Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Situationsabhängige Modellierung für die OpenTOSCA Benutzeroberfläche

Lavinia Stiliadou

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	M. Sc. Kálmán Képes
Beginn am:	19. Dezember 2019
Beendet am:	14. August 2020

Kurzfassung

Das Resultat der zunehmenden Vernetzung von Geräten mit dem Internet ist die steigende Präsenz von Smart Home Systemen in Privathaushalten. Diese erfordern eine ständige Wartung von Software- und Hardwarekomponenten. Jedoch gestaltet sich die Wartung aufgrund der Heterogenität und der Anzahl an Komponenten als unzureichend. Mit dem Paradigma des Cloud Computings kann diese automatisiert werden. Bei der Softwarekonfiguration gerät man vor die Problematik, dass viele Aspekte der Umgebung betrachtet werden müssen, sodass die Kombination und deren Interaktion miteinander zu einer komplexen Aufgabe wird. Das Betrachten von Situationen als konkrete Abstrahierungen des Kontexts erlaubt die Interpretation einer Situation als Ereignis. Jedes Ereignis hat zusätzlich das Attribut eines Status, der entweder aktiv oder inaktiv ist und zu Zustandsänderungen des Systems führen kann. Open-Source Projekte wie OpenTOSCA erlauben bereits das vollautomatisierte Deployment von Anwendungen und das Management von Situationen. Jedoch ist diese für den Benutzer nicht in der Benutzeroberfläche greifbar.

Diese Arbeit stellt deshalb ein Konzept vor mit dem die situationsabhängige Modellierung und Steuerung von Anwendungsinstanzen einer CSAR möglich ist. Dazu wird prototypisch das entwickelte Konzept sowohl in der OpenTOSCA Benutzeroberfläche als auch im Container implementiert.

Inhaltsverzeichnis

1 Einleitung **13**
- 1.1 Motivierendes Szenario . . . 14
- 1.2 Aufbau der Arbeit . . . 15

2 Grundlagen **17**
- 2.1 Internet of Things . . . 17
- 2.2 Cloud Computing . . . 18
- 2.3 TOSCA . . . 19
- 2.4 OpenTOSCA . . . 20
- 2.5 Docker . . . 23

3 Verwandte Arbeiten **25**
- 3.1 Kontextbewusste Systeme . . . 25
- 3.2 Kontextbewusste Benutzeroberflächen . . . 26
- 3.3 Kontextualisierung von Geschäftsprozessen . . . 27
- 3.4 Kontextbewusstes Management von Cloud Anwendungen . . . 28
- 3.5 SitOPT . . . 28
- 3.6 Situationsbewusste Workflows . . . 29
- 3.7 Situationsbewusstes Management von cyberphysischen Systemen . . . 30

4 Problemstellung **33**
- 4.1 Situationsbewusste Modellierung in OpenTOSCA . . . 34
- 4.2 Instanzerstellung in der Benutzeroberfläche . . . 36

5 Konzept zur situationsbezogenen Modellierung und Steuerung **37**
- 5.1 Neue OpenTOSCA UI Architektur . . . 37
- 5.2 Situationsabhängige Ausführung von Anwendungsinstanzen . . . 38
- 5.3 Aggregation von Situationen . . . 39

6 Implementierung **41**
- 6.1 Situationen . . . 41
- 6.2 Situationstrigger . . . 41
- 6.3 Situationsbezogene Ausführung von Plänen . . . 43
- 6.4 Aggregierte Situationen . . . 43

7 Zusammenfassung und Ausblick **47**

Literaturverzeichnis **49**

Abbildungsverzeichnis

1.1 Motivierendes Szenario 14

2.1 IoT Referenzarchitektur (nach [GBF+18]) 18
2.2 Struktur eines Service Templates (nach [OA13]) 20
2.3 OpenTOSCA Ökosystem (nach [BEK+16]) 21
2.4 Systemarchitektur der Vinothek (nach [BBK14]) 22
2.5 Vergleich zwischen Containern und Virtuellen Maschinen (nach [Doc]) 23

3.1 Ablauf eines kontextbewussten Systems (nach [DDF+06; KKR+13]) 26
3.2 Architektur des SitOPT Systems (nach [WSB15]) 29
3.3 Beispiel einer Anwendung von CPS (nach [KBL19]) 30

4.1 BPEL Plan für das Skalieren von Anwendungsinstanzen 33
4.2 Instanzerstellung in der OpenTOSCA UI 36

5.1 Neue OpenTOSCA UI Architektur 37
5.2 Konzept: Situationsabhängige Steuerung von Anwendungsinstanzen 39
5.3 Konzept: Aggregierte Situationen 40

6.1 Die neue Benutzeroberfläche des OpenTOSCA Ökosystems 45

Verzeichnis der Listings

4.1 Aufbau einer Situation 34
4.2 Aufbau eines Situationstriggers 35

6.1 Aufbau des neuen Situationstriggers 42
6.2 Aufbau einer aggregierten Situation 44

Abkürzungsverzeichnis

API Application Programming Interface.

BPEL Business Process Execution Language.

BPMN Business Process Model and Notation.

CEA Context-driven Emergency Application.

CPS Cyber-Physical Systems.

CSAR Cloud Service Archive.

DMDM Declarative Management Description Model.

HTML Hypertext Markup Language.

IA Implementation Artifact.

IDC International Data Corporation.

IMDM Imperative Management Description Model.

IoT Internet of Things.

NIST National Institute of Standards and Technology.

OASIS Organization for the Advancement of Structured Information Standards.

REST Representational State Transfer.

TOSCA Topology and Orchestration Specification for Cloud Applications.

UI User Interface.

VM Virtual Machine.

XML Extensible Markup Language.

1 Einleitung

Die zunehmende Vernetzung von Geräten mit dem Internet ist das Ergebnis der Entwicklungen im Bereich der Internet der Dinge und des Cloud Computings. So setzen bereits nach den Daten des Bitcom Cloud Monitors 2020 [BRK] drei Viertel der Unternehmen in Deutschland Cloud Computing ein und jeder dritte Deutsche nutzt Smart Home Sprachassistenten wie Alexa [DPA19]. Jedoch bewirken diese Entwicklungen eine Erhöhung der Anzahl verbundener und heterogener Systeme. Daraus entstehen für die Anwendungen neue Herausforderungen an die Provisionierung, Konfiguration und Verwaltung der Software- und Hardwarekomponenten dieser Systeme. Zudem gestaltet sich die manuelle Verwaltung dieser Anwendungen als ineffizient, zeitaufwändig und unzureichend [LVCD13]. Daher werden vollautomatisierte Deployment- und Managementsysteme, aus dem Bereich des Cloud-Computings, notwendig.

Jedoch muss bei der Softwarekonfiguration die Umgebung, also der Kontext, miteinbezogen werden. Dabei kann der Kontext beispielsweise durch Sensoren erfasst werden, die eine große Menge an Daten erzeugen. Dadurch entsteht die Problematik viele Kombinationen von Informationen zu betrachten, wodurch der Komplexitätsgrad der Systeme wächst. Aus diesem Grund wird die Disziplin des situationsbewussten Deployments und Managements notwendig, das heißt die Konfiguration der Anwendungen wird an die jeweiligen Situationen innerhalb der Anwendungsumgebung angepasst, wobei die Eigenschaften des Systems wie Sicherheit erhalten bleiben. Dafür wird der Kontext der Anwendung abstrahiert und die daraus entstehenden Situationen erfasst. Dazu werden Situation Templates verwendet, die eine Kombination von Kontextinformationen enthalten und basierend auf diesen Werten eine Situation ableiten [SHWM16; WSB15]. Ein System, welches die Situationserkennung und die situationsabhängige Ausführung bereits unterstützt, ist OpenTOSCA[1]. Dabei ist OpenTOSCA ein Open-Source Projekt der Universität Stuttgart. Jedoch fehlt derzeit noch die graphische Darstellung sowie Verwaltung von (aggregierten) Situationen und die situationsbezogene Steuerung von Anwendungsinstanzen einer CSAR in der Benutzeroberfläche.

Das Ziel dieser Bachelorarbeit ist es, eine Modellierungsoberfläche für Situationen innerhalb der Benutzeroberfläche des OpenTOSCA Ökosystems zu konzipieren und zu entwickeln. Darauf aufbauend erfolgt eine prototypische Implementierung der entwickelten Konzepte, unter anderem der Verwaltung von Situationen und Situationstriggern sowie von aggregierten Situationen, in der Benutzeroberfläche und dem Container des OpenTOSCA Ökosystems.

[1] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

1.1 Motivierendes Szenario

Die detaillierte Abbildung der Welt auf Kontextzustände ist eine komplexe Aufgabe [GBH05; LCG+09], da viele Entitäten und deren Wechselwirken zueinander betrachet werden müssen. Aus diesem Grund werden Situationen eingeführt, welche als vereinfachtes Modell der Realität dienen und die die Problematik der kontextabhängigen Modellierung lösen. Der folgende Anwendungsfall, der in Abbildung 1.1 zu sehen ist, demonstriert die Relevanz dieser Disziplin. Dabei wird eine Anwendung unter der Woche lokal beim Unternehmen im eigenen Datenzentrum ausgeführt und den Kunden als Software-as-a-Service angeboten. Aufgrund der ständig schwankenden Anzahl an Kunden werden Public Cloud Lösungen wie Amazon AWS notwendig. Jedoch möchte das Unternehmen die Verwaltung der Anwendung sowie die der Kundendaten möglichst firmenintern durchführen und nur in Situationen wie etwa, rapider Anstieg von Nutzerzahlen, Wochenend- und Feiertagen die Anwendung in die Public Cloud auslagern und sonst im internen Datenzentrum administrieren. Dabei ist die optimale Ausführung von Scale-Out Plänen von Bedeutung. Wird der Scale-Out Plan zu früh ausgeführt, muss das Unternehmen bereits die hinzugefügten Ressourcen bezahlen, obwohl die Auslastung zu diesem Zeitpunkt nicht hoch war. Hingegen können bei einer zu späten Ausführung des Scale-Out Plans nicht alle Kundenanfragen beantwortet werden beziehungsweise dauert die Bearbeitung der Anfragen zu lange, wodurch die Kundenzufriedenheit und die Verfügbarkeit des Rechenzentrums sinkt. Systeme wie OpenTOSCA stellen die Situationserkennung sowie die situationsbedingte Ausführung des Scale-Out Plans bereit. Jedoch möchte das Unternehmen die Verwaltung sowie die Ausführung dieser Pläne für den Nutzer transparent gestalten. Dies soll durch eine Benutzeroberfläche erfolgen.

In Abbildung 1.1 ist das Szenario zu sehen, welches zur Umsetzung der situationsabhängigen Modellierung und Steuerung innerhalb der Benutzeroberfläche verwendet wird. Dazu soll eine Anwendungsinstanz der CSAR *MyTinyToDo_Bare_Docker* skaliert werden, wenn die Auslastung zu hoch ist oder die Anzahl aktiver Kunden über 500 beträgt. Die Ausführung des Scale-Out Plans soll in der Benutzeroberfläche des OpenTOSCA Systems möglich werden.

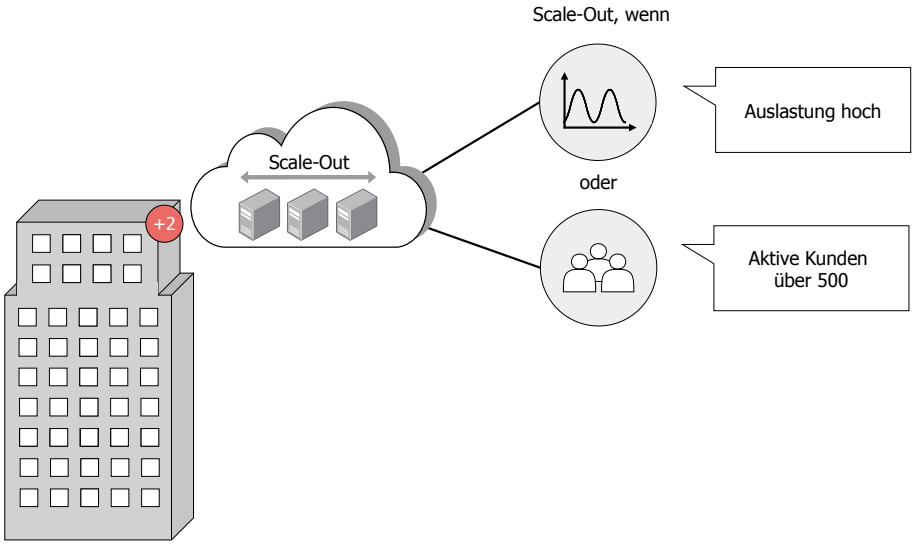

Abbildung 1.1: Skalieren von Anwendungsinstanzen, sobald die Auslastung hoch ist oder die aktive Kundenanzahl über 500 liegt.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt gegliedert:

Kapitel 2 - Grundlagen:

In diesem Kapitel sind die grundlegenden Begriffe enthalten, welche für das Verständnis der Thematik notwendig sind. Dabei werden unter anderem die Begriffe des Cloud Computings, der Internet der Dinge und OpenTOSCA vorgestellt.

Kapitel 3 - Verwandte Arbeiten:

Dieses Kapitel beschäftigt sich mit verwandten Arbeiten aus dem Bereich der kontext- sowie situationsbezogenen Adaption, die im Zusammenhang mit dieser Bachelorarbeit stehen.

Kapitel 4 - Problemstellung

Hier wird die Problemstellung, die zu dieser Arbeit geführt hat, beschrieben.

Kapitel 5 - Konzept zur situationsbezogenen Modellierung und Steuerung:

Dieses Kapitel stellt die entwickelten Konzepte zum Management von Situationen, Situationstriggern und aggregierten Situationen vor.

Kapitel 6 - Implementierung:

Das Kapitel befasst sich mit der prototypischen Implementierung der in Kapitel 5 vorgestellten Konzepte. Als Grundlage der Implementierung dienen die Benutzeroberfläche und der Container des OpenTOSCA Ökosystems.

Kapitel 7 - Zusammenfassung und Ausblick:

Das letzte Kapitel dient als Zusammenfassung der wichtigsten Aspekte dieser Arbeit und präsentiert weitere Forschungsthemen in dieser Richtung.

2 Grundlagen

Dieses Kapitel widmet sich den grundlegenden Terminologien sowie Ansätzen, die für das Verständnis dieser Arbeit relevant sind. Im ersten Abschnitt wird der Begriff der Internet der Dinge vorgestellt. Anschließend werden die Charakteristiken des Cloud Computings beschrieben. Darüber hinaus wird der TOSCA Standard vorgestellt, welcher die Modellierung von Cloud Anwendungen ermöglicht. Im Anschluss wird das OpenTOSCA Ökosystem der Universität Stuttgart vorgestellt, dass um die prototypische Implementierung der in dieser Arbeit entwickelten Konzepte erweitert wird. Im letzten Kapitel wird die Open-Source Software Docker vorgestellt, welche zum Starten von OpenTOSCA verwendet wird.

2.1 Internet of Things

Der Erfolg von IoT-Applikationen ist bereits daran erkennbar, dass fast jeder dritte Deutsche smarte Sprachassistenten wie Alexa nutzt [DPA19]. Dieses wachsende Interesse lässt sich auf die steigende Vernetzung von Geräten mit dem Internet zurückführen, geschätzt werden nach der International Data Corporation (IDC) Studie „Internet of Things in Deutschland 2016“ über 30 Milliarden vernetzte Geräte bis zum Jahr 2020 [IDC16]. Dabei wird die Verbindung der physischen Welt mit der virtuellen Welt durch den Begriff der IoT beschrieben. Die Grundlage bieten sogenannte Smart Devices, welche mit Hilfe von Sensoren und Aktuatoren den Zustand der Umgebung erfassen und sich an diese anpassen [FBH+17]. Ein konkreter Kontext, indem man von IoT spricht, sind Smart Homes. Eine beispielhafte Anwendung wäre die Regulation der Raumtemperatur, wo Medikamente gelagert werden. Dabei wird die Raumtemperatur durch Sensoren gemessen und sollte eine Temperaturschranke überschritten werden, wird die zuständige Person benachrichtigt, falls diese jedoch nicht erreichbar ist, wird die Klimaanlage automatisch eingeschaltet.

Eine IoT Referenzarchitektur ist in Abbildung 2.1 zu sehen. Der Inhalt des folgenden Abschnitts beruht im Wesentlichen auf der Beschreibung der Komponenten durch Guth et al. [GBF+18]. Sensoren und Aktuatoren sind Hardwarekomponenten, wobei Sensoren Daten aus der physischen Welt sammeln und Aktuatoren die physische Welt manipulieren. Zudem sind Sensoren sowie Aktuatoren mit Devices verbunden, die eine Verbindung zur IoT Integration Middleware ermöglichen. In IoT Umgebungen sind heterogene Sensoren sowie Aktuatoren enthalten, wodurch der Zugriff auf diese Geräte erschwert wird. Um diese Problematik zu lösen, werden die Driver der Devices verwendet. Ein Driver ist eine Software, die als einheitliche Schnittstelle für Sensoren und Aktuatoren dient. Gateways werden von Devices verwendet, wenn eine direkte Kommunikation zur IoT Integration Middleware nicht möglich ist. Hierbei übernimmt die IoT Integration Middleware die Verarbeitung der Daten aus den Devices sowie die Bereitstellung der Daten zu den verbundenen Anwendungen. Die Anwendungskomponente kann mittels der IoT Integration Middleware Daten aus den Devices empfangen sowie manipulieren.

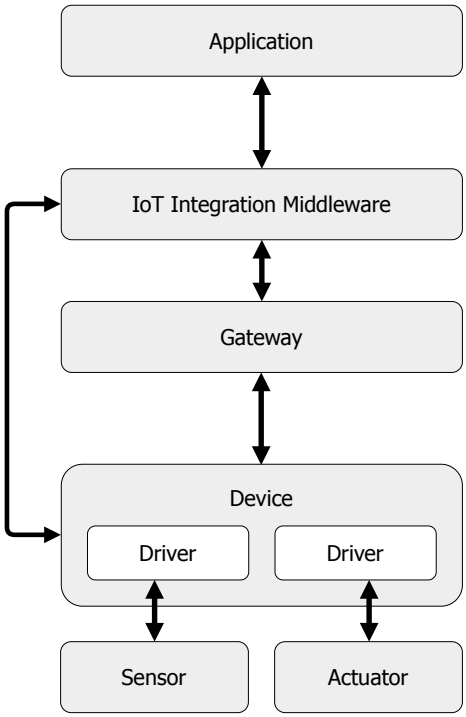

Abbildung 2.1: IoT Referenzarchitektur (nach [GBF+18])

2.2 Cloud Computing

Cloud Computing gilt als Innovation in der Verwaltung von IT-Ressourcen. Immer mehr Anwendungen nutzen das Konzept der Cloud, wie beispielsweise Google Drive[1] oder Apple iCloud[2]. Benutzer können dadurch zusätzlichen Speicher erhalten, in dem sie Daten ablegen können. Folglich erhält der Nutzer auf Nachfrage IT-Ressourcen und kann diese solange nutzen, solange sie benötigt werden [Ley11]. Dabei wird lediglich die tatsächliche Nutzung dieser Ressourcen bezahlt [Ley11]. Die Anbieter können diese Ressourcen, wie andere Ressourcen beispielsweise Strom, anbieten und managen [Ley11]. Dafür können virtuelle Maschinen (VM) beziehungsweise virtualisierte Anwendungen verwendet werden. Jedoch erfüllt eine virtualisierte Anwendung nicht zwingend die Eigenschaften einer Cloud-Anwendung [LFWW16]. Diese sind nach Fehling et al. [FLR+14] *Isolated State*, *Distribution*, *Elasticity*, *Automated Management* und *Loose Coupling*. Ein Beispiel bei dem die Elastizität einer virtualisierten Anwendung, die an eine einzelne VM geknüpft ist, verletzt wird, ist bei einem Scale-Out. Dies liegt daran, dass ein Scale-Out von einzelnen Komponenten einer VM nicht möglich ist [LFWW16]. Vaquero et al. [VRC09] Untersuchungen zeigen, dass für den Begriff des Cloud Computings verschiedene Definitionen existieren. Diese Arbeit referenziert sich auf die Definition der US-Standardisierungsbehörde National Institute of Standards and Technology (NIST) [MG11], welche die essenziellen Charakteristiken von Cloud Computing behandelt.

[1] https://www.google.com/intl/de_ALL/drive/
[2] https://www.apple.com/de/icloud/

Die fünf Charakteristiken von Cloud Computing nach der NIST Definition sind:

- *On-demand self-service:*
  On-demand self-service bezeichnet die automatische Bereitstellung von Cloud-Ressourcen, zum Beispiel Speicher, durch den Konsumenten ohne das menschliches Eingreifen nötig ist.
- *Broad network access:*
  Cloud Ressourcen sind mittels Standardprotokollen über das Netz verfügbar.
- *Resource pooling:*
  Kunden teilen sich einen Ressourcenpool. Dadurch können Ressourcen unterschiedlich auf Nutzer verteilt werden und bei Bedarf umverteilt werden.
- *Rapid elasticity:*
  Cloud-Ressourcen können, auf Nachfrage, provisioniert und freigesetzt werden. Die Skalierung kann schnell und in großer Zahl geschehen, wodurch dem Verbraucher die zur Verfügung stehenden Cloud Ressourcen als unbegrenzt erscheinen.
- *Measured service:*
  Die Ressourcennutzung wird vom Cloud System kontinuierlich überwacht und protokolliert. Der Zugriff ist für Anbieter sowie auch Nutzer möglich, wodurch Transparenz für die genutzte Dienstleistung, welche vom Nutzer bezahlt wird, entsteht.

2.3 TOSCA

Mit der zunehmenden Bedeutung von Cloud Computing und IoT, sowohl in Firmen als auch in Privathaushalten, steigt die Notwendigkeit die Portabilität der Anwendungen zu erhalten. Aus diesem Grund gibt es den *Topology and Orchestration Specification for Cloud Applications* (TOSCA) Standard der *Organization for the Advancement of Structured Information Standards* (OASIS). Dabei dient TOSCA zur Beschreibung von Cloud-Anwendungen, indem Service Templates (siehe Abbildung 2.2) verwendet werden. Diese bestehen aus den folgenden Bestandteilen: Ein *Topology Template* ist ein gerichteter Graph, dessen Knoten *Node Templates* und dessen Kanten *Relationship Templates* entsprechen. *Node Templates* stammen dabei vom abstrakten Typ *Node Type* ab und *Relationship Templates* von *Relationship Types*. Zudem enthalten *Node Types* und *Relationship Types Interfaces* und *Properties*. Jedes Interface spezifiziert eine nichtleere Menge an Operationen, sowie Eingabe- und Ausgabeparameter. Ein Beispiel für einen *Node Type* ist die *DockerEngine*, deren Interface *InterfaceDockerEngine* die Operationen *startContainer* und *removeContainer* anbietet und eine *Property* wäre die *DockerEngineURL*. Hingegen wäre ein *Relationship Type ConnectsTo* mit der Spezifikation, wenn A *ConnectsTo* B ist, ist A die Quelle und B das Ziel. *Pläne* sind BPMN oder BPEL Modelle, die zum Starten, Skalieren oder Terminieren von Anwendungsinstanzen einer CSAR verwendet werden können.

Service Templates sowie die dazugehörigen Dateien, wie XML Schemas, werden in eine ZIP-Datei verpackt [BBKL14]. Diese wird auch als Cloud Service Archive (CSAR) bezeichnet. CSARs ermöglichen die Portabilität von Anwendungen, da sie alle Dateien, die für die Installation einer Anwendung notwendig werden, enthalten [BBKL14]. Aus diesem Grund werden CSARs als *self-contained* beschrieben. Ein Beispiel einer CSAR ist die in Kapitel 1.1 erwähnte *MyTinyToDo_Bare_Docker* mit den dazugehörigen Managementplänen wie Scale-Out.

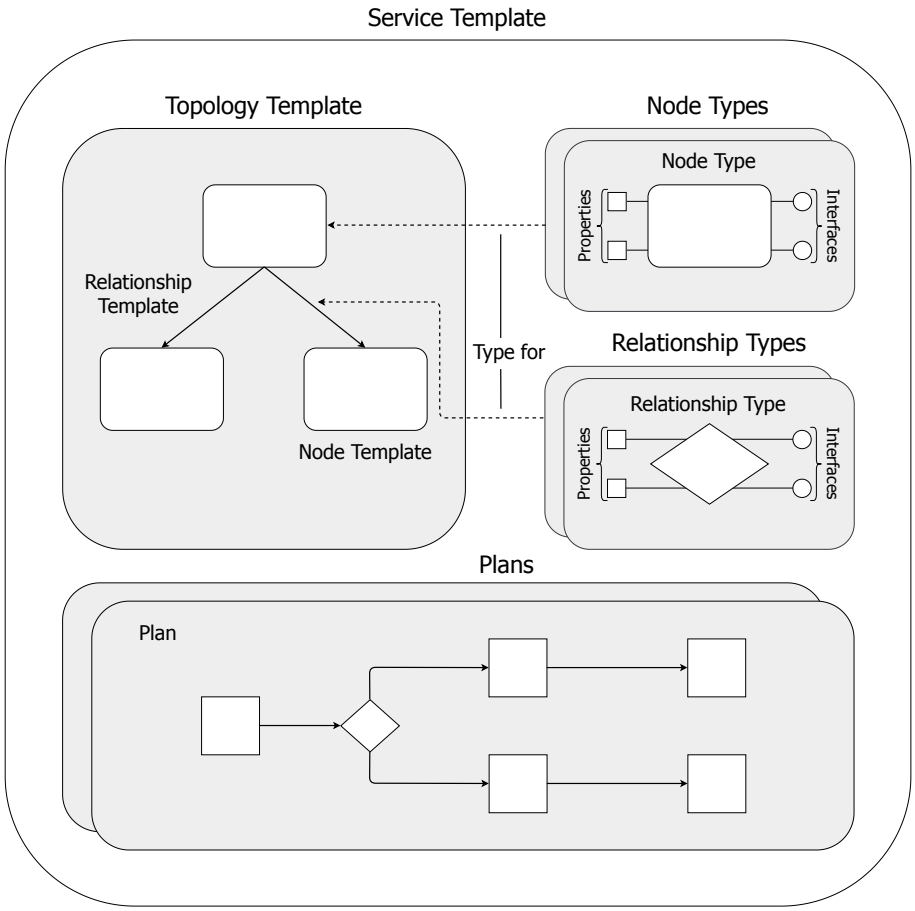

Abbildung 2.2: Struktur eines Service Templates (nach [OA13])

2.4 OpenTOSCA

OpenTOSCA[3] ist ein Open-Source Projekt der Universität Stuttgart. In Abbildung 2.3 ist eine Übersicht und die Interaktion der drei Komponenten (Winery, Container, Vinothek) des OpenTOSCA Ökosystems zu sehen [BEK+16]. Dabei ist die Winery das TOSCA-Modellierungswerkzeug, dass erlaubt die im vorherigen Abschnitt vorgestellten Topology Templates zu erstellen, die aus Node Templates und Relationship Templates bestehen. Die abstrakten Typen der Templates werden im Element Manager erstellt und verwaltet. Service Templates können als CSARs exportiert und im Container weiterverarbeitet werden. In einem Repository werden alle Daten, wie beispielsweise CSARs, gespeichert [KBBL13]. Dabei erfolgt die graphische Visualisierung durch die Modellierungssprache Vino4TOSCA [BBK+12].

Plan Engine: Die Plan Engine ist eine Laufzeitumgebung für BPEL Pläne [BBH+13], welche die Pläne der im Container installierten CSARs verwaltet. Zudem gibt die Plan Engine an, welche Anwendungsinstanzen einer CSAR aktiv sind und bei welchen es zu Fehlern kam.

[3] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

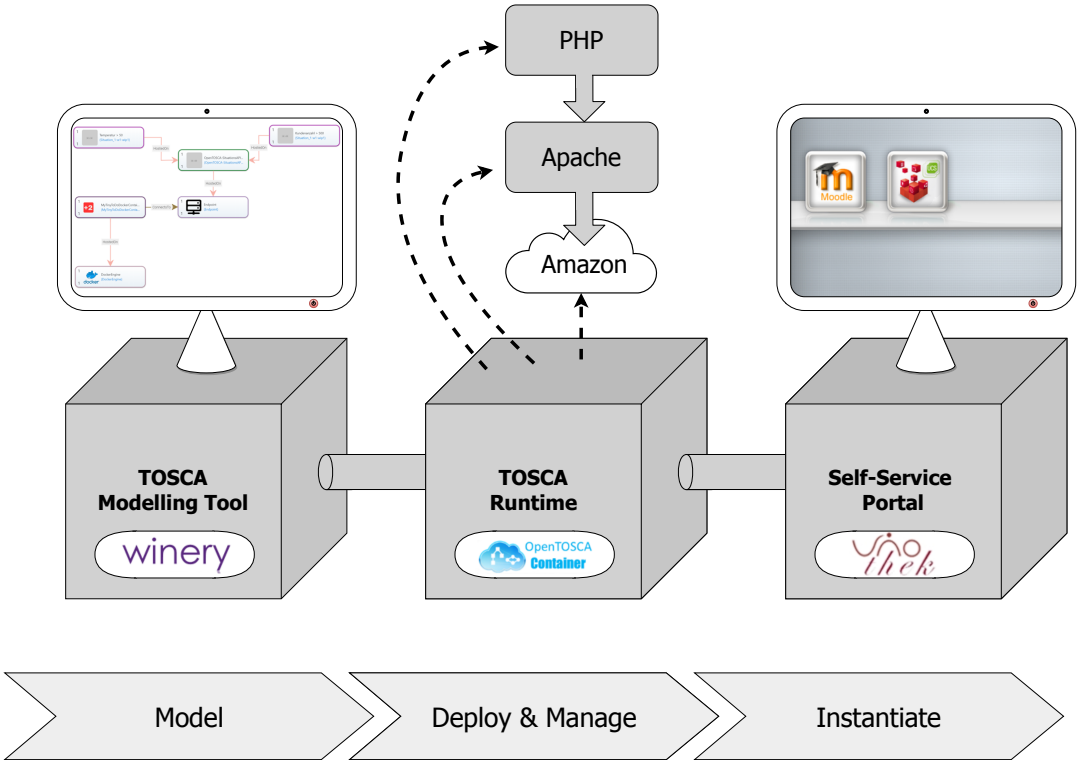

Abbildung 2.3: OpenTOSCA Ökosystem (nach [BEK+16])

Vinothek: Die Vinothek ist ein Self-Service Portal zum Verwalten von Anwendungsinstanzen der im Container installierten CSARs und dient als Abstraktionsschicht für Endbenutzer [BBK14]. Die Systemarchitektur der Vinothek ist in Abbildung 2.4 zu sehen. Dabei kommuniziert die GUI, welche auf Java Server Pages und HTML5 basiert, mittels einer REST API mit einem Server, der die Anfragen an den TOSCA Application Lifecycle Manager weiterleitet. Dieser ist für die Provisionierung und Verwaltung von Anwendungsinstanzen zuständig. Dabei wird ausgewertet, inwieweit die Operationen erfolgreich waren und basierend auf dieser Auswertung wird der GUI eine Antwort übermittelt. Somit handelt es sich um eine Client-Server-Architektur [BBK14]. Die TOSCA Runtime Integration Layer ist für die Integration von TOSCA-Laufzeitumgebungen zuständig. Die Plugins werden zur Erweiterung des TOSCA Application Lifecycle Managers und zur Implementierung der Schnittstellen, die von der TOSCA Runtime Integration Layer zur Verfügung gestellt werden, verwendet [BBK14].

Mit der neuen OpenTOSCA Version wurde die Vinothek durch die OpenTOSCA UI[4] ersetzt, die aus den drei folgenden Komponenten besteht:

> *Application:* Im Application Tab erfolgt der Upload von CSARs und das Starten von Anwendungsinstanzen einer installierten CSAR. Zudem enthält der Tab alle im Container installierten CSARs. Der Nutzer kann durch die Auswahl einer CSAR deren Anwendungsinstanzen ver-

[4] https://github.com/OpenTOSCA/ui

walten. Beispielsweise kann er Anwendungsinstanzen instanziieren, wenn der *Start Instance Button* ausgewählt wird. Werden für den Build Plan Parameter erforderlich, erscheint ein Pop-up Fenster, in dem der Nutzer diese eingeben kann.

Repository: Das Repository enthält eine Liste aller CSARs, welche in der Winery hochgeladen sind und im Container installiert werden können.

Administration: Der Administrations Tab enthält die Endpunkte zur Winery und der OpenTOSCA Runtime des Ökosystems.

Die Kommunikation mit der UI erfolgt äquivalent zur Vinothek mittels REST Anfragen. Die Benutzeroberfläche kann mittels Node.js[5] gestartet werden. Node.js ermöglicht eine effiziente Verwaltung von Netzwerkanwendungen, da es sich um eine asynchrone eventbasierte JavaScript-Laufzeitumgebung handelt [OS]. Dabei besteht die UI aus der Open-Source Plattform Angular.io [6], welche zur Entwicklung von Web Anwendungen verwendet wird und in TypeScript[7] geschrieben ist. Im Gegensatz zu JavaScript erlaubt TypeScript beispielsweise explizite Typdeklarationen.

Container: Der OpenTOSCA Container ist eine Open-Source TOSCA Runtime [BBH+13]. Der Container besteht aus zahlreichen Komponenten, jedoch ist für diese Arbeit nur die Container API relevant. Dabei ist die Kommunikation zwischen der Container API und der OpenTOSCA UI mittels REST Anfragen möglich.

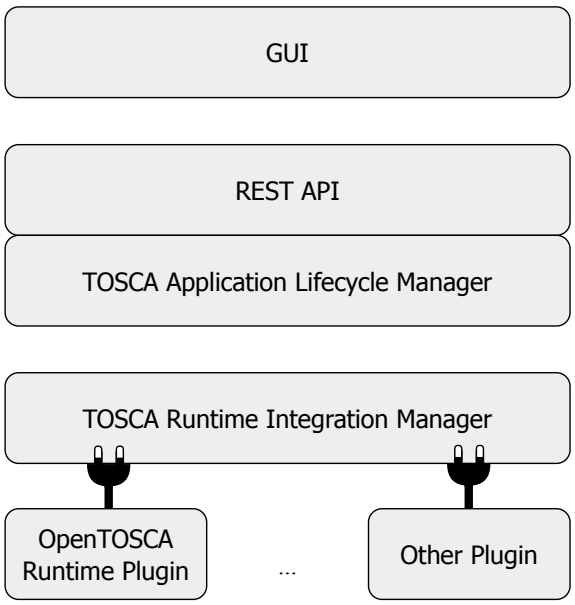

Abbildung 2.4: Systemarchitektur der Vinothek (nach [BBK14])

[5] https://nodejs.org/en/
[6] https://angular.io/
[7] https://www.typescriptlang.org/

2.5 Docker

Ein Docker ermöglicht die Isolation von Anwendungen, da es auf dem Prinzip der Containervirtualisierung basiert. Dadurch werden Techniken, wie Continuos Integration und Continuos Deployment unterstützt. Anwendungen und ihre Abhängigkeiten werden in ein Docker Image gepackt, sodass die Zeit für eine lokale Installation reduziert wird und die Funktionaltät der Anwendung beim Wechsel der Docker-Umgebung erhalten bleibt [And15]. Dabei unterscheidet sich ein Container von einer virtuellen Maschine (VM) darin, dass eine VM viele Speicherressourcen benötigt. Dies liegt daran, dass für jede neue VM das Betriebssystem sowie die Anwendungen kopiert werden müssen [Doc]. Aus diesem Grund eignen sich Container, da sie nur die Dateien speichern, welche für die Anwendung relevant sind und so schneller gestartet werden können.

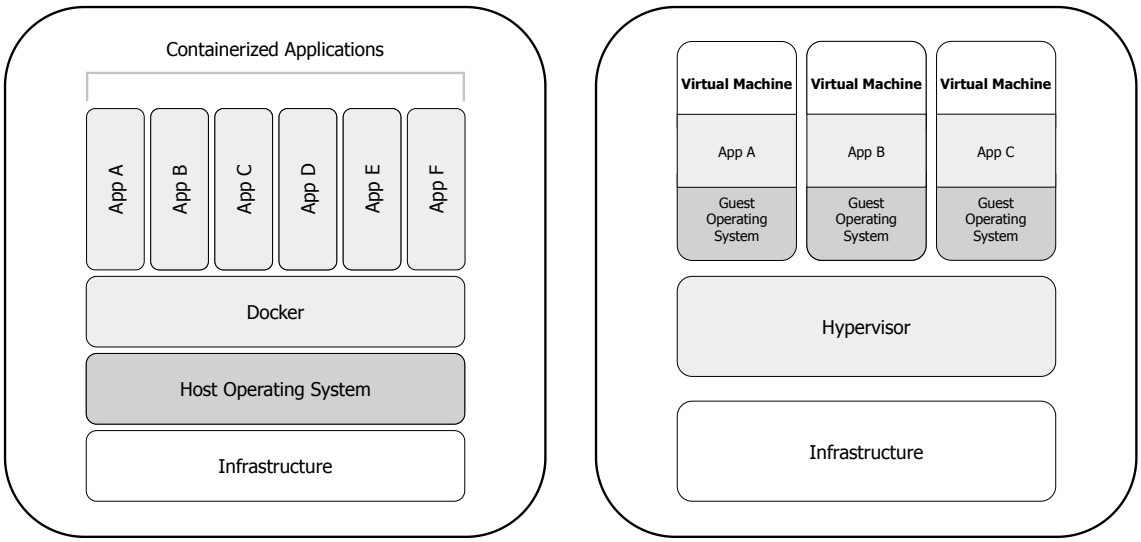

Abbildung 2.5: Vergleich zwischen Containern und Virtuellen Maschinen (nach [Doc])

3 Verwandte Arbeiten

In diesem Abschnitt werden Ansätze verwandter Arbeiten aus den Bereichen der kontekt- sowie situationsbewussten Erkennung, Modellierung und Adaption vorgestellt, die sich auf das Thema der Arbeit beziehen. Dabei wird insbesondere die Problematik der kontextbewussten Modellierung hervorgehoben, die Anlass zur Entwicklung der situationsbewussten Modellierung war.

3.1 Kontextbewusste Systeme

Es existieren viele Definitionen dessen, was ein Kontext ist. Die Definition, auf die sich diese Arbeit bezieht, ist das „ein Kontext jede Information ist, die zur Charakterisierung der Situation einer Entität verwendet wird“ [ADB99]. Dabei klassifizieren Dey et al. [ADB99] für das Ermitteln der Umgebung drei Entitäten. Diese sind Orte, Personen und Dinge. Jeder der drei Entitäten kann dabei durch Attribute, die sich in vier Klassen (Identität, Standort, Status, Zeit) unterteilen, näher beschrieben werden. Eines der ersten kontextbewussten Systeme ist das „Active Badge Location System“ von Want et al. [WHF92]. Dieses basiert auf der Technologie alle 15 Sekunden ein eindeutiges Signal der Länge einer Zehntelsekunde in ein Netzwerk aus Sensoren zu übertragen, um die Position einer Person zu ermitteln. Dabei bieten kontextbewusste Systeme den Vorteil sich automatisch an Änderungen der Umgebung adaptieren zu können, wodurch die Nutzerzufriedenheit und Effizienz steigen. Aus diesem Grund sind sie insbesondere im Bereich des Pervasive Computings relevant. Als Pervasive Computing bezeichnet man die Betrachtung „alle[r] Gegenstände der realen Welt [als] Teil eines Informations- und Kommunikationssystems“ [RH06].

Um den Kontext zu ermitteln, werden häufig Sensoren verwendet. Eine Art von Sensoren sind physikalische Sensoren wie Licht- und Geräuschsensoren. So kann beispielsweise mit Hilfe von Lichtsensoren festgestellt werden, ob sich eine Person in einem Haushalt befindet. Eine andere Art von Sensoren sind virtuelle Sensoren mit denen beispielsweise die Präferenzen eines Nutzers gemessen werden können. In Abbildung 3.1 ist der Systemablauf eines kontextbewussten Systems zu sehen. Zunächst werden Daten durch verschiedene Sensoren gesammelt. Anschließend werden die Rohdaten durch Analyse und Verarbeitungsschritte, wie beispielsweise feature extraction, gefiltert [KKR+13]. Zudem kann eine Middleware diese beiden Aufgaben übernehmen. Im folgenden Schritt dienen die Informationen als Entscheidungsbasis für die Anpassung des Systems. Auf dieser Grundlage erfolgt die Adaption des kontextbewussten Systems, der den Input für den nächsten Zyklus bereitstellt [KKR+13].

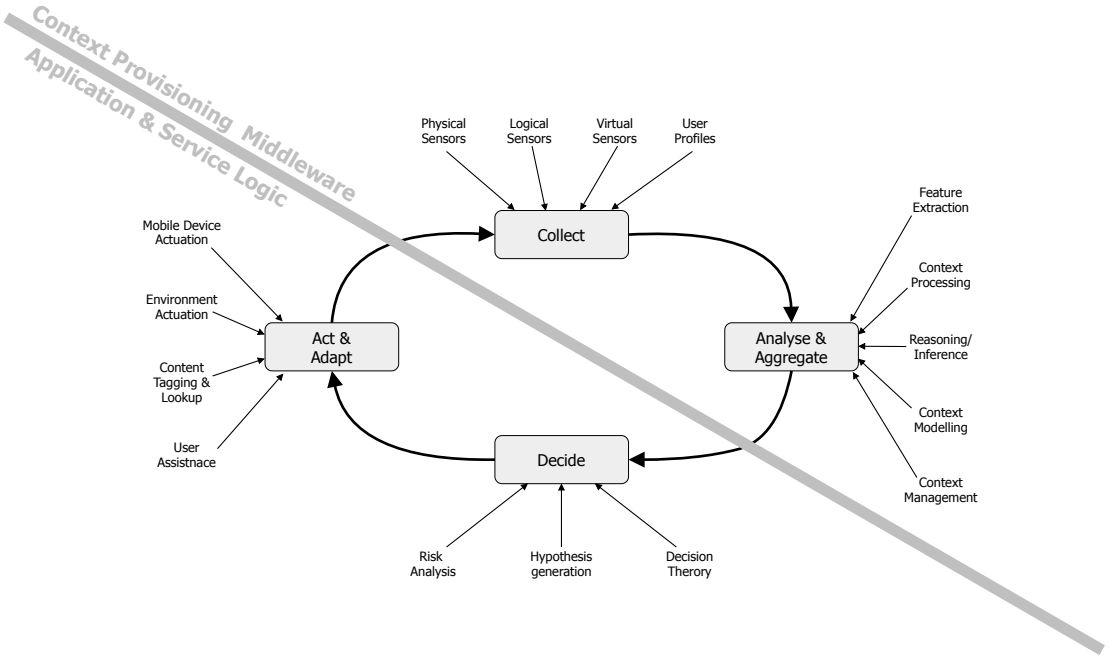

Abbildung 3.1: Ablauf eines kontextbewussten Systems (nach [DDF+06; KKR+13])

3.2 Kontextbewusste Benutzeroberflächen

Mit der zunehmenden Komplexität von Anwendungen steigt die Notwendigkeit dem Nutzer Informationen übersichtlich darzustellen. Dazu werden Benutzeroberflächen verwendet, die unter anderem durch falsches Design die Benutzbarkeit erschweren. Aus diesem Grund gibt es kontextbewusste Benutzeroberflächen. Diese bestimmen aus den Kontextinformationen des Nutzers Daten, die sie bereits ausfüllen oder vorschlagen können. Eine beispielhafte Anwendung wäre die Google Suchmaschine, welche Suchergebnisse nach den Präferenzen des Nutzers ordnet. Auch im medizinischen Bereich können kontextbewusste Benutzeroberflächen angewendet werden. Dafür wird eine neue Methode namens Context-driven Emergency Application (CEA) eingeführt [BP17]. Die CEA Methode nach Batarseh [BP17] erhält die Kontextinformationen des Benutzers aus der Notsituation, dem Persönlichkeitstyp, dem Alter, dem Geschlecht, der Sprache und der Rolle an der Universität (Mitarbeiter, Fakultät, Student in Wohnheimen, nicht wohnhaft). Anhand dieser Kontextinformationen wird die Benutzeroberfläche individuell an den Benutzer und an die Umgebung angepasst. Zum Beispiel wird die Schriftgröße basierend auf dem Alter des Benutzers adaptiert, wodurch die Benutzerfreundlichkeit erhöht wird [BP17]. Dabei ist es notwendig die relevanten Kontextinformationen herauszufiltern, um geeignetere Ergebnisse bereitzustellen. Das Bestimmen relevanter Kontextinformationen wird im nächsten Abschnitt genauer betrachtet.

3.3 Kontextualisierung von Geschäftsprozessen

Die große Menge an Daten führt zu einer hohen Komplexität, sodass kontextbewusste Systeme die Kontextinformationen filtern müssen. Insbesondere für die Darstellung und Flexibilität von Geschäftsprozessen ist es wichtig, dass die relevanten Informationen bestimmt werden, welche zu Änderungen führen können. Als Flexibilität eines Geschäftsprozesses bezeichnet man die konkrete Adaption eines Prozesses an externe Veränderungen, das heißt es werden nur die Teile des Prozesses abgeändert, die von dieser Änderung betroffen sind [RBF08]. Rosemann et al. [RBF08] stellen ein Modell zur Klassifikation von Kontextinformationen vor, welches vier Klassen identifiziert.*Immediate Context* umfasst alle Elemente, wie Ressourcen, Daten und Kontrollflüsse, die zur Beschreibung von Geschäftsprozessen verwendet werden. Diese Konstrukte sind ausreichend für die Ausführung von Geschäftsprozessen, wenn keine Änderungen im Kontext erfolgen. Der *Internal Context* betrachtet die Elemente, die Einfluss auf die interne Organisationsstruktur haben. Dies sind beispielsweise Stakeholder, die am Innenleben der Organisation beteiligt sind. Der *External Context* enthält die Elemente, die nur implizit mit der Organisationsstruktur assoziiert sind. Dies sind Faktoren wie Kunden oder Investoren. *Environmental Context* betrachtet die Elemente, welche die Ausführungsumgebung des Prozesses beeinflussen. Dabei können sich Variablen, wie beispielsweise das Wetter, sehr häufig über eine Zeitspanne ändern. Mithilfe dieser Klassifikation der Kontextinformationen wird das Betrachten der Umgebung transparenter, da nur noch die Kontextinformationen miteinbezogen werden, die für den Geschäftsprozess relevant sind. Zudem wird die Flexibilität des Geschäftsprozesses erhöht, da die Veränderungen in der Umgebung durch das eingeführte Framework klassifiziert werden.

Eine Schwäche des beschriebenen Ansatzes ist, dass nicht untersucht wird, wie das eingeführte Kontextframework in Prozessmodellierungssprachen, wie BPMN oder BPEL, integriert werden kann. Diese Thematik wird in [WKNL07] aufgegriffen. Dazu werden drei neue Konzepte (*Context Event*, *Context Query* und *Context Decision*) eingeführt, die in BPEL am Beispiel einer Smart Factory integriert wurden. Ein *Context Event* ist ein Ereignis, das basierend auf der Umgebung eingetreten ist. Hierbei wartet der Workflow asynchron auf das Ereignis und reagiert auf dieses, mithilfe eines Event Handlers [WKNL07]. Wie bereits eingangs erwähnt, kann die Menge an Kombinationen von Kontextinformationen zu einer hohen Komplexität führen. Um diese Problematik zu lösen, werden nur zuvor in der Kontextverwaltungsplattform registrierte *Context Events* betrachtet. Zusätzlich wird eine *Context Query* verwendet, welche den synchronen Zugriff auf Kontextdaten ermöglicht und anschließend die Kontextdaten filtert [WKNL07]. Das Konzept der *Context Decision* wird für Verzweigungen in Prozessen verwendet, die entweder zu true oder false ausgewertet werden [WKNL07]. Die neuen Konzepte ermöglichen dem Prozess sich, basierend auf den Kontextinformationen, automatisch anzupassen und die Flexibität des Geschäftsprozesses zu erhöhen. Im Gegensatz zu diesem Ansatz beschäftigt sich das SitOPT Projekt unter anderem mit der Abstrahierung des Kontexts durch Situationen und stellt Ansätze zur Situationserkennung sowie zur Modellierung von situationsbewussten Workflows vor.

3.4 Kontextbewusstes Management von Cloud Anwendungen

In Abschnitt 2.2 wurde bereits der Begriff des Cloud Computings erläutert. Dabei finden Cloud Dienste sowohl beim Endnutzer, durch Onlinefilesysteme wie Onedrive, als auch bei Unternehmen Anwendung [BRK]. Jedoch führt die Vielzahl an heterogenen Softwarekomponenten zu Herausforderungen für das Management dieser Komponenten. Insbesondere in zusammengesetzten Cloud Anwendungen kann es zu unerwünschten Nebeneffekten kommen, da verschiedene Technologien mit eigenen Management APIs miteinander interagieren [BBK+14]. Aus diesem Grund wird es notwendig den Kontext von Management Tasks zu betrachten.

Dazu stellen Breitenbücher et al. [BBK+14] ein sechs Phasen Modell vor, welches im folgenden Text beschrieben wird. In der ersten Phase wird die Anwendung, ihre Komponenten und deren Beziehungen beschrieben. Dazu werden zusätzlich die Typen dieser Komponenten erforderlich. Die Phase 2 beschäftigt sich mit der deklarativen Beschreibung von Management Tasks, wobei das Augenmerk nicht auf der technischen Umsetzung, sondern auf deren Auswirkungen liegt. Das Ergebnis dieser Phase wird als *Declarative Management Description Model* (DMDM) bezeichnet. Die nächste Phase beschäftigt sich mit der Analyse der Auswirkungen von Management Tasks auf die in Phase 1 beschriebenen Komponenten und deren Beziehungen. Basierend auf der Analyse aus Phase 3 wird im nächsten Schritt das DMDM adaptiert. In Phase 5 wird das *Declarative Management Description Model* zu einem ausführbaren *Imperative Management Description Model* (IMDM) transformiert. Im letzten Schritt wird das IMDM ausgeführt, welches das Management in Bezug auf den Kontext der Anwendung spezifiziert.

3.5 SitOPT

In Smart Homes steigt die Anzahl der Sensoren, welche kontinuierlich Daten übermitteln müssen. Diese große Menge an Daten muss interpretiert und effizient ausgewertet werden. Wie bereits erwähnt führt das Betrachten von Kontextinformationen zu einer hohen Anzahl von Kombinationen, die beispielsweise in Workflows durch viele Verzweigungen behandelt werden müssen [WSB15]. Die Kombinationen können reduziert werden, indem Situationen als konkrete Abstrahierungen von Kontextdaten betrachtet werden. Dabei ist SitOPT ein Projekt der Universität Stuttgart und verfolgt den Ansatz der situationsbezogenen Adaption von Workflows zur Erkennung von Situationen mittels zentraler Middleware [WSB15]. Dies wird durch die Drei Schichten Architektur [WSB15], welche in Abbildung 3.2 zu sehen ist, ermöglicht.:

1) *Sensorschicht:* Die Sensorebene ist die niedrigste Ebene und besteht aus physikalischen Sensoren, welche ihre Daten an die obere Schicht weiterleitet.

2) *Situationserkennungsschicht:* Die Situationserkennungsebene erhält Daten aus der Sensorebene und erkennt mittels einer Middleware Situationen, deren Auftreten an die Bedingungen der Sensorebene gebunden sind.

3) *Situationsbewusste Workflow Schicht:* Die situationsbewusste Workflow Schicht ist für die situationsbezogene Adaption und Modellierung verantwortlich. Dabei wird der Standard Workflow ausgeführt und sobald eine Situation eintritt, wird basierend auf der Antwort der unteren Schicht das Workflow Fragment in den Workflow integriert.

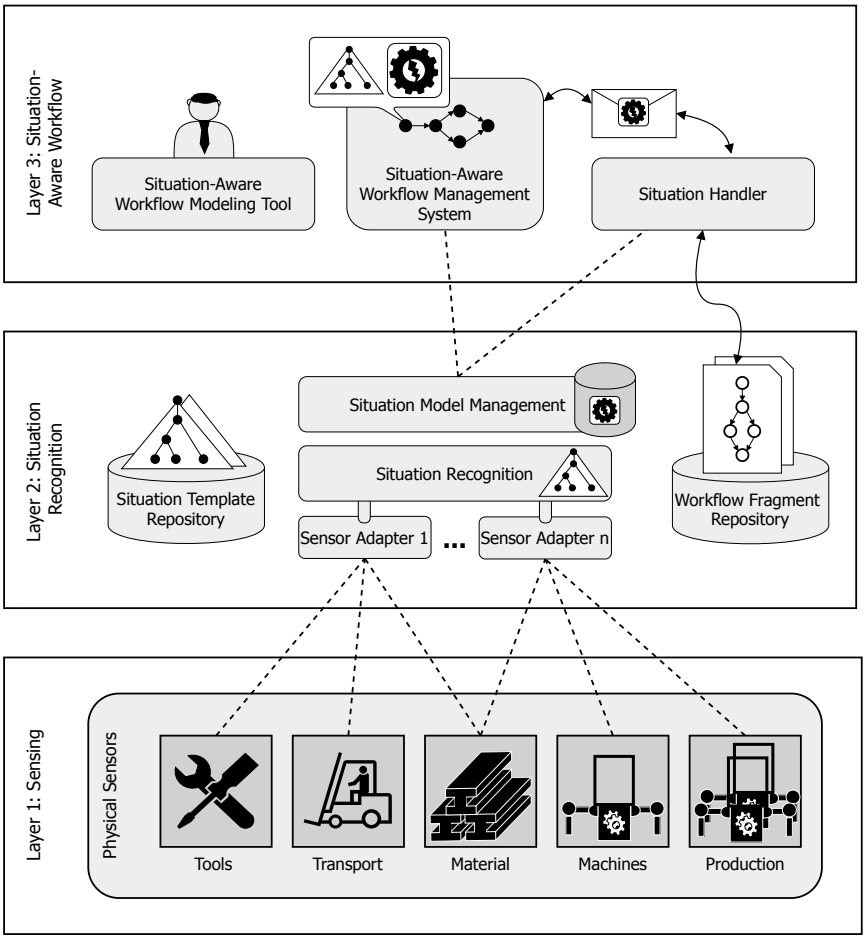

Abbildung 3.2: Architektur des SitOPT Systems (nach [WSB15])

3.6 Situationsbewusste Workflows

Konkrete Ansätze zur Modellierung von kontextbewussten Workflows wurden in Abschnitt 3.3 präsentiert. Dabei ist bereits das Modellieren der Kontextinformationen eine zeitintensive und kostspielige Aufgabe [GBH05; LCG+09]. Dies liegt daran, dass bereits eine hohe Anzahl an Verzweigungen erzeugt wird, um jede Kombination eines Kontextwerts zu betrachten. Aus dieser Problematik ist die Disziplin der situationsbewussten Workflows entstanden, indem der Kontext abstrahiert und Situationen betrachtet werden. Ein Ansatz zur Modellierung von situationsbewussten Workflows wird in [BHK+15] vorgestellt. Dabei führen Breitenbücher et al. [BHK+15] das Konzept von *Situationevents* und *Situationscopes* ein. *Situationevents* werden mit einer Aktivität verknüpft und bestimmen den Zeitpunkt ihrer Ausführung. Dieser ist an den Status der *Situationevents* gebunden, der entweder aktiv oder inaktiv ist. *Situationscopes* dienen zur Gruppierung von Aktivitäten und erlauben die Kombination mit *Situationevents*. Darüber hinaus können der Fehlerfall, dessen Behandlung und der Zeitpunkt des Eintretens spezifiziert werden. Der Zeitpunkt des Eintretens gibt an, was passiert, wenn noch nicht alle *Situationevents* zutreffen und ob die Ausführung an dieser Stelle pausiert oder weiter fortführt. Der Fehlerfall tritt ein, wenn ein *Situationevent* den

Status wechselt, obwohl der *Situationscope* noch nicht beendet ist. Dies kann dazu führen, dass beispielsweise alle Aktivitäten, die sich im *Situationscope* befinden und schon ausgeführt wurden, wieder rückgängig gemacht werden müssen.

Diese Arbeit stützt sich auf der in [BHK+15] eingeführten situationsbezogenen Modellierung, wobei *Situationscopes* gesamte Pläne situationsabhängig umfassen statt einzelne Aktivitäten situationsbezogen zu gruppieren. Dabei werden die *Situationsevents*, deren Status die Ausführung der Pläne bedingen, an die *Situationsscopes* geknüpft. Für das Szenario aus Kapitel 1.1 bedeutet das, dass der Scale-Out Plan der CSAR *MyTinyToDo_Bare_Docker* erst ausgeführt wird, wenn die *Situationevents AktiveKunden* oder *AuslastungHoch* zutreffen. Außerdem wird die in [BHK+15] präsentierte Modellierung um das Konzept der Aggregation von Situationen erweitert, wobei die Semantik einer aggregierten Situation einer Situation entspricht.

3.7 Situationsbewusstes Management von cyberphysischen Systemen

Eine zentrale Rolle für den Bereich der Internet der Dinge spielen cyber-physische Systeme (CPS). Zurückzuführen ist dies auf die zunehmende Vernetzung von physischen und virtuellen Systemen. Cyberphysische Systeme haben die Aufgabe sich dynamisch an Änderungen zur Laufzeit, die in der Umgebung sowie den Komponenten stattfinden, anzupassen und die Verwaltung dieser Komponenten zu übernehmen. In Abbildung 3.3 ist das Smart Home Szenario aus [KBL19] zu sehen. Dabei verwaltet das CPS Sensoren und Aktuatoren, die bei Bewegung das Licht ein- und ausschalten. Dieses System soll nun auf eine neuere Version upgedatet werden. Dabei wird das Update nur ausgeführt, wenn genügend Energie sowie Update Dateien vorhanden sind und der Hausbesitzer nicht daheim ist [KBL19]. Dabei ist eine Herausforderung für die Situationserkennung, dass genaue Zustände über den Kontext definiert werden müssen. Dies ist eine zeitintensive und fehleranfällige Aufgabe [KBL19]. Aus diesem Grund werden Konzepte aus dem SitOPT Projekt [WSB15] sowie aus situationsbewussten Workflows [BHK+15] für die Modellierung der Systeme übernommen.

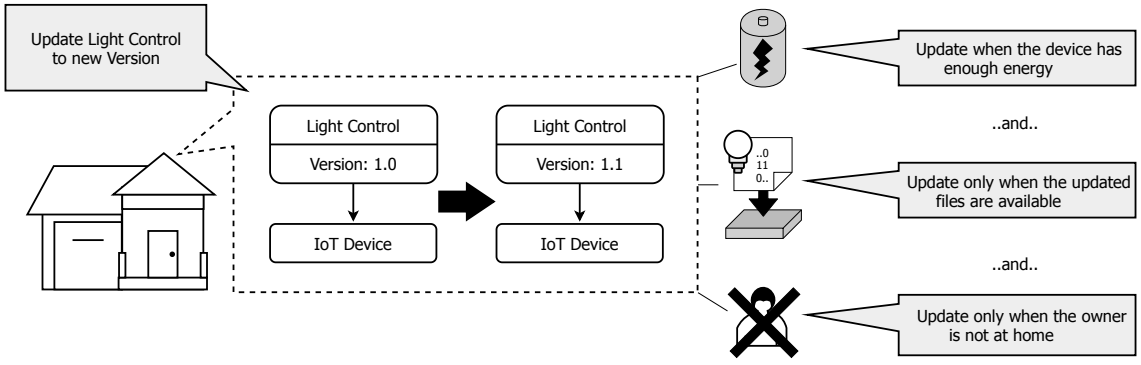

Abbildung 3.3: Update der Lichtversion, sofern alle Situationen zutreffen (nach [KBL19]).

Für den Fehlerfall, dass eine Situation während der Ausführung des Managementplans ihren Status ändert, ist in [Nie20] ein prototypischer Ansatz zur Prävention dieser Fälle in OpenTOSCA umgesetzt. Dabei wird eine Worst Case Abschätzung anhand der Dauer der einzelnen Operationen der Managementpläne berechnet und anhand dieser wird abgeschätzt, ob der Plan mit den Situationen ausgeführt werden kann [Nie20]. Dies erhöht die Verfügbarkeit und verhindert, dass bereits ausgeführte Operationen rückgängig gemacht werden müssen.

4 Problemstellung

Dieses Kapitel beschäftigt sich mit der Problemstellung dieser Arbeit und erläutert die Notwendigkeit der situationsabhängigen Modellierung und Provisionierung für die OpenTOSCA Benutzeroberfläche. Hierbei besteht die derzeitige Problematik darin, dass weder die situationsabhängige Ausführung von Plänen noch die Verwaltung von Situationen in der Benutzeroberfläche möglich ist. Die Relevanz dieser Disziplin ist in Abbildung 4.1 zu sehen.

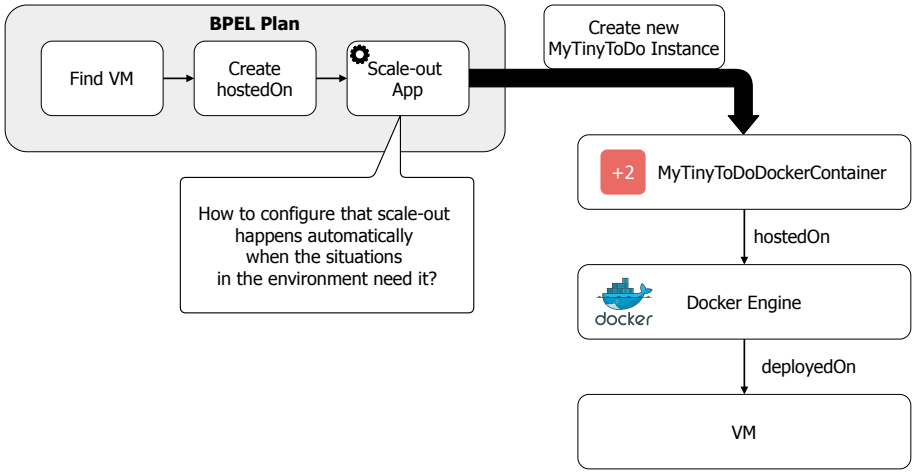

Abbildung 4.1: BPEL Plan für das Skalieren von Anwendungsinstanzen

Wie in Abschnitt 2.3 beschrieben, sind CSARs *self-contained* [BBKL14]. Das heißt sie enthalten alle Dateien, die für eine Installation einer Anwendung notwendig sind [BBKL14]. Dies sind unter anderem Service Templates, die aus einem Topology Template und unterschiedlichen Plänen bestehen. In Abbildung 4.1 ist ein möglicher BPEL Plan für das Skalieren von Anwendungsinstanzen der CSAR *MyTinyToDo_Bare_Docker* zu sehen. Dabei wird zunächst eine passende VM gesucht, sodass die im nächsten Schritt erforderliche hostedOn Beziehung zwischen der *DockerEngine* und dem *MyTinyToDoDockerContainer* realisiert werden kann. Im letzten Schritt wird ein Scale-Out der App ausgeführt, dass ist in diesem Fall die Instanziierung einer neuen Anwendung *MyTinyToDo*. Die Herausforderung ist es dabei, den Scale-Out automatisch zu konfigurieren beziehungsweise zu starten genau dann wenn die Situationen der Umgebung es erfordern. In OpenTOSCA sind Konzepte, unter anderem Situationen und Situationstrigger, umgesetzt, die sich mit dieser Herausforderung auseinandersetzen. Jedoch sind diese Konzepte nicht für den Nutzer in der Benutzeroberfläche greifbar.

4.1 Situationsbewusste Modellierung in OpenTOSCA

4.1.1 Situationen

Situationen können durch @GET, @PUT und @POST Anfragen an die REST API des OpenTOSCA Containers verwaltet werden. Dabei wird eine Situation als XML-Dokument, wie in Listing 4.1 zu sehen ist, definiert und durch eine @POST Anfrage an die API erstellt. Die *ThingId* bezieht sich auf das Objekt, zum Beispiel einen Sensor, von dem Kontextinformationen für die Situation kontinuierlich bezogen werden. Situationen werden innerhalb von *Situation Templates* modelliert. Wie in Abschnitt 3.5 beschrieben, enthalten Situation Templates die Beschreibung einer Situation. Dabei ist ein Situation Template ein gerichteter Graph, der auch als Situation Aggregation Tree (SAT) bezeichnet wird [ZHK09]. Die Blattknoten enthalten Kontextinformationen und die Wurzel ist die erkannte Situation [ZHK09]. *Active* gibt den Zustand einer Situation an, wobei dieser entweder aktiv oder inaktiv sein kann. Nach dem Erstellen von Situationen können diese durch eine @GET Anfrage an die API angefragt werden. Zudem sind alle Situationen unter `http://localhost:1337/situationsapi/situations` zu finden und jede Situation erhält eine eindeutige Zahl als ID. Diese kann verwendet werden, um @PUT Anfragen an die `http://localhost:1337/situationsapi/situations/situationID` zu stellen, um eine Situation aktiv oder inaktiv zu setzen. Desweiteren ist die ID für die im nächsten Abschnitt erwähnten Situationstrigger relevant. Denn jeder Situationstrigger enthält mindestens eine Situation, indem er auf dessen ID referenziert.

```
<?xml version="1.0" encoding="UTF-8"?>
<Situation>
    <ThingId>NetzwerkMonitor</ThingId>
    <SituationTemplateId>AuslastungHoch</SituationTemplateId>
    <Active>false</Active>
</Situation>
```

Listing 4.1: Aufbau einer Situation

4.1.2 Situationstrigger

Situationstrigger werden ebenfalls als XML-Dokument definiert und durch eine @POST Anfrage an die REST API des OpenTOSCA Containers erstellt. Dabei besteht jeder Situationstrigger aus einer nichtleeren Menge an Situationen und einer *CsarId*. Die *CsarId* bezieht sich auf die CSAR, die im Container installiert ist. Dabei spezifiziert das Element *onActivation* die Aktivierung des Situationstriggers. Stimmt der Wert des *onActivation* Elements mit dem *active* Wert aller Situationen im Situationstrigger überein, kann dieser gestartet werden. Ist das Element *isSingleInstance* auf true gesetzt, dürfen keine weiteren Instanzen von Verwaltungsprozessen ausgeführt werden. In *InputParameters* stehen die Parameter, die für das Starten des Plans erforderlich sind. Zusätzlich geben *InterfaceName* und *OperationName* an, welches Interface implementiert sein muss und welche Operation ausgeführt werden soll. In Listing 4.2 ist die Struktur eines Situationstriggers zu sehen. Dieser wird verwendet um die Anwendungsinstanz 12 der CSAR *MyTinyToDo_Bare_Docker* zu skalieren, wenn die Situationen *AuslastungHoch* und *AktiveKundenanzahlUeber500* zutreffen.

Eine Restriktion, die durch die Definition des *onActivation* Elements entsteht, ist, dass alle Situationen im Situationstrigger UND-verknüpft sind. Es ist nicht möglich zu definieren, dass der Situationstrigger gestartet wird, wenn bereits eine der beiden Situationen zutrifft. Dies ist theoretisch nur möglich, indem zwei Situationstrigger erstellt werden. Jedoch führt das zu der Problematik, dass in diesem Fall zwei Mal die Anwendungsinstanz 12 skaliert wird, wenn beide Situationen zutreffen. Somit ist es nicht möglich das Szenario aus Abbildung 1.1 in der UI zu simulieren. Dies liegt daran, dass Situationen, Situationstrigger sowie die situationsbezogene Verwaltung von Anwendungsinstanzen beziehungsweise die situationsgebundene Ausführung von Plänen nicht in der UI vorhanden sind. Dabei existiert das Starten von Managementplänen bereits in der UI. Aus diesem Grund wird im nächsten Abschnitt die Instanzerstellung von Anwendungen in der Benutzeroberfläche betrachtet.

```
<?xml version="1.0" encoding="UTF-8"?>
<SituationTrigger>
    <Situations>
        <SituationId>2</SituationId>
        <SituationId>3</SituationId>
    </Situations>
    <CsarId>MyTinyToDo_Bare_Docker.csar</CsarId>
    <onActivation>true</onActivation>
    <isSingleInstance>true</isSingleInstance>
    <InterfaceName>scaleout_dockercontainer</InterfaceName>
    <OperationName>scale-out</OperationName>
    <InputParameters>
        <InputParameter>
            <name>OpenTOSCAContainerAPIServiceInstanceURL</name>
            <Value>12</Value>
            <Type>String</Type>
        </InputParameter>
        <InputParameter>
            <name>ContainerSSHPort</name>
            <Value>9999</Value>
            <Type>String</Type>
        </InputParameter>
        <InputParameter>
            <name>ApplicationPort</name>
            <Value>9998</Value>
            <Type>String</Type>
        </InputParameter>
    </InputParameters>
</SituationTrigger>
```

Listing 4.2: Aufbau eines Situationstriggers

Außerdem gibt es derzeit keine Möglichkeit Situationen zu aggregieren, sodass wiederkehrende Situationsgruppen immer komplett neu zugewiesen werden müssen und dadurch die Fehleranfälligkeit von Eingaben steigt. Dabei bieten aggregierte Situationen den Vorteil, dass der Wahrheitswert nur einmal abgefragt werden muss, wobei bei den derzeitigen Situationstriggern dieser für jede einzelne Situation abgefragt wird. Zudem ist eine aggregierte Situation im Fall einer Situationsänderung effizienter, da nur einmal der Wahrheitswert neu ausgerechnet werden muss und bei mehreren Situationstriggern effizienter berechnet werden kann als wenn jede Situation einzeln berechnet wird. Außerdem wenn eine Situation gelöscht wird, muss der gesamte Situationstrigger gelöscht werden. Im Fall einer aggregierten Situation ist dies nicht zwingend notwendig. Dabei wird unterschieden, ob die Situation nur aus der einen aggregierten Situation entfernt werden soll oder komplett gelöscht

werden soll. Im ersten Fall, wenn die aggregierte Situation aus einer Menge an Situationen größer gleich 2 besteht, wird die zu löschende Situation entfernt und der Wahrheitswert der aggregierten Situation neu berechnet. Im zweiten Fall werden beim Löschen einer Situation alle aggregierten Situationen und Situationstrigger gelöscht, welche diese enthalten.

4.2 Instanzerstellung in der Benutzeroberfläche

In der Benutzeroberfläche ist es ebenfalls möglich eine Instanz einer CSAR zu starten. Dies wird an dieser Stelle beispielhaft an der CSAR *MyTinyToDo_Bare_Docker* beschrieben. Ist die CSAR im Container installiert, kann diese für die Verwaltung von Anwendungsinstanzen ausgewählt werden. Dabei sind für das Erstellen einer Instanz alle Parameter eines Build Plans erforderlich, diese werden von dem Benutzer eingegeben (siehe Abbildung 4.2). Anschließend kann durch Klicken des *Run Management Plan Buttons* der BuildPlan ausgeführt werden. War das Instanziieren erfolgreich, wird in der Liste der Anwendungsinstanzen der CSAR ein neuer Eintrag mit der Instanz ID, dem Status in welcher sich die Instanz befindet (*Running* oder *Terminate*) und einem Zeitstempel erstellt. Das Erstellen des Szenarios aus Kapitel 1.1 ist jedoch nicht möglich, denn eine situationsbezogene Ausführung von Plänen sowie der Zugriff auf Situationen und auf Situationstrigger ist in der Benutzeroberfläche nicht vorhanden.

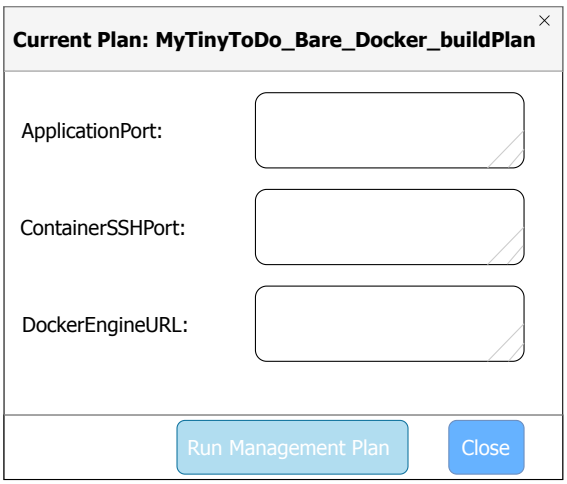

Abbildung 4.2: Instanzerstellung in der OpenTOSCA UI

5 Konzept zur situationsbezogenen Modellierung und Steuerung

Dieses Kapitel beschäftigt sich mit den Konzepten, um eine Modellierungsoberfläche für Situationen innerhalb der Benutzeroberfläche des OpenTOSCA Ökosystems bereitzustellen. Zunächst wird in Abschnitt 5.1 beschrieben, wie die Architektur der UI erweitert wurde, um diese Konzepte umzusetzen. Anschließend wird in 5.2 beschrieben, wie die situationsabhängige Ausführung von Anwendungsinstanzen einer CSAR umgesetzt wird. Zum Schluss wird das Konzept der aggregierten Situationen eingeführt und wie diese mit Situationstriggern kombiniert werden können.

5.1 Neue OpenTOSCA UI Architektur

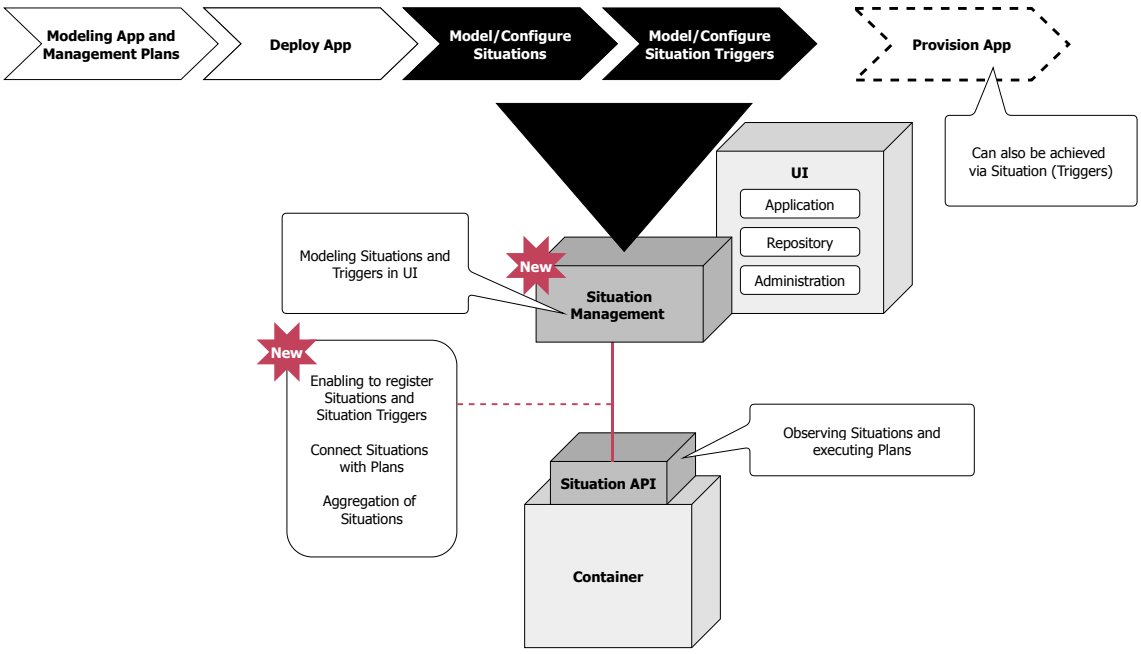

Abbildung 5.1: Neue OpenTOSCA UI Architektur

In Abbildung 5.1 ist die neue Architektur der UI zu sehen, welche neue Konzepte für das OpenTOSCA System ermöglicht. Dabei wird durch die neue *Situation Management* Komponente die Modellierung beziehungsweise Verwaltung von Situationen in der UI möglich. Diese war bisher nur in der Situations API vorhanden. Hierbei kommuniziert die *Situation Management* Komponente der Benutzeroberfläche mit der Situations API des Containers mittels REST Anfragen, wodurch zusätzlich die situationsabhängige Ausführung von Managementplänen möglich wird. Außerdem

wird durch das Verwenden von Situationstriggern die bisherige Provisionierung von Apps in der Benutzeroberfläche um das Situationsbewusstsein erweitert. Dies wird in Abschnitt 5.2 näher erläutert. Zudem kann durch die Erweiterung der Situations API das Konzept der Aggregation von Situationen eingeführt werden (siehe Abschnitt 5.3).

5.2 Situationsabhängige Ausführung von Anwendungsinstanzen

Das Konzept der situationsabhängigen Ausführung von Anwendungsinstanzen ist in Abbildung 5.2 zu sehen. Dabei erstellt der Nutzer einen Situationstrigger, der aus den Elementen, die in 4.1.2 vorgestellt wurden, besteht und einem zusätzlichen Element *AggregatedSituation*. An dieser Stelle gibt es zwei Möglichkeiten einen Situationstrigger zu starten. Die erste Variante ist es den Situationstrigger manuell durch Drücken des *Play Buttons* in der UI zu starten.

Zunächst achtet der Situationstrigger darauf, dass alle Situationen aktiv beziehungsweise inaktiv sind, damit er aktiviert werden kann. Dies geschieht durch das Element *onActivation*. Stimmt das *onActivation* Element mit jedem *active* Wert der angegebenen Situationen im Situationstrigger überein, wird er gestartet. Danach wird überprüft, ob die im Situationstrigger angegebene CSAR in der Container API installiert ist. Sofern diese installiert ist, wird kontrolliert, ob das im Situationstrigger angegebene Interface mit der zugehörigen Operation existiert. Dabei gibt der *InterfaceName* an, welches Interface der CSAR ausgewählt werden soll und der *OperationName* spezifiziert, welche Operation des Interfaces, zum Beispiel instanziieren, ausgeführt werden soll.

Um den Plan zu starten, werden die Eingabeparameter des Situationstriggers extrahiert. Im Fehlerfall wird der Situationstrigger nicht gestartet und der Nutzer erhält eine Fehlermeldung. Ist der Situationstrigger erfolgreich gestartet, übernimmt er die Rolle eines Situationsscopes [BHK+15]. Das bedeutet der Situationstrigger umschließt die Pläne der CSAR, indem er sie an die Situationen aus dem Situationstrigger bindet. Wichtig ist, dass jeder Situationstrigger nur einen Plan startet und nicht alle Pläne einer CSAR gleichzeitig. Aus diesem Grund ist in der Abbildung 1.1 die Kennzeichnung ODER zu sehen.

Für das Szenario aus Kapitel 1.1 bedeutet das, dass die Situationen *AuslastungHoch* und *AktiveKundenanzahlUeber500* dem Situationstrigger nach dem Erstellen zugewiesen werden. Sobald der Administrator merkt, dass die aktive Kundenanzahl über 500 liegt und die Auslastung der Server hoch ist, indem beispielsweise die Anfragen langsamer oder gar nicht beantwortet werden, klickt er auf den *Play Button* und startet eine neue Anwendungsinstanz der CSAR *MyTinyToDo_Bare_Docker*. Jedoch ist das Ziel nicht zu warten bis beide Situationen des Situationstriggers zutreffen, um diesen manuell zu starten, sondern diesen vollautomatisiert zu starten und zu verwalten.

Die andere Variante ist es den Situationstrigger durch eine @PUT Anfrage einer Situation zu starten. Beträgt die Zahl aktiver Kunden über 500, aber die Auslastung ist gering, kann der Administrator den Situationstrigger nicht starten. Sobald die Auslastung hoch ist, wird der Status der Situation durch eine @PUT Anfrage auf *true* gesetzt und der Situationstrigger automatisch gestartet. Nach dem erfolgreichen Instanziieren lassen sich durch einen anderen Situationstrigger, dessen *OperationName* terminate ist, Anwendungsinstanzen beenden. Dafür muss der zusätzliche Parameter

OpenTOSCAContainerAPIServiceInstanceURL definiert werden, der auf eine existierende Anwendungsinstanz der CSAR referenziert. Zusätzlich können auf der erstellten Instanz Managementpläne, wie Skalieren, ausgeführt werden. Dazu muss der *InterfaceName*, der *OperationName* sowie der Parameter *OpenTOSCAContainerAPIServiceInstanceURL* spezifiziert werden.

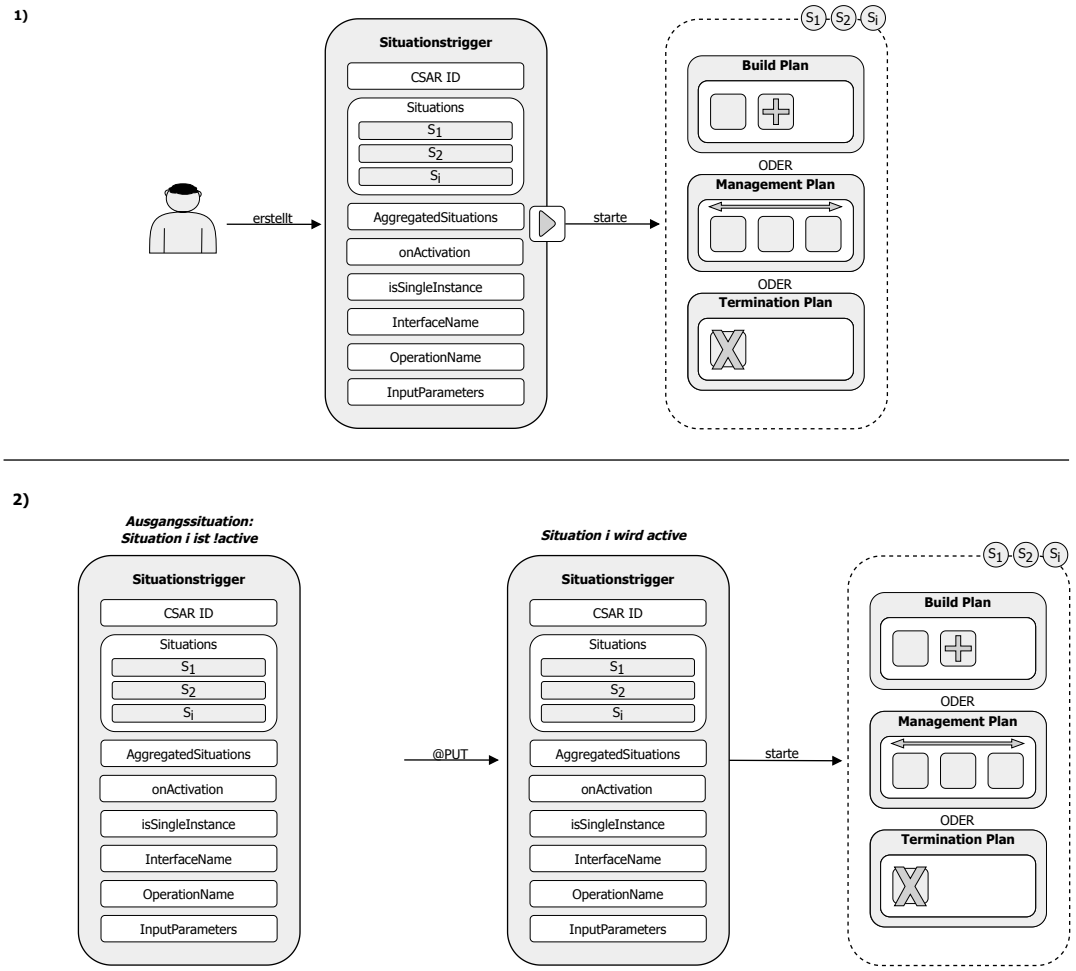

Abbildung 5.2: Konzept: Situationsabhängige Steuerung von Anwendungsinstanzen

5.3 Aggregation von Situationen

In Abbildung 5.3 ist ein weiteres Konzept zur situationsabhängigen Modellierung zu sehen. Dabei kann eine nichtleere Menge an Situationen selektiert werden, die aggregiert werden soll. Diese wird in einer aggregierten Situation A1 gespeichert, die neben der Menge an Situationen, einen logischen Ausdruck sowie ein active Element enthält. Der logische Ausdruck wird vom Nutzer spezifiziert und ermöglicht es Situationen mit einem UND oder einem ODER zu verknüpfen. Der *active* Wert basiert auf dem logischen Ausdruck und gibt an, ob dieser im Moment zutrifft.

Dieses Konzept erlaubt es das Szenario aus Abbildung 1.1 zu realisieren. Zuvor mussten alle im Situationstrigger spezifizierten Situationen zutreffen oder nicht, um diesen zu starten. Jedoch ist es für das Szenario nicht notwendig, dass sowohl die Auslastung hoch als auch die aktive Kundenanzahl über 500 ist, sondern vielmehr reicht es bereits, wenn eine Situation zu einem Zeitpunkt zutrifft. Dabei kann sich der *active* Wert nach einer @PUT Anfrage an eine Situation in der API ändern, da der Status einer aggregierten Situation an den logischen Ausdruck gebunden ist. Zudem ermöglicht das Prinzip der Aggregation eine Sammlung von Situationen mehrmals zuzuweisen. Die situationsabhängige Ausführung von Plänen mit aggregierten Situationen erfolgt äquivalent zu der mit Situationen, wobei der *onActivation* Wert des Situationstriggers zusätzlich vom *active* Wert der aggregierten Situation abhängt.

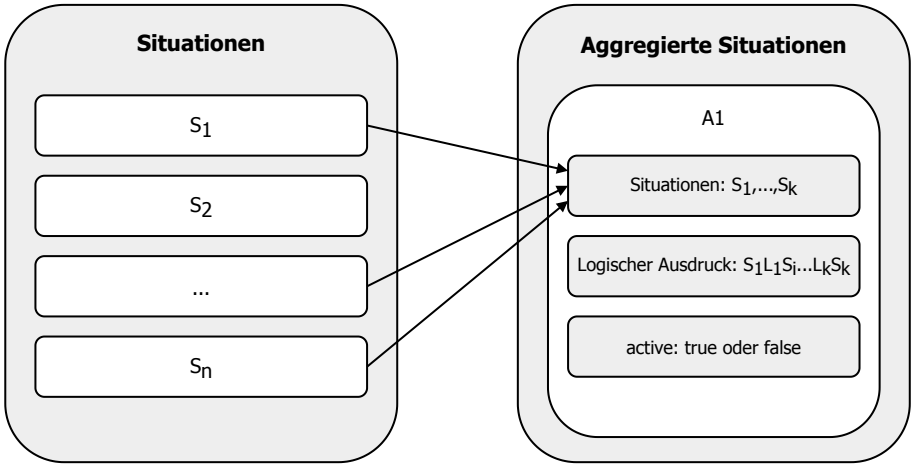

Abbildung 5.3: Konzept: Aggregierte Situationen

6 Implementierung

Dieses Kapitel beinhaltet die prototypische Implementierung des in Kapitel 5 beschriebenen Konzepts in die Benutzeroberfläche sowie den Container des OpenTOSCA Ökosystems. Zunächst wird auf die Verwaltung der Situationen und Situationstrigger in der Benutzeroberfläche eingegangen. Anschließend wird die Umsetzung der situationsbezogenen Ausführung von Plänen in der UI erläutert. Zum Schluss wird beschrieben, wie aggregierte Situationen im Container als auch in der Benutzeroberfläche realisiert werden.

6.1 Situationen

Wie in Abschnitt 4 beschrieben, ist es noch nicht möglich gewesen eine Situation in der Benutzeroberfläche zu erstellen. Durch die Implementierung ist dies nun möglich. In Abbildung 6.1 ist der neue Tab der Benutzeroberfläche zu sehen, welche zur situationsabhängigen Modellierung und Steuerung verwendet werden kann. Dabei lassen sich Situationen leicht in der UI erstellen und aktivieren. Die Struktur entspricht, der zuvor im XML-Dokument definierten Situationen. So werden in den Textfeldern die Elemente *ThingId*, *SituationTemplateId* sowie *active* eingegeben. Zudem wird die Texteingabe validiert, indem das *active* Attribut einer Situation entweder true oder false sein kann und weder die *SituationTemplateId* noch die *ThingId* leer sein dürfen. Durch Drücken des *Plus Buttons* wird eine @POST Anfrage an die API gestellt. Dabei erhält der Nutzer Feedback über das Ergebnis der @POST Anfrage. Auch das Löschen von einzelnen Situationen ist möglich, dafür muss der *Delete Button* in der entsprechenden Zeile angeklickt werden. Für das Löschen aller Situationen kann der *Delete Button* gedrückt werden. Auch das Aktivieren beziehungsweise Deaktivieren von Situationen ist in der UI möglich, indem der Haken angeklickt wird. Dies führt zu einer @PUT Anfrage, welche bei erfolgreicher Ausführung das *active* Attribut negiert. Bei fehlerhafter Anfrage erhält der Nutzer Feedback. Das Feedback erfolgt durch Angular Growl Messages um die Benutzerfreundlichkeit zu erhöhen. Mittels des *Refresh Buttons* wird eine @GET Anfrage an die API gestellt und die Tabelle mit den Situationen, die sich in der API befinden, gefüllt.

6.2 Situationstrigger

Beim Erstellen eines Situationstriggers in der UI werden neben den in Abschnitt 4.1.2 benötigten Angaben auch das Element *AggregatedSituation* erforderlich (siehe Listing 6.1). Die Parameter können dem Situationstrigger per Eingabe in die Textfelder zugewiesen werden. Ist die Eingabe in jedem Textfeld erfolgreich, wird der Situationstrigger mittels einer @POST Anfrage erstellt. Dabei werden die IDs der Situationen und aggregierten Situationen als durch Kommas getrennter String eingegeben. Hierbei wird überprüft, ob alle angegebenen Situationen beziehungsweise aggregierten Situationen in der API existieren. Für die Eingabeparameter wird eine @GET

Anfrage an die Plan Parameter gestellt, wobei zuerst die *CsarId*, der *InterfaceName* und dann der *OperationName* ausgewählt werden muss. Der Plan Parameter, der vom Nutzer ausgewählt wird, füllt die zwei Textfelder *Name* und *Type*. Der zugehörige *Value* muss jedoch vom Nutzer eingegeben werden. Anschließend kann durch Drücken des *Add Parameter Buttons* der Plan Parameter zur Liste der Eingabeparameter hinzugefügt werden. Zudem kann durch Anklicken des Eingabeparameters dieser abgeändert beziehungsweise entfernt werden. Die *onActivation* und *isSingleInstance* Elemente dürfen nur die Werte true und false annehmen. Die *CsarId* wird mittels einer Checkbox ausgewählt, die alle im Container installierten CSARs enthält. Die Checkbox für das *InterfaceName* stellt sicher, dass nur valide Interfaces ausgewählt werden können, wodurch die Fehleranfälligkeit von Eingaben reduziert wird. Der *OperationName* kann durch Eingabe in das entsprechende Textfeld festgelegt werden. Der Nutzer erhält insbesondere beim Abfragen der Plan Parameter Feedback, ob die eingegebene Operation existiert. Das Löschen von Situationstriggern erfolgt analog zu Situationen über den *Delete Button*. Wird eine Situation, die sich im Situationstrigger befindet gelöscht, wird der entsprechende Situationstrigger auch gelöscht. Die Funktionalität des *Refresh Buttons* ist analog zu dem in Abschnitt 6.1 erwähnten Button.

```
<?xml version="1.0" encoding="UTF-8"?>
<SituationTrigger>
    <Situations>
    </Situations>
    <AggregatedSituations>
        <AggregatedSituationId>4</AggregatedSituationId>
    </AggregatedSituations>
    <CsarId>MyTinyToDo_Bare_Docker.csar</CsarId>
    <onActivation>true</onActivation>
    <isSingleInstance>true</isSingleInstance>
    <InterfaceName>scaleout_dockercontainer</InterfaceName>
    <OperationName>scale-out</OperationName>
    <InputParameters>
        <InputParameter>
            <name>OpenTOSCAContainerAPIServiceInstanceURL</name>
            <Value>12</Value>
            <Type>String</Type>
        </InputParameter>
        <InputParameter>
            <name>ContainerSSHPort</name>
            <Value>9999</Value>
            <Type>String</Type>
        </InputParameter>
        <InputParameter>
            <name>ApplicationPort</name>
            <Value>9998</Value>
            <Type>String</Type>
        </InputParameter>
    </InputParameters>
</SituationTrigger>
```

Listing 6.1: Aufbau des neuen Situationstriggers

6.3 Situationsbezogene Ausführung von Plänen

Der *Play Button* ermöglicht es mittels des in Abschnitt 5 eingeführten Konzepts Build Pläne, Management Pläne und Termination Pläne einer Anwendungsinstanz einer CSAR situationsabhängig auszuführen. Zunächst wird überprüft, ob der *onActivation* Wert des Situationstriggers mit dem *active* Wert aller im Situationstrigger angegebenen Situationen übereinstimmt. Ist dies der Fall, wird eine @GET Anfrage an die API gestellt, ob die CSAR im Container installiert ist. Anschließend wird überprüft, ob die CSAR das Interface sowie die zugehörige Operation anbietet. Ist dies erfolgreich, wird der Plan zurückgegeben. An dieser Stelle werden die Eingabeparameter des Situationstriggers den für den Plan benötigten Parametern zugewiesen. Dabei wird zwischen einem Build Plan und den restlichen Planarten unterschieden. Diese benötigen im Vergleich zu einem Build Plan eine Referenz auf eine bereits existierende Anwendungsinstanz, die durch den Eingabeparameter *OpenTOSCAContainerAPIServiceInstanceURL* erfolgt.

6.4 Aggregierte Situationen

Das Aggregieren von Situationen ist ein neues Konzept um Situationen zusammenzufassen und um die Restriktion, dass alle Situationen im Situationstrigger UND-verknüpft sind, aufzulösen. Dabei ist in Listing 6.2 eine aggregierte Situation zu sehen, welche die Situationen *AuslastungHoch* und *AktiveKundenanzahlUeber500* ODER-verknüpft. Aggregierte Situationen bestehen aus einer nichtleeren Menge an Situationen, einem logischen Ausdruck und einem *active* Element. Der logische Ausdruck ermöglicht es Situationen beliebig oft durch eine ODER- beziehungsweise UND-Verknüpfung miteinander zu kombinieren. Das *active* Attribut hängt davon ab, ob der logische Ausdruck zu true oder false evaluiert und übernimmt dessen Ergebnis.

In Abbildung 6.1 ist zu sehen, wie aggregierte Situationen in der UI umgesetzt werden. Dabei können Situationen zusammengefasst werden, indem die Situation IDs in dem Textfeld als durch Kommas getrennter String übergeben werden. Der logische Ausdruck wird ebenfalls in einem Textfeld eingegeben, der aus den Operatoren || und && besteht. Dabei sind die Operatoren des logischen Ausdrucks linksassoziativ. Durch Betätigen des *Plus Buttons* wird eine @POST Anfrage an die API gestellt und der Nutzer erhält Feedback über das Ergebnis der @POST Anfrage. Allerdings muss beachtet werden, dass es bei der Nutzung von externen Anwendungen wie Postman zu einem Bad Request bei Eingabe von && kommen kann. Für diesen Fall kann der UND-Operator durch den Buchstaben A ersetzt werden.

Zudem muss beachtet werden, dass sich der Status einer aggregierten Situation durch eine @PUT Anfrage ändern kann. Dies ist beispielsweise für die aggregierte Situation in Listing 6.2 der Fall. Ist sowohl das *active* Element der Situation 2 und 3 *false*, so ist ebenfalls der *active* Wert der aggregierten Situation *false*. Wird Situation 2 *true*, dann wird aufgrund des logischen Ausdrucks der *active* Wert der aggregierten Situation ebenfalls true. Dies wird sichergestellt, indem bei jeder @PUT Anfrage überprüft wird, ob die Situation Teil einer aggregierten Situation ist. Stimmt dies zu, wird der logische Ausdruck der aggregierten Situation neu evaluiert. Der Zusammenhang zwischen aggregierten Situationen und Situationstriggern besteht darin, dass das *onActivation* Attribut nicht nur von den Wahrheitswerten der Situationen abhängt, sondern auch von den aggregierten Situationen. Dabei kann im Situationstrigger sowohl die Menge der aggregierten Situationen als auch die Menge der Situationen leer sein, aber nie beide gleichzeitig.

Außerdem ist es möglich aggregierte Situationen im Nachhinein zu verändern. Dabei können Situationen entfernt und hinzugefügt werden. Hierbei wird sichergestellt, dass beim Zurückschreiben in die Tabelle beziehungsweise beim @PUT Request nur gültige Werte übergeben werden. Dies löst auch die Problematik bei Situationstriggern, wenn festgestellt wird, dass eine Situation nicht mehr verwendet werden soll. Dabei muss nicht der Situationstrigger gelöscht werden, sondern lediglich die aggregierte Situation angepasst werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<AggregatedSituation>
    <Situations><SituationId>2</SituationId><SituationId>3</SituationId></Situations>
    <LogicExpression>2||3</LogicExpression>
</AggregatedSituation>
```

Listing 6.2: Aufbau einer aggregierten Situation

Abbildung 6.1: Die neue Benutzeroberfläche des OpenTOSCA Ökosystems

7 Zusammenfassung und Ausblick

Eine aktuelle Herausforderung für Anbieter von IoT Geräten ist die Wartung der Software-und Hardwarekomponenten in Privathaushalten und Firmen, deren Ursache auf die Heterogenität und der Anzahl der Komponenenten zurückzuführen ist. Dabei ist ein Ansatz vollautomatisierte Deployment- und Managementsysteme einzuführen. Jedoch gestaltet sich die Softwarekonfiguration aufgrund ihrer Abhängigkeit zur Umgebung als komplexer und aufwendiger Prozess, dessen Komplexität reduziert werden kann, indem nicht mehr die gesamte Umgebung und deren Zusammenwirken betrachtet wird, sondern Situationen als Abstraktion. Damit Situationen und deren Auswirkungen transparent sind, müssen diese für den Nutzer greifbar sein.

Diese Bachelorarbeit hat mehrere Konzepte zur situationsabhängigen Modellierung und Steuerung präsentiert. Dabei wurde die Problematik, dass weder die Verwaltung von Situationen und Situationstriggern noch die situationsabhängige Ausführung von Managementplänen in der UI möglich waren, gelöst. Dazu wurde eine neue Architektur zwischen dem Container und der UI präsentiert. Durch die Verbindung der Situation API mit der Benutzeroberfläche ist das Management von Situationen sowie von Situationstriggern möglich. Zusätzlich erweitert die UI die bisherige Ausführung von Plänen um das Situationsbewusstsein, wodurch es nun möglich ist, Anwendungsinstanzen einer CSAR situationsabhängig auszuführen. Außerdem wurde das Konzept der aggregierten Situationen eingeführt, welche die Gruppierung von Situationen sowie die Interpretation logischer Ausdrücke ermöglicht. Zudem erweitert dieses Konzept die bisherige Implementierung der situationsbewussten Ausführung, indem für einen Situationstrigger nicht mehr alle Situationen zutreffen müssen, damit dieser starten kann. Die Implementierung der Konzepte erfolgte in der Benutzeroberfläche und dem Container des OpenTOSCA Ökosystems. Die Strukturen der Situationen und Situationstrigger waren bereits im Container vorhanden, ebenfalls die situationsabhängige Ausführung von Management Plänen. Jedoch existierte keine Möglichkeit für den Nutzer diese Optionen in einer Benutzeroberfläche zu verwalten, dies ist nun aber möglich.

Diese Arbeit hat die UI um Situationen und Situationstrigger erweitert. Dabei könnte die Verbindung zwischen der UI und der Winery erweitert werden. Dies wäre insbesondere für die Modellierung im Topology Modeler interessant. Derzeit können zwar Situationen modelliert werden, jedoch gibt es noch keine Möglichkeit dem Situation Node Type beispielsweise beim Herausziehen aus der Palette den Namen einer Situation in der API automatisch zuzuweisen beziehungsweise auszuwählen. Außerdem könnte ein neuer Node Type *AggregierteSituation* erstellt werden, welcher mit den aggregierten Situationen aus der UI kombiniert werden kann.

Literaturverzeichnis

[ADB99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. „Towards a Better Understanding of Context and Context-Awareness“. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. HUC '99. Karlsruhe, Germany: Springer-Verlag, 1999, S. 304–307. DOI: 10.1007/3-540-48157-5_29. URL: https://doi.org/10.1007/3-540-48157-5_29 (zitiert auf S. 25).

[And15] C. Anderson. „Docker [Software engineering]“. In: *IEEE Software* 32 (Mai 2015), S. 102–c3. DOI: 10.1109/MS.2015.62 (zitiert auf S. 23).

[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*. 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 (zitiert auf S. 20, 22).

[BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, D. Schumm. „Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA“. In: *OTM 2012, Part I*. Bd. 7565. Lecture Notes in Computer Science (LNCS). Springer-Verlag, 2012, S. 416–424. DOI: 10.1007/978-3-642-33606-5_25. URL: http://dx.doi.org/10.1007/978-3-642-33606-5_25 (zitiert auf S. 20).

[BBK+14] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, M. Wieland. „Context-Aware Cloud Application Management“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, 2014, S. 499–509 (zitiert auf S. 28).

[BBK14] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. *Vinothek - A Self-Service Portal for TOSCA*. März 2014 (zitiert auf S. 21, 22).

[BBKL14] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. „TOSCA: Portable Automated Deployment and Management of Cloud Applications“. In: *Advanced Web Services*. Hrsg. von A. Bouguettaya, Q. Z. Sheng, F. Daniel. New York, NY: Springer New York, 2014, S. 527–549. ISBN: 978-1-4614-7535-4. DOI: 10.1007/978-1-4614-7535-4_22. URL: https://doi.org/10.1007/978-1-4614-7535-4_22 (zitiert auf S. 19, 33).

[BEK+16] U. Breitenbücher, C. Endres, K. Képes, O. Kopp, F. Leymann, S. Wagner, J. Wettinger, M. Zimmermann. „The OpenTOSCA Ecosystem - Concepts & Tools“. In: *European Space project on Smart Systems, Big Data, Future Internet - Towards Serving the Grand Societal Challenges - Volume 1: EPS Rome 2016,* INSTICC. SciTePress, 2016, S. 112–130. ISBN: 978-989-758-207-3. DOI: 10.5220/0007903201120130 (zitiert auf S. 20, 21).

[BHK+15] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. „A Situation-Aware Workflow Modelling Extension“. Englisch. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015)*. ACM, Dezember 2015, S. 478–484. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-57&engl=0 (zitiert auf S. 29, 30, 38).

[BP17] F. Batarseh, J. Pithadia. „Context-Aware User Interfaces for Intelligent Emergency Applications“. In: Mai 2017, S. 359–369. ISBN: 978-3-319-57836-1. DOI: 10.1007/978-3-319-57837-8_29 (zitiert auf S. 26).

[BRK] Bitkom Research GmbH und KPMG AG. *Cloud-Monitor 2020*. URL: https://www.bitkom.org/sites/default/files/2020-06/prasentation_bitkom_kpmg_pk-cloud-monitor.pdf (zitiert auf S. 13, 28).

[DDF+06] S. Dobson, S. Denazis, A. Fernández, D. Gäiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli. „A Survey of Autonomic Communications“. In: *ACM Trans. Auton. Adapt. Syst.* 1.2 (Dez. 2006), S. 223–259. ISSN: 1556-4665. DOI: 10.1145/1186778.1186782. URL: https://doi.org/10.1145/1186778.1186782 (zitiert auf S. 26).

[Doc] Docker Inc. *What is a Container?* URL: https://www.docker.com/resources/what-container (zitiert auf S. 23).

[DPA19] Deutsche Presse-Agentur. *Jeder dritte Deutsche nutzt Sprachassistenten*. 2019. URL: https://www.faz.net/aktuell/wirtschaft/digitec/jeder-dritte-deutsche-nutzt-sprachassistenten-16229830.html (zitiert auf S. 13, 17).

[FBH+17] A. C. Franco da Silva, U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, B. Mitschang, R. Steinke. „Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments“. In: Jan. 2017, S. 358–367. DOI: 10.5220/0006243303580367 (zitiert auf S. 17).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns*. Jan. 2014. DOI: 10.1007/978-3-7091-1568-8 (zitiert auf S. 18).

[GBF+18] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, L. Reinfurt. „A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences“. In: *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Springer, 2018, S. 81–101. DOI: 10.1007/978-981-10-5861-5_4 (zitiert auf S. 17, 18).

[GBH05] M. Grossmann, M. Bauer, N. Honle, U.-P. Kappeler, D. Nicklas, T. Schwarz. „Efficiently Managing Context Information for Large-Scale Scenarios“. In: Bd. 2005. Apr. 2005, S. 331–340. ISBN: 0-7695-2299-8. DOI: 10.1109/PERCOM.2005.17 (zitiert auf S. 14, 29).

[IDC16] Pressebox and IDC. *Viel Luft nach Oben: Deutsche Unternehmen vergeben Chance, ihre digitale Transformation mit IoT voranzutreiben*. Dez. 2016. URL: https://www.pressebox.de/pressemitteilung/idc-central-europe-gmbh/Viel-Luft-nach-Oben-Deutsche-Unternehmen-vergeben-Chance-ihre-digitale-Transformation-mit-IoT-voranzutreiben/boxid/829243 (zitiert auf S. 17).

[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. 20).

[KBL19] K. Képes, U. Breitenbücher, F. Leymann. „Situation-Aware Management of Cyber-Physical Systems“. In: *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*. SciTePress, Mai 2019, S. 551–560. ISBN: 978-989-758-365-0. DOI: 10.5220/0007799505510560 (zitiert auf S. 30).

[KKR+13] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, R. Tonjes. „Survey of Context Provisioning Middleware“. In: *IEEE Communications Surveys and Tutorials* 15.3 (2013), S. 1492–1519. DOI: 10.1109/SURV.2013.010413.00207 (zitiert auf S. 25, 26).

[LCG+09] R. Lange, N. Cipriani, L. Geiger, M. Großmann, H. Weinschrott, A. Brodt, M. Wieland, S. Rizou, K. Rothermel. „Making the World Wide Space Happen: New Challenges for the Nexus Context Platform“. In: Apr. 2009, S. 1–4. DOI: 10.1109/PERCOM.2009.4912782 (zitiert auf S. 14, 29).

[Ley11] F. Leymann. „Cloud Computing“. In: *In: Proc. 52th Photogrammetric Week, pp. 1-10* 53 (Juli 2011). DOI: 10.1524/itit.2011.9070 (zitiert auf S. 18).

[LFWW16] F. Leymann, C. Fehling, S. Wagner, J. Wettinger. „Native Cloud Applications: Why Virtual Machines, Images and Containers Miss the Point!“ In: *Proceedings of the 6th International Conference on Cloud Computing and Service Science (CLOSER 2016)*. SciTePress, 2016, S. 7–15 (zitiert auf S. 18).

[LVCD13] F. Li, M. Vögler, M. Claeßens, S. Dustdar. „Towards Automated IoT Application Deployment by a Cloud-Based Approach“. In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. 2013, S. 61–68. DOI: 10.1109/SOCA.2013.12 (zitiert auf S. 13).

[MG11] P. M. Mell, T. Grance. *The NIST Definition of Cloud Computing*. Techn. Ber. Gaithersburg, MD, USA, 2011 (zitiert auf S. 18).

[Nie20] F. Nieuwenhuizen. „Time-sensitive Deployment and Management for Cyber-Physical Systems using TOSCA“. Masterarbeit. Universität Stuttgart, 2020. DOI: 10.18419/opus-10710. URL: http://dx.doi.org/10.18419/opus-10710 (zitiert auf S. 31).

[OA13] OASIS. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. Nov. 2013. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (zitiert auf S. 20).

[OS] OpenJS Foundation. *About Node.js®*. URL: https://nodejs.org/en/about/ (zitiert auf S. 22).

[RBF08] M. Rosemann, J. Recker, C. Flender. „Contextualization of Business Processes“. In: *International Journal of Business Process Integration and Management* 3 (Juli 2008). DOI: 10.1504/IJBPIM.2008.019347 (zitiert auf S. 27).

[RH06] H. Reimer. „BSI Studie: „Pervasive Computing: Entwicklungen und Auswirkungen““. In: *Datenschutz und Datensicherheit - DuD* 30.11 (Nov. 2006), S. 748–748. ISSN: 1862-2607. DOI: 10.1007/s11623-006-0212-4. URL: https://doi.org/10.1007/s11623-006-0212-4 (zitiert auf S. 25).

[SHWM16] A. C. F. da Silva, P. Hirmer, M. Wieland, B. Mitschang. „SitRS XT - Towards Near Real Time Situation Recognition“. In: *J. Inf. Data Manag.* 7 (2016), S. 4–17 (zitiert auf S. 13).

[VRC09] L. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner. „A Break in the Clouds: Towards a Cloud Definition“. In: *Computer Communication Review* 39 (Jan. 2009), S. 50–55. DOI: 10.1145/1496091.1496100 (zitiert auf S. 18).

[WHF92] R. Want, A. Hopper, V. Falcão, J. Gibbons. „The Active Badge Location System“. In: *ACM Trans. Inf. Syst.* 10.1 (Jan. 1992), S. 91–102. ISSN: 1046-8188. DOI: 10.1145/128756.128759. URL: https://doi.org/10.1145/128756.128759 (zitiert auf S. 25).

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, F. Leymann. „Towards Context-Aware Workflows“. In: *Pernici, Barbara (ed.); Gulla, Jon Atle (ed.): CAiSE´07 Proceedings of the Workshops and Doctoral Consortium Vol.2, Trondheim, Norway, June 11-15th, 2007* (Juni 2007) (zitiert auf S. 27).

[WSB15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. „Towards Situation-Aware Adaptive Workflows: SitOPT — A General Purpose Situation-Aware Workflow Management System“. In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. März 2015, S. 32–37. DOI: 10.1109/PERCOMW.2015.7133989 (zitiert auf S. 13, 28–30).

[ZHK09] O. Zweigle, K. Häussermann, U.-P. Käppeler, P. Levi. „Supervised Learning Algorithm for Automatic Adaption of Situation Templates Using Uncertain Data“. In: *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ICIS '09. Seoul, Korea: Association for Computing Machinery, 2009, S. 197–200. ISBN: 9781605587103. DOI: 10.1145/1655925.1655960. URL: https://doi.org/10.1145/1655925.1655960 (zitiert auf S. 34).

Alle URLs wurden zuletzt am 12. 08. 2020 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