Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Verteilte Situationserkennung

Naumov Andriy

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf., Mathias Mormul
Beginn am:	15. Juni 2017
Beendet am:	15. Dezember 2017
CR-Nummer:	H.2.1, H.2.4, D.2.2, D.2.11

Kurzfassung

Das Interesse an Internet der Dinge (IoT) steigt durch die wachsende Anzahl der physischen Objekte (Things), die miteinander vernetzbar sind. Die dadurch entstehenden, intelligenten Netzwerke erfassen mittels Sensoren die Änderungen der dynamischen Umgebung. Durch Verarbeitung der Sensordaten können Aussagen über die Situation gemacht werden, in welcher sich ein Thing befindet. Auf diese Art kann beispielsweise ein Fehlverhalten des Objekts festgestellt werden. Sobald eine vordefinierte Situation erkannt wurde, sind entsprechende, vorgesehene Reaktionsmaßnahmen durchführbar.

Das Forschungsprojekt SitOPT beschäftigt sich mit der dynamischen Adaption von situationsbezogenen Anwendungen an die Umgebung. Dafür wird ein General-Purpose-System entwickelt, in welchem die Situationserkennung eine wichtige Rolle spielt. Das Thema dieser Arbeit ist die Erweiterung der Situationserkennung im Rahmen von SitOPT durch Verteilung. Dabei wird das Ziel verfolgt, die Vorteile von Edge Computing, wie Verringerung der Latenz und des Datenverkehrs in die Cloud, zu nutzen. Die Konzeption der verteilten Situationserkennung wird entwickelt und prototypisch implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Durch das Paradigma von Internet der Dinge (IoT) entsteht das Potential intelligente Umgebungen zu realisieren. SMART Home, SMART Factory, SMART Mobility sind Beispiele für Systeme, welche die Vernetzung von Sensoren und Aktoren ausnutzen. Derartige intelligente Umgebungen modernisieren Geschäfts- und Gesellschaftsprozesse, indem sie den Grad der erforderlichen, menschlichen Beteiligung verringern. Am Beispiel von SMART Factory lassen sich die Vorteile von IoT innerhalb der Industrie 4.0 veranschaulichen. Viele Ereignisse im Laufe der industriellen Produktion können maschinell erfasst werden, wenn die Fabrik mit Sensoren ausgestattet wird. Die Sensoren der Maschinen einer intelligenten Fabrik können die Informationen über deren Umgebung sammeln. Auf die Ereignisse in der Umgebung können die Aktoren reagieren. Jedoch, um die Ereignisse oder Situationen in der Umgebung zu erkennen, muss die Sensor-Information ausgewertet werden. Manuelle Datenauswertung kommt nicht in Frage, denn die Anzahl der Sensoren, die für ausreichende Umgebungserfassung notwendig sind, ist zu hoch. Das Volumen an produzierten Sensordaten bedarf einer rechnergestützten Auswertung.

Um die Autonomie und Intelligenz einer Umgebung zu gewährleisten, ist eine Technologie nötig, welche Informations- und Kommunikationstechnik umfasst. Zur Kommunikation der Things, welche physische Objekte sind (Machine-to-Machine Kommunikation), müssen deren ausreichend detaillierte digitale Entsprechungen erstellt werden. Ein System, in welchem die Things virtuell repräsentiert sind heißt cyber-physisch. In einem Cyber-Physischen-System (CPS) werden können die Things einen Informationsfluss aufbauen.

Ein autonomes System schließt eine Infrastruktur für die Erfassung, Kommunikation und Analyse der Sensordaten, sowie Ableitung der Situationen und Reaktionsroutinen ein. Um diese angestrebte Systeminfrastruktur zu realisieren, können die Vorteile von Cloud Computing genutzt werden. Dabei dient die Cloud-Komponente als eine Datenverarbeitungs- und Verwaltungszentrale. Ein zentral verwaltetes System kann suboptimales Verhalten in Hinblick auf Kommunikationswege haben: Alle Daten müssen durch die Zentrale durchgehen. Durch Einsatz von Edge Computing kann Verteilung der Datenverarbeitung vorgenommen werden. Edge Computing in Kombination mit IoT, Industrie 4.0 und Cloud Computing kann zum Entwurf autonomer intelligenter und verteilter Systeme genutzt werden.

1.1. Problembeschreibung

Das Ausmaß an Sensordaten, die die Dinge zur Verfügung stellen steigt kontinuierlich. Für ein System mit zentralisierter Cloudverwaltung hat es problematische Auswirkungen: Datenverkehr in die Cloud wird erhöht. Wenn die Ableitung von Situationen (aus der Menge von Sensordaten) in der Cloud stattfindet, entsteht zusätzlich die Problematik der erhöhten Latenz der Situationserkennung für die Anwendungen, welche auf die erkannten Situationen warten und nicht auf

der Cloud-Instanz, sondern auf lokalen Netzwerkknoten laufen. Die Antwortzeit vergrößert sich dabei, denn zwischen Cloud und Edge besteht eine Netzwerk-Latenz.

Ein weiteres Problemfeld sind die Sicherheitsbedenken, die mit der Nutzung von Cloud verbunden sind. Die Daten, die sensibel sind, müssen durch das Netzwerk verschickt werden. Um die Datensicherheit zu erhöhen muss ein hohe Aufwand zur Verschlüsselung betreiben werden, was nicht immer (z.B. wegen Ressourcenknappheit) möglich ist.

SitOPT [Sit] ist ein Forschungsvorhaben der DFG (Deutschen Forschungsgemeinschaft). Es beschäftigt sich mit dem Entwurf einer IoT-Architektur. Das SitOPT Projekt setzt Erarbeitung von Methoden, die es Applikationen erlauben autonom zu sein und sich der Umgebung anzupassen, als Ziel. Die SitOPT-Architektur basiert auf der Nutzung von Cloud-Services. Die Dinge (Things) erkennen ihre Umgebung durch Sensoren und leiten diese Information weiter an die Cloud-Zentrale. Dadurch entsteht ein starker Datenstrom, welcher die verfügbare Bandbreitenkapazität belasten kann. In der Cloud wird die gesammelte Information gespeichert und verwaltet: Über die Cloud können Applikationen auf die Sensordaten zugreifen. Es kann vorkommen, dass die Applikationen, welche die Sensordaten analysieren und anschließend benutzen wollen, eine kürzere (lokale) Netzverbindung zu den Things haben, die die Daten bereitgestellt haben. In diesem Fall ist der Zugriff auf die Daten über die Cloud nicht optimal, denn die Netzwerklatenz sich auf die Antwortzeit der Applikationen auswirkt.

Diese Problematik, wird durch das Edge Computing Paradigma adressiert. Als Edge-Nodes werden die Knoten im Netzwerk bezeichnet, über welche die Things mit der Cloud und miteinander kommunizieren (z.B. ein lokaler Server). Durch Verteilung der Datenverarbeitung auf Edge und Cloud kann die Latenzzeit für lokale (auf demselben Edge-Node laufende) Anwendungen verringert werden. Diese Arbeit beschreibt die Erzeugung eines Prototyps von einem autonomen System, welches die Idee der Verteilten Situationserkennung umsetzt. Die Auswirkungen der Verteilung auf hohe Latenzzeit, hohen Datenverkehr und Sicherheitsbedenken werden untersucht.

1.2. Motivierendes Beispiel

Um den Nutzen von Verteilung in Situationserkennung zu verdeutlichen wird ein hypothetisches Beispiel aus dem Industrie-Umfeld (SMART Factory) präsentiert. Eine Fabrik besteht aus mehreren Fabrikgebäuden. In einem Fabrikgebäude befinden sich mehrere Roboter. Diese Roboter besitzen Sensoren, mit welchen sie die Temperatur der Umgebung erfassen können.

Eine Situation (*Situation Nr. 1*) mit dem Namen *Roboter-Überhitzt*, die eine Roboter-Überhitzung darstellt, muss erkannt werden. Die gesammelten Sensordaten werden in die Cloud gesendet. In der Cloud werden die Sensordaten gespeichert und in einem Situationserkennungssystem analysiert. Die Analyse erfolgt mit Hilfe von einem *Situationstemplate*, das durch einen Roboter-Experten entworfen wurde. Wenn die Sensordaten, die in dem Situationstemplate definierte Threshold (z.B. 70°C) überschreiten, wird die Überhitzung erkannt.

In dem Fabrikgebäude befindet sich ein lokaler Server welcher die Sensordaten der Roboter erhält und in die Cloud weiterleitet. Auf diesem Server läuft eine Anwendung - *Kühlung eines Roboters*, welche auf die erkannte Überhitzung-Situation wartet, um diese zu verarbeiten. Die Daten der erkannten Situation befinden sich in der Cloud. Das bedeutet, dass für Kühlung-Anwendung die Latenz der Situationserkennung hoch ist, denn ein weiterer Zugriff auf die Cloud muss erfolgen.

Nachdem die Anwendung die erkannte Situation aus der Cloud abgerufen hat, kann sie ausgeführt werden.

Eine weitere Situation (*Situation Nr. 2*) schließt die Information über mehrere Roboter ein: *Mehrere-Roboter-Überhitzt* mit jeweils derselben Treshold. Diese Situation entsteht aus der Analyse der Sensordaten von mehreren Robotern. Diese Sensordaten müssen miteinander (mit einer logischen UND-Operation) verbunden werden.

Eine weitere, komplexere Situation (*Situation Nr. 3*) erkennt eine Situation, die sich aus mehreren Situationen zusammensetzt: z.B. Zusammensetzung der *Situation Nr. 2* aus unterschiedlichen Fabrikgebäuden. Diese Situation kann "Fehler in Produktionslinie"heißen und tritt ein und wird erkannt, wenn die Situation Nr. 2 in mehreren Fabrikgebäuden erkannt wird.

Das beschriebene Beispiel zeigt, dass für die Erkennung von Situationen nur ein Thing (Nr.1), mehrere Things (Nr.2) oder Eintritt von mehreren kleineren Situationen (Nr.3), überwacht werden können. Dadurch, dass eine größere Situation aus vielen kleineren Situationen bestehen darf, kann Vermutung gemacht werden, das ein Situationserkennung sich auf mehrere Netzwerkknoten verteilen lässt.

Des Weiteren lässt sich erkennen, dass die Verwendung von einer zentralen Cloud-Instanz von Nachteil sein kann. Der lokale Server kann die Situationserkennung nicht durchführen sondern braucht Verarbeitung seitens der Cloud-Zentrale. Dadurch kann die lokale Anwendung *Kühlung eines Roboters* nicht den Vorteil der Nähe (Lokalität) zu dem Server ausnutzen.

Das beschriebene Szenario gibt nur einen oberflächlichen Einblick auf die Problematik. In der realen Welt müssen mehrere weitere Aspekte beachtet werden: unterschiedliche Typen der Roboter (Things), Datenformate/Messeinheiten etc. Das Verhalten der autonomen und verteilten Situationserkennung in einem komplexen System mit einer Vielzahl (z.B. Tausende) von Knoten stellt neue Herausforderungen (z.B. eindeutige Benennung, Vermeidung von Redundanzen etc.). In der vorliegenden Arbeit (im Rahmen von SitOPT Projekt) werden Ziele gesetzt, um auf die Herausforderungen der verteilten Situationserkennung einzugehen.

1.3. Aufgabenstellung

Diese Arbeit beschäftigt sich mit dem SitOPT Projekt und baut auf bereits entworfenen Teilen der Architektur auf. Die Ziele von SitOPT müssen demnach verfolgt werden. Für diese Thesis gilt insbesondere das Ziel eine effiziente Infrastruktur zur Situationserkennung zu entwerfen. Laut Anforderungen von SitOPT [Sit], müssen die Situationen aus der dynamischen Umgebung basierend auf einem Situationstemplate erkannt werden.

Das global definierte Ziel von SitOPT gilt in dieser Arbeit umzusetzen bzw. eine vorhandene Umsetzung ist anzupassen und zu modifizieren. Die übergeordnete Aufgabe besteht darin, ein verteiltes Situationserkennungssystem zu entwerfen. Dabei basiert die Erkennung auf Situationstemplates, welche die Art der zu erkennenden Situation beschreiben. Die Unteraufgaben dieser Arbeit lassen sich wie folgt definieren:

1. Die erste Unteraufgabe besteht darin, eine Anpassung des Schemas der Situationstemplates vorzunehmen: Situationen sollen andere Situationen als Input (Eingabe) verwenden.

Dadurch können Relationen zwischen mehreren Situationstemplates entstehen. Diese Verknüpfung erlaubt Modellierung komplexer Situationstemplates und damit auch komplexer Situationen.

2. Die zweite Unteraufgabe stützt auf SitOPTs Anforderung benutzerfreundlich zu sein. Dafür wurde eine grafische Modellierungsumgebung erstellt. Im Rahmen der zweiten Unteraufgabe muss das Modellierungswerkzeug erweitert werden, um die Eigenschaften von dem neuen Schema (siehe erste Unteraufgabe) abzubilden.
3. Die dritte Unteraufgabe ist eine logische Fortsetzung der Schema-Erweiterung: Modellierung und Analyse der Verknüpfungen von mehreren Situationstemplates. Die Infrastruktur für die Realisierung der Relationen von Situationstemplates muss bereitgestellt werden, um die Situationserkennung, die auf mehreren Situationstemplates basiert, zu ermöglichen.
4. Als vierte Aufgabe wird Einführung der verteilten Situationserkennung definiert. Dabei handelt es sich um Verteilung zwischen Edge und Cloud. Ein Konzept der Verteilung basierend auf dem neuen Schema der Situationstemplates muss entworfen werden. Die Verteilung muss in Abhängigkeit von beteiligten Things (die die Sensordaten bereitstellen) erfolgen.
5. Die fünfte Aufgabe besteht darin, ein Medium für die Kommunikation der erkannten Situationen bereitzustellen. Die Kommunikationsarten dabei sind: Edge-Edge, Edge-Cloud, Cloud-Edge, Cloud-Cloud. Dabei ist eine lose Kopplung zwischen Kommunikationspartnern von Vorteil. Die Situationen müssen eindeutig identifizierbar sein, obwohl das gleiche Situationstemplate für verschiedene Things eingesetzt werden kann.

Gliederung

Diese Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen und verwandte Arbeiten: Hier werden die theoretischen Grundlagen beschrieben, zentrale Problemstellungen und Lösungsansätze präsentiert sowie verwandte Arbeiten angesprochen. Das zugrundeliegende Projekt SitOPT und die Ausgangsarbeiten für diese Thesis werden in Detail beschrieben.

Kapitel 3 – Technischer Hintergrund: Hier werden die theoretischen Grundlagen, Methoden und Technologien angesprochen, die im Rahmen dieser Arbeit verwendet werden.

Kapitel 4 – Konzept: In diesem Kapitel wird das Konzept vorgestellt, welches zur Lösung der gesetzten Ziele entworfen wurde.

Kapitel 5 – Implementierung: Dieses Kapitel befasst sich mit der Realisierung der Methode, die im Konzept entworfen wurden. Es wird auf die technischen Details der Implementierung eingegangen.

Kapitel 6 – Evaluation: Hier wird überprüft, ob die Ziele der Arbeit erreicht wurden.

Kapitel 7 – Zusammenfassung und Ausblick: Das letzte Kapitel fasst die Ergebnisse der Arbeit zusammen und stellt Problemstellungen für zukünftige Arbeiten vor.

2. Grundlagen und verwandte Arbeiten

Dieses Kapitel fasst die grundlegenden Definitionen und Begriffe zusammen, die für das Verständnis des folgenden Textes essentiell sind. Es werden die Arbeiten aus dem Ausgangsprojekt SitOPT, sowie Arbeiten, die verwandte Themengebiete betreffen, beschrieben.

2.1. Grundlegende Definitionen

Die zentralen Begriffe dieser Arbeit sind Thing, Kontext und Situation Template. Zusätzlich wird auch der Workflow-Begriff erläutert, um die Ziele der Ausgangsarbeit und grundsätzlicher SitOPT-Vision klarzustellen.

2.1.1. Things

Architektur eines Internet of Things Systems beschreibt eine Infrastruktur, in der physische Objekte miteinander vernetzt sind. Zwischen diesen Objekten erfolgt Datenaustausch. Ein derartiges physisches Objekt, ein Thing, ist fähig Daten zu produzieren. Ein Thing produziert Daten über Sensoren. Diese Daten können gesammelt und gespeichert werden. Je mehr Daten gesammelt wurden, desto genauer kann für ein physisches Objekt eine virtuelles Repräsentation erstellt werden. Die virtuellen Objekte bilden in Verbindung mit physischen Objekten cyber-physische Objekte, aus welchen cyber-physisches Systeme (CPS) [BG11] zusammengesetzt werden. In einem CPS werden die Ereignisse um ein physisches Objekt auf seinem digitalen Pendant (Digital Twin [GS12]) virtuell reproduziert/simuliert, indem die Sensordaten analysiert und verwaltet werden. In einem IoT-System haben die cyber-physischen Objekte eine Möglichkeit miteinander zu kommunizieren.

Im Zusammenhang mit Situationserkennung sind Things, die Objekte, welche überwacht werden, um den Eintritt von verschiedenen, zuvor modellierten Situationen festzustellen. Die Situationen auf Things triggern vordefinierte Reaktionsmechanismen. Diese Reaktionsmechanismen können die Anwendungen sein, die für die Bearbeitung der erkannten Situationen zuständig sind. Die Handhabung der erkannten Situationen beschreibt reagierende Maßnahmen, die das System zu einem Soll-Zustand bringen. Die Reaktionsmaßnahmen können die Things betreffen, durch welche die Situation entstanden ist. Daraus ergibt sich ein autonomes intelligentes System. Zunächst produzieren die Sensoren von einem Thing die Daten. Danach werden diese Daten für weitere Verarbeitung an die Situationserkennungskomponente versendet. Die Daten werden dort analysiert und eine Situation wird erkannt. Beim Erkennen startet ein Situationshandler die Maßnahmen, die das Thing beeinflussen, das die Sensordaten bereitstellte. Auf diese Weise, kann eine selbstorganisierte intelligente Infrastruktur für unterschiedliche Domänen (SMART Home, SMART Factory, SMART City etc.) entstehen. Je nach Domäne variieren die Einsatzszenarien

und dementsprechend die zu erkennenden Situationen. Die Aufgabe eines Domänenexperten besteht darin, die Situation hinreichend genau zu modellieren, um alle relevanten Eigenschaften der physischen Objekte zu berücksichtigen.

2.1.2. Kontext und Sensordaten

Im Bereich der Anwendungen kann sich die Bedeutung von Kontext unterscheiden. Informationen über Objekte/Personen, Standorte, Umgebung, Zeit, Jahreszeit, Temperatur etc. können als Kontext zusammengefasst werden [ADOB98; RPM99; ST94]. Dey, Abowd et al. [ADB+99] definieren Kontext als jede Art von Information, die dazu genutzt werden kann, die Situation, in der sich eine Entität befindet, zu charakterisieren. Eine Entität ist eine Person, ein Ort oder ein Objekt, welches als relevant für die Interaktion zwischen einem Benutzer und einer Anwendung angesehen wird, einschließlich des Benutzers und der Anwendung selbst. Demnach können Sensordaten als Kontext für die Entität eines physischen Objekts (Thing) gesehen werden. Daher wird eine Situation durch die semantische Bedeutung der Sensordaten charakterisiert. Denn die Daten, welche von Sensoren der physischen Objekte erfasst werden, erlauben es dem Beobachter, Schlussfolgerungen über den Zustand dieser Objekte zu ziehen. Aus den Informationen über den Zustand von einem oder mehreren Objekten ergibt sich eine Situation.

Im Rahmen dieser Arbeit sind die Begriffe Sensordaten und Kontext sehr eng miteinander verbunden. Dennoch können diese Begriffe nicht gleichgesetzt werden. Der Kontext einer Situation ist nicht immer direkt mit Sensordaten verbunden. Der Kontext beschreibt mehr als nur einen gemessenen Wert sondern enthält noch eine zusätzliche Information in welcher Relation, der Wert zu anderen Werten steht. Zum Beispiel können Kontextdaten Zeitstempel enthalten.

Kontext lässt sich hinsichtlich unterschiedlicher Aspekte klassifizieren. Man unterscheidet zwischen statischem und dynamischem Kontext. Der statische Kontext gibt konstante Daten an. Beispielsweise, ist die Bezeichnung von einem Roboter-Modell *Schweißroboter Nr. 3000*. Diese Information ändert sich nicht und wird bei der Modellierung einer Situation gebraucht, wenn das konkrete Roboter-Modell eingesetzt wird. Der dynamische Kontext beschreibt die Kontextdaten, die sich (unterschiedlich) häufig ändern können. Die Sensordaten eines *Schweißroboters Nr. 3000* definieren seinen dynamischen Kontext. Die Sensordaten ändern sich, sofern der Roboter in Betrieb ist, und entsprechend der Sensorwerte kann der aktuelle Zustand des Roboters ermittelt werden.

Ein weiteres Klassifizierungsmerkmal ist die Qualität der Kontextdaten. Die Übertragung von Sensordaten kann mit Fehlern verbunden werden. Es gibt unterschiedliche Ursachen für Fehler, z.B. Abweichungen des Messsystems, Störungen bei der Übertragung oder Ableitung der Kontextdaten (entstehen bei Operation mit mehreren abweichenden Daten - von mehreren Sensoren). Die Genauigkeit der Angaben über die Qualität des Kontextes kann bei der Auswertung benötigt werden, um die mögliche Rate der Abweichung festzustellen. Für den Kontext, der sich als unsicher erweist, können spezielle zuvor vorgesehenen Maßnahmen in Frage kommen. Es existieren mehrere Kriterien der Kontext-Qualität wie Aktualität, Genauigkeit, Zuverlässigkeit etc (siehe [MTD08; WKL+09]).

2.1.3. Situation

Der Begriff Situation in der allgemeinen Bedeutung beschreibt die aktuelle Lage oder Zustand, in dem sich etwas befindet. In kontextbasierten Anwendungen beschreibt eine Situation den Zustand von einem oder mehreren physischen Objekten. Der Zustand eines Objekts wird durch seine Kontextdaten (z.B. durch Sensordaten) festgelegt. Jedoch sind nicht alle Zustände der Objekte wichtig und ergeben aussagekräftige Situationen. So ist ein Zustand, der nicht weiteres aussagt, dass keine Änderungen eingetreten sind, nicht kritisch für die weitere Funktionalität des Systems. Dieser Zustand kann auch durch Abwesenheit von Fehlern erkannt werden. Die Erkennung der Fehlerzustände ist hingegen sehr wichtig und kann die Arbeit des Systems beeinträchtigen. Für bestimmte Anwendungsszenarien müssen Domänen-Experten die Situationen definieren, welche für die Arbeit der gesamten Anwendung wichtig oder sogar kritisch sind. Darüber hinaus, wenn eine intelligente autonome Anwendung erstellt wird, müssen durch Erkennung von Situationen Reaktionsmechanismen aktiviert werden. Um auf eine Situation reagieren zu können müssen nicht nur die Kontextdaten, sondern auch die Randbedingungen wie, Zeit, Objekt-ID (Thing-ID) etc. angegeben werden. Dafür muss eine Datenstruktur (Situationsobjekt) erstellt werden, die eine Instanz der Entität Situation eindeutig identifiziert und alle relevanten Kontextdaten enthält.

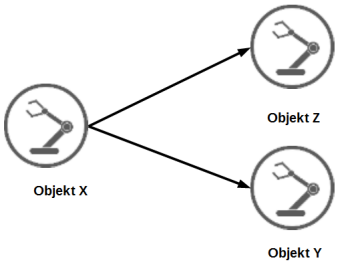

Abbildung 2.1.: Beispielhafte Anordnung von Things

Eine beispielhafte Anordnung von physischen Objekten ist in Abbildung 2.1 zu sehen. Die dargestellten Things sind z.B. die Roboter in einer Fabrik. Die Ausgangsdaten von Thing X sind die notwendigen Eingangsdaten für weitere Arbeit der Things Y, Z. Der Ausfall von X (z.B. durch Überhitzung) führt in einem nicht selbstorganisierten System dazu, dass Y und Z nicht ohne einen Eingriff auf das System von außen weiterarbeiten können. Im Gegensatz dazu wird in einem intelligenten System das Problem erkannt. Es wird ein Situationsobjekt mit der Information über die Beteiligten Things, den Eintrittszeitpunkt, sowie den auslösende Kontextdaten erstellt. Das Situationsobjekt wird an einen Situationshandler weitergeleitet. Dadurch, werden nach der Identifizierung der Problem-Situation Reaktionsmaßnahmen durchgeführt: Objekt X wird repariert bzw. ersetzt o.Ä. Um den Eintritt einer Problem-Situation festzustellen, müssen alle involvierten Objekte über entsprechende Sensorik verfügen. Durch die Interpretation von Sensordaten kann das Eintreten bzw. Nichteintreten einer Situation nachgewiesen werden. Zwar kann beliebige Kontextdaten als eine Situation definiert werden, jedoch ist oft sinnvoll viele Kontextdaten zu kombinieren und daraus hinsichtlich der Funktionalität der gesamten Anwendung aussagekräftigere Situationen abzuleiten. So ist die Information, dass das Objekt X die Temperatur von 70°C hat, nicht relevant für die Anwendung. Die Information, dass Objekt X die Temperatur von 70°C und seine Überhitzungsgrenze bei 65°C liegt, kann zu der Situation 'Objekt X ist überhitzt' führen.

Daraufhin kann beispielsweise automatisch die Abkühlungsmaßnahme gestartet werden, die das Objekt X wieder einsatzbereit macht.

2.1.4. Workflow

Ein Workflow, auch Arbeitsablauf genannt, kann als ein wiederholbarer, geordneter Vorgang beschrieben werden, in welchem benötigte Daten an alle Teilelemente (Aktivitäten von Objekten, Personen) eines Businessprozesses (siehe [LR00]) weitergeleitet werden. Dabei ist die Reihenfolge der Informationsbearbeitung durch die Teilnehmer wichtig. Im Rahmen der vorliegenden Arbeit wird der Begriff Workflow mit der Definition des *situationsbezogenen adaptiven Workflows* aus der Arbeit [WSBL15] gleichgesetzt. Um die Idee der situationsbezogenen Workflows zu erklären wird zunächst auf die länger etablierten *kontextbezogenen Workflows* eingegangen. Durch Realisierung und Entwicklung der Prinzipien von Industrie 4.0 und Internet of Things, entstehen Vernetzungen von Objekten, die mit viele Sensoren ausgestattet werden. Die Sensoren-Überwachung der Komponenten produziert große Mengen von Kontextdaten. Um den Arbeitsfluss in Systemen, die reich an Kontextdaten sind zu beschreiben, werden kontextbezogene Workflows eingesetzt. Ein kontextbezogener Workflow hat im Vergleich zum Standard-Workflow zusätzliche Eigenschaft, dass Kontextdaten eingebunden und berücksichtigt werden können. Das ist eine nicht triviale Erweiterung (z.B. siehe [WKNL07]), denn die standardisierten Workflow-Modellierungssprachen (BPEL, BPMN, YAWL, EPK, XPDL etc.) keine Einbindung der Kontextdaten vorsehen [Wie13]. Da die Aufgabe eines Workflows darin besteht, einen Businessprozess rechnergestützt zu automatisieren und zu steuern [HH95], wird innerhalb eines kontextbezogenen Workflows Information aus den Kontextdaten abgeleitet um den Zustand der Umgebung zu verstehen. Der Ansatz von Wolf et al. [WHR09] beschreibt, wie ein Workflow situationsabhängig modelliert werden kann. Der Unterschied zwischen den kontextbezogenen und situationsbezogenen Workflows besteht darin, dass in dem situationsbezogenen Fall, die Verarbeitung der Kontextdaten transparent für den Workflow ist. Das bedeutet, dass die Kontextdaten aggregiert wird und in Form von einer Situation innerhalb von einem Workflow auftreten kann. Eine Situation ist in diesem Fall eine Abweichung von einem Norm-Workflow (Ablauf ohne Situationen). Ein situationsbezogener Workflow muss Reaktionsmaßnahmen für alle vorgesehenen Situationen beinhalten. Damit passt sich der Workflow der Umgebung an und kann im autonomen Modus ablaufen.

2.2. Ausgangsarbeit

2.2.1. SitOPT-Projekt

Diese Masterthesis ist im Rahmen der Durchführung des DFG-Projekts SitOPT [Sit; WSBL15] entstanden. Ziel von SitOPT ist der Entwurf einer Drei-Ebenen-Architektur Abbildung 2.2 zur Adaption von situationsbezogenen Anwendungen an das sich ändernde Umfeld:

1. Auf der Sensorebene befinden sich die physischen Objekte. Diese Objekte besitzen Sensoren, mit welchen sie die Umgebung wahrnehmen. Die Sensorwerte der Objekte dienen als Kontextdaten-Input für die Situationen der Situationserkennungsebene.

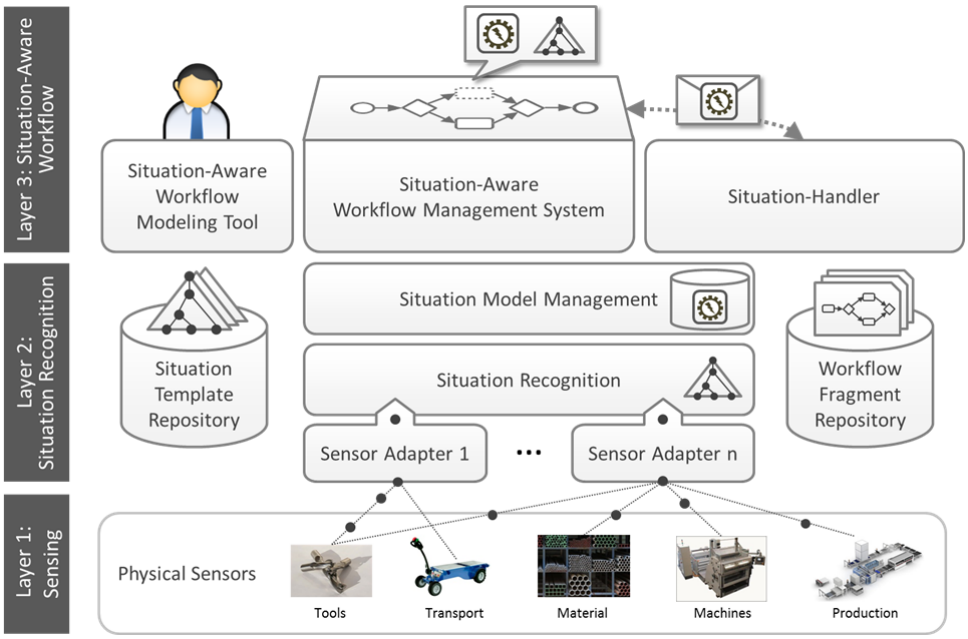

Abbildung 2.2.: Ziel-Architektur von SitOPT-Projekt [Sit].

2. Auf der Situationserkennungsebene werden aus den Kontextdaten Situationen abgeleitet. Die Situationen basieren auf den Situationstemplates (siehe nächstes Kapitel für Definition), welche im Vorfeld durch Domänenexperten modelliert wurden. Nach der Erkennung einer Situation, wird diese in der Workflow-Ebene verarbeitet.

3. Auf der Workflowebene wird auf Situationen reagiert. Für jede Abweichung von Norm-Workflow werden von Domänenexperten Maßnahmen zur Bearbeitung der eingetretenen Situation festgelegt.

Vorliegende Thesis beschäftigt sich mit Themen und Herausforderung der Situationserkennungsebene. Die Sensorebene wird für Evaluierungszwecke simuliert.

2.2.2. Situationstemplate

Der Begriff Template kommt aus dem englischen und bezeichnet eine Vorlage oder Schablone. Im Kontext der elektronischen Datenverarbeitung definiert ein Template die Struktur und den groben Aufbau eines Dokuments. Um aus einem Template ein fertiges Dokument zu erstellen, müssen die fehlenden variablen Daten festgelegt und deklariert werden (z.B. ein Formular mit offenen Feldern für Name, Adresse etc.).

Im Bereich Situationserkennung und Entwurf der intelligenten Systeme können *Situationstemplates* verwendet werden. Diese Templates beschreiben das Grundgerüst einer Situation, indem sie die Randbedingungen für die darunterliegenden physischen Objekte festlegen. Die Modellierung eines Templates, wird von Domänen-Experten gemacht, welche mit dem Verhalten der physischen Objekte vertraut sind. Die Experten bestimmen die Wertebereiche der Sensordaten der Objekte (z.B. die Temperatur eines Roboters darf im Bereich zwischen 0°C und 70°C liegen). Bei der Überschreitung der definierten Wertebereichen kann der Eintritt einer Situation erkannt

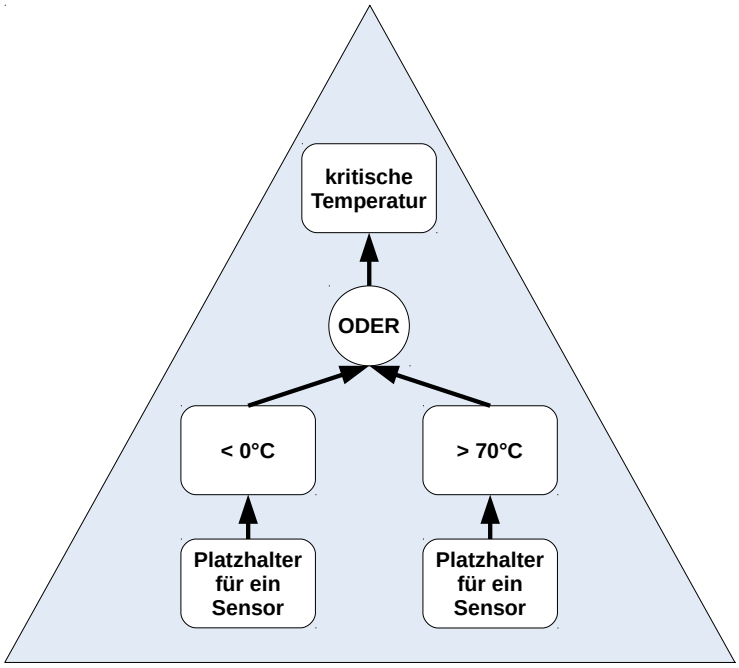

Abbildung 2.3.: Ein beispielhaftes Situationstemplate, welches eine Situation modelliert, welche durch Überwachung der Temperatursensoren zweier Things erkannt werden kann. An Stelle der Sensoren der Things sind Platzhalter welche die das Situationserkennungssystem instanziiert werden können.

werden (z.B. wird bei der Temperatur von 75°C erkannt, dass der Roboter überhitzt ist). Ähnlich wie bei Formularen offene Felder für persönliche Angaben frei gelassen werden, gibt es bei Situationstemplates Platzhalter für Sensoren der physischen Objekte.

Wenn man in den Situationstemplates konkrete Things (z.B. Maschine Nr.100) und nicht die Objekttypen, so kann das Modell nicht wiederverwendet werden. Das ist dadurch verursacht, dass mehrfache Erkennung einer Situation auf gleichen Things nicht sinnvoll ist. Wiederverwendung einer, auf diese Weise definierten, Situation, kann nur durch ihren Neustart verursacht werden. Definiert man in den Situationstemplates keine konkreten Objekte, sondern die Objekttypen, so kann das Modell potentiell häufiger wiederverwendet werden. Beispielsweise, kann das Situationstemplate Roboter-Überhitzung so modelliert werden, dass alle Roboter-Typen mit Temperatursensoren als Kontext-Lieferanten in Frage kommen. Auf diese Weise kann das Überhitzungsszenario auf mehreren Things erkannt werden.

2.2.3. Architektur

Im Rahmen der Diplomarbeit [Mor15] wurde ein Entwurf von SitOPT-Architektur erstellt. Die wesentlichen Elemente dieser Architektur sind in Abbildung 2.4 zu sehen. Die Information über die

physischen Objekte, sowie die modellierten Situationstemplates sind in der Datenbank enthalten. Auf diese Daten kann über das Situation Dashboard zugegriffen werden. Nach der Eingabe des überwachten Objekts, des Situationstemplates, des Situatuionserkennungssystems im Dashboard gelangen die eingegebenen Daten in Transformation Mapper. Diese Komponente bildet ein ausführbares Situationstemplate, welches in die Situationserkennungskomponente eingegeben wird. Dort werden die Kontextdaten über die Ressource Management Platform (RMP) dem Erkennungssystem übergeben. RMP ist ein Modul zur Beschaffung und Zwischenspeicherung der Sensordaten. Verschiedene Sensoren der Things können in RMP registriert werden und die Daten werden entweder in geregelten Zeitintevallen ausgelesen (Pull-Methode) oder via publish/subscribe-Paradigma [EFGK03] (Push-Methode) veröffentlicht. In der Situationserkennungskomponente wird der Kontext erkannt, verarbeitet und in Form von einem Situationsobjekt an die Situationsverwaltung zur Validierung, Speicherung und Weiterleitung an das Workflowsystem weitergeleitet. In Wechselwirkung mit Situationshandler kann Situationsverwaltungskomponente die erkannten Situationen dem Workflow bereitstellen.

2.2.4. Datenmodell

In der Arbeit [Mor15] wird eine Schnittstelle zwischen der Situationserkennungsebene und Workflowebene konzipiert. Das Datenmodell, das in dieser Arbeit vorgeschlagen wurde, beschreibt die gegenseitige Relationen der Entitäten, die an der Situationserkennung beteiligt sind. Die Entitäten sind die Ressourcen in der Datenbank, auf die während dem Modellierungsprozess und zur Laufzeit zugegriffen wird. Die Relationen der Ressourcen lassen sich durch ein ER-Diagramm beschreiben Abbildung 2.5. Jeder Sensor gehört zu einem Thing und erstellt Sensorwerte. Diese Sensorwerte sind in einer Situation enthalten. Eine Situation bildet sich aus einem Situationstemplate und Things, die die Kontextdaten liefern: Situation ist eine Instanz von Situationstemplate und Thing.

2.2.5. Modellierung der Situationstemplates

SitOPT sieht auf der Situationserkennungsebene ein Situationstemplates-Repository vor. In diesem Repository müssen alle Situationstemplates enthalten sein. Die Situationstemplates werden in Form von XML-Dateien verwaltet. Das Schema der Templates spielt dabei eine zentrale Bedeutung, denn da wird die allgemeine Struktur der Situationen bestimmt. Für die Modellierung von Templates wurde ein Werkzeug entwickelt [GSS15]. Die eingeführte grafische Repräsentation erleichtert es, die Situationstemplates zu modellieren, indem sie von XML-Darstellung abstrahiert. In dem grafischen Tool wird überprüft, ob alle semantischen Regeln (entsprechend dem XML-Schema) eingehalten wurden. Sofern ein grafisches Template validiert wurde, wird es (transparent für den Benutzer) auf XML abgebildet und (in der Datenbank) gespeichert werden. Die gespeicherten oder auf anderem Wege erstellten (direkt in XML erstellte) Situationstemplates können für weitere Bearbeitung ins Modellierungstool importiert werden.

Schema der Situationstemplates

Das XML-Schema der Situationstemplates sieht vier Datentypen vor: Situationen, Operationen, Konditionen und Kontext Abbildung 2.6.

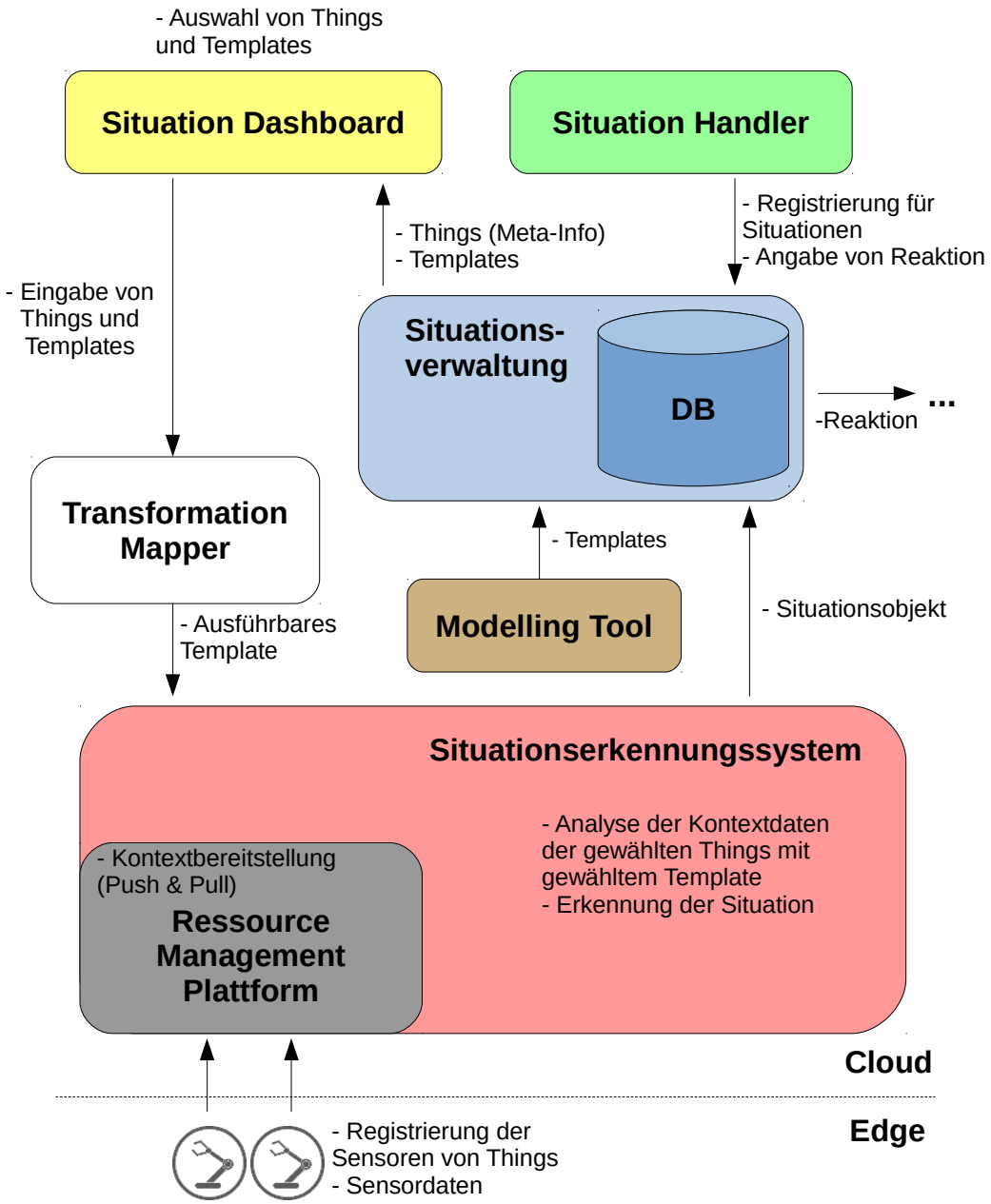

Abbildung 2.4.: Entwurf von SitOPT Architektur [Mor15].

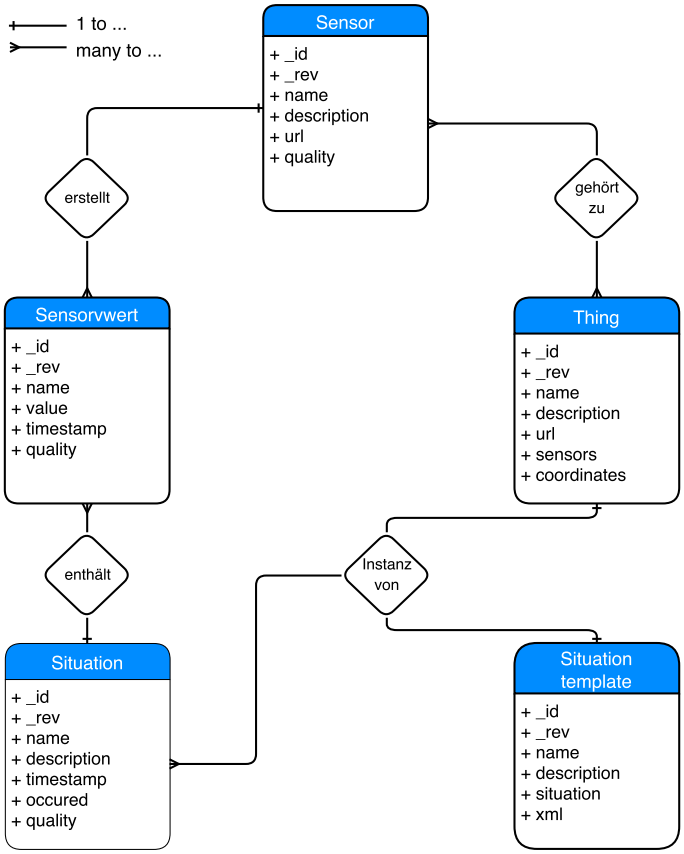

Abbildung 2.5.: Entity Relationship Diagramm für Datenmodell der Ressourcen des SitOPT-Entwurfs [Mor15]

Kontext: Ein Sensor, der Kontextdaten zur Situationserkennung bereitstellt. Ein Temperatursensor liefert Kontext in Form von einem Temperaturwert und einem Timestamp: 12°C (12:00), 12°C (12:01), 10°C(12:02), 11°C (12:03) etc.

Kondition: Eine Bedingung, auf welche die rohen Kontextdaten überprüft werden. Eine Kondition kann z.B. auf Überschreitung eines bestimmten Wertes prüfen: *"größer als 15?"*.

Operation: Eine logische Operation, die mehrere Konditionen zusammenführt. Zusammenführung der Konditionen *"größer als 15?"* und *"kleiner als 20?"* durch ein logisches UND ergibt höherwertigen Kontext: *"im Intervall zwischen 15 und 20"*.

Situation: Bei der Zusammenführung der Operationen entsteht eine Situation. Eine Situation ist eine Interpretation der Kontextdaten und dadurch eine high-Level Kontextinformation. Das heißt, dass eine Situation auch eine Kontextinformation ist. Die Situation wird modelliert, um einen Zustand erkennen und auf diesen reagieren zu können.

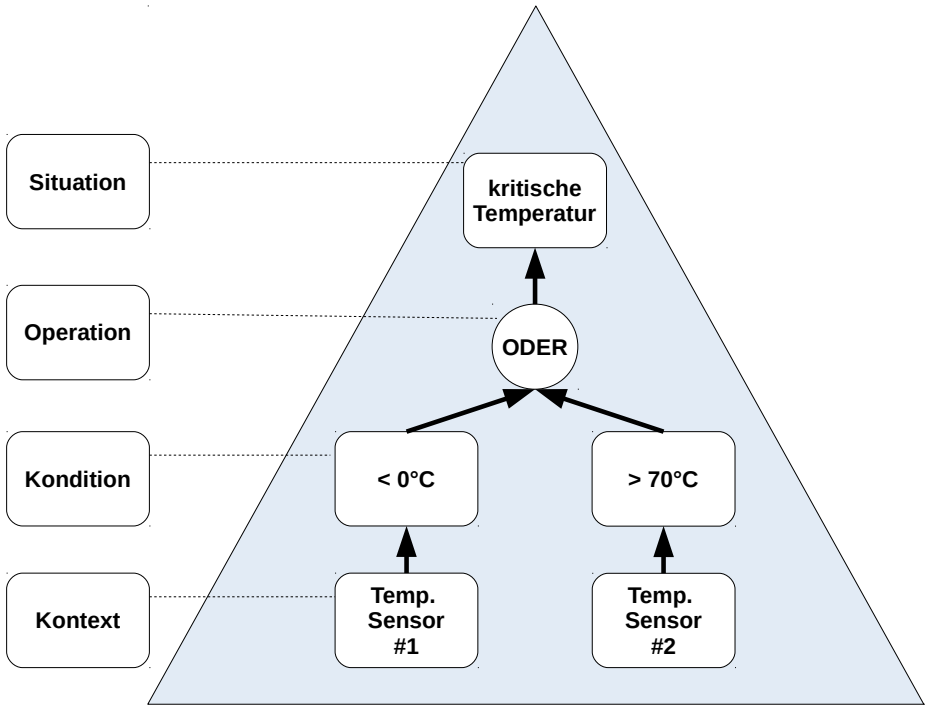

Abbildung 2.6.: Schema der Bestandteile von Situationstemplates

2.3. Verwandte Arbeiten

In Kontext von SitOPT ist eine Reihe von Arbeiten entstanden. Diese Arbeiten betreffen alle Ebenen der Zielarchitektur von Things und Sensoren [HBS+16; HWBM16a; HWBM16b] über Situationserkennung [HWS+15; MHWM17; SHWM16] bis hin zu Workflow-Management [WSBL15].

Herausforderungen der Situationserkennung wurden in den Arbeiten thematisiert, die sich mit Roboter-Verhalten auseinandersetzen. Dieser Anwendungsbereich kann als Spezifizierung der allgemein definierten SitOPT-Aufgaben interpretiert werden. In einem Ansatz [WG96] werden Entscheidungen über das Verhalten der Roboter auf der Basis der Situationen getroffen. Die Situation beschreibt in diesem Fall den Zustand der Umgebung des Roboters, sowie seine Ziele und seine Verhaltensszenarien. Die Situationen werden mit Sensoren des Roboters erfasst. In einem anderen Ansatz [DGG93] muss das entworfene Situationserkennungssystem die Umgebung in Form von Ereignissen mit Zeitstempel wahrnehmen. Eine Situation wird als Abfolge von Ereignissen und Einschränkungen definiert. Bildlich dargestellt sieht eine Situation wie ein gerichteter Graph aus. Sobald im Erkennungssystem eine zuvor definierte Abfolge von Ereignissen wahrgenommen wird, wird eine passende Reaktion getriggert.

SitRS und SitRS XT sind Ansätze, die im Rahmen von SitOPT entstanden sind und sich mit der Situationserkennung beschäftigen. Die Erkennung basiert auf Complex Event Processing (CEP) [BK09; Luc02] und alle Komponenten dieser Ansätze können in Cloud-Umgebung realisiert werden. Daher sind die Komponenten replizierbar. Dadurch ist die Erkennung mit SitRS skalierbar.

Demnach kann eine Aufgabenverteilung auf unterschiedliche Replikas erfolgen (Load Balancing). Durch Einsatz von CEP können große Datenmengen (kontinuierlicher Fluss an Sensordaten) parallel verarbeitet/überwacht werden. Die prototypische Implementierung der Situationserkennung im Rahmen von SitRS XT (erweiterte und modifizierte Variante von SitRS) kann in nahezu Echtzeit (near real-time) erfolgen.

Die Entwicklung von Adaptable Pervasive Flow (APF) ist ein Ansatz, der das klassische Workflow-Schema erweitert [WHR09]. Diese Erweiterung setzt optimierte Bereitstellung der Kontextdaten als Ziel. Dazu werden die relevanten Kontextinformationen zur Laufzeit, Ad hoc bestimmt. Die Aktivitäten eines Workflows werden gruppiert und erhalten Information über Typen der beteiligten physischen Objekte. Zur Laufzeit werden konkrete Objekte als Kontextdaten-Lieferanten gewählt. Der Kontext wird durch die gruppierte Abfolge der Aktivitäten verarbeitet. Dadurch wird die Menge der Kontextdaten verringert und irrelevanter Kontext wird herausgefiltert.

Ein System für Erkennung von Ereignissen innerhalb eines Sensornetzes wurde im Rahmen des [DW08] Ansatzes entwickelt und evaluiert. Die Aufgabe der Energieeinsparung steht dabei im Vordergrund. Um dieser Herausforderung standzuhalten wird verteilte Erkennung eingesetzt. In dem vorgeschlagenen Ansatz werden Techniken zur Mustererkennung benutzt: Das Erkennungssystem kalibriert die Sensoren und ermöglicht dadurch ein Training mit unterschiedlichen Mustern. Dadurch wird die Klassifizierung und Erfassung von Mustern durch Sensoren ermöglicht. Es werden Kommunikationsmethoden zwischen einzelnen Sensoren und Fusionsmechanismen für Teilereignisse berücksichtigt. Bei der Evaluierung des Entwurfs wurden Vorteile der verteilten Erkennung im Vergleich zur lokalen Erkennung festgestellt.

3. Technischer Hintergrund

Da vorhergehende Kapitel (Grundlagen und verwandte Arbeiten) beschreibt den Stand der Wissenschaft in verwandten Bereichen zu der vorliegenden Arbeit. In diesem Kapitel wird der technische Hintergrund der Arbeit zum Vorschein gebracht. Die im Folgenden vorgestellten Methoden, eignen sich insbesondere für den Entwurf der IoT-Systeme. Das Augenmerk wird auf die Aspekte der Datenspeicherung und Datenkommunikation gerichtet.

3.1. NoSQL

Ein etablierter Standard für die Speicherung und Verwaltung der Daten war und bleibt das relationale Datenbankmodell. Dieses Modell legt klar definierte Schemata vor, an welche sich die Daten halten müssen. Die Nutzung der relationalen Datenbanken kann allerdings bei einer sehr hohen Zahl der Lese- und Schreibzugriffe oder bei kontinuierlichen Datenänderungen zu Leistungsproblemen führen [NLIH13]. Dagegen können NoSQL-Datenspeicher, von schematischer Datenverwaltung abstrahieren und Vorteile hinsichtlich Skalierung aufweisen [MH13].

3.1.1. CAP und PACELC

Die relationalen Datenbanken gelten als zuversichtlich, weil sie einen hohen Wert auf die Konsistenz der Daten legen. Darunter leidet jedoch laut CAP-Theorem die Verfügbarkeit und Toleranz gegenüber Partitionen. Das CAP-Theorem besagt, dass ein verteiltes System die drei der folgenden Eigenschaften - **C**onsistency(Konsistenz), **A**vailability(Verfügbarkeit), **P**artition Tolerance(Toleranz gegenüber Partitionen) - nicht zur gleichen Zeit besitzen kann. Das PACELC-Theorem ist eine Erweiterung des CAP-Theorems, welche besagt, dass sogar im Falle einer Partition ein Kompromiss zwischen Latenzzeit und Konsistenz zu finden ist: Im Fall von einer Partition (**P**artition) wird Kompromiss zwischen Verfügbarkeit(**A**vailability) und Konsistenz(**C**onsistency) gemacht. Sonst(**E**lse) wird zwischen Latenz(**L**atency) und Konsistenz(**C**onsistency) entschieden.

Eine Partitionierung in einem verteilten System kann nicht vollständig ausgeschlossen werden. Dadurch müssen die Kompromissen zwischen Konsistent und Verfügbarkeit/Latenz gefunden werden.

Dadurch, dass im relationalen Modell die Konsistenz die höchste Priorität hat, eignet es sich insbesondere für sensible Daten. Jedoch, wenn es akzeptabel ist, die Daten nicht konstant konsistent zu halten, um mehr Verfügbarkeit oder geringere Antwortzeit zu gewinnen, so ist die Nutzung von NoSQL-Datenspeicher vorteilhaft.

3.1.2. ACID und BASE

ACID (auf deutsch AKID) beschreibt die Anforderungen an ein verteiltes Datenverwaltungssystem insbesondere in Hinblick auf Transaktionsverwaltung. Bei dem Management der Transaktionen spielt Verlässlichkeit die zentrale Rolle. Die Eigenschaften von AKID sind **A**tomarität, **K**onsistenz, **I**solation und **D**auerhaftigkeit.

Durch *Atomarität* werden die Transaktionen entweder vollständig oder gar nicht ausgeführt. Für die Transaktionsteile, die bereits abgeschlossen wurden müssen Rollback-Mechanismen vorgesehen werden.

Durch *Konsitenz* wird Integrität der Transaktionen gewährleistet. Das bedeutet, dass jeder Zugriff (von jedem verteilten Systemknoten) auf die durch diese Transaktion veränderten Daten die gleichen Ergebnisse liefert. Das muss gelten solang die Daten nicht durch eine andere Transaktion verändert werden.

Die *Isolation* verlangt Kapselung der Transaktionen voneinander: Die Wirkung einer Transaktion betrifft nicht die parallel ablaufenden Transaktionen.

Die *Dauerhaftigkeit* garantiert Persistenz der Änderungen, die durch die Transaktion hervorgerufen wurden.

BASE (**B**asically **A**vailable **S**oft State, **E**ventual Consistency) beschreibt die Eigenschaften eines Systems in welchem nicht die Sicherheitsanforderungen von AKID bestehen müssen. BASE ist durch weiche Konsistenz, Verfügbarkeit und daraus folgende Geschwindigkeit gekennzeichnet. *Eventual Consistency* garantiert, dass die Konsistenz der Daten früher oder später erreicht wird und erlaubt Nutzung von *stale data* - Daten, die noch nicht auf dem aktuellsten Stand sind.

Durch Fokus auf Verfügbarkeit und Ersetzung der Konsistenz durch Eventual Consistency skalieren NoSQL Datenbanken horizontal und nicht vertikal. Das bedeutet, dass die Vergrößerung der Anzahl der verteilten Knoten die Leistung des Systems verbessert.

3.1.3. REST-API und CRUD-Funktionen

Das Akronym *CRUD* bezeichnet vier Basisfunktionen für Bearbeitung von persistent gespeicherten Daten [Mar81]:

- **C**reate: Erstellung einer Ressource.
- **R**ead: Lese-Zugriff auf eine vorhandene Ressource.
- **U**pdate: Schreib-Zugriff für die Änderung einer vorhandenen Ressource.
- **D**elete: Entfernung einer vorhandenen Ressource.

Dieselbe Semantik wie die Funktionen von CRUD haben die SQL Methoden *INSERT, SELECT, UPDATE, DELETE* sowie HTTP-Anfragen(Requests) *PUT/POST, GET, PUT/POST, DELETE*. Die HTTP-Requests benutzt man häufig im Rahmen der Entwicklung von *RESTful APIs*. REST steht für Representational State Transfer (Repräsentative Zustandsübergabe) und ist ein Stil in Programmierung von Webservices [Mas11]. Die kennzeichnenden notwendigen Prinzipien von REST sind:

- **Client-Server**: Klare Trennung von Server und Client-Interfaces um separate Entwicklung dieser Teile zu ermöglichen.
- **Zustandlosigkeit**: Client und Server kommunizieren über ein zustandloses Protokoll. Jede Client-Nachricht enthält alles was für das Verstehen dieser Nachricht von dem Server nötig ist. Dadurch fehlt das Bedürfnis Session-Information zu serverseitig verwalten.
- **Caching**: Die Antworten (Responses) des Servers können von Vermittlerknoten (Proxies) gespeichert werden um direkte Zugriffe auf den Server zu vermeiden.
- **Einheitliche Schnittstelle** Die ausgetauschten Informationen werden in eine standardisierte Form (eindeutige Adressierung der Ressourcen, Manipulation der Ressourcen über (dynamisch erhaltene) Standard-Methoden/Repräsentationen) gebracht.
- **Mehrstufigkeit** Der Client hat kein Wissen darüber, ob er direkt mit dem Server oder mit einem Proxy kommuniziert. Durch Benutzung von Proxy-Vermittlung ist es möglich die Lastenverteilung und Skalierbarkeit des Systems zu verbessern.
- **Code-on-Demand** Der Client kann von dem Server den Code (in From von Applets oder Scripts) für lokale Ausführung bekommen.

3.1.4. CouchDB

Apache CouchDB ist eine Dokumentenorientierte NoSQL-Datenbank. Die Daten werden in Form von JSON-Dokumenten gespeichert welche in Collections gruppiert werden. Die Collections unterliegen keinen Schemata. Neben optionalen Feldern werden die Dokumente mit automatisch generierten IDs (*id*) und Versionsnummern (*rev*) versehen. Die Versionsnummer wird bei jedem Update inkrementiert. Um ein Dokument zu ändern wird eine Einfügeoperation benötigt: Für neues Dokument wird ein Dokument mit neuer ID eingefügt; für Update wird ein Dokument mit einer bestehenden ID und inkrementierter Versionsnummer eingefügt; für Löschoperation wird ein Dokument mit einer bestehenden ID und einem *'delete-Flag'* eingefügt. Es können Version-Konkurrenz Probleme auftreten: Während einem Dokument-Update wird darauf ein Lesezugriff ausgeführt. In diesem Fall wird die aktuelle Version, die vor dem Update-Beginn gegolten hat, beim Lesezugriff zurückgegeben. Nach der Beendigung des Updates wird die Versionsnummer erhöht und der nächste Zugriff wird das bereits geänderte Dokument zurückgeben.

CouchDB unterstützt Replikation und steigert dadurch die Performanz der Dokumenten-Zugriffe. Jedes Replica kann Dokumente bearbeiten und für die Konsistenz der Daten wird in Synchronisationsphasen gesorgt. Dadurch wird Eventual Consistency und hohe Verfügbarkeit erreicht.

Die Zugriffe auf die Inhalte der Datenbank werden mittels RESTful HTTP-Requests gemacht. Alle Elemente in CouchDB (Collection, Dokument) verfügen über eindeutige URI und HTTP-Requests GET, PUT, POST und DELETE realisieren die CRUD-Funktionen Abschnitt 3.1.3. Für komplexe Suchanfragen werden in CouchDB Views verwendet. Die Views werden mit dem MapReduce-Verfahren erstellt (Map filtert Dokumente und Reduce fasst die Werte zusammen).

Die Dokumente können Anhänge in von JSON verschiedenen Formaten (XML, YAML etc.) beinhalten.

3.1.5. NoSQL in IoT

Internet of Things Anwendungen profitieren von den wesentlichen Vorteilen der NoSQL-Datenverwaltung: horizontale Skalierbarkeit und Schemafreiheit. Die Daten im IoT Kontext kommen aus variablen Quellen und sind so unterschiedlich, dass eine flexible Speicherung der Daten vorteilhaft sein kann. Nicht nur die flexible Natur der Daten sondern auch die Datenmenge sind in IoT-Anwendungen herausfordernd. NoSQL-Datenbanken können große Datenmengen verarbeiten, ohne dass die Antwortzeit erheblich steigt.

3.2. Flowengine (NodeRed)

Die Flow-Programmierung beschreibt eine Methode, die Anwendungen als Vernetzung von Prozessen mit verborgenem Verhalten darstellt. Diese Prozesse kommunizieren miteinander, indem sie Nachrichten austauschen. Die Prozesse sind Knoten im Datenfluss welche Daten als Eingabe bekommen, verarbeiten und eine Ausgabe produzieren. Es gibt drei Arten der Knoten. Datenquellen sind Knoten, die die Daten ausschließlich produzieren - besitzen nur Ausgänge zum Versenden; Datensenken sind die Knoten, die die Daten ausschließlich konsumieren - besitzen nur Eingänge zum Empfangen; zusätzlich gibt es Knoten die Daten sowohl versenden als auch empfangen können - besitzen Ein- und Ausgänge. Die Arbeit eines Knotens kann nur starten, wenn die Daten an seinen Eingabeeingängen vorliegen.

Node-Red ist ein Werkzeug, das mittels Flow-Programmierung die Vernetzung der Elemente einer IoT-Anwendung vornehmen kann. Die Anbindung der IoT-Bausteine wird mittels Standard-Protokolle wie HTTP, TCP, MQTT etc. realisiert. Es können sowohl virtuelle Komponenten (E-Mail, WebSockets) als auch Hardware-Geräte angeschlossen werden. Node-Red besitzt eine Reihe von eingebauten Knoten-Typen die mit Hilfe der erwähnten Protokolle mächtige Input und Output Knoten bilden können. Dabei wird die Komplexität der Interaktion der Knoten mit der realen Welt transparent gehalten: Die Programmierdetails sind versteckt. Falls die Mächtigkeit der integrierten Knoten-Typen nicht ausreicht, können benutzerdefinierte Funktionen erstellt werden.

Dadurch, dass Node-Red in einer light-weight JavaScript Laufzeitumgebung (node.js) implementiert wird, ist es für Umsetzung auf Edge-Knoten geeignet. Es eignet sich auch für die Arbeit auf leistungsschwachen Raspberry Pies. Arbeit in Cloud-Umgebungen ist ebenfalls möglich.

3.3. Message Queue und MQTT

Message Queue is eine Art der Kommunikation der Prozesse untereinander, sowie der Threads innerhalb eines Prozesses [EFGK03]. Im Rahmen der Kommunikation werden Nachrichten ausgetauscht. Der Sender einer Nachricht sendet die Nachricht nicht an einen konkreten Empfänger, sondern veröffentlicht diese für alle potentiellen Interessenten. Dafür wird ein sogenannter Topic definiert, unter welchen der Sender seine Nachricht(en) verfügbar macht. Der Kommunikationspartner kann ein Topic abonnieren und alle darunter veröffentlichten Nachrichten erhalten. Dadurch entsteht eine asynchrone, lose gekoppelte Kommunikation. Dabei wird der Sender als *Publisher* und der Empfänger als *Subscriber* bezeichnet.

Ein etabliertes, asynchrones Protokoll für den Austausch von Nachrichten heißt MQTT (Message Queue Telemetry Transport) [HTS08]. MQTT baut auf TCP/IP-Protokoll und ist für den Einsatz in Netzwerken mit häufigen Verbindungsverlusten, niedriger Bandbreite und Verzögerungen geeignet. Diese Eigenschaft macht MQTT attraktiv für Kommunikation in IoT-Anwendungen: Da die Things häufig keine leistungsstarke, voll ausgebaute Rechner sind, wird geringe Leistungsaufnahme und Bandbreitenbeanspruchung von MQTT besonders benötigt. MQTT ist ein Client-Server-Protokoll in dem der Server ein Message Broker ist. Die Clients bauen Verbindung zum Server auf und können in Publish-Subscribe-Manier Nachrichten austauschen. MQTT unterscheidet drei Qualitätsstufen:

At-most-once : Nachricht wird einmal gesendet und vergessen (fire and forget).

At-least-once : Nachricht wird potentiell mehrfach gesendet um den Empfang sicherzustellen. Duplikate können vorkommen.

Exactly-once : Nachricht wird nur einmal empfangen, auch wenn die Verbindung temporär verloren wird.

MQTT ist als M2M (machine to machine) Protokoll des IoT und IIoT (Industrial IoT) standardisiert [OAS]. MQTT eignet sich für IoT-Anwendungen, denn Things (z.B. mobile Geräte) verfügen über limitierte Ressourcen und das Protokoll sehr sparsam (kleine Datenpakete, niedriger Energiekonsum, effiziente Informationsausbreitung) ist.

3.4. Edge und Cloud Computing

3.4.1. Cloud Computing

Cloud Computing Paradigma beschreibt die Methoden und Arten der Daten- und Servicebereitstellung über das Internet [AFG+10]. Dabei müssen die Daten nicht lokal heruntergeladen und die Services nicht installiert werden. Die Cloud Computing Systeme müssen skalierbar sein, unabhängig von Ort, Gerät und Plattform. Ein großer Vorteil von Cloud ist die Möglichkeit die zusätzliche Leistung bei Bedarf (on demand) zu mieten und bei fehlendem Bedarf zu stornieren. So kann der Cloud-Nutzer durch eine Abschätzung seiner Bedürfnisse über die Zeit eine korrelierende Leistungslieferung bestellen und erwarten. Die Dienstleistungen, die über Cloud zur Verfügung gestellt werden, können in drei Gruppen klassifiziert werden: Infrastruktur(IaaS), Plattform(PaaS) und Software(SaaS).

- IaaS Infrastruktur in Form von Rechner-Cluster wird als Dienstleistung angeboten. Der Benutzer kann eigenständig erwünschte Software und Umgebung installieren und aufrechterhalten.
- PaaS Rechnerleistung und Programmierungsumgebungen werden dem Benutzer angeboten damit dieser eigene Software entwickeln kann. Für die Funktionalität der Softwareumgebung ist der Cloud Provider zuständig.
- SaaS Cloud Provider stellt Infrastruktur und bereits entworfene Software zur sofortigen (on demand) Benutzung an.

Es existieren vier Modelle für die Bereitstellung der Dienstleistungen: Public Cloud, Private Cloud, Hybrid Cloud und Community Cloud.

- Public Cloud Zugang zu den Rechnerkapazitäten ist öffentlich und funktioniert nach *pay-as-you-go* Prinzip. Das bedeutet, dass die Lieferung von Leistung und die Bezahlung von dem aktuellen Bedarf abhängt. Cloud Provider kümmert sich dabei um die Infrastruktur, Hosting, Verwaltung und Lastverteilung.
- Private Cloud Private Cloud ist für einzelne Organisationen vorgesehen. Das Hosting und die Verwaltung werden von der Organisation (oder einem Drittanbieter) übernommen. Die Organisation kann die Nutzung der Cloud Dienstleistungen für eigene Kunden oder Geschäftspartner freischalten.
- Community Cloud Bereitstellung der Infrastruktur für eine Organisationsgruppe. Die Organisationen in der Gruppe haben in der Regel etwas gemeinsam z.B. gemeinsamen Aufgabenbereich oder geografische Nähe. Dadurch können Kosten für Cloud-Dienste aufgeteilt werden.
- Hybrid Cloud In der Hybrid Cloud Variante ist es möglich die Vorteile von unterschiedlichen Methoden zu kombinieren. Beispielsweise kann eine Organisation Private (Inhouse) Cloud für sensible Kundendaten und Public Cloud alles weitere verwenden.

3.4.2. Edge Computing

Bei Edge Computing geht es im Gegensatz zu Cloud Computing darum, die Daten nicht über das Netzwerk in die Cloud-Zentrale zu schicken, sondern darum, Methoden die Daten lokal auf dem 'Netzwerkrand' - der Edge - zu verarbeiten bzw. zu speichern. Dadurch, dass die Daten auf Endgeräten der Edge verarbeitet werden, kann die Antwortzeit verringert werden. Durch diese Eigenschaft ist Edge Computing für Echtzeit-Anwendungen geeignet. Eine Herausforderung für die Edge ist jedoch die (potentielle) Verarbeitung von großen Menge an Daten.

In der Arbeit von W. Shi et al. [COV; SCZ+16] wird auf die Vorteile der Nutzung von Edge Computing eingegangen. Wenn ein Teil der Information direkt auf den Edge-Knoten verarbeitet werden kann, so kann mit verkürzter Latenzzeit für lokale Anwendungen, sowie Einsparung der Bandbreite gerechnet werden. Wenn dabei keine sensiblen Daten in die Cloud gesendet werden müssen, steigert sich dadurch auch die Sicherheit.

4. Konzept

In diesem Kapitel wird die Arbeitsweise der verteilten Situationserkennung, sowie die darunterliegende Architektur beschrieben. Es werden die Veränderungen zu der Ausgangsarbeit beschrieben und begründet. Es handelt sich konzeptionell um zwei unterschiedliche Verteilungsszenarien und die jeweiligen Rollen der Edge- und Cloud-Knotenpunkte. Dabei ist die neue Rolle von Edge-Nodes eine prinzipielle Neuerung, die eine Änderung der bisherigen Rolle der Cloud-Instanz mit sich bringt. Ein wichtiger Teil der Architektur der Ausgangsarbeit - das Ressource Management Platform (RMP) - wurde komplett ersetzt. Stattdessen wird für die Kommunikation der Sensordaten Message Queue verwendet. Das Datenmodell wurde entsprechend der neuen Herausforderungen angepasst.

Zentrale Neuerung im Vergleich zum bisherigen Konzept ist die Nutzung der Vorteile des Edge Computing. Durch doe Verteilung von Situationserkennung können Vorteile, wie geringerer Datenverkehr in die Cloud, erhöhte Sicherheit, sowie verbesserte Antwortzeit für lokale Edge-Anwendungen.

4.1. Architekturaufbau

Aus der technischen Umsetzung der verteilten Situationserkennung, sowie der Datenbank, Message Queue, Node-Red Flowengine und dem Modelierungstool setzt sich eine IoT-Architektur zusammen. Die Funktionsweise dieses Systems angefangen von Situationstemplate-Modellierung bis hin zu beispielhaften Situationserkennungs-Szenarien wird präsentiert.

In der Abbildung 4.1 unten (rechts) wird die Architektur dargestellt, die als Grundlage für eine verteilte Situationserkennung dienen kann. Die Modellierung der spezifischen, je nach Einsatzbereich variierenden Situationstemplates, gilt als notwendige Voraussetzung der Situationserkennung. Zwar gehört die Modellierung nicht zum eigentlichen Erkennungsprozess, jedoch müssen für den Verlauf der Erkennung Templates vorliegen, die durch einen Domänenexperten entworfen wurden. Die modellierten Templates werden in einer zentralen Datenbank abgelegt und können über die Verteiler- und Mapper-Komponenten mit konkreten Objekten - den *Things* assoziiert werden. Die Sensoren der Things stellen Kontextdaten bereit, welche für die Erkennung der modellierten Situation mit den Konditionen des Templates verglichen werden. Wenn die Konditionen erfüllt sind, wird die modellierte Situation erkannt.

Der aktuelle Architekturentwurf Abbildung 4.1 unten (rechts) wird mit dem Stand vor Beginn der vorliegenden Arbeit Abbildung 4.1 oben (links) [Mor15] verglichen. Im Vergleich zu der vorherigen Architektur, entfällt die Komponente Resource Management Platform (RMP), welche für die Bereitstellung der Kontextdaten zuständig war. Message Queue wurde als neue Art der Kontextdaten-Bereitstellung gewählt. Dadurch wird die Gesamtarchitektur kompakter, denn das RMP muss nicht mehr aufgesetzt werden. Zusätzlich wird durch die Einführung von Message

Queue ein etablierter M2M-Kommunikationsstandard gewählt. Trotz Vorteile, die RMP verspricht, ist Benutzung eines Industrie-Standards häufig einer eigenständigen Software vorzuziehen. In der Ausgangsarbeit wurden alle Komponenten in eine zentralen Cloud-Instanz realisiert. Durch die Einführung der neuen Verteiler-Komponente ist es möglich die Vorteile von Edge Computing zu nutzen. Die Entscheidung, ob die Verteilung vorgenommen wird basiert auf Kriterien wie Overhead für Verteilung, Bandbreitennutzung, Latenz und Security. Im Kontext dieser Arbeit wird im Gegensatz zu der Ausgangsarbeit kein Bezug auf die Workflowebene von SitOPT genommen. Dadurch wird die Situationhandler Komponente durch die allgemeinere Komponente, situationsbezogene adaptive Anwendung ersetzt.

In den nächsten Abschnitten werden die wichtigsten Elemente der Architektur im Detail beschrieben.

4.2. Schema der Situationstemplates

Die Modellierung der Situationstemplates liegt dem gesamten weiteren Erkennungsprozess zugrunde. Die Vorlagen, ferner *Templates* genannt, werden in einer graphischen STMT-Modellierungsumgebung erstellt und in Form von XML-Dateien gespeichert. Die Form der XML-Templates wird durch ein XML-Schema beschrieben. Die bisherige Umsetzung der Modellierungsumgebung hat es nicht erlaubt bereits existierende Templates als Eingabe für neue komplexere Templates zu verwenden. Dadurch wurde die Komplexität der potentiell modellierbaren Situationen eingeschränkt bzw. durch unübersichtliche, redundante Modellierung, durch Kopieren der bereits existierenden vollständigen Templates in die neuen Templates, erschwert. Um dieses Problem zu umgehen, wurde ein prinzipiell unterschiedliches Schema der Situationstemplates entworfen (siehe Anhang A.1). Die bereits erstellten Situationstemplates sind über einen Zugriff auf die Datenbank als Eingabe für weitere komplexere Situationen verfügbar. Diese Neuerung ist eine Voraussetzung für das ferner beschriebene (Abschnitt 4.4) modulare Verteilungsparadigma. Der Nutzen der Schema-Änderung wird durch folgendes Beispiel verdeutlicht: Zunächst wird die Situation *Temperatur-Fehler* erstellt - siehe Abbildung 4.2. Nachfolgend wird eine komplexere *Maschine außer Betrieb*-Situation modelliert, in welcher dieselben Elemente, wie in der *Temperatur-Fehler*-Situation vorkommen. In der bisherigen Umsetzung des Template-Schemas wird die komplexe Situation komplett neu, ohne Berücksichtigung der vorhandenen Templates modelliert - siehe Abbildung 4.3. Durch das neue Schema können vorhandene Templates bei der Modellierung neuer Situationen genutzt werden - siehe Abbildung 4.4.

Im Weiteren werden Begriffe Eingabe-Template, Ziel-Template, Eingabe-Element verwendet. Diese Begriffe werden folgendermaßen definiert:

Ziel-Template: Bei der Modellierung von einem Situationstemplate wird dieses als bezeichnet, da es das ursprüngliche Ziel der Modellierung ist.

Eingabe-Template: Bei der Modellierung von einem Ziel-Template ist es möglich eine Situation als Kontextdaten-Lieferant zu verwenden. Diese Situation basiert wiederum auf einem eigenen Template. Der Ziel-Template-Designer wählt über einen Zugriff auf die Datenbank das Template, welches die erwünschte Eingabe-Situation modelliert. Dieses Template wird im Weiteren Eingabe-Template genannt.

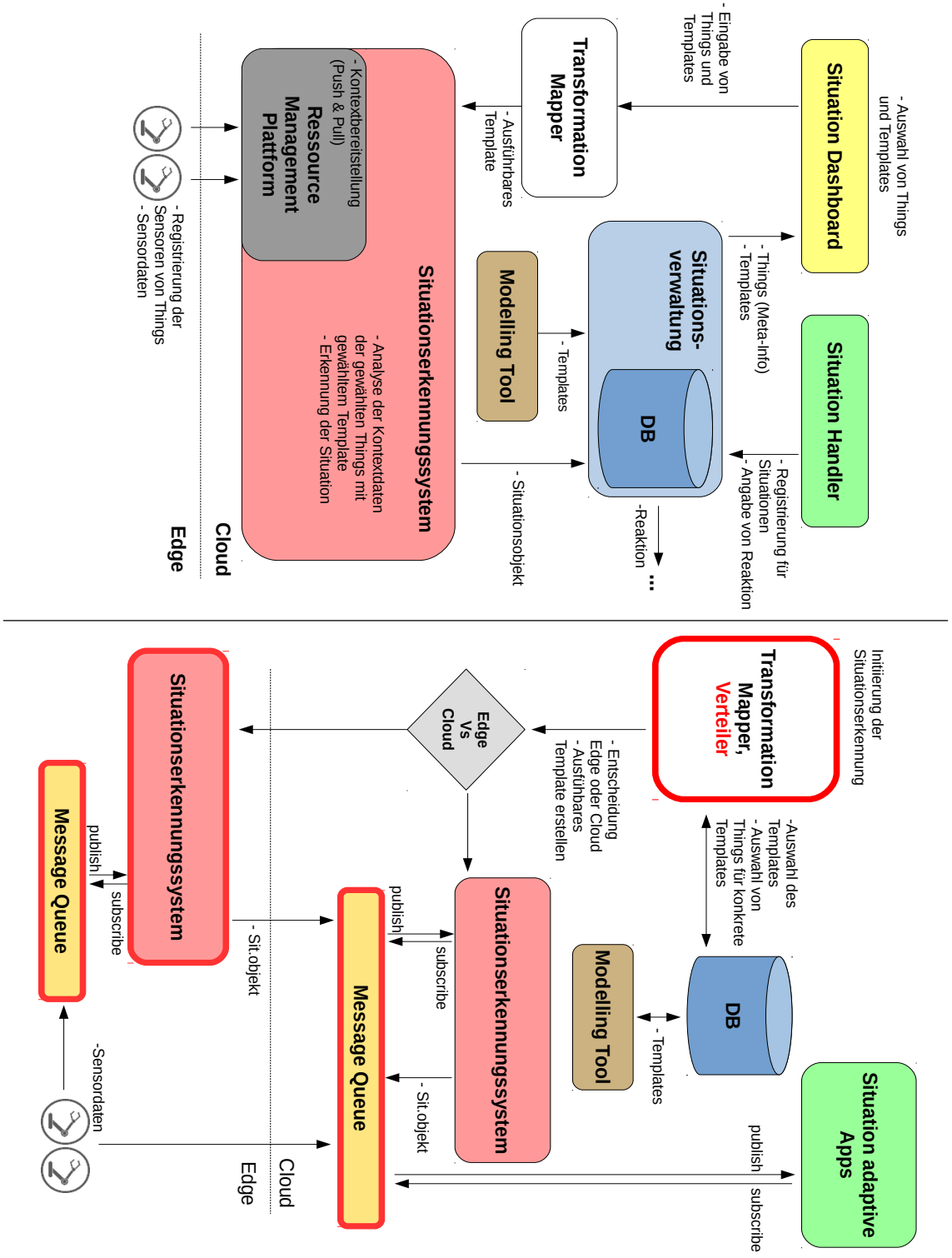

Abbildung 4.1.: Vergleich der Architektur der Ausgangsarbeit [Mor15] - oben (links) und der vorliegenden Arbeit unten (rechts). Die neu eingeführten Komponenten sind rot umrandet.

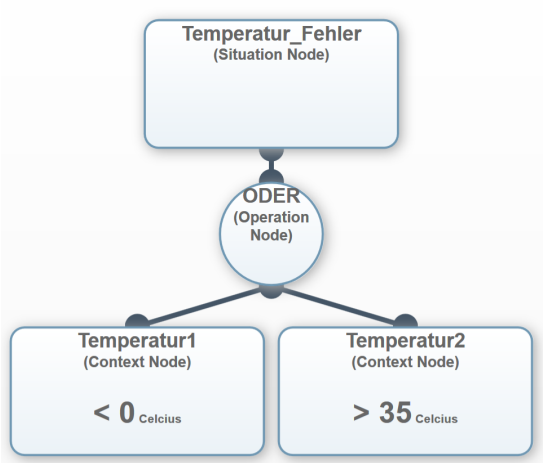

Abbildung 4.2.: Das Situationstemplate, das einen Temperaturfehler modelliert. Ein Temperaturfehler ist in diesem Beispiel ein Zustand, in dem der gültige Sensorwerte Bereich zwischen 0°C und 35°C verlassen wird.

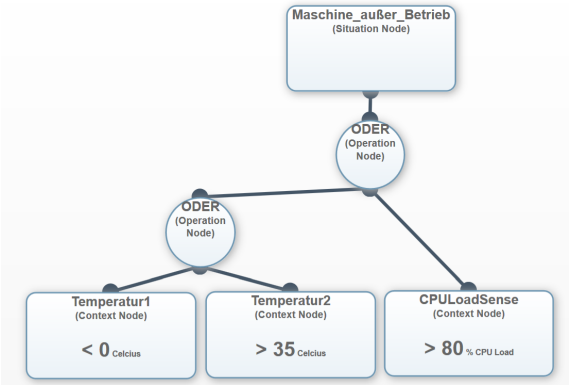

Abbildung 4.3.: Die Situation, die den Betriebstopp einer Maschine modelliert.

Eingabe-Element: Bei der Modellierung von einem Ziel-Template gibt es zwei potentielle Kontextdaten-Quellen: Sensor und Eingabe-Template (Situation). Diese Kontextdaten-Quellen sind Eingabe-Elemente für das Ziel-Template.

Eingabe-Typ: Als Eingabe-Typ wird ferner die Art des Eingabe-Elements bezeichnet. Der Eingabe-Typ ist entweder ein Sensor oder eine Situation.

Die Umsetzung des neuen XML-Schemas in dem Modellierungswerkzeug erlaubt es dem Benutzer über eine Datenbankinteraktion auf die, in der Datenbank gespeicherten, Situationstemplates zuzugreifen. So wird ein existierendes Template, bzw. dadurch modellierte Situation, als Kontextdaten-Lieferant (Eingabe) verwendet Abbildung 4.5. Dabei wird der Bezeichner von diesem Eingabe-Template in dem XML-Repräsentation des Ziel-Templates eingetragen. Zur Laufzeit

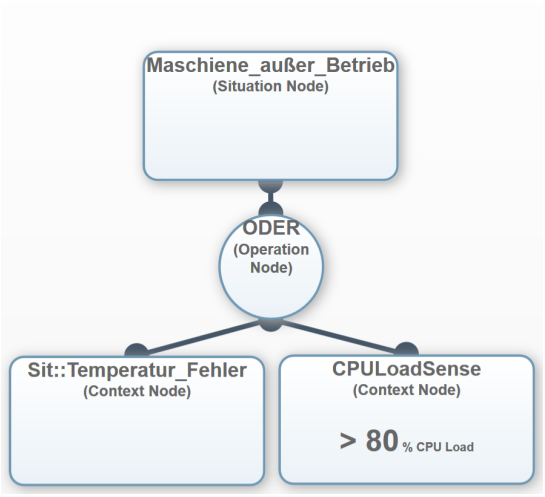

Abbildung 4.4.: Die Situation, die den Betriebstopp einer Maschine modelliert. Vorhandenes Temperatur-Fehler-Template wird aus dem Template-Repository geladen.

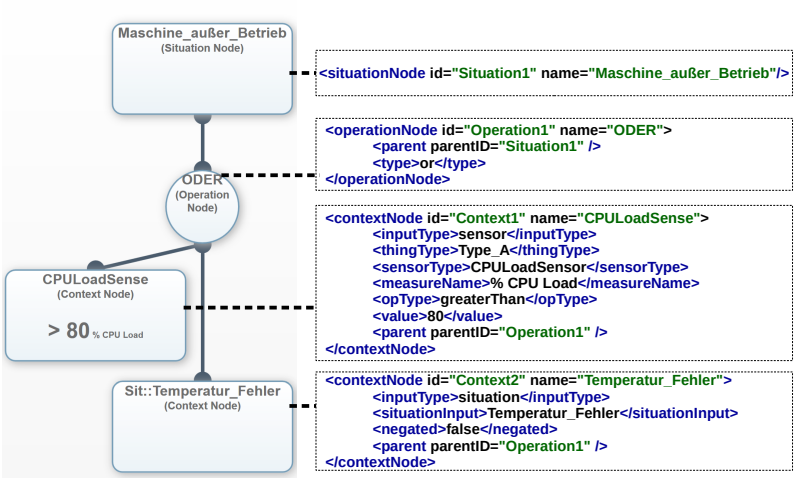

Abbildung 4.5.: Das Template Maschine außer Betrieb in der grafischen (Modellierungstool) und der schematischen (XML) Darstellungen.

wird dieser Bezeichner genutzt, um die XML-Repräsentation des Eingabe-Templates abzurufen und zu verarbeiten.

Eine weitere Optimierung des XML Schemas für Situationstemplates besteht in Zusammenführung der Kontext-Knoten und Kondition-Knoten zu Kontext-Kondition-Knoten. Dadurch, dass zwischen Kontext-Knoten und Kondition-Knoten immer eine 1 zu 1 Relation besteht, ist es sinnvoll diese Knotenarten zu verschmelzen. Der neue Kontext-Kondition-Knoten hat die gleiche semantische Bedeutung, wie die Konkatenation von alten Kontext-Knoten und Kondition-Knoten. Somit

existieren nur drei erlaubte Datentypen in dem XML-Schema: Situationen, Operationen, Kontext-Lieferanten. Die Kontext-Lieferanten können Sensoren oder Situationen sein.

4.3. Datenbank

Die Datenbank ist der Behälter für die alle Ressourcen, welche im Laufe des Situationserkennungsprozesses benötigt werden. Zugriffe auf die Datenbank erfolgen sowohl während der Template-Modellierung (z.B. für die Auswahl der Eingabe-Templates oder Thing-Typen), als auch zur Laufzeit, während der Realisierung von Templates (z.B. Auswahl der Things, dessen Sensordaten verwendet werden). In Form der Einträge in der Situationen-Collection (siehe Abschnitt 3.1) werden die Ergebnisse der Situationserkennung festgehalten.

Der innere Aufbau der zentralen Datenbank besteht aus Collections, die die gleichnamigen Ressourcen beinhalten. Die Ressourcen sind miteinander logisch verbunden und ergeben somit ein Datenmodell. Dieses Datenmodell wird in [Mor15] ausführlich beschrieben. Für die Zwecke der vorliegenden Arbeit wurden an dem beschriebenen Datenmodell Veränderungen vorgenommen. Es wurde eine neue Ressource ThingType eingeführt, die den Typ der physischen Objekte beschreibt. ThingType wird benutzt wenn ein Situationstemplate in eine ausführbare, erkennbare Form transformiert wird: ThingType gibt an welche Things in dem gewählten Template benutzt werden können. Zusätzlich wurden einzelne Attribute der anderen Ressourcen (Thing, Situation, Situationstemplate) verändert.

Auf die Eigenschaften einzelner Ressourcen, sowie deren wichtige Attribute wird in folgenden gleichnamigen Teilkapiteln im Detail eingegangen.

4.3.1. Things

Ein Thing ist ein physisches Objekt, welches im Rahmen der Situationserkennung überwacht wird und seine Sensordaten (dynamische Kontextdaten) als Kontext liefert. Die Attribute von einem Thing und die Sensordaten bilden gemeinsam ein virtuelles Objekt, welches die Ereignisse um das physische Objekt in digitaler Form repräsentieren.

Ein Thing besitzt die folgenden Attribute: **Name, Edge-IP-Adresse** und den **ThingType**. Die IP-Adresse des Edge-Nodespunktes in welchem sich das aktuelle Thing befindet. Dieses Attribut ist von zentraler Bedeutung für die Verteilung der Erkennung auf Edge und Cloud. Alle Situationen mit Things, welche anstelle von diesem Attribut null-Wert haben, können nur in der Cloud erkannt werden. Das Attribut ThingType gibt an welche Sensor-Arten das Thing besitzt. ThingType nimmt einen Wert ein, der einem Eintrag der ThingType-Collection entspricht. Die Koordinaten eines Things werden im Vergleich zu der Ausgangsarbeit nicht gespeichert, denn der Standort des Objekts ist für die Verteilung nicht von Bedeutung. Die Edge-IP reicht für die Verteilung aus

4.3.2. ThingType

ThingType-Collection ist eine Auflistung von verschiedenen Arten von Sensoren. Diese Collection wird im Rahmen der vorliegenden Arbeit eingeführt. Einträge in diese Datenbank-Collection

müssen durch Domänenexperten manuell hinzugefügt werden. Jedes Thing hat eine eindeutige Typ-Zuordnung. Zusätzlich liefert die ThingType-Collection Information über verfügbare Maßeinheiten für jeweilige Sensorart: z.B. ein Temperatur-Sensor kann Kontext-Werte in Celsius, Kelvin und Fahrenheit bereitstellen. Der ThingType wird durch das Attribut **sensors** beschrieben, welches sich aus der Info über Sensor-Art und Sensor-Messeinheit zusammensetzt.

Diese Entität ersetzt die Sensor-Ressource der Ausgangsarbeit. Bei der Modellierung von einem Situationstemplate wird die Information über die ThingTypes einzelner Kontexte durch definiert. Dann wird die Thing-Auswahl zur Laufzeit (bei der Instanziierung der Kontexte) auf nur erlaubte/vorgesehene Things beschränkt.

4.3.3. Situation und Situationstemplate

Die Situationen-Collection stellt die Ergebnisse des Erkennungsprozesses dar und beinhaltet erstellte Situationsobjekte. Die Einträge in diese Collection werden automatisch gemacht, sobald alle Sensordaten einer Situation vorliegen und entsprechend der modellierten Bedingungen und Operationen überprüft werden. Ein Situationsobjekt beinhaltet Information über das Template, die beteiligten physischen Objekte und die Zeit des Erkennung. In Situationsobjekten werden die Kontextdaten, dir zu der Erstellung von diesem Objekt geführt haben mitgeliefert. Die Attribute (**Name, Thing, Situationtemplate, Occured, Sensorwerte, Timestamp**) haben sich im Vergleich zu der Ausgangsarbeit nicht geändert. Allerdigs spielt das Attribut **Quality** (Qualität) im Rahmen der vorliegenden Arbeit keine Rolle. Das Attribut **Occured** ist besonders wichtig, denn das sagt aus ob die Situation eingetreten ist. Dieses Attribut kann separat von Anderen genutzt werden, wenn nur wichtig ist, ob eine Situation eingetreten ist oder nicht und kein Bedarf an konkreten Werten besteht.

Die in Situation Template Modeling Tool von Analytikern und Domänenexperten erstellten Vorlagen werden in der Situationstemplates-Collection abgelegt. Jedes Situationstemplate-Objekt wird als Anhang gespeichert. Durch einen Zugriff können aus dem XML-Anhang alle für Erkennung nötigen Daten abgelesen Werden. Speicherung als Anhang erspart zusätzliche Verarbeitungsschritte (JSON -> XML Transfprmation), die bei einer Speicherung als Attribut nötig wären. Neben dem **_attachment**-Attribut gibt ein Situationstemplate den ThingType vor. Davon hängt die erlaubte Auswahl an Things ab, die zur Laufzeit für Überwachung gewählt werden.

4.3.4. Sensoren und Sensorwerte

Diese Ressourcen/Entitäten der Ausgangsarbeit, sind in dem Datenmodell dieser Arbeit als Attribut in anderen Entitäten vorhanden. So dient Sensor als Attribut von ThingType und Sensorwerte sind ein Attribut von Situation.

Die Relationen der Ressourcen lassen sich durch ein ER-Diagramm (siehe **??**) beschreiben. Jedes Thing enthält ein ThingType. Ein ThingType kann mehrere Sensoren enthalten. Das Situationstemplate gibt vor, welche ThingTypes verwendet werden. Zusammen mit einem oder mehreren Things werden Instanzen von Situationen gebildet und als Situationsobjekte gespeichert. Die Sensoren produzieren Sensorwerte, welche in einem Situationsobjekt gespeichert werden.

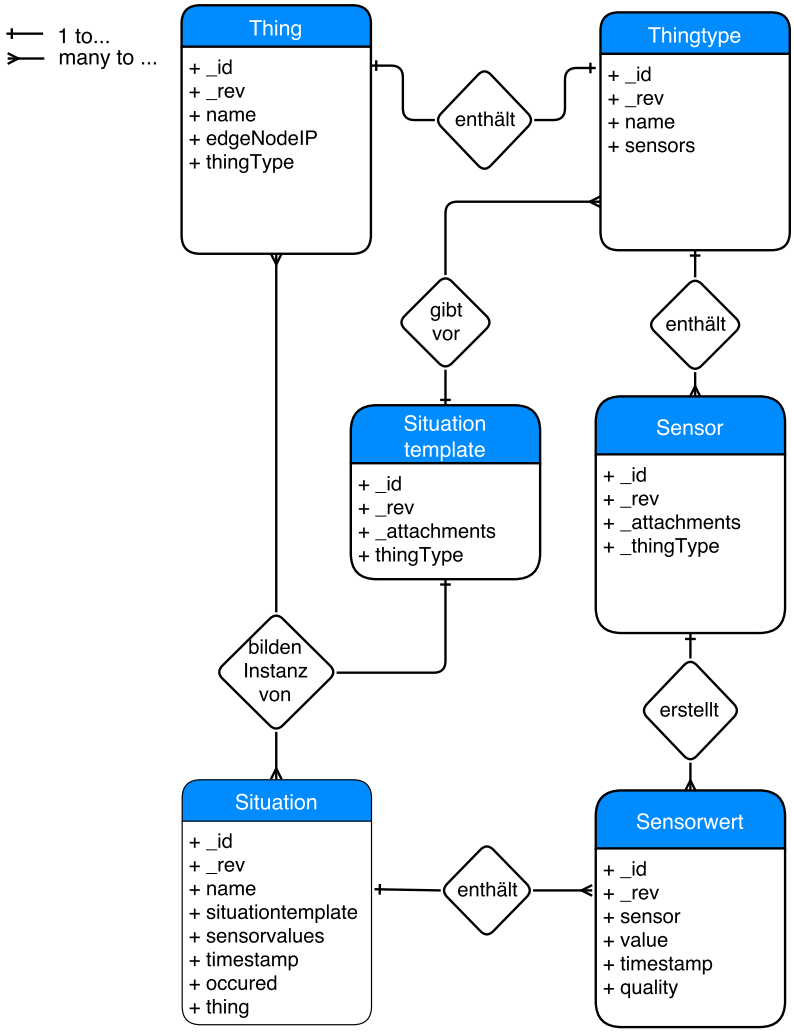

Abbildung 4.6.: Datenmodell (angelehnt an Modell aus [Mor15])

Im Vergleich dazu stehen die Ressourcen-Beziehungen des Modells aus der Ausgangsarbeit [Mor15]. Diese wurden in Abbildung 2.5 vorgestellt.

4.4. Verteilung und Mapping: Edge vs Cloud

Der eigentliche Startpunkt im Situationserkennungsprozess ist die manuelle Auswahl eines Templates (aus der Datenbank). Dieses wird durch Verteilungs- und Mapping-Komponenten auf reale Edge- oder Cloud-Knotenpunkte übertragen. Die Verteilungs-Komponente (ferner Verteiler) hat zwei Aufgaben: Auswahl von Things und Knotenpunkt-Verteilung (Edge vs Cloud). Im Rahmen der ersten Aufgabe ist der Verteiler dafür zuständig für jede Sensor-Kontext Eingabe eine manuelle Thing-Auswahl zu initiieren. Zweite Aufgabe von dem Verteiler besteht darin, die Elemente des Templates zu untersuchen und analysieren. Auf der Basis dieser Analyse entscheidet der Verteiler, ob ein Template-Element auf einem Edge-Nodespunkt oder auf einem Cloud-Knotenpunkt realisiert werden kann. Die Verteilungskriterien, die für diese Entscheidung ausschlaggebend sind,

werden im Weiteren, in einem separaten Abschnitt vorgestellt (??) Das übergeordnete Ziel ist, die Teilaufgaben des Situationserkennungsprozesses auf Edge-Nodes zu verteilen, um dadurch die Vorteile von Edge Computing zu sichern.

Der Mapper benutzt die Auswahl der Things und gegebenenfalls die Knotenpunkt-Information und erstellt eine Abbildung des schematischen Modells auf einen Flow der Node-Red-Engine. Es lassen sich zwei konzeptionell unterschiedliche Szenarien erkennen: modulare und ad-hoc Verteilung. Aus der Kombination bzw. der Konkatenation dieser Fälle kann ein weiteres Szenario abgeleitet werden - Hybrid Verteilung. Die Unterscheidung wird durch den Aufbau des modellierten Templates und die Nähe (angegeben durch die IP-Adresse) der ausgewählten Things zu Edge-Nodes bestimmt. Je nach Szenario variiert der Einsatz und die Aufgabe der Verteilungs- und der Mapping-Komponente.

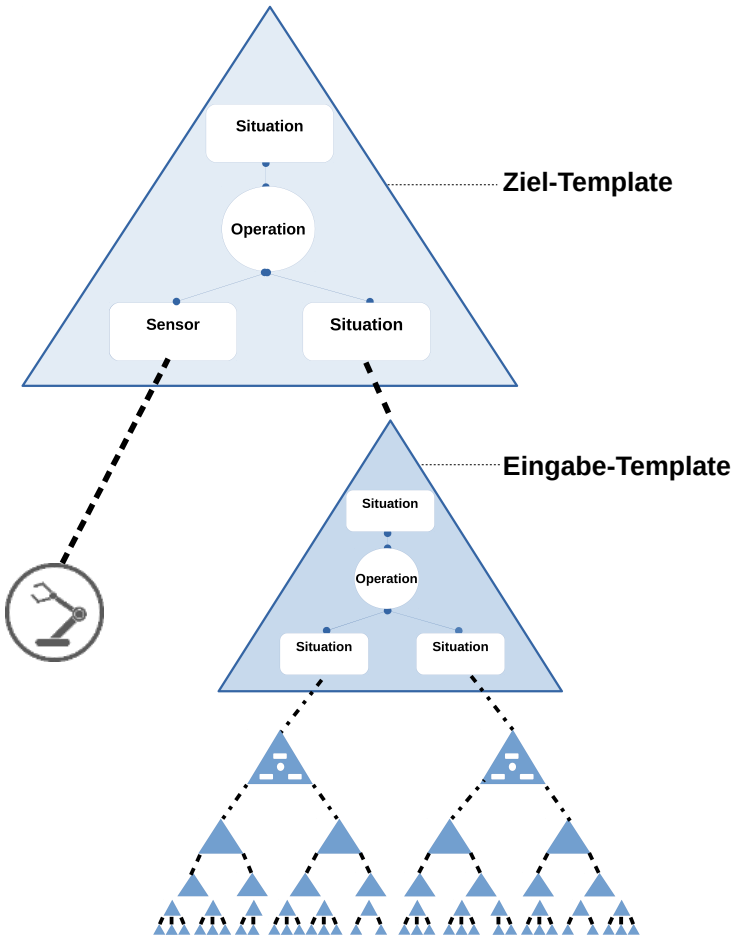

Abbildung 4.7.: Darstellung von einem mehrfach verschachtelten Template.

Modulare Verteilung Prinzipiell beschreibt dieses Szenario einen Fall, in dem das *Ziel-Template*, welches die zu erkennende Situation modelliert, mindestens ein weiteres Situationstemplate als Eingabe-Element hat - das *Eingabe-Template*. Demnach, existiert in der Datenbank bereits ein Eintrag mit dem Eingabe-Template. Zusätzlich kann das Zieltemplate eine beliebige Anzahl an Eingabe-Elementen von Typ Sensor haben. In dem Verteiler werden die Things ausgewählt und es wird entschieden, ob die Erkennung in der Cloud oder auf Edge erfolgt. Diese Entscheidung wird auf Grund dem 'edgeNodeIP'-Eintrag in der Things-Collection (Edge-IP-Adresse) der Datenbank getroffen. Diese Art der Verteilung kann als modular bezeichnet werden, denn die Eingabe-Situationen sind bereits definiert und können separat voneinander auftreten. Die Herausforderung in diesem Szenario ist es, die potentiell ineinander verschachtelte Situationstemplates (siehe Abbildung 4.7) zu identifizieren und korrekt darzustellen. Dieser Fall tritt ein, wenn ein bereits erstelltes Eingabe-Template seinerseits ein weiteres Situationstemplate als Eingabe hat und dadurch ein weiterer Level der Rekursion entsteht. Auf dem tiefsten Rekursionslevel ist ein Template, das ausschließlich Sensor-Kontext-Elemente beinhaltet.

Ein Workflow (siehe Abbildung 4.8), der die Funktionsweise der modularen Verteilung beschreibt beginnt im Situation Template Modeling Tool. Dort wird ein Template erstellt und anschließend in der Datenbank gespeichert. Dieses Template kann nun direkt als Vorlage für Situationserkennung dienen. Zusätzlich gibt es auch die Möglichkeit dieses Template als ein Bestandteil für weitere komplexere/größere Situationen zu verwenden. Ein komplexes Template greift auf die Datenbank zu und kann die dort abgelegten primitiveren/kleineren Templates als Kontext-Lieferant verwenden. Derartige Eingabe-Templates müssen erkennen, ob die durch sie modellierte Situation eingetreten ist oder nicht und einen entsprechenden boolschen Wert zurückgeben. Soll das komplexe verschachtelte Situation-Template realisiert werden, so muss die XML-Datei, die das Template enthält, durch den Distributer/Mapper bearbeitet werden. Dabei wird die XML-Datei geparst. Zunächst wird festgestellt, ob das Template einen Sensor-Kontext beinhaltet. Ist das der Fall, so wird für jedes Sensor-Element ein Thing ausgewählt und auf Node-Red-Objekte abgebildet. Die boolschen Operationen, die die Elemente verbinden und ein Ergebnis-Publisher werden ebenfalls im Node-Red Flow dargestellt. Im Folgenden findet die Überprüfung statt, ob das Ziel-Template Eingabe-Templates beinhaltet und für diese werden in Node-Red Subscriber-Elemente erstellt. Im Weiteren folgt die Verbindung der erstellen Flow-Bestandteile miteinander gemäß der Struktur des Ziel-Template. Der Inhalt des Eingabe-Templates wird von der Datenbank abgerufen und dient zusammen mit dem Topic des entsprechenden Node-Red-Subscriber als Eingangsparameter für den rekursiven Aufruf der Verteilungs- und Mapping-Komponente.

Ad-Hoc-Verteilung Anders als in dem modularen Fall ist in einem Ad-Hoc-Verteilungsszenario die dynamische Identifizierung von möglichen Teilmodulen eines Templates die zentrale Herausforderung. Das bedeutet, dass die Elemente des Ziel-Templates keine Eingabe-Templates sind. Es wird das Ziel verfolgt die Vorteile von Edge Computing, wenn möglich, zu nutzen anstatt die komplette Datenverarbeitung in der Cloud vorzunehmen. Dafür sollen mehr Informationen auf den Edge-Nodes verarbeitet werden. Dies ist nicht immer möglich. Die Entscheidung Edge oder Cloud wird in Abhängigkeit davon gemacht, welche Verteilungskriterien momentan wichtiger sind.

Ein Thing hat einen Eintrag (in der Datenbank) 'edgeNodeIP', welcher die IP-Adresse angibt, unter welcher die Sensor-Kontextdaten von diesem Thing verarbeitet werden können. Things dessen 'edgeNodeIP' auf **null** gesetzt ist, müssen ihre Sensor-Kontextdaten an die Cloud-Instanz

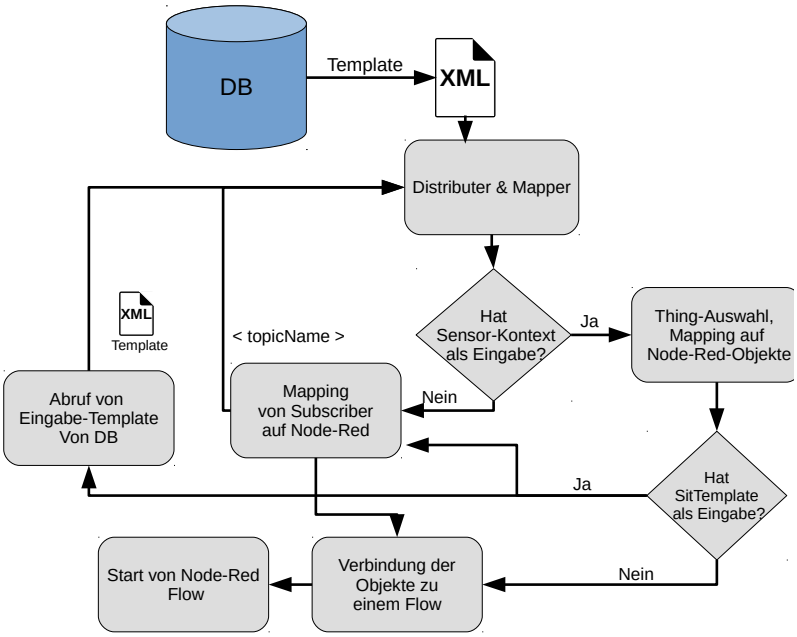

Abbildung 4.8.: Darstellung der Aktivitäten im Rahmen eines modularen Workflow.

zur weiteren Verarbeitung weiterleiten. Bei der Ad-Hoc-Verteilung wird nach der Auswahl von Things eine Analyse aller beteiligten Edge-IP-Adressen durchgeführt. Ziel der Analyse ist die Identifizierung von Sub-Templates, welche Teile des gesamten Templates repräsentieren und zusätzlich nur die Things mit gleicher IP-Adresse beinhalten.

So sind z.B. zwei Things, die über dieselbe IP-Adresse verfügen, mit demselben Edge-Node-Server verbunden. Alle Situationen, die durch Sensorwerte-Erfassung dieser Things entstehen, können direkt auf diesem Edge-Server erkannt werden.

Die Schwierigkeit des Ad-Hoc-Falls ist die dynamische Identifizierung und Gruppierung der Teilelemente, welche auf Grund deren Eigenschaften zusammengehören können. Diese Elemente können als temporäre, zur Laufzeit erstellte Sub-Templates dargestellt werden. Dafür muss der Verteiler die XML-Repräsentation des Ziel-Templates ad-hoc bearbeiten. So können zu Laufzeit mehrere Sub-Templates auf der Basis von Information über die Edge-Infrastruktur erstellt werden (siehe Abschnitt 4.4). Die ad-hoc identifizierten Teil-Templates können optional in der Datenbank gespeichert werden und ferner separate Situationen modellieren.

Hybride Verteilung Als Kombination aus den zwei zuvor beschriebenen Verteilungsarten lässt sich das Hybrid-Szenario darstellen. In diesem Fall existiert ein Template sowohl aus bereits definierten Templates (oder aus temporären in der Ad-Hoc-Phase erstellten Templates) als auch aus noch zu analysierenden Things. Dabei muss sowohl der potentiell rekursive Aufbau der modularen Templates als auch die Infrastruktur-Analyse der Edge-Nodes beachtet werden. Dieses Szenario beschreibt keine konzeptionell neue Arbeitsweise des Verteilers sondern Konkatenation der beiden zuvor definierten Verteilungsarten. Zunächst werden für alle Sensor-Kontext-Eingaben,

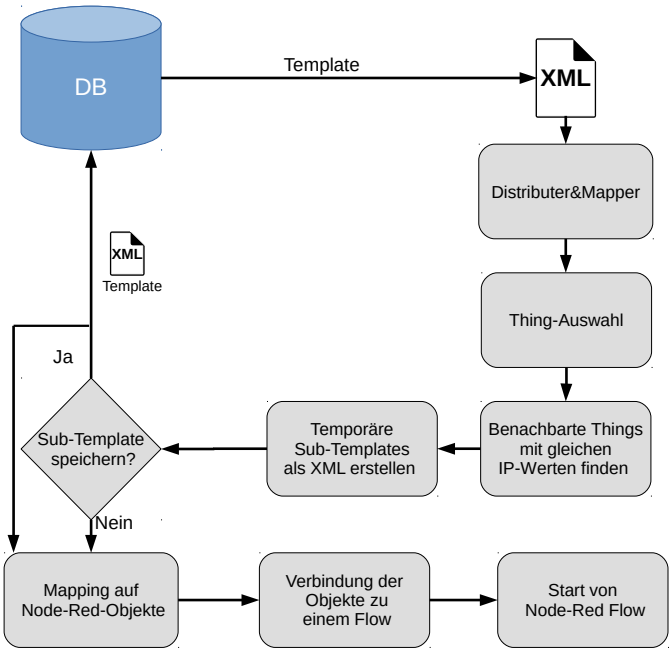

Abbildung 4.9.: Darstellung der Aktivitäten im Rahmen eines Ad-hoc Workflow.

die Things deklariert und es wird zunächst die Ad-Hoc und anschließend die modulare Verteilung angewendet (siehe Abbildung 4.10). Dieses Szenario kann potenziell häufig vorkommen, denn es bietet eine benutzerfreundliche Modellierung an: Sowohl Template-Wiederverwendung als auch dynamische Template Erstellung werden angeboten. Die dadurch verursachte erhöhte Komplexität der Verteiler-Funktion ist für den Benutzer transparent.

4.5. Verteilungskriterien

Bislang wurden Verringerung von Datenverkehr und Latenz sowie Erhöhung der Security als Berechtigung für Verteilung erwähnt. Diese Vorteile kann die Verwendung von Edge Computing mit sich bringen. Jedoch müssen dafür weitere zusätzliche Konditionen erfüllt werden. Ein weiteres Entscheidungskriterium, ob die Verteilung vorzunehmen ist, ist der Kompromiss zwischen Overhead für Verteilung und den Edge Computing Vorteilen.

4.5.1. Latenz

Wie bereits bei der Einführung zu dieser Arbeit an einem Beispiel gezeigt wurde, kann die Benutzung von zentralisierter Situationserkennung zur Erhöhung der Antwortzeiten führen. Das passiert, wenn sich die Things, die im Rahmen einer Situationserkennung auf dem gleichen Edge-Nodes (lokalen Server) befinden, mit den Applikationen, die auf die zu erkennende Situation reagieren. Wenn der lokale Server die Situationserkennung eigenständig (nicht mit Hilfe der

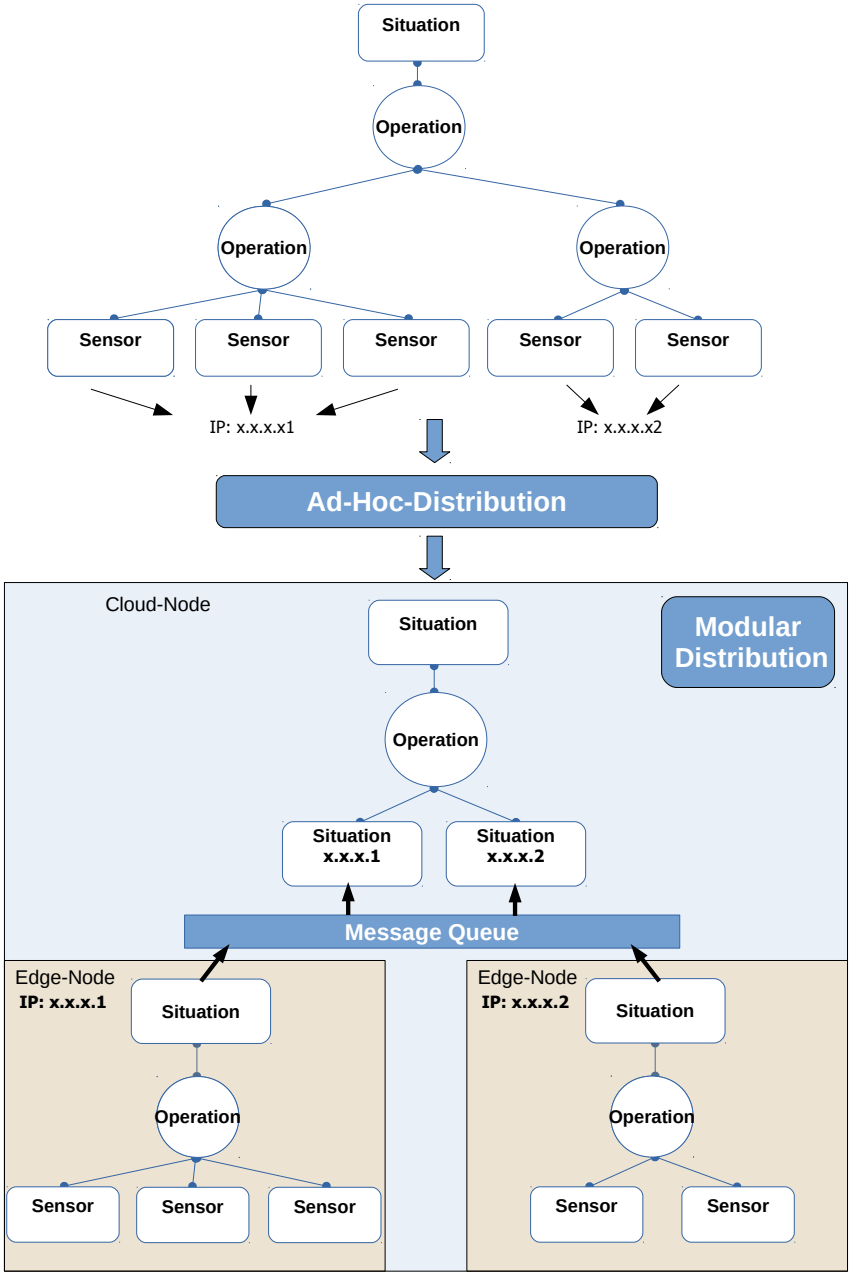

Abbildung 4.10.: Hybridverteilung: Konkatenation von ad-hoc und modularen Verteilung.

Cloud-Zentrale) vornehmen könnte, so würde sich die Antwortzeit der lokalen Anwendungen verringern: Die erkannten Situationen werden direkt am lokalen Server vorliegen und nicht mehr ausschließlich in der Cloud. Die lokal erkannten Situationen werden nach wie vor in die Cloud-Zentrale gesendet - für weitere potentiell interessierte Anwendungen. Für nicht lokale Anwendungen (Anwendungen auf anderen Edge-Nodes oder in der Cloud) ändert sich die Latenzzeit nicht.

4.5.2. Bandbreiteneinsparung

Nur durch die verteilte Situationserkennung ändert sich nichts an dem Problem der großen Menge an Daten, die zwischen Edge und Cloud ausgetauscht wird. Der Grund dafür ist, das Vorhandensein der Sensorwerte im Körper des Situationsobjekts: Die Kontextdaten, die zur Erstellung des Objekts geführt haben, werden in die Cloud für die Speicherung mitgesendet. Dadurch, dass die Situationsobjekte auch erstellt werden, wenn die Situation nicht eingetreten, ist offensichtlich, dass große Menge der Sensorwerten, in gleicher Weise, wie vor der Verteilung, durch das Netzwerk fließen und die Bandbreite ausnutzen wird. Um dieses Problem zu adressieren, werden zwei Konzepte vorgeschlagen: Kontextstripping und Situationen-Logs.

Situation-Log

Wenn eine Situation auf einem Edge-Nodes erkannt wird, kann dieses Ereignis in einem Log-Eintrag festgehalten werden. In einer Synchronisation-Phase können die Log-Einträge in der zentralen Datenbank gespeichert werden. Je nach Situationstemplate kann die synchronisierende Information sofort nach dem Eintritt der Situation oder in regelmäßigen Zeitabständen via Message-Queue in die Datenbank gesendet werden. Dieses Vorgehen ist nur im Falle von idempotenten Situationen möglich. Das heißt, dass einmaliges und mehrfaches Eintreten einer Situation semantisch gleich sind. Es ist möglich alle Situationsobjekte, die innerhalb einer definierten Zeitspanne eingetreten sind, mit einer logischen ODER-Operation zu vereinen und das Ergebnis in der zentralen Datenbank abzuspeichern. Dadurch wird der Speicheraufwand verringert und dennoch konstante Überwachung eines physischen Objektes garantiert.

Der Log kann nützlich sein, um den Datenverkehr zu verringern und dennoch die Erkennung nicht zu unterbrechen. Für die Situationen, nach Erkennung von welchen nicht unmittelbar eine Reaktion folgen muss, ist Logging sehr sinnvoll. Jedoch ist Logging bei kritischen Situationen nicht anwendbar, denn auf die kritischen Situationen/Zustände muss eine zeitnahe/schnelle Reaktion folgen. Die Synchronisationsabstände müssen so gewählt werden, dass das System auf jeden Fall genug Zeit hat, um die Reaktionsmechanismen vollständig durchzuführen. Zum Beispiel kann eine Situation *Roboter-Akku-50%* einmal in 60 Minuten synchronisiert werden, wenn die benötigte Auflade-Zeit eine Stunde beträgt und die zur Hälfte entladene Batterie zwei Stunden Betriebszeit garantiert: Sogar im ungünstigsten Fall wird eine rechtzeitige Reaktion folgen. Dementsprechend sind die Synchronisationsphasen für alle Situationen, für die Logging anwendbar ist, zu wählen.

Kontextstripping

Um den hohen Datenverkehr zu verringern kann in Kombination mit Verteilung Kontextstripping verwendet werden. Das heißt, dass die Situationsobjekte nicht vollständig in die Cloud gesendet werden, sondern nur auf das Attribut *occured* reduziert werden. Dadurch ist in der Cloud die Information vorhanden, falls eine Situation erkannt wurden, jedoch nicht bekannt welche Werte es provoziert haben. Kontextstripping ist anwendbar, wenn das Zustandekommen der Situation nicht wichtig ist, sondern nur die Feststellung ausreicht, dass sie eingetreten ist. Im Rahmen dieser Arbeit wird Kontextstripping auf allen Eingabe-Situationen angewendet: Die Ziel-Situation bekommt nur einen boolschen Wert, der aussagt, dass die Eingabe-Situation eingetreten(wenn true)/nicht eingetreten (wenn false) ist.

4.5.3. Overhead

Eine zusätzliche Aufgabe beim Systementwurf ist Kompromissfindung zwischen dem Grad der Verteilung und dem Aufwand für die Umsetzung der Verteilung. Es muss die Frage beantwortet werden, ab welcher Anzahl der Kontextelemente von Typ Sensor sich eine Verteilung auf Edge lohnt Unter Overhead ist in erster Linie der Aufwand, den der Message Broker betreibt, zu verstehen: Für jede verteilte Situationserkennungsinstanz werden Publisher, Subscriber und Topics benötigt. Die Aufgabe die Nachrichten zu verarbeiten (korrekt weiterzuleiten) liegt ausschließlich beim Broker. Also muss die Entscheidung getroffen werden, wie groß (hinsichtlich der Anzahl von Sensor-Kontext Elementen) das kleinste Template sein darf, das auf einem Edge-Nodes erkennbar ist.

Diese Entscheidung wird von einem Einsatzbereich zum anderen variieren. Wenn die Security von kritischer Bedeutung ist, so wird Maximierung der Verteilung bevorzugt. Wenn der Aufwand von Message Broker zu sparen ist, sollen nur große Situationen auf Edge erkannt werden.

Im Rahmen dieser Arbeit darf das kleinstmögliche auf einem Edge-Nodes erkennbare Template zwei Sensor-Kontext Elemente beinhalten. Diese Granularitätsstufe erlaubt es nicht, dass einzelne Sensor-Kontext Elemente auf Edge erkennbar sind. Das Minimum von zwei Sensoren wurde frei gewählt um Erkennung von kleinen Situationen zu erlauben, jedoch nicht die kleinstmögliche Granularitätsstufe von eins zu nehmen. Erkennung von Situationen, welche nur einen Sensor überwachen ist im Rahmen dieser prototypischen Implementierung unnötig, denn die Implementierung von Applikationen, die auf Erkennung warten, ist nicht Teil dieser Arbeit. Der Aufwand für eine feingranulare Verteilung bleibt dabei ungenutzt.

4.5.4. Security

Sicherheitsbedenken, die mit der Verwendung von Cloud verbunden sind, können durch Verteilung auf Edge teilweise minimiert werden. Dieser Vorteil ist unmittelbar mit Verringerung des Datenverkehrs in die Cloud verbunden. Die Bedenken über Sicherheit entstehen bei der Benutzung von sensiblen Daten. Ein Beispiel dafür ist SMART Home. Das ist eine IoT-Umgebung in welcher viele private sensible Daten ausgetauscht werden. Wenn die sensiblen Daten nicht über das Netzwerk zu der Cloud-Instanz fließen müssen, sondern auf dem lokalen Server (innerhalb der SMART Home Umgebung, z.B. im Haus selbst) verarbeitet werden, erhöht sich die Sicherheit

des Systems. In Verbindung mit Kontextstripping kann die Erhöhung der Sicherheit realisiert werden, denn die Kontextinformation (die auch sensibel sein kann - siehe SMART Home) nicht die Edge verlassen muss.

Sicherheitsbedenken können auch den Grad der Verteilung beeinflussen. Die Gewährleistung der Sicherheit kann dazu führen, dass kleine Situationen (z.B. bestehend aus nur einem Sensor) durch Situationserkennung auf Edge erfasst werden. Das ist der Fall, wenn die Sensordaten nicht in die Cloud gelangen sollen - dafür wird erneut Kontextstripping verwendet.

4.6. Message Queue

Benutzung von Message Queue als Interprozesskommunikationsmedium ist eine Veränderung des vorigen Konzepts, die im Rahmen der vorliegenden Arbeit eingeführt wurde. Damit wurde die Ressource Management Platform (RMP) ersetzt. Ein Grund für die Ersetzung ist die Verringerung der Komplexität der Architektur. Dadurch, dass die RMP nicht zusätzlich zu der Datenbank und den Erkennungssystemen aufgesetzt werden muss, wird das System einfacher und übersichtlicher. Der andere Grund Message Queueing einzuführen ist die Tatsache, dass Message Queue ein standardisiertes Konzept für M2M-Kommunikation ist.

Im Kontext der Situationserkennung ist Message Queueing besonders gut einsetzbar. Da ein physisches Objekt die Sensordaten potenziell an mehrere Interessenten (Überwachende Objekte) schicken muss, kann der Aufwand für die Speicherung und Verwaltung von Kontaktinformationen sehr hoch werden. Wenn ein Thing seine Sensordaten über die Message Queue unter einem bestimmten Topic veröffentlicht, kann sich jedes Erkennungssystem, das diese konkreten Sensordaten überwachen will, als Subscriber für dieses Topic anmelden. Dadurch, dass der Thing-Publisher nicht weiß welche Erkennungssysteme die Sensordaten überwachen, müssen keine zusätzlichen Datenlieferung-Mechanismen entworfen werden.

Die Aufgabe, die Nachrichten an die richtigen Subscriber zu liefern wird von Message Broker übernommen. Der Message Broker kann sich sowohl auf einem Edge-Nodes als auch in der Cloud befinden. Das hängt von dem Standort des Erkennungssystems ab. Unterschiedliche Erkennungssysteme können ihrerseits untereinander kommunizieren.

4.6.1. Identifizierung der Situationen

Benutzung von Message-Queue ist mit dem Veröffentlichen und Abonnieren der Nachrichten unter bestimmten Topics verbunden. Eindeutige Identifizierung der erkannten Situationen ist eine Teilaufgabe beim Systementwurf. Ein einfaches Beispiel von einem Fehlverhalten: Wenn die Topics nach den Situationstemplates benannt werden, so ist es nicht möglich zwei Situationen voneinander zu unterscheiden, die das gleiche Situationstemplate, jedoch verschiedene Things verwenden.

Eine Möglichkeit, dieses Problem zu beseitigen ist die eindeutige Namensgebung für Topics, auf welchen Situationsobjekte veröffentlicht werden. So kann beispielsweise die Konkatenation der Namen des Templates und der beteiligten Things gemacht werden:

Name des Templates: «*Maschine-Überhitzt*»;

Namen der Things: «*Roboter#1, Roboter#2*»;

Name der Publish/Subscribe-Topics: «*Maschine-Überhitzt:Roboter#1&Roboter#2*»;

Diese Methode kann eindeutige Erkennung des benötigten Topics garantieren, jedoch verringert sich die Übersicht mit Erhöhung der Anzahl der beteiligten Things.

Eine Alternative zu der Konkatenation der Namen der Komponenten ist die Erstellung eines Verzeichnisses (ferner Registry), in welchem die Information über alle aktuell laufenden Erkennungen enthalten ist. Die Registry kontrolliert die Situationserkennungen auf allen Knoten des Netzwerks. Ein Eintrag der Registry beinhaltet den Namen der Situation, welche auf einem Knoten (Edge/Cloud) erkennbar ist und die Namen der Things, die mit dieser bei der Instanziierung dieser Situation gewählt wurden. Der Nutzen der Registry geht über die Identifizierungsaufgabe hinaus: Zusätzlich kann mittels Registry-Einträge eine bereits laufende Erkennung mit als Teil der Erkennung einer komplexeren Situation verwendet werden. Zum Beispiel wird die Situation «*Maschine-Überhitzt:Roboter#1&Roboter#2*»auf einem Netzwerk-Knoten erkannt. Es wird eine neue komplexere Situation «*Maschine-Nicht-Funktionsfähig*»modelliert, welche (unter anderen) die Kontext-Lieferant Situation «*Maschine-Überhitzt:Roboter#1&Roboter#2*»beinhaltet. Durch den Registry Eintrag wird die bereits laufende Instanz dieser Eingabe-Situation verwendet und keine redundante erstellt. Dadurch tritt der Nutzen von Message Queueing zum Vorschein: Lose Kopplung von Publish-Subscribe-Technologie erlaubt mehrfache Verwendung derselben Situationen.

5. Implementierung

Für die in dem Kapitel 4 beschriebene Architektur wurde ein Prototyp implementiert. Mit diesem Prototyp wird die die Situationserkennungsebene von SitOPT realisiert. Die entworfene Architektur umfasst eine Datenbank, Modellierungsumgebung einen Message Broker, Verteiler und Mapper. Die Funktion der einzelnen Komponenten, sowie die Details deren Umsetzung werden in diesem Kapitel vorgestellt.

5.1. Datenbank

Als Datenspeicherung und -Verwaltungssystem wurde in diesem Projekt die CouchDB eingesetzt. CouchDB eignet sich für die Umsetzung der vorgestellten Konzepte. Obwohl es eine Open Source Software ist, wird CouchDB aktiv vorangetrieben und verfügt über die meisten gängigen NoSQL-Features. Innerhalb eines IoT Systems entstehen sehr viele Kontextdaten, die gespeichert werden müssen. CouchDB (als Vertreter von NoSQL-Datenbanken) kann große Datenmengen verarbeiten.

Für die vorliegende Arbeit sind die Schemalosigkeit und RESTful API von CouchDB besonders attraktiv. Dank Schemalosigkeit können heterogene Daten gespeichert werden. In flexiblen IoT Architekturen ist es ein Vorteil. REST-API ist ein angenehmer und geeigneter Weg um mit der Datenbank zu kommunizieren. Es geht sowohl aus der Modellierungsumgebung (mit JavaScript-Queries) als auch aus dem Verteiler (über JavaEE MEthoden).

5.2. Situation Template Modeling Tool

Die grafische Modellierungsumgebung für SitOPT [GSS15] wurde im Rahmen der Implementierung der verteilten Situationserkennung erweitert. Die Implementierung der Template-Modellierung umfasst die Methoden zur Modellierung von Templates; Validierung der modellierten Templates; Speicherung der Templates in der Datenbank; Laden der Templates aus der Datenbank; Importieren und Exportieren der Templates in XML-Format.

Ein Teil dieser Arbeit war die Konzipierung und Entwurf eines neuen Schemas für Situationstemplates. Das entstandene Schema erlaubt Benutzung der schon existierenden Templates für die Modellierung der neuen Templates. Durch die Erweiterung des Schemas für Situationstemplates, ist eine Interaktion zwischen dem Modellierungswerkzeug und der Datenbank notwendig. Zur Realisierung dieser Funktion wurden die HTTP-Requests der CouchDB REST API benutzt.

Nach der Erstellung von einem Situationstemplate kann es in die Datenbank eingefügt werden. Dafür wird ein POST-Request an die CouchDB-Instanz gesendet. Das ID für das Template vergibt der Benutzer. Die Erstellung des Dokuments verläuft in zwei Schritten. Zunächst wird das neue

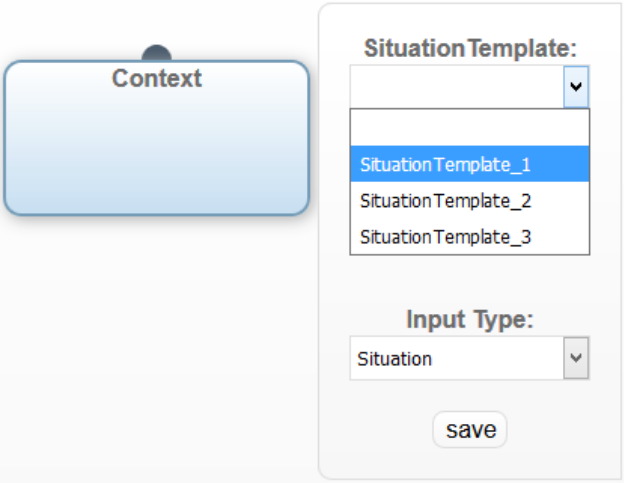

Abbildung 5.1.: Zugriff auf Stuationtemplates-Collection: Dropdown mit existierenden Templates.

Dokument mit dem vergebenen ID, über einen Aufruf von jQuery-Ajax, erstellt. Innerhalb des ersten Ajax-Aufrufs wird ein weiterer synchroner PUT-Aufruf gemacht um das bereits erstellte Dokument zu updaten - das Template in XML-Format wird angehängt. Die jQuery Aufrufe sind für den Benutzer transparent. Die Erstellung des Dokuments wird in der Modellierungsumgebung on Form von einer *alert*-Nachricht bekanntgegeben. Ein Fehler löst ebenfalls eine *alert*-Nachricht aus. Beim Laden des Dokuments aus der Datenbank wählt der Benutzer das Template aus einer Dropdown-Liste aus. Das gewünschte Template wird abgerufen: HTTP-GET-Anfrage für den Anhang des Dokuments mit der ausgewählten ID.

Die Schemaerweiterung erlaubt die Benutzung von Situationen in der Rolle eines Kontextlieferants. In der Modellierung wird dafür ein vorhandenes Template in das aktuell zu modellierende eingefügt. Der Benutzer greift dafür auf die Datenbank-Collection *Situationtemplates*. In der Modellierungsumgebung erscheint dabei eine Dropdown-Liste mit allen Dokumenten aus dieser Collection (siehe Abbildung 5.1).

Die Veränderung des Datenmodells wurde in Abschnitt 4.3 beschrieben. Die Einführung der Entität ThingType hat Auswirkungen auf das Modellierungswerkzeug: Beim Modellieren von einem Sensordaten-Lieferant sind Datenbankzugriffe nötig. Zum einen werden die ThingTypes aus der Datenbank geholt (siehe Abbildung 5.2 - 1). Nach der Auswahl des ThingTypes ist ein weiterer Zugriff nötig, um die Sensorarten (siehe Abbildung 5.2 - 2) und deren Messeinheiten (siehe Abbildung 5.2 - 3) des gewählten ThingType abzurufen.

Nachdem ein Template ausgewählt wird und die Modellierung abgeschlossen ist, wird bei der Speicherung oder beim Exportieren des fertigen Templates eine XML-Datei erstellt. Die XML-Datei kann nur gespeichert bzw. exportiert werden, wenn sie valide ist. Regeln der Validierung garantieren, dass nicht erlaubte Verbindungen und Namen der Elemente abgefangen werden. Bei der Erstellung der Templates in XML-Format wird die durch das Schema vorgeschriebene Form gesichert. Die Verbindungen des grafischen Tools werden in XML mit dem *parent*-Tag

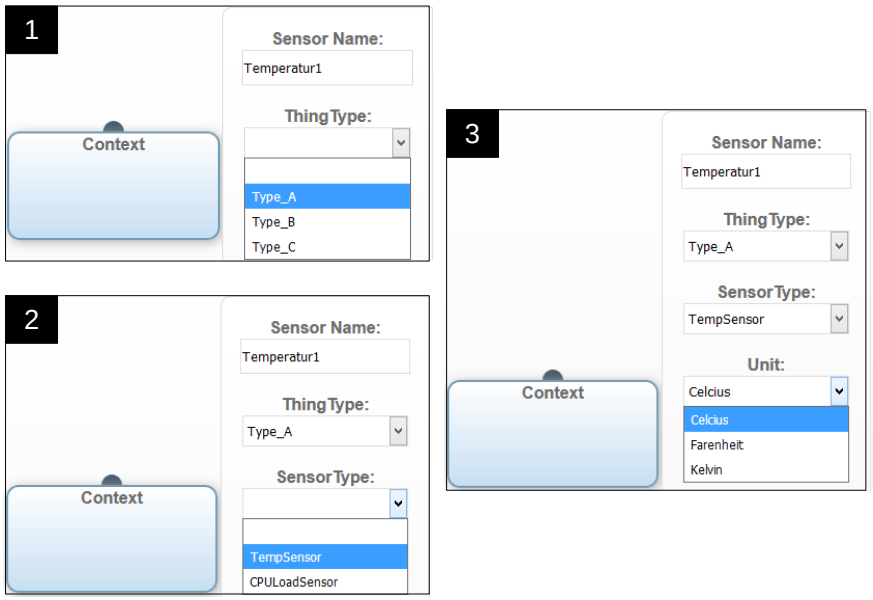

Abbildung 5.2.: DB-Zugriffe aus der Modellierungsumgebung.

realisiert. Diese Verbindungen stellen den unidirektionalen Informationsfluss von Kindknoten zum Elterknoten dar.

5.3. Verteilte Situationserkennung

Der Kernbeitrag des Konzepts der vorliegenden Thesis ist die Einführung der Verteilung in die SitOPT-Situationserkennung. Der Verteiler ist zuständig für die Auswahl von Things und Entscheidung ob die Situationserkennung lokal auf einem Edge-Nodes oder in der Cloud erfolgt. Die Verteilungsfunktion wurde im Rahmen dieser Arbeit prototypisch in Java implementiert.

5.3.1. Template-Auswahl

Der Benutzer startet die Anwendung und wird aufgefordert ein Situationstemplate zu wählen, welches in eine ausführbare Form gebracht werden muss. Dafür wird auf die Datenbank zugegriffen und die Templates der Situationtemplates-Collection werden als Ergebnis der HTTP-Anfrage zur Auswahl bereitgestellt. Für weitere Arbeit der Anwendung wird die XML-Form des Templates benötigt. Mit einem weiteren Lese-Zugriff wird der XML-Anhang abgerufen.

5.3.2. JAXB-API

Für die Anbindung der XML-Schemata in Java wird JAXB-Schnittstelle benutzt. Um die Situationstemplates in XML-Form als Java-Klassen darzustellen muss zunächst das Java Datenmodell aus dem XML-Schema erzeugt werden. So wird für jeden XML-Datentyp (z.B. Operation) eine

Java-Klasse erzeugt welche zur Laufzeit der Anwendung instanziiert wird. Für alle Attribute und Unterelemente eines XML-Datentyps werden Variablen, sowie Getter- und Setter-Methoden erstellt. Zur Laufzeit wird der XML-Anhang aus der Datenbank abgerufen und durch Unmarshalling zur Java-Repräsentation gebracht.

5.3.3. Verteilung

Nach der Instanziierung von JAXB-Klassen müssen die Things gewählt werden, welche die Sensordaten bereitstellen. Dazu erfolgt die Prüfung von ThingTypes einzelner Kontext-Lieferanten des Situationstemplates: Der Wert von dem XML-Tag "ThingType". Darauf folgt ein Zugriff auf die Datenbank, um die Things, die durch den ThingType erlaubt sind, zu ermitteln. Die erlaubten Things werden dem Benutzer zur Auswahl angeboten. Dieser Vorgang wird für jeden Kontext-Lieferant von Typ *Sensor* wiederholt. Bei der Thing-Auswahl wird neben dem Thing-Bezeichner, die IP des Edge-Node von der Datenbank abgerufen und zusammen mit dem gewählten Thing gespeichert.

Nach der Auswahl von Things kann die Ad-Hoc-Phase Abschnitt 4.4 beginnen. Anhand der IP-Adressen der Edge-Nodes lassen sich die Things im Rahmen der Ad-Hoc-Phase, in Gruppen aufteilen. Die Things mit gleicher IP-Adresse bilden eine Gruppierung. Jede derartige Gruppierung von Things kann einem Edge-Node mit der entsprechenden IP-Adresse zugeordnet werden. Im Rahmen dieser Arbeit besteht die kleinste erlaubte Thing-Gruppierung mit derselben Edge-Node-IP aus mindestens zwei Sensoren um unnötigen Overhead zu vermeiden. Wenn die Things mit derselben Edge-IP-Adresse zusätzlich über die gleiche Operation verbunden sind, so können sie direkt auf dem Edge-Node erkannt werden. Dafür werden alle Sensoren der Gruppierung, sowie die verbindende Operation in einer eigenständigen Situation zusammengefasst. Diese Situation wird durch das Situationserkennungssytem auf dem Edge-Node bearbeitet. Die Kommunikation mit dem restlichen Template erfolgt über Nachrichten einer Message Queue: auf dem Edge-Nodes entsteht ein Publisher und in der Cloud - ein Subscriber. Das Situationserkennungssytem auf Edge analysiert Sensordaten und erstellt Situationsobjekte. Der Publisher auf Edge sendet diese Situationsobjekte an den Subscriber in der Cloud. Für jede Gruppierung der Things innerhalb des ausgewählten Situationstemplate wird in der Cloud ein Subscriber erstellt. Dieser Subscriber wartet auf die Ergebnisse der jeweiligen Gruppierung (siehe Abbildung 4.10). Für jede Gruppierung der Things wird ein temporäres XML-Template erstellt. Dieses Template kann nach Wunsch des Benutzers in der Datenbank gespeichert und ferner separat verwendet werden. Nachdem alle möglichen Gruppierungen gebildet sind, kann die Phase der Modularen Verteilung beginnen.

In der Modularen Verteilung Abschnitt 4.4 geht es darum, die Eingabe-Elemente von Typ Situation zu verarbeiten. Falls das gewählte Situationstemplate Kontext-Lieferant von Typ *Situation* beinhaltet, wird zunächst das Eingabe-Template-XML aus der Datenbank abgerufen und mit JAXB instanziiert. Dann wird auf diesem verschachtelten Eingabe-Template die Verteilung-Funktion aufgerufen. Das bedeutet, dass die Auswahl von beteiligten Things und die Ad-Hoc-Verteilung auch auf das Eingabe-Template angewendet werden. Dadurch kann Rekursion entstehen. Das rekursive Schema wiederholt sich bis ein verschachteltes Eingabe-Template auftritt, dessen Eingabe-Elemente ausschließlich von Typ Sensor sind. Jedes Eingabe-Template modelliert eine Situation. Wenn diese Situation erkannt wird, braucht das äußere Template (Zieltemplate) eine Nachricht in Form von einem boolschen Wert (true oder false). Die Gründe dafür wurden im Kontext unter Kontextstripping diskutiert. Dieser Wert ersetzt das vollständige Situationsobkjekt, welches in

diesem Fall nicht nötig ist: Der Wert von Eingabe-Template wird im äußeren Template mit logischen Operatoren weiterverarbeitet. Darum spielt nur das Feld 'occurred' aus dem gewöhnlichen Situationsobjekt eine wichtige Rolle. Kommunikation zwischen einem Eingabe-Template und Zieltemplate erfolgt über Message Queue. Eingabe-Template veröffentlicht die 'occurred-'Info über einen Publisher; Das Zieltemplate bekommt diese Ergebnisse über einen Subscriber.

Die Modulare Verteilung erfolgt nach der Ad-Hoc-Verteilung. Das bedeutet, dass für jedes verschachtelte Eingabe-Template auch Ad-Hoc und Modulare Phase aufgerufen werden. Dadurch wird sichergestellt, dass auch in verschachtelten Templates die Verteilung auf Edge und Cloud funktioniert.

5.3.4. Mapping auf Node-Red

Mapping

Der Mapper ist die Komponente, welche das Template in eine ausführbare Form transformiert. Eine ausführbare Form stellt ein System dar, welches die Things anbinden kann und die Logik des modellierten Situationstemplates umsetzt. Im Rahmen der vorliegenden Arbeit wurde die ausführbare Form der Templates in Node-Red realisiert. Node-Red ist ein Werkzeug für Flow-Programmierung. In Node-Red werden die Elemente miteinander vernetzt. Das bedeutet, dass im Mapping-Schritt die JAXB-Klassen auf die Elemente von Node-Red abgebildet werden. Die Verbindungen zwischen den Node-Red Elementen entsprechen der Logik der Verbindungen in der grafischen Form der Templates und der XML-Hierarchie, die durch den *parent*-Tag entsteht. Der Mapper steht in einer engen Verbindung zum Verteiler. Diese Komponenten arbeiten in Wechselwirkung miteinander. Im Falle einer Rekursion (Modulare Verteilung) wird zunächst die Verteilung und Mapping des Zieltemplates vorgenommen. Das Eingabe-Template wird nach der Verteilung einem Knoten zugewiesen. Bei dem Mapping des Eingabe-Templates wird nicht nur die Logik des Situationstemplates auf Node-Red Elemente übertragen, sondern auch die Logik für die Kommunikation mit dem Zieltemplate. Die Node-Red-Instanz, die das Zieltemplate realisiert beinhaltet ein Subscriber-Element. Dieses Element abonniert die Ergebnisse der Situation, die das Eingabe-Template modelliertDas Eingabe-Template beinhaltet ein Publisher-Element über welches die Ergebnisse der Erkennung in die Cloud gesendet werden.

In der Abbildung 4.2 ist ein Situationstemplate zu sehen, das modelliert wurde um den gültigen Temperatur-Wertebereich für Things zu definieren und Überschreitungen der Bereichsgrenzen zu erkennen. In Abbildung 5.3 ist eine instanziierte (mit Things Roboter1 und Roboter2) ausführbare Node-Red Form von diesem Situationstemplate zu sehen. Abbildung 4.4 zeigt ein Template an, welches das Temperatur-Fehler Situationstemplate als Eingabe hat. In Abbildung 5.4 ist eine Node-Red Realisierung von diesem Situationstemplate zu sehen.

Die Node-Red Flows aus Abbildung 5.3 und Abbildung 5.4 sind entstanden nachdem auf dem Ziel-Template, *Maschine_Außer_Betrieb*, die modulare Verteilung eingesetzt wurde. Dabei wurde in dem Ziel-Template automatisch ein Subscriber zum Abonnieren der Ergebnisse des Eingabe-Templates eingebaut

Node-Red beschreibt die Flows in JSON-Format. Es ist möglich die Flows als JSON-Datei zu exportieren und zu importieren. Der Mapper übersetzt die Information aus dem Situationstem-

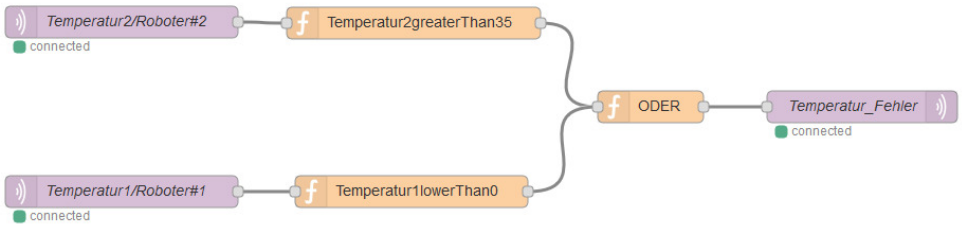

Abbildung 5.3.: Node-Red Realisierung von dem Temperatur-Fehler Situationstemplate. Diese Instanz der Situationserkennung überwacht Roboter1und Roboter2 und prüft deren Temperatursensorwerte. Temperatur-Fehler ist ein Eingabe-Template für das Ziel Template Maschine_Außer_Betrieb.

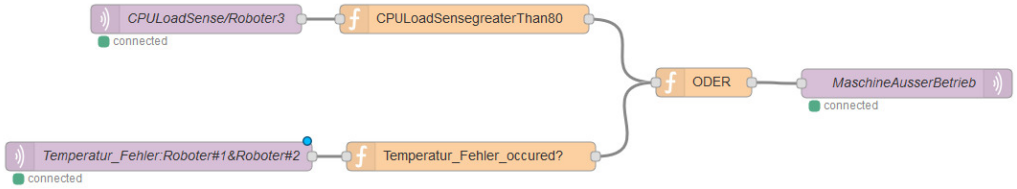

Abbildung 5.4.: Node-Red Realisierung von dem Maschine_Außer_Betrieb Situationstemplate. Diese Instanz der Situationserkennung überwacht den CPU-Sensor von Roboter3 und Ergebnisse des Eingabe-Templates Temperatur-Fehler. Maschine_Außer_Betrieb ist das Ziel-Template für Temperatur-Fehler

plate in JSON-Format. Das Template wird als JSON-Objekt in Node-Red importiert und die Situationserkennung kann gestartet werden.

In dem Mapper werden die Methoden deklariert, welche für die Verarbeitung der Kontextdaten entsprechend dem modellierten Template nötig sind. So werden die Bedingungen für Sensordaten und die Operationen als Java-Funktion-Rümpfe definiert, welche zur Laufzeit mit den Werten aus den Templates gefüllt werden. Die Funktionen mit eingesetzten Werten werden als String neben anderen Elementen in dem JSON-Objekt gespeichert.

Node-Red

Node-Red stellt eine Reihe von Bausteinen bereit. Die Elemente von Node-Red haben entweder Inputs oder Outputs oder beides. Durch Inputs und Outputs werden Nachrichten über verschiedene etablierte Protokollen gesendet. Neben HTTP,TCP und UDP hat Node-Red auch Elemente, die über das MQTT-Protokoll Nachrichten von einer Message Queue abrufen (Subscriber) und in Message Queue senden (Publisher) können. Um die Verbindung zu einer Message Queue aufzubauen wird ein Message Broker Element erstellt. Das Broker-Element enthält die Information über den realen Message Broker der das MQTT-Protokoll implementiert. Das Broker-Element hat keine grafische Repräsentation in dem Flow. Node-Red unterscheidet mqtt-in (Subscriber) und mqtt-out (Publisher) Elemente. Ein Node-Red mqtt-in Element, kann einen Topic deklarieren, auf welchem ein Thing über seine Sensoren die Kontextdaten (z.B ein Temperatursensor schickt Daten: 20,22,29,41

etc.) veröffentlicht. Die Realisierung der *Bedingungen* (z.B.' < 50'), sowie *Operationen* (z.B. logisches UND welches zwei Bedingungen verbindet) erfolgen mittels Node-Red *function*-Elemente. Das *function*-Element ist in der Lage die Information von anderen Elementen aufzunehmen und mit benutzerdefinierten JavaScript-Funktionen zu verarbeiten. Die Palette der erlaubten Funktionen (z.B. größer/kleiner als, in Intervall zwischen, UND, OR, XOR etc.) wird in dem Mapper vordefiniert und entsprechend den Anforderungen des Templates automatisch gewählt. Das Bedingungsfunktion-Element hat immer eine 1-zu-1 Input-Verbindung mit einem mqtt-in Element, welches ein Sensor darstellt und kann mehrere Outputs haben. Das Operationsfunktion-Element kann mehrere Inputs und Outputs haben. Die Situationsobjekte werden durch Operation-Elemente erstellt welch mit Situation-Elementen verbunden sind. Der Datentyp *Situation* aus dem Template wird mit einem mqtt-out Element dargestellt. Ein Operation-Element verbindet mehrere Bedingungen und/oder andere Operationen erstellt ein Situationsobjekt und leitet dieses an das mqtt-out Situation-Element. Das Situation-Element veröffentlicht das Situationsobjekt wiederum in einer Message Queue. Dort können alle interessierte Anwendungen darauf zugreifen. In jedem Fall wird das Situationsobjekt in der Datenbank gespeichert. Die hier beschriebene prototypische Implementierung benutzt einen nodeJS-Server, welcher die erstellten Situationsobjekte in die Datenbank einfügt.

5.4. Implementierungsdetails

Die Implementierung der Konzepte der vorliegenden Arbeit wurde in MS Windows 8.1 auf 64-Bit-Betriebssystem vorgenommen. Die Implementierung umfasst lokale Installation der Datenbank; Erstellung der im Datenmodell beschriebenen Entitäten; Anpassung des XML-Schema für die Situationstemplates; Anpassung der Modellierungsumgebung an das neue Schema; Modellierung von beispielhaften Templates; Entwurf der Verteilungslogik und Anpassung der existierenden Mappingfunktionen; Installation von Node-Red; Installation von Mosquitto Message Broker; Implementierung eines nodeJS-Servers für Interaktion mit der Message Queue und der Datenbank, und für Simulation von Sensorverhalten.

Im Folgenden sind technische Setup-Details der bei der Implementierung eingesetzten Technologien kurz beschrieben.

5.4.1. CouchDB

Für die vorliegende Arbeit wurde Apache CouchDB der Version 1.6.1 verwendet. Für die Installation wurde ein Windows Installer von der offiziellen Seite von CouchDB[1] heruntergeladen und ausgeführt. In dem Installations-Verzeichnis muss die couchdb.bat Datei ausgeführt werden um die Datenbank zu starten. CouchDB stellt ein grafische Oberfläche (GUI *Futon*) bereit, welche unter *localhost:5984/_utils* zu finden ist. Die Dokumente können entweder in GUI, oder über die REST-API mittels HTTP-Requests manipuliert werden.

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

5.4.2. Situation Template Modelling Tool

Für die Anpassung von der Modellierungsumgebung wurde JavaScript verwendet. Die HTTP-Zugriffe auf die REST-API der CouchDB-Datenbank wurden umgesetzt mit Hilfe von jQuery Ajax-Aufrufen. Die jQuery-Bibliothek[2] muss in der HTML-Datei des Modellierungswerkzeugs . Die Modellierungsumgebung wurde erstellt um als Webapp auf Apache Tomcat Server zu laufen. Download und Installation von Apache Tomcat erfolgt über die offizielle Webseite[3].

5.4.3. Verteilung und Mapping

Die Verteilung und Mapping Komponenten wurden mit Java SE 1.7 (JDK 1.8) entworfen. Der Code der Ausgangsarbeit ist in einem GitHub-Repository[4] vorzufinden. Für die Erstellung von Java-Repräsentation der Datentypen des XML-Schemas wurde JAXB verwendet.

5.4.4. Node-Red

Für die Installation von Node-Red wurde npm (Node Package Manager) verwendet. Installation benötigt Node.js (Versionen ab Node.js 0.12.x or 0.10.x). Nach der Installation kann die Flow-Engine mit dem Terminal-Befehl *node red* gestartet werden. Unter *localhost:1880* kann auf die grafische Oberfläche der Node-Red Instanz zugegriffen werden. Für Änderung von Port muss die Konfigurationsdatei (settings.js) in dem Installations-Verzeichnis angepasst werden (*uiPort: process.env.PORT || 1880*).

5.4.5. Mosquitto

Als Message Broker wurde Eclipse Mosquitto verwendet. Mosquitto ist ein Open Source Message Broker, welches das MQTT Protokoll umsetzt. Der Windows Installer befindet sich auf der offiziellen Seite von Mosquitto Broker[5]. Standardmäßig wird Mosquitto auf dem Port 1883 laufen. Für die Testzwecke, können Mosquitto Publisher und Subscriber mit folgenden Befehlen gestartet werden:

- **Publisher**: *start mosquitto_pub -t* [Name des Topics] *-m* [Inhalt der Nachricht];
- **Subscriber**: *start mosquitto_sub -v -t* [Name des Topics];

[2] https://jquery.com/
[3] http://tomcat.apache.org/
[4] https://github.com/SitOPT
[5] https://mosquitto.org/

6. Evaluation

In diesem Abschnitt wird rückblickend auf die Ziele der vorliegenden Arbeit eingegangen. Dabei wird kritisch betrachtet, ob die Ziele im Laufe dieser Arbeit erreicht wurden.

6.1. Zielerreichung

Das übergeordnete Ziel der Arbeit wurde in fünf Unteraufgaben aufgeteilt. Die zusammenfassende Information über die Erfüllung dieser Unteraufgaben wird im Folgenden präsentiert:

- Die erste Aufgabe, die darin bestand, das XML-Schema der Situationstemplates zu modifizieren, wurde umgesetzt. Dafür wurde das Attribut *InputType* in der XSD erweitert, um neben Sensor auch Situation als valide Eingabe zu erlauben. Das neue Schema erlaubt es, mehrere Situationstemplates miteinander zu verknüpfen, indem Situationen als Eingabe verwendet werden (zuvor konnten nur Sensorwerte als Eingabe für andere Situationen verwendet werden). Auf diese Weise können komplexere Situationen erstellt werden. Die Erweiterung des Schemas ist eine notwendige Voraussetzung für die Realisierung aller weiteren Ziele. Das erweiterte Thema wurde anschließend validiert.
- Im Rahmen der zweiten Aufgabe mussten die Eigenschaften des neuen Schemas auch in der graphischen Modellierungsumgebung umgesetzt werden. Dafür werden die Datentypen des XML-Schemas grafisch repräsentiert. Die Attribute/Unterelemente müssen von dem Benutzer ausgewählt oder eingegeben werden.
- Im Rahmen der dritten Teilaufgabe war erforderlich, die Relationen zwischen mehreren Situationstemplates modellierbar zu machen. Das bedeutet, dass in dem Modellierungswerkzeug eine Interaktion mit Datenbank erfolgt, damit die Palette der existierenden Situationen für den Benutzer verfügbar ist. Die Auswahl eines Templates, das die Eingabe-Situation modelliert wirkt sich auf die XML-Form des gesamten zu modellierenden Templates.
- Die vierte Aufgabe bestand in der Einführung der Verteilung in die Situationserkennung zu konzipieren. Dies war die zentrale Aufgabe der Arbeit. Für die Umsetzung dieser Aufgabe wurden auf Konzept-Ebene zwei Verteilungsarten entwickelt. Die Modulare Verteilung beschäftigt sich mit der Verteilung von Eingabe-Situationen. Durch Ad-Hoc-Verteilung werden die Edge-Nodes gewählt, welche die Situationserkennung selbstständig (nicht über Cloud) durchführen können. Als System zu Situationserkennung wurde Node-Red gewählt und als Kommunikationsprotokoll - MQTT.
- Die Things, Edge-Nodes sowie Cloud-Knoten kommunizieren über Message Queue, was zur losen Kopplung des Systems beiträgt. Das heißt die Komponenten ohne Wissen über Kommunikationspartner kommunizieren. Die Komplexität der Kommunikation liegt beim

Message Broker und ist für den Benutzer transparent. Dadurch können Komponenten einfach ausgetauscht werden, ohne Änderungen im Systemaufbau.

6.2. Ergebnisse

Die Ergebnisse der Arbeit wurden nach der Implementierung getestet. Die Tests haben sich auf die Erfüllung der erforderlichen Funktionalität beschränkt. Das implementierte System hat die erforderliche Funktionalität aufgewiesen. Die Laufzeit-Tests auf unterschiedlichen weit voneinander entfernten Maschinen wurden nicht durchgeführt, denn diese Tests würden keine aussagekräftigen Ergebnisse liefern. Das liegt daran, dass durch die Verteilung nur entschieden wird welche Edge-Server (welcher Rechner) die Situationserkennung durchführt. Die Laufzeit für die Kommunikation zweier voneinander entfernten Maschinen würde nur eine Aussage über die Netzwerklatenz machen. Diese Information ist für die vorliegende Arbeit irrelevant. Darum reichen die lokalen Funktionalitätstests mit Benutzung von mehreren Node-Red Instanzen aus. Die Node-Red Instanzen laufen dabei auf unterschiedlichen Ports, was die Arbeit unterschiedlicher Knoten simuliert.

Geeignete Situationstemplates für das Testen der beiden entworfenen Verteilungskonzepte wurden durchgeführt. So eignet sich beispielsweise für die modulare Verteilung ein verschachteltes Situationstemplate. Die Rekursion von drei ineinander verschachtelten Templates hat wie erforderlich drei Node-Red-Flows produziert. Für die Ad-hoc Verteilung wurde eine Situation mit mehreren Sensorknoten modelliert. Diese Knoten ließen sich unterschiedlichen Edge-Nodes zuweisen. Zur Laufzeit wurden für die Edge-Sub-Situationen temporäre XML-Templates entworfen und modelliert.

Zur eindeutigen Identifizierung der Situationen-Instanzen in der Message Queue wurde bei der Topic-Namensgebung die Konkatenation aus Namen des Situationstemplates und der beteiligten Things benutzt. Jedoch ist diese Methode bei einer hohen Anzahl von beteiligten Things nicht optimal auf Grund der Unübersichtlichkeit. Darum wurde als Alternative eine Registry (Verzeichnis) konzipiert, welche für alle Knoten im Netzwerk (Edge-Nodes und Cloud Knoten) eine Liste mit allen Situationen führt, welche durch das Situationserkennungssystem des jeweiligen Knoten erkennbar ist. Die Liste enthält den Namen des Situationstemplates und aller beteiligten Things. Durch die Überprüfung der Registry kann festgestellt werden ob eine Situation bereits existiert. Das bedeutet, dass alle nicht identische Situationen eindeutig erkannt werden.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Diese Arbeit beschreibt die Fortentwicklung der Situationserkennungsebene des Projekts SitOPT. Die Situationserkennung wird ausgebaut und kann in dem entworfenen Prototyp verteilt werden auf Edge-Knoten und Cloud-Knoten. Diese Verteilung im Sinne von Edge Computing ist potentiell vorteilhaft in Hinblick auf die Latenzzeit der Situationserkennung und Datenverkehr in die Cloud. Die Struktur der Situationstemplates wurde modifiziert: Bereits bestehende Situationstemplates können eingelesen werden. Dadurch kann Verknüpfung aus mehreren Situationstemplates entstehen. Damit kann die Modellierung komplexer Situationen potentiell vereinfacht werden und redundante Modellierung entfallen.

Die grundlegenden Begriffe, die Ausgangsarbeiten sowie weitere verwandten Arbeiten wurde im Kapitel 2 vorgestellt. Die Bedeutung von Kontext, Situation, Template und Things wird erklärt und deren Zusammenhang zu Internet Of Things vorgestellt. Zusätzlich wird auf den Begriff Workflows eingegangen um die Verbindung zu den Zielen von SitOPT-Projekts besser herzuleiten. Es folgt eine Vorstellung von Arbeiten die im Rahmen von SitOPT bereits veröffentlicht wurden. Im Anschluss findet ein Überblick über verwandte Themengebiete, Problemstellungen sowie vorgeschlagenen Lösungsansätze.

Während das zweite Kapitel sich mit dem Stand der Theorie befasst wird in Kapitel 3 der technische Hintergrund zum Vorschein gebracht. Dabei wird auf die Problematik der Speicherung und Verwaltung von IoT-Kontextdaten eingegangen. Dabei stellt sich heraus, dass die Eigenschaften der NoSQL-Datenbanken sich für die relevanten IoT-Szenarien besonders eignen. Dabei werden die Charakteristika von ACID und BASE im Hintergrund von CAP- und PACELC-Theorem diskutiert. Ferner wird auf Flow-Programmierparadigma und MQTT-Protokoll eingegangen. Die Kommunikation via Message Queues wird erläutert. Am Ende des Kapitels werden Eigenschaften von Cloud und Edge Computing angesprochen. Dabei wird auf deren Rollen beim Entwurf von IoT-Systemen Bezug genommen.

In Kapitel 4 werden die vorgestellten theoretischen Grundlagen benutzt, um auf der konzeptuellen Ebene die Umsetzung der Ziele zu erreichen. Dabei entsteht eine Systemarchitektur, welche die bereits umgesetzten Varianten modifiziert und ausbaut. Ein neues Schema der Templates wird eingeführt um Situationen in Relation zueinander zu modellieren. Als Kommunikationsmedium für die miteinander verknüpften Situationen dient eine Message Queue. Zur Laufzeit des Prototyps können die erlaubten physischen Objekte (Things) gewählt und in die Situationserkennung eingebunden werden. In Abhängigkeit von den Eigenschaften der gewählten Things (IP-Adresse des Edge-Knotens), erfolgt die Verteilung. Die verteilten Sub-Templates kommunizieren wiederum über eine Message Queue und können dadurch die erwünschte, modellierte Semantik realisieren. Durch Verzicht auf Details der erkannten Situationen kann eine Einsparung der Bandbreite

erfolgen. Durch die Situationserkennung auf Edge, verringert sich die Antwortzeit für die lokalen Anwendungen, die an der Erkennung interessiert sind.

Das Kapitel 5 beschreibt die Details der prototypischen Implementierung der Methoden, die in Kapitel 4 konzipiert wurden. Dabei wird auf die Details des Entwurfs einzelner Architektur-Komponenten eingegangen: Es werden die verwendeten Programmiersprachen, Frameworks/Bibliotheken/Schnittstellen, Software angesprochen. Die in Kapitel 3 vorgestellten Technologien, werden bei der Implementierung umgesetzt: Als Datenverwaltungssystem wird die NoSQL Datenbank CouchDB benutzt; die Verbindung der Komponenten der Erkennung erfolgt mittels Flowengine Node-Red; Als Message Broker wird Eclipse Mosquitto verwendet. Die Umsetzung der Verteilungskonzepte und die Transformation der Things und Templates in eine ausführbare Form werden im Detail beschrieben.

Anschließend werden im Kapitel 6 die Ergebnisse der Arbeit vorgestellt. Das geschieht in Form einer Analyse der Anforderungen und Beurteilung, ob diese erfüllt wurden. Die lokalen Tests der Funktionalität werden durchgeführt. Die Überprüfung der Effizienz gilt nicht als Ziel der vorliegenden Arbeit und kann in zukünftigen Arbeiten an SitOPTs Situationserkennung getestet werden.

7.2. Ausblick

Im Weiteren werden potentielle Erweiterungen von SitOPT vorgeschlagen und diskutiert. Dabei wird explizit auf die Situationserkennungsebene der SitOPT-Architektur eingegangen.

7.2.1. Verteilungskriterien

In dem Konzept von dieser Arbeit wurden solche Kriterien für Verteilung wie Sicherheit, Bandbreiteneinsparung, Latenz und Overhead definiert. In zukünftigen Arbeiten können, diese Kriterien spezifiziert und konkretisiert werden. Es können weitere Verteilungskriterien, wie z.B. Qualität zum Vorschein kommen.

Die Zielsetzung dieser Arbeit hat die Frage, über die Qualität der Sensoren, der produzierten Sensorwerte und der erkannten Situationen nicht beinhaltet. Durch Einführung dieser Eigenschaft kann die Situationserkennung aussagekräftiger werden. Durch die Einführung der Verteilung kommt die Frage über die Qualität der Edge-Nodes auf. Dieser Wert kann beispielsweise aus dem Prozentsatz, der korrekt erkannten Situationen ableitbar sein. In Abhängigkeit von diesem Wert kann z.B. der Verteiler eine Entscheidung treffen, ob der Edge-Node ausreichend qualifiziert ist oder nicht.

Das Overhead-Kriterium wurde im Rahmen der Prototyp-Entwicklung nicht ausreichend getestet. Um eine Aussage zu machen, welcher Grad an Verteilung optimal ist, müssen zusätzliche Tests durchgeführt werden. Der Aufwand, den unterschiedliche Message Broker betreiben können bedarf ebenfalls einer Untersuchung.

Im Rahmen der Bandbreiteneinsparung kann die Kontextstripping Methode erweitert werden. Durch die Interaktion mit dem Benutzer (z.B. Knopf-Druck) soll Kontextstripping ein- und ausschaltbar sein. Diverse Abstufungen von Kontextstripping können eingeführt werden: Je nach Abstufung wird mehr oder weniger Information geliefert.

Auch das Logging der Situationen kann interaktiver werden. Der Benutzer muss Logging ein und ausschalten können ohne den Quellcode zu modifizieren. Methoden zur Verwaltung des Logs müssen eingeführt werden.

7.2.2. Situationserkennungssysteme

Für die vorliegende Arbeit wurde ein System zur Erkennung von Situationen verwendet. Zwar hat sich Node-Red für die Zielsetzung dieser Arbeit geeignet, jedoch kann durchaus ein Szenario aufkommen, in welchem andere Systeme vorteilhafter sind. Das kann der Fall sein, wenn der Datentyp der Sensoren sich nicht ohne umständliche Anpassung der Mapper-Komponente für Node-Red eignet. Außerdem kann die Nutzung von Node-Red durch suboptimale Skalierbarkeit und Parallelisierung diverse Nachteile aufweisen. Als Alternative zu Node-Red kann Event-Processing-Engine CEP Esper [1] verwendet werden. Als eine Ereignis-getriebenes Tool weist Esper folgende Vorteile auf: lose Kopplung, Kommunikation durch Nachrichten, Unabhängigkeit von Geolokalisierung und Distanz zwischen Komponenten. Wie Node-Red, ist Esper ein Open Source Tool. Esper kann hohe Ereignis- und Nachrichtendichte in Echtzeit verarbeiten. Anpassung der, in dieser Arbeit entworfenen, Konzepte an Esper (oder andere Situationserkennungstools) kann sich für zukünftige SitOPT-Erweiterungen als vorteilhaft erweisen.

[1] http://esper.codehaus.org

A. Anhang

A.1. XML-Schema für Situationstemplates

Im Verlauf dieser Arbeit wurde das XML-Schema für die Situationstemplates verändert. Durch diese Veränderung wird Modellierung von Eingabe-Situationen ermöglicht. Hier werden die wichtigen Teile des Schemas für die Situationstemplates gezeigt.

Im folgenden Ausschnitt wird das Rootelement und dar Datentyp tSituationTemplate gezeigt. Das tSituationTemplate besteht aus tSituation und einer Liste von ThingTypes, (die in dem Template erlaubt sind).

```
<!-- root -->
<xs:element name="SituationTemplate" type="tSituationTemplate"/>

<xs:complexType name="tSituationTemplate">
      <xs:sequence>
            <xs:element name="Situation" type="tSituation" minOccurs="1" maxOccurs="1"/>
            <xs:element name="thingTypes">
                  <xs:complexType>
                        <xs:sequence>
                              <xs:element name="thingType" type="xs:string"
                                    minOccurs="1" maxOccurs="unbounded"/>
                        </xs:sequence>
                  </xs:complexType>
            </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
      <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

In dem nächsten Abschnitt wird der Datentyp tSituation gezeigt. Die Bestandteile von tSituation sind: tSituationNode, tOperationNode, tContextNode.

```
<xs:complexType name="tSituation">
      <xs:sequence>
            <xs:element name="situationNode" type="tSituationNode" minOccurs="1"
                maxOccurs="1"/>
            <xs:element name="operationNode" type="tOperationNode"
                maxOccurs="unbounded"/>
            <xs:element name="contextNode" type="tContextNode" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

Die Datentypen tSituationNode und tOperationNode werden im folgenden Abschnitt gezeigt. Diese Datentypen entsprechen den Elementen eines Situationstemplates: Situation, Operation. Ein tOperationNode ist eine logische (boolsche) Operation, die boolsche Werte (manchmal in Kombination mit nicht boolschen Werten) als Eingabe bekommt und eine boolsche Ausgabe produziert. Ein tSituationNode hat keine Unterelemente und dient stets als Destination für den Nachrichtenfluss während der Erkennung.

```
<xs:complexType name="tSituationNode">
      <xs:attribute name="id" type="xs:ID" use="required"/>
      <xs:attribute name="name" type="xs:string"/>
</xs:complexType>

<xs:complexType name="tOperationNode">
      <xs:sequence>
            <xs:element name="parent" type="tParent" minOccurs="1"
                maxOccurs="unbounded"/>
            <xs:element name="type">
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:enumeration value="and"/>
                              <xs:enumeration value="xor"/>
                              <xs:enumeration value="or"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
      <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

Jedes Element außer eine Situation hat einen Parent. Diese Hierarchie wird mit Hilfe von der Klasse tParent gewährleistet:

```
<xs:complexType name="tParent">
      <xs:attribute name="parentID" type="xs:IDREF"/>
</xs:complexType>
</xs:schema>
```

Für die Bereitstellung der Kontextinformation ist ein Kontextknoten verantwortlich. Dieser wird im folgenden Abschnitt durch den Datentyp tContextNode dargestellt. Dieser Datentyp kann unterschiedlichen *inputType* haben, wie z.B. Sensor oder Situation. Ein zusätzlicher *inputType*, Static (statische Kontextdaten), wird nicht im Rahmen der vorliegenden Arbeit bearbeitet. Je nach inputType variieren die Unterelement und Attribute von einem tContextNode. Ein Kontext enthölt auch eine Bedingung/Kondition (*operation*), die den Wertebereich (*value*) einschränkt. Wenn die Kondition erfüllt wird - werden die Daten weitergeleitet. Es wird außerdem der Typ von Thing und von Sensor durch die Elemente *thingType* und *sensorType* kontrolliert.

```
<xs:complexType name="tContextNode">
      <xs:sequence>
            <xs:element name="inputType" minOccurs="1" maxOccurs="1">
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:enumeration value="sensor" />
                              <xs:enumeration value="situation" />
                              <xs:enumeration value="static" />
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>

            <!--situationInput, negated - if inputType=situation -->
            <xs:element name="situationInput" type="xs:string" minOccurs="0"
                maxOccurs="1" />
            <!--Adjust type of thing to be complextype?-->
            <!--thing - if inputtyp=situation/static-->
            <xs:element name="thing" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="negated" type="xs:boolean" minOccurs="0" maxOccurs="1"/>


            <!--thingType, sensorType, measureName, opType, value - if
                inputType=sensor-->
            <xs:element name="thingType" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="sensorType" type="xs:string" minOccurs="0" maxOccurs="1"
                />
            <xs:element name="measureName" type="xs:string" minOccurs="0" maxOccurs="1"
                />
            <xs:element name="opType" minOccurs="0" maxOccurs="1">
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:enumeration value="lowerThan"/>
                              <xs:enumeration value="greaterThan"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
            <xs:element name="value" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="parent" type="tParent" minOccurs="1"
                maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
      <xs:attribute name="name" type="xs:string"/>

      <!--
      Instead of attribute for sensorType there is an element.
            <xs:attribute name="type" type="xs:string" use="required"/>-->
</xs:complexType>
```

Literaturverzeichnis

[ADB+99] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, P. Steggles. „Towards a better understanding of context and context-awareness“. In: *Handheld and ubiquitous computing*. Springer. 1999, S. 304–307 (zitiert auf S. 14).

[ADOB98] D. Abowd, A. K. Dey, R. Orr, J. Brotherton. „Context-awareness in wearable and ubiquitous computing“. In: *Virtual Reality* 3.3 (1998), S. 200–211 (zitiert auf S. 14).

[AFG+10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al. „A view of cloud computing“. In: *Communications of the ACM* 53.4 (2010), S. 50–58 (zitiert auf S. 29).

[BG11] R. Baheti, H. Gill. „Cyber-physical systems“. In: *The impact of control technology* 12 (2011), S. 161–166 (zitiert auf S. 13).

[BK09] A. Buchmann, B. Koldehofe. „Complex event processing“. In: *IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik* 51.5 (2009), S. 241–242 (zitiert auf S. 22).

[COV] C. COVER. „The Promise of Edge Computing“. In: () (zitiert auf S. 30).

[DGG93] C. Dousson, P. Gaborit, M. Ghallab. „Situation recognition: representation and algorithms“. In: *IJCAI*. Bd. 93. 1993, S. 166–172 (zitiert auf S. 22).

[DW08] N. Dziengel, G. Wittenburg. „Verteilte Ereigniserkennung in Sensornetzen.“ In: *Informatiktage*. 2008, S. 213–216 (zitiert auf S. 23).

[EFGK03] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec. „The many faces of publish/subscribe“. In: *ACM computing surveys (CSUR)* 35.2 (2003), S. 114–131 (zitiert auf S. 19, 28).

[GS12] E. H. Glaessgen, D. Stargel. „The Digital Twin paradigm for future NASA and US Air Force vehicles“. In: *53rd Struct. Dyn. Mater. Conf. Special Session: Digital Twin, Honolulu, HI, US*. 2012, S. 1–14 (zitiert auf S. 13).

[GSS15] M. Göggelmann, S. Sarangi, E. Schäfer. „Ein grafisches Werkzeug zur Modellierung von Situations-Templates“. In: (2015) (zitiert auf S. 19, 49).

[HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. In: *Complex Systems Informatics and Modeling Quarterly* 9 (2016), S. 28–43 (zitiert auf S. 22).

[HH95] D. Hollingsworth, U. Hampshire. „Workflow management coalition: The workflow reference model“. In: *Document Number TC00-1003* 19 (1995) (zitiert auf S. 16).

[HTS08] U. Hunkeler, H. L. Truong, A. Stanford-Clark. „MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks“. In: *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE. 2008, S. 791–798 (zitiert auf S. 29).

[HWBM16a] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Automated Sensor Registration, Binding and Sensor Data Provisioning.“ In: *CAiSE Forum*. 2016, S. 81–88 (zitiert auf S. 22).

[HWBM16b] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Dynamic ontology-based sensor binding“. In: *East European Conference on Advances in Databases and Information Systems*. Springer. 2016, S. 323–337 (zitiert auf S. 22).

[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“. In: *Proceedings of the 9th symposium and summer school on service-oriented computing*. 2015, S. 113–127 (zitiert auf S. 22).

[LR00] F. Leymann, D. Roller. „Production workflow: concepts and techniques“. In: (2000) (zitiert auf S. 16).

[Luc02] D. Luckham. *The power of events*. Bd. 204. Addison-Wesley Reading, 2002 (zitiert auf S. 22).

[Mar81] J. Martin. „Managing the data base environment“. In: (1981) (zitiert auf S. 26).

[Mas11] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Ö'Reilly Media, Inc.", 2011 (zitiert auf S. 26).

[MH13] A. Moniruzzaman, S. A. Hossain. „Nosql database: New era of databases for big data analytics-classification, characteristics and comparison“. In: *arXiv preprint arXiv:1307.0191* (2013) (zitiert auf S. 25).

[MHWM17] M. Mormul, P. Hirmer, M. Wieland, B. Mitschang. „Situation model as interface between situation recognition and situation-aware applications“. In: *Computer Science-Research and Development* 32.3-4 (2017), S. 331–342 (zitiert auf S. 22).

[Mor15] M. Mormul. „Entwicklung eines Situationsmodells als Schnittstelle zwischen Situationserkennung und Workflows“. Magisterarb. 2015 (zitiert auf S. 18–21, 31, 33, 36, 38).

[MTD08] A. Manzoor, H.-L. Truong, S. Dustdar. „On the evaluation of quality of context“. In: *Smart Sensing and Context* (2008), S. 140–153 (zitiert auf S. 14).

[NLIH13] C. Nance, T. Losser, R. Iype, G. Harmon. „Nosql vs rdbms-why there is room for both“. In: (2013) (zitiert auf S. 25).

[OAS] OASIS. *OASIS*. URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (zitiert auf S. 29).

[RPM99] N. Ryan, J. Pascoe, D. Morse. „Enhanced reality fieldwork: the context aware archaeological assistant“. In: *Bar International Series* 750 (1999), S. 269–274 (zitiert auf S. 14).

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge computing: Vision and challenges“. In: *IEEE Internet of Things Journal* 3.5 (2016), S. 637–646 (zitiert auf S. 30).

[SHWM16] A. C. F. da Silva, P. Hirmer, M. Wieland, B. Mitschang. „SitRS XT-Towards Near Real Time Situation Recognition“. In: *Journal of Information and Data Management* 7.1 (2016), S. 4 (zitiert auf S. 22).

[Sit] SitOPT. *Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten.* URL: https://www.ipvs.uni-stuttgart.de/abteilungen/as/forschung/projekte/SitOPT (zitiert auf S. 10, 11, 16, 17).

[ST94] B. N. Schilit, M. M. Theimer. „Disseminating active map information to mobile hosts“. In: *IEEE network* 8.5 (1994), S. 22–32 (zitiert auf S. 14).

[WG96] K. P. Wershofen, V. Graefe. „Situationserkennung als Grundlage der Verhaltenssteuerung eines mobilen Roboters“. In: *Autonome Mobile Systeme 1996*. Springer, 1996, S. 170–179 (zitiert auf S. 22).

[WHR09] H. Wolf, K. Herrmann, K. Rothermel. „Modeling dynamic context awareness for situated workflows“. In: *OTM Confederated International ConferencesÖn the Move to Meaningful Internet Systems"*. Springer. 2009, S. 98–107 (zitiert auf S. 16, 23).

[Wie13] M. Wieland. „Methoden zur Modellierung und Ausführung kontextbezogener Workflows in Produktionsumgebungen“. In: (2013) (zitiert auf S. 16).

[WKL+09] M. Wieland, U.-P. Käppeler, P. Levi, F. Leymann, D. Nicklas. „Towards Integration of Uncertain Sensor Data into Context-aware Workflows.“ In: *GI Jahrestagung*. 2009, S. 2029–2040 (zitiert auf S. 14).

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, F. Leymann. „Towards context-aware workflows“. In: *CAiSE07 Proc. of the Workshops and Doctoral Consortium*. Bd. 2. 2007, S. 25 (zitiert auf S. 16).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. „Towards situation-aware adaptive workflows: SitOPT—A general purpose situation-aware workflow management system“. In: *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*. IEEE. 2015, S. 32–37 (zitiert auf S. 16, 22).

Alle URLs wurden zuletzt am 10. 12. 2017 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