Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3718

Entwicklung eines Situationsmodells als Schnittstelle zwischen Situationserkennung und Workflows

Mathias Mormul

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. ing. habil. Bernhard Mitschang
Betreuer/in:	Dr. rer. nat. Matthias Wieland
Beginn am:	04. März 2015
Beendet am:	03. September 2015
CR-Nummer:	D.2.2, D.2.11, D.4.4, H.2.1, H.2.4

Kurzfassung

Neue Paradigmen wie das *Internet of Things* und *Industrie 4.0* erwecken das Interesse vieler Unternehmen und ermöglichen das Erstellen von *Smart Factories*, *Smart Homes* und vielen weiteren autonom agierenden Umgebungen. Grundlage hierfür sind Sensoren, die Umweltdaten der sich dynamisch ändernden Umgebungen ermitteln. Die große Anzahl von Sensorwerten und deren Verarbeitung stellen die IT-Welt vor neue Herausforderungen. Zusätzlich werden auf Anwendungsseite Mechanismen benötigt, die autonom und situationsbezogen agieren, um sich an die Umgebung anzupassen. Es existiert bereits eine Reihe von Anwendungen, die auf Situationen reagieren, wobei die Sensoren allerdings fest in die Anwendung integriert sind und sich deshalb nur für spezielle Anwendungsfälle eignen. In dieser Arbeit wird das Forschungsprojekt SitOPT vorgestellt und entwickelt, dessen Ziel es ist, ein General-Purpose-System für eine effiziente Situationserkennung als Voraussetzung für die Verwendung von adaptiven situationsbezogenen Workflows zu ermöglichen. Die vorgestellten Methoden und Konzepte sollen ein flexibles und leicht erweiterbares System gewährleisten und durch die Entkopplung der Workflowdomäne von der Situationserkennung auf beiden Seiten zur einfacheren und voneinander unabhängigen Entwicklung beitragen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Internet of Things ist ein neues Paradigma aus der Informationsbranche. Es beschreibt die fortschreitende Vernetzung von Dingen (*Things*) über das Internet miteinander. Hierbei werden physische Objekte mit ihren jeweils virtuellen Repräsentationen verknüpft. Um eine virtuelle Repräsentation zu ermöglichen, wird eine große Anzahl von Informationen über das Objekt benötigt. Einige Information wie Bezeichnung und Größe sind bereits bei der Produktion bekannt und verändern sich über die Lebenszeit des Objekts nicht. Andere Informationen, wie z.B. die Position eines Autos hingegen ändern sich häufig und bedürfen einer ständigen Überwachung des Objekts. Solch eine Überwachung kann mit verschiedenartigen Sensoren durchgeführt werden. Die Verbreitung von Sensoren in allen Bereichen des Lebens ist am Beispiel der RFID (radio-frequency identification)-Technik zu sehen, die verwendet wird, um das Identifizieren und Lokalisieren mittels Radiowellen durchzuführen. Bereits im Jahr 2006 prognostizierte [Hen06] das Marktwachstum für RFID-Systeme zwischen 2004 und 2010 global von EUR 1,5 Mrd. auf 22 Mrd. Die Informationslücke, die sich zwischen physischen Objekten und deren virtueller Repräsentation befindet, soll auf diese Weise geschlossen werden und daraus ein cyber-physisches Objekt entstehen. Der Zusammenschluss von cyber-physischen Objekte führt zu einem cyber-physischen System (CPS). Mittels einer Machine-to-Machine Kommunikation können Informationen untereinander ausgetauscht werden und auf Basis dieser Informationen Entscheidungen getroffen oder Operationen ausgeführt werden. Beispiele für CPS sind unter anderem ein intelligentes Stromnetz und Industrie 4.0.

Industrie 4.0 beschreibt das Zukunftsprojekt der deutschen Bundesregierung und der Industrie. Das Ziel dieses Projekts ist die Erstellung einer *intelligenten Fabrik*, die sich autonom an ihre dynamischen Umgebungen anpassen und Änderungen im Produktionsablauf veranlassen kann. Die fortschreitende Informatisierung im Fertigungsbereich ermöglicht diesen Schritt, indem es für ein größeres Vorkommen an Sensoren (z.B. RFID-Chips) sorgt. Zusätzlich werden Aktoren benötigt, die auf Basis der Sensorwerte Entscheidungen treffen können. Eine mögliche Wahl für Aktoren ist der Einsatz von Workflows. Workflows sind heutzutage ein Grundbaustein in der Orchestrierung von Arbeitsabläufen im Geschäftsprozessbereich und werden zum Schlüsselelement für den Erfolg einer Firma gezählt [MGP+11]. Die Koordinierung von Geschäftsprozessen führte zu deren Modularisierung, was wiederum dazu führte, dass einzelne Prozesse leichter ausgetauscht werden konnten, um Optimierungen des Gesamtsystems vorzunehmen. Die logische Konsequenz ist der Einsatz von Workflows in anderen Bereichen wie der Fertigungsbranche, um auch dort von deren Vorteile zu profitieren. Dadurch entstehen allerdings neue Herausforderungen, die es zu bewältigen gilt.

1.1. Problembeschreibung

Sich ständig ändernde Anforderungen und Umgebungszustände erschweren den geplanten Ablauf eines Workflows. Zusätzlich gestaltet sich die Modellierung eines Workflows, der alle möglichen Umgebungszustände beachtet und mögliche Alternativabläufe beinhaltet, als eine Aufgabe, deren Komplexität mit der immer weiter steigenden Anzahl von verfügbaren Sensorwerten weiterhin steigt [WSBL15]. In Abbildung 1.1 ist der Ablauf zu sehen, um ein auf Informationen basiertes, autonom organisiertes System zu erstellen.

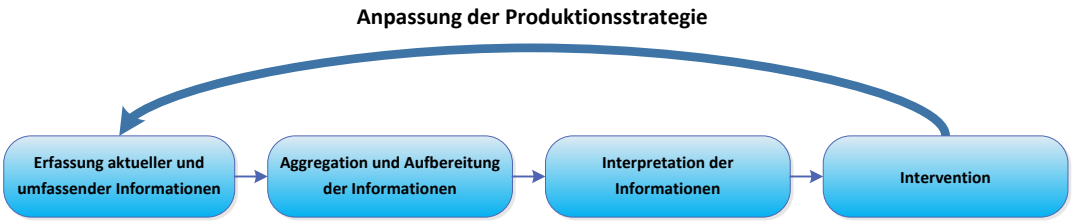

Abbildung 1.1.: Industrie 4.0 - Autonom organisiertes System [GSL14, p. 526]

Es wäre vorteilhaft, wenn Workflows nur den letzten Schritt *Intervention* durchführen müssten. Die Auslagerung der ersten drei Aufgaben führt zu einer Vereinfachung der Modellierung von Workflows. Das in dieser Arbeit vorgestellte Projekt SitOPT ermöglicht diese Auslagerung, indem es die Punkte *Erfassung aktueller und umfassender Informationen, Aggregation und Aufbereitung der Informationen* sowie *Interpretation der Informationen* in einem System vereint. Dem Interpretationsschritt folgt die Erstellung einer Situation, welche alle benötigten Informationen enthält. Diese Situationen können von Workflows verwendet werden, um entsprechende Änderungen im Ablauf durchzuführen. Die Ziele von SitOPT sind [WSBL15]:

1. Verringerung der Komplexität bei der Erstellung von Workflow-Modellen
2. Laufzeitadaption von Workflows an von SitOPT erkannten Situationen
3. Beliebiges Austauschen, Ändern und Optimieren von Situationserkennungssystemen
4. Verwendung von Situationen durch mehrere Workflows

Im Rahmen dieser Arbeit und des Projekts SitOPT wird eine Schnittstelle zwischen der Situationserkennung und Workflows erzeugt. Es muss ermittelt werden, welche Information ein Workflow benötigt, um ein Situationsmodell zu erstellen. Des Weiteren wird erforscht, wie Situationen mit Hilfe von Situationstemplates modelliert werden können.

1.2. Einführendes Beispiel

Als einführendes Beispiel wird die Überwachung von Computern gewählt. Ist ein Computer überlastet oder nicht verfügbar, soll der Benutzer darüber informiert werden, um Schritte zur Behebung des Problems einleiten zu können. Je mehr Informationen der Benutzer über den jeweiligen Computer und dessen Zustand erhält, desto durchdachter können dessen Entscheidungen sein. Ist beispielsweise bekannt, dass jeden Tag zu einem bestimmten Zeitpunkt die Auslastung eines Computers einen Hochpunkt erreicht, können die jeweiligen Prozesse identifiziert werden, die zu dieser hohen Auslastung führen. Ist der Standort aller Computer bekannt, kann das Personal eines ausgefallenen zum am nächsten stehenden Computer geführt werden, vorausgesetzt der Computer entspricht den Anforderungen, die für die aktuelle Aufgabe des Personals benötigt werden. Um diesen Vorgang mithilfe eines Workflows zu automatisieren, sind folgende Bedingungen zu erfüllen.

Wie an dem vorgestellten Beispiel zu erkennen ist, wird eine Großzahl verschiedenartiger Sensoren benötigt. Die Auslastung eines Computers wird über die prozentuale Auslastung der CPU sowie der Größe des verfügbaren Arbeitsspeichers definiert. Zusätzlich wird überwacht, ob der Computer aktiv ist, indem periodisch ein Ping gesendet wird. Der Standort eines Computers und des Personals wird mit einem GPS-Sensor ermittelt werden. Die derzeitige Aufgabe des Personals sowie deren Anforderung an einen Computer muss ebenfalls bekannt sein . Gleichzeitig muss Hintergrundwissen vorhanden sein, z.B. ob die Anforderungen des Computers der Aufgabe des Personals genügen. Die Anforderungen einer Aufgabe müssen spezifiziert werden und dem Benutzer zugänglich sein, um sie mit der Systemspezifikation des Computers zu vergleichen. Stimmen die Spezifikationen nicht überein, müssen Alternativpläne vorhanden sein, die dem Personal die Bearbeitung ihrer Aufgabe ermöglichen. Um dem Personal Anweisungen geben zu können, müssen die Kommunikationsmöglichkeiten sowie Kommunikationsprotokolle bekannt sein. All diese Informationen müssen dem Workflow mitgeteilt werden, wofür ein Netzwerk, z.B. das Internet, benötigt wird.

Dieses kleine Beispiel zeigt, wie komplex die Überwachung von Computern mit nur einem Anwendungsfall ist. Je größer die Anzahl überwachter Maschinen, desto größer die Anzahl möglicher Situationen und benötigter Alternativpläne. Heute bereits vorhandene Anwendungsfälle sind beispielsweise ortsbezogene Werbungen auf mobilen Endgeräten, welche auf den Standort des Benutzers zugreifen. Die meisten Anwendungsfälle beschränken sich auf eine überschaubare Anzahl von Situationen. Die Automatisierung einer gesamten Fabrik mit Hunderten Maschinen und Tausenden von Sensorwerten stellt dagegen ein hoch komplexes System mit weitaus höheren Anforderungen dar. Um diesen Herausforderungen entgegenzutreten, müssen bestimmte Anforderungen von SitOPT erfüllt werden.

1.3. Anforderungen an diese Arbeit & SitOPT

Das Ziel ist es, ein System zu entwickeln, dass für verschiedene Anwendungsfälle verwendet werden kann und situationsbezogenen Workflows Situationsdaten effizient und anforderungsgerecht bereitstellt. Hierfür soll ein Situationsmodell erstellt werden, dass als Schnittstelle zwischen einer Situationserkennung und situationsbezogenen Workflows dient. Anhand dieses Situationsmodells soll ermittelt werden, wie sich Situationen mithilfe sogenannter Situationstemplates modellieren

Anforderungen	Beschreibung
A1: Schnittstelle	Die Hauptaufgabe dieser Arbeit besteht in der Bereitstellung einer Schnittstelle, um die Interaktion zwischen einem Situationserkennungssystem und situationsbezogenen Workflows zu ermöglichen. Auf diese Weise werden die verschiedenen Domänen entkoppelt, was die Modellierung von Workflows vereinfacht und die Verwendung und Optimierung verschiedener Situationserkennungssysteme ermöglicht. Zusätzlich muss die Schnittstelle alle benötigten Funktionen bereitstellen, um die Interaktion aller beteiligten Module zu ermöglichen.
A2: Flexibilität	Der prototypische Charakter der Implementierung erfordert, dass darauf aufbauende Arbeiten möglichst einfach zu gestalten sind. Das Austauschen oder Hinzufügen von Modulen soll einfach sein, weshalb in allen Bereichen der Implementierung auf eine modulare Bauweise geachtet werden soll.
A3: Skalierbarkeit	Abhängig von der Anzahl der überwachten Objekte sowie der Anzahl der zugrundeliegenden Sensorwerte können bei der Überwachung große Datenmengen entstehen. Gleichzeitig stellt die Überwachung selbst einen ressourcenintensiven Prozess dar. Die zugrundeliegenden Systeme zur Datenverwaltung und Situationserkennung müssen einfach und effizient skalierbar sein. Sowohl die verwendeten Technologien als auch die implementierungsspezifischen Methoden sollen einfach Skalierbarkeit ermöglichen.
A4: Benutzerfreundlichkeit	Die Verwendung von SitOPT soll möglichst einfach gestaltet werden, dass eine einfache und intuitive Schnittstellenbenutzung ermöglicht wird. Darüber hinaus soll eine detaillierte Dokumentation der Schnittstellenfunktionen erfolgen, um die Benutzung zusätzlich zu vereinfachen.
A5: Verschiedenartige Sensordaten	Wie im einführenden Beispiel zu sehen ist, müssen verschiedene Sensordaten mit SitOPT integriert werden. Diese unterscheiden sich in verschiedene Datentypen, Qualitätswerten, usw. Um ein anwendungsunabhängiges System zu erstellen, muss SitOPT mit allen möglichen Sensordaten interagieren können.

Tabelle 1.1.: Anforderungen an SitOPT

lassen. Die Verwaltung der erkannten Situationen sowie der Situationstemplates soll mit Hilfe eines Repository stattfinden, um deren Wiederverwendung zu ermöglichen. Die funktionalen sowie nichtfunktionalen Anforderungen an SitOPT werden in Tabelle 1.1 dargestellt.

1.4. Gliederung dieser Arbeit

Im folgenden Kapitel 2 werden die Grundlagen und Begriffsdefinitionen für diese Arbeit vorgestellt. Des Weiteren werden verwandte Arbeiten vorgestellt, auf denen manche Bereiche von SitOPT aufbauen. Kapitel 3 beschreibt die in SitOPT verwendeten Technologien. In Kapitel 4 wird das Konzept und die Architektur von SitOPT vorgestellt. Zusätzlich werden alternative Methoden aufgezeigt, die im Laufe der prototypischen Implementierung zur Wahl standen. In Kapitel 5 werden Details der Implementierung gezeigt, um die in Kapitel 5 vorgestellten Konzepte umzusetzen. Kapitel 6 zeigt die Evaluation des implementierten Prototyps. Die Anforderungen aus Tabelle 1.1 werden hier im Hinblick auf ihre Erfüllung evaluiert. Kapitel 7 enthält offene Punkte, die sich im weiteren Projektverlauf für die Implementierung anbieten. Kapitel 7 weist darüber hinaus eine Zusammenfassung dieser Arbeit auf.

2. Grundlagen und verwandte Arbeiten

2.1. Situationsbezogene Workflows

Die Workflow Management Coalition [wmc] definiert den Begriff Workflow als *computergesteuerte Unterstützung oder Automatisierung von Geschäftsprozessen, ganz oder teilweise.* Geschäftsprozesse und deren Ausführung werden modelliert und bieten Unternehmen eine bessere Übersicht über den Prozessablauf. Diese Transparenz erleichtert es, Optimierungen im Prozessablauf zu erkennen. Zusätzlich führt die Modellierung der einzelnen Prozesse zur Modularisierung des Gesamtsystems. Einzelne Prozesse können ausgetauscht werden, ohne dass Veränderungen an anderen Prozessen notwendig sind (unter der Voraussetzung, dass die Schnittstellen nicht verändert werden). Im Fertigungsbereich ist die Verwendung von Workflows hingegen selten verbreitet. Stattdessen werden PPS-Systeme (Produktionsplanungs- und Steuerungssysteme) verwendet, um kurze Durchlaufzeiten, Termineinhaltung, optimale Bestandshöhen und die wirtschaftliche Nutzung der Betriebsmittel zu ermöglichen [WKNL07]. Die Durchsetzung von Workflows im Fertigungsbereich würde zu den selben Vorteilen führen wie bei Geschäftsprozessen. Fertigungsprozesse sind stärker auf physische Ereignisse (Maschine überhitzt, Materialknappheit, usw.) fokussiert als Geschäftsprozesse. Die Umsetzung von *Industrie 4.0* führt zusätzlich zu einem stetigen Ausbau von Sensortechnologien im Fertigungsbereich, wodurch Workflows mehr und mehr Informationen über den derzeitigen Status von Maschinen mitgeteilt werden können. Diese Informationen - auch Kontext genannt - führen dazu, dass aus statischen Workflows kontextbezogene Workflows entstehen. Kontextbezogenene Workflows verarbeiten den erhaltenen Kontext, um höherwertigen Kontext abzuleiten. Mit immer größer steigender Anzahl von Kontextdaten entsteht dadurch eine hohe Komplexität für Workflowdesigner. Verschiedene Kontextdaten müssen erfasst werden können und darauf folgende Abläufe integriert werden. Situationsbezogene Workflows dagegen lagern die Verarbeitung der Kontextdaten aus. Um die Modellierung zu vereinfachen, wird zuerst ein Standard-Workflow modelliert, der bei Nichtauftreten von Situationen ausgeführt wird. Um eine Adaption an erkannte Situationen zu ermöglichen, werden alle möglichen Ausnahmen und die dazugehörigen alternativen Abläufe des Workflows definiert. Diese sogenannten Workflow-Fragmente werden gestartet, sobald die entsprechende Situation erkannt wird. Zur Vereinfachung wird im Folgenden der Begriff Worfklow für situationsbezogene adaptive Workflows verwendet [WSBL15].

2.2. Kontext - Grundlagen und Definition

Es existieren mehrere Definitionen für den Begriff *Kontext*. In [Zwe11] wird strikt zwischen dem Begriff *Sensordaten*, welcher für reine Messdaten ohne logische Zusammenhänge steht, und *Kontext*, welcher die Relationen zwischen den gegebenen Sensordaten darstellt, unterschieden. Nach [ADB+99]

dagegen ist Kontext *jegliche Information, welche die Situation einer Entität charakterisiert. Eine Entität ist eine Person, ein Ort oder Objekt, welche relevant für die Interaktion zwischen Benutzer und Anwendung, einschließlich des Benutzers und der Anwendung selbst* . Im Rahmen dieser Arbeit ist die zweite allgemeinere Definition ausreichend.

Um zu verstehen, welche Schwierigkeiten im Umgang mit Kontext zu meistern sind, müssen zuerst die Eigenschaften analysiert werden. Eine sehr allgemeine Gliederung teilt Kontext in statischen und dynamischen Kontext auf [GBH+05].

- **Statischer Kontext** bleibt immer gleich (je nach Definition auch nur über einen sehr langen Zeitraum). Als Beispiel gelten personenbezogene Daten wie Geburtsdatum oder Geburtsort. Als statischer Kontext, welcher sich, falls überhaupt, nur sehr selten ändert, kann der Name einer Person betrachtet werden. Der Verlust dieser Daten ist sehr schwerwiegend.
- **Dynamischer Kontext** dagegen ändert sich häufig. Hierbei lassen sich zusätzlich weniger dynamischer Kontext (Beruf einer Person) sowie hochdynamischer Kontext (GPS-Position einer Person) unterscheiden. Der Verlust dieser Daten ist bei der nächsten Aktualisierung bereits unwichtig.

Um die Qualität des bereitgestellten Kontexts wiederzugeben, müssen weitere Merkmale betrachtet werden. Die obige Grobgliederung wird durch drei Untergruppen ergänzt.

- **Sensordaten** stellen dynamischen Kontext bereit. Je nach Bereich und verwendetem Sensor ist eine gewisse Fehlermarge zu erwarten. Auch Übertragungsfehler können zu verfälschten Ergebnissen führen. Mit einer immer größer werdenden Anzahl von Sensoren - ob physikalischen oder virtuellen - ist dies die Grundlage für die autonome Bereitstellung von Kontext.
- **Abgeleiteter Kontext** beschreibt die Verarbeitung von bereits vorhandenem Kontext (beispielsweise Sensordaten mehrerer Sensoren), um neue Informationen daraus zu gewinnen (Durchschnittswert aller Sensoren). Zu beachten ist, dass Fehler durch bestimmte Operationen (Abstand von zwei Smartphones) aufgrund einzelner fehlerhafter Sensordaten (GPS-Position eines Smartphones) vergrößert werden können. Geben in diesem Szenario die einzelnen Sensoren einen Fehler von einem Meter in entgegengesetzter Richtung, so entsteht durch die Operation *Abstand berechnen* ein Fehler von zwei Metern.
- **Vom Benutzer bereitgestellter Kontext** ist sehr zuverlässig, allerdings wird dieser bei Veränderungen selten aktualisiert. Ohnehin sollte diese Art von Kontext wenn möglich eine Ausnahme bleiben. Kontextbezogene Anwendungen, welche ausschließlich auf Sensordaten beruhen, sind völlig autonom.

Auch mit dieser Unterteilung sind keine genauen Angaben zur Qualität von Kontext (QvK) gegeben. Ergänzend müssen weitere Metadaten angegeben werden, um zuverlässige Aussagen über den Kontext treffen zu können. Da vom Benutzer bereitgestellter Kontext fehlerfrei sein sollte und abgeleiteter Kontext nur Fehler weiterführen kann, werden im Folgenden nur noch Sensordaten betrachtet. Wichtige Metadaten sind [WKL+09]

- **Genauigkeit** beschreibt die exakte Fehlermarge, die aufgrund der physikalischen Gegebenheiten eines Sensors gegeben sind (ist bei Produktion bekannt).

- **Aktualität** steht für den Zeitraum zwischen der Erfassung von Sensordaten und deren Verarbeitung. Je höher dieser Wert ist, desto geringer die QvK (Zeitstempel bei Erfassung wird mit Zeitstempel bei Verarbeitung verglichen - synchrone Uhren erforderlich).
- **Durchschnittliche Lebenszeit** beschreibt die durchschnittliche Zeit, in welcher die Sensordaten korrekt sind und ist speziell in Abhängigkeit der Aktualität zu betrachten. Ist die Lebenszeit höher, ist ein höherer Wert der Aktualität akzeptabel, ohne eine Verringerung der QvK herbeizuführen.
- **Korrektheit** beschreibt die Wahrscheinlichkeit, dass Sensordaten korrekt sind. Für diesen Wert müssen über einen gewissen Zeitraum diese speziellen Sensordaten beobachtet werden und anschließend eine Evaluation durchgeführt werden, die angibt, wie oft die gelieferten Sensordaten korrekte bzw. fehlerhafte Information lieferten.
- **Zuverlässigkeit des Kontext-Provider** charakterisiert den Provider im Hinblick auf die Korrektheit seiner Kontextdaten. Je öfter der Provider nicht korrekte Daten liefert, desto weiter sinkt dessen Zuverlässigkeit und die QvK seiner Daten.
- **Deduktionsgeschichte** ist speziell im Hinblick auf abgeleiteten Kontext zu betrachten. Je mehr Sensordaten oder andere abgeleitete Kontextdaten einem Kontext zugrunde liegen, desto niedriger ist seine QvK.

Die Kombination all dieser Metadaten ermöglicht eine qualitative Aussage über die QvK. Diese lässt sich nun beispielsweise als numerischer Wert zwischen 0 und 1 darstellen, wobei 1 für *sehr hochwertig* und 0 für *sehr unzuverlässig* stehen könnte. Des Weiteren können prozentuale oder absolute Abweichungen vom Normwert angegeben werden. Mit einer Standardisierung der QvK ist die Erstellung von Kontext-Providern möglich, die als Kontextbasis für viele verschiedene kontextbezogene Anwendungen dienen.

2.3. Systeme zur Verwaltung von Kontext

Das im Jahr 1999 gestartete Nexus-Projekt der Universität Stuttgart hat zum Ziel, eine dem Internet ähnliche Struktur namens *World Wide Space* zu erzeugen - eine globale Plattform von Kontextprovidern, die Umgebungsmodelle für mobile kontextbezogene Anwendungen bereitstellt. Die Architektur von Nexus ist in drei Ebenen unterteilt [LCG+09]:

- **Context Information Layer.** Auf dieser Ebene wird Kontext von Datenbanken mehrerer Anbieter bereitgestellt. Abhängig davon, welche Art von Kontext bereitgestellt wird, soll eine geeignete Datenbank verwendet werden. Zum Beispiel soll der Verlauf von sich bewegenden Objekten in speziellen *History Servern* abgelegt sein. Um die Datenmenge auf den Servern zu reduzieren, werden Kompressionsalgorithmen vorgeschlagen.
- **Federation Layer.** Der Federation Layer beinhaltet von Nexus bereitgestellte Services. Diese werden in *Core Service* (von Nexus benötigte Services) und *Platform Services* (optionale Services) unterteilt. Der Platform Service *Context Reasoning* ermöglicht die Deduktion von höherwertigem Kontext aus low-level-Kontext mithilfe von Situationstemplates (siehe Kap. 2.5), welche im Core

Service *Situation Repository* gespeichert sind. Context Reasoning erfolgt mittels klassischer Logik und Bayes'schen Netzen.

- **Application & Middleware Layer.** Die Anwendungsebene besteht aus kontextbezogenen Anwendungen und Workflows sowie zusätzlicher Middleware. Middleware Services können beispielsweise verwendet werden, um Anwendungslogik auszulagern und somit deren wiederholte Verwendung zu ermöglichen.

In [WKL+09] wird das Augenmerk auf unsichere Sensordaten und deren Verwendung in kontextbezogenen Workflows gelegt. Die Arbeit ist Teil des Nexus-Projekts und baut somit auf der vorgestellten Architektur auf. Um die QvK für kontextbezogene Workflows zu beachten, müssen dem Kontext auf dem Context Information Layer Metadaten über die verwendeten Sensoren hinzugefügt werden. Um stochastische Fehler zu verringern, werden Sensorwerte zu einem bestimmten Ereignis von mehreren Sensoren gemessen und anschließend miteinander vereint. Zusätzlich werden Bewertungen über die Sensoren und deren Qualität abgegeben, die mithilfe der Verwendung der *normalized weighted arithmetic mean method* [KBZ+08] und darauf aufbauendem Fuzzy Clustering dazu führen, dass abweichende Sensorwerte herausgefiltert werden können. Kontextbezogene Workflows können anschließend mit Context Quality Policies versehen werden. Bei jeder Anfrage für Kontextdaten wird die Policy mitgegeben, um sicherzustellen, dass nur Kontextdaten verwendet werden, die den Qualitätsansprüchen des Workflows genügen.

2.4. Situation - Grundlagen und Definition

Am Beispiel einer beliebigen Maschine A in einer Fertigungsfabrik soll der Nutzen der Deduktion von Kontext und einer daraus entstehenden Situation verdeutlicht werden. Maschine A produziert ein Produkt, welches von Maschine B weiterverwendet wird. Bei Ausfall von Maschine A ist zugleich die Produktion von Maschine B und möglichen weiter folgenden Maschinen eingeschränkt. Ein Workflow muss dieses Problem erkennen und alternative Prozesse starten beziehungsweise die nötigen Schritte zur Wiederinstandsetzung von Maschine A durchführen. Bezogen auf das Konzept *Industrie 4.0* sollte jede Komponente, die zu einem Fehler der Maschine führen könnte, mit Sensoren ausgestattet sein. Beispiele sind Temperatursensoren, die vor Überhitzung warnen sollen, Spannungssensoren, die bei Überspannung eine Notfallabschaltung einleiten und viele weitere. Ein situationsbezogener Workflow, der die gesamte Produktion überwacht, ist an diesen einzelnen Werten (z.B. der Werte aller Temperatursensoren) nicht interessiert. Wichtig ist der allgemeine Zustand einer Maschine, der sich beispielsweise in die Situationen *funktionsfähig*, *kritisch* und *funktionsunfähig* einteilen lässt. Die Deduktion zu diesen Situationen ist oftmals maschinenspezifisch und obliegt nicht der Workflow-Domäne. Das Ziel ist es somit, einen situationsbezogenen Workflow direkt mit den benötigten Situationen zu versorgen. Die zugrundeliegenden Kontextdaten sind jedoch nicht gänzlich zu vernachlässigen, da die Wiederherstellung der Funktionsfähigkeit einer Maschine davon abhängt.

[HHL+10] definiert eine Situation als *die Charakterisierung eines spezifischen, wiederkehrenden Umstands oder einer Konstellation in der realen Welt, die 1) idealisiert beschrieben werden könnte und 2) als Evaluationsbasis für die Adaption und Reaktion von kontextbezogenen Anwendungen dient.* Es gibt dabei keine klare Definitionsgrenze zwischen Kontext und Situationen. Vielmehr stellt eine Situation

Situation	Benötigter Kontext
Fabrik 100% funktionsfähig	Alle Maschinen funktionsfähig
Maschine funktionsfähig	Werkzeug funktionsfähig, Material vorhanden, Maschine nicht überhitzt
Werkzeug funktionsfähig	Benutzungsdauer des Werkzeug nicht überschritten, Werkzeug zeigt keine Mängel
Material vorhanden	Maschine besitzt ausreichendes Material
Maschine nicht überhitzt	Sensorwerte von Temperatursensoren unter kritischem Niveau

Tabelle 2.1.: Kontext als Situation

eine Abstraktion mehrerer Kontextdaten dar, wobei diese gleichzeitig auch Situationen sein können, was in Tabelle 2.1 beispielhaft dargestellt ist. Die Eigenschaften einer Situation hängen dabei stark von dem zugrundeliegenden Kontext ab, welcher zur Deduktion verwendet wurde.

2.4.1. Situationsmodell & Situationsobjekt

Die im Beispiel genannten Situationen sind stark vereinfacht. Erhält ein Workflow als Situation nur den String *kritisch*, fehlen wichtiger Kontext, um darauf basierend agieren zu können.

- Auf welche Maschine bezieht sich die Situation?
- Wann ist die Situation erkannt worden?
- Was bedeutet diese Situation?
- Wie wahrscheinlich ist es, dass bei der Erkennung der Situation kein Fehler aufgetreten ist?

Um diese und weitere Fragen zu beantworten, wird ein Situationsmodell gefordert, das alle vom Workflow benötigten Informationen beinhaltet. Des Weiteren besteht die Notwendigkeit, diese Informationen an den Workflow übergeben zu können. Das Situationsmodell dient als standardisierte Grundlage für die Erzeugung von Situationsobjekten. Diese Situationsobjekte können weitergegeben, gespeichert und von einem Workflow *verstanden* werden.

2.5. Situationstemplates - Grundlagen und Definition

Das Aufstellen der Konditionen sowie Interpretationen von Kontext stellen ab einer gewissen Größe einen großen Aufwand dar und erfordern domänenspezifisches Wissen der Sensoren. Um Konditionen an die Sensorwerte stellen zu können, muss bekannt sein, welcher Datentyp produziert wird (z.B Strings oder Zahlen, weiter unterteilt in Floats oder Integers), in welchem Format die Sensorwerte dargestellt werden (wird die Auslastung des Arbeitsspeichers prozentual angegeben oder die Anzahl verfügbarer Megabyte beziehungsweise Gigabyte dargestellt) und welche Qualitätsmerkmale vorhanden sind (z.B. um Toleranzabweichungen bestimmen zu können). Workflowentwickler können und sollten von diesem Prozess abgekoppelt werden. Dies führt zu einer neuen Entwicklungsebene

- dem Erstellen von Situationstemplates. Workflowentwickler geben an, welche Situationen von Interesse sind und welche Konditionen an die Sensorwerte gestellt werden. Die Details werden von Situationstemplatedesignern übernommen, die über sensorspezifisches Wissen verfügen.

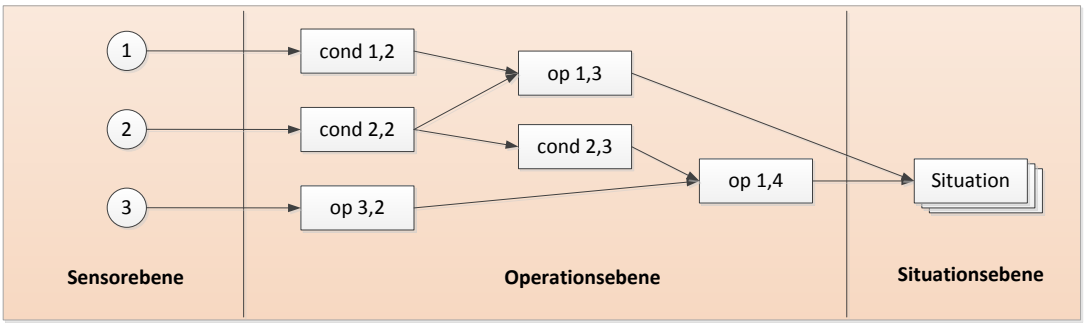

Abbildung 2.1.: Situationstemplate

Ein Situationstemplate stellt die Gesamtheit an benötigten Sensoren und den Konditionen, welche an die von den Sensoren gelieferten Sensorwerten gestellt werden, dar. Des Weiteren werden alle Operationen, dargestellt als logische Ausdrücke, die zur Deduktion des Kontexts benötigt werden, definiert. Werden alle Konditionen erfüllt und alle Operationen erfolgreich ausgeführt, ist eine Situation gültig, andernfalls ungültig. In dieser Variante definiert ein Situationstemplate genau eine Situation. Bildlich kann ein solches Situationstemplate wie in Abbildung 2.1 dargestellt werden. Die *Sensorebene* beinhaltet alle Sensoren und stellt den low-level-Kontext bereit. In der Operationsebene werden alle Konditionen überprüft (das Überprüfen einer Kondition stellt auch eine Operation dar) sowie Operationen zur Deduktion des low-level-Kontext durchgeführt, wodurch mid-level-Kontext entsteht. Die Zusammenführung des mid-level-Kontexts in der Situationsebene führt zum höchsten Abstraktionsgrad - dem high-level-Kontext. Diese Situation wird, wie in Kapitel 2.4 beschrieben, über ein Situationsmodell in ein Situationsobjekt transformiert. Die fertigen Situationstemplates können in einer Datenbank gespeichert werden und bei Bedarf in einem Situationserkennungssystem instanziiert werden, um Situationen zu erkennen.

Eine Alternative ist, mehrere Situationen in einem Situationstemplate zu definieren. Angenommen es existieren die Situationen *Maschine voll funktionsfähig*, *Maschine teilweise funktionsfähig*, *Maschine im kritischen Zustand* und *Maschine überlastet* und alle diese Situationen benötigen dieselben Kontextdaten. In diesem Fall ist es denkbar, diese vier Situationen in einem einzigen Situationstemplate zu definieren. Das Beispiel in Tabelle 2.2 zeigt, dass bei einer CPU- und RAM-Auslastung von beispielsweise 90% die Konditionen der Situationen *S2* und *S3* erfüllt sind. Um zu verhindern, dass mehrere Situationen gleichzeitig gültig sind, müssen den Situationen zusätzlich Prioritäten zugewiesen werden. Um in diesem Beispiel die *wichtigere* Situation *S3* zu erkennen, würde diese eine höhere Priorität als Situation *S2* erhalten. Gleichzeitig gilt, dass wenn eine Situation als gültig erkannt wird, alle anderen Situationen automatisch ungültig sind. Die Vorteile dieser Alternative sind die vielschichtige Unterteilung mehrerer Situationen. Allerdings nimmt mit der Anzahl von Situationen und zugehöriger Kontextdaten die logische Komplexität eines Situationstemplates stark zu, was wiederum

Situation	Konditionen
S1: Maschine voll funktionsfähig	CPU- und RAM-Auslastung unter 50% Auslastung.
S2: Maschine teilweise ausgelastet	CPU- und RAM-Auslastung über 50% Auslastung.
S3: Maschine im kritischen Zustand	CPU- und RAM-Auslastung über 80% Auslastung.
S4: Maschine überlastet	CPU- und RAM-Auslastung bei 100% Auslastung.

Tabelle 2.2.: Mehrere Situationen in einem Situationstemplate

die Erstellung von Situationstemplates erschwert. Im Rahmen dieser Arbeit werden deshalb vorerst Situationstemplates verwendet, die genau eine Situation definieren.

2.6. Systeme zur Situationserkennung

Es besteht bereits eine große Anzahl kontextbezogener Applikationen. Die Bereitstellung des Kontexts geschieht in der Regel mittels eine Adhoc-Variante. Alle Module sind stark miteinander gekoppelt. Die Weiterverwendung durch andere Applikationen beziehungsweise Workflows ist auf diese Weise nicht realisierbar. Bei Neuerstellung einer solchen Anwendung muss das gesamte Kontexterfassungssystem, alle Sensoren, Server und diverse Middleware neu implementiert werden. Der Grund dafür ist, dass die Entwicklung einzelner Anwendungen auf diese Art deutlich einfacher und schneller abläuft. Hierbei werden die Sensorentreiber direkt in der Applikation verwendet. Die Deduktion der Kontextdaten zu Situationen erfolgt auf Anwendungsseite. Die Alternative ist die Modularisierung aller beteiligten Systeme. Sensoren, Middleware und Applikation sind streng voneinander getrennt, wodurch deren Weiterverwendung ermöglicht wird. Neu erstellte Anwendungen können die bereits vorhandenen Module über Schnittstellen ansprechen [DAS00].

In [DAS00] wird auf dieser Basis ein Smart Environment am Beispiel eines intelligenten Hauses vorgestellt. Die Architektur der Kontextbereitstellung wird in drei Bereiche eingeteilt - Context Widgets, Context Servers und Context Interpreters. Context Widgets sind die direkte Schnittstelle zu den Sensoren. Anwendungen können sich direkt bei Context Widgets un/subscriben. Der Kontext kann entweder durch Polling oder Notifikationen bereitgestellt werden. Context Server stellen eine Subklasse der Context Widgets dar. Bei der Instanziierung eines Context Servers wird angegeben, bei welchen Context Widgets sich dieser anmelden soll. Die Anwendung kann sich daraufhin bei einem Context Server subscriben, anstatt sich bei allen einzelnen Context Widgets anmelden zu müssen. Die Speicherung von Daten obliegt den jeweiligen Modulen. Auf diese Weise kann eine Historie erstellt werden, die Verlaufsinformationen der jeweiligen Context Widgets beziehungsweise Context Server ermöglicht. Auf Basis dieser Informationen sind Anwendungen möglich, die Kontext sowie deduzierten Kontext mit gewissen Wahrscheinlichkeiten vorhersagen können, um noch intelligentere Systeme zu entwickeln. Context Interpreter sind optional und können zur Deduktion von low-level-Kontext (reine Sensorwerte) in higher-level-Kontext (Zustand) verwendet werden.

Problematisch ist die fehlende Verwaltung der einzelnen Module. Im Falle einer Vielzahl großer Anwendungen basierend auf vielen Sensorwerten entsteht eine enorm große Anzahl von Context Widgets. Trotz der Aggregation durch einen Context Server müssen die Context Widgets bei der

Instanziierung des Context Servers bekannt sein. Hierfür muss eine zusätzliche Verwaltungsebene erstellt werden, die alle Context Widgets sowie Context Server enthält, sodass Anwendungsentwickler auf einfache Weise benötigte Module finden können, ohne domänenspezifische Details zu kennen. Die Konditionen, die an die von den Context Widgets und Servers gestellt werden, stellen im Gegenzug einen unerwünschten Verwaltungsaufwand dar. Anstatt, dass sich eine Anwendung bei jedem einzelnen Modul, ob Context Widget oder aggregiert bei einem Context Server, anmeldet, ist aufwändig und unübersichtlich. Gleichzeitig wird zusätzlich ein Context Interpreter benötigt, welcher für die Deduzierung des Kontexts verwendet wird. Ein simpler logischer Ausdruck wie *if (Sensorwert1>10 & Sensorwert2>50) then Sensorwert3 = 30;* wird mit dieser Variante auf zwei Context Widgets und einen Context Interpreter aufgeteilt.

In [Zwe11] wird am Beispiel des RoboCup Szenarios das Verhalten von Robotern mithilfe von Situationen gesteuert. Hierbei spielen mehrere Roboter Fußball gegeneinander, weshalb Reaktionen im Millisekundenbereich verlangt werden, was eine effiziente Situationserkennung voraussetzt. Im Gegensatz zu [DAS00] werden Situationstemplates verwendet, um aufgestellte Konditionen zu vereinen. Ein Situationstemplate besteht aus einer SituationsID und einem Situationsschema. Zur Laufzeit werden mit einem Erkennungsalgorithmus, beispielsweise Bayes'schen Netzen, Situationsobjekte aus dem Situationstemplate mit Umweltdaten (Kontextdaten) instanziiert. Im Situationsschema sind alle Konditionen definiert, die für die Gültigkeit einer Situation benötigt werden. Mit einem Matching-Algorithmus wird das Situationsobjekt mit dem Situationsschema verglichen und falls alle Konditionen erfüllt sind, als gültig deklariert. Darüber hinaus ist auch die Schachtelung von Situationstemplates möglich, so dass ein Situationsobjekt mit mehreren Situationstemplates verglichen wird. Eine Möglichkeit zur Persistierung der Situationsobjekte ist nicht vorhanden.

3. Stand der Technik

3.1. NoSQL

NoSQL-Datenbanken stellen eine Alternative zu herkömmlichen relationalen Datenbanken dar. Der größte Unterschied besteht darin, dass NoSQL-Datenbanken kein Schema für die Strukturierung von Daten erfordern. Müssen bei relationalen Datenbanken zwei Elemente derselben Tabelle dieselben Attribute enthalten, können Elemente von NoSQL-Datenbanken komplett unterschiedlich aufgebaut sein. Diese Schemafreiheit bietet für Anwendungen mit heterogenen Datensätzen eine sinnvolle Alternative. Laut des CAP-Theorems [GL12] können Datenbanken nur zwei der drei folgenden Eigenschaften Konsistenz (Consistency), Verfügbarkeit (Availability) und Partitionstoleranz (Partition Tolerance) erfüllen. NoSQL-Datenbanken konzentrieren sich dabei auf Verfügbarkeit und Partitionstoleranz, im Gegenteil zu relationalen Datenbanksystemen, die statt Partitionstoleranz Konsistenz bevorzugen. Die bekanntesten Systeme, die (zumindest teilweise) auf NoSQL-Datenbanksystemen basierten, sind Facebook [FB] und Twitter [TWT]. Dies sind stark verteilte Systeme, die immer verfügbar sein müssen. Wann eine Benutzernachricht bei einem Facebook-Nutzer eintrifft, ist dagegen kein zeitkritischer Prozess und kann in dieser Hinsicht vernachlässigt werden. Trotzdem bieten einige NoSQL-Datenbanken (z.B. MongoDB [MDb]) die Möglichkeit an, den Konsistenzlevel auf Kosten der Partitionstoleranz selbst zu bestimmen.

3.1.1. Skalierbarkeit

Das beste Argument für die Verwendung von NoSQL-Datenbanksystemen liegt in deren Skalierbarkeit. Um die verbesserte Skalierbarkeit zu erklären, müssen die zugrundeliegenden Datenbankparadigmen ACID und BASE verglichen werden. Das Akronym ACID ist transaktionsbasiert, wird von relationalen Datenbanksystemen verwendet und beschreibt die folgenden Eigenschaften [Pri08]:

- **Atomicity.** Atomarität beschreibt die Eigenschaft, dass eine Transaktion entweder ganz oder gar nicht erfolgt. Wenn Teile der Transaktion nicht durchgeführt werden können (z.B. weil eine Datenbank nicht erreichbar ist), so müssen alle bereits erfolgten Änderungen wieder rückgängig gemacht werden. Dies erhöht den Verwaltungsaufwand und erschwert somit die Skalierung.
- **Consistency.** Konsistenz (in diesem Sinne *strong consistency*) beschreibt, dass bei einem Schreibvorgang die Änderungen an jeden weiteren Knoten weitergeleitet und durchgeführt werden müssen. Ein darauffolgender Lesevorgang auf einem beliebigen Knoten gibt jeweils das selbe Ergebnis zurück. Bei stark verteilten Datenbanksystemen, bei denen zwischenzeitlich Knoten ausgefallen sein könnten oder nicht erreichbar sind (keine Internetverbindung), ist die Einhaltung eines konsistenten Zustands nur schwer zu erreichen.

- **Isolation.** Isolation beschreibt, dass Transaktionen isoliert voneinander ablaufen. Daraus folgt, dass das gleichzeitige Ausführen von Transaktionen zu dem gleichen Ergebnis kommt wie deren sequentielle Ausführung.
- **Durability.** Dauerhaftigkeit besagt, dass sobald eine Transaktion erfolgreich abgeschlossen wurde, der neu herbeigeführte Zustand dauerhaft bestehen bleibt, auch nach Fehlern oder Systemausfällen, weswegen die Daten nach einem Schreibvorgang auf ein sicheres Speichermedium *geflusht* werden müssen. Im Gegensatz dazu basieren einige NoSQL-Datenbanken wie Redis oder Memcache auf Cache-basierten Speichersystemen, um die Zugriffszeiten deutlich zu verringern.

Hauptsächlich wurde BASE (dt. Lauge) als Akronym verwendet, um den Gegensatz zu ACID (dt. Säure) darzustellen und steht für ***B**asically **A**vailable, **S**oft State, **E**ventually Consistent*. Es bestehen keine eindeutigen Definitionen für *Basically Available* und *Soft State*. *Eventually Consistent* beschreibt die Eigenschaft, dass nach einem Schreibvorgang irgendwann alle Daten wieder in einem konsistenten Zustand sind, dieser Zeitpunkt allerdings nicht genauer spezifiziert werden kann. Somit ist es in Ordnung, veraltete Daten zu verwenden, Diese Abschwächung des Konsistenzlevels ermöglicht deutlich schnellere Schreibvorgänge, da nicht zuerst auf die Erfüllung der in ACID beschriebenen Eigenschaften gewartet werden muss [Pri08].

Systeme, bei denen Skalierbarkeit und Geschwindigkeit wichtiger ist als die Konsistenz der Daten, finden in NoSQL-Datenbanken eine sinnvolle Alternative zu relationalen Datenbanken.

3.2. Cloud Computing

Herkömmliche Rechnerinfrastrukturen von Unternehmen sind darauf ausgelegt, das Maximum an benötigten Kapazitäten bereitzustellen. Dieses Maximum wird oftmals durch kurzzeitige Kapazitätsspitzen definiert. In der restlichen Zeit werden die bereitgestellten Kapazitäten nicht benötigt, obwohl trotzdem Kosten für die Verwaltung, Instandhaltung und den Lagerplatz entstehen. Die Bereitstellung neuer Kapazitäten kann mehrere Wochen dauern und muss zusätzlich in das bestehende System integriert werden [AFG+10].

Cloud Computing beschreibt das Konzept, IT-Infrastrukturen virtualisiert zur Verfügung zu stellen. Hierbei werden die physischen Ressourcen einer Maschine in logische Ressourcen unterteilt. Voneinander unabhängige virtuelle Maschinen greifen auf die logischen Ressourcen zu, wodurch der Nutzungsgrad um ein Vielfaches gesteigert werden kann. Auf diese Weise können Benutzern und Prozessen virtuelle Maschinen zugeteilt werden, die optimal auf die ausgeführten Prozesse abgestimmt ist [Ley09].

Die Bereitstellung einer Cloud lässt sich allgemein in vier Service-Modelle einteilen [MG09]:

1. **Public Cloud.** Eine Public Cloud beschreibt das Konzept, das eine Cloud öffentlich bereitgestellt wird und von jedem verwendet werden kann. Virtuelle Maschinen können gebührenpflichtig bei einem Cloud Provider (z.B. Amazon AWS [aws]) für die Dauer der Benutzung gemietet werden und bieten die einfachste Möglichkeit, eine anforderungsgerechte Infrastruktur zu erzeugen. Zusätzlich werden Funktionalitäten wie Load Balancing (gleichmäßige Verteilung

der Last auf alle gemieteten virtuellen Maschinen) oder Auto Scaling, dass bei Auslastung der Maschinen automatisch weitere Maschinen mietet. Die Verwaltung und Instandhaltung der Infrastruktur sowie die Erfüllung nicht funktionaler Anforderungen wie Sicherheitsrichtlinien obliegt dem Cloud Provider, was dem Benutzer zusätzliche Kosten einspart. Kapazitätsspitzen können durch Mieten weiterer virtueller Maschinen innerhalb von Minuten abgefangen werden. Nicht benötigte Maschinen können entfernt werden, um Kosten zu sparen. Vor allem schnell wachsende Startup-Unternehmen profitieren von der schnellen Bereitstellung und dem Pay-as-you-go Konzept.

2. **Private Cloud.** Die Private Cloud ermöglicht die exklusive Nutzung für einzelne Organisationen. Das Managen dieser Cloud kann durch ein Drittunternehmen stattfinden. Der Zugang findet über das Intranet des Unternehmens statt. Hier wird meist das firmeneigene Rechenzentrum zur Bereitstellung verwendet, wodurch nur eine indirekte Kosteneinsparung erfolgt, dafür allerdings die Sicherheitsrichtlinien selbst gesetzt werden können. Dies ist das meist verwendete Modell, da das Unternehmen die gesamte Kontrolle über die Cloud behält. Die Verwaltung einer Private Cloud kann von einem Drittunternehmen übernommen werden.

3. **Hybrid Cloud.** Deutsche Unternehmen sind über das Datenschutzgesetzes dazu verpflichtet, personenbezogene Daten innerhalb Deutschlands aufzubewahren. Weitere Gründe wie der Patriot Act [pa] bewegen deutsche Unternehmen dazu, amerikanische Cloud Provider zu meiden und entweder eine Private Cloud oder deutsche Cloud Provider zu verwenden. Eine Alternative ist die Verwendung einer Hybrid Cloud, die eine Kombination der Modelle Public und Private Cloud darstellt, wobei die einzelnen Clouds über Schnittstellen miteinander verbunden sind.

4. **Community Cloud.** Die Community Cloud ist eine Private Cloud, die von mehreren Organisationen gemeinsam genutzt wird. Beispielsweise können Firmen und deren Zulieferer dieselbe Cloud verwenden, um die Kommunikation untereinander zu vereinfachen. Wie auch die Private Cloud kann die Community Cloud von einem Drittunternehmen verwaltet werden.

Insgesamt wird die Kosteneinsparung durch Nutzung von Cloud Computing zwischen den Jahren 2010 und 2015 auf etwa 50 Milliarden US-Dollar geschätzt [ceb10]. Für den Einsatz in SitOPT ist vor allem die Eigenschaft der Elastizität von Interesse. Elastizität beschreibt, dass Kapazitäten dem Benutzer immer verfügbar sind und sich die Menge verwendeter Kapazitäten an die aktuellen Anforderungen anpasst. Diese Eigenschaften werden als Verfügbarkeit und Skalierbarkeit definiert [Ley09]:

1. **Skalierbarkeit.** Anwendungen in der Cloud können vor allem bei der Verwendung einer Public Cloud vergleichsweise schnell und beliebig hoch skalieren. Durch unterstützende Funktionalitäten wie Auto Scaling wird die Skalierung automatisiert. Voraussetzung dafür ist, dass die Anwendung parallelisierbar ist und somit auf verschiedenen Maschinen gleichzeitig ausgeführt werden kann.

2. **Verfügbarkeit.** Verfügbarkeit beschreibt die Eigenschaft, wie oft es zu Ausfallzeiten der Rechnerinfrastruktur kommt. Hochverfügbare Systeme sollten Ausfallzeiten von höchstens wenigen Minuten im Jahr besitzen. Bei der Verwendung einer Cloud können ausgefallene Maschinen sofort durch neue Maschinen ersetzt werden, wodurch die Ausfallzeiten für den Benutzer zusätzlich minimiert werden.

Für den Anwendungsfall einer Situationserkennung sind die Voraussetzungen optimal. Will der Benutzer eine Situationserkennung starten, stehen immer ausreichend Kapazitäten zur Verfügung. Werden die überwachten Objekte ausgeschaltet und die Situationserkennung gestoppt, werden die gemieteten Kapazitäten wieder entfernt. Hochsensiblen Produktionsabläufen wird durch die Minimierung von Ausfallzeiten eine durchgehend zuverlässige Situationsüberwachung gewährleistet. Die Verwendung einer Private Cloud bietet vor allem für den Prototyp von SitOPT die Vorteile einer schnellen und ressourcensparenden Bereitstellung.

3.3. Complex Event Processing (CEP)

CEP - das Verarbeiten komplexer Ereignisse - beschreibt das Verfahren, auf einer Menge logisch verknüpfter Ereignisse höherwertige Informationen zu gewinnen. Ein Ereignis ist hierbei ein Objekt, dass eine Aktivität in einem System darstellt. Ereignisse sind voneinander abhängig und können in Folge ihrer Verarbeitung neue Ereignisse produzieren [Rob10]. Zu den wichtigsten Anwendungsgebieten zählen das *Business Activity Monitoring*, der Einsatz in Sensornetzwerken sowie die Erforschung von Marktdaten. Vor allem der Einsatz in Sensornetzwerken entspricht dem Anwendungsfall einer Situationsüberwachung. Der Ablauf eines CEP-Systems wird in Abbildung 3.1 gezeigt. Im Folgenden werden die einzelnen Schritte am Beispiel einer Situationserkennung vorgestellt.

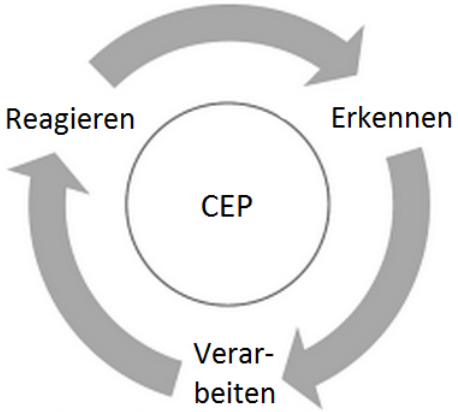

Abbildung 3.1.: CEP-Zyklus [RB15, pp. 6]

- **Erkennen.** Das Erkennen von Ereignissen (Situationen) erfolgt über das Bereitstellen von Sensorwerten durch Sensoren. Gleichzeitig werden statischer Kontext über Wissensdatenbanken bereitgestellt. Mithilfe dieses Kontext soll der Zustand eines virtuellen oder physischen Objekts abgeleitet werden. Von hoher Bedeutung ist die Geschwindigkeit, mit der dynamischer Kontext erkannt und weiterverarbeitet wird, um einen aktuellen Zustand zu erkennen. Bei Sensorwerten, die sich beispielsweise jede Sekunde ändern, ist eine Situation, die auf fünf Minuten alten Sensorwerten basiert, längst obsolet.
- **Verarbeiten.** Bei der Verarbeitung wird zuerst eine Analyse der Kontextdaten durchgeführt. Hierbei können beispielsweise die von einem Situationstemplate definierten Konditionen an

die Sensorwerte überprüft werden. Darauf folgend können beliebige Operationen auf den Kontextdaten durchgeführt werden.

- **Reagieren.** Abhängig davon, welche Ergebnisse in der Verarbeitungsphase entstanden sind, können vorab definierte Aktionen durchgeführt werden, beispielsweise die Benachrichtigung von Personal oder das Weitersenden der deduzierten Kontextdaten an weitere Services.

CEP verwendet SQL ähnliche EPL (Event Processing Language) Statements, um zu überprüfen, ob der Kontext die definierten Konditionen erfüllt. Das folgende Statement

```
EPStatement cepStatement = cepAdm.createEPL("select * from Situation having cpu > 80.0 AND ram >
    600.0");
cepStatement.addListener(new CEPListener());
```

überprüft Objekte der Klasse *Situation* im Hinblick auf die Werte *cpu* und *ram* und vergleicht diese mit den angegebenen Schwellenwerten. Anschließend wird dem Statement ein Listener hinzugefügt, um neue Situationen zu erkennen.

4. Konzept und Architektur von SitOPT

Im Folgenden werden die Module von SitOPT konzeptionell beschrieben. Zusätzlich wird der Ablauf von der Situationsüberwachung bis hin zur Situationsbereitstellung an einen Workflow demonstriert. Abbildung 4.1 demonstriert einen Gesamtüberblick über die einzelnen Module von SitOPT. Diese werden in den folgenden Kapiteln detailliert vorgestellt.

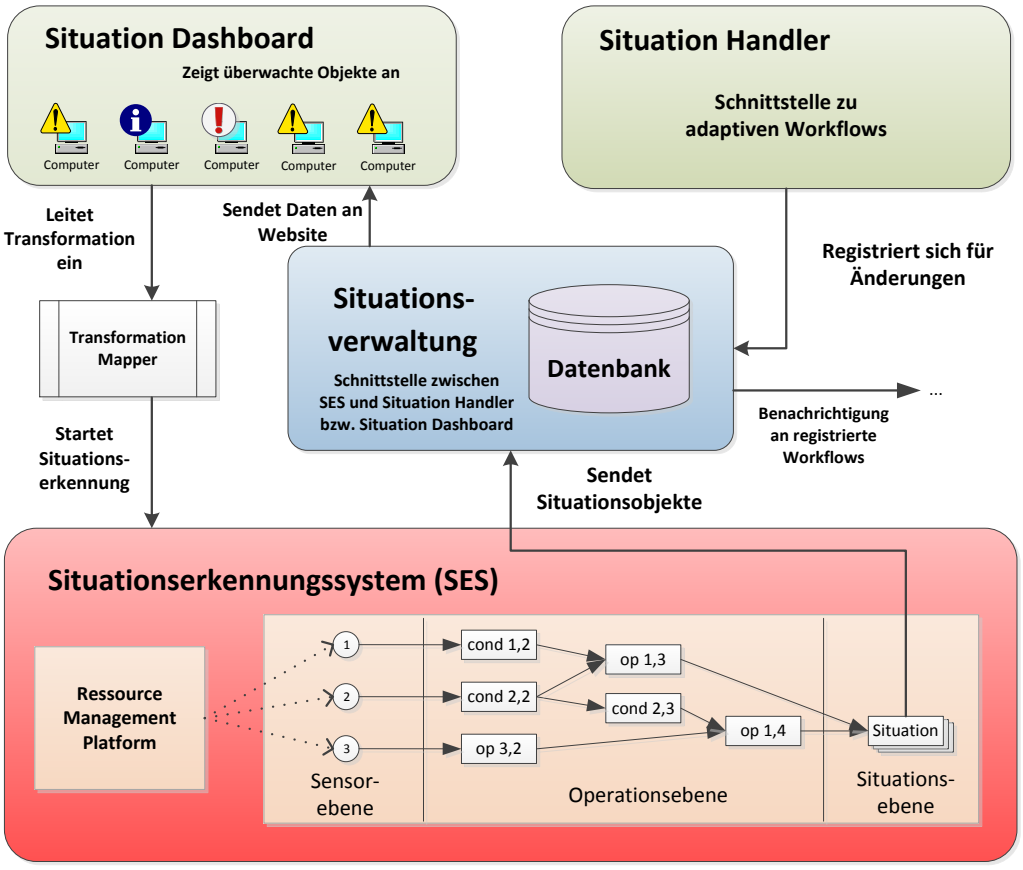

Abbildung 4.1.: Gesamtarchitektur von SitOPT

Bei der Architektur wurde auf eine modulare Bauweise geachtet, um SitOPT flexibel zu gestalten und zugrundeliegende Module oder Technologien erweitern beziehungsweise austauschen zu können.

Dabei sind die Module *Ressource Management Platform, Transformation Mapper* und *Situation Handler* nur indirekt Teil dieser Arbeit und werden dementsprechend weniger detailliert beschrieben. In der Abbildung 4.1 ist die Schnittstellenfunktion des Datenbanksystems zwischen den Modulen des Situationserkennungssystems sowie der adaptiven Workflows beziehungsweise des Situation Dashboards zu erkennen. Um die entstandenen Situationen an das Workflowsystem weiterleiten zu können, werden diese in einer Document-Based Datenbank gespeichert sowie nötige Veränderungen vorgenommen, um die Spezifikationen der Datenmodelle zu erfüllen. Daraufhin können die Workflows die von ihnen benötigten Daten aus der Datenbank auslesen. Das Situationserkennungssystem sowie die Datenbank befinden sich in einer Cloud und sind somit von überall zugreifbar und skalierbar. Im Folgenden wird ein detaillierter Überblick über die wichtigsten Komponenten von SitOPT geliefert.

4.1. Situationserkennungssystem

Um die Abkopplung der Situationserkennung von der Workflow-Ebene zu ermöglichen, werden Situationserkennungssysteme (SES) vorgestellt. Hier werden auf Basis des verwendeten Situationstemplates Situationen erkannt, die im späteren Verlauf Workflows zur Verfügung gestellt werden. Des Weiteren ermöglicht die Modularisierung die Verwendung verschiedener SES, ohne Änderungen an anderen Modulen vornehmen zu müssen. Wichtige Eigenschaften eines SES sind unter anderem Effizienz, Skalierbarkeit und Parallelisierung. Die Visualisierung eines SES kann von Vorteil sein, um Abläufe in der Situationserkennung zu betrachten und zu verstehen. Die Abläufe innerhalb eines SES können in folgende Bereiche gegliedert werden:

1. **Kontext erfassen:** Die Erfassung von Kontext dient als Grundlage für das Erkennen von Situationen. Die im Situationstemplate definierten Sensoren werden in der Sensorebene ausgelesen. Beim Start einer Situationserkennung muss dem SES mitgeteilt werden, wo die benötigten Sensorwerte vorliegen (z.B. URL des Sensors). Dies erfolgt über das überwachte Objekt, dass vorhandene Sensoren referenziert.
2. **Kontext verarbeiten:** In der Operationsebene werden zuerst die im Situationstemplate definierten Konditionen an die Sensorwerte überprüft. Wird mindestens eine Kondition nicht erfüllt, wird die Situation als ungültig betrachtet. Des Weiteren können beliebige Operationen auf dem Kontext durchgeführt werden. Werden alle Konditionen erfüllt und alle Operationen erfolgreich ausgeführt, gilt eine Situation als gültig.
3. **Situationsobjekt erstellen:** Unabhängig davon, ob eine Situation gültig oder ungültig ist, wird auf der Situationsebene ein Situationsobjekt erstellt, dass alle im Verlauf der Situationserkennung verwendeten Daten enthält.

Wie zu sehen ist, entspricht diese Vorgehensweise den beiden Punkten *Erkennen* und *Verarbeiten* aus dem in Kapitel 3.1 vorgestellten CEP-Zyklus. Workflows, die auf Basis erkannter Situationen *reagieren*, vervollständigen den CEP-Zyklus. Somit können beliebige CEP-Systeme für eine Situationserkennung verwendet werden. Die anschließende Verarbeitung des Situationsobjekts kann auf zwei Möglichkeiten realisiert werden:

1. Die erste Möglichkeit beschreibt die Vorgehensweise, dass Situationsobjekte nur dann gespeichert werden, wenn sich die Gültigkeit ändert. Solange alle Situationen ungültig sind, wird der Standard-Workflow ausgeführt. Erst sobald eine Situation erkannt wird, muss das entsprechende Workflow-Fragment initialisiert werden. Wird dieselbe Situation im Anschluss nochmals als gültig erkannt, ist dies für den Workflow nicht von Interesse, da das Workflow-Fragment bereits ausgeführt wird und somit keine weiteren Schritte eingeleitet werden müssen. Sobald daraufhin die Situation wieder als ungültig erkannt wird, kann der Workflow die Ausführung des Workflow-Fragments wieder beenden.
2. Obwohl Workflows keinen direkten Nutzen aus nacheinander mehrfach erkannten Situationen mit derselben Gültigkeit haben, kann es von Vorteil sein, diese in der Datenbank zu speichern. Die Speicherung aller Situation führt zu einer Erstellung einer Historie von Situationen. Monitoringsysteme können diese Historie auswerten und mögliche Optimierungen im Ablauf aufweisen. Die Speicherung aller Situationen kann zu einem großen Overhead bezüglich des gespeicherten Datenvolumens führen.

Resource Management Platform (RSM)

Die im SES verwendeten Sensordaten können über die *Resource Management Platform*, die auf dem Konzept des OSLC (Spezifikation für die Integration von Werkzeugen) basiert, ausgelesen werden. Die RSM verwendet eine Push&Pull Methode. Um jederzeit eine Situationserkennung durchführen zu können, werden spezielle OSLC-Adapter verwendet, um die Sensorwerte von den Sensoren in einen Data Cache der RSM zu pushen und zu speichern. Anderenfalls sind möglicherweise keine Sensorwerte vorhanden, um Situationen abzuleiten. Zu beachten ist, dass mit der Lebenszeit von Sensorwerten die Qualität enorm sinken kann. Wird eine Situationserkennung durchgeführt, werden die benötigten Sensorwerte von einem OSLC Service aus dem Data Cache gepullt und können somit dem SES als REST Ressource zur Verfügung gestellt werden [HWS+15]. Die Implementierung der RSM ist nicht Teil dieser Arbeit.

4.2. Transformation Mapper

Situationstemplates liegen in einem XML-Format vor. Die Verwendung mehrerer Systeme zur Situationserkennung fordert daher eine Transformation eines Situationstemplates in ein ausführbares Situationstemplate für das gewählte SES. Der Transformation Mapper enthält alle Mappingalgorithmen verfügbarer SES und wird beim Start einer Situationserkennung durch das Situation Dashboard aufgerufen. Nach erfolgreicher Transformation startet der Transformation Mapper das SES mit dem ausführbaren Situationstemplate. Die Implementierung des Transformation Mappers ist nicht Teil dieser Arbeit.

4.3. Situationsverwaltung

Die Situationsverwaltung dient in erster Linie dazu, Workflows und dem *Situation Dashboard* Situationen bereitzustellen. Darüber hinaus stellt dieses Modul die Schnittstelle zur Datenbank dar. Auf diese Weise können von der Datenbank nicht vorhandene Funktionen durch die Situationsverwaltung bereitgestellt werden. Zusätzlich müssen alle von SitOPT benötigten Funktionen implementiert werden. Da kein direkter Datenbankzugriff möglich ist, stellt die Situationsverwaltung die grundlegenden Funktionen Erstellen, Lesen, Ändern sowie Löschen von Ressourcen zur Verfügung. Eine Ressource stellt im Kontext von REST ein einzelnes Dokument dar, eine Listenressource eine Menge von Ressourcen. Im Rahmen dieser Arbeit ist eine Listenressource eine Menge von Ressourcen desselben Typs (z.B. nur Situationen). Im Folgenden werden Anforderungen an die bereitgestellten Funktionen beschrieben:

- **Erstellen von Ressourcen.** Diese Funktion ermöglicht dem SES das Speichern von Situationsobjekten und den dazugehörigen Sensorwerten innerhalb der Datenbank. Des Weiteren ermöglicht sie Benutzern die Erstellung von Sensoren, Things und Situationtemplates, welche anschließend von SitOPT verwendet werden können. Um zu überprüfen, ob die erzeugte Ressource alle von SitOPT benötigten Attribute enthält, wird eine Schemavalidierung mithilfe der in Kapitel 4.7 gezeigten Datenmodelle durchgeführt. Zusätzlich fügt die Situationsverwaltung eigenständig Metadaten hinzu. Beispielsweise ist dem SES der Name einer erkannten Situation nicht bekannt, welcher allerdings im Situationstemplate definiert ist. Da dem SES das ausgeführte Situationstemplate bekannt ist, wird beim Einfügen einer Situation über das referenzierte Situationstemplate der Name der Situation ausgelesen und dem Situationsobjekt hinzugefügt.
- **Ändern von Ressourcen.** Um die Verwaltung von Situationen zu vereinfachen, soll das kontinuierliche Erkennen einer einzigen Situation nicht zur Erstellung multipler Ressourcen führen. Stattdessen wird eine Ressource geändert, wobei ältere Versionen beibehalten werden. Des Weiteren muss es möglich sein, statische Ressourcen wie Sensoren, Things und Situationstemplates zu verändern.
- **Lesen von Ressourcen.** Das Lesen von Ressourcen soll dem Benutzer die Möglichkeit geben, Ressourcen zu erhalten. Hierfür sollen neben der ID einer Ressource weitere Attribute als Abfragekriterium möglich sein. Beispielsweise lässt sich eine Situation neben der ID eindeutig als Kombination eines Things mit einem Situationstemplate definieren.
- **Löschen von Ressourcen.** Um die referenzielle Integrität von Ressourcen zu gewährleisten, muss ein kaskadierender Löschvorgang gestartet werden. Dies bedeutet, dass wenn eine Ressource A gelöscht werden soll, alle Ressourcen, die Ressource A referenzieren, ebenso gelöscht werden müssen.

Um Fehlverhalten von Benutzern zu vermeiden, soll zusätzlich eine Benutzerverwaltung integriert werden, die die Sicherheit und Funktionsweise von SitOPT gewährleisten soll. Auf diese Weise kann beispielsweise sichergestellt werden, dass nur ein SES die Möglichkeit besitzt, Situationsobjekte zu erzeugen.

4.3.1. Situationsbereitstellung

In Abbildung 4.2 ist die anfängliche Methode zur Situationsbereitstellung zu sehen. Statt jeden Zugriff über eine Datenbank durchzuführen, werden alle Situationen des SES an eine Messaging Queue gesendet. Eine Queue kann mehrere Topics enthalten, auf welchen Situationen bereitgestellt werden können. Der Topic *Maschine1/sitTemp1* enthält alle Situationsobjekte, die bei der Situationserkennung für die Maschine *Maschine1* mit dem Situationstemplate *sitTemp1* erkannt wurden. Um über erkannte Situationen informiert zu werden, muss sich der Situation Handler bei einem Topic *subscriben*. Das Messaging-System *published* bei jeder Änderung auf diesem Topic die neuen Situationen an alle *Subscriber*. Bei dieser Variante fehlen Funktionalitäten wie z.B. die oben angesprochene Methode, nur Situationen zu speichern (und damit auch an den Situation Handler weiterzuleiten), die sich geändert haben. Der Vorteil dieser Methode ist eine höhere Geschwindigkeit der Situationsbereitstellung im Vergleich zur zweiten Methode, da keine Datenbankzugriffe benötigt werden. Das SES sendet zwecks Historienerstellung weiterhin Situationsobjekte an die Datenbank.

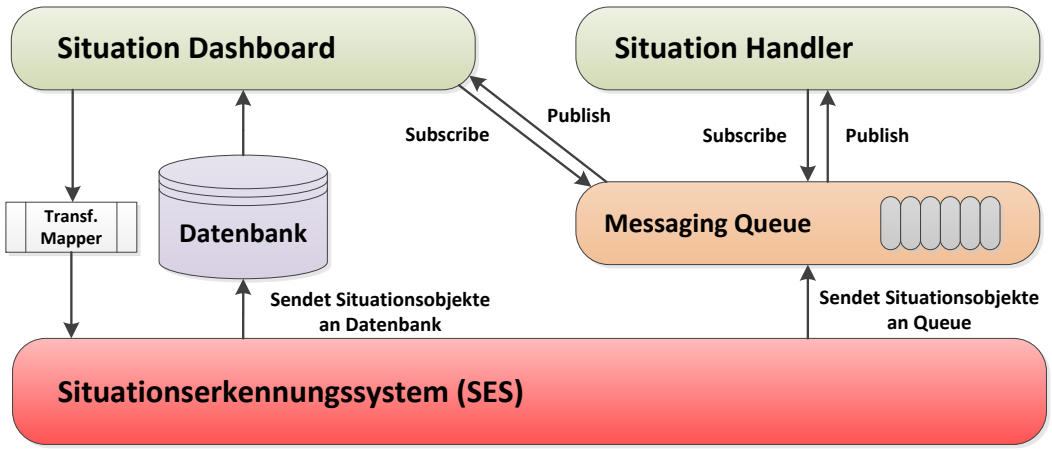

Abbildung 4.2.: Message Queue als Situationsbereitstellung - vereinfachte Darstellung

Die in SitOPT derzeit verwendete Methode, ist die Situationsbereitstellung mittels der Situationsverwaltung, wie es in Abbildung 4.1 zu sehen ist. Alle Situationsobjekte werden zuerst in einer Datenbank gespeichert. Daraufhin kann sich der Situation Handler bei der Situationsverwaltung auf spezifische Situationen anmelden. Des Weiteren ist es möglich, sich auf alle möglichen Situationen anzumelden. Bei einer Veränderung oder Neuentstehung einer Situation wird überprüft, ob auf dieser Situation Registrierungen vorhanden sind. Falls ja, wird die Situation an eine von dem Situation Handler bei der Registrierung angegebene URL gesendet. Der Situation Handler kann bei der Anmeldung

angeben, ob fortlaufend Benachrichtigungen gesendet werden sollen oder nur beim erstmaligen Auftreten einer Situation. Bei Angabe der letzteren Variante wird die Anmeldung im Anschluss an die Benachrichtigung gelöscht. Soll ein Workflow nicht weiterhin über Situationen informiert werden, muss der Situation Handler eine Abmeldung bei der Situationsverwaltung durchführen.

4.3.2. Datenbank

Die Datenbank verwaltet alle von SitOPT benötigten und daraus entstehenden Daten. Aufgrund der Situationsverwaltung sind die Anforderungen an die Funktionalitäten der Datenbank geringer. Sicherheitsspezifische Funktionen wie Benutzerverwaltung können beispielsweise von der Situationsverwaltung übernommen werden. Nichtsdestotrotz ist es von Vorteil, bereits bestehende Funktionalitäten übernehmen zu können. Das Hauptaugenmerk liegt deshalb auf den Eigenschaften der Datenbank. Vor allem die Skalierung der Datenbank ist für SitOPT von größter Wichtigkeit. Angenommen alle 5 Sekunden wird eine Situation für ein Objekt erkannt und jede erkannte Situation des SES wird gespeichert. Werden in diesem Szenario 20 Objekte über einen Zeitraum von einem Tag überwacht mit einer Datengröße von 5 Kilobyte pro Situationsobjekt, so entsteht in diesem Zeitraum eine Datengröße von 1,728 Gigabyte. Zu entscheiden ist, wie die Daten sinnvoll auf verschiedene Datenbanken aufgeteilt werden können, um Zugriffszeiten zu minimieren, die Anzahl der Datenbanken möglichst gering zu halten und ein ausfallsicheres System ohne *Single Point of Failure* zu gewährleisten. Gleichzeitig müssen Sicherungskopien der vorhandenen Daten erstellt werden.

4.4. Situation Handler

Wie bereits in Kapitel 2.1 beschrieben, ist die Komplexität von Workflows für industrielle Prozesse enorm hoch. Das Erzeugen von Situationsobjekten abseits der Workflow-Ebene erleichtert den Umgang mit erkannten Situationen. Wird ein Workflow über eine neu erkannte Situation informiert, können auf dieser Basis Workflow Fragmente ausgeführt werden, um die Situation zu handhaben. Der Situation Handler dient als Komponente für den Umgang mit erkannten Situationen als Schnittstelle zwischen SitOPT und einzelnen Workflows. Dazu meldet sich der Situation Handler bei der Situationsverwaltung an und wird über Änderungen und neu erkannte Situationen informiert. Die Implementierung des Situation Handlers ist nicht Teil dieser Arbeit.

4.5. Situation Dashboard

Das Situation Dashboard stellt eine visuelle webbasierte Möglichkeit dar, die von SitOPT verwendeten Informationen darzustellen. Jeder Ressourcentyp soll auf einer eigenen Seite aufgelistet werden. Hierbei ist vor allem die Überwachung der Things von Interesse. Der Benutzer soll auf einfache und übersichtliche Weise über alle überwachten Objekte und deren derzeitige Situationen informiert werden. Neben der Auflistung aller Objekte sollen zugehörige Situationen mit Icons intuitiv dargestellt werden.

Des Weiteren besteht die Möglichkeit, die Situationserkennung zu starten. Eine Situationserkennung wird mit folgenden Optionen gestartet:

- **Objekt.** Der Benutzer wählt das Objekt, das überwacht werden soll. Es können nur Objekte überwacht werden, die in der Datenbank gespeichert sind. Objekte können mehrfach überwacht werden.
- **Situationstemplate.** Situationstemplates definieren, welche Situationen für ein Objekt erkannt werden können. Zu beachten ist, dass nicht jedes Situationstemplate für jedes Objekt verwendet werden kann. Das Objekt muss die im Situationstemplate definierten Sensoren enthalten. Um die Benutzung zu vereinfachen, sollen dem Benutzer deshalb nur Situationstemplates angezeigt werden, die für das ausgewählte Objekt möglich sind.
- **SES.** Der Benutzer soll bei jeder neu gestarteten Situationserkennung zwischen den durch SitOPT unterstützten SES wählen können. Unterschiede verschiedener SES sind beispielsweise bezüglich Skalierbarkeit, Effizienz und Parallelisierung zu finden. Zugleich bieten manche SES eine Visualisierung der Situationserkennung und somit eine Visualisierung des Situationstemplates bereit.
- **Speichermodus.** In Kapitel 4.1 wurden zwei Möglichkeiten besprochen, wann Situationsobjekte gespeichert werden sollen. Per Angabe des Speichermodus soll der Benutzer bei jeder individuellen Situationserkennung selbst entscheiden, ob alle erkannten Situation gespeichert werden sollen oder nur jede, deren Gültigkeit sich geändert hat und bestimmt den Kompromiss zwischen Monitoring und geringerem Datenvolumen.

Des Weiteren sollen Objekte in ihrer Umgebung dargestellt werden können. Über Positionsangaben soll eine interaktive Karte mit allen Objekten und Situationen angezeigt werden, um einen schnellen Überblick über alle überwachten Objekte zu erhalten.

Zusätzlich sollen alle von der Situationsverwaltung bereitgestellen Funktionen angezeigt werden. Um den Umgang mit SitOPT erheblich zu vereinfachen, soll neben der Dokumentation der Funktionen die Möglichkeit bestehen, über das Situation Dashboard alle Funktionen der Situationsverwaltung zu testen, ohne einen eigenen Client implementieren zu müssen. Hierbei werden zugleich die Datenmodelle der einzelnen Ressourcen sichtbar.

Visualisierbare SES sollen über das Situation Dashboard aufgerufen werden können, um laufende Situationserkennungen anzeigen zu lassen. Wird eine Situationserkennung mit einem visualisierbaren SES gestartet, soll der Aufruf automatisch erfolgen, um die aktuelle und bereits laufende Situationserkennungen anzuzeigen.

4.6. Funktionsweise

Nachdem alle beteiligten Module vorgestellt worden sind, wird im Folgenden der Gesamtablauf demonstriert, der zum Erstellen von Situationsobjekten führt. In Abbildung 4.3 ist das Zusammenspiel aller Module zu erkennen. Die Datenbank enthält die Objekte und Situationstemplates, die für eine Situationsüberwachung verwendet werden können.

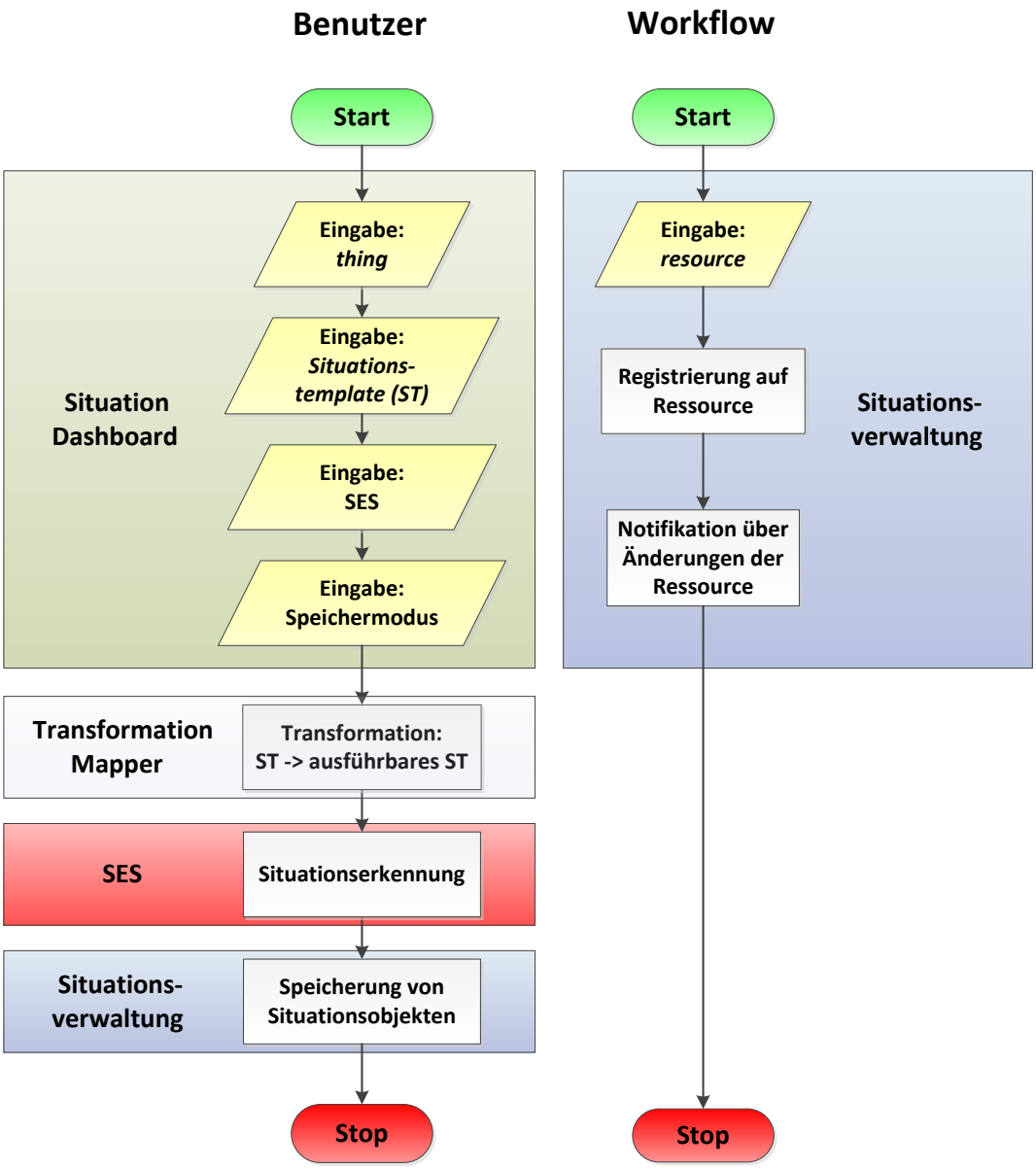

Abbildung 4.3.: Ablauf für das Erkennen und Erhalten von Situationen

Der Benutzer greift über das Situation Dashboard auf die Daten zu und kann über die Eingabe des zu überwachenden Things, des verwendeten Situationstemplates und SES sowie des Speichermodus die Situationserkennung starten. Zuvor transformiert der Transformation Mapper das Situationstemplate in ein ausführbares Situationstemplate auf Basis des verwendeten SES. Daraufhin erkennt das SES kontinuierlich Situationen und erstellt Situationsobjekte, die an die Situationsverwaltung gesendet

werden. Die Situationsverwaltung validiert das Situationsobjekt, überprüft Referenzen und fügt weitere Metadaten hinzu.

Parallel dazu führt der Situation Handler eine Anmeldung bei der Situationsverwaltung durch. Hierbei gibt der Situation Handler an, beim Auftreten welcher Situationen eine Benachrichtigung erfolgen soll. Die Situationsverwaltung wird über jede neu erkannte Situation informiert und überprüft sämtliche Anmeldungen. Wenn eine neu erkannte Situation Anmeldungen enthält, wird das Situationsobjekt an eine von dem Situation Handler spezifizierte URL gesendet.

4.7. Datenmodell

Die in Kapitel 4.3 erwähnte Schemavalidierung legt ein Datenmodell aller verwendeten Ressourcen zu Grunde. Abbildung 4.4 zeigt die verschiedenen Ressourcen, ihre Attribute sowie Relationen zueinander auf. Die Datenmodelle beschreiben, wie die Ressourcen in der Datenbank vorliegen. Diese unterscheiden sich von den Datenmodellen, die für die Eingabe der jeweiligen Ressourcen verwendet werden. Beispielsweise besitzt jede Situation das Attribut *Name*. Das SES ruft hierbei die Funktion der Situationsverwaltung zum Erstellen einer Situationsressource auf, gibt den Namen allerdings nicht an, da dieser dem SES nicht bekannt ist. Stattdessen fügt die Situationsverwaltung den Namen selbstständig hinzu, damit das im folgenden aufgezeigte Datenmodell einer Situation erfüllt ist.

Im Folgenden werden die Datenmodelle jeder Ressource ausführlich erklärt. Hierbei werden die einzelnen Attribute beschrieben sowie deren Ursprung. Jeder Ressource wird eine eindeutige ID (wahlweise vom Benutzer definiert) sowie eine Revisionsnummer zugewiesen. Die Revisionsnummer wird zur Konfliktbehandlung bei nebenläufigen Schreibzugriffen benötigt.

4.7.1. Situation

Das Datenmodell einer Situation muss alle Attribute aufweisen, die für die Weiterverarbeitung durch Workflows benötigt werden. Des Weiteren sollen Metadaten vorhanden sein, um schnell und intuitiv Situationen im Situation Dashboard erkennen zu können.

- **Name.** Der Name dient neben der ID zur intuitiven Identifikation der Situation. Dieser wird im Situationstemplate definiert. Verschiedene Situationstemplates können den selben Namen für Situationen definieren. Als Richtlinie sollte darauf geachtet werden, dass Namen nicht doppelt existieren.
- **Thing.** Das Attribut Thing referenziert die ID des überwachten Objekts. Bei der Instanziierung wird die ID des Objekts dem SES übergeben und bei Erkennung einer Situation dem Situationsobjekt hinzugefügt.
- **Timestamp.** Der Zeitstempel beschreibt den Zeitpunkt, an dem die Situation erkannt wurde. Der Zeitstempel wird durch das SES zum Zeitpunkt der Situationserkennung gesetzt. Dieser kann verwendet werden, um die Aktualität einer Situation zu beschreiben. Zusätzlich lässt sich der Situationsverlauf einzelner Situationen zeitlich darstellen.

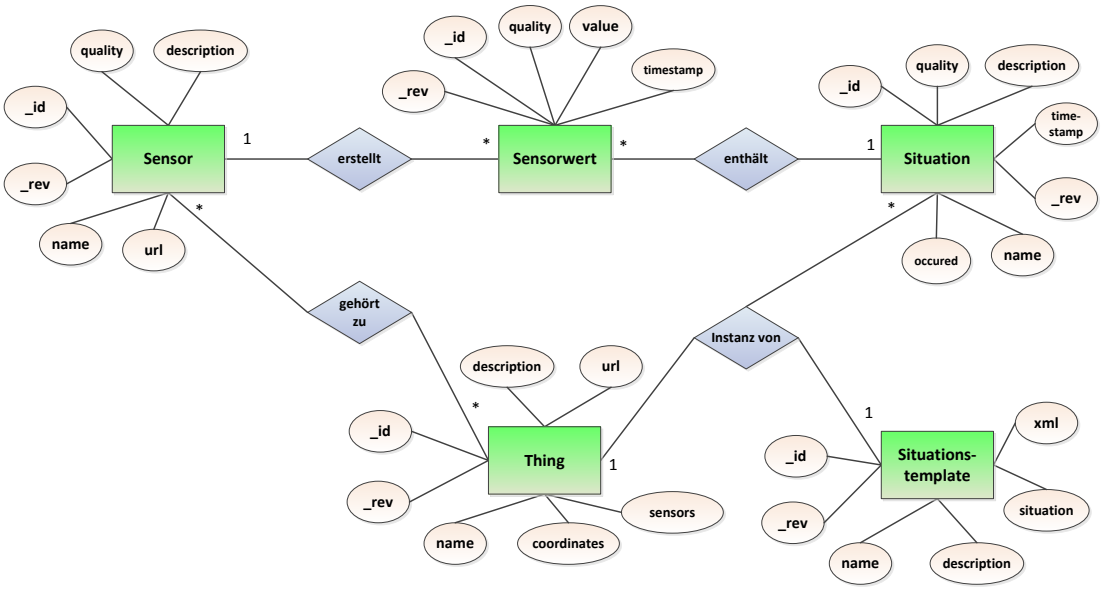

Abbildung 4.4.: ER-Diagramm der Ressourcen

- **Sensorwerte.** Die Sensorwerte, die bei der Erkennung der Situation verwendet wurden, werden dem Situationsobjekt innerhalb des SES hinzugefügt. Die Situationsverwaltung extrahiert die Sensorwerte und ersetzt sie durch Referenzen auf die jeweiligen Sensorwerte. Als Referenz wird die ID des separat gespeicherten Sensorwerts verwendet.
- **Beschreibung.** Um Situationen verständlicher darzustellen, wird eine kurze Beschreibung über die Bedeutung der Situation hinzugefügt. Diese wird im Situation Dashboard angezeigt. Diese Beschreibung wird im Situationstemplate definiert und durch die Situationsverwaltung anhand der referenzierten ID des Situationstemplates ausgelesen und dem Situationsobjekt hinzugefügt.
- **Situationstemplate.** Beim Aufruf einer Situationserkennung erhält das SES die ID des Situationstemplates. Beim Erstellen des Situationsobjekts wird diese ID als Referenz hinzugefügt.
- **Occured.** Die Gültigkeit einer Situation wird über das Attribut *Occured* als Wahrheitswert angegeben. Ob eine Situation gültig ist oder nicht, wird im SES erfasst.
- **Qualität.** In Abhängigkeit der verwendeten Sensoren und Sensorwerten soll die Qualität einer Situation bestimmt werden. Das SES besitzt die verwendeten Sensorwerte, jedoch keine näheren Informationen über die verwendeten Sensoren. Daher wird die Qualitätsbestimmung einer Situation in der Situationsverwaltung durchgeführt.

4.7.2. Situationstemplate

Ein Situationstemplate liegt als XML-Datei vor, in welcher alle Konditionen und Operationen definiert sind, die für das Erkennen einer Situation benötigt werden. Zusätzlich enthält es wichtige Metadaten einer Situation. Zur besseren Übersichtlichkeit wird der Inhalt der XML-Datei nicht als Attribut gespeichert, sondern als Anhang dem jeweiligen Dokument beigefügt.

- **Name.** Der Name dient neben der ID zur intuitiven Identifikation des Situationstemplates. Verschiedene Situationstemplates können den selben Namen besitzen. Als Richtlinie sollte darauf geachtet werden, dass Namen nicht doppelt existieren.
- **Situation.** Dieses Attribut enthält den Namen der Situation, die durch das Situationtemplate erzeugt werden kann. Es sollte Wert auf eine ausdrucksstarke und intuitive Namensgebung gelegt werden.
- **Beschreibung.** Hier wird eine kurze Beschreibung der Situation angegeben, welche im Situation Dashboard angezeigt wird.

4.7.3. Sensor

Sensoren besitzen eine Großzahl an Spezifikationen. Die hier aufgezählten Attribute schränken diese Spezifikationen deutlich ein, um ein standardisiertes Sensormodell zu erzeugen und verschiedenartige Sensortypen zu unterstützen.

- **Name.** Der Name dient neben der ID zur intuitiven Identifikation des Sensors. Verschiedene Sensoren können den selben Namen besitzen. Als Richtlinie sollte darauf geachtet werden, dass Namen nicht doppelt existieren.
- **URL.** Unter dieser URL sind die vom Sensor bereitgestellten Sensorwerte zu finden.
- **Beschreibung.** Hier wird eine kurze Beschreibung des Sensors und möglichen Sensorwerten geliefert.
- **Qualität.** Um die Qualität einer Situation zu bestimmen, werden die Qualitäten der verwendeten Sensoren benötigt. Die Qualität lässt sich zwischen 0 (sehr unzuverlässig) und 1 (keine Fehler) einordnen.
- **Datentyp** beschreibt den Datentyp der Sensorwerte, die durch den Sensor entstehen.

4.7.4. Sensorwert

Sensorwerte werden ausschließlich dann gespeichert, wenn das Situationsobjekt, denen die Sensorwerte zugrunde liegen, gespeichert wird. Analog zum Speichern von Situationsobjekten ist es denkbar, alle Sensorwerte zu speichern, um weitere Informationen über das überwachte Objekt zu erhalten, was erneut zu einem Anstieg des gespeicherten Datenvolumens führt.

- **Sensor.** Das Attribut *Sensor* enthält eine Referenz auf den Sensor, der diesen Sensorwert produziert hat. Dem SES ist die ID des Sensors bekannt.
- **Wert.** Hier wird der eigentliche Sensorwert angezeigt. Um verschiedenartige Sensortypen zu unterstützen, dürfen keine Einschränkungen an den Datentyp gesetzt werden. Der Wert wird durch das SES erfasst.
- **Timestamp.** Durch einen Zeitstempel wird definiert, wann ein Sensorwert entstanden ist. Der Zeitstempel muss bereits vor der Situationserkennung durch das SES gesetzt werden.
- **Qualität.** Neben der Qualität des Sensors kann auch jeder Sensorwert einen zusätzlich definierten Qualitätswert besitzen. Dieser liegt zwischen 0 und 1.

4.7.5. Things

Things stellen die virtuelle Repräsentation eines physischen Objekts dar, um diese anschließend überwachen zu können. Dabei gibt es keinerlei Einschränkungen bezüglich des Objekts, solange es Sensoren besitzt.

- **Name.** Der Name dient neben der ID zur intuitiven Identifikation des Sensors. Verschiedene Sensoren können den selben Namen besitzen. Als Richtlinie sollte darauf geachtet werden, dass Namen nicht doppelt existieren.
- **URL.** Dieses Attribut beschreibt die URL, unter welcher das Objekt verfügbar ist.
- **Koordinaten.** Um im Situation Dashboard eine Karte aller überwachten Objekte anzeigen zu können, muss jedes Objekt Koordinaten besitzen, die den Standort spezifizieren.
- **Sensor.** Ein Objekt kann mehrere Sensoren besitzen. Diese werden innerhalb eines Arrays über die jeweilige ID referenziert.

5. Implementierung des Prototyps

In diesem Kapitel wird eine prototypische Implementierung der im vorangegangenen Kapitel vorgestellten Konzepte entwickelt. Zunächst wird die von SitOPT verwendete Datenbank vorgestellt, im Anschluss die darauf aufbauende Situationsverwaltung. Es werden Details der implementierten Funktionen dargestellt sowie das Framework *Swagger*, welches zum Dokumentieren und Implementieren der Situationsverwaltung verwendet wurde. Zusätzlich wird in Kapitel 5.6 das Setup dieses Prototyps ausführlich beschrieben.

5.1. Datenbank

SitOPT verwendet die dokumentenorientierte Datenbank CouchDB [CDbb]. Zu beachten ist, dass im Rahmen des Projekts auch andere Arten von Datenhaltungssystemen zum Einsatz kommen. Beispielsweise werden die Sensoren in ihrer nativen Form als Ontologien gespeichert, die Daten auf der Workflowebene innerhalb eines SQL-Datenbanksystems.

CouchDB (Couch für "cluster of unreliable commodity hardware") ist seit 2005 unter der Apache-Lizenz verfügbar und eine in Erlang von Damien Katz entwickelte dokumentenorientierte NoSQL-Datenbank [SSK11]. In CouchDB werden die Daten in JSON-Dokumenten gespeichert. Die Dokumente können in verschiede *Databases* untergliedert werden. Dabei werden die Daten innerhalb des Dokuments in Schlüssel-Wert-Paaren geordnet. Zusätzlich wird der Schlüssel _id für eine eindeutige Identifizierung hinzugefügt. Das Feld _rev enthält die Revisionsnummer, welche zur Versionsverwaltung der Dokumente verwendet wird. Um bei gleichzeitigem Schreibzugriff auf ein Dokument Konflikte zu vermeiden, muss die Revisionsnummer bei einem Schreibvorgang mitangegeben werden. CouchDB verwendet *Optimistic Locking*, d.h. der Nutzer, der zuerst einen Schreibvorgang beendet, ist privilegiert, dem zweiten Benutzer wird ein Konflikt gemeldet [ALS10, pp. 39-40]. Dokumente können Referenzen zu anderen Dokumenten aufweisen, allerdings wird die referenzielle Integrität nicht überprüft, was dazu führen kann, dass ein referenziertes Dokument bereits gelöscht oder verschoben wurde [SSK11].

5.1.1. Views

Einzelne Dokumente können direkt per Angabe der ID angefordert werden. Werden andere Attribute des Dokuments für die Suche verwendet, müssen Views verwendet werden. Views werden mittels Map/Reduce erzeugt. Die Map-Funktion iteriert über alle Dokumente und gibt die Ergebnisse zurück, die durch eine Funktion gefiltert worden sind. Das Ergebnis wird als Array von Key-Value-Paaren zurückgegeben. Die Reduce-Funktion ist optional und dient dem Zusammenfassen von Werten, die bei

der Map-Funktion entstanden sind. Wird eine View das erste Mal mittels der Map-Funktion erstellt, wird ein B-Baum erstellt, der diese View enthält. Jeder weitere Zugriff auf Daten, die in dieser View enthalten sind, erfolgen deutlich schneller, da direkt auf den B-Baum zugegriffen wird [ALS10, pp. 53-55].

```
function(doc){
      emit([doc.thing, doc.situationtemplate], doc._id);
}
```

Listing 5.1: CouchDB Views mit 2 Attributen

Dieser View geht über alle Dokumente und enthält als *key* ein Array mit den Inhalten *thing* und *situationtemplate* und als *value* die ID des Dokuments. Dieser View wird verwendet, um zu überprüfen, ob eine vom SES gelieferte Situation bereits vorhanden ist oder nicht. Eine Situation lässt sich eindeutig über die ID identifizieren. Das SES besitzt diese ID nicht. Stattdessen lässt sich eine Situation auch eindeutig über die Kombination von Objekt und Situationstemplate identifizieren, welche dem SES bekannt sind. Der obere View ermöglicht die Suche nach dieser Kombination und gibt, wenn vorhanden, die ID des Dokuments als Value zurück. Mit dieser ID wird die neue Situation in der Datenbank gespeichert - es wird ein Update durchgeführt. Ansonsten wird ein neues Dokument mit einer neuen ID für die Situation angelegt. Ein Aufruf dieses Views in Node.js sieht wie folgt aus:

```
var database = conn.database('situations');
database.view('situations/existing,
      {key: [req.body.thing, req.body.situationtemplate]},
      function(err,doc)
            // doc == Array mit dem gesuchten Dokument
      ...
```

Listing 5.2: Aufruf in *Node.js* mit *cradle*

Views existieren für bestimmte Databases. Wie zu sehen ist, wird zuerst eine Verbindung zu der Database *situations* aufgebaut und anschließend der View *existing* abgefragt. Views werden in Designdokumenten, im Beispiel mit Namen *situations*, untergeordnet, um die Verwaltung von Views zu vereinfachen (ähnlich einer Ordnerstruktur). Als *key* wird das referenzierte Thing und Situationtemplate übergeben. Im Anhang werden alle von der Situationsverwaltung verwendeten CouchDB Views aufgelistet.

5.1.2. Changes Feed

CouchDBs API besitzt eine Ressource namens *_changes*. Angewendet auf eine Listenressource liefert *_changes* jede bis dahin erfolgte Veränderung an den Benutzer. Das wiederholte Anfragen nach Veränderungen - genannt Polling - ist unerwünscht. Bei jeder Abfrage würde eine neue Verbindung aufgebaut werden müssen, auch wenn es zu keiner Veränderung kam. Die Einstellung *feed=continuous* umgeht dieses Problem, indem die Verbindung so lange bestehen bleibt, bis sie entweder explizit geschlossen wird oder nachdem nach einem benutzerdefiniertem Intervall keine Veränderungen eingetreten sind. Solange die Verbindung offen ist, wird jede Veränderung an den Benutzer gesendet. Wird das Attribut *include_docs* gesetzt, wird das Dokument, das die Veränderung herbeigeführt hat, mitgesendet [CDba].

5.1.3. Replikation

CouchDB verwendet eine Peer-to-Peer-Architektur. Der Benutzer kann selbst definieren, zwischen welchen Knoten Replikationen stattfinden sollen. Da dies für jeden Knoten im System erfolgen muss, entsteht bei hoher Anzahl von Knoten ein erheblicher Aufwand, weshalb man in der Praxis bei größerer Anzahl von Knoten schließlich auf eine Master-Slave-Replikation zurückgreift [Kra].

Um Daten zu replizieren, müssen die Quelldatenbank und die Zieldatenbank bestimmt werden. Die Replikation kann unidirektional oder bidirektional erfolgen. CouchDB überprüft daraufhin die Unterschiede der Datenbanken und repliziert die Dokumente der Quelldatenbank, die nicht in der Zieldatenbank enthalten sind. Sind in beiden Datenbanken jeweils Dokumente mit der selben _id, so lässt sich mittels der Versionsnummer _rev herausfinden, welches das aktuellere Dokument ist. Die Replikation ist inkrementell. Sollte also durch einen Netzwerkfehler die Replikation unterbrochen werden, kann CouchDB genau an der Stelle weitermachen, wo es unterbrochen wurde. Mittels der Eigenschaft continuous überprüft CouchDB stets die Unterschiede der Datenbanken und repliziert bei auftretenden Änderungen automatisch [CDbd].

5.2. Situationsverwaltung

CouchDB besitzt bereits eine REST-konforme HTTP Schnittstelle [CDbc]. Im Rahmen dieser Arbeit fehlen der Schnittstelle jedoch essentielle Funktionalitäten wie das Überprüfen referenzieller Integrität und Schemavalidierung, was dazu führte, dass auf Basis der CouchDB API die Situationsverwaltung implementiert wurde. Alle an die Datenbank gerichteten Anfragen gehen zuerst an die Situationsverwaltung, welche anschließend an CouchDB weitergeleitet werden. Da großer Wert auf die Weiterentwicklung dieser Arbeit gesetzt wird, wurde Swagger verwendet - ein Framework für Schnittstellen, welches eine einfache Möglichkeit darstellt, die Funktionalitäten einer Schnittstelle visuell darzustellen und zu dokumentieren. Zusätzlich lassen sich alle Funktionen ohne Implementierung eines Clients testen, was für die Entwicklung eines Prototyps von Vorteil ist. Der Swagger Editor dient der Erstellung einer von Swagger lesbaren Schnittstellenspezifikation. Standardmäßig wird YAML (wahlweise JSON) als Auszeichungssprache verwendet. An folgendem Beispiel soll die Vorgehensweise zur Erstellung einer mittels Swagger spezifizierten Funktion erläutert werden:

```
...
/situations/byID:
  x-swagger-router-controller: situations
  get:
    tags:
      - situation
    summary: Get situation by ID
    description: Get the specified situation. Returns a document.
    operationId: situationByID

    parameters:
      - name: ID
        in: query
        description: ID of situation
        required: true
```

```
      type: string

  responses:
    "200":
      description: Situation found
      schema:
        $ref: "#/definitions/Situation
    "404":
      description: Not found
      schema:
        $ref: "#/definition/ErrorResponse
...
```

Listing 5.3: Swagger Spezifikation

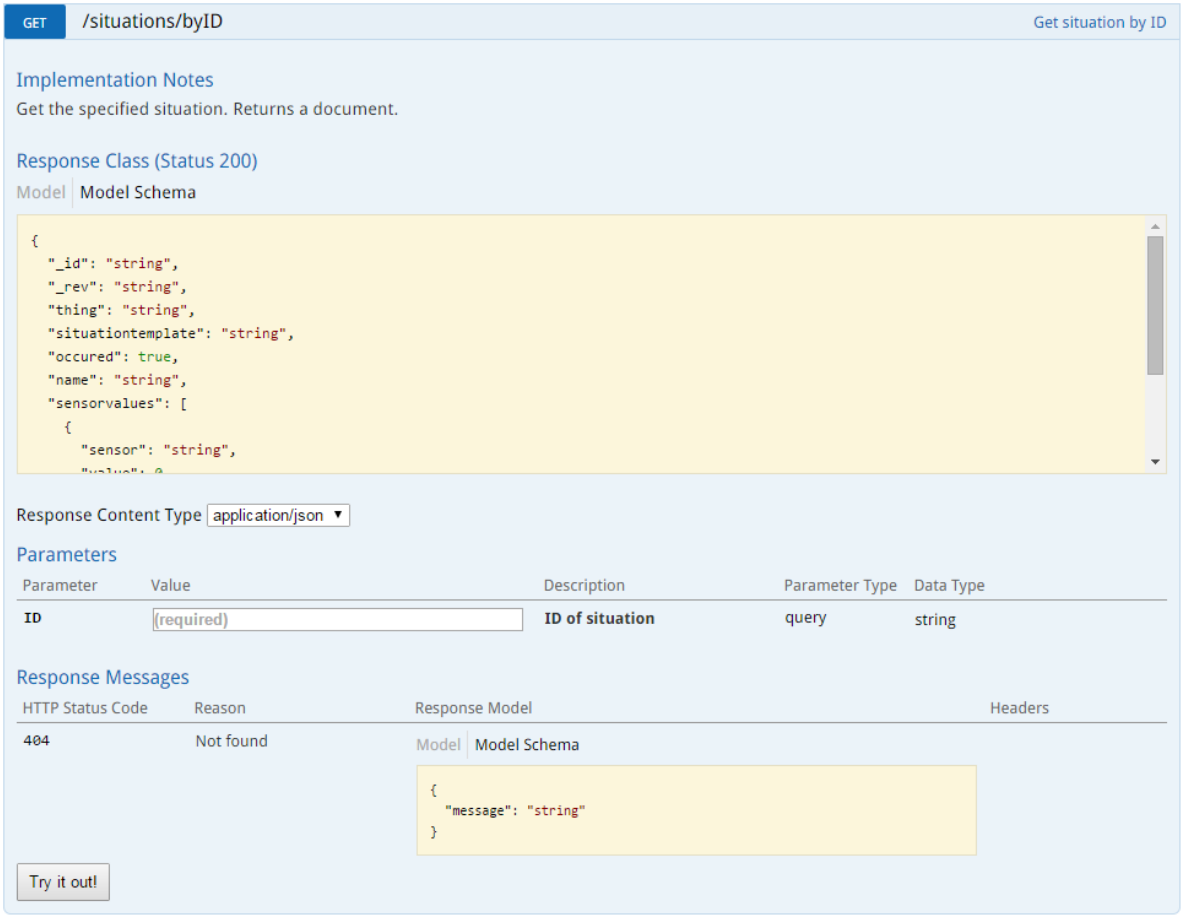

Abbildung 5.1.: Swagger Spezifikation - *GET /situations/byID* (1)

Das Modul SwaggerUI erstellt auf Basis dieser Spezifikation die in Abbildung 5.1 zu sehende Darstellung. Der Codeausschnitt erstellt den HTTP-Request *GET /situations/byID*. *x-swagger-router-controller* definiert, in welchem Modul die unter *operationID* spezifizierte Funktion *situationByID* zu finden ist, um den HTTP GET Request auszuführen. *summary* wird für eine kurze Beschreibung (in Abbildung 5.1 oben rechts) verwendet, *description* für eine ausführlichere Beschreibung und weitere Details

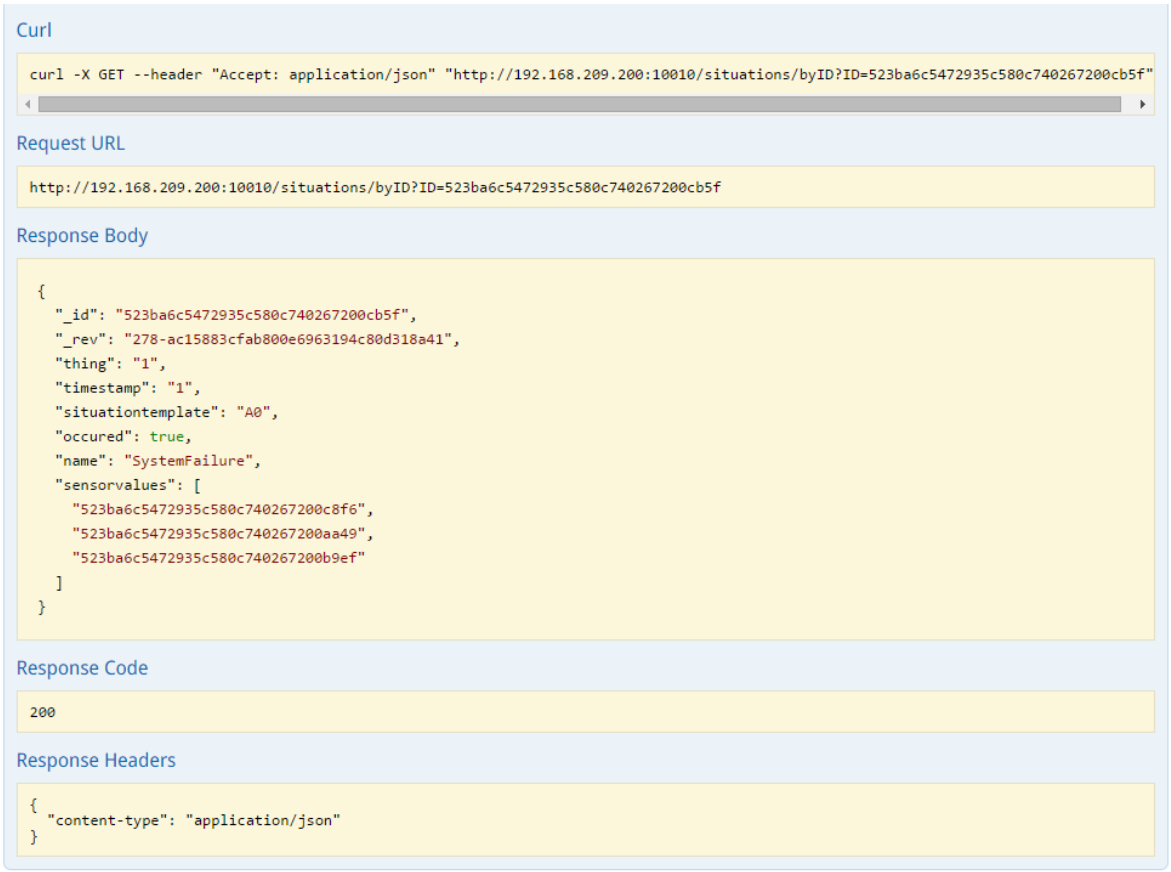

Abbildung 5.2.: Swagger Spezifikation - *GET /situations/byID*(2)

(beides optional). *parameters* definiert die Benutzereingabe und zugehörige Eigenschaften, in diesem Fall die ID, die zur Identifikation der Situation verwendet werden soll. Unter *responses* werden alle möglichen Statuscodes angegeben, die von der Situationsverwaltung zurückgegeben werden können. Das zugehörige Attribut *schema* definiert das Modell der Antwort. Hierfür werden Referenzen auf (auch in der YAML Spezifikation) vordefinierte Modelle gesetzt. Unter *#/definition/Situation* ist das in Kapitel 4.7.1 beschriebene Situationsmodell definiert. Unter Angabe einer ID und dem Klick auf den Button *Try it out!* wird die in Abbildung A.5 zu sehende Ansicht ausgefahren. Es wird ein äquivalenter HTTP GET Request mit dem Kommandozeilentool cURL angezeigt sowie die Request URL. Der *Response Body* enthält die Antwort auf den Request und zeigt die gesuchte Situation an. Zusätzlich werden die Metadaten der Antwort - der *Response Code 200* und der *Content-Type* (das Datenformat der Antwort) *application/json*. Der gesendete und erhaltene Content-Type wurde zu Beginn der YAML-Datei (nicht im Listing enthalten) spezifiziert und gilt somit für alle Funktionen. Es ist möglich, einzelnen Funktionen andere Content-Types zuzuweisen beziehungsweise mehrere Content-Types zuzulassen.

5.2.1. Funktionen

Im Folgenden werden Details zu den von der Situationsverwaltung bereitgestellten Funktionen vorgestellt. Die Funktionen jeder Ressource, dargestellt in Swagger, sind im Anhang dieser Arbeit zu finden.

- **GET.** Jede Ressource enthält verschiedene HTTP GET Request Methoden. Wird die ID als Suchparameter angegebenen, wird das jeweilige Dokument zurückgegeben. Wird der Name verwendet, wird ein Array mit allen Dokumenten zurückgegeben, die den Namen enthält, mit Ausnahme der Ressource *sensorvalues*, die keinen Namen enthält. Ein weiterer HTTP GET Request ermöglicht, die gesamte Listenressource zu erhalten. Die Ressource *situation* enthält zudem einen HTTP GET Request, dessen Suchparameter *thing* und *situationtemplate* sind (Kombination identifiziert Situation eindeutig). Des Weiteren kann der Benutzer mittels *GET situations/changes* alle derzeitigen Anmeldungen des Situation Handlers erhalten. Bei der Ressource *situationtemplates* ist ein HTTP GET Request vorgesehen, der den XML-Inhalt zurückgibt (nicht implementiert).
- **POST.** Jede Ressource enthält eine HTTP POST Request Methode, mit der es möglich ist, eine neue Ressource zu erstellen. Mittels *POST /situations/changes* können Anmeldungen durchgeführt werden. Mit *POST /situationtemplates/{id}/{templatename}* kann der Benutzer die XML-Datei des Situationstemplates als Anhang an das jeweilige Dokument hinzufügen.
- **DELETE.** Ein Löschvorgang einer Ressource erfolgt über deren ID. Hierbei findet kein kaskadierender Löschvorgang statt. Über *DELETE /situations/changes* können Anmeldungen gelöscht werden.
- **PUT.** Im Prototyp ist es derzeit nicht möglich, Ressource mit einem HTTP PUT Request zu verändern. Die einzige HTTP PUT Request Methode ist *PUT /situations/occured*, mittels derer das *occured*-Attribut einer Situation zu Testzwecken verändert werden kann, ohne eine Situationserkennung starten zu müssen.

5.2.2. Changes Feed

Der in Kapitel 5.1.2 vorgestellte Changes Feed von CouchDB stellt die Grundlage für den Changes Feed der Situationsverwaltung dar. Folgende Herausforderungen an den Changes Feed der Situationsverwaltung sind zu erfüllen. Die Anfrage an die CouchDB *changes*-Ressource liefert alle Änderungen aller Situationen, was selten erwünscht ist. Workflows sind nur an Situationen interessiert, deren Gültigkeit sich ändert (*occured*-Attribut wechselt von *true* auf *false* beziehungsweise umgekehrt). Zusätzlich benötigt nicht jeder Workflow alle Situationsänderungen. Demnach muss es möglich sein, sich nur auf bestimmte Situationen registrieren zu können und nur benachrichtigt zu werden, wenn sich die Gültigkeit dieser Situation verändert. Ein weiterer Nachteil ist, dass CouchDB die Veränderungen direkt an den Benutzer zurücksendet. Da es sinnvoller ist, die Situationen direkt an die angemeldeten Workflows zu senden, muss bei der Anmeldung die Möglichkeit bestehen, eine Callback URL anzugeben. Ändert sich die Gültigkeit einer Situationen, wird überprüft, ob Anmeldungen für diese Situation bestehen. Falls ja, wird die Situation an alle angegebenen Callback URLs gesendet. Wie in Abbildung 5.3 zu sehen ist, fragt die Situationsverwaltung bei CouchDB die *changes*-Ressource an. Benutzer und

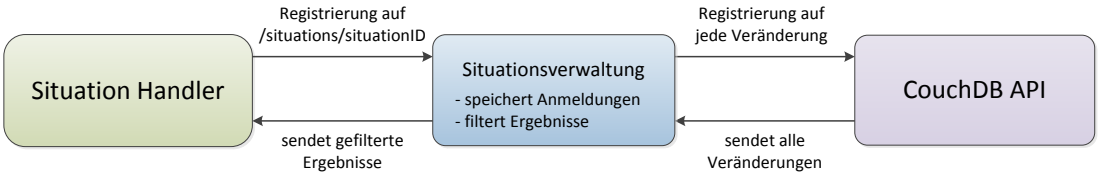

Abbildung 5.3.: API Kommunikation

Workflows registrieren sich bei der Situationsverwaltung. Jede Veränderung von CouchDB wird an die Situationsverwaltung gesendet, welche die Dokumente filtert und an die entsprechenden Callback URLs sendet.

Die von der Situationsverwaltung bereitgestellten Operationen sind:

- **POST situations/changes.** Die Anmeldung erfolgt über einen HTTP POST Request. Da die ID der Situation dem Benutzer im Normalfall nicht bekannt ist, werden als Parameter die ID des Situationstemplates und die ID des things angegeben, um sich auf eine bestimmte Situation anzumelden. Werden diese Parameter nicht angegeben, erfolgt die Anmeldung auf alle neu erkannten Situationen. Die Angabe einer Callback URL ist obligatorisch. Der Boolean-Parameter *once* gibt an, ob Situationsänderungen dauerhaft an die Callback URL gesendet werden sollen oder nur einmalig. Beim Setzen des Parameters auf *true* wird die nächste auftretende Situationsänderung an die Callback URL gesendet und daraufhin die Anmeldung gelöscht. Meldet sich ein Benutzer auf alle Situationsänderungen an, werden alle vorherigen Anmeldungen dieses Benutzers gelöscht und durch die neue Anmeldung ersetzt. Mögliche Fehlermeldungen sind *404*, falls eine vom Benutzer spezifizierte Situation nicht gefunden werden konnte und *400*, falls nur einer der beiden Parameter *ThingID* und *SituationtemplateID* angegeben wurde. Sendet CouchDB Veränderungen an die Situationsverwaltung, wird ermittelt, ob diese Situation Anmeldungen enthält. Falls ja, wird diese an die hinterlegte Callback URL mittels eines HTTP POST Requests gesendet.
- **GET situations/changes.** Diese Funktion gibt alle bestehenden Anmeldungen zurück. Der optionale Parameter *Callback URL* wird angegeben, wenn nur Anmeldungen, welche die spezifizierte Callback URL verwenden, zurückgegeben werden sollen. Mögliche Fehlermeldung ist *404*, wenn unter der angegebenen Callback URL keine Anmeldungen gefunden werden konnten.
- **DELETE situations/changes.** Um nicht weiterhin über Situationsänderungen benachrichtigt zu werden, können mit dieser Funktion Anmeldungen gelöscht werden. Soll eine bestimmte Anmeldung gelöscht werden, erfolgt dies unter der Angabe der Parameter *ThingID*, *SituationtemplateID* sowie *CallbackURL*. Wird nur die Callback URL angegeben, werden alle Anmeldungen, die diese Callback URL enthalten, gelöscht. Die Fehlermeldungen sind analog zur Funktion *POST situations/changes*.

5.2.3. Situationsobjekt

Um zu überprüfen, ob alle benötigten Attribute eines Dokuments bei einem POST-Request vorhanden sind, führt Swagger eine Schemavalidierung anhand des definierten Situationsobjekts durch. Ein Situationsobjekt, das von einem SES an die Situationsverwaltung geschickt wird, sieht wie folgt aus:

```
{
  "thing": "ThingID",
  "situationtemplate": "SitTempID",
  "timestamp": "string",
  "occured": true,
  "sensorvalues": [
    {
      "sensor": "SensorID",
      "value": 0,
      "timestamp": "string",
      "quality": 0
    }
  ]
}
```

Listing 5.4: Situationsobjekt

Zu beachten ist, dass dieses Situationsobjekt nicht mit dem Datenmodell aus Kapitel 4.7.1 übereinstimmt. Metadaten wie z.B. der Name der Situation werden über das referenzierte Situationstemplate ausgelesen und dem Situationsobjekt in der Situationsverwaltung hinzugefügt. Die Attribute *thing* und *situationtemplate* referenzieren die IDs des überwachten things sowie das verwendete Situationstemplate. Diese sind zum Zeitpunkt der Instanziierung bekannt und müssen dem Situationsobjekt hinzugefügt werden, um eine Situation eindeutig zu definieren. Die ID einer Situation ist dem SES nicht bekannt, zumal neu erkannte Situationen noch keine ID besitzen. Mittels der Funktion *checkID(documentID, databaseID, callback)* wird zuerst überprüft, ob die ID des things vorhanden ist, um fehlerhafte Referenzen zu verhindern. Da nur überprüft werden muss, ob diese ID vorhanden ist, ist es von Vorteil, nur die Metadaten des Dokuments mittels der *documentID* abzufragen, um die Geschwindigkeit zu erhöhen und die Bandbreite weniger zu belasten. Um den Namen der Situation zu bestimmen, wird das referenzierte Situationstemplate mittels einer GET Abfrage unter Angabe der ID gelesen. Werden beide IDs gefunden, wird überprüft, ob sich die Gültigkeit der Situation (*occured*-Attribut) geändert hat. Falls ja, werden zuerst die Sensorwerte der Situation *sensorvalues* mit der Funktion *SaveSV* eingefügt, um diese anschließend im Situationsobjekt referenzieren zu können. Anhand des *sensor*-Attributs eines Sensorwerts wird erkannt, ob bereits Sensorwerte dieses Sensors in der Datenbank vorhanden sind. Falls ja, wird ein Update dieses Sensorwerts durchgeführt. Ansonsten wird ein neues Dokument für diesen Sensorwert erstellt und eine neue ID zugewiesen. Die IDs der Sensorwerte sind nicht bekannt, weshalb der View *sensorvalues/existing* verwendet wird, welcher als key die Sensor-ID *sensor* enthält und das als *value* das Dokument. Der Callback der Funktion *SaveSV* enthält alle IDs der Sensorwerte der einzufügenden Situation, so dass diese referenziert werden können. Anschließend wird überprüft, ob Situationen dieses Typs (selbe ID) bereits bestehen. Wie bereits beschrieben, lässt sich eine Situation neben der ID über die Kombination der IDs des überwachten things und des verwendeten Situationstemplates identifizieren. Der View *situations/existing* enthält diese Kombination als key und als value das Dokument. Wird eine bestehende Situation identifiziert, wird ein Update ausgeführt, anderenfalls ein neues Dokument erstellt. Mögliche HTTP Statuscodes

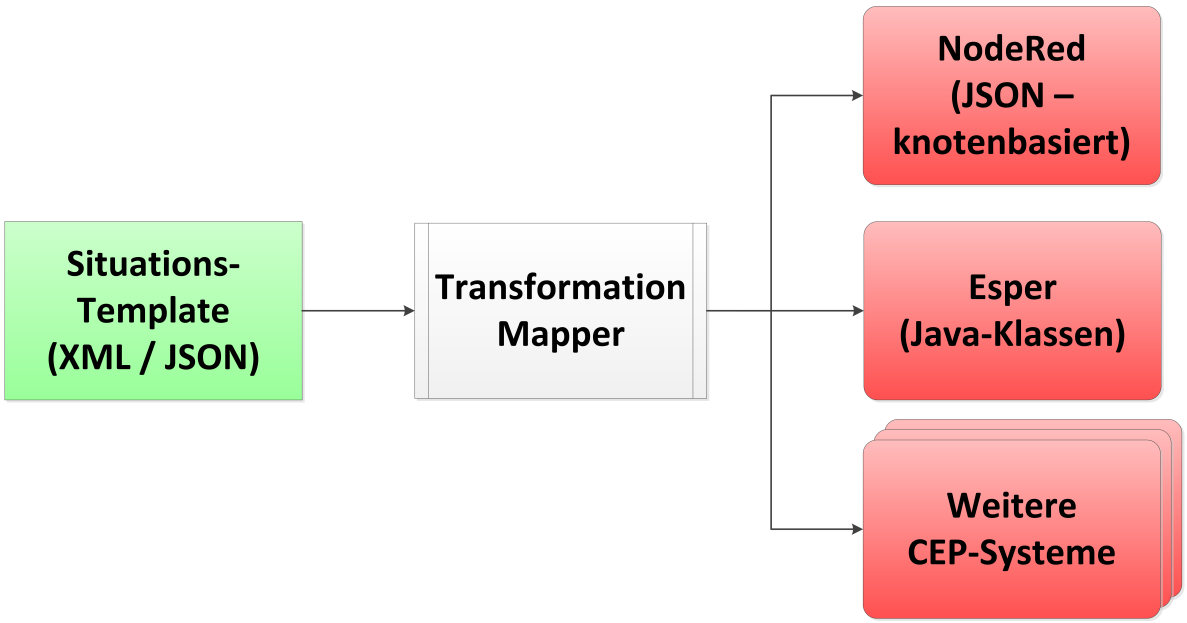

Abbildung 5.4.: Transformation Mapper

sind *Reference Errors (404)*, wenn die IDs des things und des Situationstemplates nicht gefunden werden. Kann keine Verbindung zu CouchDB aufgebaut werden, entsteht der Fehlercode *504*. Wird eine neue Situation erstellt, wird *201*, bei Updaten einer Situation der Statuscode *200* zurückgesendet.

5.3. Transformation Mapper

Der Transformation Mapper verwaltet alle Transformationen von Situationstemplates. Da im Rahmen dieser Arbeit nur Node-RED als SES zur Verfügung steht, wird das Situationstemplate in ein Node-RED-spezifisches Format transformiert. Dazu wird der in Java geschriebene Transformation Mapper als .jar-Datei im Situation Dashboard aufgerufen, wenn eine Situationserkennung gestartet wird. Der beigefügte XML Inhalt des Situationstemplates wird ausgelesen und als Argument beim Aufruf des Transformation Mappers verwendet. Zusätzlich wird die URL des Sensors mitgegeben. In Zukunft soll die URL des Sensors angegeben werden, die über das Attribut *sensor* des zu überwachenden Things und anschließend über das Attribut *url* des Sensors referenziert ist. Das Boolean-Argument *doOverwrite* gibt an, ob bereits laufende Situationserkennungen überschrieben werden sollen. Wie in Abbildung 5.3 zu sehen ist, soll der Transformation Mapper in Zukunft eine Vielzahl unterschiedlicher SES unterstützen.

5.4. Situationserkennung

Node-RED [nr] stellt auf Basis von Node.js ein visuelles Tool bereit, um das *Internet Of Things* darzustellen. Node-RED bietet die Möglichkeit, verschiedene *things* (z.B. Hardwaregeräte, Schnittstellen

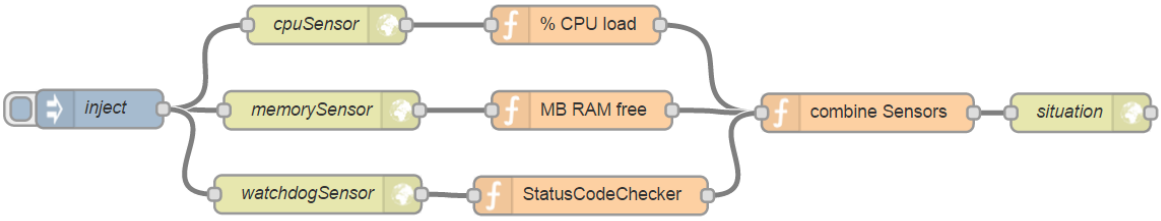

Abbildung 5.5.: Node-RED Flow

oder Onlinedienste) als Knoten zu verbinden und deren Interaktion zu ermöglichen (siehe Abbildung 5.5). Ein *Inject*-Knoten wird als Startpunkt gesetzt und definiert die Intervallgröße der Situationserkennung. Zusätzlich ist spezifizierbar, dass die Situationserkennung nur zu bestimmten Zeitpunkten oder Zeiträumen durchgeführt werden soll (z.B. müssen nachts stillgelegte Maschinen nicht überwacht werden). Die Sensorwerte werden über eine URL bereitgestellt, welche bei der Instanziierung der Überwachung angegeben wird. Mittels HTTP Request Knoten werden anschließend die Sensorwerte von RAM und CPU ausgelesen. Zusätzlich wurde ein *Watchdog* eingesetzt, der überprüft, ob die überwachte Maschine verfügbar ist. Funktionsknoten können für die Erstellung von in JavaScript geschriebenen Funktionen verwendet werden, um aus den Sensorwerten deduzierten Kontext zu erstellen. Zusätzlich werden Metadaten wie der Zeitstempel der Situationserkennung erstellt. Der Prototyp verwendet die Funktionsknoten hauptsächlich zur Generierung des Situationsobjekts. Der Funktionsknoten *combine Sensors* wird verwendet, um die im Situationstemplate definierten Konditionen an die Sensorwerte zu überprüfen und anschließend festzulegen, ob die Situation gültig oder ungültig ist. Ein weiterer HTTP Request Knoten wird benötigt, um einen HTTP POST Request an die Situationsverwaltung zu senden. Im *body* des Requests befindet sich das Situationsobjekt mit allen Sensorwerten. Grüne Knoten dienen ausschließlich Debug-Zwecken. Folgendes Listing 5.5 zeigt das zur Abbildung 5.5 zugehörige Situationstemplate im XML-Format.

```
 <?xml version="1.0" encoding="UTF-8"?>
<SituationTemplate id="A0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="situation_template_draft01.xsd"
      name="SystemObservation">
      <Situation id="A1" name="SystemFailure">
            <operationNode id="A3" name="combine Sensors">
                  <type>or</type>
                  <parent parentID="A10"/>
            </operationNode>
            <conditionNode id="A4" name="% CPU load">
                  <type>type</type>
                  <measureName>measureName</measureName>
                  <opType>greaterThan</opType>
                  <condValue>
                        <value>70</value>
                  </condValue>
                  <parent parentID="A3"/>
            </conditionNode>
            <conditionNode id="A8" name="MB RAM free">
```

```
                <type>type</type>
                <measureName>measureName</measureName>
                <opType>lowerThan</opType>
                <condValue>
                      <value>10</value>
                </condValue>
                <parent parentID="A3"/>
          </conditionNode>
          <conditionNode id="A9" name="StatusCodeChecker">
                <type>type</type>
                <measureName>measureName</measureName>
                <opType>notEquals</opType>
                <condValue>
                      <value>200</value>
                </condValue>
                <parent parentID="A3"></parent>
          </conditionNode>
          <contextNode id="A5" name="memorySensor">
                <parent parentID="A8"></parent>
          </contextNode>
          <contextNode id="A6" name="cpuSensor">
                <parent parentID="A4"></parent>
          </contextNode>
          <contextNode id="A7" name ="watchdogSensor">
                <parent parentID="A9"/>
          </contextNode>
          <situationNode name="machine_failed" id="A10"/>
     </Situation>
</SituationTemplate>
```

Listing 5.5: Situationstemplate in XML

Das Situationstemplate *SystemObservation* definiert die Situation *SystemFailure*. Alle Funktionsknoten von Node-RED sind in der XML-Datei spezifiziert. Inject-, Debug-, und HTTP Request-Knoten sind Node-RED-spezifisch und werden bei der Transformation erzeugt. Die Konditionen für die einzelnen Sensorwerte sind in den jeweiligen *conditionNodes* enthalten. Wenn der *CPU load* größer als 70(%) ist, weniger als 10 *MB RAM free* sind, und der *StatusCodeChecker* einen anderen HTTP-StatusCode als 200 (200 = *OK*) liefert, ist die Situation gültig, anderenfalls ungültig. Die Javascript-Funktionen sind ebenfalls Node-RED-spezifisch und werden im Transformation Mapper deklariert. Die *contextNodes* definieren die benötigten Sensorwerte.

Node-RED unterstützt zudem benutzerspezifizierte Knoten. Direktverbindungen mit Datenbanken (z.B. MongoDB) oder Messaging Services (MQTT) verringern den Entwicklungsaufwand erheblich. In Zukunft sollen weitere SES wie Esper oder Odyseus unterstützt werden. Dementsprechend muss der Transformation Mapper die zugehörigen Transformationen implementieren.

5.5. Situation Dashboard

Das Situation Dashboard stellt die Benutzerschnittstelle zu SitOPT dar. Für das Situation Dashboard wurde das asynchrone Event-basierte Framework Node.js verwendet. In Kombination mit dem darauf

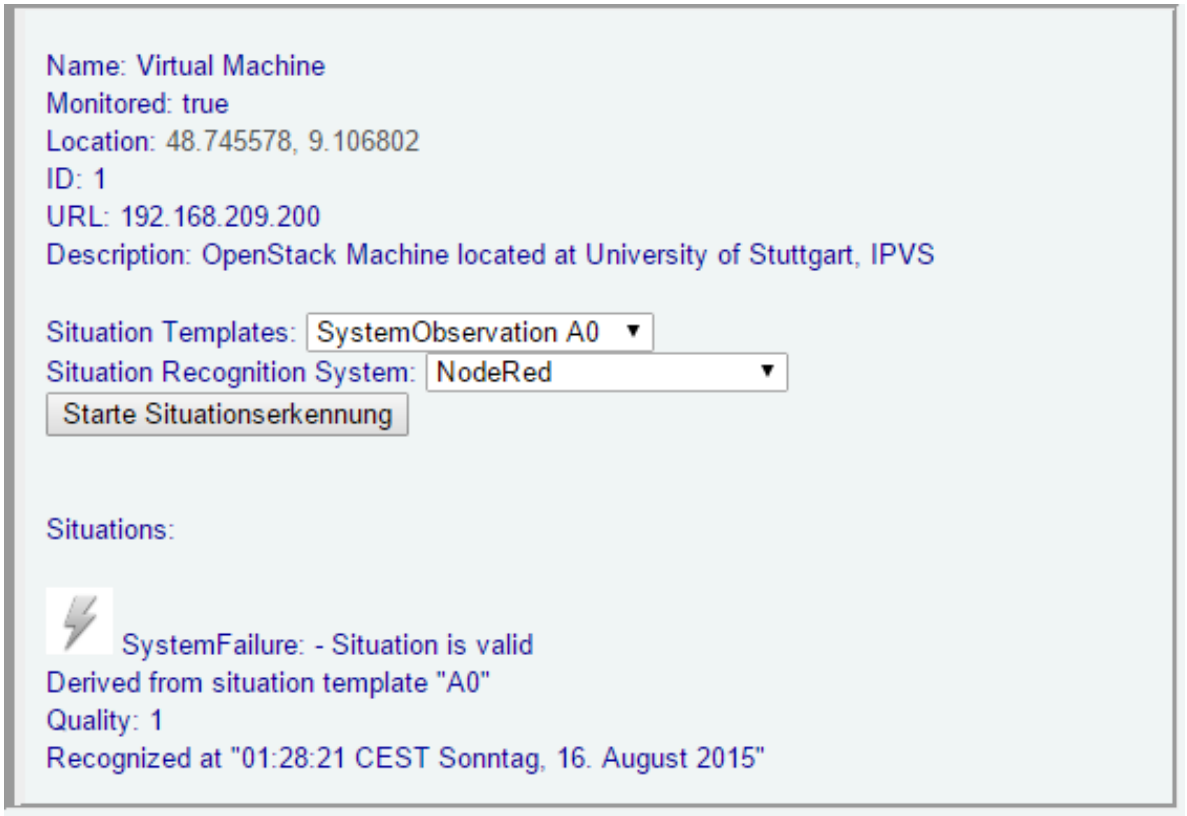

Abbildung 5.6.: Situation Dashboard - Things

aufbauenden Web-Framework Express.js lässt sich auf einfache Weise eine Website mit zahlreichen Middlewares und externen Bibliotheken aufbauen. Aufgabe des Situation Dashboards ist es, dem Benutzer eine visuelle Darstellung der Datenbank ohne direkten Datenbankzugriff zu ermöglichen. Abbildung 5.6 zeigt einen Ausschnitt der Seite Things. Zu sehen ist ein Thing und dessen Attribute. Ob das Thing derzeit überwacht wird, gibt das Attribut *monitored* an. Über einen Klick auf die Koordinaten des Things wird eine Google Maps Seite mit diesen Koordinaten geöffnet. Genauere Koordinaten und Umgebungen können ein Bild von Maschinen innerhalb einer Fabrik anzeigen, was die Übersicht und Verwaltung der Things erleichtert. Über zwei Dropdown Boxes kann das zum Überwachen verwendete Situationstemplate und SES ausgewählt werden. Wird der Button *Starte Situationserkennung* betätigt, startet die Situationserkennung und eine neue Seite mit einer Instanz von Node-RED wird geöffnet. Voraussetzung dafür ist, dass Node-RED gestartet wurde. Zusätzlich wird eine Liste aller Situationen des Things angezeigt. Ein Icon stellt die Situation bildlich dar.

Um entscheiden zu können, welches Situationstemplate verwendet werden soll, werden alle Situationstemplates auf der Seite *Situationstemplates* aufgelistet. Eine kurze Beschreibung soll erklären, welche Situationen aus dem entsprechenden Situationstemplate deduziert werden können. Unter *API Reference* wird die bereits besprochene Swagger-Dokumentation detailliert angezeigt. Zusätzlich lassen sich auf dieser Seite alle Funktionen testen. Der Reiter *NodeRed* öffnet eine Instanz von Node-RED

und zeigt alle laufenden Situationsüberwachungen an. Beim Starten einer Situationserkennung wird Node-RED automatisch geöffnet.

5.6. Setup

Im Folgenden werden die notwendigen Schritte für das Setup von SitOPT beschrieben. Alle benötigten Module werden im Repository mormulms/SitOPT bereitgestellt. Das Setup wurde auf einer virtuellen Maschine von OpenStack mit dem 32-Bit Betriebssystem Windows 7.0 Professional mit Service Pack 1 getestet.

Die Basis aller Module ist das Framework Node.js [1]. Nach der Installation können mittels des Kommandozeilenaufrufs *node* mit Node.js erstellte Anwendungen gestartet werden. Sollte das Kommando *node* nicht erkannt werden, muss Node.js dem Systempfad hinzugefügt werden. Als Kommandozeilentool wird die Windowskonsole *cmd* verwendet.

5.6.1. CouchDB

Die Installation der Datenbank CouchDB [2] erfolgt über den Windows (x86) Installer der Version 1.6.1. Nach erfolgreicher Installation startet CouchDB und wird auch bei Neustart der Maschine automatisch gestartet. Unter 127.0.0.1:5984/_utils (alternativ localhost:5984/_utils) wird die grafische Oberfläche *Futon* angezeigt. Wird der Port, unter dem CouchDB bereitgestellt wird, verändert, müssen entsprechende Änderungen in der Situationsverwaltung und dem Situation Dashboard vorgenommen werden. Um einen Remotezugriff auf CouchDB zu ermöglichen, müssen Veränderungen an der Konfiguration durchgeführt werden. Über Futon kann die Konfiguration angezeigt werden. Unter dem Feld *bind_address* muss der Wert (standardmäßig 127.0.0.1) in 0.0.0.0 geändert werden, um allen IPs den Zugriff zu ermöglichen. Zusätzlich muss der Port 5984 in der Windows Firewall unter *Eingehende Regeln* als neue Regel definiert werden. Der Remotezugriff ermöglicht die Replikation von Datenbanken. Auf diese Weise können die benötigten Views aus der Datenbank, die unter 192.168.209.246:5984 definiert sind und für die Verwendung von SitOPT benötigt werden, repliziert werden. Abbildung 5.7 zeigt das über Futon erreichbare Replicator Tool.

Abbildung 5.7.: CouchDB Replicator Tool

Unter 192.168.209.246 wird der Prototyp von SitOPT ausgeführt. Die dortige CouchDB Instanz enthält alle benötigten Views und Daten, um das in dieser Arbeit vorgestellte Beispiel zu demonstrieren. Die

[1] https://nodejs.org/
[2] http://couchdb.apache.org/

Replikation funktioniert nur für einzelne Databases (in Abbildung 5.7 am Beispiel von *things* dargestellt), weshalb diese für jede Database *things, situations, sensors, sensorvalues* und *situationtemplates* separat durchzuführen ist.

5.6.2. Node-RED

Node-RED kann über den Package Manager *npm* von Node.js installiert werden. Dieser wird standardmäßig bei der Installation von Node.js installiert. Über *npm install node-red* wird Node-RED heruntergeladen und im aktuell befindlichen Ordner installiert. Fehler bei der Installation bezüglich *node-gyp* schränken die benötigten Funktionen für SitOPT nicht ein und können ignoriert werden [NRi]. Nach erfolgreicher Installation befindet sich das ausführbare Skript *red.js* im Unterordner *node_modules/node-red* und kann in diesem Ordner mittels *node red* gestartet werden. Standardmäßig wird der Port 1880 verwendet, somit kann unter 127.0.0.1:1880 die gestartete Node-RED Instanz erreicht werden.

5.6.3. Situation Dashboard

Im Ordner *Situation Dashboard* wird mit dem Kommando *node server.js* das Situation Dashboard gestartet und unter 127.0.0.1:3001 aufrufbar. Über das Kommando *npm install* innerhalb des Ordners werden die in der Datei *package.json* definierten Module installiert. Sollte es zu Problemen bei der Installation kommen, ist auf GitHub zusätzlich der Ordner als .rar-Datei vorhanden, in welchem alle Module bereits installiert sind. Beim Start der Situationserkennung wird der Transformation Mapper aufgerufen. Dieser ist als .jar-Datei *mappingString.jar* als GitHub-Release bereitgestellt und ist im Unterordner */public/mapper/nodeRed* einzufügen. Die alternative *mapping.jar*-Datei ermöglicht den Aufruf des Transformation Mappers mit einer XML-Datei statt des XML-Inhalts. Da ein Java-Aufruf benötigt wird, muss das Java Runtime Environment (JRE [3]) installiert sein.

5.6.4. Situationsverwaltung

Wie bei dem Situation Dashboard erfolgt zuerst die Installation der benötigten Node.js-Module über das Kommando *npm install*. Da die Situationsverwaltung mit Swagger implementiert worden ist, muss das Node.js spezifische Modul *swagger-node* [4] installiert werden. Dies kann ebenfalls über den Package Manager mit *npm install -g swagger* installiert werden. Unter Umständen wird die Umgebungsvariable nicht definiert. In diesem Fall muss der Pfad *%Username%/AppData/Roaming/npm* dem Systempfad manuell hinzugefügt werden. Im Ordner *Situationsverwaltung* kann daraufhin mit dem Kommando *swagger project start* die Situationsverwaltung gestartet werden und die API Dokumentation, die auf dem Situation Dashboard verlinkt wird, ist zugänglich. Das Kommando *swagger project edit* startet den webbasierten Swagger Editor, der die .yaml-Datei und die entsprechende Visualisierung mittels

[3] https://www.java.com/de/download/
[4] https://github.com/swagger-api/swagger-node

SwaggerUI anzeigt. SwaggerUI ist bereits integriert und muss nicht zusätzlich installiert werden. Die Situationsverwaltung ist unter 192.168.209.246:10010 erreichbar.

5.6.5. Sensoren

Die Sensoren, die Node-RED als Eingabe erhält, werden durch das Node.js Skript *app.js* im Ordner *ComputeSensor* bereitgestellt. Nachdem die benötigten Module mit *npm install* installiert worden sind, wird mit dem Kommando *node app.js* ein Server gestartet, der HTTP GET Requests über den Port 8080 und den entsprechenden Zusatz */cpuusage, /ramusage* und */ping* entgegennimmt und Node-RED die Werte liefert.

6. Evaluation

6.1. Testumgebung

Als Testumgebung wurde eine 64 Bit Version von Windows 8.1 Professional verwendet. Das zugrunde liegende System besteht aus einer Intel Core i5-4300 CPU @ 1.90 GHz und einem 8 GB Arbeitsspeicher.

6.2. Erfüllung der Anforderungen

Im Folgenden wird überprüft, ob die in Tabelle 1.1 definierten Anforderungen erfüllt werden.

A1: Erstellung einer Schnittstelle

Die Situationsverwaltung ist als Schnittstelle zwischen einem SES und situationsbezogenen Workflows erfolgreich implementiert worden. Die Entkopplung dieser beiden Module führt auf beiden Seiten zu deutlichen Vorteilen. Die Erstellung von situationsbezogenen Workflows wird aufgrund der geringeren Komplexität durch den Wegfall der Kontextverarbeitung vereinfacht. Gleichzeitig ist es möglich, beliebige SES zu verwenden, ohne Änderungen an den Workflows vornehmen zu müssen. Die Implementierung des Situation Handlers kann ohne Kenntnisse von SitOPT durchgeführt werden und ist derzeit Teil anderer Arbeiten. Über die in Kapitel 5.2.1 vorgestellten Funktionen der Situationsverwaltung können SES Situationsobjekte speichern und Workflows Situationen beziehen.

A2: Flexibilität

Die Erstellung der Situationsverwaltung als Schnittstelle ermöglicht gleichzeitig die Modularisierung aller beteiligten Systeme. Das SES und die situationsbezogenen Workflows agieren unabhängig voneinander. Die Verwendung alternativer SES wie z.B. Esper ist damit einfach zu bewerkstelligen. Der Transformation Mapper liegt dem Situation Dashboard als separate Bibliothek bei und kann somit beliebig erweitert beziehungsweise verändert werden.

A3: Skalierbarkeit

Die zugrunde liegende Systemarchitektur der Cloud Computing Plattform Openstack bietet die Möglichkeit für ein leicht skalierbares System. Darüber hinaus wird die NoSQL-Datenbank CouchDB verwendet, die gegenüber relationalen Datenbanken besser skaliert. Zusätzlich werden in Kapitel 7.1 die Bereiche von SitOPT beschrieben, die Skalierbarkeit erfordern, sowie mögliche Lösungsvorschläge, die in späteren Arbeiten implementiert werden können.

A4: Benutzerfreundlichkeit

Das Situation Dashboard dient Benutzern als übersichtliches Interface zur Interaktion mit SitOPT. Der Benutzer kann von hieraus alle Things überwachen. Es wurde Wert auf die Übersichtlichkeit gelegt, dass ein schneller Überblick über aktuelle Situationen eines Objekts möglich ist. Die Instanziierung einer Situationsüberwachung ist dahingehend automatisiert, sodass keine Fehlbedienung des Benutzers möglich ist. Der direkte Zugriff auf die Datenbank wurde verhindert und stattdessen die Bedienung über die Situationsverwaltung ermöglicht. Somit können Benutzer nur zugelassene Funktionen verwenden und nicht beispielsweise benötigte Ressourcen wie Views aus der Datenbank löschen. Die Funktionen der Situationsverwaltung wurden mit Swagger bereitgestellt und dokumentiert. Über die auf dem Situation Dashboard bereitgestellte Swagger-Dokumentation kann der Benutzer alle möglichen Funktionen einsehen und direkt testen, ohne einen Client zu implementieren.

A5: Verschiedenartige Sensordaten

Das in SitOPT verwendete Sensormodell beschränkt sich auf die benötigten Attribute eines Sensors beziehungsweise Sensorwertes. Auf diese Weise ist die Verwendung und Weiterverarbeitung verschiedenartiger Sensordaten möglich.

6.3. Ergebnisse

Das in der Einleitung vorgestellte Szenario wurde als Testszenario verwendet, um Laufzeit und Datenverbrauch des Prototyps zu ermitteln. Im ersten Testlauf wurde über einen Zeitraum von 60 Minuten eine Situationserkennung durchgeführt, wobei 63 Situationen erkannt worden sind. Die Situationserkennung wurde hierbei lokal durchgeführt. Eine allgemeine qualitative Aussage über die Häufigkeit einer gültigen Situationserkennung ist meist irrelevant, da diese vom verwendeten Situationstemplate abhängig ist. Der erste Testlauf dient zur differenzierten Laufzeiterkennung, dessen Ergebnisse in Tabelle 6.1 aufgelistet werden. Die Zeile *Gesamt* beschreibt den Zeitraum zwischen einer Situationserkennung durch Node-RED und dem erfolgreichen Einfügen des Situationsobjekts durch die Situationsverwaltung in CouchDB. Die Gesamtzeit lässt sich aufteilen in die Zeit, die der HTTP POST Request benötigt (Erkennen der Situation durch Node-RED bis Empfangen des Requests durch die Situationsverwaltung) und der Zeit, die die Situationsverwaltung benötigt, um das Situationsobjekt zu speichern. Wie zu sehen ist, benötigt die Situationsverwaltung im Durchschnitt 216,13 ms. Der Großteil der Zeit wird für den HTTP POST Request verwendet, welcher durchschnittlich 680,07 ms in

Laufzeiten	Durchschnitt	Max	Min
Gesamt	896.21 ms	1199 ms	221 ms
Situationsverwaltung	216,13 ms	300 ms	124 ms
HTTP POST	680,07 ms	1003 ms	12 ms
CouchDB changes	4,6 ms	8 ms	1 ms

Tabelle 6.1.: Verschiedene Laufzeiten von SitOPT

Bezug nimmt. Hierbei handelt es sich offensichtlich um Verzögerungen bei Node-RED und hat nichts mit der Situationsverwaltung zu tun. Des Weiteren wurde gemessen, wie lange CouchDB benötigt, um die Veränderungen über die _changes-Ressource wieder an die Situationsverwaltung zu senden. Wie in der Zeile *CouchDB changes* zu sehen ist, benötigt dieser Vorgang mit einer durchschnittlichen Laufzeit von 4,6 ms einen unerheblichen Anteil an der Gesamtzeit. Der Zeitpunkt, an welchem die Situationen bei dem Situation Handler ankommen, wird durch einen weiteren HTTP POST Request bestimmt und ist für die Evaluation des Prototyps nicht von Bedeutung. Mit der Annahme, dass ein weiterer HTTP POST Request die selbe Laufzeit wie der vorherige besitzt, benötigt eine Situationsbereitstellung von dem Zeitpunkt, an dem die Situation erkannt wurde und dem Eintreffen der Situation bei dem Situation Handler mindestens 149 ms (Minimalzeiten addiert, HTTP POST x 2) und maximal 2,134 Sekunden (Maximalzeiten addiert, HTTP POST x 2).

Ein Situationsobjekt ist 4 KB groß, ein Sensorwertobjekt je etwa 2,6 KB. In diesem Szenario besitzt eine Situation drei verschiedene Sensorwerte, was bedeutet, dass die Sensorwerte circa 66 Prozent des Datenvolumens erzeugen. Szenarien, in denen mehr Sensorwerte verwendet werden, erhöhen den Anteil am Gesamtvolumen drastisch. Bei der Überwachung von 20 Maschinen mit jeweils drei Sensorwerten über einen Zeitraum von einer Woche mit einem Intervall von einer Sekunden entsteht somit ein Datenvolumen von etwa 6,97 Gigabyte. Erhöht sich die Sensoranzahl auf 10, erhöht sich das Datenvolumen auf etwa 17,7 Gigabyte, wobei die Sensorwerte 86,6 Prozent des Datenvolumens erzeugen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung dieser Arbeit

Die vorliegende Arbeit hat Konzepte vorgestellt, mit denen das General-Purpose-System SitOPT weiterentwickelt wurde. Es wurde gezeigt, wie die Erstellung einer Schnittstelle die Situationserkennung von der Anwendungsebene abgekoppelt hat, um eine anwendungsunabhängige Situationsbereitstellung gewährleisten zu können. Auf Basis eines Situationsmodells können Situationsobjekte erzeugt werden, die von adaptiven situationsbezogenen Workflows verwendet werden können. Des Weiteren eröffnet die Speicherung der Situationsobjekte die Möglichkeit, Situationsverläufe zu erstellen und damit detaillierte Informationen über die überwachte Umgebung zu erhalten.

Zunächst wurden in Kapitel 2 die Grundlagen und Definitionen der zugrundeliegenden Begriffe erklärt. Es wurde gezeigt, welche Eigenschaften von Kontext von Bedeutung sind und wie sich diese auf die Qualität des Kontexts auswirken. Anschließend wurde eine zusätzliche Abstraktionsebene hinzugefügt, um aus den Kontextdaten Situationen zu deduzieren. Es wurde vorgestellt, wie mithilfe von Situationstemplates Situationen definiert werden können.

In Kapitel 3 wurde der Stand der Technik, der für diese Arbeit relevant ist, vorgestellt. Hier ist gezeigt worden, welche Vorteile SitOPT aus den Technologien NoSQL und Cloud Computing erschließen kann. Des Weiteren wurde das Complex Event Processing vorgestellt, die Grundlage vieler Systeme, die für eine Situationserkennung verwendet werden können.

Kapitel 4 stellt die erarbeiteten Konzepte und Architektur von SitOPT dar. Es ist zu sehen, dass großer Wert auf die Modularisierung aller beteiligten Systeme gelegt wurde, um SitOPT flexibel und erweiterbar zu gestalten. Die Grundlage dazu stellt die Situationsverwaltung als Schnittstelle zwischen der Situationserkennung und Workflows dar. Des Weiteren wurden die Datenmodelle beschrieben, die in SitOPT zum Einsatz kommen. Vor allem das Situationsmodell trägt dazu bei, Situationsobjekte erzeugen zu können und eine anwendungsunabhängige Situationsbereitstellung zu ermöglichen. Durch die Beschränkung auf wenige Sensorattribute wird die Verwendung verschiedenartiger Sensoren ermöglicht.

In Kapitel 5 wurde die prototypische Implementierung der in Kapitel 4 vorgeschlagenen Konzepte durchgeführt. Es wurde näher auf die zugrundeliegende Datenbank CouchDB und deren Funktionalitäten eingegangen. Insbesondere wurde der Blick auf den Changes Feed von CouchDB vorgestellt, welcher die Basis für die Situationsbereitstellung an Workflows dient. Weiterhin wurden die Funktionen und Implementierung der Situationsverwaltung mittels Node.js und Swagger detailliert präsentiert. Die Verwendung von Swagger trägt maßgeblich zur einfachen Benutzung und Dokumentation der implementierten Funktionen bei. Bezüglich der Situationserkennung wurde das von SitOPT verwendete Situationserkennungssystem Node-RED und die Transformation eines Situationstemplates in

ein Node-RED kompatibles und äquivalentes Format vorgestellt. Anschließend wurde das Situation Dashboard - der visuelle Einstiegspunkt des Benutzers in SitOPT - und dessen Funktionalitäten vorgestellt. Von hoher Bedeutung war die schnelle Übersicht über alle überwachten Objekte und deren aktuelle Situationen. Abschließend wurde das Setup des Prototyps an einem Beispielsetup auf einer virtuellen Maschine in OpenStack vorgestellt, um weitere Arbeiten an SitOPT zu vereinfachen.

Die Evaluation des Prototyps wird in Kapitel 6 vorgestellt. Es wurde festgestellt, dass die in der Einleitung definierten Anforderungen erfüllt worden sind. Anschließend wurden multiple Testdurchläufe durchgeführt, um die Effizienz von SitOPT zu überprüfen und vorgeschlagen, welche Methodiken implementiert werden könnten, um die Effizienz weiterhin zu verbessern.

Der Prototyp von SitOPT ist derzeit Teil weiterer Arbeiten an der Universität Stuttgart:

- **Situationserkennung basierend auf Complex Event Processing.** In dieser Masterarbeit wird SitOPT um weitere Situationserkennungssysteme auf Basis von CEP erweitert. Zusätzlich wird der Transformation Mapper um die entsprechende Transformation erweitert, um Situationstemplates zu äquivalenten CEP Queries umzuwandeln [MCE].
- **Konzept und Implementierung eines Situation Handlers.** Die Masterarbeit für das Institut für Architektur von Anwendungssystemen hat die Implementierung eines Situation Handlers zum Ziel, um einerseits die Anmeldungen bei der Situationsverwaltung zu verwalten, und andererseits auf Basis erkannter Situationen die Ausführung von Workflow Fragmenten einzuleiten.
- **Flexible Modellierung und Ausführung von Datenverarbeitungs- und Integrationsflüssen.** In dieser Dissertation wird der Einsatz von Situationserkennungssystemen in SitOPT erforscht und dient als Grundlage für die in dieser Arbeit vorgestellt Situationserkennung. Des Weiteren werden die RSM erforscht sowie die Verwendung und Entwicklung der vorgestellten Situationstemplates und deren Transformation mittels des Transformation Mappers.
- **CupCake.** Cupcake ist ein Studienprojekt der Universität Stuttgart. Aufgabe ist es, die Situationserkennung von SitOPT auf die Lernfabrik umzusetzen. Des Weiteren soll die Integratrion weiterer, meist unstrukturierter Datenquellen stattfinden [cc].

In den folgenden Kapiteln werden Vorschläge und Ideen für zukünftige Arbeiten an SitOPT vorgestellt

7.2. Skalierung von SitOPT

SitOPT baut auf Technologien auf, die einfache Skalierbarkeit erlauben. Im Rahmen der Entwicklung des Prototyps ist die funktionale Skalierung nicht implementiert worden. Stattdessen werden in diesem Kapitel Ideen sowie mögliche Alternativen angesprochen, mit denen ein skalierbares System ohne Single Point Of Failure (SPOF) entstehen kann. Die zwei Hauptbereiche, aufgrund derer Skalierbarkeit verlangt wird, ist die Situationserkennung und die Datenbank.

Die durch das SES entstehende CPU-Auslastung liegt einerseits an der Anzahl gleichzeitig ausgeführter Situationstemplates, andererseits durch die gewählte Intervallgröße (wie oft werden Situationen

erkannt). Hochsensible Systeme benötigen eine möglichst geringe Intervallgröße, um über jede Situationsänderung so schnell wie möglich informiert werden und darauf folgende Maßnahmen treffen zu können. Große Fertigungsanlagen müssen unbedingt gleichzeitig alle Maschinen überwachen, um ein autonomes System erstellen zu können. Daraus folgt, dass weder die Anzahl noch die Intervallgröße begrenzt werden darf. Aus diesem Grund verwendet SitOPT Cloud Computing als Infrastruktur. Bereits der Prototyp wird auf einer virtuellen Maschine einer OpenStack Instanz bereitgestellt. Ist das SES ausgelastet, kann eine zweite virtuelle Maschine gestartet werden und die Arbeitslast unter diesen beiden Maschinen aufgeteilt werden. Sinkt die Auslastung, kann eine virtuelle Maschine wieder beendet und die frei gewordenen Kapazitäten anderweitig genutzt werden. Auf diese Weise lässt sich SitOPT (in Abhängigkeit der zugrundeliegenden Rechnerinfrastruktur) beliebig hoch skalieren. Funktionalitäten wie Load-Balancing können diesen Prozess automatisieren, um eine bestmögliche Auslastung der Infrastruktur zu erreichen und kontinuierlich eine ausreichende Anzahl von virtuellen Maschinen bereitzustellen.

Je ausgelasteter das SES ist, desto mehr Daten werden produziert. Wie bei der Evaluation zu sehen war, kann es zu großen Datenmengen kommen, wenn alle erkannten Situationen gespeichert werden. Das Speichern aller Situationen hat allerdings gewisse Vorteile. Monitoring-Systeme können auf diese Weise genauere Schlüsse ziehen, um Systeme effizienter zu gestalten. Neuronale Netze können mit genügend großer Anzahl an Daten Voraussagen über auftretende Situationen treffen. Auf diese Weise wird (in bestimmten Anwendungsfällen) nicht erst bei Auftreten einer unerwünschten Situation gehandelt, sondern bereits zuvor, damit das Auftreten verhindert wird. Die Skalierung der Datenbank folgt demselben Prinzip wie die Skalierung des SES. Bei Bedarf werden weitere virtuelle Maschinen mit einer Datenbank gestartet. Zu jeder neuen Datenbank wird auch eine neue Instanz der Situationsverwaltung gestartet. Das hierbei entstehende Problem ist die Verwaltung der Daten. Wird ein Datum abgefragt, ist vorerst nicht bekannt, auf welcher Datenbank sich dieses befindet. Es gibt drei mögliche Lösungen für dieses Problem:

1. Die einfachste Methode ist, die Anfrage an jede Datenbank zu senden. Hierbei ist nicht garantiert, dass zwei Dokumente auf verschiedenen Datenbanken die gleiche ID zugewiesen bekommen haben. Zugleich entsteht bei vielen Anfragen ein großer Overhead auf allen Datenbanksystemen.
2. Jede Anfrage geht zuerst an einen Föderationslayer. Der Föderationslayer beinhaltet einen Index über alle vorhandenen Daten und deren Speicherorte. Der Vorteil ist, dass nur eine einzige Anfrage gesendet werden muss. Ein Nachteil ist, dass bei dieser Methode ein SPOF entsteht. Gegenmaßnahmen sind das Erstellen von Backup-Datenbanken, die im Falle eines Ausfalls in Kraft treten. Diese Methode profitiert von CouchDBs inkrementellen Replikationsmechanismus. Replikationen können zwischen mehreren Datenbanken automatisch erfolgen. Sollte eine Datenbank zeitweise offline sein, wird, sobald die Datenbank wieder online ist, die Replikation fortgesetzt. Somit ist sichergestellt, dass die Datenbank im Föderationslayer und alle Backups denselben Index besitzen (Eventual Consistency)(Master-Slave). Ein weiteres Problem ist, dass der Föderationslayer einen Flaschenhals darstellt und bei einer großen Anzahl von Anfragen für Ineffizienz sorgt.
3. Um das Flaschenhalsproblem der zweiten Möglichkeit zu beseitigen, besitzt jede Situationsverwaltung einen Index. Jede Anfrage kann an jede beliebige Instanz gesendet werden. Auch diese Methode profitiert von CouchDBs Replikationsmechanismus. Hierbei entsteht ein größeres

Datenvolumen, da auch der Index ein großes Datenvolumen besitzen kann. Zusätzlich ist aufgrund von *Eventual Consistency* nicht sichergestellt, dass alle Datenbanken zu einem beliebigen Zeitpunkt über denselben Index verfügen.

Nützlich bei der Reduzierung des Datenvolumens ist die CouchDB Funktion *Compaction*. Bei Aufruf dieser Funktion auf einer Listenressource werden alle früheren Versionen eines Dokuments gelöscht und nur die aktuellste Version beibehalten. Auf diese Weise behalten alle Dokumente ihre ursprüngliche ID. Würden alle Dokumente gelöscht werden, würden neu erkannte Situationen, die bereits zuvor in der Datenbank vorhanden waren, neue IDs erhalten. Situationsbezogene Workflows, die bereits auf Situationen angemeldet sind (über die ID), würden aufgrund der neuen ID dieser Situation nicht mehr über Veränderungen informiert werden.

7.3. Erstellung von Situationstemplates

Wie in Kapitel 5.3 zu sehen war, ist bereits ein kleines Situationstemplate aufwändig und mühselig zu schreiben. Am Beispiel von Node-RED ist zu sehen, wie eine visuelle Darstellung eines Situationstemplates benutzerfreundlicher und einfacher zu verstehen ist. Mit diesem Ansatz sollten Tools bereitgestellt werden, um die Erstellung von Situationstemplates visuell zu unterstützen. Nur auf diese Weise ist die Erstellung weitaus komplexerer Situationstemplates sinnvoll. Zugleich sollte es möglich sein, bereits bestehende Situationstemplates einzulesen und auf einfache Art und Weise zu ändern, erweitern oder kombinieren.

Durch Verwendung eines Tools und der damit geringeren Komplexität können Situationstemplates erstellt werden, die mehrere Situationen enthalten. Auf diese Weise kann die Situationserkennung effizienter und ressourcenschonender ablaufen. Die daraus resultierenden .xml-Dateien können mit einem bereits bestehenden XML-Schema validiert werden.

Abschnitt: Relation zwischen Things und Templates definieren! nicht jedes Thing kann mit jedem beliebigen Template instanziiert werden. weil sensoren fehlen.

7.4. Verwendung weiterer Situationserkennungssysteme

Die Situationserkennung wurde dahingehend gestaltet, sodass verschiedene SES und verschiedene Sensordatenquellen wie Events, Streams und Messages verwendet werden können . Neben Node-RED sollen in Zukunft Datenstromverarbeitungssysteme (z.B. Odysseus [od]) und CEP Systeme (z.B. Esper [esp]) unterstützt werden. Je nach Sensordatentyp kann verglichen werden, welches SES am besten geeignet ist. Im Folgenden wird die grobe Funktionsweise einer einfachen Situationserkennung mittels Esper dargestellt.

```
//Definition des Situationsobjekts
public static class Situation {
   Float cpu;
   Float ram;
   String name;
   Date timestamp;
```

```
   public Situation(Float sCpu, Float sRam, Date sTimestamp){
      cpu = sCpu;
      ram = sRam;
      timestamp = sTimestamp;
      name = "MachineFailure"
   }

   public float getCpu() {return cpu;}
   public float getRam() {return ram;}
}

//Abfrage des Situationsobjekts auf Gueltigkeit
EPStatement cepStatement = cepAdm.createEPL("select * from Situation having cpu > 80.0 AND ram >
    600.0");

//Methode des Sensors
public static Float GetCPU() throws IOException{
      URL ram = new URL("http://localhost:1338/");
   ...
}
public static Float GetRAM() throws IOException{
      URL ram = new URL("http://localhost:1337/");
      ...
}
```

Listing 7.1: Situationstemplate in Esper

Ein Situationstemplate wird hierbei in drei Bereiche aufgeteilt. Die Java-Klasse *Situation* definiert das Situationsobjekt. Eine EPL Statement vergleicht die Situationsobjekte mit den Konditionen *cpu > 80.0* und *ram > 600.0*. Über die Methoden *getCpu()* und *getRam()* hat das Statement Zugriff auf die Sensorwerte. Die Sensoren werden als Methoden *GetCPU()* und *GetRAM()* implementiert.

```
Configuration cepConfig = new Configuration();
cepConfig.addEventType("Situation", Situation.class.getName());
EPRuntime cepRT = cep.getEPRuntime();
```

Listing 7.2: CEP Konfiguration

Die Klasse *Situation* muss als EventType registriert werden. Zusätzlich wird ein Objekt *cepRT* instanziiert, das zur Laufzeit Events erzeugen kann.

```
Situation situation = new Situation(GetCPU(), GetRAM(), new
      Timestamp(date.getTime()));
cepRT.sendEvent(situation);
```

Listing 7.3: Situationserkennung

In einer while-Schleife können anschließend kontinuierlich Objekte der Klasse *Situation* mit den Sensormethoden als Attribute instanziiert werden und als Events an die CEP-Engine gesendet werden. Ein Listener, der dem in Listing 7.1 definierten Statement hinzugefügt ist, erkennt jedes neue Event und führt das Statement aus.

A. Anhang

CouchDB Views sind ausschließlich in einer CouchDB Instanz vorzufinden. Um beim Ausfall einer virtuellen Maschine weiterhin Zugriff auf die implementierten Views zu haben, werden diese im Folgenden, unterteilt in die jeweiligen Ressourcen, aufgelistet. Des Weiteren werden zu jeder Ressource die jeweiligen Funktionen anhand der Swagger-Repräsentation gezeigt.

A.1. Listenressource situations

Abbildung A.1.: Funktionen der Listenressource *situations*

```
function(doc){
      emit([doc.thing, doc.situationtemplate], doc);
}
```

Listing A.1: CouchDB View _design/situations/existing

```
function(doc){
      emit(null, doc);
}
```

Listing A.2: CouchDB View _design/situations/all

```
function(doc){
      emit(doc.name, doc);
```

```
}
```

Listing A.3: CouchDB View _design/situations/byName

```
function(doc){
      emit([doc.thing, doc.situationtemplate], doc);
}
```

Listing A.4: CouchDB View _design/situations/byThingAndTemplate

```
function(doc){
      emit(doc.thing, doc);
}
```

Listing A.5: CouchDB View _design/situations/monitoring

A.2. Listenressource situationtemplates

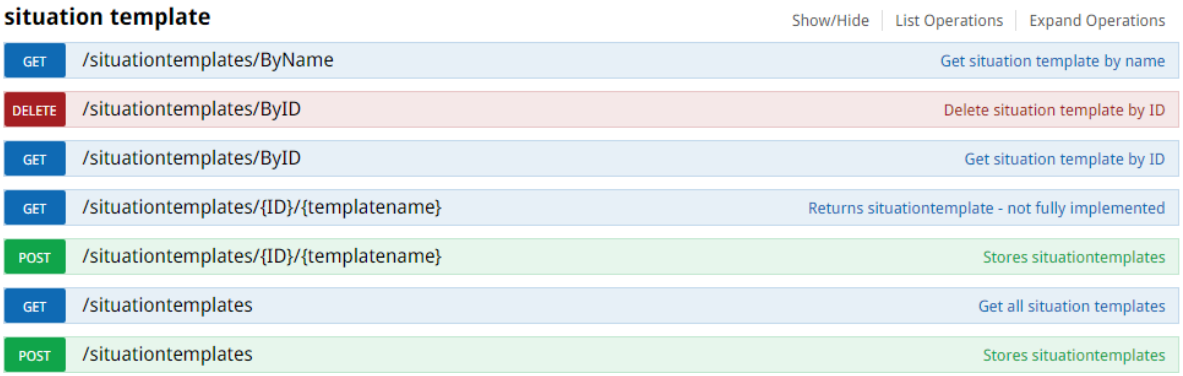

Abbildung A.2.: Funktionen der Listenressource *situationtemplates*

```
function(doc){
      emit(null, doc);
}
```

Listing A.6: CouchDB View _design/situationtemplates/all

```
function(doc){
      emit(doc.name, doc);
}
```

Listing A.7: CouchDB View _design/situationtemplates/byName

```
function(doc){
      emit(doc._id, doc._rev);
}
```

Listing A.8: CouchDB View _design/situationtemplates/idAndRev

A.3. Listenressource things

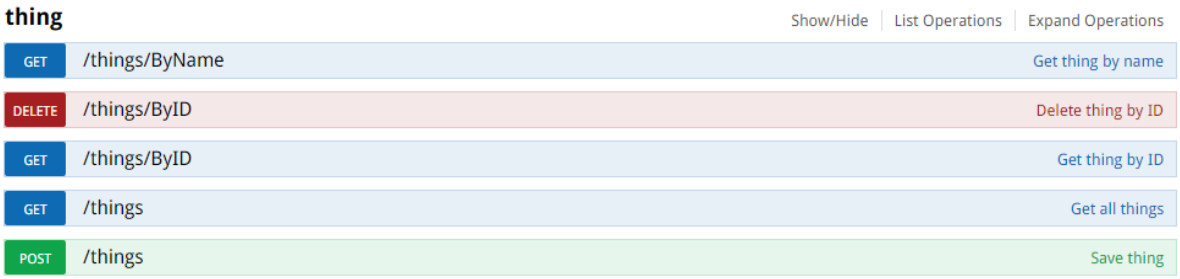

Abbildung A.3.: Funktionen der Listenressource *things*

```
function(doc){
      emit(null, doc);
}
```

Listing A.9: CouchDB View _design/things/all

```
function(doc){
      emit(doc.name, doc);
}
```

Listing A.10: CouchDB View _design/things/byName

A.4. Listenressource sensors

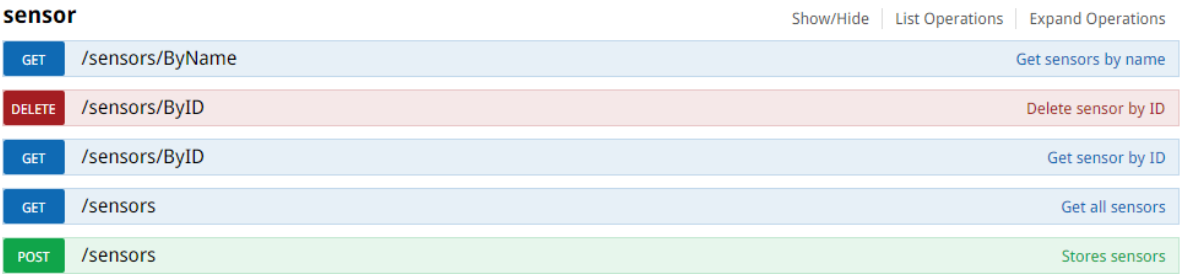

Abbildung A.4.: Funktionen der Listenressource *sensors*

```
function(doc){
      emit(null, doc);
}
```

Listing A.11: CouchDB View _design/sensors/all

```
function(doc){
      emit(doc.name, doc);
}
```

Listing A.12: CouchDB View _design/sensors/byName

A.5. Listenressource sensorvalues

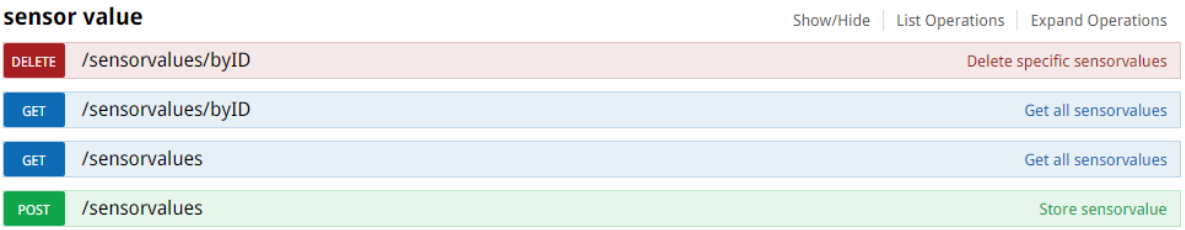

Abbildung A.5.: Funktionen der Listenressource *sensorvalues*

```
function(doc){
      emit(null, doc);
}
```

Listing A.13: CouchDB View _design/sensorvalues/all

```
function(doc){
      emit(doc.sensor, doc._id);
}
```

Listing A.14: CouchDB View _design/sensorvalues/existing

Literaturverzeichnis

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, S. 304–307. Springer, 1999. (Zitiert auf Seite 15)

[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. *Communications of the ACM*, 53(4):50–58, 2010. (Zitiert auf Seite 24)

[ALS10] J. C. Anderson, J. Lehnardt, N. Slater. *CouchDB: the definitive guide*. Ö'Reilly Media, Inc.", 2010. (Zitiert auf den Seiten 41 und 42)

[aws] Amazon Web Services. URL https://aws.amazon.com/de/. (Zitiert auf Seite 24)

[cc] Esper. URL https://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/lehrveranstaltungen/studienprojekte/SS15/Stupro.CUPCAKE.html. (Zitiert auf Seite 62)

[CDba] CouchDB changes. URL http://docs.couchdb.org/en/latest/api/database/changes.html. (Zitiert auf Seite 42)

[CDbb] A Database for the Web. URL http://couchdb.apache.org/. (Zitiert auf Seite 41)

[CDbc] HTTP API Reference. URL http://docs.couchdb.org/en/latest/http-api.html. (Zitiert auf Seite 43)

[CDbd] Replication. URL http://wiki.apache.org/couchdb/Replication. (Zitiert auf Seite 43)

[ceb10] cebr. THE CLOUD DIVIDEND: Part One The economic benefits of cloud computing to business and the wider EMEA economy. Technischer Bericht, centre for economics and business research ltd, 2010. (Zitiert auf Seite 25)

[DAS00] A. K. Dey, G. D. Abowd, D. Salber. A context-based infrastructure for smart environments. In *Managing Interactions in Smart Environments*, S. 114–128. Springer, 2000. (Zitiert auf den Seiten 21 und 22)

[esp] Esper. URL http://www.espertech.com/products/esper.php. (Zitiert auf Seite 64)

[FB] Facebook. URL https://www.facebook.com. (Zitiert auf Seite 23)

[GBH+05] M. Grossmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, T. Schwarz. Efficiently managing context information for large-scale scenarios. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, S. 331–340. IEEE, 2005. (Zitiert auf Seite 16)

[GL12] S. Gilbert, N. A. Lynch. Perspectives on the CAP Theorem. Institute of Electrical and Electronics Engineers, 2012. (Zitiert auf Seite 23)

[GSL14] D. Gorecky, M. Schmitt, M. Loskyll. Mensch-Maschine-Interaktion im Industrie 4.0-Zeitalter. In *Industrie 4.0 in Produktion, Automatisierung und Logistik*, S. 525–542. Springer, 2014. (Zitiert auf den Seiten 7 und 10)

[Hen06] S. Heng. Rfid Chips: Future Technology on Everyone's Lips. *Deutsche Bank Research, E-conomics*, 2006. (Zitiert auf Seite 9)

[HHL+10] K. Häussermann, C. Hubig, P. Levi, F. Leymann, O. Simoneit, M. Wieland, O. Zweigle. Understanding and designing situation-aware mobile and ubiquitous computing systems. In *Proc. of intern. Conf. on Mobile, Ubiquitous and Pervasive Computing*, S. 329–339. 2010. (Zitiert auf Seite 18)

[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 C. Nikolaou, F. Leymann, Herausgeber, *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*, S. 35–49. IBM, 2015. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-34&engl=0. (Zitiert auf Seite 31)

[KBZ+08] U.-P. Käppeler, R. Benkmann, O. Zweigle, R. Lafrenz, P. Levi. Resolving Inconsistencies in Shared Context Models using Multiagent Systems. *Intelligent Autonomous Systems 10: IAS-10*, S. 298, 2008. (Zitiert auf Seite 18)

[Kra] M. Kramer. NoSQL-Datenbanken. (Zitiert auf Seite 43)

[LCG+09] R. Lange, N. Cipriani, L. Geiger, M. Grossmann, H. Weinschrott, A. Brodt, M. Wieland, S. Rizou, K. Rothermel. Making the world wide space happen: New challenges for the nexus context platform. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, S. 1–4. IEEE, 2009. (Zitiert auf Seite 17)

[Ley09] F. Leymann. Cloud Computing: The Next Revolution in IT. In *Photogrammetric Week '09*, S. 3–12. Wichmann Verlag, 2009. (Zitiert auf den Seiten 24 und 25)

[MCE] Masterarbeit - Situationserkennung basierend auf Complex Event Processing. URL https://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/studentische_arbeiten/masterarbeiten/MA_SitRec_CEP.html. (Zitiert auf Seite 62)

[MDb] MongoDB. URL https://www.mongodb.org/. (Zitiert auf Seite 23)

[MG09] P. Mell, T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009. (Zitiert auf Seite 24)

[MGP+11] S. Mitsch, W. Gottesheim, F. H. Pommer, B. Pröll, W. Retschitzegger, W. Schwinger, R. Hutter, G. Rossi, N. Baumgartner. Making workflows situation aware: an ontology-driven framework for dynamic spatial systems. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, S. 182–188. ACM, 2011. (Zitiert auf Seite 9)

[nr] Node-RED. URL http://nodered.org/. (Zitiert auf Seite 49)

[NRi] Installation. URL http://nodered.org/docs/getting-started/installation.html. (Zitiert auf Seite 54)

[od] Odysseus. URL http://odysseus.informatik.uni-oldenburg.de/. (Zitiert auf Seite 64)

[pa] Patriot Act. URL http://thomas.loc.gov/cgi-bin/query/z?c107:H.R.3162.ENR:. (Zitiert auf Seite 25)

[Pri08] D. Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008. (Zitiert auf den Seiten 23 und 24)

[RB15] J. D. Ralf Bruns. *Complex Event Processing: Komplexe Analyse von massiven Datenströmen mit CEP*. Springer, 2015. Seite 6. (Zitiert auf den Seiten 7 und 26)

[Rob10] D. Robins. Complex event processing. In *Second International Workshop on Education Technology and Computer Science. Wuhan*. 2010. (Zitiert auf Seite 26)

[SSK11] C. Strauch, U.-L. S. Sites, W. Kriha. NoSQL databases. *Lecture Notes, Stuttgart Media University*, 2011. (Zitiert auf Seite 41)

[TWT] Twitter. URL https://twitter.com/?lang=de. (Zitiert auf Seite 23)

[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*, S. 2029–2040. Citeseer, 2009. (Zitiert auf den Seiten 16 und 18)

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, F. Leymann. Towards context-aware workflows. In *CAiSE07 Proc. of the Workshops and Doctoral Consortium*, Band 2, S. 25. 2007. (Zitiert auf Seite 15)

[wmc] Workflow Management Coalition. URL http://www.wfmc.org/. (Zitiert auf Seite 15)

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbucher, 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*, S. 32–37. IEEE, 2015. (Zitiert auf den Seiten 10 und 15)

[Zwe11] O. G. Zweigle. *Erweiterung kognitiver Fähigkeiten in Multiagentensystemen durch Kommunikation, Rollenverteilung und Situationsanalyse*. Shaker, 2011. (Zitiert auf den Seiten 15 und 22)

Alle URLs wurden zuletzt am 02. 09. 2015 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