Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Dipomarbeit Nr. 3707

Eine OSLC-Plattform zur Unterstützung der Situationserkennung in Workflows

Paul Jansa

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Ing. habil. B. Mitschang
Betreuer/in:	Dipl.-Inf. Pascal Hirmer
Beginn am:	2. Februar 2015
Beendet am:	3. August 2015
CR-Nummer:	B.4.3, B.8.0, C.0, C.2.4

Kurzfassung

Das Internet der Dinge gewinnt immer mehr an Bedeutung durch eine starke Vernetzung von Rechnern, Produktionsanlagen, mobilen Endgeräten und weiteren technischen Geräten. Derartige vernetzte Umgebungen werden auch als SMART Environments bezeichnet. Auf Basis von Sensordaten können in solchen Umgebungen höherwertige Situationen (Zustandsänderungen) erkannt und auf diese meist automatisch reagiert werden. Dadurch werden neuartige Technologien wie zum Beispiel „Industrie 4.0“, „SMART Homes“ oder „SMART Cities“ ermöglicht. Komplexe Vernetzungen und Arbeitsabläufe in derartigen Umgebungen werden oftmals mit Workflows realisiert. Um eine robuste Ausführung dieser Workflows zu gewährleisten, müssen Situationsänderungen beachtet und auf diese entsprechend reagiert werden, zum Beispiel durch Workflow-Adaption. Das heißt, erst durch die Erkennung höherwertiger Situationen können solche Workflows robust modelliert und ausgeführt werden. Jedoch stellen die für die Erkennung von Situationen notwendige Anbindung und Bereitstellung von Sensordaten eine große Herausforderung dar. Oft handelt es sich bei den Sensordaten um Rohdaten. Sie sind schwer extrahierbar, liegen oftmals nur lokal vor, sind ungenau und lassen sich dementsprechend schwer verarbeiten. Um die Sensordaten zu extrahieren, müssen für jeden Sensor individuelle Adapter programmiert werden, die wiederum ein einheitliches Datenformat der Sensordaten bereitstellen müssen und anschließend mit sehr viel Aufwand untereinander verbunden werden.

Im Rahmen dieser Diplomarbeit wird ein Konzept erarbeitet und entwickelt, mit dessen Hilfe eine einfache Integration von Sensordaten ermöglicht wird. Dazu werden die Sensoren über eine webbasierte Benutzeroberfläche oder über eine programmatische Schnittstelle in einer gemeinsamen Datenbank registriert. Die Sensordaten werden durch REST-Ressourcen abstrahiert, in RDF-basierte Repräsentationen umgewandelt und mit dem Linked-Data Prinzip miteinander verbunden. Durch die standardisierte Schnittstelle können Endbenutzer oder Anwendungen über das Internet auf die Sensordaten zugreifen, neue Sensoren anmelden oder entfernen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1 Einleitung

Im ersten Abschnitt dieses Kapitels werden die Motivation und Zielsetzung der Diplomarbeit beschrieben. Anschließend werden Begriffe erklärt, die für das Verständnis dieser Diplomarbeit wichtig sind.

1.1 Motivation und Zielsetzung

Mit dem Einzug der Industrie 4.0 [18] findet die vierte Stufe der industriellen Revolution statt. Angetrieben durch das Internet verschmelzen die reale und virtuelle Welt zu einem Internet der Dinge [22] zusammen. So werden in Zukunft immer mehr Industriemaschinen aber auch Alltagsgegenstände mit Sensoren und Funkchips ausgestattet sein, damit sie selbstständig untereinander kommunizieren können. Aufgrund der starken Vernetzung von Rechnern, Industrieanlagen und mobilen Endgeräten weit über die Produktionsstätten hinaus, können intelligente Systemumgebungen realisiert werden. Unter dem Oberbegriff „SMART Environments [23]“ entstehen neuartige Technologien wie zum Beispiel *„SMART Homes [11]“*, *„SMART Cities“*, *„SMART Mobility / Connected Cars“* oder auch *„SMART Agriculture“*. Vorrangig sollen durch diese vernetzten Umgebungen effiziente Energieauslastung, Sicherheit und die Erhöhung der Wohn- und Arbeitsqualität verbessert werden. Auf Basis von Sensordaten lassen sich in derartigen Umgebungen höherwertige Situationen erkennen, analysieren und auswerten. Dies ermöglicht beispielsweise die Modellierung komplexer situationsabhängiger Workflows. Um Situationen effizient und zuverlässig erkennen zu können, ist eine einfache Anbindung und Bereitstellung von Sensordaten essentiell. Das Problem dabei ist jedoch, dass Sensordaten oft ungenau sind und nur lokal an den jeweiligen Geräten zur Verfügung stehen. Des Weiteren fehlt es ihnen an einheitlichen Schnittstellen, um auf die Daten zugreifen zu können. Eine Integration von Sensordaten innerhalb eines Systems oder einer intelligenten Umgebung ist bisher mit sehr hohem Aufwand verbunden.

Diese Diplomarbeit zeigt eine Lösung für eine einfache Anbindung und Bereitstellung heterogener Sensordaten an neue oder bestehende Systeme, sowie die zentrale Bereitstellung der Daten im Internet. Dabei werden die Sensordaten mit Hilfe des REST[1]-Frameworks [8] über einzelne REST-Ressourcen bereitgestellt. Jede dieser Ressourcen erhält ihre eigene eindeutige

[1] Representational State Transfer

URI[2], die zentral über das Internet abgerufen werden kann. Mittels der vier einfachen Operationen GET, PUT, POST und DELETE können diese Daten über das HTTP-Protokoll [7] abgefragt, erstellt, aktualisiert oder auch gelöscht werden. Des Weiteren wird die Möglichkeit geschaffen, die Sensordaten untereinander zu verknüpfen.

1.2 Aufbau und Kapitelübersicht

Dieses Dokument wurde in acht Abschnitte unterteilt. Im vorliegenden Einführungskapitel wird die Motivation der Aufgabenstellung dieser Diplomarbeit beschrieben. Anschließend werden für das bessere Verständnis einige Begriffe eingeführt, die erläutert und voneinander abgegrenzt werden. Im anschließenden Hintergrundkapitel wird das Projekt SitOPT [24] vorgestellt. SitOPT bietet ein System, mit dem auf Basis von Sensordaten und Kontextinformationen Situationen erkannt und situationsabhängige Workflows ermöglicht werden. Im darauffolgenden Kapitel werden die Grundlagen der eingesetzten Technologien, die in dieser Diplomarbeit eingesetzt werden, beschrieben. Kapitel 4 beschreibt die Problemstellung und Herausforderung sowie die Aufgabenstellung dieser Arbeit, die im nachfolgenden Kapitel 5 als konzeptionelle Lösung vorgestellt wird. Anschließend wird in Kapitel 6 die technische Umsetzung des vorgestellten Konzeptes geschildert. Kapitel 7 behandelt themenbezogene Arbeiten, welche sich mit ähnlichen Ansätzen beziehungsweise ähnlichen Herausforderungen befassen. Der letzte Abschnitt gibt eine Zusammenfassung dieser Diplomarbeit, aus der dann ein Fazit gezogen wird. Es wird zudem ein Ausblick über zukünftige Forschungen in diesem Themenbereich gegeben. Die Grafiken in dieser Diplomarbeit wurden mit Keynote 6.2.2[3] erstellt.

1.3 Begriffserläuterung

In diesem Abschnitt werden Begriffe definiert, die im Rahmen dieser Arbeit häufig verwendet werden.

- **Sensor**
  Sensoren sind mittlerweile in fast allen technischen Geräten verbaut. Sie befinden sich beispielsweise in Computern, Smartphones, Produktionsanlagen sowie Robotern und Haushaltsgeräten. Sensoren können Positionsdaten, physikalische oder chemische Eigenschaften einer Umgebung erfassen. Diese Daten können dann zum Beispiel an einen Computer geschickt und von diesem verarbeitet werden. Ein Sensor ist immer mit einem zu überwachenden Objekt verbunden.

[2]Uniform Resource Identifier
[3]https://www.apple.com/de/mac/keynote/

- **Objekt**
  Ist von Objekten die Rede, dann ist damit zum Beispiel eine zu überwachende Umgebung oder Geolokation, eine Produktionsanlage oder auch nur ein Teil dieser Anlage gemeint. So kann beispielsweise ein Roboterarm als Teil einer Industrieanlage ein Objekt darstellen, dessen Bewegungsablauf mit Hilfe von Sensoren überwacht wird. Wichtig dabei ist, dass jedes Objekt mit Sensoren ausgestattet sein muss. Objekte können auch wiederum weitere Objekte enthalten. Kein Objekt in diesem Sinne ist zum Beispiel ein Objekt aus der objektorientierten Programmierung.

- **Situation**
  Eine Situation stellt eine Zustandsänderung dar, die von Sensoren wahrgenommen werden kann. Solange sich ein Zustand im Objekt nicht ändert, tritt auch keine neue Situation auf. So kann zum Beispiel eine Situation auftreten, indem ein Sensor einen kritischen Temperaturanstieg innerhalb eines Computerprozessor erfasst. Auf diese Situation muss entsprechend reagiert werden, um die Temperatur wieder auf Normalniveau zu bringen.

- **SMART Environment**
  Von SMART Environment (deutsch: intelligente Umgebung) wird dann gesprochen, wenn eine Umgebung mit Sensoren, Funkmodulen und Prozessoren ausgestattet ist und diese miteinander kommunizieren können. Als Beispiel kann hier das intelligente Haus genannt werden, dessen sämtliche Einrichtungen (Heizung, Licht, Rolläden, etc.) mit Smartphones oder Computern von überall aus bedienbar sind und sich an die Bedürfnisse der Bewohner anpassen.

2 Hintergrund

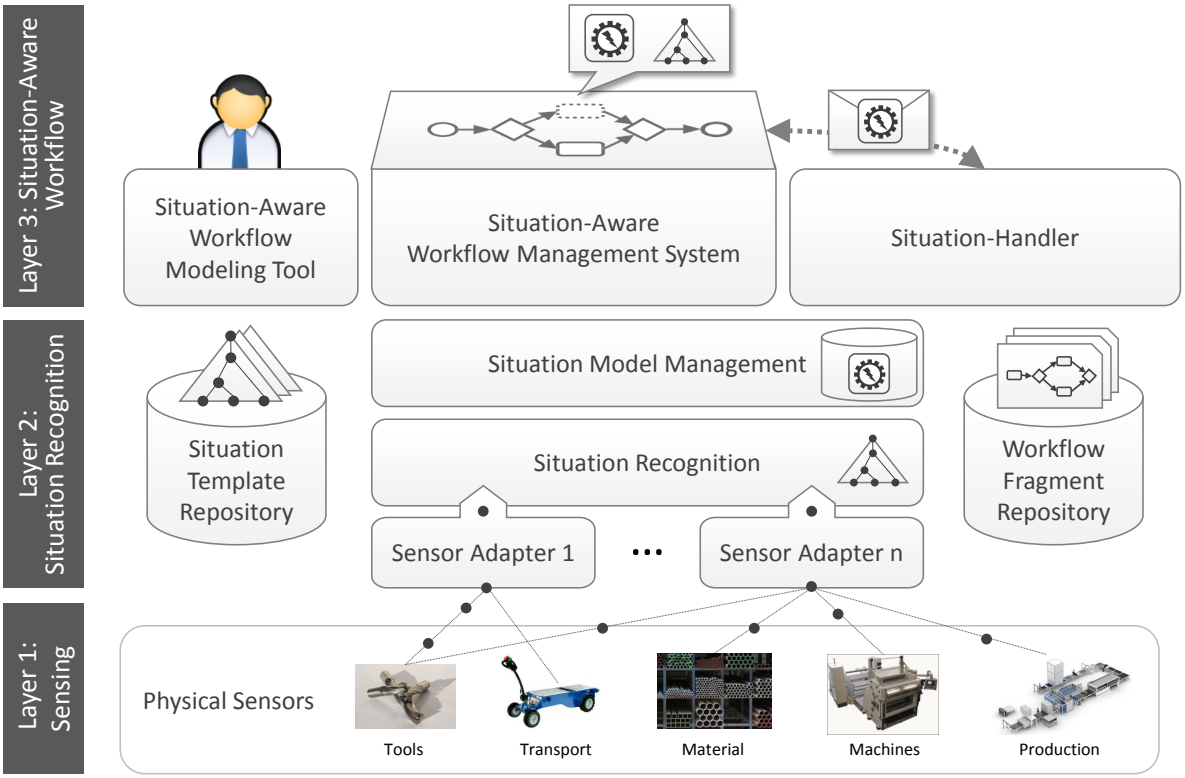

Abbildung 2.1: Die SitOPT - Softwarearchitektur [24]

Dieses Kapitel beschreibt das SitOPT-Forschungsprojekt [24], welches am Institut für Parallele und Verteilte Systeme[1] entwickelt und von der Deutschen Forschungsgemeinschaft unterstützt wird.

Das Projekt SitOPT [24] befasst sich mit der automatischen Anpassung von situationsbezogenen Anwendungen in dynamischen Umgebungen, wie beispielsweise in SMART-Homes oder SMART-Cities. In dem Projekt werden Konzepte und Methoden entwickelt, die es erlauben, fragmentbasierte Workflowmodellierung zu erweitern, um situationsbezogene Anpassungen

[1] https://www.ipvs.uni-stuttgart.de

dieser Anwendungen zu ermöglichen. Dabei muss eine effiziente Erkennung von Situationen basierend auf Sensoren und Kontextinformationen erfolgen, um eine dynamische Adaptierung zu realisieren. Die in Abbildung 2.1 dargestellte „Sensing"-Schicht dient der Bereitstellung der Sensordaten und Kontextinformationen. Diese Schicht wird als Konzept und Implementierung in dieser Diplomarbeit entwickelt. In der „Situation Recognition"-Schicht erfolgt die Situationserkennung und in der „Situation-Aware-Workflow"-Schicht die Workflowmodellierung sowie die Anpassung. Nähere Informationen zum SitOPT-Projekt finden sich in [24].

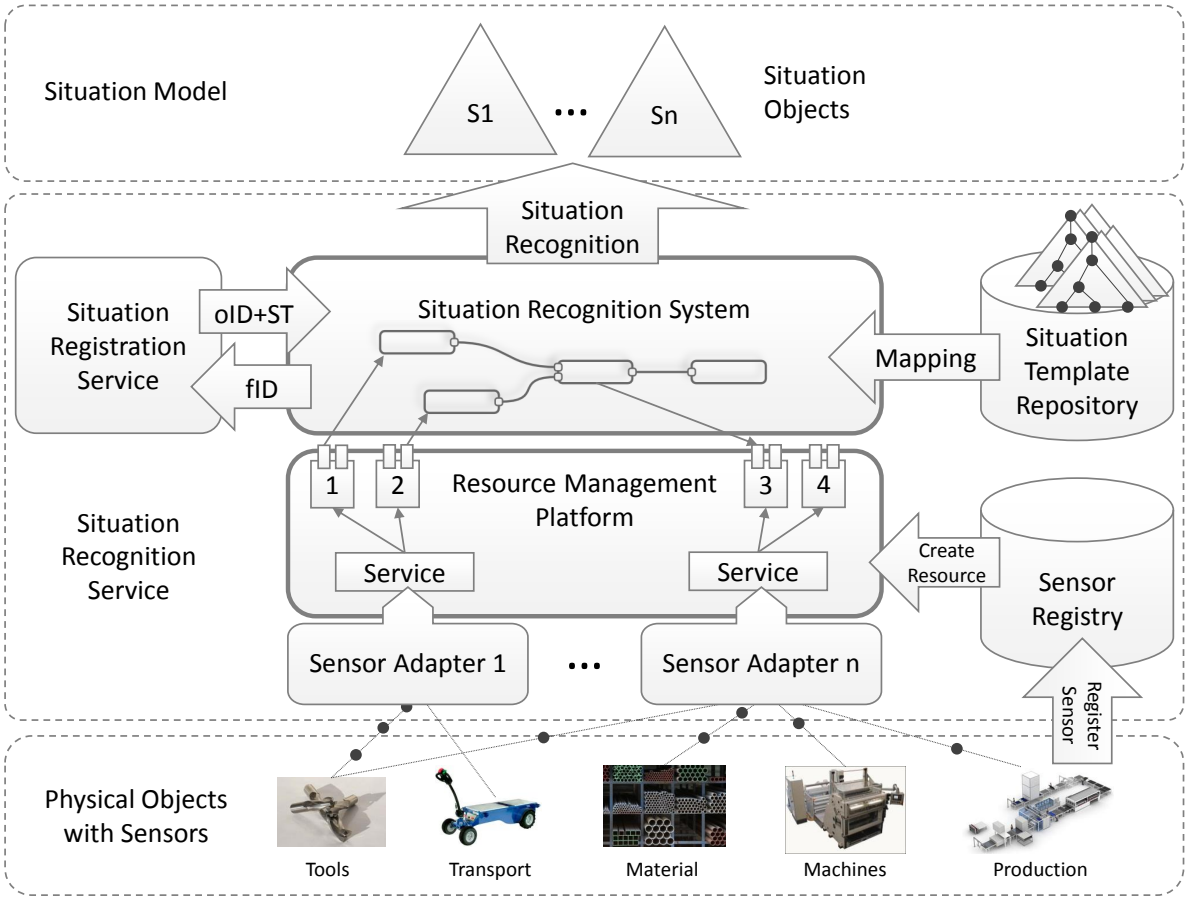

Abbildung 2.2: Die SitRS - Softwarearchitektur [12]

Das SitOPT-Konzept [24] beruht auf der Aufteilung der Aufgaben in (i) Workflowmodellierung, (ii) Workflowanpassung und (iii) Situationserkennung. Bei der Workflowmodellierung und -anpassung werden auf der einen Seite standardisierte Arbeitsabläufe modelliert, diese spiegeln eine Standardsituation, beispielsweise einen Produktionsablauf, wider. Auf der anderen Seite werden alle möglichen Abweichungen und Fehler die auftreten können, als Sub-Workflows modelliert. Diese werden dann zu den jeweiligen Standard-Workflows zuge-

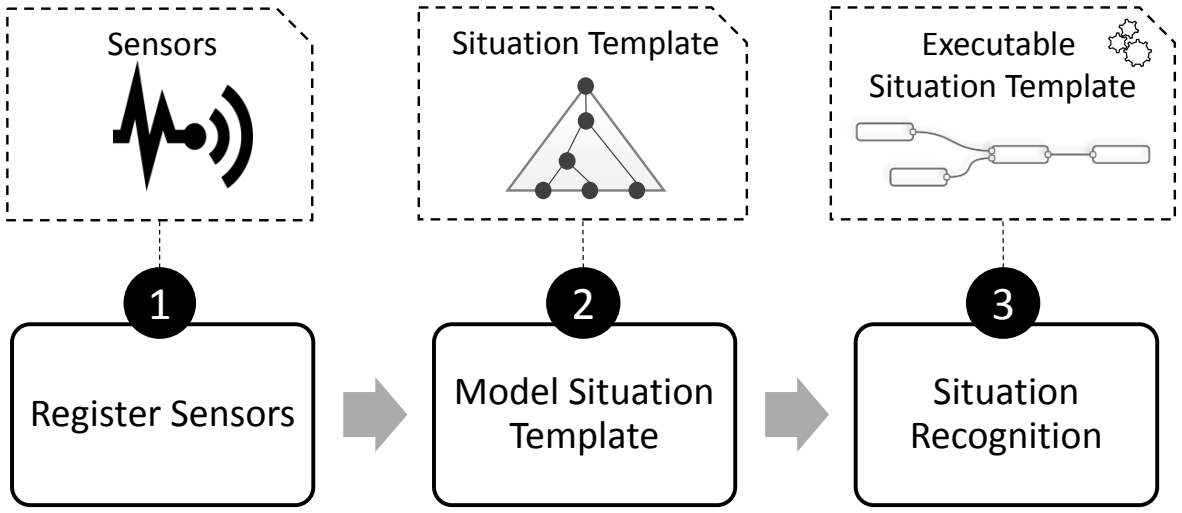

Abbildung 2.3: SitRS - Methoden für die Situationserkennung [12]

ordnet in denen eine Abweichungen auftreten kann. Tritt beispielsweise ein Fehler in einem Produktionsablauf auf, werden statt dem Standardablauf die Sub-Workflows ausgeführt. Für die Selektion des entsprechenden Sub-Workflows wird eine automatische Situationserkennung vorausgesetzt. Diese erhält, wie in Abbildung 2.2 gezeigt, die Sensordaten aus der, in dieser Diplomarbeit beschriebenen Resource-Management-Platform. Diese erstellt aus den Sensordaten OSLC-basierte REST-Ressourcen, die über eine Webschnittstelle an das Situationserkennungs-System (SitRS [12]) gesendet werden.

Das Situationserkennungs-System, als Teil des SitOPT-Projekts, erkennt basierend auf diesen Sensordaten und weiteren Kontextinformationen eine vorliegende Situation. Diese Situationerkennung wird mit sogenannten „Situation Templates (ST)“ modelliert. Ein Situation Template ist ein gerichteter Graph, dessen Endknoten die Sensoren und Kontextinformationen darstellen. Diese werden anschließend von den Elternknoten, die die Verarbeitungslogik darstellen, verarbeitet. Der Wurzelknoten repräsentiert am Ende eine erkannte Situation. Die Templates werden, wie in Abbildung 2.2 dargestellt, in einem Situation Template Repository gespeichert. Diese werden zu den jeweiligen Objekten zugeordnet und ausgeführt. Mit dem Registrierungsservice werden die Situationen basierend auf den Situation Templates registriert.

Die Schritte für eine automatische Situationserkennung sind, wie in Abbildung 2.3 dargestellt, (1) die Registrierung von Sensordaten, welche den Hauptteil dieser Diplomarbeit ausmacht, (2) die Modellierung von Situationstemplates und (3) die anschließende Situationserkennung.

3 Grundlagen

Dieses Kapitel behandelt die technischen Grundlagen, die für das Verständnis dieser Diplomarbeit wichtig sind. Weiterführende Informationen sind aus den jeweiligen Quellenangaben zu entnehmen.

3.1 Internet of Things

Der Begriff „Internet of Things“ wurde das erste Mal von Kevin Ashton im Jahr 1999 im Zuge der Auto-ID Labs-Technologie [1] geprägt. Diese Technologie beschreibt die automatische und berührungslose Identifikation und Lokalisierung von Objekten und Lebewesen mittels Radiowellen. Heute versteht man unter dem Begriff die Verknüpfung und Interaktion von physischen Objekten jeglicher Art mit dem Internet [9]. Dabei werden die Objekte im Internet durch eine virtuelle Repräsentation dargestellt. Somit soll jedes Objekt, welches in der realen Welt einen bestimmten Zustand besitzt, diese Zustandsinformationen auch im Internet bereitstellen. Die Objekte können untereinander kommunizieren und Informationen austauschen. Auf Basis von Sensordaten kann der Zustand der Objekte bestimmt werden. Damit wird eine Überwachung sowie automatisierte Steuerung ermöglicht und die Objekte können selbständig Aufgaben übernehmen, die weit über die Informationsbereitstellung hinaus gehen. Beispielsweise kann ein Drucker, dessen Füllstand einen festgelegten Wert unterschreitet, automatisch eine neue Druckerpatrone nachbestellen, ohne dass dessen Besitzer eine Aktion ausführen muss.

3.2 REST

Dieser Abschnitt gibt einen allgemeinen Überblick über „Representational State Transfer (REST)“. In der Dissertation von Roy Fielding [8] wird REST als ein Architekturstil beschrieben, der schwerpunktmäßig die Maschinen-Maschinen-Kommunikation (M2M) charakterisiert. Überwiegend wird dieser Architekturstil für das World Wide Web angewandt. REST folgt keiner konkreten Implementierung mit eigener Syntax, sondern ist viel mehr ein Programmierparadigma. So gibt REST lediglich Richtlinien für eine zustandslose Übertragung

von Daten vor. Als Übertragungsprotokolle werden hauptsächlich HTTP und HTTPS eingesetzt. Theoretisch können auch andere Implementierungen dieser Architektur existieren, dies ist aber für diese Diplomarbeit nicht relevant und wird nicht näher beschrieben.

REST fordert weiterhin, dass eine bestimmte Web-Adresse – die URI –, genau einen Seiteninhalt, respektive genau eine Ressource im Internet darstellt. Ressourcen sind das zentrale Konzept in REST. Alles was eindeutig indentifizierbar ist, stellt eine Ressource dar. Dies kann zum Beispiel eine Bestellung, ein Produkt, ein Geburtstagstermin, ein Service oder eine intelligente Umgebung im Internet darstellen. Dabei kann eine einzelne Ressource mehrere unterschiedliche Repräsentationen besitzen, die durch vordefinierte Formate (XML, JSON, HTML, etc.) festgelegt sind. Unterschiedliche Klienten können so jeweils das Format anfordern, das am besten ihren Bedürfnissen entspricht. So kann ein Endanweder mit einem Browser eine HTML-Darstellung anfordern, ein progammatischer Client wiederum eine JSON-oder XML-Repräsentation. Der Vorteil der eindeutig indentifizierten Ressourcen besteht hauptsächlich darin, dass sich die URI nicht ändert und man diese zum Beispiel als Favorit im Browser speichern oder an andere Personen versenden kann. Ein weiterer Vorteil gegenüber den ähnlichen Verfahren wie SOAP [6] und WSDL [5] besteht darin, dass REST als Webservice-Implementierung die Methodeninformationen nicht in der URI kodiert. Die URI gibt nur den Ort und den Namen der Ressource an, nicht aber die Funktionalität die dahinter steckt. REST gibt weiterhin vor, dass jede Ressource die gleichen Operationen unterstützen muss. Im HTTP-Protokoll wird diese standardisierte Schnittstelle zu den bekannten Methoden GET, PUT, POST und DELETE konkretisiert. Da das World Wide Web bereits einen Großteil der Technologie für das REST-Framework bereitstellt, müssen keine neuen Spezifikationen erstellt werden. Die Anzahl der möglichen Operationen ist durch die gemeinsame Schnittstelle und das HTTP-Protokoll beschränkt. Diese REST-Grundprinzipien sowie die Operationen werden im folgenden Abschnitt näher beschrieben.

3.2.1 REST Grundprinzipen

Ein REST-basierter Webservice ist dank einer Vielzahl an Entwicklungsumgebungen und vorgefertigten Frameworks leicht erstellt. Damit dieser jedoch das HTTP-Protokoll so verwendet, wie es dem REST-Architekturstil entspricht, müssen einige Grundprinzipien eingehalten werden.

Die folgenden vier Eigenschaften muss ein REST-Dienst in jedem Fall besitzen:

- **Adressierbarkeit**
  Jede REST-Ressource muss über eine eindeutige URI erreichbar sein. Zudem muss jeder REST-konforme Webservice über eine eindeutige Adresse, genauer gesagt, über eine

eindeutige URL[1] aufrufbar sein. Diese Adressierbarkeit ermöglicht einen konsistenten Zugriff auf einen vom Webservice angebotenen Dienst oder auf eine Ressource.

- **Unterschiedliche Repräsentationen**
  Ein vom Webservice angebotener Dienst, der unter einer bestimmten Adresse erreichbar ist, kann verschiedene Repräsentationen einer Ressource für eine Anfrage zurückliefern. Je nach Anwendung kann die Darstellung beispielsweise in XML, JSON oder HTML erfolgen.

- **Zustandslosigkeit**
  Die Kommunikation mit einer Ressource muss zustandslos geschehen. Dies bedeutet, dass bei der verarbeitenden Ressource keine Informationen über vorangegangene Aktionen implizit vorhanden sind. So ist ein Client etwa nicht davon abhängig, dass der Server den Zustand der Kommunikation intern speichert. Vielmehr werden bei jeder Anfrage alle zur Ausführung einer Operation benötigten Informationen mitgeschickt. Somit ist jede Anfrage des Clients an den Server in sich geschlossen. Dies bedeutet nicht, dass die eigentliche Anwendung keinen Zustand halten kann. Nur die Kommunikation selbst ist hiervon betroffen. Dies begünstigt zudem auch die Skalierbarkeit des REST-basierten Dienstes. So können die eingehenden Anfragen im Zuge der Lastenverteilung unkompliziert auf mehrere Server verteilt werden.

- **Operationen**
  REST-basierte Dienste bieten verschiedene Operationen an, um Ressourcen auszuliefern, zu verändern oder zu löschen. Diese Operationen werden durch eine gemeinsame Schnittstelle standardisiert. Mit dem HTTP-Protokoll werden die Operationen auf einige wenige reduziert. So definiert **GET** einen lesenden Zugriff auf eine Ressource. Diese Operation wird auch als „sicher“ bezeichnet, da die wiederholte Anfrage den Zustand der Ressource nicht verändert beziehungsweise nicht verändern darf. **PUT** aktualisiert eine bestehende Ressource. Diese Operation wird als „idempotent“ bezeichnet, da die wiederholte Ausführung der Operation stets das selbe Ergebnis zurückliefert. Die Ressource wird dementsprechend nur einmal verändert. Zu den idempotenten Operationen gehört auch die löschende Operation **DELETE**. Die **POST**-Operation legt eine neue Ressource unter einer bestimmten URI an. Diese Operation ist weder sicher noch idempotent, da die wiederholte Ausführung mehrere dieser Ressourcen anlegen kann.

Folgende Prinzipien werden in einigen Literaturen von Autoren nicht oder nur teilweise erwähnt, sie sollten aber trotzdem eingehalten werden:

- **Hypermedia / Linking**
  Diese Eigenschaft sagt aus, dass neben den Daten selbst auch Metadaten mitgesendet

[1] Uniform Resource Locator

werden (Hypermedia). Diese Metadaten steuern den Zustand des Clients (HATEOAS[2]). Ein bekanntestes Beispiel für solche Metadaten sind Links auf andere Webseiten. Diese verknüpfen unterschiedliche Ressourcen, welche unter Umständen auch auf unterschiedlichen Servern liegen, miteinander. Die Verbindung geschieht über Zusatzinformationen in den jeweiligen Repräsentationen einer Ressource. Sie werden meist über XML-Strukturen definiert.

- **Cacheable**
  Dieses Prinzip beschreibt die implizite oder explizite Deklaration eines Web-Dokuments (der Antwort eines Webservers) als zwischenspeicherbar. Durch eine solche eindeutige Kennzeichnung wird verhindert, dass ungültige Antworten weiterbenutzt werden (zum Beispiel weil eine Antwort gespeichert wurde, obwohl das nicht zulässig war). Ferner können durch zwischengespeicherte Elemente die Skalierbarkeit und Performanz gesteigert werden.
- **Layered System**
  Bei Webanwendungen, die aus mehreren Schichten bestehen, geschieht die Kommunikation transparent für die jeweiligen Endpunkte. Dies bedeutet, dass etwa ein Client nicht bestimmen kann, ob er eine direkte Verbindung zu dem Server hat oder ob die Kommunikation über Zwischenknoten geleitet wird. Dadurch wird der Einsatz von Zwischenservern ermöglicht, welche der Skalierbarkeit zuträglich sind, etwa indem sie eine angemessene Lastenverteilung bereitstellen. Ferner können Sicherheitsrichtlinien zum Beispiel durch eine Firewall leichter umgesetzt werden. Dieses Prinzip geht mit der Zustandslosigkeit einher.
- **Code on Demand**
  Mit dem optionalen Prinzip des „Code on Demand“ bezeichnet man die Erweiterung der Clients um Funktionalitäten, die durch die Übermittlung von ausführbarem Code erreicht wurden. Ein synonymer Begriff ist das „client-side scripting“.

3.3 Resource Description Framework und Web Ontology Language

In diesem Kapitel wird ein allgemeiner Überblick über das Resource Description Framework [14] (RDF) und die damit zusammenhängende Web Ontology Language [17] (OWL) gegeben. Aufgrund der Komplexität und der vielen Begrifflichkeiten in RDF und OWL, beschränkt sich dieses Kapitel auf die für die Diplomarbeit relevanten Themen und Begriffe.

[2]Hypermedia as the Engine of Application State

3.3.1 Resource Descripton Framework

Das Resource Description Framework ist ein Standard des World Wide Web Consortium[3] (W3C) und wurde ursprünglich für die Beschreibung von Metadaten konzipiert. Mittlerweile ist RDF ein wichtiger Bestandteil des Semantischen Webs [21] und beschreibt als Teil der Aussagenlogik die technische Vorgehensweise zur Formulierung von Ressourcen im Internet. Der Hauptaspekt liegt jedoch nicht auf der Lesbarkeit und Verständlichkeit für Menschen, sondern der von Computern. RDF-Datenmodelle können mit Hilfe von XML beschrieben und implementiert werden. XML ist ein sehr weit verbreitetes Datenformat, welches unabhängig von Plattform oder Sprache zum Austausch von Informationen verwendet werden kann. XML nutzt aber keine semantische Eingrenzung dieser Dokumente.

Das RDF-Modell ist ein Datenmodell mit wohldefinierter Semantik, das auf einem gerichteten mathematischen Graphen *(G= V, E)* basiert. Jede Aussage im RDF-Modell besteht aus einem Subjekt, Prädikat und Objekt, wobei eine Ressource (das Subjekt) über eine Eigenschaft (das Prädikat - muss eine weitere Ressource sein) mit einem Objekt (kann eine Ressource sein oder auch nur ein einziges Literal) in Verbindung steht. Dabei bildet die Menge der Triple einen mathematischen Graphen. Dieses Format kann programmatisch verarbeitet werden, um Beziehungen von Objekten zueinander zu erkennen sowie eigenständiges Schlussfolgern zu ermöglichen.

Im folgenden Listing 3.1 wird ein einfacher Aufbau eines RDF-Modells in XML-Darstellung gezeigt und anschließend erläutert. Die Syntax eines XML-Dokuments ist nicht relevant für diese Diplomarbeit. Weiterführende Informationen zum Aufbau und zur Syntax von RDF/XML sind aus den jeweiligen Quellen zu entnehmen.

Listing 3.1 Beispiel eines RDF-Modell

```
1  <?xml version="1.0"?>

3  <rdf:RDF
4  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5  xmlns:OSLCAdapter="http://www.example.com/OSLCAdapter#">

7  <rdf:Description rdf:about="http://www.example.com">
8    <OSLCAdapter:title>OSLCAdapter</OSLCAdapter:title>
9    <OSLCAdapter:type>Adapter</OSLCAdapter:type>
10 </rdf:Description>

12 </rdf:RDF>
```

Der Kopfbereich der XML-Datei gibt mit dem RDF-Tag `<rdf:RDF>` explizit an, dass es sich um ein RDF/XML-Dokument handelt. Die festgelegten Vokabulare, einschließlich Kommentar, Angabe der Signatur bei Prädikatressourcen, Angabe der Oberklasse bei Subjektressourcen für RDF finden sich in dem zugehörigen Namensraum. Des Weiteren muss auch der Namensraum

[3] http://www.w3.org

für selbst definierte Vokabulare angegeben werden. Dies geschieht beispielsweise in Zeile 5. Das „Description“ Element enthält die eigentliche Beschreibung der Ressource, die mit dem Attribut „about“ deklariert wird. Die Elemente `title` und `type` sind selbstdefinierte Vokabulare und beschreiben die Eigenschaften der Ressource.

Somit stellt das folgende Triple:
`OSLCAdapter isTitle` von `http://www.example.com/OSLCAdapter#` einen gerichteten Graphen dar, der in Abbildung 3.1 beispielhaft dargestellt ist.

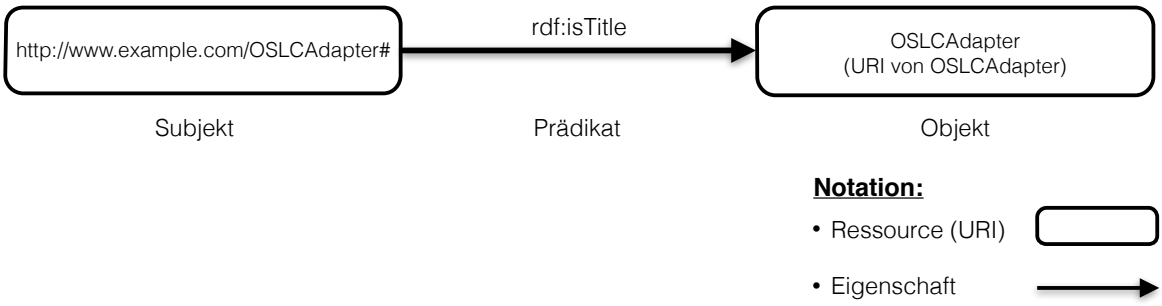

Abbildung 3.1: Einfaches RDF-Modell als Graph

3.3.2 RDF-Schema

Das RDF-Schema (RDFS) definiert das Basisvokabular für die Beschreibung der Eigenschaften und Klassen von RDF-Ressourcen mit einer Semantik für verallgemeinernde Hierarchien solcher Eigenschaften und Klassen. Das Schema beschreibt:

- Klassen
- Eigenschaften
- Hierarchie
- Sammlung
- Versachlichung
- Folgebeziehung

Dabei ist zu beachten, dass das RDF-Schema keine anwendungsspezifischen Klassen oder Eigenschaften definiert, stattdessen wird der Rahmen dafür geboten. Die Klassen ähneln den Klassen von objektorientierten Programmiersprachen. Auf diese Weise können Ressourcen als Instanzen von Klassen und als Unterklassen von Klassen definiert werden.

So ist zum Beispiel, wie in Abbildung 3.2 dargestellt, die RDFS-Klasse **OSLCAdapter** eine Unterklasse der Klasse **Adapter**.

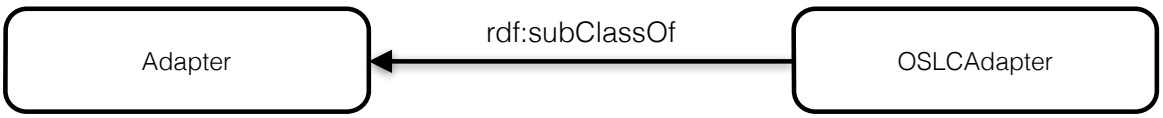

Abbildung 3.2: Beispiel eines einfachen RDFS-Modells als Graph

Im folgenden Listing 3.2 ist der Aufbau dieses RDFS-Modells in XML-Notation dargestellt:

Listing 3.2 Beispiel eines RDFS-Modells

```
1  <rdfs:Class rdf:ID="URI#Adapter">
2      <rdfs:subClassOf rdf:ressource="URI#OSLCAdapter"/>
3  </rdfs:Class>
```

Das RDF-Schema bietet nur einen Bruchteil dessen, was für den Aufbau des Semantischen Webs erforderlich ist. Deshalb wird die Web Ontology Language eingeführt, die in Kapitel 3.3.3 näher beschrieben wird.

3.3.3 Web Ontology Language

Die Web Ontology Language (OWL) baut auf dem RDF-Modell auf. Mit einem erweiterten Vokabular in Verbindung mit formaler Semantik erleichtert OWL die Interpretationsmöglichkeiten von Webinhalten als dies RDF und RDFS ermöglichen. Unter anderem ist es möglich Relationen zwischen Klassen zu definieren sowie eine Kardinalität und Äquivalenzen zwischen diesen anzugeben. Des Weiteren werden mehr Eigenschaftstypen und Charakteristika von Eigenschaften, wie beispielsweise Symmetrie, definiert als bei RDF-Schema. In OWL gibt es drei Untersprachen mit aufsteigender Ausdrucksstärke. Sie sind für verschiedene Anwendungsgruppen konzipiert. Diese werden im Folgenden näher beschrieben.

- **OWL Lite**
  Diese Untersprache wurde für Anwender konzipiert, die eine einfache Restriktion und Klassenhierarchie benötigen. Die formale Komplexität ist geringer als bei den anderen Untersprachen. Sie erlaubt zum Beispiel nur Kardinalitätsrestriktion mit den Werten 0 oder 1. Dadurch ist es einfacher Werkzeuge für OWL Lite zu entwickeln. Außerdem wird dadurch eine schnelle Migration von Thesauri und anderen Taxonomien ermöglicht.
- **OWL DL**
  Das DL kommt von *Description Logic*, ein Forschungsfeld der Logik, welche auch die formelle Basis von OWL bildet. OWL DL ist für diejenigen Anwender konzipiert, die die höchste Ausdruckstärke mit vollständiger Verarbeitbarkeit benötigen. Dadurch lassen sich alle Entscheidungen vollständig verarbeiten. Die OWL Sprachkonstrukte sind aber an bestimmte Bedingungen geknüpft. So kann zum Beispiel eine Klasse eine Unterklasse vieler Klassen sein, aber sie kann nicht gleichzeitig eine Instanz einer anderen Klasse sein.
- **OWL Full**
  Die OWL Full Sprache ist für Anwender gedacht, die ein Maximum an Ausdruckstärke und die syntaktische Freiheit von RDF benötigen. Allerdings gibt es hierbei keine Garantie der vollständigen Verarbeitbarkeit. So kann in OWL Full eine Klasse gleichzeitig als Sammlung von Individuen behandelt werden und gleichzeitig auch als ein eigenes Individuum. Als „Individuen“ werden in OWL Instanzen von Klassen bezeichnet. Derzeit gibt es jedoch noch keine OWL Full Implementierung.

Im folgenden Listing 3.3 ist der Aufbau eines RDF-Dokuments mit OWL Vokabular in XML-Notation dargestellt. Aufgrund des großen Umfangs wird nur auf die für die Arbeit wichtigen Elemente eingegangen.

Im Kopfbereich werden zusätzlich zum RDF Namensraum, die RDFS- und OWL-Namensräume definiert. Beispielsweise wird in Zeile 5 mit dem „dc:“ Präfix ein bestimmter

Namensraum eingebunden. Dieser referenziert die *Dublin Core Metadata Initiative*[4] *(DCMI)*. Hierbei handelt es sich um eine Sammlung einfacher und standardisierter Elemente, wie zum Beispiel „title“ und „description“, die in dieser Ontologie verwendet werden. Die Zeilen 8 bis 12 sind optional und müssen für eine Ontologie nicht angegeben werden, es erleichtert aber das Verständnis für Anwender. Zudem können hier auch Informationen über die jeweilige Version angegeben werden. Um Dinge in Bezug auf Semantik oder Bedeutung zu klassifizieren, können in OWL Klassen und Unterklassen gebildet werden. In den Zeilen 16 bis 21 wird die Klasse `Adapter Type` instanziiert. Sie stellt die Oberklasse aller Klassen dar und enthält alle Gemeinsamkeiten von Adaptertypen. Im Bezug auf das Semantische Web wird die Oberklasse auch als `Taxonomie` bezeichnet. In den Zeilen 24 bis 32 und 35 bis 43 werden die Unterklassen `OSLC Adapter` und `Sensor Adapter` definiert. Sie erben durch die Angabe `subClassOF` alle Eigenschaften ihrer Oberklasse. Abbildung 3.3 stellt diese Taxonomie-Hierarchie grafisch dar.

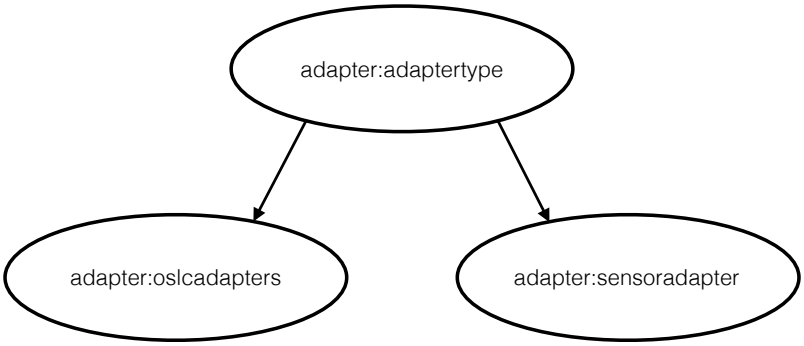

Abbildung 3.3: Beispiel einer Taxonomie Hierarchie

Um ein Beispiel von OWL-Individuen aufzuzeigen, wird in den Zeilen 46 bis 51 ein solches Individuum instanziiert. Der `PC Sensoradapter` ist keine weitere Unterklasse der Klasse `Adapter Types`, sondern eine Instanz der Klasse `Sensor Adapter`. Durch die Vererbungshierarchie erbt die Klasse `PC Sensoradapter` trotzdem alle Eigenschaften der Hauptklasse.

[4] http://dublincore.org

Listing 3.3 Beispiel eines OWL-Dokuments

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:adapters="http://www.example.com/adapters#">

  <!-- OWL Header Example -->
  <owl:Ontology rdf:about="http://www.example.com/adapters">
    <dc:title>The Example Adapter Ontology</dc:title>
    <dc:description>An example ontology for adapters</dc:description>
  </owl:Ontology>


  <!-- OWL Class Definition - Adapter Type -->
  <owl:Class rdf:about="http://www.example.com/adapters#adaptertype">

    <rdfs:label>The adapter type</rdfs:label>
    <rdfs:comment>The class of all adapter types.</rdfs:comment>

  </owl:Class>

  <!-- OWL Subclass Definition - OSLC Adapter -->
  <owl:Class rdf:about="http://www.example.com/adapters#oslcadapters">

    <!-- OCLC Adapter is a subclassification of adaptertype -->
    <rdfs:subClassOf rdf:resource="http://www.example.com/adapters#adaptertype"/>

    <rdfs:label>OSLC Adapter</rdfs:label>
    <rdfs:comment>An OSLC Adapter to receive and store sensordata to database</rdfs:comment>

  </owl:Class>

  <!-- OWL Subclass Definition - Sensor Adapter -->
  <owl:Class rdf:about="http://www.example.com/adapters#sensoradapter">

    <!-- Sensor Adapter is a subclassification of adaptertype -->
    <rdfs:subClassOf rdf:resource="http://www.example.com/adapters#adaptertype"/>

    <rdfs:label>Sensor Adapter</rdfs:label>
    <rdfs:comment>An Adapter for Sensordata</rdfs:comment>

  </owl:Class>

  <!-- Individual (Instance) Example RDF Statement -->
  <rdf:Description rdf:about="http://www.example.com/adapters#pcsensoradapter">

    <!-- PC Sensor Adapter is a type (instance) of the Sensor Adapter classification -->
    <rdf:type rdf:resource="http://www.example.com/adapters#sensoradapters"/>

  </rdf:Description>

</rdf:RDF>
```

3.4 Linked Data

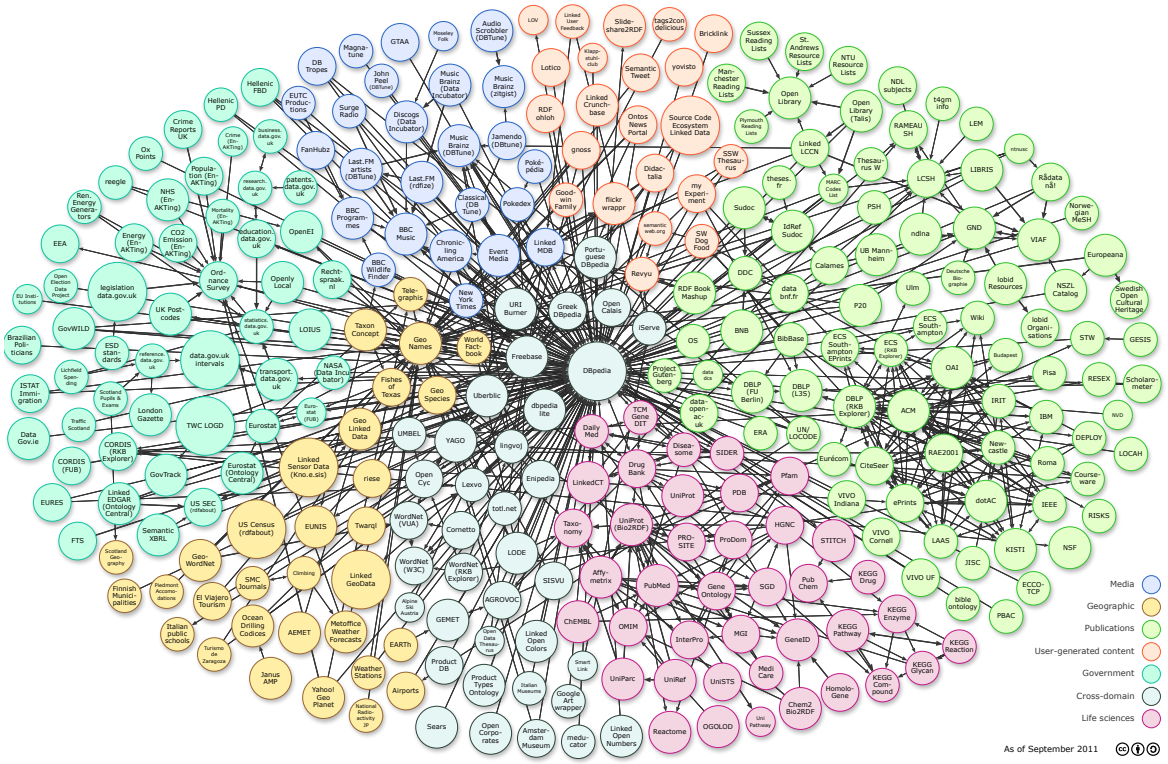

Abbildung 3.4: Verknüpfung zwischen Linked Open Data Datenbeständen (Sept. 2011)
Quelle: http://lod-cloud.net/versions/2011-09-19/lod-cloud_colored.png

Linked Data eröffnet durch die Vernetzung von Daten über das Internet neue Möglichkeiten, diese über Domänen und Organisationsgrenzen hinweg, zu nutzen. Durch die Nutzung von Semantic Web Technologien (RDF, RDFS und OWL), wie sie in Kapitel 3.3 beschrieben wurden, ist es Anwendungen möglich, diese Daten abzufragen, zu verarbeiten und Schlussfolgerungen zu ziehen. Um dies zu erreichen, müssen alle Daten, egal ob es sich um einen Sensorwert, einen Termin, eine Teilenummer oder um Wetterdaten handelt, mittels einer eindeutigen URI identifiziert werden. Diese Daten selbst können ebenfalls per URI auf andere Daten verweisen. Diese Verlinkung entspricht einer Eigenschaft in RDF (siehe Kapitel 3.3.1). Das Konzept von Linked Open Data geht im Wesentlichen auf Tim Berners Lee zurück, der auch Ende 2007 die Bezeichnung „Giant Global Graph“ [3] erwähnte. Er prägte vier Grundregeln für Linked Data:

1. Verwende zur Bezeichnung von Objekten URIs.
2. Verwende HTTP URIs, so dass sich die Bezeichnungen nachschlagen lassen.
3. Stelle zweckdienliche Informationen bereit, sobald jemand eine URI nachschlagen möchte (mit Hilfe des RDF Standards).
4. Zu diesen Informationen gehören insbesondere Links auf andere URIs, über die weitere Objekte entdeckt werden können.

Abbildung 3.4 zeigt die Darstellung einer Verknüpfung zwischen Linked Open Data und den Datenbeständen von DBpedia[5] und GeoNames[6].

3.5 Open Services for Lifecycle Collaboration

Die Open Services for Lifecycle Collaboration (OSLC) [10] ist eine offene Gemeinschaft und seit 2013 Mitglied von OASIS[7]. Diese Gemeinschaft beschäftigt sich mit einer Spezifikation, die eine einfache Integration von Werkzeugen für die Softwareentwicklung durch Standardisierungen ermöglicht. Dies wird in Abbildung 3.5 beispielhaft dargestellt.

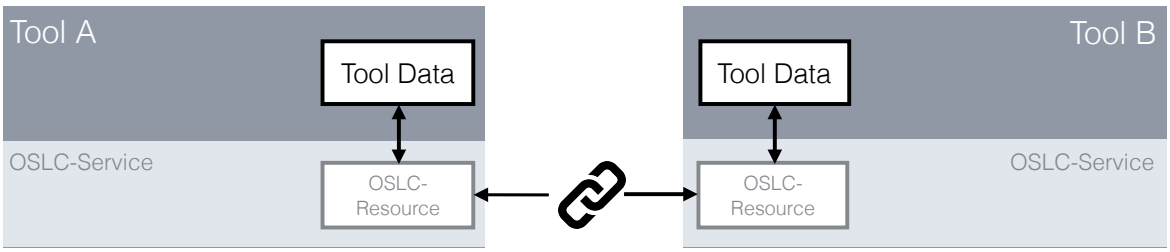

Abbildung 3.5: OSLC Beispielintegration mit zwei Werkzeugen

Die OSLC-Gemeinschaft beschäftigt sich hauptsächlich mit Themen aus dem Bereich des Software-Lebenszyklus [20]. Darunter gehören beispielsweise „Fallbearbeitungssysteme (Bugtracker)“, „Application-Lifecycle-Management (ALM)“-, „Product-Lifecyle-Management (PLM)“- und „Software-Test-Management“-Systeme. Diese Themengebiete werden von OSLC als „**domains**“ bezeichnet und sind in verschiedene Arbeitsgruppen unterteilt. Jede dieser Arbeitsgruppen untersucht für sich die jeweiligen Integrationsszenarien und erstellt daraus ein gemeinsames Vokabular für Artefakte, die erforderlich sind, um die Integration zu unterstützen.

[5]http://de.dbpedia.org
[6]http://www.geonames.org/
[7]Organization for the Advancement of Structured Information Standards

Darüber hinaus hat die OSLC-Kernarbeitsgruppe die „OSLC-Core-Spezifikation“ herausgebracht, die eine Kohärenz zwischen den verschiedenen „domains“ gewährleistet. Jede Spezifikation muss sich an die Integrationstechniken dieser Kernspezifikation halten. Überwiegend besteht diese Spezifikation aus Standardregeln und Techniken für die Verwendung des HTTP-Protokolls in Zusammenhang mit RDF und Linked Data. Die Kernspezifikation kann nicht allein genutzt werden, vielmehr wird sie in Verbindung mit der für die jeweilige **Domäne** existierenden Spezifikation verwendet.

OSLC-Spezifikationen bauen auf REST und Linked Data auf, die in Kapitel 3.2 (REST) und Kapitel 3.4 (Linked Data) beschrieben werden. So stellt jedes Artefakt in OSLC eine HTTP-Ressource dar und kann mit den HTTP-Methoden **GET**, **PUT**, **POST** und **DELETE** abgefragt und manipuliert werden. Ein Artefakt kann zum Beispiel eine Quellcodedatei, ein Testfall oder sonstige Software darstellen. Wird die dritte Grundregel aus Linked Data herangezogen *(Stelle zweckdienliche Informationen bereit, sobald jemand eine URI nachschlagen möchte)*, so ist jede HTTP-Ressource auch eine RDF-Ressource, die als RDF/XML repräsentiert ist. Andere Repräsentation wie zum Beispiel JSON sind auch möglich. OSLC beschreibt zwei Integrationstechniken, mit denen Software-Werkzeuge verbunden werden können. Diese werden im nächsten Abschnitt beschrieben.

3.5.1 OSLC Integrationstechniken

OSLC bietet zwei Integrationstechniken an, um Software-Werkzeuge miteinander zu verbinden. Zum einen die *programmatische Verlinkung von Daten (über HTTP)* und zum anderen die *Verlinkung von Daten über eine webbasierte Benutzeroberfläche.*

- **Verlinkung von Daten über HTTP**
  Diese Technik beschreibt die programmatische Verlinkung zwischen Software-Werkzeugen. OSLC spezifiziert ein gemeinsames Protokoll, welches Anwendungen erlaubt, Lifecycle-Daten über das HTTP-Protokoll zu erstellen, abzurufen, zu aktualisieren und zu löschen. Dieses Protokoll kann von jeder Anwendung oder einem programmatischen Client verwendet werden, um mit einem Werkzeug, welches auch die Spezifikation implementiert hat, zu kommunizieren. Die Bindung wird durch Einbetten der URI in der jeweiligen anderen Ressource erreicht.
- **Verlinkung von Daten über ein webbasierte Benutzeroberfläche**
  Bei dieser Technik spezifiziert OSLC ein Protokoll, welches einem Software-Werkzeug erlaubt, einen Teil der Benutzeroberfläche der jeweils anderen Software zu verwenden. Dies vereinfacht dem Benutzer, neue oder vorhandene Ressourcen miteinander zu verlinken und zu verarbeiten, da die Information über die Ressourcen als Vorschau in einem Fenster dargestellt wird. Dabei wird die URI der Benutzeroberfläche in der jeweiligen anderen Implementierung eingebettet. In vielen Fällen ist dies effizienter und

bietet mehr Benutzerfunktionen als die Umsetzung einer neuen Benutzeroberfläche und die Integration über programmatische Schnittstellen.

Nachfolgend werden der OSLC-Service-Provider sowie der OSLC-Service beschrieben.

3.5.2 OSLC-Service-Provider und OSLC-Service

Die zentralen Konzepte in OSLC stellen der OSLC-Service-Provider und der OSLC-Service dar. In den Service-Providern werden alle Artefakte wie beispielsweise Testfälle und Fehler aus Bugtrackingtools zusammengefasst. Die OSLC-Services spezifizieren Eigenschaften für die Erstellung dieser Artefakte als Ressourcen. Jeder OSLC-Service gehört zu einem Service-Provider. Die Service-Provider legen URIs fest, unter denen Ressourcen mittels **POST** erstellt und mittels **GET** eine Liste der existierenden Ressourcen angefordert werden können.

OSLC-Service-Provider können wie folgt charakterisiert werden:

- Alle OSLC-Ressourcen gehören zu einem oder mehreren Service-Providern. Mit der optionalen Eigenschaft `<oslc:serviceProvider>` kann explizit der zuständige Service-Provider angegeben werden.
- Clients können eine Liste existierender Service-Provider über den optionalen *Service-Provider-Catalog* anfordern.
- Der einzige Weg der in OSLC definiert ist, um eine OSLC-Ressource zu erstellen, ist sie in einem OSLC-Service zu erstellen, welcher vom OSLC-Service-Provider verwaltet wird.

OSLC-Service-Provider besitzen drei fundamentale Eigenschaften:

- **`<oslc:creation>`**
  Sie gibt die URI an, unter der neue Ressourcen mittels POST erstellt werden können.
- **`<oslc:queryBase>`**
  Sie gibt die URI an, unter der bestehende Ressourcen mittels **GET** angefordert werden können. Es wird eine Liste der existierenden Ressourcen im jeweiligen Service Provider zurückgeliefert.
- **`dialog`**
  Diese Eigenschaft betrifft die zweite Integrationstechnik in OSLC, die in Kapitel 3.5.1 beschrieben wurde. Mit dieser Eigenschaft können für die Erstellung von Ressourcen Dialog-Fenster eingebunden werden, die eine Vorschau der Ressource in der jeweils anderen Anwendung anzeigen.

4 Problembeschreibung

In diesem Abschnitt wird die Aufgabenstellung dieser Diplomarbeit erläutert. Beginnend mit der Problembeschreibung wird auf die Herausforderung dieser Arbeit eingegangen. Anschließend wird die Aufgabenstellung erläutert.

4.1 Problembeschreibung und Herausforderung

Das Internet der Dinge ermöglicht die Überwachung und Steuerung von Objekten über webbasierte Dienste. Hauptsächlich wird dafür auf Sensordaten zurückgegriffen, die physische Objekte durch eine virtuelle Repräsentation im Internet widerspiegeln. Allerdings ist ein Zugriff auf diese heterogenen Sensoren vor allem bei älteren technischen Geräten oft sehr schwierig. Den meisten Objekten fehlt es an einer einheitlichen Schnittstelle, um Sensordaten über das Internet bereitzustellen. Oft besitzen derartige Geräte gar keine Verbindung nach außen. Dies macht die Vernetzung der Daten untereinander teils unmöglich und die Anbindung und Verknüpfung mit anderen Datenquellen ist mit sehr hohem Aufwand verbunden. Heutzutage sind viele Plattformen auf dem Markt, die sich mit diesem Problem befassen. Allerdings ist eine Anpassung an neue Datenquellen oder das Anbinden neuer Sensoren oft schwergewichtig und es fehlt immer noch ein Standard für das Bereitstellen der Sensordaten über eine einheitliche Schnittstelle.

Die Herausforderung dieser Diplomarbeit besteht darin, heterogenen Sensoren zu identifizieren, deren Sensorwerte auszulesen und über das Internet bereitzustellen, welche anschließend über Verknüpfungen untereinander verbunden werden sollen.

4.2 Aufgabenstellung

In dieser Arbeit soll ein Ressourcenbereitstellungs-System entwickelt werden, das mit Hilfe der OSLC-Spezifikation eine einfache Integration von Sensordaten und Services ermöglicht. Die Aufgabe dieser Diplomarbeit besteht darin, heterogene Sensoren aus Objekten zu erfassen und diese mit OSLC-basierten REST-Ressourcen zu provisionieren. Dabei sollen OSLC-Adapter und -Services mit OSLC-Service-Providern entwickelt und anhand eines Anwendungsszenarios implementiert werden. Zudem soll untersucht werden, inwieweit

neue Datenquellen möglichst effizient an diese Plattform angeschlossen werden können. Das Ziel ist es, eine möglichst einfache OSLC-basierte Schnittstelle zu schaffen, um neue Ressourcen einer Informationsquelle zu erstellen bzw. bestehende Ressourcen zu verändern oder zu löschen.

5 Konzeptionelle Lösung

In diesem Kapitel werden die Konzepte möglicher Lösungsansätze der in Kapitel 4 beschriebenen Problemstellung vorgestellt. Das Lösungskonzept wird als Softwarearchitektur beschrieben. Für die Übersicht wird ein vereinfachtes, schematisches Gesamtbild (Abbildung 5.1) aufgezeigt. Dieses ist für das bessere Verständnis in drei Schichten unterteilt. Die Schichten werden im Folgenden kurz erläutert und in den nachfolgenden Kapiteln detailliert beschrieben.

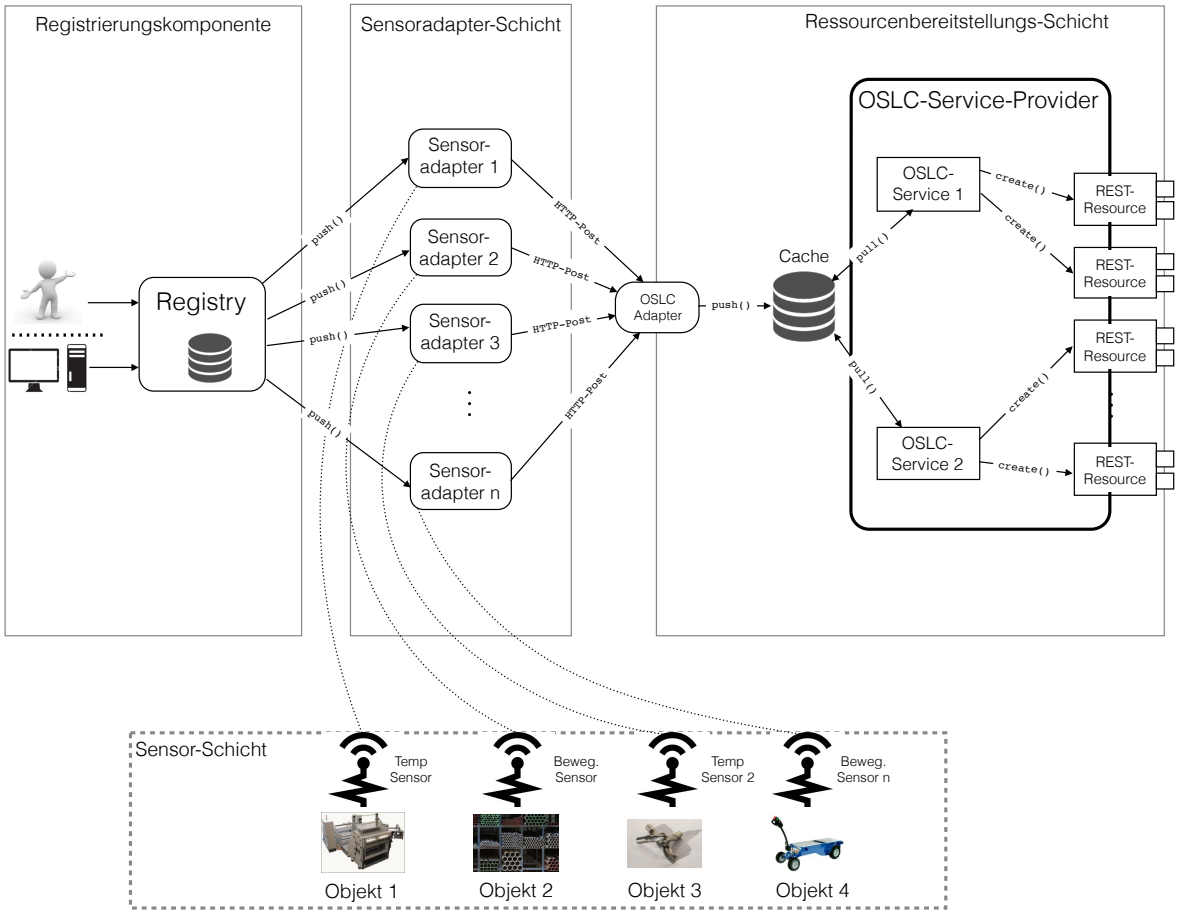

Abbildung 5.1: Konzeptionelle Lösung als Gesamtbild

5.1 Konzeptioneller Überblick

Für einen besseren Überblick wurde das entwickelte Konzept in drei Schichten untergliedert. Dies verdeutlicht die Abbildung 5.1. Die Sensor-Schicht ist nicht Teil dieser Lösung. Diese Schicht demonstriert lediglich die Verbindung zwischen den Sensoren und dem vorgestellten Lösungsansatz. Die auf der linken Seite dargestellte Registrierungskomponente (englisch: Registry) stellt den Einstiegspunkt dieser Lösung dar. Sie ist die einzige Schnittstelle nach außen und bietet eine graphische und programmatische Schnittstelle für die Registrierung, Deregistrierung und Verwaltung von Sensoren. Dabei kommuniziert die Registry mit der Sensoradapter-Schicht. In dieser Schicht sind die Sensoradapter enthalten. Ein Sensoradapter stellt die Verbindung zwischen einem Sensor und der Ressourcenbereitstellungs-Schicht her. Für jeden Sensor muss ein individueller Sensoradapter implementiert werden. Anschließend werden die Sensordaten vom Adapter verarbeitet und an die Ressourcenbereitstellungs-Schicht weitergeleitet. Diese Schicht provisioniert die Sensordaten und erstellt daraus die OSLC-basierten REST-Ressourcen. Diese werden dann für die weitere Verwendung über eine URI im Internet bereitgestellt.

5.2 Registrierungskomponente

Die Registrierungskomponente ist als Einstiegspunkt für die Verwaltung der zu überwachenden Sensoren konzipiert. Sie ist die einzige Schnittstelle, um diese von außen zu registrieren, zu editieren und zu deregistrieren. Sie wird als webbasierte Anwendung implementiert und verwendet zur Speicherung der Sensorinformationen eine relationale Datenbank.

5.2.1 Funktion der Registrierungskomponente

Für die Verwaltung von Sensoren werden folgende Methoden bereitgestellt:

- **Sensoren registrieren**
  Für die Registrierung der Sensoren wird eine Methode aufgerufen, welche die eingegebenen Sensordaten an die Sensoradapter-Schicht weiterleitet und gleichzeitig eine weitere Methode für die Speicherung der eingegebenen Sensordaten in die Registrierungsdatenbank ausführt. Dabei wird überprüft, ob der Sensor bereits registriert ist. Ist dies der Fall, wird ein entsprechender Hinweis angezeigt und der Benutzer kann entweder die Sensordaten ändern oder die Registrierung abbrechen.
- **Sensorinformationen anzeigen**
  Für das Bereitstellen der Sensorinformationen wird eine Methode zur Verfügung

gestellt, die eine Datenbankabfrage über die bereits registrierten Sensordaten ausführt und diese dann als Listenansicht zurückliefert.

- **Sensorinformationen editieren**
  Die Registrierungskomponente bietet eine weitere Methode an, um die Sensorinformationen zu editieren, falls die zuvor eingegebenen Sensorinformationen aktualisiert werden sollen. Dabei wird die Eingabemaske für die Sensorregistrierung als Dialog eingeblendet, in der die Sensorinformationen des jeweiligen Datenbankeintrags bereits enthalten sind. Diese können editiert und anschließend durch eine Update-Funktion in der Datenbank überschrieben werden. Die geänderten Sensorinformationen werden anschließend an die Sensoradapter-Schicht übertragen.
- **Sensoren deregistrieren**
  Soll ein Sensor deregistriert werden, wird eine Methode zum Löschen der Datenbankeinträge implementiert. Diese Methode löscht den entsprechenden Eintrag für den jeweiligen Sensor aus der Datenbank und führt gleichzeitig eine weitere Methode aus, die ein Deregistrierungsereignis an die Sensoradapter-Schicht übermittelt.

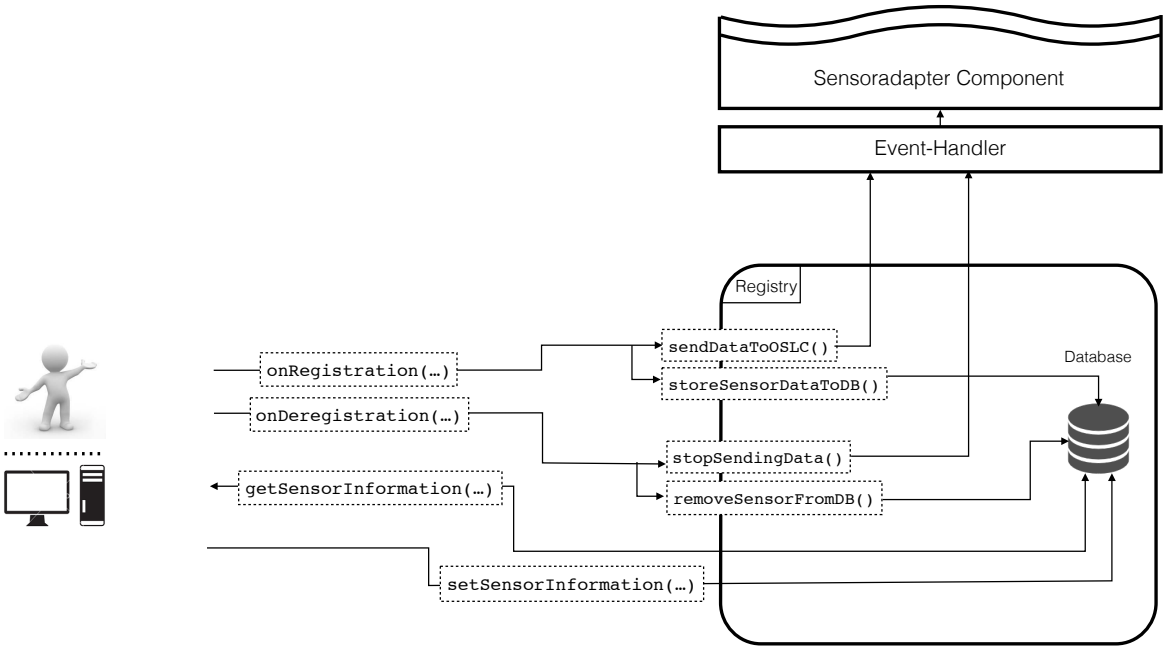

Abbildung 5.2: Sensor Registrierungskomponente

Für die Registrierung, Deregistrierung und Verwaltung der Sensordaten wie in Abbildung 5.2 dargestellt, bietet die Registrierungskomponente zwei verschiedene Verfahren an:

- Eine **benutzergesteuerte Sensorregistrierung** mit Hilfe einer webbasierten grafischen Oberfläche.

- Eine **anwendungsgesteuerte Sensorregistrierung** über eine programmatische Webschnittstelle.

Dadurch ergeben sich zwei Anwendungsfälle, die im Folgenden näher beschrieben werden.

5.2.2 Benutzergesteuerte Sensorregistrierung

Für die Registrierung und Verwaltung der Sensoren durch einen Benutzer wird ein Webformular bereitgestellt. Dem Benutzer werden für die Eingabe der Sensorinformationen unter anderem folgende Eingabefelder zur Verfügung gestellt:

- **ObjectID:**
  In dieses Feld gibt der Benutzer die ID des zu registrierenden Objekts ein. Jedes Objekt muss einen eindeutigen Identifikator besitzen. Eine Objekt-ID kann zum Beispiel der Eintrag „PC_Rechenzentrum“ repräsentieren.
- **SensorType:**
  In dieses Feld wird der zu registrierende Sensor des zuvor genannten Objekts eingetragen. Der Sensortyp kann zum Beispiel vom Typ „Temperatursensor“ sein und zum Objekt „PC_Rechenzentrum“ gehören.
- **Sensoradapter_URI:**
  In dieses Feld wird die URI eingetragen, die den jeweiligen Sensoradapter adressiert.
- **SensorQuality:**
  In dieses Feld wird die Qualität eines Sensors eingetragen. Die Qualität ist gleichbedeutend mit der Genauigkeit der Sensorwerte und kann aus der jeweiligen Sensor-Spezifikation entnommen werden.
- **Frequency:**
  In dieses Feld wird das Intervall der Sensormessung vom Benutzer eingetragen. Jeder Sensor liefert Messwerte in einem bestimmten Intervall. Dadurch lässt sich eine Aussage über die Messwerte in Bezug auf die Genauigkeit machen. Liegt zwischen zwei Messungen ein große Zeitspanne, ist ein Sensorwert für eine Situationserkennung weniger genau als dies bei einer kontinuirlichen Übertragung der Messwerte der Fall ist.
- **DataType:**
  Jeder Sensor kann die Daten in einem bestimmten Datentyp messen und verschicken. Ein Datentyp kann zum Beispiel einem Integer-, Float- oder Doublewert entsprechen. Diese Information ist für den Anwender der Sensordaten wichtig, um entsprechend die Ausgabe der Daten verarbeiten zu können.

- **Geolocation:**
  Für die Standortbestimmung der Sensoren wird die Geolokation in maschinenlesbaren Koordinaten eingetragen. Dadurch lassen sich später Sensoren aus bestimmten Umgebungen anzeigen.

Das Webformular wird in Abbildung 5.3 zur Verdeutlichung beispielhaft dargestellt.

ObjectID	PC_Data-Center
SensorType	CPU
Sensoradapter_URI	http://www.uni-stuttgart.de/rechenzentrum
SensorQuality	93 %
DataFrequenz	0.5 sec
DataType	float
Geolocation	48.7818021,9.172806,535m

Abbildung 5.3: Beispielmaske für ein Registrierungsformular

5.2.3 Anwendungsgesteuerte Sensorregistrierung

Zur Registrierung von Sensoren mittels einer Anwendung wird eine programmatische Webschnittstelle über das HTTP-Protokoll implementiert. Es werden URIs definiert, auf die mittels der vier HTTP-Operatoren zugegriffen werden kann. Für die Registrierung eines Sensors wird mit dem **POST**-Operator ein neuer Sensor in der Datenbank registriert. Gleichzeitig wird ein Registrierungsereignis an den Sensoradapter geschickt. Der **PUT**-Operator aktualisiert einen bereits registrierten Sensor in der Datenbank mit den aktuellen Sensorinformationen. Wird ein Sensor deregistriert, wird dieser mit dem **DELETE**-Operator mit der entsprechenden Angabe der Sensorinformationen aus der Datenbank entfernt und ein Deregistrierungsereignis an den Sensoradapter geschickt. Eine **GET**-Anfrage sendet alle registrierten Sensoren zurück. Die Sensorinformationen werden, wie in Tabelle 5.1 dargestellt, als HTTP-Parameter an die URI der Registrierungskomponente übergeben.

ObjectID	SensorType	Sensoradapter_URI	SensorQuality	Frequency	DataType	Geolocation

Tabelle 5.1: Parameterangaben zur anwendungsgesteuerten Sensorregistrierung

5.2.4 Speichern der Sensorinformationen in der Datenbank

Für die Speicherung der Sensordaten wird eine relationale Datenbank verwendet. Eine derartige Datenbank hat den Vorteil, dass sich durch datenbankspezifische Abfragen Such- und Filterkriterien leicht umsetzen lassen. Die erhöhte Zugriffszeit bei einem großen Datenaufkommen kann hierbei vernachlässigt werden, da es sich nicht um zeitkritische Daten handelt.

Die Sensorinformationen werden in einer Datenbanktabelle gespeichert. Die Spalten der Tabelle lassen sich beliebig erweitern, falls für den jeweiligen Sensor weitere Informationen zur Verfügung stehen und gespeichert werden sollen. Werden Sensorinformationen gespeichert, wird gleichzeitig immer ein aktueller Zeitstempel angefügt. Dadurch lässt sich eine Suche auf einen gewissen Zeitraum eingrenzen sowie Auskunft darüber geben, wann ein Sensor gespeichert wurde. Die Sensordaten können tabellarisch über eine Webschnittstelle im Browser angezeigt werden. Dabei kann der Benutzer über die Editierungsfunktion die gespeicherten Sensorinformationen ändern oder löschen. Die tabellarisch aufgelisteten Sensoren können mit einer Such- und Filteroption für eine sortierte Darstellung entsprechend in der gewünschten Ausgabe angezeigt werden. Möchte der Benutzer beispielsweise einen bestimmten Sensortyp, der zu einer bestimmten Zeit für eine bestimmte Umgebung registriert wurde anzeigen, so kann er dies über die Suchmaske eintragen und die Suche ausführen.

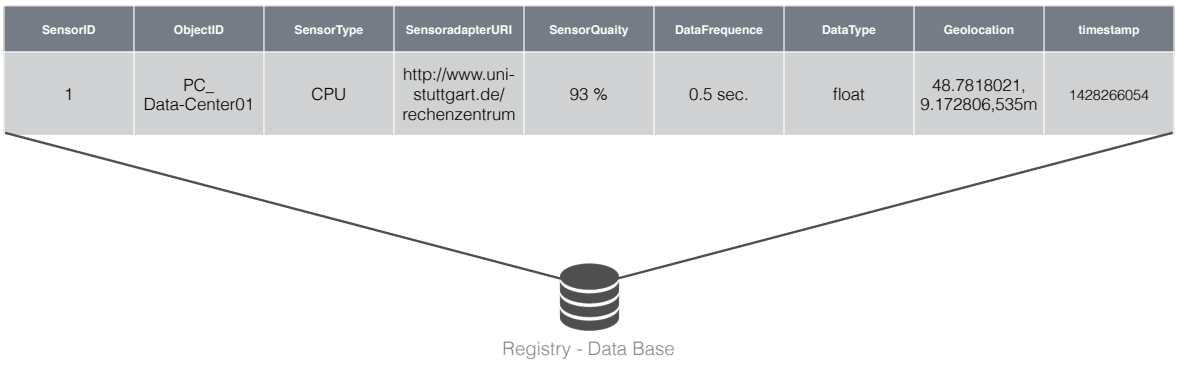

Abbildung 5.4: Registrierungskomponente und deren Datenbankfelder

5.2.5 Anwendungsfälle

Im Folgenden wird die im Lösungskonzept vorgestellte Interaktion der benutzergesteuerten Sensorregistrierung als Anwendungsfälle festgehalten und beschrieben. Ein Anwendungsfall beschreibt die Arbeitsschritte aus Sicht des Benutzers, die nötig sind, um eine Handlung durchzuführen und das beschriebene Ziel zu erreichen. Die Anwendungsfälle werden anhand des implementierten Konzeptes beschrieben. Die in diesem Use-Case beschriebenen Akteure sind die Personen, die über die Webschnittstelle Sensoren registrieren. Aufgrund des Umfangs wird die Ausnahmebehandlung der Sonderfälle in den folgenden Use-Cases nicht beschrieben.

5.2.5.1 Sensor registrieren

Use-Case	**Sensor registrieren**
Ziel	Der Benutzer möchte einen Sensor registrieren.
Vorbedingung	• Der Benutzer hat die Benutzerschnittstelle der Registrierungskomponente geöffnet.
Regulärer Ablauf	1. Der Benutzer wählt die Aktion: „Sensor registrieren“. 2. Der Benutzer gibt unter anderem folgende Daten in die Formularfelder ein: • ObjectID • SensorType • Sensoradapter_URI • SensorQuality • Frequency • DataType • Geolocation 3. Der Benutzer wählt die Aktion: „Sensor speichern“. 4. Die Registrierungskomponente speichert die Sensorinformation in der Datenbank. 5. Die Registrierungskomponente schickt die Daten an den entsprechenden Sensoradapter.
Sonderfälle	4. Der Sensor kann nicht gespeichert werden, da die Verbindung zur Datenbank fehlerhaft ist. 5. Die Registrierungskomponente kann keine Verbindung zum Sensoradapter herstellen.
Nachbedingung	Der Sensor ist in der Datenbank gespeichert und der Sensoradapter verarbeitet die empfangenen Daten.

5.2.5.2 Sensorinformationen anzeigen

Use-Case	Sensorinformationen anzeigen
Ziel	Der Benutzer möchte die registrierten Sensoren anzeigen.
Vorbedingung	• Der Benutzer hat die Registrierungskomponenente geöffnet.
Regulärer Ablauf	1. Der Benutzer wählt die Aktion: „Sensorinformationen anzeigen“. 2. Die Suchmaske wird geöffnet. 3. Der Benutzer kann nach der Sensoradapter URI suchen. 4. Das System zeigt den gesuchten Sensor an.
Sonderfälle	4. Der gesuchte Sensor ist nicht registriert.
Nachbedingung	Die Sensorinformationen können editiert werden.

5.2.5.3 Sensorinformationen editieren

Use-Case	Sensorinformationen editieren
Ziel	Der Benutzer möchte die registrierten Sensorinformationen editieren.
Vorbedingung	• Der Benutzer hat den zu editierenden Sensor in der Suchmaske eingegeben.
Regulärer Ablauf	1. Der Benutzer wählt die Aktion: „Sensorinformationen editieren“. 2. Das Formularfeld zur Registrierung öffnet sich und die Sensorinformationen sind eingetragen. 3. Der Benutzer kann unter anderem die folgenden Felder editieren: • ObjectID • SensorType • Sensoradapter_URI • SensorQuality • Frequency • DataType • Geolocation 4. Der Benutzer speichert die neuen Informationen ab. 5. Das System aktualisiert die Sensorinformation in der Datenbank.
Sonderfälle	4. Es fehlen Schreibrechte für die Speicherung.
Nachbedingung	Die aktualisierten Sensorinformation können angezeigt werden.

5.2.5.4 Sensor deregistrieren

Use-Case	Sensor deregistrieren
Ziel	Der Benutzer möchte einen Sensor deregistrieren.
Vorbedingung	• Der Benutzer hat den zu deregistrierenden Sensor in der Suchmaske eingegeben.
Regulärer Ablauf	1. Der Benutzer wählt die Aktion: „Sensor deregistrieren“. 2. Der Sensor wird aus der Datenbank entfernt. 3. Die Registrierungskomponente schickt ein Ereignis an den Sensoradapter.
Sonderfälle	3. Es kann keine Verbindung zum Sensoradapter hergestellt werden.
Nachbedingung	Der Sensor ist in der Datenbank entfernt und der Sensoradapter deregistriert.

5.3 Sensoradapter-Schicht

Der Sensoradapter dient als Schnittstelle zwischen einem Sensor und der Ressourcenbereitstellungs-Schicht. Der Sensoradapter wird verwendet, um die Sensorwerte des jeweiligen Objekts auszulesen. Diese werden anschließend an die entwickelte Ressourcenbereitstellungs-Schicht im erforderlichen Format gesendet. Jedes Objekt ist von seiner Grundart unterschiedlich und stellt dementsprechend in der Praxis auch unterschiedliche Anforderungen an die Anbindung eines Sensoradapters. Es gibt Objekte, auf denen der Sensoradapter direkt implementiert und ausgeführt werden kann. Das kann beispielsweise ein Computer oder eine rechnergestützte Produktionsanlage mit Zugang zum Internet sein.

Ältere technische Geräte, wie zum Beispiel alte Produktionsanlagen ohne computergestütze Steuerung, besitzen weder eine Internetverbindung, noch können die Sensoradapter direkt darauf ausgeführt werden. Hierbei muss mit Zwischenlösungen gearbeitet werden, wie beispielsweise zwischengeschaltete Minicomputer (z.B. Raspberry Pi[1]), die über eine analoge oder digitale Schnittstelle die Signale der Sensoren auslesen und anschließend die Sensorwerte an den jeweiligen Sensoradapter weiterleiten.

[1] https://www.raspberrypi.org

Eine weitere Herausforderung stellen die Sensoren selbst dar. Es muss überprüft werden, inwieweit die Sensorwerte bereitgestellt werden. Bei einem Push-Verfahren liegt immer ein aktueller Messwert am Sensorausgang und die Daten können direkt vom Sensoradapter gelesen werden. Dem gegenüber stehen Sensoren, die über das Pull/Push-Verfahren Sensorwerte anbieten. Diese müssen erst über einen Zwischenschritt vom Sensoradapter im bestimmten Intervall angefordert werden, woraufhin dieser den aktuellen Messwert schickt. Somit ist der Aufbau eines Sensoradapters technisch- und bauartbedingt für jedes Objekt und jeden Sensortyp individuell und der Sensoradapter muss entsprechend implementiert werden.

Der unterschiedliche Aufbau wird in zwei Gruppen zusammengefasst und im Folgenden näher beschrieben.

5.3.1 Direkte Sensoradapteranbindung

Wie bereits in Kapitel 5.3 beschrieben, werden die Sensoradapter bei modernen Anlagen und technischen Geräten, die bereits über eine Internetverbindung verfügen, direkt ausgeführt. Das heißt, besitzt ein Objekt ein eigenständiges Betriebsystem oder eine rechnergestützte Steuerung mit einer Internetverbindung, kann der Sensoradapter in der jeweiligen Umgebung direkt integriert werden. Der Adapter greift über die bereitgestellte Schnittstelle über das Push-Verfahren auf die Sensordaten zu. Ist keine direkte Schnittstelle zum Sensor vorhanden, müssen die Sensorwerte beispielsweise über ein Zusatzprogramm ausgelesen und an den jeweiligen Sensoradapter weitergeleitet werden (Pull/Push-Verfahren). Dieser Aufbau wird in Abbildung 5.5 verdeutlicht. Der Sensoradapter wird nachfolgend als Pseudocode in 5.1 aufgezeigt:

Algorithmus 5.1 Pseudo-Code: Sensordaten extrahieren und senden

```
boolean abort = false;
procedure SendSensorData
    while true do
        getSensorData();
        sendSensorData();
        if abort then
            break;
        end if
    end while
end procedure
```

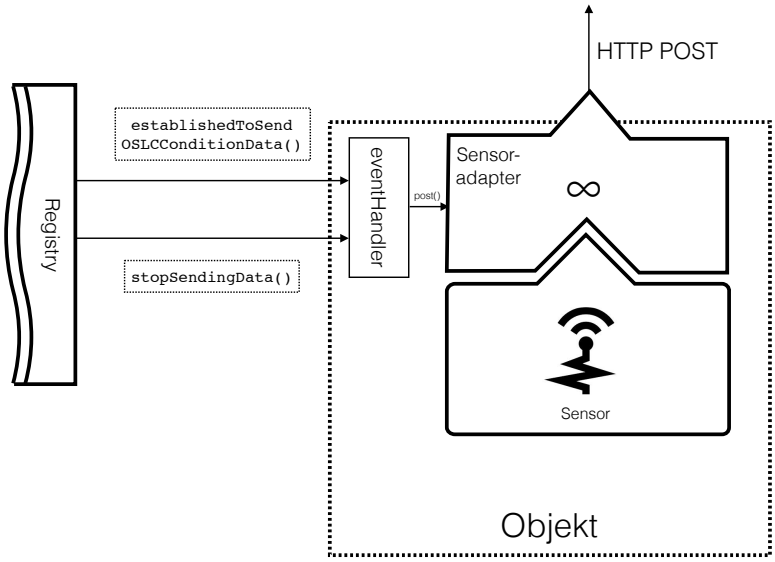

Abbildung 5.5: Beispiel eines Sensoradapters mit direkten Zugang zum Internet

5.3.2 Sensoradapteranbindung mit einer Zwischenkomponente

Ältere technische Geräte oder frei in der Umgebung platzierte Einzelsensoren ohne direkte Verbindung zum Internet müssen über eine gesonderte Schnittstelle mit dem Internet verbunden werden. Ein derartiger Adapter wird in Abbildung 5.6 dargestellt. Dies betrifft beispielsweise Produktionsanlagen, die nur mittels Microcontrollern gesteuert werden. Die analogen Sensorwerte müssen von Einplatinencomputer, wie zum Beispiel einem Raspberry PI[2] erfasst und verarbeitet werden. Diese Mini-Computer verfügen über eine Netzwerkverbindung und haben die Möglichkeit, die analogen Sensorwerte zu digitalisieren, um sie anschließend an einen Sensoradapter zu senden. Erst danach können die Werte an die Ressourcenbereitstellungs-Schicht gesendet werden. Diese Adapter sind somit erst über eine Zwischenkomponentente mit dem Sensor verbunden.

Im Folgenden werden die gemeinsamen Eigenschaften der unterschiedlichen Sensoradapter zusammengefasst.

Der Sensoradapter besteht im Wesentlichen aus einem Webserver, der über einen Ereignishandler (engl. Eventhandler) gesteuert wird. Ein Ereignis stellt dabei entweder die Sensor-*Registrierung* oder *-Deregistrierung* dar. Auf diese Ereignisse wird im Folgenden näher eingegangen.

[2]https://www.raspberrypi.org

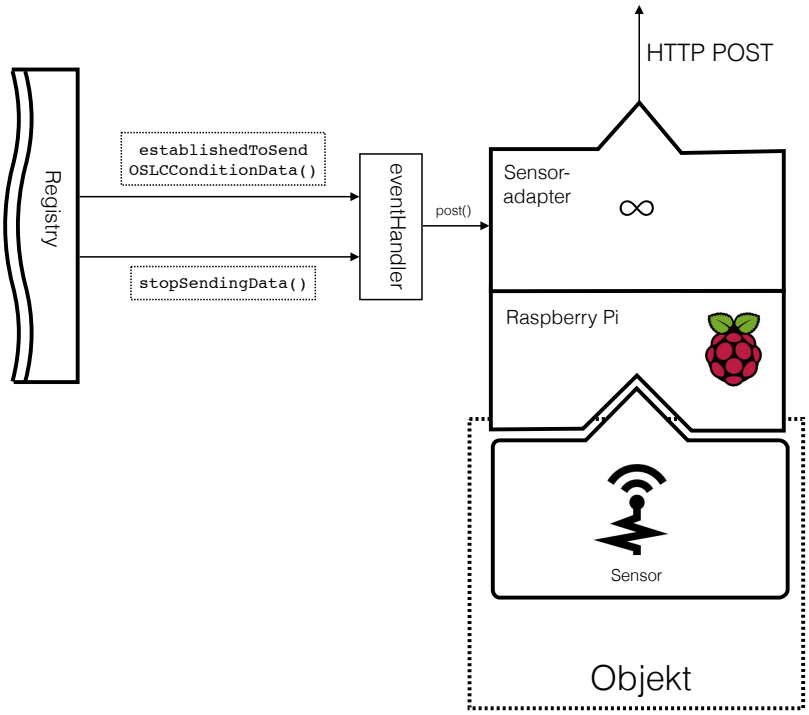

Abbildung 5.6: Beispiel eines Sensoradapters über eine gesonderte Schnittstelle (Rasberry Pi)

5.3.2.1 Ereignisgesteuerte Sensorregistrierung

Bei der eventgesteuerten Sensorregistrierung werden die Daten aus den zuvor eingetragenen Formularfeldern **ObjectID** und **SensorType** aus der Registrierungskomponenten an den Sensoradapter geschickt. Dieses Registrierungsereignis startet den Sensoradapter, der eine Kommunikation mit dem Sensor eröffnet und die Sensordaten ausliest. Jeder Sensorwert wird mit einem aktuellen Zeitstempel versehen. Der Zeitstempel gibt Auskunft darüber, zu welcher Zeit der Wert gemessen wurde. Fällt beispielsweise der Sensor aus, kann nachvollzogen werden, wann der letzte Wert aufgetreten ist. Gleichzeitig werden die Daten bestehend aus ObjectID, SensorType und dem aktuellen Zeitstempel über das HTTP-Protokoll an die Ressourcenbereitstellungs-Schicht geschickt.

5.3.2.2 Ereignisgesteuerte Sensorderegistrierung

Bei der eventgesteuerten Sensorderegistrierung wird von der Registrierungskomponente unter Angabe der Sensorinformationen ein Deregistrierungsereignis an den Sensoradapter geschickt. Dieses Ereignis wird an die Ressourcenbereitstellungs-Schicht weitergeleitet.

Gleichzeitig wird die Kommunikation zwischen dem Sensor und dem Adapter beendet, so dass keine Daten mehr verschickt werden und die Ressource nicht mehr erreichbar ist.

5.4 Die Ressourcenbereitstellungs-Schicht

In diesem Kapitel werden die Ressourcenbereitstellungs-Schicht und die darin enthaltenen Komponenten beschrieben.

Auf der Suche nach einem geeigneten Konzept für die Bereitstellung und Vernetzung von OSLC-basierten Sensordaten als REST-Ressourcen wurden die Technologien, die in Kapitel 3 beschrieben wurden, in Betracht gezogen.

Die Ressourcenbereitstellungs-Schicht wird als eigenständiger Service in die Cloud ausgelagert und über das Internet bereitgestellt. Dies hat gegenüber dem lokalen Betrieb auf einem Rechner den Vorteil, dass ein zentraler Zugriff über das Internet auf den Service ermöglicht wird. Außerdem kann die Rechenleistung nach Bedarf skaliert werden. Müssen beispielsweise eine große Menge an Sensordaten gleichzeitig verarbeitet werden, kann die Leistung auf mehrere Server verteilt werden, ohne den Programmablauf zu beeinflussen. Die Sensordaten sind zeitkritisch, das heißt, sie müssen stets den aktuellen Messwert der Sensoren widerspiegeln. Veraltet ein Sensorwert während einer langsamen Verarbeitung durch den Service, gehen wichtige Informationen verloren und die vorliegende Situation basiert nicht mehr auf den aktuellen gemessenen Sensordaten.

Für die Kommunikation zwischen den Sensoradaptern und der Ressourcenbereitstellungs-Schicht dient ein sogenannter OSLC-Adapter als Schnittstelle. Der Adapter empfängt die verschickten Sensordaten und speichert diese im Cache. Der Cache wird als Key-Value Store realisiert und ermöglicht einen permanenten Zugriff auf die Sensordaten, selbst wenn gerade keine neuen Messwerte von den Sensoradaptern verschickt werden. Im OSLC-Service-Provider werden die einzelnen OSLC-Services spezifiziert und verwaltet. Dazu erhält jeder OSLC-Service unter anderem einen eindeutigen Bezeichner sowie spezifische Eigenschaften für die Erstellung der REST-Ressourcen. Die OSLC-Services beziehen die Daten aus dem Cache und generieren daraus die OSLC-basierten REST-Ressourcen. Diese stellt der jeweilige Service anschließend über eine URI im Internet für die weiteren Verwendung zur Verfügung.

5.4.1 OSLC-Service-Provider

Im OSLC-Service-Provider werden die enthaltenen OSLC-Services verwaltet. Die Angaben zur Verwaltung werden mit einer RDF/XML-Datei spezifiziert. Im Kopfbereich der Spezifikationsdatei wird die URI definiert, unter der ein OSLC-Service-Provider erreichbar ist. Anschließend wird dem OSLC-Serivce-Provider ein eindeutiger Name und optional eine Beschreibung zugeteilt. Darauf folgen die Namensräume, die in den jeweiligen Services als Vokabular verwendet werden. Das Vokabular ist abhängig vom jeweiligen Sensortyp und muss für neu hinzugefügte Sensortypen entsprechend angepasst werden. Anschließend werden die zum OSLC-Service-Provider zugehörigen OSLC-Services aufgelistet. In jedem dieser Services werden für die Erstellung und für die Abfrage der REST-Ressourcen eindeutige URIs definiert. Diese geben an, unter welcher Adresse eine Ressource erstellt wird und mit welcher URI eine Ressource vom OSLC-Service abgefragt werden kann. Ein solches Konzept wird in Abbildung 5.7 verdeutlicht. Die Spezifikationsdatei eines OSLC-Service-Providers wird beispielhaft in Listing 5.1 dargestellt.

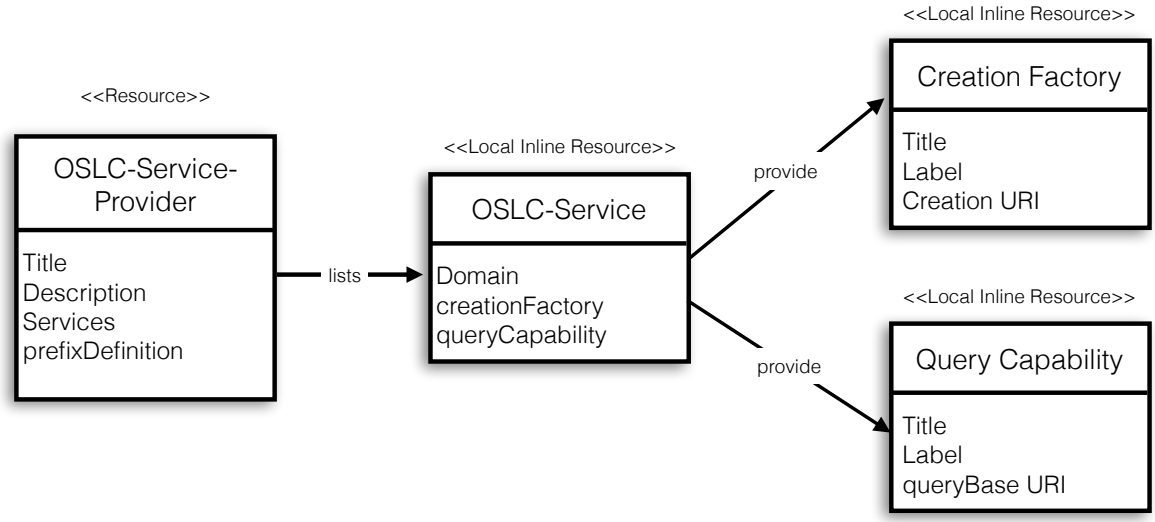

Abbildung 5.7: Konzeptionelle Lösung eines OSLC-Service-Providers

Listing 5.1 Beispiel-XML-Spezifikation eines OSLC-Service-Providers

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:oslc="http://open-services.net/ns/core#">

<oslc:ServiceProvider
   rdf:about="http://example.com/service-provider">

   <dcterms:title>Sensor Service</dcterms:title>
   <dcterms:description>Example OSLC Sensor Service</dcterms:description>

   <oslc:prefixDefinition>
      < oslc:PrefixDefinition >
         <oslc:prefix>rdf</oslc:prefix>
         <oslc:prefixBase rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      </oslc:PrefixDefinition >
   </oslc:prefixDefinition >

   < oslc:prefixDefinition >
      < oslc:PrefixDefinition >
         <oslc:prefix>oslc</oslc:prefix>
         <oslc:prefixBase rdf:resource="http://open-services.net/ns/core#" />
      </oslc:PrefixDefinition >
   </oslc:prefixDefinition >

   < oslc:prefixDefinition >
      < oslc:PrefixDefinition >
         <oslc:prefix>oslc_sensor</oslc:prefix>
         <oslc:prefixBase rdf:resource="http://example/ns/sensorsvokabular/sensors#" />
      </oslc:PrefixDefinition >
   </oslc:prefixDefinition >

   <oslc:service>
      <oslc:Service>
         <oslc:domain rdf:resource="http://example.com/xmlns/example-cm#" />

         <oslc:creationFactory>
            <oslc:CreationFactory>
               <dcterms:title>Location for creation Sensor Resources for Personal Computer</dcterms:title>
               <oslc:label>Sensor Creation for Object Personal Computer</oslc:label>
               <oslc:creation rdf:resource="http://example.com/creation/sensors" />
            </oslc:CreationFactory>
         </oslc:creationFactory>


         <oslc:queryCapability>
            <oslc:QueryCapability>
               <dcterms:title>Sensor Query</dcterms:title>
               <oslc:label>Sensor Query</oslc:label>
               <oslc:queryBase rdf:resource="http://example.com/query" />
            </oslc:QueryCapability>
         </oslc:queryCapability>

      </oslc:Service>
   </oslc:service>

</oslc:ServiceProvider>
</rdf:RDF>
```

5.4.2 OSLC-Adapter

In diesem Abschnitt werden der OSLC-Adapter sowie seine Funktionalität beschrieben. Der OSLC-Adapter ist als Schnittstelle zwischen den Sensoradaptern und der Ressourcenbereitstellungs-Schicht konzipiert. Sobald die Daten **Objekt-ID**, **Sensor-Typ**, **Sensor-Wert** und der **Zeitstempel** vom jeweiligen Sensoradapter verschickt werden, werden diese vom OSLC-Adapter über den HTTP-POST Aufruf empfangen. Diese Daten werden anschließend als Schlüssel-/Wert-Paare über das Push-Verfahren in den Cache gespeichert.

Eine Implementierung mit einem Pull-Verfahren würde dazu führen, dass keine aktuellen Sensordaten vorliegen, da an dieser Stelle ein Zwischenschritt erfolgt, um die Daten in vordefinierten Abständen vom Sensoradapter abzuholen. Die Anwender der Daten würden dementsprechend mit veralteten Messwerten arbeiten und eine andere Situation vorliegen haben als diejenige, die dem aktuellen Zeitpunkt entspricht. Dementsprechend kann nicht zeitnah auf eine Sitution reagiert werden. Mit dem Push-Dienst ist es möglich, die Sensordaten direkt nach dem Empfangen weiterzuleiten und die Zeitverzögerung auf ein Minimum zu reduzieren. Ein weiterer Vorteil liegt in der Reduzierung der Verbindungen. Diese sind in der Regel teuer und müssen bei einem Pull-Verfahren ständig neu aufgebaut und wieder geschlossen werden, selbst wenn keine Sensordaten vom Sensoradapter bereitgestellt werden. Das Push-Verfahren öffnet nur dann eine Verbindung, wenn der Sensor neue Werte über den Sensoradapter verschickt.

Die Kommunikation und der Aufbau des OSLC-Adapters wird in Abbildung 5.8 verdeutlicht.

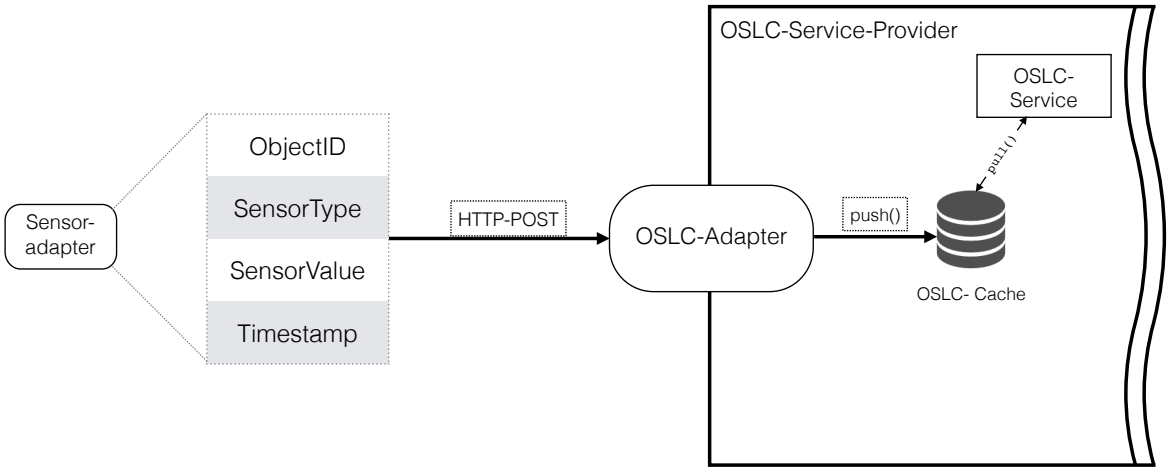

Abbildung 5.8: Der OSLC-Adapter

Der OSLC-Adapter hat folglich die Aufgabe, die Sensordaten als Schlüssel-/Wert-Paare zu empfangen und diese anschließend in den Cache zu speichern. Bei einer Deregistrierung

werden die Schlüssel-/Wert-Paare aus dem Cache entfernt. Dadurch wird die REST-Ressource nicht mehr generiert und der Aufruf der URI liefert einen HTTP 404-Statuscode (*404 = Not Found*).

5.4.3 Cache

Dieser Abschnitt beschreibt den Cache und die Motivation für dessen Einsatz in diesem Lösungskonzept.

Der Cache ist für eine permanente Bereitstellung der Sensordaten zuständig. Dadurch wird erreicht, dass zu jeder Zeit Sensordaten für die Verarbeitung zur Verfügung stehen, selbst wenn gerade keine aktuellen Sensorwerte vorliegen. Dadurch ist die Ressourcenbereitstellungs-Schicht unabhängig vom Sendeintervall der Sensoren. Der Cache wird als Key-Value Store realisiert. Derartige Speicherlösungen sind optimal für hohe Datenraten ausgelegt, wie zum Beispiel das ständige Aktualisieren und Abfragen der Sensordaten. Sobald ein Sensor neue Messdaten über den Sensoradapter bereitstellt und diese an den OSLC-Adapter schickt, wird auch der jeweilige Sensorwert im Cache aktualisiert.

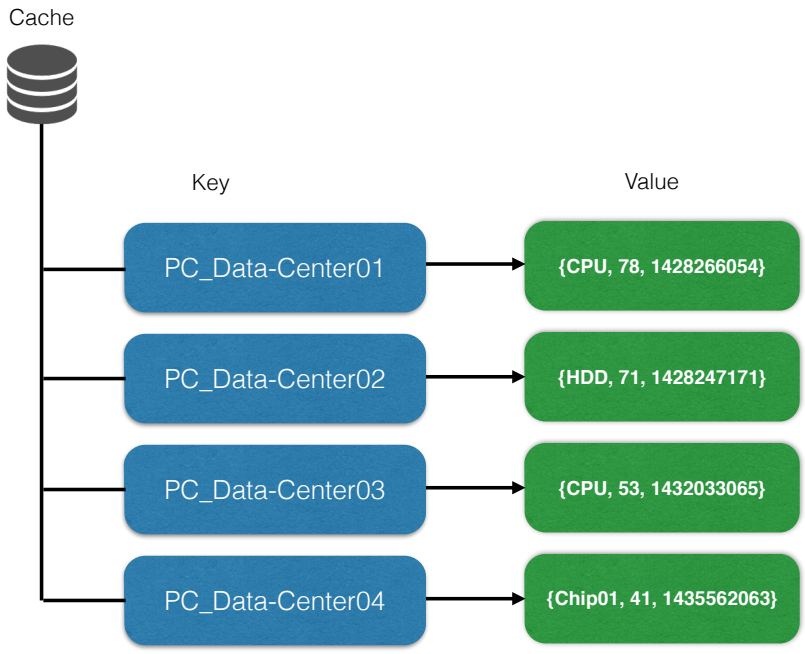

Abbildung 5.9: Beispiel der Sensordaten als Schlüssel-/Wert-Paare im Cache

Für eine optimale Laufzeit sollte der Cache zusammen mit dem OSLC-Service-Provider auf einem Server installiert werden. Dies verringert die Zugriffszeit bei sehr datenintensiven

Abfragen. Für eine bessere Skalierbarkeit können vom Cache mehrere Instanzen erstellt und auf verschiedenen Servern verteilt werden (Replikation). Dadurch können auch einzelne Serverausfälle ausgeglichen werden.

Wie bereits am Anfang dieses Abschnitts angesprochen, werden die Daten als Schlüssel-/Wert-Paare gespeichert. Als Schlüssel wird die ObjectID verwendet. Der Wert ergibt sich aus dem jeweiligen Sensortyp, dem gemessenen Sensorwert und dem zum Zeitpunkt der Messung erstellten Zeitstempel. Dies wird in Abbildung 5.9 verdeutlicht:

Für die Registrierung, Verwaltung und Deregistrierung der Sensordaten in der Datenbank ergeben sich drei Anwendungsfälle:

- **Sensordaten im Cache speichern:**
  Die durch den OSLC-Adapter verschickten Daten werden mittels Push-Nachrichten als Schlüssel-/Wert-Paare empfangen. Gleichzeitig wird geprüft, ob der entsprechende Sensor bereits registriert ist. Ist dies nicht der Fall, wird ein neuer Eintrag mit den empfangenen Sensordaten im Key-Value Store angelegt. Dabei werden die Daten **Objekt-ID**, **Sensor-Typ**, **Sensor-Wert** und der aktuelle **Zeitstempel** gespeichert.
- **Sensordaten aktualisieren:**
  Bei den empfangenen Sensordaten wird überprüft, ob der entsprechende Schlüssel bereits registriert ist. Ist dies der Fall, wird nur der neue **Sensor-Wert** und ein aktueller **Zeitstempel** aktualisiert. Ist kein Eintrag mit dem entsprechenden Schlüssel vorhanden, wird ein neuer Sensor angelegt.
- **Sensordaten löschen:**
  Wird ein Sensor deregistriert, werden auch die Daten im Cache gelöscht. Dabei wird überprüft, ob der entsprechende Sensorschlüssel bereits existiert. Ist dies der Fall, werden alle Daten, die zum entsprechenden Sensor gespeichert sind, gelöscht.

5.4.4 OSLC-Service

In diesem Abschnitt wird der OSLC-Service beschrieben.

Die Definition des OSLC-Services wird vom OSLC-Service-Provider vorgenommen. Im OSLC-Service wird die Spezifizierung für das Erstellen und Abfragen der OSLC-basierten REST-Ressourcen vorgenommen.

Die Sensordaten werden aus dem Cache mittels Push-Verfahren extrahiert, sobald vom Benutzer die entsprechende URI zur jeweiligen Ressource aufgerufen wird. Dabei wird immer der aktuellste der gemessenen Werte übertragen und in der entsprechenden Darstellung zurückgeliefert.

5.4.4.1 Aufbau der Spezifikationsdatei für einen OSLC-Service

Für die Erstellung und Vernetzung der Sensordaten untereinander werden nach dem Linked Data Prinzip und der OSLC-Spezifikation die Eigenschaften für jede erzeugte Ressource im jeweiligen OSLC-Service definiert. Das Link Data Prinzip wurde in Kapitel 3.4 erläutert. Die OSLC-Spezifikation wurde in Kapitel 3.5 beschrieben Die Spezifikation wird, wie beim OSLC-Service-Provider, mittels einer RDF/XML-Datei erstellt und besitzt folgende Eigenschaften:

- **Name (String):**
  Name des jeweiligen OSLC-Services.
- **URI:**
  Die URI des jeweiligen OSLC-Services. Die URI setzt sich aus dem Namensraum und dem Namen des OSLC-Services zusammen. Zum Beispiel sieht die URI für einen Service folgendermaßen aus: `http://open-services.net/ns/core#Service`
- **Properties:**
  Mit Hilfe der Eigenschaften (engl. Properties) werden für die Ressourcen weitere Kennzahlen festgelegt.
  - *Timestamp*:
    Die Angabe für den Zeitpunkt der Sensordatenerstellung.
  - *OSLC-Services*:
    Diese Eigenschaft gibt die Zugehörigkeit zu weiteren OSLC-Services an, die von einem OSLC-Service-Provider verwaltet werden. Dadurch wird die Vernetzung der Sensordaten über die Objektgrenzen hinaus ermöglicht.

5.4.4.2 URI-Aufbau der REST-Ressourcen

In diesem Abschnitt wird der Aufbau der URI einer REST-Ressource beschrieben. Der Aufbau wird als Grammatik beschrieben und setzt sich aus folgenden Elementen zusammen:

- **`<prefix>`**
  Der Prefix gibt das Übertragungsprotokoll für die Sensordaten an. Üblicherweise wird das HTTP-Protokoll verwendet. Für eine verschlüsselte Übertragung kann das HTTPS-Protokoll eingesetzt werden.
- **`<rmp_url>`**
  Die URI der Ressource-Management-Plattform (kurz: rmp_url). Diese wird vom OSLC-Service-Provider festgelegt.
- **`OBJECT_ID`**
  Die zum jeweiligen Objekt zugehörige ID.
- **`SENSOR_TYPE`**
  Der Sensor-Typ des jeweiligen Objektes.

Der Aufbau der URI im Gesamten wird im Folgenden anhand eines Beispiels dargestellt:

`http://www.example.com/OBJECT_ID/SENSOR_TYPE`

Bei der konzeptionellen Lösung wurden unterschiedliche Implementierungsvarianten untersucht, um eine effektive Strategie für den Aufbau des OSLC-Services zu erreichen. Dabei sind die Lösungsvarianten (i) **Ein OSLC-Service für alle Sensoren**, (ii) **Ein OSLC-Service pro Sensortyp** und (iii) **Ein OSLC-Service für jedes Objekt** entstanden, die sich am effektivsten erwiesen haben. Diese werden im Folgenden aufgelistet und mit den Vor- und Nachteilen der jeweiligen Lösungsansätze verglichen.

5.4.4.3 Ein OSLC-Service für alle Sensoren

Abbildung 5.10 zeigt die Realisierung eines einzigen OSLC-Services. Dieser ist für alle registrierten Objekte und Sensoren zuständig. Es wird dabei keine Trennung zwischen den verschiedenen Objekten und Sensortypen vorgenommen. Der OSLC-Service verwaltet alle Sensortypen und erstellt daraus die OSLC-basierten REST-Ressourcen.

Vorteile:

- Für die Verwaltung der Sensordaten ist genau ein OSLC-Service zuständig. Dieser muss nur einmal für die gesamte Implementierung definiert werden.
- Da nur ein einziger OSLC-Service existiert, kann auf die Verwendung eines Service-Providers verzichtet werden.

- Eine Auflistung aller angemeldeten Sensoren wird ermöglicht.
- Für das Erstellen und Anzeigen der REST-Ressourcen wird nur eine URI definiert. Diese verweist dann auf den OSLC-Service. Es muss nur der Sensortyp in der URI angegeben werden.

Nachteile:

- Die dargestellte Lösungsvariante ist bei einer sehr großen Anzahl der angemeldeten Sensoren unübersichtlich.
- Wird ein Sensor deregistriert, muss in der gesamten Auflistung nach dem richtigen Sensor gesucht werden. Ein solches Vorgehen erhöht die Laufzeit bei sehr großen Datenmengen.
- Wird ein neuer Sensor angemeldet, muss entsprechend der gesamte OSLC-Service angepasst werden, dabei kann es zu Seiteneffekten kommen und es kann nicht garantiert werden, dass alle Sensoren mit dem angepassten Service kompatibel sind.
- Für eine Übersicht fehlt der entsprechende Bezug zwischen einem Objekt und seinen Sensoren. Die entsprechende Zuordnung in der URI ist nicht ersichtlich.
- Der Service wächst mit der Anzahl der Objekte und Sensoren.

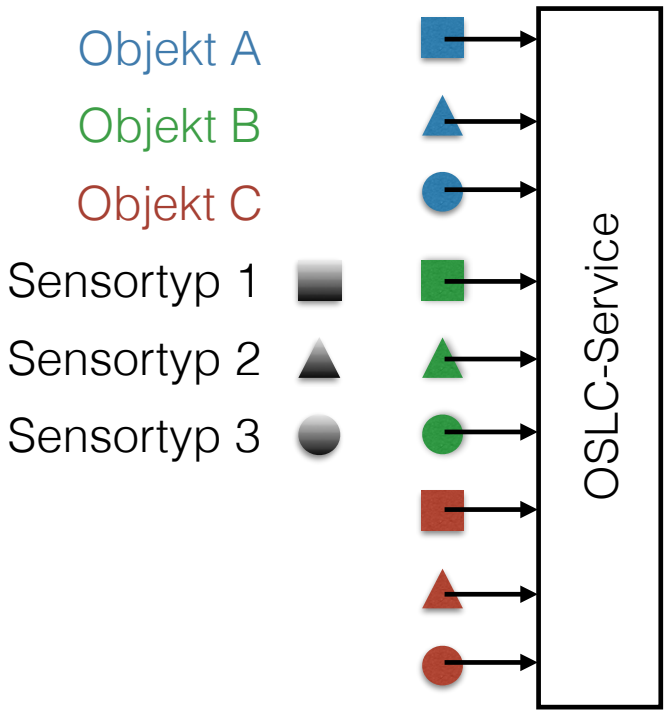

Abbildung 5.10: Ein OSLC-Service für alle Sensoren

5.4.4.4 Ein OSLC-Service pro Sensortyp

Abbildung 5.11 zeigt die Realisierung mit mehreren OSLC-Services in einem Service-Provider. Ein OSLC-Service ist dabei für einen bestimmten Sensortyp konzipiert. Es können gleiche Sensortypen aus verschiedenen Objekten zu einem Service zusammengefasst werden.

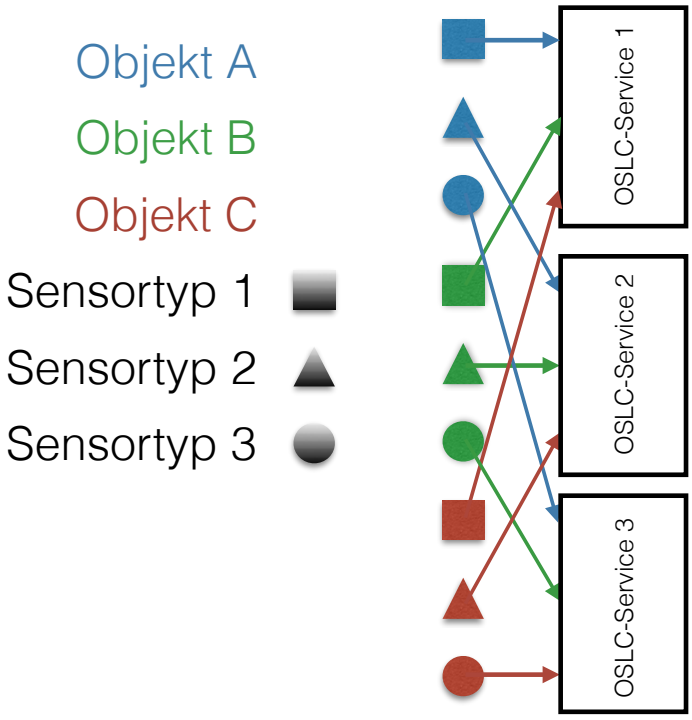

Abbildung 5.11: Ein OSLC-Service pro Sensortyp

Vorteile:

- Für jeden Sensortyp existiert jeweils ein eigener OSLC-Service. Dadurch lassen sich alle Sensoren vom gleichen Typ auflisten.
- Eine Anpassung an neu angemeldete Sensoren ist kaum beziehungsweise nicht notwendig, da alle angemeldeten Sensoren in diesem Service vom gleichen Typ sind.
- Die OSLC-Services werden kleiner.

Nachteile:

- Für die Verwaltung der OSLC-Services muss ein Service-Provider implementiert werden.

- Im realen Prozess gibt es eine große Anzahl an verschiedenen Sensortypen. Dies erhöht auch die Anzahl der jeweiligen Services.
- Aufgrund der Vermischung der Sensordaten aus verschiedenen Objekten zu einem OSLC-Service, müssen die OSLC-basierten REST-Ressourcen über Parameter in der URI aufgerufen werden. Dies entspricht nicht der reinen REST-Architektur. Aufgrund dessen ist die gesamte Ressourcenbereitstellungs-Schicht nicht *RESTful*.
- Wird ein Objekt abgemeldet, muss in den verschiendenen OSLC-Services nach den dazugehörigen Sensoren gesucht werden. Dies erhöht wiederum die Laufzeit.

5.4.4.5 Ein OSLC-Service für jedes Objekt

Die hier dargestellte Abbildung 5.12 zeigt die Realisierung eines OSLC-Services pro Objekt. Jeder OSLC-Service ist für ein Objekt zuständig und verwaltet die enthaltenen Sensordaten.

Vorteile:

- Für die Registrierung von Objekten genügt es, jeweils einen OSLC-Service zu erstellen. Dadurch lassen sich Objekte einfach hinzufügen und wieder entfernen.
- Wird ein Objekt deregistriert, genügt es nur den jeweiligen OSLC-Service zu entfernen.
- Die Übersichtlichkeit wird dadurch verbessert.
- Eine Anpassung für ein neuen Sensortyp muss nur im jeweiligen OSLC-Service vorgenommen werden. Die übrigen Services bleiben unangetastet.
- Durch die objektspezifischen OSLC-Services sind HTTP-Anfragen ohne Parameter möglich. Dies entspricht dem reinen REST-Architekturstil.

Nachteile:

- Es wird auch hier ein OSLC-Service-Provider benötigt.
- Wird ein neuer Sensortyp angemeldet, muss im jeweiligen Service eine Anpassung stattfinden. Diese Anpassung muss auch bei gleichen Sensortypen aus anderen Objekten in den jeweils anderen Services vorgenommen werden.
- Die OSLC-Services werden für jeden neu angemeldeten Sensor größer.

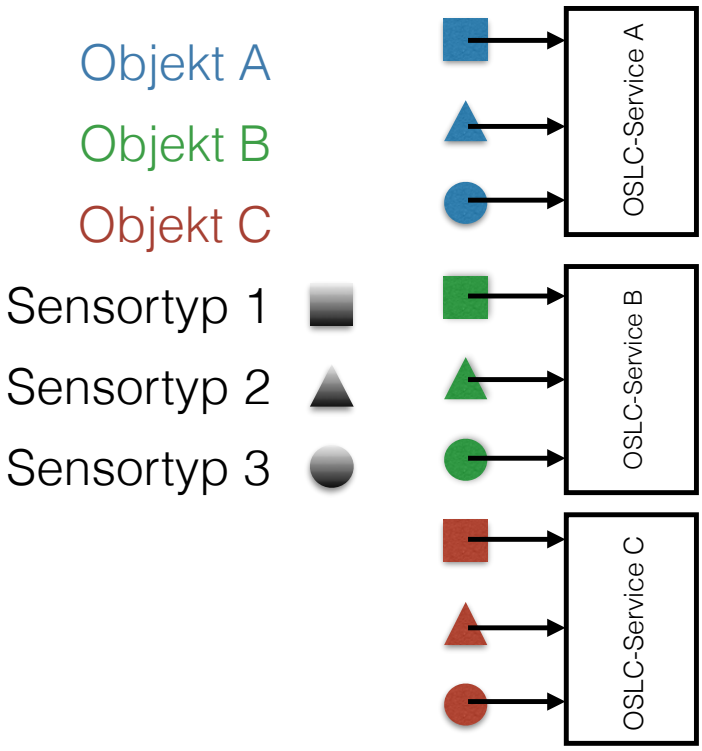

Abbildung 5.12: Ein OSLC-Service pro Objekt

5.4.4.6 Fazit aus dem Vergleich

Durch die Gegenüberstellung der drei Implementierungsvarianten für die OSLC-Services hat sich die Letzte der vorgestellten Varianten als die beste Wahl herausgestellt (siehe Kapitel 5.4.4.5). Obwohl auch diese Lösung einige Nachteile mit sich bringt, sind diese gering und vernachlässigbar. Es wurde auf einen Kompromiss zwischen Übersichtlichkeit und der einfachen Handhabung im praktischen Betrieb geachtet. Ein einziger Service für alle angemeldeten Sensoren würde in der Praxis einen sehr großen Service entstehen lassen und eine Zuordnung zu bestimmten Objekten könnte nicht bestimmt werden. Die Variante mit einem OSLC-Service für jeden Sensortyp würde die gleiche unstrukturierte Darstellung ergeben und es ist zudem ein größerer Aufwand, einen Sensor vom bestimmten Objekt zu deregistrieren. Aufgrund dieser Nachteile wird im weiteren Vorgehen dieser Diplomarbeit nur noch auf Services eingegangen (siehe Kapitel 5.4.4.5), die für jeweils ein Objekt zuständig sind. Hier wird die Zuordnung zwischen Objekt und Sensor durch einen Service klar geregelt und die Deregistrierung eines ganzen Objektes mitsamt allen darin enthaltenen Sensoren wird ermöglicht.

5.5 Sicherheit

Dieser Abschnitt gibt ein kurzen Ausblick über das Thema Sicherheit. Dies ist nicht Teil dieser Diplomarbeit, es werden aber trotzdem praxisbezogene Sicherheitsvorschläge geben, da dieses Thema nach wie vor berücksichtigt werden muss.

Der Einsatz von Webtechnologien birgt auch immer ein gewisses Risiko der Manipulation und des Fremdzugriffs von außen. Die Ressource-Management-Plattform wird als Service in der Cloud ausgelagert. Dadurch lässt sich ein Fremdzugriff über das Internet nur dann vermeiden, wenn Sicherheitsmechanismen eingebaut werden. Bei der konzeptionellen Lösung wurde das Thema Sicherheit nicht berücksichtigt, da dies zum einen nicht Gegenstand dieser Diplomarbeit ist und zum anderen den Rahmen übersteigen würde.

Mir ist bewusst, dass durch Fremdzugriff auf die Plattform die Sensordaten in der Registrierungskomponente gelöscht oder verändert werden können. Die Kommunikation zwischen der Registrierungskomponente und den Sensoradaptern sowie den Sensoradaptern und der Ressourcenbereitstellungs-Schicht kann ebenfalls abgehört werden. Auch der Cache bietet keinen Zugriffschutz vor Datendiebstahl oder Manipulation.

Aus diesem Grund, werden im Folgenden einige praxisbezogene Vorschläge gemacht, um die Sicherheit zu erhöhen. Trotz dieser Vorschläge, kann kein hundertprozentiger Schutz garantiert werden.

Für eine verschlüsselte Kommunikation kann das HTTPS-Protokoll verwendet werden. Dadurch werden die übermittelten Sensordaten vor Man-in-the-Middle-Angriffen [2] geschützt. Der Zugriff auf die Registry kann mit einem Zugriffschutz versehen werden. Erst durch die Eingabe von Benutzername und Passwort wird ein Zugang gewährt. Bei den Sensoradaptern kann die Übertragung der Sensorwerte auch mittels entsprechender Firewalls geschützt werden und der Versand der Daten an den OSLC-Adapter erfolgt über das HTTPS-Protokoll. Der Cache kann ebenfalls wie die Registry mit Hilfe von Zugangsdaten geschützt werden. Dies erschwert den Zugriff und die Manipulation der gespeicherten Sensordaten.

Für den OSLC-Service-Provider wird OAuth[16] als Authentifizierungmechanismus akzeptiert. Die OSLC-Gemeinschaft definiert jedoch keine Richtlinien für die Verwendung von OAuth. Für weitere Informationen wird auf die Quellenangabe verwiesen.

6 Technische Umsetzung

Dieses Kapitel beschreibt die technische Umsetzung der in Kapitel 5 vorgestellten konzeptionellen Lösung. Die entwickelte Resource-Management-Platform wird für das SitOPT-Projekt [12] verwendet, welches am Institut für Parallele und Verteilte Systeme[1] entwickelt wird. Die entwickelte Plattform stellt Sensordaten mittels OSLC-basierten REST-Ressourcen für eine automatische Situationserkennung zur Verfügung.

Nach der Beschreibung eines Anwendungsfalls in Kapitel 6.1, welcher als Grundlage für die Implementierung dieser Arbeit dient, wird anschließend in Kapitel 6.2 das Ergebnis der Implementierung für diesen Anwendungsfall beschrieben. Dabei werden die einzelnen Komponenten und ihre Funktionalität erläutert. Für ein besseres Verständnis wird auf die Hilfe von Screenshots und Listings zurückgegriffen. Die Abfolge der Komponentenbeschreibung lehnt sich an die Reihenfolge, wie sie bereits im Konzeptkapitel 5 verwendet wurde.

6.1 Anwendungsfall für die technische Umsetzung

Dieser Abschnitt beschreibt einen Anwendungsfall für den implementierten Prototypen. Dieser Anwendungsfall wird im weiteren Verlauf der Diplomarbeit referenziert.

Das Ziel dieses Anwendungsfalls ist es, verschiedene Temperatursensoren eines technischen Geräts zu registrieren, die Temperatursensorwerte auszulesen und über das Internet als OSLC-basierte REST-Ressourcen bereitzustellen. Dabei soll durch die Angabe einer URI zum jeweiligen Sensors die dazugehörige Temperatur und ein zum Zeitpunkt der Temperaturmessung angegebener Zeitstempel bereitgestellt werden. Die Sensorwerte sollen mit Verknüpfungen zu allen anderen registrierten Sensordaten im Browser angezeigt werden.

Für diesen Anwendungsfall wird ein Versuchsrechner verwendet, der über mehrere Temperatursensoren verfügt, auf deren Werte zugegriffen werden kann. Diese werden mit Hilfe der in Kapitel 6.2.1 beschriebenen Registrierungskomponente registriert und deren Sensoreigenschaften in einer Datenbank gespeichert. Anschließend werden die Daten eines jeweiligen Sensors vom entsprechenden Sensoradapter abgefragt, der wie in Kapitel 5.3.1 beschrieben, lokal auf dem Versuchsrechner ausgeführt wird. Der Adapter erfasst die Temperaturwerte und

[1] https://www.ipvs.uni-stuttgart.de

schickt diese zusammen mit den Sensorinformationen aus der Registry an die in Kapitel 6.2.3 beschriebene Ressourcenbereitstellungs-Schicht. Mit Hilfe der OSLC-Spezifikation wird ein OSLC-Service-Provider realisiert, der für die Sensoren einen OSLC-Service spezifiziert und bereitstellt. Der OSLC-Service erstellt aus den Sensordaten anschließend OSLC-basierte REST-Ressourcen, die untereinander mittels RDF verknüpft sind. Dabei erhält jeder registrierte Sensor eine URI, die über einen Aufruf in einem Web-Browser die Temperaturwerte und den Zeitpunkt der Sensorwert-Messung einschließlich Verbindungen zu allen registrierten Sensoren anzeigt. Die technische Umsetzung einer prototypischen Implementierung für diesen Anwendungsfall wird in Kapitel 6.2 beschrieben.

6.2 Umsetzung der Resource-Management-Platform

In diesem Kapitel wird das Ergebnis der prototypischen Implementierung für die Resource-Management-Platform beschrieben. Dabei lehnt sich die Reihenfolge der Beschreibungen, der einzelnen Komponenten an das Architekturbild 5.1 an, welches in Kapitel 5 beschrieben ist.

6.2.1 Umsetzung Registrierungskomponente

In diesem Abschnitt wird die technische Realisierung der Registrierungskomponente beschrieben.

Für den Anwendungsfall, der im oberen Abschnitt (siehe Kapitel 6.1) beschrieben ist, wird die Registrierungskomponente auf einem lokal eingesetzten Apache Tomcat Webserver[2] ausgeführt. Diese wird als Webservice umgesetzt und besitzt sowohl eine grafische Oberfläche, die mit einen Browser aufgerufen werden kann, als auch eine programmatische Schnittstelle für die anwendungsgesteuerte Sensorregistrierung, auf der direkte HTTP-Anfragen durchgeführt werden können. Für die Speicherung der zu registrierenden Sensoren wird eine MySQL-Datenbank verwendet.

Für die Umsetzung der grafischen Oberfläche wird ein Formular mit Hilfe von HTML5 und PHP 5.4 erstellt. Das Formular wird über eine URL im Browser aufgerufen. Der Vorteil der Umsetzung als Webformular ist eine vom Betriebsystem unabhängige Darstellung sowie eine einfache Anpassung der grafischen Ausgabe mit Hilfe von CSS[3].

[2] http://tomcat.apache.org
[3] Cascading Style Sheets

6.2.1.1 Benutzergesteuerte Sensorregistrierung

Für die Registrierung der Sensoren über die grafische Oberfläche trägt der Benutzer die URI der Registrierungskomponente in den Browser ein. Dies öffnet das Webformular für das Eintragen der Sensorinformationen. In dieser Implementierung werden die Felder **ObjectID**, **SensorType** und **Sensoradapter_URI** bereitgestellt. Die ObjectID ist in diesem Anwendungsfall der Versuchsrechner, der SensorType entspricht den verbauten Temperatursensoren und die Sensoradapter_URI adressiert den Sensoradapter, der in Kapitel 6.2.2 beschrieben wird. Die Felder werden als Pflichtfelder definiert, da diese die notwendigen Mindestanforderungen für die Sensorregistrierung darstellen. In Abbildung 6.1 wird die Darstellung dieses Web-Formulars im Browser angezeigt.

Abbildung 6.1: HTML-Darstellung eines Sensorregistrierungs-Formulars

Diese Sensorinformationen sind für den Anwendungsfall ausreichend, da die Informationen beispielsweise zur Sensor-Qualität oder der Frequenz zum Zeitpunkt der Implementierung nicht zur Verfügung standen. Für die prototypische Umsetzung sind diese Angaben jedoch nicht notwendig. Durch eine Anpassung der Webformular-Datei sowie der Änderung der Datenbankstruktur lassen sich zukünftig ohne großen Aufwand beliebig viele weitere Felder hinzufügen.

Speicherung der Sensordaten in die Datenbank:
Nachdem die Sensorinformationen durch den Benutzer eingetragen wurden, werden diese an die MySQL-Datenbank weitergeleitet und gespeichert. Parallel dazu wird auch ein Unix-Zeitstempel mit der PHP Funktion „time()“ generiert, der ebenfalls an die Datenbank geschickt und in einer weiteren Tabellenspalte gespeichert wird. Die Datenbank generiert für jeden Eintrag eine fortlaufende ID. Diese steht am Anfang jeder Zeile in der Tabelle.

Sensordaten an Sensoradapter senden:
Für eine eindeutige Zuordnung zwischen einem Objekt, seinem Sensor und dem gemessenen Sensorwert werden diese Angaben als Schlüssel-/Wert-Paare in einem Key-Value Store, welcher in Kapitel 6.2.3.2 detailliert beschrieben wird, gespeichert. Damit die Struktur einheitlich aus Schlüssel-/Wert-Paaren bestehen bleibt, werden die eingetragenen Sensorinformationen, bestehend aus der ObjectID, gefolgt von einem Unterstrich und dem SensorType zu einer Zeichenkette konkateniert. Diese Zeichenkette definiert dabei einen eindeutigen Schlüssel und der gemessene Sensorwert mit einem aktuellen Zeitstempel aus dem Sensoradapter entspricht dem Wert zum jeweiligen Schlüssel. Anstatt eines Unterstrichs, kann jedes beliebige „utf8“-Zeichenliteral verwendet werden. Wichtig dabei ist, dass dies einheitlich für alle Sensordaten verwendet wird. Damit die Zeichenkette keine weiteren Unterstriche, beispielsweise in der ObjectID, enthält, findet bei der Registrierung eine Überprüfung statt, die gegebenenfalls eine Meldung aufzeigt, dass bei den einzutragenden Feldern keine Unterstriche erlaubt sind.

Die Zeichenkette wird anschließend über einen HTTP-POST Aufruf über eine URI als Parameterangabe an den Sensoradapter übergeben und in diesem gespeichert. Um den entsprechenden Sensoradapter zu adressieren, wird die zuvor eingetragene Sensoradapter_URI verwendet. Dazu wird eine Methode implementiert, die unter Angabe dieser URI ein Registrierungsereignis mit der zuvor erwähnten Zeichenkette an den entsprechenden Adapter übermittelt.

In Abbildung 6.2 wird der Datenfluss, der sich daraus ergibt, verdeutlicht. Die grau eingefärbten Elemente sind noch nicht aktiv, das heißt, diese Daten wurden noch nicht übermittelt. Die inaktiven Elemente werden in den nachfolgenden Kapiteln detailliert beschrieben.

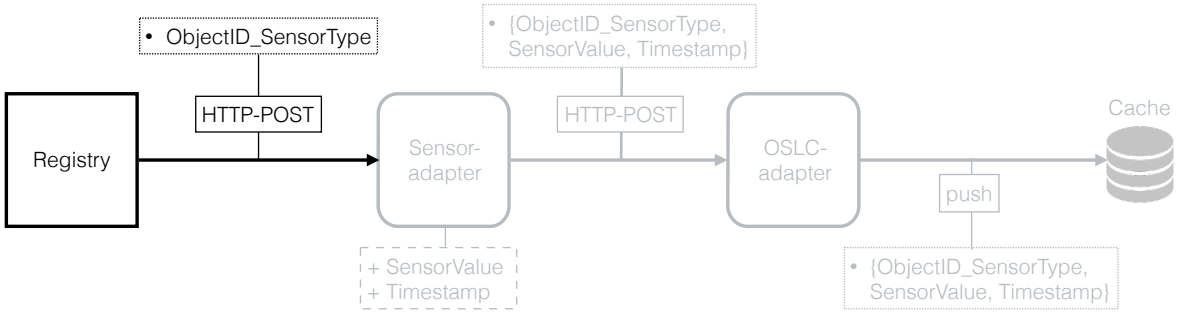

Abbildung 6.2: Datenfluss aus der Registrierungskomponente

6.2.1.2 Sensorinformationen editieren

Der Benutzer hat die Möglichkeit, sich einen gespeicherten Sensor anzuzeigen und dessen Informationen zu editieren. Dabei wird eine SQL-SELECT Abfrage generiert, die alle registrierten Sensoren tabellarisch im Browser auflistet. Hier hat der Benutzer anschließend die Möglichkeit, die Sensorinformationen zu ändern. Für das Ändern der Sensorinformationen wird erneut das Registrierungsformular geöffnet. Die zuvor eingetragenen Informationen sind bereits in den entsprechenden Feldern eingetragen. Nach der Änderung werden die Sensorinformationen mittels einer SQL-UPDATE Abfrage in der Datenbank überschrieben und gespeichert. Anschließend werden die geänderten Angaben erneut zu einer Zeichenkette konkateniert und an den Sensoradapter geschickt.

6.2.1.3 Sensor deregistrieren

Für eine Deregistrierung werden erneut alle registrierten Sensoren aufgelistet. Anschließend hat der Benutzer die Möglichkeit, einen Sensor auszuwählen und diesen zu deregistrieren. Bei der Deregistrierung wird der entsprechende Sensor aus der Datenbank mit einer SQL-DELETE Abfrage entfernt und gleichzeitig ein Deregistrierungsereignis an den Sensoradapter geschickt. Dieser leitet das Ereignis an den OSLC-Adapter weiter, der im Cache die entsprechenden Sensorwerte löscht. Dieser Vorgang wird in Kapitel 6.2.3.1 detailliert beschrieben.

6.2.1.4 Anwendungsgesteuerte Sensorregistrierung

Für die anwendungsgesteuerte Sensorregistrierung wird eine programmatische Webschnittstelle bereitgestellt, die über die gleiche URI erreichbar ist, die auch für die benutzergesteuerten Sensorregistrierung verwendet wird.

Sensorregistrierung
Die Sensorregistrierung über eine Anwendung erfolgt über einen HTTP-POST Aufruf. Dabei werden an die URI der Registrierungskomponente die Parameter **ObjectID**, **SensorType** und **Sensoradapter_URI** gesendet und in der Datenbank gespeichert. Gleichzeitig werden das Registrierungsereignis und die konkatenierte Zeichenkette bestehend aus ObjectID und SensorType an den Sensoradapter geschickt.

Sensorinformationen editieren
Für das Editieren der Sensorinformationen über eine Anwendung wird der entsprechende Sensor mit der URI über die Parameter **ObjectID**, **SensorType** und **Sensoradapter_URI** referenziert und die zu ändernden Daten mit einem HTTP-PUT Aufruf gesendet. Diese werden mit der SQL-UPDATE Abfrage in der Datenbank aktualisiert und gleichzeitig als Zeichenkette erneut an den Sensoradapter geschickt.

Sensoren deregistrieren
Für das Deregistrieren eines Sensors kann zuvor mit Hilfe eines HTTP-GET Aufrufs eine Liste der gespeicherten Sensoren aufgelistet werden. Wird die URI unter Angabe der entsprechenden Parameter mit einem HTTP-DELETE aufgerufen, wird der jeweilige Sensor mit einer SQL-DELETE Abfrage aus der Datenbank gelöscht und das Deregistrierungsereignis an den Sensoradapter geschickt und weiterverarbeitet.

6.2.2 Umsetzung Sensoradapter

In diesem Kapitel wird die Realisierung eines Sensoradapters beschrieben, der für den beschriebenen Anwendungsfall (siehe Kapitel 6.1) konzipiert wurde.

Für die prototypische Umsetzung wird nur ein Sensoradapter implementiert. Dieser kann drei unterschiedliche Temperatursensoren auslesen, die anschließend an einen OSLC-Adapter geschickt werden. In Listing 6.1 ist die Implementierung dieses Sensoradapters dargestellt. An dieser Stelle weicht die Implementierung vom Konzept ab, da nur ein Sensoradapter für alle Sensoren verwendet wird, anstatt einem Sensoradapter pro Sensor. Dies verringert den Umfang und Redundanz im Programmcode.

Der Sensoradapter wurde in der Skriptsprache JavaScript implementiert. Dieser wird in einem eigenständigen NodeJS-basierten Webserver ausgeführt. Dies soll die lose Koppelung der drei unterschiedlichen Schichten ermöglichen, die in Kapitel 5 eingeführt wurden. Node.js[4] basiert auf der JavaScript-Laufzeitumgebung „V8“, die eine ereignisgesteuerte Ausführung ermöglicht.

Für das Auslesen der Sensorwerte muss in diesem Anwendungsfall auf die Hilfe eines Dritthersteller-Programms (Hardwaremonitor[5]) zurückgegriffen werden, da sich auf dem Versuchsrechner keine direkte Schnittstelle zu den verbauten Temperatursensoren befindet. Das Programm ist im Internet als Freeware verfügbar und wird auf dem Versuchsrechner installiert. Die lizenzfreie Version bietet dabei nur Zugriff auf die Temperatursensoren. Dies ist aber für den beschriebenen Anwendungsfall ausreichend. Die Applikation bietet eine grafische Oberfläche für die Darstellung der Sensordaten und eine programmatische Schnittstelle, die über einen Konsolenaufruf angesprochen werden kann. Für die Implementierung wird nur auf die programmatische Schnittstelle zurückgegriffen, da diese die Sensordaten als JSON-Objekt zur Verfügung stellt. Dies ermöglicht eine einfache Extraktion der Sensordaten.

[4]https://nodejs.org/api/
[5]http://www.bresink.com/osx/HardwareMonitor-de.html

6.2.2.1 Implementierung ereignisgesteuerten Sensorregistrierung

Sobald die Sensoren registriert werden, wird eine HTTP-POST Anfrage an den Ereignishandler des Sensoradapters gesendet. Dabei werden die zuvor eingetragenen Sensorinformationen, bestehend aus ObjectID und SensorType zu Zeichenketten konkateniert und als einzelne Parameter an die URI der Anfrage angehängt. Das Ereignis startet, wie in Listing 6.1 gezeigt, in Zeile 20 das erwähnte Programm zum Auslesen der Sensordaten. Da das Programm keine kontinuierliche Ausgabe der Sensordaten ermöglicht, muss wie in Zeile 14 dargestellt, dessen Ausführung in einer Endlosschleife immer wieder erneut gestartet und anschließend die aktuellen Sensorwerte an den OSLC-Adapter geschickt werden.

Um den entsprechenden Sensorwert zum jeweiligen Temperatursensor zu erhalten, wird das JSON-Objekt zeilenweise ausgelesen und der Temperaturwert extrahiert. Gleichzeitig wird mit der JavaScript-Funktion „Date.now()“ ein aktueller Unix-Zeitstempel erstellt und zum jeweiligen Sensorwert hinzugefügt.

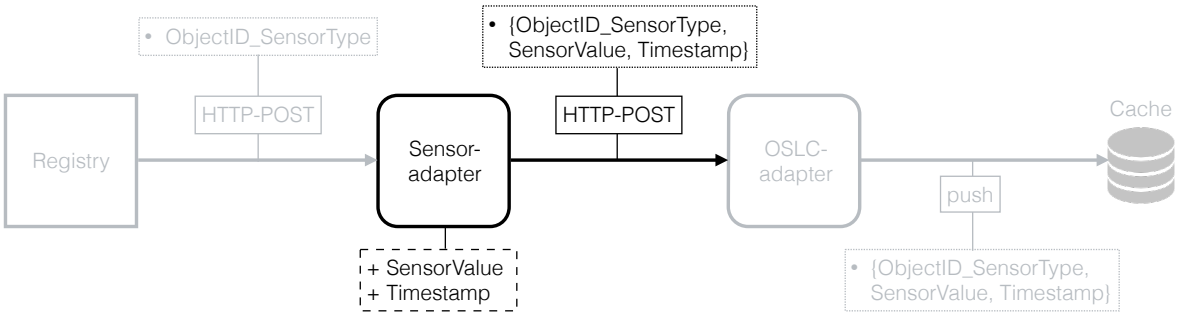

Abbildung 6.3: Datenfluss aus dem Sensoradapter

In Zeile 25 werden die Parameter der URI einzeln extrahiert und in Variablen gespeichert. Diese entsprechen den Schlüsseln und werden in Zeile 32 den jeweiligen Sensorwerten zugeordnet. Dadurch wird die Schlüssel-/Wert-Paar Struktur eingehalten.

Anschließend wird in Zeile 36 ein JSON-Objekt aus den Schlüssel-/Wert-Paaren erstellt und an den OSLC-Adapter geschickt. Der sich daraus ergebene Datenfluss ist in Abbildung 6.3 dargestellt.

6.2.2.2 Implementierung ereignisgesteuerten Sensorderegistrierung

Wird ein Sensor von einem Benutzer oder einer Anwendung deregistriert, wird ein entsprechende Deregistrierungsereignis an den Eventhandler des Sensoradapters geschickt. Dieser leitet das Ereignis an den OSLC-Adapter weiter, damit die Sensorwerte aus dem Cache gelöscht werden. Gleichzeitig wird der Prozess mit der Endlosschleife vom Eventhandler beendet, sodass keine Sensordaten ausgelesen und an den OSLC-Adapter gesendet werden.

Listing 6.1 Der Sensoradapter als NodeJS-basierter Webserver

```
1  http.createServer(function(req,res){
2    // Parse the uri from the registry
3    var urlObj = url.parse(req.url, true);
4    req.on('data', function () {});
5
6    req.on('end', function () {
7
8      // Option parameters for the request to the oslc-adapter
9      var options = {host: 'localhost', port: 8080, path: '/OSLCAdapter/',
10        method: 'POST', headers: {'Content-Type': 'application/json'}
11     };
12
13     // Repeat the request to oslc-adapter until the dereg-event is received
14     while (true) {
15
16       // Request to oslc-adapter with given options
17       http.request(options, function(resToOSLCAdapter) {
18
19         // Starts the hardwaremonitor and extract all data
20         exec('/Applications/HardwareMonitor.app/Contents/MacOS/hwmonitor',
21           function (error, stdout, stderr) {
22          var alltempsensors = stdout.split('\n');
23
24          // Path parameter as object names
25          var sensor1 = urlObj.query.sensor1;
26          // -- sensor2 and sensor 3 analogous --
27
28          // Create an object of all sensors
29          var tempsensors = {};
30
31          // Assign object ids to values of temperature sensor
32          tempsensors[sensor1] = alltempsensors[1].split(':')[1] + ', ' + Date.now();
33          // -- sensor2 and sensor 3 analogous --
34
35          // Creation of the temperature sensors to an JSON-Object
36          var jsonObject = JSON.stringify(tempsensors);
37
38          // Sends the JSON-Object to OSLC-Adapter
39          reqToOSLCAdapter.write(jsonObject);
40        });
41        res.end();
42      });
43    }
44   });
45 }).listen(8181);
```

6.2.3 Umsetzung Ressourcenbereitstellungs-Schicht

In diesem Abschnitt wird die prototypische Umsetzung der Ressourcenbereitstellungs-Schicht beschrieben. Das Architekturbild ist in Abbildung 5.1 dargestellt.

Die Komponenten der Ressourcenbereitstellungs-Schicht wurden in Java umgesetzt. Für die Implementierung wurde das Eclipse Lyo[6]-Werkzeug verwendet. Eclipse Lyo ist ein auf Eclipse basierendes Entwicklungswerkzeug und unterstützt die einfache Umsetzung von OSLC-Spezifikationen.

Für einen zentralen Zugriff kann die Ressourcenbereitstellungs-Schicht auf einem entfernten Server ausgeführt und über das Internet zugänglich gemacht werden. Für diesen Anwendungsfall wird ein zusätzlicher, lokaler Apache Tomcat Webserver eingesetzt.

6.2.3.1 Umsetzung OSLC-Adapter

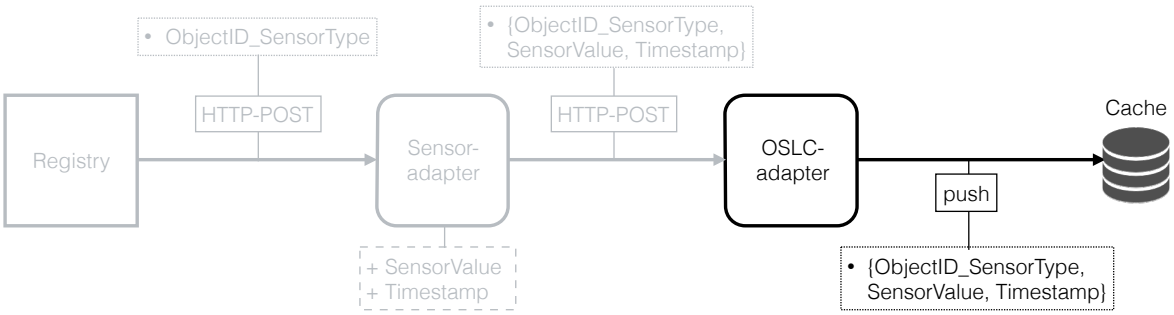

Abbildung 6.4: Datenfluss aus dem OSLC-Adapter

Der OSLC-Adapter wird als eigenständige Java-Klasse in der Ressourcenbereitstellungs-Schicht implementiert und verwendet für das Empfangen der Sensordaten aus dem Sensoradapter das HTTP-Protokoll. Das empfangene JSON-Objekt bestehend aus den Schlüssel-/Wert-Paaren der registrierten Sensoren wird in einzelne JSON-Objekte aufgespalten und direkt, wie in Abbildung 6.4 dargestellt, an den Cache weitergeleitet. Das Push-Verfahren ist effizient und kann die Sensordaten in Echtzeit an den Cache weiterleiten, beziehungsweise die vorhandenen Sensordaten aktualisieren. Ein weiterer Vorteil ergibt sich daraus, dass dadurch die Anzahl der Verbindungen zwischen dem Sensoradapter und dem OSLC-Adapter reduziert werden kann. Der Sensoradapter muss die Sensordaten nicht vorhalten und auf die nächste Anfrage des OSLC-Adapters warten. Stehen keine neuen Sensorwerte bereit, wird auch keine neue Verbindung aufgebaut.

[6]http://eclipse.org/lyo/

Nachdem der OSLC-Adapter ein solches Schlüssel-/Wert-Paar extrahiert hat, wird unter Verwendung des Schlüssels überprüft, ob bereits Sensordaten im Cache gespeichert sind. Liegt kein Eintrag vor, wird das Schlüssel-/Wert-Paar im Key-Value Store, wie in Kapitel 6.2.3.2 beschrieben, angelegt. Befindet sich ein entsprechender Eintrag im Key-Value Store, wird unter Verwendung des Schlüssels nur der entsprechende Wert aktualisiert. Erhält der OSLC-Adapter ein Deregistrierungsereignis, wird das Schlüssel-/Wert-Paar aus dem Key-Value Store mit einer DELETE-Abfrage entfernt.

6.2.3.2 Umsetzung Key-Value Store

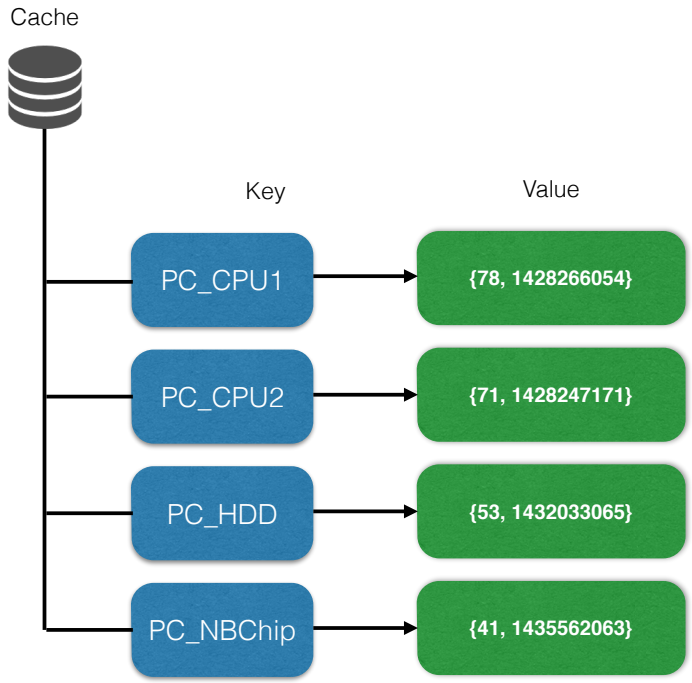

Abbildung 6.5: MongoDB mit der Collection Cache

In diesem Abschnitt wird die Implementierung des Caches als Key-Value Store beschrieben. Während der Implementierung wurde für das Speichern der Sensordaten auch ein alternativer Lösungsansatz untersucht. Dieser Ansatz wird im zweiten Abschnitt dieses Kapitels mit seinen Vor- und Nachteilen näher beschrieben und erklärt, weshalb die im ersten Abschnitt beschriebene Lösung gewählt wurde.

Für die Realisierung eines Key-Value Stores wurde auf eine NoSQL-Datenbank zurückgegriffen. Der große Vorteil von NoSQL-Datenbanken liegt in einer sehr guten horizontalen Skalierbarkeit [15]. Dies ermöglicht das Speichern von sehr großen Datenmengen, welche

sich durch das Erstellen von Instanzen der Datenbank auf mehrere Server verteilen lässt, ohne diese vorher herunterzufahren oder sonst den Betrieb zu stören. Als Key-Value Store wird MongoDB eingesetzt. MongoDB[7] ist eine dokumentenorientierte Datenbank, die im Gegensatz zu relationalen Datenbanken schemafrei ist. Sie gehört zu den „On Disk“ Datenbanken, welche die Daten direkt auf der Festplatte speichern. MongoDB ist mit vielen gängigen Treibern ausgestattet und bietet eine gut dokumentierte Java-API an.

Die Herausforderung bei der Umsetzung des Key-Value Stores war es, die eindeutige Zuordnung zu einem Objekt, dem Sensortyp sowie dessen Sensorwert und dem dazugehörigen Zeitstempel als Schlüssel-/Wert-Paare in der Datenbank abzubilden. Aus diesem Grund wurden die „ObjectID“ und der „SensorType“ zu einer Zeichenkette verbunden. Diese dient als Schlüssel für die eindeutige Zuordnung zu einem bestimmten Wert. Das Schlüssel-/Wert-Paar wird vom OSLC-Adapter in eine Datensammlung (engl. Collection) in dem Key-Value Store gespeichert. Dadurch wird eine einheitliche Struktur aus eindeutigen Schlüsseln und den dazugehörigen Werten erreicht. Die Struktur der Datenbank wird in Abbildung 6.5 dargestellt. Die Collection wird im weiteren Verlauf als „Cache“ bezeichnet. Hier werden nun nacheinander die Sensordaten gespeichert. Der OSLC-Service kann daraufhin zum jeweiligen Schlüssel den entprechenden Sensorwert auslesen und zu einer OSLC-basierten REST-Ressource verarbeiten. Sobald ein neuer Sensorwert und Zeitstempel vorliegt, wird der zuvor gemessene Wert mit einer UPDATE-Abfrage überschrieben.

Lösungsansatz mit einem ID-Manager
Eine andere Möglichkeit für die Zuordnung zwischen einem Objekt, dem Sensortyp, dem Sensorwert und dem dazugehörigen Zeitstempel verfolgt der Ansatz der Verwendung eines ID-Managers. Hierbei wird dem Objekt und dem Sensortyp eine eindeutige ID zugeordnet und in einer separaten Collection namens „ID-Manager“ gespeichert. Der Sensorwert und der Zeitstempel werden unter Verwendung der gleichen ID in die Collection Cache gespeichert. Daraus ergeben sich zwei Key-Value Stores für die Schlüsselverwaltung und der Sensordatenverwaltung. Dies wird in Abbildung 6.5 dargestellt. Um einen bestimmten Sensorwert für den Sensortyp eines Objekts zu erhalten, wird im ID-Manager zuerst nach dem entsprechenden Eintrag in der Wert-Spalte gesucht. Der dazugehörige Schlüssel (die ID) wird anschließend im Cache einem Wert zugeordnet. Diese Lösungsvariante ist robust für die eindeutige Zuordnung von verschiedenen Objekten und Sensortypen, da hier die Sensorinformationen getrennt gespeichert werden. Dies erhöht auch die Übersichtlichkeit und ermöglicht die Verwendung von komplexen Bezeichnungen für Objekte und Sensortypen.

Allerdings erhöht sich die Zugriffszeit auf die Sensordaten gegenüber der zuvor vorgestellten Lösung, aufgrund der doppelten Abfrage in den Collections. Außerdem müssen für die Sensorregistrierung und -deregistrierung mehr Daten ausgetauscht werden. Zusätzlich müsste eine Logik im OSLC-Adapter implementiert werden, die eine Abfrage in beiden Collections durchführt, um die entsprechenden Einträge anzulegen oder zu löschen.

[7] abgeleitet vom engl. hu**mongo**us, „gigantisch“, https://www.mongodb.org

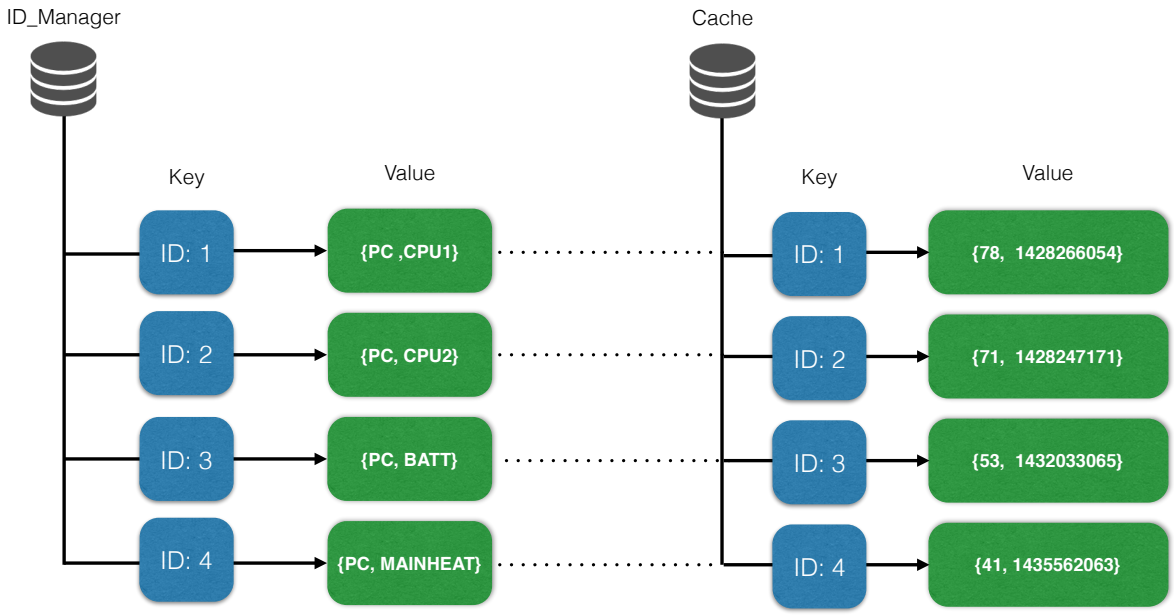

Abbildung 6.6: MongoDB mit zwei Collections: ID_Manager und Cache

6.2.3.3 Umsetzung des OSLC-Service-Provider

Der Service-Provider dient der Verwaltung und Auflistung der zur Verfügung stehenden OSLC-Services. In diesem Abschnitt wird auf den Service-Provider nur kurz eingegangen, da in diesem Anwendungsfall nur ein generischer OSLC-Service implementiert wird. Dieser verarbeitet immer die gleichen Sensordaten und somit kann im Wesentlichen auf einen Service-Provider verzichtet werden. Um die OSLC-Spezifikation einzuhalten wird der Service-Provider zwar implementiert aber sehr einfach gehalten. Zukünftig ist es aber denkbar, dass die Services heterogen werden und mehr als nur Temperatursensoren verwalten. In diesem Fall kann der rudimentär implementierte Service-Provider einfach um weitere Services erweitert werden.

6.2.3.4 Umsetzung OSLC-Service

In diesem Abschnitt wird die Umsetzung eines OSLC-Services dieser prototypischen Implementierung vorgestellt. Dieser kann, wie in Kapitel 5.4.1 bereits beschrieben, mit einer XML-Datei spezifiziert werden. Eine weitere Möglichkeit ist es, den Service mit Hilfe des

Listing 6.2 Beispiel einer Java-Klasse mit Annotationen

```
1 //sample simple string attribute
2 @OslcDescription("Descriptive text about resource.")
3 @OslcPropertyDefinition(OslcConstants.DCTERMS_NAMESPACE +
     "description")
4 @OslcTitle("Description")
5 @OslcValueType(ValueType.XMLLiteral)
6 public String getDescription()
7 {
8       return description;
9 }
```

Java-Werkzeuges OSLC4J[8] zu beschreiben. Für die Spezifikationsumsetzung wird im weiteren Verlauf der OSLC-Service mit dem OSLC4J-Werkzeug umgesetzt.

OSLC4J
Das OSLC4J ist ein Java-Werkzeug und Teil des Eclipse Lyo Projekts. Das Werkzeug unterstützt die Entwicklung von OSLC-Service-Providern, OSLC-Services und OSLC-REST-Ressourcen. Dabei werden Java-Objekte implementiert und mit Annotationen versehen, mit dessen Hilfe sich die OSLC-Spezifikation einfach umsetzen lässt. Die mitgelieferten OSLC4J Java-Bibliotheken vereinfachen die Erstellung von Service-Providern und Ressourcen sowie die Serialisierung und Deserialisierung von OSLC-Ressourcen in RDF- oder JSON-Repräsentationen. Das OSLC4J-Werkzeug beinhaltet außerdem eine Beispielapplikation sowie deren Testklassen. In Listing 6.2 ist ein Beispiel derartiger Annotationen abgebildet.

Wie in Kapitel 3 beschrieben, stellt die OSLC-Spezifikation[9] eine Vielzahl an Funktionen bereit. Für den implementierten Prototyp wird, basierend auf dem Anwendungsfall, nur auf die Erstellung und die Darstellung von OSLC-basierten REST-Ressourcen eingegangen. Das Aktualisieren und das Löschen von OSLC REST-Ressourcen sind für diese Diplomarbeit nicht relevant, da dies bereits von der Registrierungskomponente vorgenommen wird.

[8] http://wiki.eclipse.org/Lyo/LyoOSLC4J
[9] http://open-services.net/bin/view/Main/CmSpecificationV2

OSLC-Service
Der OSLC-Service dient der Rückgabe von Sensordaten- und informationen. Dabei bietet er zwei Darstellungsformen an, die die Sensordaten auf verschiedene Arten repräsentieren: einerseits die Darstellung der Sensordaten in JSON-Repräsentation, andererseits eine RDF-basierte Repräsentation, bei der die Metainformationen der Sensoren zurückgegeben werden.

Bei der JSON-Repräsentation werden der Sensorwert und der zugehörige Zeitstempel ausgegeben. Diese Darstellung lässt sich beispielsweise durch ein Situationserkennungs-System programmatisch verarbeiten. Die RDF/XML-Darstellung der Sensordaten hingegen liefert Informationen über die Verknüpfung der Sensoren. Beispielsweise können auf diese Art alle Sensoren eines bestimmten Objektes erhalten werden. Für die Umsetzung der beiden Repräsentationen werden im OSLC-Service zwei Methoden implementiert, welche im Folgenden näher beschrieben werden.

JSON-Repräsentation
Für die Darstellung der Sensordaten in JSON-Repräsentation wird die Methode „getSensordata“ implementiert. Diese Methode wird, wie in Listing 6.3 gezeigt, über die Annotation **@Produces(MediaType.APPLICATION_JSON)** gekennzeichnet.

Diese Methode wird aufgerufen, sobald ein HTTP-GET Aufruf über einen Browser oder einen anderen Webclient aufgerufen wird und im HTTP-Header die Information „application/json“ im Feld „Accept“ eingetragen ist. Um für einen bestimmten Sensor die Sensorwerte zu erhalten, wird die HTTP-GET Anfrage über die URI `http://HOST/objectID/sensorType` verschickt.

Die URI besteht dabei aus folgenden Elementen:

- **http://**
  diese Angabe gibt das verwendete Protokoll an.
- **HOST**
  diese Angabe ist für den Host bzw. Hostnamen. In dieser Implementierung ist der Host der lokal eingesetzte Apache Tomcat Webserver.
- **objectID**
  Hier ist das entsprechende Objekt einzutragen. In dieser Implementierung ist das Objekt der Versuchsrechner.
- **sensorType**
  Dieser Platzhalter steht für den entsprechenden Sensortyp. In dieser Implementierung wird einer der Temperatursensoren eingetragen.

Sobald die URI an den OSLC-Service übertragen wird, werden die Pfadangaben „objectID“ und „sensorType“ zu einem String konkateniert. Mit Hilfe des Strings wird der entsprechende Schlüssel im Cache gesucht und der dazugehörige Wert wird zurückgeliefert. Dies geschieht über das Pull-Verfahren. Anschließend kann das Ergebnis in einem Browser oder einem anderen Webclient dargestellt werden. Abbildung 6.7 stellt den Kommunikationsaufbau und die JSON-Repräsentation eines Sensorwerts mit dem dazugehörigen UNIX-Zeitstempel im Browser dar.

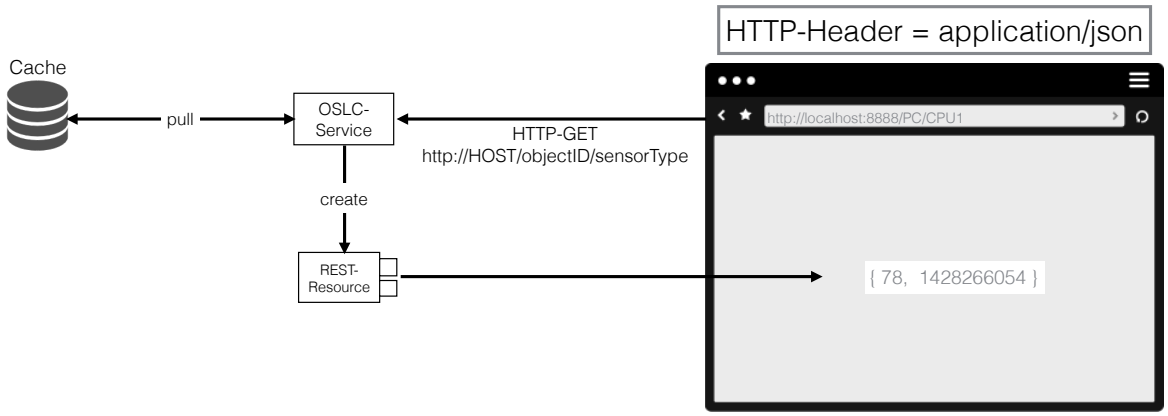

Abbildung 6.7: JSON-Repräsentation mit Beispieldaten

Listing 6.3 OSLC-Service für die JSON-Darstellung

```
1  @GET
2  @OslcService(ResourceManagementPlatform.RMP_DOMAIN) // the uri of the
       resource management platform
3  @Path("{objectID}/{sensorType}")
4  @Produces(MediaType.MediaType.APPLICATION_JSON)
5  public getSensordata(
6                      // transform the pathparameter to objectID and
                          sensorType to string
7                      @PathParam("objectID") final String objectID,
8                      @PathParam("sensorType") final String sensorType)
9  {
10       // concatenate objectID and sensorType to key
11       String sensordata = objectID + "_" + sensorType;

13       // create a connection to the key-value store
14       MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
15       DB db = mongoClient.getDB("keyValueStore");
16       DBCollection collection = db.getCollection("cache");

18       // create a query for the value by the given key
19       BasicDBObject query = new BasicDBObject(sensordata, new
             BasicDBObject("$exists", true));
20       DBCursor cursor = collection.find(query);

22       // grab the value
23       String sensorValue = (String) cursor.next().get(sensordata);

25       // if no value exists, response a not-found status code
26       if (value == null) {
27             throw new
                  WebApplicationException(Response.Status.NOT_FOUND);
28       }
29             // else show value on response
30             return
                  Response.status(200).entity(sensorValue.toString()).build();
31       }

33 }
```

RDF/XML-Repräsentation
Für die Darstellung der Sensordaten als RDF/XML-Repräsentation wird die Methode „getRDFSensordata“ bereitgestellt. Diese wird, wie in Listing 6.4 dargestellt, über die Annotation **@Produces(MediaType.RDF_XML)** gekennzeichnet und liefert Metainformation über alle Sensordaten, die von einem OSLC-Service verwaltet werden, zurück.

Dabei muss für den HTTP-Header die Information „application/rdf+xml“ im Feld „Accept“ über den HTTP-GET Aufruf geschickt werden. Die Angaben können zum Beispiel über ein Plugin (Advanced Rest Client Application[10]) für den Google Chrome Webbrowser übertragen werden. Der URI-Aufbau sowie die anschließend ausgeführte Aktion geschieht analog zur JSON-Repräsentation. In Abbildung 6.8 wird beispielhaft die Ausgabe der RDF/XML-Repräsentation eines Sensors dargstellt. Dabei werden neben dem Sensorwert (`<rdf:value>`) und Zeitstempel (`<rdf:date>`) alle registrierten Sensoren mit dem Attribut `<oslc:isSibling>` aufgelistet, die von einem OSLC-Service verwaltet werden.

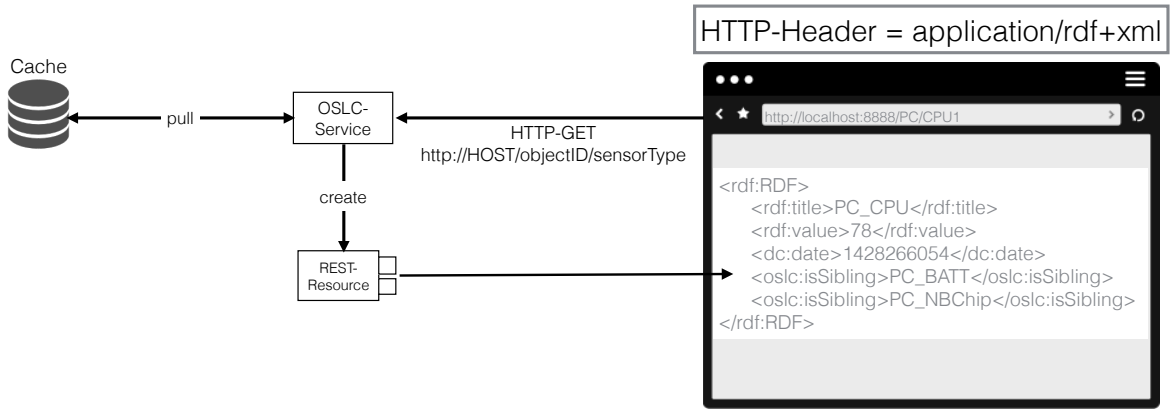

Abbildung 6.8: RDF/XML-Repräsentation mit Beispieldaten

[10]http://chromerestclient.appspot.com/

Listing 6.4 OSLC-Service für die RDF/XML-Darstellung

```
1  @GET
2  @Path("{objectID}/{sensorType}")
3  // produces the rdf/xml representation of all registred sensors
4  @Produces(MediaType.RDF_XML)
5  public getRDFSensordata(
6                          @PathParam("objectID") final String objectID,
7                          @PathParam("sensorType") final String
                               sensorType)
8  {
9        // Connection to the key-value-store
10       MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
11       DB db = mongoClient.getDB("keyValueStore");
12       DBCollection collection = db.getCollection("cache");

14       // Concatenate path parameters to string
15       String sensordata = objectID + "_" + sensorType;

17       // Search in key-value-store for the value by given key
18       BasicDBObject query = new BasicDBObject(sensordata, new
             BasicDBObject("$exists", true));
19       DBCursor cursor = collection.find(query);

21       String sensorValue = (String) cursor.next().get(sensordata);

23       // get sensors of same object
24       Map<String, String> siblings = getSiblings(objectID, sensorType,
             collection);
25       JSONObject siblingsAsJSON = JSON.parse(siblings);
26       // add sensor value
27       siblingsAsJSON.put("value", sensorValue);

29       if (value == null) {
30       throw new WebApplicationException(Response.Status.NOT_FOUND);
31       }
32       return
             Response.status(200).entity(siblingsAsJSON.toString()).build();
33       }

35 }
```

7 Themenbezogene Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt, die zum selben Themenbereich dieser Diplomarbeit gehören. Diese werden außerdem von dieser Diplomarbeit abgegrenzt.

OpenMTC
Die OpenMTC [19] Plattform (Open Machine Type Communication) ist eine, in Zusammenarbeit mit der Technischen Universität Berlin und der Fraunhofer Fokus Forschungsgruppe, entwickelte Middleware-Kommunikationsplattform für standardisierte Maschinen-zu-Maschinen-Anwendungen (M2M) und -Dienste, vor allem im Bereich des Internet der Dinge. Diese Plattform wird nicht nur von großen Telekommunikationsunternehmen im M2M-Bereich angewandt, sondern bietet auch für viele Gerätehersteller und Integratoren einen Mehrwert. OpenMTC ist eine offene, cloud-fähige Plattform, die eine Vielzahl verschiedener Sensor- und Gerätetechnologien unterstützt und lässt sich mit anderen Dienstplattformen kombinieren. Die OpenMTC-Lösungsplattform wurde als eine horizontale Konvergenzschicht für die Maschinen-zu-Maschinen Kommunikation entwickelt und kann in vielen Anwendungsbereichen, den so genannten vertikalen M2M-Sektoren, angewandt werden. Hierzu zählen die Einsatzfelder Versorgungsunternehmen, Automobilindustrie, eHealth sowie Transport und Logistik. OpenMTC besteht im Wesentlichen aus zwei Schichten: der Frontend-Schicht und der Backend-Schicht, die auf verschiedenen Hardware-Plattformen laufen. Die Frontend-Schicht kann zum Beispiel über eine HTTP-Schnittstelle auf einem Smartphone dargestellt werden und ermöglicht die Steuerung der angeschlossenen Geräte, deren Sensoren über die Backend-Schicht registriert wurden. Das Backend kann zum Beispiel auf einem Raspberry Pi oder ähnlichen Gerät ausgeführt werden. Die beiden Schichten unterstützen unterschiedliche Internet-Kommunikationsprotokolle sowie lokale Zugangstechnologien wie Zigbee, FS20 oder Bluetooth. Über offene Schnittstellen können Anwendungen gemeinsame Service-Funktionen ausführen.

Im Gegensatz zu dieser Diplomarbeit werden bei der Lösung die Sensordaten nicht als REST-Ressourcen zur Verfügung gestellt. Die einzelnen Sensoren lassen sich nicht über das Internet ansprechen und es muss immer die Middleware adressiert werden. Die Verknüpfung der Sensordaten erfolgt mit der Middleware-Plattform und neu hinzugefügte Sensoren müssen über das Backend registriert und mit der jeweilige Umgebung verknüpft werden.

FIWARE
FIWARE [4] ist eine quelloffene OpenStack-basierte Cloud Middleware-Plattform. FIWARE verfolgt einen ähnlichen Ansatz wie die OpenMTC-Plattform und bietet sowohl ein Backend

für die Verwaltung und Registrierung von Sensoren – beispielsweise in intelligenten Umgebungen – und eine Frontend-Ansicht für die Darstellung, Überwachung sowie Steuerung von Objekten, wie beispielsweise mit einem Smartphone oder einem anderen internetfähigen Endgerät. Allerdings kann die Plattform modular um weitere Softwareapplikationen erweitert werden. Dazu wird eine Vielzahl an Erweiterungen zur freien Verwendung von FIWARE bereitgestellt. Zudem können durch die quelloffene lizenzfreie API-Spezifikation zusätzliche Erweiterungen erstellt und angebunden werden. Für die Anbindung von Sensoren und anderen kontextbasierten Informationen sowie für die Kommunikation zwischen den verschiedenen Software-Modulen wird ein Protokoll namens *„OMA NGSI-9/10“* bereitgestellt. Für das Zusammenführen aller Anfragen ist ein sogenannter *„Context-Broker“* zuständig. Dieser leitet alle angeforderten Daten an die angebundenen Software-Module und Sensoren weiter.

FIWARE verfolgt die gleichen Ziele wie die in dieser Diplomarbeit vorgestellten. Die Plattform ist kostenlos, es lassen sich unterschiedliche Sensoren anbinden und diese werden untereinander verknüpft. Sensoren verfügen über einen IoT Agent, einer Art Adapter, der die Daten an einen Context-Broker schickt. Alle Informationen werden in einer dokumentenbasierten Datenbank gespeichert. Allerdings werden Einschränkungen bezüglich der Programmierschnittstelle und des zu verwendenden Übertragungsprotokolls vorgenommen. So muss die Kommunikation ausschließlich über das angesprochene OMA NGSU-9/10 Protokoll erfolgen. Vorhandene oder neue Systeme müssen die vorgegebene API ansprechen können. In dieser Diplomarbeit wird lediglich die Softwarearchitekur vorgegeben, sodass auf unterschiedlichen Protokollen gearbeitet werden kann.

Sensor Instance Registry
Der Artikel „Discovery Mechanisms for the Sensor Web“ [13] beschäftigt sich hauptsächlich mit der Sensorregistrierung und der Sensorbereitstellung für verschiedene Webservices basierend auf dem „OGC Sensor Web Enablement Framework (OGC SWE)“. Das Open Geospatial Consortium (OGC)[1] ist eine gemeinnützige Organisation, bestehend aus über 370 Mitgliedern aus der Industrie, Forschung und Entwicklung. Die Organisation hat sich zum Ziel gemacht, die Entwicklung von raumbezogener Informationsverarbeitung (insbesondere Geodaten) auf Basis allgemeingültiger Standards zum Zweck der Interoperabilität festzulegen. So wurden in Kooperation mit der Internationalen Organisation für Normung, ISO-Standards für Sensorinformationen und deren Metadaten entwickelt sowie deren Schnittstellenbeschreibung. In [13] wird das dienstbasierte SWE-Rahmenwerk beschrieben, welches eine Plattform für die Entwicklung und Verknüpfung diverser sensorbezogener Standards im Sinne der OGC Organisation schafft. Dabei werden zusätzliche Dienste zur Webintegration von Sensoren und Sensornetzwerken beschrieben. Des Weiteren werden Lösungsansätze für das Aufsuchen von Sensoren und ihre semantische Verknüpfung beschrieben sowie das Sammeln von Sensor-

[1] http://www.opengeospatial.org

Metainformationen und die Integration dieser Sensoren in bereits bestehende Sammlungen von Sensorinformationen, den sogenannten „Sensorkatalogen“.

Wie auch in den anderen zuvor vorgestellten Arbeiten ist das Ziel, eine lizenzfreie und quelloffene Plattform zu schaffen, die mit Hilfe von Sensoren und Metadaten bestimmte Situationen überwachen und über das Internet zur Verfügung stellen kann. Allerdings gibt es auch hier Einschränkungen bezüglich des zu verwendenden Software-Frameworks. Das Standardisieren der Webintegration von Sensoren und Sensornetzwerken ist ein guter Ansatz, allerdings ist diese zeit- und kostenaufwendig und es müssen sich alle an diese Standards halten.

8 Zusammenfassung und Ausblick

In diesem Abschnitt wird die Diplomarbeit zusammengefasst und ein abschließendes Fazit gegeben. Im darauffolgenden Kapitel wird ein Ausblick auf zukünftige Arbeiten und Weiterentwicklungen gegebeben, die in Rahmen dieser Diplomarbeit auf Grund des großen Umfangs nicht umgesetzt werden konnten.

8.1 Zusammenfassung

In dieser Diplomarbeit wurde ein Konzept entwickelt und umgesetzt, welches es erlaubt, unterschiedliche Sensoren an eine Plattform anzumelden und deren Daten mit Hilfe der OSLC-Spezifikation als REST-Ressourcen über das Internet bereitzustellen. Dabei sollten vorhandene Technologien verwendet werden, die im Internet einen Quasi-Standard repräsentieren, um so einfach wie möglich alle Sensordaten zu verknüpfen und über eine Webschnittstelle nach außen zur Verfügung zu stellen.

Diese Aufgabenstellung konnte durch einen im Rahmen dieser Diplomarbeit entwickelten Prototypen umgesetzt werden. Dabei wurde zunächst ein Konzept erarbeitet, in dem die verschiedenen Herausforderungen dieser Diplomarbeit untersucht und umgesetzt wurden. Das Konzept wurde anschließend anhand eines Prototyps begründet. Mit dem Prototypen wird eine webbasierte Resource-Management-Platform bereitgestellt, die eine einfache Sensorregistrierung über eine grafische Benutzeroberfläche oder mit Hilfe einer programmatischen Webschnittstelle über das Internet ermöglicht. Für jeden registrierten Sensor werden Adapter implementiert, deren Sensordaten an eine Ressourcenbereitstellungs-Schicht über das HTTP-Protokoll gesendet werden. Anschließend werden die Sensordaten als OSLC-basierte REST-Ressourcen provisioniert. Diese lassen sich als JSON-Objekte darstellen, die eine einfache Weiterverarbeitung ermöglichen. Außerdem werden alle angemeldeten Sensoren als RDF-Repräsentation bereitgestellt, die die Sensoren über Metadaten miteinander verknüpfen.

Durch die einfache Darstellung und Bereitstellung der Sensordaten über URIs wird eine standardisierte Schnittstelle ermöglicht mit der bestehende oder neue Systeme interagieren können.

8.2 Ausblick

Dieses Kapitel gibt einen Ausblick auf mögliche Ansätze und Weiterentwicklungen, die im Rahmen dieser Diplomarbeit nicht umgesetzt werden konnten.

8.2.1 Automatische Sensorregistrierung

Bisher müssen für die Sensoranbindung individuelle Adapter programmiert werden, die die unterschiedlichen Bauweisen sowie Schnittstellen berücksichtigen. Zukünftig wäre eine Lösung vorstellbar, die beispielsweise einem Nutzer erlaubt, über ein Fernzugriff die Adapter auf den Objekten zu installieren und automatisch auszuführen. Dadurch kann der Konfigurationsaufwand für den Nutzer gesenkt werden, in dem vorgefertigte Skripte für die Sensoren bereitgestellt werden.

8.2.2 RDF-Repräsentation erweitern

In dieser Arbeit wurde die Basis für eine RDF-Repräsentation der Sensoren geschaffen. Diese zeigt alle angemeldeten Sensoren, die zu einem OSLC-Service zugehörig sind. Wie in Kapitel 5.4.4.5 beschrieben, repräsentiert dabei der OSLC-Service ein Objekt mit allen darin enthaltenen Sensoren. Zukünftig wäre eine Erweiterung dieser Repräsentation möglich, in dem beispielsweise Verbindungen zu anderen Objekten aus anderen intelligenten Umgebungen oder Produktionsstätten aufgezeigt werden.

8.2.3 Aktoren unterstützen

Im vorgestellten Konzept werden bisher nur Sensoren angebunden und deren Werte ausgelesen. Diese können für eine Situationserkennung in intelligenten Umgebungen genutzt werden. Die Signale der Sensoren werden an die Resource-Management-Platform gesendet und verarbeitet. Mit einer zusätzlichen Funktionalität sollen zukünftig auch Aktoren angesprochen werden. Diese werden wie die Sensoren angemeldet und empfangen Signale, die aus der Resource-Management-Platform gesendet werden. Dadurch lässt sich eine Aktion im Objekt ausführen, die basierend auf Sensordaten beispielsweise vor einer kritischen Situation nicht nur warnt, sondern auch aktiv entgegenwirkt.

8.2.4 Sicherheit

Aufgrund des Umfangs dieser Diplomarbeit konnte das Thema Sicherheit nicht berücksichtigt werden. In Kapitel 5.5 werden Sicherheitskonzepte beschrieben, die zukünftig umgesetzt werden sollen.

8.2.5 Sensordaten streamen

Mit der Resource-Management-Platform wurde die Grundlage geschaffen, Sensordaten mit Hilfe eines Pull-Ansatzes aus dem Cache zu laden und als OSLC-basierte REST-Ressourcen bereitzustellen. Die Plattform lässt sich um eine weitere Komponente erweitern, um Sensorwerte ohne Umweg über den Cache direkt zu streamen. Dadurch lassen sich zum Beispiel Complex-Event-Processing Systeme bedienen, die komplexe Ereignisse zeitnah und kontinuierlich verarbeiten können.

Literaturverzeichnis

[1] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7):97–114, 2009. (Zitiert auf Seite 19)

[2] Nadarajah Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *Security Protocols*, pages 28–41. Springer, 2005. (Zitiert auf Seite 59)

[3] Tim Berners-Lee. Giant global graph. *Decentralized Information Group*, 2007. (Zitiert auf Seite 29)

[4] Denis Butin, Marcos Chicote, and Daniel Le Métayer. Strong accountability: beyond vague promises. In *Reloading Data Protection*, pages 343–369. Springer, 2014. (Zitiert auf Seite 79)

[5] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1, 2001. (Zitiert auf Seite 20)

[6] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, (2):86–93, 2002. (Zitiert auf Seite 20)

[7] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999. (Zitiert auf Seite 12)

[8] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. In *Architectural Styles and the Design of Network-based Software Architectures*, 2000. (Zitiert auf den Seiten 11 und 19)

[9] Elgar Fleisch and Friedemann Mattern. *Das Internet der Dinge*. Springer, 2005. (Zitiert auf Seite 19)

[10] Open Services for Lifecycle Collaboration. Oslc primer. In *OSLC Primer - Learning the Concepts of OSLC*, 2008. (Zitiert auf Seite 30)

[11] Richard Harper. *Inside the smart home*. Springer Science & Business Media, 2006. (Zitiert auf Seite 11)

[12] Pascal Hirmer, Matthias Wieland, Holger Schwarz, Uwe Breitenbücher, and Frank Leymann. Sitrs - a situation recognition service based on modeling and executing situation templates. In *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing (SummerSOC)*, 2015. (Zitiert auf den Seiten 8, 16, 17 und 61)

[13] Simon Jirka and D Nüst. Sensor instance registry discussion paper. *Open Geospatial Consortium*, 2010. (Zitiert auf Seite 80)

[14] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. 2006. (Zitiert auf Seite 22)

[15] Markus Kramer. Nosql-datenbanken. (Zitiert auf Seite 70)

[16] Barry Leiba. Oauth web authorization protocol. *IEEE Internet Computing*, 16(1):74–77, 2012. (Zitiert auf Seite 59)

[17] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004. (Zitiert auf Seite 22)

[18] Dr. Georg Schütte. Industrie 4.0. *Bundesministerium für Bildung und Forschung (BMBF)*, April 2014. (Zitiert auf Seite 11)

[19] Frank Schulze Sebastian Wahle, Thomas Magedanz. The openmtc framework – m2m solutions for smart cities and the internet of things. *The OpenMTC Framework – M2M Solutions for Smart Cities and the Internet of Things*, 2012. (Zitiert auf Seite 79)

[20] Harald Störrle. *UML 2 für Studenten*, volume 320. Pearson Studium, 2005. (Zitiert auf Seite 30)

[21] Ora Lassila Tim Berners-Lee, James Hendler. The semantic web. In *I know what you mean*, 2001. (Zitiert auf Seite 23)

[22] Mark Weiser. Scientific american. In *The Computer for the 21st Century*, 1991. (Zitiert auf Seite 11)

[23] Mark Weiser. The origins of ubiquitous computing research at parc in the late 1980s. In *The origins of ubiquitous computing research at PARC in the late 1980s*, 1999. (Zitiert auf Seite 11)

[24] Matthias Wieland, Holger Schwarz, Uwe Breitenbücher, and Frank Leymann. Towards situation-aware adaptive workflows. In *Proceedings of the 11th Workshop on Context and Activity Modeling and Recognition (COMOREA) at the IEEE Conference on Pervasive Computing (PerCom)*, 2015. (Zitiert auf den Seiten 8, 12, 15 und 16)

Alle URLs wurden zuletzt am 30. 07. 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