Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 44

Konzept und Implementierung eines Situation Handlers

Stefan Fürst

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Dipl.-Inf. Uwe Breitenbücher
Beginn am:	1. Juni 2015
Beendet am:	1. Dezember 2015
CR-Nummer:	H.4.1

Kurzfassung

Im Rahmen von Industrie 4.0 werden Konzepte zur Automatisierung von Industrieanlagen erforscht. Eine wichtige Rolle spielen dabei Sensoren zur Erkennung von Kontextinformationen und Workflows zur automatisierten Ausführung von Geschäftsprozessen. Um einen hohen Grad an Autonomie zu erreichen, müssen Workflows den von Sensoren erkannten Kontext berücksichtigen und angemessen auf gegenwärtige Situationen reagieren. Die Berücksichtigung von Kontextinformationen in Workflows macht deren Modellierung jedoch außerordentlich komplex, da durch die Kontextbehandlung sehr viele Situationen individuell behandelt werden müssen.

Eine Möglichkeit zur Handhabung dieser Komplexität besteht in der Aufteilung von Workflows in Workflow-Fragmente, die in Abhängigkeit zur vorherrschenden Situation zur Ausführung einer bestimmten Aktivität ausgewählt werden. Dies ermöglicht die gezielte Modellierung von Workflow-Fragmenten, die für eine bestimmte Situation die jeweils am besten geeignete Lösung beschreiben. In dieser Arbeit wird ein Situation Handler zur Handhabung von Kontextinformationen entwickelt, der von Workflows aufgerufen werden kann, um automatisiert ein geeignetes Workflow-Fragment zur Ausführung einer bestimmten Aktivität auszuwählen. Dabei wird die Auswahl des Fragments von der gegenwärtigen Situation beeinflusst. Zur Erkennung von vorherrschenden Situationen wird ein Situation Recognition System eingesetzt, das feingranulare, technische Kontextinformationen zu höherwertigen Situationen abstrahiert. Das Situation Recognition System wurde im Rahmen eines Forschungsprojektes der Universität Stuttgart entwickelt. Es wird zudem der situationsabhängige Versand von Notifikationen unterstützt. Zum Versand wird ein Pluginsystem eingesetzt, das die Verwendung von verschiedenen Technologien zum Versand erlaubt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Algorithmen

1. Einleitung

Eines der in der Wirtschaft derzeit meist geförderten Paradigmen heißt Industrie 4.0. Unter diesem Schlagwort werden verschiedene Möglichkeiten zur Weiterentwicklung von Fertigungsprozessen in der Industrie erforscht. Ziel dabei ist insbesondere eine Automatisierung von Fertigungsprozessen, die in weitgehend autonomen *Smart Factories* eingesetzt werden sollen. Die Automatisierung soll vor allem durch eine umfassende Digitalisierung in der Fertigung erreicht werden [LFK+14].

Um das Ziel der Digitalisierung zu erreichen, werden unter anderem Workflows eingesetzt, um die Ausführung von industriellen Prozessen zu automatisieren. Damit ein Workflow möglichst unabhängig von menschlichen Eingriffen abläuft, muss er selbstständig den aktuellen Zustand der Umgebung, das heißt die vorherrschende, gegenwärtige Situation, berücksichtigen. Der Zustand, zum Beispiel von eingesetzten Maschinen, kann über verschiedene Sensoren überwacht werden. Die gesammelten Informationen der Sensoren werden auch als *Kontextinformationen* bezeichnet. Workflows zur Abbildung von industriellen Prozessen unter Berücksichtigung von Kontextinformationen sind jedoch außerordentlich komplex, da durch die Behandlung der Kontextinformationen häufig eine Vielzahl verschiedener Pfade im Workflow benötigt werden, um alle möglicherweise eintretenden Situationen abzudecken. Beispielsweise ist es möglich, dass vor der Verwendung einer Maschine innerhalb eines Fertigungsprozesses aufgrund von Verschleiß ein Teil ausgetauscht werden muss. Der Verschleiß des Teils wird automatisch durch durch einen Sensor erkannt. Der Austausch muss jedoch in dem Workflow als extra Aktion dargestellt werden, die nur in Falle von großem Verschleiß durchgeführt wird. Die Austauschaktion muss im Workflow folglich in einem eigenen Pfad ausgeführt werden. Wenn ein Fertigungsprozess viele Stufen umfasst, muss im Workflow selbst eine Vielzahl von verschiedenen Pfaden modelliert werden, um alle möglichen Situationen abzudecken. Das Workflow-Modell wird dadurch enorm komplex. Dies wirkt sich negativ auf die Wartbarkeit und Erweiterbarkeit des Modells aus.

In den letzten Jahren wurden zahlreiche Ansätze entwickelt, welche die Berücksichtigung von Kontextinformationen in Workflows behandeln. Diese haben häufig die Eigenschaft, dass die feingranularen Kontextinformationen innerhalb des Workflows behandelt werden, was zu dem beschriebenen Problem führt, dass sehr viele Pfade zur Behandlung der Information benötigt werden. Das Workflow-Modell wird so in einer gewissen Weise verschmutzt und verliert an Übersichtlichkeit [BHK+15]. In dieser Arbeit soll das Problem auf eine andere Weise gelöst werden. Dabei werden im Wesentlichen die beiden folgenden Konzepte berücksichtigt:

- Als Kontextinformation werden nicht direkt die Informationen von Sensoren verwendet. Stattdessen werden diese in Form von *Situationen* abstrahiert, die Daten von mehr als einem Sensor berücksichtigen. Damit müssen nicht mehr die feingranularen Sensordaten beachtet werden,

sondern nur noch die grobgranularen Situationen. Dadurch kann die Erkennung der gegenwärtigen Situation an externe Systeme ausgelagert werden, wodurch die Aggregation von Kontextinformationen zu Situationen nicht mehr vom Workflow selbst getätigt werden muss.

- Die situationsabhängigen Aktionen eines Workflows werden nicht mehr direkt im Workflow modelliert, sondern in Workflow-Fragmente ausgelagert. Die einzelnen Pfade werden folglich durch diese Fragmente dargestellt. Es muss in diesem Fall ein „Haupt-Workflow“ erstellt werden, der keine Situationen berücksichtigt, sondern nur die gewünschte Aktion angibt. Die Auswahl der situationsabhängig richtigen Umsetzung der Aktion (Workflow-Fragment) wird nicht im Workflow durchgeführt, sondern durch den in dieser Arbeit entwickelten Situation Handler.

Zur Umsetzung dieser Konzepte werden zwei Komponenten benötigt. Eine, die das Konzept der Situation klar definiert und in der Lage dazu ist, anhand von Sensordaten solche Situationen zu erkennen. Eine andere, um abhängig von der Situation ein bestimmtes Workflow-Fragment auszuwählen, auszuführen und das Ausführungsresultat des Fragments an den aufrufenden Workflow zurückzusenden. Das Situationskonzept [WSBL15] sowie die Komponente SitRS [HWS+15] zur Erkennung von Situationen wurden bereits im Rahmen eines Forschungsprojektes der Universität Stuttgart entwickelt. Im Rahmen dieser Arbeit wurden die Konzeption und prototypische Implementierung des Situation Handlers zur situationsabhängigen Ausführung von Workflow-Fragmenten durchgeführt. Die entwickelte Komponente kann mit nahezu beliebigen BPEL-Workflows kommunizieren, die dem SESE-Schema [KELU10] entsprechen. Sie kooperiert mit dem bereits entwickelten Situationserkennungssystem SitRS [HWS+15], um Situationen zu erkennen und leitet die Anfragen von Workflows entsprechend an verschiedene Workflow-Fragmente weiter. Die Workflow-Fragmente können zur Laufzeit über verschiedene Möglichkeiten als Endpunkt angegeben werden. Der Situation Handler wählt aus den verschiedenen Endpunkten den geeignetsten aus und leitet Anfragen an die vom Endpunkt vorgegebene Adresse weiter. Der Situation Handler kann damit als situationsbewusster Service Bus (engl. *Situation-Aware Service Bus*) beschrieben werden. Die Auswahl des Endpunktes erfolgt über einen hierfür entworfenen Algorithmus, der eine beliebige Anzahl von Situationen berücksichtigt. Dabei kann der Zustand einer Situation festgelegt werden (eine Situation liegt vor oder nicht). Des Weiteren unterstützt der Situation Handler den situationsabhängigen Versand von Notifikationen beziehungsweise die situationsabhängige Ausführung von Aktionen zum Versand von Notifikationen. Die Aktionen werden mithilfe von Plugins ausgeführt, die zur Laufzeit jederzeit hinzugefügt oder verändert werden können. Es wurden Plugins zum Versand von E-Mails, HTTP-Anfragen und Android Push-Benachrichtigungen entwickelt. Zudem können die Plugins dazu verwendet werden, um situationsabhängig Workflow-Fragmente aufzurufen. Dadurch können Konzepte wie das *Situation Event* von Breitenbücher et al. [BHK+15] realisiert werden.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Aufgabenstellung: In diesem Kapitel wird die genaue Aufgabenstellung der Arbeit erläutert.

Kapitel 3 – Grundlagen und Terminologie: In den Grundlagen werden Begriffe erläutert, die wichtig für das Verständnis der Arbeit sind. Zudem werden die Grundlagen von SitOPT erklärt, dem Forschungsprojekt, in dem das System zur Erkennung von Situationen entwickelt wurde.

Kapitel 4 – Konzept des Situation Handlers: Das Konzept der Arbeit wird in diesem Kapitel vorgestellt. Hierbei wird nicht auf konkrete Technologien eingegangen. Stattdessen erfolgt eine Beschreibung, wie die Aufgabenstellung auf konzeptioneller Ebene umgesetzt wird.

Kapitel 5 – Implementierung des Situation Handlers: Im Implementierungskapitel wird die Umsetzung des Entwurfs beschrieben. Dabei wird auf ausgewählte Technologien, die Architektur und auf wichtige Implementierungsdetails eingegangen.

Kapitel 6 – Verwandte Arbeiten: Ziel dieses Kapitels ist die Vorstellung von ähnlichen Arbeiten und die Herausarbeitung der Unterschiede zwischen diesen Arbeiten und dem vorliegenden Ansatz.

Kapitel 7 – Zusammenfassung und Ausblick: Abschließend werden die Ergebnisse zusammengefasst und es wird ein Ausblick auf zukünftige Herausforderungen geliefert.

2. Aufgabenstellung

In diesem Abschnitt wird die detaillierte Aufgabenstellung für diese Arbeit erläutert und es werden die daraus resultierenden Anforderungen an das Ergebnis festgehalten.

Ziel der Arbeit ist der Entwurf und die Implementierung einer Komponente zur Behandlung von Situationen, nachfolgend **Situation Handler** genannt. Das Konzept einer Situation wird in Kapitel 3 vorgestellt. Eine Komponente zur Modellierung und Erkennung von Situationen in verschiedenen Umgebungen ist das im Rahmen von SitOPT entwickelte System SitRS [HWS+15], welches ebenfalls in Kapitel 3 vorgestellt wird. Diese Komponente und alle zugehörigen Artefakte werden der Einfachheit halber als Situationserkennungssystem (**SES**) bezeichnet. Dieses ist nicht zu verwechseln mit dem Situation Recognition System (**SRS**), welches eine einzelne Komponente von SitRS darstellt. Der in dieser Arbeit entwickelte Situation Handler soll mit dem SES zusammenarbeiten. Dies bedeutet, dass Situationen, die durch das SES erkannt werden, vom Situation Handler behandelt werden. Dabei sollen zwei verschiedene Arten der Behandlung möglich sein:

(i) In der ersten Form soll beim Auftreten oder Verschwinden einer Situation eine bestimmte Aktion durchgeführt werden. Diese soll möglichst frei spezifizierbar sein. Die Grundidee ist jedoch, dass in erster Linie Notifikationen auf beliebige Art versandt werden. (ii) Bei der zweiten Art der Situationsbehandlung soll für einen Dienstaufruf durch einen Workflow, abhängig von aktuell vorliegenden Situationen, eine passende Implementierung des aufgerufenen Dienstes ausgewählt werden. Der Dienst ist dabei als Workflow-Fragment implementiert. Für Workflow-Anfragen muss zudem ein Rollbackhandling durchgeführt werden können. Hierbei soll das ausgeführte Workflow-Fragment benachrichtigt werden, wenn sich eine relevante Situationsänderung ergibt. Das Fragment muss daraufhin bereits ausgeführte Aktionen rückgängig machen. Die Auswahl der Workflow-Fragmente für Dienstaufrufe sowie die Aktionen beim Auftreten einer bestimmten Situation sollen frei konfigurierbar sein. Zur Konfiguration sollen sowohl eine Benutzeroberfläche als auch eine API zur Verfügung gestellt werden.

Um den Versand von Notifikationen bei einer Situationsänderung möglichst flexibel zu gestalten, soll der Situation Handler ein Pluginsystem zur Verfügung stellen. Dieses soll es erlauben, verschiedene Plugins zur Kommunikation in den Situation Handler einzubinden und zum Versand von Notifikationen zu verwenden. Zur Aufgabe dieser Arbeit gehört zudem die Entwicklung von einem oder mehreren Kommunikationsplugins für den Situation Handler. Der Versand von Notifikationen soll zudem in einem Log aufgezeichnet werden, das über die Oberfläche zugänglich sein soll. Um die Verwendung des Situationserkennungssystems zu erleichtern, soll das Deployment und die Verwaltung von Situation Templates ebenfalls über die Oberfläche des Situation Handlers ermöglicht werden.

3. Grundlagen und Terminologie

In diesem Kapitel werden Begriffe und Konzepte erläutert, die eine wichtige Grundlage für das Verständnis der vorliegenden Arbeit darstellen. Hierbei wird insbesondere in Abschnitt 3.3 Projekt SitOPT vorgestellt. Der in dieser Arbeit entwickelte Situation Handler stellt einen Teil von SitOPT dar und greift auf bereits im Rahmen von SitOPT entwickelte Komponenten zurück.

3.1. Workflows und dienstorientierte Architekturen

SitOPT setzt auf die Anpassung von Anwendungen an Situationen durch adaptive Workflows [WSBL15]. An dieser Stelle wird daher zunächst der Begriff des *Workflows* erklärt, wobei die Definition des Begriffs durch Leymann und Roller [LR00, S.1] als Vorlage dient. Als Grundlage für Workflows dienen Geschäftsprozesse. Ein Geschäftsprozess kann als Folge von Aktivitäten definiert werden, die von verschiedenen Personen ausgeführt werden. Der Prozess kann verschiedene Dokumente als Ergebnis produzieren. Typische Beispiele für Geschäftsprozesse sind der Prozess zur Vergabe eines Kredites in einer Bank oder die Aktivitäten, die nacheinander an einem Fließband durchgeführt werden. Die Aktivitäten werden häufig wiederholt und folgen dabei immer demselben Muster. Dieses Muster wird als Prozessmodell bezeichnet (vgl. [LR00, S.1]).

Geschäftsprozesse interagieren häufig miteinander. Ein typisches Szenario besteht darin, dass ein Geschäftsprozess einen anderen Geschäftsprozess startet und dieser danach unabhängig vom ersten Geschäftsprozess abläuft. Häufig ist es der Fall, dass die Geschäftsprozesse in ihrem weiteren Verlauf weiterhin miteinander interagieren. In einem anderen möglichen Szenario startet ein Geschäftsprozess einen anderen und wartet solange, bis dieser abgeschlossen ist (zum Beispiel, da er das Resultat dieses Geschäftsprozesses benötigt, um fortfahren zu können). Der gestartete Geschäftsprozess wird in diesem Fall als *Subprozess* bezeichnet.

Als Workflow wird ein Geschäftsprozess oder ein Teil eines Geschäftsprozesses bezeichnet, der auf einem Computer ausgeführt wird. Das Verhältnis zwischen Geschäftsprozessen und Workflows wird in Abbildung 3.1 dargestellt. Links im Bild wird das Modell des Geschäftsprozesses in der realen Welt gezeigt. Ein Geschäftsprozess bezeichnet das Modell in der Ausführung beziehungsweise eine Instanz des Prozessmodells. Wird das Prozessmodell auf den Computer übertragen, wird dieses als Workflow-Modell beschrieben. Ein Workflow stellt wiederum eine ausgeführte Instanz des Workflow-Modells dar (vgl. [LR00, S.7]).

Mit der zunehmenden Verbreitung von dienstorientierten Architekturen wurden Workflows auch für diese relevant. Dienstorientierte Architekturen zeichnen sich dadurch aus, dass sämtliche Funktionalitäten des System als Dienste (engl. *Services*) angeboten werden. Ein Dienst besitzt insbesondere die Eigenschaft, dass er zu jedem Zeitpunkt an einer bestimmten (Netzwerk-) Adresse verfügbar ist

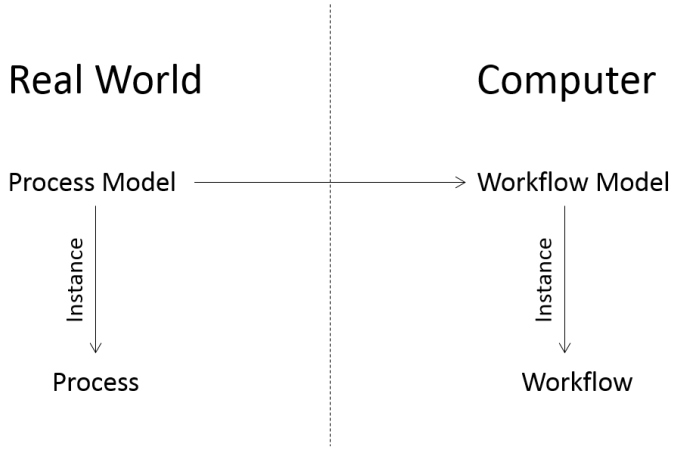


Abbildung 3.1.: Workflows und Geschäftsprozesse [LR00, S.7]

(vgl. [WCL+05, S.13ff]). Der Dienst bietet eine bestimmte Funktionalität an und stellt Informationen darüber bereit, wie der Dienst zu verwenden, das heißt aufzurufen, ist. In einer dienstorientierten Architektur wird eine Funktionalität häufig durch die Komposition von mehreren Diensten implementiert. Zum Beispiel greift ein Dienst zur Buchung einer Reise auf Dienste zu, mit denen ein Hotel und ein Flug gebucht werden. Die Komposition von mehreren Diensten kann dabei auf Grundlage der Workflow-Technologie umgesetzt werden. Ein Workflow-Modell beschreibt in diesem Fall in welcher Reihenfolge bestimmte Dienste verwendet werden. Dabei ist es möglich, dass die verwendeten Dienste als Workflow implementiert sind. Dadurch ergibt sich ein rekursives Aggregationsmodell für Workflow-Modelle [LR00].

Eine Sprache zur Spezifikation von Workflow-Modellen, bei der die Aktivitäten durch Web Services implementiert werden, ist BPEL (Business Process Execution Language). Bei BPEL handelt es sich um eine XML-basierte Sprache, die von OASIS[1] spezifiziert wurde. Die bei BPEL verwendeten Web Services implementieren meist Teile der WS-* Spezifikationen (erstellt vom W3C[2] und OASIS). Hierbei wird insbesondere WSDL (Web Service Description Language) zur Beschreibung von Web Service Schnittstellen verwendet und SOAP[3] über HTTP zum Austausch von Nachrichten.

3.2. Industrie 4.0 und das Internet der Dinge

Einer der Haupteinsatzzwecke von SitOPT ist die Automatisierung von industriellen Prozessen unter dem Schlagwort *Industrie 4.0*. Mit dem Begriff Industrie 4.0 wird ein von der deutschen Bundesregierung gefördertes Zukunftsprojekt beschrieben, das insbesondere auf eine engere Verbindung von Produktions- mit Informationstechnik zur Modernisierung von industriellen Produktionsanlagen und

[1] https://www.oasis-open.org/
[2] http://www.w3.org/
[3] http://www.w3.org/TR/soap/

somit auf eine Digitalisierung der Industrie setzt [BMBb]. Dadurch soll „eine starke Individualisierung der Produkte unter den Bedingungen einer hoch flexibilisierten (Großserien-) Produktion“ [BMBb] erreicht werden. Aufgrund der Tragweite der erwarteten Veränderungen wird die Umsetzung von Industrie 4.0 als Beginn einer vierten industriellen Revolution bezeichnet [BMBb]. Ziele der Industrie 4.0 sind vor allem die Entwicklung von Technologiestandards und Werkzeugen, um Unternehmen bei der Modernisierung zu unterstützen [BMBb]. Als Zukunftsvision wird hierbei häufig das Bild einer *Smart Factory* beschrieben [LFK+14]. Bei einer *Smart Factory* handelt es sich um eine Produktionsanlage, die sich weitestgehend selbst organisiert und ohne menschliche Eingriffe zurechtkommt [Wik]. Sie zeichnet sich durch Anpassungsfähigkeit und Effizienz aus [BMBa].

Eine der Grundlagen für *Industrie 4.0* stellt das *Internet der Dinge* (englisch: *Internet of Things*, kurz *IoT*) dar. Atzori et al. [AIM10] beschreiben die grundlegende Idee des IoT als die durchgängige Anwesenheit von verschiedenen Dingen, die miteinander interagieren und kooperieren, um gemeinsame Ziele zu erreichen. Dabei können Dinge insbesondere über eine eindeutige Adresse angesprochen werden. Dinge sind hier beispielsweise Sensoren, RFID (Radio-Frequency Identification) Transponder oder auch Mobiltelefone. Zu beachten ist hierbei, dass der Begriff *Internet der Dinge* die Bezeichnung für die beschriebene Grundidee beziehungsweise Vision ist. Es existieren verschiedene Ansätze, wie genau die Grundidee interpretiert und umgesetzt wird [AIM10]. Im Zusammenhang mit Industrie 4.0 beschreibt das IoT vor allem die Verwendung von Sensoren in Produktionsanlagen (*Smart Factories*), um eine weitgehende Automatisierung zu erreichen.

3.3. SitOPT

Bei SitOPT handelt es sich um ein Forschungsprojekt der Universität Stuttgart[4]. Das Ziel von SitOPT besteht darin, „Konzepte und Methoden zu entwickeln, die es situationsbezogenen Anwendungen erlauben, sich autonom an die dynamische Umgebung, in der sie ablaufen, anzupassen“ [Stu]. Dies umfasst zum einen die Entwicklung von Konzepten, um Situationen zu beschreiben und zu erkennen und zum anderen die Anpassung von Anwendungen an diese Situationen.

Der in dieser Arbeit entwickelte Situation Handler ist Teil von SitOPT und interagiert mit anderen Komponenten, die im Rahmen von SitOPT entwickelt wurden. Die relevanten Komponenten von SitOPT werden an dieser Stelle vorgestellt. Dabei wird zunächst in Abschnitt 3.3.1 der Begriff der Situation näher erläutert, anschließend werden *Situation Templates* in 3.3.2 erklärt. Danach wird in Abschnitt 3.3.3 der Situationserkennungsservice *SitRS* vorgestellt. Die Anpassung von Anwendungen an Situationen erfolgt durch Workflows, die aus situationsabhängigen Workflow-Fragmenten zusammengesetzt werden. Die Anpassung der Workflows ist Teil der vorliegenden Arbeit und wird daher im Hauptteil beschrieben.

[4] `https://www.ipvs.uni-stuttgart.de/abteilungen/as/forschung/projekte/SitOPT`

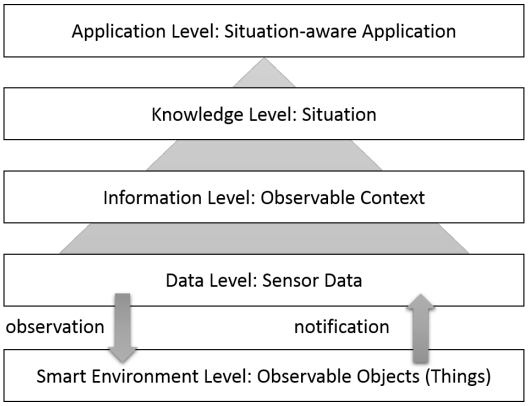


Abbildung 3.2.: Kontext und Situationen [HWS+15]

3.3.1. Kontext und Situationen

Eine der Hauptaufgaben von SitOPT stellt die Erkennung von Situationen dar. Wieland et al. [WSBL15] bezeichnen Situationen als eine Aggregation verschiedener Kontextinformationen, wodurch diese interpretiert und zu höherwertigen Aussagen zusammengefasst beziehungsweise abstrahiert werden. *Kontext* hingegen wird als die Information bezeichnet, die auf niedrigerem Level direkt aus den Daten von Sensoren gewonnen wird.

Hirmer et al. [HWS+15] beschreiben den Zusammenhang von Sensordaten, Kontext und Situationen wie in Abbildung 3.2 dargestellt. Auf der untersten Ebene befinden sich *Dinge*, die durch Sensoren überwacht werden. Die Sensoren liefern Rohdaten, die interpretiert werden müssen, bevor sie verwendet werden können. Durch Auswertung der Rohdaten entsteht Information in Form von Kontextdaten. Hierbei werden die Daten Objekten aus der realen Welt zugeordnet. Aus den Kontextinformationen kann wiederum Wissen gewonnen werden. Dieses wird durch Situationen repräsentiert. Eine Situation stellt folglich eine weitere Interpretation und Aggregation von Kontextdaten dar. Das Wissen über Situationen kann durch situationsbewusste Anwendungen genutzt werden. Ein mögliches Beispiel (vgl. [HWS+15]) beschreibt die Überwachung eines Servers. Dabei werden Sensoren zur Überwachung von CPU und RAM des Servers benutzt. Auf der Datenebene werden hier einfache Daten, wie zum Beispiel die Temperaturwerte der CPU oder der belegte RAM in MB, gesammelt. Aus diesen Daten können Kontextinformationen für CPU und RAM erstellt werden, das heißt die Sensordaten werden dem jeweiligen Objekt zugeordnet. Dadurch kann beispielsweise der Zustand des RAMs festgestellt werden. Im nächsten Schritt werden die Kontextinformationen zu Situationen abstrahiert. Als Beispiel für eine Situation auf dem Server könnte die Situation *Zustand kritisch* dienen. Die Situation *Zustand kritisch* könnte so definiert werden, dass über 90 Prozent des RAMs belegt sein müssen und die CPU eine kritische Temperatur erreicht hat. Die Situation könnte von einem Load-Balancer verwendet werden, der Anfragen nur an Server weiterleitet, bei denen die Situation *Zustand kritisch* nicht vorliegt.

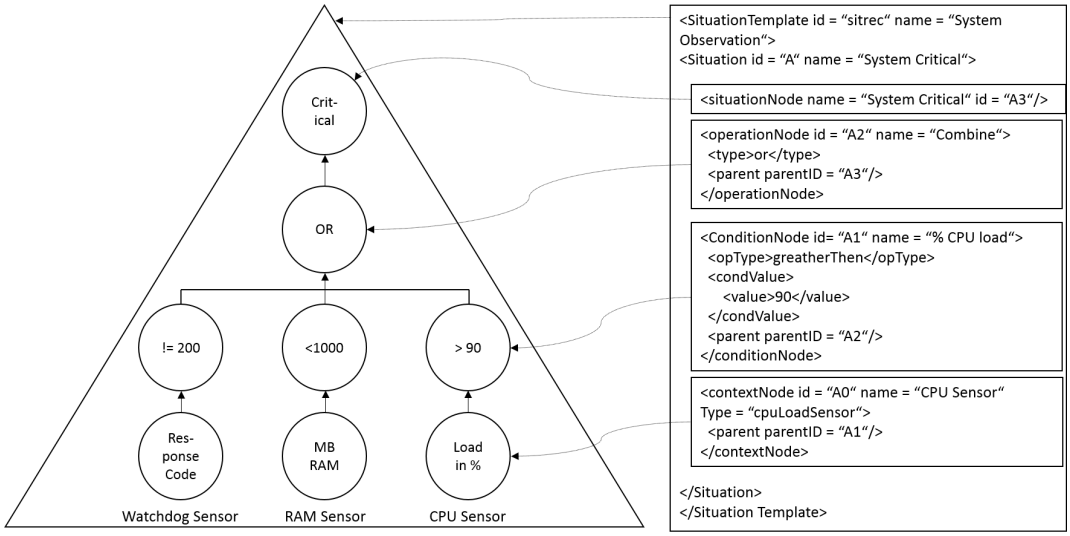


Abbildung 3.3.: Situation Template [HWS+15]

3.3.2. Situation Templates

Die Beschreibung von Situationen erfolgt bei SitOPT mithilfe von sogenannten *Situation Templates*. Ein Situation Template wird durch das *Situation Recognition System* (SRS) ausgeführt (siehe Abschnitt 3.3.3). Zur Ausführung wird das Situation Template mit einem *Ding* assoziiert. Das Resultat der Ausführung ist eine Situation, die durch das Situation Template beschrieben wird und auf dem Ding vorliegt. Ein Ding kann ein beliebiges Objekt sein, an dem Situationen durch Sensoren erkannt werden können. Ein Ding muss im System durch eine ID eindeutig identifiziert werden können. Auch Situation Templates besitzen eine eindeutige ID. Die Kombination von Situation Template und Ding ist hierbei eindeutig. Dies bedeutet, dass auch die Situation, die auf einem Ding von einem bestimmten Situation Template erkannt wird, eindeutig definiert ist. Es ist zu beachten, dass auf einem Ding mehrere Situation Templates ausgeführt werden können, wodurch dem Ding auch mehrere Situationen zugeordnet werden können.

Ein Situation Template wird als sogenannter *Situation-Aggregation-Tree* [ZHKL09] dargestellt. Bei einem *Situation-Aggregation-Tree* handelt es sich um einen gerichteten Graphen, der in Form einer Baumstruktur aufgebaut ist. Die Zweige des Baumes werden von den Blättern zur Wurzel aggregiert. Ein Beispiel für ein Situation Template von Hirmer et al. [HWS+15] wird in Abbildung 3.3 dargestellt.

Auf der linken Seite der Abbildung wird das Situation Template schematisch dargestellt. Die Knoten auf der unteren Ebene werden als *Kontextknoten* bezeichnet. Diese stellen Sensoren dar. Kontextknoten sind mit *Bedingungsknoten* verbunden, die eine Bedingung definieren, die der Wert des Kontextknotens erfüllen muss. Mehrere Bedingungsknoten können von einem *Operationsknoten* zusammengefasst werden. Ein Operationsknoten definiert einen logischen Ausdruck. Es können beliebig viele Operationsknoten zur Aggregation des Baumes verwendet werden (in Abbildung 3.3 nur ein einzelner

`OR`-Knoten). Der letzte Operationsknoten definiert den Übergang zum Wurzelknoten. Der Wurzelknoten repräsentiert die Situation, die das Resultat der Auswertung des Situation Templates darstellt. Bei der Auswertung werden die definierten Bedingungen überprüft. Sobald eine Bedingung verletzt wird, wird die Auswertung abgebrochen. In diesem Fall kann sofort festgestellt werden, dass die Situation nicht vorliegt.

Auf der rechten Seite von Abbildung 3.3 wird die XML-Darstellung des Situation Templates gezeigt. XML wird als Sprache zur Beschreibung von Situation Templates verwendet, da XML maschinell verarbeitet werden kann und gute Strukturierungsmöglichkeiten bietet. Ein längeres Beispiel für ein Situation Template befindet sich im Anhang (Abschnitt A.1). Das Template in Abbildung 3.3 beschreibt die Situation `System Critical` eines Webservers. Der Webserver besitzt die Sensoren `Watchdog`, `RAM` und `CPU`. Diese Sensoren werden als Kontextknoten definiert. Um eine Bedingung für den Wert des Sensors festzulegen, werden die Kontextknoten jeweils mit einem Bedingungsknoten verbunden. Der CPU-Sensor misst beispielsweise die Last der CPU in Prozent. Der Bedingungsknoten spezifiziert dazu die Bedingung, dass die Last der CPU über 90 Prozent liegen muss. Die Bedingungsknoten sind alle mit demselben Operationsknoten verbunden, der als Operation ein logisches `OR` definiert. Der Operationsknoten ist in diesem Fall direkt mit dem Situationsknoten verbunden. Dies bedeutet, dass sobald der Operationsknoten ein positives Ergebnis liefert, die Situation `System Critical` vorliegt. Zusammengefasst liegt die Situation folglich vor, sobald entweder die Last der CPU einen Wert größer als 90 Prozent annimmt oder weniger als 1000MB RAM zur Verfügung stehen oder der Watchdog einen anderen Antwortcode zurückliefert als 200.

3.3.3. SitRS

Wie am Anfang des Kapitels beschrieben, sollen in SitOPT im Wesentlichen zwei Kernaufgaben durchgeführt werden: Die Erkennung von Situationen und die Anpassung von Workflow-basierten Anwendungen an Situationen. Die Erkennung von Situationen wird mit dem Dienst *SitRS* [HWS+15] durchgeführt. Die Architektur von SitRS besteht aus drei Schichten und wird in Abbildung 3.4 dargestellt. Die erste Schicht unten in der Abbildung beinhaltet Objekte mit Sensoren. Die zweite Schicht in der Mitte stellt das eigentliche SitRS System dar. Die dritte Schicht im oberen Bereich des Bildes besteht aus den erkannten Situationen.

Das SitRS System besteht im Wesentlichen aus zwei Komponenten: der *Resource Management Platform* und dem *Situation Recognition System*. Zudem ist ein Archiv zur Verwaltung von Situation Templates und ein anderes Archiv zur Verwaltung von Sensoren vorhanden. Die Sensoren können in der *Sensor Registry* registriert werden. Die Resource Management Platform beginnt nach der Registrierung eines Sensors die Daten der Sensoren mittels geeigneter Adapter zu extrahieren. Nach der Extraktion werden die Daten dem Situation Recognition System als Dienst bereitgestellt. Das Situation Recognition System ist zuständig für die Ausführung von Situation Templates, die im Archiv für Situation Templates gespeichert werden. Die Ausführung von Situation Templates im Situation Recognition System wird durch den Situation Registration Dienst gestartet. Hierfür wird die ID eines Situation Templates und die ID eines Dings benötigt. Im Gegenzug wird eine ID zur Überwachung der Ausführung generiert.

Es ist zu beachten, dass die Ausführung eines Situation Templates nicht nur einmal erfolgt, sondern kontinuierlich stattfindet. Wird ein Situation Template gestartet, werden die Sensordaten ununterbro-

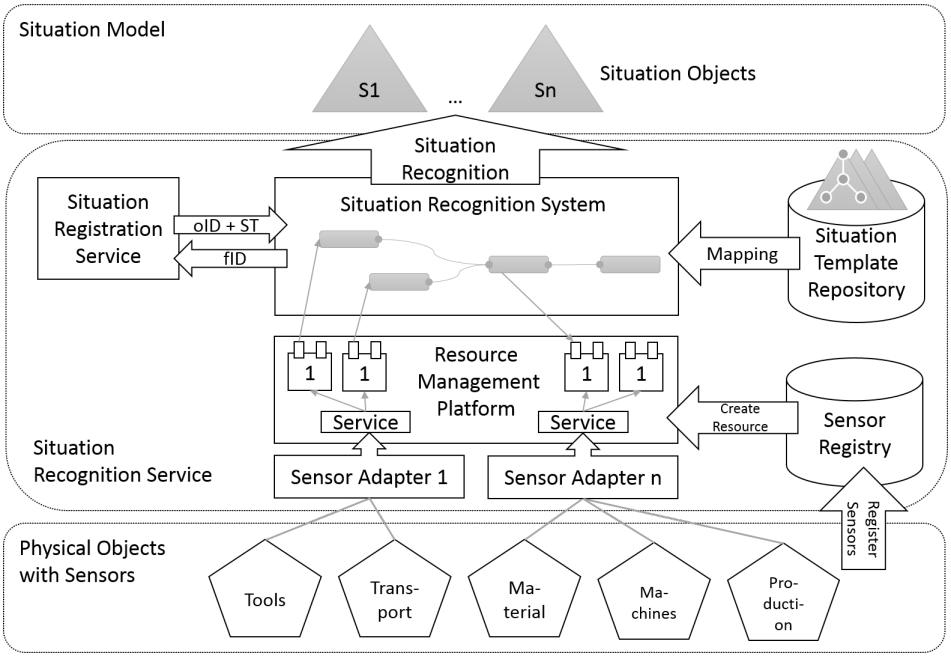


Abbildung 3.4.: SitRS - Architektur [HWS+15]

chen ausgewertet, bis die Ausführung wieder gestoppt wird. Es wird demnach bei jeder Auswertung festgestellt, ob die Situation vorliegt oder nicht. Die Auswertung der Sensoren erfolgt derzeit einmal pro Sekunde. Der Wert von einer Sekunde ist jedoch nur exemplarisch und kann je nach Einsatzgebiet angepasst werden.

4. Konzept des Situation Handlers

In diesem Kapitel wird das Konzept beschrieben, mit dem die Aufgabenstellung aus Kapitel 2 umgesetzt wird. Das Kapitel ist strukturiert wie folgt: Zunächst wird eine grobe Übersicht über die Funktionen des Situation Handlers und seine Interaktion mit dem Situationserkennungssystem (SES) gegeben (siehe Abschnitt 4.1). Als SES werden die SitOPT Komponenten bezeichnet, die für die Erkennung und Verwaltung von Situationen zuständig sind (siehe Kapitel 3). In den darauf folgenden Abschnitten wird die konzeptionelle Umsetzung des Situation Handlers näher erläutert. Dabei wird in 4.2 zunächst auf die Rolle des ER-Directories eingegangen, das zur Speicherung von Endpunktdaten und Notifikationsregeln dient. In Abschnitt 4.3 wird die Situationsbehandlung erläutert, wobei der Versand von Notifikationen und die Behandlung von Dienstaufrufen durch Workflows getrennt beschrieben werden. Abschließend wird auf das Pluginsystem (Abschnitt 4.4), die Interaktion mit dem SES (Abschnitt 4.5) und die Konfigurationsmöglichkeiten des Situation Handlers (Abschnitt 4.6) eingegangen.

4.1. Übersicht

Zunächst wird eine Übersicht über die Integration des Situation Handlers mit dem SES und anderen Komponenten gegeben. Die Übersicht wird in Abbildung 4.1 illustriert. Die in blau gehaltenen Bestandteile gehören zum Umfang dieser Arbeit. Hierbei handelt es sich um den Situation Handler selbst, sowie die zur Konfiguration erforderliche Oberfläche und die API. Diese dienen als Schnittstellen für den Nutzer, der im oberen Bereich des Bildes in schwarz dargestellt wird. Zudem greift der Situation Handler auf verschiedene Plugins zur Kommunikation, beziehungsweise zum Versand von Notifikationen zurück. Die Notifikationen werden durch Aktionen versandt, die durch die Plugins ausgeführt werden. Durch eine Aktion kann beispielsweise auch ein Workflow-Fragment aufgerufen werden. Das Verzeichnis auf der rechten Seite des Bildes dient zur Speicherung von Informationen über Workflow-Fragmente und Notifikationen. Bei den in orange gehaltenen Bestandteilen der Abbildung handelt es sich um externe Komponenten. Hierbei hat das Situationserkennungssystem im unteren Bereich des Bildes eine große Bedeutung. Es liefert dem Situation Handler Informationen über aktuell vorliegende Situationen und Änderungen von Situationen. Bei Situationsänderungen, die dem Situation Handler vom Situationserkennungssystem gemeldet werden, wird der Versand der dafür registrierten Notifikation ausgelöst. Die Interaktion des Situation Handlers mit Workflows wird auf der linken Seite des Bildes dargestellt. Der Workflow, beziehungsweise die Workflow Engine, sendet eine Anfrage an den Situation Handler. Dieser wählt anhand aktuell vorliegender Situationen ein passendes Workflow-Fragment (rechts oben im Bild) aus und startet die Ausführung des Workflow-Fragmentes. Nachdem das Workflow-Fragment vollständig ausgeführt wurde, schickt es dessen Antwort an den Situation Handler zurück. Dieser leitet die Antwort an den anfragenden Workflow

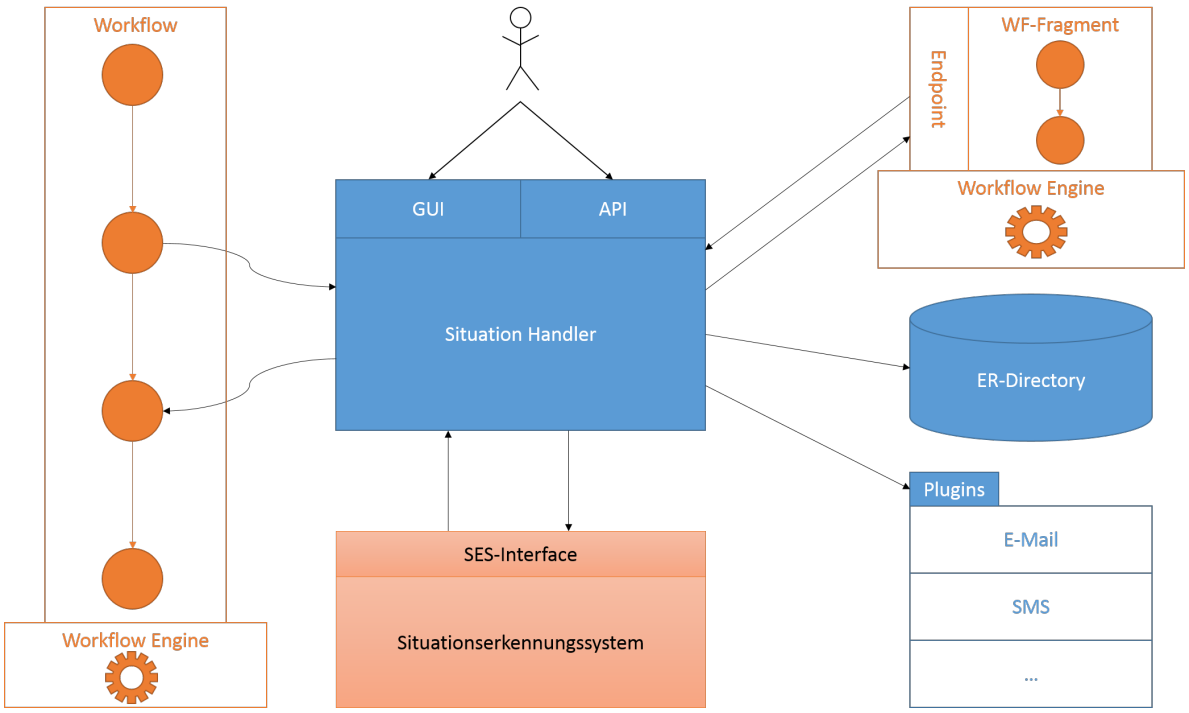


Abbildung 4.1.: Übersicht über die Integration des Situation Handlers.

weiter. Die Workflow-Fragmente entsprechen dabei dem SESE-Schema [KELU10]. Die Kommunikation zwischen den Komponenten wird in der vorliegenden Arbeit grundsätzlich asynchron umgesetzt. Eine synchrone Umsetzung ist jedoch in ähnlicher Weise möglich, was in der vorliegenden Arbeit aber nicht weiter behandelt wird. Eine genauere Beschreibung der Funktionalitäten des Situation Handlers wird in den folgenden Abschnitten gegeben.

4.2. Endpunkt- und Regelverzeichnis

Das Endpunkt- und Regelverzeichnis (**ER-Directory**) dient als Speicher für die Endpunkte der bereitgestellten Workflow-Fragmente, das heißt von Fragmenten, die auf einer Workflow Engine bereitgestellt wurden und somit über einen Endpunkt ausgeführt werden können. Außerdem speichert das ER-Directory die Regeln, nach denen Notifikationen versandt werden. Eine Beschreibung des Aufbaus von Regeln und Endpunkten wird in Abschnitt 4.3 gegeben. Für diese wird eine persistente Speicherung und effizienter Zugriff benötigt. Diese Eigenschaften werden durch das ER-Directory garantiert. Endpunkte und Notifikationsregeln müssen zudem dynamisch verändert werden können. Dementsprechend bietet das ER-Directory die Möglichkeit, dort abgelegte Endpunkte und Regeln zu modifizieren oder zu löschen. Außerdem können dem Verzeichnis neue Endpunkte und Regeln hinzugefügt werden. Eine Bearbeitung von Endpunkten und Regeln erfolgt in erster Linie über die API und die Benutzeroberfläche. GUI und API dienen dem Nutzer folglich als Schnittstelle zur Konfiguration des ER-Directories und damit des Situation Handlers. Der Situation Handler greift

bei Bedarf auf das ER-Directory zu, um Regeln und die Endpunkte der als Dienst bereitgestellten Workflow-Fragmente abzufragen.

4.3. Situationsbehandlung

Die Behandlung von Situationen wird auf zwei verschiedene Arten durchgeführt. Zum einen ist der Versand von Notifikationen bei der Änderung einer Situation möglich. Die Notifikationen werden über generische Aktionen versandt. Eine Aktion kann beispielsweise auch zum situationsabhängigen Aufruf eines Workflow-Fragments verwendet werden. Zum anderen werden Dienstaufrufe, die an den Situationen Handler übermittelt werden, abhängig von den aktuell vorliegenden Situationen an einen Endpunkt weitergeleitet, der eine der Situation angemessene Implementierung für diesen Dienstaufruf bietet. Beide Varianten werden im Folgenden vorgestellt.

Zunächst wird an dieser Stelle jedoch die Verwendung des Begriffes Situation erklärt, da sich die Bedeutung des Begriffes leicht zu der Verwendung in Kapitel 3 unterscheidet. Hier war eine Situation das Resultat der Ausführung eines Situation Templates auf einem Ding. Das Situation Template und das Ding wurden hierbei durch eine eindeutige ID identifiziert. Eine Situation lies sich durch die Kombination der IDs von Situation Template und Ding oder die ID der Situation identifizieren. In der vorliegenden Arbeit wird der Begriff der Situation nachfolgend anders verwendet. Eine Situation besteht hier aus dem Namen der Situation und einem Identifikator für das Objekt. Der Name der Situation entspricht dem Situation Template. Das Objekt entspricht einem Ding. Zur Identifikation des Objekts wird ebenfalls die ID verwendet. Eine Situation ist in der vorliegenden Arbeit demnach eindeutig durch einen Namen (dieser kann zum Beispiel der ID des Situation Templates entsprechen) und ein Objekt (der ID des Dinges), auf dem die Situation vorliegt, identifiziert. Es wird folglich ausgeschlossen, dass dieselbe Situation mehrfach auftritt und dabei unterschiedliches aussagt. Eine Situation kann jedoch zwei verschiedene Zustände besitzen: Entweder sie liegt vor oder sie liegt nicht vor. Es wird angenommen, dass eine Situation vorliegt, wenn vom SES gemeldet wurde, dass die Situation neu aufgetreten ist. Das Vorliegen dieses Zustandes wird solange angenommen, bis das SES meldet, dass die Situation nicht mehr vorliegt. Der Wechsel des Zustands einer Situation wird in der vorliegenden Arbeit auch als *Situationsänderung* bezeichnet.

Als Beispiel dient ein Szenario, in dem eine Situation mit dem Namen „zu heiß“ existiert, die auf einem Gerät mit einem Temperatursensor überwacht wird. Das SES meldet ein Auftauchen der Situation, wenn der Temperatursensor einen Wert von mehr als 90°C misst. In diesem Fall würde für die Situation der Zustand „liegt vor“ gelten. Wenn das SES meldet, dass die Situation verschwunden ist, gilt für die Situation der Zustand „liegt nicht vor“. Die Situation hat sich also geändert. Jedoch ist es möglich, dass das SES die Änderung erst mit einer gewissen Verzögerung erkennt oder das die Temperatur gesunken ist und dies mit einer Verzögerung erkannt wird. In diesem Fall würde die Situationsänderung erst mit einer Verzögerung erkannt werden.

4.3.1. Notifikationen

Ein Teil der Behandlung von Situationen besteht aus dem Versand von Notifikationen beziehungsweise der Ausführung von Aktionen. Dieser Teil der Behandlung wird in diesem Abschnitt beschrieben.

Hierfür werden zunächst Notifikationsregeln und Aktionen näher erläutert. Danach wird der Ablauf von diesem Teil der Situationsbehandlung erklärt.

Notifikationsregeln und Aktionen

Die Aktionen zum Versand von Notifikationen werden in Form von Notifikationsregeln verwaltet. Eine Notifikationsregel ist dabei aufgebaut wie folgt:

Situation —> Aktion 1, Aktion 2, ... , Aktion n

Eine Notifikationsregel entspricht somit einem vereinfachten Event-Condition-Action-Konzept, das auf Situationen als Zusammenfassung von „Event“ und „Condition“ definiert ist. Da Situationen eindeutig sind, kann nur eine Regel pro Situation existieren. Es sind jedoch beliebig viele Aktionen pro Regel erlaubt, das heißt dies stellt im Allgemeinen keine Einschränkung dar. Regeln können vom Nutzer erstellt und im ER-Directory abgelegt werden. Eine Regel kann dort anhand der aufgetretenen Situation abgefragt werden, das heißt bei Eingabe einer Situation liefert das Verzeichnis die Regel und damit alle Aktionen zurück, die mit dieser Situation assoziiert sind.

Eine Aktion steht allgemein für die Ausführung einer beliebigen Aufgabe - die eigentliche Intention ist jedoch, dass durch eine Aktion eine beliebige Nachricht versandt wird. Dies könnte beispielsweise eine E-Mail an eine bestimmte Person sein, um diese über die Situationsänderung zu benachrichtigen. Ein weiteres Beispiel für eine Notifikation wäre der Aufruf eines Workflow-Fragments, um dieses beim Auftreten oder Verschwinden einer Situation zu starten. Grundsätzlich gibt es bei dem Konzept der Aktion keine Einschränkungen, was eine Aktion bewirken kann oder darf. Es könnte zum Beispiel auch lediglich eine lokale Berechnung durchgeführt werden. Zur Ausführung von Aktionen werden Plugins verwendet, die durch das Pluginsystem verwaltet werden. Das Pluginsystem wird in Abschnitt 4.4 beschrieben. Wenn der Nutzer eine Aktion definiert, muss dieser sicherstellen, dass ein Plugin existiert, welches die beschriebene Aktion ausführen kann. Eine Aktion definiert sich über die folgenden Eigenschaften:

Plugin
: Eine eindeutige Beschreibung (ID), welchem Plugin die Aktion zugeordnet ist, beziehungsweise von welchem Plugin die Aktion behandelt werden muss.

Empfänger
: Der Empfänger der Nachricht, die bei der Aktion versandt wird. Für ein E-Mail-Plugin wäre dies die E-Mail-Adresse.

Nachricht
: Die Nachricht in einem beliebigen Format, das der Versandart entspricht. Wenn beispielsweise eine E-Mail versandt werden soll, würde hier der Text der E-Mail angegeben werden.

Optionale Parameter
: Optionale Parameter, die durch das Plugin spezifiziert werden. Zum Versand einer E-Mail könnte hier der Betreff der E-Mail angegeben werden. Dieser Parameter muss jedoch von einem entsprechenden E-Mail-Plugin vorgegeben werden.

Ausführungszeitpunkt
: Eine Aktion kann entweder beim Auftreten, Verschwinden oder generell bei der Änderung der Situation ausgeführt werden.

Eine genauere Beschreibung, wie diese Eigenschaften von einem Plugin verarbeitet werden, wird in Abschnitt 4.4 gegeben. Allerdings ist zu beachten, dass für die Verarbeitung einer Aktion bei der Situationsbehandlung nur ein Plugin spezifiziert sein muss, damit die Aktion zugeordnet werden kann. Die anderen Eigenschaften sind vollständig vom verwendeten Plugin abhängig. Da die Intention betont werden soll, dass eine Aktion den Versand einer Nachricht beschreibt, besitzen Aktionen immer die Eigenschaften *Empfänger* und *Nachricht*. Diese beiden Felder wären ansonsten nicht notwendig, da eine beliebige Anzahl von optionalen Parametern spezifiziert werden kann. *Empfänger* und *Nachricht* könnten somit auch als optionale Parameter definiert werden.

Ablauf des Notifikationsversandes

Der Versand von Notifikationen beziehungsweise die Ausführung von Aktionen läuft in mehreren Schritten ab, bei denen verschiedene Komponenten involviert sind. Die Interaktion zwischen den verschiedenen Komponenten wird in Abbildung 4.2 dargestellt. In der Abbildung wird dieselbe farbliche Kodierung verwendet wie zuvor. Externe Komponenten wie das SES werden orange dargestellt. Komponenten, die zu dieser Arbeit gehören, in blau. Die Plugins des Situation Handlers werden in gelb dargestellt - diese stammen teilweise aus dieser Arbeit, es kann sich aber auch um externe Komponenten handeln. Die Situationsbehandlung wird stets durch die Änderung einer Situation ausgelöst. Situationsänderungen werden dem Situation Handler stets vom SES gemeldet. Das SES wird in Abbildung 4.2 unten im Bild dargestellt. Wenn das SES eine Situationsänderung erkennt, wird unmittelbar eine Benachrichtigung an den Situation Handler versandt. Dieser schickt eine Anfrage an das ER-Directory (links im Bild), um die gespeicherten Aktionen abzufragen. Die Aktionen werden daraufhin ausgeführt, wozu das Pluginsystem verwendet wird. Die Plugins sind in den Situation Handler integriert, wie es in Abbildung 4.2 dargestellt ist.

Algorithmus 4.1 fasst den Ablauf zur Situationsbehandlung als Pseudocode zusammen. Die genauen Abläufe werden im Folgenden erläutert. Die Prozedur wird gestartet, wenn durch das SES eine Situationsänderung gemeldet wird. Hierbei wird die Situation und ihr neuer Zustand mitgeteilt. Um die erforderlichen Aktionen abzufragen, muss eine Anfrage an das ER-Directory gestellt werden. Dies wird in Zeile 2 des Algorithmus beschrieben. Hierbei erfolgt die Abfrage einer Notifikationsregel. Als Eingabe wird hierzu die Situation verwendet. Das Ergebnis ist eine Liste der Aktionen, die in der Regel dieser Situation zugeordnet wurden. Da Situationen eindeutig definiert sind, werden nur die Aktionen zurückgeliefert, die genau dieser Situation zugeordnet sind. Da Aktionen einen Ausführungszeitpunkt festlegen, dürfen nur die Aktionen ausgeführt werden, die für den vorliegenden Zustandswechsel definiert wurden. Wenn eine Situation zum Beispiel verschwindet, werden nur die Aktionen ausgeführt, bei denen als Ausführungszeitpunkt das Verschwinden der Situation oder

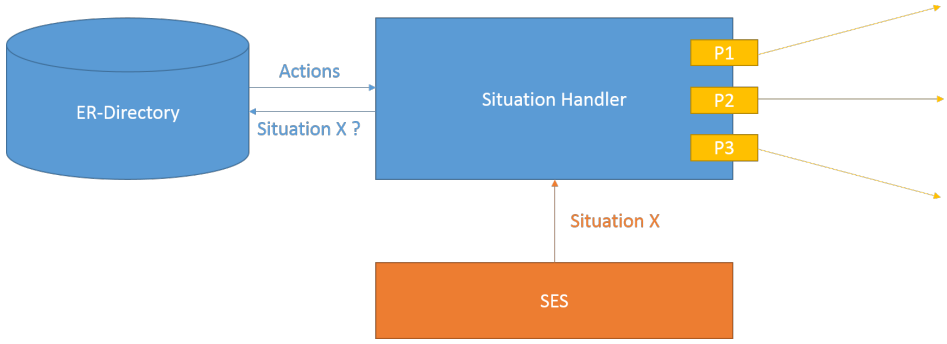


Abbildung 4.2.: Interaktion der Komponenten bei einer Situationsänderung

Algorithmus 4.1 Situationsbehandlung: Notifikationen

```
1:  procedure onSituationChange(situation)
2:      actions ← queryDirectory(situation)          // get actions associated with situation
3:      for all (action ∈ actions) do
4:          plugin ← getPlugin(action.Plugin)
5:          if (exists(plugin)) then
6:              invokePlugin(plugin, action)
7:          else
8:              throw(new Error(plugin))
9:          end if
10:     end for
11: end procedure
```

die Änderung der Situation definiert ist. Falls als Zeitpunkt das Auftreten der Situation angegeben ist, wird die Aktion in diesem Fall nicht ausgeführt. Der Ausführungszeitpunkt wird dabei vom Situation Handler nicht berücksichtigt, sondern bei der Anfrage in Zeile 2. Bei der Anfrage werden die Aktionen herausgefiltert, bei denen der Ausführungszeitpunkt nicht der aktuellen Situationsänderung entspricht. Die Filterung findet somit im ER-Directory statt. Dadurch wird auch vermieden, dass eine doppelte Überprüfung der Aktionen stattfindet - Aktionen werden in einem Durchlauf auf die Situation (Regel) und den Ausführungszeitpunkt geprüft.

Nach der Abfrage der Aktionen liegen dem Situation Handler genau die Aktionen vor, die ausgeführt werden müssen. Die Ausführung der Aktionen wird in Zeile 3 bis 10 beschrieben. Um eine Aktion ausführen zu können, wird ein Plugin benötigt, welches dazu in der Lage ist. Um dieses Plugin zu bestimmen, wird in Zeile 4 eine Anfrage an das Pluginsystem gestellt. Um das benötigte Plugin zu identifizieren, wird die Beschreibung des Plugins verwendet, die der Aktion beigefügt ist. Es ist jedoch möglich, dass das angeforderte Plugin nicht existiert, da der Nutzer beliebige Aktionen spezifizieren kann und Plugins auch gelöscht werden können. Bei der Definition von Aktionen kann nicht immer sichergestellt werden kann, dass ein existierendes Plugin angegeben wird (siehe Abschnitte 4.4 und 4.6). Daher findet in Zeile 5 eine Überprüfung statt, ob das angegebene Plugin gefunden wurde. Die Ausführung der Aktion erfolgt letztlich in Zeile 6 durch die Verwendung des Plugins. Hierzu wird

dem Plugin die Aktion übergeben. Wie genau ein Plugin eine Aktion ausführt, wird in Abschnitt 4.4 näher beschrieben. Falls kein Plugin gefunden wird, wird ein Fehler ausgelöst, beziehungsweise die Aktion wird nicht ausgeführt.

4.3.2. Situationsabhängige Dienstaufrufe

Der zweite Teil der Situationsbehandlung besteht aus der situationsabhängigen Ausführung von Dienstaufrufen. In der vorliegenden Arbeit liegt der Fokus auf dem Aufruf von Diensten durch Workflows und der Implementierung von Diensten durch Workflow-Fragmente.

Situationsbewusste Endpunkte

Die Grundidee bei den situationsabhängigen Dienstaufrufen besteht darin, dass Workflows den Situation Handler als Endpunkt für Dienstaufrufe verwenden. Dienstaufrufe werden auch als *Workflow-Operationen* bezeichnet. Der Situation Handler untersucht vorliegende Situationen und wählt dementsprechenden eine geeignete Implementierung für diese Operation aus. Eine Operation wird durch einen anderen Workflow (Workflow-Fragment nach dem SESE-Schema [KELU10]) umgesetzt. Dementsprechend übernimmt der Situation Handler in diesem Fall die Rolle eines Service Busses: Nachdem der Situation Handler eine Anfrage erhält, führt er in einer Registry einen Lookup durch, um eine geeignete Implementierung der Anfrage zu finden. Der Situation Handler übernimmt in diesem Fall auch den Aufruf des Dienstes. Die Funktion der Registry wird vom ER-Directory übernommen. Das ER-Directory speichert die Adressen, an denen die verschiedenen Implementierungen einer Operation zur Verfügung stehen, als **situationsbewusste Endpunkte** ab. Ein Endpunkt besitzt hier folgende Eigenschaften:

Situation Die Situation, in der dieser Endpunkt verwendet werden kann.

Operation Die Operation, die dieser Endpunkt implementiert.

Adresse Die Adresse des Endpunktes, der einen Dienst zur Verfügung stellt.

Eine Situation besteht aus dem Namen der Situation und einem Objekt, wie zu Beginn von Kapitel 4 beschrieben. Eine Operation wird durch den Namen der Operation und einem zusätzlichen Kennzeichner (**Qualifier**) definiert. Der Qualifier kann grundsätzlich beliebig gewählt werden. Seine Aufgabe besteht darin, die Operation zusammen mit dem Operationsnamen eindeutig zu identifizieren. Es wäre damit möglich, dass mehrere Endpunkte denselben Namen für verschiedene Operationen verwenden. Der Qualifier sorgt in diesem Fall dafür, dass die zu wählende Operation eindeutig bestimmt werden kann. Eine Operation bestehend aus einem Operationsnamen und Qualifier ist daher eindeutig und vergleichbar mit dem Konzept der qualifizierten Namen (*QName*) bei XML/WSDL. Der Operationsname sowie der Qualifier werden dem Situation Handler bei dem Dienstaufruf durch einen Workflow mitgeteilt. Die Umsetzung der Operation durch den Endpunkt erfolgt in der Regel durch die Ausführung eines Workflow-Fragments. Die Adresse des Endpunktes ist beispielsweise eine URL, die zur Verwendung des Endpunktes dient. Unter einem Endpunkt wird folglich ein Workflow-Fragment verstanden, das an einer bestimmten Adresse bereitgestellt wurde und das zur Ausführung einer bestimmten Operation in einer ausgewählten Situation geeignet ist.

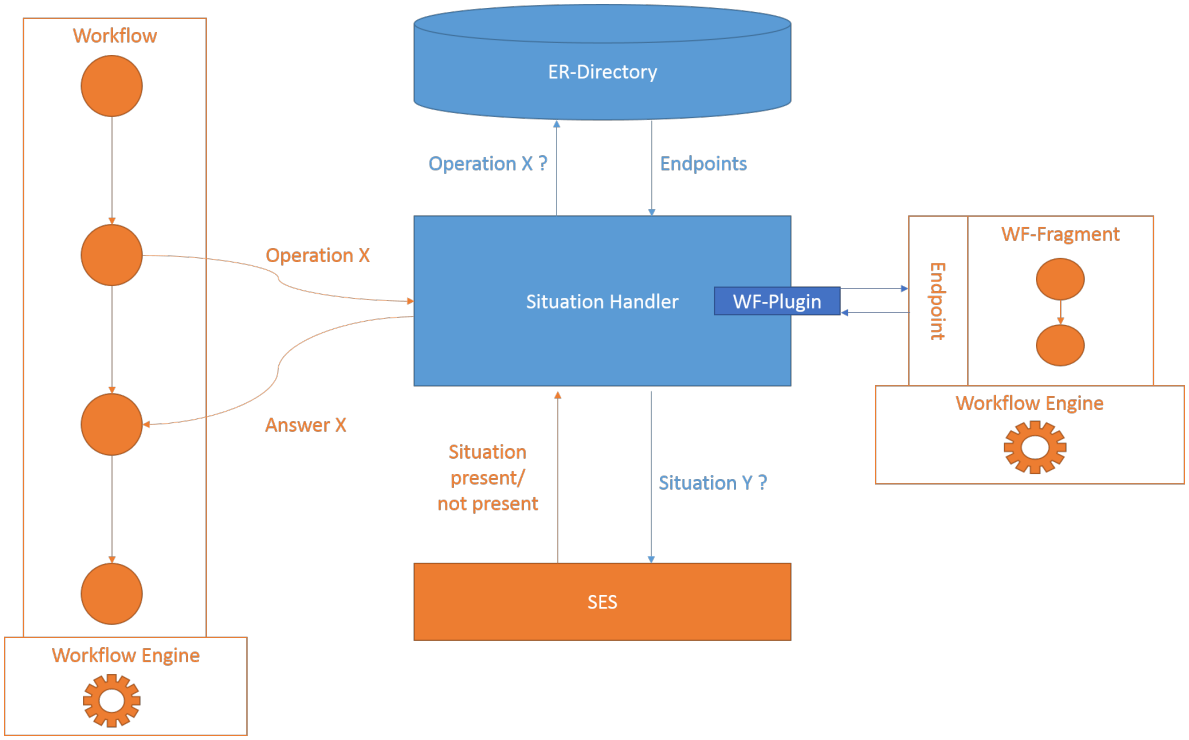


Abbildung 4.3.: Grafische Darstellung der Auswahl eines Operationsendpunktes.

Ablauf der Endpunktauswahl

Der grundsätzliche Ablauf zur Auswahl eines Endpunktes wird in den Algorithmen 4.2 und 4.3 dargestellt, die Kommunikation der verschiedenen Komponenten in Abbildung 4.3. Erneut werden in der Abbildung externe Komponenten in orange gezeigt, während die Teile dieser Arbeit in blau gehalten sind. Die Kommunikation zwischen den Komponenten wird durch die Pfeile gezeigt. Auch bei den Pfeilen wird eine farbliche Kodierung verwendet, um zu zeigen, von welcher Komponente die Kommunikation ausgeht.

Es gibt folgende Kommunikationsbeziehungen: Der Workflow auf der linken Seite des Bildes übermittelt dem Situation Handler eine Anfrage zu einem Operationsaufruf sowie die dazu nötigen Parameter. Der Situation Handler verwendet das ER-Directory zur Abfrage von Endpunkten, die diese Operation anbieten und das SES zur Abfrage von aktuell vorliegenden Situationen. Zum Aufruf eines Workflow-Fragments (rechts im Bild) wird ein eigenes Plugin verwendet. Dieses Plugin kann auch zum Versand von Notifikationen und somit zum situationsabhängigen Aufruf von Workflow-Fragmenten verwendet werden.

Der genaue Ablauf bei der Situationsbehandlung wird durch Algorithmus 4.2 beschrieben. Anders als bei der situationsabhängigen Ausführung von Operationen ist der Startpunkt hier stets ein Operationsaufruf (Verwendung eines Dienstes) durch einen Workflow. Der erste Schritt zur Verarbeitung dieses Aufrufes ist eine Anfrage an das ER-Directory, die im Algorithmus in Zeile 2 ausgeführt wird.

Algorithmus 4.2 Situationsbehandlung: Endpunkte

```
1: procedure onRequestReceived(operation, client)
2:     endpoints ← queryDirectory(operation)        // get endpoints that implement operation
3:     selectedEndpoint ← chooseEndpoint(endpoints)
4:     result ← invoke(selectedEndpoint)
5:     forwardAnswer(result, client)
6: end procedure
```

Algorithmus 4.3 Situationsbehandlung: Auswahl eines Endpunktes

```
1: procedure chooseEndpoint(endpoints)
2:     selectedEndpoint ← null
3:     while (selectedEndpoint = null) do
4:         currentEndpoint ← endpoints.Next()
5:         if (situationPrevalent(currentEndpoint.Situation)) then
6:             selectedEndpoint ← currentEndpoint
7:         end if
8:     end while
9:     return selectedEndpoint
10: end procedure
```

Der Verzeichnis liefert auf diesen Aufruf einen oder mehrere Endpunkte zurück, die diese Operation implementieren. Es können mehrere Endpunkte zurückgeliefert werden, da es situationsabhängig verschiedene Implementierungen für eine Operation geben kann. Diese werden an verschiedenen Endpunkten zur Verfügung gestellt. Aus der Auswahl an Endpunkten muss ein Endpunkt für den Aufruf ausgewählt werden. Die Auswahl wird in Algorithmus 4.3 beschrieben. Dabei wird in Zeile 3 bis 8 solange die Liste der Endpunkte durchsucht, bis ein passender Endpunkt gefunden wurde. Hierzu wird in Zeile 5 überprüft, ob die Situation des aktuell geprüften Endpunktes vorliegt. Ist dies der Fall, wird dieser Endpunkt ausgewählt und die Suche kann beendet werden. Der ausgewählte Endpunkt wird zurückgeliefert. Anschließend wird in Algorithmus 4.2, Zeile 4 der Endpunkt beziehungsweise die Operation, die dort implementiert wird, aufgerufen. Das Ergebnis des Aufrufs wird abschließend an den Klienten weitergeleitet. Mit diesem Vorgang ist die Behandlung der Workflow-Operation (Dienstaufruf) abgeschlossen.

Annahmen

Das in Algorithmus 4.3 beschriebene Vorgehen zur Auswahl eines Endpunktes setzt mehrere Annahmen voraus:

Annahme 1 Wenn der Situation Handler eine Operation ausführen soll, gibt es für diese mindestens einen Endpunkt mit einer zum Zeitpunkt der Ausführung vorliegenden Situation. Das heißt Algorithmus 4.3 findet immer ein Ergebnis oder es wird ein Fehler an den aufrufenden Workflow zurückgesendet.

Annahme 2 Es gibt höchstens einen Endpunkt für eine Operation, beziehungsweise es kann über die vorliegenden Situationen eindeutig aufgelöst werden, welcher Endpunkt verwendet werden soll. Falls mehr als ein Endpunkt in Frage kommt, kann eine beliebige Auswahl aus diesen Endpunkten getroffen werden. Dies hat keinen Einfluss auf das Ergebnis, da die Semantik der Dienste aller Endpunkte identisch ist.

Annahme 1 ist valide und kann ohne große Einschränkungen vorausgesetzt werden. Wenn es nicht mindestens einen Endpunkt gibt, kann davon ausgegangen werden, dass es einen Fehler beim Entwurf des Workflows gab oder ein Endpunkt nicht registriert wurde. Es gibt für den Situation Handler keine Möglichkeit, einen Endpunkt zu erraten, falls kein Endpunkt gefunden wird. Daher kann in diesem Fall ein Fehler gemeldet werden. Der Entwickler des Workflows muss dafür sorgen, dass die verschiedenen Situationen durch Endpunkte abgedeckt sind und mindestens ein Endpunkt für jede sinnvolle Kombination aus Situation und Operation zur Verfügung steht.

Es ist zu bemerken, dass nicht für jede existierende Kombination aus Operation und Situation ein Endpunkt angegeben werden muss. Nur für die Kombinationen, die aus dem Kontext heraus sinnvoll sind, muss ein Endpunkt definiert werden. Der Entwickler der Workflows und der Situation Templates muss sich bewusst sein, welche Kombinationen sinnvoll sind und welche nicht. Dies beinhaltet ebenso die Überlegung, welche Situationen für welche Objekte auftreten können/werden. Die Modellierung der Workflows wird dadurch maßgeblich beeinflusst.Ein kurzes Beispiel verdeutlicht dies: Angenommen es wurden zwei Situation Templates entworfen. Beide werden zur Überwachung von Gerätesensoren in einer Fabrik verwendet. Template 1 dient zur Überwachung eines Fließbandes und liefert die Situation „Fließband klemmt“ beziehungsweise die gegenteilige Situation, dass das Fließband nicht klemmt. Template 2 überwacht die Temperatursensoren einer Säge und liefert die Situation „Säge zu heiß“. Beide überwachten Geräte haben in diesem Beispiel nichts miteinander zu tun. Als Operationen stehen für das Fließband „starte Fließband“ und für die Säge „säge Holz“ zur Verfügung. Offensichtlich hat die Situation „Fließband klemmt“ nichts mit Operation „säge Holz“ zu tun. Es ist folglich unwahrscheinlich, dass eine Implementierung von „säge Holz“ existiert, die das Fließband repariert. Daher muss keine Kombination der Situation „Fließband klemmt“ und der Operation „säge Holz“ angegeben werden.

Annahme 2 ist hingegen nicht allgemein gültig. Diese impliziert, dass es nur eine sinnvolle Kombination von Situation und Operation gibt, beziehungsweise das nur eine der Situationen gleichzeitig auftreten kann. Ein einfaches Beispiel zeigt, dass dies keine sinnvolle Annahme ist und das diese zu starke Einschränkungen mit sich bringen würde. Als Umgebung wird in diesem Beispiel erneut eine Fabrik angenommen. Dabei wird als zu überwachende Maschine nur das Fließband berücksichtigt. Für dieses werden die Situationen „Fließband klemmt“ und „Fließband zu heiß“ überwacht. Beide Situationen können gleichzeitig auftreten. Es existiert nur eine Operation und zwei verschiedene Endpunkte, die diese Operation implementieren. Ein Endpunkt berücksichtigt die Situation „Fließband zu heiß“ und ein anderer Endpunkt die Situation „Fließband klemmt“. Wenn beide Situationen gleichzeitig auftauchen ist keine eindeutige Auflösung mehr möglich. Egal welcher der Endpunkte gewählt wird, bleibt eine Situation unberücksichtigt. Dies würde sehr wahrscheinlich zu fehlerhaftem Verhalten führen. Eine Lösung dafür wird im nächsten Abschnitt vorgestellt.

4.3.3. Workflow-Behandlung mit Auflösung von Mehrdeutigkeiten

Am Ende des vorigen Abschnitts wurde gezeigt, dass die dort vorgeschlagene Behandlung von Dienstaufrufen problematisch ist, wenn Mehrdeutigkeiten entstehen, die nicht eindeutig aufgelöst werden können. Ein vollständiges Verbot von Mehrdeutigkeiten wäre jedoch eine zu starke Einschränkung. In den folgenden Abschnitten wird eine Lösung für das Problem vorgestellt, bei der möglichst wenige Einschränkungen definiert werden müssen. Hierzu werden zunächst mehrere Anwendungsszenarien entworfen, bei denen Mehrdeutigkeiten entstehen können. Für die Probleme dieser Szenarien werden Lösungsmöglichkeiten für die dortigen Probleme entwickelt. Anschließend werden neue Annahmen getroffen und das bisherige Modell zur Auswahl eines Endpunktes entsprechend erweitert.

Szenarien mit Mehrdeutigkeit

Die verschiedenen Szenarien werden in Abbildung 4.4 dargestellt. Die Abbildung stellt insbesondere die Endpunkte dar und auf welche Situationen, Objekte und Workflow-Fragmente sich diese beziehen. Eine textuelle Beschreibung der Szenarien, die die Problematik des jeweiligen Szenarios und die gewünschte Lösung darlegt, folgt an dieser Stelle.

Szenario 1 - Gleichwertige Endpunkte

Beschreibung

Auf mehreren gleichartigen Objekten, die dieselbe Operation unterstützen, wird dasselbe Situation Template verwendet. Es existieren beispielsweise fünf verschiedene Sägen, die alle die Operation „säge Holz“ unterstützen und deren Verwendung das identische Resultat nach sich zieht. Auf allen fünf Sägen wird ein Template verwendet, um eine Situation mit dem Namen „zu heiß“ zu erkennen. Es werden daher fünf verschiedene Endpunkte angegeben (für jede Säge einen), die die Operation „säge Holz“ in der Situation „zu heiß“ unterstützen.

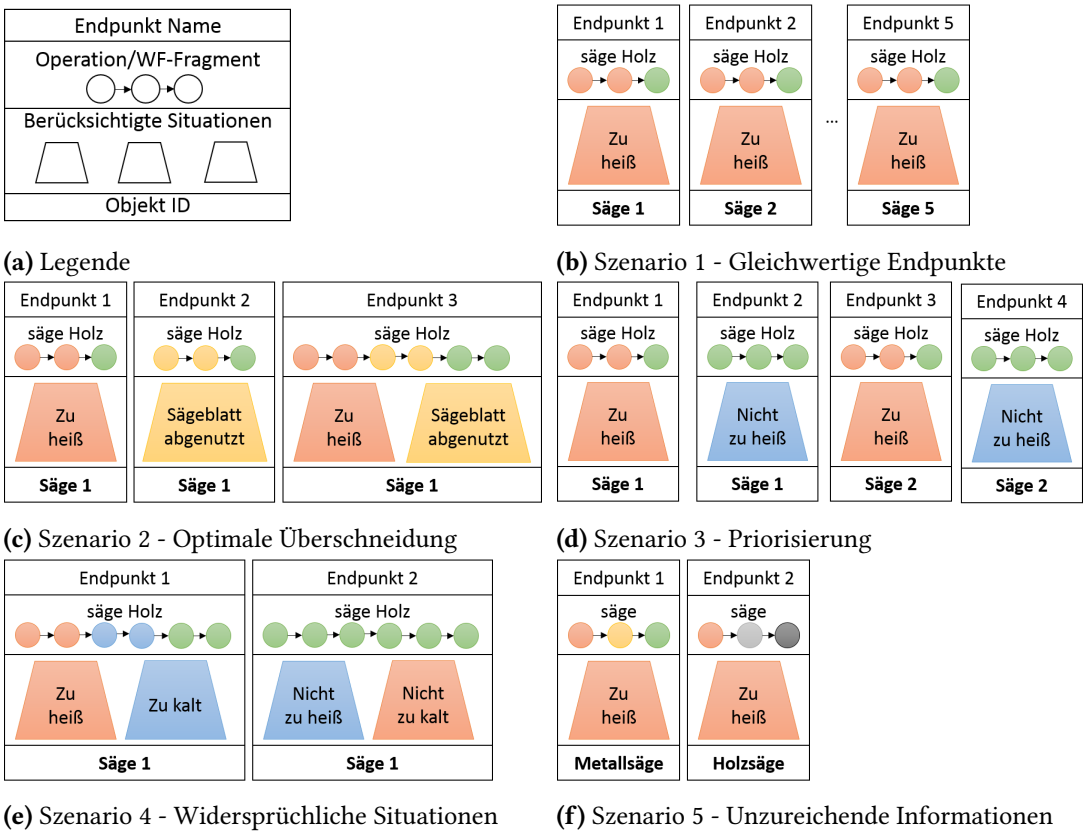
Gewünschtes Ergebnis

Es spielt in diesem Szenario keine Rolle, welcher der Endpunkte ausgewählt wird, da alle dieselbe Operation mit dem identischen Resultat ausführen. Es kann daher ein beliebiger Endpunkt ausgewählt werden, sofern die entsprechende Situation „zu heiß“ erfüllt ist.

Szenario 2 - Optimale Überschneidung

Beschreibung

Es existieren eine oder mehrere verschiedene Sägen, die situationsabhängig die Operation „säge Holz“ unterstützen. Auf einer Säge werden zwei verschiedene Templates zur Überwachung verwendet. Dabei wird durch Template 1 die Situation „zu heiß“ erkannt und durch Template 2 die Situation „Sägeblatt abgenutzt“. Für die verschiedenen Kombinationen der Situationen werden verschiedene Endpunkte angegeben.



(a) Legende

(b) Szenario 1 - Gleichwertige Endpunkte

(c) Szenario 2 - Optimale Überschneidung

(d) Szenario 3 - Priorisierung

(e) Szenario 4 - Widersprüchliche Situationen

(f) Szenario 5 - Unzureichende Informationen

Abbildung 4.4.: Verschiedene Szenarien mit mehreren Endpunkten zur Auswahl

Gewünschtes Ergebnis

Es wird genau der Endpunkt ausgewählt, der im Vergleich zu den anderen Endpunkten die meisten Übereinstimmungen mit den real vorliegenden Situationen aufweist. Wenn nur die Situation „zu heiß“ vorliegt, soll ein anderer Endpunkt ausgewählt werden, als wenn die Situationen „zu heiß“ und „Sägeblatt abgenutzt“ gleichzeitig vorliegen.

Szenario 3 - Priorisierung

Beschreibung

Es existieren zwei Sägen, die situationsabhängig die Operation „säge Holz“ unterstützen. Auf beiden Sägen wird ein Situation Template verwendet, das die Situation „zu heiß“ überwacht. Es werden für jede Säge zwei verschiedene Endpunkte angegeben. Ein Endpunkt unterstützt die Situation „zu heiß“ und leitet vor der Durchführung der Sägeoperation eine Kühlungsmaßnahme ein. Der andere Endpunkt unterstützt das Gegenteil von „zu heiß“, das heißt die Situation darf explizit nicht vorliegen. Hier beginnt die Sägeoperation sofort und ohne eine Kühlungsmaßnahme. Beide Sägen sind unabhängig voneinander, das heißt auf beiden Sägen können verschiedene Situationen vorliegen.

Gewünschtes Ergebnis

Liegen auf beiden Sägen die gleiche Situationen vor, wird eine beliebige Säge ausgewählt. Falls verschiedene Situationen bestehen, soll die Säge verwendet werden, bei der die Situation „nicht zu heiß“ vorliegt, da dort keine Kühlungsmaßnahme erforderlich ist und die Sägeoperation somit schneller durchgeführt werden kann. Es soll folglich eine Priorisierung von Endpunkten möglich sein.

Szenario 4 - Widersprüchliche Situationen

Beschreibung

Es existiert eine Säge, die situationsabhängig die Operation „säge Holz“ unterstützt. Auf der Säge wird ein Template verwendet, das die Situation „zu heiß“ überwacht und eines, das die Situation „zu kalt“ überprüft. Beide Situationen schließen sich gegenseitig aus. Die Abwesenheit der einen Situation führt aber nicht zwingend zum Vorliegen der anderen. Wenn die Säge nicht zu heiß ist, bedeutet dies nicht automatisch, dass sie zu kalt ist. Es werden zwei Endpunkte angegeben: Ein Endpunkt deckt die Situationen ab, dass die Säge entweder zu heiß oder zu kalt ist, der andere, dass die Säge Normaltemperatur besitzt und keine der beiden Situationen vorliegt.

Gewünschtes Ergebnis

Es wird eine Auswahl entsprechend den vorliegenden Situationen getroffen. Hierfür ist es erforderlich, dass bei der Auswahl auch sich ausschließende Situationen angegeben und ausgewertet werden können.

Die eben beschriebenen Szenarien eins bis vier decken alle Möglichkeiten ab, die in der vorliegenden Arbeit explizit behandelt werden.

Ablauf mit Auflösung der Mehrdeutigkeiten

Um die beschriebenen Szenarien lösen zu können, muss das Vorgehen zur Auswahl eines Endpunktes verändert werden. Zudem muss das Modell des Endpunktes angepasst werden. Ein Endpunkt ist von nun an definiert wie folgt:

Situationen Liste Einem Endpunkt sind nun mehrere Situationen zugeordnet.

Operation Die Operation, die dieser Endpunkt umsetzt (unverändert).

Adresse Die Adresse, an der dieser Endpunkt zur Verfügung steht (unverändert).

Für eine Situation kann außerdem festgelegt werden, ob das Vorliegen oder das Nichtvorliegen der Situation geprüft werden soll. Damit ein Endpunkt verwendet werden darf, müssen alle Situationen, die dem Endpunkt zugeordnet sind, den vorgegebenen Zustand besitzen. Allerdings kann eine Situation als *optional* markiert werden. In dem Fall kann ein Endpunkt mit dieser Situation umgehen, es ist aber nicht zwingend erforderlich, dass die Situation vorliegt.

Zur Bestimmung des Endpunktes wird Algorithmus 4.2 verwendet. Zur Auswahl des Endpunktes wird allerdings nicht mehr die Prozedur `chooseEndpoint` aus Algorithmus 4.3 verwendet, sondern

Algorithmus 4.4 Situationsbehandlung: Auswahl eines Endpunktes mit Auflösung von Mehrdeutigkeiten

```
1:  procedure chooseAmbiguousEndpoint(endpoints)
2:      selectedEndpoint ← null
3:      bestScore ← −1
4:      for all (currentEndpoint ∈ endpoints) do
5:          currentScore ← 0
6:          for all (currentSituation ∈ currentEndpoint.Situations) do
7:              if (situationPrevalent(currentSituation) = currentSituation.State) then
8:                  if (optional(currentSituation)) then
9:                      currentScore ← currentScore + 1
10:                 else
11:                     currentScore ← currentScore + 2
12:                 end if
13:             else
14:                 if (Not optional(currentSituation)) then
15:                     currentScore ← −∞
16:                     skipEndpoint(currentEndpoint)          // Check next endpoint
17:                 end if
18:             end if
19:         end for
20:         if (currentScore > bestScore) then
21:             bestScore ← currentScore
22:             selectedEndpoint ← currentEndpoint
23:         end if
24:     end for
25:     return selectedEndpoint
26: end procedure
```

die Prozedur `chooseAmbiguousEndpoint` aus Algorithmus 4.4. Der Algorithmus unterscheidet sich deutlich von Algorithmus 4.3. Grundsätzlich werden in diesem Algorithmus alle Endpunkte untersucht, die für eine Operation in Frage kommen. Dabei wird für jeden Endpunkt eine Bewertung erstellt. Der Endpunkt mit der höchsten Bewertung wird ausgewählt. Bei mehreren Endpunkten mit derselben Bewertung wird ein beliebiger Endpunkt aus dieser Menge ausgewählt. Die Bewertung der Endpunkte geschieht im Algorithmus in den Zeilen 4 - 24. Dabei wird in Zeile 7 für jede einzelne Situationen eines Endpunktes geprüft, ob diese denselben Zustand besitzt, wie im Endpunkt vorgegeben. Das heißt es wird abgefragt, ob die Situation vorliegt oder ob sie nicht vorliegt. Das Ergebnis der Abfrage wird mit der Vorgabe des Endpunktes verglichen. Wenn sich die Situation im vorgegebenen Zustand befindet, wird die Bewertung des Endpunktes erhöht. Hierbei wird in Zeile 8 unterschieden, ob es sich um eine optionale Situation handelt oder nicht. Die Bewertung wird weniger stark erhöht, wenn es sich um eine optionale Situation handelt. Falls der aktuelle Zustand der Situation nicht erfüllt ist, muss geprüft werden, ob es sich um eine optionale Situation handelt (Zeile 14). Handelt es sich nicht um eine optionale Situation, wird der Endpunkt aus der Auswahl entfernt und kann nicht mehr gewählt werden. Nachdem alle Situationen des Endpunktes geprüft wurden, wird die Bewertung des

Endpunktes in Zeile 20 mit dem Endpunkt mit der aktuell höchsten Bewertung verglichen. Ist die Bewertung des aktuellen Endpunktes größer als die bisher höchste Bewertung, wird der aktuelle Endpunkt zum derzeit besten Kandidaten erklärt.

Zusätzlich zu dem in Algorithmus 4.4 beschriebenen Verhalten gibt es die Möglichkeit, bei mehreren Endpunkten mit derselben (maximalen) Punktzahl den Nutzer entscheiden zu lassen, welcher der Endpunkte ausgewählt werden soll. Hierfür muss in der Anfrage des Workflows festgelegt werden, wer die Entscheidung trifft. Dazu muss eine Möglichkeit angegeben werden, den Entscheider zu kontaktieren. Im Prototypen des Situation Handlers wird der Entscheider über eine Push-Nachricht auf ein Smartphone kontaktiert. Eine genauere Beschreibung dieses Vorgangs erfolgt in Abschnitt 5.3.2. Wird kein Entscheider für die Anfrage festgelegt, wird ein zufälliger Endpunkt aus den Kandidaten mit der höchsten Punktzahl ausgewählt.

Behandlung der Szenarien

Mit dem neuen Endpunktmodell und dem neuen Ablauf zur Auswahl eines Endpunktes, wird die gewünschte Lösung für sämtliche Szenarien aus dem vorigen Abschnitt erreicht.

Szenario 1 kann gelöst werden, indem für jede Säge ein Endpunkt erstellt wird, dem jeweils nur die Situation „zu heiß“ zugeordnet wird. Der beschriebene Algorithmus erstellt bei einem Aufruf der Operation „säge Holz“ für jeden der Endpunkte eine Bewertung. Angenommen, bei allen Sägen liegt die Situation „zu heiß“ vor. Dann erhalten alle Endpunkte die gleiche Bewertung. Aus diesen Endpunkten wird ein beliebiger ausgewählt.

Szenario 2 wird durch die Angabe von mehreren Situationen pro Endpunkt gelöst. Hierbei gibt es grundsätzlich zwei verschiedene Möglichkeiten dies zu tun. Bei der ersten Möglichkeit werden die Situationen angegeben wie folgt:

Endpunkt 1 Behandelt die Situation „zu heiß“.

Endpunkt 2 Behandelt die Situation „Sägeblatt abgenutzt“.

Endpunkt 3 Behandelt die Situationen „Sägeblatt abgenutzt“ und „zu heiß“.

Endpunkt 4 Behandelt die Situationen „Sägeblatt nicht abgenutzt“ (bedeutet: die Situation „Sägeblatt abgenutzt“ liegt nicht vor) und „nicht zu heiß“ (bedeutet: die Situation „zu heiß“ liegt nicht vor).

Beachtet werden muss, dass auf diese Weise durchaus mehrere valide Endpunkte gleichzeitig auftreten können. Die korrekte Auswahl erfolgt über die Bewertung. Wenn zum Beispiel die Situationen „Sägeblatt abgenutzt“ und „zu heiß“ vorliegen, sind die Endpunkte 1 - 3 zunächst valide Auswahlmöglichkeiten. Es wird jedoch immer Endpunkt 3 gewählt, da dieser durch Algorithmus 4.4 die höchste Bewertung erhält. Die andere Möglichkeit zur Lösung von Szenario 2 sieht folgende Endpunkte vor:

Endpunkt 1 Behandelt die Situation „zu heiß“ und „Sägeblatt nicht abgenutzt“.

Endpunkt 2 Behandelt die Situation „Sägeblatt abgenutzt“ „nicht zu heiß“.

Endpunkt 3 Behandelt die Situationen „Sägeblatt abgenutzt“ und „zu heiß“.

Endpunkt 4 Behandelt die Situationen „Sägeblatt nicht abgenutzt“ und „nicht zu heiß“.

Auf diese Weise wird ausgeschlossen, dass mehrere Endpunkte gleichzeitig in Betracht gezogen werden. Wenn hier die Situationen „Sägeblatt abgenutzt“ und „zu heiß“ gültig sind, kommen die Endpunkte eins und zwei nicht in Frage, da dort explizit angegeben wird, dass die Situationen „Sägeblatt abgenutzt“ (Endpunkt 1) beziehungsweise „zu heiß“ (Endpunkt 2) bei der Verwendung nicht auftreten beziehungsweise nicht vorliegen dürfen. Eine komplexere Logik zur Auswahl der Endpunkte kann und sollte auf Ebene der Situation Templates erstellt werden. Auf diese Weise könnten die Situationen „zu heiß“ und „Sägeblatt abgenutzt“ zum Beispiel zu einer Situation vereinigt werden. Dies hat den Vorteil, dass weniger verschiedene Endpunkte benötigt werden und keine exponentiell wachsende Anzahl an Pfaden mehr modelliert werden muss.

Szenario 3 Die in Szenario 3 gewünschte Priorisierung kann mithilfe von optionalen Situationen durchgeführt werden. Bei dem konkreten Beispiel könnten die Endpunkte hierfür aussehen wie folgt:

Endpunkt 1 Behandelt die Situation „zu heiß“ als optionale Situation.

Endpunkt 2 Behandelt die Situation „nicht zu heiß“(bedeutet: „zu heiß“ liegt nicht vor).

Diese Endpunkte müssen für jeweils beide Sägen aus Szenario 3 erstellt werden. Bei der Auswertung wird automatisch der Endpunkt mit der Situation „nicht zu heiß“ bevorzugt, da optionale Situationen weniger stark in die Bewertung einfließen.

Szenario 4 Das vierte Szenario kann gelöst werden, indem ein Endpunkt erstellt wird, der sowohl „zu heiß“ als auch „zu kalt“ als optionale Situationen angibt. Dadurch wird der Endpunkt ausgewählt, sobald eine der Situationen auftritt. Würde eine der Situationen nicht als optional markiert werden, würde der Endpunkt nicht mehr berücksichtigt werden, sobald die nicht-optionale Situation nicht auftritt.

Annahmen bezüglich Mehrdeutigkeit

Wie im vorigen Abschnitt gezeigt, können durch das erweiterte Modell zur Auswahl von Endpunkten deutlich mehr Szenarien behandelt werden, als mit dem einfachen Modell aus Abschnitt 4.3.2. Jedoch können prinzipiell beliebige Kombinationen aus Situationen und Operationen für Endpunkte angegeben werden. Diese sind jedoch häufig nicht sinnvoll, wie zum Beispiel folgendes Szenario zeigt:

Szenario 5 - Unzureichende Informationen

Beschreibung

Es existiert eine Holz- und eine Metallsäge. Beide unterstützen die Operation „säge“. Beide Sägen besitzen die gleichen Temperatursensoren, weshalb auf beiden dasselbe Situation Template betrieben wird. Das Template erkennt die Situation „zu heiß“. Für beide Sägen wird ein Endpunkt angegeben, der „säge“ implementiert und die Situation „zu heiß“ berücksichtigt.

Gewünschtes Ergebnis

Bei einem Aufruf der Operation „säge“ wird automatisch erkannt, welche Säge verwendet werden soll. Dementsprechend wird, wenn beide Endpunkte zur Auswahl stehen, der Endpunkt der richtigen Säge ausgewählt.

Dieses Szenario kann auch mit der erweiterten Endpunktauswahl nicht gelöst werden. Der Situation Handler verfügt nicht über die Informationen, die für eine derartige Auswahl notwendig sind. Hierfür müsste der Situation Handler über Kontextinformationen verfügen, um herauszufinden, ob Holz oder Metall gesägt werden soll. Eine Lösung wäre der Einsatz eines weiteren Situation Templates zur Erkennung von Holz und Metall. Das Szenario dient jedoch als Beispiel dafür, dass die Verantwortung, ein funktionierendes Szenario zu erstellen, beim Modellierer liegt und zeigt, dass bestimmte Annahmen darüber getroffen werden müssen, wie Endpunkte modelliert werden müssen. Annahme 1 aus Abschnitt 4.3.2 ist dabei immer noch gültig, das heißt es muss mindestens ein passender Endpunkt gefunden werden.

Annahme 1 Es existiert immer ein gültiger Endpunkt, siehe Abschnitt 4.3.2.

Annahme 2 Ein Endpunkt behandelt genau die Situationen, die für ihn spezifiziert wurden. Zusätzliche Situationen, die eventuell zeitgleich auftreten und die nicht im Widerspruch zu den anderen vom Endpunkt behandelten Situationen stehen, sind für den Endpunkt irrelevant.

Annahme 3 Ein Endpunkt, der mehr Situationen spezifiziert als ein anderer, ist tiefer spezialisiert und führt die entsprechende Aufgabe besser an die Situation angepasst durch, als ein allgemeinerer Endpunkt.

Annahme 4 Kann ein Endpunkt explizit nicht mit einer Situation umgehen, wird angegeben, dass das Gegenteil der Situation vorherrschen muss. Alternativ wird ein Endpunkt angegeben, der diese Situation behandelt. Dadurch kann außerdem eine Priorisierung erzielt werden: Je mehr Situationen für einen Endpunkt angegeben werden, desto höher ist der Endpunkt priorisiert (vgl. die beiden verschiedenen Lösungen für Szenario 2).

Annahme 5 Der Situation Handler benötigt für die Auswahl eines Endpunktes keine weiteren Informationen als die, die durch die Situationen und die Spezifikation des gewünschten Operationsaufrufs bereitgestellt werden (siehe Szenario 5).

Annahme 6 Falls am Ende der Bewertung durch den Situation Handler mehrere Endpunkte zur Wahl stehen, sind diese gleichwertig und deren Verwendung führt zum selben Resultat.

Diskussion

Die in den vorigen Abschnitten beschriebene Workflow Behandlung ist dazu in der Lage, verschiedene Szenarien zu lösen. Bei der Definition der Endpunkte beziehungsweise der Situationen und Operationen muss dabei jedoch stets beachtet werden, dass keine Mehrdeutigkeiten entstehen, die der Situation Handler nicht mithilfe der gegebenen Mittel auflösen kann. Die Kombination der einem Endpunkt zugeordneten Situationen stellt einen logischen Ausdruck dar. Hierbei wird ein logisches `AND` zwischen allen Situationen eines Endpunktes durchgeführt. Dabei können Situationen auch negiert werden.

Da Situationen als optional markiert werden können, ist es auch möglich einfache `OR`-Ausdrücke darzustellen (siehe Szenario 4). Ein vollständiges Logiksystem wird durch diese Möglichkeiten jedoch nicht umgesetzt. Zum Beispiel ist es nur schwer, möglich komplexere Ausdrücke mit mehreren `AND`- und `OR`-Verknüpfungen zu erstellen. Dies ist auch nicht das Ziel des Situation Handlers. Es wird hier eine höhere Priorität darauf gelegt, Endpunkte für verschiedene Situationen möglichst schnell und unkompliziert erstellen zu können. Der Hintergrundgedanke besteht darin, dass in den meisten Fällen nicht sehr viele verschiedene Situationen behandelt werden, da ansonsten auch sehr viele verschiedene Endpunkte angegeben werden müssten. Komplexere logische Ausdrücke (Prädikatenlogik erster Stufe) können auf Ebene der Situation Templates erstellt werden. Es ist daher empfehlenswert, so viel Logik wie möglich in die Situation Templates auszulagern. Der Situation Handler soll dabei grundsätzlich auf einem hohen Abstraktionslevel arbeiten. Dies entspricht auch dem Grundgedanken der Abstraktion aus Abschnitt 3.3: Die Situationen sollen die Low-Level-Kontextinformationen der Sensoren abstrahieren - dementsprechend sollte der Situation Handler nicht den Schritt zurück auf ein niedriges Abstraktionslevel gehen. Dennoch kann der Situation Handler als weitere Schicht gesehen werden, in der Situationen noch weiter abstrahiert werden. Dabei können mehrere Situationen in einem Endpunkt zusammengefasst werden, der diese handhabt. Zu beachten ist auch, dass dadurch die einfache Verknüpfung von Situationen über mehrere Objekte hinweg möglich ist. Situation Templates sind hierbei insofern eingeschränkt, dass sie nur auf einem einzelnen Objekt (Ding) betrieben werden und dort Situationen erkennen. Die Situationen auf Ebene des Situation Handlers können von verschiedenen Objekten stammen, wodurch auch eine objektübergreifende Situationsbehandlung ermöglicht wird. Weitere Überlegungen zur Kombination von Situationen über verschiedene Objekte hinweg stellen den Gegenstand zukünftiger Forschungsarbeiten dar.

Eine Priorisierung kann wie in Szenario 3 durch optionale Situationen erreicht werden. Eine alternative Möglichkeit hierfür wäre ein vollständiges Priorisierungssystem, zum Beispiel indem jedem Endpunkt eine Priorität durch eine Zahl von 1 bis 100 zugewiesen wird. Es könnte dann der Endpunkt ausgewählt werden, dessen Situationen vorliegen und der die höchste Priorität besitzt. Dadurch könnten deutlich mehr Abstufungen erreicht werden, wie über das beschriebene Prinzip der kleineren Gewichtung von optionalen Situationen. Hierauf wird verzichtet, um keine unnötige Komplexität bei der Definition von Endpunkten einzuführen. Die Definition von beliebigen Prioritätsleveln scheint zudem nur wenig zusätzlichen Nutzen zu versprechen - die meisten Anwendungsfälle sehen eine Auswahl über aktuell vorliegende Situationen und ansonsten gleichwertige Endpunkte vor.

4.3.4. Message Routing

Bisher wurde für die Behandlung von Dienstaufrufen durch Workflows beschrieben, wie die Auswahl eines Endpunktes abläuft - jedoch nicht, wie die Kommunikation zwischen dem Situation Handler, Workflows und Workflow-Fragmenten abläuft. Das grundsätzliche Szenario sieht hierbei vor, dass der Workflow dem Situation Handler eine Anfrage schickt und der Situation Handler einen Endpunkt (Workflow-Fragment) für die Anfrage auswählt. Die Anfrage wird an den Endpunkt weitergeleitet. Der Endpunkt bearbeitet die Anfrage und schickt dem Situation Handler eine Antwort. Dieser leitet die Antwort direkt an den Workflow, der die Anfrage geschickt hat, weiter.

Dieses Szenario könnte sowohl über synchrone als auch über asynchrone Kommunikation umgesetzt werden. Synchrone Kommunikation würde den Nachteil bieten, dass der Workflow blockiert wäre,

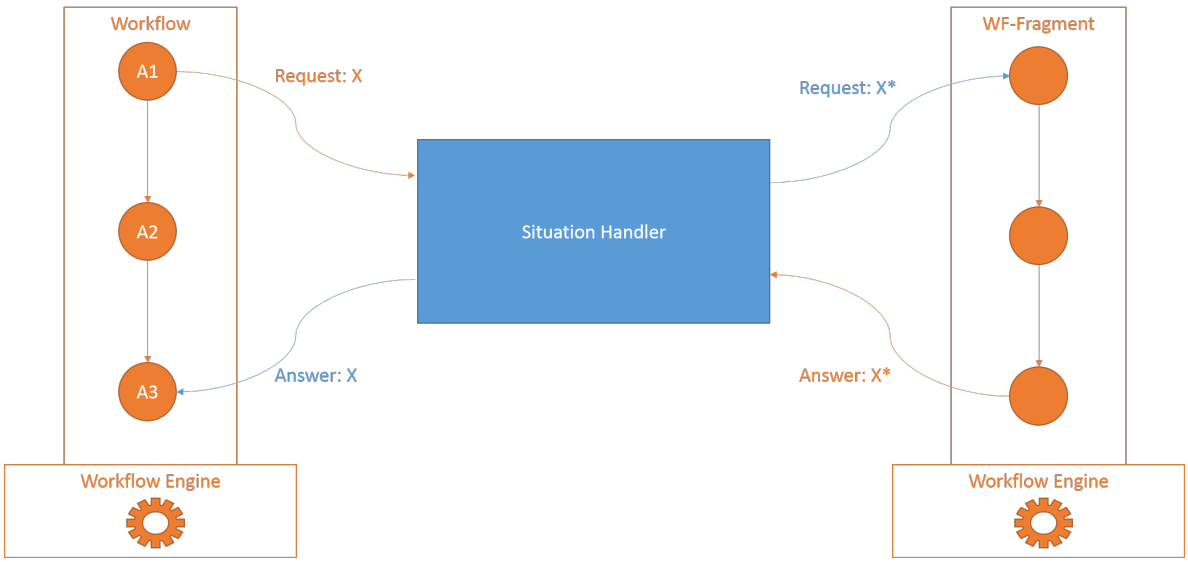


Abbildung 4.5.: Asynchrone Kommunikation zwischen Workflow, Situation Handler und Workflow-Fragment.

bis die Antwort vom Endpunkt beziehungsweise vom Situation Handler eingegangen wäre. Auch der Situation Handler müsste die Verbindung zum Workflow und zum Endpunkt offen halten, bis die Bearbeitung der Anfrage vollständig abgeschlossen wäre. Asynchrone Kommunikation bietet diese Nachteile nicht. Hierbei werden weder der Workflow blockiert, noch muss der Situation Handler die Verbindung zum Workflow ununterbrochen geöffnet halten. Der Workflow kann somit andere Aktivitäten ausführen, während er auf die Antwort des Situation Handlers wartet. Asynchrone Kommunikation ist jedoch komplexer zu realisieren, da Anfrage- und Antwortnachrichten einander manuell zugeordnet (korreliert) werden müssen.

Für den Situation Handler wird angenommen, dass sämtliche Kommunikation auf eine asynchrone Art stattfindet, da diese insgesamt mehr Möglichkeiten bietet. Ein synchrones Anwendungsszenario kann zudem einfach mit asynchroner Kommunikation umgesetzt werden, indem der Workflow sofort nach dem Senden der Anfrage beginnt auf die Antwort zu warten und seine weitere Ausführung vom Eingang der Antwortnachricht abhängig macht das heißt die weitere Ausführung entsprechend bis zum Eintreffen der Antwort blockiert.

Ein Beispiel für die asynchrone Kommunikation zwischen Workflow, Situation Handler und Workflow-Fragment wird in Abbildung 4.5 dargestellt. Die Workflows und Nachrichten ausgehend von Workflows sind in der Abbildung orange dargestellt, der Situation Handler und vom Situation Handler gesendete Nachrichten in blau. Andere Komponenten, wie das SES, werden zwar bei der Situationsbehandlung verwendet, spielen für die Kommunikation zwischen Workflow und Situation Handler aber keine Rolle und werden daher nicht dargestellt. Das Beispiel beginnt mit der Ausführung von Aktivität A1 durch den Workflow auf der linken Seite der Abbildung. Dabei wird Anfrage X an den Situation Handler gesendet. Anfrage X ist eindeutig identifizierbar und enthält zudem Informationen darüber, wohin die Antwort gesendet werden soll. Der Workflow kann nach dem Senden der Nachricht direkt

fortfahren und A2 ausführen. Erst für die Ausführung von A3 wird die Antwort auf Anfrage X benötigt. Der Situation Handler wählt nach dem Erhalt von Anfrage X ein geeignetes Workflow-Fragment aus und leitet die Anfrage an den Endpunkt weiter, der dieses Workflow-Fragment ausführt. Vor der Weiterleitung muss Anfrage X jedoch modifiziert werden. Die Antwortadresse, die in Anfrage X enthalten ist, wird hierbei durch die Adresse des Situation Handlers ersetzt. Das Resultat ist Anfrage X*. Der Situation Handler speichert dabei die Antwortadresse, die ursprünglich in Anfrage X enthalten war. Das Workflow-Fragment schickt die Antwort auf Anfrage X* an den Situation Handler. Dieser überprüft, welche Antwortadresse ursprünglich für Anfrage X vorgesehen war und schickt die Antwort dorthin. In diesem Fall wird die Antwort zurück an den Workflow geschickt und dort von Aktivität A3 verarbeitet.

4.3.5. Situationsbedingte Rollbacks

Geht beim Situation Handler eine Anfrage von einem Workflow ein, wird abhängig von aktuell vorliegenden Situationen entschieden, an welches Workflow-Fragment (beziehungsweise an welchen Endpunkt, der das Workflow-Fragment ausführt) die Anfrage weitergeleitet wird. Der Zustand einer Situation kann sich jedoch jederzeit ändern, das heißt sie kann verschwinden oder neu auftauchen, wenn sie vorher nicht vorlag. In diesem Fall wäre es möglich, dass ein Workflow-Fragment, das gerade eine Anfrage verarbeitet, nicht mehr zur Verarbeitung dieser Anfrage geeignet ist. Um eine fehlerhafte Verarbeitung der Anfrage zu vermeiden, muss die Verarbeitung durch dieses Workflow-Fragment abgebrochen und ein anderes Workflow-Fragment ausgewählt werden, das die neue Situation handhaben kann. Das bisher zuständige Workflow-Fragment muss zudem sämtliche Aktionen, die bereits durchgeführt wurden, rückgängig machen (**Rollback**).

Ein Rollback ist jedoch nicht für jede Situationsänderung sinnvoll. Dies verdeutlicht folgendes Beispiel: Es wird ein Workflow-Fragment spezifiziert, das die Operation „säge Holz“ umsetzt und in der Situation „zu heiß“ eingesetzt wird (Workflow-Fragment 1). Eine andere Variante des Workflow-Fragments setzt die gleiche Operation um und wird in der Situation „nicht zu heiß“ eingesetzt, das heißt in dem Fall, dass „zu heiß“ nicht vorliegt (Workflow-Fragment 2). Angenommen die Situation „zu heiß“ liegt vor und entsprechend wird Workflow-Fragment 1 ausgewählt. In diesem Fall wird bei der Ausführung von „säge Holz“ durch Workflow-Fragment 1 die Säge zuerst gekühlt und erst danach wird die eigentliche Sägeoperation ausgeführt. Offensichtlich wird währenddessen die Situation „zu heiß“ verschwinden. Dennoch kann Workflow-Fragment 1 weiter verwendet werden und es ist kein Rollback nötig. Das Modell des Endpunktes aus Abschnitt 4.3.3 wird daher geringfügig angepasst. Ein Endpunkt erlaubt in dem bisherigen Modell die Angabe von mehreren Situationen, die überprüft werden. Für jede einzelne dieser Situationen kann nun angegeben werden, ob ein Rollback bei der Änderung der Situation durchgeführt werden soll oder nicht.

Für einen Rollback ergibt sich folgender Ablauf: Zu Beginn erhält der Situation Handler eine Anfrage von einem Workflow. Mithilfe des Algorithmus aus Abschnitt 4.3.3 wird ein passender Endpunkt (der ein Workflow-Fragment ausführt) zur Verarbeitung der Anfrage ausgewählt. Für die Situationen, die diesem Endpunkt zugeordnet sind, wird überprüft, ob eine Situationsänderung einen Rollback erfordern würde. Wenn dies der Fall ist, wird für diese Situation ein Eintrag angelegt, dass die Änderung dieser Situation einen Rollback für genau diesen Endpunkt erfordert. Da mehrere Situationen pro Endpunkt festgelegt werden können, können pro Endpunkt entsprechend mehrere Verweise auf

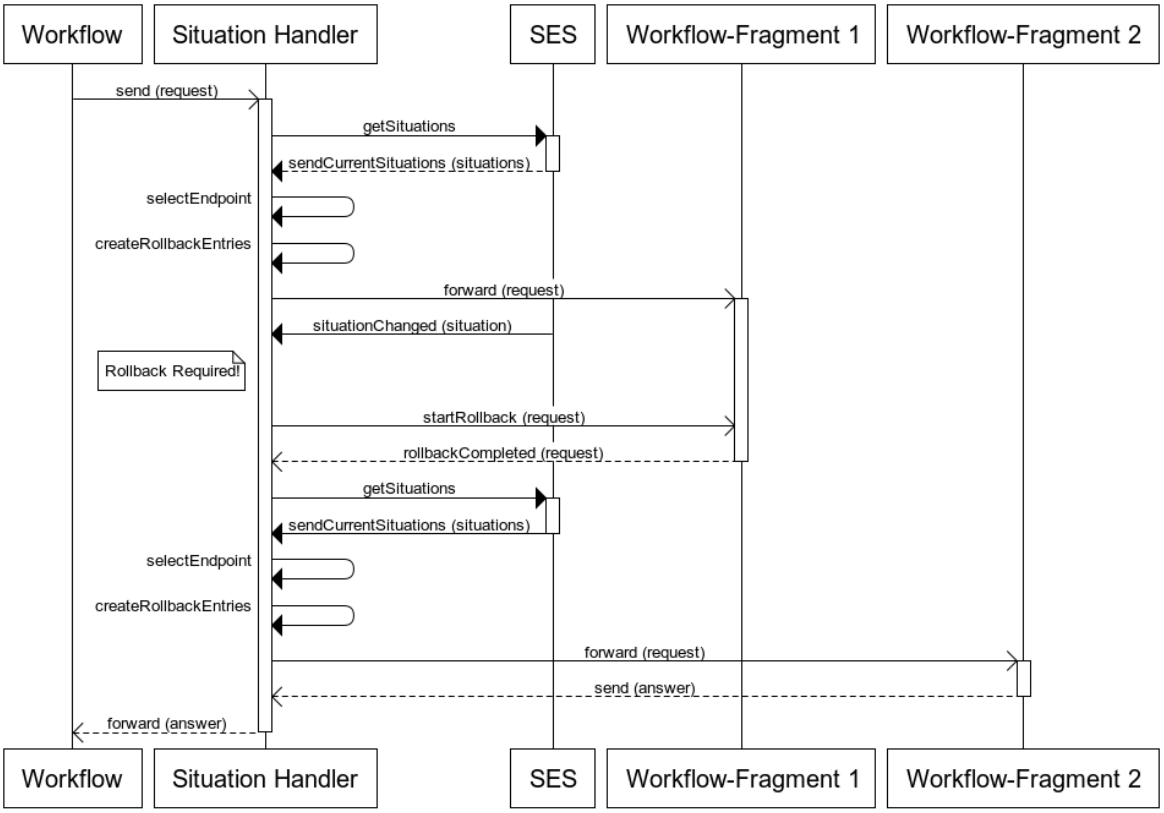


Abbildung 4.6.: Die Prozedur bei einem situationsbedingten Rollback, einschließlich den Kommunikationsbeziehungen.

Rollbacks angelegt werden. Die Anfrage wird an den gewählten Endpunkt weitergeleitet und dort verarbeitet. Tritt nun eine Situationsänderung auf, schickt der Situation Handler eine Rollback-Nachricht an den Endpunkt, die sich auf die vorige Anfrage bezieht. Der Endpunkt führt daraufhin einen Rollback durch. Nachdem dieser abgeschlossen ist, schickt der Endpunkt eine Nachricht an den Situation Handler, die den Abschluss des Rollbacks bestätigt. Darauf wählt der Situation Handler einen anderen Endpunkt zur Verarbeitung aus. Wenn dieser die Anfrage vollständig verarbeitet hat, schickt er wie im Standardfall eine Antwort an den Situation Handler. Dieser leitet die Antwort an die Antwortadresse aus der ursprünglichen Anfrage weiter. Der eben beschriebene Ablauf bei einem Rollback wird in Abbildung 4.6 als UML-Sequenzdiagramm dargestellt.

Zu beachten ist jedoch, dass eine Situation sich unter Umständen in einem kurzen Zeitraum sehr häufig ändern kann. Wenn diese Situationsänderung mit einem Rollback verbunden ist, würde bei jeder Änderung ein Rollback durchgeführt und ein neuer Endpunkt gewählt werden. Unter Umständen würde die Bearbeitung der Anfrage auf diese Weise sehr lange dauern oder nie abgeschlossen werden. Workflows können in ihren Anfragen daher ein Limit für die Anzahl der Rollbacks beziehungsweise für die Anzahl der Verarbeitungsversuche festlegen. Der Situation Handler überwacht, wie oft für eine Anfrage bereits ein Rollback durchgeführt wurde. Wird das festgelegte Limit überschritten,

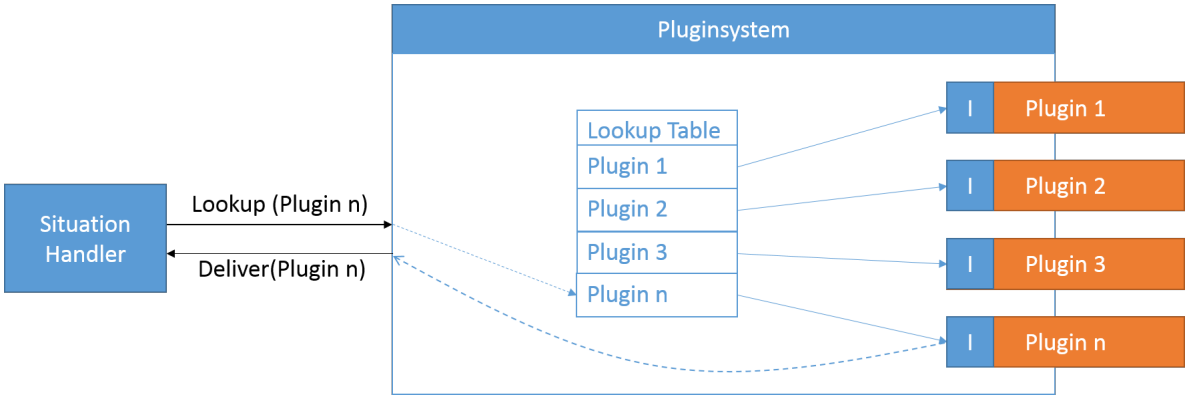


Abbildung 4.7.: Pluginsystem und dessen Verwendung.

wird eine entsprechende Fehlernachricht an den aufrufenden Workflow versandt. Nachdem das Limit überschritten wurde, wird kein neuer Endpunkt mehr für die Anfrage ausgewählt und die Verarbeitung wird abgebrochen.

4.4. Pluginbasiertes Kommunikationskonzept

In vorigen Abschnitten wurde auf das Kommunikationssystem des Situation Handlers verwiesen, das mit Plugins arbeitet. Dieses wird in diesem Abschnitt näher erläutert. Zunächst wird in Abschnitt 4.4.1 das Pluginsystem allgemein beschrieben. Danach wird in 4.4.2 auf die Anforderungen an ein Plugin eingegangen, die für die Funktion innerhalb des Pluginsystems erfüllt werden müssen. Im darauf folgenden Unterabschnitt 4.4.3 wird erklärt, welche Plugins im Rahmen dieser Arbeit geliefert werden. Abschließend werden in Abschnitt 4.4.4 Vor- und Nachteile des Pluginsystems diskutiert.

4.4.1. Idee des Pluginsystems

Das Pluginsystem bietet Zugriff auf Plugins, die eine bestimme Art von Kommunikation realisieren oder zur Ausführung einer sonstigen Aktion dienen. Der konzeptionelle Aufbau des Pluginsystems wird in Abbildung 4.7 gezeigt. Das Pluginsystem in der Mitte des Bildes besitzt eine Tabelle, in der Informationen über alle momentan vorhandenen Plugins gespeichert werden. Die Plugins sind auf der rechten Seite des Bildes als einsteckbare Teile des Pluginsystems dargestellt. Jedes Plugin implementiert eine standardisierte Schnittstelle, über die es verwendet werden kann. Eine Beschreibung der Schnittstelle erfolgt im nächsten Abschnitt. Die Schnittstelle wird wird durch den blauen Teil im Plugin dargestellt. Will der Nutzer (in diesem Fall der Situation Handler) ein Plugin verwenden, schickt er eine Anfrage an das Pluginsystem. Dieses durchsucht die Plugin-Tabelle (*Lookup Table*) nach dem geforderten Plugin und liefert es an den Situation Handler zurück. In der Abbildung wird der Situation Handler auf der linken Seite dargestellt. Die Anfrage wird durch den Pfeil symbolisiert. Vorgänge innerhalb der Komponente werden gestrichelt dargestellt. Zur Identifikation eines Plugins

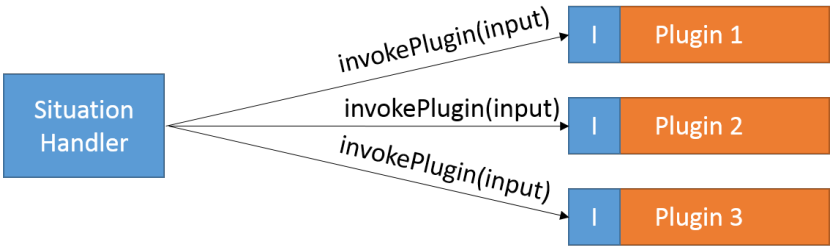


Abbildung 4.8.: Plugins können über eine standardisierte Schnittstelle auf dieselbe Weise verwendet werden.

bei einer Anfrage durch den Situation Handler wird ein Identifikator benötigt, welches das Plugin eindeutig identifiziert.

Zur Verwendung eines Plugins wird die standardisierte Schnittstelle des Plugins verwendet. Die Interaktion mit verschiedenen Plugins erfolgt, wie in Abbildung 4.8 dargestellt: Mit jedem Plugin, wird auf die gleiche Weise interagiert. Dies wird durch die standardisierte Schnittstelle ermöglicht. Damit die einheitliche Interaktion ermöglicht wird, folgen auch die Eingabewerte des Plugins einem Standard (siehe nächster Abschnitt).

Um das Pluginsystem so dynamisch wie möglich zu gestalten, wird es erlaubt, die Plugins während der Laufzeit des Situation Handlers zu verändern. Dabei ist es möglich, neue Plugins hinzuzufügen und bestehende zu löschen. Zur Ersetzung eines Plugins durch eine neuere Version, muss das existierende Plugin gelöscht und die neue Version hinzugefügt werden. Neue Plugins stehen dem Nutzer sofort zur Verfügung, nachdem sie hinzugefügt wurden. Der Löschvorgang eines Plugins ist hingegen keine Aktion mit sofortiger Wirkung. Ein altes Plugin kann noch verwendet werden, sofern es vor der Löschaktion vom Nutzer angefordert wurde. Es ist jedoch ab dem Moment, in dem der Befehl zur Löschung des Plugins einging, nicht mehr möglich, das Plugin für die Verwendung neu anzufordern.

Abhängigkeiten zwischen Plugins werden nicht erlaubt, das heißt es ist nicht möglich, dass ein Plugin auf die Funktionalität eines anderen Plugins zugreift. Zudem können die Plugins nicht auf sonstige Funktionalitäten des Pluginsystems oder des Situation Handlers zugreifen. Ein Plugin ist demnach eine vollständig isolierte Einheit. Verwendet ein Plugin externe Funktionalitäten, müssen diese in das Plugin integriert werden. Das Verbot von Abhängigkeiten zwischen Plugins wird in Abschnitt 4.4.4 begründet.

4.4.2. Anforderungen und Vorgaben an ein Plugin

Wie bereits im vorigen Abschnitt angedeutet, muss ein Plugin diverse Vorgaben befolgen. Aus dem Entwurf des Pluginsystems ergeben sich folgende Anforderungen an ein Plugin:

Anforderung 1 Das Plugin muss eine standardisierte Schnittstelle zur Verwendung bereitstellen.

Anforderung 2 Die Eingabe für das Plugin folgt einer standardisierten Form.

Anforderung 3 Das Plugin stellt relevante Information zur Laufzeit bereit.

Anforderung 4 Sämtliche externe Abhängigkeiten müssen in das Plugin integriert sein, das heißt das Plugin ist eine selbstständige Einheit.

Bis auf diese Vorgaben gibt es keine Einschränkungen für ein Plugin, das heißt es kann prinzipiell eine beliebige Funktionalität umsetzen. Grundidee ist es jedoch wie in den vorigen Abschnitten erläutert, dass Plugins zum Versand von Notifikationen bei der Situationsbehandlung. (siehe Abschnitt 4.3) verwendet werden. Außerdem ist durch das Pluginsystem der Aufruf von Workflow-Fragmenten möglich, um auf Situationen zu reagieren. Dadurch lassen sich situationsabhängig Workflow-Fragmente aufrufen, wodurch sich auch das von Breitenbücher et al. [BHK+15] entwickelte Konzept der *Situation Events* umsetzen lässt. Die Situation Events werden in Kapitel 6 näher erläutert.

Zur Umsetzung von **Anforderung 1** muss ein Plugin eine Schnittstelle mit folgender Funktionalität implementieren: Zunächst benötigt die Schnittstelle eine Möglichkeit, die Funktion des Plugins zu verwenden, also seine Ausführung zu starten. Beim Start des Plugins können die Nutzereingaben an das Plugin übergeben werden. Die Eingaben, die ein Plugin erhalten kann, orientieren sich an dem Aufbau einer Aktion aus Abschnitt 4.3.1. Eine Aktion beinhaltet mehrere Informationen, die zur Verwendung des Plugins benutzt werden:

Plugin
: Die Beschreibung des Plugins wird verwendet, um das korrekte Plugin vom Pluginsystem zu erhalten.

Empfänger
: Der Empfänger dient als Ziel zum Versand der Nachricht.

Nachricht
: Die Nachricht wird vom Plugin an den Empfänger versandt.

Optionale Parameter
: Die optionalen Parameter werden zur Ausführung ebenfalls an das Plugin übergeben.

Ausführungszeitpunkt
: Wird vom Plugin nicht benötigt.

Als Eingabe zur Ausführung des Plugins dienen folglich Empfänger, Nachricht und die optionalen Parameter. Die optionalen Parameter können dabei frei vom Plugin spezifiziert werden. Um **Anforderung 2** zu erfüllen, reicht es, diese Eingaben zu akzeptieren. Dem Plugin ist freigestellt, wie diese Eingaben verarbeitet werden. Es ist die Aufgabe des Plugin-Erstellers eine angemessene Dokumentation für das Plugin bereitzustellen, in der die Form der Eingaben für das Plugin dokumentiert wird. Dies trifft insbesondere auch für die optionalen Parameter zu. Um **Anforderung 3** zu erfüllen, muss das Plugin folgende Informationen zur Laufzeit bereitstellen können:

Identifikator
: Ein eindeutiger Identifikator des Plugins. Kann vom Pluginsystem gelesen werden, um das Plugin zu identifizieren. Kann außerdem vom Nutzer verwendet werden, um das Plugin anzusprechen.

Name
Der Name des Plugins. Dieser soll insbesondere eine gute Lesbarkeit und Aussagekraft bieten.

Anzahl der optionalen Parameter
Eine Angabe, wie viele optionale Parameter das Plugin benötigt.

Beschreibung der optionalen Parameter
Eine Beschreibung für jeden optionalen Parameter.

Diese Informationen erlauben eine sinnvolle Interaktion mit dem Plugin und der Nutzer wird befähigt Aktionen für das Plugin zu erstellen.

4.4.3. Standard-Plugins des Situation Handlers

Da sämtliche ausgehende Kommunikation des Situation Handlers auf Plugins basiert, müssen dem Situation Handler einige Standard-Plugins zur Verfügung gestellt werden. Erforderlich für die Behandlung von Dienstaufrufen (siehe 4.3.2) ist ein Plugin, um externe Workflow-Fragmente zu starten. Zum Start verschiedener Workflow-Fragmente können unter Umständen verschiedene Technologien verwendet werden. Das Plugin muss diese berücksichtigen. Informationen über dort verwendete Technologien sowie den Umgang des „Workflow-Plugins“ mit diesen werden im Implementierungsteil gegeben. Zu beachten ist, dass dieses Plugin unabdingbar für die Funktion der situationsabhängigen Workflow-Behandlung ist. Ohne es können keine Workflow-Fragmente mehr gestartet werden. Dadurch würden Workflows, die den Situation Handler verwenden, blockiert werden, da sie vom Situation Handler keine Antwort erhalten. Das Plugin darf folglich keinenfalls dauerhaft entfernt werden. Um beispielsweise ein Update des Plugins zu ermöglichen, wird die Entfernung des „Workflow-Plugins“ durch das Pluginsystem nicht unterbunden. Es liegt in der Verantwortung des Nutzers, einen reibungslosen Austausch des Plugins durchzuführen.

Des Weiteren werden Plugins zur Verfügung gestellt, die weit verbreitete Kommunikationswege nutzen. Hierunter fällt ein E-Mail-Plugin, welches den Versand von E-Mails erlaubt. Der Versand erfolgt dabei von einer festen Adresse. Empfänger, Betreff und Inhalt der E-Mail können frei spezifiziert werden. Ein weiteres Plugin dient dazu, Notifikationen an Mobilgeräte zu senden.

4.4.4. Diskussion: Vor- und Nachteile eines Pluginsystems

Die Realisierung der ausgehenden Kommunikation auf Basis von Plugins ergibt beim Situation Handler aus mehreren Gründen Sinn. Der Hauptgrund besteht darin, dass ein Pluginsystem ideal dazu geeignet ist, um verschiedene Arten von Notifikationen bei der Situationsbehandlung zu unterstützen. So ist es möglich, flexibel neue Arten von Notifikationen hinzuzufügen, ohne den Situation Handler verändern zu müssen. Auch ein Update oder das Entfernen eines Plugins ist möglich, ohne dass die Hauptkomponente des Situation Handlers verändert werden muss. Das Hinzufügen, Ändern und Entfernen von Plugins kann zudem während dem Betrieb des Situation Handlers geschehen, das heißt dieser kann die Situationsbehandlung ohne Unterbrechungen fortsetzen. Fest integrierte Kommunikationskomponenten würden bei einer Änderung einen Stopp des Situation Handlers

erfordern. Auch können Plugins jederzeit von externen Parteien bereitgestellt werden, ohne dass diese genaue Kenntnisse über die Umsetzung des Situation Handlers besitzen müssen. Es reicht die Vorgaben für ein Plugin zu kennen und diese umzusetzen. Ferner ist ein weiterer Vorteil eines Pluginsystems, dass auf diese Weise eine modulare Architektur unterstützt wird. Einzelne Komponenten sind dadurch besser voneinander abgekoppelt, wodurch Einflüsse von unbekannten Komponenten ausgeschlossen werden können. Durch einen modularen Aufbau wird zudem die Wiederverwendbarkeit von einzelnen Komponenten gefördert, da diese so streng nach ihrer Funktionalität gekapselt sind und so in andere Anwendungen integriert werden können, die dieselbe Funktionalität benötigen.

Die strenge Abkapselung bietet auf der anderen Seite den Nachteil, dass generell keine Abhängigkeiten zwischen verschiedenen Plugins möglich sind. So ist es denkbar, dass ein Plugin die Funktionalität eines anderen benötigen würde und diese selbst umsetzen muss, anstatt auf das andere Plugin zugreifen zu können. So muss eine Funktionalität unnötigerweise mehrfach umgesetzt werden. Dieses Problem könnte behoben werden, indem das Pluginsystem zulässt, dass Abhängigkeiten zu anderen Plugins spezifiziert werden können. Eine solche Möglichkeit bringt jedoch andere Probleme mit sich: Bei Abhängigkeiten von externen Plugins ist es leicht möglich, dass diese stillschweigend ihre Funktionalität ändern und so auch das Verhalten des abhängigen Plugins geändert wird. Dies führt möglicherweise zu Fehlern, die im Normalfall schwierig zu erkennen und zu beheben sind. Zudem erfordern Abhängigkeiten zwischen verschiedenen Plugins komplexe Prüfungen seitens des Pluginsystems, um festzustellen, ob sämtliche Abhängigkeiten erfüllt sind. Wenn zum Beispiel ein Plugin deaktiviert wird, sind sämtliche andere Plugins, die von diesem Plugin abhängig sind, nicht mehr funktionsfähig. Die beschriebenen Probleme verschärfen sich zudem durch transitive Abhängigkeiten. Beispielsweise wäre es möglich, dass Plugin A von Plugin B abhängt und Plugin B wiederum von Plugin C abhängig ist. Ändert Plugin C seine Funktionalität, kann dies zu einer Verhaltensänderung bei Plugin B führen, die sich auch auf Plugin A fortpflanzt. Für den Entwickler von Plugin A wäre es schwierig, den Grund für diese Verhaltensänderung zu erkennen und das Verhalten zu korrigieren. Ähnliche Probleme würden entstehen, wenn Plugin C deaktiviert wird. Dies würde auch zu Fehlern bei Plugin A führen. Für den Entwickler von Plugin A wäre dies auch in diesem Fall schwer nachzuvollziehen, da dort nur eine Abhängigkeit zu Plugin B vorhanden ist. Aufgrund dieser Probleme werden keine Abhängigkeiten zwischen Plugins erlaubt.

4.5. SES-Interaktion

Das Situationserkennungssystem (SES) ist eine wichtige externe Komponente, mit der der Situation Handler zusammenarbeitet. Die Funktionen des SES werden dem Situation Handler durch SitRS [HWS+15] bereitgestellt. Insbesondere bei der Situationsbehandlung wird häufig mit dem SES interagiert, um Informationen über aktuelle Situationen vom SES zu beziehen. Das SES bietet zwei verschiedene Varianten um Situationen abzufragen.

Bei der ersten Variante wird vom Situation Handler eine Anfrage geschickt, um aktiv abzufragen, ob eine Situation aufgetreten ist. Hier können statt der Abfrage einer einzelnen Aktion auch alle im SES vorhandenen Situationen abgefragt werden. Bei der situationsabhängigen Behandlung von Dienstaufrufen wird die aktive Abfrage einzelner Situationen eingesetzt. Dort interessiert nur der aktuelle Zustand einer Situation, das heißt ob die Situation vorliegt oder nicht. Um die Anzahl der Anfragen an

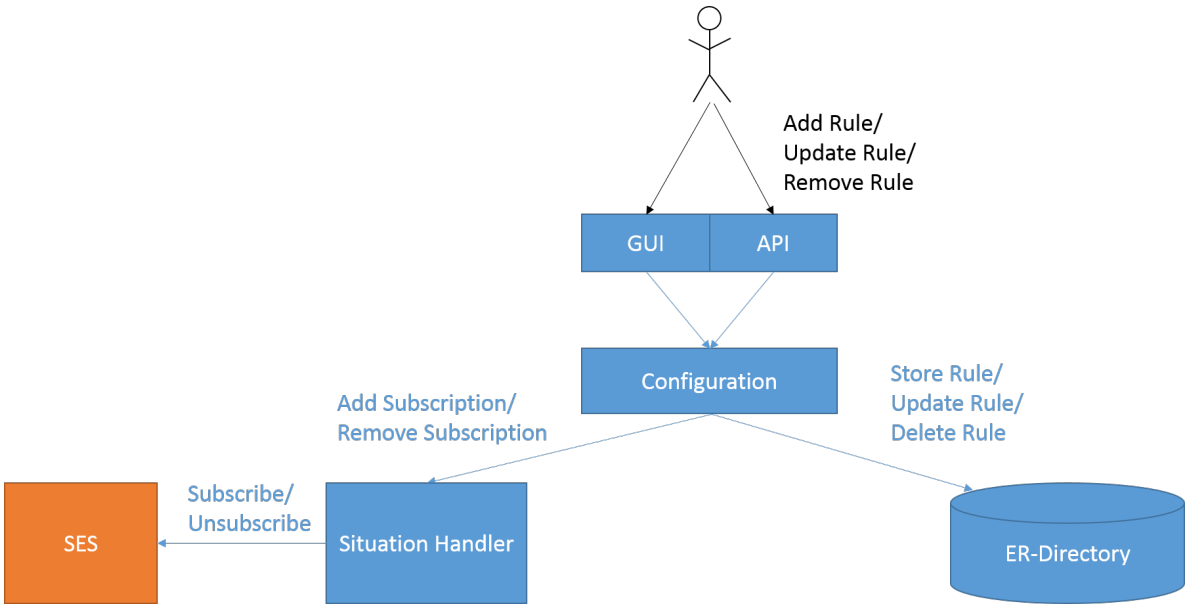


Abbildung 4.9.: Durchführung von Subscribe-Operationen.

das SES so gering wie möglich zu halten, wird das Ergebnis der Anfragen zwischengespeichert. Der verwendete Zwischenspeicher wird abgefragt, bevor eine Anfrage an das SES gesendet wird. Durch die Verwendung eines lokal gespeicherten Ergebnisses wird die Behandlung von Workflow-Operationen beschleunigt, da der Overhead durch die Kommunikation mit dem SES entfällt.

Bei der zweiten Variante werden Situationsänderungen mittels Publish/Subsribe vom SES mitgeteilt. Der Subscriber hat hierbei die Möglichkeit, entweder ein Subscribe auf einzelne Situationen durchzuführen oder auf alle Situationen. Wird auf eine Situation ein Subscribe durchgeführt, erhält der Nutzer eine Benachrichtigung, sobald die Situation auftaucht oder verschwindet. Das SES bietet auch die Möglichkeit, ein Subscribe wieder rückgängig zu machen, um die Benachrichtigungen bezüglich einer Situation zu deaktivieren. Die Publish/Subscribe-Benachrichtigungen werden bei der Situationsbehandlung für den Versand von Notifikationen verwendet. Publish/Subscribe bietet hier den Vorteil, dass Änderungen der Situation sofort mitgeteilt werden. Als Alternative kämen nur regelmäßige aktive Abfragen der Situation in Frage (*polling*). Hierbei würde eine Situationsänderung aber nur mit einer gewissen Verzögerung erkannt werden. Außerdem würde es zu vielen unnötigen Abfragen kommen. Daher ist für den situationsabhängigen Versand von Notifikationen Publish/Subscribe die bessere Wahl.

Um unnötige Subscribe-Operationen zu vermeiden, wird kein Subscribe auf alle Situationen durchgeführt. Stattdessen wird ein Subscribe auf eine Situation nur dann durchgeführt, wenn eine Notifikation bei der Situationsänderung versandt werden soll. Die Durchführung der Subscribe-Operation wird in Abbildung 4.9 dargestellt. Der Nutzer wird in der Abbildung oben gezeigt. Die Komponenten die im Rahmen dieser Arbeit entstehen sind blau markiert, externe Komponenten orange. Da neue Notifikationsregeln nur über GUI und API durch den Nutzer erstellt werden können, wird der Aufruf dort abgefangen und an eine Schicht zur Konfiguration weitergeleitet. Hier werden alle Schritte zur

Verarbeitung der Nutzereingaben durchgeführt. Es wird hier sowohl die Notifikationsregel gespeichert, als auch eine Benachrichtigung an den Situation Handler geschickt. Der Situation Handler führt daraufhin die Subscribe-Operation am SES durch. Die Subscribe-Operation bezieht sich in diesem Fall auf die Situation, die dieser Regel zugeordnet ist. Es ist zu beachten, dass für jede Situation nur genau eine Notifikationsregel existieren kann (siehe Abschnitt 4.3.1) und daher höchstens eine Subscribe-Nachricht pro Situation geschickt wird. Nach der Subscribe-Operation für die Situation erhält der Situation Handler Informationen über Änderungen dieser Situation. Dies bedeutet, dass bei einer Situationsänderung eine Behandlung wie in Abschnitt 4.3.1 beschrieben durchgeführt werden kann. Löscht der Nutzer eine Regel, wird der Löschvorgang analog zum Hinzufügen einer Regel durchgeführt. Auch das Löschen ist für den Nutzer nur über die GUI/API möglich. Daher wird dieser Aufruf dort ebenfalls abgefangen und entsprechend an das ER-Directory und den Situation Handler weitergeleitet (siehe Abbildung 4.9). Bei der Änderung einer Regel beziehungsweise der Situation, die der Regel zugeordnet ist, muss die Subscription auf die alte Regel ebenfalls entfernt werden. Im gleichen Schritt wird die Subscription für die neue Situation erstellt.

Für die Verwendung des Publish/Subscribe-Systems gibt es zusätzlich zum Versand von Notifikationen noch einen weiteren Anwendungsfall. Dieser betrifft die in Abschnitt 4.3.5 beschriebenen Rollbacks. Hierbei wird ein Rollback ausgelöst, sobald sich der Zustand einer Situation verändert. Eine Situationsänderung wird am besten erkannt, indem ein Subscribe auf diese Situation durchgeführt und die Benachrichtigung bei einer Änderung entsprechend verarbeitet wird. Daher werden auch bei der Erzeugung von Endpunkten Subscriptions erstellt. Dabei wird eine Subscription für jede Situation erzeugt, die dem Endpunkt zugeordnet ist. Dies geschieht analog zur Erzeugung der Notifikationsregeln, wie in Abbildung 4.9 dargestellt. Dementsprechend werden die Eingaben des Nutzers über die GUI/API an die Konfigurationsschicht weitergeleitet. Dort werden die Endpunkte im ER-Directory abgelegt und die Subscriptions für die durch den Endpunkt behandelten Situationen werden erstellt. Wird ein Endpunkt oder eine vom Endpunkt behandelte Situation entfernt, kann auch die Subscription entfernt werden.

Durch die Verwendung von Subscriptions für die Endpunkte ergibt sich jedoch folgendes Problem: Regeln und Endpunkte können sich auf die gleichen Situationen beziehen, wodurch es mehrere Subscriptions für ein und dieselbe Situation geben kann. Da sich grundsätzlich beliebig viele Endpunkte auf dieselbe Situation beziehen können, werden auch entsprechend viele Subscriptions erstellt. Um mehrfache Subscriptions zu vermeiden, schickt der Situation Handler maximal eine Subscription an das SES, speichert jedoch gleichzeitig, wie viele Subscribe-Anfragen eingegangen sind. Wird eine Subscription gelöscht, wird der Löschvorgang nicht sofort an das SES weitergegeben. Stattdessen wird geprüft, ob die Subscription noch von anderen Endpunkten oder Regeln benötigt wird. Nur wenn dies nicht der Fall ist, wird die Subscription tatsächlich beim SES gelöscht.

Der Situation Handler verwendet die Benachrichtigungen über Situationsänderungen zudem zur Aktualisierung des Zwischenspeichers für Situationen. Wenn der Situation Handler eine Änderungsnachricht erhält, legt er einen entsprechenden Eintrag im Zwischenspeicher an. Da für sämtliche von Endpunkten behandelten Situationen eine Subscription angelegt wird, enthält der Zwischenspeicher für diese Situationen den aktuellen Zustand. Damit muss der Zustand einer Situation nur noch selten aktiv vom SES abgefragt werden.

Subscriptions auf Situationen, die von Endpunkten behandelt werden, könnten alternativ erst dann angelegt werden, wenn der Endpunkt verwendet wird, indem eine Anfrage zu ihm geschickt wird.

Damit würde der Situation Handler Benachrichtigungen über Änderungen immer noch rechtzeitig erhalten. Zudem müssten insgesamt weniger Subscriptions erstellt werden. Allerdings könnte der Zwischenspeicher dadurch nicht mehr in demselben Maße durch die Änderungsnachrichten aktuell gehalten werden und könnte veraltete Einträge enthalten. Dies könnte zur Auswahl eines ungeeigneten Endpunktes führen, wenn aus dem Zwischenspeicher eine Situation abgefragt wird, deren Zustand sich verändert hat, aber der Situation Handler über diese Änderung nicht benachrichtigt wurde. Eine Zwischenspeicherung wäre damit insgesamt nicht mehr sinnvoll, weshalb bei der Behandlung von Operationen die aktuellen Situationen stets vom SES abgefragt werden müssen. Da dies zugunsten einer schnelleren Verarbeitung (siehe oben) vermieden werden soll, werden die Subscriptions auf die behandelten Situationen sofort beim Erstellen des Endpunktes und nicht erst bei der Verwendung erzeugt.

4.6. Konfigurationsmanagement

Bei der Situationsbehandlung werden Endpunkte und Notifikationsregeln ausschließlich aus dem ER-Directory gelesen. Die dort eingetragenen Endpunkte/Regeln werden ausschließlich durch den Nutzer erstellt. Es gibt keine Möglichkeit für den Situation Handler, neue Endpunkte oder Regeln herzuleiten. Es ist daher essentiell, dass der Nutzer verschiedene Möglichkeiten zur Konfiguration erhält. Die Konfigurationsmöglichkeiten werden in Abbildung 4.10 dargestellt. Der Nutzer wird im Bild oben gezeigt. Sämtliche Konfigurationsmöglichkeiten werden über die GUI beziehungsweise über die API angeboten. GUI und API greifen auf eine eigene Konfigurationsmanagementkomponente zurück, die die Eingaben des Nutzers verarbeitet. Die Komponenten, auf die sich die Konfiguration direkt auswirkt, sind das ER-Directory und das Pluginsystem des Situation Handlers. Wie im vorigen Abschnitt beschrieben, werden dabei zusätzlich die Subscriptions für die Situationen verwaltet. Im ER-Directory werden Endpunkte und Regeln gespeichert, das Pluginsystems ist für die Verwaltung der Plugins zuständig. Damit sind sämtliche Komponenten mit Konfigurationsmöglichkeiten abgedeckt. Die sonstigen Komponenten benötigen keine Konfiguration durch den Nutzer.

Für Endpunkte gibt es die Möglichkeit völlig neue Endpunkte anzulegen und vorhandene Endpunkte zu bearbeiten oder zu löschen. Die Eigenschaften eines Endpunktes können dabei frei definiert werden. Es ist die Aufgabe des Nutzers sicherzustellen, dass alle Vorgaben eingehalten werden, damit eine sinnvolle Situationsbehandlung möglich ist. Eine inhaltliche Überprüfung der Endpunkte durch den Situation Handler kann nicht realisiert werden. Dazu müsste geprüft werden, welche Kombinationen aus Situation und Operation generell sinnvoll sind. Solches Wissen steht dem Situation Handler nicht zur Verfügung - auch wäre die Realisierung einer solchen Überprüfung selbst mit Hintergrundinformationen über Situationen und Operationen äußerst komplex. Um Mehrdeutigkeiten überprüfen zu können, wäre außerdem Wissen über die jeweiligen Implementierungen der Endpunkte erforderlich. Bei der Konfiguration wird daher lediglich sichergestellt, dass der Nutzer sämtliche erforderliche Eigenschaften für einen Endpunkt angibt. Außerdem wird überprüft, ob durch den Nutzer eine syntaktisch korrekte Adresse übermittelt wird. Eine Überprüfung, ob an dieser Adresse tatsächlich eine Funktion bereitgestellt wird, findet nicht statt.

Im Kontext der Notifikationsregeln ist es ebenfalls möglich, neue Regeln zu erstellen, beziehungsweise existierende Regeln zu bearbeiten oder zu löschen. Zudem ist es jederzeit möglich, die Aktionen,

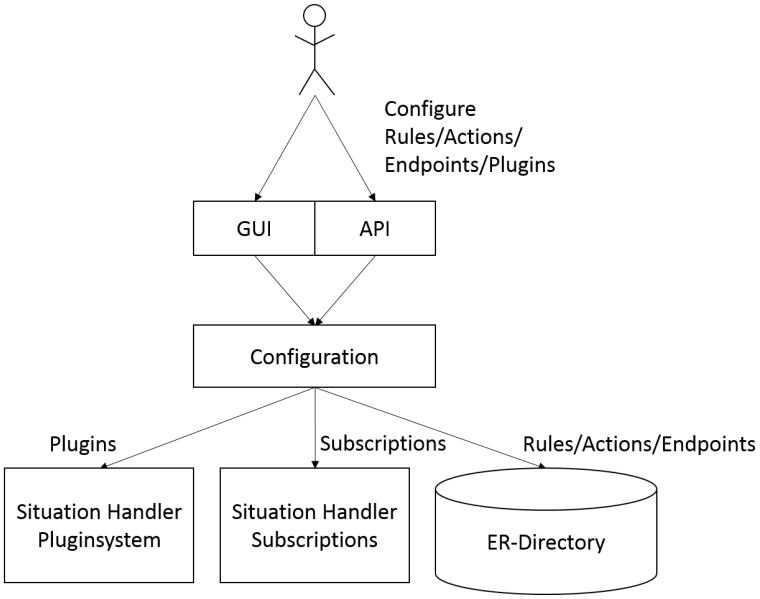


Abbildung 4.10.: Beteiligte Komponenten beim Konfigurationsmanagement.

die mit einer Regel assoziiert sind, zu bearbeiten. Auch hier können neue Aktionen erstellt und vorhandene Aktionen gelöscht oder bearbeitet werden. Zur Erstellung einer Regel reicht es zunächst aus die Situation anzugeben, in der die Regel angewandt werden soll. Dabei wird überprüft, ob für diese Situation bereits eine Regel existiert. Ist dies der Fall, werden eventuell angegebene Aktionen zur bestehenden Regel hinzugefügt. Wird die Situation einer Regel bei einer Aktualisierung verändert, wird ebenfalls auf Konflikte mit anderen Regeln geprüft. Gibt es einen Konflikt, wird die Aktualisierung der Situation unterbunden. Bei der Erstellung einer Regel wird zudem überprüft, ob alle erforderlichen Eigenschaften der Regel festgelegt wurden. Falls eine Regel gelöscht wird, werden auch alle Aktionen gelöscht, die der Regel zugeordnet sind. Beim Erstellen von Aktionen werden dem Nutzer wenige Einschränkungen gemacht. Es ich überprüft, ob das angegebene Plugin existiert und ob für dieses die erforderlichen Parameter angegeben wurden. Zudem muss ein Ausführungszeitpunkt für eine Aktion angegeben werden. Dieselben Überprüfungen werden auch bei der Veränderung einer Aktion durchgeführt. Eine Überprüfung, ob Aktionen doppelt angelegt werden, findet nicht statt.

Die letzte Konfigurationsmöglichkeit betrifft die Plugins des Pluginsystems. Hier können neue Plugins hinzugefügt und existierende Plugins entfernt werden. Um ein Plugin hinzuzufügen, muss dieses in einer eigenständigen Datei dem Situation Handler übergeben werden. Dabei muss zusätzlich der Identifikator des Plugins festgelegt werden. Dieser muss so gewählt werden, dass es zu keinen Konflikten mit anderen Plugins kommt. Dabei muss es sich um dieselbe ID handeln, die das Plugin auf Abfrage zur Laufzeit zurückliefert. Bei der Entfernung eines Plugins kann angegeben werden, ob die Aktionen, die dieses Plugin zur Ausführung benötigen, gelöscht werden sollen oder nicht. Diese Aktionen können nach der Entfernung des Plugins nicht mehr ausgeführt werden. Es wird jedoch ermöglicht die Aktionen bestehen zu lassen, um ein Update eines Plugins zu ermöglichen. Für ein Update eines Plugins muss das Plugin zuerst entfernt und danach die neue Version des Plugins

hinzugefügt werden. Würden beim Entfernen des Plugins immer alle assoziierten Aktionen gelöscht werden, müssten diese nach dem Update des Plugins neu erstellt werden.

5. Implementierung des Situation Handlers

In diesem Kapitel wird die prototypische Implementierung des im vorigen Kapitel vorgestellten Konzeptes beschrieben. Hierzu werden in Abschnitt 5.1 zunächst wichtige Technologien erklärt, die zur Implementierung verwendet wurden. Danach wird in 5.2 die Architektur des gesamten Systems beschrieben. Anschließend folgen ausgewählte Details über die Implementierung des Situation Handler Prototyps (Abschnitt 5.3). Am Ende des Kapitels werden in Abschnitt 5.4 einige Aspekte der Implementierung diskutiert.

5.1. Verwendete Technologien

Die Implementierung des Situation Handlers erfolgte in Java 8. Hierbei wurden verschiedene Frameworks und Technologien eingesetzt, die wesentlichen Einfluss auf den Aufbau und die Umsetzung des Situation Handlers hatten. Diese werden in diesem Abschnitt vorgestellt.

5.1.1. Apache Camel

Bei Apache Camel[1] handelt es sich um ein Java Framework der Apache Software Foundation[2]. Zu den Hauptfunktionen von Camel gehören das Routing von Nachrichten und die Transformation zwischen verschiedenen Nachrichtenformaten. Camel eignet sich daher zur Integration von verschiedenen Kommunikationstechnologien in einer Komponente. Da dies auch zu den Aufgaben des Situation Handlers gehört, bot sich Apache Camel an, um die verschiedenen Technologien auf eine einheitliche Weise zu integrieren.

Camel bietet Implementierungen von verschiedenen Enterprise Integration Patterns [HW12] und unterstützt zahlreiche Transportprotokolle. Dies ermöglicht die Umsetzung von häufigen Aufgaben im Bereich Routing/Mediation auf eine einheitliche Weise. Die Verwendung von Camel erfolgt über verschiedene DSLs (*Domain Specific Languages*) oder über eine Java API. Die Hauptbestandteile der Architektur von Camel sind Komponenten, Endpunkte und Prozessoren. Der Zusammenhang dieser Bestandteile wird in Abbildung 5.1 dargestellt. Die Camel-Komponenten sind in der mittleren Schicht der Abbildung dargestellt. Eine Camel-Komponente stellt eine bestimmte Kommunikationstechnologie über eine einheitliche Schnittstelle zur Verfügung. Dies bietet den Vorteil, dass auf alle Technologien, die als Komponente zur Verfügung stehen, auf die gleiche Weise über Camel zugegriffen werden kann. Damit ist keine Einarbeitung in die verschiedenen APIs der Technologien erforderlich. Ein Beispiel für

[1] `http://camel.apache.org/`
[2] `http://www.apache.org`

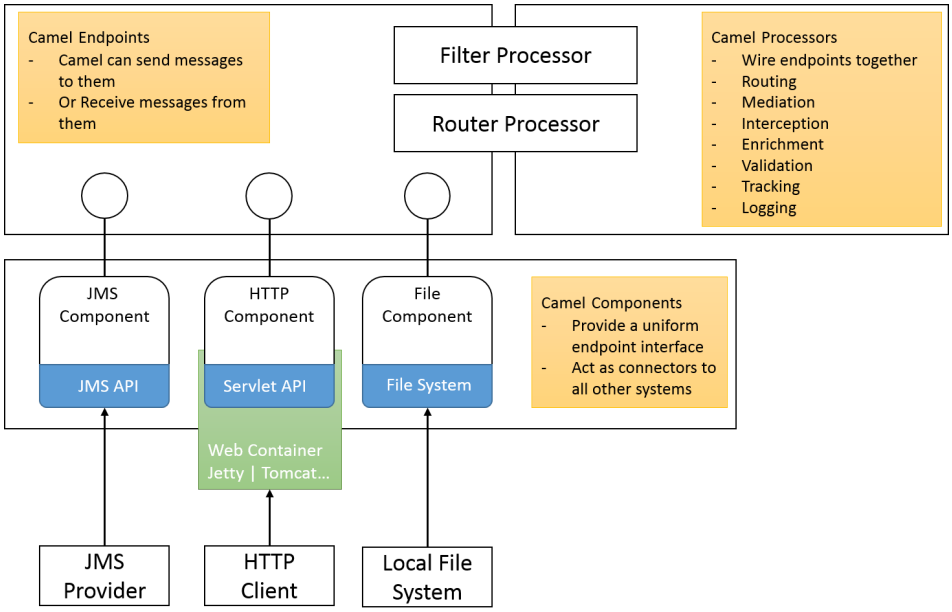


Abbildung 5.1.: Architektur Apache Camel [Apa]

eine Camel-Komponente ist die File-Komponente. Diese ermöglicht den Zugriff auf das Dateisystem und kann verwendet werden, um Dateien zu schreiben und zu lesen. Der Zugriff erfolgt zum Beispiel über die Camel-Java-DSL. Durch das Komponentenmodell ist Camel flexibel erweiterbar. Zudem müssen nicht alle verfügbaren Camel-Komponenten verwendet werden. Die Camel-Endpunkte stellen den Schnittpunkt zwischen der Anwendung, die Camel verwendet und den Camel-Komponenten dar. Ein Endpunkt kann als Konsument oder als Produzent von Nachrichten dienen. Zum Beispiel kann eine HTTP-Nachricht versendet werden, indem ein HTTP-Endpunkt definiert und die Nachricht an den Endpunkt übergeben wird. Der Endpunkt wird in diesem Fall als Produzent verwendet. Die Camel HTTP-Komponente übernimmt die Zustellung der Nachricht.

Zur Verbindung von mehreren Endpunkten werden sogenannte Routen verwendet. Auf den Routen können Camel-Prozessoren verwendet werden, um Nachrichten zu filtern oder sonstige Verarbeitungsschritte für Nachrichten durchzuführen. Auf den Routen finden insbesondere auch Formattransformationen statt. Als Endpunkte von Routen können auch Java Beans verwendet werden. Auf diese Weise können die über Camel-Endpunkte empfangene Nachrichten nahtlos an eine Java Anwendung übergeben werden.

Eine weitere Möglichkeit zur direkten Verwendung von Camel innerhalb einer Java Anwendung stellen das `Producer Template` und das `Consumer Template` dar. Hierbei handelt es sich um gewöhnliche Java Objekte, die vom Camel Framework instantiiert werden und in der Anwendung verwendet werden können. Das `Producer Template` dient zur Erzeugung von Nachrichten für einen Endpunkt. Mit dem `Consumer Template` können Nachrichten von einem Endpunkt abgefragt werden (beziehungsweise die Nachricht wird vom Endpunkt konsumiert).

5.1.2. WS-Addressing

Bei WS-Addressing [W3C] (**WSA**) handelt es sich um eine Spezifikation, bei der Mittel zur Adressierung von Web Services beschrieben werden. WSA beschreibt Möglichkeiten zur Identifikation von Web Service Endpunkten und einen Mechanismus zum Austausch der Identifikationsinformation mit SOAP-Nachrichten (vgl. [WCL+05, S.88]). WSA stellt einen Teil der WS-* Spezifikationen dar. Zur Beschreibung der Adressinformationen wird das Konzept der Endpunktreferenz (englisch: *Endpoint Reference*) eingeführt. Dies ist eine Datenstruktur, die alle Informationen beinhaltet, um einen Web Service zu adressieren und Nachrichten mit ihm auszutauschen. Der zweite Teil der WSA-Spezifikation beschreibt die Übertragung von Endpunktreferenzen in SOAP-Header. Auf eine vollständige Beschreibung der Endpunktreferenz wird an dieser Stelle verzichtet. Stattdessen werden die durch WSA definierten SOAP-Header beschrieben, da diese für die vorliegende Arbeit eine höhere Relevanz besitzen. Die Beschreibung von Endpunktreferenzen kann zum Beispiel in der WSA-Spezifikation [W3C] des W3C nachgelesen werden. Folgende SOAP-Header werden im WSA-Standard definiert (vgl. [WCL+05, S.95ff]):

Source Der Ursprung beziehungsweise der Sender der Nachricht.

To Der Empfänger der Nachricht.

ReferenceProperty Weitere Eigenschaften zur Identifikation eines Endpunktes oder der Instanz eines Endpunktes.

Reference Parameter Parameter, die nicht zur Identifikation benötigt werden.

MessageID Eine eindeutige ID der Nachricht.

ReplyTo Die Adresse, an die die Antwort gesendet werden soll.

FaultTo Die Adresse, an die im Fehlerfall eine Nachricht gesendet werden soll.

Action Die Aktion zur Verarbeitung der Nachricht.

RelatesTo Der RelatesTo-Header kann verwendet werden, um eine andere Nachricht zu referenzieren. Hierzu wird die ID der anderen Nachricht angegeben. Zudem besitzt dieser Header ein Attribut *RelationshipType*, um die Art der Beziehung zwischen den Nachrichten zu beschreiben.

Sämtliche Header bis auf *To* und *Action* sind optional.

In der WSA-Spezifikation wird außerdem ein Pattern für asynchrones Request-Reply definiert [WCL+05, S.97ff]. Dadurch wird die inhärente Synchronität bei einem Aufruf eines Web Services über HTTP aufgebrochen. Der Web Service akzeptiert hierbei die Anfrage und schickt die Antwort zu einem späteren Zeitpunkt an einen anderen Endpunkt. Die hierfür benötigten Adressinformationen werden über WSA-SOAP-Header ausgetauscht. Das von WSA definierte Pattern wird in Abbildung 5.2 dargestellt.

Die Anfrage (*Request*) wird auf der linken Seite des Bildes dargestellt - die Antwort in der Mitte. Die Anfrage muss im Header *MessageID* eine eindeutige ID enthalten und im *ReplyTo*-Header eine Antwortadresse spezifizieren. Dazu können optional *ReferenceProperties* und *ReferenceParameter*

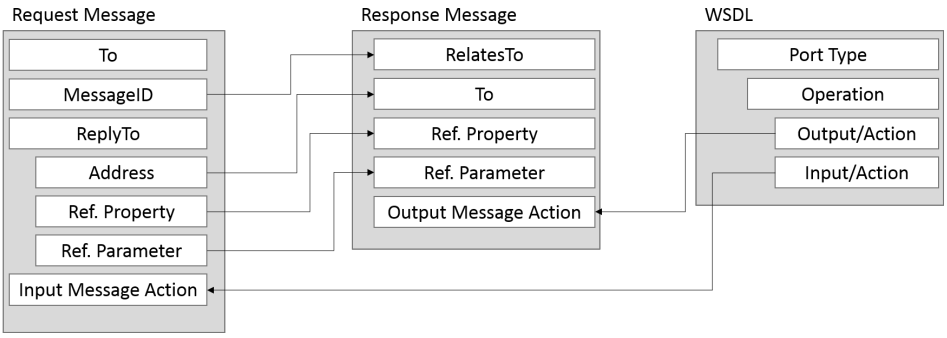


Abbildung 5.2.: WS-Addressing Request-Reply [WCL+05, S.99]

kommen. Bei der Antwort wird die *MessageID* der Anfrage in den *RelatesTo*-Header der Antwort übertragen. Für Request-Reply wird in der Antwort zudem das Attribut *RelationshipType* auf „Reply“ gesetzt [W3C]. Der *To*-Header der Antwort wird auf den Wert des *ReplyTo*-Headers der Anfrage gesetzt. *ReferenceProperties* und *ReferenceParameter* werden eins-zu-eins von der Anfrage in die Antwort übertragen. Die jeweiligen Werte für die *Action*-Header werden, wie in der Abbildung dargestellt, direkt aus der WSDL übernommen, die die Schnittstelle der Web Services definiert.

5.2. Architektur

In diesem Abschnitt wird die Architektur des Situation Handlers vorgestellt. Hierfür werden zunächst in Abschnitt 5.2.1 die eingesetzten Technologien in eine Drei-Schichten-Architektur eingeordnet und die Verbindung zwischen den Schichten wird erläutert. Im darauf folgenden Abschnitt 5.2.2 wird der Situation Handler in Komponenten aufgeteilt. Dazu erfolgt eine Einordnung der Komponenten in die Architektur.

5.2.1. Technologien in der Drei-Schichten-Architektur

Der Situation Handler setzt eine Drei-Schichten-Architektur um. Die in den verschiedenen Schichten eingesetzten Technologien werden in Abbildung 5.3 dargestellt. Die Datenschicht wird in der Abbildung in blau dargestellt, die Anwendungslogik in grün und die Präsentationsschicht in orange.

Der Situation Handler wird in der unteren Hälfte des Bildes dargestellt. In der Datenhaltungsschicht wird eine MySQL Datenbank eingesetzt. Diese dient in erster Linie zur Speicherung von Regeln und Endpunkten. Die Anwendungslogik des Situation Handlers ist in Java implementiert. Hier wird vor allem die Situationsbehandlung (Notifikationen, Aktionen und Workflows), das Pluginsystem und die weitere in Kapitel 4 beschriebene Logik umgesetzt. Zur Kommunikation mit der Datenbank wird der MySQL-JDBC-Konnektor eingesetzt. Auf der Präsentationsschicht bietet der Situation Handler mehrere Schnittstellen. Die grafische Oberfläche wird als Web-Applikation zur Verfügung gestellt. Diese besteht aus statischen HTML und Javascript Webinhalten. Durch die Oberfläche wird keine Logik implementiert - sie greift lediglich auf die REST API zu, die zur Konfiguration des Situation

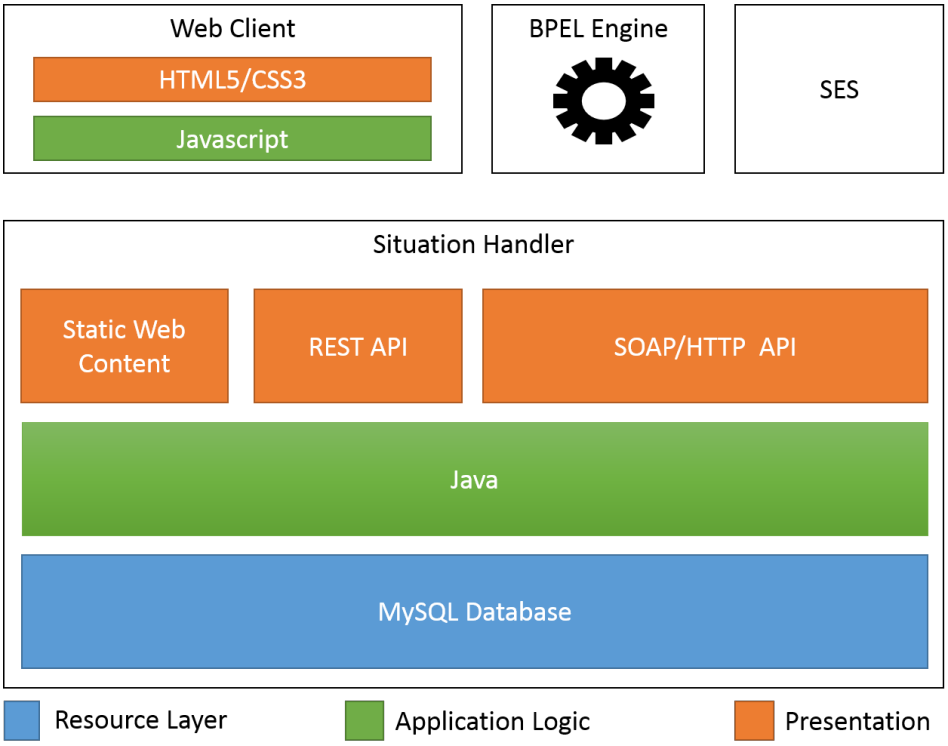


Abbildung 5.3.: Eingesetzte Technologien in der Drei-Schichten-Architektur des Situation Handlers

Handlers dient. Zudem bietet der Situation Handler mehrere HTTP-Schnittstellen, über die vor allem SOAP-Nachrichten an den Situation Handler übertragen werden.

Auf der Client Seite existiert unter anderem die Weboberfläche. Hier werden die statischen Webinhalte, die durch den Situation Handler bereitgestellt werden, in einem Browser gerendert. Die Logik des Clients dient zur Kommunikation mit der REST API des Situation Handlers und einer Überprüfung der Eingaben. Sie wird mit Javascript umgesetzt. Es wird angenommen, dass zur Modellierung von Workflows ausschließlich BPEL und zur Ausführung eine BPEL Engine verwendet wird. Bei BPEL werden SOAP-Nachrichten über HTTP zur Kommunikation mit Web Services verwendet. Der Situation Handler bietet daher eine entsprechende HTTP-Schnittstelle um SOAP-Nachrichten empfangen zu können. Auch zur Kommunikation mit dem Situationserkennungssystem bietet der Situation Handler eine HTTP-Schnittstelle.

Zur Bereitstellung der Schnittstellen wird Apache Camel verwendet. Für die SOAP/HTTP-Schnittstellen wird entweder die Camel-Jetty-Komponente[3] oder die Camel-Servlet-Komponente[4] verwendet, um verschiedene HTTP-Endpunkte zu erzeugen. Die Jetty-Komponente verhält sich hierbei wie ein einfacher Webserver und stellt einen Wrapper für einen Jetty[5] Webserver dar. Die

[3] http://camel.apache.org/jetty.html
[4] http://camel.apache.org/servlet.html
[5] http://www.eclipse.org/jetty/

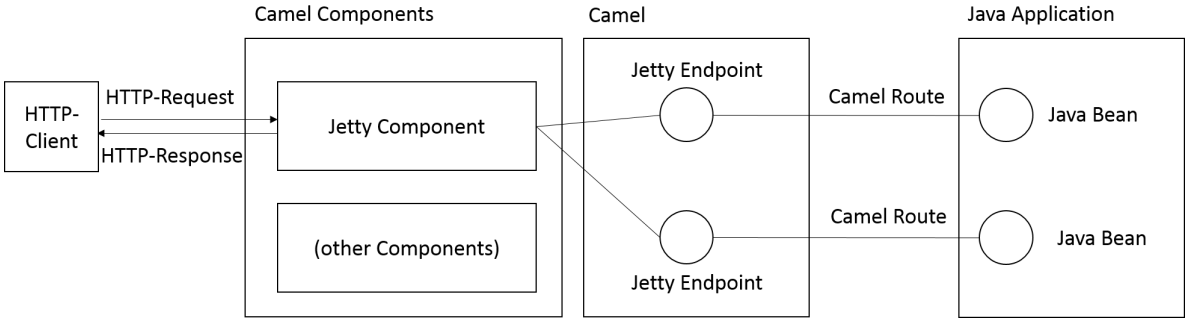


Abbildung 5.4.: Verbindung der Jetty-Komponente mit der Java Anwendung

Webinhalte zur Darstellung der Weboberfläche sowie die REST API werden ebenfalls über die Jetty-Komponente bereitgestellt. Die Verbindung der Camel-Jetty-Komponente mit der Java Anwendung wird in Abbildung 5.4 dargestellt. Das Camel Framework wird in der Mitte dargestellt. Dort werden mehrere Jetty-HTTP-Endpunkte definiert, um die verschiedenen Funktionen bereitzustellen. Die Jetty-Endpunkte werden über eine Camel-Route mit einer Java Bean verbunden, die ebenfalls einen Camel-Endpunkt darstellt. Die Java Bean wird von der Java Anwendung bereitgestellt. Die Jetty-Komponente in der Mitte links nimmt die HTTP-Anfragen der Clients entgegen. Camel übernimmt die Weiterleitung der Anfrage vom Jetty-Endpunkt zu der entsprechenden Java Bean. Dort kann die Anfrage verarbeitet und eine Antwort definiert werden. Camel leitet die Antwort zurück zur Jetty-Komponente, die aus der Antwort wiederum eine HTTP-Antwort erstellt. Wenn statt der Jetty-Komponente die Servlet-Komponente verwendet wird, bleibt das Verhalten und die Interaktion mit dem Situation Handler gleich. Bei Verwendung der Servlet-Komponente wird der Situation Handler als Java-War-Datei gepackt und kann auf einem Server betrieben werden.

Wenn die Jetty-Komponente verwendet wird, muss kein Server verwendet werden. Stattdessen dient Camel zur Bereitstellung sämtlicher HTTP-Schnittstellen. Jedoch muss beachtet werden, dass die Trennung zwischen Camel und der Java Anwendung nicht so scharf ist, wie in Abbildung 5.4 dargestellt. Vielmehr werden die Camel Routen und Komponenten innerhalb der Java Anwendung mit einer von Camel bereitgestellten DSL definiert. Die Funktionalität der Java Beans, die als Schnittstelle zwischen Camel und der Java Anwendung dienen, wird in einem von Camel definierten Kontext ausgeführt. Teile der Java Anwendung werden dementsprechend innerhalb des Camel-Kontextes ausgeführt, der wiederum beim Start der Java Anwendung erzeugt und konfiguriert wird.

5.2.2. Komponenten und Architektur

In Kapitel 4 wurden die Funktionalitäten des Situation Handlers auf konzeptioneller Ebene beschrieben. Diese Funktionalitäten sind in vielen Teilen unabhängig voneinander und greifen nur vereinzelt auf andere Funktionalitäten zurück. Der Prototyp des Situation Handlers wird bei der Implementierung dementsprechend in verschiedene Komponenten aufgeteilt. Dabei implementiert jede Komponente eine bestimmte Funktionalität. Die Komponenten können über Schnittstellen auf die Funktionalität von anderen Komponenten zugreifen und diese so verwenden. Die Architektur, die sich durch die

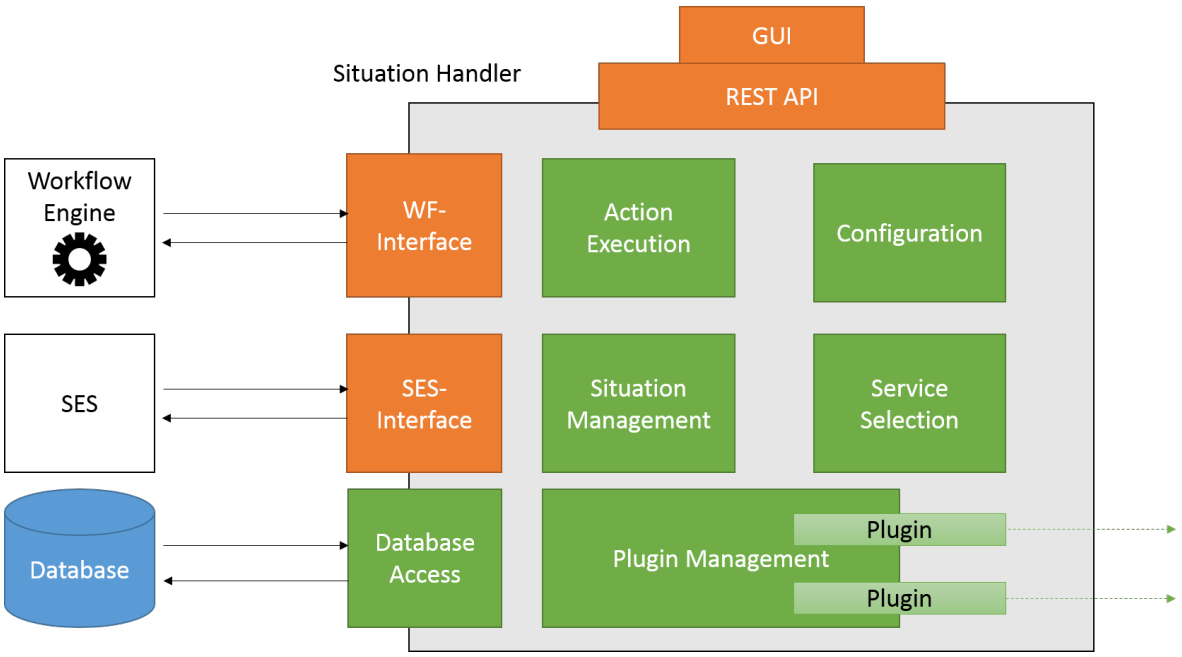


Abbildung 5.5.: Die Architektur des Situation Handlers.

Aufteilung in verschiedene Komponenten ergibt, wird in Abbildung 5.5 dargestellt. Die Komponenten sind dabei entsprechend der Schichten aus Abschnitt 5.2.1 eingefärbt. Die Komponenten der Datenschicht in blau, die Komponenten der Anwendungslogik in grün und die der Präsentationsschicht in orange.

Die Datenbank links unten im Bild wird zur Speicherung von Regeln und Endpunkten verwendet (ER-Directory, Abschnitt 4.2). Die Interaktion mit der Datenbank findet über die Datenbankzugriffskomponente statt. Die Situationsbehandlung (Abschnitt 4.3) wird auf die Komponenten *Action Execution* für den Versand von Notifikationen und *Service Selection* für die Behandlung von Workflow-Anfragen aufgeteilt. Für Workflow-Anfragen wird eine eigene HTTP-Schnittstelle (*WF-Interface*) zur Verfügung gestellt. Die Interaktion mit dem SES (Abschnitt 4.5) wird durch die Komponente *Situation Management* umgesetzt. Für Benachrichtigungen vom SES steht eine eigene HTTP-Schnittstelle zur Verfügung (*SES-Interface*). Die Konfigurationskomponente nimmt Anfragen der REST API entgegen und überprüft diese (siehe Abschnitt 4.6) bevor sie die entsprechende Konfiguration vornimmt. Die Funktionalität der GUI ist über die REST API umgesetzt. Die Plugin-Management-Komponente ist verantwortlich für die Verwaltung der Plugins (Abschnitt 4.4).

Die Komponenten, die die Anwendungslogik implementieren, sind stets auf eine ähnliche Weise umgesetzt: Für jede Komponente existiert ein (Java) Interface, das die Methoden bereitstellt, die für die anderen Komponenten relevant sind. Eine Instanz der Komponente kann über eine Factory erzeugt werden, was die Verwendung der Komponente ermöglicht. Die Factory nimmt dabei die Initialisierung der Komponente vor. Die Factory sorgt dafür, dass alle Instanzen einer Komponente denselben Zustand besitzen, sofern dies erforderlich ist. Damit kommt es zu keinen Inkonsistenzen zwischen den verschiedenen Instanzen einer Komponente. Auf diese Weise können die Komponenten

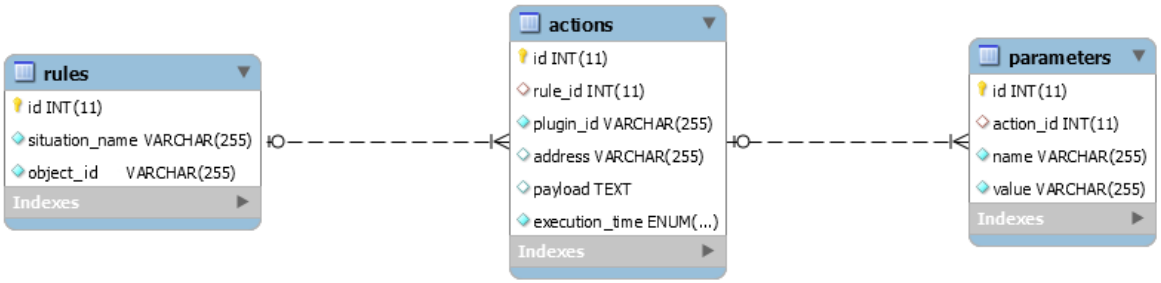


Abbildung 5.6.: Regeln Datenbankschema

weitestgehend unabhängig voneinander agieren. Zudem muss sich eine Komponente nicht darum kümmern, den Zustand einer anderen Komponente zu verwalten. Stattdessen kann über die Factory einfach eine neue Instanz der anderen Komponente bezogen werden, die den gleichen Zustand wie die alte Instanz besitzt, sofern es sich um relevante Zustandsinformation handelte. Ein Nachteil dieser Umsetzungsweise besteht darin, dass die Factories statischen Code verwenden müssen, um die Komponenten zu initialisieren und den Zustand zu verwalten.

5.3. Implementierungsdetails

In diesem Abschnitt wird auf Implementierungsdetails des Situation Handler Prototyps eingegangen. Dabei wird die Implementierung der Komponenten erklärt, die in Abschnitt 5.2.2 eingeführt wurden.

5.3.1. Datenbank und Datenbankzugriff mit Konfiguration

In Kapitel 4 wurden die Konzepte von Endpunkten und Regeln vorgestellt. Diese werden persistent in einer MySQL-Datenbank gespeichert. Das Datenbankschema zur Speicherung von Regeln wird in Abbildung 5.6 dargestellt. Für die Speicherung der Regeln werden insgesamt drei Tabellen verwendet: Die Tabelle `rules` zur Speicherung der Regeln selbst, die Tabelle `actions` zur Speicherung der Aktionen und `parameters` zur Speicherung der Parameter von Aktionen. Die Aktionen werden über einen Fremdschlüssel einer Regel zugeordnet. Die Parameter verwenden einen Fremdschlüssel zur Zuordnung zu einer Aktion. Für die Tabelle `rules` wurde für die Spalten `situation_name` und `object_id` ein einzigartiger Index (MySQL: „*UNIQUE_INDEX*“) angelegt, um sicherzustellen, dass nur eine Regel für jede Kombination aus Situationsname und Objekt ID angelegt werden kann.

Die Endpunkte werden in dem Schema aus Abbildung 5.7 gespeichert. Hierzu wird die Tabelle `endpoints` zur Speicherung des Endpunkts selbst verwendet und die Tabelle `handled_situations` zur Speicherung der durch die Endpunkte behandelten Situationen. Die behandelten Situationen verwenden einen Fremdschlüssel zur Zuordnung zu dem jeweiligen Endpunkt.

Zusätzlich zu den Endpunkten und Regeln wird in der Datenbank eine History gespeichert, in der Informationen über im Rahmen der Situationsbehandlung durchgeführten Aktionen gespeichert werden. Das Datenbankschema der History wird in Abbildung 5.8 gezeigt. Hierbei erhält jeder Eintrag

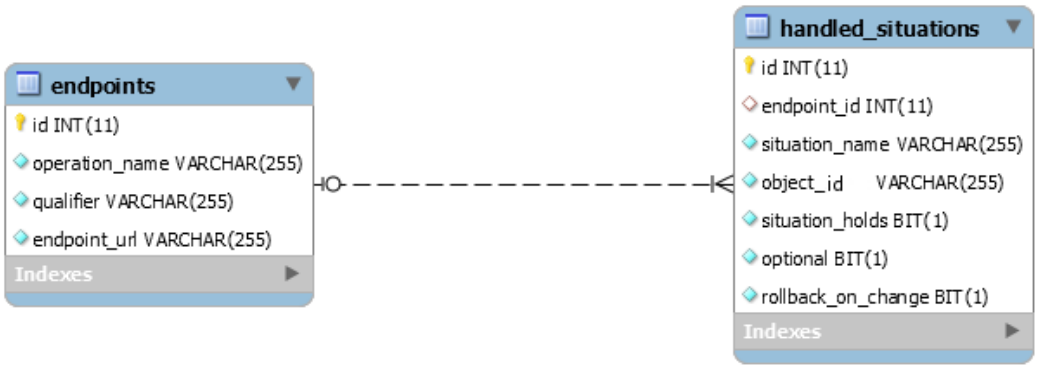


Abbildung 5.7.: Endpunkte Datenbankschema

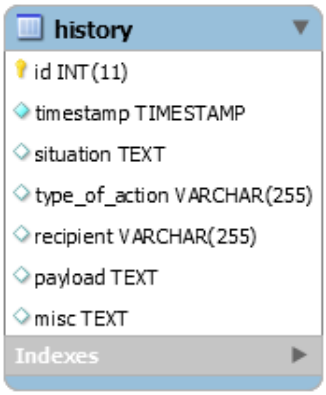


Abbildung 5.8.: History Datenbankschema

einen Zeitstempel, der von der Datenbank generiert wird. Es gibt keine festen Vorgaben darüber, wie ein Eintrag in dieser Tabelle auszusehen hat, weshalb für die meisten Felder der Datentyp `TEXT` gewählt wurde. Sämtliche Felder sind optional. Es ist im Allgemeinen jedoch sinnvoll, Angaben über die jeweilige Situation, die Art der durchgeführten Aktion und den Empfänger zu machen. Zusätzlich kann zum Beispiel der Inhalt einer Nachricht oder ein beliebiger Text zur Beschreibung einer Aktion gespeichert werden.

Allgemein werden die IDs durch die Datenbank generiert. Pflichtattribute werden umgesetzt, indem die entsprechenden Spalten mit `NOT NULL` markiert werden. Weitere Details zu den jeweils verwendeten Datentypen etc. können den Abbildungen entnommen werden.

Der Zugriff auf die Datenbank wird in der Datenbankzugriffskomponente gekapselt. Diese implementiert die Queries, die von anderen Komponenten benötigt werden, um zum Beispiel Regeln und Endpunkte abzufragen, zu ändern oder neue hinzuzufügen. Die anderen Komponenten interagieren also nicht direkt mit der Datenbank, sondern nur mit der Zugriffskomponente. Die Konfigurationskomponente ist in die Datenbankzugriffskomponente integriert, beziehungsweise dieser vorgeschaltet. Die Konfigurationskomponente kann als zusätzliche Schicht gesehen werden, durch die eine An-

frage geleitet wird, bevor sie in der Datenbank gespeichert wird. Um einen anderweitigen Zugriff einer anderen Komponente zu verhindern, stellt die Komponente öffentlich nur Interfaces zur Verfügung, sowie Factory Klassen, um eine Implementierung der Interfaces zu erhalten. Die Factories liefern stets Implementierungen der Konfigurationskomponente zurück, die wiederum auf die Datenbankkomponente zugreift. Die Konfigurationskomponente interagiert dabei vor allem mit der Situation-Management-Komponente, um Subscriptions zu erstellen, wie im konzeptionellen Teil in Abschnitt 4.5 beschrieben.

Die Überprüfungen, ob zum Beispiel neu erstellte Endpunkte oder Regeln valide sind, finden sowohl durch die Datenbank als auch durch die Zugriffskomponente statt. Die Datenbank übernimmt die Prüfungen, die direkt als Einschränkungen auf der Datenbank spezifiziert wurden (zum Beispiel Überprüfung auf *NULL*-Werte). Die Zugriffskomponente fängt in einem Fehlerfall den von der Datenbank geworfenen Fehler ab und liefert der anfragenden Komponente eine entsprechende Java `Exception` zurück. Weitere Überprüfungen, wie die Überprüfung auf syntaktische Korrektheit von Endpunkt-URLs, werden vorgenommen, bevor die Werte in der Datenbank abgelegt werden. Auch hier werden Fehler über Java `Exception`s zurückgeliefert.

Zur Kommunikation mit der MySQL Datenbank beziehungsweise zur Umsetzung von Datenbankanfragen wird die Java Persistence API (JPA) mit Hibernate eingesetzt. Dafür werden entsprechend annotierte Klassen verwendet.

5.3.2. Situationsabhängige Dienstauswahl

Die *Service-Selection*-Komponente (**SSEL-Komponente**) implementiert die Behandlung von Workflow-Operationen (Dienstaufrufe durch Workflows), wie sie in Abschnitt 4.3.2 und den folgenden Unterabschnitten beschrieben wurde. Die SSEL-Komponente bietet anderen Komponenten Schnittstellen zur Übermittlung von neuen Anfragen durch Workflows und Antworten auf Anfragen durch Workflow-Fragmente. Diese werden von der *WF-Interface*-Komponente bedient. WF-Interface wiederum verwendet Camel-Endpunkte zum Empfang von HTTP-Anfragen. Die Anfragen werden an die SSEL-Komponente weitergeleitet. Die SSEL-Komponente besitzt eine weitere Schnittstelle zum Empfang von Situationsänderungen. Diese wird von der *Situation-Management*-Komponente verwendet, um die SSEL-Komponente über Situationsänderungen zu benachrichtigen.

Der Zugriff auf die SSEL-Komponente erfolgt über das Java Interface `OperationHandler`. Eine Instanz der Komponente wird über die Factory Klasse `OperationHandlerFactory` erzeugt.

Die SSEL-Komponente erlaubt ausschließlich SOAP-Anfragen. Dabei wird vor der Verarbeitung überprüft, ob es sich um eine valide Nachricht handelt. Die Verarbeitung der SOAP-Nachricht, das heißt die Auswahl eines Endpunktes abhängig von Situation und Operation (siehe Abschnitt 4.3.3), kann parallel für mehrere Nachrichten gleichzeitig durchgeführt werden. Hierfür werden die Anfragen mit der Camel-SEDA-Komponente[6] zwischengespeichert und von mehreren Threads abgearbeitet. Bei SEDA handelt es sich um eine Architektur, welche insbesondere Wert auf hohe Parallelität legt [Wel].

[6] `http://camel.apache.org/seda.html`

Bei Camel werden zur Umsetzung der SEDA-Architektur Queues verwendet, die eine beliebige Anzahl an konsumierenden Threads erlauben. Die Threads werden vom Camel Framework verwaltet.

Zur Umsetzung der asynchronen Kommunikation wird WS-Addressing (**WSA**) verwendet. Hierbei wird das Request-Reply-Pattern aus der W3C-Spezifikatio [W3C] umgesetzt. Das Pattern wurde in Abschnitt 5.1.2 beschrieben. Die Verwendung von WS-Addressing setzt voraus, dass auch die Workflow Engines und die verwendeten Endpunkte WS-Addressing unterstützen. Die SSEL-Komponente analysiert die WSA-Header und ersetzt dort insbesondere den Wert des `To`-Headers durch den Zielendpunkt und den `ReplyTo`-Header durch die Adresse des Situation Handlers. Zudem werden für die Nachrichten neue IDs generiert und in den WSA-Headern gesetzt. Um eine korrekte Zustellung der Antworten zu ermöglichen, werden die ID und die `ReplyTo`-Adresse der original Anfragen gespeichert.

Wenn bei der Auswahl des Endpunktes ein Entscheider kontaktiert werden soll, muss dies in der Anfrage des Workflows festgelegt werden. Hierfür wurden zwei zusätzliche SOAP-Header eingeführt. Der Header mit dem Namen `Decider` legt eine Adresse des Entscheiders fest. Der Header mit dem Namen `DecisionDescription` kann verwendet werden, um die Entscheidung für den Entscheider näher zu beschreiben. Die Entscheidungsnachricht wird als Android-Push-Benachrichtigung versandt. Zum Versand und Empfang dienen das Android-Plugin und die zugehörige Android App (siehe Abschnitt 5.3.7). Da der Versand der Nachricht über *Google Cloud Messaging* [Goo] (**GCM**) erfolgt, muss zur Adressierung des Empfängers ein GCM-Token verwendet werden. Dies wird durch die zugehörige Android App generiert. Dem Entscheider wird eine Liste der möglichen Endpunkte sowie der festgelegte Text geschickt. Der Entscheider wählt einen der Endpunkte aus. Die Auswahl wird als HTTP-Post zurück an den Situation Handler geschickt. Für den Empfang der Antwort wird mit Apache Camel ein HTTP-Endpunkt geöffnet, der nach dem Empfang der Antwort geschlossen wird. Über die URL des HTTP-Endpunktes erfolgt eine Zuordnung der Antwort zur Anfrage. Um die Zuordnung zu ermöglichen, wird die ID der Anfrage in die URL aufgenommen. Falls für den Entscheider keine Adresse festgelegt wird, wird ein zufälliger Endpunkt aus der Liste der gleichwertigen Kandidaten gewählt.

Wenn sich Situationsänderungen ergeben während ein Endpunkt/Workflow-Fragment eine Anfrage verarbeitet, muss unter Umständen ein Rollback eingeleitet werden. Ein Rollback muss eingeleitet werden, wenn sich der Zustand einer Situation ändert und der Endpunkt einen Rollback für diese Situationsänderung vorschreibt. Um Rollbacks zu ermöglichen, wird von der SSEL-Komponente vor der Weiterleitung der Anfrage an einen Endpunkt überprüft, ob ein Rollback in einer bestimmten Situation nötig ist. Ist dies der Fall, wird für die Situation ein `RollbackHandler` erstellt, welcher sich auf den gewählten Endpunkt und die Situationen, die diesem Endpunkt zugeordnet sind, bezieht. Erhält die SSEL-Komponente die Information über eine Situationsänderung, werden sämtliche dieser Situation zugeordneten `RollbackHandler` gestartet. Der `RollbackHandler` schickt eine Rollback-Nachricht an den ihm zugeordneten Endpunkt. Der Endpunkt schickt nach Abschluss des Rollbacks eine Antwort an den Situation Handler zurück. Der Situation Handler beginnt den Vorgang zur Auswahl eines Endpunktes daraufhin von vorne. Hierbei wird kein neuer `RollbackHandler` erstellt, sondern der alte verwendet. Dieser zählt die Anzahl der durchgeführten Rollbacks. Wird die Anzahl der maximalen Rollbacks erreicht, wird eine Fehlernachricht an den Anfragesteller gesandt.

Für die Rollback-Anfragen und Antworten wurden eigene SOAP-Nachrichten definiert. Bei einer Rollback-Nachricht handelt es sich um eine SOAP-Nachricht, die nur das Element `RollbackRequestElement` beinhaltet. Die Nachricht wird an denselben (WSDL) Port geschickt,

wie die ursprüngliche Anfrage. Das heißt, dass der WSDL `PortType`, der die eigentliche Operation anbietet, auch die Rollback-Operation anbieten muss. Diese muss den Namen `startRollback` tragen. Zur Adressierung wird WSA verwendet. Der WSA-*RelatesTo*-Header wird verwendet, um die ursprüngliche Anfrage zu referenzieren. Der Header erhält hierzu als Wert die Nachrichten ID der Anfrage, für die ein Rollback durchgeführt werden soll. Das Attribut `RelationshipType` wird auf den Wert „Rollback“ gesetzt. Die Rollback-Antwort beinhaltet nur das Element `startRollbackResponse`. Der WSA-*RelatesTo*-Header referenziert die Rollback-Anfrage. Als *RelationshipType* wird hierbei „RollbackResponse“ verwendet. Sowohl bei der Rollback-Anfrage als auch bei der Antwort werden im SOAP-Body die ID der Anfrage versandt, auf die sich der Rollback bezieht. In der Antwort existiert ein zusätzliches Element, welches den Erfolg des Rollbacks beschreibt. Jedoch wird angenommen, dass ein Rollback immer erfolgreich ist - ein erfolgloser Rollback wird nicht speziell gehandhabt. Die Erfolgsnachricht dient lediglich dazu, die Vorgänge bei einem Rollback-Vorgang besser nachvollziehen zu können.

Es gibt verschiedene Situationen, in denen der Situation Handler eine Fehlernachricht an den Anfragesteller schickt. Dies ist insbesondere der Fall, wenn die maximale Anzahl an Rollbacks erreicht wurde, bei der Auswahl des Endpunktes kein Ergebnis gefunden wurde oder ein Endpunkt nicht erreicht werden kann. Da asynchrone Kommunikation verwendet wird, können die Fehlernachrichten nicht in einer WSDL Datei als Faults definiert werden - Faults sind nur für „in/out“-Operationen möglich, während für asynchrone Kommunikation stets „in“-Operationen verwendet werden. Daher muss der Anfragesteller, die Operation `situationHandlerFault` implementieren, wenn er Fehlernachrichten empfangen möchte. Die Fehlernachrichten verwenden ein vom Situation Handler vorgegebenes Nachrichtenformat. Eine Fehlernachricht beinhaltet die ID der Anfrage, die den Fehler ausgelöst hat, eine Nachricht, die den Fehler näher beschreibt und einen SOAP-Fault-Code[7]. Außerdem muss vom Anfragesteller der Wert des WSA-*FaultTo*-Headers auf die gewünschte Antwortadresse gesetzt werden. Zudem kann ein weiterer Header (`FaultCorrelationInfo`) gesetzt werden, um eine ID für Nachrichtenkorrelation zu spezifizieren. Der Situation Handler verarbeitet diesen Header und setzt die ID im Body der Fehlernachricht als ID der Anfrage. Existiert der `FaultCorrelationInfo` Header nicht, wird stattdessen die im WSA-*MessageID*-Header spezifizierte ID verwendet.

Ein Workflow kann für eine Anfrage eine maximale Anzahl an Rollbacks beziehungsweise erneuten Versuchen festlegen. Die Anzahl kann in einem vom Situation Handler definierten SOAP-Header festgelegt werden. Dieser trägt den Namen „MaxRetries“. Das `Actor`-Attribut muss auf den Wert „Situation_Handler“ gesetzt werden. Der Wert des Headers ist eine Zahl größer oder gleich Null. Der Header wird nach der Verarbeitung durch die SSEL-Komponente aus der Nachricht entfernt. Ein Beispiel für den Header (mit maximal zwei erneuten Versuchen nach Rollback) sieht aus wie folgt:

```
<MaxRetries soapenv:actor = "Situation_Handler"> 2 </MaxRetries>
```

Die Informationen über behandelte Anfragen sowie über durchgeführte Rollbacks werden in der History vermerkt.

[7] `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383510`

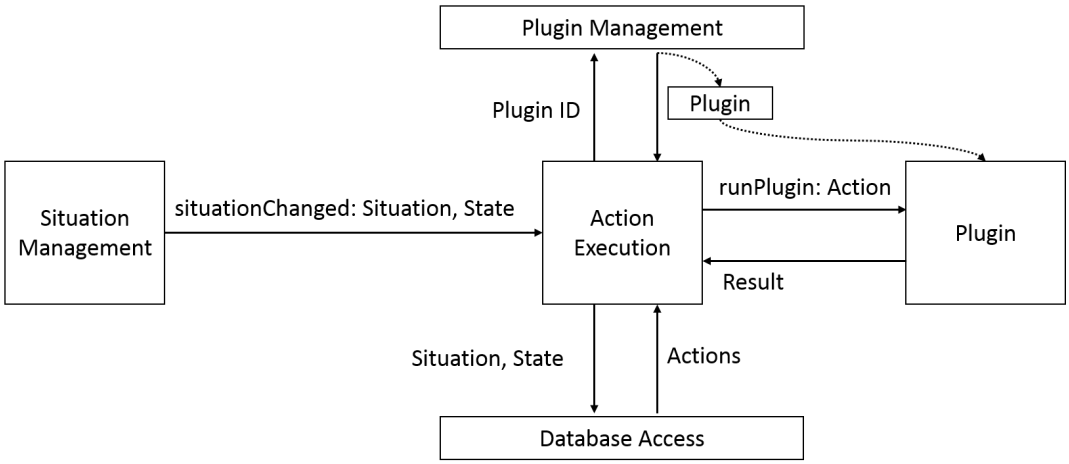


Abbildung 5.9.: Interaktion der Action-Execution-Komponente mit anderen Komponenten

5.3.3. Situationsbedingte Aktionsausführung und Notifikationsversand

Die *Action-Execution*-Komponente implementiert den situationsabhängigen Versand von Notifikationen beziehungsweise die Ausführung der entsprechenden Aktionen. Wie genau Situationsänderungen hier behandelt werden, wurde bereits in Abschnitt 4.3.1 beschrieben. Hierfür ist die Interaktionen mit anderen Komponenten erforderlich. Diese ist in Abbildung 5.9 dargestellt. Die einzige Funktion, die die Action-Execution-Komponente nach außen anbietet ist eine Möglichkeit, um die Komponente über Situationsänderungen zu benachrichtigen. Diese wird, wie in der Abbildung auf der linken Seite dargestellt, von der Situation-Management-Komponente verwendet, sobald diese die Information über eine Situationsänderung erhält. Dabei wird der Komponente die Situation sowie der neue Zustand der Situation („liegt vor“ oder „liegt nicht vor“) mitgeteilt. Die vorhandenen Aktionen werden aus der Datenbank über die Komponente zur Datenbankkommunikation abgefragt. Diese bietet eine Möglichkeit, um die Aktionen abzufragen, die bei einer bestimmten Situationsänderung auszuführen sind. Die Filterung der Aktionen findet durch eine entsprechende SQL-Query auf Datenbankebene statt. Eine Aktion spezifiziert das Plugin, mit welchem diese ausgeführt werden soll. Für jede Aktion wird von der Pluginkomponente das entsprechende Plugin angefordert und dieses wird anschließend ausgeführt. Die Funktionsweise der Pluginkomponente sowie die Ausführung von Plugins wird in Abschnitt 5.3.4 beschrieben. Die Ausführung der Aktionen beziehungsweise Plugins geschieht dabei auf parallele und asynchrone Weise unter Verwendung des Java `ExecutorServices`[8]. Dieser wird in diesem Fall nicht selbst durch die Anwendung erstellt, sonder von Camel angefordert. Die Threads werden bei dieser Vorgehensweise von Camel verwaltet. Das Ausführungsergebnis, dass das Plugin zurückliefert, wird in der History vermerkt. Hierfür wird ein extra Thread verwendet, der auf das Ergebnis des Plugins wartet und dieses über die Datenbankkomponente in der History vermerkt. Der Zugriff auf die Action-Execution-Komponente erfolgt über das `NotificationComponent` Interface. Eine Instanz der Komponente wird über die `NotificationComponentFactory` erzeugt.

[8] `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html`

5.3.4. Plugin Management

Die Plugin-Management-Komponente ist zuständig für die Verwaltung von existierenden Plugins und die Bereitstellung von Plugins zur Verwendung für andere Komponenten. Für andere Komponenten werden folgende Möglichkeiten geboten:

- Abfrage von Informationen über existierende Plugins
- Anforderung eines Plugins zur Verwendung
- Das Hinzufügen von neuen Plugins und das Entfernen von existierenden Plugins

Zugriff auf die Plugin-Management-Komponente erfolgt über das Java Interface `PluginManager`. Eine Implementierung des Interfaces kann für andere Komponenten über die `PluginManagerFactory` bereitgestellt werden. Die Factory sorgt zudem für die Initialisierung der Komponente und führt beim Beenden der Komponente Aufräumaufgaben durch, wie das Freigeben von verwendeten Ressourcen.

Zur Implementierung der Plugin-Verwaltung wird der Java `ServiceLoader`[9] verwendet. Hierbei handelt es sich um eine Java Klasse, die das einfache Laden von Erweiterungen (Services) in eine Java Anwendung ermöglicht. Ein Service wird durch ein Java Interface oder eine abstrakte Klasse definiert. Ein Service Provider implementiert dieses Interface oder leitet von der abstrakten Klasse ab. Die Implementierung des Services muss als Jar-Datei zur Verfügung gestellt und zur Laufzeit der Hauptanwendung von dem Service Loader geladen werden. In der Jar-Datei muss sich zudem eine Textdatei befinden, in der der vollständige Namen der Klasse angegeben wird, die den Service implementiert.

In diesem Fall wurde ein Interface für Plugins definiert, in dem die Methoden vorgegeben sind, die ein Plugin implementieren muss. Über das Interface können vor allem die Informationen abgefragt werden, die ein Plugin zur Verfügung stellen muss und es wird eine Möglichkeit geboten, das Plugin auszuführen. Außerdem muss eine Methode zum „Cleanup“ beim Herunterfahren eines Plugins implementiert werden. Ein Plugin stellt folgende Informationen zur Verfügung: eine ID, einen Namen, eine Beschreibung der optionalen Parameter und eine Anleitung zur Verwendung. Die ID kann vom Plugin selbst gewählt werden und muss global eindeutig sein. Die ID sollte daher die Form *com.example.examplePlugin* besitzen. Die Beschreibung der optionalen Parameter besteht aus einer Menge von Strings. Um ein Plugin auszuführen, kann von diesem eine Implementierung des Java Interfaces `Callable`[10] angefordert werden. Die `call()` Methode der Implementierung sollte das Plugin starten. Auf diese Weise kann eine andere Komponente des Plugin verwenden, indem die `Callable`-Implementierung beispielsweise in einem `Thread` ausgeführt wird. Wenn die Ausführung des Plugins bei der Plugin-Management-Komponente angefordert wird, können dort als Parameter ein Empfänger, ein Nachrichteninhalt sowie die optionalen Parameter festgelegt werden. Die optionalen Parameter werden in einem Wrapper für eine Java `Map`[11] übergeben. Hierbei werden als Schlüssel die Beschreibungen der Parameter verwendet, die vom Plugin selbst spezifiziert werden. Der Wert des Parameters wird vom Nutzer festgelegt. Die Parameter werden an das Plugin übergeben und

[9] `https://docs.oracle.com/javase/8/docs/api/java/util/ServiceLoader.html`
[10] `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html`
[11] `https://docs.oracle.com/javase/8/docs/api/java/util/Map.html`

dieses erzeugt eine Instanz von `Callable`. Die `Callable`-Implementierung kann als Antwort eine *Map* zurückliefern, die die Antwortparameter enthält. Die Antwortparameter werden ebenfalls frei vom Plugin spezifiziert. Zu beachten ist, dass die Antwort in der Regel nicht gesondert verarbeitet wird, sondern für die History beziehungsweise Logs dient.

Die Plugins werden der Plugin-Management-Komponente als Jar-Datei in einem von der Komponente definierten Ordner auf dem Dateisystem zur Verfügung gestellt. Plugins, die sich in diesem Ordner befinden, werden beim Start der Komponente geladen. Dem Java `Service Loader` muss ein `Classloader` zur Verfügung gestellt werden, der die Klassen dieser Jar-Dateien lädt. In diesem Fall wurde ein `URLClassLoader`[12] verwendet, dem die URLs sämtlicher verwendeter Jar-Dateien übergeben werden. Das Hinzufügen von neuen Plugins kann über diesen so geregelt werden, dass die URL der neuen Jar-Datei dem Classloader hinzugefügt wird. Zum Hinzufügen eines neuen Plugins wird der Komponente die ID des neuen Plugins sowie eine URL übergeben, die auf die Jar-Datei verweist. Die Jar sollte sich hierbei an einem Ort im Dateisystem befinden. Beim Hinzufügen des Plugins wird die Jar-Datei von der Plugin-Management-Komponente in ein eigenes Verzeichnis kopiert. Das Löschen eines Plugins gestaltet sich schwieriger, da das Laden einer Klasse in Java nicht einfach rückgängig gemacht werden kann. Dies ist jedoch wünschenswert, um nicht unnötig viele „alte“ Klassen geladen zu haben. Eine Klasse wird erst vom Garbage Collector entfernt, wenn sämtliche Instanzen der Klasse sowie der Classloader selbst vom Garbage Collector entfernt wurden. Daher wird beim Entfernen eines Plugins der Classloader, der für die Plugins zuständig ist, ausgetauscht. Der neue Classloader lädt sämtliche alte Plugins bis auf das, das entfernt wurde. Zudem wird beim Entfernen des Plugins die entsprechende Jar-Datei vom Dateisystem gelöscht. Hier muss unter Umständen darauf gewartet werden, bis der nicht mehr verwendete Classloader vom Garbage Collector entfernt wurde, da die Jar-Datei ansonsten immer noch als geöffnet gilt und das Betriebssystem die Löschung der Datei blockiert.

5.3.5. Situation Management

Die *Situation-Management*-Komponente ist für die Interaktion mit dem Situationserkennungssystem zuständig. Sie implementiert die Interaktion mit dem SES, die in Abschnitt 4.5 beschrieben wurde. Dazu gehört insbesondere auch die Verwaltung des Zwischenspeichers für Situationen (*Situationscache*) und die Verwaltung der Subscriptions auf der Seite des Situation Handlers. Zugriff auf die Situation-Management-Komponente erfolgt über die `SituationManagementFactory` beziehungsweise das `SituationManager` Interface. Über die Factory kann zudem das Herunterfahren der Komponente initiiert werden. Außerdem lässt sich darüber der Cache aktivieren und deaktivieren. Für andere Komponenten bietet die Situation-Management-Komponente die Möglichkeit, eine Situation aktiv abzufragen und neue Subscriptions für Situationen zu erstellen oder bestehende Subscriptions zu entfernen. Die Situation-Management-Komponente leitet Nachrichten über Situationsänderungen außerdem an die Service-Selection-Komponente und die Action-Execution-Komponente weiter. Die Kommunikation mit dem SES erfolgt über eine REST API, die vom SES zur Verfügung gestellt wird. Der Zustand einer Situation kann über eine HTTP-GET-Anfrage abgefragt werden. Als Query-Parameter werden für diese Anfrage die ID des Situation Templates und die ID des Dinges angegeben. Diese

[12] `https://docs.oracle.com/javase/8/docs/api/java/net/URLClassLoader.html`

beiden Werte identifizieren eine Situation eindeutig (siehe Abschnitt 3.3). Eine Subscription wird über eine POST-Anfrage erstellt. Als Query-Parameter werden hier erneut die ID des Templates und die des Dinges zur Identifikation der Situation sowie eine Callback-URL verwendet. Die Callback-URL muss auf einen HTTP-Endpunkt verweisen. An diesen werden die Nachrichten über Situationsänderungen vom SES geschickt. Für die Situation-Management-Komponente wird hierfür über Apache Camel ein eigener HTTP-Endpunkt erzeugt (entspricht dem SES-Interface aus Abbildung 5.5). Dieser Endpunkt wird innerhalb einer Camel-Route verwendet, die als Ziel eine Java Bean (`SituationEndpoint`) besitzt. Die Java Bean „parst“ die Nachricht des SES und leitet die Information über die Situationsänderung an die Service-Selection-Komponente und die Action-Execution-Komponente weiter. Außerdem nimmt sie ein Update des Situationscaches vor, falls dieser aktiviert ist. Die Anfragen an das SES werden mit der Camel-HTTP-Komponente erzeugt. Dabei wird keine Route definiert, sondern eine Nachricht mit dem Camel *Producer Template* versandt.

Bei der Verwaltung der Subscriptions wird vor allem darauf geachtet, dass keine überflüssigen Subscribe-Nachrichten geschickt werden und dass die Subscriptions erst endgültig gelöscht werden, wenn Informationen über den Zustand dieser Situation keinesfalls mehr benötigt werden. Daher wird für jede Subscription beim ersten Subscribe ein Wrapper erstellt, der die Anzahl der Subscriptions für diese Situation zählt. Erhält die Situation-Management-Komponente einen weiteren Subscribe-Auftrag für diese Situation, wird der Zähler erhöht. Geht ein Unsubscribe-Auftrag einer anderen Komponente ein, wird der Zähler dekrementiert. Der Wrapper schickt bei seiner Erzeugung eine Subscribe-Anfrage an das SES. Wenn der Zähler des Wrappers den Wert 0 erreicht, schickt der Wrapper eine Unsubscribe-Anfrage an das SES. Die (Un)Subscribe-Aufträge erhält die Situation-Management-Komponente in der Regel, wenn neue Endpunkte und Regeln erzeugt beziehungsweise vorhandene gelöscht werden. Es ist jedoch möglich, dass bereits Regeln und Endpunkte in der Datenbank existieren, wenn der Situation Handler gestartet wird. Daher wird zu Beginn über die Datenbankzugriffskomponente abgefragt, welche Regeln und Endpunkte existieren. Für die dort verwendeten Situationen werden in diesem Schritt Subscriptions erstellt. Beim Shutdown der Situation-Management-Komponente werden sämtliche existierende Subscriptions beim SES gelöscht.

Zur Implementierung des Caches wird die `LRUMap`[13] aus der Bibliothek Apache Commons-Collections4 verwendet. Für die LRUMap lässt sich eine maximale Kapazität festlegen. Als Ersetzungsstrategie wird Least Recently Used (LRU) verwendet. Der Cache wird aktualisiert, wenn eine Situation aktiv vom SES abgefragt wird und wenn eine Änderungsnachricht aufgrund einer Subscription eingeht. Da für alle verwendeten Situationen Subscriptions vorliegen, wird der Cache auf diese Weise stets aktuell gehalten und kann keine veralteten Einträge enthalten. Eine Aktualisierung des Caches findet nur dann statt, wenn der Cache aktiviert ist. Wenn der Cache deaktiviert wird, wird er geleert. Dies vermeidet, dass der Cache veraltete Einträge enthält, wenn er erneut aktiviert wird.

5.3.6. REST API und GUI

Die REST API und die GUI dienen zur Interaktion mit dem Nutzer. Hierunter fällt die Konfiguration von Regeln, Endpunkten und Plugins, wie im konzeptionellen Teil in Abschnitt 4.6 beschrieben.

[13] `https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/map/LRUMap.html`

Zur Umsetzung der REST API wurde Apache Camel verwendet. Für REST-Schnittstellen bietet Camel die REST DSL[14], die die einfache Beschreibung von REST-Schnittstellen innerhalb einer Java Anwendung erlaubt. Die Schnittstellen werden über die Jetty-Komponente bereitgestellt. Ein Beispiel für die Definition einer REST-Operation mit der REST DSL sieht aus wie folgt:

```
rest("/config/endpoints").get().outTypeList(Endpoint.class)
                                    .to("bean:endpointApi?method=getEndpoints");
```

Dieser Auszug stammt aus der Implementierung der REST API und ermöglicht die Abfrage aller vorhandenen Endpunkte. Dabei wird nach dem Schlüsselwort `rest` der Pfad definiert. Außerdem wird festgelegt, dass es sich um eine HTTP-GET-Operation handelt. Das Ergebnis ist eine Liste, die Elemente vom Typ `Endpoint` enthält. Die Anfrage wird an die Java Bean `endpointApi` (Name der Bean in der Camel Registry) und dort an die Methode `getEndpoints` weitergeleitet. Diese Methode verarbeitet die Anfrage und erzeugt eine Antwort. Die Antwort wird automatisch durch die Camel-Jackson-Komponente[15] in das JSON-Format umgewandelt.

Bei der GUI handelt es sich um eine Web-Anwendung, die mit dem Javascript Framework AngularJS realisiert wurde. Da die Web-Anwendung ausschließlich aus statischen Artefakten besteht (HTML und Javascript), kann sie auf einem einfachen HTTP-Server bereitgestellt werden. In diesem Fall übernimmt dies ebenfalls der Situation Handler mithilfe der Camel-Jetty-Komponente. Die Web-Anwendung greift dabei ausschließlich auf die REST API zu. Sie könnte somit auch auf einem anderen Server als der Situation Handler betrieben werden. Einige Screenshots der Oberfläche befinden sich in Anhang A.2.

5.3.7. Beispiel Plugins

Bei den Plugins, die mit dem Situation-Handler-Prototypen zur Verfügung gestellt werden, handelt es sich um ein HTTP-Plugin, ein E-Mail-Plugin und ein Android-Plugin.

Das HTTP-Plugin besitzt einen sehr hohen Stellenwert, da der Situation Handler es selbst zur Kommunikation mit Workflow-Fragmenten verwendet. Außerdem ist es mit dem Plugin möglich, bei Situationsänderungen Workflow-Fragmente auszuführen, wodurch *Situation Events* realisiert werden können, wie von Breitenbücher et al. [BHK+15] beschrieben. Das HTTP-Plugin bietet die Möglichkeit, für Anfragen beliebige Querystrings und HTTP-Header zu definieren. Es können GET und POST Anfragen ausgeführt werden. Die zu verwendende HTTP-Methode, Query-Parameter und Header können als optionale Parameter übergeben werden. Zur Umsetzung verwendet das Plugin die Bibliothek Apache HttpComponents[16].

Das E-Mail-Plugin ermöglicht den Versand von E-Mails von einer festen E-Mail-Adresse an einen beliebigen Empfänger. Als optionaler Parameter kann der Betreff spezifiziert werden. Zum Versand der E-Mails wird die JavaMail API verwendet. Der Prototyp des E-Mail-Plugins kann E-Mails über beliebige SMTP-Server versenden, sofern das Plugin korrekt konfiguriert wird.

[14] `http://camel.apache.org/rest-dsl.html`
[15] `http://camel.apache.org/json.html`
[16] `https://hc.apache.org/`

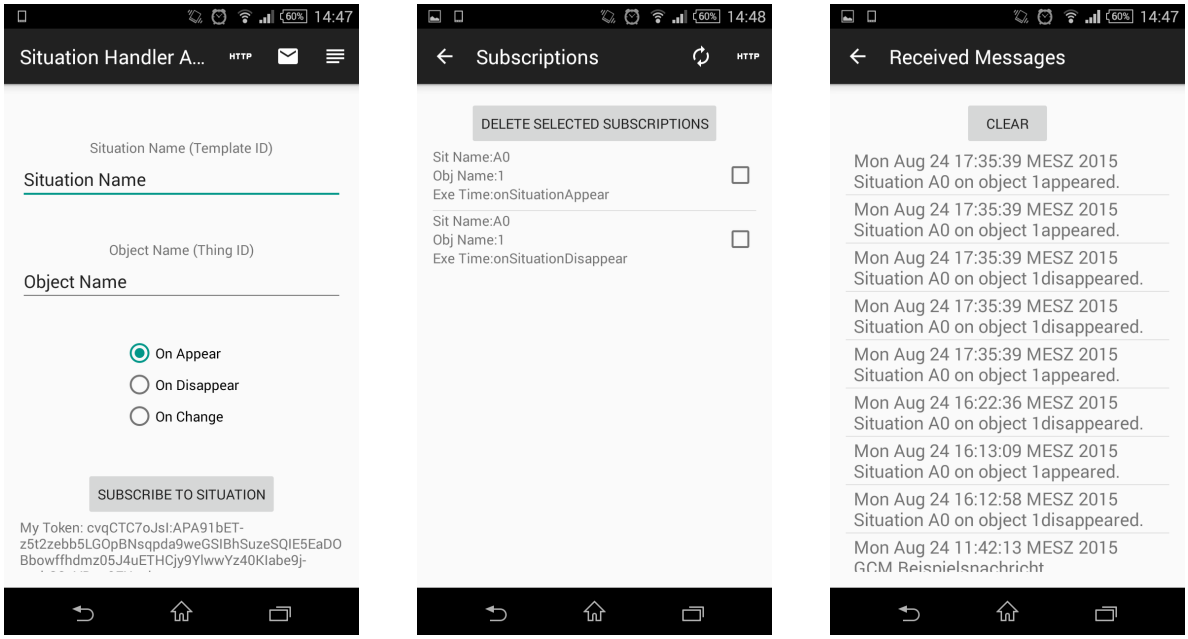


Abbildung 5.10.: Die Android App zum Empfang von Notifikationen.

Das Android-Plugin ermöglicht den Versand von Push-Nachrichten an Android-Geräte mithilfe von Google Cloud Messaging [Goo] (**GCM**). Der Versand einer Nachricht durch das Android-Plugin erfolgt über eine HTTP-POST Nachricht an einen GCM-Server. Zum Empfang der Nachrichten wurde eine Android App entwickelt. Für die Verwendung von GCM auf dem Gerät, müssen auf dem Gerät die Google Play Services installiert sein. Diese sind auf den meisten Geräten standardmäßig durch den Google Play Store installiert. Die Android App ermöglicht zudem das Erzeugen von Aktionen, die den Versand einer Nachricht an dieses Gerät nach sich ziehen. Die Adressierung des Gerätes erfolgt mithilfe eines Tokens, das dem Gerät durch GCM zur Verfügung gestellt wird. Die Android App ermöglicht zudem das Scannen von QR-Codes. Durch den Scan eines QR-Codes kann eine Subscription erstellt oder eine Situationsänderung ausgelöst werden. Zur Auslösung einer Situationsänderung wird mit der REST API des SES kommuniziert. Screenshots der Android App sind in Abbildung 5.10 zu sehen. Die Ansicht links in der Abbildung ermöglicht das Erzeugen von Aktionen (Subscriptions). Die Ansicht in der Mitte zeigt alle Subscriptions für dieses Gerät beziehungsweise diese Instanz der App. In der Ansicht rechts werden die letzten empfangenen Notifikationen angezeigt.

5.4. Diskussion

In diesem Kapitel werden einige grundsätzliche Entscheidungen bei der Implementierung des Situation Handlers und der Integration mit dem SES diskutiert. Der Situation Handler arbeitet mit Workflows zusammen. Die Implementierung beschränkt sich dabei auf die Einbindung des Situation Handlers in BPEL-Workflows, bei denen mit SOAP-Nachrichten kommuniziert wird. Es existieren jedoch weitere ausführbare Workflow-Sprachen, die ebenfalls hätten berücksichtigt werden können,

wie zum Beispiel YAWL[17]. Aufgrund der momentanen Dominanz von BPEL in diesem Bereich, kann jedoch davon ausgegangen werden, dass die Einschränkung auf BPEL irrelevant für die meisten Anwendungsfälle ist. Zudem kann angenommen werden, dass der Situation Handler kompatibel zu anderen Workflows ist, solange diese SOAP und WS-Addressing zur Kommunikation einsetzen. Der Situation Handler nutzt keine BPEL spezifischen Eigenschaften aus, sondern kann standardmäßig mit allen durch WSDL spezifizierten Schnittstellen mit SOAP-Binding kommunizieren. Eine Erweiterung des Situation Handlers für andere Kommunikationstechnologien wäre außerdem möglich - die situationsabhängige Auswahl von Endpunkten ist unabhängig von einer bestimmten Workflow-Technologie. Eine Erweiterung um bestimmte Funktionen könnte auch aufgrund des Pluginsystems einfach durchgeführt werden. Zudem sollte berücksichtigt werden, dass es sich bei dem Situation Handler um eine prototypische Umsetzung handelt, die nicht zwingend für einen produktiven Einsatz geeignet ist. Das in Kapitel 4 beschriebene Konzept gilt unabhängig von einer bestimmten Technologie.

Bei der Integration mit dem Situationserkennungssystem muss berücksichtigt werden, dass hier unter Umständen Verzögerungen bei der Erkennung und Weitergabe von Situationsänderungen entstehen. Das Situationserkennungssystem wertet die Sensordaten im jetzigen Stand einmal pro Sekunde aus und meldet dementsprechend das Auftauchen oder Verschwinden von Situationen. Folglich kann eine Verzögerung von bis zu einer Sekunde entstehen, bis die Benachrichtigung über die Situationsänderung beim Situation Handler ankommt. Abhängig von den Latenzen bei der Kommunikation, kann sich diese Verzögerung noch weiter erhöhen. Wenn im Zeitraum zwischen der tatsächlichen Änderung und der Erkennung dieser Änderung eine Fragmentselektion durchgeführt wird, wird dort unter Umständen ein Endpunkt ausgewählt, der nicht für die real vorliegende Situation geeignet ist. Dieses Problem kann durch den Situation Handler nicht behoben werden. Es liegt beim Anwender zu beurteilen, ob diese Verzögerung akzeptabel ist oder nicht. Im Zweifelsfall muss das Situationserkennungssystem auf einer leistungsfähigeren Maschine betrieben werden, die eine häufigere Auswertung der Sensordaten erlaubt.

Ein weiteres potentielles Problem kann ebenfalls durch das Situationserkennungssystem ausgelöst werden. Wenn dieses sehr schnell hintereinander Zustandsänderungen einer einzelnen Situation meldet, würde der Situation Handler für jede Änderung einen Rollback auslösen und (sofern der Rollback bereits abgeschlossen ist) einen neuen Endpunkt auswählen. Dies würde schnell zum Scheitern der Endpunktauswahl führen, da die Anzahl der maximalen Rollbacks schnell überschritten werden würde. Ein solches Szenario wäre möglich, wenn der Wert, der durch einen Sensor geliefert wird, um eine bestimmte Schwelle pendelt. Wenn beispielsweise im Situation Template eine Situation „zu heiß“ definiert wird, die bei einem Temperaturwert von größer gleich 90°C auftritt und bei einem Wert von kleiner 90°C wieder verschwindet, würde eine nur geringe Temperaturänderung zu ständigen Zustandsänderungen bei der Situation führen. Zur Lösung dieses Problems wäre ein Zugriff auf die genauen Sensordaten erforderlich, um abschätzen zu können, wie „sicher“ eine Situation vorliegt. Eine Lösung für das Problem auf Ebene des Situation Handlers ist daher nicht möglich, da dieser keinen Zugriff auf die genauen Sensordaten besitzt und generell auf einem höheren Abstraktionslevel arbeitet. Der Situation Handler geht davon aus, dass er nur korrekte Situationen erhält, die stabil vorliegen. Unsicherheit und Instabilität von Situationen muss über die Sensorwerte auf Ebene des Situationserkennungssystems aufgelöst werden. Das Problem kann außerdem gelöst werden, indem

[17] http://www.yawlfoundation.org/

bei der Modellierung des Situation Templates mehr Sensoren einbezogen werden. Dadurch ist die Situation weniger stark von dem Wert eines einzelnen Sensors abhängig und ändert sich nur, wenn mehrere Sensoren auf eine Änderung der Situation hindeuten.

6. Verwandte Arbeiten

Die möglichen Vorteile der Ausnutzung von Kontext zur Steuerung von Workflows wurden von Forschern erkannt und untersucht. In diesem Kapitel werden verschiedene Ansätze vorgestellt, bei denen ebenfalls Kontextinformationen zur situationsabhängigen Anpassung von Workflows verwendet werden. Zudem werden die Unterschiede zwischen Ansätzen der vorgestellten Arbeiten und dem Ansatz der vorliegenden Arbeit herausgearbeitet. Die vorgestellten Ansätze lassen sich dabei in verschiedene Kategorien einteilen. Hierbei wird zwischen Ansätzen unterschieden, die entweder

- völlig neue Modellierungssprachen entwickeln,
- eine vorhandene Sprache erweitern
- oder die Kontextbehandlung ausgelagern.

Der in der vorliegenden Arbeit verwendete Ansatz lässt sich in die letzte Kategorie einordnen.

6.1. Entwicklung und Erweiterung von Workflow-Sprachen

In diesem Abschnitt werden Ansätze vorgestellt, die kontextabhängige Workflows realisieren, indem entweder *a*) eine neue Modellierungssprache entwickelt oder *b*) eine existierende Sprache erweitert wird.

6.1.1. Entwicklung von neuen Sprachen und Metamodellen

Es gibt verschiedene Möglichkeiten, bei denen durch die Entwicklung von neuen Workflow-Sprachen oder Metamodellen kontextabhängige Workflows ermöglicht werden. Zu beachten ist, dass für diese oftmals Transformationen zu Standardsprachen zur Verfügung gestellt werden. Dadurch können die neu entwickelten Sprachen auf Standard-Workflow-Engines ausgeführt werden.

Breitenbücher et al. [BHK+15] stellen mit *SitME* eine Möglichkeit vor, um Situationen in Workflows zu modellieren. Das Konzept der Situation ist das gleiche, wie das in der vorliegenden Arbeit verwendete. Hierbei werden die Konstrukte *Situation Event* und *Situational Scope* eingeführt. Unter einem Situation Event wird eine Aktivität verstanden, die ausgelöst wird, sobald eine bestimmte Situation auftritt. Ein Situation Event kann beispielsweise den Start eines Workflows auslösen. Ein Situational Scope kann verwendet werden, um eine oder mehrere Aktivitäten einzuschließen und diese damit situationsabhängig zu machen. Die Aktivitäten im Situational Scope werden nur ausgeführt, wenn bestimmte Situationen vorliegen. Die Spezifikation der Situationen für Situational Scopes erfolgt durch die Verbindung der Situational Scopes mit einem oder mehreren Situation Events. Dabei kann

über verschiedene Attribute definiert werden, welche Aktionen vorgenommen werden sollen, wenn die erforderlichen Situationen nicht vorliegen oder sich während der Ausführung des Situational Scopes ändern. SitME stellt keine eigene Workflow-Sprache dar, sondern nur ein Konzept zur Modellierung von situationsabhängigen Workflows, das prinzipiell auf jede Sprache angewandt werden kann. Entsprechend muss eine Transformation des SitME-Modells in eine andere Sprache stattfinden. Breitenbücher et al. beschreiben diese Transformation für die Sprache BPEL. Zur Erkennung von Situationen verwenden Breitenbücher et al. das Situationserkennungssystem SitRS [HWS+15], auf das auch in der vorliegenden Arbeit zurückgegriffen wird. Auch das Konzept der Situation Templates zur Beschreibung von Situationen wird verwendet. Mit den Situation Events ist zudem die Arbeit mit Workflow-Fragmenten möglich. Die Situation Events lassen sich auch durch das in dieser Arbeit entwickelte Konzept umsetzen. Dies wird erreicht, indem eine Aktion erstellt wird, die das Workflow-Plugin verwendet und bei der Änderung der gewünschten Situation ausgelöst wird. Gleich wie in der vorliegenden Arbeit ist es dabei das Ziel, das originale Modell des Workflows möglichst wenig mit der Modellierung von Situationen zu "verschmutzen".

Han et al. [HCC05] entwickelten mit *uWDL* (*Ubiquitous Workflow Description Language*) eine eigene Sprache zur Beschreibung von kontextabhängigen Workflows. Bei uWDL werden Workflows durch Graphen repräsentiert. Aktionen werden durch Knoten dargestellt, die mit Kanten verbunden sind. Eine Aktion beschreibt meist die Verwendung eines Web Services. Die Kontextinformationen werden auf den Übergängen zwischen verschiedenen Knoten modelliert. Die Strukturierung der Kontextinformationen erfolgt durch Ontologien. Dabei verwendet uWDL Tripel aus Subjekt, Verb und Objekt, um Kontextinformation darzustellen. Zur Sammlung von Kontextinformationen wird ein sogenannter *Context Mapper* verwendet. Dieser sammelt Informationen aus einem Sensornetzwerk und ordnet sie den Tripeln aus Subjekt, Verb und Objekt zu. Das Problem bei einem solchen Ansatz besteht darin, dass die Einbringung des Kontextes auf den Kanten zu einem sehr großen Modell führt, das sehr viele verschiedene Pfade besitzt. In der vorliegenden Arbeit wird versucht dieses Problem zu vermeiden, indem die Pfade für verschiedene Situationen in Fragmente ausgelagert werden. Zudem bietet die Entwicklung einer eigenen Sprache wie uWDL den Nachteil, dass diese inkompatibel zu bestehenden Workflow-Sprachen und Engines ist. Die Verwendung von Ontologien beziehungsweise den Tripeln aus Subjekt, Verb und Objekt bietet jedoch den Vorteil, dass auf Basis dieser Regeln eine komplexere Ableitung von Schlussfolgerungen möglich ist.

Ein System zur dynamischen Anpassung von Workflows an Kontextdaten entwickelten Choi et al. [CCSC07]. Dabei wird uWDL [HCC05] als Sprache zur Beschreibung von Workflows und Kontext eingesetzt. Ziel hierbei war es insbesondere, Workflows zur Laufzeit an die geänderte Situation eines Nutzers anzupassen. Die Anpassung erfolgt durch das Einfügen von neuen Diensten in einen Workflow oder durch die Anpassung der verwendeten Dienstes. Hierzu werden sogenannte *DItrees* (*Document Instance Trees*) eingesetzt. Bei einem DItree handelt es sich um eine Baumstruktur, die aus mehreren Unterbäumen besteht. Die Unterbäume sind kontextabhängig und können angepasst werden, wenn dem System eine Änderung des Kontextes mitgeteilt wird. Dabei muss nur der betroffene Unterbaum angepasst werden, während die Struktur des gesamten Baumes unberührt bleibt. Die Änderung des Baumes wird dynamisch von der Situationsänderung abgeleitet, wobei neu zu verwendende Dienste vom Nutzer definiert werden müssen. Zur Ausführung des Workflows werden die Unterbäume ausgeführt. Die Auswahl der Unterbäume erfolgt, indem der im Baum definierte Kontext mit dem real vorliegenden Kontext verglichen wird. Der initiale Workflow wird durch uWDL beschrieben und kann durch das von Choi et al. entwickelte System in einen DItree übersetzt werden. Choi

et al. legen bei ihrer Arbeit einen Fokus auf die dynamische Anpassbarkeit von Workflows. Es ist zu beachten, dass eine dynamische Anpassung auch in der vorliegenden Arbeit möglich ist, da die Workflow-Fragmente jederzeit ausgetauscht oder verändert werden können. Zudem können neue Situationen jederzeit berücksichtigt werden. Jedoch ist es nicht möglich, eine Aktivität zur Laufzeit vollständig aus dem Modell zu entfernen - dies würde eine BPEL Engine erfordern, die diesen Vorgang unterstützt. Die Anpassung erfolgt auf verschiedene Weisen: Das Modell und System von Choi et al. erlauben eine direkte Anpassung des Modells, während in der vorliegenden Arbeit eine externe Komponente die Auswahl von Fragmenten übernimmt, die dort ausgetauscht werden. Da Choi et al. auf die Verwendung von uWDL setzen, gelten die Vor- und Nachteile von uWDL im Vergleich zu der vorliegenden Arbeit auch für den Ansatz von Choi et al.

Ein weiteres Modell zur Behandlung von situationsabhängigen Workflows entwickelten Wolf et al. [WHR09]. Dieses greift verschiedene Aspekte der Objektorientierung auf. Dabei wird die *Flow Modelling Language* [HRKD08] erweitert, die Workflows anpassbarer und interaktiver machen soll. Wesentliche Elemente des Modells von Wolf et al. stellen *Entities*, *Flows*, *Context Frames* und *Context Events* dar. Eine Entity entspricht einem beliebigen Objekt aus der realen Welt, das Sensoren zur Überwachung von Kontext besitzt und einem bestimmten Typen entspricht. Bei Flows handelt es sich um Workflows, die mit Entities kombiniert und so kontextabhängig werden. Mit einem Context Frame können mehrere Entities zusammengefasst werden. Context Events erlauben die Behandlung von neu auftretenden Situationen. Diese Konstrukte erlauben eine kontextabhängige Modellierung und Ausführung von Workflows. Auch hier wird im Gegensatz zum vorliegenden Ansatz eine eigene Modellierungssprache verwendet, in der Kontextinformationen direkt in das Modell übernommen werden.

Ein anderer Ansatz zur Anpassung von Workflows wurde von Modaferri et al. [MBCP05] entwickelt und basiert, ähnlich wie SitME von Breitenbücher et al., auf kontextsensitiven Regionen. In diesen wird, abhängig vom Kontext, eine bestimmte Konfiguration geladen, beziehungsweise ein spezielles Verhalten gewählt. Dabei kann das Verhalten auch während der Ausführung eines Workflows angepasst werden. Ändert sich eine Situation, können Aktionen kompensiert werden, sofern sie für die neue Situationen nicht mehr passend sind. Für die Verarbeitung der Kontextinformation wird ein ähnliches Konzept gewählt, wie in dieser Arbeit. Auch hier wird aus der Low-Level-Information, High-Level-Information abgeleitet, die von den darüber liegenden Komponenten verwendet wird. Dies entspricht in etwa der Verwendung von Situation Templates im Situationserkennungssystem. Die genaue Ableitung der High-Level-Information beziehungsweise von Situationen, unterscheidet sich jedoch. Der Ansatz von Modaferri et al. sieht ein eigenes Modell zur Darstellung des Workflows und den Situationsinformationen vor. Dieses kann in ein BPEL-Modell transformiert werden. Hier liegt der wesentliche Unterschied zu dem Ansatz der vorliegenden Arbeit darin, dass der Ansatz von Modaferri et al. ein neues Modell für den Workflow erfordert, das erst transformiert werden muss, um mit Standard-Workflow-Engines kompatibel zu sein. Der hier präsentierte Ansatz lässt das ursprüngliche Workflow-Modell unberührt und verwendet eine externe Komponente zur Behandlung des Kontexts.

6.1.2. Verwendung vorhandener Sprachen

Eine weitere Möglichkeit zur Realisierung von kontextabhängigen Workflows stellt die Verwendung von vorhanden Sprachen dar, die offen und erweiterbar gestaltet sind. Diese lassen sich durch ihre bereits vorhandenen Fähigkeiten auch für die Darstellung von Kontextabhängigkeiten verwenden.

Mitsch et al. [MGP+11] präsentieren mit *ProFlow* einen Ansatz, der auf Ontologien zur Beschreibung von Situationen und Workflows setzt. Als Ontologiesprache wird OWL[1] verwendet. In ProFlow werden Ontologien für Workflows und Situationen definiert. Die Ontologien dienen als Grundlage zur Bewertung von Situationen und zur Auswahl eines entsprechenden Workflows. Um die Ontologie verwenden zu können, werden in der realen Welt erkannte Objekte auf Objekte mit Attributen in der Ontologie abgebildet beziehungsweise es wird ein Objekt der Ontologie instantiiert. Dadurch kann auf Basis der vorhandenen Objekte und deren Beziehungen das Auftreten einer bestimmten Situation erkannt werden. Mitsch et al. unterteilen bei ProFlow in ein sogenanntes *perception subsystem*, welches zur generischen Darstellung von Objekten und deren Attributen dient und ein *comprehension subsystem*, das die Beziehungen zwischen den Objekten repräsentiert. Insgesamt wird bei ProFlow ein anderer Fokus bei der Bewertung von Situationen gelegt als in dieser Arbeit. Zudem wird bei ProFlow mehr darauf abgezielt, einen Workflow auszuwählen und weniger darauf, die Workflows zu fragmentieren. Ein weiterer wesentlicher Unterschied besteht in der Verwendung von Ontologien zur Beschreibung von Situationen und Workflows.

Wang et al. [WLZ14] benutzen ebenfalls OWL um kontextabhängige Workflows umzusetzen. Hierbei definieren die Autoren verschiedene OWL-Klassen und Eigenschaften, um Kontext und Workflows zu modellieren. Für Workflows definieren Wang et al. verschiedene Arten von Knoten. Dabei gibt es Knoten zur Modellierung von Aufgaben (zum Beispiel Aufgaben für Menschen oder Anwendungen) und Routen. Mit den Routen kann ein Kontrollfluss dargestellt werden. Die OWL-Klassen zur Darstellung von Kontext beinhalten vor allem den Ort, die Zeit, die Akteure und Sonstiges. Diese Elemente wurden von Wang et al. als Hauptkategorien für Kontext identifiziert. Zur Verbindung der Konstrukte aus Workflows und Kontext werden Eigenschaften (*OWL properties*) verwendet. Hierbei definieren Wang et al. typische Kontextarten, erlauben dem Nutzer jedoch auch die Definition von eigenen Properties. Zudem legen Wang et al. Regeln fest, um Workflows mithilfe des von ihnen beschriebenen Modells zu erstellen. Wang et al. konzentrieren sich in ihrer Arbeit auf die Modellierung von kontextbewussten Workflows und nicht auf deren Ausführung. Aufgrund der Ähnlichkeit des Konzepts von Wang et al. zu dem von Mitsch et al., sind auch die Gemeinsamkeiten und Unterschiede zur vorliegenden Arbeit ähnlich. Es geht bei Wang et al. um die Darstellung von Kontext direkt im Workflow-Modell und nicht um die Auslagerung der Behandlung.

Mit *Context4BPEL* stellen Wieland et al. [WKNL07] ein Konzept für kontextabhängige Workflows und eine Implementierung des Konzeptes für BPEL vor. Die Kernelemente bei Context4BPEL stellen *Context Events*, *Context Queries* und *Context Decisions* dar. Ein Context Event beschreibt asynchrones Warten auf Kontextinformation, eine Context Query eine Anfrage um aktuellen Kontext abzurufen und eine Context Decision eine kontextabhängige Entscheidung im Kontrollfluss des Workflows. Context4BPEL stellt eine BPEL-Erweiterung dar, mit der die genannten Konstrukte für BPEL implementiert wurden.

[1] `http://www.w3.org/TR/owl-ref/`

Dafür wird der Workflow Engine eine Kontext-Management-Plattform zur Verfügung gestellt, die als Ziel für Context Queries dient und Context Events auslöst. Das Konzept von Context4BPEL unterschiedet sich vom hier vorliegenden Ansatz im Wesentlichen darin, dass Context4BPEL die Workflowsprache BPEL erweitert und die Kontextentscheidungen somit in das Workflow-Modell mit aufgenommen werden. Im vorliegenden Ansatz werden die Kontextentscheidungen jedoch in den Situation Handler ausgelagert.

Hsieh und Lin [HL14] verwenden die *Petri Net Markup Language*[2] (**PNML**), um kontextabhängige Workflows zu realisieren. Dabei werden Petri-Netze zur Modellierung von Workflows eingesetzt. Ein wichtiges Ziel ist hierbei die Allokation von Ressourcen. Hierzu werden *resource activity* Modelle erstellt, um die Verbindung zwischen Arbeitern und Ressourcen zu modellieren. Durch die Verbindung mit den Petri-Netzen, die Workflows darstellen, entsteht hierbei ein vollständiges Modell, dass die Überwachung und Zuweisung von Ressourcen erlaubt. Damit sollen insbesondere kontextabhängige Benutzeroberflächen erstellt werden. Die Benutzeroberflächen werden abhängig von dem aktuellen Kontext und der Aktivität des Workflows angepasst. So können zum Beispiel individuelle Arbeitslisten für die Nutzer generiert werden, abhängig zum Beispiel von deren Standort. Die Benutzeroberfläche wird dabei mittels XML beschrieben und kann auf einem mobilen Gerät dargestellt werden. Dieser Ansatz unterscheidet sich wesentlich von dem hier vorliegenden, da mit PNML eine andere Sprache verwendet wird und das Ziel (kontextabhängige GUI) ein anderes ist. Grundsätzlich wird aber auch in der Arbeit von Hsieh und Lin die Anpassung von Workflows an aktuellen Kontext thematisiert.

Ein weiteres Framework zur dynamischen Anpassung von Prozessen wurde von Bucchiarone [BMPR12] et al. entwickelt. Dabei setzen Bucchiarone et al. insbesondere auf Selbstanpassung durch die Anwendungen, die wiederum aus Prozess-Fragmenten bestehen. Das Modell des Frameworks sieht Entitäten vor, deren Verhalten durch Geschäftsprozesse beschrieben wird. Entitäten können ein System dynamisch betreten oder verlassen und bieten ihre Funktionalität in Form von Prozess-Fragmenten an. Die Geschäftsprozesse werden durch *Adaptable Pervasive Flows* (**APF**s) [HRKD08] beschrieben, die es erlauben, Geschäftsprozesse mit Kontext aus der realen Welt zu verbinden. Die APFs wurden durch Bucchiarone et al. insbesondere um abstrakte Aktivitäten erweitert. Eine abstrakte Aktivität besitzt in der Designphase nur ein bestimmtes Ziel, für welches zur Laufzeit ein existierendes Prozessfragment ausgewählt wird, das unter Berücksichtigung des aktuellen Kontextes das vorgegebene Ziel erfüllt. Das Ziel kann auch durch eine Komposition von mehreren Prozessfragmenten erreicht werden. Weitere Möglichkeiten der dynamischen Adaption sind *Local Adaption* und *Compensation*. Local Adaption setzt den Prozess zurück und beginnt bei einer früheren Aktivität von vorne. Compensation berechnet dynamisch einen Kompensationsprozess aus existierenden Prozessfragmenten. Die Kombination der beschriebenen Adaptionsmechanismen führt zu sogenannten Adaptionsstrategien, die komplexere Anpassungsprobleme lösen können. Der Ansatz von Bucchiarone et al. zeigt durch die Verwendung von Prozess/Workflow-Fragmenten Gemeinsamkeiten mit der vorliegenden Arbeit. Jedoch liegt ein größerer Fokus auf der selbstständigen Anpassung der Anwendung an den aktuellen Kontext. Das Konzept der abstrakten Aktivitäten ähnelt der Auswahl von Workflow-Fragmenten durch den Situation Handler. Es gibt jedoch Unterschiede bei der Art der Auswahl. Die Auswahl erfolgt bei der vorliegenden Arbeit durch eine externe Komponente, während die Auswahl bei Bucchiarone et al. durch das System selbst vorgenommen wird.

[2] XML-basierte Sprache zum Austausch von Petri-Netzen [WK03]

Tang et al. [TGD+08] verwenden kontextbewusste Workflows im Bereich *Ubiquitous Computing*. Dabei setzen sie ähnlich wie Hsieh und Lin [HL14] auf Petri-Netze zur Darstellung von Workflows. Ziel hierbei ist eine kontextbewusste Navigation. Ein Unterschied zur Arbeit von Hsieh und Lin besteht in der Modellierung von Kontext. Tang et al. definieren Kontext in mehreren Hauptdimensionen (zum Beispiel die Dimension der beteiligten Personen) und verwenden *Entitäten* mit Attributen und Beziehungen zu anderen Entitäten. Zusätzlich wurde ein Algorithmus entwickelt, der, unter Berücksichtigung des aktuellen Kontexts, die nächste Aktivität in einem Workflow auswählt. In den Petri-Netzen werden zu diesem Zweck selektive Übergänge eingeführt, die nur ausgeführt werden, wenn die entsprechende Bedingung erfüllt wird. Die Version der kontextbewussten Workflows, die in [TGD+08] vorgestellt wird, ist stark auf den Bereich Navigation eingeschränkt, da der entwickelte Auswahlalgorithmus nur die vorgegebenen Hauptdimensionen berücksichtigt. Diese sind wiederum auf das Beispiel der Navigation zugeschnitten. Jedoch ist das Modell der Dimensionen und der Auswahlalgorithmus erweiterbar. Das Konzept von Tang et al. unterscheidet sich wesentlich von dem der vorliegenden Arbeit. Die Modellierung von Kontext und auch die Behandlung von Kontext haben wenige Gemeinsamkeiten. Lediglich Workflows zur Ausführung von Prozessen unter der Berücksichtigung von Kontext können als gemeinsames Mittel identifiziert werden.

6.2. Auslagerung der Kontextbehandlung

Die bislang vorgestellten Arbeiten sehen eine Modifikation des Workflow-Modells zur Behandlung von Kontextinformationen und deren Abhängigkeiten vor. Ein anderer Ansatz besteht in der Behandlung von Kontextinformationen durch eine externe Komponente. Das ursprüngliche Modell des Workflows wird also wenig mit situationsabhängigen Konstrukten belastet. Dennoch gibt es meist minimale Modifikationen des Modells, die dazu dienen, die Situationsbehandlung einzuleiten. Der wesentliche Unterschied zu den zuvor vorgestellten Ansätzen besteht folglich darin, dass Modelle ohne Transformationen von Standard-Workflow-Engines ausgeführt werden können und im Workflow-Modell keine ausführliche Behandlung von Situationen beziehungsweise Kontext stattfinden muss. Der in der vorliegenden Arbeit vorgestellte Ansatz lässt sich ebenfalls in diese Kategorie einordnen.

Eine Mischung aus der Verwendung und Erweiterung einer vorhandenen Prozessmodellierungsart und der Auslagerung der Kontextbehandlung in eine externe Komponente stellen Mundbrod et al. [MGKR15] mit CaPI (*context-aware process injection*) vor. Dabei verwenden Mundbrod et al., gleich wie in dieser Arbeit, Workflow-Fragmente zur Dynamisierung von Workflows. Jedoch setzen Mundbrod et al. auf die Injektion der Fragmente zur Laufzeit, während in der vorliegenden Arbeit der Situation Handler für die Ausführung der Workflow-Fragmente zuständig ist. Zur Darstellung von kontextbewussten Workflows erweitern Mundbrod et al. ein Prozessmodell um sogenannte CPFs (*context-aware process family*). Eine CPF besteht aus den Elementen *Extension Area*, *Contextual Situation*, *Process Parameter*, *Process Fragment* und *Injection Specification*. Eine Extension Area stellt den Bereich in einem Workflow dar, der durch die Injektion von einem Workflow-Fragment zur Laufzeit an Kontextdaten angepasst werden kann. Die Kontextdaten werden durch Contextual Situations dargestellt. Eine Contextual Situation besteht aus mehreren Process Parameters, die in einem logischen Ausdruck (Prädikatenlogik erster Stufe) zusammengefasst werden. Die Process Parameters enthalten aktuelle Kontextdaten. Die Process Fragments stellen die Workflow-Fragmente dar, die injiziert werden. Mit einer Injection Specification kann der Vorgang der Injektion näher spezifiziert und der Datenfluss

festgelegt werden. Die Auswertung der Kontextdaten und der Vorgang der Injektion finden zur Laufzeit statt. Jedoch muss zur Designphase festgelegt werden, welche Process Parameters in den Contextual Situations berücksichtigt werden. Hier bietet die vorliegende Arbeit den Vorteil, dass zur Designphase nur eine Operation festgelegt werden muss, zu einem situationsbewussten Endpunkt jedoch jederzeit neue Situationen hinzugefügt werden können. Zur Ausführung des Prozesses verwenden Mundbrod et al. das System AristaFlow [DR09], das die Modifikation von Prozessen zur Laufzeit ermöglicht. Zur Erweiterung von *AristaFlow* entwickelten Mundbrod et al. eine Komponente zur Auswertung der Kontextdaten. Die vorliegende Arbeit bietet hier den Vorteil, dass sie zu dem weiter verbreiteten Standard BPEL kompatibel ist.

Adams et al. [AHEA06] verwenden zur Flexibilisierung von Workflows sogenannte *Worklets*. Hierbei handelt es sich um eigenständige und vollständige Workflows, die eine Teilaufgabe eines größeren Workflows implementieren. Ein Workflow wird in diesem Modell als Komposition von Worklets dargestellt. Die Worklets von Adams et al. ähneln dem Konzept der Workflow-Fragemente der hier vorliegenden Arbeit. Die Auswahl eines passenden Worklets erfolgt zur Laufzeit, durch eine Auswertung der Kontextinformationen. Zur Auswahl des Worklets werden sogenannte *Ripple Down Rules* verwendet. Eine Ripple Down Rule stellt strukturell einen Binärbaum dar. Auf den Knoten sind hierbei Bedingungen und Schlussfolgerungen annotiert, deren Erfüllung oder Nichterfüllung zu einem anderen Pfad führt, was wiederum die Auswertung einer anderen Bedingung und eine andere Schlussfolgerung zur Folge hat. Hierbei wird für jede Aufgabe in einem Workflow, die durch Worklets umgesetzt werden soll, eine Menge an Regeln verwaltet, die ausgewertet wird [ATHEA06]. Die Auswahl wird anhand aktueller Kontextdaten über die Regeln getroffen. Hält ein Nutzer die Wahl für unangemessen, kann er die Wahl ablehnen und eine neue Regel hinzufügen, die zu einer besseren Auswahl führt. Das ausgewählte Worklet wird nach der Auswahl dynamisch in den Workflow eingebunden. Das System von Adams et al. wurde mithilfe von *YAWL* [AH05] realisiert. Hierbei handelt es sich um eine Open-Source-Workflow-Sprache, einschließlich einer Engine zur Ausführung. Obwohl das Konzept der Worklets und das der Workflow-Fragmente in der vorliegenden Arbeit sehr ähnlich sind, gibt es dennoch einige deutliche Unterschiede. Zur Ausführung der Worklets werden diese bei Adams et al. in dieselbe Engine geladen, während in der vorliegenden Arbeit ein anderer Workflow gestartet wird. Dieser kann an einer beliebigen Netzwerkadresse zur Verfügung gestellt werden und muss nur über eine entsprechende Schnittstelle (WSDL mit SOAP-Binding) verfügen. Letztlich unterscheiden sich auch die verwendeten Workflow-Technologien, da in der vorliegenden Arbeit der Fokus auf BPEL gelegt wurde.

Eine weitere Umsetzung von kontextabhängigen Workflows stellt das *CAWE-Framework* von Ardissono et al. [AFG+07a] dar. Hierbei werden für Workflows abstrakte Aktivitäten eingeführt, für die es verschiedene kontextabhängige Implementierungen geben kann. Um einen Workflow, der solche abstrakten Aktivitäten enthält, auszuführen, wird bei der Ausführung der abstrakten Aktivität ein sogenanntes *Personalisierungsmodul* aufgerufen. Das Personalisierungsmodul liefert abhängig vom Kontext eine geeignete Implementierung der Aktivität zurück. Die Auswahl der Implementierung der abstrakten Aktivität erfolgt über die Werte von Kontextvariablen [AFG+07b]. Die aktuellen Werte der Kontextvariablen werden vom Personalisierungsmodul über einen Web Service abgefragt. Die kontextabhängigen Implementierungen definieren Bedingungen, die für eine Verwendung der jeweiligen Implementierung vorliegen müssen. Die Bedingungen werden über die Werte der Kontextvariablen evaluiert. Dabei ist zu beachten, dass die Kontextvariablen, die dem Personalisierungsmodul zur Eingabe dienen, bei der Modellierung des Workflows festgelegt werden müssen. Lediglich der Wert

der Kontextvariablen wird zur Laufzeit evaluiert. Die Implementierung der abstrakten Aktivität wird als Subprozess des Workflows ausgeführt. Zur Ausführung eines Workflows mit abstrakten Aktivitäten kann laut Ardissono et al. eine Standard-Workflow-Engine verwendet werden. Das Konzept des CAWE-Frameworks ähnelt in einigen Punkten dem in dieser Arbeit vorgestellten Konzept. Bei Ardissono et al. wird ebenfalls eine externe Komponente - in diesem Fall das Personalisierungsmodul - verwendet, um eine geeignete Implementierung einer Aktivität auszuwählen. Der Situation Handler kümmert sich jedoch selbst darum, das ausgewählte Workflow-Fragment zu starten. Zudem wird hier auf das Konzept der abstrakten Aktivitäten verzichtet. Stattdessen werden Standard BPEL-Workflows verwendet, die den Situation Handler adressieren, anstatt eines Web Services, der die Aktivität selbst implementiert. Dadurch kann der Workflow beinahe unverändert bleiben. Auch die Art der Darstellung von Kontext unterscheidet sich. Der Kontext wird im Situation Handler durch Situationen dargestellt, die vom Situationserkennungssystem erkannt werden. Beim CAWE-Framework wird der Kontext in eigenen Variablen, die im Workflow festgelegt werden, gespeichert. Die Behandlung von Kontextdaten beim Situation Handler bietet den Vorteil, dass sie unabhängig vom Workflow stattfindet und jederzeit neue Situationen berücksichtigt werden können. Bei CAWE muss sämtliche Kontextinformation durch die im Workflow definierten Variablen abgedeckt werden. Um eine neue Variable beziehungsweise andere Kontextinformation zu behandeln, muss der Workflow modifiziert werden.

Andrikopoulos et al. [ABS+13] setzen adaptive Workflow-Choreographien durch sogenannte *Collaborative Adaptive Systems* (**CAS**) um. Hierbei wird das CAS verwendet, um Workflows zur Laufzeit anzupassen. Als CAS wird ein System definiert, in welchem Entitäten kommunizieren, um eigene und gemeinsame Ziele zu erreichen. Zur Erreichung der gemeinsamen Ziele formen die Entitäten sogenannte *Ensembles*. Zur Interaktion zwischen den Entitäten stellt jede Entität eine *Zelle* bereit. Zellen stellen Einheiten dar, die in einem größeren (zellulären) System eine konkrete Funktionalität bereitstellen. Zur Umsetzung der Funktionalität kann mit anderen Zellen interagiert werden. Für die Interaktion zwischen den Zellen werden vorgegebene Protokolle eingesetzt. Um das Verhalten einer Zelle beschreiben zu können, werden abstrakte Aktivitäten eingesetzt, die andere Zellen zur Umsetzung ihrer Funktionalität benötigen. Die Auswahl dieser Zellen kann zur Laufzeit geschehen. Durch das Konzept der abstrakten Aktivitäten können Kontextinformationen berücksichtigt werden, die zur Laufzeit durch die Entitäten bereitgestellt werden. Zur Anpassung von Zellen definieren Andrikopoulos et al. das Konzept der *Differenzierung*. Hierunter wird die Anpassung des Verhaltens einer existierenden Zelle verstanden. Beispiele sind das Einfügen und Löschen von Aktivitäten oder von Konnektoren in den Kontrollfluss. Zur Realisierung dieses Konzeptes wird eine Workflow Engine benötigt, die die Ad-hoc-Anpassung von Aktivitäten beziehungsweise des Kontrollflusses erlaubt. Hierfür haben Andrikopoulos et al. die BPEL Engine Apache ODE[3] erweitert. Das Konzept der abstrakten Aktivitäten von Andrikopoulos et al. ähnelt dem von Ardissono et al. und dem Konzept der Workflow-Fragmente dieser Arbeit. Jedoch wird hier ein sehr viel größerer Fokus auf die Kollaboration von verschiedenen Workflows gelegt und der Kontext wird im Modell berücksichtigt, beziehungsweise das Modell wird an den Kontext angepasst.

Ein Ansatz zur Umsetzung von kontextbewussten Workflows, bei dem die Kontextbehandlung ebenfalls in eine externe Komponente ausgelagert wird, wurde von Abbasi et al. [AS09] entwickelt. Abbasi

[3] `http://ode.apache.org/`

et al. schlagen ein Framework vor, welches eine geringe Modifikation der Aktivitäten im Workflow sowie eine externe Komponente zur Auswertung von Kontext (*Context Manager*) benötigt. Im Workflow werden kontextbewusste Aktivitäten (*Smart Activities*) eingeführt. Den Smart Activities werden *Context Queries* zugeordnet, die vom Context Manager ausgewertet werden. Zur Auswertung einer Context Query greift der Context Manager auf ein Sensornetzwerk zu, das Daten aus der realen Welt liefert. Für jede Smart Activity existiert eine Menge von Aktivitäten, die die Smart Activity umsetzen können. Es können auch andere Workflows zur Umsetzung einer Smart Activity verwendet werden. Der Context Manager ist für die Auswahl einer konkreten Aktivität zur Umsetzung einer Smart Activity zuständig. Hierzu werden die Context Query, die der Smart Activity zugeordnet ist und die Sensordaten aus der realen Welt verwendet. Eines der Ziele von Abbasi et al. ist die Kontextbehandlung in Workflows mit einem möglichst einfachen Modell. Dabei soll es, ähnlich wie in der vorliegenden Arbeit, vermieden werden, die Kontextbehandlung über verschiedene Pfade im Workflow umzusetzen. Dazu wird auf Smart Activities gesetzt, die konzeptionell den abstrakten Aktivitäten ähneln, wie sie beispielsweise von Ardissono et al. [AFG+07a] eingesetzt werden. Die Auswahl der Implementierung einer Smart Activity erfolgt durch eine externe Komponente. Dabei wird durch die Context Query jedoch bereits zur Designphase festgelegt, welche Kontextinformationen berücksichtigt werden sollen. In der vorliegenden Arbeit wird zur Design Phase nur die Operation festgelegt. Die Situationen, die berücksichtigt werden sollen, können jedoch jederzeit geändert werden. Auch die Ausführung der Smart Activities unterscheidet sich von der Ausführung der Workflow-Fragmente in dieser Arbeit. Bei Abbasi et al. wird die Implementierung der Smart Activity durch den Context Manager ausgewählt und zur Laufzeit durch die Workflow Engine gebunden. In der vorliegenden Arbeit übernimmt der Situation Handler die Ausführung beziehungsweise den Start des gewählten Workflow-Fragments und liefert die Antwort an den Workflow zurück.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Komponente zur situationsabhängigen Auswahl von Workflow-Fragmenten konzipiert und implementiert, die als Situation Handler bezeichnet wird. Der Situation Handler ist dazu in der Lage, Workflow-Fragmente zur Ausführung einer bestimmten Operation, abhängig von Situationen, aus einer Liste auszuwählen und zu kontaktieren. Zur Auswahl wurde ein Algorithmus entwickelt, der Bewertungen für die Workflow-Fragmente erstellt und das Workflow-Fragment mit der besten Bewertung auswählt. Ändert eine Situation ihren Zustand, kann ein Rollback eingeleitet und ein anderes Workflow-Fragment ausgewählt werden. Bei Situationsänderungen können zudem Benachrichtigungen versandt oder Aktionen angestoßen werden. Auf dieses Weise können beim Auftreten oder Verschwinden von Situationen auch Workflow-Fragmente ausgeführt werden. Damit kann das Konzept der Situation Events von Breitenbücher et al. [BHK+15] umgesetzt werden. Der Versand der Nachrichten beziehungsweise die Ausführung von Aktionen erfolgt über ein Pluginsystem, das die Verwendung verschiedener Technologien für den Versand auf eine dynamische Art ermöglicht. Mit dem Situation Handler wurden Plugins für den Versand von E-Mails, HTTP-Anfragen und Android-Push-Nachrichten entwickelt. Zudem wurde eine Android App umgesetzt, die zum Empfang der Push-Nachrichten dient. Der Situation Handler ist über eine Web-Oberfläche und eine REST API konfigurierbar. Die Abfrage von aktuellen Situationen erfolgt über das Situationserkennungssystem SitRS [HWS+15], das im Rahmen eines Forschungsprojektes der Universität Stuttgart entwickelt wurde. Für den Situation Handler wurde mit der Programmiersprache Java ein Prototyp implementiert. Ein wichtige Bedeutung kam hierbei dem Framework Apache Camel zu, das für das Routing und die Verarbeitung von Nachrichten eingesetzt wurde.

Für die Interaktion mit Workflows und die situationsabhängige Auswahl von Workflow-Fragmenten wurde zuerst ein einfaches Modell entworfen, das die möglichen Anwendungsszenarien des Situation Handlers allerdings unzureichend abdeckt. Daraufhin wurde dieses Modell erweitert, sodass insbesondere mehr Situationen auf verschiedene Weisen berücksichtigt werden. Das resultierende Metamodell eines Endpunktes (der ein Workflow-Fragment ausführt) sieht für einen Endpunkt mehrere Situationen, eine Operation sowie die Adresse des Endpunktes vor. Für eine Situation kann dabei definiert werden, welchen Zustand die Situation besitzen soll, ob die Einhaltung des Zustandes optional ist oder nicht und ob bei einer Zustandsänderung ein Rollback durchgeführt werden soll. Erhält der Situation Handler die Anfrage eines Workflows eine bestimmte Operation durchzuführen, erstellt der Situation Handler eine Bewertung für jedes Workflow-Fragment, das diese Operation implementiert. Es wird das Workflow-Fragment ausgewählt, für das alle Situationen den geforderten Zustand besitzen. Definiert ein Endpunkt (Workflow-Fragment) mehr Situationen als ein anderer Endpunkt, wird dieser bevorzugt behandelt. Ändert eine Situation ihren Zustand während ein Workflow-Fragment eine Anfrage verarbeitet, kann der Situation Handler einen Rollback initiieren und einen anderes Workflow-Fragment auswählen.

Für die Zukunft soll der Situation Handler um weitere Funktionalitäten erweitert werden. Dies betrifft insbesondere die Behandlung von Workflow Anfragen in einem Szenario, in dem kein passendes Workflow-Fragment zur Auswahl gefunden wird. Für diesen Fall soll es zukünftig möglich sein, auf ein Verzeichnis mit archivierten Workflow-Fragmenten zuzugreifen. Die Workflow-Fragmente sind hierbei so verpackt, dass sie automatisch in einer Cloud-Umgebung bereitgestellt werden können. Workflow-Fragmente können damit zukünftig durch den Situation Handler bereit gestellt werden, sofern momentan kein passendes Workflow-Fragment existiert. Diese Funktion bietet außerdem den Vorteil, dass weniger Workflow-Fragmente aktiv gehalten werden müssen und so Ressourcen gespart werden können. Die Workflow-Fragmente werden bei Bedarf in der Cloud bereitgestellt und können wieder entfernt werden, wenn sie nicht mehr benötigt werden.

A. Anhang

A.1. Situation Template

Nachfolgend ein Beispiel für ein Situation Template 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"/>
           </conditionNode>
           <contextNode id="A5" name="memorySensor">
                 <parent parentID="A8"/>
           </contextNode>
           <contextNode id="A6" name="cpuSensor">
                 <parent parentID="A4"/>
```

```
            </contextNode>
            <contextNode id="A7" name="watchdogSensor">
                  <parent parentID="A9"/>
            </contextNode>
            <situationNode name="machine_failed" id="A10"/>
      </Situation>
</SituationTemplate>
```

A.2. WebApp

Nachfolgend einige Screenshots der wichtigsten Funktionalitäten der Benutzeroberfläche des Situation Handlers.

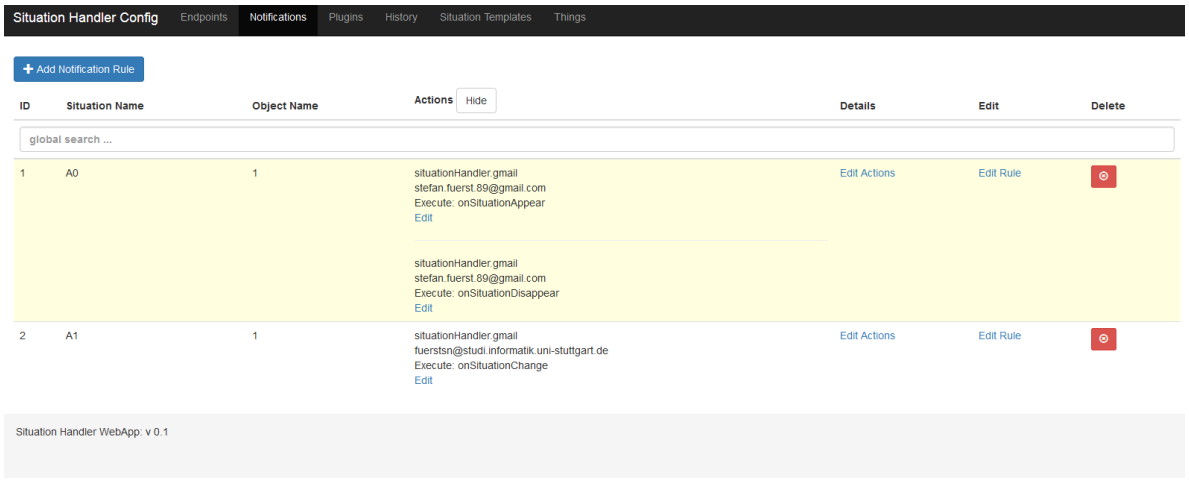


Abbildung A.1.: Die Anzeige der Notifikationsregeln in der Benutzeroberfläche. Zusätzlich wird eine Kurzfassung der Aktionen angezeigt (kann ausgeblendet werden). Es kann in die Bearbeitungsansicht der Regeln (Abb. A.2) sowie in die Ansicht der Aktionen (Abb. A.3) gewechselt werden.

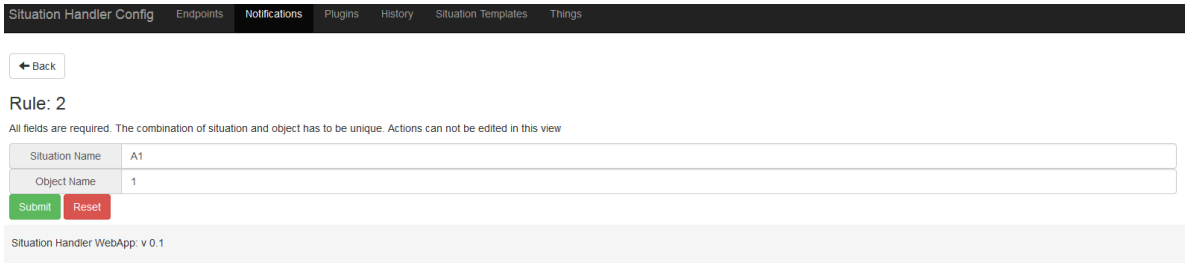


Abbildung A.2.: Die Bearbeitung/Erzeugung von Notifikationsregeln in der Benutzeroberfläche. Situationsname und Objekt ID müssen eingegeben werden.

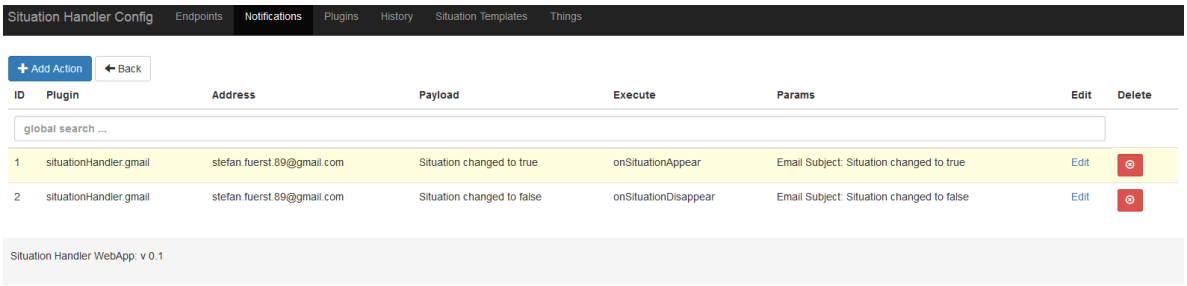


Abbildung A.3.: Die Anzeige von Aktionen in der Benutzeroberfläche. Alle wichtigen Eigenschaften einer Aktion, inklusive Parameter, werden angezeigt. Es kann zur Bearbeitung einer Aktion gewechselt werden (Abb. A.4).

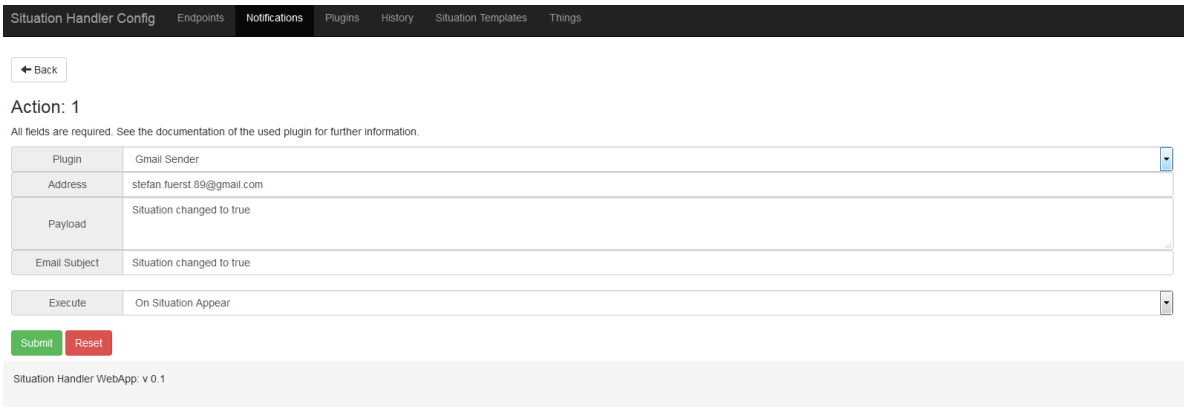


Abbildung A.4.: Die Bearbeitung von Aktionen in der Benutzeroberfläche. Es werden die existierenden Plugins zur Auswahl angeboten. Die Felder zur Eingabe der Parameter werden abhängig von den Vorgaben des Plugins erzeugt.

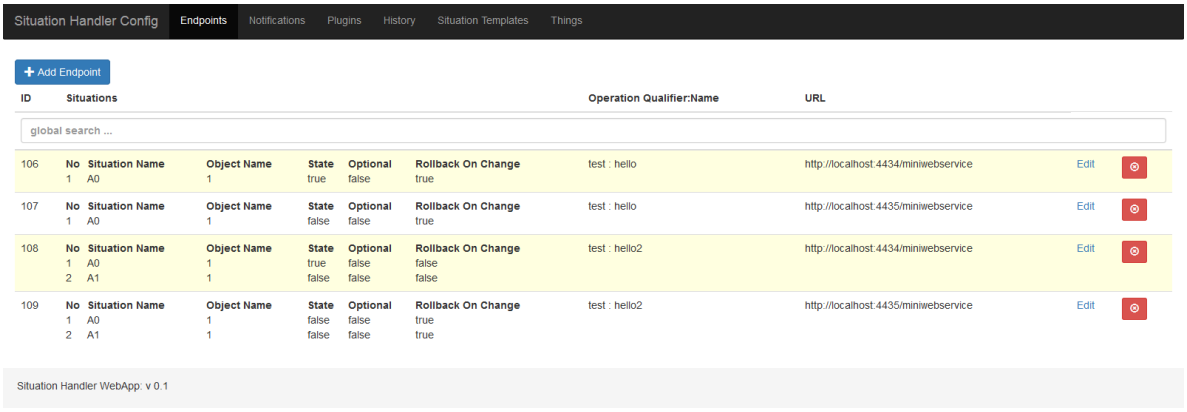


Abbildung A.5.: Die Anzeige von Endpunkten in der Benutzeroberfläche. Eine Übersicht aller wichtigen Eigenschaften eines Endpunktes wird angezeigt. Es kann zur Ansicht zur Bearbeitung eines Endpunktes gewechselt werden (Abb. A.6).

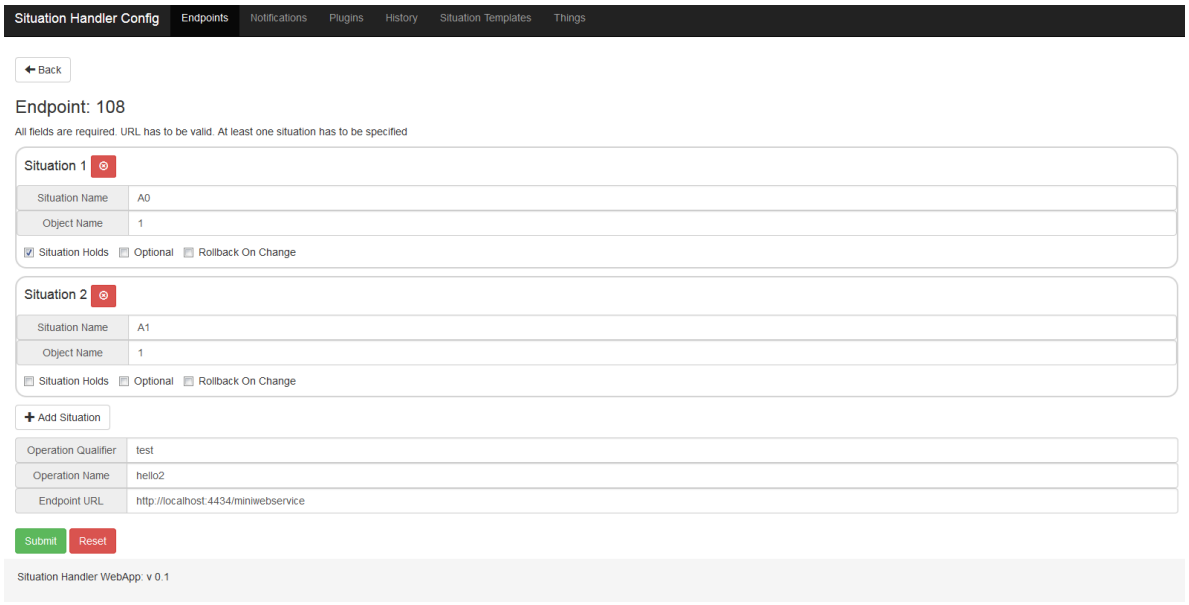


Abbildung A.6.: Die Bearbeitung von Endpunkten in der Benutzeroberfläche. Mit dem Button „Add Situation“ können einem Endpunkt beliebig viele Situationen zugeordnet werden.

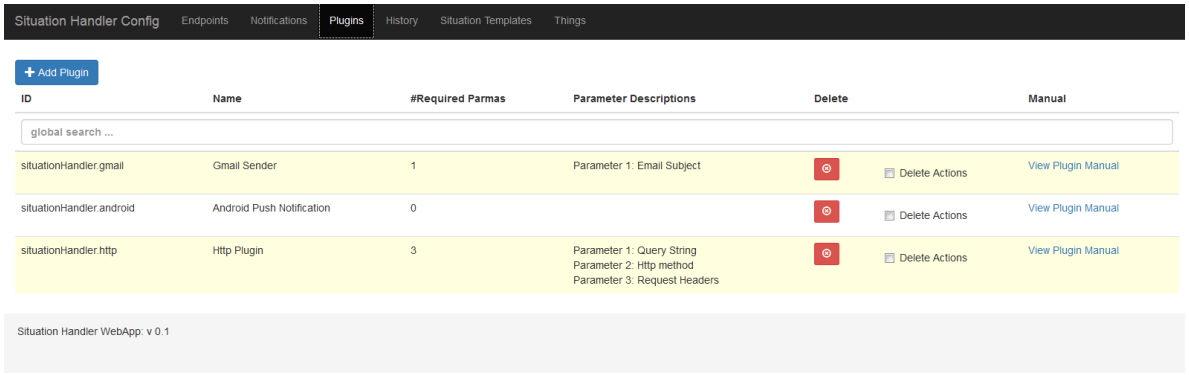


Abbildung A.7.: Die Anzeige der Plugins in der Benutzeroberfläche. Zeigt Informationen über alle aktuell geladenen Plugins. Ermöglicht das Entfernen eines Plugins zur Laufzeit (mit und ohne Entfernung der zugeordneten Aktionen), die Anzeige der Anleitung des Plugins und das Hinzufügen neuer Plugins (Abb. A.8).

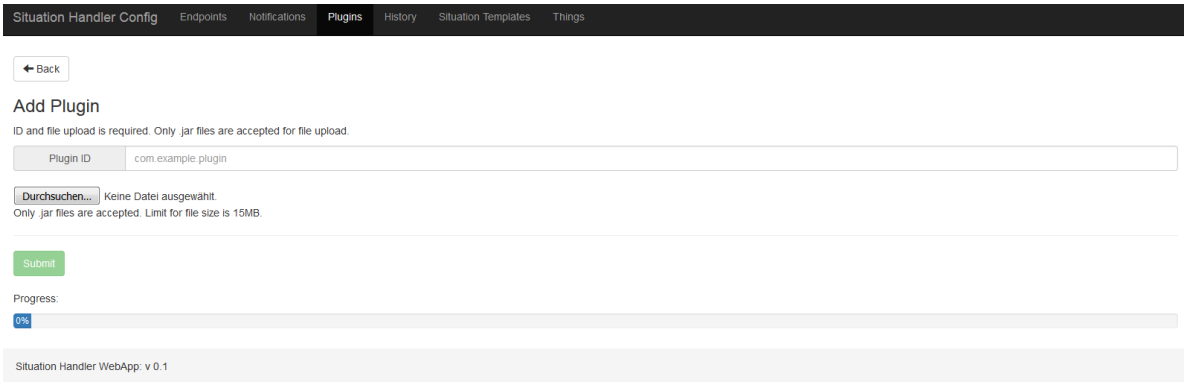
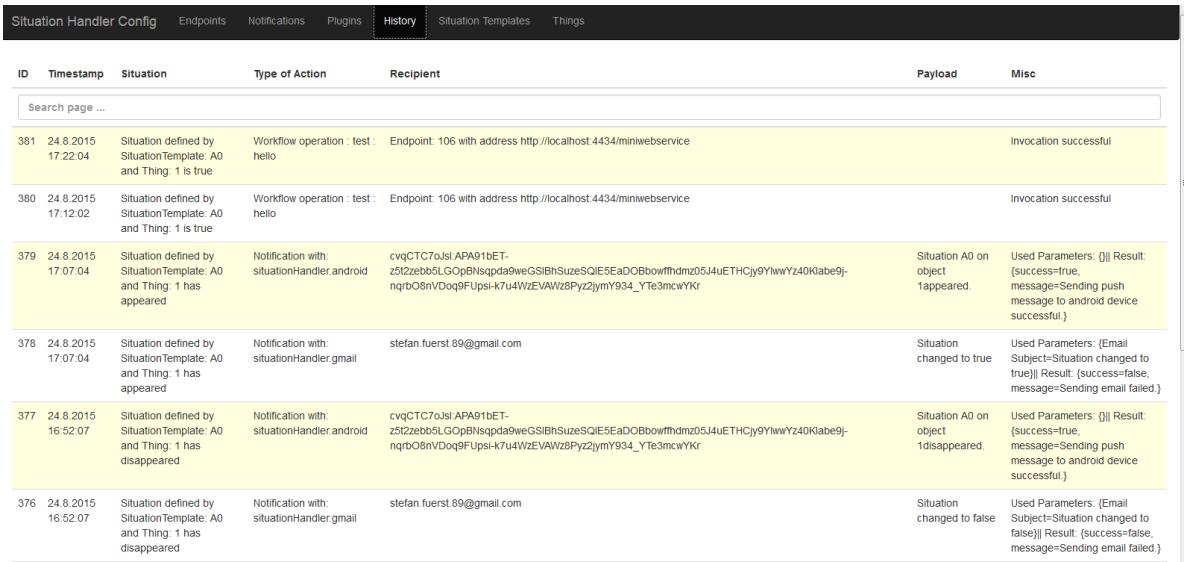


Abbildung A.8.: Das Hinzufügen von Plugins in der Benutzeroberfläche. Es muss die ID des Plugins angegeben werden. Außerdem muss eine Jar-Datei, die das Plugin enthält, hochgeladen werden.



ID	Timestamp	Situation	Type of Action	Recipient	Payload	Misc
381	24.8.2015 17:22:04	Situation defined by SituationTemplate: A0 and Thing: 1 is true	Workflow operation : test : hello	Endpoint: 106 with address http://localhost:4434/miniwebservice		Invocation successful
380	24.8.2015 17:12:02	Situation defined by SituationTemplate: A0 and Thing: 1 is true	Workflow operation : test : hello	Endpoint: 106 with address http://localhost:4434/miniwebservice		Invocation successful
379	24.8.2015 17:07:04	Situation defined by SituationTemplate: A0 and Thing: 1 has appeared	Notification with: situationHandler.android	cvqCTC7oJsl:APA91bET-z5t2zebb5LGOpBNsqpda9weGSlBhSuzeSQlE5EaDOBbowffhdmz05J4uETHCjy9YlwwYz40Klabe9j-nqrbO8nVDoq9FUpsi-k7u4WzEVAWz8Pyz2jymY934_YTe3mcwYKr	Situation A0 on object 1appeared.	Used Parameters: {}|| Result: {success=true, message=Sending push message to android device successful.}
378	24.8.2015 17:07:04	Situation defined by SituationTemplate: A0 and Thing: 1 has appeared	Notification with: situationHandler.gmail	stefan.fuerst.89@gmail.com	Situation changed to true	Used Parameters: {Email Subject=Situation changed to true}|| Result: {success=false, message=Sending email failed.}
377	24.8.2015 16:52:07	Situation defined by SituationTemplate: A0 and Thing: 1 has disappeared	Notification with: situationHandler.android	cvqCTC7oJsl:APA91bET-z5t2zebb5LGOpBNsqpda9weGSlBhSuzeSQlE5EaDOBbowffhdmz05J4uETHCjy9YlwwYz40Klabe9j-nqrbO8nVDoq9FUpsi-k7u4WzEVAWz8Pyz2jymY934_YTe3mcwYKr	Situation A0 on object 1disappeared.	Used Parameters: {}|| Result: {success=true, message=Sending push message to android device successful.}
376	24.8.2015 16:52:07	Situation defined by SituationTemplate: A0 and Thing: 1 has disappeared	Notification with: situationHandler.gmail	stefan.fuerst.89@gmail.com	Situation changed to false	Used Parameters: {Email Subject=Situation changed to false}|| Result: {success=false, message=Sending email failed.}

Abbildung A.9.: Die History kann hier komplett eingesehen werden. Diese wird stückweise vom Situation Handler geladen.

Literaturverzeichnis

[ABS+13] V. Andrikopoulos, A. Bucchiarone, S. G. Sáez, D. Karastoyanova, C. A. Mezzina. Towards Modeling and Execution of Collective Adaptive Systems. In *Proceedings of the 9th International Workshop on Engineering Service-Oriented Applications (WESOA'13)*, S. 1–12. Springer, 2013. (Zitiert auf Seite 82)

[AFG+07a] L. Ardissono, R. Furnari, A. Goy, G. Petrone, M. Segnan. Context-Aware Workflow Management. In *Web Engineering*, Band 4607 von *Lecture Notes in Computer Science*, S. 47–52. Springer Berlin Heidelberg, 2007. (Zitiert auf den Seiten 81 und 83)

[AFG+07b] L. Ardissono, R. Furnari, A. Goy, G. Petrone, M. Segnan. A Framework for the Management of Context-aware Workflow Systems. In *WEBIST (1)*, S. 80–87. 2007. (Zitiert auf Seite 81)

[AH05] W. van der Aalst, A. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005. (Zitiert auf Seite 81)

[AHEA06] M. Adams, A. ter Hofstede, D. Edmond, W. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, Band 4275 von *Lecture Notes in Computer Science*, S. 291–308. Springer Berlin Heidelberg, 2006. (Zitiert auf Seite 81)

[AIM10] L. Atzori, A. Iera, G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. (Zitiert auf Seite 17)

[Apa] Apache. Architecture - Diagram. `https://camel.apache.org/architecture.html`. (Zitiert auf Seite 56)

[AS09] A. Abbasi, Z. Shaikh. A Conceptual Framework for Smart Workflow Management. In *Information Management and Engineering, 2009. ICIME '09. International Conference on*, S. 574–578. 2009. (Zitiert auf Seite 82)

[ATHEA06] M. Adams, A. H. Ter Hofstede, D. Edmond, W. M. van der Aalst. Implementing dynamic flexibility in workflows using worklets. *BPMCenter Report*, 6(06), 2006. (Zitiert auf Seite 81)

[BHK+15] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. A Situation-Aware Workflow Modelling Extension. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015)*. 2015. (Zitiert auf den Seiten 9, 10, 46, 71, 75 und 85)

[BMBa] BMBF. Zukunftsbild „Industrie 4.0". `http://www.bmbf.de/pubRD/Zukunftsbild_Industrie_40.pdf`. (Zitiert auf Seite 17)

[BMBb] BMBF. Zukunftsprojekt Industrie 4.0. `http://www.bmbf.de/de/9072.php`. (Zitiert auf Seite 17)

[BMPR12] A. Bucchiarone, A. Marconi, M. Pistore, H. Raik. Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, S. 33–41. 2012. (Zitiert auf Seite 79)

[CCSC07] J. Choi, Y. Cho, K. Shin, J. Choi. A Context-aware Workflow System for Dynamic Service Adaptation. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications - Volume Part I*, ICCSA'07, S. 335–345. Springer-Verlag, Berlin, Heidelberg, 2007. (Zitiert auf Seite 76)

[DR09] P. Dadam, M. Reichert. The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science - Research and Development*, 23(2):81–97, 2009. (Zitiert auf Seite 81)

[Goo] Google. Cloud Messaging. `https://developers.google.com/cloud-messaging/`. (Zitiert auf den Seiten 65 und 72)

[HCC05] J. Han, Y. Cho, J. Choi. Context-Aware Workflow Language Based on Web Services for Ubiquitous Computing. In *Computational Science and Its Applications - ICCSA 2005*, Band 3481 von *Lecture Notes in Computer Science*, S. 1008–1017. Springer Berlin Heidelberg, 2005. (Zitiert auf Seite 76)

[HL14] F.-S. Hsieh, J.-B. Lin. Development of context-aware workflow systems based on Petri Net Markup Language. *Computer Standards & Interfaces*, 36(3):672 – 685, 2014. (Zitiert auf den Seiten 79 und 80)

[HRKD08] K. Herrmann, K. Rothermel, G. Kortuem, N. Dulay. Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, S. 108–113. 2008. (Zitiert auf den Seiten 77 und 79)

[HW12] G. Hohpe, B. Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Amsterdam, 1. aufl. Auflage, 2012. (Zitiert auf Seite 55)

[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 *SummerSOC*. 2015. (Zitiert auf den Seiten 10, 13, 18, 19, 20, 21, 48, 76 und 85)

[KELU10] O. Kopp, H. Eberle, F. Leymann, T. Unger. The Subprocess Spectrum. In *3rd International Conference on Business Processes and Services Computing (BPSC) – Informatik 2010*, Band P-177 von *Lecture Notes in Informatics (LNI)*, S. 267–279. Gesellschaft für Informatik e.V. (GI), 2010. (Zitiert auf den Seiten 10, 24 und 29)

[LFK+14] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, M. Hoffmann. Industrie 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014. (Zitiert auf den Seiten 9 und 17)

[LR00] F. Leymann, D. Roller. *Production Workflow Concept and Techniques*. Prentice Hall PTR, 2000. (Zitiert auf den Seiten 15 und 16)

[MBCP05] S. Modafferi, B. Benatallah, F. Casati, B. Pernici. A Methodology for Designing and Managing Context-Aware Workflows. In *Mobile Information Systems II*, Band 191 von *IFIP - The International Federation for Information Processing*, S. 91–106. Springer US, 2005. (Zitiert auf Seite 77)

[MGKR15] N. Mundbrod, G. Grambow, J. Kolb, M. Reichert. Context-Aware Process Injection. In C. Debruyne, H. Panetto, R. Meersman, T. Dillon, G. Weichhart, Y. An, C. A. Ardagna, Herausgeber, *On the Move to Meaningful Internet Systems: OTM 2015 Conferences*, Band 9415 von *Lecture Notes in Computer Science*, S. 127–145. Springer International Publishing, 2015. (Zitiert auf Seite 80)

[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*, iiWAS '11, S. 182–188. ACM, New York, NY, USA, 2011. (Zitiert auf Seite 78)

[Stu] U. Stuttgart. SitOPT Forschungsprojekt. `https://www.ipvs.uni-stuttgart.de/abteilungen/as/forschung/projekte/SitOPT`. (Zitiert auf Seite 17)

[TGD+08] F. Tang, M. Guo, M. Dong, M. Li, H. Guan. Towards Context-Aware Workflow Management for Ubiquitous Computing. In *Embedded Software and Systems, 2008. ICESS '08. International Conference on*, S. 221–228. 2008. (Zitiert auf Seite 80)

[W3C] W3C. Web Services Addressing (WS-Addressing). `http://www.w3.org/Submission/ws-addressing/`. (Zitiert auf den Seiten 57, 58 und 65)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, 2005. (Zitiert auf den Seiten 16, 57 und 58)

[Wel] M. Welsh. SEDA: An Architecture for Highly Concurrent Server Applications. `http://www.eecs.harvard.edu/~mdw/proj/seda/`. (Zitiert auf Seite 64)

[WHR09] H. Wolf, K. Herrmann, K. Rothermel. Modeling Dynamic Context Awareness for Situated Workflows. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, Band 5872 von *Lecture Notes in Computer Science*, S. 98–107. Springer Berlin Heidelberg, 2009. (Zitiert auf Seite 77)

[Wik] Wikipedia. Smart Factory. `https://de.wikipedia.org/wiki/Smart_Factory`. (Zitiert auf Seite 17)

[WK03] M. Weber, E. Kindler. The Petri Net Markup Language. In *Petri Net Technology for Communication-Based Systems*, Band 2472 von *Lecture Notes in Computer Science*, S. 124–144. Springer Berlin Heidelberg, 2003. (Zitiert auf Seite 79)

[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 78)

[WLZ14] P. Wang, H. Li, B. Zhang. Context-aware workflow modeling approach using OWL. In *Control and Decision Conference (2014 CCDC), The 26th Chinese*, S. 4161–4165. 2014. (Zitiert auf Seite 78)

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. Towards Situation-Aware Adaptive Workflows. In *Proceedings of the 13th Annual IEEE Intl. Conference on Pervasive Computing and Communications Workshops: 11th Workshop on Context and Activity Modeling and Recognition*, S. 32–37. IEEE, 2015. (Zitiert auf den Seiten 10, 15 und 18)

[ZHKL09] O. Zweigle, K. Häussermann, U.-P. Käppeler, P. Levi. Supervised Learning Algorithm for Automatic Adaption of Situation Templates Using Uncertain Data. In *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, S. 197–200. ACM, New York, NY, USA, 2009. (Zitiert auf Seite 19)

Alle URLs wurden zuletzt am 16. 11. 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