Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3721

Automatisierte, Ontologie-basierte Sensorintegration für das Internet der Dinge

Sven Grumbein

Studiengang:	Informatik
Prüfer/in:	PD Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Dipl.-Inf. Pascal Hirmer
Beginn am:	4. Mai 2015
Beendet am:	3. November 2015
CR-Nummer:	C.0, C.2.4, E.2, H.2.8, H.3.3, H.3.5

Kurzfassung

Das Internet der Dinge (IoT) bezeichnet im Allgemeinen die Vernetzung von technischen Geräten. Diese Geräte, auch intelligente Objekte genannt, enthalten dabei meist Sensoren, die Informationen über ein Objekt und dessen Umgebung bereitstellen. Derartige vernetzte Umgebungen werden als intelligente Umgebungen bezeichnet. Die „Industrie 4.0“-Initiative greift diese technologische Entwicklung auf und wendet sie auf Produktions- und Logistiksysteme an. Die vernetzten Daten ermöglichen es beispielsweise Prozesse und Workflows situationsbedingt zu adaptieren.

Das DFG-Forschungsprojekt `SitOPT` entwickelt ein System zur Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten. Teil dieses Systems ist eine Situationserkennung. Sie leitet aus Sensorwerten Situationen ab. Um dies zu ermöglichen, müssen Sensoren in das System integriert werden.

In dieser Arbeit wird ein Konzept zur automatisierten ad-hoc Sensorintegration entworfen und entwickelt. Dazu wird eine Ontologie als semantische Wissensbasis entworfen, die Objekte, Sensoren und ihre Eigenschaften semantisch verknüpft. Über einen REST-basierten Service wird die Ontologie bedient und eine Sensorintegration initiiert. Adapter fungieren als Schnittstelle zwischen Sensoren und der Situationserkennung und werden im Prozess der Sensorintegration auf das Objekt übertragen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1 Einleitung

Das Internet der Dinge (IoT) ist eine derzeit aufstrebende Technologie [For15]. In ihm werden stetig mehr werdende technische Geräte in sogenannte intelligente Umgebungen eingebunden. Diese Geräte besitzen meist eine Vielzahl Sensoren und liefern entsprechend viele Sensordaten. All diese Sensorendaten vernetzt in einer intelligenten Umgebung offerieren eine Menge neuer Möglichkeiten. Zum Beispiel können höherwertige Situationen erkannt und auf diese automatisiert reagiert werden.

Die Industrie 4.0 versteht sich als technologische Revolution zum bisherigen Stand [BMBF15]. Sie beruht auf der Idee des Internets der Dinge und wendet diese auf Fertigung- und Logistikprozesse an. Diese Prozesse können – z. B. als Workflow modelliert – bedarfsorientiert und situationsbedingt ausgeführt werden.

Das DFG-Forschungsprojekt `SitOPT` entwickelt ein System zur Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten. Eine Situationserkennung (`SitRS`) nimmt sich der Aufgabe an, eine Situation aus ermittelten Sensordaten abzuleiten. Tritt eine Situation ein, kann `SitOPT` einen Workflow entsprechend adaptieren.

Sensoren müssen in `SitRS` integriert werden, um an die Sensordaten zur Situationserkennung zu kommen. Bisher mussten Sensoren aufwendig manuell in das System integriert werden.

Ziel und Motivation dieser Arbeit ist es, eine automatisierte ad-hoc Sensorintegration zu schaffen. Über eine Ontologie erhält diese Sensorintegration semantisches Wissen über Objekte und Sensoren.
Der Hintergrund und die Motivation werden in Kapitel 2 detailliert vertieft.

Gliederung

Diese Diplomarbeit ist in folgender Weise gegliedert:

Kapitel 2 – Hintergrund und Motivation beschreibt `SitOPT` als Hintergrund dieser Arbeit und die Motivation einer Sensorintegration.

Kapitel 3 – Grundlagen erklärt die Grundlagen der eingesetzten Technologien.

Kapitel 4 – Verwandte Arbeiten weist auf verwandte Arbeiten hin und gibt einen Einblicke in diese.

Kapitel 5 – Konzept zeigt die konzeptionelle Lösung.

Kapitel 6 – Implementierung schildert die technische Umsetzung der konzeptionellen Lösung.

Kapitel 7 – Zusammenfassung und Ausblick fasst die Arbeit zusammen und gibt einen Ausblick wie die Lösung weiterentwickelt werden kann.

Begriffserklärung

In dieser Diplomarbeit werden verschiedene Begriffe wiederholt verwendet. An dieser Stelle wird erklärt welche Bedeutung ihnen inne liegt.

Situation – eine Situation ergibt sich aus einer bedingten Zustandsänderung gemessen durch Sensoren.

Internet der Dinge – auch *Internet of Things* bzw. *IoT*. Die Differenzierung in diesem Themenbereich ist schwierig und in der Literatur wird dieser und die folgenden Begriffe (s.u.) teils synonym verwendet. In dieser Diplomarbeit ist mit dem Begriff das Konzept der Vernetzung und Adressierung verschiedener Objekte über das Internet gemeint. Kapitel 3.1 geht detaillierter auf *IoT* ein.

Objekt – ein Objekt ist der Gegenstand oder das Gerät respektive das „Ding“ aus dem Internet der Dinge.

SMART <AUSDRUCK> – <AUSDRUCK> ist eine intelligente Einheit, die mit ihren beinhaltenden Elementen bzw. mit seiner Umgebung vernetzt ist. In der Vernetzung findet ein Informationsaustausch statt, mittels der intelligent agiert wird.
Beispiele: *SMART Factory*, *SMART Environment*, *SMART Car*, *SMART Home*

Über dieses Dokument

Diese Diplomarbeit wurde mit LaTeX mittels der MiKTeX-Distribution gesetzt. Das Dokument basiert auf der inoffiziellen Vorlage[1] für Diplomarbeiten für Informatik an der Universität Stuttgart.

Als Texteditoren wurden sowohl Notepad++[2], als auch TeXnicCenter[3] (Version 2.02) eingesetzt.

Vektorgrafiken, sofern nicht zitiert, wurden mit yEd[4] und Inkscape[5] erstellt.

[1] https://github.com/latextemplates/uni-stuttgart-computer-science-template
[2] https://notepad-plus-plus.org/
[3] http://www.texniccenter.org/
[4] http://www.yworks.com/en/products/yfiles/yed/
[5] https://inkscape.org/

2 Hintergrund und Motivation

Diese Arbeit entstand vor dem Hintergrund des Forschungsprojektes SitOPT. Einleitend vor der Motivation dieser Arbeit wird als erstes das Forschungsprojekt SitOPT (2.1) in Grundzügen beschrieben. Danach folgt die Komponente der Situationserkennung (SitRS, 2.2), die ein Bestandteil von SitOPT darstellt. Anschließend wird die Motivation der Sensorintegration (SeInt, 2.3) – also dieser Diplomarbeit – beschrieben. Diese Sensorintegration knüpft an die Situationserkennung an.

2.1 SitOPT

Das Internet der Dinge verbreitet sich im derzeitigen Trend[1] immer stärker. Die zugehörige Hardware wird günstiger und die Anzahl der Sensoren und der gewonnenen Sensordaten wächst. Wirtschaft und Industrie verfolgen unter dem Begriff „Industrie 4.0“ das Ziel Prozesse automatisch zu steuern [Jas12]. Dazu muss sich ein System seiner Umgebung bewusst sein und auf Situationen reagieren können.

SitOPT als ein universelles situationsbezogenes und adaptives Workflow Management System nimmt sich dieser Entwicklung an. Die gewachsene Informationsmenge moderner Informationssysteme, des Internets der Dinge und der Industrie 4.0 gestalten eine robuste Workflow-Modellierung aufwendig und komplex [WSBL15].

Mit der Anzahl der Datenquellen, die ein System erfasst, wächst auch die Anzahl der Zustände im System. Aus einer Zustandsänderung kann sich eine neue Situation ableiten, welche einen geänderten Workflow notwendig macht.

> *Beispiel einer Situation:*
> *Die Temperatur einer Maschine steigt über einen Grenzwert. Es stellt sich die Situation ein, dass die Maschine heiß gelaufen ist. Die Workflow-Änderung lässt die Maschine nun langsamer laufen oder pausiert sie gar, damit sie abkühlen kann.*

[1] Google Trends zu *Internet of Things* https://www.google.com/trends/explore#q=internet%20of%20things

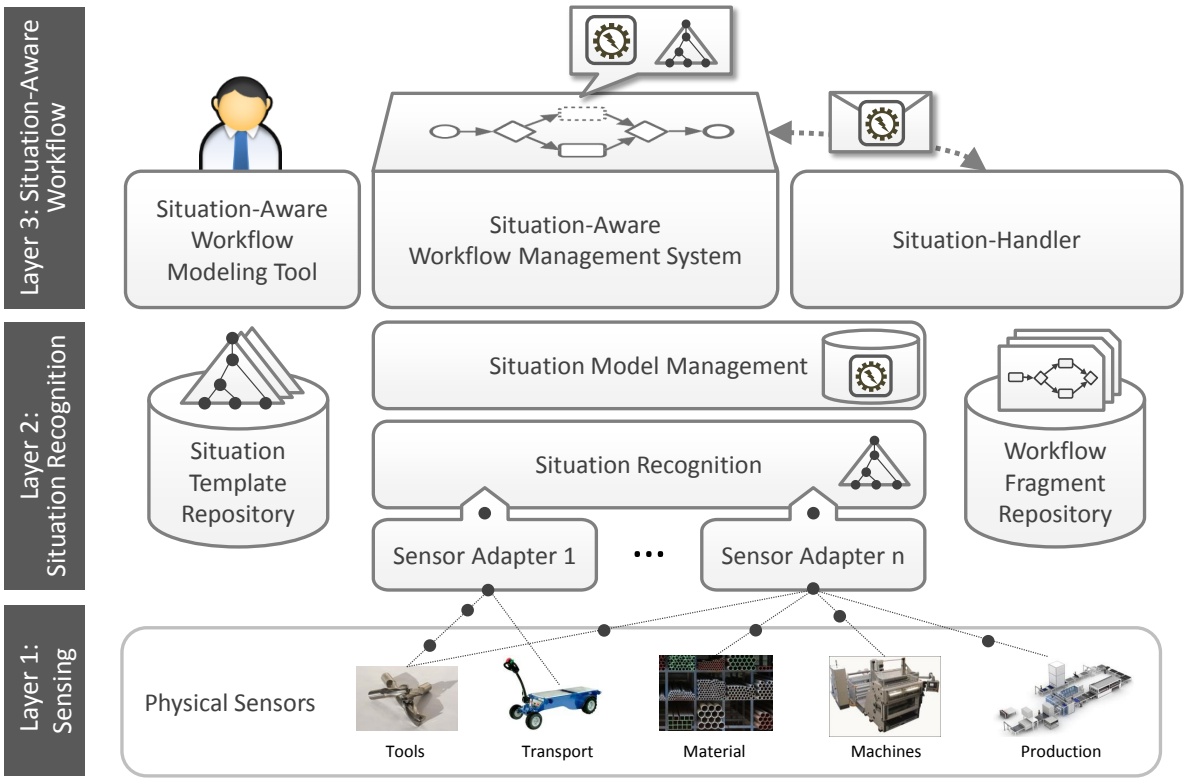

Abbildung 2.1: `SitOPT` Übersicht (Quelle: [WSBL15], Fig. 3)

`SitOPT` lässt sich in drei Schichten untergliedern (Abb. 2.1). Die Sensorebene (*Layer 1: Sensing*) erfasst Sensordaten und dient als Datenquelle für die darüber liegende Ebene der Situationserkennung (*Layer 2: Situation Recognition*). Die Situationserkennung bewertet die Daten aus den Sensoren und signalisiert gegebenenfalls einen Situationseintritt. Die Workflow-Ebene (*Layer 3: Situation-Aware Workflow*) adaptiert Workflows anhand von erkannten Situation mit vordefinierten Workflow-Fragmenten (Abb. 2.2).

`SitOPT` ist ein durch die Deutsche Forschungsgemeinschaft (DFG) gefördertes Projekt[2].

[2]Deutsche Forschungsgemeinschaft (DFG), Zuwendung 610872
„Optimierung und Adaption situationsbezogener Anwendungen basierend auf Workflow-Fragmenten“
`http://gepris.dfg.de/gepris/projekt/252975529`

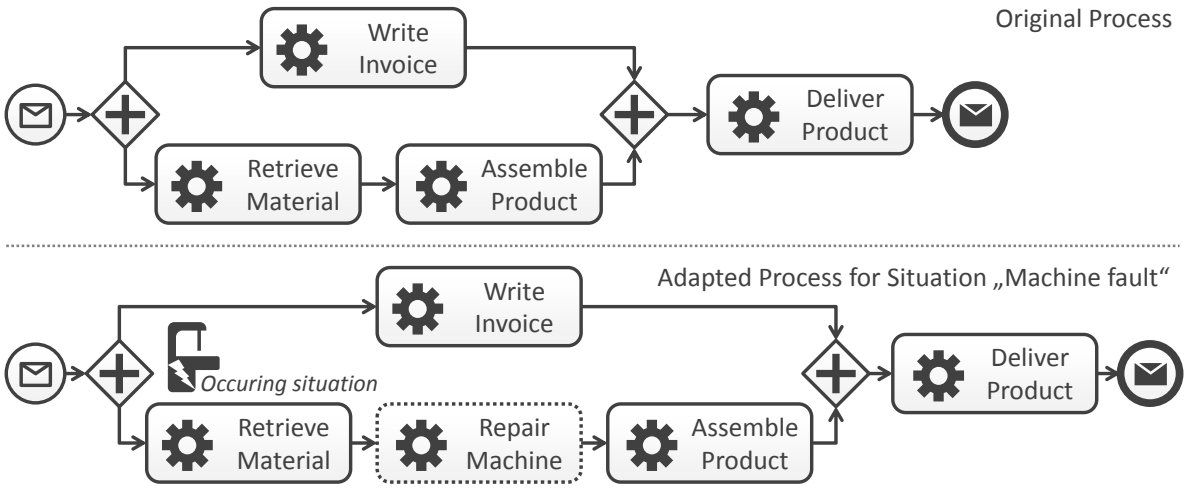

Abbildung 2.2: Workflow Adaption (Quelle: [BHK+])

2.2 Situationserkennung – SitRS

Der Situationserkennungs-Service (SitRS) [HWS+15] ist ein cloud-basiertes System. Es abstrahiert aus systemnahen („low-level“) Sensordaten eine Situation, die von einer kontextsensitiven Anwendung weiterverarbeitet werden kann. In SitOPT beispielsweise ist die Workflow-Ebene (Abb. 2.1 – Layer 3) eine solche kontextsensitive Anwendung.

Eine Situation ist das Ergebnis einer gegen Bedingungen geprüfte Auswertung von Sensordaten.

SitRS besteht aus mehreren Komponenten (siehe Abb. 2.3):

- **Situation Registration Service**
  Hier wird eine Situation registriert, die erkannt werden soll. Tritt eine solche Situation ein, wird dieses Ereignis über *Push*- oder *Pull*-Benachrichtigungen bekannt gegeben.
- **Situation Template Repository**
  Diese Ablage enthält vorab definierte Modelle einer Situation, sogenannte *Situation Templates* (ST). Ein ST ist ein *Situation-Aggregation-Tree* (SAT) [ZHKL09]. Abbildung 2.4 zeigt ein Beispiel-ST einer Maschinenüberhitzung. Die Blätter des Baumes enthalten die Sensordaten und bilden die Kontextknoten, diese werden in den Bedingungsknoten ausgewertet und bilden, verknüpft über Operationsknoten, die ausgewertete Situation an der Wurzel.

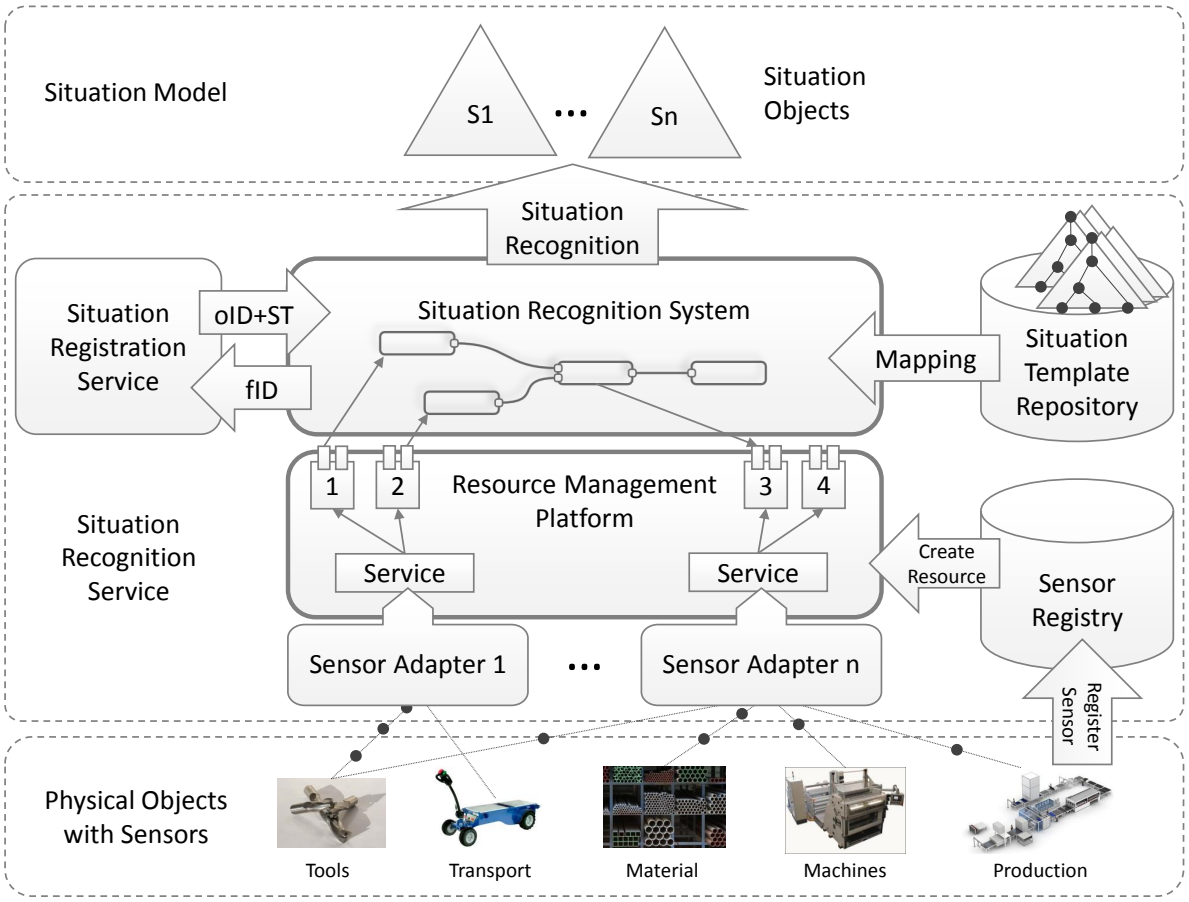

Abbildung 2.3: `SitRS` Architektur (Quelle: [HWS+15], Fig. 2)

- **Situation Recognition System**
  Die Situationserkennung stellt eine der Kernkomponenten von `SitRS` dar. Mit der Situationsregistrierung wird ein ST eingelesen und in eine ausführbare Repräsentation, z. B. als CEP-Query[3], überführt. Diese ausführbare Repräsentation wird dem ausführenden System übergeben. In der derzeitigen prototypischen Implementierung von `SitRS` wird hierzu Node-RED[4] verwendet.
- **Resource Management Platform**
  Die *Resource Management Plattform* (`RMP`) ist die andere Kernkomponente. Sie stellt die Sensordaten als einheitliche REST-Ressourcen zur Verfügung. Die `RMP` ist das Thema einer anderen Diplomarbeit [Jan15].

[3] complex event processing – `https://en.wikipedia.org/w/index.php?title=Complex_event_processing&oldid=680354106`
[4] `http://nodered.org/`

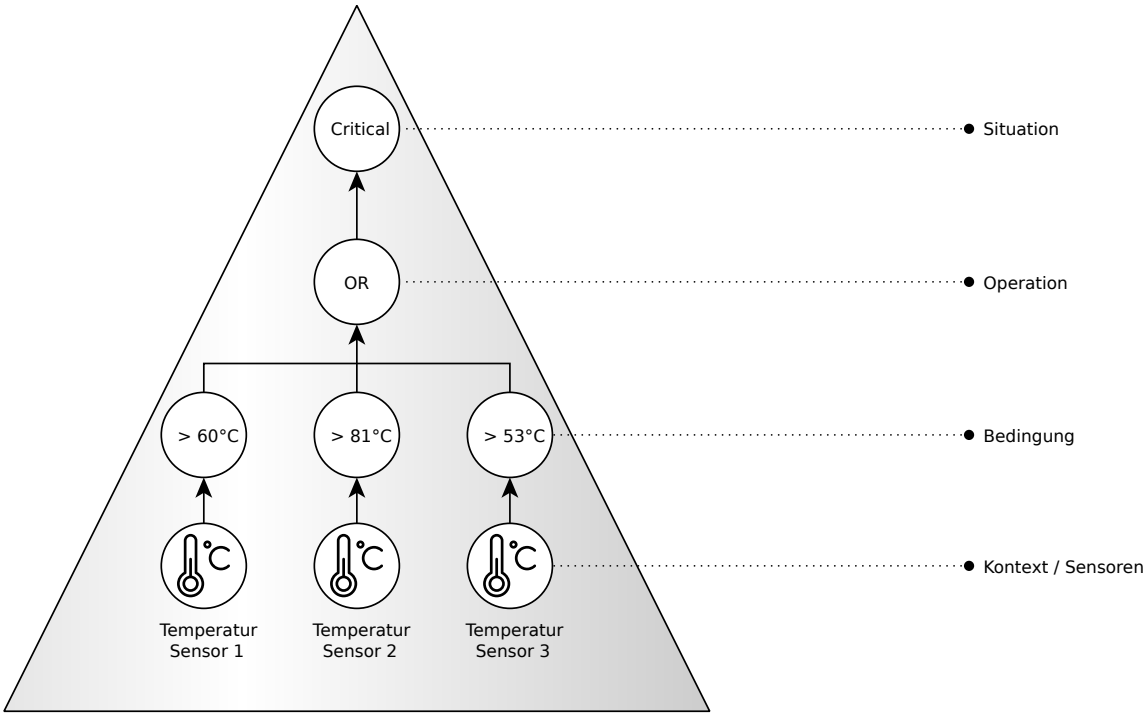

Abbildung 2.4: Beispiel eines *Situation Template* (nach [HWS+15], Fig. 4)

- **Sensor Registry**
  Hier werden Sensoren registriert. Sie steht im Verbund mit der RMP und verknüpft das Objekt, den Sensor und die Sensordaten in einem Kontext miteinander.

Diese Diplomarbeit zur Sensorintegration knüpft direkt an die *Resource Management Platform* und die *Sensor Registry* an.

2.3 Sensorintegration – SeInt

Bisher geschah die Sensorregistrierung manuell. Ebenso mussten bisher die gewonnenen Sensordaten manuell in das System eingebunden werden. Die manuelle Integration der Sensoren gestaltet sich komplex. Sie bereitet einen hohen Aufwand und ist zeit- und kostenintensiv. Auf manuellem Wege ist sie ist wenig flexibel, wenn es darum geht weitere Objekte und Sensoren dem System hinzuzufügen. Darüber hinaus ist die manuelle Integration der Sensoren fehleranfällig, da keine Automatismen über eine definierte Schnittstelle greifen.

Die Sensorintegration (SeInt) als Ziel dieser Diplomarbeit ermöglicht eine automatisierte ad-hoc Integration der Sensoren.

`SeInt` bietet eine Schnittstelle an die *Resource Management Plattform* und *Sensor Registry* der Situationserkennung, um einen Sensor zu registrieren. Hierzu verwendet `SeInt` eine Ontologie als Wissensbasis, um Informationen über Objekte und Sensoren zu verwalten. Zu dieser Wissensbasis gehören neben Informationen über die Lokalisierung insbesondere Informationen zu Sensoreigenschaften wie z. B. die Messgenauigkeit.

Im Rahmen dieser Diplomarbeit kommen an einen *Raspberry Pi*[5] angebundene Sensoren zum Einsatz. Um diese Sensoren an die *Resource Management Plattform* anzubinden, liefert `SeInt` einen Mechanismus, der im Zuge der Registrierung an der *Sensor Registry* einen Adapter ausliefert. Dieser Adapter hat zwei Aufgaben: zum einen liest er die Sensordaten eines Sensors aus, zum anderen sendet er die Sensordaten an die *Resource Management Plattform*.

Die automatisierte ad-hoc Integration der Sensoren reduziert den Aufwand und die Kosten gegenüber einer manuellen Integration. Des Weiteren ist die Lösung durch `SeInt` flexibler und ermöglicht es, weitere Sensoren auf einfache Weise in das System einzubinden.

Motivation und Ziel dieser Diplomarbeit ist es, die hier formulierte Sensorintegration zu konzipieren und prototypisch zu implementieren.

[5] Raspberry Pi Foundation – `https://www.raspberrypi.org/`

3 Grundlagen

Dieses Kapitel beschreibt die notwendigen Grundlagen, die für das Verständnis dieser Arbeit erforderlich sind.

3.1 Internet der Dinge

Der Begriff „*Internet of Things*“ (Internet der Dinge, IoT) geht auf Kevin Ashton zurück, der diesen in einer Präsentation 1999 verwendete [Ash09].

Das Konzept des Internets der Dinge wurde bereits 1991 in einer Zukunftsversion von Mark Weiser beschrieben [Wei91]. In seiner Zukunftsversion vernetzt er unterschiedlichste Geräte und Computer miteinander, mit dem Begriff *Internet der Dinge* werden Objekte miteinander vernetzt. Die Vernetzung ist dabei sowohl kabelgebunden, als auch drahtlos mit mobilen Geräten und Computern. Die Technologie – Computer – verschwindet in den Objekten. Diese können eine Vielzahl unterschiedlicher Gegenstände der realen Welt darstellen, z. B. Lichtschalter, Thermostate, RFID-Systeme, Sensoren. Sie wird auf diesem Wege unsichtbar für den Menschen und verschwindet aus seiner bewussten Wahrnehmung. Sie soll ihn durch intelligentes Agieren unmerklich bei seinen Tätigkeiten unterstützen.

All diese vernetzten Objekte tauschen beständig Informationen aus bzw. berechnen Daten, was Mark Weiser als „ubiquitous computing“[1] bezeichnete. Angereichert durch diese vernetzten Informationen können die Objekte intelligent agieren. Um intelligent agieren zu können benötigen die Objekte eine Kenntnis ihrer Umgebung wie z. B. den Ort an dem sie sich befinden.

Das Internet der Dinge schließt die Lücke zwischen der virtuellen und der realen Welt. Ein jedes Objekt ist dazu eindeutig in einem Netzwerk identifizierbar und adressierbar. Ein Objekt kann so seine Daten im Netzwerk mit anderen Objekten austauschen [ITU12]. Als Netzwerk dient hierbei die Infrastruktur, Technologie und die Protokolle des Internets.

[1] allgegenwärtige Datenverarbeitung

3.2 Industrie 4.0

Das Zukunftsprojekt *Industrie 4.0* des Bundesministerium für Bildung und Forschung steht nach der Mechanisierung, der Massenfertigung und der Digitalisierung für die vierte industrielle Revolution.

Der Begriff selbst ist unscharf und vage definiert – Ziel ist allerdings die intelligente Fabrik (*SMART Factory*). Sie reagiert mit Produktions- und Logistikprozessen dynamisch und bedarfsorientiert auf Betriebsbedingungen und Aufträge [BMBF15].

Durch das Internet getrieben wandeln sich heutige Produktionssysteme zu *cyber-physischen Systemen* (CPS) auf den gleichen technologischen Grundlagen wie das Internet der Dinge. CPS beschreibt die Integration von Datenverarbeitung in physikalische Prozesse [LS11].

Heute bestimmen weitestgehend die Produktionssysteme und Produktionsprozesse über das Produkt. Mit der Industrie 4.0 findet ein Paradigmenwechsel statt. Das Produkt nimmt eine aktive Rolle ein und gibt selbst vor wie mit ihm agiert werden soll [KLW11], [Jas12].

3.3 Sensoren

Das englische Wort *sensor* wird in DIN 1319-1 als „(Meßgrößen-)Aufnehmer (4.4)“ bezeichnet. Er ist „Teil eines Meßgerätes oder einer Meßeinrichtung, der auf eine Meßgröße unmittelbar anspricht“. Als Beispiel nennt die Norm den Schwimmer eines Flüssigkeitsstand-Anzeigers [1319-1].
In der Literatur finden sich weitere Synonyme für Sensor[2], z. B. (Mess-)Fühler oder (Messwert)-Aufnehmer.

In dieser Diplomarbeit soll als Sensor ein jeder Datengeber dienen, der Daten in einer datenverarbeitungskompatiblen Form liefert. Die gewonnenen Messdaten physikalischer Sensoren werden so konvertiert, dass sie in Form von Daten im informationstechnischen Sinne vorliegen. Außerdem wird der Begriff dahingehend erweitert, dass als Sensor auch die Bereitstellung einer Statusinformation eines Objektes gilt. Zum Beispiel ist die Messung der Rechnerauslastung (*CPU load*) nicht das direkte Resultat einer physikalischen Messung, sondern die Auswertung einer Rechnerressource, die den Auslastungsstatus widerspiegelt.

Ein aktuelles Smartphone – Objekt – besitzt diverse Sensoren, z. B. Beschleunigungssensoren, Helligkeitssensor, Gyroskop, Höhenmesser, GPS und weitere. Auch kann man die Kamera als optischen Sensor bzw. Sende-/Empfangsmodule wie GSM, LTE oder WLAN als Sensoren für ein Funknetzwerk auffassen.

[2]Deutschen Nationalbibliothek, Sachbegriff „Sensor“ – `http://d-nb.info/gnd/4038824-4`

Sensoren zeichnen sich durch diverse Merkmale und Eigenschaften aus, die in ihrer Gesamtheit einen Sensor beschreiben und die Messergebnisse in ihrer Qualität quantifizieren.

Jeder Sensor besitzt einen Messbereich für den er ausgelegt ist. Messungen außerhalb dieses Bereiches führen zu keinem oder einem unzuverlässigen bzw. gänzlich falschen Messergebnis. Die Messgenauigkeit oder auch Messauflösung bestimmt wie nahe Messwerte beieinander liegen können, bevor sie nicht mehr zu unterscheiden sind. Die Einstelldauer respektive Einschwingzeit definiert die Dauer bis ein Messwert bestimmt ist, d.h. ein einzelner Messvorgang abgeschlossen ist. Die bisher genannten Merkmale lassen sich im Allgemeinen durch einfache Zahlenwerte zzgl. Messeinheit beschreiben. Für gewöhnlich sind die Werte innerhalb des Messbereiches auch linear abhängig von der Messgröße.

Es gibt aber auch komplexe Merkmale wie zum Beispiel die Hysterese. Dabei wird ein aktueller Messwert durch vorherige Messungen beeinflusst. Dieses Verhalten wird mit einer Hysteresekurve beschrieben.
Auch kann der Sensor selbst einen signifikanten Einfluss auf die Messung in Abhängigkeit der Messgröße haben.

3.4 Ontologie

> Ontologie erklärt die
> Beschaffenheit der Welt
>
> *(Heinz von Förster)*

Ontologie ist ein Begriff aus der Philosophie und beschäftigt sich mit der „Natur des allgemeinen Seins“ [Bra96], sie ist Teilgebiet der Metaphysik, die sich mit der Wesen der Existenz bzw. des Seienden auseinandersetzt. Sie fragt unter anderem nach dem Warum und den Ursprung des Seins von allem.

In einem fiktiven Dialog verschiedener wissenschaftlicher Disziplinen beschreiben Busse et al. den Begriff. Gemeinhin wird Ontologie zur Beschreibung der Welt für ein gemeinsames Verständnis verwendet, wobei die Informatik den Begriff in einer übertragenen Bedeutung verwendet und die Existenz als gegeben hinnimmt [BHL+14].

3.4.1 Ontologie in der Informatik

Thomas R. Gruber definiert eine Ontologie prägnant mit: „*An ontology is an explicit specification of a conceptualization*“ [Gru93]. In der Literatur wird diese Definition meist erweitert zitiert:

> „ *An ontology is a formal, explicit specification of a shared conceptualisation*“
> Eine Ontologie ist eine formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung[3]

Wobei diese Definition auf Studer et al. zurück zurückgeht [SBF98].

Die Konzeptualisierung umfasst `Klassen`[4], `Instanzen`[5] und `Relationen`[6] einer Wissensdomäne. Die `Klassen` und `Relationen` sind häufig in einer hierarchischen Struktur aufgebaut, können aber allgemein in einem beliebigen gerichteten Graph angeordnet sein. Prinzipiell werden `Klassen` in `Relationen` zueinander gesetzt, womit dem Vokabular eine semantische Bedeutung zugeordnet wird. Weiterhin kann es `Axiome` geben, die weitere Regeln und Einschränkungen enthalten können. Formalisiert repräsentieren Ontologien eine maschinenlesbare und teilbare Wissensbasis.

Abbildung 3.1 zeigt ein Beispiel einer einfachen Ontologie. Die `Klasse` *Gemälde* ist der `Klasse` *Werk* untergeordenet. Die `Relationen` beschreibt die Beziehung dieser beiden `Klassen` zueinander. *Norwegen* ist eine `Instanz` der `Klasse` *Land*.

Formal lässt sich eine Ontologie mit folgenden Tuple beschreiben [Yil06]:
$\mathcal{O} = \left\{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, \mathcal{H}^{\mathcal{R}}, \mathcal{I}_{\mathcal{C}}, \mathcal{I}_{\mathcal{R}}, \mathcal{A}^{\mathcal{O}}\right\}$, wobei

$\mathcal{C}$	:=	die Menge aller `Klassen` (engl. *concepts*)
$\mathcal{R}$	:=	die Menge der `Relationen` zwischen `Klassen` mit $r_i \in \mathcal{R}$ und $r_i \rightarrow \mathcal{C} \times \mathcal{C}$
$\mathcal{H}^{\mathcal{C}}$	:=	die Hierarchie der `Klassen` $\mathcal{C}$ mit der Relation $\mathcal{H}^{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$, wobei $\mathcal{H}^{\mathcal{C}}(c_i; c_j)$ mit $c_i, c_j \in \mathcal{C}$ ausdrückt, dass c_i eine Unterklasse von c_j ist.
$\mathcal{H}^{\mathcal{R}}$	:=	die Hierarchie der `Relationen` $\mathcal{R}$ mit der Relation $\mathcal{H}^{\mathcal{R}} \subseteq \mathcal{R} \times \mathcal{R}$, wobei $\mathcal{H}^{\mathcal{R}}(r_i; r_j)$ mit $r_i, r_j \in \mathcal{R}$ ausdrückt, dass r_i eine Unterrelation von r_j ist.
$\mathcal{I}_{\mathcal{C}}$	:=	die Menge der `Instanzen` der `Klassen` $\mathcal{C}$
$\mathcal{I}_{\mathcal{R}}$	:=	die Menge der `Instanzen` der `Relationen` $\mathcal{R}$
$\mathcal{A}^{\mathcal{O}}$	:=	die Menge der `Axiome` der Ontologie $\mathcal{O}$

[3] Begriffsbildung
[4] in der Literatur auch *Begriff* oder *Konzept*
[5] in der Literatur auch *Individuum*
[6] in der Literatur auch *Beziehung*, seltener *Attribut* oder *Eigenschaft*; engl. auch *slot* oder *role*

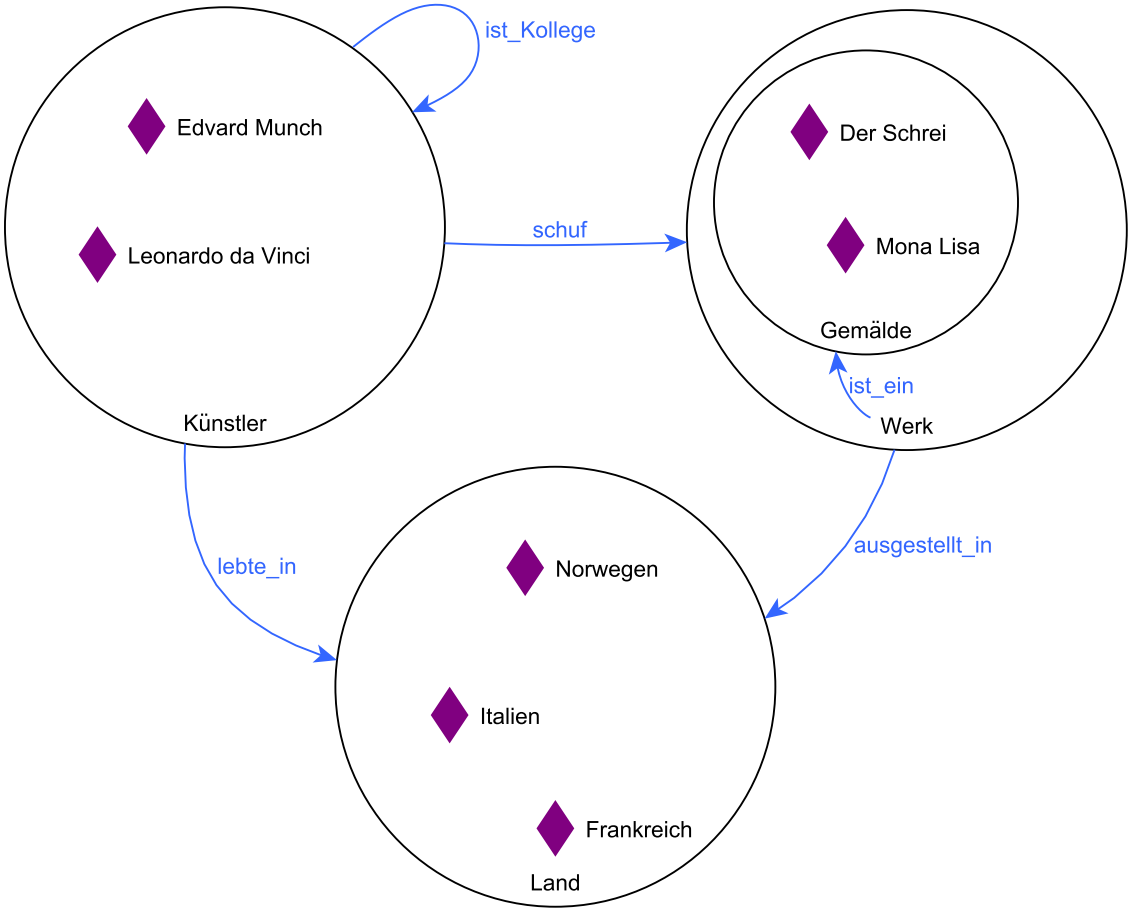

Abbildung 3.1: Ontologie Beispiel

Andere formale Beschreibung finden sich z. B. in [GOS09] oder [Men02].

Die einfachste Ontologie – eine zusammenhangslose Sammlung von `Klassen` – stellt ein Glossar da. Durch eine hierarchische Ordnung der `Klassen` eingeschränkt, erhält man eine Taxonomie.

Einer Beschreibungslogik folgend erhält man eine komplexe Ontologiesprache. Schränkt man die Beschreibungslogik auf $\mathcal{SHOIN}(D)$ ein (vgl. z. B. [BN03]), gelangt man zu einer vollständig berechenbaren und entscheidbaren Untermenge der Prädikatenlogik erster Stufe. Dies ermöglicht Inferenz mit der sich Wissen schlussfolgern lässt.

Angenommen man wüsste aus der Ontologie nach Abbildung 3.1 nicht, dass *Leonardo da Vinci* (`Instanz`) ein *Künstler* (`Klasse`) ist. Dann kann man aus dem Wissen, dass *Leonardo da Vinci* (`Instanz`) das *Werk* (`Klasse`) *Mona Lisa* (`Instanz`) und das ein *Künstler* (`Klasse`)

ein *Werk* (`Klasse`) *schuf* (`Relation`), ableiten, dass *Leonardo da Vinci* (`Instanz`) ein *Künstler* (`Klasse`) sein muss.

Für Ontologien gibt es eine Reihe formaler Beschreibungssprachen mit dem Ziel, von Software verarbeitet und semantisch verstanden werden zu können. In dieser Diplomarbeit wird *Web Ontology Language* (OWL) verwendet. OWL basiert technisch auf dem im Folgenden erläuterten *Resource Description Framework*.

3.4.2 Resource Description Framework und RDF Schema

Das *Resource Description Framework* (RDF) ist ein abstraktes Datenmodell, um Ressourcen im Internet in Form eines gerichteten Graphen darzustellen. Es ist computerlesbar und dient vor allem dem Informationsaustausch zwischen verschiedenen Applikationen.

Das zentrale Element in RDF sind Tripel bestehend aus einem Subjekt, einem Prädikat und einem Objekt. Aussagen werden mit Hilfe solcher Tripel gebildet. Die Menge aller Tripel wird RDF-Graph genannt.

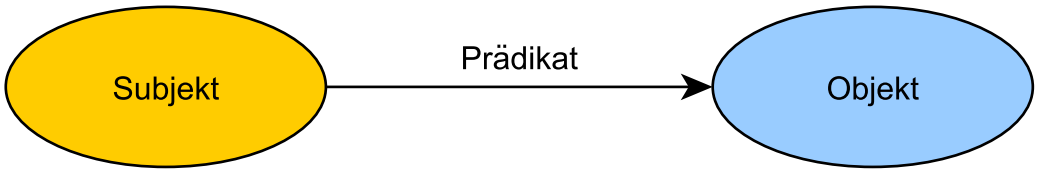

Abbildung 3.2: RDF Tripel – Subjekt, Prädikat, Objekt (nach [RDF1.1], Fig. 1)

Subjekt und Prädikat bestehen aus einem IRI[7]. Das Objekt kann sowohl ein IRI als auch ein wertbehaftetes Literal sein. Darüber hinaus können Subjekt und Objekt auch durch einen leeren Knoten repräsentiert werden. Ein IRI in RDF 1.1 ist die verallgemeinerte Version eines URI[8] aus RDF 1.0. Beide bezeichnen eine Ressource eindeutig und entsprechen den `Instanzen` einer Ontologie.

Es gibt unterschiedlich Serialisierungen von RDF-Graphen, z. B. RDF/XML[9] oder Turtle[10]. Die Syntax und Konzepte von RDF beschreibt [RDF1.1] detailliert. [RDFSE] beschreibt die Semantik im Detail.

[7]Internationalized Resource Identifiers – http://www.ietf.org/rfc/rfc3987.txt
[8]Uniform Resource Identifier – http://www.ietf.org/rfc/rfc3986.txt
[9]RDF 1.1 XML Syntax – `http://www.w3.org/TR/rdf-syntax-grammar/`
[10]RDF 1.1 Turtle – `http://www.w3.org/TR/turtle/`

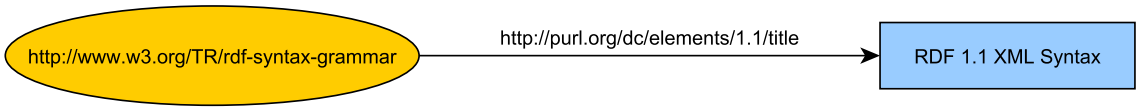

Abbildung 3.3: RDF Graph - Beispiel (nach [RDFX], Fig. 1)

Zu einer Beschreibungssprache für eine Ontologie mit Hierarchien wird RDF erst durch RDF-Schema. RDF-Schema ist eine semantische XML Erweiterung für RDF. Es erweitert das Vokabular von RDF um `Klassen` und `Relationen`[11].

Mit folgenden Präfixen werden die Namensräume von RDF und RDF-Schema voneinander abgegrenzt:

`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`	für RDF-Schema
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	für RDF

Abbildung 3.4 verdeutlicht einige RDF-Schema Konstrukte am Beispiel von zwei Hierarchien: Künstler-Maler und Kunstwerk-Gemälde.

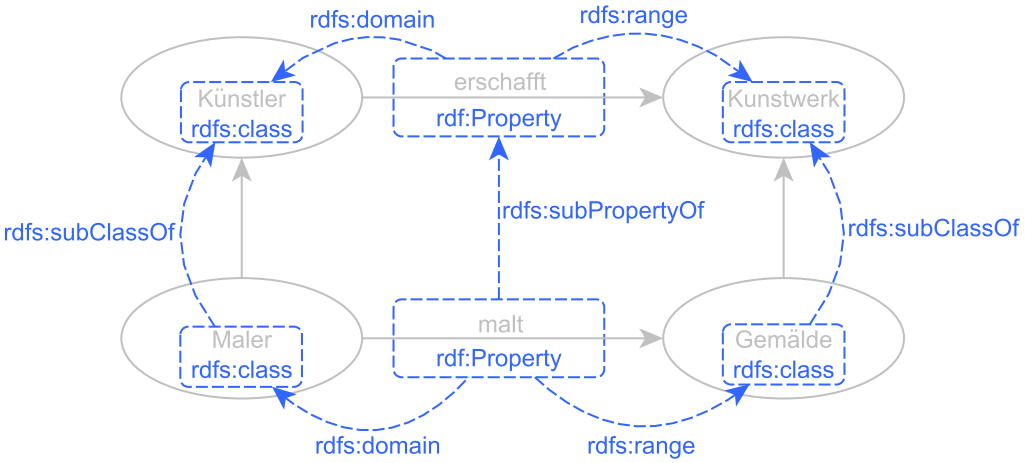

Abbildung 3.4: Beispiel RDF-Schema Graph

[11] In [RDFS] und weiterer Literatur unglücklich als Eigenschaften (*property*) bezeichnet – verwechselbar mit der `Klasse` namens `rdf:Property`

Wichtige Klassen-Konstrukte von RDF-Schema sind:

rdfs:Resource	alle RDF Ressourcen sind Instanzen dieser Klasse; alle anderen Klassen sind der Klasse rdfs:Resource untergeordnet
rdfs:Class	weist einer Ressource eine Klasse zu
rdfs:Literal	die Klasse aller Literale wie String und Integer-Werte
rdf:Property	stammt aus dem RDF Vokabular und zeichnet eine Relation aus

Zu den wichtigen Relations-Konstrukten von RDF-Schema zählen:

rdfs:domain	gibt die Quelle der gerichteten Relation an
rdfs:range	gibt das Ziel der gerichteten Relation an
rdfs:subClassOf	ordnet Klassen hierarchisch zueinander, transitive Relation
rdfs:subPropertyOf	ordnet Relationen hierarchisch zueinander, transitive Relation
rdf:type	gibt an, dass eine Instanz aus einer bestimmten Klasse ist

Eine vollständige und detaillierte Beschreibung findet sich unter [RDFS].

RDF-Schema liefert notwendige Grundlagen, um eine einfache Ontologie zu entwerfen. Jedoch fehlen RDF-Schema einige Merkmale für eine umfangreiche Beschreibungssprache für Ontologien. Es wird ein erweitertes Vokabular benötigt, um beispielsweise Relationen stärker einzuschränken. So fehlt RDF-Schema die Möglichkeit Relationen mit einer Kardinalität zu versehen. Auch benötigt es weitere Regeln, um mehr Wissen schlussfolgern zu können wie es beispielsweise disjunkte Klassen böten [Lac05].

Für umfangreichere Beschreibungsmöglichkeit, leistungsfähigere Inferenz und verfeinerte semantische Konzepte wurde *Web Ontology Language* (OWL) entworfen.

3.4.3 Web Ontology Language – OWL

Web Ontology Language (OWL) ist eine Beschreibungssprache für das Semantische Web, mit der sich Wissen über Dinge und Beziehungen zwischen Dingen beschreiben lässt. OWL ist eine formale Beschreibung für eine Ontologie. Die Entwicklung von OWL entstammt den Erfahrungen und Erkenntnissen, die man durch DAML+OIL[12] gewann. Technisch setzt OWL auf RDF und RDF-Schema auf (vgl. Abbildung 3.5) [OWL].

[12] http://www.w3.org/TR/daml+oil-reference

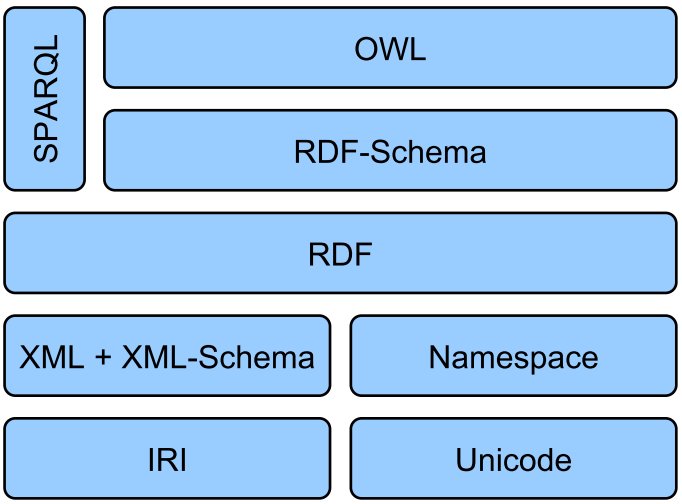

Abbildung 3.5: OWL Stack (nach [WSEM])

OWL gliedert sich in die drei Untersprachen, die sich über erlaubte beziehungsweise eingeschränkte OWL-Konstrukte definieren:

- OWL Lite
  besteht aus der geringsten Menge der OWL-Konstrukte und eignet sich für einfache Klassifikationen in Hierarchien mit einfachen Einschränkungen der erlaubten OWL-Konstrukte. So erlaubt OWL Lite z. B. keine disjunkten `Klassen` oder schränkt Kardinalitäten auf Werte von 0 oder 1 ein.
- OWL DL
  besteht aus der Menge aller OWL-Konstrukte. Es entspricht einer vollständig entscheidbaren Beschreibungslogik bei größtmöglicher Ausdrucksstärke. Dafür unterliegen OWL-Konstrukte Einschränkungen, so müssen `Klassen`, `Instanzen`, `Relationen` paarweise verschieden bezeichnet sein. Die Menge aller Objekt-`Relationen` und Datentyp-`Relationen` ist disjunkt, weswegen es z. B. keine inverse Datentyp-Relation geben darf.
- OWL Full
  besitzt die größte Ausdrucksstärke und erlaubt einschränkungsfrei alle OWL-Konstrukte. Es ist aber nicht mehr vollständig entscheidbar, damit ungeeignet für Inferenzsoftware.

OWL Lite ist eine syntaktisch echte Teilmenge von OWL DL und diese wiederum von OWL FULL.

OWL gibt es mittlerweile in der Version OWL 2, wobei eine gültige Ontologie in OWL auch eine gültige Ontologie in OWL 2 ist. OWL 2 hingegen definiert nur zwei Ausdrucksformen – OWL 2 DL und OWL 2 Full. Diesen liegt das selbe Konzept bzgl. der Entscheidbarkeit wie bei OWL DL und OWL Full zugrunde [OWL2].

Darüber hinaus definiert OWL 2 drei verschiedene Sprachprofile. Jedes dieser unabhängigen Sprachprofile ist eine syntaktische Teilmenge von OWL 2 DL mit dem Ziel, eine Balance zwischen Ausdrucksstärke und Rechenaufwand zu schaffen:

- OWL 2 EL
  ist für Ontologien mit einer Vielzahl `Klassen` und `Relationen` gedacht.
- OWL 2 QL
  ist für Ontologien mit einer Vielzahl `Instanzen` mit einem starken Fokus auf Anfragen (*query*) gedacht. Es ist mit einer datenbank-zentrierten Sicht entworfen.
- OWL 2 RL
  zielt auf gute Skalierung ab, ohne dabei viel Ausdrucksstärke zu verlieren. Es eignet sich für leichtgewichtige Ontologien mit vielen `Instanzen` und einer regelbasierten Implementierung.

Für jedes dieser Sprachprofile kann eine Inferenzsoftware implementiert werden, die in Polynomialzeit zur Ontologiegröße schlussfolgern kann. Sie unterscheiden sich in den Einschränkungen bzgl. der OWL 2-Konstrukte und sind unter [OWL2b] im Detail beschrieben.

OWL 2 fügt OWL weitere Elemente hinzu und definiert neue Syntaxbeschreibungen wie den funktionellen Stil. Sie fügt weitere Typen von `Relationen` hinzu, beispielsweise asymmetrische oder reflexive `Relationen`.

Andere Neuerungen machen Beschreibungen bequemer und kürzer, z. B. können mehrere `Klassen` in einer Anweisung paarweise disjunkt deklariert werden. In OWL muss jede `Klasse` einzeln disjunkt zu anderen `Klassen` deklariert werden [OWL2a].

In dieser Diplomarbeit wird OWL 2 DL verwendet, aber es wird sich weitestgehend auf den bereits von OWL gebotenen Umfang beschränkt.

Zu den Sprachkonstrukten von OWL zählen unter anderen:

`owl:Thing`	ist die ⊤-`Klasse` – jede `Klasse` ist `owl:Thing` untergeordnet (`rdfs:subClassOf`)
`owl:Nothing`	ist die ⊥-`Klasse` – die leere `Klasse` und jeder anderen `Klasse` untergeordnet
`owl:ObjectProperty`	beschreibt eine `Relation` zwischen `Instanzen`
`owl:DatatypeProperty`	verknüpft eine `Instanz` mit einem Literal
`owl:NamedIndividual`	deklariert eine nicht anonyme `Instanz`

Einen Überblick über weitere OWL 2-Konstrukte bietet [OWL2c].

Listing 3.1 zeigt das Beispiel aus Abbildung 3.4 in OWL mit einer RDF/XML-Syntax.

Listing 3.1 Beispiel OWL in RDF/XML (vgl. Abbildung 3.4)

```
<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.w3.org/2002/07/owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:kunst="http://localhost/kunst#">
    <Ontology rdf:about="http://localhost/kunst"/>
    <!-- Object Properties -->
    <ObjectProperty rdf:about="&kunst;erschafft">
        <rdfs:range rdf:resource="&kunst;Kunstwerk"/>
        <rdfs:domain rdf:resource="&kunst;Künstler"/>
    </ObjectProperty>
    <ObjectProperty rdf:about="&kunst;malt">
        <rdfs:range rdf:resource="&kunst;Gemälde"/>
        <rdfs:domain rdf:resource="&kunst;Maler"/>
        <rdfs:subPropertyOf rdf:resource="&kunst;erschafft"/>
    </ObjectProperty>
    <!-- Data properties -->
    <DatatypeProperty rdf:about="&kunst;name"/>
    <!-- Classes -->
    <Class rdf:about="&kunst;Gemälde">
        <rdfs:subClassOf rdf:resource="&kunst;Kunstwerk"/>
    </Class>
    <Class rdf:about="&kunst;Kunstwerk">
        <disjointWith rdf:resource="&kunst;Künstler"/>
    </Class>
    <Class rdf:about="&kunst;Künstler"/>
    <Class rdf:about="&kunst;Maler">
        <rdfs:subClassOf rdf:resource="&kunst;Künstler"/>
    </Class>
    <!-- Individuals -->
    <NamedIndividual rdf:about="&kunst;munch">
        <rdf:type rdf:resource="&kunst;Maler"/>
        <kunst:name>Edvard Munch</kunst:name>
        <kunst:malt rdf:resource="&kunst;schrei"/>
    </NamedIndividual>
    <NamedIndividual rdf:about="&kunst;schrei">
        <rdf:type rdf:resource="&kunst;Gemälde"/>
        <kunst:name>Der Schrei</kunst:name>
    </NamedIndividual>
</rdf:RDF>
```

3.4.4 Abfragesprache SPARQL

SPARQL ist eine Abfragesprache für RDF-Graphen und steht für „**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage“. Die Syntax ist an SQL[13] angelehnt. In der Syntax vorkommende RDF-Graphen bzw. einzelne RDF-Tripel werden in der [Turtle]-Notation angeben. Variablen, die als Platzhalter in RDF-Tripel dienen, werden mit einem „?“ gekennzeichnet. Namensräume können mit einem Präfix abgekürzt werden.

Das Listing 3.2 zeigt eine Beispielanfrage für die Kunst-Ontologie aus Listing 3.1. Die Anfrage liefert für das Variablenpaar (`?kuenstler`, `?gemaelde`) das Ergebnis („Edvard Munch“, „Der Schrei“).

Listing 3.2 Beispiel SPARQL-Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kunst: <http://localhost/kunst#>

SELECT ?kuenstler ?gemaelde
WHERE
{
    ?subject kunst:malt ?object .
    ?subject kunst:name ?kuenstler .
    ?object kunst:name ?gemaelde
}
```

Die Spezifikation von SPARQL teilt sich in mehrere Teile auf – z. B. [SPA13a] für Abfragen und [SPA13b] zur Datenmanipulation.

3.5 Representational State Transfer – REST

REST leitet sich aus der Architektur des Internets ab und wurde von Roy Thomas Fielding im Rahmen seiner Dissertation abstrahiert [Fie00].

REST besteht aus einer Reihe Randbedingungen für das Architekturdesign. Diese ermöglichen eine skalierbare und performante Architektur und zeichnet sich durch einfache Schnittstellen aus. Auf Grund seines Ursprungs im HTTP-Protokoll eignet sich REST besonders für Webservices.

[13] Structured Query Language

Zu den Randbedingungen gehören

- *Klient-Server-Architektur*

  Der Server bietet Services an und wartet auf Anfragen eines Klienten. Der Server bearbeitet eine Anfrage und sendet eine Antwort an den Klient zurück.

- *Zustandslos*

  Eine Anfrage muss alle notwendigen Informationen enthalten, die zum Bearbeiten der Anfrage notwendig sind. Der Server verwaltet keinen Zustand und eine Anfrage ist unabhängig von vorangegangen Anfragen. Die Zustandsverwaltung bleibt dem Klient überlassen.

- *Einheitliche Schnittstelle*

  Jeder Klient und Server bedient sich der selben Methoden, um miteinander kommunizieren zu können. Die Service-Implementierung eines Servers ist dadurch von einem standardisiertem Kommunikationsweg entkoppelt. Diese Schnittstelle definiert sich wie folgt:

  · *Adressierung der Ressourcen* – jede Ressource ist durch einen eindeutige URI bzw. IRI bestimmt.

  · *Repräsentation der Ressourcen* – ein Klient kann eine Ressource in unterschiedlichen Repräsentationen anfordern – dem *MIME-Type*[14] bzw. *Content-Type*. Dies kann z. B. formatiert als XML oder JSON[15] geschehen.

  · *Selbstbeschreibende Nachrichten* – jede Nachricht enthält alle notwendigen Informationen, die zu ihrer Bearbeitung notwendig ist. Bezogen auf Webservices sind dies z. B. die Operationen gegeben durch die HTTP-Methoden[16] wie `GET`, POST, `PUT`, `DELETE` und weitere.

  · *Hypermedia as the Engine of Application State* (HATEOAS) – mit Ausnahme des Einstiegspunktes navigiert ein Klient nur zu Ressourcen, die er vom Server mitgeteilt bekommen hat [Fie08].

- *Schichtensystem*

  Ein Klient weißt nicht, ob er direkt mit dem Server oder mit einer Zwischenschicht verbunden ist. Zwischenschichten können die Skalierbarkeit durch Lastenverteilung verbessern oder einen Zwischenspeicher (*Cache*) bieten.

[14] Multipurpose Internet Mail Extensions
[15] JavaScript Object Notation
[16] Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content – `https://tools.ietf.org/html/rfc7231`

- *Zwischenspeicher*

  Antworten auf Anfragen können vom Klient oder einer Zwischenschicht zwischengespeichert werden. Um Zustandsproblemen vorzubeugen, müssen Antworten als zwischenspeicherbar bzw. nicht zwischenspeicherbar deklariert sein. Dies kann auch implizit geschehen.

- *Code-On-Demand*

  Der Server übermittelt einen ausführbaren Code an den Klient, um dessen Funktionalität bei Bedarf zu erweitern. *Code-On-Demand* ist eine optionale Randbedingung.

Sind alle Randbedingungen erfüllt, ergibt das einen *RESTful*-Service. [Bet] bzw. [Fow] beschreiben das *Richardson Maturity Model* – es klassifiziert wie streng sich ein Webservice an die REST-Bedingungen hält.

Diese umgeordnete REST-Beschreibung folgt inhaltlich der von [Fie00]. In der Literatur finden sich ähnliche Beschreibungen, z. B. [RR07].

4 Verwandte Arbeiten

In diesem Kapitel werden zum Themengebiet dieser Diplomarbeit verwandte Arbeiten beschrieben. Diese beschäftigen sich mit ähnlichen Problemen oder behandeln Teilaspekte des Themengebietes.

4.1 SWE und SensorML

Das Open Geospatial Consortium (OCG)[1] als gemeinnützige Organisation für raumbezogene Informationsverarbeitung entwickelt eine Reihe offener Standards zum Zwecke der Interoperabilität. Mit der „Sensor Web Enablement"-Initiative (SWE)[2] hat es eine Reihe sensorbezogener Standards entworfen.

Als einer dieser Standards liefert SensorML ein allgemeines Standardmodell, um Sensoren und in Bezug stehende Prozesse zu beschreiben. Zu diesen Prozessen zählen die Messprozesse eines Sensors, aber auch Prozessanweisungen, um Informationen aus Beobachtungen eines Sensors ableiten zu können. Die Prozesse und Modelle hinter SensorML sind mit UML[3] modelliert. Ein Prozess definiert sich aus seinen Eingaben, Ausgaben, Parametern, Methoden sowie Metadaten. Er erfasst ein beobachtbares Phänomen und wandelt dieses in Daten um [Sen14].

Als SWE-Standard steht SensorML in direktem Zusammenhang mit weiteren Standards, insbesondere *Observations & Measurements*, *Sensor Observation Service* und *Sensor Planning Service*. Im Zusammenspiel ermöglichen diese das Veröffentlichen, Auffinden und Einbinden von Sensor-bezogenen Daten. Für *Observations & Measurements* hat z. B. Simon Cox eine Ontologie in OWL implementiert [Cox13].

[1] Open Geospatial Consortium – `http://www.opengeospatial.org/`
[2] Sensor Web Enablement – `http://www.opengeospatial.org/ogc/markets-technologies/swe`
[3] Unified Modeling Language – `http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/`

Formal besteht SensorML aus einer Reihe XML Schematas. Diese hängen von diversen weiteren SWE-Standards ab und bauen auf diesen wie z. B. SWE Common Data Model[4], GML 3.2[5] sowie Normen zu Geoinformationen und Geodaten[6] auf.

Durch GML modellierte Geoinformationen und Geodaten enthalten u.a. präzise Ortsangaben und sind Bestandteile der durch SensorML definierten Konzepte.

> *Beispiel Ozeanvermessung:*
> *Die Messungen im Ozean beruhen auf den Messwerten vieler Messbojen. Setzt man die Messwerte und Positionen der Messbojen in Bezug zueinander, erhält man ein Bild der aktuellen Bedingungen im Ozean. So ergibt sich die Wellenhöhe aus Messwerten von Wellental und Wellenberg verschiedener benachbarter Messbojen. Auch lassen sich Strömungen erst aus dem Zusammenspiel vieler Messbojen detailliert bestimmen.*

SensorML ist eine vorwiegend syntaktische Beschreibung mit einer unterliegenden semantischen Bedeutung. Die Spezifikation referenziert Beispielontologien [Sen14, S. 71], die sich allerdings auf ein zusammenhangsloses Begriffswörterbuch beschränken bzw. nicht mehr verfügbar sind.

4.2 OntoSensor – ontologische Übersetzung von SensorML

Russomanno et al. beschreiben in ihrem Artikel die Konstruktion einer Sensor Ontologie namens *OntoSensor* in OWL [RKT05]. Dazu gehört auch eine in Prolog implementierte Inferenzsoftware. Die Konstruktion der Ontologie basiert dabei auf SensorML, *IEEE Suggested Upper Merged Ontology* (SUMO) und ISO 19115.

Im ersten Teil ihrer Arbeit beschreiben sie die Unzulänglichkeiten, die SensorML mit sich bringt. Sie verweisen darauf, dass XML nur ein syntaktisches Konstrukt ist. Auch bemängeln sie, dass SensorML keine formale Definitionen seiner Klassen und Relationen liefert. Weiterhin stellen sie fest, dass SensorML keine logische bzw. axiomatisch fundierte Grundlage für eine gemeinsame Konzeptualisierung nach der Ontologiedefinition durch Gruber [Gru93] liefert. Positiv bewerten sie, dass SensorML ein generisches Datenmodell für Informationen über Sensoren ist, und dass SensorML eine wohl-fundierte Grundlage zur Entwicklung einer Ontologie bietet.

[4] SWE Common Data Model Encoding Standard – `http://www.opengeospatial.org/standards/swecommon`
[5] Geography Markup Language – `http://www.opengeospatial.org/standards/gml`
[6] ISO 19103, ISO 19108, ISO 19111, ISO 19115 und weitere

Zur Kontruktion der Ontologie identifizieren sie als erstes die Konzepte hinter SensorML, um sie anschließend in OWL zu implementieren. Als Werkzeug verwenden sie hierfür Protégé[7], einen Ontologie Editor. Einige der SensorML-Konzepte beruhen auf den durch GML modellierten Ortsangaben, um Sensordaten lokal einzuordnen oder in Bezug zu anderen Sensoren zu setzen. *OntoSensor* wiederverwendet hierzu eine Ontologie der GML-Konzepte, beruhend auf der Arbeit eines anderen Teams. Die Quelle dieser Arbeit ist nicht mehr auffindbar.

Am Ende des Artikels präsentieren sie noch eine Beispielinstanz eines Sensors und das Resultat ihrer eigenen Inferenzsoftware.

4.3 Sensoren und deren Datenströme als verknüpfte und frei verfügbare Daten

Danh Le-Phuoc und Manfred Hauswirth beschreiben in ihrem Artikel eine Plattform namens SensorMashup, um Sensordaten allgemein über das Internet verfügbar zu machen und so der Vision eines Internet der Dinge näher zu kommen [PH09]. Sie folgen dabei dem Konzept von *linked (open) data*[8] als Teil des *Semantic Webs*.

Eine Verknüpfung aus Sensoren, deren Datenströme und Metadaten nennen sie *Sensor Mashup*. Die Plattform ermöglicht es solche *Mashups* aufzufinden und neue Sensoren sowie Datenströme zu veröffentlichen. Mit einem *Composer* können neue *Mashups* erzeugt werden.

Die Plattform bietet einen Webservice als REST-Schnittstelle an, um definierte *Mashups* abzurufen. Sie basiert auf den SWE[9]-Standard, wobei die Webservice-Beschreibung in WSDL[10] mit RDFa[11] um semantisches Wissen ergänzt wird. Für die Sensoren wurde eine Ontologie mit OWL-DL basierend auf den Konzepten von SensorML entworfen. Ihr Entwurf ist dabei vergleichbar zu dem Ontologieentwurf von [RKT05] (vgl. Kapitel 4.2). Weiterhin bietet sie noch einen SPARQL-Endpunkt für komplexe *Mashups* an.

Aus Effizienzsgründen verwalten sie die Mashups als virtuelle RDF-Graphen in einem internen Speicher. Der virtuelle RDF-Graph wird in einer komprimierten Repräsentation vorgehalten. Dieser Ansatz ist leistungsfähiger, dennoch sind semantische Anfragen mit SPARQL weiterhin möglich. Die Plattform bleibt auf diesem Wege kompatibel zu anderen ontologischen Systemen.

[7] Protégé Ontologie Editor – `http://protege.stanford.edu/`
[8] Linked Data – `http://www.w3.org/DesignIssues/LinkedData.html`
[9] Sensor Web Enablement
[10] Web Services Description Language
[11] RDF in Attributes – `http://www.w3.org/TR/rdfa-primer/`

Ein Schwerpunkt dieses Projektes liegt auf der Verwaltung und der Speicherung der Datenströme mit einem „*data stream management system*“ (DSMS). Im Gegensatz zur Sensorintegration dieser Diplomarbeit liegt ihr Fokus darin, Sensoren und ihre Datenströme im Internet in einem offenen System verfügbar zu machen.

4.4 OpenTOSCA

OpenTOSCA[12] ist ein quelloffenes Ökosystem der Universität Stuttgart für den „OASIS[13] Topology and Orchestration Specification for Cloud Applications“-Standard [TOSCA].

TOSCA ist ein Standard zur portablen Beschreibung von Cloud-Anwendungen. Eine solche Beschreibung ermöglicht eine automatisierte Provisionierung cloud-basierten Anwendung.

Er enthält eine Anwendungstopologie und deren Orchestration in XML. Unter Orchestration versteht man die Koordination und das Management von Anwendungen, Middleware und Computersystemen. Sie wird als Plan hinterlegt und beschreibt die Schritte, die vollzogen werden müssen, um eine Anwendung in einer Cloud-Umgebung zu installieren. Die Anwendungstopologie (*Topology Template*) beschreibt die Anwendung und ihre Abhängigkeiten zu ihrer Umgebung, auf der sie laufen soll. Beispielsweise läuft eine PHP-Anwendung auf einem Apache-Webserver mit einem PHP-Interpreter auf einem Linux-Betriebssystem in einer Cloud-Instanz.

Das OpenTOSCA-Ökosystem besteht aus drei Komponenten:

- winery – zur graphisch unterstützten Modellierung. Das Modell wird in einem *Cloud Service Archive* (CSAR) zusammengefasst.
- OpenTOSCA Container – Laufzeitumgebung, die die im CSAR enthaltende Beschreibung ausführt [BBH+13].
- Vinothek – webbasierter Service, mit dem sich CSAR verwalten und installieren lassen.

OpenTOSCA beschäftigt sich nicht im Speziellen mit Sensoren oder semantischen Beziehungen zwischen Objekten und Sensoren, sondern mit dem Prozess, eine Software auf einem anderen System zu installieren und zu starten. Die Sensoradapter-Provisionierung dieser Diplomarbeit ist ein vergleichbarer Prozess.

[12] `http://www.opentosca.org`
[13] Organization for the Advancement of Structured Information Standards – `https://www.oasis-open.org/`

5 Konzept

In diesem Kapitel wird das Konzept der Sensorintegration erläutert. Vorab wird die Gesamtstruktur beschrieben, bevor auf die einzelnen Komponenten eingegangen wird. Abschließend werden noch Optimierungen für die Sensorintegration, eine autonome Sensorintegration und Sicherheitsaspekte konzeptioniert.

5.1 Gesamtstruktur

Es wurde eine Sensorintegration (`SeInt`) geschaffen, mittels welcher sich Objekte und ihre untergeordneten Sensoren automatisiert und ad-hoc in `SitRS` (vgl. Kapitel 2.2) integrieren lassen. Dabei werden die `SitRS`-Komponenten *Resource Management Platform* (`RMP`) und *Sensor Registry* integriert.

Abbildung 5.1 zeigt das Architekturkonzept der Sensorintegration und ihre Anbindung an die *Resource Management Platform* (`RMP`) sowie die *Sensor Registry* von `SitRS`. `SitRS` – gestrichelt dargestellt – ist nicht Bestandteil dieser Diplomarbeit.

Die Sensorintegration besteht aus zwei Kernkomponenten:

- ein „Zentraler Service“ (ZS), der eigene Schnittstellen bietet bzw. die Schnittstellen von `SitRS` bedient
- eine Ontologie, die als Wissensbasis zu Objekten und Sensoren dient. Sie wird über den ZS angesprochen.

Weiterhin gibt es noch ein Adapter-Repository. Es beinhaltet Adapter zu Objekt-Sensor-Paaren.

Der ZS wird nach außen hin über einen Klienten gesteuert und vermittelt dessen Anfragen an die Ontologie. Die Ergebnisse liefert der ZS an den Klienten zurück.

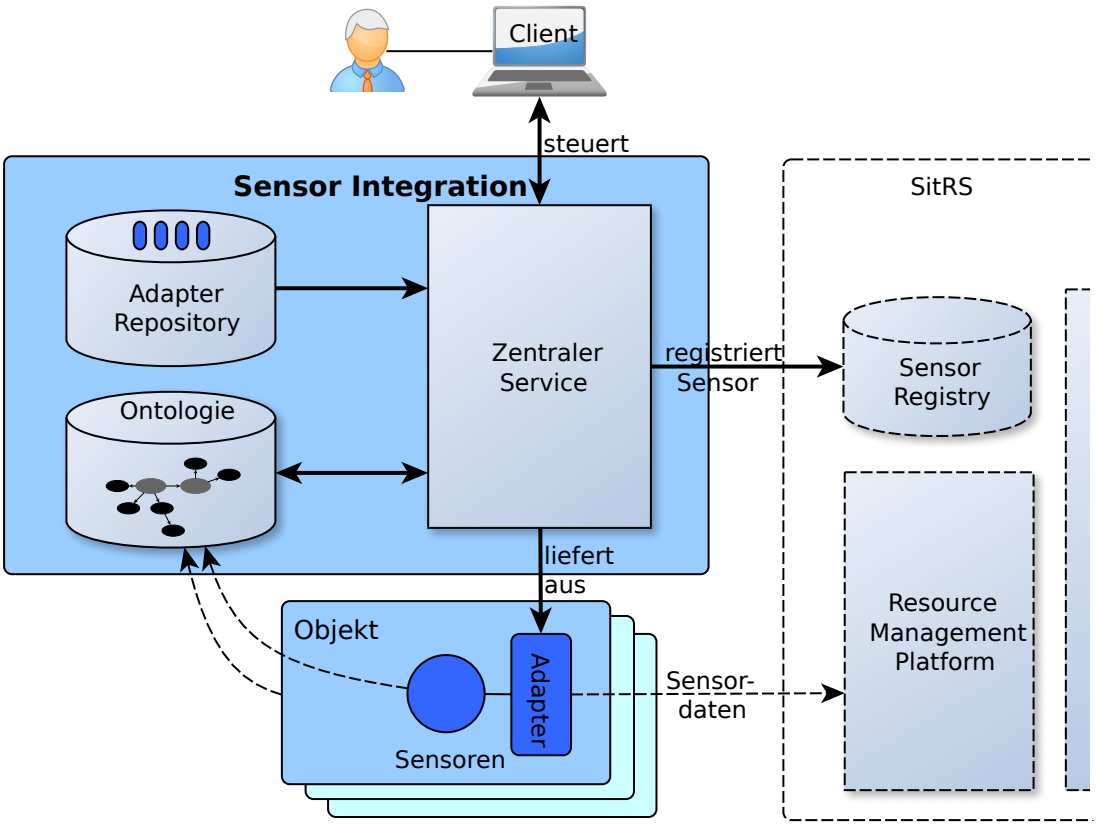

Abbildung 5.1: Konzept Übersicht

Das Sequenzdiagramm in Abbildung 5.2 zeigt den Ablauf einer Sensorintegration:

1. der Klient fordert die Integration eines Objektes und dessen Sensoren an
2. der Zentrale Service (ZS) nimmt die Anfrage des Klienten entgegen und verarbeitet sie
3. der ZS fragt die Ontologie nach den Informationen über das Objekt und die Sensoren
4. die Ontologie sendet die Informationen über das Objekt und die Sensoren an den ZS
5. der ZS registriert mittels den Informationen aus der Ontologie das Objekt und die Sensoren an der *Sensor Registry* von SitRS und wartet auf das Resultat
6. der ZS provisioniert den Adapter für die Sensoren auf dem Objekt
7. der Klient bekommt die Integration vom ZS bestätigt

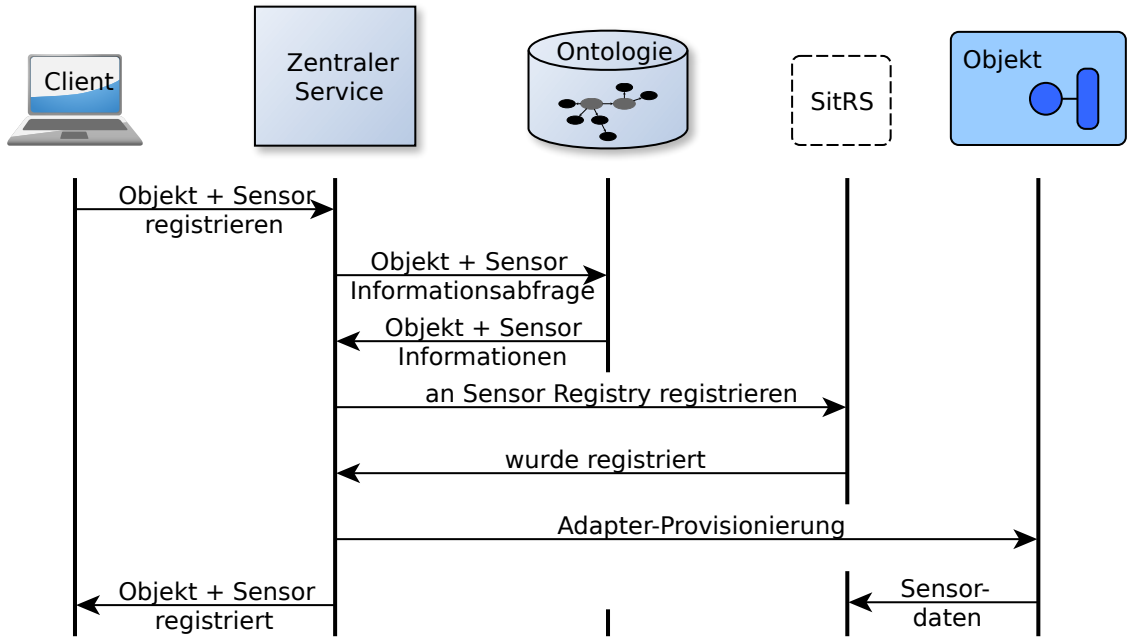

Abbildung 5.2: Konzept Sequenzdiagramm einer Sensorintegration

5.2 Zentraler Service – ZS

Am Prozess einer Sensorintegration sind viele Komponenten beteiligt. Jede dieser Komponenten hat ihre eigenen Schnittstellen. Der ZS kommuniziert mit all diesen Schnittstellen bzw. steuert die Komponenten über deren Schnittstellen.

Der „ZS“ vereinigt diverse Schnittstellen, die sich wie folgt zusammensetzen:

- eine interne Schnittstelle zum Informationsaustausch mit der Ontologie. Der ZS ruft Informationen zu Objekten und Sensoren ab bzw. legt diese an.
- eine eigene äußere Schnittstelle für einen Klienten zur allgemeinen Steuerung. Sie ermöglicht es dem Klient die Sensorintegration zu steuern. Außerdem fungiert der ZS als Vermittler zwischen Klient und Ontologie.
- eine Schnittstelle zu Objekten mit ihren Sensoren. Es wird ein Adapter provisioniert und somit einem Objekt ermöglicht, seine Sensordaten an die RPM zu senden.
- Schnittstellen zu den Komponenten von SitRS. Sie verbindet sich mit der *Sensor Registry*, um dort Objekte und Sensoren zu registrieren. Die Registrierung macht Objekte und Sensoren für SitRS verfügbar. Schlägt die Registrierung fehl, wird die Sensorintegration abgebrochen und kein Adapter provisioniert.

Die Schnittstellen zu den Komponenten von SitRS sind vorgegeben. Die weiteren Schnittstellen werden in den folgenden Kapiteln präzisiert.

5.3 Ontologie als Wissensbasis

Die andere zentrale Komponente der Sensorintegration ist die Ontologie als Wissensbasis. In ihr werden Informationen über die Objekte und Sensoren abgelegt.

Die Struktur der Ontologie setzt sich aus zwei Teilen zusammen:

- zum einen das semantische Wissen in Form von Klassen und Relation
- zum anderen konkrete Instanzen von Orten, Objekten und Sensoren

Die folgenden Abbildungen der Ontologien zeigen zur besseren Übersicht Teilausschnitte.

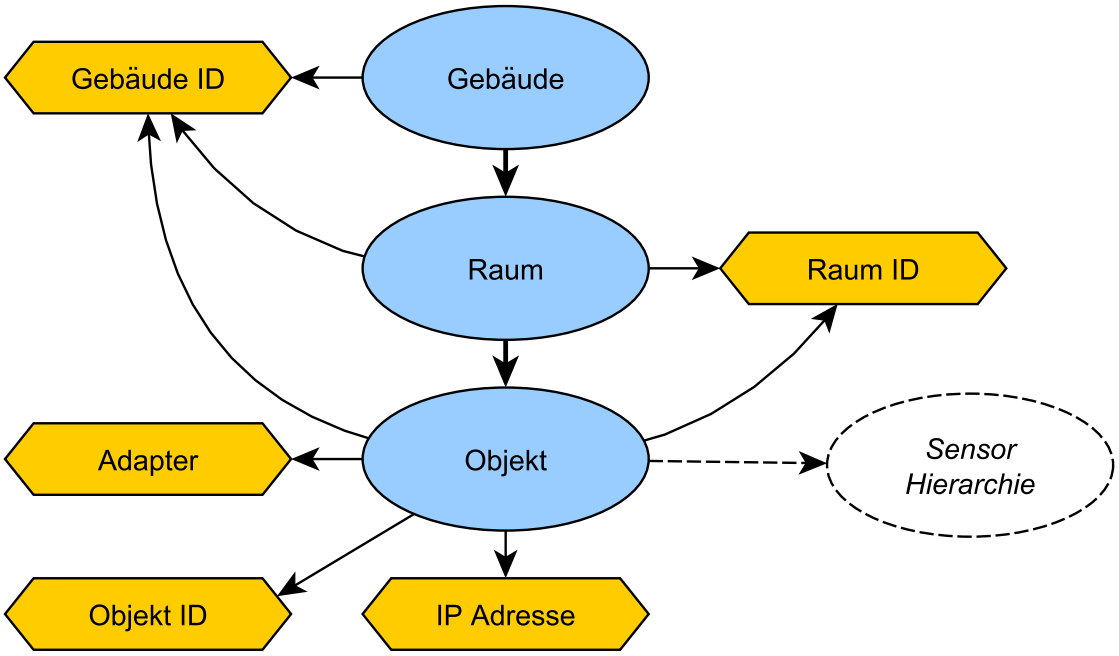

Abbildung 5.3: Konzept Ontologie des Objektes

Abbildung 5.3 zeigt das Konzept der Ontologie für das Objekt. Die Sensor-Ontologie ist in dieser Abbildung nur angedeutet. Die Klassen sind in zwei Kategorien eingeteilt. Hauptklassen als vorrangige Hierarchie sind in bläulichen Ellipsen gehalten. Instanzen dieser Hauptklassen repräsentieren Gegenstände aus der realen Welt, welche durch Nebenklassen

näher beschrieben werden. Die Nebenklassen sind als gelbliche Hexagone eingezeichnet. Sie stehen in Bezug zu Hauptklassen und versehen eine `Instanz` einer Hauptklasse mit beschreibenden Informationen.

Die `Instanz` eines Objekt erhält auf diesem Wege Informationen zu Objekteigenschaften wie die Adresse, zugehörige Adapter und Sensoren sowie weitere Informationen.

In Hexagonen sind die Nebenklassen eingezeichnet, die die Informationen zu einer `Instanz` aus der vorrangigen Hierarchie tragen. Eine `Instanz` eines Objektes wird über die Informationen der Nebenklassen detailliert beschrieben.

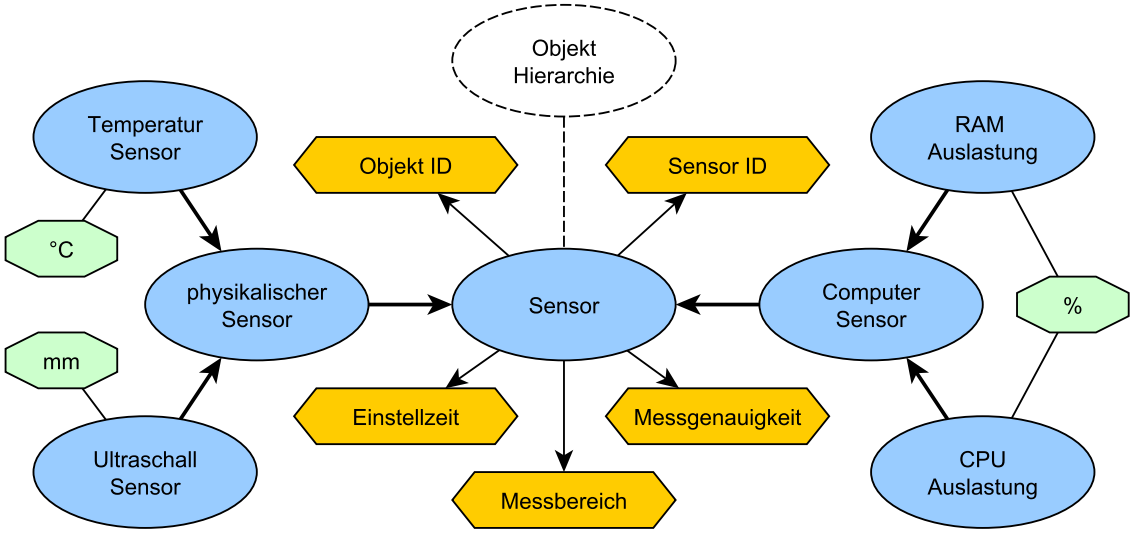

Abbildung 5.4: Konzept Ontologie der Sensoren

Die Hierarchie für Sensoren (Abbildung 5.4) ist analog konzipiert. Sensoren werden in ihre Unterarten aufgegliedert, z. B. physikalische Sensoren oder Sensoren, die Rechnereigenschaften wie die CPU-Auslastung widerspiegeln. Andere Hierarchien sind denkbar. So können Sensoren nach Messprinzip in `Klassen` wie mechanische, optische, elektrische und weitere Messmethoden geordnet werden.

Jeder Sensor-`Klasse` sind bestimmte Parameter vorgegeben, die sich über Axiome – grüne Oktogone – beschreiben lassen. Der `Instanz` eines Temperatursensors ist auf diesem Wege vorgegeben, dass „°C“ ihre physikalische Einheit ist.

Zur Vervollständigung der Ontologie werden diverse `Relationen` hinzugefügt (vgl. Abbildung 5.5). Sie beschreiben die Beziehungen zwischen den `Instanzen` der jeweiligen `Klassen`. Die `Relation` „`besitzt(Objekt, Sensor)`“ erklärt welche Sensoren zu einem Objekt gehören. Die dazugehörige inverse `Relation` lautet „`gehört_zu(Objekt, Sensor)`“.

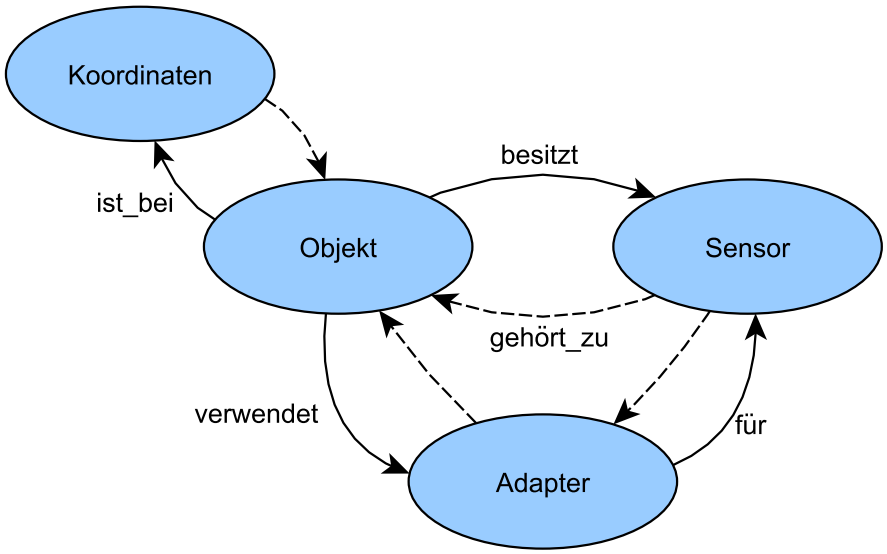

Abbildung 5.5: Konzept Ontologie Relationen

Zu den weiteren `Relationen` gehören:

- `ist_bei(Objekt, Koordinaten)` zur Geolokalisierung eines Objektes
- `verwendet(Objekt, Adapter)`, um Objekt und Adapter miteinander zu verknüpfen
- `für(Adapter, Sensor)`, um anzuzeigen welchen Sensor ein Adapter bedient

Zu jeder `Relation` kann eine inverse `Relation` definiert werden. Mit diesen können `Instanzen` aus verschiedenen Blickwinkeln definiert werden. Man kann z. B. von einer Objekt-`Instanz` ausgehen und alle ihre Sensoren definieren oder man definiert umgekehrt Sensor-`Instanzen` und erklärt, zu welchen Objekten sie gehören.

Die Ontologie wird in einem Speicher – vergleichbar zu einer Datenbank – verwaltet. In diesem Speicher werden die Informationen und die Struktur als Daten abgelegt. Darüber hinaus wird auch die semantische Bedeutung der Daten als formale Beschreibung in Form von Regeln über Daten sowie Beziehungen zwischen Daten gespeichert. Über diese semantische Bedeutung der Daten ist es einer Inferenzsoftware möglich, die Daten auf ihre Konsistenz und Widerspruchsfreiheit hin zu überprüfen. Auch kann fehlendes Wissen aus den vorhandenen Daten rückgeschlossen bzw. ergänzt werden.

Eine solche Inferenzsoftware gehört zu der Schnittstelle der Ontologie. Die Schnittstelle ermöglicht es `Instanzen` der Ontologie hinzuzufügen bzw. abzufragen. Auch kann Wissen in Form von `Klassen` und `Relationen` geändert oder ergänzt werden. Die Methoden der Schnittstelle sind vergleichbar zu den klassischen Datenbankoperationen (CRUD[1]).

Die Schnittstelle kann `Klassen`, `Relationen` und `Instanzen`

- neu erzeugen,
- vorhandene lesen,
- vorhandene aktualisieren und
- vorhandene entfernen.

Sie wird vom Zentralen Service gesteuert und bedient.

5.4 Klient

Um den zentralen Service abseits einer programmatischen Schnittstelle für einen Benutzer steuerbar und bedienbar zu gestalten, benötigt es einen Klient.

Der Klient ermöglicht es dem Benutzer

- Objekte und deren Sensoren zur Sensorintegration auszuwählen.

  Der Klient bietet dem Benutzer in einem ersten Schritt an, ein Objekt auszuwählen. Im nächsten Schritt bekommt er die zu diesem Objekt gehörenden Sensoren angezeigt. Er wählt die gewünschten Sensoren aus und startet die Sensorintegration. Der Klient kommuniziert dem `ZS` die Auswahl und wartet darauf, dass der `ZS` die Sensorintegration mit all ihren Schritten vollzogen hat.

- Informationen zu Objekten und Sensoren abzufragen.

  Der Klient zeigt dem Benutzer zu gewünschten Objekten bzw. Sensoren die in der Ontologie hinterlegten Informationen an. Das Auswahlverfahren gleicht dem der Sensorintegration, er kann aber auch Informationen über Sensoren unabhängig vom Objekt abfragen.

[1] *create, read, update, delete*

- Objekte sowie Sensoren der Ontologie hinzufügen.

  Der Benutzer bekommt über den Klient eine Eingabemaske angeboten. Über diese Maske kann er Objekte zzgl. den notwendigen Informationen zu dem Objekt anlegen. Weiterhin kann er dem Objekt Sensoren hinzufügen. Ein so geleiteter Prozess sorgt dafür, dass die Daten in der Ontologie konsistent und widerspruchsfrei bleiben.

5.5 Sensordatenanbindung über Adapter

Im Zuge der Sensorintegration werden Objekte mit ihren Sensoren registriert. Das Objekt hat nun zwei Aufgaben zu erfüllen:

- es verbindet sich mit dem registrierten Sensor und liest dessen Sensordaten aus.
- es sendet die erfassten Sensordaten an die *Resource Management Platform* (RMP).

Ein auf dem Objekt ausgeführter Adapter wird diese Aufgaben übernehmen. Das Objekt-Sensor-Paar identifiziert den Adapter eindeutig.

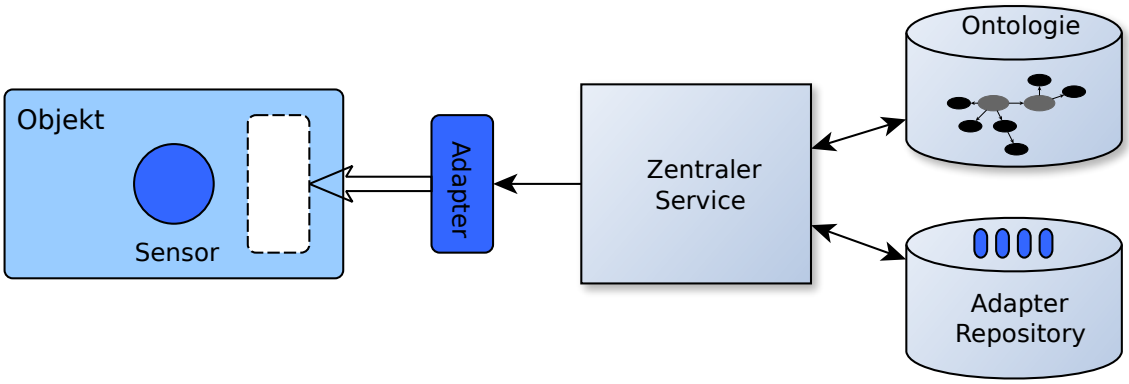

Abbildung 5.6: Konzept Adapter Provisionierung

Die Vielzahl verschiedener Sensoren bedingen ebenso viele Methoden wie ein Sensor ausgelesen werden kann. Deswegen wird für jeden konstruktionsgleichen Sensor ein individueller Adapter benötigt. Je nach Art des Sensors kann es genügen, den Messwert über die Sensorschnittstelle direkt zu erfassen. Im Allgemeinen wird man den Messwert aber erst noch einer Transformation unterziehen müssen. Diese kann z. B. einer physikalischen Vorschrift folgen.

Weiterhin muss der Adapter befähigt sein, die erfassten Sensordaten an die RMP zu senden. Dazu benötigt der Adapter die Information auf welche Weise die RMP anzusprechen ist. Die Methode der Schnittstelle zur RMP ist für alle Adapter identisch.

Jeder Adapter repräsentiert ein Objekt-Sensor-Paar. In einem *Adapter Repository* (AR) werden die Adapter als Vorlage vorgehalten. Die Vorlage eines Adapters ist in einer parametrisierten Form hinterlegt. Zu den Parametern gehört der URI der RMP sowie eine Objekt-Id und eine Sensor-Id. Werden die Parameter der Vorlage zugeführt, kann der Adapter auf dem Objekt ausgeführt werden.

Im Prozess der Sensorintegration erhält der ZS Informationen über die Art des Objektes und des Sensors und kann damit den Adapter bestimmen. Der ZS konfiguriert nun den Adapter mit allen notwendigen Parametern.

Als letzter Schritt wird der Adapter auf das Objekt übertragen und dort ausgeführt.

OpenTosca bietet einen umfassenden Ansatz um ein Objekt gänzlich zu konfigurieren, Systemumgebung für den Adapter schaffen und Adapter ausliefern [BBH+13].

5.6 Optimierung

Eine Situation in SitRS kann unter Umständen durch mehrere Sensoren gleichermaßen bestimmt werden. So kann es alternative Sensoren für ein Messdatum geben, wobei sich die alternativen Sensoren hinsichtlich ihrer Eigenschaften unterscheiden.

> *Beispiel – Füllstand in einem Flüssigkeitstank:*
> *Der Füllstand kann über verschiedene physikalische Methoden bestimmt werden. Über den Differenzdruck, Gesamtgewicht, elektrische Leitfähigkeit der Flüssigkeit, Ultraschallmessung und diverse weitere Messmethoden. Jede Messmethode hat ihre Vor- und Nachteile bzgl. ihrer Messgenauigkeit und anderer Messeigenschaften.*

Ein Sensor kann gegenüber einem anderen Sensor für dieselbe Situation beispielsweise einen höheren Energiebedarf, eine geringere Einstelldauer oder eine höhere Messgenauigkeit besitzen. Es ergeben sich unterschiedliche Optimierungskriterien nach denen eine Situationserkennung erfolgen kann:

- Energetische Optimierung
  Sensoren haben einen unterschiedlichen Energiebedarf. Hier gilt es den Sensor mit dem geringsten Energiebedarf auszuwählen.
- Zeitliche Optimierung
  Sensoren erfassen ihren Messwert mit unterschiedlicher Einstelldauer. Es gilt den Sensor mit der geringsten Einstelldauer auszuwählen.

- Präzisionsoptimierung
  Sensoren erfassen ihren Messwert mit unterschiedlicher Messgenauigkeit. Es gilt den Sensor mit der höchsten Messgenauigkeit auszuwählen.
- Kostenoptimierung
  Die Kostenoptimierung kann aus verschiedenen Blickwinkeln betrachtet werden. Energiekosten, Anschaffungskosten in Relation zur Lebensdauer. Es gilt den Sensor mit den geringsten Kosten auszuwählen.
- Ressourcen Optimierung
  Damit ist der Umstand gemeint, dass es für eine neue Situationserkennung zwei äquivalente Sensoren gibt. Einer der Sensoren wird bereits für eine andere Situationserkennung herangezogen. Statt neue Ressource im Gesamtsystem zu reservieren, können die Messdaten des sich bereits im Einsatz befindlichen Sensors für diese neue Situationserkennung wiederverwendet werden.

Die verschiedenen Optimierungskriterien können sich gegenseitig ausschließen, was sich allgemein wie folgt darstellt:

- *Sensor A* misst mit hoher Messgenauigkeit, benötigt dafür mehr Energie und hat eine längere Einstelldauer als *Sensor B*.
- *Sensor B* misst mit geringer Messgenauigkeit, dafür bei geringerem Energiebedarf und einer kürzeren Einstelldauer als *Sensor A*.

Sensor A und *Sensor B* schließen dabei eine zeitgleiche Optimierung nach Messgenauigkeit und Energiebedarf aus.

Algorithmisch gilt es dabei zu jeder Optimierung die in Frage kommenden Sensoren zu bestimmen und entsprechend der Art der Optimierung auszuwählen. Dazu müssen entweder zueinander alternative Sensoren in der Ontologie vermerkt werden oder über die Geolokalisierung und den Sensortyp zueinander alternative Sensoren ermittelt werden, um entsprechend der Art der Optimierung auswählen zu können. Für eine Ressourcenoptimierung sollte verwaltet werden, welche der alternativen Sensoren bereits im Einsatz sind.

Darüber hinaus bedarf es einer stetigen Aktualisierung sobald sich eine Sensorenkonfiguration ändert. Wird ein Sensor, der durch eine Optimierung ausgewählt wurde, entfernt, benötigt die dahinterliegende Situationserkennung einen entsprechenden alternativen Sensor, um weiterhin ihren Dienst verrichten zu können. Wird ein neuer Sensor hinzugefügt, kann dieser in genutzten Optimierungen den Vorzug gegenüber vorherigen Sensoren erhalten.

5.7 Autonome Sensorintegration

Rein konzeptionell wird eine autonome Sensorintegration beschrieben. Damit ist der Eintritt eines intelligenten Objektes (*SMART Object*) in eine intelligente Umgebung (*SMART Environment*) gemeint.

Prinzipiell kann der Eintritt eines intelligenten Objektes in die intelligente Umgebung an zwei Stellen registriert und behandelt werden:

- Entweder auf der Seite des intelligenten Objektes, welches selbstständig seine Ankunft im Netzwerk feststellt,
- oder auf der Seite der intelligenten Umgebung, die die Ankunft einen neuen Objektes bemerkt.

Es stellt sich das Problem, wie beide Seiten Kenntnis voneinander erlangen können. Insbesondere, wenn man davon ausgeht, dass weder das Objekt noch die intelligente Umgebung Kenntnis von der Adresse des jeweils anderen besitzt. Damit beide Seiten sich finden, um miteinander kommunizieren zu können, ist ein standardisierter und wohldefinierter Prozess notwendig. Dazu gehört, dass die Nachricht des Eintrittes als solche erkannt werden kann.

Im Zusammenhang mit der Sensordatenanbindung ist das vorangegangene Konzept der Adapterauslieferung nicht mehr ausreichend und muss durch eine universellere Lösung ersetzt werden. Dazu sind grundsätzlich zwei Wege denkbar:

- Das Objekt stellt seinen eigenen Adapter bereit. Bei Eintritt in das Netzwerk teilt die intelligente Umgebung dem Objekt die Datensenke für die Sensordaten der *Resource Management Platform* (`RMP`) mit. Der Adapter muss vom Objekt selbst entsprechend konfiguriert werden.

  Dieser Weg gibt einen generellen Standard für Adapter vor. Die Methode wie ein Adapter seine Daten an die `RMP` sendet ist durch die Architektur vorgegeben und jeder Adapter muss dieser Methode folgen. Eine nachträgliche und signifikante Änderung der `RMP` würde sämtliche Adapter obsolet gestalten und ein jedes Objekt bräuchte ein eigenes Update seiner Adapter. Anderseits ist die Sensorintegration flexibler, da sie weder Adapter vorhalten muss, noch detaillierte Kenntnisse über technischen Details der Sensoren eines Objektes benötigt.

- Das Objekt teilt der intelligenten Umgebung alle seine Parameter für eine Provisionierung eines Adapter mit. Die intelligente Umgebung generiert aus diesen Parametern einen auf dem Objekt lauffähigen Adapter, liefert diesen aus und startet ihn auf dem Objekt.

  Dieser Weg ist wesentlich komplexer. Die Sensorintegration benötigt umfangreiches Wissen über diverse Objekte, z.B. in welcher Form ein Adapter vorliegen muss. Mit einem Raspberry Pi des vorherigen Kapitels 5.5 bestehen Adapter aus Python-Skripten, denkbar ist aber auch ein Objekt mit einem programmierbaren Mikrocontroller, dem ein Adapter als Binärcode übermittelt werden muss, z.B. per I²C [NXP14]. Bei signifikanten Änderungen an der `RMP` hingegen können sämtliche Adapter zentral angepasst werden.

Für die Eintrittserkennung des Objektes werden in den folgenden Unterkapiteln mehrere Ansätze und deren Probleme erläutert. Zur besseren Übersicht werden die Ansätze mit klassischem IP[2] und beschränkt auf ein lokales Netzwerk erläutert. Auch besitzen die Objekte in den folgenden Ansätzen nur einen Sensor und beherbergen ihren eigenen Adapter.

5.7.1 Eintrittsankündigung

DHCP[3] zur automatischen Netzwerkkonfiguration funktioniert über Broadcasts auf festgelegten UDP-Ports. Ein Client sendet an alle im Netzwerk eine Anfrage, ein DHCP-Server fängt diese auf und beantwortet sie [RFC2131]. Eine vergleichbare Technik lässt sich zur Eintrittsankündigung verwenden.

Über die Broadcast-Zieladresse `255.255.255.255` wird der Eintritt im lokalen Netzwerk kommuniziert. Die intelligente Umgebung reagiert auf solche Broadcasts und antwortet darauf mit seiner IP-Adresse.

Das Objekt nimmt eine aktive Rolle ein. Es muss feststellen, wann es in ein Netzwerk eintritt. Das auslösendes Ereignis kann der Erhalt einer Netzwerkkonfiguration per DHCP darstellen. Das Objekt überwacht aktiv, wann es ein solche Netzwerkkonfiguration erhält, um daraufhin seinen Eintritt per Broadcast anzukündigen.

Abbildung 5.7 zeigt ein Sequenzdiagramm mit dem Ablauf der Kommunikation zwischen intelligentem Objekt und der intelligenten Umgebung. Das Objekt ist dem Netzwerk bereits beigetreten und hat seinen eigenen Adapter, der nur noch konfiguriert werden muss.

Um die Anzahl der Nachrichten zu reduzieren, könnte man den Broadcast der Eintrittsankündigung bereits um die Information über die Sensortypen erweitern. Die Größe eines UDP-Pakets ist durch IPv4 limitiert und ein einzelnes Paket reicht eventuell nicht aus, um

[2] *Internet Protocol Version 4* – `https://tools.ietf.org/html/rfc791`
[3] *Dynamic Host Configuration Protocol*

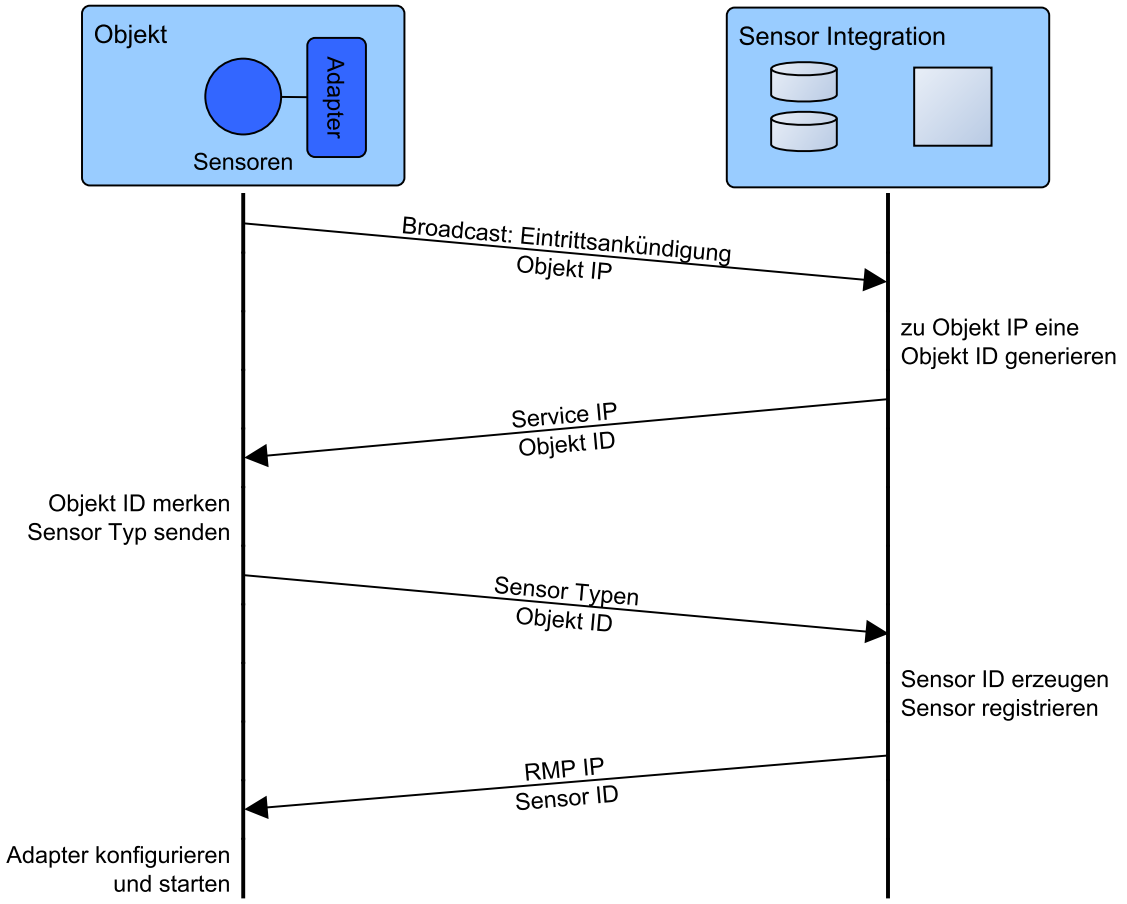

Abbildung 5.7: Sequenzdiagramm einer autonomen Sensorintegration per Eintrittsankündigung

alle Information unterzubringen – insbesondere für Objekte mit einer Vielzahl Sensoren. Da die Broadcast-Nachricht an alle Geräte im Netzwerk gesendet wird, sollten solche Pakete möglichst klein gehalten werden, um die Netzlast gering zu halten.

5.7.2 Eintrittsaufforderung

Eine andere Möglichkeit besteht darin, dass die intelligente Umgebung in periodischen Abständen eine Eintrittsaufforderungsnachricht in das Netzwerk absetzt. Neue Objekte fangen diese Eintrittsaufforderung auf und melden sich an die intelligenten Umgebung zurück.

Diese Variante ändert das Sequenzdiagramm wie in Abbildung 5.8 zu sehen ist. Wenn es eine große Anzahl neuer und nicht registrierter Objekte im Netzwerk gibt, wird eine Eintrittsaufforderung eine entsprechende Anzahl zeitgleicher Antworten auslösen. Die intelligente Umgebung muss in der Spitzenlast alle Antworten zuverlässig verarbeiten und beantworten können. Darüber hinaus muss das Netzwerk selbst solche Spitzen verkraften können, ohne dass Nachrichten verloren gehen.

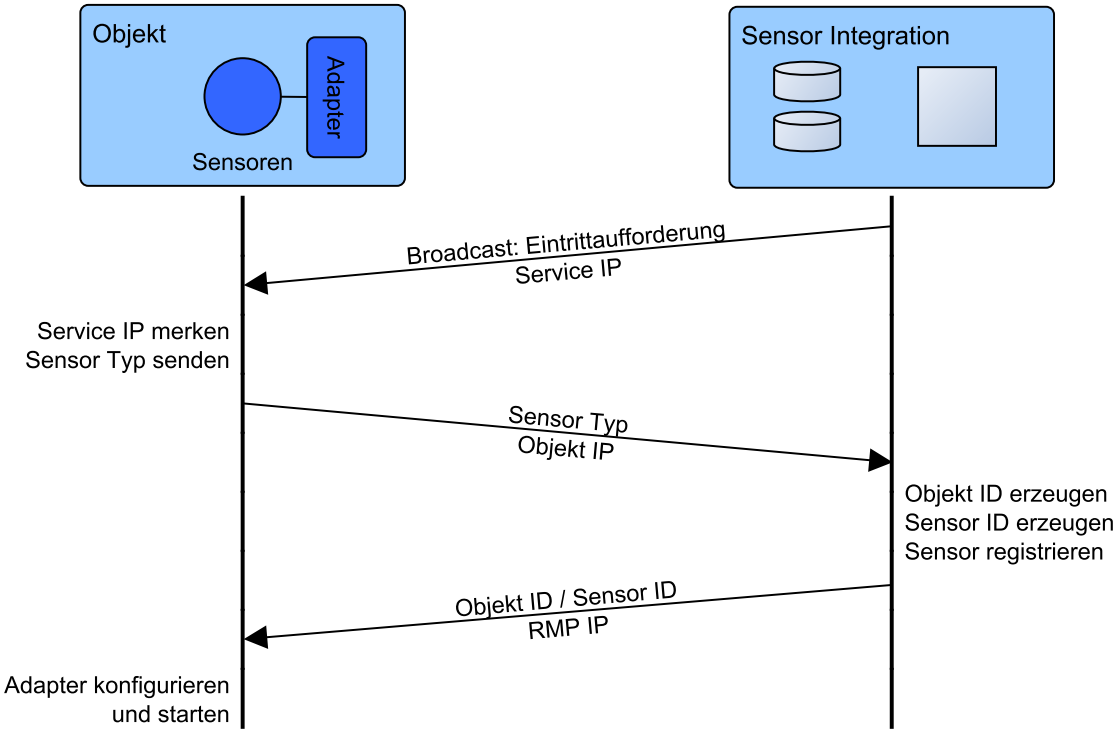

Abbildung 5.8: Sequenzdiagramm einer autonomen Sensorintegration per Eintrittsaufforderung

5.7.3 Weitere Probleme einer autonomen Sensorintegration

Autonome intelligente Objekte müssen komplexer gestaltet sein. Das macht ihre Entwicklung aufwendiger, zeitintensiver und damit auch teurer. Davon abgesehen ergeben sich noch weitere Probleme.

Zur erfolgreichen Geolokalisierung muss ein autonomes intelligentes Objekt seinen Ort bestimmen oder durch die intelligente Umgebung bestimmt werden. Dies kann beispiels-

weise ein GPS-Modul leisten, sofern die Bestimmung unter freiem Himmel stattfindet. In geschlossene Räumen schlägt eine GPS-Ortung im Allgemeinen fehl.

Ausgehend von einem intelligenten Objekt, welches per WLAN in das Netzwerk eingebunden wird, könnte eine Lösung in der Auswertung der WLAN-Signalstärke liegen. Bei mehreren *Access Points* (AP) kann das intelligente Objekt die jeweiligen Signalstärken messen. Diese Messwerte teilt das intelligente Objekt der intelligenten Umgebung mit. Diese kann die Position des intelligenten Objekts näherungsweise über die Ortskenntnis der AP triangulieren. Eine aufwendige und vermutlich unpräzise Methode, die einen weiteren Kommunikationsschritt erfordert. Alternativ könnte man auch spezialisierte Signalgeber installieren und das GPS-Konzept auf Innenräume übertragen. Objekte müssten entsprechend mit vermeintlich teurer Spezialhardware angepasst werden. Bei kabelgebundenen Objekten wird der Ort näherungsweise über die Kabellänge bestimmt. Über die Signallaufzeit können moderne Netzwerkkomponenten die Kabellänge bestimmen. Moderne Switches nutzen die Kabellängenerkennung zum Energiemanagement [Cis14].

Eine Geolokalisierung kann technisch gelöst werden, ist aber je nach Methode nur in einer Näherung möglich. Sie benötigt weiteres Wissen über die Umgebung, welches von Menschen zusammengetragen und verwaltet werden muss. Unter Umständen wird teurere und weitere Hardware benötigt.

Ein generelles Problem ergibt sich aus der Erreichbarkeit im Netzwerk. So bilden Router und Gateways natürliche Netzwerkgrenzen, die Broadcast-Nachrichten normalerweise nicht über Netzwerkgrenzen hinaus tragen. IPv6[4] bietet mit dem Multicast eine Lösungsmöglichkeit [RFC4291].

Gibt es grundlegende Änderungen im gesamten Prozess der Sensorintegration, kann eine Abwärtskompatibilität entweder nicht mehr gegeben oder erwünscht sein. Eine Versionsnummer im Protokoll einer autonomen Sensorintegration hilft zu entscheiden, ob intelligentes Objekt und intelligente Umgebung miteinander kommunizieren können bzw. eine Integration möglich ist. Im Fall einer Inkompatibilität kann dies auf beiden Seiten zur Fehlerdiagnose protokolliert werden.

[4]Internet Protocol Version 6 – https://www.ietf.org/rfc/rfc2460.txt

Änderungen an der Schnittstelle kann man mit einer dynamischen Schnittstellenbindung vorbeugen. Die intelligente Umgebung sendet hierzu seine Schnittstellenbeschreibung dem intelligen Objekt. Für eine solche Beschreibung sind mehrere Wege denkbar. WSDL[5] ist hier ein etablierter Standard, für REST-basierte gibt es WADL[6] – beiden fehlt jedoch semantisches Wissen. Möchte man der Schnittstellenbeschreibung eine semantische Bedeutung verleihen bieten sich dafür SAWSDL[7], OWL-S[8] oder WSML[9] an.

Probleme, die sich aus Sicherheitsaspekten ergeben werden in Kapitel 5.8 gesondert betrachtet.

5.8 Sicherheit

Die Prämisse dieser Diplomarbeit ist eine private Cloud. Das gesamte System wird als geschlossen und somit sicher betrachtet. In der Praxis muss dieser Ansatz jedoch überdacht werden.

Das sogenannte Stuxnet-Schadprogramm[10] manipuliert gezielt Industriesteuerungen und sabotierte z. B. Uranzentrifugen im Iran. Überträgt man dies auf `SitOPT` und `SitRS`, können manipulierte Sensordaten Situationen auslösen, die ungewollte Workflow-Adaptionen nach sich ziehen.

> *Beispiel einer manipulierten Situation:*
> *Der Temperatursensor einer Maschine wird manipuliert, so dass die Temperatur immer über dem Grenzwert liegt. Die fälschlich erkannte Situation führt zu einem adaptierten Workflow, der die Maschine zum Abkühlen pausiert bis die Temperatur wieder im erlaubten Bereich liegt.*

Verschiedene Sicherheitsmaßnahmen können einer (Industrie-)Spionage bzw. Sabotage vorbeugen oder solche Angriffe erschweren.

Ein erster Schritt sind verschlüsselte Kommunikationswege. Dadurch wird verhindert, dass die Kommunikation zum einen manipuliert, zum anderen ausspioniert werden kann. REST-Schnittstellen über HTTPS[11] sichern die Kommunikation kryptographisch gegen „*Man in the middle*“-Angriffe ab.

[5] Web Services Description Language
[6] Web Application Description Language
[7] Semantic Annotations for WSDL and XML Schema
[8] `http://www.w3.org/Submission/OWL-S/`
[9] Web Service Modeling Language
[10] `https://de.wikipedia.org/w/index.php?title=Stuxnet&oldid=147449547`
[11] Hypertext Transfer Protocol Secure – `https://tools.ietf.org/html/rfc2818`

Eine Benutzer-Authentifizierung – z. B. in Form eines Benutzernamens und Kennwortes – schützt das System vor unberechtigtem Zugriff durch nicht autorisierte Personen. Zusätzlich kann eine Benutzer-Authentifizierung um ein Rechtesystem ergänzt werden. Mit einem solchen Rechtesystem kann ein Benutzer beispielsweise auf einen rein lesenden Zugriff beschränkt werden und kann nur Daten aus der Ontologie abfragen. Einem anderen Benutzer hingegen kann der volle Zugriff gewährt werden.

Eine autonome Sensorintegration (vgl. Kapitel 5.7) erfordert ein komplexeres Sicherheitskonzept, da die Kontrolle aus der Hand der Benutzer rückt. Ein bösartig konzipiertes Objekt könnte in das System eingeschleust werden. In ein geschlossenes System eingebracht, kann so ein bösartiges Objekt eine Schnittstelle nach außen hin öffnen. Über diese Schnittstelle ist dann eine Manipulation oder ein Ausspionieren des Systems möglich.

Dem kann z. B. durch ein einfaches Zertifikatssystem vorgebeugt werden. Objekte, welche kein gültiges Zertifikat vorweisen können, werden abgewiesen. Eine technisch ausgereiftere und komplexere Lösung lässt sich durch den XACML[12]-Standard erreichen. Mit ihm lassen sich Sicherheitsrichtlinien wie Authentifizierungsrichtlinien und Autorisierungen zentral steuern [XACML].

Vollständige Sicherheit ist allerdings nicht erreichbar. Am Ende bleibt immer der Mensch als Sicherheitsrisiko bestehen.

[12] eXtensible Access Control Markup Language

6 Implementierung

> Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live
>
> *(John F. Woods)*

Dieses Kapitel beschreibt die prototypische Implementierung der Sensorintegration `SeInt`. Es werden dabei einzelne Designentscheidungen, Probleme und deren Lösungsmöglichkeiten diskutiert. Die Implementierung besteht einerseits aus einer Software und andererseits aus der Ontologie.

6.1 Technologieentscheidung für die Software

Am Anfang der Software-Implementierung galt es zu entscheiden, welche Technologien sich empfehlen und geeignet sind, das in Kapitel 5 vorgestellte Konzepte umzusetzen.

Sowohl die *Resource Management Platform* (RMP) als auch die *Sensor Registry* bieten REST-basierte Schnittstellen. Dementsprechend folgt `SeInt` dem gleichen Schnittstellenkonzept, um mit den anderen Teilen konform zu bleiben. Als Alternative böte sich z. B. SOAP[1] an. REST ist allerdings im Vergleich zu SOAP direkter und weniger umfangreich zu implementieren. Beide bieten eine umfangreiche Infrastruktur wie Web- bzw. Anwendungsserver (*application server*) sowie HTTP als Transportprotokoll.

Aus dem Bereich der Webtechnologien bieten sich diverse Programmiersprachen zur Entwicklung einer REST-basieren Schnittstelle an. Beispiele, zu denen es teilweise Unterstützung für Ontologien in Form von Bibliotheken gibt, sind:

- PHP – eine Skriptsprache entworfen für dynamische Webanwendungen.
- node.js – JavaScript in einer serverseitigen Laufzeitumgebung, eignet sich vor allem für eine ereignisgesteuerte Entwicklung.

[1] Simple Object Access Protocol

- ASP.Net – ist von Microsoft und limitiert damit die Infrastruktur auf der ein Webservice läuft.
- JAX-RS – *Java API for RESTful Web Services* als API Spezifikation.

Des Autors Vorkenntnisse präferierten eine Lösung in PHP oder node.js.

Allerdings ergaben Recherchen, dass es für Java die größte Unterstützung durch Bibliotheken gibt. Java bietet mit

- Jersey[2] – als Referenzimplementierung für die JAX-RS API Spezifikation
- Jena[3] – eine Bibliothek u.a. für OWL und SPARQL
- jSch[4] – eine SSH2-Implementierung, kommt bei der Adapter-Provisionierung zum Einsatz.

die notwendigen Bibliotheken, um `SeInt` als prototypische Implementierung umzusetzen. Die Bibliotheken werden mit einer Javadoc-Dokumentation ausgeliefert, diese beschreibt i.A. jedoch nur die Struktur einer API Bibliothek, erklärt aber nicht, wie sie zu verwenden ist. Weitere Dokumentation zu den API Bibliotheken ist zwar vorhanden, benötigt jedoch ein tieferes Verständnis der jeweiligen Bibliothek.

Eine Implementierung in Java bietet die Möglichkeit die gesamte Anwendung in ein portables *Web Application Archive* (WAR) zu packen. Ein Anwendungsserver wie z. B. GlassFish[5] genügt, um WAR auszuführen.

6.2 Ontologie mit Protégé

Es gibt nicht „die eine richtige Methode“ wie Noy und McGuinness in einer Anleitung zum Entwurf einer Ontologie betonen [NM01]. Der Ontologie-Entwurf geschah in einem iterativen Prozess und wurde stetig neu geordnet, erweitert und geändert.

Technisch wurde die Ontologie in OWL-DL implementiert. OWL ist XML-basiert und der händische Entwurf komplexer XML Dokumente ist zeitaufwendig und fehleranfällig. Ontologie-Editoren sind hierbei eine hilfreiche Unterstützung. Es gibt eine Vielzahl Ontologie-Editoren, z. B. Protégé und weitere[6].

[2] `https://jersey.java.net/`
[3] `http://jena.apache.org/`
[4] `http://www.jcraft.com/jsch/`
[5] `https://glassfish.java.net/`
[6] Übersicht verschiedener Ontologie-Editoren – `http://www.w3.org/wiki/Ontology_editors`

Protégé war der erste untersuchte Ontologie-Editor. Er erwies sich direkt als mächtig und komfortabel genug, um eine Ontologie zu entwerfen. Weitere Editoren wurden deswegen nicht untersucht. Protégé macht es sehr einfach Hierarchien graphisch-gestützt zu entwerfen.

Nach einer ersten Begriffssammlung für die Ontologie galt es diese Begriffe in einen Zusammenhang zu setzen. Ein Problem war dabei z. B. die Einordnung der Koordinaten. Wie in Abbildung 6.1 dargestellt, können Koordinaten entweder als eigene `owl:Class` mit einer `owl:ObjectProperty` oder als `owl:DatatypeProperty` zu einer Objekt-`Instanz` strukturiert werden.

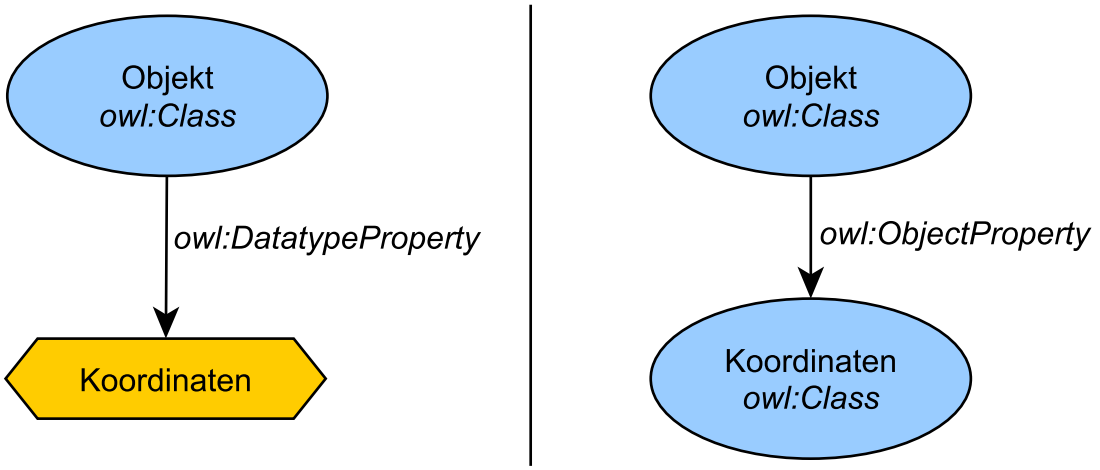

Abbildung 6.1: Koordinaten als Klasse oder Wert

Koordinaten als eigene `owl:Class` bieten den Vorteil, dass eine `Instanz` dieser `Klasse` verschiedenen Objekt-`Instanzen` zugeordnet und damit wiederverwendet werden kann.

Bei der Adapter-`Klasse` stellte sich die Frage, ob die Adapter in Form eines Python-Skripts (vgl. Kapitel 6.3.4) selbst oder nur eine Referenz dieser in einem *Adapter Repository* (`AR`) in der Ontologie platziert werden. Das Speichern in der Ontologie hätte den Vorteil, dass alles zusammen an einem Ort verwaltet werden kann. Sie hätte auch keine Abhängigkeiten zu einem `AR`. Allerdings müsste ein Adapter serialisiert in der Ontologie hinterlegt werden, um sicher in ein XML-Dokument eingebettet werden zu können. Die Datengröße der Ontologie ist mit einer Referenz zum `AR` geringer. Zudem sind Adapter und Ontologie entkoppelt, wodurch `SeInt` modularer gestaltet ist.

Bereits vorhandene Sensor-Ontologien (vgl. Kapitel 4.2 und 4.3) sind zu umfangreich und komplex, um im Zuge dieser Diplomarbeit Verwendung zu finden. Die hier entworfene Sensor-

Ontologie – inspiriert durch SensorML – beschränkt sich deswegen auf ein Hierarchiemodell zur Klassifizierung der Sensortypen.

`Instanzen`, die mit Protégé erzeugt werden, sind vom Typ `owl:NamedIndividual`. Sie tragen ihre Id in der Bezeichnung. Anonyme Instanzen fehlt die Id und eignen sich vorwiegend für einen internen Ontologie-Gebrauch. In dieser Ontologie werden nur `owl:NamedIndividual` verwendet.

Listing 6.1 OWL Instanz eine Sensors – Ultraschall Messmodul HC-SR04

```
<owl:NamedIndividual rdf:about="&seint;HC-SR04">
  <rdf:type rdf:resource="&seint;ultrasonicPulseEcho"/>
  <powerConsumption rdf:datatype="&xsd;integer">10</powerConsumption>
  <measureInterval rdf:datatype="&xsd;integer">20</measureInterval>
  <rangeLowerLimit rdf:datatype="&xsd;integer">20</rangeLowerLimit>
  <measurePrecision rdf:datatype="&xsd;integer">3</measurePrecision>
  <rangeUpperLimit rdf:datatype="&xsd;integer">30000</rangeUpperLimit>
  <sensorId rdf:datatype="&xsd;integer">5</sensorId>
  <unitName>millimeter</unitName>
</owl:NamedIndividual>
```

Neben der Id des `owl:NamedIndividual` gibt es für jedes Objekt und jeden Sensor noch ein eigenes Id-Feld, z. B. `sensorId` (vgl. Listing 6.1). Diese zweite Id ist dafür gedacht, Objekte und Sensoren projektweit – getrennt von der Ontologie – eindeutig identifizieren zu können.

6.3 Sensorintegration – `SeInt`

`SeInt` besteht wie im Konzept beschrieben (vgl. Kapitel 5.1) aus mehreren Komponenten, deren Implementierung in den folgenden Unterkapiteln erläutert wird.

6.3.1 Ontologie-Schnittstelle

Die Ontologie-Schnittstelle wurde mit Hilfe der Jena-Bibliothek implementiert.

Die Namensräume innerhalb der Ontologie werden in einem Enum – einem Aufzählungstypen – verwaltet. Dieser Enum enthält zu jedem Namensraum eine Konstante und eine Methode, um den Wert einer Konstante auszulesen.

Bestimmte Werte einer Instanz der Ontologie werden direkt über die OWL-API von Jena ausgelesen. Hierzu wurde eine generische Methode entwickelt, um zu einer beliebigen `Instanz` eine bestimmte Eigenschaft abzufragen (vgl. Listing 6.2).

Listing 6.2 Methode zum direkten Auslesen eines speziellen Wertes einer Instanz

```
public String getPropertyValueOfIndividual(String individualName, String propertyName)
{
    Individual individual = ontology.getIndividual(Prefix.SEINT.value() +
 ↪   individualName);
    DatatypeProperty property = ontology.getDatatypeProperty((Prefix.SEINT.value() +
 ↪   propertyName));
    return individual.getPropertyValue( property ).toString();
}
```

Geht es um die Abfrage einer gesamten Instanz aus der Ontologie wird die SPARQL-Query-API von Jena verwendet. Es gibt verschiedene SPARQL-Abfragen, um beispielsweise die Instanz eines Objekts oder Sensors abzufragen (vgl. Listing 6.3).

Die Jena-Bibliothek bietet verschiedene Ausgabeformate für Ontologie-Abfragen. Davon wird in Kombination mit der REST-basieren Schnittstelle Gebrauch gemacht, um entsprechende Ausgaben zu generieren.

Die SPARQL-Query-API erlaubt keine Manipulationen der Ontologie. Hierfür gibt es in dem sehr speziellen Bibliotheksdesign von Jena eine gesonderte SPARQL-Update-API, die sich allerdings identisch zur SPARQL-Query-API bedienen lässt.

Listing 6.5 zeigt die Ausgabe zu einer Beispiel Instanz des „Ultraschall Messmodul HC-SR04“.

6.3.2 REST-basierte Schnittstelle nach außen

Diese Schnittstelle ist mit Jersey implementiert. Jersey bringt eine Menge Annotationen mit, die den Quellcode sehr klein und übersichtlich halten.

Listing 6.4 zeigt die Implementierung um alle Sensoren der Ontologie aufzulisten. Bei einem HTTP-`GET` auf die REST-Ressource `/sensor/list` wird die Methode `getSensorListAsText()` aufgerufen. Auf diese Art und Weise sind sämtliche REST-Ressourcen deklariert.

Wie im Konzept erläutert, folgt die Bedienung der Ontologie dem CRUD-Schema (vgl. Kapitel 5.3). HTTP bietet verschiedene Methoden an, die wie folgt verwendet werden:

`PUT`	Erzeugen neuer Daten in der Ontologie
`GET`	Abfrage der Ontologie
`POST`	Ändern von Daten in der Ontologie
`DELETE`	Löschen von Daten in der Ontologie

Listing 6.3 Methode zum Auslesen der Sensordaten

```
/**
 * Gets sensor data by its name (owl:NamedIndividual) and outputs in a given format
 *
 * @param name String owl:NamedIndividual name of a sensor
 * @param outputFormat String
 * @return String sensor data in a given format
 */
public String getSensorByName(String name, OutputFormat outputFormat)
{
        String queryString =
                        "SELECT ?sensor ?predicate ?object \n" +
                        "WHERE {\n" +
                        "   ?class "+Prefix.RDFS.prefix()+":subClassOf*
 ↪   "+Prefix.SEINT.prefix()+":sensor . \n" +
                        "   ?sensor "+Prefix.RDF.prefix()+":type ?class . \n" +
                        "   ?sensor ?predicate ?object . \n" +
                        "FILTER ( ?sensor = "+Prefix.SEINT.prefix()+":"+name+" ) \n" +
                        "}";

        return query(queryString, outputFormat);
}
```

Listing 6.4 Beispiel einer REST-Ressource in Jersey

```
@Path("/sensor")
public class Sensor extends RestResource
{
    @GET
    @Path("/list")
    @Produces({MediaType.TEXT_PLAIN, MediaType.WILDCARD})
    public Response getSensorListAsText()
    {
        String output = ontology.getSensorList( OntologyManager.OutputFormat.TEXT );
        return out(output);
    }
```

6.3.3 Klient für SeInt

Die Benutzer-Klient zur Bedienung der REST-basierten Schnittstelle wurde als HTML5-Webseite in Kombination mit JavaScript entworfen. Webbrowser gibt es für alle gängigen Betriebssysteme, auch wenn derzeitig noch nicht alle Webbrowser HTML5 in vollem Umfang unterstützen.

Für JavaScript wird die jQuery[7]-Bibliothek verwendet. Sie erleichtert DOM[8]-Manipulationen und AJAX[9] wesentlich.

In HTML5 wurde für die einzelnen REST-Ressourcen jeweils ein Formular beschrieben, um die Parameter eines URI eingeben zu können (vgl. Abbildung 6.2). Jedes dieser Formulare enthält Buttons zum ausführen der Anfrage.

HTML5 Formulare erlauben nur `POST` und `GET` als HTTP-Methoden. Außerdem definiert der Browser den HTTP Request-Header, insbesondere die MIME-Typen des `Accept`-Feldes. Dieses Problem wird mit JavaScript und der AJAX-Technik gelöst. Mittels jQuery lassen sie AJAX-Request sehr einfach gestalten, vor allem kann man das `Accept`-Feld frei definieren.

Listing 6.6 zeigt eine vollständige HTML5-*Section*, inklusive dem Formular und JavaScript, das die Formularbehandlung übernimmt. Die Buttons werden auf ein auslösendes Ereignis – ein Klick oder Tastendruck – hin überwacht. Mit dem Ereignis wird die Anfrage an die REST-Ressource ausgelöst. Es werden die Parameter der Ressource mit den Formularfeldern ausgefüllt. Der Button bestimmt das Ausgabeformat, entsprechend wird im AJAX-Objekt von jQuery der `dataType` gesetzt.

Listing 6.5 zeigt die Antwort einer Anfrage im einfachen Textformat.

6.3.4 Adapter auf einem Raspberry Pi

Im Rahmen dieser Diplomarbeit werden die Objekte durch einen Raspberry Pi dargestellt. An diesen Minicomputer sind die Sensoren über die GPIO[10]-Schnittstelle angeschlossen. Als Sensor kam ein *Ultraschall Messmodul HC-SR04*[11] zum Einsatz.

Auf dem Raspberry Pi läuft ein Linux-Betriebsystem, des Weiteren gibt es SSH und einen Python-Interpreter. Beides wird für die Adapter-Provisionierung eingesetzt.

[7] `https://jquery.com/`
[8] Document Object Model
[9] `AsynchronousJavaScriptandXML`
[10] General-purpose input/output
[11] Ultraschall Messmodul HC-SR04 – `http://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf`

Listing 6.5 Ontologie Sensor-`Instanz` des Ultraschall Messmodul HC-SR04

sensor	property	value
seint:HC-SR04	seint:unitName	"millimeter"
seint:HC-SR04	seint:sensorId	5
seint:HC-SR04	seint:rangeUpperLimit	30000
seint:HC-SR04	seint:measurePrecision	3
seint:HC-SR04	seint:rangeLowerLimit	20
seint:HC-SR04	seint:measureInterval	20
seint:HC-SR04	seint:powerConsumption	10
seint:HC-SR04	rdf:type	seint:ultrasonicPulseEcho
seint:HC-SR04	rdf:type	owl:NamedIndividual
seint:HC-SR04	rdf:type	rdfs:Resource
seint:HC-SR04	rdf:type	seint:sensor
seint:HC-SR04	rdf:type	seint:physicalSensor

Die Adapter sind als Vorlage in einem *Adapter Repository* (AR) hinterlegt, wobei das AR sich auf eine einfache Dateiablage beschränkt. Die Vorlage ist ein Python-Skript mit Platzhaltern für die notwendigen Parameter wie die URL der *Resource Management Platform* (`RMP`) sowie ObjektId und SensorId.

Das Python-Skript besteht aus zwei Funktionen:

1. eine misst die Signale an der GPIO-Schnittstelle. Dort kommen nur Rechtecksignale mit den Werten 1 bzw. 0 an. Diese Signale müssen erst noch als Sensorwert interpretiert werden. Beim Beispiel des *Ultraschall Messmodul HC-SR04* genügt es, die Dauer zwischen dem Ultraschall-Ping und dem Echo zu messen. Aus dieser Zeit lässt sich der gemessene Abstand als Sensorwert berechnen.
2. der Sensorwert wird als HTTP-POST an die `RMP` gesendet. Die python-Bibliothek Requests[12] wird zu diesem Zwecke eingesetzt.

Beide Funktionen laufen in einer Endlosschleife.

Im Prozess der Sensorintegration wird die Vorlage in einen String eingelesen und die Platzhalter durch ihre die entsprechenden Parameter ersetzt. Hierbei wird ein lauffähiges Python-Skript erzeugt. Im Anschluss wird mit Hilfe der jSch-Bibliothek eine SSH Verbindung zum Raspberry Pi aufgebaut. Dazu wurden die Verbindungsdaten, bestehend aus IP-Adresse, Benutzername und Passwort, der Ontologie entnommen. Über diese Verbindung wird das

[12]Requests: HTTP for Humans –`http://docs.python-requests.org/en/latest/`

Listing 6.6 Quellcodebeispiel des Klient

```
<!-- section to get a sensor by name -->
<section id="getSensorByName" class="get" tabindex="30">
  <h2>GET Sensor by: NAME</h2>
  <div>
    <fieldset>
      <legend>GET <a href="">/rest/sensor/name/{name}</a></legend>
      <label>Sensor {name}: <input class="getSensorByName" type="search"></label>
      <button class="getSensorByName" >search</button>
      <button class="getSensorByName json" >search (json)</button>
    </fieldset>
    <script type="text/javascript">
      (function(){
        var trigger = '.getSensorByName';
        var path    = 'rest/sensor/name/';

        $(trigger).bind('keypress click',function(e){
          var value = $('input'+trigger).val();
          if (value && (e.keyCode == 13 || $(this).is(':button'))) {
            var outputFormat = ( $(this).is('.json') ? "json" : "text" );
            $.ajax({
              url: path + value,
              dataType: outputFormat,
              success: function(data, status, jqXHR){
                console.log('ajax success', arguments);
                var w = window.open();
                var output = ( outputFormat == "json" ? JSON.stringify(data,null,4) :
 ↪  data );
                $(w.document.body).append("<pre></pre>");
                $(w.document).find("pre").text(output);
              },
              error: function(jqXHR, status, errorThrown){
                console.log('ajax error', arguments);
                alert('ERROR '+jqXHR.status+'\n\n'+status+'\n'+errorThrown);
              }
            });
          }

          //update url of fieldset legend
          value = value || '{name}';
          $(this).parents('fieldset').find('legend a').attr('href', path +
 ↪  value).text(path + value);
        });
      }());
    </script>
  </div>
</section>
```

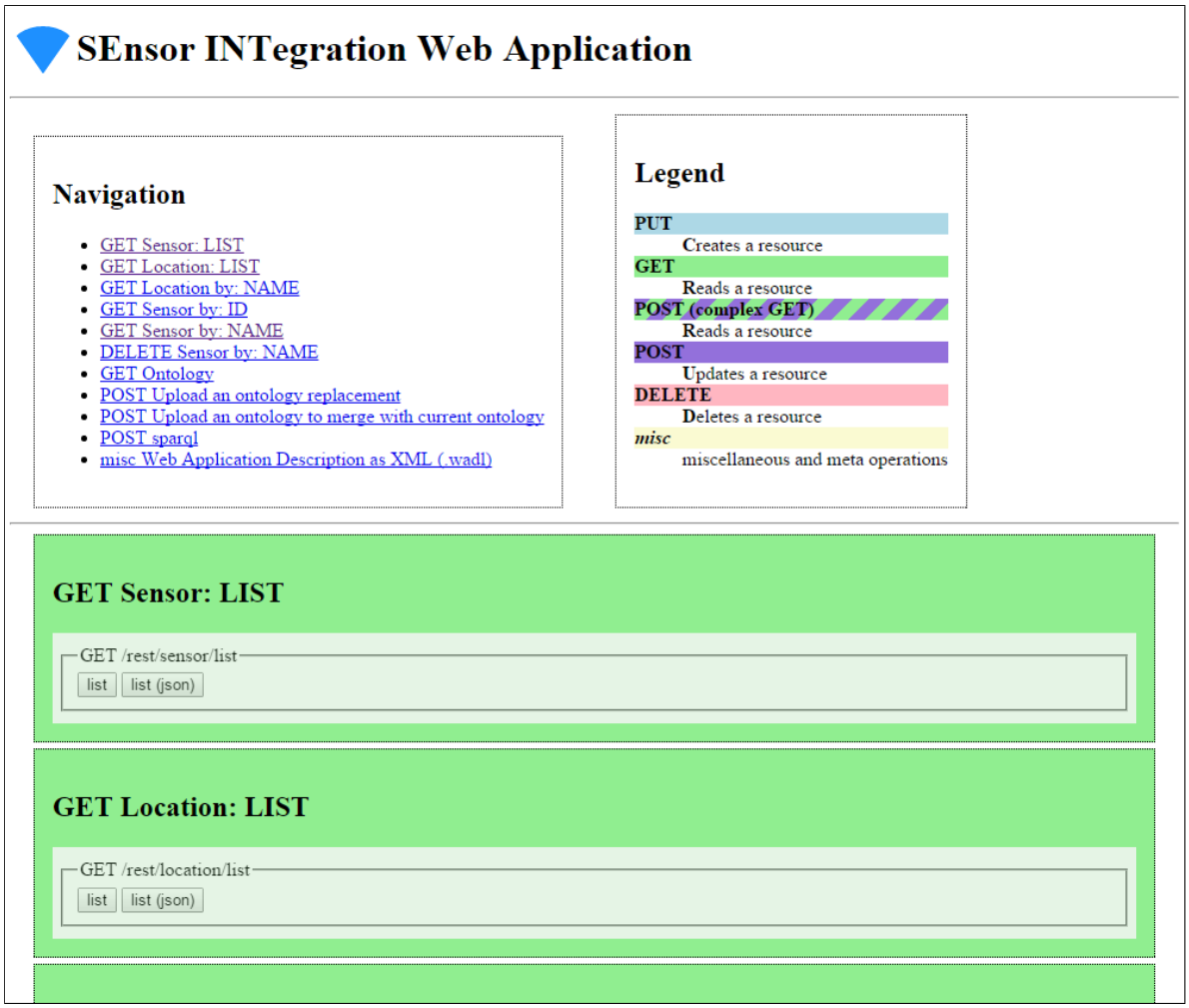

Abbildung 6.2: Weboberfläche des Klients

Python-Skript auf den Raspberry Pi kopiert. Abschließend wird das Python-Skript im Hintergrund gestartet und die SSH-Verbindung beendet.

Das Python-Skript soll nach beenden der SSH-Verbindung noch laufen. Mit jSch ist es nicht ganz trivial einen Prozess per SSH im Hintergrund zu starten. Die Standardmethode einen Prozess in einer Shell in den Hintergrund zu legen funktioniert nicht mit jSch. Dazu wird der `sudo`-Befehl unter Linux verwendet.

7 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine automatisierte ad-hoc Sensorintegration basierend auf einer Ontologie zu schaffen. Das Konzept und die prototypische Implementierung bieten eine REST-basierte Schnittstelle. Über diese lassen sich Objekte und ihre Sensoren auswählen. Die Zugehörigkeit wird über eine Ontologie ermittelt. Sensoren können also dem Ziel dieser Arbeit entsprechend ad-hoc und automatisiert in das System von `SitRS` integriert werden.

Am Anfang dieser Arbeit wurde der Hintergrund zu `SitOPT` erläutert und welche Rolle `SitRS` in diesem System besitzt. Die Motivation dieser Arbeit leitet sich aus den Anforderung einer Sensorintegration in `SitRS` ab.

Es wurden die notwendigen Grundlagen zum Verständnis dieser Arbeit erklärt. Anschließend wurden verschiedene themenverwandte Arbeiten vorgestellt, die Lösungen zu Teilaspekten dieser Arbeit liefern.

In Kapitel 5 ist ein umfangreiches Konzept zur Sensorintegration entworfen worden. Dieses Konzept beinhaltet den Entwurf einer Ontologie, um Wissen über Objekte und Sensoren semantisch zu repräsentieren und nutzbar zu machen. Außerdem wurde ein modularisierte Softwaresystem für interagierende Komponenten konzipiert. Im Zusammenspiel stellen die Komponenten die Sensorintegration dar.

Abschließend wurde die prototypische Implementierung beschrieben und die darin verwendeten Technologien und Methoden diskutiert.

Fazit und Ausblick

Ontologien bieten einen guten Ansatz, um semantisches Wissen über die Umgebung zu repräsentieren. Metadaten über Sensoren und Objekte lassen sich gut damit verwalten und die Bedeutung der Metadaten ist durch die Ontologie gegeben. Über die vorhandene Schnittstelle der Sensorintegration können sich andere Dienste dieses semantischen Wissens bedienen.

Für die Zukunft kann die hier vorgestellte Ontologie weiter verfeinert werden. Speziell die Sensoren können noch feiner granuliert werden. Dazu gehört z. B. die Auffächerung unterschiedlicher Sensortypen, aber auch der Ausbau der Sensoreigenschaften wie sie in den Grundlagen angesprochen worden sind. Auch können die Möglichkeiten der Inferenz in der Ontologie deutlich ausgebaut werden. Beispielweise um Formulierungen der Art „integriere alle Temperatorsensoren die sich einem bestimmten Raum befinden“ Folge leisten zu können.

Die Sensorintegration kann in verschiedenen Aspekten gegenüber der prototypischen Implementierung erweitert werden. Der Mechanismus der Adapter-Provisionierung wurde über einen speziellen Ansatz gelöst. Er definiert sich durch die Infrastruktur der verwendeten Minicomputer – Raspberry Pi. In Zukunft ist hier ein allgemeiner Ansatz denkbar, der auch mit verschiedenen Infrastrukturen funktioniert. Die Konzepte hinter OpenTOSCA können hier als Vorbild dienen.

Eine autonome Sensorintegration stellt den gesamten Prozess vor große Herausforderungen und wurde in dieser Arbeit nur konzeptionell beschrieben. Sie geht mit dem Sicherheitskonzept einher. Generell wurden Sicherheitsaspekte in der Implementierung dieser Arbeit außen vor gelassen. Das Konzept vermittelt Ideen und Anhaltspunkte, welche Aspekte hier angegangen werden können.

Literaturverzeichnis

[Ash09] K. Ashton. That 'Internet of Things' Thing. RFID Journal, 2009. URL http://www.rfidjournal.com/articles/pdf?4986. (Zitiert auf Seite 21)

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *11th International Conference on Service-Oriented Computing*, LNCS. Springer, 2013. (Zitiert auf den Seiten 40 und 51)

[Bet] S. Betten. Richardson Maturity Model. URL http://www.se.uni-hannover.de/pub/File/kurz-und-gut/ws2011-labor-restlab/RESTLab-Richardson-Maturity-Model-Sascha-Betten-kurz-und-gut.pdf. (Zitiert auf Seite 36)

[BHK+] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. A Situation-Aware Workflow Modelling Extension. Sumbitted to the 17th International Conference on Information Integration and Web-based Applications & Services (IIWAS), 2015. (Zitiert auf Seite 15)

[BHL+14] J. Busse, B. Humm, C. Lübbert, F. Moelter, A. Reibold, M. Rewald, V. Schlüter, B. Seiler, E. Tegtmeier, T. Zeh. Was bedeutet eigentlich Ontologie? *Informatik-Spektrum*, 37(4):286–297, 2014. doi:10.1007/s00287-012-0619-2. URL http://dx.doi.org/10.1007/s00287-012-0619-2. (Zitiert auf Seite 23)

[BN03] F. Baader, W. Nutt. The Description Logic Handbook. Kapitel Basic Description Logics, S. 43–95. Cambridge University Press, New York, NY, USA, 2003. URL http://dl.acm.org/citation.cfm?id=885746.885749. (Zitiert auf Seite 25)

[Bra96] C. Braig. *Vom Sein. Abriß der Ontologie.* Herder, Freiburg im Breisgau, 1896. URL https://www.freidok.uni-freiburg.de/fedora/objects/freidok:806/datastreams/FILE1/content. (Zitiert auf Seite 23)

[BMBF15] Bundesministerium für Bildung und Forschung. *Industrie 4.0 - Innovation für die Produktion von morgen.* Bundesministerium für Bildung und Forschung, 2. Auflage, 2015. URL http://www.bmbf.de/pub/Industrie_4.0.pdf. (Zitiert auf den Seiten 9 und 22)

[Cis14] Cisco Systems. Cisco Switches der Serie 300 Cisco Small Business, 2014. URL http://www.cisco.com/c/dam/en/us/products/collateral/switches/small-business-smart-switches/300_Series_Switches_DS_FINAL_2757.pdf. (Zitiert auf Seite 57)

[Cox13] S. Cox. An explicit OWL representation of ISO/OGC Observations and Measurements. In *Proceedings of the 6th International Workshop on Semantic Sensor Networks co-located with the 12th International Semantic Web Conference (ISWC 2013)*, Band 1063, S. 1–18. 2013. URL http://ontolog.cim3.net/file/work/OntologyBasedStandards/2013-10-17_Ontologies-for-Geospatial-Standards/wip/Cox_OM-OWL_20131017b1.pdf. (Zitiert auf Seite 37)

[1319-1] DIN-Normenausschuss Technische Grundlagen (NATG). *Grundlagen der Meßtechnik - Teil 1: Grundbegriffe*. Deutsches Institut für Normung e. V., Am DIN-Platz, Burggrafenstraße 6, 10787 Berlin. DIN 1319-1 : 1995-01. (Zitiert auf Seite 22)

[RFC2131] R. Droms. Dynamic Host Configuration Protocol. Technischer Bericht, Network Working Group, Request for Comments, 1997. URL https://tools.ietf.org/html/rfc2131. RFC2131. (Zitiert auf Seite 55)

[Fie00] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, 2000. URL http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. (Zitiert auf den Seiten 34 und 36)

[Fie08] R. T. Fielding. REST APIs must be hypertext-driven. Privater Blog, 2008. URL http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. (Zitiert auf Seite 35)

[For15] W. E. Forum©. Industrial Internet of Things: Unleashing the Potential of Connected Products and Services, 2015. URL http://www3.weforum.org/docs/WEFUSA_IndustrialInternet_Report2015.pdf. (Zitiert auf Seite 9)

[Fow] M. Fowler. Richardson Maturity Model. URL http://martinfowler.com/articles/richardsonMaturityModel.html. (Zitiert auf Seite 36)

[GOS09] N. Guarino, D. Oberle, S. Staab. What Is an Ontology?, 2009. URL http://iaoa.org/isc2012/docs/Guarino2009_What_is_an_Ontology.pdf. (Zitiert auf Seite 25)

[Gru93] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, Band 5(Nummer 2):199–220, 1993. URL http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html. (Zitiert auf den Seiten 24 und 38)

[RFC4291] R. Hinden, S. Deering. IP Version 6 Addressing Architecture. RFC 4291, Network Working Group, Request for Comments, 2006. URL https://tools.ietf.org/html/rfc4291. (Zitiert auf Seite 59)

[HWS+15] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, F. Leymann. SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates. In C. Nikolaou, F. Leymann, Herausgeber, *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*, S. 35–49. IBM, 2015. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-34&engl=. (Zitiert auf den Seiten 16, 17 und 18)

[ITU12] ITU-T Study Group 13. Overview of the Internet of things, 2012. URL http://handle.itu.int/11.1002/1000/11559. Recommendation ITU-T Y.2060. (Zitiert auf Seite 21)

[Jan15] P. Jansa. *Eine OSLC- Plattform zur Unterstützung der Situationserkennung in Workflows.* Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2015. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3707&engl=0. (Zitiert auf Seite 16)

[Jas12] P. D.-I. J. Jasperneite. Internet und Automation – Was hinter Begriffen wie Industrie 4.0 steckt. *Computer & AUTOMATION*, 2012. URL http://www.computer-automation.de/steuerungsebene/steuern-regeln/artikel/93559/. (Zitiert auf den Seiten 13 und 22)

[KLW11] H. Kagermann, W.-D. Lukas, W. Wahlster. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. *VDI Nachrichten : Technik, Wirtschaft, Gesellschaft. - [N.F.] 65 (2011), H. 1-17*, 13, 2011. URL http://www.vdi-nachrichten.com/Technik-Gesellschaft/Industrie-40-Mit-Internet-Dinge-Weg-4-industriellen-Revolution. Sammelband 65 (2011), H. 1-17. (Zitiert auf Seite 22)

[Lac05] L. W. Lacy. *OWL: representing information using the Web Ontology Language.* Trafford Publishing, 2005. ISBN: 1-4120-3448-5. (Zitiert auf Seite 28)

[LS11] E. A. Lee, S. A. Seshia. *Introduction to Embedded Systems – A Cyber-Physical Systems Approach.* 1. Auflage, 2011. URL http://leeseshia.org/releases/LeeSeshia_DigitalV1_08.pdf. ISBN 978-0-557-70857-4. (Zitiert auf Seite 22)

[Men02] C. Menzel. Ontology Theory. In *Ontologies and Semantic Interoperability, CEUR Workshop Proceedings*, Band 64. Citeseer, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.7057&rep=rep1&type=pdf. (Zitiert auf Seite 25)

[NM01] N. F. Noy, D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technischer Bericht, Stanford University, 2001. URL http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880. (Zitiert auf Seite 64)

[NXP14] NXP Semiconductors. UM10204 – I²C-bus specification and user manual, 2014. URL http://www.nxp.com/documents/user_manual/UM10204.pdf. Rev. 6. (Zitiert auf Seite 55)

[XACML] OASIS TC. eXtensible Access Control Markup Language (XACML) Version 3.0, 2010. URL http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf. (Zitiert auf Seite 61)

[TOSCA] OASIS Standard. Topology and Orchestration Specification for Cloud Applications Version 1.0, 2013. URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf. (Zitiert auf Seite 40)

[OWL2a] OWL Working Group. OWL 2 Web Ontology Language – Document Overview (Second Edition). W3C Recommendation, 2012. URL http://www.w3.org/TR/owl2-overview/. (Zitiert auf Seite 32)

[OWL2] OWL Working Group. OWL 2 Web Ontology Language – Primer (Second Edition). W3C Recommendation, 2012. URL http://www.w3.org/TR/owl2-primer/. (Zitiert auf Seite 31)

[OWL2b] OWL Working Group. OWL 2 Web Ontology Language – Profiles (Second Edition). W3C Recommendation, 2012. URL http://www.w3.org/TR/owl2-profiles/. (Zitiert auf Seite 32)

[OWL2c] OWL Working Group. OWL 2 Web Ontology Language – Quick Reference Guide (Second Edition). W3C Recommendation, 2012. URL http://www.w3.org/TR/owl2-quick-reference/. (Zitiert auf Seite 32)

[PH09] D. L. Phuoc, M. Hauswirth. Linked open data in sensor data mashups. *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09), in conjunction with ISWC 2009*, 522, 2009. URL http://hdl.handle.net/10379/1113. CEUR. (Zitiert auf Seite 39)

[RDF1.1] RDF Working Group. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014. URL http://www.w3.org/TR/rdf11-concepts/. (Zitiert auf den Seiten 26 und 27)

[RDFSE] RDF Working Group. RDF 1.1 Semantics. W3C Recommendation, 2014. URL http://www.w3.org/TR/rdf11-mt/. (Zitiert auf Seite 27)

[Turtle] RDF Working Group. RDF 1.1 Turtle. W3C Recommendation, 2014. URL http://www.w3.org/TR/turtle/. (Zitiert auf Seite 32)

[RDFX] RDF Working Group. RDF 1.1 XML Syntax. W3C Recommendation, 2014. URL http://www.w3.org/TR/rdf-syntax-grammar/. (Zitiert auf Seite 27)

[RDFS] RDF Working Group. RDF Schema 1.1. W3C Recommendation, 2014. URL http://www.w3.org/TR/rdf-schema/. (Zitiert auf Seite 28)

[RKT05] D. J. Russomanno, C. R. Kothari, O. A. Thomas. Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models. In *The 2005 International Conference on Artificial Intelligence*, S. 637–643. 2005. URL https://wwwnew.memphis.edu/eece/cas/docs/ica3194.pdf. Las Vegas, NV. (Zitiert auf den Seiten 38 und 39)

[RR07] L. Richardson, S. Ruby. *RESTful Web Services*, Band 1. O'Reilly Media, 2007. ISBN 978-0-596-52926-0. (Zitiert auf Seite 36)

[SBF98] R. Studer, V. R. Benjamins, D. Fensel. Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998. doi:10.1016/S0169-023X(97)00056-6. URL http://dx.doi.org/10.1016/S0169-023X(97)00056-6. (Zitiert auf Seite 24)

[Sen14] OGC© SensorML: Model and XML Encoding Standard, 2014. URL https://portal.opengeospatial.org/files/?artifact_id=55939. OGC 12-000. (Zitiert auf den Seiten 37 und 38)

[SPA13a] SPARQL Working Group. SPARQL 1.1 Query Language. W3C Recommendation, 2013. URL http://www.w3.org/TR/sparql11-query/. (Zitiert auf Seite 34)

[SPA13b] SPARQL Working Group. SPARQL 1.1 Update. W3C Recommendation, 2013. URL http://www.w3.org/TR/sparql11-update/. (Zitiert auf Seite 34)

[OWL] Web Ontology Working Group . OWL Web Ontology Language – Overview. W3C Recommendation, 2004. URL http://www.w3.org/TR/owl-features/. (Zitiert auf Seite 30)

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991. URL https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf. Reprint Pervasive Computing 2002. (Zitiert auf Seite 21)

[WSEM] Wikipedia. Semantic Web Stack — Wikipedia, The Free Encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Semantic_Web_Stack&oldid=681457479. [Online; accessed 28-October-2015]. (Zitiert auf Seite 30)

[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, St. Louis, Missouri, USA, 2015. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-24&engl=1. (Zitiert auf den Seiten 13 und 14)

[Yil06] B. Yildiz. Ontology Evolution and Versioning - The state of the art. Vienna University of Technology, Institute of Software Technology & Interactive Systems (ISIS), 2006. URL http://publik.tuwien.ac.at/files/pub-inf_4603.pdf. (Zitiert auf Seite 24)

[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*, S. 197–200. ACM, New York, NY, USA, 2009. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2009-137&engl=0. (Zitiert auf Seite 16)

Alle URLs wurden zuletzt am 01. November 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