mit zwei beispielhaften Node Instances, die Events der ersten Instance sind ausgeblendet.

Control Panel und Filter

Ähnlich dem generischen Frontend gibt es ebenfalls ein Control Panel am Beginn der Seite, von dem aus die Abfrage von neuen Events begonnen oder angehalten werden sowie das Intervall festgelegt werden kann (Anforderung A5). Mittels des Buttons „Delete all events“ lassen sich alle OpenTOSCA Events löschen (Anforderung A2), insbesondere betrifft dies jedoch keine Events eines anderen Eventtyps.

Weiterhin befinden sich unterhalb dieser Buttons zwei OpenTOSCA spezifische Filter, die bei ihrer Aktivierung nur Node Instances einer ausgewählten *Service Instance* oder eines *Node Types* anzeigen (Anforderung A21). Die Service Instance ist hierbei eine TOSCA-spezifische ID, der mehrere Node Instances untergeordnet sind. Der entsprechende Filter listet

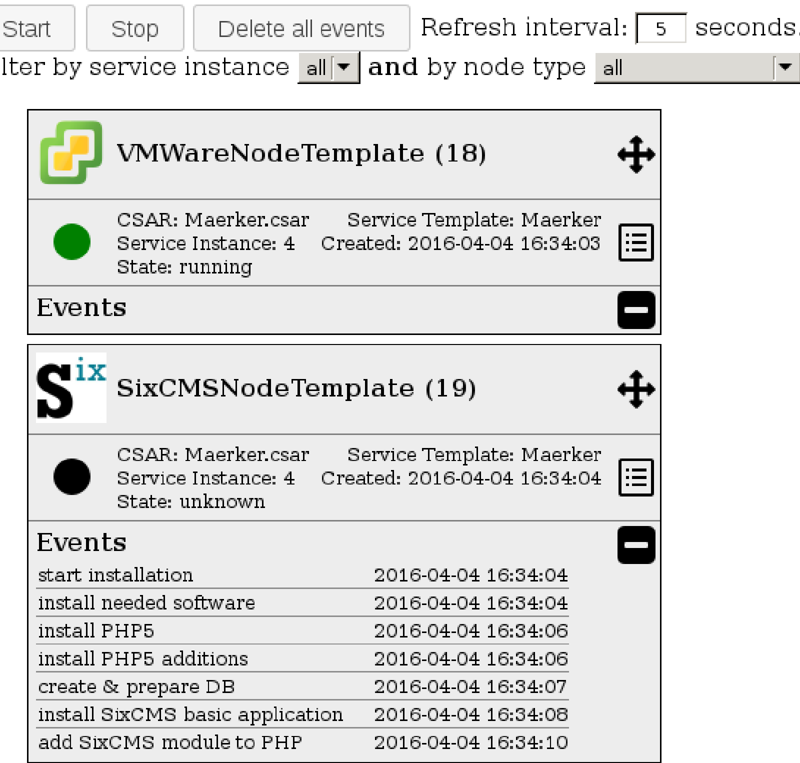

Abbildung 5.4: OpenTOSCA Frontend mit zwei *Node Instances*

alle Instance IDs, die bisher bei der Abfrage von Node Instance Daten aufgetreten sind.

Selbiges gilt auch für den Filter für Node Types, dieser kann ebenfalls nach beliebigen Typen filtern. Die korrekte grafische Darstellung durch ein entsprechendes Icon ist jedoch derzeit nur für folgende Typen vorgesehen. Diese wurden aus dem bisherigen GUI-Prototypen abgeleitet:

- UbuntuNodeType
- EC2NodeType
- VMWareNodeType
- MySQLNodeType
- ApacheWSNodeType
- SixCMSNodeType

Davon abweichende Typen erhalten keine grafische Kennzeichnung innerhalb einer Node Instance. Die Erweiterung dieser Liste ist jedoch sowohl

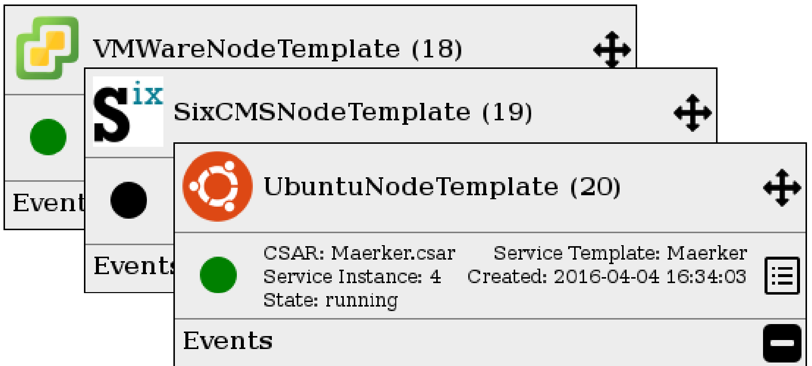

Abbildung 5.5: Überlappend gestapelte Node Instances

im Code als auch im Design vorgesehen und einfach umzusetzen. In der Funktion `insertNodeInstance()` existiert ein `switch`-Statement, das für jeden bekannten Typen eine CSS-Klasse vergibt. Das Hinzufügen einer weiteren `case`-Unterscheidung und einem zugehörigen Icon via CSS ist ausreichend, um einen neuen Typen zu unterstützen.

Standardmäßig sind keine Filter gesetzt, beide Auswahlfelder stehen auf „all“. Sobald einer der Filter geändert wird, werden alle Node Instances, auf die das gewählte Kriterium nicht zutrifft, ausgeblendet. Die verbleibenden oder jetzt ausgewählten Instances werden angezeigt und neu nebeneinander sortiert, um zu verhindern, dass sich zuvor vom Nutzer verschobene Instances außerhalb des sichtbaren Bereichs befinden. Werden beide Filter gemeinsam eingesetzt, so müssen alle Kriterien gleichzeitig erfüllt werden.

Node Instances

Die Anforderung A12 wurde umgesetzt, indem jede Node Instance durch einen eigenen Kasten in der Benutzeroberfläche repräsentiert wird. Er enthält nahezu alle Informationen, die OpenTOSCA zu einer Node Instance zur Verfügung stellt, und gruppiert diese in sinnvolle Teile (Anforderung A14).

Der erste Teil innerhalb des Kastens enthält ein dem `nodeType` entsprechendes Logo, sofern der Typ bekannt ist (siehe obige Liste) und entspricht somit der Anforderung A15. Es folgen die `nodeTemplateID` und der Namen der Node Instance. In der rechten oberen Ecke befindet sich außerdem ein *Move*-Icon, an dem der Kasten mittels Drag'n'Drop beliebig neu positioniert werden kann, siehe Abbildung 5.5. Dies erfüllt somit die Anforderung A13. Die Position bleibt bis zum Neuladen der Seite oder dem Anwenden bzw. Deaktivieren eines Filters bestehen und ermöglicht die komfortable Anordnung oder Gruppierung von für den Nutzer wichtigen Node Instances.

Der zweite Teil beginnt mit einem Icon, das den aktuellen `state` der

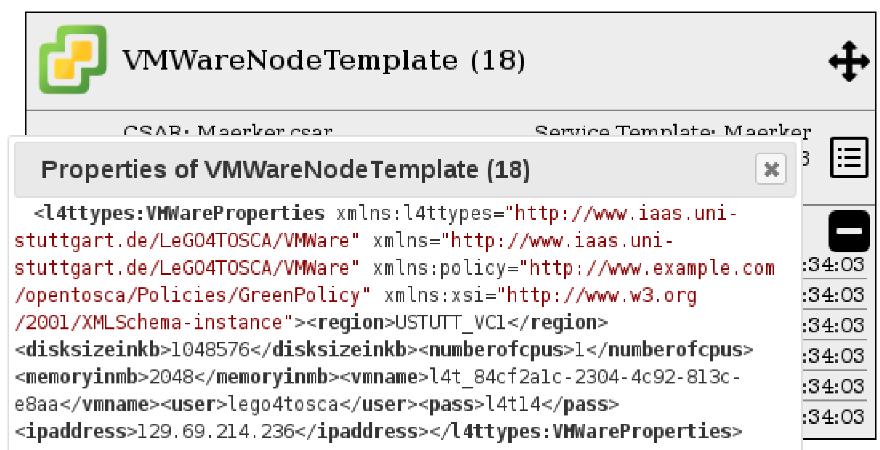

Abbildung 5.6: Node Instance mit darüber gelegtem *Properties* Pop-up

Instance repräsentiert (Anforderung A15). Bekannte States sind *running* (grün) und *installed* (blau), anderenfalls wird das Icon schwarz dargestellt. Darauf folgt eine Liste weiterer Eigenschaften: Die `csarID`, die `serviceInstanceId` (nach welcher gefiltert werden kann), eine textuelle Repräsentation des `state`, die `serviceTemplateID`, sowie der Zeitstempel des frühesten Events dieser Instance im `created` Feld.

Die Anforderungen A18 und A19 wurden umgesetzt, indem ein Klick auf das nebenstehende Icon ein Pop-up öffnet, in dem die `Properties` der Instance dargestellt werden, siehe Abbildung 5.6. Diese werden zusätzlich mit XML Syntax-Highlighting hervorgehoben. Das Pop-up lässt sich ebenfalls verschieben und mit einem Klick auf das Icon rechts oben schließen.

Der dritte Teil des Kastens besteht aus einer Tabelle mit allen bisher erhaltenen Events, die zu der aktuellen Node Instance gehören (Anforderung A16). Die Tabelle wird standardmäßig angezeigt, kann jedoch über einen Klick auf das *Minus*-Symbol aus- und eingeblendet werden um die Anforderung A17 zu erfüllen, siehe bspw. Abbildung 5.4.

5.3.2 Technische Umsetzung

Das OpenTOSCA Frontend besteht ebenfalls aus einer einzelnen HTML-Datei, die das Control Panel und die Auswahlfelder der Filter definiert, sowie die folgenden Dateien einbindet:

opentosca.css Das gesamte, OpenTOSCA spezifische Layout dieses Frontends

jquery-ui.min.css Design-Anpassungen für Elemente der jQuery UI Erweiterung

highlight.css Der „Original“-Style für *highlight.js*

jquery-2.2.3.min.js Die aktuelle Version von jQuery

jquery-ui.min.js Die aktuelle Version von jQuery UI

highlight.js Eine JavaScript Bibliothek für Syntax-Highlighting

opentosca.js Die eigentliche Implementierung und der gesamte Workflow des Frontends

Abgesehen von der eben aufgelisteten, genauen Auswahl und Benennung der referenzierten Dateien unterscheidet sich die Architektur des OpenTOSCA Frontends nicht signifikant von der des generischen Frontends, wie sie in Abbildung 5.3 zu sehen ist.

Aufbau

Das Frontend ist als direkte Erweiterung von jQuery selbst implementiert und in der Funktion `OpenTosca` innerhalb von jQuery definiert. Es wird in der `index.html` mittels `new $.OpenTosca();` initialisiert.

Der bisherige Prototyp des Monitoring GUIs hat einen *Model View Controller* (MVC) Ansatz gewählt. Die Implementierung dieser Bachelorarbeit verwendet jedoch keine derartige Struktur, sondern beschränkt sich auf eine einzelne „Klasse“ bzw. Memberfunktion des jQuery Objekts, sowie innerhalb des Objekts globale Variablen und weitere Unterfunktionen. Der gesamte Umfang des Codes beschränkt sich somit trotz eines größeren Funktionsumfangs auf nur 600 Zeilen, wohin gegen die bisherige Implementierung knapp 1500 Codezeilen benötigte. Hinzu kommen noch weitere JavaScript-Frameworks sowie Assets, die nicht länger benötigt werden und den Umfang in weiterem Maße reduzieren.

Initialisierung

Bei der Instanziierung des Frontends werden nur wenige Variablen und Objekte initialisiert. Unter anderem wird, ähnlich dem generischen Frontend, die höchste bekannte Event ID in `lastEventId` gespeichert, das Objekt `nodeInstances` hält hingegen alle bekannten Node Instances sowie die von OpenTOSCA dazu gelieferten Daten.

Die Filterfunktionen benötigen außerdem eine Liste aller bekannten *Service Instances* sowie der verschiedenen *Node Types*, die jeweils in den Objekten `serviceInstanceIds` und `nodeTypes` gespeichert und beim Erstellen neuer Node Instances aktualisiert werden.

Abschließend werden einige Event-Handler gebunden, um die Funktionen des Control Panels zu ermöglichen. Hierzu gehören der „Start“-Button, der die Abfrage der Events im angegebenen Intervall anstößt, sowie der „Stop“-Button, der dieses Polling unterbricht. Ferner werden die Handler für

das Löschen aller OpenTOSCA Events, das Ein- und Ausklappen von Eventlisten pro Node Instance und die Änderung der Filterauswahl gebunden.

Workflow

Beim Klick auf den „Start“-Button wird die Funktion `refreshStart()` aufgerufen. Diese liest und überprüft zunächst das aktuell vom Benutzer gesetzte Polling-Intervall. Ist es ungültig wird die Eingabe farblich markiert und an dieser Stelle abgebrochen. Anderenfalls wird ein möglicherweise gesetzter alter Timer deaktiviert, und das übergebene Intervall verwendet, um in diesem Abstand die Funktion `refresh()` aufzurufen (Anforderung A3). Damit der Ablauf des Timers nicht abgewartet werden muss, wird `refresh()` außerdem direkt aufgerufen.

Eine spätere Verwendung des „Stop“-Buttons deaktiviert den zuvor gesetzten Timer und behält den aktuellen Zustand der Benutzeroberfläche bei.

Die Funktion `refresh()` iteriert zuerst über alle bereits bekannten Node Instances, und erfragt mittels eines GET Requests im Backend an der Ressource `/monitoring/eventstorages/opentosca/nodeinstancelist/{id}/state` den aktuellen Zustand jeder Instance. Die Response ist ein einfacher String (insbesondere kein JSON). Dieser wird mit dem zuletzt bekannten Zustand abgeglichen und bei einer Änderung wird die grafische und textuelle Repräsentation in der Benutzeroberfläche entsprechend angepasst (Anforderung A20).

Hiernach wird ein weiterer GET Request an die Eventlist unter der Ressource `/monitoring/eventstorages/opentosca/eventlist&afterEventID={id}` gestellt, die Response enthält alle seit der letzten Anfrage neu hinzugekommenen Events und wird weiter an `insertNewEvents()` übergeben. Da explizit nur neuere Events beim Backend erfragt werden, ist somit die Anforderung A4 erfüllt.

Verarbeitung neuer Events und Node Instances

Die Funktion `insertNewEvents()` erhält neu empfangene Events als JSON Objekt. Die Datenstruktur ist eine Obermenge der generischen Events, bei OpenTOSCA wurde ausschließlich das Feld `nodeInstanceId` hinzugefügt, es entspricht inhaltlich jedoch der `source`. Ein Beispieldatensatz des JSON Objekts ist in Listing 3 zu sehen.

Nun wird über jedes dieser ankommenden Events iteriert und die Node Instance, zu der es gehört, aus dem Cache abgerufen. Sofern diese bereits vorhanden ist, kann das Event direkt zur Event-Tabelle hinzugefügt werden und erscheint in der Benutzeroberfläche.

```
[
  {
    "id": 1,
    "message": "start install",
    "datetime": 1462372442552,
    "source": "http://localhost:1337/containerapi/instancedata/nodeInstances/20",
    "nodeInstanceId": "http://localhost:1337/containerapi/instancedata/nodeInstances/20"
  }
]
```

Listing 3 JSON Datenstruktur für OpenTOSCA Events

Ist die zugehörige Node Instance bisher noch unbekannt, erfolgt ein GET Request mit der `nodeInstanceId` des Events an die Ressource `/monitoring/eventstorages/opentosca/nodeinstancelist/{id}`. Die Response ist ebenfalls ein JSON-Objekt, das alle OpenTOSCA Informationen zu dieser Node Instance enthält. Ein Beispiel hierfür ist in Listing 4 zu sehen.

Die erhaltenen Informationen werden dem Cache hinzugefügt und die neue

```
{
  "nodeInstanceID": "http://localhost:1337/containerapi/instancedata/nodeInstances/20",
  "name": "20",
  "created": 1462372442552,
  "nodeType": "UbuntuNodeType",
  "csarID": "Maerker.csar",
  "nodeTemplateID": "UbuntuNodeTemplate",
  "serviceInstanceID": "4",
  "serviceTemplateID": "Maerker",
  "properties": "(...)",
  "state": "running"
}
```

Listing 4 JSON Datenstruktur für eine Node Instance in OpenTOSCA

Node Instance wird als HTML Element in die Benutzeroberfläche eingefügt (Anforderung A11). Dabei werden einige Felder entsprechend formatiert und angepasst, bspw. eine nutzerfreundliche Darstellung von Datum und Uhrzeit oder das Syntax-Highlighting der XML-Properties mittels *highlight.js*. Ferner wird das Pop-up für die Anzeige der Properties vorbereitet und die

Positionierung mittels Drag'n'Drop ermöglicht, beide Funktionen werden von jQuery UI zur Verfügung gestellt.

Abschließend werden die Auswahlfelder für den *Service Instance*- und *Node Type*-Filter aktualisiert, sofern dies nötig ist.

Filter

Bei der Änderung von einer der beiden Filter über die Eingabemasken wird die Funktion `filterNodes()` aufgerufen. Da durch das Filtern möglicherweise nur noch eine Teilmenge der bisher angezeigten Node Instances verbleibt, werden zu Beginn des Vorgangs die Positionen aller Instances zurückgesetzt, wodurch diese nebeneinander aufgereiht werden. Dieses Verhalten verhindert, dass eine durch Drag'n'Drop manuell platzierte Instance plötzlich außerhalb des direkt sichtbaren Bereichs der Benutzeroberfläche gerät.

Darauf hin wird über alle bekannten Node Instances iteriert und dabei überprüft, ob beide gewählten Filter zutreffen. Entsprechend der Überprüfung wird das aktuelle Element ein- oder ausgeblendet.

Löschen von Events

Um alle OpenTOSCA Events zu löschen, wird ein POST Request an die Ressource `/monitoring/eventstorages/opentosca/eventdropper` des Backends geschickt. Bei erfolgreicher Durchführung wird dem Nutzer ein grafisches Feedback angezeigt, der Cache aller bekannten Node Instances gelöscht und selbige in der Benutzeroberfläche entfernt. Abschließend werden alle Filter zurückgesetzt, sowie die davon verwendeten Listen der *Service Instances* und *Node Types* geleert.

6 Zusammenfassung und Ausblick

Der Ablauf langlaufender Prozesse ohne direkten Einblick kann sich aus Nutzersicht oft schlecht auf die Nutzererfahrung auswirken. Ebenfalls erschwert dies, im Hintergrund laufende Prozesse verstehen und nachvollziehen zu können.

Insbesondere die Provisionierung oder Verwaltung einer mit TOSCA beschriebenen Cloud Anwendung besteht in der Regel aus komplexen Prozessen, die längere Zeit bis zu ihrer Fertigstellung benötigen können. Diese werden durch den OpenTOSCA Container angestoßen und erzeugen teilweise Events, die mitunter ihren Ablauf und den Fortschritt repräsentieren. Diese sind für den Nutzer allerdings nicht sichtbar oder nachverfolgbar.

Durch das im Rahmen dieser Arbeit entwickelte Monitoring System können die Events, die im Laufe der Prozesse erzeugt werden, graphisch dargestellt werden, um dem Nutzer Einsicht in aktuelle oder vergangene Abläufe zu geben. Dies ermöglicht zum einen, den Fortschritt der Prozesse zu betrachten oder einschätzen zu können, und bietet zum anderen eine Übersicht mit zusätzlichen Informationen über die Vielzahl möglicherweise gleichzeitig ablaufender Prozesse.

Die Grenzen von allgemeinen Monitoring Events, wie sie im generischen Frontend dargestellt werden, zeigen sich hierbei jedoch schnell. Ohne domänenspezifisches Wissen lassen sich die angezeigten Informationen nur schlecht zuordnen oder verwerten und bieten dem Nutzer nur eine geringe Übersicht und eingeschränkte Handlungsmöglichkeiten.

Ähnlich dem implementierten Generischen- und OpenTOSCA-Frontend lassen sich jedoch aufgrund der REST-API des Backends einfach weitere, spezialisierte Benutzeroberflächen realisieren. Diese könnten für verschiedenste Anwendungsfälle eingesetzt werden. Die Benutzeroberfläche kann dabei entsprechend eine gewünschte, an die Anwendung angepasste Logik verwenden, sowie eine zugeschnittene Darstellungsform umsetzen.

Auch die beiden realisierten Frontends bieten einige Möglichkeiten zur Erweiterung. Insbesondere weitere Filterfunktionen oder Einschränkungen, welche Events dem Nutzer präsentiert werden, sind denkbar. Ein Beispiel hierfür wäre die Option, Events speziell nach ihrem Datum zu filtern. So könnten nur Events aus einem benutzerdefinierten Zeitraum angezeigt werden, oder einzelne Ober- und Untergrenzen für das Datum gesetzt werden.

Literatur

[1] *Bootstrap*. URL: http://getbootstrap.com/ (besucht am 30. 04. 2016).

[2] *CSS Specification*. URL: https://www.w3.org/Style/CSS/ (besucht am 30. 04. 2016).

[3] *DataTables*. URL: https://www.datatables.net/ (besucht am 30. 04. 2016).

[4] *ECMAScript Language*. URL: http://www.ecmascript.org/ (besucht am 30. 04. 2016).

[5] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. University of California, Irvine, 2000.

[6] *HTML5 Specification*. URL: https://www.w3.org/TR/2014/REC-html5-20141028/ (besucht am 30. 04. 2016).

[7] N. Borenstein N. Freed. *Multipurpose Internet Mail Extensions*. 1996. URL: http://www.ietf.org/rfc/rfc2046.txt (besucht am 08. 05. 2016).

[8] *OpenTOSCA*. URL: http://opentosca.org/ (besucht am 30. 04. 2016).

[9] L. Masinter R. Fielding T. Berners-Lee. *Uniform Resource Identifier*. 2005. URL: http://www.ietf.org/rfc/rfc3986.txt (besucht am 08. 05. 2016).

[10] TOSCA Spezifikation. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html (besucht am 30. 04. 2016).

[11] *jQuery*. URL: http://jquery.com/ (besucht am 30. 04. 2016).

Erklärung

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

Ort, Datum, Unterschrift