Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 345

Automatische, TOSCA-basierte Provisionierung des Situationserkennungssystems SitOPT

Armin Hüneburg

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.–Ing. habil. Bernhard Mitschang
Betreuer/in:	M. Sc. Ana Cristina Franco da Silva Dipl.–Inf. Pascal Hirmer
Beginn am:	6. Juni 2016
Beendet am:	6. Dezember 2016
CR-Nummer:	D.0, D.2.1, D.2.2, D.2.3, D.2.6, D.2.9, J.1

Kurzfassung

Das Internet of Things (IoT) übernimmt immer größere Teile unseres Lebens. Dabei beschreibt IoT die Verwendung von Geräten, welche vernetzt werden. Diese Geräte sind mit Sensoren und Aktuatoren ausgestattet. Systeme, die diese Technologie benutzen, werden häufig als intelligent (SMART) bezeichnet. So gibt es beispielsweise SMART Homes, SMART Factories und SMART Cities. Durch die Verwendung der Sensoren werden die einzelnen Bereiche der Umgebung überwacht, wodurch Änderungen schnell festgestellt werden können. Dadurch ist IoT für Firmen sehr interessant. Jedoch wollen Firmen nicht nur über Änderungen in Kenntnis gesetzt werden. Es ist wichtig, dass sofort auf die geänderten Bedingungen reagiert wird und sich die Prozesse in der Firma automatisch an diese anpassen. Ein Projekt, das versucht diese Anforderungen umzusetzen, ist das Projekt *Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten (SitOPT).* Jedoch muss ein Ansatz wie SitOPT auch flexibel und skalierbar sein. Um dies zu erreichen, eignet es sich, das Programm in einer Cloud–Umgebung bereitzustellen. Um eine fehleranfällige, zeitaufwändige manuelle Provisionierung zu vermeiden, wird eine automatische Provisionierung durch den Topology and Orchestration Specification for Cloud Applications (TOSCA)–Standard in dieser Arbeit bereitgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

In den letzten Jahren ist der Einsatz von Sensoren zur Überwachung von Systemen immer weiter gestiegen. Diese sogenannten SMARTEN Systeme werden in immer mehr Gebieten eingesetzt. So gibt es beispielsweise SMART Homes, bei denen die Wohnung im Vordergrund steht. Zudem setzt auch die Industrie immer mehr auf Sensoren. In SMART Factories [LCW08] oder in der Industrie 4.0 [LBK15] werden Maschinen und Prozesse mit Hilfe von Sensoren überwacht. Dazu werden Sensoren in einem Sensorennetzwerk zusammengeschlossen. In diesem Netzwerk werden die Daten der Sensoren ausgelesen und analysiert. Die Analyse kann beispielsweise bei der Verbesserung von Arbeitsprozessen, oder auch der Erkennung von Ereignissen, beitragen. Dabei kann diese automatisiert werden, wodurch sie jeden Tag, rund um die Uhr, stattfinden kann. Zudem werden Fehlerquellen, wie menschliches Versagen, ausgeschlossen.

Cloud Computing [MG11] erlaubt es Nutzern bequem Computerressourcen, beispielsweise Netzwerke oder Server, zu erstellen. Diese Ressourcen sind virtuell, können nach Bedarf vom Nutzer erstellt werden, und sind an den Anwendungsfall angepasst. Die entstehenden Kosten werden bei den Anbietern nach der verwendeten Leistung berechnet. Dies wird „pay–as–you–go“–Prinzip [AG10] genannt. Diese Vorteile des Cloud Computings machen es besonders attraktiv für verteilte Programme.

Verteilte Programme oder Dienste können automatisch auf den erstellten Ressourcen bereitgestellt werden. Dadurch ergibt sich für den Nutzer kaum Aufwand, um neue Instanzen von bekannten Umgebungen zu erstellen. Der Nutzer muss lediglich eingreifen, wenn Besonderheiten benötigt werden, die für die Automatisierung nicht ersichtlich sind. Dazu können beispielsweise besondere Einstellungen der Ressourcen gehören.

Durch die Kombination von Cloud Computing, automatischer Bereitstellung und SMARTEN Systemen, wird ein variables System erzeugt. Dieses lässt sich bequem an die Anforderungen der Umgebung anpassen. So können neue SMART Factories beispielsweise dadurch erstellt werden, dass bereits vorhandene Instanzen der Fabrik mit all ihren Ressourcen kopiert werden. Diese müssen gegebenenfalls noch angepasst werden. Somit ist die Kombination von Cloud Computing und SMARTEN Systemen in sich erweiternden Umgebungen von Vorteil.

Mit der Provisionierung in der Cloud werden die Programme für Anwender zur Verfügung gestellt. Dieser Schritt benötigt oft viel Zeit, da Cloud–Applikationen oft viele Einstellungsmöglichkeiten bieten, wobei der Großteil nur in Spezialfällen nötig ist. Aus diesem Grund gibt es Provisionierungsplattformen, die viele dieser Einstellungen automatisch vornehmen. Zudem

muss sich die Person, die versucht die Applikation aufzusetzen, nicht damit beschäftigen, wo die Einstellungen vorgenommen werden müssen, da die Provisionierungsplattform das komplette Bereitstellen übernimmt. In dieser Arbeit wird die Anwendung SitOPT mit Hilfe des Provisionierungsstandards TOSCA bereitgestellt. SitOPT dient zur Erkennung von Situationen. Dazu werden Sensordaten in einen zentralen Speicher, RMP genannt, eingelesen, welcher als Zwischenspeicher verwendet wird. Aus diesem können sie jederzeit wieder ausgelesen und zur Erkennung von Situationen genutzt werden. TOSCA ist ein Standard, der zur Provisionierung und Management von Diensten entworfen wurde.

Gliederung

Diese Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden die Grundlagen erklärt, die zum Verständnis der Arbeit benötigt werden. Die Grundlagen beinhalten den TOSCA–Standard, OpenTOSCA sowie SitOPT.

Kapitel 3 – Verwandte Arbeiten: TOSCA ist nicht der einzige Provisionierungsstandard. Aus diesem Grund werden in diesem Kapitel Andere erwähnt und beschrieben. Zudem werden Arbeiten beschrieben, die ein ähnliches Ziel hatten wie diese, die Provisionierung eines Systems.

Kapitel 4 – Provisionierung des Situationserkennungssystems SitOPT: Dieses Kapitel stellt den Kern der Arbeit dar. Hier wird beschrieben, wie die einzelnen Module von SitOPT mit Hilfe von TOSCA provisioniert werden können. Zunächst wird die Topologie des gesamten Systems erklärt um anschließend die Implementierung von zwei Teilen, Ressource–Management–Platform und Situation–Template–Modeling–Tool, zu erklären.

Kapitel 5 – Zusammenfassung: Das letzte Kapitel fasst die Inhalte dieser Arbeit zusammen. Zudem wird ein Überblick über mögliche Arbeiten gegeben, die die Konzepte und Implementierung erweitern.

2 Grundlagen

In diesem Kapitel werden Grundlagen erklärt, die zum allgemeinen Verständnis der Arbeit notwendig sind. Diese beinhalten den TOSCA–Standard, der von der Advancing Open Standards for the Information Society (OASIS)–Organisation entworfen wurde, sowie OpenTOSCA[1] und SitOPT. OpenTOSCA ist eine Implementierung des TOSCA–Standards, die an der Universität Stuttgart entwickelt wurde. Diese Implementierung wird für die Umsetzung der Arbeit verwendet. SitOPT ist eine Cloud–Applikation, von der in dieser Arbeit Teile durch OpenTOSCA bereitgestellt werden.

2.1 Der TOSCA–Standard

TOSCA [OAS13] ist eine Spezifikation für das Bereitstellen und Verwalten von Diensten oder Bestandteilen dieser. Der TOSCA–Standard besteht aus zwei Teilen. Diese Teile sind Topologien und die Orchestration von Diensten. Topologien beschreiben Komponenten von Diensten, die notwendig sind, um den diesen zu betreiben (z.B. Applikationsserver, Datenbank, Betriebssystem). Dabei kann der Modellierer entscheiden, welche Teile durch die automatisierte Provisionierung umgesetzt werden sollen und welche durch den Modellierer bereitgestellt werden. So kann er die Infrastruktur, beispielsweise eine virtuelle Maschine für einen Server, beschreiben, damit diese miterstellt wird. Allerdings kann er die Infrastruktur selbst bereitstellen. Nachträglich wird die Nutzung der vorhandenen Infrastruktur beschrieben. Beispielsweise kann eine bereits vorhandene virtuelle Maschine genutzt werden, wenn die benötigten Daten bereitgestellt werden. Genauso kann er durch die automatische Provisionierung nur einen Teil bereitstellen lassen, zum Beispiel die Infrastruktur, um daraufhin den gewollten Dienst manuell zu provisionieren. Es ist auch möglich das gesamte System zu beschreiben, damit TOSCA alle notwendigen Komponenten bereitstellt.

Topologien werden als Graph dargestellt. Dabei sind die Knoten sogenannte *NodeTemplates*. Diese stellen die Komponenten des Dienstes dar, welche durch TOSCA bereitgestellt werden sollen. Gleichzeitig müssen auch deren Eigenschaften, in TOSCA genannt *Properties*, mit angegeben werden. Zwischen den Komponenten gibt es Relationen, die als Kanten im Graphen dargestellt werden. Diese Kanten werden *RelationshipTemplates* genannt. RelationshipTemplates bestimmen, wie die Komponenten zusammenhängen. So können über RelationshipTemplates

[1] http://opentosca.org

Abhängigkeiten, wie „muss installiert sein“ oder „muss gestartet sein“, festgelegt werden. Ein Beispiel einer Topologie wird in Abbildung 2.1 gezeigt. Diese würde eine WebApp mit ihren Abhängigkeiten, Java und Tomcat, bereitstellen, die eine Verbindung zu einer Datenbank aufbaut, welche auf einem anderen Server bereitgestellt ist. Die Datenbank wird ebenfalls von TOSCA bereitgestellt.

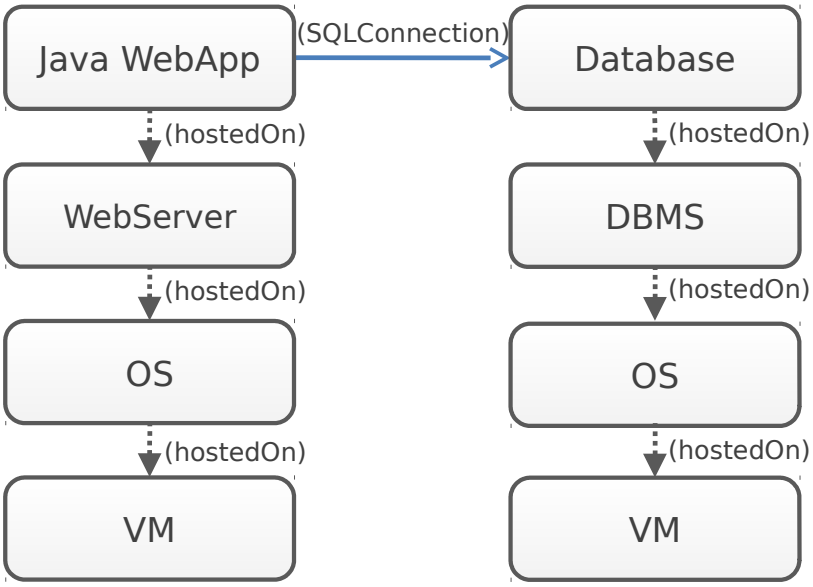

Abbildung 2.1: Topologie einer Webanwendung, die eine Datenbank auf einem anderen Server nutzt (aus [BBK+14]).

NodeTemplates und RelationshipTemplates werden durch *NodeTypes* und *RelationshipTypes* typisiert. Diese enthalten *DeploymentArtifacts* und *ImplementationArtifacts*, die beschreiben, wie eine Komponente des Dienstes bereitgestellt werden soll. DeploymentArtifacts beinhalten Dateien, die für die Bereitstellung des Diensts benötigt werden, beispielsweise zu installierende Pakete, der Quellcode, der kompiliert werden muss, oder auch das fertige Kompilat, das gestartet werden muss. ImplementationArtifacts beschreiben, wie verschiedene Aufgaben, die bei der Provisionierung benötigt werden, ablaufen. So enthalten ImplementationArtifacts beispielsweise Installationsskripte und Startskripte. Allerdings muss kein DeploymentArtifact vorhanden sein, wenn das ImplementationArtifact auch auf andere Weise, beispielsweise durch das Herunterladen von einer Webseite, auf die benötigten Daten zugreifen kann. So kann ein Java Service, der ein Bild verarbeiten soll, durch ImplementationArtifacts bereitgestellt werden. Das Bild, als nicht ausführbare Komponente, wird durch ein DeploymentArtifact bereitgestellt. Ein Beispiel für ein ImplementationArtifact wird im Listing 2.1 gezeigt. Die Elemente, die von ImplementationArtifacts und DeploymentArtifacts benötigt werden, werden durch *ArtifactTemplates* bereitgestellt.

Topologien können sowohl grafisch beschrieben werden, als auch durch XML [KBBL13] oder YAML [OAS16]. Ein grafischer Editor ist beispielsweise im OpenTOSCA–Ökosystem enthalten, welches in Unterabschnitt 2.2.1 erklärt wird.

Listing 2.1 Teil eines ImplementationArtifacts zum Installieren eines MySQL–Servers

```
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get install -y mysql-server
```

Die Orchestrierung der Services kann auf zwei Arten vonstatten gehen. Die erste Art ist deklarativ, wobei die TOSCA–Laufzeitumgebung alle Komponenten des Dienstes kennen muss. Allerdings reicht es, wenn der Nutzer eine Topologie erstellt. TOSCA kann dann die einzelnen Komponenten aus der Topologie identifizieren und bereitstellen. Die andere Art ist imperativ. Bei dieser Vorgehensweise wird ein Provisionierungsplan erstellt, der beschreibt, wie Dienste bereitgestellt werden. Dabei müssen alle notwendigen Schritte beschrieben werden. Dadurch bleiben TOSCA–Implementierungen generisch, da diese genutzte Dienste nicht kennen muss.

Die mit TOSCA beschriebenen Komponenten können in Cloud Service Archives (CSARs) gepackt werden. Diese sind Zip-Archive, die einen standardisierten Aufbau haben.

2.2 OpenTOSCA

OpenTOSCA wurde von der Universität Stuttgart entwickelt und setzt den TOSCA–Standard um. OpenTOSCA besteht aus mehreren Modulen, die dem Nutzer erlauben den TOSCA Standard zu nutzen. Diese Module sind:

1. Winery Modeling Tool [KBBL13]
   Modul zum Erstellen von Topologien
2. Administrative UI
   Modul zum Hochladen von CSARs
3. Vinothek Self–Service Portal [BBK+14]
   Modul zum Starten von Provisionierungsplänen
4. OpenTOSCA Container [BBH+13]
   Modul zur Durchführung von Provisionierungsplänen

2.2.1 Winery Modeling Tool

Dieses Modul erlaubt das Erstellen von Topologien sowie von allen in TOSCA benötigten Typdefinitionen. Dafür können beispielsweise NodeTypes und RelationshipTypes erstellt und mit ImplementationArtifacts oder DeploymentArtifacts versehen werden. Diese können wiederum mit ArtifactTemplates ausgestattet werden. Um ArtifactTemplates mit Aktionen zu verknüpfen, werden sogenannte *Interfaces* genutzt, die angeben, welche Aktionen vorhanden sind und mit welchen Parametern diese aufgerufen werden sollen. Zusätzlich werden Rückgabewerte

angegeben. Zudem können Properties für die NodeTypes angegeben werden. Die fertigen NodeTypes können dann in Topologien als angeleitete NodeTemplates modelliert werden, in welchen Properties mit Werten befüllt werden können. Ein Beispiel einer Topologie ist in Abbildung 2.2 zu sehen.

Das Winery Modeling Tool erlaubt dem Nutzer jedes Element als CSAR zu exportieren. So können beispielsweise einzelne NodeTypes, RelationshipTypes, aber auch ganze Topologien exportiert werden. Zudem können CSARs importiert werden, um dem Winery Modeling Tool die einzelnen Elemente hinzuzufügen.

2.2.2 Administrative UI

Die administrative UI erlaubt es dem Nutzer CSARs hochzuladen. Diese können beispielsweise aus dem Winery Modeling Tool stammen. Nach dem Hochladen werden die CSARs überprüft, ob sie eine Topologie beinhalten. Wenn dies der Fall ist, werden sie zum OpenTOSCA Container weitergeleitet, der einen Provisionierungsplan erstellt. Sobald der Import erfolgt ist und keine Fehler aufgetreten sind, kann der Nutzer den Inhalt der CSAR untersuchen. In Abbildung 2.3 wird die Oberfläche des Moduls gezeigt.

2.2.3 Vinothek Self–Service Portal

Das Vinothek Self–Service Portal bietet dem Nutzer die Möglichkeit Provisionierungspläne zu starten. Dazu kann er die deploymentspezifischen Daten von noch leeren Properties befüllen. Deploymentspezifische Daten sind Daten, die bei jeder Durchführung anders sein können, wie zum Beispiel der Nutzername einer virtuellen Maschine. In Abbildung 2.4 wird die Abfrage eines solchen Datensatzes gezeigt. Sobald versucht wird eine Provisionierung zu starten, wird überprüft, ob benötigte Datensätze fehlen. Falls dies der Fall ist, wird eine Fehlermeldung angezeigt. Wenn alle Daten ausgefüllt sind, sieht der Nutzer, dass der Provisionierungsplan ausgeführt wird. Nachdem diese erfolgreich beendet wurde, bekommt der Nutzer eine Rückmeldung. Diese kann auch Daten des Provisionierungsplans enthalten, beispielsweise die IP–Adresse einer erstellten virtuellen Maschine.

2.2.4 OpenTOSCA Container

Der OpenTOSCA Container dient zur Generierung von Provisionierungsplänen aus Topologien [Kep13]. Für diesen Schritt werden zunächst die Abhängigkeiten für die Installation über die RelationshipTemplates ermittelt. Anschließend wird bestimmt, wie die Kommunikation zwischen den Diensten stattfindet, damit die Reihenfolge der ImplementationArtifacts festgestellt werden kann. Die Kommunikation der verschiedenen Dienste ist über die RelationshipTemplates festgelegt. Nach der Ablaufermittlung wird ein Provisionierungsplan, der

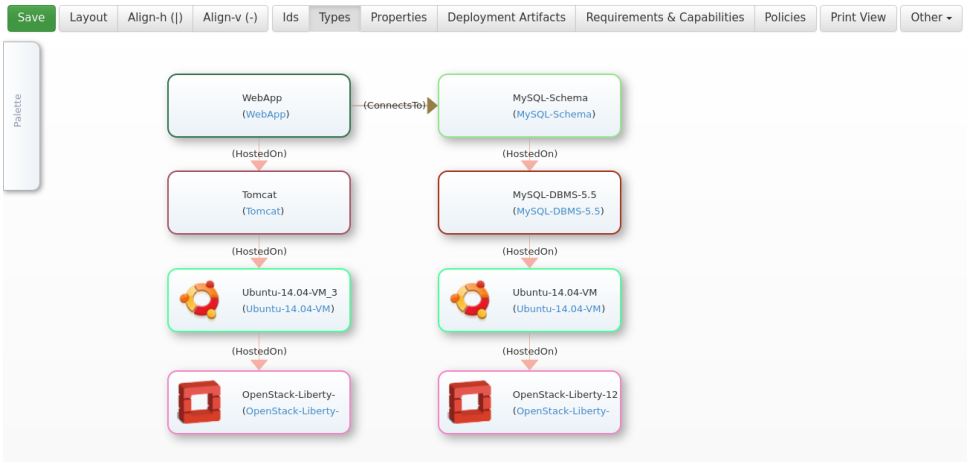

Abbildung 2.2: Eine Topologie im Winery Modeling Tool

Abbildung 2.3: Administrative UI

Abbildung 2.4: Abfrage einer fehlenden Information im Vinothek Self–Service Portal

auf BPEL basiert, erstellt. Ein Beispiel, wie RelationshipTemplates die Ausführungsreihenfolge beeinflussen können wird in Abbildung 2.5 dargestellt. In der linken Grafik wird gezeigt, wie die Architektur einer Webanwendung aussieht. Diese wurde im rechten Bild so umgewandelt, dass die richtige Bereitstellungsreihenfolge sichergestellt wird.

Zudem dient der OpenTOSCA Container zur Durchführung von Provisionierungen. Dazu werden die Provisionierungspläne ausgeführt, welche beispielsweise durch das Hochladen von Topologien in die Administrative UI erstellt wurden. Der Container stellt zwei Operationsmöglichkeiten zur Verfügung. Entweder werden Dienste direkt angesprochen, zum Beispiel die Amazon EC2 Schnittstelle, oder es werden ImplementationArtifacts genutzt [BBH+13]. Der Container führt ImplementationArtifacts oder die Anfragen an die Endpunkte von Diensten aus. Nach der Ausführung wartet er auf eine Antwort. Diese wird daraufhin geprüft, ob sie eine Fehlermeldung beinhaltet. Wenn dies nicht der Fall ist, wird die nächste Anfrage oder das nächste ImplementationArtifact gestartet.

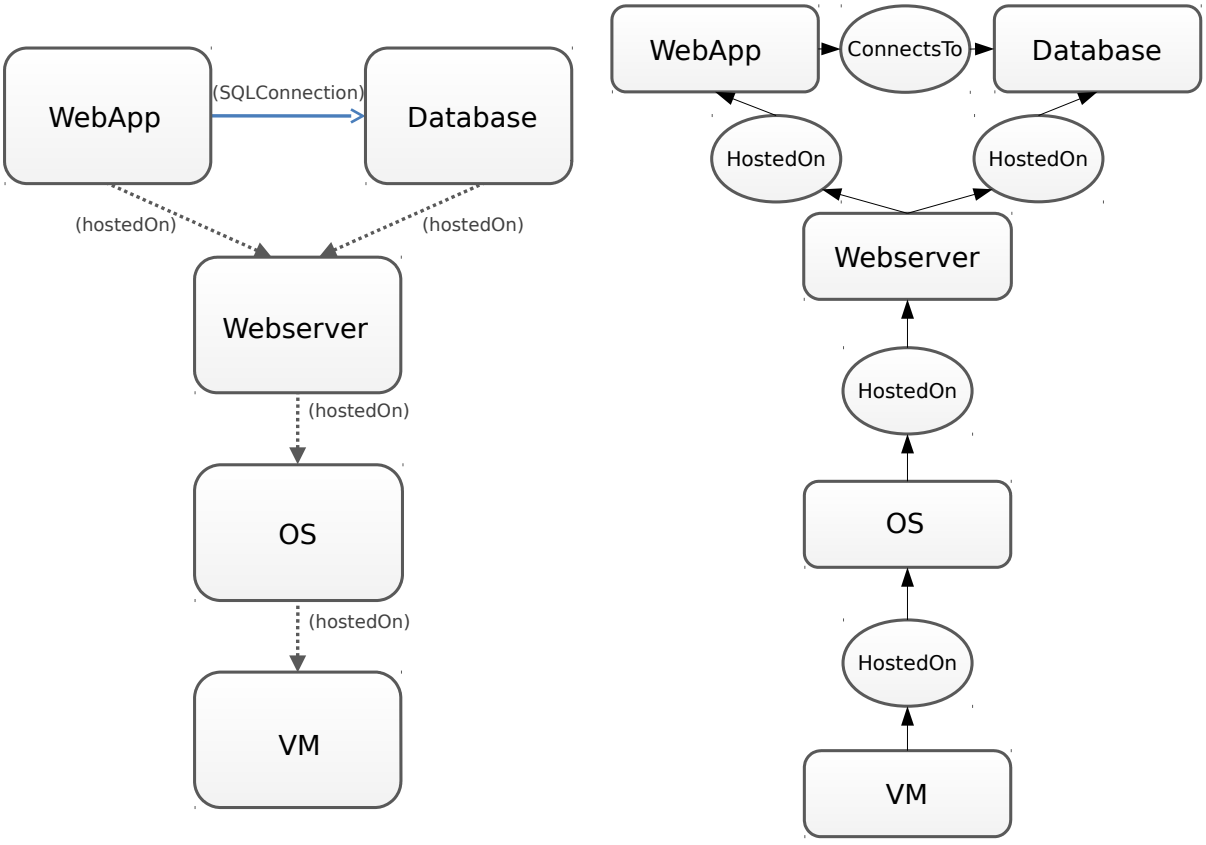

Abbildung 2.5: Schema zur Umsetzung einer Topologie zur Ausführungsreihenfolge, nach [Kep13]

2.3 SitOPT

SitOPT ist eine Zusammenstellung von Modulen, die es erlauben, Situationen auf Basis von Sensordaten zu erkennen [HWS+15; HWS+16; SHWM16; WSBL15]. Für diesen Zweck werden diese Objekten zugeordnet, die die virtuelle Repräsentation von „realen Dingen“ sind. Beispielsweise wird ein Temperatursensor einem virtuellen Objekt *Raum* zugeordnet, welcher einen realen Raum darstellt. Aus den Daten, die von den verschiedenen Sensoren der Umgebung bereitgestellt werden, lassen sich allerdings keine komplexeren Zusammenhänge erkennen. Aus diesem Grund wurde von Hirmer et al. in [HWS+15] ein Vorgehen beschrieben, das dieses Problem löst. Dieses ist in Abbildung 2.6 dargestellt.

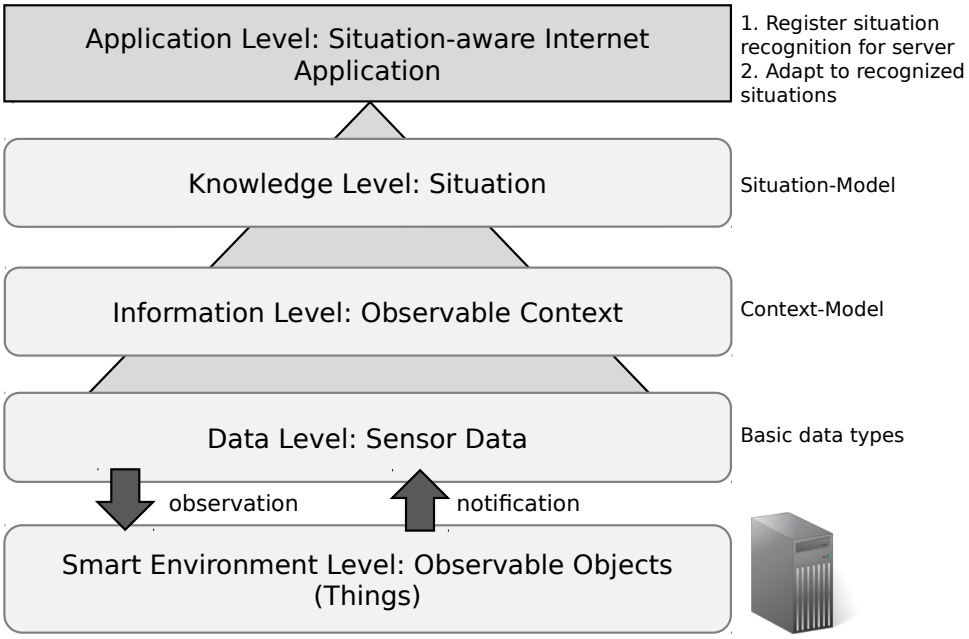

Abbildung 2.6: Weg der Daten zu komplexen Informationen aus [HWS+15]

In der untersten Schicht befindet sich die mit Sensoren ausgestattete Umgebung, die als SMART bezeichnet wird. Diese Sensoren, liefern Daten, welche von der zweiten Schicht – der Datenschicht – dargestellt werden. Die gesammelten Daten beinhalten jedoch noch kaum zusätzliches Wissen.

Um diese Daten besser nutzen zu können, werden sie in einen Kontext gebracht, der durch die virtuellen Objekte realisiert ist. Die Eigenschaften dieser virtuellen Objekte werden mit Hilfe der Sensordaten befüllt. In der Abbildung 2.6, stellt die Assoziation mit dem Kontext das Kontext–Level dar.

Aus den Eigenschaften der Objekte kann komplexeres Wissen, auch *Situation* genannt, abgeleitet werden. Situationen bezeichnen einen Zustand, den ein virtuelles Objekt und somit auch das

entsprechende reale Ding, hat. So kann beispielsweise aus den verschiedenen Eigenschaften eines Computers hergeleitet werden, ob dieser überlastet ist.

In SitOPT werden Situationen von den virtuellen Objekten abstrahiert. Hierfür wurden SituationTemplates, die in XML beschrieben werden, definiert [HWS+15] um Bedingungen für eine Situation zu beschreiben. In diesen ist beschrieben, welche Eigenschaften an einer Situation beteiligt sind, welche Bedingungen für die Sensorwerte gelten müssen und wie diese miteinander verknüpft sind. Dabei werden die Sensorwerte der Objekte mit Werten verglichen, die Grenzen für das Eintreffen der Situation festlegen. Da an einer Situation mehrere Sensorwerte beteiligt sein können, müssen diese zusammengeführt werden. Dies wird durch logische Operationen erreicht. Diese Berechnung bestimmt, ob die Situation eintritt oder nicht. So kann beispielsweise die Situation *Feuer* erkannt werden, wenn die Eigenschaft *Sauerstoffgehalt* eines *Raums* unter 12% und die Eigenschaft *Temperatur* des *Raums* über 100°C liegt. In Abbildung 2.7 ist zu sehen, wie diese Situation erkannt werden kann. Die Knoten *O2Sensor* und *TempSensor* repräsentieren die Werte, die von den Sensoren ausgelesen werden. Durch die Knoten *wenigerAls12* und *mehrAls100* wird überprüft, ob der entsprechende Wert diese Bedingung erfüllt. Die logischen Werte der Vergleiche werden anschließend durch den Knoten *UND* mit einer und–Operation vereinigt. Der *Feuer*–Knoten wird genutzt, um die Situation zu speichern.

SitOPT wurde in verschiedene Module eingeteilt, die die einzelnen Schritte des Vorgehens zur Erkennung von Situationen übernehmen. Diese werden im restlichen Kapitel beschrieben. Eine Zuordnung der Module in die einzelnen Schichten von Abbildung 2.6 ist in Abbildung 2.8 zu sehen.

2.3.1 Resource Management Platform

Die RMP[2] [HSW+16; HWBM16a; HWBM16b] beinhaltet die Schichten des Empfangens der Sensordaten sowie die Zuordnung dieser zu Objekten. Um dies zu ermöglichen, bietet die RMP registrierten Sensoren die Möglichkeit, ihre Werte zu speichern. Dazu müssen diese zur RMP geschickt werden. Bei der Sensorenregistrierung müssen das virtuelle Objekt sowie Metadaten zu dem Sensor angegeben werden. Diese Metadaten beinhalten die Art (bspw. Temperatursensor) und die Qualität des Sensors sowohl als auch die verwendete Einheit und deren Symbol.

Sobald der Sensor registriert ist, wird eine REST–Ressource erstellt, die die Metadaten und den letzten Wert des Sensors anbietet. Zudem können die Metadaten und der Wert über eine REST–Schnittstelle aktualisiert werden. Die REST–Ressource wird über das virtuelle Objekt und die Kennung des Sensors spezifiziert. Dies erlaubt dem Nutzer mehrere Sensoren mit gleicher Kennung zu definieren, die allerdings unterschiedlichen Objekten zugewiesen sind. So können beispielsweise zwei Maschinen mit der Kennung „Maschine1“ und „Maschine2“ jeweils

[2] https://github.com/SitOPT/Resource-Management-Platform

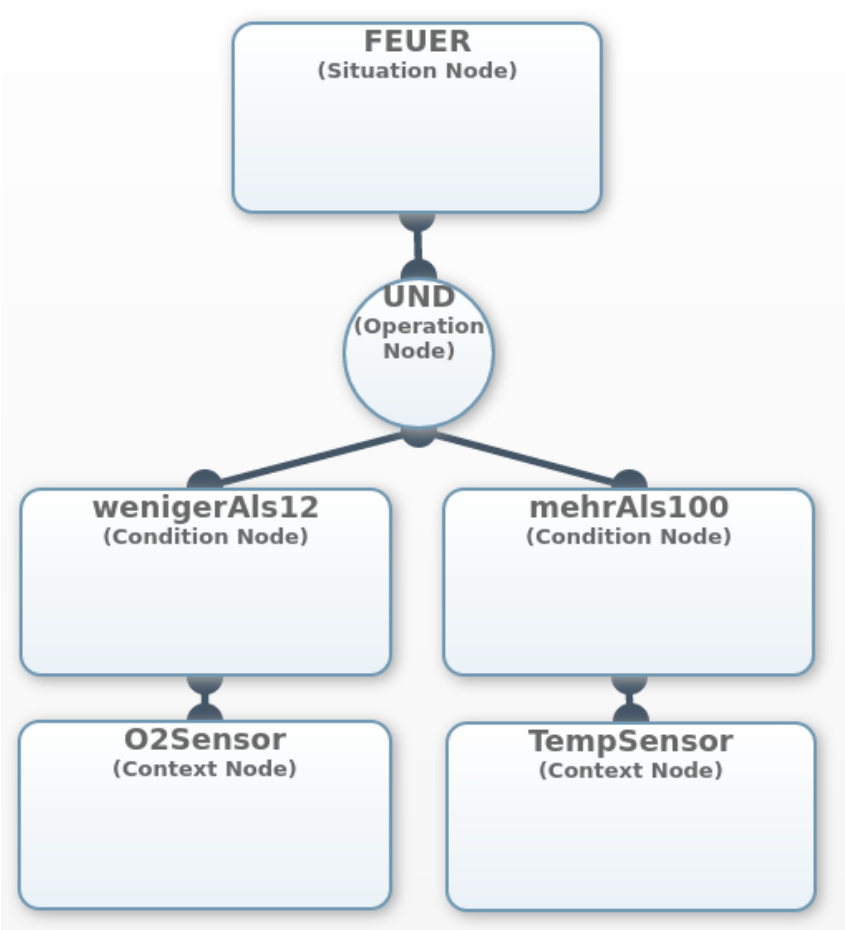

Abbildung 2.7: Erkennung der Situation *Feuer*

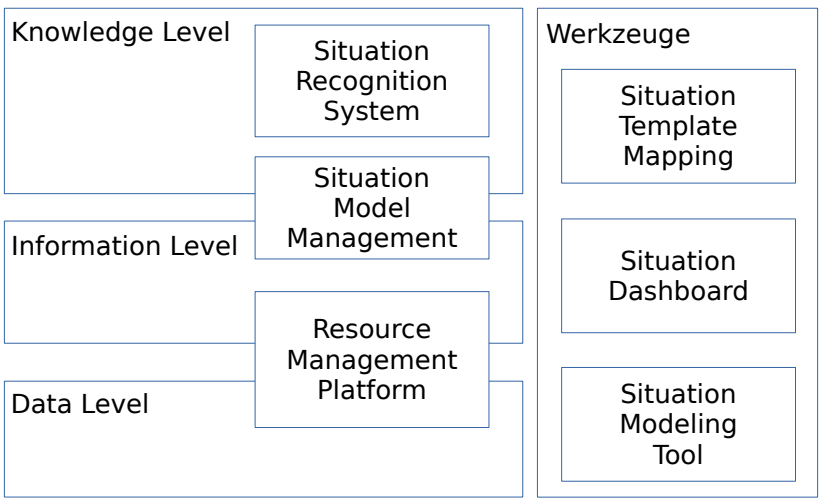

Abbildung 2.8: Einteilung der Module von SitOPT in die Schichten von Abbildung 2.6 sowie in zusätzliche Werkzeuge

einen Sensor haben, der die Kennung „TempSensor“ hat. Diese können zwei unterschiedliche Sensoren sein.

2.3.2 Situation-Model-Management

Das Modul Situation-Model-Management[3] speichert Sensoren, virtuelle Objekte, Situationen und SituationTemplates. Zudem werden Daten für Berechtigungen, Reaktionen auf und Erkennungen von Situationen gespeichert. Das Modul erstellt REST–Ressourcen für jedes gespeicherte Element. Dabei werden Objekte und SituationTemplates mit eindeutigen Kennungen versehen. Sensoren werden ebenfalls mit Kennungen versehen, allerdings müssen sie auch einem virtuellen Objekt zugeordnet werden. Um Sensoren zu identifizieren, müssen sowohl seine eigene als auch die Kennung des virtuellen Objektes angegeben werden. Ebenso werden Situationen über die Kennung des Templates sowie über die zugeordneten des Objektes identifiziert. Überwachungen von Situationen sind auf die gleiche Weise eindeutig erkennbar. Um auf Änderungen von Situationen reagieren zu können, müssen Registrierungen auf Situationen vorgenommen werden. Um dies zu bewerkstelligen, werden die Kennungen des SituationTemplates und des virtuellen Objekts genutzt. Dabei wird eine URL angegeben, die das Modul ansteuert, sobald sich der Zustand einer Situation (erkannt oder nicht erkannt) ändert.

2.3.3 Situation Recognition System

Das Situation Recognition System nutzt die beiden Technologien Node–Red[4] und Esper[5]. Diese werden zur Erkennung von Situationen in SitOPT verwendet. Node–Red eine Flow–Engine, die zur Verarbeitung von Daten dient, die über verschiedene Quellen eingelesen werden können. Zu diesen Quellen gehören beispielsweise MQTT[6], HTTP-Requests oder Websockets. Die zurückgelieferten Daten können anschließend über selbstgeschriebene Funktionen transformiert und verschiedene Mechanismen, wie MQTT oder UDP, weitergesendet werden. Ein Beispiel für eine komplette Verarbeitung befindet sich in Abbildung 2.9.

Dabei haben die Knoten folgende Funktionen:

Der blaue Knoten startet die Verarbeitung in regelmäßigen Abständen.

Die gelben Knoten stellen HTTP–Requests dar. Die linken Knoten holen die Anfragen aus der RMP, während der rechte Knoten die Situation speichert.

Die orangen Knoten beinhalten Funktionen zur Datenverarbeitung.

[3] https://github.com/SitOPT/Situation-Model-Management
[4] http://nodered.org/
[5] http://www.espertech.com/esper/
[6] http://mqtt.org/

Esper ist ein Complex Event Processing (CEP)–System, das eine Anbindung von Java und .Net ermöglicht. Es berechnet die komplexen Events durch Anfragen an die Engine. Diese Anfragen werden komplett intern behandelt, wodurch Esper auch als Speicher für Daten genutzt wird. Um die Ergebnisse der Anfragen auszulesen, müssen diese registriert werden.

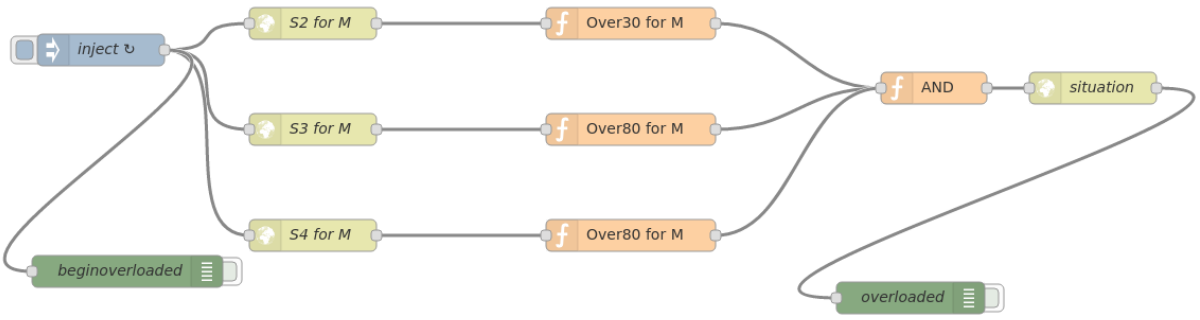

Abbildung 2.9: Beispiel eines Node–Red Datenverarbeitungsstroms.

2.3.4 Situation-Template-Mapping

Dieses Modul besteht aus zwei Untermodulen; eines für Esper[7] und eines für Node–Red[8] (siehe Unterabschnitt 2.3.3). Sie wandeln ein SituationTemplate in eine Anfrage für Esper oder einen Flow für Node–Red um. Die Eingaben in die Untermodule sind der eindeutige Name des Objekts, für das die Situation erkannt werden soll, als auch das Template – entweder als XML–Datei oder als XML–String. Nach der Umwandlung werden die transformierten Erkennungen von Situationen in die entsprechende Engine hochgeladen.

2.3.5 Situation–Dashboard

Das Situation–Dashboard[9] bietet die Möglichkeit, die Schnittstelle des Situation–Model–Managements mit Hilfe einer Weboberfläche anzusprechen. Dabei ist sie in mehrere Abschnitte aufgeteilt. Unter dem Reiter *Things* werden Situationserkennungen für virtuelle Objekte gestartet. Hier wird das Objekt gesucht, für das die Erkennung gestartet werden soll. Anschließend wird das SituationTemplate gewählt, mit dem diese durchgeführt werden soll. Sobald die Auswahl getroffen wurde, ob die Erkennung mit Esper oder mit Node–Red stattfinden soll, kann sie gestartet werden. Die Darstellung eines virtuellen Objektes mit den entsprechenden Feldern

[7] https://github.com/SitOPT/Situation-Template-Mapping_Esper
[8] https://github.com/SitOPT/Situation-Template-Mapping_Node-Red
[9] https://github.com/SitOPT/Situation-Dashboard

und Einstellmöglichkeiten wird in Abbildung 2.10 gezeigt. Unter dem Reiter SituationTemplates werden alle im Situation–Model–Management gespeicherten SituationTemplates gezeigt. Die Darstellung eines SituationTemplates im Situation–Dashboard wird in Abbildung 2.11 gezeigt. Der Reiter *API Reference* erlaubt es direkt auf die Schnittstelle des Situation–Model–Managements zuzugreifen, mit einer Swagger[10]–Oberfläche dargestellt wird. Swagger bietet die Möglichkeit REST–Schnittstellen zu dokumentieren und zu testen. Um dies zu erreichen, werden die einzelnen Endpunkte, ihre Dokumentation, sowie Felder zum Ausfüllen von Anfragen angezeigt. Swagger wird in Abbildung 2.12 dargestellt. Der Reiter *NodeRed* zeigt alle Erkennungen, die derzeit auf Node–Red laufen, und erlaubt diese zu beenden. Die Darstellung einer Erkennung wird in Abbildung 2.13 gezeigt.

Abbildung 2.10: Die Things–Ansicht im Situation–Dashboard

Abbildung 2.11: Ein SituationTemplate, wie es im Situation–Dashboard dargestellt wird

[10] http://swagger.io/

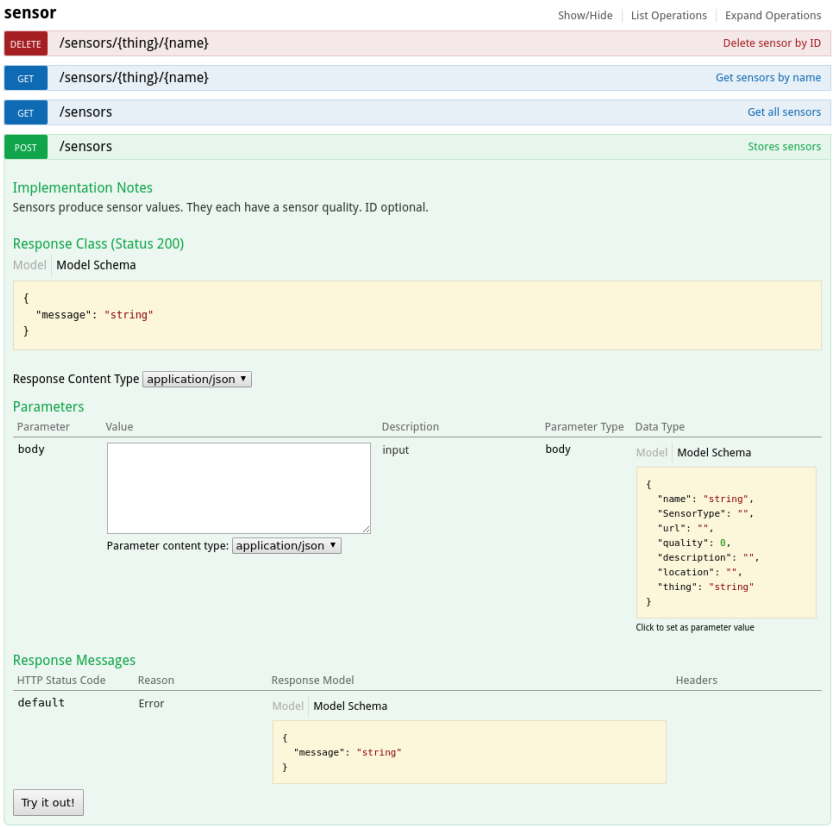

Abbildung 2.12: Eine Ansicht von Swagger

Abbildung 2.13: Darstellung einer Situationserkennung im Situation–Dashboard

2.3.6 Situation-Template-Modeling-Tool

Das Situation–Template–Modeling–Tool[11] erlaubt dem Benuter das einfache Modellieren von SituationTemplates. Dabei kann der Nutzer die verschiedenen Knotenarten per Drag–and–Drop in eine Modellierungsfläche ziehen. Diese Knotenarten sind:

[11] https://github.com/SitOPT/Situation-Template-Modeling-Tool

- Context Node
  Diese Knotenart repräsentiert die Eingabe von Daten, wie sie beispielsweise durch Anfragen an die RMP bereitgestellt werden.
- Condition Node
  Diese Art erlaubt es den Vergleich von Daten mit Konstanten zu modellieren. Die Daten werden dabei durch Context Nodes bereitgestellt, während die Konstanten direkt eingetragen werden.
- Operation Node
  Operation Nodes erlauben es verschiedene Condition Nodes zu verbinden. Dies wird durch logische Verknüpfungen erreicht.
- Situation Node
  Ein Situation Node markiert welches Ergebnis der logischen Kombinationen bestimmt, ob die Situation zutrifft.

Zwischen Knoten können Kanten gezogen werden, die beschreiben, wie der Informationsfluss stattfinden soll. Dabei gibt es einige Einschränkungen:

- Situation Nodes können nur eine eingehende Kante haben
- Situation Nodes können Kanten von Operation Nodes, Condition Nodes oder Context Nodes haben
- Operation Nodes können andere Operation Nodes, sowie Condition Nodes, als auch Context Nodes als direkte Kinder haben
- Condition Nodes können nur Kanten von Context Nodes als Eingang haben

Das Situation–Template–Modeling–Tool erlaubt dem Benutzer die Verifikation seines modellierten Templates. Dabei werden sowohl die genannten Einschränkungen überprüft, als auch folgende:

- Es dürfen keine Zyklen im Graphen vorhanden sein
- Es müssen alle Felder ausgefüllt sein
- Es muss jeder Knoten verwendet werden

Der Nutzer kann das Template exportieren und anschließend im Situation–Model–Management hochladen. Das dann zur Erkennung von Situationen genutzt werden. Die Oberfläche des Situation–Template–Modeling–Tools mit einem beispielhaften SituationTemplate ist in Abbildung 2.14 sehen.

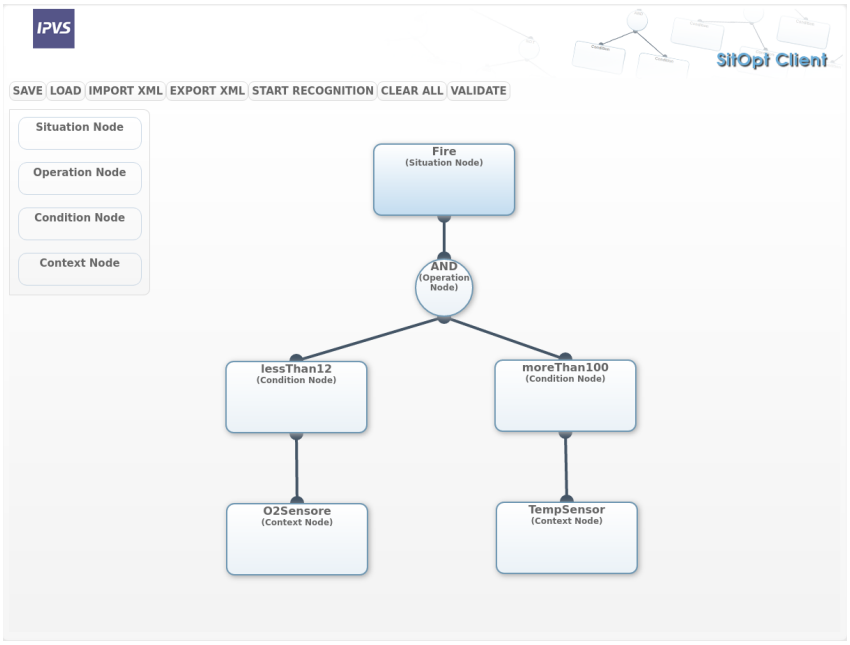

Abbildung 2.14: Oberfläche des Situation–Template–Modeling–Tools mit einem beispielhaften SituationTemplate

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt. Diese befassen sich vor allem mit dem Vergleich von verschiedenen Orchestrierungsplattformen, wie TOSCA oder OpenStack Heat[1]. Zudem werden Arbeiten beschrieben, die bereits Dienste durch OpenTOSCA provisioniert haben.

3.1 Automatische Provisionierung von Anwendungen

Der TOSCA–Standard bietet die Möglichkeit Cloud–Applikationen zu Provisionieren. Allerdings gibt es noch weitere Orchestrierungsplattformen. Die Großen neben TOSCA sind OpenStack Heat und AWS CloudFormation[2].

AWS CloudFormation ist Amazons Orchestrierungsplattform für die eigene Cloudlösung AWS. Dabei werden Services mit Hilfe von Templates und Stacks beschrieben [MCRG15]. In Templates werden Ressourcen und ihre Eigenschaften in JSON definiert. Stacks sind Sammlungen von Ressourcen. Diese können gemeinsam erstellt, verändert oder gelöscht werden. Änderungen am Stack werden dabei in den einzelnen Ressourcen übernommen.

OpenStack Heat wurde für die OpenStack–Plattform entworfen. Heat erlaubt sowohl die Verwendung von AWS Cloudformation Templates, als auch von Heat Orchestration Templates [MCRG15]. Heat Orchestration Templates werden in YAML verfasst und beinhalten, genauso wie bei AWS CloudFormation, die Beschreibungen von Ressourcen. Allerdings können noch weitere Elemente beschrieben werden, wie beispielsweise Autoskalierung [MCRG15]. Heat verwendet ebenfalls Stacks zum Gruppieren von Ressourcen.

Da sowohl AWS CloudFormation als auch Heat speziell von Plattformen unterstützt werden müssen, wurde sich gegen diese Systeme entschieden. TOSCA erlaubt, dass Nutzer eigene Bibliotheken bereitstellen können, um Schnittstellen selbst anzusprechen und ist somit generischer. So gibt es eine Bibliothek, die bereits die OpenStack–Schnittstelle anspricht [Das14], um Instanzen dort anzulegen. Somit wurde ein Teil von Heat bereits für OpenTOSCA bereitgestellt.

[1] https://wiki.openstack.org/wiki/Heat
[2] https://aws.amazon.com/cloudformation/

Für den TOSCA–Standard gibt es zwei unterschiedliche Implementierungen, OpenTOSCA und Cloudify[3], welches unter anderem von ARIA[4] genutzt wird. Hier wurde sich für OpenTOSCA entschieden, da es an der Universität Stuttgart entwickelt wurde, wodurch bereits Expertise vorhanden war. Zudem gibt es Werkzeuge, wie Winery, die den Umgang mit und die Modellierung für OpenTOSCA erleichtern.

Ein weiteres Programm, das zur Provisionierung genutzt werden kann, ist Puppet[5], worin eine virtuelle Maschine deklarativ beschrieben wird. Dabei wird festgelegt, welchen Zustand die Maschine haben soll, z. B. welche Programme installiert sein sollen.

Puppet hat das Problem, dass es deklarativ ist. Somit erlaubt es nur über Umwege, beliebige Programme bereitzustellen. Aus diesem Grund wurde es nicht für die Umsetzung dieser Arbeit gewählt.

Ebenso kann Chef[6] zur Provisionierung genutzt werden. Dafür wird, mit Hilfe von sogenannten Rezepten, die Konfiguration einer Maschine beschrieben. Da es allerdings keine Werkzeuge, wie beispielsweise Editoren, gibt, wurde sich gegen den Einsatz von Chef entschieden.

3.2 Automatische Provisionierungen durch OpenTOSCA

Da die Universität Stuttgart OpenTOSCA in einem gefördeten Projekt implementierte, wurden verschiedene Arbeiten zu dieser Software geschrieben. Zu den studentischen Arbeiten zählen auch Provisionierungen, die mit Hilfe von OpenTOSCA erstellt wurden. Dieses Kapitel listet einige Arbeiten auf und beschreibt, was in diesen Arbeiten umgesetzt wurde. Die Arbeiten sind:

- TOSCA4Mashups – Provisionierung und Ausführung von Data Mashups in der Cloud [Gau16]
- Datenmanagement in der Cloud für den Bereich Simulationen und Wissenschaftliches Rechnen [RWWS14]
- Development of TOSCA ServiceTemplates for provisioning portable IT Services [Liu13]
- Integrating Cloud Service Deployment Automation with Software Defined Environments [Das14]

[3] http://getcloudify.org/
[4] http://ariatosca.org/
[5] https://puppet.com/
[6] https://www.chef.io/chef/

3.2.1 TOSCA4Mashups – Provisionierung und Ausführung von Data Mashups in der Cloud

In der Vergangenheit wurden IT–Experten herangezogen, wenn mehrere Datenquellen verbunden werden sollten. Dies ändert sich jedoch und es wird heutzutage vor allem von Domänenexperten durchgeführt, welche sich jedoch nicht unbedingt mit dem Erstellen eines solchen Systems auskennen. Aus diesem Grund wurde in dieser Arbeit ein Programm bereitgestellt, das es Domänenexperten erlaubt, verschiedene Datenquellen zu verknüpfen. Die Bereitstellung dient vor allem dafür, neue Instanzen zu kreieren. Dazu werden in einer Cloud–Umgebung neue Ressourcen erstellt, die der Instanz zugeordnet werden. Dem Domänenexperten wird schließlich die neue Instanz präsentiert.

3.2.2 Datenmanagement in der Cloud für den Bereich Simulationen und Wissenschaftliches Rechnen

Bestimmte Simulationen werden oft nur gelegentlich eingesetzt. Aus diesem Grund haben Unternehmen, die vereinzelt auf Simulationen zurückgreifen, keine Expertise in diesem Gebiet im Haus. Um das Problem zu beheben, haben Reimann et al. die Simulationssoftware *Pandas* bereitgestellt. Diese Provisionierung erlaubt es Domänenexperten Simulationsumgebungen aufzusetzen. Die Umgebungen werden in einer Cloud–Umgebung provisioniert, damit das Erstellen einer neuen Simulation vereinfacht wird, da Ressourcen dynamisch zugewiesen werden können. Dies erlaubt auch das Löschen der Instanz nach der Simulation.

3.2.3 Development of TOSCA ServiceTemplates for provisioning portable IT Services

Firmen nutzen oft *Enterprise Content Management*–Systeme um intern Dokumente zu verwalten. Das Provisionieren dieser Anwendungen ist ein anstrengender und langer Prozess. Um ihn zu erleichtern, wurde in dieser Arbeit das *SmartCloud Content Management*–System von IBM bereitgestellt. Dazu wurden Kernkomponenten durch OpenTOSCA provisioniert.

3.2.4 Integrating Cloud Service Deployment Automation with Software Defined Environments

Viele Cloud–Dienste benötigen eine Infrastruktur, die oft speziell für den Dienst erstellt wird. Zu dieser Infrastruktur gehören beispielsweise virtuelle Maschinen und IP–Adressen. Die Infrastruktur wird oft durch eine Virtualisierungsumgebung wie OpenStack verwaltet. Da es viele Virtualisierungsumgebungen gibt, ist es für OpenTOSCA schwierig alle zu bedienen. Aus

diesem Grund wurde in dieser Arbeit ein NodeType entwickelt, der es dem OpenTOSCA–Nutzer erlaubt virtuelle Maschinen in OpenStack zu erstellen.

4 Automatische, TOSCA-basierte Provisionierung des Situationserkennungssystems SitOPT

Um SitOPT durch OpenTOSCA bereitzustellen, müssen NodeTypes erstellt werden, die die einzelnen Komponenten des Systems provisionieren. Basierend auf diesen NodeTypes wird anschließend eine Topologie modelliert. Diese dient dann als Grundlage für eine Provisionierungsumgebung, wie z. B. OpenTOSCA. In diesem Kapitel werden sowohl die Topologie des Systems beschrieben, als auch die Implementierung der umgesetzten Teile erklärt.

4.1 Konzept

Die Gesamttopologie ist in Abbildung 4.1 dargestellt. Sie wird in den folgenden Kapiteln erklärt. Dabei werden zunächst deren Bestandteile beschrieben und in welcher Reihenfolge die Provisionierung durchgeführt wird. Hierfür wird auf die eingezeichneten Einzeltopologien jeweils getrennt eingegangen.

Eine Topologie ist ein Graph, der aus Knoten und Kanten besteht. Die Knoten sind NodeTypes, während Kanten RelationshipTypes sind. Es gibt mehrere RelationshipTypes:

- HostedOn
  Dieser RelationshipType sagt aus, dass ein NodeType erst bereitgestellt werden kann, sobald die Provisionierung des anderen abgeschlossen ist. So kann beispielsweise ein Webserver erst installiert werden, wenn die virtuelle Maschine komplett aufgesetzt ist.
- DependsOn
  DependsOn wird verwendet, wenn eine Komponente vor einer anderen installiert werden muss. So muss beispielsweise eine Laufzeitumgebung, wie Java, vor Programmen installiert werden, die diese verwenden.
- ConnectsTo
  Es wird ausgedrückt, dass zum Start alle Abhängigkeiten, die mit diesem RelationshipType verbunden sind, bereits gestartet sein müssen. Wenn zum Beispiel ein Programm eine Datenbank verwenden soll, muss diese zuvor gestartet werden, damit sofort eine Verbindung aufgebaut werden kann.

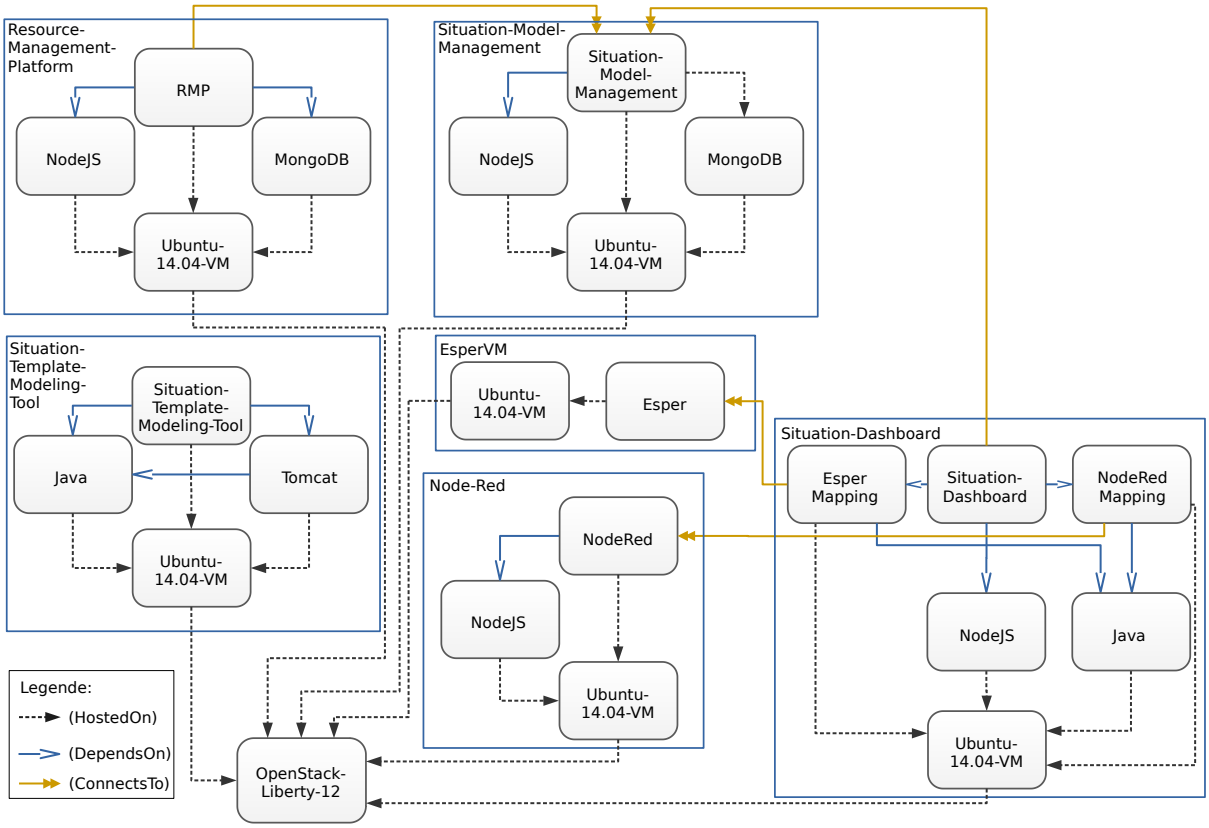

Abbildung 4.1: Gesamttopologie des SitOPT–Systems.

Der Knoten, der von allen Einzeltopologien verwendet wird, und somit keiner speziellen zugeordnet ist, ist vom Typ *OpenStack–Liberty–12*. Dieser NodeType erlaubt es eine Verbindung zu einer OpenStack–Instanz aufzubauen. Die Eigenschaften des Knoten legen fest, wie sich OpenTOSCA zu OpenStack verbinden soll. Damit lassen sich Ressourcen, die auf OpenStack angelegt sind, mit anderen NodeTypes verwalten. Zudem können neue Ressourcen angelegt werden, zum Beispiel kann die Verbindung aus diesem NodeType und dem NodeType *Ubuntu–14.04–VM* eine neue virtuelle Maschine auf OpenStack anlegen. Die Eigenschaften des NodeTypes *OpenStack–Liberty–12* sind:

- HypervisorEndpoint
  URL der OpenStack–Instanz, mit der sich OpenTOSCA verbinden soll.
- HypervisorTenantID
  Tenant ID oder Projektname, mit dem sich OpenTOSCA verbinden soll. Die Tenant ID ist die ID eines Projekts.
- HypervisorUserName
  Nutzername, den OpenTOSCA verwenden soll.

- HypervisorUserPassword
  Passwort für den entsprechenden Nutzer.

Eine komplette Beschreibung des NodeTypes befindet sich in [Das14]. Ebenso kann Node–Red durch den NodeType bereitgestellt werden, der von del Gaudio in [Gau16] beschrieben wurde.

4.1.1 Resource–Management–Platform

In diesem Abschnitt wird die Teiltopologie der Resource–Management–Platform erklärt. Diese ist in Abbildung 4.2 dargestellt und stellt die RMP bereit. Die RMP dient als zentraler Speicher von Sensorwerten und kann über eine REST–Schnittstelle ausgelesen werden.

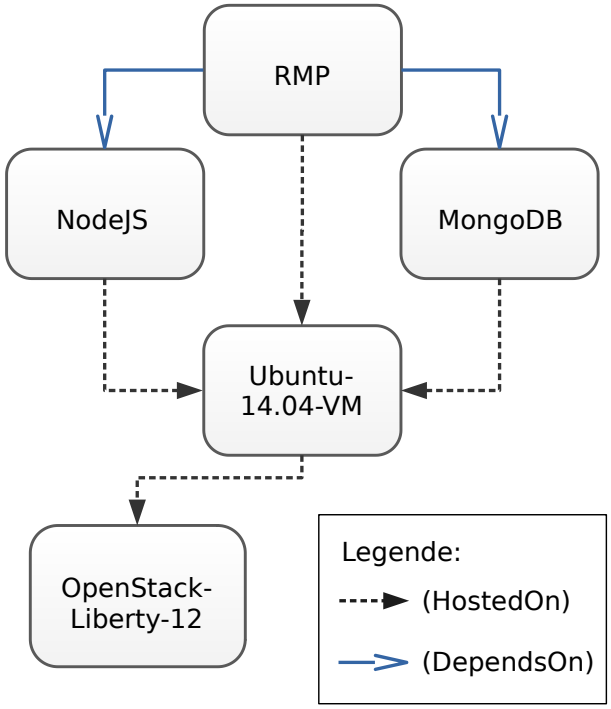

Abbildung 4.2: Teiltopologie der RMP

Um die RMP bereitzustellen, müssen Node.JS und MongoDB bereitgestellt sein. Bei MongoDB muss die Datenbank „RBS“ angelegt werden, da diese zum Speichern der Daten verwendet wird. Zusätzlich muss der Nutzer „RBS“ angelegt werden, der Lese– und Schreibrechte auf der gleichnamigen Datenbank hat. Dieser wird zum Lesen und Schreiben der Daten verwendet. Außerdem müssen die *Collections* „caches“ und „sensors“ zur Kategorisierung der gespeicherten Objekte angelegt werden. In der Collection „sensors“ werden Sensorenbeschreibungen gespeichert, in „caches“ werden die Sensorenwerte gespeichert.

Für die Provisionierung der RMP wird zunächst eine virtuelle Maschine mit dem NodeType *Ubuntu–14.04–VM* erstellt. Dieser NodeType legt eine virtuelle Maschine an, bevor er das Betriebssytem „Ubuntu“ mit der Version 14.04 installiert. Die Eigenschaften des NodeTypes beschreiben sowohl die virtuelle Maschine als auch Elemente des Betriebssystems. Der NodeType hat folgende Eigenschaften:

- VMType
  OpenStack speichert im VMType welche Hardwareeigenschaften eine virtuelle Maschine haben soll. Dazu gehören beispielsweise wie viel RAM und wie viele Prozessoren der Maschine zur Verfügung stehen sollen.
- VMUserName
  Der Accountname des Benutzers, über den alle Aktionen ausgeführt werden sollen.
- VMPassword
  Das Passwort des Benutzers, über den alle Aktionen ausgeführt werden sollen.
- VMPrivateKey
  Der private SSH–Schlüssel, über den sich OpenTOSCA mit der virtuellen Maschine verbinden soll. Dieser muss mit allen Zeilenümbrüchen komplett eingetragen werden.
- VMPublicKey
  Der öffentliche SSH–Schlüssel, der zu dem Privaten passt. Er wird nicht benötigt, wenn das Schlüsselpaar gespeichert ist und der Name angegeben wird.
- VMKeyPairName
  Der Name des gespeicherten SSH–Schlüsselpaares.

Auf der virtuellen Maschine werden drei Module installiert, welche jeweils von einem eigenem NodeType repräsentiert werden:

- NodeJS
  Dieser NodeType installiert Node.JS auf der virtuellen Maschine. Da Node.JS keine angepasste Konfiguration benötigt, hat dieser Knoten keine Eigenschaften. Zur Installation gibt es ein Paket, welches die Paketverwaltung *apt*[1] heruntergeladen und installiert werden kann. Dieser Ansatz könnte komplett als ImplementationArtifact realisiert wird. Alternativ kann das Paket als DeploymentArtifact zu dem NodeType hinzugefügt werden. Bei diesem Ansatz wird trotzdem ein ImplementationArtifact benötigt, das das Paket entpackt und installiert. Der NodeType wird in dieser Arbeit bereitgestellt und die Implementierung wird in Abschnitt 4.2 beschrieben. Sie nutzt den ersten Ansatz, da mit diesem immer die neueste Version installiert wird.

[1] https://wiki.debian.org/Apt

- MongoDB
  Der *MongoDB*–NodeType installiert MongoDB und startet den Dienst. Er trifft bereits Vorbereitungen für die Ressource–Management–Platform, indem der Datenbanknutzer und die Kollektionen angelegt werden. Analog zum *NodeJS* NodeType gibt es zwei Ansätze zum Installieren von MongoDB. Der Erste ist, das Paket mit dem NodeType als DeploymentArtifact bereitzustellen. Der Zweite installiert MongoDB aus einem Repository von *apt*. Auch hier wurde der erste Ansatz gewählt, um die Aktualität des Pakets zu gewährleisten. Die Implementierung dieses NodeTypes wird in Abschnitt 4.2 beschrieben.
- RMP
  Dieser NodeType klont den aktuellen Master–Zweig der Ressource–Management–Platform von GitHub. Der Quellcode könnte auch als DeploymentArtifact bereitgestellt werden, jedoch müsste das DeploymentArtifact nach jedem Update aktualisiert werden. Anschließend werden die Abhängigkeiten installiert und das Modul gestartet. Die Funktionsweise wird in Abschnitt 4.2 erklärt.

Provisionierung

Zunächst wird eine Verbindung zu einer OpenStack–Instanz und einem bestimmten Projekt aufgebaut. Sobald diese besteht, wird in dem Projekt auf OpenStack eine neue virtuelle Maschine erstellt. Dann versucht OpenTOSCA die restlichen NodeTypes bereitzustellen. Anschließend wird der Start für die einzelnen NodeTypes vorbereitet, beispielsweise wird Node.JS installiert. Danach werden die benötigten Dienste, das heißt zunächst MongoDB und anschließend die RMP, gestartet.

Da der NodeJS NodeType lediglich eine Abhängigkeit installiert, die später genutzt wird, muss dieser nicht gestartet werden. Sobald die Dienste laufen, ist die Ressource–Management–Platform einsatzbereit.

4.1.2 Situation–Model–Management

Die Topologie des Situation–Model–Managements wird in Abbildung 4.3 erklärt. Das Situation–Model–Management speichert Situationen, Sensoren, virtuelle Objekte und SituationTemplates. Außerdem speichert es zusätzliche Daten, wie Aktuatoren, Berechtigungen und gestartete Erkennungen von Situationen. Für jedes gespeicherte Element wird eine REST–Ressource angelegt. Das System dient zum Aktivieren der Aktuatoren, wenn sich der Zustand einer Situation von *erkannt* auf *nicht erkannt*, oder umgekehrt, ändert.

Als Grundlage dient eine virtuelle Maschine, die mit dem *Ubuntu–14.04–VM*–NodeType bereitgestellt wird, welcher in Unterabschnitt 4.1.1 erklärt wurde. Genauso werden die NodeTypes

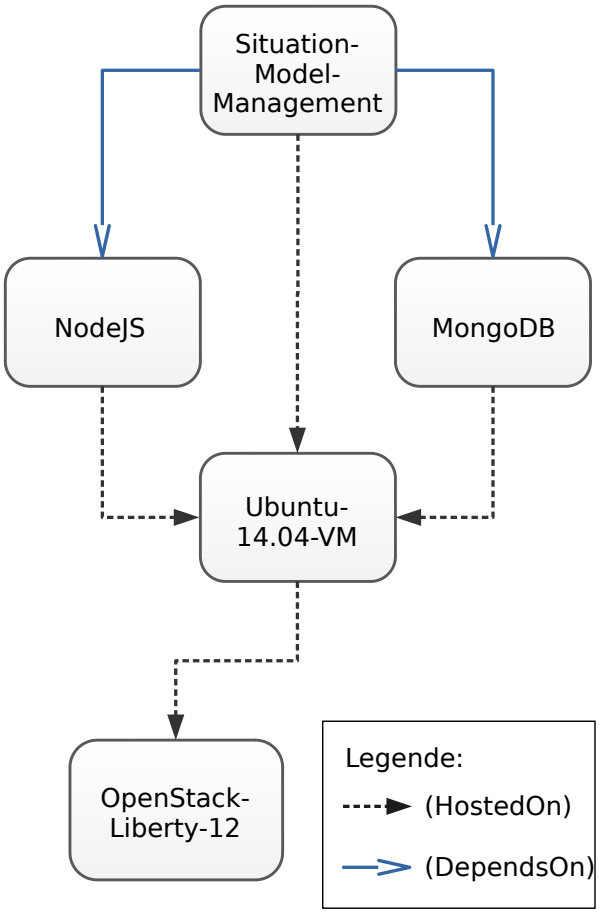

Abbildung 4.3: Teiltopologie des Situation–Model–Managements

NodeJS und MongoDB wiederverwendet, die ebenso bei der Bereitstellung der RMP genutzt wurden.

Der *Situation–Model–Management*–NodeType lädt den aktuellen Stand des Codes von GitHub herunter oder lädt ihn aus einem DeploymentArtifact nach. Anschließend werden die Abhängigkeiten mit Hilfe von *NodeJS* installiert. Schließlich wird der Dienst gestartet.

Provisionierung

Die Provisionierung findet analog zu Abschnitt 4.1.1 statt. Zunächst wird eine Verbindung mit OpenStack aufgebaut. Anschließend wird eine virtuelle Maschine erstellt. Auf dieser werden Node.JS und MongoDB über die NodeTypes *NodeJS* und *MongoDB* bereitgestellt. Zum Schluss wird das Situation–Model–Management gestartet.

4.1.3 Situation–Dashboard

Das Situation–Dashboard bietet eine Weboberfläche für das Situation–Model–Management. So bietet es die Möglichkeit Situationserkennungen zu starten und zu stoppen. Zudem wird eine graphische Oberfläche für die REST–Schnittstelle des Situation–Model–Managements angeboten. Das Topologiediagramm des Situation–Dashboards ist in Abbildung 4.4 dargestellt. Dazu werden auf einer virtuellen Maschine die beiden genutzten Programmiersprachen Node.JS und Java 8 bereitgestellt. Java 8 wird von den Modulen *NodeRedMapping* und *EsperMapping* benötigt, Node.JS wird vom Webdienst des Dashboards verwendet. Die Pakete für Node.JS und Java 8 können sowohl als DeploymentArtifacts an die entsprechenden NodeTypes angehängt werden, als auch durch *apt* heruntergeladen werden. Die Programme *NodeRedMapping* und *EsperMapping* können entweder als Quellcode oder als gepackte JAR–Datei ausgeliefert werden. Beide Ansätze können über DeploymentArtifacts realisiert werden. Der erste Ansatz kann zusätzlich mit ein ImplementationArtifact umgesetzt werden, da der Code von GitHub heruntergeladen werden kann.

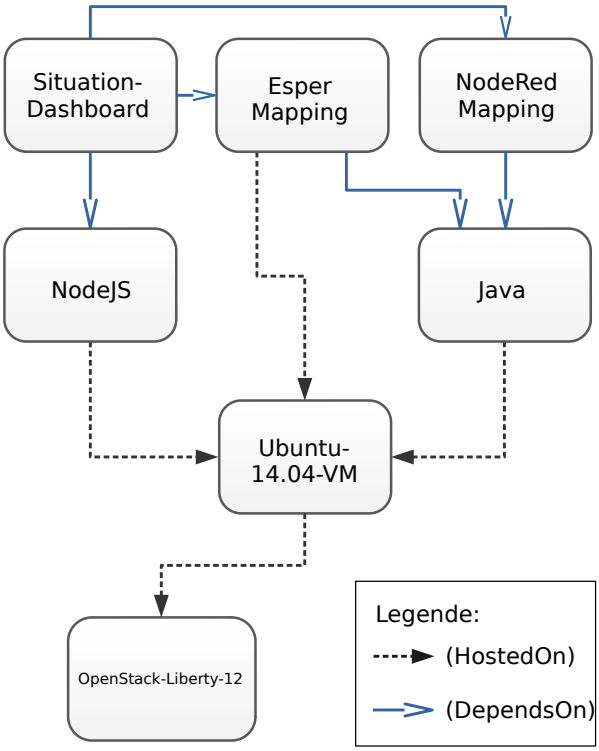

Abbildung 4.4: Einzeltopologie des Situation–Dashboards

Provisionierung

Nachdem eine Verbindung zu OpenStack aufgebaut und eine virtuelle Maschine eingerichtet wurde, werden Java und Node.JS installiert. Anschließend werden beide Mappings bereitgestellt. Nach der Installation des Dashboards, wird dieses gestartet.

4.1.4 Situation–Template–Modeling–Tool

Das Situation–Template–Modeling–Tool erlaubt das graphische Modellieren von SituationTemplates. Dabei können die Knotenarten, die von SitOPT unterstützt werden, per Drag–and–Drop dem Template hinzugefügt werden. Zudem können die Eigenschaften der Knoten gesetzt werden, die festlegen, wie eine Situation erkannt werden soll. Die Topologie des Situation–Template–Modeling–Tools wird in Abbildung 4.5 gezeigt.

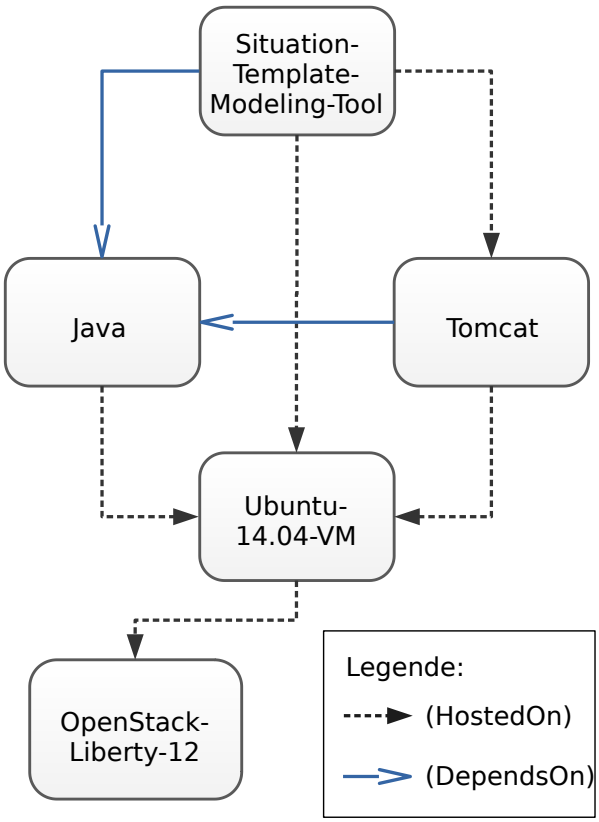

Abbildung 4.5: Subtopologie des Situation–Template–Modeling–Tools

Das Modeling–Tool benötigt eine Tomcat[2]–Instanz und Java 8. Tomcat und das Situation–Template–Modeling–Tool können ebenfalls komplett über ImplementationArtifacts als auch

[2] https://tomcat.apache.org/

über ImplementationArtifacts und DeploymentArtifacts realisiert werden. Dabei werden entweder die benötigten Dateien heruntergeladen oder von DeploymentArtifacts ausgeliefert. Die Installation der Programme findet immer über ImplementationArtifacts statt.

Provisionierung

Zunächst wird eine Verbindung zu OpenStack aufgebaut und eine virtuelle Maschine bereitgestellt. Anschließend werden Java und Tomcat bereitgestellt. Danach wird das Modeling–Tool provisioniert und gestartet.

4.2 Implementierung

In diesem Abschnitt wird erklärt, wie die durch diese Arbeit bereitgestellten NodeTypes implementiert sind.

MongoDB

Dieser NodeType enthält drei ArtifactTemplates, welche die Funktionen *install*, *configure* und *start* implementieren. Beim Installieren wird ein Personal Package Archive eingebunden, das das Paket *mongodb-org* bereitstellt. Um Probleme vorzubeugen, wird der zuvor Besitzer des Archives dem lokalen PGP–Schlüsselring[3] hinzugefügt. Durch das Einbinden des Archives kann das Paketverwaltungssystem *apt* MongoDB installieren. Nachdem die lokale Datenbank von *apt* auf den neuesten Stand gebracht wurde, wird das Paket *mongodb-org* installiert. Die genauen Schritte sind in Listing 4.1 zu sehen.

Listing 4.1 Installationsskript von MongoDB

```
sudo bash -c "echo '127.0.0.1' ${hostname} >> /etc/hosts"
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
    --recv EA312927
echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.0 multiverse
    sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org
```

Der *configure* Schritt legt die Kollektionen *caches* und *sensors* in der Datenbank *RBS* an. Diese werden zur Kategorisierung der verschiedenen gespeicherten Objekte genutzt. In *sensors* werden Sensoren gespeichert, die zum Erstellen von REST–Ressourcen benötigt. In *caches* werden

[3] http://www.pgpi.org/doc/pgpintro/

die Werte der Sensoren gespeichert. Diese werden einem gespeicherten Sensor zugeordnet. Anschließend wird der Datenbankbenutzer *RBS* mit dem Datenbankpasswort *RBS* angelegt. Der Datenbanknutzer hat der Lese– und Schreibrechte auf der gleichnamigen Datenbank hat. Diese werden für das Speichern und Lesen der Sensoren und ihrer Werte benötigt. Das Skript zur Konfiguration ist in Listing 4.2 dargestellt.

Listing 4.2 Konfigurationsskript von MongoDB

```
mongo RBS --host localhost --eval 'db.createCollection("caches")'
mongo RBS --host localhost --eval 'db.createCollection("sensors")'
mongo RBS --host localhost --eval "db.createUser({user: 'RBS',
    pwd: 'RBS', roles: ['readWrite'] })"
```

Das Startskript startet den Dienst, der durch das Paket „mongodb–org“ bereitgestellt wurde. Das Skript ist in Listing 4.3 zu sehen.

Listing 4.3 Startskript von MongoDB

```
sudo /etc/init.d/mongod start
```

NodeJS

Dieser NodeType beinhaltet lediglich ein ArtifactTemplate, welches die *install*–Funktion bereitstellt. Das Skript lädt zunächst ein anderes Skript herunter, das die Installation vorbereitet. Nach der Installationsvorbereitung, wird Node.JS installiert. Listing 4.4 zeigt diese Schritte.

Listing 4.4 Installationsskript von Node.JS

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Ressource–Management–Platform

Dieser NodeType beinhaltet zwei ArtifactTemplates, welche die Funktionen *install* und *start* bereitstellen. Zum Installieren wird Git[4] benötigt. Dazu wird es zunächst installiert. Anschließend wird das Gitrepository der RMP geklont. Danach werden die Abhängigkeiten mit Hilfe des Paketverwaltungssystem npm installiert. Bevor der Dienst gestartet werden kann, müssen die Konfigurationsdateien kopiert werden. Das Skript ist in Listing 4.5 zu sehen.

[4] https://git-scm.com/

Listing 4.5 Installationsskript der RMP

```
sudo apt-get install git
sudo sh -c "echo '127.0.0.1' $(hostname) >> /etc/hosts"
git clone https://github.com/SitOPT/Resource-Management-Platform.git
cd Resource-Management-Platform
npm install
cp config/database.config.js.example config/database.config.js
cp config/sitdb.config.js.example config/sitdb.config.js
```

Zum Starten der RMP ist das mitgelieferte Startskript auszuführen. Zur besseren Kontrolle wird die RMP im Hintergrund als *upstart–Dienst*[5] gestartet. Dazu wird der Dienst in die Datei „rmp.conf“ geschrieben, welche nach „/etc/init“ kopiert wird. Anschließend wird der Dienst über den Befehl „service“ gestartet. Die Implementierung der Schritte ist in Listing 4.6 dargestellt.

Listing 4.6 Startskript der RMP

```
cd Resource-Management-Platform
sudo chown ${SUDO_USER} -R .
{
    cat << EOF
    description "A test for starting the RMP"
    author "Armin Hüneburg"
    start on runlevel [2345]
    script
        cd $(pwd)
        npm start
    end script
    EOF
} > rmp.conf
sudo cp rmp.conf /etc/init/
sudo service rmp start
```

Tomcat

Der Tomcat–NodeType beinhaltet drei ArtifactTemplates. Diese implementieren die Operationen *install*, *configure* und *start*. Da Ubuntu 14.04 kein Paket für Tomcat8 bereitstellt, muss es explizit bereitgestellt werden. Das Skript, das dies übernimmt, ist in Listing 4.7 zu sehen. Zunächst werden die Gruppe und der Nutzer „tomcat8“ angelegt. Anschließend wird die Version

[5] http://upstart.ubuntu.com/

8.0.23 von Tomcat heruntergeladen. Danach wird das heruntergeladene Archiv entpackt und in das Installationsverzeichnis kopiert, dessen Berechtigungen angepasst werden.

Listing 4.7 Installationsskript für Tomcat8

```
sudo groupadd tomcat8
sudo useradd -s /bin/false -g tomcat8 -d /opt/tomcat8 tomcat8
wget http://mirror.sdunix.com/apache/tomcat/tomcat-8/v8.0.23/bin/
    apache-tomcat-8.0.23.tar.gz
sudo mkdir /opt/tomcat8
sudo tar xf apache-tomcat-8*.tar.gz -C /opt/tomcat8 --strip-components=1
cd /opt/tomcat8
sudo chgrp -R tomcat8 conf
sudo chmod g+rwx conf
chmod g+r conf/*
sudo chown -R tomcat8 work/ temp/ log/
```

Die Konfiguration von Tomcat ist in Listing 4.8 dargestellt. Es wird zunächst die Datei „tomcat8.conf“ angelegt, welche den upstart–Dienst von Tomcat beschreibt. Diese Datei wird anschließend nach „/etc/init“ kopiert.

Listing 4.8 Konfigurationsskript des Tomcat–NodeTypes

```
{
    cat << EOF
    description "Tomcat8 Server"
    start on runlevel [2345]
    stop on runlevel [!2345]
    respawn
    respawn limit 10 5
    setuid tomcat8
    setgid tomcat8
    env JAVA_HOME=/opt/java8
    env CATALINA_HOME=/opt/tomcat8
    env JAVA_OPTS="-Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom"
    env CATALINA_OPTS="-Xms512M -Xmx1024M -server -XX:+UseParallelGC"
    exec $CATALINA_HOME/bin/catalina.sh run
    post-stop script
        rm -rf $CATALINA_HOME/temp/*
    end script
    EOF
} > tomcat8.conf
cp tomcat8.conf /etc/init
```

Zum Starten von Tomcat muss lediglich der upstart–Dienst gestartet werden. Listing 4.9 zeigt den Befehl.

Listing 4.9 Startskript des Tomcat–NodeTypes

```
sudo service tomcat8 start
```

Java

Der Java–NodeType installiert Java 8, welches standardmäßig nicht im Ubuntu 14.04 Repository ist. Aus diesem Grund wird ein *Personal Package Archive* eingebunden, das ein Installationspaket für Java 8 beinhaltet. Um dieses nutzen zu können muss auch die Oracle–License akzeptiert werden. Die Installationsschritte werden in Listing 4.10 gezeigt.

Listing 4.10 Installationsskript zu Java

```
sudo apt-get install -y python-software-properties debconf-utils
add-apt-repository -y ppa:webupd8team/java
sudo apt-get update
echo „oracle-java8-installer shared/accepted-oracle-license-v1.1 select true“ |
    sudo debconf-set-selections
sudo apt-get install -y oracle-java8-installer
```

Situation–Template–Modeling–Tool

Der NodeType *Situation–Template–Modeling–Tool* stellt die ArtifactTemplates für *install* und *start* zur Verfügung. Für die Installation werden zunächst Git und Ant[6] benötigt. Nach deren Installation, werden die Repositories geklont, die den Code für das Modeling–Tool und das global genutzte XML–Schema beinhalten. Anschließend werden aus dem Schema die Java–Klassen generiert. Danach wird der Quellcode des Tools kompiliert. Diese Schritte sind in Listing 4.11 beschrieben.

Zum Starten des Tools muss lediglich die kompilierte *SitTempModelingTool.war* nach „/opt/-tomcat8/webapps“ kopiert und der Tomcat–Dienst neu gestartet werden. Das Skript, das diese Schritte ausführt, ist in Listing 4.12 zu sehen.

[6] https://ant.apache.org/

Listing 4.11 Installationsskript für Situation–Template–Modeling–Tool

```
sudo apt-get install -y git ant
git clone https://github.com/SitOPT/Situation-Template-Modeling-Tool.git
git clone gttps://github.com/SitOPT/Situation-Template-Schema.git
xjc -d Situation-Template-Modeling-Tool/src
    -p mapping Situation-Template-Schema/situation_template.xsd
cd Situation-Template-Modeling-Tool
ant
```

Listing 4.12 Startskript für Situation–Template–Modeling–Tool

```
cd Situation-Template-Modeling-Tool
cp SitTempModelingTool.war /opt/tomcat8/webapps
sudo service tomcat8 restart
```

4.2.1 Evaluation

Die Verwendung von OpenTOSCA hat sich als vorteilhaft erwiesen um TOSCA–Topologien bereitzustellen. Dies liegt vor allem an der Integration der Winery, da diese das Modellieren von Topologien stark vereinfacht. Zudem können Topologien aus der Winery in den OpenTOSCA Container übernommen werden. Dieser generiert aus den Topologien Provisionierungspläne, die aus dem Container gestartet werden können.

Die erstellten NodeTypes sind nutzbar, jedoch bieten sie nicht alle Konfigurationsmöglichkeiten an, die möglich wären. So können beispielsweise nicht der Name des Datenbankbenutzers oder der Datenbank in der RMP konfiguriert werden. Allerdings behindert dies nicht die Nutzung der NodeTypes.

5 Zusammenfassung

In dieser Arbeit wurden Konzepte erstellt, die die automatische Provisionierung von SitOPT auf der Cloud–Plattform OpenStack erlaubt. Hierfür wurde der TOSCA–Standard genutzt. Es wurde OpenTOSCA der Universität Stuttgart verwendet, welches diesen Standard umsetzt. OpenTOSCA wurde hauptsächlich gewählt, da es eine große Unterstützung von Werkzeugen genießt. Dazu zählen auch das Modellierungswerkzeug Winery, welches das graphische Editieren von TOSCA–Topologien erlaubt. Zudem gestattet OpenTOSCA auch die Ausführung von Topologien, die in der Winery erstellt wurden. Außerdem wurden zwei Teile von SitOPT, die RMP und das Situation–Template–Modeling–Tool, durch OpenTOSCA provisioniert.

Die RMP ist ein Zwischenspeicher, der in SitOPT genutzt werden kann. Jedoch kann sie auch allein genutzt werden. In der RMP werden Sensoren angelegt, für die REST–Ressourcen angelegt werden. Für Sensoren können Sensordaten gespeichert werden, die ebenfalls über REST–Ressourcen bereitgestellt werden.

Das Situation–Template–Modeling–Tool dient zur Erstellung von SitOPT–SituationTemplates. Dafür gibt es einen graphischen Editor, der es erlaubt, SituationTemplates als Graphen mit Hilfe von verschienen Knotentypen zu bearbeiten. Die entstandenen Knoten haben Eigenschaften, die ebenfalls mit dem Editor mit Werten versehen werden können.

5.1 Zukünftige Aufgaben

In der Zukunft müssen die restlichen Module von SitOPT bereitgestellt werden. Dazu gehören das Situation–Dashboard, Esper und das Situation–Model–Management. Um dies zu erreichen müssen die einzelnen NodeTypes erstellt werden. Um eine problemlose Provisionierung zu erlauben muss auch noch festgestellt werden, welche Properties die einzelnen NodeTypes benötigen. Die RMP und das Situation–Template–Modeling–Tool haben Eigenschaften, die bisher mit Standardwerten sind, die allerdings deploymentspezifisch sein müssen. In zukünftigen Arbeiten können die Eigenschaften der bisherigen NodeTypes überarbeitet werden.

Abkürzungsverzeichnis

Abkürzung	Bedeutung	Erstes Vorkommen
CEP	Complex Event Processing	23
CSAR	Cloud Service Archive	15
IoT	Internet of Things	3
OASIS	Advancing Open Standards for the Information Society	13
RMP	Resource Management Platform	7
SitOPT	Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten	3
TOSCA	Topology and Orchestration Specification for Cloud Applications	3

Literaturverzeichnis

[AG10] N. Antonopoulos, L. Gillam. *Cloud computing: Principles, systems and applications*. Springer Science & Business Media, 2010 (zitiert auf S. 11).

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. „OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications“. In: *Service-Oriented Computing: 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*. Hrsg. von S. Basu, C. Pautasso, L. Zhang, X. Fu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 692–695. ISBN: 978-3-642-45005-1. DOI: 10.1007/978-3-642-45005-1_62. URL: http://dx.doi.org/10.1007/978-3-642-45005-1%7B%5C_%7D62 (zitiert auf S. 15, 18).

[BBK+14] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, J. Wettinger. „Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA“. In: *Proceedings of the IEEE International Conference on Cloud Engineering (IEEE IC2E 2014)*. IEEE Computer Society, März 2014, S. 87–96. DOI: DOI10.1109/IC2E.2014.56 (zitiert auf S. 14, 15).

[Das14] D. Das. „Integrating Cloud Service Deployment Automation With Software Defined Environments“. Magisterarb. Universität Stuttgart, 2014 (zitiert auf S. 29, 30, 35).

[Gau16] D. D. Gaudio. „TOSCA4Mashups - Provisionierung und Ausführung von Data Mashups in der Cloud“. bathesis. Universität Stuttgart, 2. Mai 2016 (zitiert auf S. 30, 35).

[HSW+16] P. Hirmer, A. C. F. da Silva, M. Wieland, U. Breitenbücher, K. Képes, B. Mitschang. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. In: *Complex Systems Informatics and Modeling Quarterly* (2016) (zitiert auf S. 20).

[HWBM16a] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Automated Sensor Registration, Binding and Sensor Data Provisioning“. Englisch. In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016)*. Bd. 1612. CEUR Workshop Proceedings. Ljubljana, Slovenia: CEUR-WS.org, Juni 2016, S. 81–88. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-22&engl=0 (zitiert auf S. 20).

[HWBM16b] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Dynamic Ontology-based Sensor Binding“. Englisch. In: *Advances in Databases and Information Systems. 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, 2016, Proceedings*. Bd. 9809. Information Systems and Applications, incl. Internet/Web, and HCI. Prague, Czech Republic: Springer International Publishing, Aug. 2016, S. 323–337. ISBN: 978-3-319-44038-5. DOI: 10.1007/978-3-319-44039-2. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2016-25&engl=0 (zitiert auf S. 20).

[HWS+15] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, F. Leymann. „SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates“. Englisch. In: *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*. Hrsg. von J. Barzen, R. Khalaf, F. Leymann, B. Mitschang. Bd. RC25564. Technical Paper. IBM Research Report, Dez. 2015, S. 113–127. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-34&engl= (zitiert auf S. 19, 20).

[HWS+16] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, F. Leymann. „Situation recognition and handling based on executing situation templates and situation-aware workflows“. Englisch. In: *Computing* (Okt. 2016), S. 1–19. DOI: 10.1007/s00607-016-0522-9. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2016-12&engl=0 (zitiert auf S. 19).

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. „Winery – Modeling Tool for TOSCA-based Cloud Applications“. In: *11th International Conference on Service-Oriented Computing*. LNCS. Springer, 2013 (zitiert auf S. 14, 15).

[Kep13] K. Kepes. *Konzept und Implementierung eine Java-Komponente zur Generierung von WS-BPEL 2.0 BuildPlans für OpenTOSCA*. Deutsch. Bachelorarbeit: Universität Stuttgart, Institut für Architektur von Anwendungssystemen. Bachelorarbeit. Juli 2013. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=BCLR-0043&engl=0 (zitiert auf S. 16, 18).

[LBK15] J. Lee, B. Bagheri, H.-A. Kao. „A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems“. In: *Manufacturing Letters* 3 (Jan. 2015), S. 18–23. DOI: 10.1016/j.mfglet.2014.12.001. URL: http://dx.doi.org/10.1016/j.mfglet.2014.12.001 (zitiert auf S. 11).

[LCW08] D. Lucke, C. Constantinescu, E. Westkämper. „Smart Factory - A Step towards the Next Generation of Manufacturing“. In: *Manufacturing Systems and Technologies for the New Frontier: The 41st CIRP Conference on Manufacturing Systems May 26–28, 2008, Tokyo, Japan*. Hrsg. von M. Mitsuishi, K. Ueda, F. Kimura. London: Springer London, 2008, S. 115–118. ISBN: 978-1-84800-267-8. DOI: 10.1007/978-1-84800-267-8_23. URL: http://dx.doi.org/10.1007/978-1-84800-267-8_23 (zitiert auf S. 11).

[Liu13] K. Liu. „Development of TOSCA Service Templates for provisioning portable IT Services“. Universität Stuttgart, 2013 (zitiert auf S. 30).

[MCRG15] E. Markoska, I. Chorbev, S. Ristov, M. Gusev. „Cloud portability standardization overview“. In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*. Mai 2015, S. 286–291. DOI: 10.1109/MIPRO.2015.7160281 (zitiert auf S. 29).

[MG11] P. Mell, T. Grance. „The NIST definition of cloud computing“. In: (2011) (zitiert auf S. 11).

[OAS13] OASIS. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. https://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf. 2013 (zitiert auf S. 13).

[OAS16] OASIS. *TOSCA Simple Profile in YAML Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2016. URL: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html (zitiert auf S. 14).

[RWWS14] P. Reimann, T. Waizenegger, M. Wieland, H. Schwarz. „Datenmanagement in der Cloud fur den Bereich Simulationen und Wissenschaftliches Rechnen“. In: *44. Jahrestagung der Gesellschaft für Informatik INFORMATIK 2014*. 2014 (zitiert auf S. 30).

[SHWM16] A. C. F. da Silva, P. Hirmer, M. Wieland, B. Mitschang. „SitRS XT – Towards Near Real Time Situation Recognition“. Englisch. In: *Journal of Information and Data Management* 7.1 (Apr. 2016), S. 4–17. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2016-14&engl=0 (zitiert auf S. 19).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. „Towards Situation-Aware Adaptive Workflows“. Englisch. In: *Proceedings of the 13th Annual IEEE Intl. Conference on Pervasive Computing and Communications Workshops: 11th Workshop on Context and Activity Modeling and Recognition*. St. Louis, Missouri, USA: IEEE, März 2015, S. 32–37. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-24&engl= (zitiert auf S. 19).

Alle URLs wurden zuletzt am 5. Dezember 2016 geprüft.

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