UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Richard Mietz

Technologien für eine zustandsbasierte Suche im Internet der Dinge

Dissertation.

R. Mietz

Aus dem Institut für Technische Informatik
der Universität zu Lübeck
Institutsdirektor: Prof. Dr. Erik Mähle

Technologien für eine zustandsbasierte Suche im Internet der Dinge

Inauguraldissertation zur Erlangung der Doktorwürde
der Universität zu Lübeck

— aus der Sektion Informatik/Technik —

vorgelegt von Richard Mietz aus Emden.

Lübeck, Juli 2014

1. Berichterstatter	Prof. D
2. Berichterstatter	Prof. D
Tag der mündlichen Prüfung	18.07.2

Zum Druck genehmigt.
Lübeck, den 18.07.2014

gez. Prof. Dr. Stefan Fischer
— Vorsitzender der Sektionen Informatik / Technik und Naturwissenschaften —

Vorwort

> So eine Arbeit wird eigentlich nie fertig, man muss sie für fertig erklären, wenn man nach der Zeit und den Umständen das Mögliche getan hat.
>
> — *Johann Wolfgang von Goethe*

Zur Anfertigung einer Dissertation gehört nicht nur eine Idee, Fleiß und etwas Glück, sondern auch ein Umfeld, das einen während dieses Lebensabschnittes begleitet. Mein Dank gilt all jenen, die mich bei der Arbeit an dieser Dissertation unterstützt haben, sei es unmittelbar durch Anregungen, Ratschläge oder Kritik oder sei es mittelbar durch die notwendige Ablenkung und Ausgleich in der Freizeit.

An erster Stelle möchte ich meinem Doktorvater Prof. Dr. Kay Römer für das Vertrauen danken, dass er von Anfang an in mich gesetzt hat und für die Möglichkeit im Rahmen meiner wissenschaftlichen Tätigkeit eine Dissertation anzufertigen. Die Freiheiten, die ich durch seinen Betreuungsstil erfahren habe, haben mich in meiner persönlichen und wissenschaftlichen Entwicklung gefördert und zu der vorliegenden Arbeit geführt. Mit seiner kritischen, ehrlichen, aber stets objektiven Meinung umzugehen, war manchmal auf den ersten Blick entmutigend. Bei der zweiten Betrachtung habe ich dadurch allerdings viel gelernt. Ein weiterer Dank geht an Prof. Dr. Volker Linnemann für die Übernahme des Korreferats.

Eine Vielzahl weiterer Personen haben den Promotionsalltag zu einer wundervollen Zeit gemacht. Sowohl meinen ehemaligen Kollegen am Institut für Technische Informatik, insbesondere aus meiner Arbeitsgruppe, als auch den darauffolgenden Kollegen am Institut für Telematik gilt mein Dank für die Freundlichkeit und Hilfsbereitschaft, die mir entgegengebracht wurde. Hervorheben möchte ich Dennis Pfisterer, mit dem ich schon während des Studiums zusammen an den ersten Projekten gearbeitet habe. In der Promotionszeit hatten wir eine inspirierende Zusammenarbeit, die sehr bereichernd und motivierend war. Stefan Fischer möchte ich für die freundliche Aufnahme am Institut für Telematik danken, nachdem meine Arbeitsgruppe am ITI gen Graz gezogen war.

Auch im privaten Umfeld möchte ich einigen Leuten meinen besonderen Dank aussprechen. Als erstes möchte ich meiner Familie, insbesondere meinen Eltern danken, die mir durch ihre finanzielle Unterstützung das Studium ermöglicht haben. Viel wichtiger war jedoch die mentale Unterstützung, die mir zuteilwurde. Für die Unterstützung in Form von Zuhören, Verständnis für lange Arbeitstage und motivierende Worte in schwierigen Situationen sowie die schöne Zeit, die wir gemeinsam verbracht haben, möchte ich mich weiterhin bei Nina und Iris bedanken.

Sport ist all die Jahre ein wichtiger Bestandteil und guter mentaler Ausgleich zur Schreibtischtätigkeit gewesen. Ohne diesen wäre ich nicht so entspannt und gesund über die Dissertationszeit gekommen. Dafür möchte ich besonders allen Mitgliedern der LG Uni Lübeck danken. Die vielen Stunden, laufend am Kanal, im Wald oder an der Straße, haben mir so viel gegeben, dass ich nicht missen möchte.

Letztlich möchte ich Lukas, David, Ronny, Moe, Sergej und Jo hervorheben, die seit meinem Studienbeginn im Jahre 2004 in Lübeck eine zentrale Rolle gespielt hat. Diese Gruppe hat mich zum großen Teil durch das Studium und während der wissenschaftlichen Tätigkeit begleitet. Bei unzähligen Kaffeetreffen und gemeinsamen Mittagessen wurde diskutiert, geredet, gelacht und herumgealbert. David und Ronny gilt zudem ein großer Dank für die Korrekturen und Verbesserungsvorschläge für die vorliegende Arbeit.

Kurzfassung

Die fortschreitende technologische Entwicklung erlaubt immer kleinere Strukturbreiten bei elektronischen Bauteilen. Dies ermöglicht mittlerweile eine wirtschaftliche Produktion von kleinen, fast unauffälligen, eingebetteten Systemen mit Sensorik, Aktuatorik, Funkschnittstelle und autarker Energieversorgung. Die Integration solcher Bauteile in z. B. Smartphones, Autos und Unterhaltungselektronik ist schon heute Realität.

Die Vision des Internet der Dinge, bei der Gegenstände der realen Welt wie Räume, Plätze, Fahrzeuge und selbst Haustiere durch das Einbetten bzw. Anbringen von vernetzter Elektronik eine virtuelle Repräsentation im Internet erhalten und dank der integrierten Sensoren ihren aktuellen Zustand weltweit zur Verfügung stellen können, wird somit zunehmend Wirklichkeit.

Prognosen verschiedener Unternehmen wie Cisco [1], IBM [2] und Gartner [3] gehen davon aus, dass innerhalb des nächsten Jahrzehnts Milliarden bis hin zu Trilliarden von Geräten mit dem Internet verbunden sein werden. In einem Großteil dieser Geräte wird Sensorik integriert sein, die den Zustand der Geräte bzw. der Umgebung erfasst. Die Geräte stellen Metainformationen über sich und die integrierten Komponenten, wie Sensoren, bereit. Dies beinhaltet z. B. Typ, Genauigkeit, Produzent und Herstellungsdatum. Außer diesen statischen Informationen werden durch die Sensorik periodisch oder aufgrund von Ereignissen Messwerte erfasst, ggf. in Form von Aggregation oder Kompression weiterverarbeitet und zum Abrufen bereitgestellt oder an ein Backend verschickt.

Eine Herausforderung bei der Menge an generierten Informationen ist ein skalierbarer und effizienter Umgang mit diesen Sensordaten. Die Daten haben isoliert betrachtet für die Industrie und den Endverbraucher noch keinen Nutzen. Erst durch das Sammeln, Speichern, Verknüpfen und Aggregieren der Daten sowie das Bereitstellen von Diensten, die auf diesen Daten basieren, können neuartige Smart-X-Konzepte wie Smart Cities, Smart Grids, Smart Homes, Smart Agriculture und Smart Healthcare aufbauend auf dem Internet der Dinge realisiert werden.

Ein Basisdienst, der für höherwertige Services benötigt wird, ist eine Suchfunktion für das Internet der Dinge. Dabei ist nicht nur die Suche anhand von statischen Metadaten interessant, sondern auch die Suche nach Gegenständen, die zur Suchzeit einen bestimmten Zustand aufweisen. Im Gegensatz zum traditionellen Internet, in dem sich die Informationen auf Webseiten nur sporadisch

ändern, ist die Änderungshäufigkeit bei Sensoren, die Phänomene der realen Welt überwachen, um ein Vielfaches höher. Heutige Techniken, wie das wiederkehrende Abfragen (engl. crawling), können nicht ohne weiteres übernommen werden, da die verbauten Geräte oftmals autark mit Energie versorgt werden. Die Entwicklung bei Energiespeichern hat im Gegensatz zu der Entwicklung der Strukturbreiten nicht so rasante Fortschritte gemacht. Da die Geräte möglichst klein und unauffällig sein sollten, ist die Energiedichte und damit die Laufzeit der Geräte der limitierende Faktor. Auch weitere Ressourcen, wie die Prozessorgeschwindigkeit, der Speicher und die Funkbandbreite, sind aufgrund der geringen Größe beschränkt. Um die Batterielaufzeit zu maximieren, müssen daher ressourcenschonende Verfahren und Algorithmen entwickelt und die Kommunikation mit den Geräten so gering wie möglich gehalten werden.

In dieser Arbeit werden verschiedene Technologien konzipiert, realisiert und evaluiert, welche die Basis für eine skalierbare, zustandsbasierte Suche im Internet der Dinge bilden. Im ersten Schritt wird eine Nutzerbefragung zum Thema *Suche im Internet der Dinge* durchgeführt, welche die Anforderungen an die verschiedenen Komponenten aus Nutzersicht untersucht. Anschließend werden flexible Mechanismen vorgestellt, um auf verschiedenen Plattformen aus rohen Sensorwerten höherwertige Zustände der realen Welt zu ermitteln, die für den Nutzer semantisch verständlich sind. Es wird erläutert, wie durch die Verwendung von Wahrscheinlichkeitsmodellen für die Zustände bei der Suche Ressourcen gespart werden können. Zur Erzeugung dieser Modelle werden ebenfalls mehrere Verfahren präsentiert. Es wird eine Möglichkeit der Modellierung von Metainformationen und dynamischen Sensordaten, wie den Zuständen und Modellen, auf Basis von Semantic-Web-Technologien vorgestellt. Diese Technologien erlauben eine einfache Verknüpfung mit anderen Datenquellen. Dadurch können die Sensordaten in einen Kontext gebracht und sogar neue Fakten automatisch aus existierendem Informationen hergeleitet werden. Die Speicherung der anfallenden Daten wird durch DecentSPARQL, eine verteilte semantische Datenbank, realisiert. Die anschließende Evaluation zeigt, dass das Konzept von DecentSPARQL skaliert und für die prognostizierte Menge an Daten geeignet ist. Abschließend werden zwei Schnittstellen zur Nutzung der Suche beschrieben. Dies sind eine REST-Schnittstelle für Entwickler und eine grafische Benutzungsoberfläche für die Endnutzer. Eine Evaluation der Gebrauchstauglichkeit der Benutzungsoberfläche mit 20 Personen zeigt, dass die zu Beginn in der Umfrage identifizierten Anforderungen zufriedenstellend umgesetzt werden konnten.

Abstract

Ongoing technological advances constantly allow higher transistor densities in electronic components. Thus, an economic production of small, nearly invisible, embedded systems with sensors, actuators, radio interface and self-sufficient energy supply becomes possible. The integration of these components in everyday objects such as smartphones, cars and consumer electronics is already reality.

The vision of the Internet of Things is that objects of the real world such as rooms, places, vehicles and even pets get a virtual representation in the Internet. This should be achieved by equipping them with networked electronics. Thanks to integrated sensors, the *Things* will be able to publish their current state on the Internet thus making it available worldwide and in real-time.

Forecasts of various companies such as Cisco [1], IBM [2] and Gartner [3] expect that, within the next decade, billions or even up to trillions of devices will be connected to the Internet. A majority of these will have sensors integrated which are capable of perceiving the device's or the environment's state. The devices will provide meta information such as type, accuracy, producer and date of manufacturing about themselves and the built-in components. Besides such static information, the sensors will collect measurements in a periodic or event-based fashion. Where applicable, sensors may use aggregation or compression. Finally they provide means for downloading the data or send it actively to a backend.

A challenge in the upcoming Internet of Things will be the scalable and efficient handling of the huge amount of generated data. The isolated data has no benefit for the industry and the end-user. Only by collecting, storing, linking and aggregating data as well as offering services on the basis of this information, novel smart-x concepts such as Smart Cities, Smart Grids, Smart Homes, Smart Agriculture and Smart Healthcare can be realized.

A fundamental service, which is needed for more sophisticated services, is a search function for the Internet of Things. Not only the search for static information is important, but also the search for objects, which perceive a certain state at query time. In contrast to the traditional Internet, where information on web pages is changing only sporadically, the change frequency of sensors, which monitor phenomena of the real world, can be orders of magnitude higher. Current techniques such as periodic crawling consume resources on a regular basis and often collide with the energy constraints of battery powered devices.

Unfortunately, the progress in energy storage is much slower than the advances in transistor densities. Because embedded devices should be small in size and inconspicuous, the energy density and therefore the lifetime of devices is the limiting factor. Additionally, also other resources such as processor speed, storage capacity and radio bandwidth are limited due to the small size. Thus, to prolong the battery lifetime, new resource-efficient mechanisms and algorithms need to be developed and the communication between devices should be kept to a minimum.

In this work different technologies are designed, realized and evaluated to constitute the foundation for a scalable, state-based search for the Internet of Things. First, a user survey on the topic *Search in the Internet of Things* is conducted to investigate the requirements for the different components from a user's perspective. Afterwards, flexible mechanisms to infer semantically understandable high-level states of the real world from raw sensor readings on different platform are presented. This is followed by the introduction of state probability models and a discussion on how the use of these can minimize resource consumption when searching the Internet of Things. Different methods to generate such models are presented subsequently. Next, a modeling approach for static metadata as well as dynamic sensor data based on semantic web technologies is described. These technologies allow for simple linking with other data sources. Hence, the sensor data can be combined with other information and even new facts can be inferred automatically. Data storage is handled by DecentSPARQL, a distributed semantic database, which is evaluated to show the feasibility of the approach. Finally, two interfaces to use the search service are described. This is on the one hand a REST interface for developers and on the other hand a graphical user interface for consumer. A usability evaluation of the user interface with 20 participants shows that the requirements identified with the help of the user survey are realized in a satisfying manner.

Inhaltsverzeichnis

Vorwort **v**

Kurzfassung **vii**

Abstract **ix**

1. Einleitung **1**
1.1. Suche als Dienst für das Internet der Dinge 2
1.2. Stand der Forschung 4
1.2.1. Existierende Systeme 4
1.2.2. Vergleich der Systeme 11
1.3. Herausforderungen 16
1.4. Aufbau und Beiträge dieser Arbeit 17

2. Grundlagen **21**
2.1. Technologische Entwicklung 21
2.1.1. Sensornetze 23
2.1.2. Internet der Dinge 27
2.2. RESTful Webservices 30
2.2.1. Constrained Application Protocol 30
2.2.2. CoAP Observe 31
2.2.3. CoAP Conditional Observe 32
2.3. Semantic Web 34
2.3.1. Resource Description Framework 34
2.3.2. SPARQL Protocol and RDF Query Language 37
2.3.3. GeoSPARQL 39

3. Nutzerorientierte Entwicklung **41**
3.1. Verwandte Arbeiten 42
3.2. Umfrage 43
3.2.1. Aufbau der Umfrage 43
3.2.2. Ergebnisse der Umfrage 45
3.2.3. Fazit 59

4. Zustandsermittlung und -wahrscheinlichkeiten **61**
4.1. Verwandte Arbeiten 62

4.2. Verfahren zur Zustandsermittlung 65
4.2.1. Lousy Camera Framework: Zustandsermittlung auf kamerabestückten Sensorknoten 67
4.2.2. High-Level-State-Option für das Constrained Application Protocol 82
4.2.3. Skriptbasierte Zustandsermittlung 96
4.2.4. Browser-Erweiterung zur Erzeugung virtueller Sensoren . 104
4.2.5. Fazit 109
4.3. Zustandswahrscheinlichkeiten 111
4.3.1. Definition 111
4.3.2. Korrelationsbasierte Modelle mit Bayes'schen Netzen . . 114
4.3.3. Modellgenerierung mittels Skriptsprache 119
4.3.4. Modellgenerierung mittels CoAP 119
4.3.5. Fazit 121

5. Informationsmodellierung 123
5.1. Verwandte Arbeiten 124
5.2. RDF-Modellierung 124
5.2.1. Metadaten 125
5.2.2. Zustände 126
5.2.3. Historie von Zuständen 128
5.2.4. Basiswahrscheinlichkeitsmodell 131
5.2.5. Temporale Erweiterungen 132
5.2.6. Räumliche Erweiterungen 135
5.2.7. Erweiterung um Korrelationen 136
5.2.8. Generelle Erweiterbarkeit 137
5.2.9. Zustandsabfrage 139
5.2.10. Rankingfaktoren 140
5.2.11. Evaluation 143
5.2.12. Fazit 144
5.3. Wahrscheinlichkeitsmodellgenerierung mittels SPARQL 144
5.3.1. Fazit 146

6. Informationsspeicherung und -präsentation 149
6.1. Verwandte Arbeiten 151
6.2. DecentSPARQL 152
6.2.1. GeoSPARQL-Erweiterung für LUPOSDATE 154
6.2.2. Evaluation 155
6.2.3. Fazit 158
6.3. Suchschnittstelle 159
6.3.1. REST-Schnittstelle 159
6.3.2. Grafische Benutzungsschnittstelle 160
6.3.3. Fazit 165

7. Zusammenfassung und Ausblick 167
7.1. Beiträge der Arbeit 168
7.2. Ausblick 171

A. Anhang **173**
A.1. Umfrage 174
A.2. XML Schemata 179
A.3. Aufgabenstellungen der Usability-Evaluation 181
A.4. Fragebogen der Usability-Evaluation 182

B. Verzeichnisse **183**
Tabellenverzeichnis 183
Abbildungsverzeichnis 185
Quelltextverzeichnis 189
Abkürzungsverzeichnis 191
Eigene Publikationen 193
Literaturverzeichnis 197

1 Einleitung

> A lot of times people don't know what they want until you show it to them.
>
> — *Steve Jobs*

Die technologische Entwicklung der letzten Jahrzehnte hat zu einer fortschreitenden Miniaturisierung technischer Komponenten und einer immer stärker werdenden Durchdringung unserer Umwelt mit Elektronik geführt. Am Anfang der Computerentwicklung standen Großrechner, die von wenigen Personen genutzt wurden. Durch die Entwicklung des Personal Computers wurde der private Gebrauch vorangetrieben. Der nächste Schritt war die mobile Nutzung durch die Erfindung von Notebooks, Tablets und Smartphones. Parallel dazu haben Fortschritte in der Funktechnologie und die Entstehung des Internets zum rasanten Anstieg der Nutzerzahlen und der Omnipräsenz von computerisierten Systemen geführt.

Die entwickelten Technologien finden zudem zunehmend in immer mehr Alltagsgegenständen, wie Waschmaschinen, Kühlschränke und Autos, als eingebettete Systeme Verwendung. Hier dienen sie der Überwachung, Steuerung oder Regelung. Dies wird durch die Weiterentwicklung im Bereich der Sensorik und Aktuatorik ermöglicht.

Auf wissenschaftlicher Ebene sind seit Beginn des 21. Jahrhundert drahtlose Sensornetze ein großes Forschungsthema. Dabei werden viele sogenannte Sensorknoten durch Funktechnologie miteinander zu einem Netz verbunden. Die Sensorknoten sind u. a. mit Prozessor, Speicher, autarker Energieversor-

gung und Sensorik ausgestattet. Dies erlaubt ihnen Umweltparameter mittels Sensoren zu messen, diese zu speichern, weiterzuverarbeiten und an andere Sensorknoten bzw. eine Basisstation zu verschicken. Sensornetze haben vielfältige Einsatzmöglichkeiten: von der Natur- und Tierüberwachung über das Monitoring von Gebäuden und anderen Bauwerken, wie Brücken, bis hin zur gesundheitlichen Beobachtung von Patienten und dem Einsatz in der Industrie, wie in der Logistik.

Anfangs waren die Sensorknoten und Sensornetze isoliert, aber nach und nach wurden diese durch Protokollübersetzung bzw. Adaption der Internetprotokolle in das Internet integriert. Auch die oben beschriebenen, in die Alltagsgegenstände eingebetteten Systeme werden zunehmend mit dem Internet verbunden und erlauben so, Informationen über sie online abzurufen. Marktforschungsunternehmen wie Gartner [3] und Firmen wie Cisco [1] und IBM [2] schätzen, dass innerhalb des nächsten Jahrzehnts Milliarden oder sogar Trilliarden von internetfähigen Geräten, vor allem Alltagsgegenstände, verbunden sein werden. Diese Vision nennt sich das Internet der Dinge[1] (IoT). Das IoT stellt eine Erweiterung des heutigen Internets dar, bei der die Dinge der realen Welt eine virtuelle Repräsentation besitzen, über die sie statische Informationen, wie Typ und Hersteller, aber auch dynamische Informationen, wie Ort und Zustand, in Echtzeit bereitstellen. Aber selbst der Zustand von Personen und Tieren, wie z. B. die Position oder die derzeitige Aktivität, könnte erfasst werden.

Die prognostizierte Menge an Geräten wird höchstwahrscheinlich nicht nur für industrielle Zwecke, beispielsweise in der Logistik, und gesellschaftliche Zwecke, wie in der Wettervorhersage, genutzt werden, sondern wird auch für den persönlichen Gebrauch zur Verfügung stehen. Die Vision ist, dass die Geräte in Haushalten und Städten miteinander kommunizieren können, um u. a. das Leben der Bewohner komfortabler und sicherer zu gestalten und Ressourcen, wie Wasser und Energie, zu sparen.

1.1. Suche als Dienst für das Internet der Dinge

Heute sind Suchmaschinenseiten die am häufigsten genutzten Webseiten im Internet [4]. Diese erlauben das Finden von Internetseiten und Dateien anhand statischer Metadaten, wie Inhalt, Name, Dateityp und Alter. Es ist eine naheliegende Idee, eine Suchmaschine zu entwickeln bzw. bestehende so zu erweitern, dass auch die Geräte und Daten eines zukünftigen IoT einbezogen werden. Der erste, noch sehr an heutigen Suchmaschinen orientiert Schritt wäre, dass die Sensoren bzw. Geräte des IoT anhand ihrer statischen Metadaten auffindbar sind. Ein Beispiel ist die Suche nach Sensoren in der näheren Umgebung, die Lichteinstrahlung messen. Sobald der Nutzer das Ergebnis mit den Lichtsensoren erhalten hat, kann er die aktuellen Sensorwerte von den Sensoren abfragen,

[1] Im Englischen: *Internet of Things*, abgekürzt *IoT*. *Internet der Dinge*, *Internet of Things* und *IoT* werden in der vorliegenden Arbeit synonym verwendet.

sofern diese eine Funktion dafür bereitstellen. Somit ist es für ihn möglich, sonnige und schattige Plätze zu finden.

Dieses Vorgehen hat drei Nachteile für den Nutzer einer solchen Suchmaschine:

1. Der Arbeitsfluss ist nicht sehr komfortabel für den Nutzer. Nacheinander müssen alle Sensoren der Ergebnismenge manuell abgefragt werden, um schlussendlich hoffentlich einen Sensor mit dem für den Nutzer passenden Ausgabewert zu finden.
2. Es ist nicht (direkt) ersichtlich, was der Sensor beobachtet. Ein Lichtsensor könnte in einem Raum installiert sein, um das Bürolicht zu überwachen. Dieser Sensor wäre für das Beispielszenario nutzlos, da er nicht die Sonneneinstrahlung misst und somit keine Aussage über sonnige oder schattige Orte erlaubt.
3. Der Nutzer würde in dem Beispielszenario die Werte von Lichtsensoren empfangen. Diese messen die Beleuchtungsstärke typischerweise in Lux. Der Nutzer bräuchte Vorwissen über die Messeinheit, um den Wert als *sonnig* oder *schattig* interpretieren zu können.

Zur Bewältigung dieser drei Problematiken sollte die Suchmaschine folgende Konzepte umsetzen:

1. Die Suche sollte nicht nur für statische Daten ermöglicht werden, sondern auch für dynamische, wie dem aktuellen Sensorwert. Dadurch müsste der Nutzer nicht mehr selbst die Werte von Sensoren abrufen.
2. Im IoT besitzen die Dinge der realen Welt und nicht die einzelnen Sensoren virtuelle Repräsentationen ihrer selbst im Internet. Des Weiteren ist die Position eines Sensors nicht zwingend mit der Position des Objektes, das dieser überwacht, identisch. Eine Kamera, die den Belegungszustand von Parkplätzen überwacht, kann in einiger Entfernung angebracht sein. Deshalb sollte das Suchsystem nicht die Suche nach Sensoren, sondern nach Objekten ermöglichen, die durch diese Sensoren überwacht werden.[2]
3. Bei einer Suchmaschine, welche die Suche anhand des aktuellen Sensorrohwertes anbietet, benötigt der Nutzer schon während der Sucheingabe ein Verständnis der Ausgabe des Sensors für eine Entität. Für das Beispielszenario zur Suche nach sonnigen Plätzen muss der Nutzer wissen, in welcher Einheit die Lichtstärke gemessen wird und ab welchem Wert von *sonnig* gesprochen werden kann. Für den Nutzer ist demnach nicht der Rohwert eines Sensors von Bedeutung, sondern der höherwertige Zustand einer Entität, da dieser für ihn direkt begreifbar und semantisch verständlich ist.

Durch eine Suchmaschine, welche die beschriebenen Konzepte umsetzt, eröffnet sich eine Vielzahl von neuen Anwendungen und Möglichkeiten. Beispiele

[2] Die Begriffe Entität, Ding, Objekt und Gegenstand werden in der vorliegenden Arbeit im Kontext der Suche synonym verwendet.

sind die Suche nach freien Parkplätzen mit Hilfe von Parkplatz- und Parkhaussensoren, die Suche nach Restaurants mit freien Tischen, das Finden von sonnigen bzw. schattigen Plätzen oder die Abfrage der Wartezeit an Freizeitparkattraktionen, beim Arzt oder am Imbiss.

1.2. Stand der Forschung

Im weiteren Verlauf wird der Stand der Forschung zum Thema *Suche im IoT* vorgestellt. Dazu werden Arbeiten diskutiert, die Ansätze und Realisierungen für eine solche Suchmaschine vorstellen. Eine anschließende Analyse und Bewertung der Systeme erörtert die Nachteile und zeigt offenen Fragen auf. Darauf aufbauend werden in Abschnitt 1.3 die Herausforderungen für eine Suchmaschine für das IoT beschrieben.

1.2.1. Existierende Systeme

Es existieren einige wissenschaftliche Prototypen [5–17], die eine Echtzeitsuche nach Entitäten der realen Welt ermöglichen. Auch kommerzielle Anbieter erkennen zunehmend den Trend des IoT, entwickeln Dienste zur Integration von Sensoren in das Internet, zur Verarbeitung von Sensordaten und zur Suche nach Sensoren. Die Funktionsweise und Leistungsmerkmale dieser Systeme werden in den folgenden Abschnitten präsentiert, verglichen und evaluiert. Zuerst werden sechs Systeme gezeigt, welche eine schlüsselwort- oder entitätsbasierte Suche erlauben, d. h. eine Suche anhand statischer Beschreibungen bzw. nach bestimmten Objekten. Anschließend werden drei Systeme vorgestellt, die eine zustandsbasierte Suche erlauben, also Entitäten finden können, die zur Suchzeit einen gewissen Zustand aufweisen. Die Systeme werden jeweils in chronologischer Reihenfolge ihrer Veröffentlichung aufgelistet.

MAX

MAX [5] erlaubt die Suche nach verlorenen Gegenständen. Dabei nutzt es eine Drei-Schichten-Architektur. Auf der untersten Ebene werden die suchbaren Objekte mit Radio-Frequency-Identification-Tags (RFID-Tags) bestückt, welche eine Beschreibung des Gegenstandes beinhalten. Auf der mittleren Schicht sind sogenannte *substations* angesiedelt. Dies sind Sensorknoten, die an vornehmlich statischen Objekten, wie Sofas und Tischen, angebracht werden und die mit den RFID-Tags kommunizieren können. Die oberste Schicht bilden *base stations*, die dauerhaft statisch sind. Eine *base station* ist üblicherweise für die *substations* in einem Raum zuständig.

Suchanfragen werden von der obersten bis zur untersten Schicht und wieder zurück gereicht. Die Anzahl gefundener Schlüsselwörter in der Beschreibung des Gegenstandes wird als Rankingkriterium genutzt. Wird ein passender Gegen-

stand mittels der Schlüsselwörter gefunden, so wird der ungefähre Ort anhand der *substation* bestimmt, die den besten Signalempfang (RSSI-Wert) für den Gegenstand misst.

In einer Nutzerstudie mit zehn Teilnehmern wurde die Funktionalität des Systems untersucht. Dazu bat man die Probanden, zwei Objekte in einem Raum mit Hilfe von *MAX* bzw. ohne zu finden. Unter Zuhilfenahme des Systems ließen sich alle Objekte finden, während bei der manuellen Suche drei Teilnehmer aufgaben. Die Zeit bis zum Finden war mit dem System durchgehend geringer. Eine Evaluation der Benutzerzufriedenheit fand allerdings nicht statt. Zudem ist, wie die Autoren selbst schreiben, die Teilnehmerzahl zu gering für aussagekräftige Ergebnisse.

Der Nachteil des Systems ist die Einschränkung auf die Suche nach Gegenständen anhand statischer Schlüsselwörter. Außerdem ist die Skalierbarkeit sehr eingeschränkt, da Suchanfragen immer in das gesamte System geflutet werden.

Objects Calling Home

Objects Calling Home (OCH) [6] ist ähnlich dem *MAX*-System, da ebenfalls eine Suche nach verlorengegangenen Objekten ermöglicht wird. An die Objekte werden Sensorknoten angebracht. Ein zentraler Server nimmt Anfragen entgegen und verteilt diese an *object nodes*. Im Beispielszenario werden Mobiltelefone mit Bluetooth als *object nodes* verwendet. Die *object nodes* versuchen, das zu suchende Objekt in ihrer Umgebung über Bluetooth zu finden. Im Gegensatz zu *MAX* beinhalten die Sensorknoten allerdings keine Beschreibung der Objekte, sondern nur die Identität des Objektes. Somit ist es nicht möglich, mittels Schlüsselwörtern allgemein nach Objekten zu suchen, sondern nur mit Hilfe der Identität jeweils nach einem bestimmtem Objekt.

OCH versucht durch verschiedene Mechanismen den Ressourcenverbrauch zu optimieren. Einerseits kann eine maximale Suchzeit t, andererseits eine maximale Nachrichtenzahl q angegeben werden. Ist die Beantwortung der Anfrage unter Einhaltung dieser Parameter nicht möglich, so wird sie aus dem System gelöscht und kein Ergebnis zurückgeliefert. Weiterhin werden drei Heuristiken beschrieben, um den Suchraum einzugrenzen. Die ersten beiden beruhen auf einem Schlüsseltausch mit dem Nutzer, um diesen zu identifizieren. Suchanfragen des Nutzers werden dann vornehmlich in die Bereiche geroutet, an denen sich der Nutzer häufig aufhält bzw. dorthin, wo dieser zuletzt war, da es wahrscheinlich ist, dass er dort das gesuchte Objekt verloren hat. Die dritte Heuristik routet die Anfrage zu der Region, in der das Objekt zuletzt gesehen wurde. Der Nutzer hat die Möglichkeit die Heuristiken nach seinen Wünschen zu gewichten.

OCH hat den gleichen Nachteil wie *MAX*: die Einschränkung auf eine statische Suche nach Objekten. In diesem System müssen die suchbaren Objekte dem Nutzer sogar bekannt sein. Als Verbesserung der Skalierbarkeit werden im Gegensatz zu *MAX* Heuristiken verwendet, um die Anfrage nicht in das komplette

Netz zu fluten. Diese Heuristiken gehen aber mit neuen Nachteilen einher, da bei einigen die Objekte bzw. die Nutzer verfolgt werden, was einerseits Ressourcen verbraucht, andererseits die Privatsphäre verletzen kann. Zwar kann durch die Anfrageparameter *t* und *q* der Ressourcenverbrauch reduziert werden, allerdings kann es dadurch passieren, dass ein Objekt nicht gefunden wird, selbst wenn es theoretisch auffindbar wäre.

Snoogle/Microsearch

Das Suchsystem *Snoogle* [7, 8] / *Microsearch* [9, 10] erlaubt, ähnlich wie *MAX*, eine schlüsselwortbasierte Suche nach Entitäten der realen Welt. Auf der unteren Schicht der Zwei-Schichten-Architektur sind Gegenstände, die suchbar sein sollen, wie z. B. Bücher oder Ordner, mit Sensorknoten ausgestattet. Neben diesen *Object Nodes* genannten Elementen sind zudem *Index Points* auf dieser Schicht angesiedelt. *Object Nodes* speichern beschreibende Schlüsselwörter zu den Objekten, an denen sie angebracht sind. Die *Index Points* sind statische Systeme, die *Object Nodes* in ihrer Kommunikationsreichweite verwalten. Die *Index Points* wiederum werden von dem *Key Index Point* auf der oberen Schicht administriert. Da die *Object Nodes* an mobilen Entitäten angebracht sind, treten diese dynamisch einem *Index Point* in ihrer Nähe bei und verlassen diesen auch wieder. Um zu prüfen, ob ein *Object Node* noch aktiv ist, gibt es zwei Varianten: Im *timer*-Modus schickt der *Object Node* in regelmäßigen Abständen seine Daten an den *Index Point*, während im *beacon*-Modus der *Index Point* von allen *Object Nodes* regelmäßig die textuelle Beschreibung mit einer Broadcast-Nachricht anfordert. Aus den Schlüsselwörtern erzeugt der *Index Point* einen invertierten Index, um eine effiziente Suche anbieten zu können. Dieser Index wird auch an den *Key Index Point* weitergereicht, so dass dieser Indizes aller *Index Points* besitzt.

Mit einem Smartphone kann entweder lokal ein *Index Point* oder global der *Key Index Point* angefragt werden. Dabei kann der Nutzer spezifizieren, wie viele Ergebnisse *k* er erhalten möchte. Bei der lokalen Suche gibt der *Index Point* die ersten *k* Ergebnisse zurück, wobei die Sortierung anhand der Zahl übereinstimmender Suchwörter erfolgt. Auch der *Key Index Point* gibt die besten *k* Ergebnisse zurück. Um dabei nicht alle Treffer von allen *Index Points* abrufen zu müssen, geht er wie folgt vor: Zu Beginn wird der jeweils am höchsten eingestufte *Object Node* von allen *Index Points* abgefragt. Daraus wird eine sortierte Liste erzeugt. Enthält diese Liste weniger als *k* Ergebnisse, so werden die Restlichen wie folgt bestimmt: Zuerst wird an den *Index Point*, von dem der am höchsten eingestufte *Object Node* geschickt wurde, der zweite Eintrag der sortierten Liste geschickt. Dieser *Index Point* schickt daraufhin alle *Object Nodes*, die besser, also höher, eingestuft sind als der Gesendete, zum *Key Index Point*. Diese *Object Nodes* werden in die Liste einsortiert. Anschließend wird der Prozess mit dem zweiten *Index Point* und dem ursprünglich dritten *Object Node* der sortierten Liste fortgeführt, usw.

Nachteile des Systems sind die Einschränkung auf eine statische Suche nach Schlüsselwörtern und eine Zentralisierung durch den *Key Index Point*, wodurch das System schlecht skaliert.

Distributed Image Search

Statt auf einer Suche mittels Schlüsselwörtern baut das System *Distributed Image Search (DIS) [11]* auf einer Suche mit Bildern auf. Dazu nehmen kamerabestückte Sensorknoten Fotos der Umgebung auf. Nutzer können Suchanfragen in Form von Bildern an das System stellen. Dieses versucht Sensoren als Ergebnis zurückzuliefern, deren geschossene Bilder die größte Ähnlichkeit zu dem Suchbild aufweisen. Somit kann nach Objekten und in gewissem Maße auch nach Zuständen der realen Welt gesucht werden, nämlich sofern das Bild den gesuchten Zustand, wie z. B. einen leeren Parkplatz, widerspiegelt.

Neben den Sensorknoten existiert eine zentrale Basisstation. Statt Bilder, die üblicherweise mehrere Kilobyte (kB) groß sind, an diese zu übertragen, was zu großen Ressourcenaufwand bedeuten würde, werden interessante Merkmale, *visterms*, aus den Bildern extrahiert. Diese kompakte Repräsentation stellt eine Analogie zu Schlüsselwörtern dar. Für ein Bild werden rund 200 *visterms* berechnet, die nur rund 1,6 kB Speicher benötigen. Je mehr *visterms* im Suchbild und im aufgenommenen Bild übereinstimmen, desto besser ist der *ranking score* für das Bild. Gibt es keine übereinstimmenden *visterms*, so ist die Ergebnismenge leer. Die Suche kann in zwei Modi betrieben werden. Bei der zentralen Variante werden die *visterms* von den Sensorknoten zur Basisstation gesendet, die, ähnlich wie bei *MAX*, einen invertierten Index aufbaut. Dagegen werden bei der dezentralen Variante die *visterms* des Suchbildes zu den Sensorknoten geschickt. Diese schicken daraufhin die *k* besten *ranking scores* an die Basisstation zurück. Die Basisstation fügt nach dem Empfang die Ergebnisse der Sensorknoten zusammen. Anschließend werden die Fotos der besten Ergebnisse von den Sensorknoten abgerufen.

Der Nachteil von *DIS* ist wie bei *MAX*, die Skalierbarkeit, da immer alle Sensorknoten in die Suche einbezogen werden. Während der Textvergleich von Suchwörtern mit Schlüsselwörtern bei den bisher vorgestellten Systemen für den Nutzer nachzuvollziehen ist, sind die technischen Details beim Vergleich mit Hilfe von *visterms* nicht so einfach für den Nutzer zu verstehen. Außerdem kann es für Nutzer schwierig sein, geeignete Suchbilder bereitzustellen. Des Weiteren ist der Ressourcenverbrauch durch die Extraktion der *visterms* sowie das Speichern und Verschicken von Bildern höher als bei anderen Systemen.

Xively

Xively [12], vormals *Pachube* (02/2008–05/2012) und *Cosm* (05/2012–05/2013), ist ein cloud-basiertes Sensorverzeichnis, bei dem Nutzer ihre Sensoren registrieren können. Für jeden Sensor können Metainformationen, wie Beschreibung,

Titel und Ort, eingetragen werden. Anschließend können die Messwerte der Sensoren auf zwei Arten an den Dienst übermittelt werden. Entweder senden die Sensoren aktiv ihre Werte an den Dienst oder er kann die Daten von den Sensoren abrufen, sofern diese über ein entsprechendes Application Programming Interface (API) verfügen.

Die historischen Sensorwerte stehen, je nach Einstellung, nur für den Besitzer oder für weitere Nutzer zur Verfügung. Dabei können die Daten direkt auf der Webseite des Dienstes visualisiert oder mittels API von Anwendungen genutzt werden. Eine schlüsselwortbasierte Suche erlaubt das Finden von Sensoren anhand der vom Nutzer eingegebenen Beschreibung.

Der Nachteil der Plattformen ist vor allem die statische Suche. Eine Suche anhand von Werten und/oder einer bestimmten Region ist nicht vorhanden. Ein weiterer Nachteil ist, dass die Nutzer die Sensoren unterschiedlich detailliert beschreiben. So wird auf *Xively* der Sensortyp von einigen Personen im Titel angegeben, andere schreiben ihn in den Beschreibungstext und manche geben ihn gar nicht an. Dies führt zu einer Inhomogenität der Daten wodurch eine automatische Verarbeitung mittels API in Anwendungen schwierig bis unmöglich wird. Der volle Funktionsumfang ist bei *Xively* zudem kostenpflichtig.

Presto

Presto's [13] untere Schicht seiner Zwei-Schichten-Architektur besteht aus Sensorknoten, während die obere Schicht vom Presto-Proxy gebildet wird. Dieser sammelt Daten der Sensoren, um daraus zeitreihenbasierte Vorhersagemodelle zu berechnen. Für die Modelle wird zusätzlich ein Fehlerintervall berechnet, in den die Vorhersagen fallen. Die Modellparameter werden zurück zu den Sensoren geschickt, so dass diese periodisch ihre Messwerte mit den Vorhersagen vergleichen können und im Falle von Ausreißern diese dem Proxy mitteilen. Nach einer einstellbaren Anzahl an Ausreißern wird ein Modell neu berechnet.

Anfragen bestehen aus dem Sensor, für den der Messwert ermittelt werden soll, sowie einem Fehler, der tolerierbar ist. Fällt der Fehler in das berechnete Fehlerintervall, so wird das Vorhersagemodell genutzt, anstatt den Sensor zu kontaktieren. Dadurch wird Kommunikation mit dem Sensornetz eingespart. Andernfalls muss der Wert vom Sensor abgerufen werden.

Der Proxy speichert alle Sensorwerte, die er aus dem Netzwerk erhält. Diese nutzt er zur Modellberechnung und zur Beantwortung von Anfragen nach historischen Daten.

Der Nachteil des Systems ist zum einen das initiale Übertragen von ausreichend Daten zur Modellberechnung, zum anderen die Spezifizierung des Sensors für den ein Wert ermittelt werden soll. Das System könnte durch eine Modifizierung allerdings als Suchmaschine für Sensorwerte genutzt werden. Dafür müsste eine Anfrage statt des Sensors den gesuchten Wert enthalten. Die Vorhersagemodelle

aller Sensoren könnten dann dazu genutzt werden Sensoren zu finden, die mit hoher Wahrscheinlichkeit aktuell diesen Wert messen.

Dyser

Eine Suchmaschine, die nicht nur die Suche nach Sensoren anhand von Metadaten erlaubt, sondern auch die Suche anhand des aktuellen Zustandes, wird von Ostermaier et al. [14] vorgestellt. Das *Dyser*-System bindet Sensoren über ein Sensor Gateway an das Internet an. Die Sensoren überwachen Entitäten der realen Welt. Dabei wird für jede überwachte Entität eine Webseite generiert. Diese enthält zum einen statische Informationen der Entität, wie eine Beschreibung und die Zustände, welche die Entität annehmen kann. Zum anderen beinhaltet sie dynamische Informationen in Form eines Wahrscheinlichkeitsmodells, mit dem die Wahrscheinlichkeiten für jeden annehmbaren Zustand zu einem bestimmten Zeitpunkt berechnet werden können. Im Falle des Parkplatzbeispiels sind diese Informationen eine Ortsangabe, die Information, dass es sich um einen Parkplatz handelt, die beiden Zustände *frei* und *belegt* sowie ein Modell, das Wahrscheinlichkeiten für diese Zustände berechnen kann. Dieses Modell wird von dem Gateway mittels historischer Daten des Sensors erstellt. Dabei stehen drei Modelltypen zur Verfügung: Das einfachste berechnet die Häufigkeit von jedem Zustand, während die Mächtigeren versuchen, periodische Muster (entweder eines oder mehrere) in den Zustandsdaten zu identifizieren.

Die *Crawler*-Komponente von *Dyser* indiziert die Webseiten inklusive der Modelle und speichert diese in einer Datenbank ab. Nutzer können Anfragen stellen, die aus statischen und dynamischen Teilen aufgebaut sind. Der statische Teil sind Schlüsselwörter, welche die Entität beschreiben (z. B. Parkplatz) während der dynamische Teil die Zustandsart (z. B. Belegungsstatus) und den gesuchten Zustand (z. B. frei) umschreibt. Die *Resolver*-Komponente sucht anhand des statischen Teils passende Webseiten der Entitäten in der Datenbank. Danach wird das sogenannte Sensor Ranking [18] angewendet. Dabei werden die entsprechenden Modelle für die Entitäten, die anhand des dynamischen Teils vom *Resolver* ausgewählt werden, ausgeführt und eine Wahrscheinlichkeit für den gesuchten Zustand berechnet. Die Liste der Entitäten wird nach ihrer Wahrscheinlichkeit, dass sie gerade den Zustand haben, sortiert. Als letzter Schritt werden die Sensoren, welche die jeweilige Entität beobachten, in absteigender Reihenfolge der Wahrscheinlichkeit abgefragt, bis k Sensoren gefunden wurden, die aktuell den gesuchten Zustand messen. Sofern die Wahrscheinlichkeitsmodelle eine hohe Anpassungsgüte haben, kann dadurch die Zahl der anzufragenden Sensoren im Gegensatz zum zufälligen Abfragen reduziert werden.

Ein Nachteil ist die Berechnung der Modelle auf dem Gateway, da dafür ausreichend Daten von den Sensoren vorliegen müssen. Bei der Berechnung auf den Sensorknoten würde dagegen die Kommunikation minimiert werden. Auf die Modellierung und Erweiterbarkeit der Wahrscheinlichkeitsmodelle wird nicht eingegangen. Die restlichen verwendeten Technologien sind dagegen

zum Großteil standardisiert. Ein Nachteil für den Nutzer ist allerdings, dass er sich an eine spezielle Suchsyntax halten muss.

Gander

Das *Gander*-System [15, 16] ist eine Sucharchitektur für Personalized Networked Spaces (PNetSs). PNetSs sind pervasive Umgebungen, in dem das Smartphone eines Nutzers mit anderen digitalen Ressourcen, wie weiteren Smartphones, Sensoren und Aktuatoren, via W-LAN ad-hoc ein lokales Netz etabliert, um Dienste anzubieten und zu konsumieren. Eine Internetverbindung ist dafür nicht zwingend notwendig.

Für iOS-Geräte steht das *myGander*-Interface zur Verfügung. Es bietet dem Nutzer die Möglichkeit, Entitäten im aktuellen PNetS anhand von *Constraints*, welche die Entität erfüllen soll und Relevanzkriterien, nach denen die Ergebnisse sortiert werden sollen, zu finden. So könnte ein Nutzer in einem Freizeitpark eine Achterbahn suchen, deren Wartezeit weniger als 20 Minuten beträgt. Die Ergebnisse könnten dabei nach geringster Distanz sortiert werden. Zum Einsparen von Kommunikationsaufwand im PNetS kann der Nutzer aus verschiedenen Routing-Verfahren wählen. So kann man das Versenden von Nachrichten auf eine gewisse Region oder auf einen gewissen Umkreis begrenzen.

Das System wurde in einer Nutzerstudie an zwölf Teilnehmern evaluiert. Dazu wurde eine auf dem C++-basierten Netzwerksimulator OMNet++[3] geschriebene Simulationsumgebung genutzt. Bei der Frage, ob die Nutzer interessiert seien, ein solches System in der Zukunft zu nutzen, und bei der Frage nach der Gebrauchstauglichkeit (engl.: Usability) der grafischen Benutzungsoberfläche (GUI) antworteten alle zwölf Teilnehmer auf der 4-Punkte-Skala entweder mit der höchsten oder mit der zweithöchsten Bewertung.

Ein Nachteil des Systems ist, dass es nur unter Laborbedingungen in einer Simulation getestet wurde. Die Teilnehmerzahl war mit zwölf sehr gering und somit nicht aussagekräftig. Ein weiterer Nachteil ist, dass nur in seinem lokalen PNetS gesucht werden kann. Möchte man nach Entitäten an einem entferntem Ort suchen, so ist dies nicht möglich.

Das Suchsystem von Truong

Das Suchsystem von Truong [17] erlaubt die Suche nach Sensoren anhand von Ganzzahlintervallen und einem Zeitfenster. Das System findet Sensoren, deren Werte in dem definierte Zeitfenster vom Suchzeitpunkt in die Vergangenheit „almost continuously“ [17, S. 121], also fast durchgehend, in das angegebene Ganzzahlintervall fallen. Es nutzt als Datenbasis den bereits vorgestellten Cloud-Dienst Xively[4], der eine API bereitstellt, mit der Sensoren ihre Daten

[3] `http://www.omnetpp.org/` [Zugriff am 21. März 2014]
[4] `https://www.xively.com/` [Zugriff am 21. März 2014]

übertragen und somit online abrufbar machen können. Die GUI, eine Java-Applikation, sendet dabei Suchanfragen an die *Global-Sensor-Search*-Komponente, die wiederum die Anfragen an *Local-Sensor-Search*-Einheiten weiterleitet. Diese besitzen *Crawler*, welche die Werte bzw. Wahrscheinlichkeitsmodelle für eine gewisse Menge an Sensoren periodisch abrufen bzw. generieren. Intern nutzt das Suchsystem auf Fuzzy-Sets [19] basierende temporale Wahrscheinlichkeitsmodelle. Wie bei Dyser wird Sensor Ranking genutzt, um die Zahl anzufragender Sensoren zu minimieren. Das System betrachtet die Wahrscheinlichkeit für einen gewissen Zeitraum, anstatt, wie für eine Echtzeitsuche wichtig, nur die momentane Wahrscheinlichkeit heranzuziehen. Durch das Wählen eines möglichst kleinen Zeitfensters kann die momentane Wahrscheinlichkeit aber angenähert werden.

Ein Nachteil ist die Suche mittels Ganzzahlintervallen. Dies setzt beim Nutzer das Wissen über die Semantik der vom Sensor genutzten Einheit voraus. Dies kann zu Problemen führen, wenn Nutzer entweder nicht wissen, was eine Einheit bedeutet oder falls nicht klar ist, in welcher Einheit ein Sensor die Messung vornimmt. Die Messung einer Temperatur kann etwa in Grad Celsius (°C), Grad Fahrenheit (°F), Grad Rømer (°Rø) oder Kelvin (K) erfolgen. Weiterhin erlaubt das Suchsystem keine Einschränkung der Sensorart oder der gesuchten Entität. Es werden lediglich Sensoren gesucht, deren Werte mit der Anfrage übereinstimmen. Bei einer Suche nach dem Intervall [18,21] kann die Ergebnisliste somit Sensoren enthalten, die $20\,°$C, $20\,°$F (dies entspricht jedoch $-6.7\,°$C) oder 20 % Luftfeuchtigkeit messen.

1.2.2. Vergleich der Systeme

Da die vorgestellten Systeme in ihrer Ausrichtung und Funktionalität sehr unterschiedlich und die Implementierungen zudem nicht frei verfügbar sind, ist eine quantitative Evaluation nicht möglich. Deswegen findet an dieser Stelle ein qualitativer Vergleich der Systeme statt.

Für jedes vorgestellte System zeigt Abbildung 1.1 ein Netzdiagramm, welches die Ausprägung der Systeme in acht qualitativen Kategorien darstellt, deren Fragestellung und Werte in Tabelle 1.1 aufgeführt sind. Es werden Kategorien gewählt, die verschiedene wichtige Aspekte für eine effiziente, skalierbare und erweiterbare Suche im IoT betrachten. Weiter außen liegende Punkte auf den Kategorienskalen bedeuten eine bessere Bewertung. Je größer also die aufgespannte, farbige Fläche, desto besser ist das System für eine Suche im IoT geeignet.

Die erste Kategorie betrachtet die Skalierbarkeit der Systeme auf Netze verschiedener Größe. Die Verteiltheit von Speicher- und Suchkomponenten wird in der zweiten Kategorie behandelt. Wie flexibel das System in Bezug auf Erweiterbarkeit durch Entwickler und Sensorbesitzer ist, bewertet die dritte Kategorie. Ein Teilaspekt dieser Flexibilität, die Heterogenität der nutzbaren sensorbestückten Geräte, wird als eigene, vierte Kategorie separat evaluiert. Die fünfte

Kategorie	Fragestellung	Werte
Skalierbarkeit	Wie gut skaliert das System für große Netze mit Millionen von Sensoren?	raumweit, gebäudeweit, weltweit
Verteilung	Sind die Komponenten verteilt oder gibt es eine zentrale Instanz?	zentral, teilweise zentral, dezentral
Flexibilität	Wie flexibel ist das System?	keine, mittlere, große
Heterogenität	Können verschiedene oder nur eine Art von Geräten integriert werden?	keine, mittlere, große
Effizienzsteigerungs-methoden	Werden Methoden zur Steigerung der Sucheffizienz eingesetzt?	keine, einige, viele
Offene Standards	Werden offene Standards verwendet?	keine, teilweise, komplett
Suchfunktion	Wie kann gesucht werden?	entitätsbasiert, schlüssel-wortbasiert, zustandsba-siert
Zielgruppe	Wer ist die Zielgruppe des Systems?	Experte, Fortgeschritte-ner, Jedermann

Tabelle 1.1.: Fragestellungen und mögliche Werte der Kategorien der Netz-diagramme.

Kategorie beschäftigt sich mit der Anzahl der in die Systeme integrierten Methoden zur Steigerung der Effizienz bei der Suche. Dies umfasst Aspekte wie die Verringerung der Antwortzeit und die Minimierung der Kommunikation und des Energieverbrauchs. Die Verwendung offener Standards, z. B. zur Abfrage, Beschreibung und Speicherung von Meta- und Sensordaten werden in Kategorie sechs bewertet. Der Umfang der Suchfunktionalität sowie die Zielgruppe, d. h. welches Vorwissen zur Nutzung notwendig ist, werden in der siebten und achten Kategorie eingestuft.

Einige der Kategorien beeinflussen sich in ihrer Bewertung gegenseitig. So beschränkt der Verzicht auf effizienzsteigernde Methoden sowie die Zentralisierung der Systemkomponenten auch die Skalierbarkeit der Systeme. Die Erweiterbarkeit als Teilaspekt der Flexibilität kann sowohl durch die Verwendung nicht offener Standards als auch durch die Homogenität von Geräten negativ beeinflusst werden.

Vergleicht man die Diagramme in Abbildung 1.1, so fällt vor allem auf, dass die Systeme in den Kategorien Verteilung, Flexibilität und Heterogenität niedrige Bewertungen haben. Die meisten Systeme nutzen proprietäre oder nicht-offene Protokolle, wodurch eine einfache Nutzung und Erweiterung schwierig ist. Die Suchfunktion, die ein zentrales Element der Systeme ist, erlaubt nur bei Gander, Dyser und dem Suchsystem von Truong eine zustandsbasierte Suche.

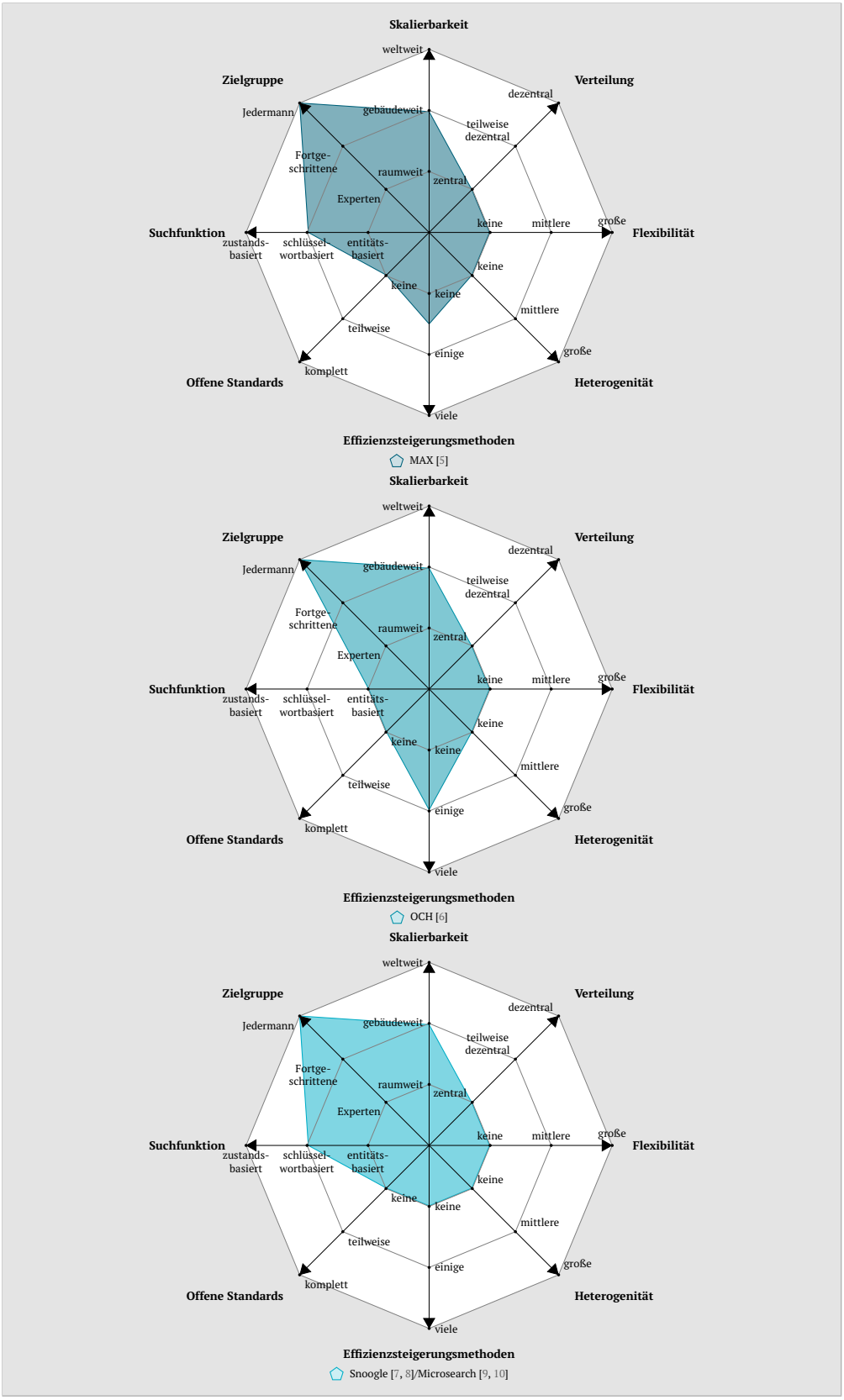


Abbildung 1.1.: Qualitativer Vergleich der vorgestellten Systeme.

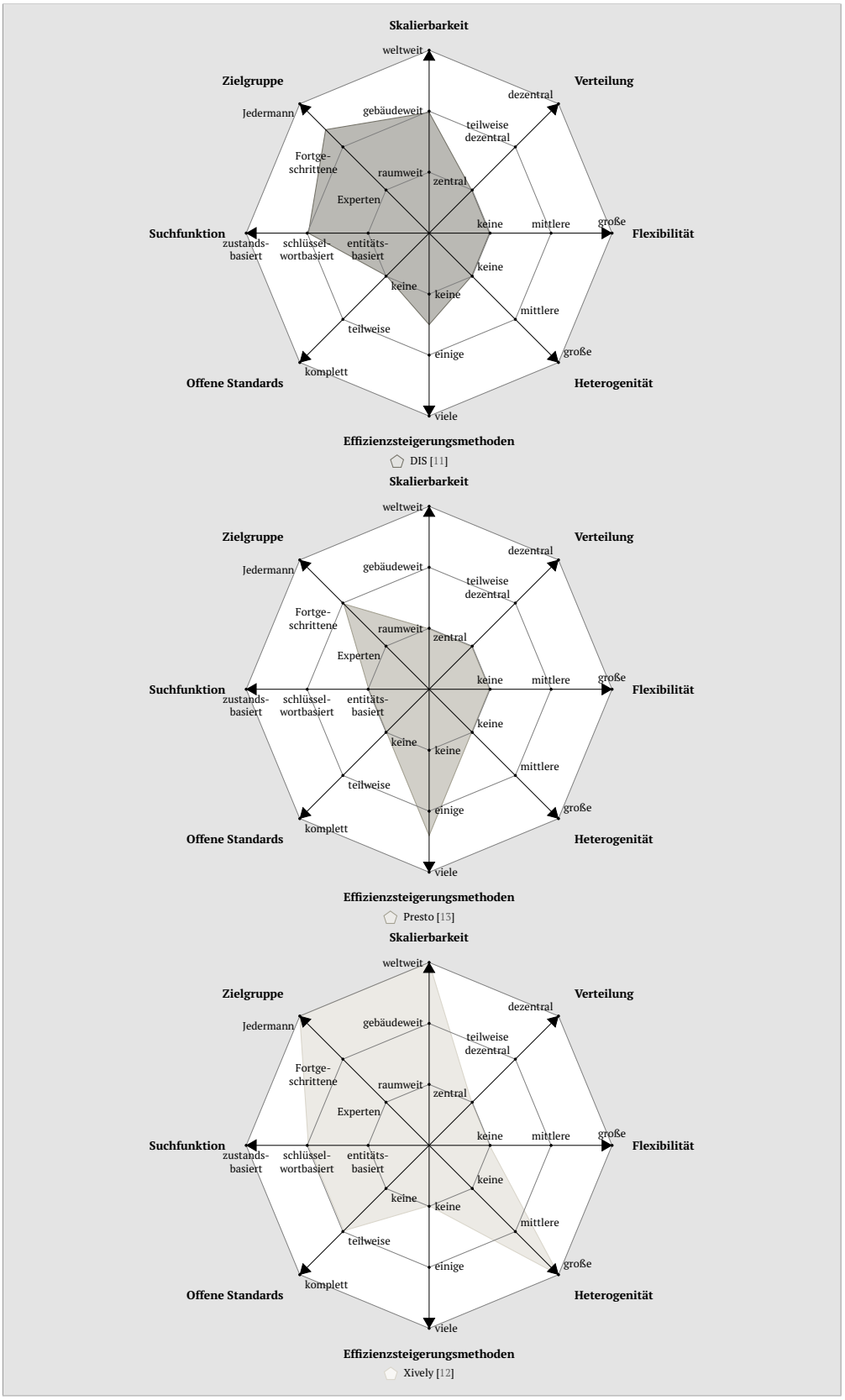


Abbildung 1.1 (fortgesetzt): Qualitativer Vergleich der vorgestellten Systeme.

Abbildung 1.1 (fortgesetzt): Qualitativer Vergleich der vorgestellten Systeme.

1.3. Herausforderungen

Im Vergleich bereits existierender Systeme im vorherigen Abschnitt wurden acht Evaluationskriterien genutzt. Dies sind Aspekte, die bei der Entwicklung von Technologien für eine zustandsbasierte Suche im IoT von Bedeutung sind. Zusammenfassen kann man diese Aspekte und somit die Herausforderungen unter den Begriffen *Skalierbarkeit* und *Akzeptanz*. Diese Herausforderungen müssen dabei jeweils aus der Sicht der Nutzer und aus der Sicht der Entwickler und Betreiber betrachtet werden.

Bei der Skalierbarkeit ist es wichtig, dass eine horizontale Skalierbarkeit unterstützt wird, d. h. dass zur Leistungssteigerung des Systems neue Rechner und im Kontext des IoT auch neue Sensorknoten hinzugefügt werden, anstatt wie bei der vertikalen Skalierung nur die Ressourcen (z. B. Festplatten, Prozessoren) der existierenden Rechner aufzurüsten. Die horizontale Skalierung soll dabei möglichst optimal die vier Arten der Skalierbarkeit, die Lastskalierbarkeit, die räumliche, die zeitlich-räumliche und die strukturelle Skalierbarkeit, gewährleisten, da die Arten sich teilweise gegenseitig negativ beeinflussen können [20]. Bei einer guten Lastskalierbarkeit verhält sich das System unter verschiedenen Lastintensitäten immer performant. Eine räumliche Skalierbarkeit bedeutet, dass durch das Hinzufügen von zu verwaltenden Elementen der notwendige Speicherbedarf lediglich geringfügig ansteigt, idealerweise sub-linear. Dass das Hinzukommen neuer Elemente die Verarbeitungszeit möglichst nicht beeinflusst, beschreibt die zeitlich-räumliche Skalierbarkeit. Der letzte Punkt, die strukturelle Skalierbarkeit, bedeutet, dass die Implementierung und der Aufbau des Systems das Ergänzen neuer Elemente nicht beeinträchtigt. Aus Nutzersicht bedeutet eine gute Skalierbarkeit somit, dass die Antwortzeit, unabhängig davon, wie viele andere Personen das System gerade verwenden, im selben Bereich liegt. Für Entwickler und Betreiber muss es auf einfachem Wege möglich sein, neue Komponenten, wie Sensorknoten, ggf. Gateway-Rechner zur Anbindung der Sensorknoten und Rechner zur Verwaltung der Daten, hinzuzufügen. Zudem darf sich dadurch die Performanz des Systems nicht verschlechtern.

Auch die Akzeptanz ist ein vielschichtiger Begriff, der aus Nutzer-, Entwickler- und Betreibersicht betrachtet werden muss. Auf Nutzerseite beinhaltet dies vor allem die Benutzungsfreundlichkeit. Diese wird in Normen üblicherweise Gebrauchstauglichkeit genannt und umfasst laut DIN EN ISO 9241 Teil 11 [21] die drei Aspekte Effektivität, Effizienz und Zufriedenheit. Effektivität bedeutet, dass der Nutzer bestmöglich unterstützt wird, um die Aufgabe, in diesem Fall die zustandsbasierte Suche, zu erledigen. Dabei ist es für Computersysteme vor allem schwierig die Sucheingabe zu verstehen, da die Nutzer ihre Suchintention mit sehr unterschiedlichen Begriffen umschreiben [22]. Die Effizienz als Teil der Gebrauchstauglichkeit beschreibt, dass die Aufgabenerfüllung durch das System in einer angemessenen Zeit geschehen soll. Dies ist verknüpft mit der Lastskalierbarkeit, da die Antwortzeiten immer möglichst klein gehalten werden sollen. Die Zufriedenheit ist dagegen ein sehr subjektiver Aspekt, der von Nutzer zu Nutzer sehr unterschiedlich ist. Nutzerzufriedenheit kann unter

anderem durch eine problemlose Bedienbarkeit (übersichtliche Bedienoberfläche, gute Anpassbarkeit, einfache Syntax), die Bereitstellung der Suchfunktion für verschiedene Systeme, wie Smartphones, Notebooks und Navigationsgeräte, oder Transparenz über die Erhebung und Verwendung privater Daten erreicht werden. Auf der Seite der Entwickler und Betreiber kann eine Akzeptanz neuer Technologien durch die Verwendung etablierter, offener Standards erreicht werden. Eine patentfreie, kostengünstige oder kostenlose Bereitstellung der Technologien sowie offene, gut dokumentierte APIs können einen weiteren Beitrag leisten.

Die Herausforderungen bei der Entwicklung von Technologien zur Realisierung einer zustandsbasierten Suche für das IoT sind somit sehr vielseitig und teilweise diametral zueinander. Es sollten immer alle Aspekte, sowohl die technischen als auch nutzerzentrierten, berücksichtigt und gegeneinander abgewogen werden. Die These dieser Arbeit ist es, dass *auf offenen Standards basierende Technologien entwickelt werden können, die eine zustandsbasierte Suche für das IoT realisierbar machen, die aus technischer Sicht skalierbar und ressourceneffizient arbeitet und zugleich eine hohe Benutzerakzeptanz durch eine gute Gebrauchstauglichkeit erzielt*. Im nächsten Abschnitt wird erläutert durch welche Technologien diese These gestützt werden kann.

1.4. Aufbau und Beiträge dieser Arbeit

Im Folgenden werden die Gliederung und die Beiträge der vorliegenden Arbeit beschrieben. Abbildung 1.2 zeigt dies zudem grafisch. Die Pfeile verdeutlichen welche Teile aufeinander aufbauen bzw. voneinander abhängig sind.

In Kapitel 1 wurden bereits die für den Gesamtkontext wichtigen verwandten Arbeiten vorgestellt. In den jeweiligen Kapiteln werden weitere verwandte Arbeiten präsentiert, die im Kontext des jeweiligen Kapitels relevant sind.

In Kapitel 2 werden die für das Verständnis der Arbeit wichtigen Grundlagen erklärt. Zuerst wird in Abschnitt 2.1 auf die technologische Entwicklung von Sensorknoten hin zum IoT eingegangen. Als zweites werden in Abschnitt 2.2 RESTful Webservices und das Constrained Application Protocol (CoAP) als spezielle Variante für ressourcenbeschränkte Geräte vorgestellt. Abschließend beschreibt Abschnitt 2.3 die Grundlagen des Semantic Web, indem die semantische Modellierungssprache Resource Description Framework (RDF) sowie die für RDF entwickelte Abfragesprache SPARQL Protocol and RDF Query Language und deren Erweiterung GeoSPARQL vorgestellt werden.

Eine Nutzerumfrage [23], deren Ergebnisse in die Entwicklung verschiedener Teile dieser Arbeit eingeflossen ist, wird in Kapitel 3 vorgestellt, ausgewertet und diskutiert. Dafür werden die Teilnehmer gefragt, wie sie sich eine Suche für das IoT vorstellen und wie sie diese nutzen würden.

Kapitel 4 stellt zuerst eine gemeinsame Regelbasis zur Ermittlung von Zuständen aus Sensorwerten vor. Anschließend werden verschiedene Systeme präsen-

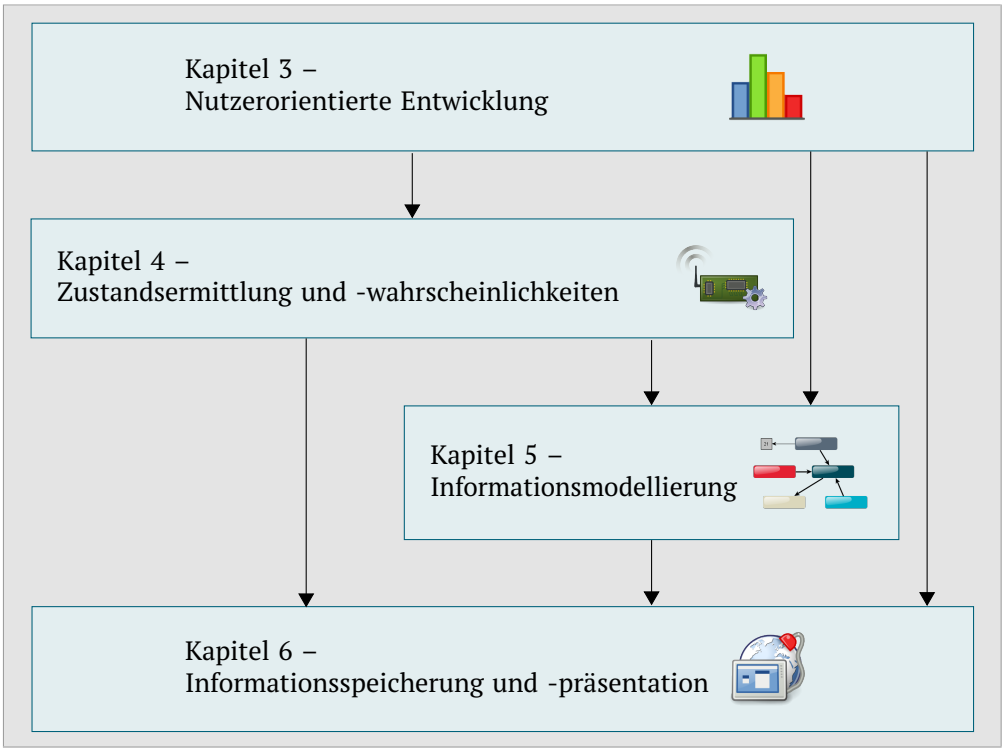


Abbildung 1.2.: Grafische Darstellung der Beiträge dieser Arbeit und deren Zusammenhänge.

tiert, die diese Regelbasis nutzen, um Zustände zu ermitteln. In Abschnitt 4.2.1 wird das Lousy Camera Framework [24] vorgestellt. Dieses Framework kann aus Bildern, die von kamerabestückten Sensorknoten aufgenommenen werden, mit Hilfe einer flexiblen Regelsprache Zustände ableiten und auf verschiedene Weise im Internet veröffentlichen. Abschnitt 4.2.2 behandelt die High-Level-State-Option [25, 26] für CoAP, die das dynamische Erstellen von Zustandsressourcen auf Sensorknoten erlaubt. Anstatt roher Sensorwerte können damit höherwertige Zustände direkt vom Knoten abgefragt werden. Für Sensorknoten, die CoAP nicht unterstützen, wird in Abschnitt 4.2.3 eine alternative Lösung vorgestellt, die auf einer Skriptsprache [27] basiert. Diese erlaubt u. a. die Ermittlung von höherwertigen Zuständen. Abschließend wird in Abschnitt 4.2.4 eine Erweiterung für Browser vorgestellt, welche die Erstellung *virtueller Sensoren* erlaubt. Für solch einen virtuellen Sensor markiert der Nutzer ein Textfragment einer Webseite (z. B. eine Temperaturangabe auf einer Wetterseite), die dann periodisch extrahiert wird. Aus diesem Textfragment wird anschließend mit Hilfe einer Regelsprache ein höherwertiger Zustand bestimmt und in einer Datenbank gespeichert. Nach der Beschreibung der Zustandsermittlung wird im Weiteren die Nutzung von Zustandswahrscheinlichkeiten zur effizienten Abfrage von Sensorzuständen behandelt. Dazu wird auf unterschiedliche Möglichkeiten zur Repräsentation und Berechnung von Wahrscheinlichkeitsmodellen eingegangen.

In Kapitel 5 wird beschrieben, wie die Zustände und Modelle, die in Kapitel 4 ermittelt bzw. berechnet wurden, semantisch modelliert werden können. Weiterhin wird die Modellierung weiterer statischer und dynamischer Sensordaten und Informationen für das Suchsystem dargelegt. Dazu gehören u. a. die Historie von Zuständen (Abschnitt 5.2.3) und Faktoren zum Ranken von Ergebnissen (Abschnitt 5.2.10). Abschließend wird in Abschnitt 5.3 gezeigt, wie Wahrscheinlichkeitsmodelle aus den vorliegenden Daten im Backend erzeugt werden können.

Die Informationsspeicherung und -präsentation ist Thema des Kapitels 6. Abschnitt 6.2 behandelt, DecentSPARQL [28], eine Abstraktions- und Adaptionsschicht für die semantische Datenbank LUPOSDATE. Diese erlaubt durch Verwendung der Peer-to-Peer-Technologie die dezentrale, verteilte und skalierbare Speicherung von semantischen Daten, wie sie z. B. durch die in Kapitel 5 beschriebene Modellierung entstehen. Durch die Erweiterung von LUPOSDATE mit GeoSPARQL ist auch DecentSPARQL in der Lage geografische Daten, wie z. B. die Position der Geräte des IoT, zu verarbeiten. Danach werden in Abschnitt 6.3 die Schnittstelle für Anwendungsentwickler sowie eine prototypische GUI vorgestellt. Zum Abschluss wird eine Nutzerstudie zur Gebrauchstauglichkeit der GUI durchgeführt und evaluiert. Diese zeigt, wie gut die in Kapitel 3 hergeleiteten Anforderungen umgesetzt wurden.

Kapitel 7 gibt eine Zusammenfassung der Arbeit, eine Bewertung der entwickelten Technologien im Vergleich zu den in der Einleitung vorgestellten Systemen sowie einen Ausblick über offene Forschungsfragen.

Die Beiträge dieser Arbeit sind daher

- die Analyse von Nutzerwünschen und -verhalten für eine zustandsbasierte Suche im IoT,
- Mechanismen zur flexiblen und dynamischen Ermittlung von Zuständen aus Sensoren,
- eine erweiterbare, maschinenlesbare Modellierung von statischen und dynamischen Sensorinformationen,
- eine skalierbare Architektur zur Speicherung von Informationen im IoT und
- eine effiziente und effektive Informationsverarbeitung und gebrauchstaugliche Informationspräsentation der Sensor- und Entitätsinformationen,

die eine zustandsbasierte Suche im IoT ermöglichen. Dabei wird durchgehend auf

- Skalierbarkeit auf große Netze und Datenmengen,
- eine dezentrale Architektur,
- Erweiterbarkeit,
- Heterogenität der verwendbaren Geräte,

- den Einsatz offener Standards und
- eine einfache Nutzbarkeit

bei der Konzeption und Realisierung geachtet.

Danksagung an Mitwirkende

Die Technologien, die in dieser Dissertation vorgestellt werden, wären ohne die Kooperation und die wertvollen Diskussionen mit Kollegen am Institut für Technische Informatik, dem Institut für Telematik und dem Institut für Informationssysteme sowie Studenten nicht möglich gewesen. DecentSPARQL ist eine gemeinsame Arbeit mit Prof. Dr.-Ing. Dennis Pfisterer, Daniel Bimschas, Oliver Kleine und PD Dr. rer. nat. Sven Groppe. Die Implementierungen wurden hauptsächlich in einer Fallstudie durchgeführt. Dankeschön gebührt den Studenten Philipp Abraham, Benjamin Meyer und Christopher Gudat für die Hilfe bei Implementierungen und Evaluationen einiger Bestandteile dieser Arbeit.

2 Grundlagen

> People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones.
>
> — *Donald Knuth*

In diesem Kapitel werden die für diese Arbeit wichtigen Grundlagen erklärt und mit Beispielen erläutert. In Abschnitt 2.1 wird auf die technologische Entwicklung und die dadurch entstehenden Möglichkeiten in Form von neuen Technologien wie Sensorknoten eingegangen. RESTful Webservices und die Umsetzung dieser für ressourcenbeschränkte Systeme, wie z. B Sensorknoten, werden in Abschnitt 2.2 behandelt. Abschnitt 2.3 beschäftigt sich abschließend mit dem Konzept des Semantic Web und den zugrundeliegenden Technologien und Standards.

2.1. Technologische Entwicklung

Mitte der sechziger Jahre untersuchte Gordon Moore die Komplexität von Schaltkreisen unter ökonomischen Gesichtspunkten. Dabei betrachtete er die Anzahl an Schaltkreiskomponenten, die wirtschaftlich herzustellen sind. Durch Untersuchung unterschiedlicher Chips kam er dabei zunächst zu der Beobachtung, dass sich die Komplexität alle 12 Monate verdoppelt [29]. Mitte der siebziger

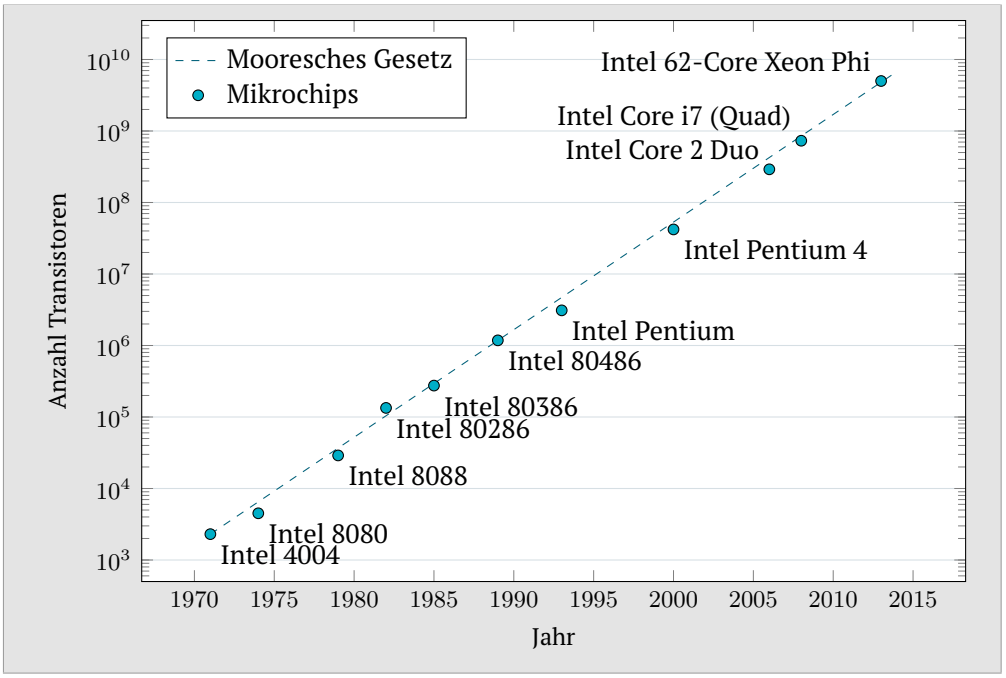


Abbildung 2.1.: Mooresches Gesetz und Anzahl Transistoren verschiedener Mikrochips, aufgetragen nach ihrer Markteinführung [Daten von 30].

Jahre korrigierte er dies auf eine Verdopplung alle 24 Monate. Die Vermutung ist heute als Mooresches Gesetz bekannt.

Abbildung 2.1 zeigt das Jahr der Markteinführung und die Anzahl der Transistoren verschiedener Intel-Prozessoren sowie die Vorhersage der Integrationsdichte durch das Mooresche Gesetz. Dabei ist die logarithmische y-Achse zu beachten. Es ist gut zu erkennen, wie die Werte relativ genau zu der Vorhersage des Mooreschen Gesetzes passen. Fälschlicherweise wird eine Verdopplung der Komplexität dabei oft mit einer Verdopplung der Rechenleistung gleichgesetzt. Da ein Teil der Komponenten allerdings für Caches genutzt wird, ist dies nicht der Fall. Dennoch steigt die Rechenleistung von Computerchips bei gleichbleibender Größe kontinuierlich an. Dies heißt im Umkehrschluss, dass die identische Rechenleistung auf immer weniger Chipfläche untergebracht werden kann. Zudem sinken auch die Produktionskosten für die gleiche Rechenleistung stetig.

Der Chef der Digital-Enterprise-Sparte von Intel, Pat Gelsinger, vermutete auf dem Intel Entwicklerforum 2008, dass die Vorhersage noch bis mindestens 2029 erfüllt werden kann [31]. Bewahrheitet sich die Vermutung von Moore für die nächsten 15 Jahre, wird Computertechnik immer unauffälliger, leistungsfähiger und günstiger werden. Diese Entwicklung ermöglicht es, immer mehr eingebettete Systeme zu verbauen. Eingebettete Systeme sind Systeme, die, weitestgehend unsichtbar für den Benutzer, in Gegenstände integriert werden, um diese zu steuern oder regeln. Die Einsatzmöglichkeiten eingebetteter Systeme werden somit zunehmend steigen.

Neben der Entwicklung, die durch das Mooresche Gesetz beschrieben wird, ist auch bei weiteren Komponenten, die in Informationstechnik verbaut werden, zu beobachten, dass die technologischen Parameter in ähnlichen Zeiträumen gesteigert werden konnten. Es ist davon auszugehen, dass dieser Trend auch bei diesen Komponenten noch eine Zeit lang anhält. Dazu gehören u. a. die Speicherdichte, Speichertransferraten sowie die Bandbreite kabelgebundener und kabelloser Kommunikation. Auf der anderen Seite ist die Steigerungsrate bei der Energiedichte von Batterien nur sehr gering. Deshalb ist Energie der oftmals limitierende Faktor bei batteriebetriebenen Geräten. Koomey et al. [32] haben durch Untersuchungen folgende Schlussfolgerung gezogen:

> *„The power needed to perform a task requiring a fixed number of computations will fall by half every 1.5 years, enabling mobile devices performing such tasks to become smaller and less power consuming and making many more mobile computing applications feasible.“*

Dies wird auch als Koomey's Gesetz bezeichnet. Diese immer effizientere Verwendung von Energie ermöglicht somit, trotz langsam steigender Energiedichte von Batterien immer mehr Einsatzmöglichkeiten und längere Laufzeiten von Geräten.

Weitere Triebfedern, die eine Durchdringung unserer Umwelt und der Gegenstände in ihr mit Informationstechnologie voranbringen, sind neue Materialien, verbesserte Sensorik, die bereits erwähnte Verbesserung von Kommunikationsbandbreite sowie neue Konzepte zur Vernetzung von Geräten.

Berührungsempfindliche Displays, Gesten- und Spracherkennung erweitern den Raum möglicher Mensch-Computer-Interaktion. Sensorik wird immer kleiner und genauer und lässt sich somit in Geräte wie Smartphones verbauen. Neuartige Displaytechnologien erlauben flache, biegsame und faltbare Bildschirme. Im Bereich der Kommunikation ermöglichen Entwicklungen wie W-LAN, Bluetooth, Nahfeldkommunikation, Ad-Hoc Netzwerke und Service-Discovery-Mechanismen wie UPnP spontane, breitbandige Vernetzung zwischen Geräten.

All diese Entwicklungen führen zu neuen Geräteklassen, die zunehmend unsichtbar in unsere Umwelt eingebettet werden und mit ihr verschmelzen. Dieser Trend des in-den-Hintergrund-treten von Technologie wurde schon in den neunziger Jahren von Marc Weiser als *Ubiquitous Computing* beschrieben [33]. Eine dieser neu entstandenen Geräteklassen sind Sensorknoten.

2.1.1. Sensornetze

Ein drahtloser Sensorknoten ist ein kleines eingebettetes System, das mindestens aus einem Prozessor, Speicher, einem Funkmodul, einem oder mehreren Sensoren und einer Energieversorgung besteht. Weitere Komponenten können u. a. verschiedene Schnittstellen, Analog-Digital-Wandler, Taster und Aktuatorik, wie LEDs und Lautsprecher, sein. Sensorknoten können mit Hilfe ihrer

	Core Module	Core Module 2	Core Module 3
Jahr	2007	2009	2011
Prozessortakt	16 MHz	4–32 MHz	4–32 MHz
RAM	96 kB	128 kB	128 kB
Flashspeicher	128 kB	512 kB	512 kB
Genauigkeit der Uhr	20 ppm	20 ppm	6 ppm
Verbrauch im Betrieb	~9 mA	~6 mA	~6 mA
Verbrauch beim Schlafen	~10 µA	~3 µA	~3 µA
Datenrate	250 kbit/s	250–667 kbit/s	250–667 kbit/s
Dimensionen	45x30 mm	45x30 mm	45x30 mm
Arbeitsbereich	-20–70 °C	-20–70 °C	-20–70 °C

Tabelle 2.1.: Verschiedene Versionen der iSense-Plattform [37] der Firma Coalesenses [Daten aus 38–40].

Sensoren Messwerte erfassen, diese Daten verarbeiten, speichern und mittels des Funkmoduls verschicken. Durch die Verwendung von Batterien ist, wie oben beschrieben, die verfügbare Energie der limitierende Faktor beim Betrieb von Sensorknoten. Der Energieverbrauch des Funkmoduls kann bis zu 90 % des Gesamtverbrauchs ausmachen [34]. Deshalb wird häufig versucht so selten wie möglich das Funkmodul zu nutzen und im Falle der Nutzung die Menge an zu kommunizierenden Daten auf ein Minimum zu beschränken.

Die Entwicklung dieser Geräteklasse begann Anfang der Jahrtausendwende an der University of California, Berkeley mit dem mica-Sensorknoten [35]. Dieser basiert auf einem 8-Bit Mikrocontroller mit einem 4 MHz Prozessor und besitzt u. a. 128 kB Programmspeicher, 4 kB statischem RAM sowie ein Funkmodul, das eine Übertragungsgeschwindigkeit von bis zu 115 kbit/s unterstützt. Ein 51-Pin Erweiterungsstecker erlaubt den Anschluss verschiedener Sensorik und Aktuatorik. Die Energieversorgung erfolgt durch zwei AA Batterien.

Üblicherweise bietet ein spezialisiertes Betriebssystem Zugriff auf die Komponenten eines Sensorknotens und kapselt Funktionalitäten wie Funkmedienzugriff, Routing und Zeitsynchronisation. Die mica-Plattform nutzt das Betriebssystem TinyOS, das ebenfalls an der UC Berkeley entwickelt wurde [36].

Die iSense-Sensorknoten der Firma Coalesenses[1], die mit dem hauseigenen iSense-Betriebssystem betrieben werden, bilden eine weitere Sensorknotenplattform. Tabelle 2.1 zeigt die drei Versionen dieses Sensorknotens. Es ist zu erkennen, wie die Leistung, d. h. der Speicher, die Geschwindigkeit und Genauigkeit der Uhr, über die Jahre gestiegen ist. Dennoch ist die Größe gleich geblieben und der Energieverbrauch wurde gesenkt. Dieses Verhältnis von Grö-

[1] `http://www.coalesenses.com/` [Zugriff am 21. März 2014]

ße zu Leistung ist auf die bereits beschriebenen technologischen Fortschritte zurückzuführen. Dennoch ist die Geschwindigkeit und der zur Verfügung stehende Speicher von Sensorknoten eher als gering einzustufen. Dies ist eine Herausforderung bei der Programmierung von Sensorknoten-Anwendungen. Durch die Energieversorgung mit Batterien ergibt sich zusätzlich die Anforderung, möglichst sparsam mit den Ressourcen des Knotens umzugehen.

Für die iSense-Plattform ist eine Vielzahl von Sensorik verfügbar. Dazu zählen Sensoren zur Messung von Temperatur, relativer Luftfeuchte, barometrischem Druck, Lichtintensität, Position mittels Global Positioning System (GPS), Bewegung durch Passive-Infrared-Sensoren (PIR-Sensoren), Beschleunigung mit Hilfe von 3-Achsen-Beschleunigungssensoren und magnetischen Massen mit AMR-Sensoren. Außerdem gibt es ein Kameramodul zur Aufnahme von Bildern. Die in dieser Arbeit implementierten Systeme sind für die iSense-Sensorknoten realisiert.

Durch die Vernetzung von mehreren Sensorknoten kann ihr Potential ausgeschöpft werden. Die entstehenden drahtlosen Sensornetze (engl. Wireless Sensor Network (WSN)) können große Gebiete überwachen. Es gibt eine Vielzahl von Einsatzmöglichkeiten für WSNs. Einige dieser Anwendungsbereiche werden im Folgenden beschrieben und exemplarisch reale Sensornetzinstallationen für das jeweilige Gebiet genannt.

Eines der häufigsten Einsatzfelder, von dem in der Literatur berichtet wird, ist die Überwachung natürlicher Phänomene, das Environmental Monitoring. Eine Installation von Sensorknoten auf *Great Duck Island* in Maine aus dem Jahr 2002 wird in [41, 42] beschrieben. Ein Teil der Knoten ist in den Nestern von Wellenläufern platziert, um die Anwesenheit der Vögel sowie Temperatur und Feuchtigkeit zu messen. Weitere Knoten protokollieren außerhalb der Nester die Wetterbedingungen. Dadurch wird das Brutverhalten der Tiere untersucht. Mittels eines Multi-Hop-Netzwerkes werden die Informationen an einen zentralen Computer, die Basisstation, zur Speicherung und Auswertung übertragen. Im *PODS*-Projekt [43] liegt der Fokus dagegen auf seltenen und gefährdeten Pflanzen im *Hawai'i-Volcanoes-Nationalpark*. Ein Sensornetz misst Umgebungsparameter wie Temperatur, Luftfeuchte, Niederschlag, Wind und Sonneneinstrahlung. Spezifischer ist das *Redwood-Ökophysiologie*-Projekt [44, 45], bei dem das Mikroklima von Redwood-Riesenmammutbäumen in Sonoma im US-Bundesstaat Kalifornien untersucht wird. Dazu werden die Lufttemperatur, relative Luftfeuchte und die photosynthetisch aktive Sonneneinstrahlung an den Bäumen gemessen. Im *GlacsWeb*-Projekt [46] werden in einem PVC-Gehäuse untergebrachte Sensorknoten in vorgebohrte Löcher in einem Gletscher untergebracht. Dort messen sie Temperatur, Druck und die Orientierung im Eis. Insbesondere die Kommunikation durch Wasser und Eis ist eine Herausforderung bei dieser Installation. Die Schneehydrologie, also die Menge, Verteilung und Beschaffenheit von Schnee, wird im *Snowcloud*-Projekt [47] untersucht. Eine weitere Kategorie der Umweltüberwachung ist das Monitoring von Waldbrandgefahr. Doolin und Sitar [48] beschreiben eine Sensornetzinstallation im *Pinole Point Regional Park* für diesen Einsatzzweck.

Mehrere Sensornetzanwendungen beschäftigen sich mit der Überwachung von Tierverhalten, insbesondere den Bewegungsmustern der beobachteten Tiere. In *ZebraNet* [49, 50] und beim Projekt von Sommer et al. [51] werden Zebras bzw. Fledermäuse mit GPS ausgestattet, um ihre Bewegungen zu verfolgen. Sobald die Tiere wieder in die Nähe von Basisstationen kommen, übertragen die Sensorknoten die zwischengespeicherten Koordinaten der Bewegung.

Der Einsatz für gewerbliche und industrielle Zwecke ist gleichermaßen eine Triebfeder für die Weiterentwicklung der Technologie. In der Landwirtschaft nutzt das *LOFAR-agro*-Projekt [52] die Technik zur Untersuchung von Pilzbefall in Kartoffelfeldern. Beckwith et al. [53] sowie Burrell et al. [54] beschreiben die Überwachung des Mikroklimas von Weinbergen mit Sensornetzen. In [55] wird ein System für Kühe beschrieben, das neben Sensoren auch Aktuatoren enthält, welche den Kühen kleine Stromstöße versetzen, sobald sie sich aus einem definierten Gebiet bewegen. Dadurch entstehen virtuelle Zäune. Dieses System kann für große Farmen Geld einsparen, falls, statt realer Zäune, nur Sensorknoten an den Kühen angebracht werden müssen. Um die Vibrationen von Industrieanlagen zu überwachen, wird in [56] beschrieben, wie rund 150 Bewegungssensoren, u. a. am Motor eines Öltankers, angebracht werden.

Der Einsatz von Sensornetzen im kulturellen Kontext wird ebenfalls erfolgreich umgesetzt. Li et al. [57] berichten von einer Installation, welche schädliche klimatische Bedingungen für antike Relikte in der *Verbotenen Stadt* in China feststellt.

Neben industriellen Anlagen kann ebenso die Integrität von Bauwerken, wie Gebäuden und Brücken, überwacht werden. Kim et al. [58] beschreiben ein System zur Messung von Vibrationen an der *Golden Gate Bridge* in San Francisco. Auch das *iBAST*-Projekt [59], in dem Teile dieser Arbeit entstanden sind, beschäftigt sich mit der Thematik des Brückenmonitoring mit Hilfe drahtloser Sensornetze, insbesondere der Überwachung und der Veränderung von bestehenden Schäden. Ein anderes System, beschrieben in [60], überwacht den Zustand des unter Weltkulturerbestatus stehenden Turms *Torre Aquila* in Trento während naheliegender Bauarbeiten.

Der Gesundheitssektor hat ebenfalls den Nutzen von Sensorknoten erkannt. Die immer kleiner werdenden Geräte und ihre drahtlose Kommunikation eignen sich ideal zur Überwachung von Vitalparametern wie Puls und Blutsauerstoffgehalt. Dies wird u. a. im Projekt von Chipara et al. [61] angewendet. In [62, 63] werden Sensoren bei Marathonveranstaltungen genutzt, um Geschwindigkeit, Position, Temperatur und Puls während des Laufens zu messen.

Wie bei der Tierüberwachung mit virtuellen Zäunen gibt es auch im menschlichen Umfeld Sensornetze, die aktuatorische Elemente besitzen, um auf die Erkenntnisse aus sensorischen Messungen direkt mit Aktionen antworten zu können. Deshpande et al. [64] beschreiben ein System, das die Heizung, Lüftung und Klimaanlage eines Gebäudes auf Basis der aktuellen klimatischen Bedingungen und der Vorgaben des Nutzers steuert.

Weiterführende Informationen vieler hier vorgestellter Arbeiten, eine ausführliche Liste mit weiteren Beispielen sowie eine Taxonomie finden sich bei Oppermann et al. [65].

Weitere Entwicklung

Die ersten Installationen von WSNs waren isolierte Netze, deren Daten zentral auf einem Rechner gesammelt und anschließend ausgewertet wurden. Obwohl ggf. eine Verbindung des Rechners mit dem Internet existierte, war eine direkte Kommunikation mit dem Sensornetz oder einzelnen Sensorknoten nicht vorgesehen.

Um auch vom Internet aus auf die Sensorknoten zuzugreifen, wurde die Entwicklung von Proxy-Anwendungen vorangetrieben. Diese Proxys vermitteln zwischen den etablierten, standardisierten Internetprotokollen und den proprietären Sensornetzprotokollen, indem sie diese ineinander übersetzen [66, 67].

Solche Speziallösungen sind aber aufgrund des Bestrebens abgelöst worden, Sensorknoten direkt per Internet Protocol (IP) ansprechen zu können und so nahtlos in das Internet zu integrieren. Um den Anforderungen von Sensornetzen gerecht zu werden, wurde für das Internet Protocol Version 6 (IPv6) die Adaptionsschicht IPv6 over Low power Wireless Personal Area Network (6LoWPAN) standardisiert, um IPv6 auf ressourcenbeschränkten Systemen nutzen zu können.

Die Integration von Sensornetzen ins Internet erlaubt es, den Zustand der realen Welt, der durch Sensoren gemessen wird, via Internet überall und in Echtzeit abzufragen. Dies ist ein Schritt zur Verwirklichung der Vision des IoT.

2.1.2. Internet der Dinge

Der Begriff *Internet of Things* wird Kevin Ashton zugeschrieben, der diesen in einer Präsentation im Jahr 1999 zum ersten Mal verwendet hat [68]. Das Konzept des IoT wurde allerdings bekannt durch die Aktivitäten des Auto-ID Labs[2], das aus dem Auto-ID Center des Massachusetts Institutes of Technology hervorging. Dieses beschäftigte sich um die Jahrtausendwende mit der Nutzung der RFID-Technologie zur Warenidentifikation. Der Vorteil gegenüber dem Strichcode ist, dass die auf dem RFID-Chip gespeicherten Informationen ohne Sichtkontakt drahtlos über kurze Distanzen ausgelesen werden können. Passive Chips benötigen dafür nicht einmal eine Energieversorgung, da sie durch elektromagnetische Induktion vom Lesegerät mit Energie versorgt werden. Die RFID-Technologie, die eine eindeutige Identifikation von Gegenständen ermöglicht, wurde als Grundlage für das IoT angesehen.

Marktforscher, Firmen und Wissenschaftler interessierten sich zunehmend für die Idee, Dinge der realen Welt eindeutig identifizierbar und über das Internet

[2] `http://www.autoidlabs.org/` [Zugriff am 21. März 2014]

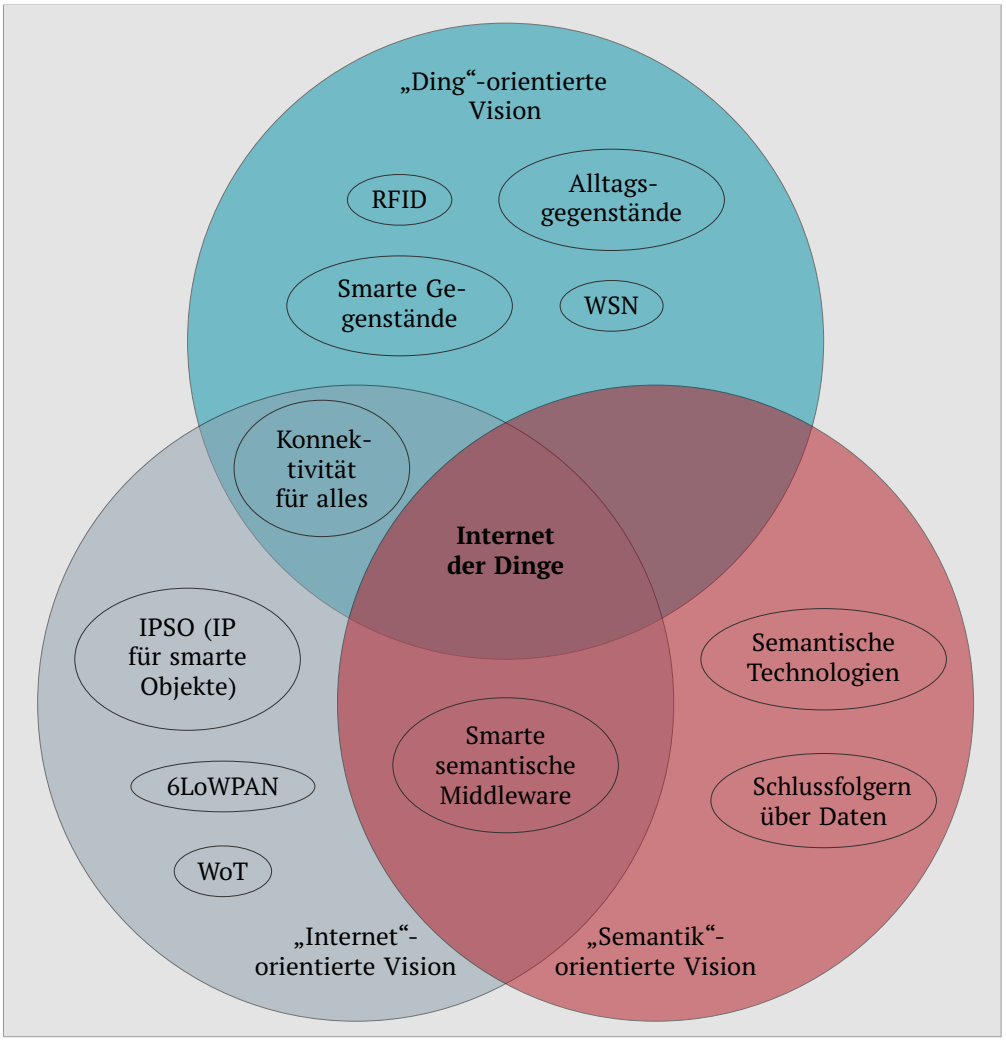


Abbildung 2.2.: Unterschiedliche Dimensionen bei der Definition des Begriffes *Internet der Dinge* [Nach 69, Abb. 1].

verfügbar zu machen. Sie nutzen die Entwicklungen ihrer Fachgebiete, wie z. B. WSNs, um die Vision weiterzuentwickeln. Eine Vielzahl von Definitionen entstand, so dass es heute keine eindeutige Begriffserklärung gibt. Dies liegt vor allem daran, dass die zahlreichen Definitionen aus verschiedenen Blickrichtungen auf die Thematik entstanden sind. Atzori et al. [69] erläutern in ihrem Artikel drei Richtungen, aus denen der Begriff IoT entstanden ist. Diese Facetten sind in Abbildung 2.2 dargestellt.

Der Begriff *Internet der Dinge* gibt bereits zwei Sichtweisen vor. Die erste ist die „Ding“-orientierte Sicht, aus der das Auto-ID Lab das Thema behandelt. Bei dieser Betrachtungsweise geht es um die Ausrüstung realer Dinge mit Technik, um diese eindeutig identifizierbar zu machen. Zudem sollen die Objekte Metadaten wie Hersteller und Produktionsdatum zur Verfügung stellen können. Forscher aus dem Sensornetzbereich sehen zusätzlich Sensorknoten als mögliche Basis

für das IoT. Diese bieten den Nutzen, dass sie zusätzlich zu statischen Metadaten auch dynamische Daten wie die Position oder den Zustand der Umgebung oder der Dinge messen und bereitstellen können. Gegenstände, die mit einer solchen Technologie ausgerüstet sind, werden häufig auch als *smart* bezeichnet, da ein gewisses Bewusstsein für den eigenen Zustand besteht.

Die zweite Blickrichtung ist die „Internet"-orientierte, die sich weniger mit der Identifikation von Objekten, sondern mehr mit der Integration in das Internet beschäftigt. Dabei geht es hauptsächlich um die Fragen, wie die Objekte möglichst dauerhaft erreichbar sind und wie gut sich bestehende Protokolle verwenden bzw. anpassen lassen, um mit den neuen Herausforderungen zu Recht zu kommen. Diese Herausforderungen sind vor allem die oftmals knappen Ressourcen, wie es bei WSNs der Fall ist, als auch die Mobilität der Dinge. Die IPSO Allianz, ein Zusammenschluss verschiedener Firmen aus der IT-Branche, möchte die Nutzung der IP-Protokollfamilie vorantreiben und fördern dies mit Interoperabilitätstests und Wettbewerben rund um diese Protokolle. Dies sind u. a. das oben erwähnte 6LoWPAN als Anpassung des IPv6-Standards und CoAP als Pendant zum Hypertext Transfer Protocol (HTTP). CoAP ist auch ein Teil der Vision des Web of Things (WoT), bei dem bestehende Webstandards wiederverwendet werden, um Geräte nicht nur im Internet, sondern auch im Web verfügbar zu machen.

Prognosen von Firmen wie Cisco [1] und IBM [2] sowie von Marktforschungsunternehmen wie Gartner [3] gehen von Milliarden oder sogar Trilliarden von verbundenen Geräten innerhalb des nächsten Jahrzehnts aus. Neben den Metadaten wird ein Großteil dieser Geräte dynamische Daten generieren. Diese Menge an Daten zu speichern, zu verwalten, zu verknüpfen und zu durchsuchen wird eine große Herausforderung [70]. Eine mögliche Lösung ist die Nutzung von Technologien des Semantic Web (siehe Abschnitt 2.3). Deshalb ist die dritte und letzte Sichtweise zur Definition des IoT die „Semantik"-orientierte. Die Technologien erlauben es, den Daten eine Semantik zu geben, sie zu verifizieren, verschiedene Datenquellen durch ein gemeinsames Format zu verknüpfen und neues Wissen aus bekannten Fakten herzuleiten.

Zusammenfassend kann gesagt werden, dass es die Vision des IoT ist, Gegenständen der realen Welt eine virtuelle Repräsentation im Internet zu geben. Über diese sollen sowohl Metadaten als auch der Zustand der Entitäten abgefragt und ggf. auch beeinflusst werden können. Dies führt zu einer Verschmelzung von digitaler und realer Welt. Realisiert werden soll das IoT durch die Anbringung bzw. Integration verschiedener, sehr heterogener, technischer Systeme in Entitäten der realen Welt. Dazu zählen u. a. Sensorknoten, RFID-Chips, Smartphones, Webcams, GPS-gestützte Geräte wie Navigationssysteme und Bluetooth-basierte Geräte wie Schrittzähler.

2.2. RESTful Webservices

RESTful Webservices sind Dienste, welche die Prinzipien von Representational State Transfer (REST) nutzen. Dies ist ein Architekturstil, der in der Dissertation von Fielding [71] beschrieben wird. Ein Hauptziel von REST ist es eine gute Skalierbarkeit der Dienste zu erreichen. REST ist allerdings kein Standard, der vom World Wide Web Consortium (W3C) oder einer anderen Standardisierungsorganisation definiert wird, sondern ein Satz von Regeln und Ideen.

In REST-Architekturen kommunizieren Client und Server nach dem Request-Response-Muster, d. h. die Anfragen von Clients werden von Servern beantwortet. Dabei verschickt der Server Repräsentationen von Ressourcen. Eine Ressource ist dabei ein Konzept, z. B. eine Datenbank, eine Datei oder Ähnliches, das adressierbar und somit vom Client abrufbar ist. Die Repräsentation spiegelt den Zustand der Ressource wider.

Im Folgenden werden die wichtigsten Grundprinzipien von REST erläutert:

Identifikation von Ressourcen Jede Ressource ist eindeutig identifizierbar, z. B. durch einen Uniform Resource Identifier (URI) in webbasierten RESTful Systemen.

Repräsentation Die eigentliche Repräsentation einer Ressource ist unabhängig von ihrer Adresse. Der Inhalt einer Ressource kann dabei auch in mehreren Formaten angeboten werden. Der Client kann in seinem Request festlegen, welches Datenformat er bevorzugt.

Zustandslosigkeit Die Elemente eines RESTful Webservices sind zustandslos, d. h. die Nachrichten und Repräsentationen enthalten immer alle nötigen Informationen zur Verarbeitung. Dies erlaubt eine gute Skalierbarkeit, da der Server keinerlei Informationen zur Kommunikation mit einem Client speichern muss.

Manipulation Ressourcen können durch Clients angelegt, modifiziert oder gelöscht werden, sofern diese die Rechte dazu besitzen. In webbasierten RESTful Systemen, die auf HTTP aufbauen, werden dazu die HTTP-Methoden GET, PUT, POST und DELETE verwendet.

Hypermedia RESTful Webservices nutzen Hypermedia-Formate zur Repräsentation von Ressourcen. Üblich sind u. a. die Extensible Markup Language (XML), die Hypertext Markup Language (HTML) und die Javascript Object Notation (JSON).

2.2.1. Constrained Application Protocol

RESTful Webservices mittels HTTP sind eine einfache Methode, um Daten für Clients bereitzustellen. Im Kontext des IoT bietet es sich daher an, die Metainformationen von Sensorknoten und Sensoren, sowie die Sensorwerte in Form von Ressourcen auf Basis von RESTful Webservices anzubieten. Aller-

dings hat HTTP den Nachteil der Nutzung von Zeichenketten, um die Methoden, Optionen und Parameter zu spezifizieren. Dies ist im Umfeld von ressourcenbeschränkten Systemen sowohl bei der Übertragung als auch bei der Verarbeitung ungeeignet.

Daher hat die Constrained-RESTful-Environments-Arbeitsgruppe (CoRE-Arbeitsgruppe) der Internet Engineering Task Force (IETF) das Constrained Application Protocol (CoAP) [72] definiert. Dieses hat u. a. die Aufgabe, die für ressourcenbeschränkte Systeme sinnvollen Teile von HTTP effizient umzusetzen. Dazu werden für die bekannten Funktionen GET, PUT, POST und DELETE sowie einige Headerfelder, wie *Content-Format*, *Max-Age* und *Accept*, binäre Kodierungen definiert. Zusätzlich zu dieser Untermenge von HTTP-Funktionen bietet CoAP Möglichkeiten zum Auffinden von Ressourcen (Discovery), Multicast-Unterstützung, Caching- und Proxyfunktionalitäten sowie asynchrone Nachrichtenübertragung.

CoAP ist vor allem für die machine-to-machine-Kommunikation konzipiert worden, da die Protokollnachrichten effizient zu übertragen und zu verarbeiten sind. Es nutzt dabei ebenso wie HTTP eine Request-Response-Interaktion. Konsequenterweise werden dabei 6LoWPAN, eine für ressourcenbeschränkte Systeme konzipierte Variante des IPv6, und das User Datagram Protocol (UDP) genutzt, um auf dem kompletten OSI-basiertem Protokollstapel ressourcenschonende Protokolle zu verwenden. Durch die Verwendung von verbindungslosem UDP statt dem verbindungsorientierten Transmission Control Protocol (TCP) wird auch der Overhead auf der Transportebene verringert. Dadurch und durch weitere Mechanismen, wie stellvertretende Proxys, ist es für die Geräte möglich, die meiste Zeit in einem Schlafzustand zu verbringen und so Energie zu sparen.

Neben den vordefinierten Headerfeldern, die bei CoAP Optionen heißen, ist es möglich, eigene Optionen zu definieren. Dabei kann die Verwendung einer Option in einer Anfrage bzw. Antwort als optional (*elective*) oder notwendig (*critical*) definiert werden. Weiterhin ist es möglich, Optionen als wiederholbar (*repeatable*) zu kennzeichnen, wodurch sie in einer Nachricht mehrfach vorkommen können.

2.2.2. CoAP Observe

CoAP bietet die Möglichkeit, bei Interesse an dem Zustand einer Ressource diese abzurufen. Möchte ein Client allerdings die Zustandsänderungen der Ressource über einen Zeitraum überwachen, so muss er diese regelmäßig anfordern. Dabei kann es dennoch geschehen, dass Änderungen erst spät erkannt werden und zwar dann, wenn sich der Zustand der Ressource genau nach einer Abfrage ändert. Erst mit der nächsten Anfrage wird diese Änderung erkannt. Um so eine Änderung schnell festzustellen, bleibt nur die Möglichkeit, dass der Client die Zeit zwischen Anfragen herabsetzt. Allerdings ist es wahrscheinlich, dass meistens der gleiche Zustand zurückgegeben wird und nur selten Änderungen geschehen. Außerdem würde ein Herabsetzen die aufzuwendenden Ressourcen

erhöhen. Für eine dauerhafte Überwachung wurde daher die *CoAP-Observe-Option* [73] definiert.

Die CoAP-Observe-Option erlaubt es Clients, einem Server mitzuteilen, dass er über Änderungen einer Ressource informiert werden möchte. Dazu fügt er einem GET-Request an eine Ressource die Observe-Option hinzu. Der Server speichert dies und schickt dem Client jedes Mal eine Benachrichtigung (*Notification*), wenn sich der Zustand der Ressource ändert, ohne, dass der Client erneut Requests schicken muss. Das Speichern der Observe-Beziehung widerspricht dabei dem Grundsatz der Zustandslosigkeit von RESTful Webservices. Dies wird allerdings toleriert, um den Vorteil des Ressourcensparens zu erhalten.

Die *Max-Age-Option*, die in einer Response gesetzt werden kann, gibt an, wie lange der Inhalt der Antwort Gültigkeit besitzt. Nach dieser Zeit darf der Client den Inhalt nicht mehr nutzen, da er als veraltet gilt. Ist die Option nicht enthalten, so gilt der Standardwert von 60 Sekunden. Da Clients, welche die Observe-Option nutzen, immer den aktuellen Zustand einer Ressource haben möchten, sollen Server nicht nur bei Änderungen des Zustands eine Benachrichtigung schicken, sondern auch vor Ablauf der durch *Max-Age* definierten Gültigkeitszeit.

Eine Observe-Beziehung kann durch einen Client gekündigt werden, indem bei Erhalt einer Notification mit einer Reject-Nachricht geantwortet wird.

2.2.3. CoAP Conditional Observe

CoAP Conditional Observe [74] ist eine Option, welche die CoAP-Observe-Option erweitert. Die Motivation für die Erweiterung ist, dass Clients, obwohl sie an Änderungen der Ressource interessiert sind, ggf. nicht jede kleine Änderung mitgeteilt bekommen möchten. So kann es z. B. sein, dass ein Client nur ganzzahlige Temperatursprünge einer Temperaturressource erhalten möchte oder es reicht ihm aus, wenn er erfährt, wann die Temperatur einen bestimmten Wert unterschreitet. Durch die Einschränkung auf solche Bedingungen können noch einmal Kommunikationsressourcen gespart werden, da weniger Nachrichten zwischen Client und Server versendet werden.

Die Conditional-Observe-Option wird zusätzlich zur Observe-Option dem GET-Request zugefügt. Die Option beschreibt, welche Bedingung eintreten muss, damit der Server eine Benachrichtigung an den Client sendet. Tabelle 2.2 listet die verfügbaren Bedingungen (Condition-Typ) auf. *Cancellation* signalisiert, dass eine bestehende Conditional-Observe-Beziehung gelöscht werden soll, d. h. der Server soll keine weiteren Benachrichtigungen schicken. Bei dem Typ *Time Series* erhält der Client eine Benachrichtigung für jede Änderung. Dies ist vergleichbar mit der normalen Observe-Option. Im Gegensatz zu dieser wird allerdings keine Nachricht verschickt, falls Max-Age überschritten wird. *Cancellation* und *Time Series* beinhalten keinen zusätzlichen Parameter, während alle folgenden Typen einen Wert mitgegeben bekommen, der die Bedingung parametrisiert. *Minimum response time* und *Maximum response time* legen die

Condition-Typ	Parameter
Cancellation	nein
Time series	nein
Minimum response time	ja
Maximum response time	ja
Step	ja
AllValues<	ja
AllValues>	ja
Value=	ja
Value<>	ja
Periodic	ja

Tabelle 2.2.: Condition-Typen der CoAP-Conditional-Observe-Option [nach 74, Abschnitt 4, Tabelle 3].

minimale bzw. maximale Zeit in Sekunden fest, die der Server zwischen zwei Benachrichtigungen warten soll. Das Beispiel mit ganzen Temperaturschritten lässt sich mit dem *Step*-Typ realisieren. Er gibt die minimale Änderung seit der letzten Benachrichtigung an, nach der die nächste gesendet werden soll. Bei den Typen *AllValues*<, *AllValues*> und *Value*= sollen Benachrichtigungen immer dann verschickt werden, wenn bei Änderungen der neue Wert kleiner, größer bzw. gleich dem angegebenem Parameter ist. *Value*<> dagegen verschickt nicht bei jedem Wert, der ungleich ist, eine Benachrichtigung, sondern kann als Schwellwert betrachtet werden, der über- bzw. unterschritten werden muss. Angenommen, 20 ist der Schwellwert und der Zustand der Ressource wechselt von 18 auf 19, so wird keine Benachrichtigung verschickt. Sobald ein Wert über 20 auftritt, gibt es eine Benachrichtigung. Anschließend wird erst dann erneut eine verschickt, wenn der Wert unter 20 fällt usw. Mit dem letzten Typ, *periodic*, kann der Client ein Intervall in Sekunden angeben, zu denen er jeweils eine Benachrichtigung erhalten möchte.

Die Conditional-Observe-Option ist wiederholbar. Eine mehrfache Nutzung in einem Request wird als logische UND-Verknüpfung vom Server interpretiert. Somit kann ein Client z. B. fordern, dass er Benachrichtigungen nur für Werte über 20 (*AllValues*>), die sich jedoch um mindestens 2 seit der letzten Benachrichtigung verändert haben (*Step*), erhalten möchte. Problematischerweise können damit auch Bedingungen erzeugt werden, die nie eintreten und somit keine Benachrichtigungen auslösen, aber dennoch zur Prüfung auf dem Server Ressourcen verbrauchen. Ein Beispiel wäre die Nutzung von *AllValues*< und *AllValues*> mit dem gleichen Wert, denn der Zustand einer Ressource kann nie gleichzeitig größer und kleiner als ein bestimmter Wert sein.

Auch eine logische ODER-Verknüpfung ist möglich. Dazu muss jede Teilaussage in einem eigenen GET-Request geschickt werden. Wichtig ist, dass jeder Request von einem anderen *source endpoint*, d. h. einer anderen Adresse kommt. Dies kann z. B. durch Verwendung unterschiedlicher Ports auf Clientseite realisiert werden. Das Senden mehrerer Requests vom gleichen *source endpoint* wird als

Update-Mechanismus genutzt, d. h. der Server löscht die bisherige Conditional-Observe-Beziehung und legt dafür die in dem Request spezifizierte als neue Beziehung an.

2.3. Semantic Web

Zur Darstellung von Informationen im Internet entwickelte Tim Burners Lee 1989 HTML und legte damit die Basis für das World Wide Web (WWW). Nach HTML 4.01 [75] im Jahr 1999 wurde im Jahr 2000 der Standard Extensible Hypertext Markup Language (XHTML) 1.0 [76] verabschiedet, der die Konzepte von HTML mit den Vorteilen von XML [77] verbindet. Der Vorteil von XML gegenüber HTML ist die Vorschrift, dass alle öffnenden Auszeichnungselemente, auch Tags genannt, wieder mit den entsprechenden Pendants geschlossen werden müssen. Dadurch ergibt sich eine wohlgeformte Dokumentenstruktur. Dies wiederum erlaubt eine bessere, fehlerrobuste Verarbeitung durch Maschinen. XHTML schreibt zwar die Struktur von Dokumenten vor, aber nicht die des Inhalts. Der unstrukturierte Inhalt ist somit weiterhin nicht für Maschinen verständlich.

Die Idee des Semantic Web ist es daher, Standards zu definieren, die es ermöglichen das von Menschen in Dokumenten abgelegte Wissen für Maschinen „verständlich“ zu machen. Dazu gehört, dass die Maschinen die gespeicherten Informationen verarbeiten, deren Semantik verstehen und aus bestehendem Wissen sogar neues erzeugen können. Das Semantic Web hilft u. a. dabei Doppeldeutigkeiten von Begriffen zu beseitigen. Außerdem lassen sich Beziehungen zwischen Wörtern, z. B. gleiche Bedeutungen (Synonyme), modellieren. Dies kann genutzt werden, um das Problem, dass Menschen oftmals unterschiedliche Begriffe für die gleiche Sache nutzen [22] zu lösen. Das Semantic Web soll das WWW nicht ersetzen, sondern es ergänzen.

Im Nachfolgenden werden zwei grundlegende Standards des Semantic Web, das Resource Description Framework und die SPARQL Protocol and RDF Query Language, erklärt. Der Abschnitt endet mit einer Beschreibung zwei weiterer Standards: SPARQL/Update und GeoSPARQL.

2.3.1. Resource Description Framework

Das Resource Description Framework (RDF) in Version 1.0 [78] und 1.1 [79] ist ein vom W3C definierter Standard im Bereich des Semantic Web zur strukturierten Beschreibung von logischen Aussagen.

Eine logische Aussage wird in RDF aus den drei Elementen Subjekt, Prädikat und Objekt gebildet. Aufgrund der drei Elemente wird eine solche Aussage auch Tripel genannt.

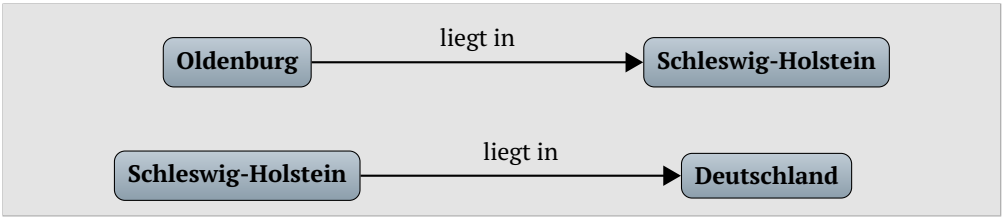


Abbildung 2.3.: Zwei Aussagen in Form von Tripeln dargestellt als gerichtete Graphen.

Abbildung 2.3 zeigt, wie Aussagen als gerichteter Graph interpretiert werden können. Subjekt und Objekt stellen Knoten der Graphen dar, während das Prädikat eine gerichtete Kante vom Subjekt zum Objekt ist. Das Prädikat im Beispiel ist zum besseren Verständnis um eine Präposition ergänzt.

Sofern das Prädikat *liegt in* als transitive Eigenschaft definiert ist, kann daraus die neue Aussage

Oldenburg liegt in Deutschland

und somit neues Wissen geschlussfolgert werden.

Da in der menschlichen Sprache viele Doppeldeutigkeiten auftreten, „weiß“ die Maschine bei der ersten Aussage nicht, um welches Oldenburg es sich handelt. Es kann Oldenburg in Holstein, Oldenburg in Oldenburg oder eines von vier weiteren Oldenburgs in den Vereinigten Staaten gemeint sein. Um Doppeldeutigkeiten aufzulösen, werden in RDF deshalb keine normalsprachlichen Worte genutzt, sondern URIs in Version 1.0 und Internationalized Resource Identifiers (IRIs) in Version 1.1. Im weiteren Verlauf der Arbeit wird nur noch URI genutzt, da der Standard 1.1 während der Durchführung der Arbeit noch nicht verabschiedet war. So würde man in RDF für jede dieser Städte eine eigene URI verwenden, um diese eindeutig zu identifizieren. Man spricht anstatt von URIs vereinfacht von Ressource. Dies ist ähnlich zu dem Konzept der Ressource in RESTful Webservices, bei dem es ebenfalls um eindeutige Identifizierung geht.

Subjekt, Prädikat und Objekt können jeweils URIs sein. Subjekt und Objekt dürfen außerdem *blank nodes* sein. Dies sind anonyme Ressourcen, die vor allem bei der Gruppierung von Aussagen genutzt werden. Das Objektelement darf im Gegensatz zu Subjekt und Prädikat auch ein Literal sein. Ein Literal ist eine beliebige Zeichenkette. In Version 1.0 kann optional ein Datentyp angegeben werden, während dies in Version 1.1 obligatorisch ist. Datentypen werden über IRIs referenziert. Die meisten vordefinierten Datentypen sind dem XML-Schema-Namensraum [80] entnommen. Es können auch eigene Datentypen definiert werden. Der GeoSPARQL-Standard bedient sich z. B. dieser Möglichkeit (siehe Abschnitt 2.3.3).

Abbildung 2.4 zeigt die Hierarchie der in XML Schema verfügbaren Datentypen. Die Typen unterhalb von anySimpleType sind in RDF nutzbar, wobei einige laut

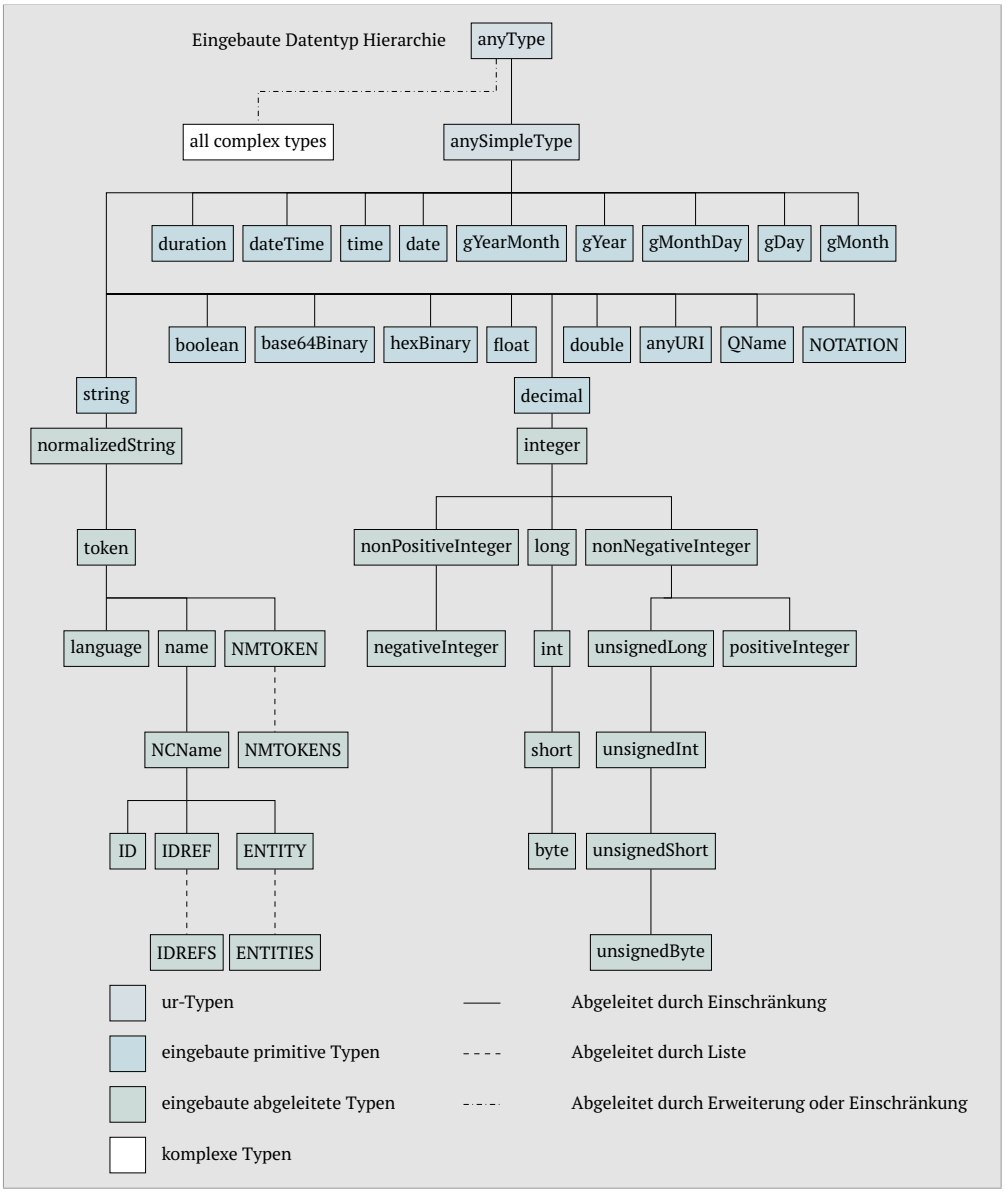


Abbildung 2.4.: XML Schema Datentypen [nach 81].

Standard nicht genutzt werden sollten. Damit können verschiedene Zahlen-, Datums- und String-Typen spezifiziert werden.

Durch die Wiederverwendung von Ressourcen können sich aus kleinen Graphen große ergeben. Ersetzt man im Eingangsbeispiel die Wörter durch entsprechende URIs, ergibt sich der Graph in Abbildung 2.5. Hier ist exemplarisch auch ein Tripel hinzugefügt worden, das mit einem Literal als Objekt eine Aussage über die Einwohnerzahl von Oldenburg trifft. URIs werden im Verlauf der Arbeit als Elemente mit runden Kanten repräsentiert, während Literale als Rechtecke dargestellt werden. Die Umrandung der Rechtecke visualisiert dabei den

Abbildung 2.5.: Ein exemplarischer RDF-Graph für das Beispiel aus Abbildung 2.3. URIs werden als Elemente mit runden Kanten repräsentiert, während Literale als Rechtecke dargestellt werden.

Literaltyp	Rahmentyp	Beispiel
Zahlentypen	durchgezogen	**21**
Stringtypen	gepunktet	**”Beispiel”**
Datums- und Zeittypen	gestrichelt	**2014-01-01T00:00:00**
Well-known Text-Typen	strichpunktiert	**”POINT(53.8 10.7)”**

Tabelle 2.3.: Literaltypen und deren Visualisierung durch verschiedene Rahmentypen.

Datentyp des Literals. Tabelle 2.3 listet die genutzten Umrandungen und den zugehörigen Datentyp auf.

Um RDF maschinenlesbar darzustellen, stehen verschiedene Serialisierungsformen zur Verfügung. Quelltext 2.1 zeigt das Beispiel aus Abbildung 2.5 in der Terse RDF Triple Language (Turtle) [82]. In der vorliegenden Arbeit wird ausschließlich die Turtle-Notation genutzt, da sie kompakter und leichter lesbar ist als die XML-basierte Serialisierung [83]. Des Weiteren erlaubt sie im Gegensatz zur N-Triple-Serialisierung [84] das Definieren von Präfixen, welches die Lesbarkeit erhöht und die Datenmenge oftmals reduziert (siehe Quelltext 2.2).

2.3.2. SPARQL Protocol and RDF Query Language

Die SPARQL Protocol and RDF Query Language (SPARQL) ist eine Abfragesprache für RDF, die vom W3C standardisiert wird. Zurzeit existieren die Version 1.0 [85] und 1.1 [86]. Die Abfrage ist graph-basiert, d. h. es wird versucht den Graphen, der durch die Anfrage definiert wird, mit dem RDF-Graphen zur Deckung

```
<http://de.dbpedia.org/page/Oldenburg_in_Holstein> <http://www.
    example.de/liegt-in> <http://de.dbpedia.org/page/Schleswig-
    Holstein> .
<http://de.dbpedia.org/page/Schleswig-Holstein> <http://www.
    example.de/liegt-in> <http://de.dbpedia.org/page/Deutschland>
     .
<http://de.dbpedia.org/page/Oldenburg_in_Holstein> <http://www.
    example.de/hat-Einwohner> "9615"^^<http://www.w3.org/2001/
    XMLSchema#int> .
```

Quelltext 2.1: Turtle-Notation der Tripel aus dem Beispiel aus Abbildung 2.5.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
@prefix ex: <http://www.example.de/>
@prefix db: <http://de.dbpedia.org/page/>

db:Oldenburg_in_Holstein ex:liegt-in db:Schleswig-Holstein .
db:Schleswig-Holstein ex:liegt-in db:Deutschland .
db:Oldenburg_in_Holstein ex:hat-Einwohner "9615"^^xsd:int .
```

Quelltext 2.2: Turtle-Notation der Tripel mit Präfixen aus dem Beispiel in Abbildung 2.5.

zu bringen. Die Elemente des RDF-Graphen, die an der Stelle der Variablen des Anfragegraph stehen, sind die Ergebnisse der Anfrage.

Die Syntax von SPARQL-Anfragen ist an die der Structured Query Language (SQL) angelehnt. Quelltext 2.3 zeigt eine Beispielanfrage, um die 20 größten Städte mit mehr als 5.000 Einwohnern sowie die Bundesländer, in denen diese Städte liegen, zu selektieren. In Zeile 1 wird ein Präfix definiert. Durch die Nutzung von Präfixen wird, wie in den RDF-Daten auch, eine höhere Übersichtlichkeit erreicht. Variablen werden in SPARQL durch ein vorangestelltes Fragezeichen gekennzeichnet. Das *SELECT* gefolgt von den zwei Variablen in Zeile 3 gibt an, welche Variablen das Ergebnis enthalten soll. Anschließend wird im *WHERE*-Block durch die Angabe von Tripelmustern (engl.: triple pattern) der Anfragegraph definiert (Zeile 5–7). Hier kommen erneut die aus dem *SELECT*-Block bekannten Variablen vor. Es können auch weitere Variablen Verwendung finden, die später nicht in der Ergebnismenge auftauchen. Die Anfrage nach einem triple pattern wird atomare Anfrage (engl.: atomic query) genannt. Mehrere triple pattern ergeben eine konjunktive Anfrage (engl.: conjunctive query). In Zeile 6 ist zu erkennen, dass sowohl vollständige als auch durch Präfix abgekürzte IRIs verwendet werden können. Durch die Angabe von *FILTERn* (Zeile 8) lasen sich bestimmte Tripel herausfiltern. Im Beispiel werden nur Städte mit mehr als 5.000 Einwohnern selektiert. Je nach Datentyp der Variablen können im *FILTER* u. a. Vergleichsoperatoren, String- und Datumsfunktionen (siehe [86]) zur Filterung genutzt werden. Die Nutzung von *FILTERn* nennt man auch Bereichsanfragen (engl.: range query), da nur ein Bereich der Daten selektiert wird. Weiterhin ist es möglich, in *FILTERn* benutzerdefinierte Funktionen anhand ihrer IRI aufzurufen, um eigene *FILTER* zu realisieren.

```
1  PREFIX db: <http://de.dbpedia.org/page/>
2
3  SELECT ?city ?state
4  WHERE {
5    ?city <http://www.example.de/liegt-in> ?state .
6    ?state <http://www.example.de/liegt-in> db:Deutschland .
7    ?city <http://www.example.de/hat-Einwohner> ?inhabitants .
8    FILTER (?inhabitants > 5000)
9  }
10 ORDER BY DESC(?inhabitants)
11 LIMIT 20
```

Quelltext 2.3: SPARQL-Anfrage, um die 20 größten Städte mit mehr als 5.000 Einwohnern sowie die Bundesländer in denen diese liegen zu selektieren.

Diese Funktionen müssen dafür im SPARQL-Endpunkt implementiert sein. Die selektierten Ergebnisse können noch geordnet (Zeile 10) und in ihrer Anzahl beschränkt (Zeile 11) werden.

Neben *SELECT*-Anfragen gibt es noch drei weitere Abfragetypen: Die *CONSTRUCT*-Abfrage hat als Ergebnis im Gegensatz zu den tabellarischen Ergebnissen einer *SELECT*-Anfrage wieder einen RDF-Graphen. Um zu testen, ob bestimmte triple pattern in den RDF-Daten existieren, gibt es die *ASK*-Anfragen, die als Ergebnis einen Wahrheitswert zurückgeben. *DESCRIBE*-Anfragen geben einen RDF-Graphen zurück, der eine Beschreibung der Ressourcen, die durch die Anfrage angegeben wurden, darstellt. Der Standard schreibt nicht genau vor, wie der resultierende Graph aufgebaut ist. Dies wird vom SPARQL-Endpunkt entschieden.

Weitere Operatoren erlauben das Vereinigen (*UNION*) von mehreren RDF-Graphen, das Filtern doppelter Ergebnisse (*DISTINCT*) und das versetzte Beginnen innerhalb der Ergebnismenge (*OFFSET*). Mit *OPTIONAL*-Blöcken ist es möglich, triple pattern zu definieren, die nicht notwendigerweise in den RDF-Daten vorkommen müssen. Falls sie vorkommen, werden sie mitselektiert und können z. B. für das Ergebnis genutzt werden. Dies entspricht somit einem left outer join in der Relationenalgebra.

Mit SPARQL können vorhandene Daten abgefragt werden. Sie erlaubt jedoch nicht das Modifizieren von Daten. Dazu wurde SPARQL/Update (SPARUL) [87] definiert. SPARUL ist eine Erweiterung von SPARQL, die das Einfügen, Aktualisieren und Löschen von Tripeln mit SPARQL-ähnlicher Syntax ermöglicht.

2.3.3. GeoSPARQL

GeoSPARQL [88, 89] ist eine Spracherweiterung für SPARQL, welche vom Open Geospatial Consortium (OGC)[3] definiert wird. Im Standard werden Datentypen und Funktionen zur Repräsentation und Abfrage von Geometrien definiert.

[3] `http://www.opengeospatial.org/` [Zugriff am 21. März 2014]

Simple Feature	Egenhofer	Region Connection Calculus 8
equals	equal	equal
disjoint	disjoint	disconnected
intersects	¬disjoint	¬disconnected
touches	meet	externally connected
within	inside + coveredBy	non-tangential proper part + tangential proper part
contains	contains + covers	non-tangential proper part inverse + tangential proper part inverse
overlaps	overlap	partially overlapping

Tabelle 2.4.: Relationen von Simple Feature, Egenhofer und RCC8 [nach 89].

Der Standard dient zum Beschreiben von geometrischen Formen, wie Punkten, Linien, Polygonen, Multipunkten, Multilinien und Multipolygonen in RDF. Dazu kann eine Repräsentation als Geography Markup Language (GML)[4], einem XML-Dialekt, oder als Well-known Text (WKT)[5], eine kompakte textuelle Darstellung, genutzt werden. Ein Beispiel zur Definition eines Polygons mit WKT findet sich in Abbildung 5.8 in Kapitel 5. Das Polygon wird durch mehrere Punkte beschrieben. Die Geometrie, in diesem Fall das Polygon, wird mit dem durch GeoSPARQL definierten Prädikat *geo:hasGeometry* an ein Subjekt gebunden. Dadurch wird ausgedrückt, dass das Subjekt diese Geometrie besitzt. Es können verschiedene Geoetrien für ein Objekt angegeben werden (z. B. unterschiedlich genaue Geometrien). Mittels des Prädikats *geo:hasDefaultGeometry* kann die standardmäßig zu verwendende Geometrie festgelegt werden.

Randell et al. [90] beschreiben Funktionen und Relationen, um Aussagen über Geometrien zu treffen. Damit kann u. a. Gleichheit, Schnitt, Überlappung und Berührung ausgedrückt werden. Eine Untermenge dieser Region Connection Calculus (RCC) genannten Logik ist RCC8, die aus acht Relationen besteht, mit der sich alle weiteren Relationen von RCC herleiten lassen. Ähnliche Relationen wurden von Egenhofer [91] und vom OGC als Simple Feature [92] definiert. Tabelle 2.4 zeigt, wie die Relationen der drei Definitionen im Verhältnis zueinander stehen. GeoSPARQL unterstützt alle drei Varianten, indem es für die jeweils acht Relationen entsprechende Filter-Funktionen definiert.

Eine weitere Klasse von Funktionen erlaubt das Erzeugen neuer Geometrien aus vorhandenen. So kann u. a. die konvexe Hülle, die Vereinigung, der Schnitt, die Differenz oder die symmetrische Differenz berechnet werden. Zusätzliche Funktionen erlauben das Berechnen der Distanz zwischen zwei Geometrien, das Umwandeln zwischen GML und WKT, sowie boolesche Überprüfungen zur Ermittlung, ob eine Geometrie leer oder einfach ist. Eine Geometrie ist einfach, wenn sie keine Überschneidungen mit sich selbst hat.

[4] `http://www.opengeospatial.org/standards/gml/` [Zugriff am 21. März 2014]
[5] `http://www.opengeospatial.org/standards/sfa/` [Zugriff am 21. März 2014]

3 Nutzerorientierte Entwicklung

> The old computing was about what computers could do; the new computing is about what users can do. Successful technologies are those that are in harmony with users' needs. They must support relationships and activities that enrich the users' experiences.
>
> — *Ben Shneiderman*

Bei der Realisierung neuer Produkte und Systeme ist die Einbeziehung von Nutzern in den Design- und Entwicklungsprozess von großem Nutzen, da dadurch z. B. die Akzeptanz bei Nutzern gesteigert und Probleme mit der Gebrauchstauglichkeit minimiert werden können. Dabei ist es möglich, (potentielle) Nutzer zu verschiedenen Zeitpunkten im Entwicklungsprozess einzubeziehen. Eine Marktanalyse kann das Erfolgspotential untersuchen und z. B. offenlegen, ob ein Produkt bestimmte Eigenschaften besitzen soll, andere wiederum nicht benötigt werden oder eventuell gar kein Markt für das Produkt existiert. Dadurch lassen sich unnötige Entwicklungskosten vermeiden. Existiert bereits ein Prototyp, ist es möglich, diesen auf die Gebrauchstauglichkeit zu untersuchen und entsprechend dem Feedback zu verbessern. Eine Evaluation eines Produktes, das bereits auf dem Markt verfügbar oder gar etabliert ist, bietet die Möglichkeit, auch an dieser Stelle auf Feedback der Nutzer einzugehen und durch Software-Updates Funktionen zu ergänzen oder Fehler zu beseitigen [93].

In diesem Kapitel wird anhand einer Befragung betrachtet, ob, wann, wie und wo Personen eine Suche für das IoT nutzen würden. Diese Analyse fragt nicht nur nach der generellen Akzeptanz solch einer Suche, sondern auch welche Funktionen sich Nutzer dafür wünschen. Die Ergebnisse der Umfrage helfen, die Technologien, die an der Schnittstelle zum Nutzer liegen bzw. diejenigen, die von Nutzerwünschen beeinflusst werden, auf die Bedürfnisse dieser zuzuschneiden.

Nach einer Diskussion verwandter Arbeiten, werden die Struktur, die Ergebnisse und Schlussfolgerungen der Umfrage in den folgenden Abschnitten beschrieben. Die wesentlichen Inhalte wurden vorab in [23] veröffentlicht.

3.1. Verwandte Arbeiten

Im Rahmen der Entwicklung von Gander [15, 16] (vgl. Abschnitt 1.2.1) wurde eine Nutzerstudie mit zwölf Teilnehmern durchgeführt. Dies geschah mit einem Prototypen, der auf dem in C++ implementierten Netzwerksimulator OMNet++[1] basiert. Die Studie hat ergeben, dass den meisten die Echtzeitsuchmöglichkeit gefällt. Es wurde gefragt, ob die Nutzer interessiert seien, ein solches System in der Zukunft zu nutzen und ob die GUI eine gute Gebrauchstauglichkeit aufweist. Auf der 4-Punkte-Skala antworteten die Teilnehmer durchweg entweder mit der besten oder mit der zweitbesten Bewertung. Allerdings erlauben die geringe Zahl von zwölf Studienteilnehmern und die Beschränkung auf bestimmte Szenarien durch die Nutzung eines konkreten Systems keine allgemeinen Schlussfolgerungen für eine Sucharchitektur für das IoT.

Gaonkar et al. [94] stellen ein System für Mobiltelefone vor, bei dem Nutzer sogenannte *Micro-Blogs* erstellen können. Dies sind kleine Informationseinheiten, die statische Daten wie Text und dynamische Daten von Sensoren des Handys beinhalten. Aufgrund der Position des Nutzers werden diese auf einer öffentlich verfügbaren Karte angezeigt und sind so von anderen Interessierten betrachtbar. Zusätzlich können Nutzer Anfragen für bestimmte Informationen in bestimmten Regionen erstellen. Als Beispiel wird die Frage nach Parkmöglichkeiten an einer Promenade genannt. Nutzer in dieser Region können dann einen Micro-Blog als Antwort erstellen. In einer Studie mit zwölf Freiwilligen wurde das System getestet. Das Ergebnis war, dass die Nutzer, sofern sie Zeit hatten, das System mit Spaß genutzt und auf Anfragen anderer Nutzer reagiert haben.

In den Arbeiten [95–98] werden ebenfalls Nutzerstudien durchgeführt. Allerdings ist der Fokus bei diesen Arbeiten auf Entwickler als Nutzer technischer Systeme gerichtet. Bei Guinard et al. [95] werden 69 Programmieranfänger gebeten, eine Anwendung für Mobiltelefone zu schreiben, die Sensordaten sowohl per REST als auch mit WS-* abruft. Entwickler präferieren laut dieser Umfrage REST, da dieses leichter zu erlernen sei. Andererseits würden

[1] `http://www.omnetpp.org/` [Zugriff am 21. März 2014]

sie für Anwendungen mit Quality-of-Service- und Sicherheitsanforderungen eher auf WS-* setzen. In den Arbeiten von Bai et al. [96], Miller et al. [97] und Eriksson et al. [98] werden jeweils Personen mit unterschiedlichen Programmierkenntnissen gebeten, einfache Anwendungen, meist zum Datensammeln, für Sensorknoten zu programmieren. Dabei werden existierende Lösungen mit neuen Ansätzen der Autoren verglichen. Diese neuen Ansätze sollen es sogar Programmierunerfahrenen ermöglichen, Programme für Sensorknoten zu schreiben. Die Ergebnisse zeigen, dass die eingeführten Abstraktionen und Erleichterungen die Programmierung vereinfachen und beschleunigen. Während sich diese Arbeiten auf Entwickler fokussieren, steht in der nachfolgend vorgestellten Umfrage der Nutzer im Mittelpunkt. Dennoch sollen die Ergebnisse Programmierern und Interfacedesignern bei der Entwicklung von Technologien für eine IoT-Suchmaschine zu Gute kommen.

Das IoT comic book [99] beschreibt verschiedene mögliche Zukunftsszenarien, die durch das IoT realisiert werden können. Einige der vorgestellten Anwendungsfälle werden als Anregung für die in der folgenden Umfrage verwendeten Szenarien genutzt. Da das IoT comic book für eine breite Öffentlichkeit gestaltet ist, gibt es weder Entwicklungshinweise noch wirft es offene Forschungsfragen auf.

3.2. Umfrage

Im weiteren Verlauf werden der Aufbau der Umfrage, die Techniken zur Auswertung sowie die Ergebnisse vorgestellt. Aus den Resultaten der thematischen Frageblöcke werden jeweils Schlussfolgerungen für das Design der verschiedenen Technologien gezogen, die in den folgenden Kapiteln vorgestellt werden.

3.2.1. Aufbau der Umfrage

Die Umfrage soll zwei Aspekte einer Suche im IoT beleuchten: einerseits das Nutzerverhalten bei der Verwendung einer solchen Suche und andererseits die Erwartungen der Nutzer an eine solche Suche. Beim Nutzerverhalten ist vor allem von Interesse, wann, wo, mit welchem Gerät, in welchem Kontext und wie häufig die Suche genutzt werden würde. Bei den Nutzererwartungen geht es vor allem um die Präsentation der Suchergebnisse, da Nutzer eine Suchmaschine nur regelmäßig nutzen, wenn sie mit den Ergebnissen und deren Darstellung zufrieden sind. Zusätzlich ist es interessant, die demographische Struktur der Teilnehmer zu erfassen, um Aussagen über die Relevanz der Ergebnisse geben zu können.

Aufgrund der beschriebenen Anforderungen wird eine Umfrage mit insgesamt 44 Fragen erstellt. Sofern nicht anders angegeben, ist pro Frage nur eine Antwort erlaubt. Die genauen Fragestellungen und Antwortmöglichkeiten sind in Anhang A.1 nachzulesen. Die Fragen verteilen sich auf fünf Frageblöcke. Zum besseren Verständnis werden diese Blöcke im weiteren Verlauf *Allgemeine*

Fragen, *Freitext Fragen*, *Allgemeine Nutzung*, *Szenario Nutzung* und *Ergebnis* genannt.

Der Block *Allgemeine Fragen* beinhaltet Fragen zu Geschlecht, Alter und Bildung sowie zu Vorwissen im Bereich IT, Google-Diensten und Google-Suchfunktionen. In der Umfrage wird Google verwendet, da dieses Unternehmen ein breites Angebot verschiedener Dienste anbietet und in Deutschland im Bereich Internetsuche den größten Marktanteil besitzt [100]. Am Ende des Blockes wird der Teilnehmer gefragt, ob er den Begriff „Internet der Dinge“ kennt und gebeten diesen ggf. per Freitext zu beschreiben. Die Fragen dieses Blockes dienen dazu, die demographische Struktur der Teilnehmer und deren Vorwissen zu untersuchen.

Alle nachfolgenden Fragen beschäftigen sich mit der Suche im IoT. Damit die Teilnehmer ein Grundverständnis vom IoT haben und somit die Fragen verstehen, ist dem zweiten Themenbereich eine kurze Erklärung der Idee des IoT vorangestellt. Danach folgt der Block *Freitext Fragen*, in dem sich die Teilnehmer Suchszenarien überlegen sollen, die sie sich für eine zustandsbasierte Suche vorstellen könnten. Anschließend folgen drei stetig komplexer werdende Szenarien, für welche die Teilnehmer aufschreiben sollen, wie sie bei Google danach suchen würden. Mit diesem Block soll evaluiert werden, wie kreativ Nutzer bei solch einer neuen Technologie sind und wie Suchanfragen aussehen, d. h. ob Teilnehmer ihre Anfragen in einer speziellen Syntax, als ganze Sätze oder mit Hilfe von Stichwörtern stellen.

Im mittleren Fragenblock, *Allgemeine Nutzung*, soll von den Teilnehmern ermittelt werden, wie sie eine Suche für das IoT nutzen würden. Dieser Block umfasst somit Fragen, an welchem Ort und mit welchem Gerät sie eine solche Suche nutzen würden. Weiterhin wird gefragt welche persönlichen Daten ein Teilnehmer von sich preisgeben würde, um bessere Suchergebnisse zu erhalten. So kann die Preisgabe, dass man jüdischen Glaubens ist, bei einer Suche nach Restaurants nur koschere Gaststätten als Ergebnis liefern. Die Freigabe der Freundesliste aus einem sozialen Netzwerk kann Orte höher stufen, die von Freunden als gut bewertet wurden.

Ähnliche Fragen zum Nutzerverhalten werden im Block *Szenario Nutzung* gestellt. In diesem Frageblock geht es allerdings um eine detailliertere Nutzungsanalyse spezieller Suchszenarien. Dazu werden den Teilnehmern die in Tabelle 3.1 aufgelisteten elf Szenarien vorgestellt. Eine Befragung zu jedem Szenario untersucht, mit welchem Gerät, wann, wie oft, wo und in welchem Kontext die Teilnehmer danach suchen würden. Diese beiden Blöcke sollen Ähnlichkeiten und Unterschiede im allgemeinem und szenarioabhängigem Nutzungsverhalten aufzeigen.

Die Umfrage schließt mit dem Block *Ergebnis*, in dem die Teilnehmer angeben sollen, wie sie die Ergebnisse einer Suche präsentiert haben möchten. Dies umfasst, wie viele Ergebnisse sie auf der ersten Seite sehen wollen, welche Informationen angezeigt werden sollen, in welcher Genauigkeit sie den Zustand der Entitäten angezeigt bekommen möchten und wonach die Ergebnisse

Kurzname	Beschreibung
Fahrrad	Suche nach einem freien Fahrrad an einer Ausleihstation
Auto	Suche nach einem freien Auto beim Car-Sharing
Taxi	Suche nach einem freien Taxi
Parken	Suche nach einem freien Parkplatz
Tisch	Suche nach einem freien Tisch in einem Restaurant
Warteschlange	Suche nach einem Imbiss mit kurzen Wartezeiten
Schwimmen	Suche nach Schwimmmöglichkeiten mit angenehmen Wassertemp.
Verlustsache	Suche nach verlorenem Gegenstand (mit Positionssensor)
Nahverkehr	Suche wo der öffentliche Nahverkehr ist (Bus/Zug)
Route	Suche nach staufreien Routen
Erholung	Suche nach einem sonnigen und ruhigen Ort

Tabelle 3.1.: Für die Umfrage verwendete Szenarien.

sortiert werden sollen. Das Ziel dieses Blockes ist, Hinweise auf die Gestaltung der Ergebnisseiten zu erhalten, um dem Nutzer ein zufriedenstellendes Sucherlebnis zu bieten, d. h. seine Erwartungen zu erfüllen und seine Suchanfrage zu beantworten.

Die Umfrage wurde online mit der Open-Source-Software LimeSurvey[2] durchgeführt. Zwar hat man im Gegensatz zu papierbasierten und persönlichen Umfragen keinen Einfluss auf die Auswahl der Teilnehmer, aber Gosling et al. [101], Bishop et al. [102] und Denscombe [103] zeigen, dass die Qualität der Ergebnisse von online durchgeführten Umfragen vergleichbar ist mit denen von traditionell durchgeführten. Die Umfrage war sieben Wochen lang geöffnet und wurde über verschiedene Mailverteiler der Universität zu Lübeck sowie über diverse Facebook-Gruppen verbreitet.

3.2.2. Ergebnisse der Umfrage

Die Antworten der Teilnehmer wurden aus der Umfragesoftware als Excel-Tabelle exportiert und mit Hilfe der Pivot-Tabellen-Funktion von Excel ausgewertet. Die Freitextantworten wurden manuell gruppiert und ausgezählt.

Zusammensetzung der Teilnehmer

Insgesamt schlossen 184 Teilnehmer die Umfrage ab. Weitere 91 Personen haben dagegen die Umfrage zwischenzeitlich abgebrochen. Die Teilantworten dieser Personen werden nicht in der Auswertung berücksichtigt.

Die Geschlechtsverteilung der Teilnehmer ist Abbildung 3.1 zu entnehmen. Sowohl über alle Teilnehmer hinweg als auch in den jeweiligen Altersgruppen ist die Verteilung nahezu ausgewogen zwischen männlichen und weiblichen

[2] `http://www.limesurvey.org/` [Zugriff am 21. März 2014]

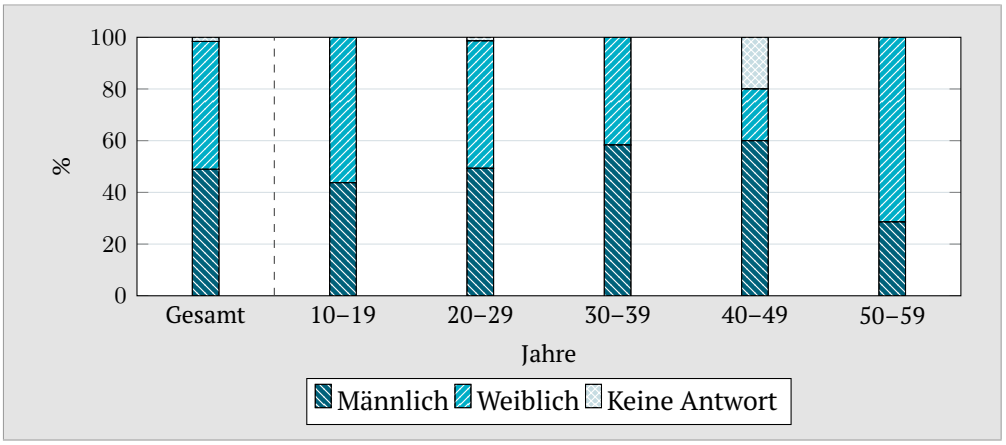


Abbildung 3.1.: Gesamtverteilung und nach Alter gruppierte Verteilung des Geschlechts.

Teilnehmern. Dies ist dagegen für die Altersverteilung nicht der Fall. Die Altersgruppe von 20 bis 29 ist mit 78,26 % überrepräsentiert. Dies lässt sich mit der Umfragewerbung erklären. Da hauptsächlich im universitären Umfeld für die Umfrage geworben wurde, ist anzunehmen, dass vor allem Studenten und Mitarbeiter der Universität erreicht wurden. Diese Hypothese lässt sich an zwei weiteren Punkten festmachen. Zum einen spiegelt die Geschlechterverteilung das Verhältnis der Studierendenschaft wider. Zum anderen legen auch die Verteilungen des Berufs in Abbildung 3.2 und des Bildungsgrades in Abbildung 3.3 nahe, dass es sich um Personen der Universität bzw. der Universitätsklinik handelt: Die meisten Teilnehmer sind Studenten (61,57 %), wissenschaftliche Mitarbeiter (9,26 %) oder Personen die im medizinischen Sektor angestellt sind (7,41 %).

Somit sind die Ergebnisse für eine gesamtdeutsche bzw. weltweite Population im Bezug auf die Altersverteilung nicht als repräsentativ anzusehen. Der interessante Aspekt, das Wissen um und die Nutzung von Internettechnologien, ist weit gestreut. So ist von Anfängern bis hin zu Experten jede Nutzergruppe vertreten. Rund ein Viertel (22,04 %) haben den Begriff „Internet der Dinge“ bereits vorher gehört und haben zumindest eine ungefähre Vorstellung seiner Bedeutung. Fast alle Teilnehmer kennen die Google-Dienste *Suche* (98,81 %), *Maps* (100 %) sowie deren Routing-Funktion (78,38 %). Dagegen sind ortsbasierte Dienste wie *Places* (48,37 %) und *Latitude* (32,61 %) weit weniger bekannt. *Latitude* ist inzwischen eingestellt. Des Weiteren ist zu erkennen, dass die Nutzung eines Dienstes unter denen, die diesen kennen umso geringer ausfällt, je weniger Personen den Dienst insgesamt kennen.

Auch wenn die Gruppe an Teilnehmern keine repräsentative Menge widerspiegelt, so gibt sie dennoch interessante Einblicke, wie zukünftige Nutzer sich eine Suche für das IoT vorstellen und wie sie eine solche Suche nutzen würden. Somit können die Ergebnisse gute Ansätze und Hinweise liefern, wie Technologien für eine Suche im IoT zu entwickeln sind.

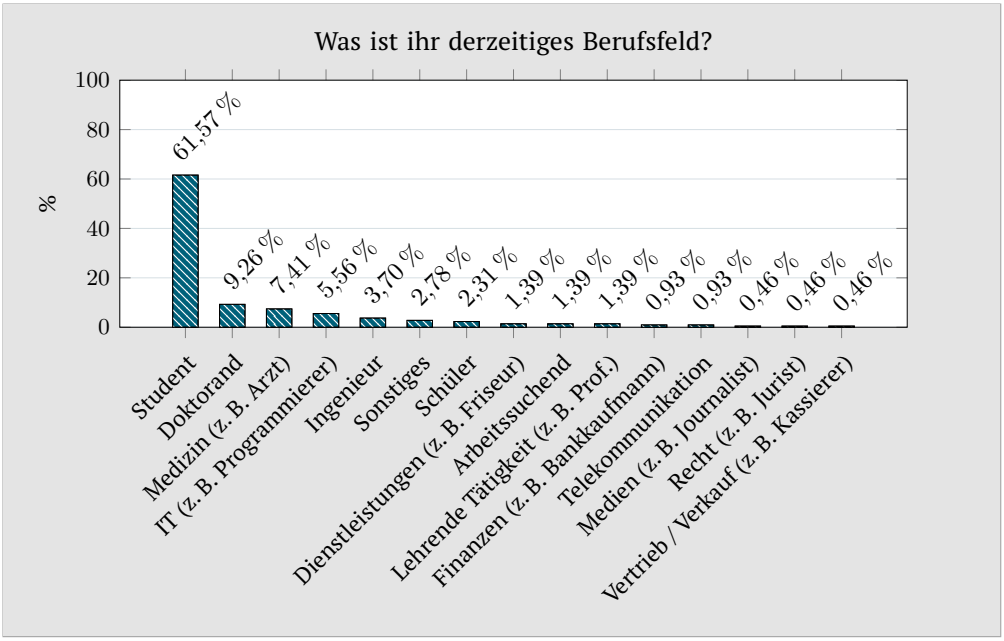


Abbildung 3.2.: Ergebnisse der Frage über den derzeitigen Beruf.

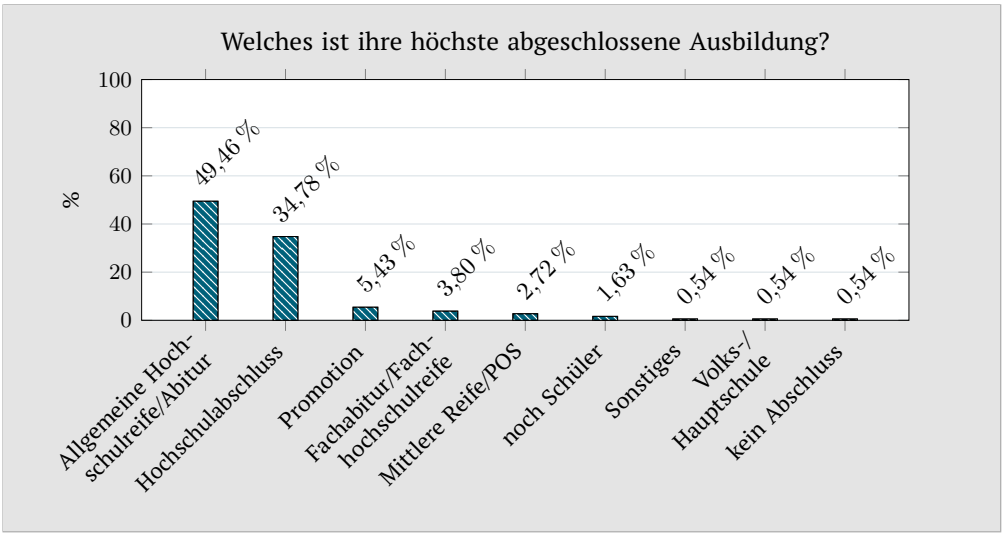


Abbildung 3.3.: Ergebnisse der Frage zur höchsten abgeschlossenen Ausbildung.

Wie stellen sich die Teilnehmer Suche im Internet der Dinge vor? – Freitextfragen

Im Fragenblock *Freitext Fragen* wird die Kreativität der Teilnehmer gefordert. Mit dem Ziel, die Umfrageteilnehmer nicht vorher in zu viele Anwendungsszenarien einzuführen, wird bis zu diesem Zeitpunkt der Umfrage nur ein konkretes Szenario, die Suche nach einem freien Tisch in einem Restaurant, genannt. Die

Wartezeit	Verfügbarkeit	Ort	Zustand
Kasse	Tisch in Restaurant	Nächster Nahverkehr	Stimmung in Bars
Attraktion im Freizeitpark	Kino-, Theater- oder Konzertticket	Verlorener Gegenstand	Temperatur / Wetter
Dienstleister	Parkplatz	Paketverfolgung	Hausgeräte
Behörde	Produkt im Supermarkt	Carsharing-Fahrzeug	Licht
Friseur	Sitz in Bibliothek	freie Bank	Benzinpreis
Arzt	Sitz im öffentlichen Nahverkehr	Wellen fürs Surfen	Füllstand von Papiercontainer
	Buch in Bibliothek	Freund	Heizung
	Carsharing-Fahrzeug	Haustier	Feuchtigkeit von Pflanzen

Tabelle 3.2.: Von den Teilnehmern genannte Szenarien.

Teilnehmer sollen überlegen, welche Suchszenarien sie sich vorstellen können. Rund 70 % beantworteten die Frage und gaben ein oder mehrere Szenarien an. Insgesamt wurden 297 Antworten gegeben, die 55 verschiedene Szenarien umschreiben. Die am häufigsten genannten Szenarien sind in Tabelle 3.2 aufgelistet und nach vier Charakteristiken gruppiert, welche die abgefragten Eigenschaften für die Szenarien allgemeiner beschreiben. Bei den Charakteristiken handelt es sich um die Wartezeit, die Verfügbarkeit, den Ort und den Zustand. Die Position in der Tabelle ist rein zufällig und sagt nichts über die Häufigkeit aus. Das am Häufigsten genannte Szenario war das Tisch-Szenario. Dies wird auf die vorherige Erwähnung als Beispiel im Fragebogen zurückgeführt. Einige Szenarien lassen sich unter verschiedenen Gesichtspunkten betrachten, d. h. sie können in unterschiedlichen Kategorien eingeordnet werden. Die Suche nach einem Carsharing-Fahrzeug kann bedeuten, dass der Nutzer die Anzahl an möglichen Fahrzeugen finden möchte oder einfach das nächstgelegene freie.

Von täglichen Aktivitäten, wie dem Einkaufen und dem Haushalt (Heizung und Haushaltsgeräte), über Lernen (Bücherei) und Freizeitaktivitäten (Erholung, Kultur und Sport) bis zu sozialen Interaktionen (Haustier, Freunde, Ausgehen) wird fast jeder Lebensbereich genannt. Nur das Arbeitsleben ist nicht vertreten. Dies lässt sich wiederum mit der Teilnehmermenge erklären. Diese besteht, wie eingangs erwähnt, hauptsächlich aus Studenten und nur aus wenigen Arbeitnehmern. Somit können die Szenarien, welche die Bibliothek adressieren als „Arbeitsumgebung“ eines Studenten gewertet werden.

In den weiteren drei Fragen dieses Blockes werden die Teilnehmer gebeten, Suchanfragen an eine fiktive Suchmaschine zu stellen. Dabei sollen sie

1. nach einem freien Tisch in einem Restaurant,
2. nach einem freien Tisch in einem warmen Restaurant und
3. nach einem freien Tisch in einem Restaurant mit einem freien Parkplatz in der Nähe

suchen. Die zu stellenden Anfragen werden zunehmend komplexer, um zu untersuchen, wie sich dies auf die Suchanfragen der Nutzer auswirkt. Etwa 90 % der Teilnehmer gaben Antworten bei diesen Freitextfragen. Die meisten von ihnen stellten die Anfragen in der Art wie heutzutage Suchmaschinen hauptsächlich genutzt werden: sie verwendeten Stichwörter. Allerdings lässt sich auch der Wunsch erkennen, dass die Suchmaschine natürliche Sprache versteht. Nur wenige nutzten dagegen eine spezielle Syntax, wie z. B. SQL. Der Durchschnitt/Median an genutzten Wörtern in den drei Suchanfragen betrug 4,23/3, 5,93/5 und 6,11/5. Dies zeigt deutlich, dass die Anzahl der benutzten Wörter trotz steigender Komplexität nur gering ansteigt. Der Median der zweiten und dritten Anfrage bleibt sogar identisch. Auch insgesamt werden für eine Suchanfrage nur wenige Worte verwendet. Die Suchabsicht, d. h. die Entität und den Zustand, hinter einer Suchanfrage mit so wenigen Suchwörtern zu verstehen, ist eine schwierige Aufgabe.

Schlussfolgerung. Die Ergebnisse dieses Frageblockes legen nahe, dass die Teilnehmer die Idee der Suche im IoT verstanden haben. Dabei unterscheidet sich die Suchformulierung kaum von heutigen Suchanfragen. Somit ist das Verarbeiten von natürlicher Sprache und das Verstehen des semantischen Inhalts der Anfragen ein wichtiger Punkt. Dies wird noch dadurch verstärkt, dass Smartphones und Desktop-Browser die Eingabe per Sprache vorantreiben. Gerade bei einer solchen Eingabemethode werden Menschen ihre Suchabsicht als Frage formulieren, anstatt einzelne Stichwörter zu sagen. Deswegen kann es sinnvoll sein, neben der eigentlichen Anfrage weitere Daten zu erheben, um der Suchmaschine das Erschließen der Semantik zu erleichtern. Zum Beispiel können der Ort, an dem sich der Anfragende befindet, die Uhrzeit, vorherige Anfragen, persönliche Daten und weitere Kontextinformationen Hinweise darauf geben was der Nutzer sucht. Auf diese Nutzungsmuster wird im nächsten Abschnitt genauer eingegangen.

Wie würden die Teilnehmer eine Suche im Internet der Dinge nutzen? – Allgemeine Nutzungsmuster

Im Abschnitt *Allgemeine Nutzung* geht es um allgemeine Nutzungsweisen einer Suche für das IoT. Bei allen Fragen in diesem Block sind Mehrfachantworten erlaubt.

Einleitend werden die Teilnehmer gefragt, mit welchen Geräten und an welchem Ort sie die Suche nutzen würden. Die Antworthäufigkeiten sind in Abbildung 3.4 zu sehen. Die meisten Teilnehmer würden das Smartphone (78,8 %) oder ein Notebook (72,28 %) nutzen. Im Verlauf der Umfrage wird die gleiche Frage für die einzelnen Szenarien aufgegriffen, um zu untersuchen, ob von diesen abhängt, wie die Nutzung ausfällt. Das Zuhause ist mit 84,78 % der am häufigsten genannte Ort. Dies deckt sich mit der Nutzung von Notebooks und Desktop-Computern, welche vornehmlich als stationär anzusehen sind. Weitere Einsatzgebiete, wie z. B. im öffentlichen Nah- und Fernverkehr oder als

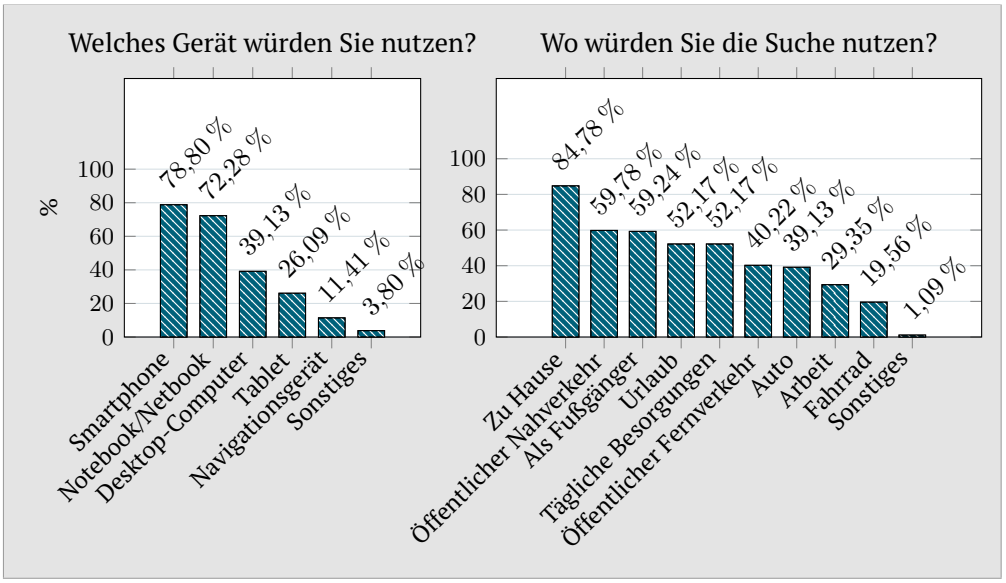


Abbildung 3.4.: Antworten auf die Fragen nach dem generellen Suchverhalten.

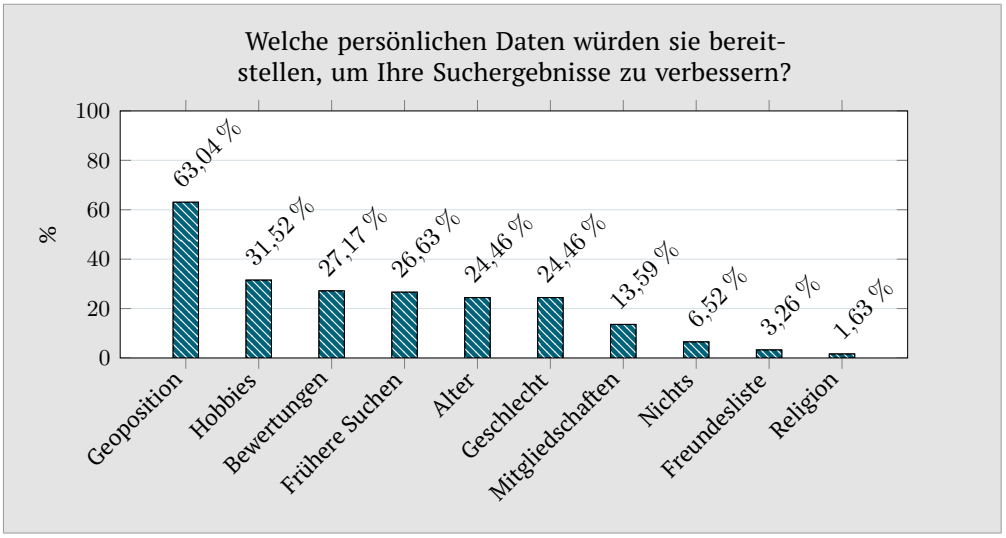


Abbildung 3.5.: Antworten auf die Frage über das Bereitstellen persönlicher Daten.

Fußgänger, werden ebenfalls oft genannt. Insgesamt wird die mobile Nutzung häufiger genannt als der Einsatz zu Hause. Dabei ist das Fahrrad am unbeliebtesten. Dies könnte an der Ablenkung und schwierigen Bedienung während des Fahrens liegen. Durch neue Eingabemethoden (z. B., Sprachsteuerung wie bei Android Wear für Wearables [104]) und Ausgabemethoden (akustisch oder Geräte wie Google Glass [105]) könnte die Suche in solchen Kontexten in Zukunft aber zunehmen.

In der letzten Frage des Blockes geht es um die Freigabe persönlicher Daten. Diese können zum einen helfen, die Suchanfrage besser zu verstehen, da Kontextinformationen zur Anfrage erfasst werden, und zum anderen die Ergebnisse zu verbessern. Es ließen sich z. B. Entitäten finden, die in der Nähe des Suchenden liegen, sofern dieser seine Position freigibt oder ein passendes Restaurant, falls der Suchende seine Essgewohnheiten preisgibt. Ein besseres Sucherlebnis steht damit der Freigabe von persönlichen Daten gegenüber. Erfahrungsgemäß stößt dies nicht auf große Zustimmung, was auch die Abbildung 3.5 widerspiegelt. Während die Position noch von der Mehrheit freigegeben würde (63 %), würden nur zwischen einem Viertel und einem Drittel der Teilnehmer andere Daten wie das Geschlecht, Alter oder Hobbys zur Verfügung stellen. Die Freundesliste aus einem sozialen Netzwerk oder die Religion würden nur 3,26 % bzw. 1,63 % teilen. Ein Anteil von 6,52 % würde sogar alle persönlichen Informationen geheim halten wollen. Einerseits war ein solches Ergebnis zu erwarten, andererseits ist anzunehmen, dass, wenn Nutzer den Vorteil der Freigabe von Daten sehen, sie eher bereit sind, die Nutzung zu erlauben. Diesen Sachverhalt kann man unter Umständen an den Zahlen zur Freigabe der Position herleiten. Moderne Smartphones senden standardmäßig Positionsdaten, die z. B. genutzt werden, um in Kartenanwendungen direkt die Umgebungskarten zu laden oder beim Routing die Auslastung von Straßen anzuzeigen [106]. Viele Nutzer machen sich wenig Gedanken über diese Freigabe ihrer Position und nutzen die Vorteile wie selbstverständlich.

Schlussfolgerung. Abbildung 3.4 zeigt, dass verschiedene Geräteklassen genutzt würden. Somit muss auf die individuellen Eigenschaften jeder Klasse beim Design von GUIs, d. h. der Such- und Ergebnisseite, geachtet werden. Sowohl zwischen diesen Geräten als auch innerhalb einiger Geräteklassen können die Unterschiede hinsichtlich Bildschirmgröße, -format, -auflösung, verfügbarer Bandbreite und der unterstützen Ein- und Ausgabemodalitäten stark variieren. Während z. B. auf einem Display mit hoher Auflösung eine Karten- und Listenansicht von Ergebnissen gleichzeitig präsentiert werden kann, sollte auf einem kleinen Bildschirm nur eine Ansicht angezeigt werden und dafür ein Wechsel zwischen den Ansichten möglich sein. Bei den Designentscheidungen sollten auch verschiedene Kontexte berücksichtigt werden. Eine Eingabe für die Bedienung während des Rad- oder Autofahrens, die möglichst wenig ablenkt und im Idealfall freihändig geschieht, wie z. B. per Sprache, könnte Unfälle durch abgelenkte Verkehrsteilnehmer verhindern. Eine der größten Herausforderungen ist die Handhabung persönlicher Daten. Dies ist eine generelle Schwierigkeit im IoT, in dem möglicherweise dutzende bis hunderte Sensoren eine Person überwachen und identifizieren. Um das Vertrauen der Nutzer nicht zu verlieren, sollten diese stets die Möglichkeit haben, sich zu informieren, welche Sensoren Daten erfassen und was für Daten dies sind. Weiterhin sollten sie selbst entscheiden dürfen, welche Daten sie von sich freigeben möchten.

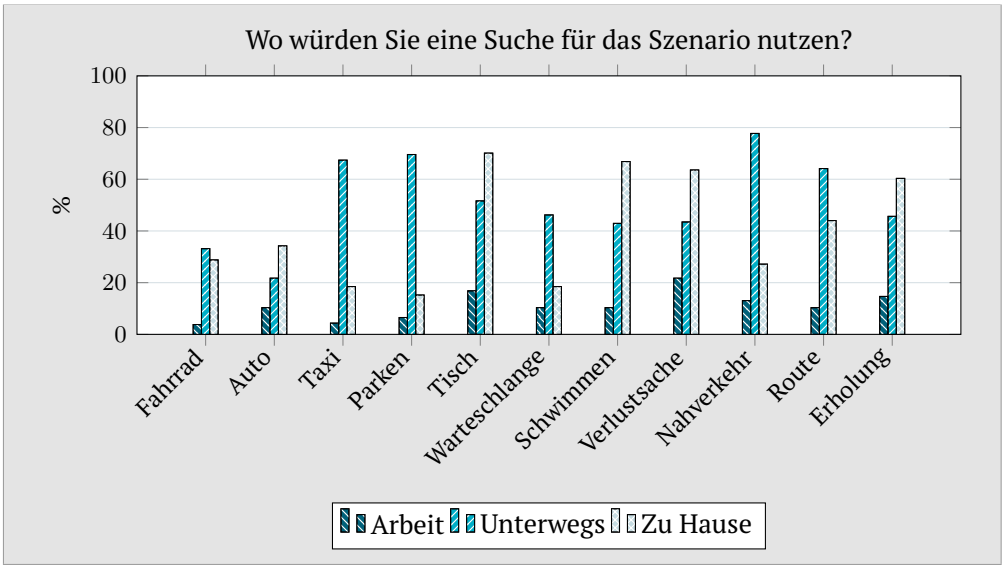


Abbildung 3.6.: Antworten auf die Frage zum Nutzungsort.

Wie würden die Teilnehmer eine Suche im Internet der Dinge nutzen? – Nutzungsmuster für spezifische Szenarien

Im vierten Frageblock werden Fragen zur Nutzung von spezifischen Suchszenarien gestellt. Die Szenarien sind in Tabelle 3.1 zu sehen. Es sind wiederum Mehrfachantworten erlaubt. Zuerst wird gefragt wo, in welchem Kontext und mit welcher Geräteklasse ein Szenario gesucht würde. Anschließend geht es um die Tageszeit und die Häufigkeit der Suche.

Für die Frage nach dem Ort gibt es nur noch die drei allgemeineren Antwortmöglichkeiten *Arbeit*, *Unterwegs* und *zu Hause*. Dafür wird die Frage zum Nutzungskontext hinzugefügt. Somit ist eine genauere Aufschlüsselung der Verwendung möglich. Die Ergebnisse zur örtlichen Nutzung sind in Abbildung 3.6 aufgetragen. Der Einsatz unterwegs und zu Hause dominiert die Verwendung während der Arbeitszeit. Wie schon bei vorherigen Antworten beschrieben, kann das durch die hohe Zahl an Studenten und somit geringe Anzahl an Arbeitnehmern erklärt werden. Von elf Szenarien werden sechs häufiger unterwegs und fünf häufiger von zu Hause aus genutzt. Betrachtet man die Verteilung der Gerätenutzung (Abbildung 3.7), so bestätigt sich dieses Ergebnis. Am häufigsten wird das Smartphone zur Suche eingesetzt, was auf eine mobile Nutzung, d. h. vor allem unterwegs, schließen lässt. Bei der Aufschlüsselung nach Szenarien kann man gut erkennen, wann eine Suche mit einem Navigationsgerät unternommen wird. Wie zu vermuten war, ist dies für die Szenarien zur Parkplatz- und Routensuche der Fall. Dennoch werden diese Szenarien noch häufiger mit dem Smartphone gesucht. Dies zeigt auch, dass die Grenzen zwischen Geräteklassen verschwinden und der Trend zu Multifunktionsgeräten geht. Schaut man letztlich die Ergebnisse zur Kontextnutzung in Abbildung 3.8 an, so ist erneut die mobile Nutzung auf Reisen und in Teilen der Freizeit sehr dominant. Einige

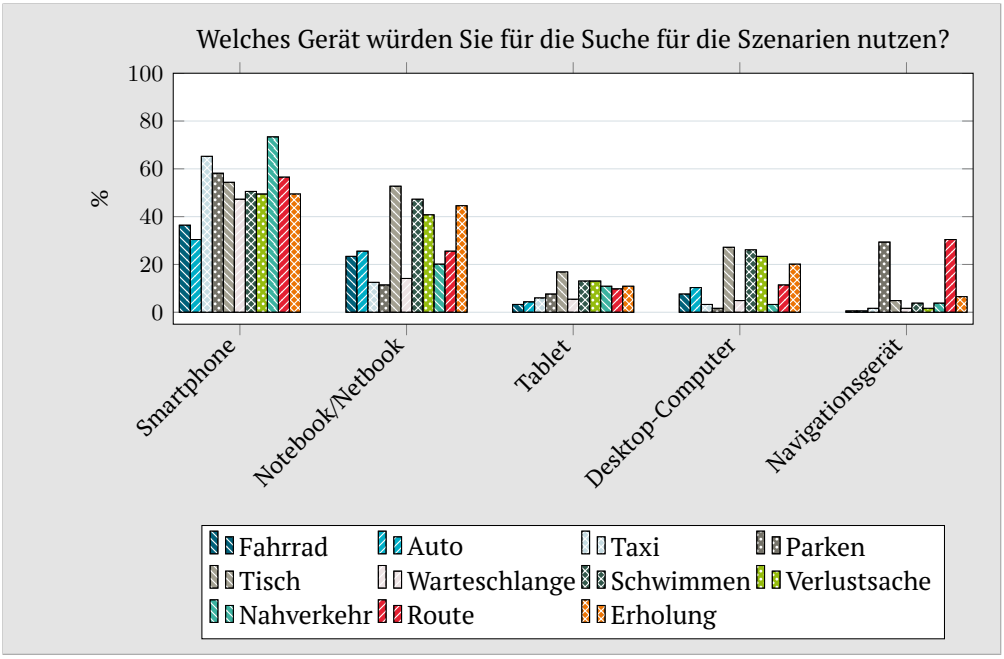


Abbildung 3.7.: Antworten auf die Frage zur Gerätenutzung.

Szenarien, wie die Tischsuche, sind in einigen Kontexten, wie z. B. im Kontext Sport, nicht vorhanden, während andere, wie z. B. *Parken*, überall vorkommen. Über fast alle Szenarien hinweg ist die Nutzung in der Freizeit der häufigste Kontext.

Auch beim Betrachten der Zahlen zum Tageszeitraum (Abbildung 3.9) und der Häufigkeit (Abbildung 3.10) der Nutzung, lassen sich interessante Sachverhalte erkennen. So werden Szenarien, wie *Taxi* und *Tisch*, hauptsächlich in bestimmten Zeiträumen des Tages, wie der Nacht oder dem Abend, genutzt. Es gibt sogar Zeiträume, in denen eine Verwendung sehr gering oder nicht vorhanden ist. Die Nutzung anderer Szenarien, wie z. B. *Verlustsache*, ist hingegen über den Tag sehr gleichmäßig verteilt. Bei der Häufigkeit gibt es einige Szenarien, die nie genutzt werden. Es ist allerdings zu vermuten, dass sich diese Einstellung ändert und die Nutzer eine solche Suche, sobald sie verfügbar ist und der mögliche Nutzen erkannt wird, auch verwenden werden. Viele neue Technologien, wie soziale Netzwerke, Smartphones und sogar der Computer, wurden anfangs belächelt, anschließend aber ein Massenprodukt. Sie sind heute nicht mehr aus der Gesellschaft wegzudenken.

Schlussfolgerung. Ein eindeutiges Nutzungsmuster hinsichtlich Gerät, Kontext, Zeitpunkt und Häufigkeit kristallisiert sich bei keinem Szenario heraus. Wie schon bei den allgemeinen Nutzungsmustern heißt dies, dass ein System auf jegliche Kombination der genannten Faktoren vorbereitet sein muss. Das Design einer Sucharchitektur und der GUI muss somit möglichst flexibel gestaltet sein. Dennoch kann man einige Erkenntnisse zur Optimierung von

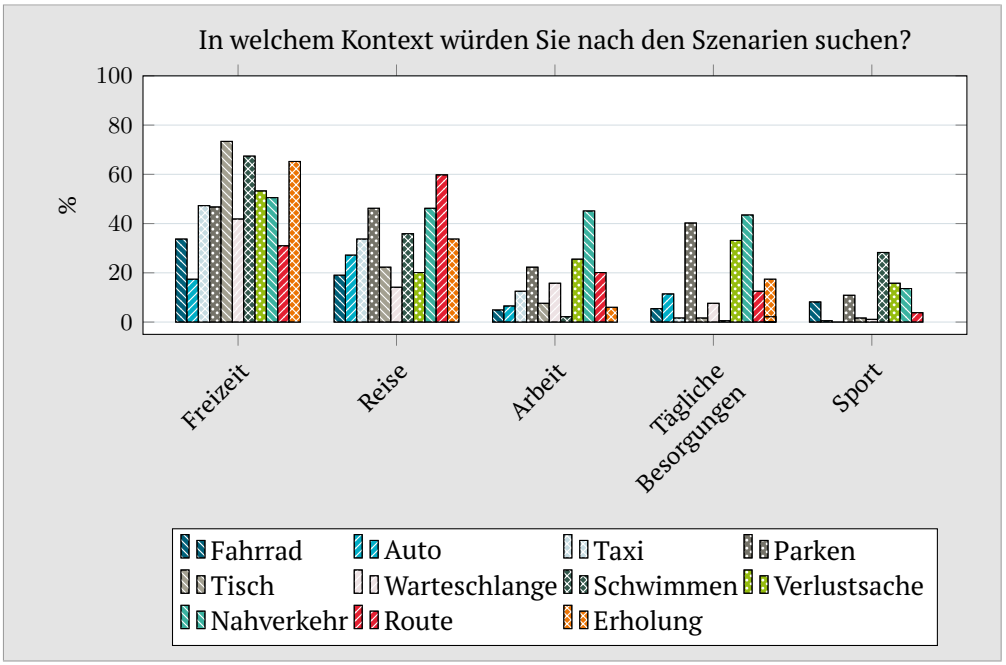


Abbildung 3.8.: Antworten auf die Frage zum Suchkontext.

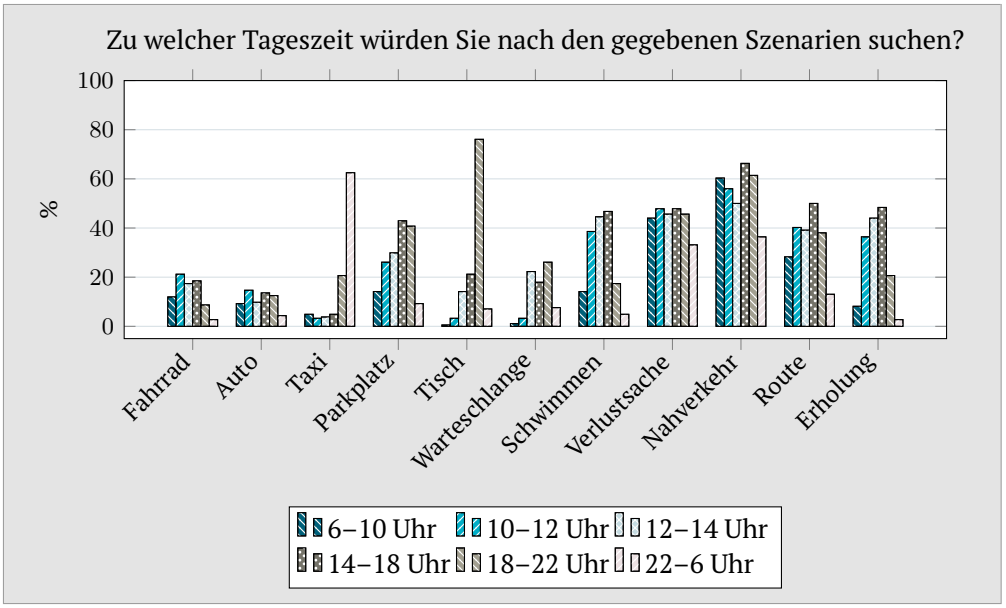


Abbildung 3.9.: Antworten auf die Fragen zum Nutzungszeitpunkt.

Prozessen bei der Suche ausnutzen. So lassen sich Sensoren, sofern sie selten bzw. gar nicht abgefragt werden, häufiger bzw. länger in einen Schlafmodus schicken, um Energie zu sparen. Außerdem kann die Zustandsübertragung von Sensoren adaptiv an das Anfragemuster angepasst werden. Wird der Zustand eines Sensors nur selten bzw. unregelmäßig benötigt, so kann dieser aktiv ange-

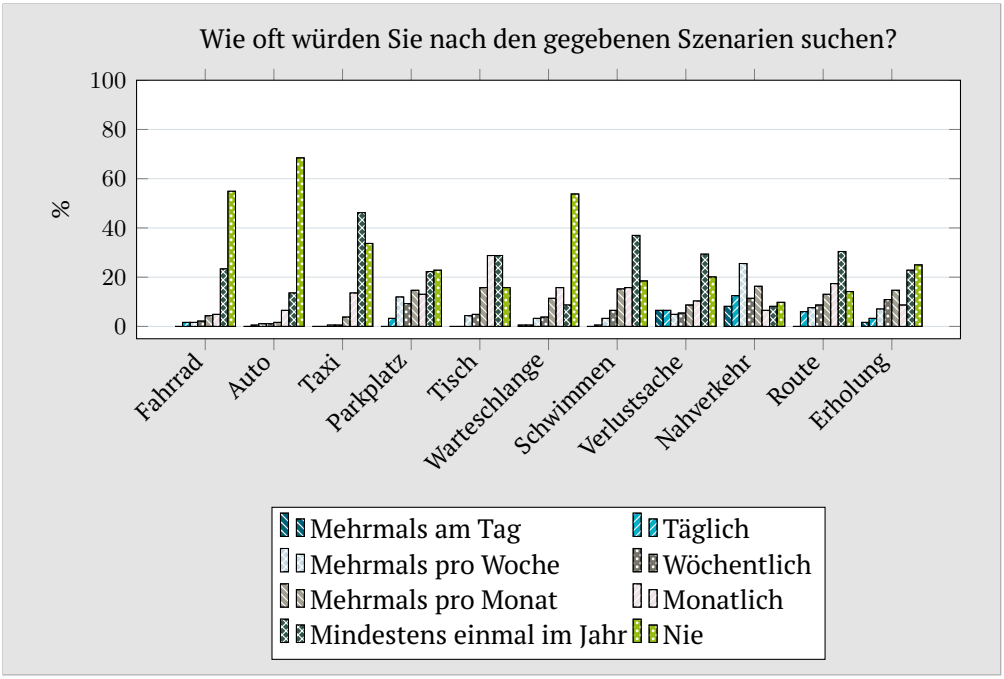


Abbildung 3.10.: Antworten auf die Frage zur Nutzungshäufigkeit.

fragt werden. Wird dagegen sein Zustand häufig gebraucht, so kann der Sensor diesen proaktiv mitteilen. Damit ist im optimalen Fall die Anzahl benötigter Nachrichten und somit die nötige Energie reduzierbar.

Wie möchten Teilnehmer Suchergebnisse angezeigt bekommen? – Präsentation von Suchergebnissen

Der letzte Fragenblock soll Aufschluss darüber geben, wie die gefunden Entitäten dem Nutzer auf der Ergebnisseite zu präsentieren sind. Dazu ist es von Interesse, wie viele Ergebnisse ein Nutzer als Ergebnis haben möchte, mit welcher Genauigkeit Sensorzustände angezeigt, welche Informationen über die Ergebnisse dargestellt und wie diese sortiert werden sollen.

Mehr als $\frac{3}{4}$ (77,14 %) der Umfrageteilnehmer möchte gerne zehn Ergebnisse auf der ersten Seite präsentiert bekommen. Die weiteren Antwortmöglichkeiten von 100, 3 oder allen Ergebnissen erhielten mit 8,57 %, 7,43 % bzw. 6,86 % bedeutend weniger Zuspruch. Dies deckt sich auch mit der von Google standardmäßig angezeigten Anzahl von Treffern auf der ersten Seite. Die Analyse von Logdateien [107] haben ergeben, dass die meisten Nutzer nur die erste Ergebnisseite betrachten. Auch andere Ansätze wie Sensor Ranking machen implizit die Annahme, dass nur wenige Ergebnisse ausreichen, um den Nutzer zufriedenzustellen. Bei Sensor Ranking wird dies ausgenutzt, um nicht alle möglichen Sensoren nach ihrem Zustand anfragen zu müssen. Stattdessen werden nur so viele kontaktiert bis eine bestimmte Menge passender gefunden

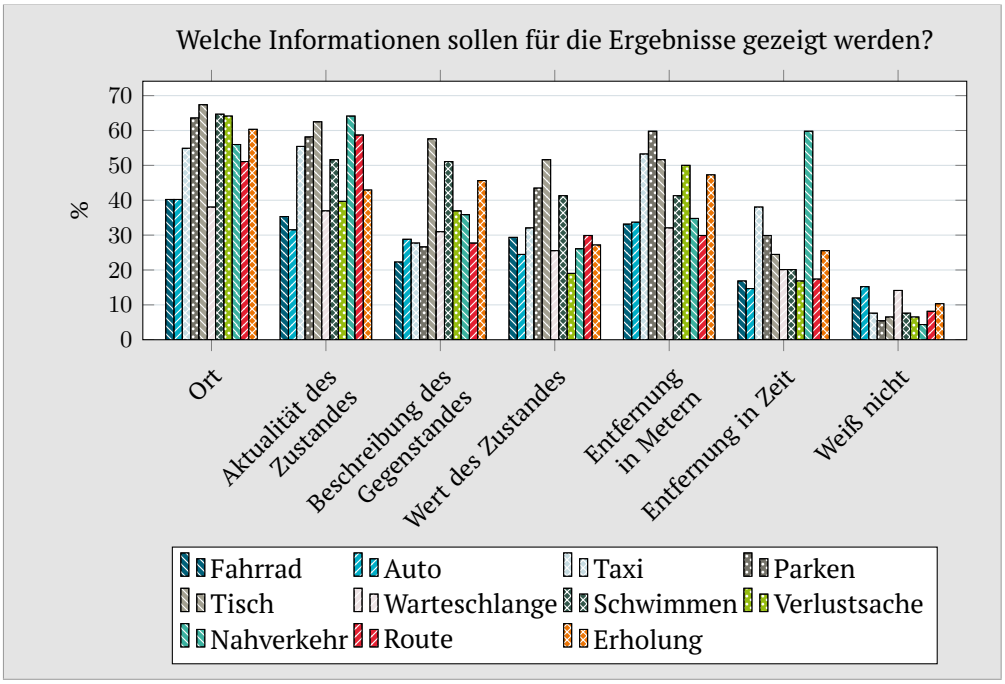


Abbildung 3.11.: Antworten auf die Frage zur Informationsdarstellung bei Ergebnissen.

wird. Was „passende“ Treffer für den Nutzer sind, soll mit den weiteren Fragen herausgefunden werden.

Für den Nutzer muss die Ergebnisseite die richtigen Informationen beinhalten, um feststellen zu können, ob das Ergebnis seinen Suchwunsch erfüllt. Darum wird gefragt, welche Informationen angezeigt werden sollen. Abbildung 3.11 zeigt, dass der Ort der Entität und die Aktualität des Zustandes den Teilnehmern am wichtigsten sind. Dies lässt vermuten, dass die Teilnehmer ein Bewusstsein dafür haben, dass eine Suche für das IoT nur mit aktuellen Daten einen Nutzen hat. Weiterhin ist erkennbar, dass jede der möglichen Informationen unabhängig vom Szenario von mindestens 15 % gefordert wird. Es sollten somit jegliche Informationen angezeigt werden. Dennoch können die Ergebnisse Hinweise geben, welche Information präsenter darzustellen sind als andere. Weiterhin wäre es sinnvoll, den Nutzer zu informieren, wenn ein Sensorzustand sich nur selten ändert. Der Nutzer wüsste somit, dass der Zustand noch „aktuell“ ist und die geringe Aktualität nicht bedeutet, dass der Sensor fehlerhaft ist.

Wie oft sich ein Sensorzustand ändert, hängt auch von der Anzahl und der Granularität der möglichen Zustände ab. In Tabelle 3.2 in Abschnitt 3.2.2 wurden die vier Kategorien Wartezeit, Verfügbarkeit, Ort und Zustand vorgestellt. Exemplarisch für diese Kategorien wird gefragt, welche Genauigkeit ein Zustand haben soll. Abbildung 3.12 zeigt, dass die gewünschte Genauigkeit sehr unterschiedlich ist. So soll diese für die Verfügbarkeit beim *Tisch*-Szenario möglichst hoch sein. Beim Ort einer gesuchten Entität soll neben einer Karte dagegen nicht die präziseste Einheit in Form von Koordinaten, sondern eine

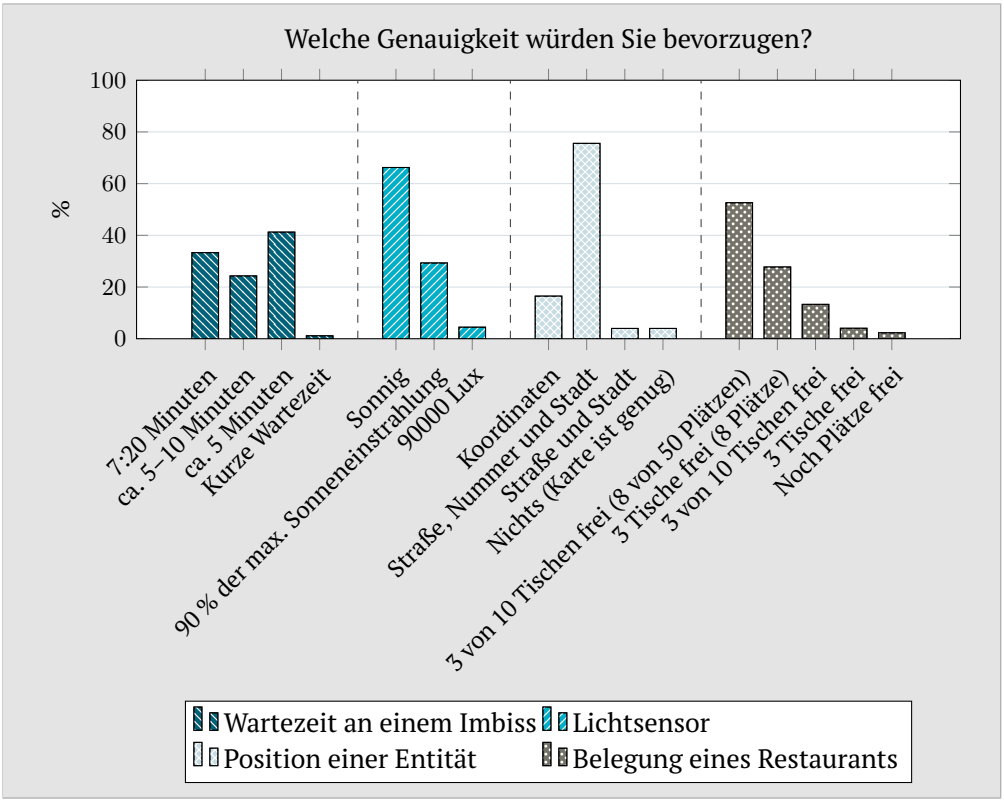


Abbildung 3.12.: Antworten auf die Frage zur Zustandsgenauigkeit.

etwas ungenauere, dafür besser verständliche Kodierung als Adresse ausgegeben werden. Für die Genauigkeit eines Lichtsensors wünschen sich die Nutzer sogar die ungenauste Angabe. Statt der genauen Lux-Zahl oder dem Anteil der Sonneneinstrahlung, soll ein abstrakter Zustand wie *sonnig* genutzt werden. Solch eine abstrakte Beschreibung ist bei der Wartezeit nicht erwünscht. Allerdings gibt es dort auch keine Genauigkeit, die wesentlich mehr Stimmen als die anderen Antworten auf sich vereinen kann.

Die abschließende Frage behandelt das Sortierkriterium. Das Diagramm in Abbildung 3.13 zeigt die Ergebnisse der Frage. Die Distanz als Kriterium, sei es in Metern oder in Zeit, zusammen mit der Aktualität des Zustandes liegen weit vor den anderen Antwortmöglichkeiten. Für einzelne Szenarien erreichen auch Bewertungen als Sortiermerkmal höhere Prozentzahlen. Der hohe Wert des Szenarios *Nahverkehr* bei der zeitlichen Entfernung zeigt ein weiteres Mal deutlich die hohe Anzahl an studentischen Teilnehmern, die generell sehr unzufrieden mit der Pünktlichkeit des Lübecker Nahverkehrs sind.

Schlussfolgerung. Es sollte immer nur eine gewisse Menge an Resultaten angezeigt und auf Nutzerwunsch weitere dynamisch nachgeladen werden. Dieses inkrementelle Abfragen weiterer passender Entitäten kann durch die Nutzung von Mechanismen wie Sensor Ranking Ressourcen bei der Abfrage von Sensoren

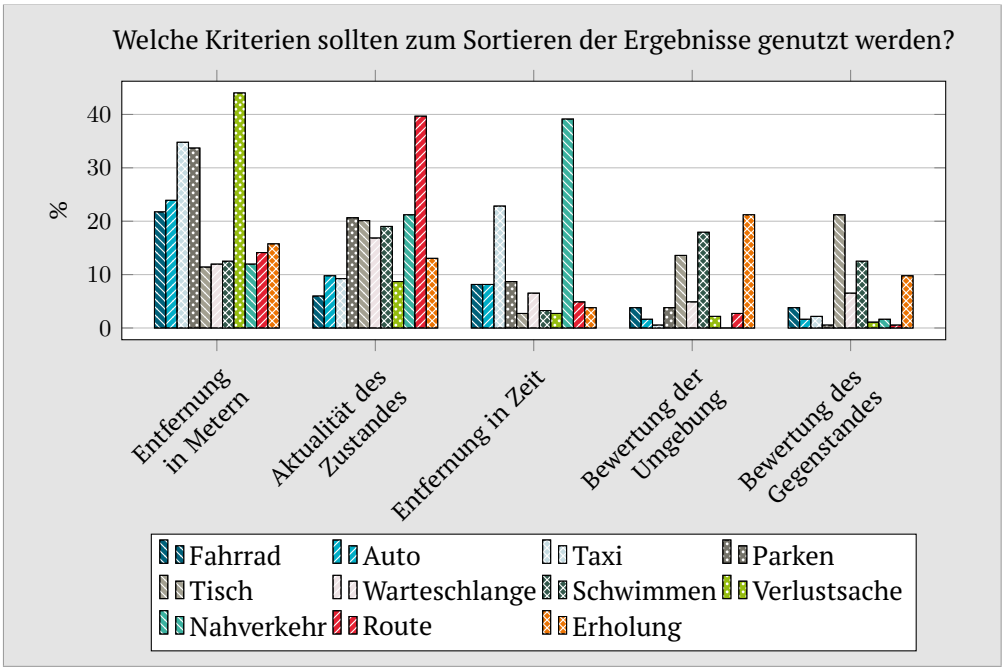


Abbildung 3.13.: Antworten auf die Frage zur Sortierung der Ergebnisse.

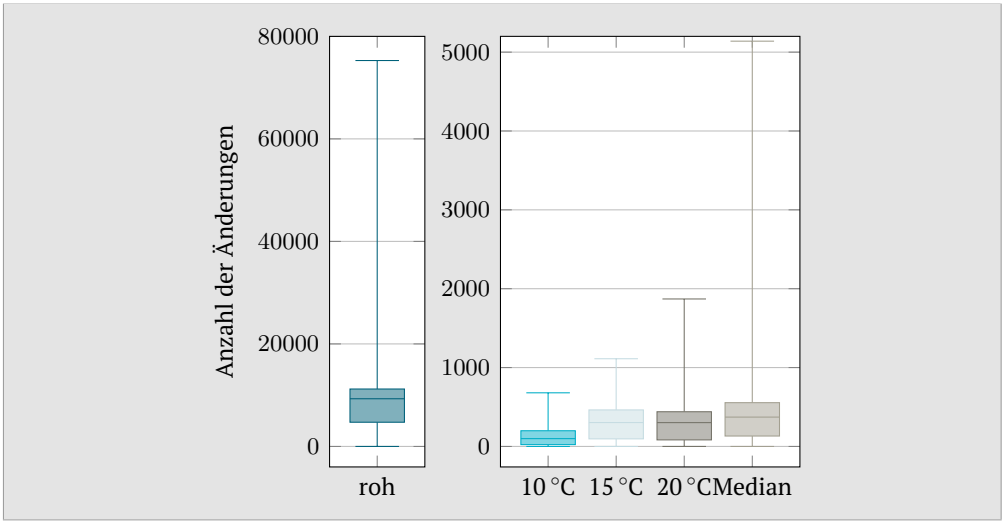


Abbildung 3.14.: Boxplots der Anzahl von Änderungen der Sensorrohwerte ( ) bzw. Anzahl von Zustandsänderungen mit zwei Zuständen. Die Zustandsgrenzen liegen bei 10 °C ( ), 15 °C ( ), 20 °C ( ) und der Mediantemperatur ( ).

sparen. Wie schon bei den anderen Frageblöcken, sind auch in diesem Block die Ergebnisse sehr szenarioabhängig. Somit benötigt auch die Ergebnisdarstellung eine gewisse Flexibilität. Eine Möglichkeit ist, dem Nutzer zu erlauben, die Sortierkriterien selbst priorisieren zu können, so dass die Liste nach seinen Bedürfnissen sortiert wird. Dennoch sollte für Nutzer, die sich nicht diesen

Aufwand machen wollen, eine Standardsortierung eingestellt sein. Die Genauigkeit von Zuständen der Entitäten muss ebenfalls flexibel sein. Diese Zustände werden aus rohen Sensorwerten ermittelt. Im Allgemeinen ändern sich die höherwertigen Zustände, wie *sonnig*, seltener als die zugrundeliegende Rohwerte. Dadurch lassen sich ggf. Ressourcen sparen, da nicht jede Sensorwertänderung übertragen werden muss, sondern nur Zustandsänderungen. Dies ist in Abbildung 3.14 anhand eines Datensatzes von ca. 8,5 Millionen Werte von über 700 Temperatursensoren des Smart-Santander-Projektes[3] zu erkennen. Die Werte sind über einen Zeitraum von einem Jahr gemessen worden. Dargestellt sind die Anzahl der Änderungen aufeinanderfolgender Sensorrohwerte bzw. die Änderungen von Zuständen. Bei den Boxplots der Zustandsänderungen werden dafür jeweils die Zustände *kalt* und *warm* definiert, wobei der Zustandsübergang bei 10 °C, 15 °C, 20 °C und jeweils dem Median aller Werte eines Sensors gesetzt wird. Während für die Rohwerte der Median der Änderungen bei fast 10.000 liegt und sich im Schnitt 72 % der aufeinanderfolgender Werte unterscheiden, ist er bei den Zuständen um den Faktor 25–100 kleiner (man beachte die unterschiedlichen y-Achsen-Skalierung).

3.2.3. Fazit

Die Ergebnisse der Umfrage zeigen die Wünsche der Nutzer bzgl. einer Suchmaschine für das IoT und wie sie diese Suchmaschine nutzen würden. Diese Erkenntnisse können von Entwicklern genutzt werden, um die Technologien für eine solche Suchmaschine auf die Bedürfnisse des Nutzers hin zu entwickeln. Außerdem lässt sich das Wissen über das Nutzerverhalten verwenden, um ressourcenschonende Verfahren zur Abfrage von Sensoren zu entwickeln. Ein wichtiger Punkt, der bei fast allen Fragen beobachtet werden kann, ist, dass es meistens eine Tendenz hin zu einer bestimmten Antwort gibt, jedoch ebenso die anderen Antwortmöglichkeiten von einer Teilmenge der Teilnehmer präferiert wird. Dies zeigt, dass die zu entwickelnden Technologien so flexibel sein müssen, dass sie die verschiedenen Wünsche und Anforderungen erfüllen können.

Die Ergebnisse dienen im Folgenden als Ausgangspunkt für die Entwicklung der Technologien in dieser Arbeit. Laut Umfrage würden Nutzer verschiedene, heterogene Geräte, wie Smartphones, Navigationsgeräte und Laptops, zur Suche verwenden. Auf die Nutzbarkeit der Suche mit verschiedenen Geräteklassen geht Abschnitt 6.3.2 ein. Damit Entwickler die Suche für weitere Geräteklassen entwickeln können, z. B. als eigenständige App für Smartphones, wird zusätzlich eine API angeboten, welche in Abschnitt 6.3.1 beschrieben wird. Die vornehmlich mobile Nutzung, u. a. in Kontexten wie auf Reisen, als auch die Tatsache, dass nach realen Dingen mit einer Position gesucht wird, bedingt den Einsatz von Geoinformationsverfahren. Dazu wird bei der Modellierung von geografischen Daten in Kapitel 5 GeoSPARQL eingesetzt. Um diese Daten speichern und abfragen zu können, wird der in Abschnitt 6.2 als Grundlage

[3] EU FP7 Projekt SmartSantander: `http://smartsantander.eu/` [Zugriff am 21. März 2014]

genutzte Triplestore LUPOSDATE um GeoSPARQL-spezifische Datentypen und Funktionen erweitert. Die eigene Position würde von den meisten Nutzern zur Verbesserung der Suchergebnisse zur Verfügung gestellt werden. Deshalb kann der Nutzer beim Aufrufen der GUI zulassen, dass seine Position ermittelt wird. Dies lässt sich anschließend nutzen, um Ergebnisse nach Distanz zu sortieren und zu filtern. Zusätzlich können Routen zu den Ergebnissen abgerufen werden, was gerade im Kontext von Reisen eine wichtige Funktion für Nutzer ist. Da die Suche oft genutzt würde, wenn der Nutzer nur wenig Zeit hat oder kurz abgelenkt sein darf und auch weil viele Szenarien wiederholt genutzt würden, ist eine schnelle Suchmöglichkeit wünschenswert. Dies wird in der GUI durch das Abspeichern und spätere Wiederausführen von Sucheinstellungen umgesetzt. Für die Ergebnisse hat sich gezeigt, dass Nutzer volle Kontrolle bei der Sortierung und möglichst umfangreiche Informationen wünschen. Deswegen wird für das Ranking der Ergebnisse eine offene und erweiterbare Modellierung (siehe Abschnitt 5.2.10) und für den Nutzer eine frei konfigurierbare Sortierung und Filterung (siehe Abschnitt 6.3.2) verwirklicht. Zudem werden möglichst viele statische und dynamische Informationen dem Nutzer in der Ergebnisansicht präsentiert. Im nächsten Kapitel werden allerdings zunächst Verfahren vorgestellt, um Zustände aus den Sensorwerten zu ermitteln. Dabei liegt ein besonderes Augenmerk auf der Flexibilität der Verfahren, da sich die gewünschte Zustandsgenauigkeit laut Umfrage je nach Szenario stark unterscheidet.

4 Zustandsermittlung und -wahrscheinlichkeiten

> Information is not knowledge.
>
> — *Albert Einstein*

Wie in Abschnitt 3.2.2 dargelegt, möchten Nutzer oftmals nicht rohe Sensorwerte, sondern höherwertige, verständliche Zustände präsentiert bekommen. Es hat sich aber herausgestellt, dass die Granularität bzw. Genauigkeit stark vom Szenario bzw. vom verwendeten Sensortyp abhängt. Es ist somit nicht ratsam eine fixe Anzahl von Zuständen für einen Sensortyp zu definieren und verwenden. Vielmehr ist es notwendig Mechanismen bereitzustellen, um aus Sensorwerten Zustände zu ermitteln. Diese Verfahren sollten auf der einen Seite flexibel sein, d. h. z. B. für typische, von Sensoren generierte, Datentypen einsetzbar sein. Auf der anderen Seite ist es hilfreich, wenn sie leicht erlern- und benutzbar sind, damit sie auch von Laien verwendet werden können.

Die Umfrageergebnisse in Abschnitt 3.2.2 zeigen weiterhin, dass die Aktualität der Zustände ein wichtiges Kriterium bei der Suche im IoT ist. Der naive Ansatz, um immer eine möglichst hohe Aktualität zu gewährleisten, ist, den Zustand in kurzen Abständen von den Sensoren abzufragen. Dies steht aber im Widerspruch zu den begrenzten Energie- und Kommunikationsressourcen von Sensorknoten. Zudem ist das häufige, periodische Abfragen ggf. unnötig, da sich der Zustand nicht so häufig ändert wie er abgefragt wird. Eine andere Möglichkeit ist es deshalb, dass der Sensorknoten Zustandsänderungen proak-

tiv an das Backend verschickt. Wird der Zustand eines Sensors allerdings nur selten benötigt, d. h. durch einen Nutzer angefordert, werden weiterhin unnötig Ressourcen verbraucht. Eine Möglichkeit, die Nachteile dieser beiden Varianten zu beseitigen, ist die Verwendung von Wahrscheinlichkeits- bzw. Vorhersagemodelle, um den derzeitigen Zustand eines Sensors abzuschätzen. Dies zeigt u. a. die Arbeit von Elahi et al. [18].

Eine Herausforderung bei der Ermittlung von Zuständen und der Erzeugung von Modellen ist die zu erwartende Heterogenität von Geräten im IoT. Deshalb ist es schwierig, einen Ansatz für alle Gerätetypen zu entwickeln. Nach einer Diskussion verwandter Arbeiten, wird in Abschnitt 4.2 erläutert, wie sich im Allgemeinen aus rohen Sensorwerten verschiedener Sensortypen höherwertige Zustände ermitteln lassen. Anschließend werden vier Ansätze für verschiedene Geräteklassen vorgestellt. Abschnitt 4.2.1 behandelt das Ermitteln von Zuständen aus Bildern, die von einem kamerabestückten Sensorknoten aufgenommen werden. Eine Option für CoAP, die aus Sensorwerten von CoAP-Ressourcen Zustände ableitet, wird in Abschnitt 4.2.2 beschrieben. Eine weitere Lösung für Sensorknoten, die mit proprietären Protokollen betrieben werden, folgt in Abschnitt 4.2.3. Abschließend zeigt eine browserbasierte Lösung in Form einer Chrome-Erweiterung in Abschnitt 4.2.4, wie sich virtuelle Sensoren und Sensorzustände aus Webseiten ermitteln lassen. Die vier präsentierten Lösungen werden in Abschnitt 4.2.5 einander in Bezug gesetzt und verglichen. In Abschnitt 4.3.1 wird eine formale Definition für Wahrscheinlichkeitsmodelle eingeführt und verschiedene Modelltypen vorgestellt. Anschließend wird in Abschnitt 4.3.2 beispielhaft ein Verfahren zur Erstellung korrelationsbasierter Modelle auf Basis von Bayes'schen Netzen umgesetzt und evaluiert. In den beiden abschließenden Abschnitten 4.3.3 und 4.3.4 werden zwei Technologien präsentiert, die es ermöglichen, auf Sensorknoten mittels CoAP und der zuvor vorgestellten Skriptsprache verschiedene Modelltypen zu erzeugen.

4.1. Verwandte Arbeiten

In den meisten Systemen werden Sensorrohwerte weiterverarbeitet. Sei es zur Vorhersage des Wetters, von Staus oder zur Erzeugung von Bewegungsprofilen von Tieren oder Menschen. Auch dies sind Formen höherwertiger Zustände. Das Hauptaugenmerk der Zustandsermittlung in diesem Kapitel liegt allerdings in der Anreicherung der rohen Sensorwerte mit einer Semantik, also einer für den Nutzer verständlichen, intuitiven Bedeutung. Deshalb werden Systeme vorgestellt, die genau diese Art der Zustandsermittlung realisieren oder zu mindestens einem Nutzer ermöglichen.

Diverse Systeme [108–113] bestimmen, ob Parkplätze frei oder belegt sind. In den Arbeiten [108–112] werden Fotoaufnahmen von Parkplätzen ausgewertet, um deren Zustand zu ermitteln. ParkNet [113] dagegen nutzt Autos, die mit GPS und Ultraschallsensoren ausgestattet sind. Diese scannen den Straßenrand beim Fahren nach freien Plätzen ab.

Auch die Arbeiten [114, 115] nutzen Kamerabilder, um die Umgebung zu überwachen. In beiden werden Sensorknoten mit Bewegungssensoren und Kameras verwendet, um bei der Detektion von Bewegungen Fotos von Tigern im Dschungel [114] bzw. Käfigmäusen [115] aufzunehmen. Diese Fotos werden zu einem zentralen Server weitergeleitet und dort gespeichert. Wissenschaftler können die Fotos auswerten, um das Verhalten der Tiere zu analysieren. Eine automatische Auswertung erfolgt jedoch nicht.

Bolliger et al. [116] präsentieren *Facet*, ein Framework, das die Kameras von Handys nutzt, um die Umgebung zu überwachen. Ein Handy kann dabei die Events *in* und *out* generieren, die Auftreten, sobald ein bewegtes Objekt die Szene betritt bzw. verlässt. Per Bluetooth lassen sich mehrere Handys zu einem Netz zusammenschließen. In einem Experiment wird gezeigt, wie die Bewegung von Personen durch einen Büroflur verfolgt werden kann.

Das *SoundOfTheCity*-Projekt [117] stellt eine offene API und eine Android-Anwendung bereit, die es erlauben Audioaufnahmen und positionsbasierte Lautstärkemessungen durchzuführen. Die Messungen werden zentral auf einem Server gespeichert und auf der Projektwebseite[1] auf einer Karte angezeigt. Die Lautstärke wird dabei je nach Intensität und Zoomstufe der Karte in verschieden großen, farbigen Kacheln visualisiert. Somit können sich Nutzer einen Lautstärkezustand eines Ortes anzeigen lassen. Entwicklern steht eine API zur Verfügung, um die Lautstärkezustände abzurufen.

SurroundSense [118] nutzt neben dem Mikrofon auch die Kamera für Licht- und Farbmessungen, den Beschleunigungssensor und das W-LAN-Modul von Handys. Aus den Informationen wird ein digitaler Fingerabdruck generiert, der mit einer Datenbank abgeglichen wird, um die Position des Handys zu bestimmen. Dabei geht es allerdings nicht um die physikalische Position, d. h. eine GPS-Koordinate, sondern um die logische Position, z. B. in welchem Laden sich das Handy befindet.

TaDaweb[2] erlaubt per Webinterface oder eigenständiger Applikation sogenannte *Rezepte* zu erstellen. Ein *Rezept* beschreibt eine Reihenfolge von Operationen, die auf eine Webseite angewendet werden sollen. Der Nutzer bestimmt dazu die Webseite, die Elemente, wie Listen, Text oder Bilder, die daraus extrahiert und wie diese weiterzuverarbeiten sind. Die resultierenden Daten werden auf der Webseite von TaDaweb angezeigt. Eine API, um die Daten abzurufen und maschinell weiterzuverarbeiten, steht nur gegen Gebühr zur Verfügung. Das Ableiten von Zuständen ist zwar in begrenztem Maße möglich, aber zugleich schwierig, da dazu viele komplizierte Funktionen zu kombinieren sind.

Menschliche Emotionen, die aus der Körpersprache und dem Verhalten ableitbar sind, können als höherwertige Zustände angesehen werden. Verschiedene Systeme beschäftigen sich mit der Erkennung von Emotionen [119–126]. Die häufigste Methode ist das Erkennen in Gesichtern mittels Bildersequenzen [119–121]. Die Systeme basieren dabei im Allgemeinen auf dem *Facial Action*

[1] `http://citysound.itm.uni-luebeck.de/` [Zugriff am 21. März 2014]
[2] `http://www.tadaweb.com/` [Zugriff am 21. März 2014]

Coding System [127], welches 44 Bewegungseinheiten der Gesichtsmuskulatur und fünf Stufen der Ausprägungsstärke definiert. Emotionen, wie Ärger, Freude, Wut, Angst und Ekel, sind als Kombination verschiedener Bewegungseinheiten definiert. Die Arbeit von Picard et al. [122] versucht die Emotionen mit vier Sensoren zur Messung der Gesichtsmuskelanspannung, des Blutdrucks, des elektrischen Leitungswiderstandes der Haut und der Atmung zu bestimmen. Weitere Systeme versuchen die Emotionserkennung aufgrund der Stimme [123, 124], genauer der Tonlage, Geschwindigkeit, Intonation und Intensität oder als Kombination verschiedener Modalitäten [125, 126] durchzuführen. Übersichten zahlreicher Systeme zur Erkennung von Emotionen finden sich in [128, 129].

Ebenfalls ein menschlicher höherwertiger Zustand ist die ausgeführte Aktivität von Personen. Eine Vielzahl von Systemen versucht die Aktivität von einzelnen Personen zu erkennen [130–138]. Dabei geht es meistens um einfache Aktivitäten wie Gehen, Laufen und Sitzen. Der Großteil der Systeme basiert auf in der Umgebung installierten Kameras [130, 131] oder Sensoren, die von der zu überwachenden Person am Körper zu tragen sind. Die sensorbasierten Systeme nutzen entweder dedizierte [132] bzw. in Mobiltelefone integrierte [133, 134] Beschleunigungssensoren oder eine Kombination verschiedener Sensortypen, ebenfalls dediziert [135–137, 139] oder mittels Mobiltelefon [138]. Übersichten von Arbeiten zu diesem Thema sind in [140] und [141] zu finden. Der Fokus von [140] liegt dabei auf kamerabasierten Systemen, während [141] sensorbasierte Ansätze behandelt.

Ein noch breiteres Spektrum bildet die Erkennung des Kontextes einer Situation. Kontext ist ein vielschichtiger Begriff, der in der Wissenschaft unterschiedliche Ausprägungen annimmt. In einigen Arbeiten wird schon die Position als Kontext bezeichnet, während in anderen Kontext aus verschiedenen Faktoren zusammengesetzt ist. *Ambient Dynamix* [142] ist ein Framework zur Realisierung von Aktivitäts- und Kontexterkennung. Es ist als Hintergrunddienst für Android realisiert, der zwischen der Hardware und beliebig vielen Dynamix-Anwendungen angesiedelt ist. Über Plugins, die zur Laufzeit nachladbar sind, erhält das Framework beliebige Kontextinformationen. Diese Informationen können von Dynamix-Anwendungen aktiv angefordert werden bzw. lassen sich ereignisgesteuert an diese verteilen, sofern die Anwendung sich vorher für die Kontexttypen registriert hat. Die Plugins können dabei den Kontext aus Informationen beliebiger Quellen wie z. B. Sensoren, dem Dateisystem oder externen APIs ermitteln.

Es existiert eine Vielzahl weiterer Systeme und Frameworks zur Kontexterkennung [143–146]. Der Aufbau ist dabei immer ähnlich und lässt sich im Allgemeinen in vier Schichten einteilen. Auf der untersten werden Kontextinformationen aus verschiedenen Quellen wie Sensoren gesammelt. Die gesammelten Daten werden anschließend auf der 2. Schicht in eine Modellierung wie z. B. eine semantische Beschreibungssprache überführt. Die 3. Schicht verarbeitet die Daten, um höherwertige Kontextinformationen herzuleiten. Dazu gehören Operationen wie Filtern, Aggregieren und Schlussfolgern (z. B. durch reasoning

oder maschinelles Lernen). Auf der obersten Schicht werden die gewonnenen Informationen weiterverteilt, z. B. an Anwendungen. Eine gute Zusammenfassung zur Kontexterfassung mit einer Analyse von 50 Systemen findet sich in [147].

Die meisten der vorgestellten Systeme haben den Nachteil, dass sie unflexibel sind, da sie für bestimmte Plattformen, Szenarien oder Sensortypen entwickelt wurden. Es ist allerdings wünschenswert ein System zu haben, das auf möglichst vielen verschiedenen Plattformen für unterschiedliche Szenarien und diverse Sensortypen einsatzfähig ist. Vor allem die Systeme zur Kontext- und Aktivitätserkennung bieten diese Flexibilität durch einen großen Funktionsumfang. Allerdings sind sie entsprechend komplex. Außerdem werden für die Erkennung meist Daten verschiedener Sensortypen herangezogen. Dies und die häufige Nutzung von Sensoren mit hohem Datenvolumen, wie Kameras, bedeutet zudem ein hohes Datenaufkommen, welches im Kontext ressourcenbeschränkter Systeme ein Nachteil ist.

Im Bezug auf die Erzeugung von Vorhersagemodellen für Sensorwerte, um Anfragen nach Sensorwerten effizient zu beantworten, gibt es eine Vielzahl von Arbeiten [13, 148–153]. Bei diesen Arbeiten geht es darum vorauszusehen, welcher Sensorwert als nächstes gemessen wird. Ziel dieser Arbeit ist jedoch zu ermitteln, welcher Zustand zum derzeitigen Zeitpunkt mit hoher Wahrscheinlichkeit auftritt. Außerdem wird nach den Werten eines Sensors gefragt, statt nach den Sensoren, die zur Anfragezeit einen bestimmten Wert messen. Die zugrundeliegenden Ideen, wie ein periodisches Verhalten des beobachteten Phänomens zu nutzen, sind jedoch identisch. Die genutzten Modelle, die entweder auf jedem Knoten, an ausgewählten Cluster-Knoten oder auf einem zentralen Server berechnet werden, basieren auf mathematischen Modellen für kontinuierliche Daten und nicht auf diskreten Zuständen wie es für diese Arbeit benötigt wird. Dabei werden korrelations- und zeitreihenbasierte Modelle verwendet. Ortsbasierte Modelle werden ebenso wenig behandelt wie die Erweiterbarkeit von Modellen oder das Kombinieren verschiedener Modelle. Sensor Ranking [18], welches Wahrscheinlichkeitsmodelle zur ressourcensparenden Beantwortung von Anfragen nutzt, wurde bereits in Abschnitt 1.2.1 im Zusammenhang mit Dyser erläutert.

4.2. Verfahren zur Zustandsermittlung

Das IoT wird viele heterogene Geräte wie Sensorknoten, Smartphones und Haushaltsgeräte umfassen. Diese werden eine Vielzahl verschiedener Sensortypen besitzen. Eine Auswahl von heutzutage häufig eingesetzten Sensortypen ist in Tabelle 4.1 aufgeführt. Für jeden Sensortyp ist der von ihm typischerweise ausgegeben Datentyp eingetragen. Einige Sensortypen haben auch mehrere übliche Ausgabeformate. Ein Mikrofon kann z. B. die Lautstärke in Dezibel oder den gesamten Audiostream ausgeben. Ein weiteres Beispiel sind Beschleunigungssensoren, welche die Beschleunigung als skalare Einheit messen. Übli-

	Belegung	**Bewegung**	**Feuchte**	**Licht**	**Luftdruck**	**Neigung**	**Niederschlag**	**Temperatur**	**Aktivität**	**Windrichtung**	**Beschleunigung**	**Position**	**Mikrofon**	**Fotokamera**	**Abschnitt**
Boolsch	✓	✓													4.2.2–4.2.4
Skalar	(✓)	(✓)	✓	✓	✓	✓	✓	✓			✓		✓		4.2.2–4.2.4
String	(✓)	(✓)							✓	✓					4.2.2–4.2.4
Vektor											✓	✓			
1D-Signal													✓		
2D-Signal														✓	4.2.1

Tabelle 4.1.: Sensortypen und von diesen produzierte Ausgabetypen.

cherweise messen diese Sensoren aber mehrere Achsen gleichzeitig, wodurch ein Vektor entsteht.

Der am häufigsten auftretende Ausgabetyp ist der Skalar. Die anderen Typen sind weniger häufig vertreten. Bedenkt man zusätzlich, dass boolesche Typen üblicherweise als 0 und 1 bzw. *true* und *false* dargestellt werden, ist dieser Ausgabetyp alternativ auch als Skalar bzw. String betrachtbar. Vektoren können theoretisch auch als Skalare angesehen werden, wenn man die Komponenten einzeln betrachtet. So könnte bei einem Beschleunigungssensor z. B. nur die zur Erdoberfläche orthogonale Achse interessant sein, um ein Fallen zu erkennen. Allerdings gibt es auch viele Szenarien, bei denen die verschiedenen Vektorelemente gemeinsam untersucht werden müssen, um einen aussagekräftigen Zustand zu ermitteln.

Um aus den rohen Sensorwerten einen höherwertigen Zustand zu ermitteln, gibt es verschiedene Möglichkeiten. Die einfachste ist die Betrachtung eines einzelnen Sensorwertes und die Abbildung auf einen Zustand. Es sind auch komplexere Verfahren denkbar, bei denen eine Reihe von Sensorwerten einbezogen werden, um daraus einen Zustand zu ermitteln. Dies sind im Allgemeinen betrachtet *wenn*→*dann*-Regeln, wobei die Bedingungen im *wenn*-Teil bei der Berücksichtigung mehrerer Sensorwerte komplexer sind.

In den im Folgenden vorgestellten Technologien werden zwei verschiedene Typen von *wenn*→*dann*-Regeln genutzt. Der erste Typ dient zur Abbildung von skalaren Werten auf einen Zustand, während der andere von Strings auf Zustände abbildet. Eine Betrachtung mehrerer Werte im *wenn*-Teil ist nicht vorgesehen. Dies hat mehrere Gründe. Es müsste eine komplexe Regelsprache definiert werden, die nicht so einfach erlernbar wäre, wie die Regelsprache zur Betrachtung nur eines Wertes. Außerdem müsste die Regelsprache bei Technologien, die auf Sensorknoten laufen, trotz der beschränkten Ressourcen, geparst und interpretiert werden. Zusätzlich müssten entsprechend den Regeln eine Menge von historischen Sensorwerten zwischengespeichert werden, um

jederzeit den Zustand ermitteln zu können. Obwohl das in Abschnitt 4.2.3 vorgestellte skriptbasierte System nicht explizit eine Unterstützung für komplexe Regeln besitzt, ist es mit der Skriptsprache möglich gewisse komplexere Regeln, die mehrere Werte einbeziehen, zu erstellen. Durch zusätzliche Logik an der Basisstation ließe sich so eine noch mächtigere Zustandsgenerierung mit den vorverarbeiteten Daten realisieren.

Der Regeltyp zur Abbildung von skalaren Werten auf einen Zustand, Intervall-Abbildung genannt, hat die allgemeine Syntax $[u{:}o) \rightarrow z$. Der links von $\rightarrow$ stehende Teil beschreibt den *wenn*-Teil. Dabei wird durch u und o ein rechtsoffenes Intervall definiert. Fällt der Skalar des Sensors in dieses Intervall, so wird der Zustand z angenommen. Die Nutzung von rechtsoffenen Intervallen erlaubt die Definition von fortlaufenden, kontinuierlichen, nicht-überschneidenden Bereichen.

Der Regeltyp für Strings, String-Abbildung genannt, folgt der allgemeinen Syntax $s \rightarrow z$. Dabei bezeichnet s den String, der vom Sensor ausgegeben wird und z den Zustand, der für diesen String angenommen wird.

Für ein System lassen sich beliebig viele Regeln definieren. Nacheinander werden die definierten Regeln auf ein Matching evaluiert. Sobald der Sensorwert auf den *wenn*-Teil passt, wird der Zustand angenommen und die Evaluation abgebrochen. Dadurch ist ausgeschlossen, dass durch die mehrfache Verwendung des gleichen Sensorwertes in verschiedenen *wenn*-Teilen Widersprüche entstehen.

Die beiden Regeltypen werden in Abschnitt 4.2.2, 4.2.3 und 4.2.4 für CoAP-basierte Sensorknoten, für Sensorknoten mit einem Skriptinterpreter und für eine Browser-Erweiterung, die aus Textfragmente von Webseiten Zustände ermittelt, genutzt. Bei dem letzten System werden aufgrund besserer Ressourcenverfügbarkeit Reguläre Ausdrücke genutzt, anstatt nur einfache Strings im *wenn*-Teil zu verwenden. Wie oben erwähnt, sind die Regeltypen nicht für Vektoren, 1D- und 2D-Daten anwendbar. Eine Möglichkeit besteht, wie bereits beschrieben, darin, einzelne Werte aus diesen Ausgabetypen zu nehmen und die Regeln damit zu evaluieren. Eine andere Möglichkeit ist, neue Werte oder Zeichenketten aus den Daten zu erzeugen, wie es z. B. bei der Aktivitätserkennung aus rohen Sensordaten geschieht. In Abschnitt 4.2.1 wird ein System vorgestellt, bei dem aus Bildern zwei Kennzahlen extrahiert werden, nämlich die Anzahl bewegter Objekte in einer Szene und die Fläche im Bild, die von diesen eingenommen wird. Diese Werte lassen sich anschließend wie bei den anderen drei Systemen als Eingabe für die Regeln heranziehen.

4.2.1. Lousy Camera Framework: Zustandsermittlung auf kamerabestückten Sensorknoten

„Ein Bild sagt mehr als tausend Worte“ ist ein bekanntes Sprichwort. Das Sprichwort legt nahe, dass sich aus einem Bild entsprechend viele Dinge erfassen und ableiten lassen. Somit ist es mit geeigneten Verfahren auch möglich, den

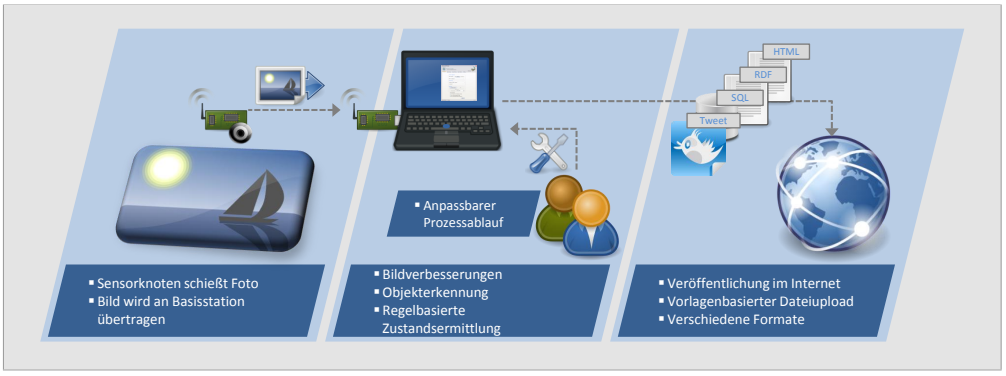


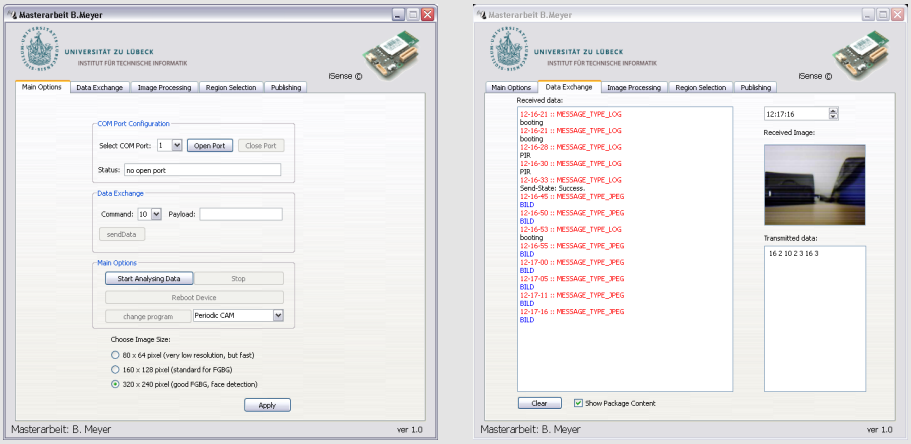
Abbildung 4.1.: LoCaF-Architektur.

Zustand der Umgebung oder von Entitäten, die ein Bild zeigt, zu extrahieren. Die Schwierigkeit liegt in der Wahl der richtigen Verfahren, der Parameter und der Weiterverarbeitung dafür.

Die Bilder, die genutzt werden, um Zustände zu ermitteln, sollten möglichst aktuell sein. So hat der Zustand noch einen Nutzen und ist nicht schon ungültig, also veraltet. Deshalb bieten sich z. B. Webcams an, die regelmäßig Bilder einer bestimmten Szene aufnehmen. Eine andere Quelle können Sensorknoten mit einer angeschlossenen Kamera sein. Mit dem Lousy Camera Framework (LoCaF) wird ein System entwickelt, das genau dies ermöglicht. Es verarbeitet Bilder eines kamerabestückten Sensorknotens und ermittelt Zustände der Szene, die das Bild darstellt. Ziel des Frameworks ist es, bei möglichst einfacher Bedienung eine große Flexibilität zu gewährleisten, damit es in verschiedenen Situationen einsetzbar ist. Im Folgenden werden die Gesamtarchitektur und die einzelnen Bestandteile des Systems vorgestellt. Anschließend wird das System anhand verschiedener Szenarien evaluiert und somit Möglichkeiten und Grenzen der automatischen Auswertung gezeigt. Die wesentlichen Inhalte wurden vorab in [24] veröffentlicht.

Architektur

Die Architektur des Systems ist in Abbildung 4.1 visualisiert. Sie ist in drei Bereiche unterteilt. Den Anfang bildet ein iSense-Sensorknoten mit einer Kamera. Die verwendete Kamera kann Farbfotos bis zu einer Auflösung von 640×480 Pixel im Joint-Picture-Expert-Group-Format (JPEG-Format) aufnehmen. Ein aufgenommenes Bild wird an einen weiteren Sensorknoten, der an eine Basisstation, z. B. einen Laptop, angeschlossen ist, gesendet. Die Basisstation verarbeitet das Bild mit Hilfe von Bildverbesserungs- und Objekterkennungsalgorithmen und ermittelt dann mittels einer Regelsprache daraus einen Zustand. Dieser Prozess ist durch den Nutzer an vielen Stellen beeinflussbar, so dass das Framework an verschiedene Szenarien und Gegebenheiten angepasst werden kann. Der letzte Schritt ist die Veröffentlichung des ermittelten Zustandes



(a) Verbindungsherstellung. **(b)** Datenaustausch.

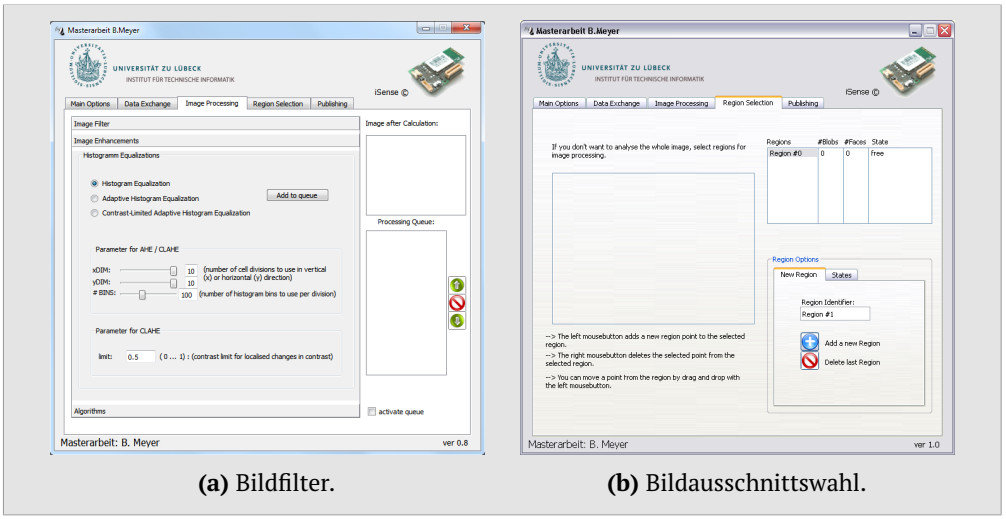
Abbildung 4.2.: Dialoge zur Verwaltung der Verbindung zum Sensorknoten.

im Internet, wodurch der Zustand der beobachten Szene weltweit in Echtzeit zugänglich ist.

Bildaufnahme Die Kamera erlaubt über zehn Bilder pro Minute aufzunehmen, aber die dabei entstehende Datenmenge kann selbst bei störungsfreiem Funkkanal nicht ausreichend schnell übertragen werden. Dies ist aber auch nicht erforderlich bzw. gewollt. Zum einen ist davon auszugehen, dass sich in den meisten Szenarien der Zustand nicht so häufig ändert, zum anderen würde die Energie des Sensorknotens bei Dauerbetrieb des Funkmoduls sehr schnell aufgebraucht sein und der Sensorknoten somit unbrauchbar.

Es sind zwei verschiedene Modi für den Sensorknoten implementiert. Im periodischen Modus schickt der Sensorknoten in vom Nutzer vorgegebenen Zeitabständen ein neues Bild. Dies ist nützlich, sofern man immer eine gewisse Aktualität des Zustandes garantieren möchte. Zudem ist so ersichtlich, ob der Sensorknoten noch aktiv ist, da das Bild auch als Alive-Nachricht interpretiert werden kann. Der zweite Modus schickt dagegen nur ein Bild, sobald ein am Sensorknoten angeschlossener PIR-Sensor etwas detektiert. Ein PIR-Sensor schlägt aus, sobald er eine Temperaturänderung der Wärmestrahlung von z. B. Personen, Tieren oder Kraftfahrzeugen wahrnimmt.

Die Aufnahme von Bildern geschieht im JPEG-Format. Ein Foto ist, je nach Bildinhalt und Auflösung, ca. 1–30 kB groß. Es wird nach der Aufnahme an den Sensorknoten, der mit der Basisstation verbunden ist, geschickt. Der Durchsatz des Funkmoduls erlaubt es theoretisch bis zu 15 Bilder pro Minute in höchster Auflösung zu senden. Der Dialog zur Herstellung der Verbindung und Festlegung des Modus sowie der Auflösung (80×64 , 160×124 und 320×240) ist in Abbildung 4.2a dargestellt. Die höchste Auflösung von 640×480 ist nicht integriert, da Bilder in dieser Auflösung wesentlich mehr Speicher benötigen,



(a) Bildfilter. **(b)** Bildausschnittswahl.

Abbildung 4.3.: Dialoge zur Konfiguration der Bildverarbeitungsschritte.

die Übertragungs- und Verarbeitungsdauer stark ansteigt und somit die aufzuwendenden Ressourcen in keinem sinnvollen Verhältnis mehr zum Nutzen stehen. Zudem sind trotz höherer Auflösung kaum Verbesserungen bei der Bildauswertung zu erwarten. Abbildung 4.2b zeigt dem Nutzer das aktuelle Bild und Log-Ausgaben des Sensorknotens zu Debugging-Zwecken. So kann u. a. geprüft werden, ob die Verbindung zum Knoten erfolgreich war und die vorgenommenen Einstellungen vom Knoten umgesetzt werden.

Bildverarbeitung Durch den Einsatz in Innenräumen und Außenbereichen sowie sich ändernde Lichtverhältnisse durch künstliches Licht, Wolken und Sonneneinstrahlung, sind die aufgenommenen Bilder stets von unterschiedlicher Qualität mit wechselnden Kontrasten und Helligkeitsverteilungen. Durch das Ausrichten der Kamera können weiterhin uninteressante oder sogar störende Objekte im Bild erscheinen. Um bestmögliche Ergebnisse bei der anschließenden Objekterkennung zu erzielen, muss auf diese wechselhaften Herausforderungen eingegangen werden. Deshalb sind verschiedene Mechanismen in das Framework integriert. Diese lassen sich je nach Bedarf als Vorverarbeitungsschritt der Objekterkennung hinzufügen. Nachfolgend werden die verschiedenen Verfahren kurz vorgestellt und erläutert für welche Bedingungen sie geeignet sind.

Abbildung 4.3a zeigt den Dialog zum Hinzufügen und Konfigurieren der verschiedenen integrierten Bildverarbeitungsmechanismen. Dies sind:

- Homomorphischer Filter
- Anisotropischer Filter
- Histogrammangleichung
- Adaptive Histogram Equalization (AHE)

- Contrast Limited Adaptive Histogram Equalization (CLAHE)

Der homomorphische Filter [154] ermöglicht eine Kompensation von schlechten Lichtverhältnissen, z. B. inhomogener Ausleuchtung. Als Ergebnis des Filters werden schattige Regionen aufgehellt und überbelichtete Bildteile angepasst, so dass die Helligkeit normalisiert wird. Somit haben wechselnde Lichtverhältnisse weniger Einfluss auf die spätere Objekterkennung.

Durch die geringe Auflösung können einige Strukturen im Bild schlecht abgebildet sein. Zur Verbesserung der Textur, d. h. Schärfung von Kanten, Rauschunterdrückung und Glättung von Bereichen zwischen markanten Kanten, kann der anisotropische Filter [155] eingesetzt werden. Damit sind weit entfernte Objekte schärfer zu erkennen.

Zur Verbesserung des Bildkontrastes sind mehrere Verfahren in LoCaF integriert. Diese ordnen jedem Pixel einen neuen Helligkeitswert zu, um so den Kontrast zu erhöhen. Dies kann zu einer besseren Unterscheidbarkeit von Objekten im Bild führen und somit die nachfolgende Objekterkennung verbessern.

Zur Kontrastverbesserung stehen drei Histogrammlinearisierungen zur Verfügung. Die einfachste Form betrachtet das Histogramm des ganzen Bildes und verteilt die Häufigkeit in diesem so um, dass sie gleichmäßiger verteilt wird. AHE [156] dagegen untersucht immer nur eine lokale Nachbarschaft pro Pixel für die Umverteilung. Das CLAHE [157]-Verfahren erweitert AHE, indem ein weiterer Parameter die maximale Kontrastanhebung angibt.

Der Einsatz der Bildverarbeitungsalgorithmen kann vollständig durch den Nutzer konfiguriert und an die Bedürfnisse des Einsatzszenarios angepasst werden. Abbildung 4.3a zeigt in der linken Spalte die verschiedenen, zur Verfügung stehenden Algorithmen. Der Nutzer kann dort selektieren, welche Algorithmen mit welchen Parametereinstellungen angewendet werden sollen. In der rechten Spalte werden die gewählten Algorithmen in einer Liste dargestellt. Der Nutzer kann dieser Liste beliebig viele Algorithmen hinzufügen und sie nach Belieben umsortieren. Die gewählte Reihenfolge spiegelt die Sequenz der Anwendung auf das Bild wider. So kann z. B. erst ein homomorphischer Filter und anschließend AHE oder erst CLAHE und danach ein anisotropischer Filter ausgeführt werden. Auch die mehrfache Ausführung des gleichen Algorithmus ist möglich.

Wie anfangs erwähnt, können einige Bildbereiche uninteressant sein oder störende Objekte wie wehende Flaggen oder Bäume enthalten. Abbildung 4.3b zeigt einen Dialog, der es erlaubt, mit Hilfe eines Polygons einen oder mehrere Bildausschnitte, sogenannten Regionen, einzuzeichnen. Jede erzeugte Region wird im weiteren Verlauf für sich behandelt und entspricht somit später einem Zustand, der aus der Region ermittelt wird. Dies erlaubt z. B. ein Bild von benachbarten Parkplätzen in mehrere Regionen, eine für jeden Parkplatz, aufzuteilen und für jeden Parkplatz einen eigenen Zustand zu ermitteln und zu veröffentlichen.

Objekterkennung Einige der in Tabelle 3.1 und 3.2 in Abschnitt 3.2 genannten Szenarien hängen unmittelbar mit der An- und Abwesenheit von Personen zusammen. Um das Framework für diese Szenarien nutzen zu können, ist eine, auf den Arbeiten von Viola und Jones [158] basierende, Gesichtserkennung integriert. Diese kann auf einem einzelnen Bild schnell Gesichter mit einer hohen Erfolgsrate erkennen. Zur erfolgreichen Erkennung müssen die Köpfe von Personen allerdings fast gerade abgebildet sein. Außerdem dürfen sie keine Sonnenbrillen oder andere maskierenden Gegenstände im Gesicht tragen. Somit ist die erfolgreiche Verwendung nur begrenzt möglich.

Um auch Personen zu erkennen, die seitlich oder rücklings zur Kamera orientiert sind, ist ein weiteres, flexibleres Verfahren zu integrieren. Da in einigen Szenarien, wie der Parkplatzbelegung, auch Objekte erkannt werden müssen, ist eine Hintergrundsubtraktion/-segmentierung beinhaltet. Das Verfahren versucht den statischen Hintergrund von zwei oder mehr Bildern durch Subtraktion zu erkennen. Dadurch werden sich ändernde Pixel im Bild erkannt, welche die Bewegungen im Bild repräsentieren und als Vordergrund klassifiziert werden. Einfache Verfahren klassifizieren dabei jedes bewegte Pixel als Vordergrund. Dies kann jedoch unerwünscht sein. So werden auch leichte Kontrastveränderungen durch wechselnde Lichtverhältnisse erkannt. Diese werden z. B. durch Wolken oder im Wind wehende Bäume hervorgerufen. Deswegen ist die auf den Arbeiten von Stauffer und Grimson [159] basierende adaptive Hintergrundsubtraktion implementiert. Der Algorithmus besitzt die Fähigkeit gleichförmige Bewegungen im Bild zu lernen und als Hintergrund zu klassifizieren. Die Sensitivität lässt sich dabei über Parameter einstellen.

Das Ergebnis der Hintergrundsegmentierung ist ein Schwarz-Weiß-Bild. Die weißen Pixel markieren den Vorder-, die schwarzen den Hintergrund. Aus diesem Bild lassen sich zwei interessante Aspekte extrahieren: zum einen, wie viele verschiedene bewegende Objekte erkannt werden, zum anderen, wie viel Fläche diese einnehmen. Die Fläche ist dabei die Menge weißer Pixel im Verhältnis zur Gesamtpixelzahl. Für die Anzahl an Objekten markiert eine Blob-Erkennung [160] die Pixel einer zusammenhängenden weißen Fläche mit dem gleichen Bezeichner und zählt anschließend, wie viele Bezeichner verwendet werden. Jeder Blob wird mit einer zufälligen Farbe eingefärbt, so dass die erkannten Objekte leicht zu identifizieren sind.

Zustandsermittlung Die Ausgabe der Objekterkennung dient als Eingabe für die Zustandsermittlung. Dies ist zum einen die Anzahl an Blobs (Parametername *count*), welche die Anzahl der beweglichen Objekte der Szene widerspiegelt und zum anderen die Fläche der Blobs in Prozent der Szene (Parametername *area*). Eine einfache Regelsprache mit zwei verschiedenen Funktionen, *map* und *switch*, erlaubt aus diesen Eingabeparametern einen Zustand zu ermitteln.

Die *map*-Funktion, die auf der Intervall-Abbildung basiert, dient zum Abbilden einer Zahl oder eines Zahlintervalls auf einen Zustand. Die Syntax ist *{count|area}:map:u:o:z*, wobei *count* und *area* die oben beschriebenen Eingabepa-

```
count:map:0:1:frei
count:map:1:-1:belegt
```

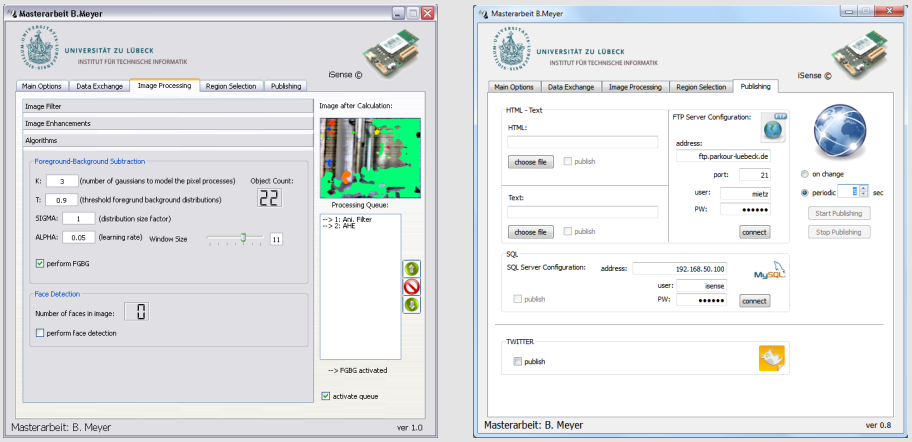
Quelltext 4.1: Beispiel von zwei *map*-Regeln für einen Tagungsraum.

```
area:switch:frei:80:belegt
area:switch:belegt:80:frei
```

Quelltext 4.2: Beispiel von zwei *switch*-Regeln für einen Parkplatz.

rameter bezeichnen. Das Schlüsselwort *map* bezeichnet die genutzte Funktion. Die weiteren Parameter sind aus der Intervallabbildung bekannt. Quelltext 4.1 zeigt ein Beispiel mit zwei Regeln für einen Tagungsraum. Werden keine beweglichen Objekte erkannt, so ist der Raum im Zustand *frei*. Sind dagegen ein oder mehr Objekte erkannt worden, -1 ist hierbei ein Alias für unendlich, so hat der Raum den Zustand *belegt*.

Mit den Regeln der *map*-Funktion lassen sich nicht alle Szenarien zufriedenstellend beschreiben. Dies ist darauf zurückzuführen, dass als Eingabe die Anzahl bewegter Objekte im Bild genutzt wird. Bewegt sich ein Objekt nicht mehr, so wird es auch nicht mehr erkannt. Würden z. B. alle Personen in einem Tagungsraum still sitzen, so würde dieser wieder als *frei* markiert werden. Bei Personen ist ein absoluter Stillstand eher unwahrscheinlich. Bei Autos auf Parkplätzen dagegen nicht. Somit würde ein Parkplatz, auf dem ein Auto steht, nach einiger Zeit, nämlich sobald das Auto nicht mehr als bewegtes Objekt klassifiziert wird, fälschlicherweise als *frei* ausgezeichnet werden. Um mit solchen Szenarien umgehen zu können, ist die *switch*-Funktion integriert. Die Syntax ist *{count|area}:switch:dz:s:zz*. Die Schlüsselwörter *count* und *area* be-



(a) Objekterkennung. **(b)** Veröffentlichung.

Abbildung 4.4.: Einstellungsdialoge für die Objekterkennung und die Veröffentlichung von Sensorzuständen.

Schlüsselwort	Ersetzung
$region#$x$	Name der Region x
$blobs#$x$	Anzahl erkannter Objekte in Region x
$state#$x$	Zustand der Region x
$time	Aktuelle Zeit t
$forLoop[$region]	Fügt für jede Region eine vordefinierte Zeichenkette ein

Tabelle 4.2.: Schlüsselwörter für die Vorlagen und entsprechende Ersetzungen.

zeichnen wieder die Eingabe. Das Element *switch* ist der Funktionsname. Durch *dz* und *zz* werden der derzeitige und zukünftige Zustand definiert. Der Schwellwert *s*, der überschritten werden muss, gibt an, wann von *dz* zu *zz* gewechselt wird. Durch mehrere Regeln dieser Form lässt sich ein Zustandsautomat konstruieren, durch den bei Schwellwertüberschreitung von einem Zustand zum nächsten übergegangen wird. Allerdings geschieht ein Übergang nur, wenn der Eingabeparameter zwischenzeitlich einmal unter den Schwellwert gefallen ist. Ansonsten würde bei jedem Bild mit entsprechender Eingabegröße ein Wechsel durchgeführt, was nicht gewünscht ist. Dies lässt sich gut am Beispiel in Quelltext 4.2 nachvollziehen. Dieser beschreibt zwei Regeln für einen Parkplatz. Sobald 80 % der Szene mit einem bewegten Objekt gefüllt ist, wechselt der Zustand von *frei* auf *belegt* bzw. zurück. Zwischenzeitlich muss das Auto allerdings still stehen und darf nicht mehr erkannt werden. Angenommen, das Einparken dauert mehrere Bildaufnahmen an. Würde ohne die Bedingung der zwischenzeitlichen Unterschreitung des Schwellwertes zwischen Zuständen gewechselt werden, so würde mehrfach der Zustand von *frei* zu *belegt* und zurück wechseln. Schlussendlich könnte der Parkplatz somit trotz Auto als *frei* gelten.

Veröffentlichung des Zustandes Hat das Framework einen oder mehrere Zustände aus einem Bild ermittelt, so können diese im Internet veröffentlicht werden und sind somit weltweit, in Echtzeit, für jedermann zugänglich.

Als Ergebnis aus den vorherigen Verarbeitungsschritten und der Nutzerangaben stehen folgende Informationen zur Verfügung:

- Bezeichnung *E* einer Region bzw. einer Entität, die durch diese Region repräsentiert wird
- Anzahl *n* der erkannten Objekte in der Region
- Zustand *Z* der Region
- Aktuelle Zeit *t*

Um diese Informationen zu veröffentlichen, stehen drei Varianten zur Verfügung:

- Mit Hilfe von Vorlagen als Textdokument
- Als Eintrag in einer SQL-Datenbank

- Als Kurznachricht im sozialen Netzwerk Twitter

Der Zeitpunkt bzw. die Häufigkeit der Veröffentlichung kann wie folgt gesteuert werden:

- Periodisch in einem vom Nutzer definierten Abstand
- Jedes Mal, wenn sich der Zustand ändert

Die Kombination dieser Einstellungen erlaubt eine Vielzahl von Möglichkeiten und somit eine höchstmögliche Flexibilität für den Nutzer. Abbildung 4.4 zeigt den Einstellungsdialog für die Veröffentlichung.

Bei der vorlagenbasierten Variante gibt der Nutzer eine Textdatei in einem beliebigen Format als Vorlage an. Somit können z. B. durch die Verwendung von TXT-, HTML- oder TTL-Dateien (Format der Turtle-Notation) eine simple Textdatei, eine Webseite oder ein RDF-Dokument erzeugt werden. Die Datei kann neben beliebigem Text bestimmte Schlüsselwörter enthalten, die Variablen darstellen, welche durch einen Parser entsprechend ersetzt werden. Eine Liste der unterstützten Schlüsselwörter befindet sich in Tabelle 4.2. Dadurch wird die Zeichenkette

> *Der $region#1 hat den Zustand $state#1, da um $time Uhr $blobs#1 Personen in ihm gesessen haben.*

durch den Parser umgeformt zu

> *Der Hörsaal T1 hat den Zustand belegt, da um 13:37:00 Uhr 21 Personen in ihm gesessen haben.*

Die verarbeitete Datei wird anschließend auf den vom Nutzer definierten FTP-Server geladen.

Die zweite Variante, die Speicherung in einer Datenbank, fügt die Werte E, n, Z und t in eine vom Nutzer zu spezifizierende Datenbank ein. Dafür muss er die Adresse, den Port, den Datenbanknutzer und das Passwort angeben.

Die letzte Variante ist das Schreiben einer Kurznachricht mit maximal 140 Zeichen, auch Tweet genannt, in dem sozialen Netzwerk Twitter[3]. Es wird eine Zeichenkette, ähnlich dem obigen Beispiel, für jede der definierten Regionen gepostet. Die Nachrichten sind für jeden zugänglich und können auch abonniert werden. Dazu muss man sich selbst anmelden und dem entsprechendem Account folgen. Abbildung 4.5 zeigt ein paar der Tweets, die das Framework getätigt hat.

Evaluation

Zur Bewertung der Funktionsweise der Objekterkennung und Regelverarbeitung des Frameworks werden mehrere Testreihen durchgeführt. Im ersten Szenario

[3] `http://www.twitter.com/` [Zugriff am 21. März 2014]

Abbildung 4.5.: Screenshot einiger Tweets durch LoCaF.

wird versucht den Belegungszustand eines Hörsaals zu ermitteln. Das zweite Szenario befasst sich mit der Belegung eines Parkplatzes.

Hörsaal-Szenario Im ersten Szenario wird die Belegung eines Hörsaals erfasst. Dazu ist die Kamera auf die Sitzreihen eines Vorlesungssaals gerichtet. Die Aufnahme startet vor Beginn der Vorlesung, wenn der Raum noch leer ist. Das Rohbild wird ohne jegliche Szenenauswahl oder Filterung für die Objekterkennung genutzt. Die Aufnahmen werden in drei Abschnitte aufgeteilt. In der ersten Phase betreten die Studenten den Hörsaal. Hier ist mit viel Bewegung zu rechnen. Im zweiten, eher ruhigen Abschnitt werden die Studenten während der Vorlesung aufgenommen. Die letzte Phase ist das Verlassen, bei dem wiederum viel Bewegung zu erwarten ist.

Abbildung 4.6 zeigt ausgewählte Originalbilder und die zugehörigen Bilder der Objekterkennung. Die Buchstaben in den oberen linken Ecken sind Identifikatoren. Die Zahlen in der rechten unteren Ecke zeigen die tatsächliche bzw. durch die Objekterkennung erkannte Anzahl an Studenten beim Betreten des Hörsaals. Abbildung 4.7 zeigt den Verlauf für die gesamte erste Phase inklusive der Markierung, zu welchem Zeitpunkt die Zahl unter- bzw. überschätzt wird.

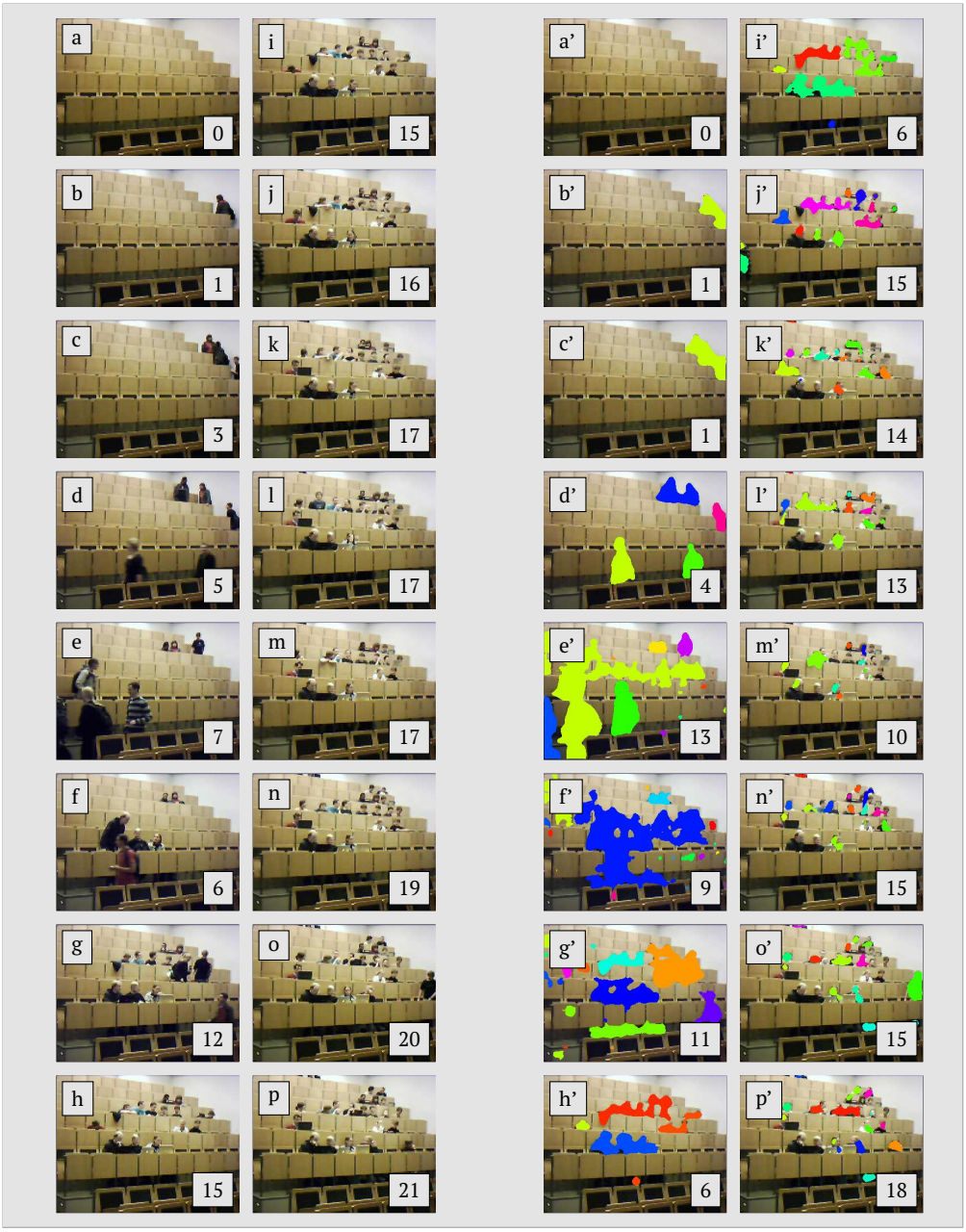


Abbildung 4.6.: Originalbilder (links) und nach der Blob-Erkennung (rechts). Die Anzahl an tatsächlichen (links) und erkannten (rechts) Studenten beim Betreten des Hörsaals ist in der unteren rechten Ecke angegeben.

Während sich anfangs Unter- und Überschätzung noch abwechseln, erfolgt ab Bild 18 bis zum Ende grundsätzlich eine Unterschätzung. Schaut man sich die Bilder in Abbildung 4.6 an, so kann man zwei Gründe dafür feststellen: Zum einen werden, wie in den Bildern c und c' zu erkennen ist, mehrere Personen als ein gemeinsamer Blob klassifiziert, da sie sich nahe beieinander aufhalten. Zum anderen, wie in den Bildern k und k' sichtbar, werden stillsitzende Personen

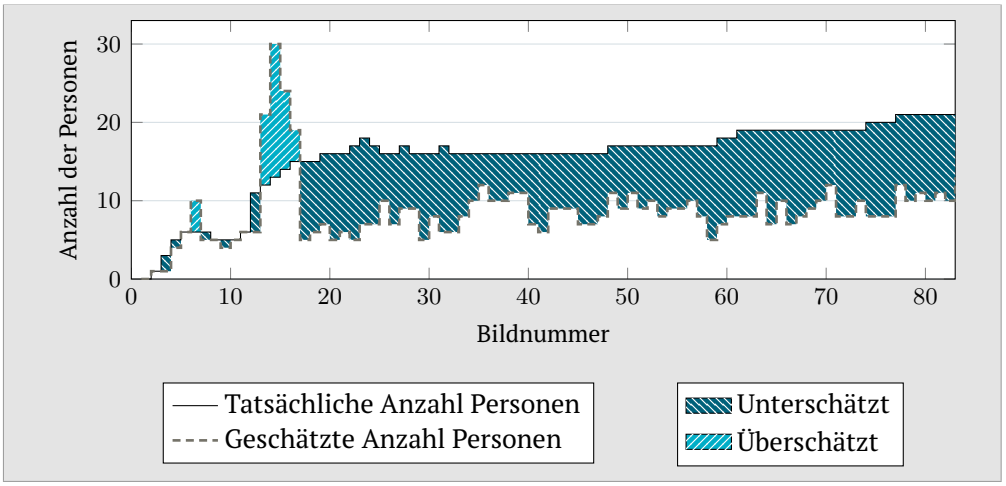


Abbildung 4.7.: Geschätzte und tatsächliche Anzahl Personen beim Betreten des Hörsaals.

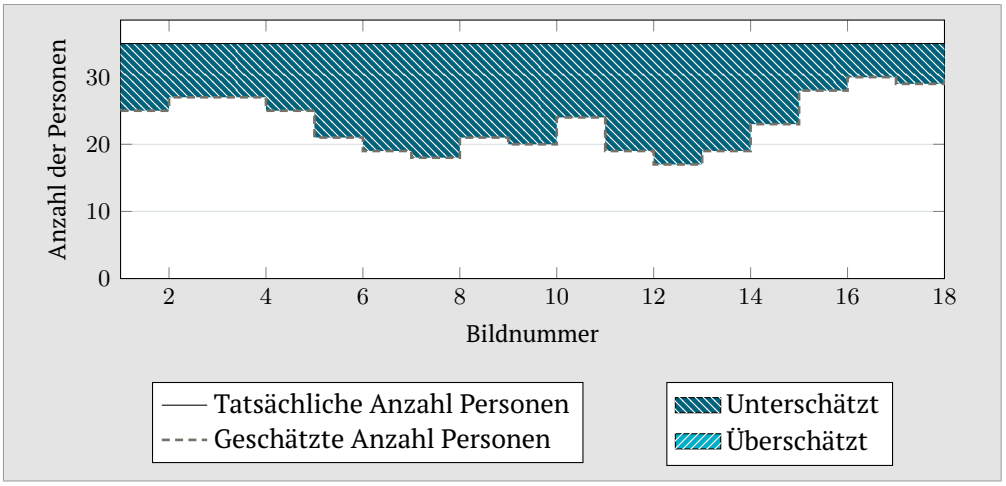


Abbildung 4.8.: Geschätzte und tatsächliche Anzahl während der Vorlesung.

nicht mehr erkannt. Die Überschätzung wiederum kann z. B. auf zusätzliche Blobs an Händen und Füßen zurückgeführt werden, wie sie in den Bildern h/h' und p/p' auftreten.

Das Unterschätzen durch Nichterkennung von stillsitzenden Personen aus der ersten Phase ist noch deutlicher in der zweiten Phase zu erkennen. Abbildung 4.8 zeigt, dass die Anzahl Personen durchgehend unterschätzt wird.

Ebenfalls unterschätzt wird die Anzahl zu Beginn der dritten Phase, dem Verlassen des Hörsaales (siehe Abbildung 4.9). Zu beobachten ist, dass die Zahl der Studenten rapide abnimmt, die Hintergrundsubtraktion allerdings nicht so schnell reagiert und somit Personen erkennt, obwohl der Raum bereits leer ist. Dies zeigt die Grenzen des Verfahrens auf. Der Algorithmus benötigt aber im Allgemeinen nur eine kleine Anzahl Bilder, bis er wieder einen komplett

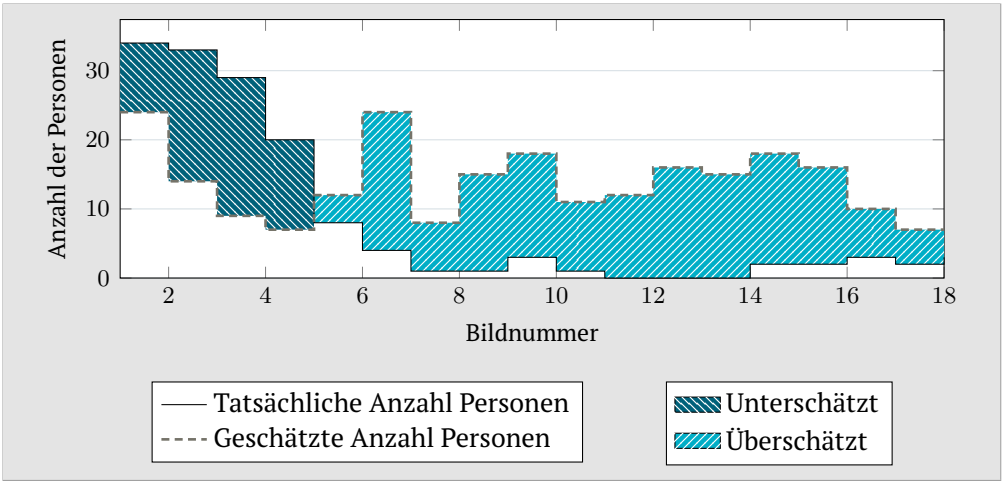


Abbildung 4.9.: Geschätzte und tatsächliche Anzahl Personen beim Verlassen des Hörsaals.

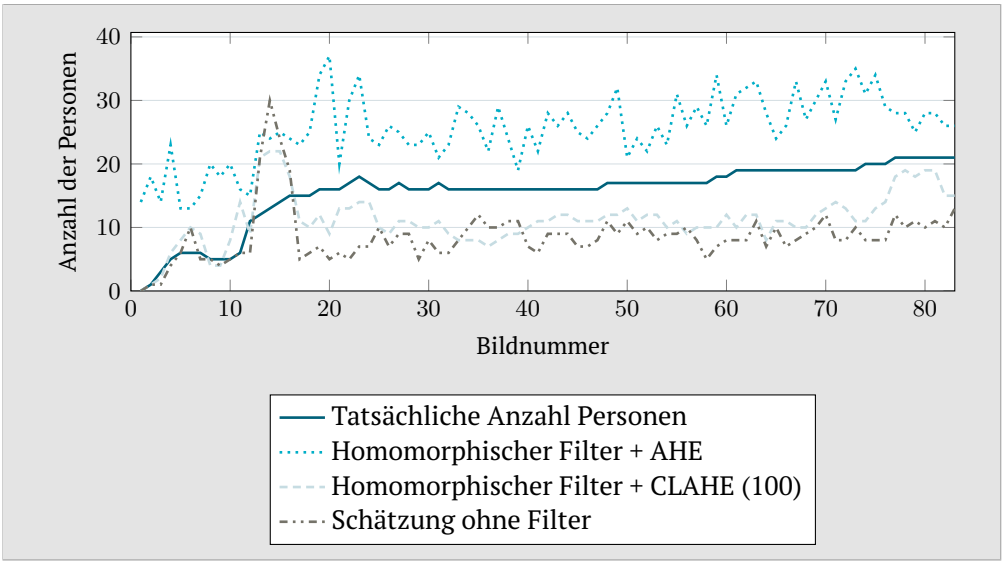


Abbildung 4.10.: Vergleich der tatsächlichen Anzahl an Personen mit der geschätzten Anzahl ohne und mit Anwendung von Filtern.

leeren Raum erkennt. In der Evaluation ist es dazu nicht gekommen, da bereits Studenten für die nächste Vorlesung den Hörsaal betreten. Diese werden allerdings sofort wieder vom Algorithmus erkannt.

Obwohl die korrekte Anzahl der Personen in allen drei Phasen fast immer unter- oder überschätzt wird, ist die Tendenz der Schätzung gut. Für ein genaues Abzählen der Personen eignet sich das Framework daher nur bedingt. Wendet man dagegen die Regel aus Quelltext 4.1 an, kann die Belegung des Raumes fast zu 100 % korrekt geschlussfolgert werden. Nur in Phase 3 wird der Raum als *belegt* klassifiziert, obwohl er kurzzeitig frei ist.

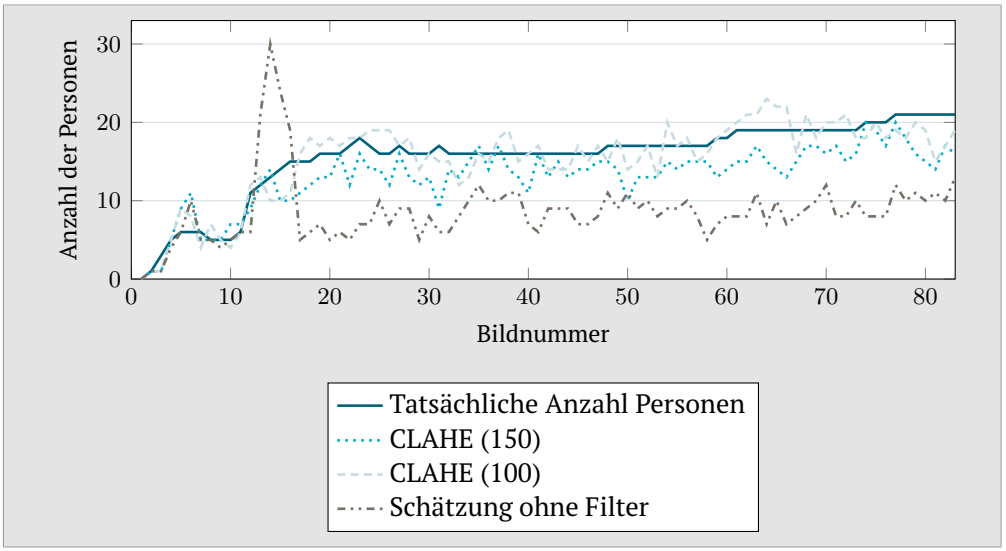


Abbildung 4.11.: Vergleich der tatsächlichen Anzahl an Personen mit der geschätzten Anzahl ohne und mit Anwendung von CLAHE.

Als nächstes wird versucht die Ergebnisse durch Nutzung von Filtern zu verbessern. Dazu werden die Bilder der ersten Phase wiederholt ausgewertet. Der homomorphische Filter wird im ersten Durchlauf mit AHE kombiniert, im zweiten mit CLAHE mit einem Wert von 100. Abbildung 4.10 zeigt die Schätzungen der beiden Durchläufe mit Filter, die Schätzung ohne Filter sowie die tatsächliche Anzahl an Personen. Die Kombination mit AHE führt zu einer dauerhaften Überschätzung. Bei der CLAHE-Filterung fallen sowohl Über- als auch Unterschätzung nicht mehr so stark aus wie bei der Schätzung ohne Filter.

Abschließend werden zwei Durchläufe mit alleiniger Filterung durch CLAHE mit den Parametern 100 und 150 durchgeführt. In Abbildung 4.11 ist zu erkennen, dass diese Filterung die besten Ergebnisse aller Versuchsdurchführungen erbringt. Es werden zwar immer noch nicht die genauen Personenzahlen geschätzt, die Schwankungen um den korrekten Wert sind jedoch sehr gering.

Parkplatz-Szenario Das zweite Szenario untersucht abermals den Belegungszustand. Allerdings geht es dieses Mal um einen Parkplatz. Ein weiterer Unterschied ist, dass die Versuchsreihe unter freiem Himmel stattfindet und somit wechselnden Lichtverhältnissen ausgesetzt ist. Am Versuchstag war es sonnig mit vereinzelten Wolken. Die Parameter für die Hintergrundsubtraktion müssen deshalb im Vergleich zum Hörsaal-Szenario geändert werden, um die erhöhte Anzahl an Hintergrundbewegungen, wie die von Wolken, korrekt als Hintergrundprozesse zu klassifizieren.

Abbildung 4.12 zeigt ein paar Originalbilder und die dazugehörigen Bilder nach der Blob-Erkennung. Das Auto wird auf jedem Bild erkannt. Nur Lampen und Scheiben werden zeitweise nicht detektiert. Dies liegt vermutlich an der Transparenz. Dadurch kommt es zeitweise, wie in Bild f', vor, dass das Fahrzeug

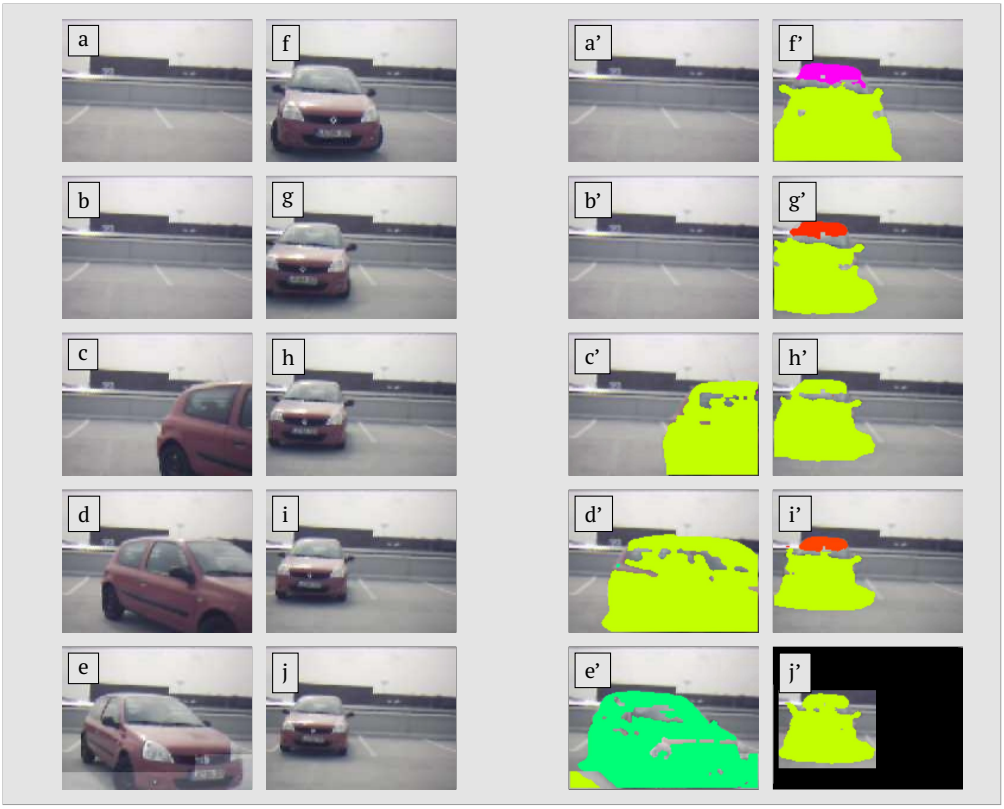


Abbildung 4.12.: Originalbilder (links) und nach der Blob-Erkennung (rechts) eines einparkenden Autos auf einem Parkplatz. Das letzte Bild stellt eine mögliche Szenenauswahl eines einzelnen Parkplatzes dar.

als zwei Objekte klassifiziert wird. Beschränkt man die Szene auf den linken Parkplatz der beiden sichtbaren (siehe Bild j' in Abbildung 4.12) und wendet die Regeln aus Quelltext 4.2 an, so wechselt der Zustand von *frei* auf *belegt*, da über 80 % der Szene durch den Blob des Fahrzeuges ausgefüllt ist. Bleibt das Auto anschließend eine Zeit stehen, so dass ein Bild aufgenommen wird, auf dem es nicht erkannt wird und fährt danach wieder los, so löst die 2. Regel aus und der Parkplatz gilt wieder als *frei*. Würde man dagegen die gleichen Regeln wie beim Hörsaal-Szenario anwenden (Quelltext 4.1), so würde schon die Erkennung eines vorbeilaufenden Menschen oder eines Vogels die Regel aktivieren und den Parkplatz als *belegt* kennzeichnen.

Fazit

Das vorgestellte Framework nutzt einen drahtlosen kamerabestückten Sensorknoten und eine Software, die auf einer Basisstation läuft, um aus den aufgenommenen Bildern in Echtzeit mittels Bildfiltern, Objekterkennung, Blob-Erkennung und einer einfachen Regelsprache Zustände der beobachteten Szene zu ermitteln und ins Internet zu stellen. Dabei kann der Nutzer die Sensorknoten, Bildverarbeitung, Zustandsermittlung und -veröffentlichung flexibel

konfigurieren. Die Evaluationen zeigen, dass verschiedene Szenarios im Innen- und Außenbereich realisierbar sind. Weiterhin wird dargelegt, dass man aus Bildern mit niedriger Auflösung und schlechter Bildqualität aussagekräftige höherwertige Zustände mit einer hohen Erfolgsquote ermitteln kann. Somit könnte aus Webcam-Bildern, die heutzutage in großer Anzahl und weltweit verfügbar sind, mit einer ähnlichen Software Zustände der realen Welt in Echtzeit ermittelt und publiziert werden.

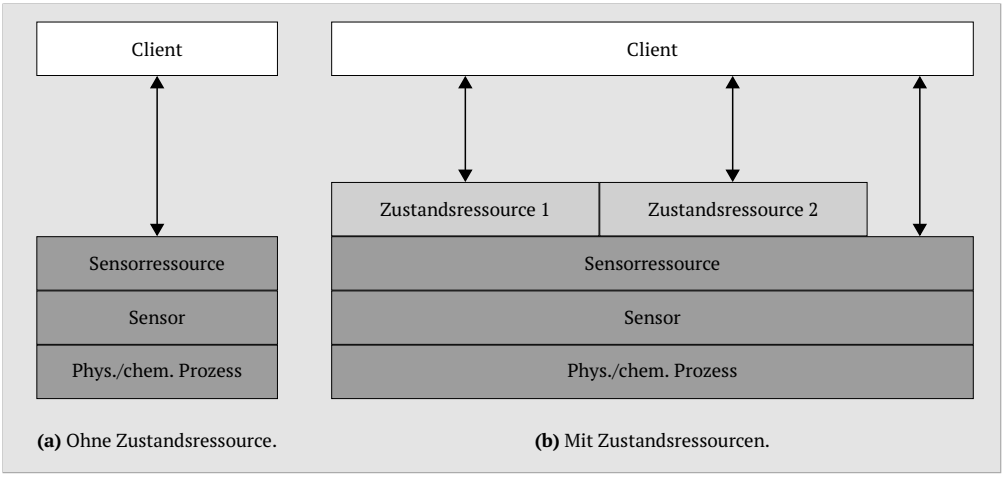
4.2.2. High-Level-State-Option für das Constrained Application Protocol

In Abschnitt 4.2.1 wurde gezeigt, wie aus Bildern ein Zustand ermittelt werden kann. Allerdings besitzen die meisten handelsüblichen Sensorknoten keine Kamera, sondern nur einfache Sensorik, z. B. für Temperatur und Luftfeuchte. Diese messen meist skalare Werte. Auch für diese Typen von Sensoren sollte eine generische Lösung zur Zustandsermittlung existieren, anstatt für jeden Sensortyp eine eigene zu entwickeln. Weiterhin ist es wünschenswert, wenn diese Lösung auf weit verbreiteten, standardisierten und nicht auf proprietären Protokollen aufsetzt, so dass sie auf vielen Sensorknoten lauffähig ist.

Ein u. a. in Sensornetzen immer mehr Beachtung findendes Protokoll ist CoAP (vgl. Abschnitt 2.2.1). In CoAP werden Sensoren üblicherweise als Ressourcen repräsentiert. Der Wert des Sensors lässt sich durch Abfrage der Ressource abrufen (siehe Abbildung 4.13a). In diesem Abschnitt wird die High-Level-State-Option (HLS-Option) für CoAP beschrieben, die es erlaubt, dynamisch neue Ressourcen zu erzeugen, die aus den Werten einer existierenden Sensorressource einen Zustand ermitteln. Neben der unveränderten Sensorressource kann ein Client nach der Erzeugung auch die neue Zustandsressource abfragen (siehe Abbildung 4.13b). Im Folgenden werden der Aufbau und die Verwendung, d. h. die Erzeugung, Abfrage und das Löschen von Zustandsressourcen mit der HLS-Option, beschrieben. Im Anschluss daran werden diese Operationen an zwei Anwendungsbeispiele illustriert. Abschließend wird eine Evaluation der Option durchgeführt. Die wesentlichen Inhalte wurden vorab in [25, 26] veröffentlicht.

Aufbau der High-Level-State-Option

Die HLS-Option ist *elective*, d. h. sie kann in CoAP-Anfragen genutzt werden, muss es allerdings nicht. Weiterhin ist sie *repeatable*. Das bedeutet, falls sie genutzt wird, kann sie ein- oder auch mehrmals in einer Anfrage auftauchen. Der Grund und die Anwendung der Wiederholbarkeit sind in Abschnitt 4.2.2 beschrieben. Die Erweiterung ist nur in Anfragen und nicht in Antworten zu verwenden, wobei die Semantik der Erweiterung von der verwendeten Anfrage-Methode abhängt. Dies erläutern die folgenden Abschnitte genauer. Das generelle Format ist in Abbildung 4.14 gezeigt. Das T-Feld ist ein 2 Bit Ganzzahlwert, der den Typ angibt. In POST-Anfragen spezifiziert der Typ das Format des Payloads, der mit dem 2. Byte beginnt und maximal 255 Bytes lang sein kann. In GET-Anfragen wird durch T das vom Client erwartete Format angegeben. Die



(a) Ohne Zustandsressource. **(b)** Mit Zustandsressourcen.

Abbildung 4.13.: Veranschaulichung der Ressourcenabfrage durch einen Client. Die Sensorressource greift auf den Sensor zu, der wiederum den phys./chem. Prozess überwacht. Die Zustandsressourcen erzeugen ihren Zustand aus dem Wert der Sensorressource.

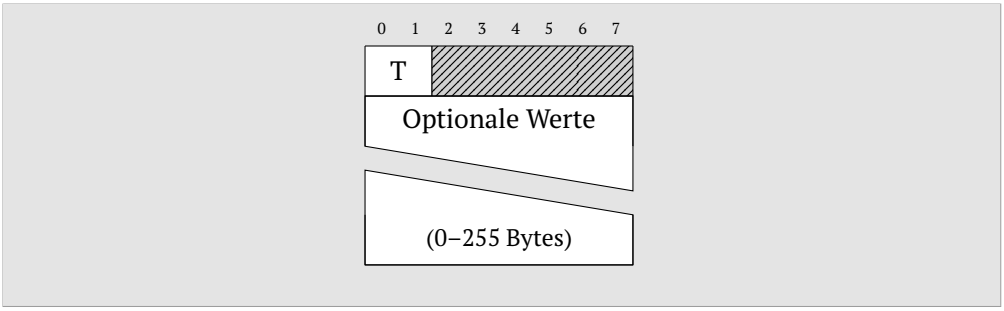


Abbildung 4.14.: Format der HLS-Option.

Bits 2–7 werden ignoriert und können für spätere Erweiterungen der Option genutzt werden.

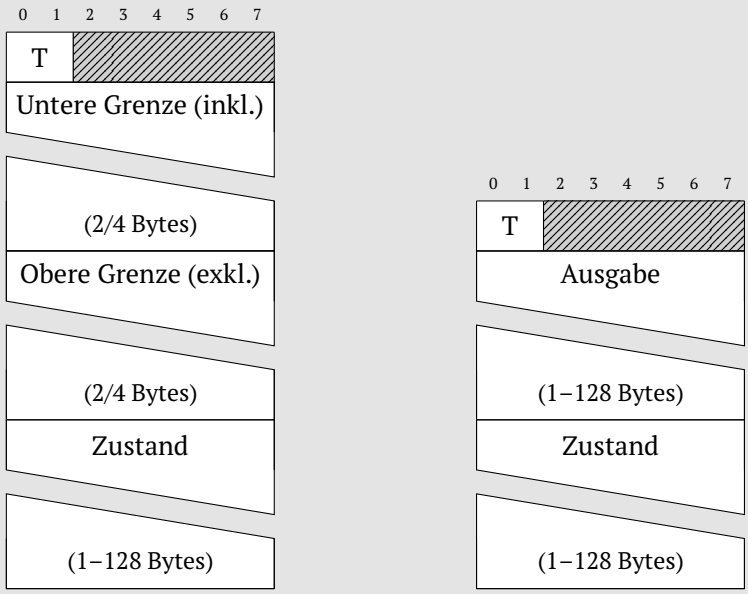
Nutzung der Erweiterung

Das Verhalten der HLS-Option hängt von der verwendeten Anfragemethode ab. Die Funktionen POST und DELETE werden zur Erzeugung und zum Löschen von Zustandsressourcen genutzt. Die GET-Methode dagegen ist zum Abrufen des Zustandes bzw. einer Beschreibung der existierenden Zustandsressourcen vorgesehen.

Erstellung von Zustandsressourcen Um eine neue Zustandsressource für eine Sensorressource zu erstellen, muss ein Client eine POST-Anfrage mit der HLS-Option an die Sensorressource richten. Der Server erstellt daraufhin eine Subressource, welche die durch die POST-Anfrage definierten Zustände aus

Datentyp	Typ	Format	Länge (Bytes)
integer	0	1	2
float	1	1	4
string	2	2	1–128

Tabelle 4.3.: Datentypen und deren Länge für die HLS-Option.



(a) Format der Intervall-Abbildung. **(b)** Format der String-Abbildung.

Abbildung 4.15.: Formate für die Zustandsermittlung.

den Sensorwerten herleitet und auf Anfrage zurückgibt. Abbildung 4.13 zeigt, wie unterhalb einer Sensorressource zwei Zustandsressourcen angelegt werden und nutzbar sind. Auch die Sensorressource lässt sich weiterhin verwenden, um nach wie vor den Sensorrohwert abzufragen.

Der Client spezifiziert in der POST-Anfrage über das T-Feld, ob die Intervall-Abbildung oder die String-Abbildung verwendet werden soll. Tabelle 4.3 listet die möglichen T-Werte, den zugehörigen Datentyp und das dadurch festgelegte Format sowie dessen Länge auf. Abhängig von der Variante ist auch das Format der Option, d. h. welche Daten ab dem 2. Byte folgen. Die beiden Formate sind in Abbildung 4.15 dargestellt.

Abbildung 4.15a zeigt das Format, welches für $T = 0$ und $T = 1$ Verwendung findet. Es beinhaltet die untere Intervallgrenze u (inkl.) und die obere Grenze o (exkl.) sowie den Zustand z, der für dieses Intervall angenommen wird. Das Format in Abbildung 4.15b, das bei $T = 2$ zum Einsatz kommt, wird für die String-Abbildung verwendet. Es besteht aus dem String s und dem Zustand z, auf den s abgebildet wird.

T	Antwort
0	Zustand
1	Zustandsnummer
2	Beschreibung

Tabelle 4.4.: Antworttypen.

Wie bereits erwähnt ist die HLS-Option *repeatable*. Jedes Vorkommen der Option in einer Anfrage definiert eine Abbildung. Allerdings muss jedes Vorkommen den gleichen Wert für T besitzen. Ansonsten antwortet der Server mit einem Fehler. Es ist erlaubt, mehrfach auf den gleichen Zustand abzubilden. Dadurch können zwei Zahlintervalle, die nicht zusammenhängend sind, oder verschiedene Zeichenketten auf den gleichen Zustand abgebildet werden.

Falls dem Server ausreichend Ressourcen zur Verfügung stehen, erzeugt er eine Zustandsressource und teilt dem Client die Adresse der Ressource mit. Die Adresse wird dabei zufällig vom Server generiert und darf vorher nicht schon einmal genutzt worden sein. Der Hintergrund der zufälligen Wahl und weshalb der Client die Adresse nicht selbst bestimmen darf ist folgender: Existiert eine Zustandsressource unter einem bestimmten Pfad, könnte ein Client diese Ressource löschen und anschließend mit einer anderen Semantik, also mit anderen Zuständen, eine neue Ressource unter derselben Adresse anlegen. Ein anderer Client, z. B. der ursprünglich erstellende, würde diese Veränderung nicht mitbekommen und ggf. unbekannte oder andersartig definierte Zustände erhalten.

Hat der Server nicht ausreichend Ressourcen zur Verfügung, weil er z. B. bereits einige Zustandsressourcen unterhält, so kann er das Erstellen der Zustandsressource ablehnen und mit einer Fehlermeldung quittieren. Das Erzeugen einer Zustandsressource ist nur für Ressourcen möglich, die Sensorwerte ausgeben. Für Ressourcen, die z. B. Metadaten bereitstellen, ist dies nicht vorgesehen und wird ebenfalls mit einer Fehlermeldung quittiert.

In dem unwahrscheinlichen Fall, dass ein Client eine Zustandsressource erstellen möchte, die eine identische Semantik zu einer bestehenden Zustandsressource hat, so erstellt der Server keine neue Ressource, sondern antwortet mit dem Pfad der existierenden.

Abfragen von Zustandsressourcen Zur Abfrage des aktuellen Zustandes von einer Zustandsressource ist eine GET-Anfrage ausreichend. Die HLS-Option muss dabei nicht in der Anfrage enthalten sein. Allerdings ist es möglich, die Option in der Anfrage mitzuschicken. Dadurch kann der Client kontrollieren, was der Server zurückgeben soll. Dies wird durch das T-Feld signalisiert. Tabelle 4.4 zeigt die erlaubten TYP-Werte.

Wenn $T = 0$ ist, so gibt der Server den aktuellen Zustand zurück. Dies ist identisch mit dem Weglassen der Option bei der Anfrage. Falls $T = 1$ ist, wird eine

Ganzzahl, die den Zustand repräsentiert, zurückgegeben. Diese Ganzzahl leitet sich wie folgt her: Beim Erzeugen der Zustandsressource wird jeder Zustand entsprechend der Reihenfolge in der Anfrage bei 0 beginnend durchnummeriert. Dieser Rückgabetyp ist nützlich, falls der Client alle möglichen Zustände der Ressource und die dazugehörigen ganzzahligen Werte kennt. Statt bei jedem Abruf eine Zeichenkette zu übertragen, wird nur eine Zahl verschickt. Dies spart Ressourcen bei der Datenübertragung ein. Die letzte Option, $T = 2$, gibt eine Beschreibung der Zustandsressource zurück, d. h. die möglichen Zustände und die Intervalle bzw. Strings, die auf diese abbilden. Der Quelltext A.1 in Anhang A.2 enthält ein XML Schema, welches das XML-Rückgabeformat definiert. Durch die *Accept-Option* lässt sich eine Inhaltsvereinbarung (Content Negotiation) realisieren und es kann so statt der wortreichen XML-Darstellung, auch eine kompaktere Repräsentation wie JSON angefordert werden. Kann der Server keine passende Abbildung für den Sensorwert finden, d. h. es ist kein Zustand für den aktuellen Wert definiert, so gibt er *undefined* zurück.

Die HLS-Option mit $T = 2$ in einer Anfrage kann auch an eine Sensorressource geschickt werden. Der Server antwortet darauf mit einer Auflistung und Beschreibung aller Zustandsressourcen, die für die Sensorressource existieren. Dies geschieht mittels einer XML-Darstellung, die dem XML Schema aus Quelltext A.2 in Anhang A.2 folgt.

Löschen von Zustandsressourcen Falls eine Zustandsressource nicht mehr benötigt wird, sollte diese gelöscht werden, um die von ihr belegten Ressourcen in Form von Speicher wieder freizugeben. Dazu muss eine DELETE-Anfrage an die Zustandsressource geschickt werden. Der CoAP-Server löscht daraufhin die Zustandsressource. Wie bereits erwähnt, darf die Adresse dieser, daraufhin nicht mehr existierenden Ressource, nicht erneut für eine Zustandsressource verwendet werden.

Anwendungsfälle

Zur Veranschaulichung der HLS-Option werden im Folgenden die Anfragen von Clients und Antworten des Servers für zwei Anwendungsfälle grafisch und textuell ausgeführt.

Anwendungsfall 1 Der erste Anwendungsfall umfasst vier Clients, die nicht an den rohen Sensorwerten eines Temperatursensors, sondern an einem Zustand interessiert sind. Deswegen wollen sie Zustandsressourcen, die auf der Sensorressource basieren, erzeugen.

Abbildung 4.16 zeigt die Erzeugung einer Zustandsressource durch Client 1. Dabei werden die Zustände *kalt* (von $-50\,°$C bis $10\,°$C) und *warm* (von $10\,°$C bis $50\,°$C) mittels einer POST-Anfrage angelegt. Die Zustandsressource lässt sich anschließend unter */temp/x42y* anfragen.

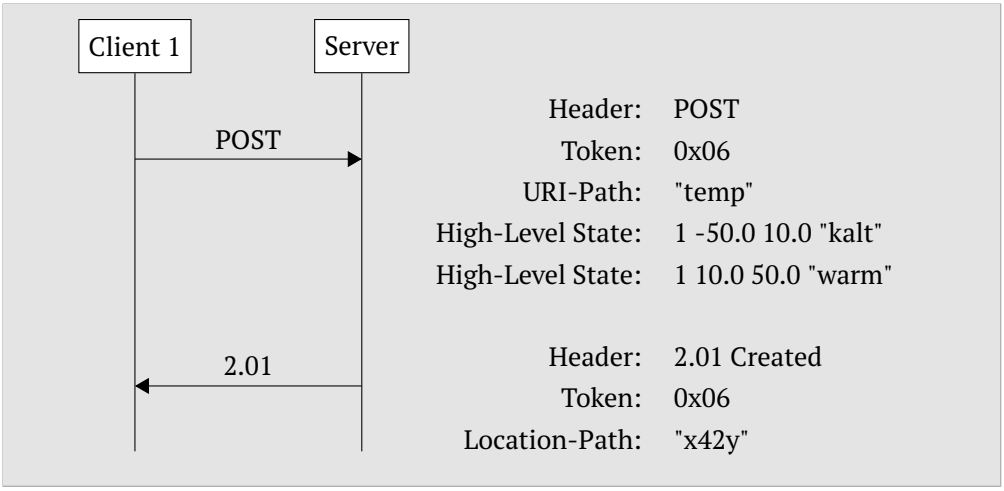


Abbildung 4.16.: Kommunikation zwischen Client 1 und dem Server zur Erzeugung einer Zustandsressource mit zwei Zuständen.

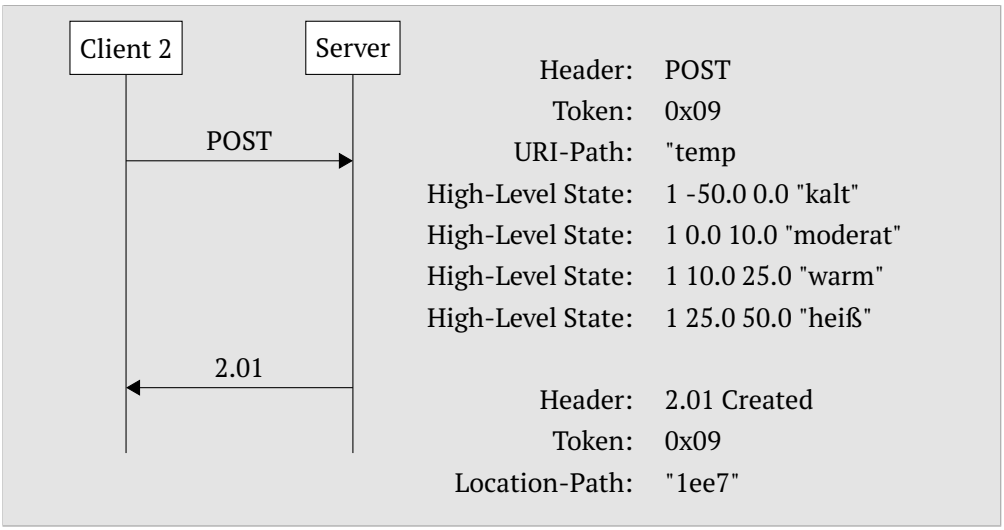


Abbildung 4.17.: Kommunikation zwischen Client 2 und dem Server. Client 2 erzeugt eine Zustandsressource mit vier Zuständen.

Darauffolgend legt Client 2 eine weitere Zustandsressource an, die unter dem Pfad */temp/1ee7* erreichbar ist und vier Zustände besitzt. Anfrage und Antwort sind in Abbildung 4.17 zu sehen.

Als Client 3 versucht eine Zustandsressource anzulegen, erkennt der Server, dass die geforderte Semantik dieselbe ist, die schon für die Ressource */temp/x42y* von Client 1 existiert. Deshalb erzeugt er keine neue Ressource, sondern schickt den Pfad der bestehenden zurück (siehe Abbildung 4.18).

Beim Versuch durch Client 4 eine weitere Zustandsressource zu erzeugen, quittiert der Server dies mit einer Fehlermeldung, weil nicht ausreichend Ressourcen zur Verfügung stehen (siehe Abbildung 4.19). Client 4 muss somit warten,

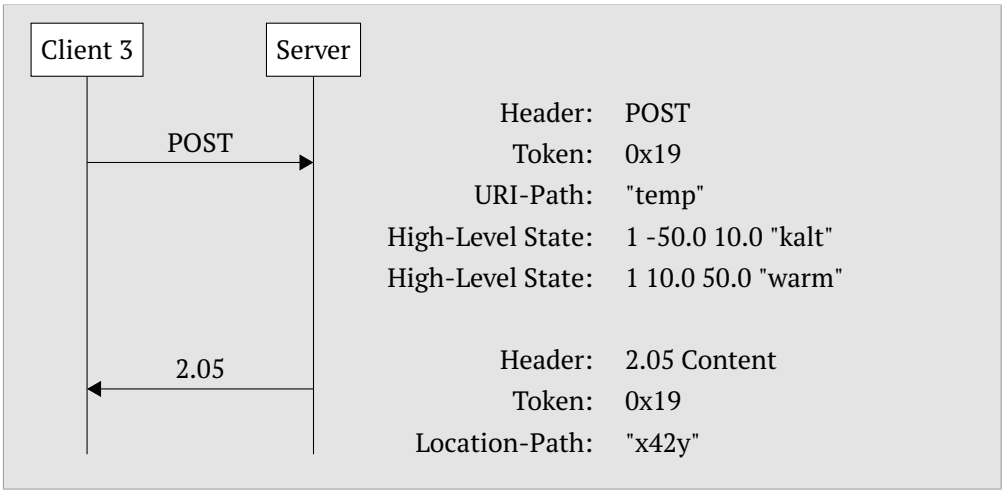


Abbildung 4.18.: Kommunikation zwischen Client 3 und dem Server. Client 3 möchte eine Zustandsressource mit gleicher Semantik wie eine bereits existierende Ressource erzeugen. Die Antwort beinhaltet die Adresse der bestehenden Ressource.

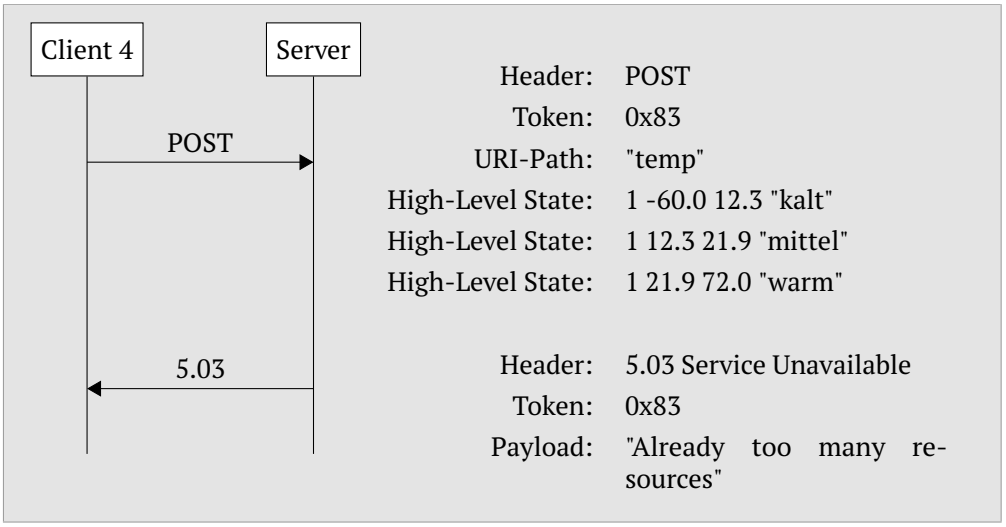


Abbildung 4.19.: Kommunikation zwischen Client 4 und dem Server. Die angeforderte Zustandsressource kann durch den Server nicht erstellt werden, da zu wenig Ressourcen vorhanden sind.

bis wieder Ressourcen bereit stehen (z. B. durch Löschen einer anderen Zustandsressource) oder er nutzt eine der bestehenden. Dazu kann er sich, wie in Abbildung 4.20 dargestellt, alle existierenden Ressourcen auflisten lassen.

Anwendungsfall 2 Der zweite Anwendungsfall behandelt einen Wettersensor, der Zeichenketten als Ergebnis ausgibt. Ein Client erzeugt mit Hilfe des String-Abbildungs-Mechanismus eine Zustandsressource, fragt diese ab und löscht sie nach der Nutzung wieder.

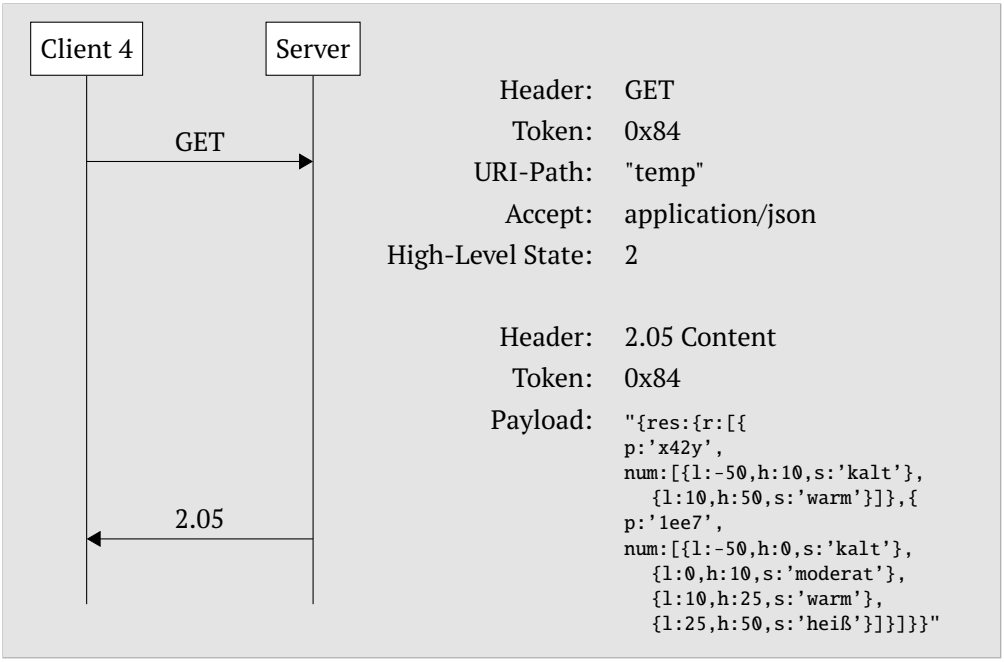


Abbildung 4.20.: Kommunikation zwischen Client 4 und dem Server zur Abfrage aller existierenden Zustandsressourcen.

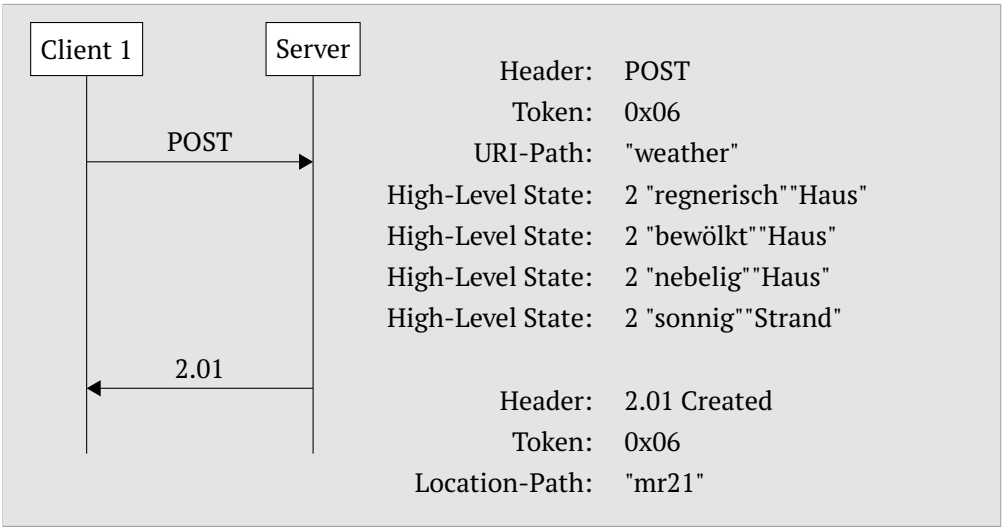


Abbildung 4.21.: Kommunikation zwischen Client 1 und dem Server zur Erzeugung einer Zustandsressource mit vier Zuständen unter Nutzung der String-Abbildung.

Abbildung 4.21 zeigt die Anfrage zur Erzeugung der neuen Ressource. Dabei werden die Strings *regnerisch*, *bewölkt* und *nebelig* auf *Haus* sowie der String *sonnig* auf den Zustand *Strand* abgebildet.

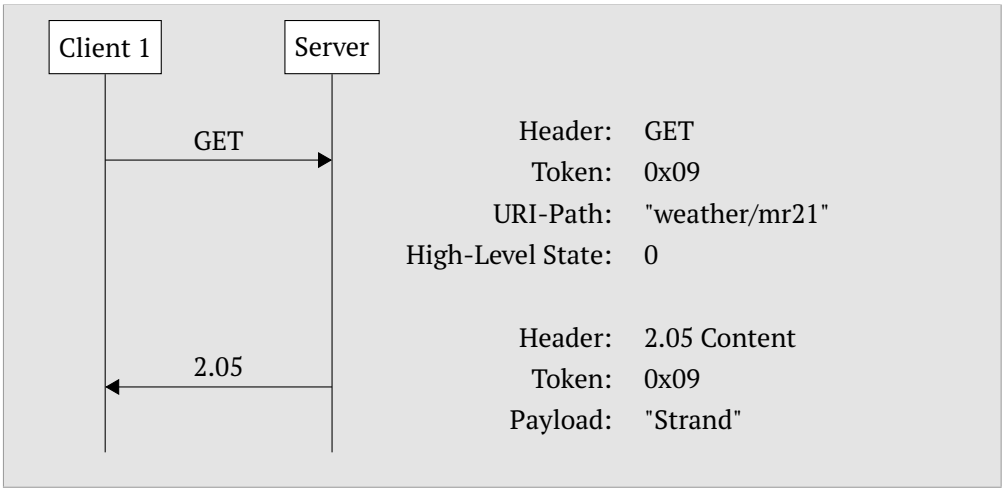


Abbildung 4.22.: Kommunikation zwischen Client 1 und dem Server zur Abfrage des aktuellen Zustandes.

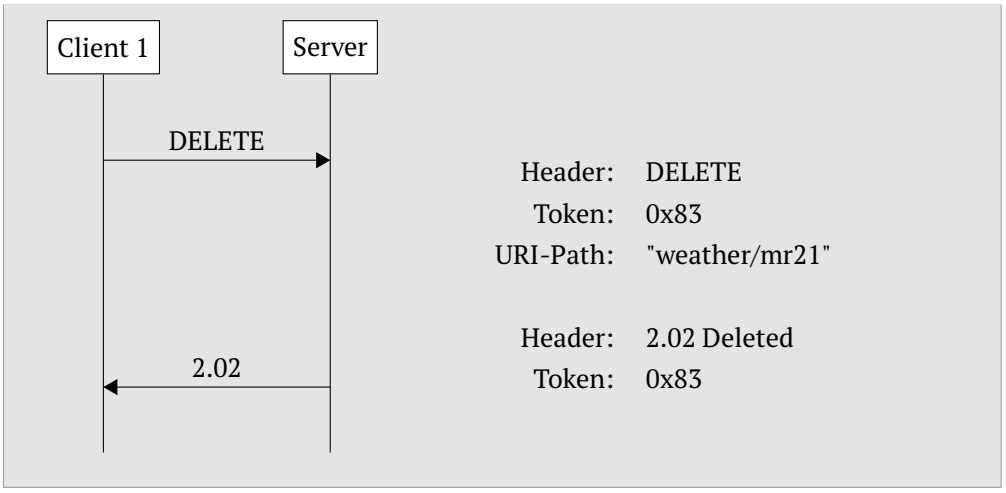


Abbildung 4.23.: Kommunikation zwischen Client 1 und dem Server zum Löschen der Zustandsressource.

Der Client ruft den aktuellen Zustand, wie in Abbildung 4.22 gezeigt, ab. Bei der dargestellten Abfrage ist es gerade *sonnig*, so dass die Anfrage mit dem Zustand *Strand* beantwortet wird.

Abschließend löscht der Client die Zustandsressource mit der Anfrage in Abbildung 4.23 wieder und die Ressourcen stehen auf dem Sensorknoten wieder zur Verfügung.

Evaluation

Die HLS-Option sowie weitere CoAP-Optionen wurden im Rahmen des Google Summer of Code 2013[4] für die Wiselib [161] programmiert. Die Wiselib ist eine, in C++ geschriebene, Algorithmenbibliothek, die es erlaubt, plattformunabhängige, generische Anwendungen zu schreiben, die sich anschließend ohne Mehraufwand für verschiedene Geräte bzw. Sensorknotenbetriebssysteme kompilieren lassen. Von Haus aus beinhaltet die Wiselib diverse Algorithmen aus verschiedenen Klassen wie Lokalisierung, Routing und Verschlüsselung. Sie bietet volle Unterstützung für iSense[5], Contiki[6], TinyOs[7] und den Simulator Shawn[8] sowie teilweise Unterstützung für iOS[9], PC Betriebssysteme, TriSOS[10], Scatterweb[2][11] und Feuerware[12].

Die Evaluation wird zur besseren Reproduzierbarkeit mit echten Datensätzen im Simulator Shawn anstatt auf realen Sensorknoten durchgeführt. Dadurch ist es möglich, Einflussfaktoren realer Hardware und des Kommunikationskanals, die für die Funktionsweise und die Bewertung der HLS-Option keine Rolle spielen, zu beseitigen. Die *Max-Age-Option* wird auf einen hohen Wert gesetzt, so dass nur Änderungen des Sensorwertes bzw. Zustandes mitgeteilt werden. Somit findet keine erneute Übertragung bei gleichem Wert statt.

Für den ersten Test werden Sensorwerte eines Innenraumlichtsensors aus dem Intel Lab Datensatz[13] genutzt. Abbildung 4.24 zeigt den Verlauf eines Tages, wobei alle 30 Sekunden ein Wert gemessen wird. Die farbige Hinterlegung kennzeichnet die zwei Zustände *dunkel* und *hell*, deren Übergang bei 100 Lux liegt. Unter dem Diagramm ist markiert, wann durch Observe (Obs) bzw. durch HLS in Kombination mit Observe eine Benachrichtigung für einen neuen Wert versendet wird. Jeder Strich entspricht dabei einer Nachricht. Während bei Observe 301 Nachrichten verschickt werden, sind es bei HLS + Obs nur 3. Selbst mit einem niedrigeren *Max-Age* Wert bzw. dem Standardwert von 60 Sekunden würde HLS + Obs nur maximal ca. die Hälfte der Anzahl an Nachrichten wie reines Observe verschicken.

Für die nachfolgenden Versuche wird ein Datensatz des WebWerdis-Dienstes des Deutschen Wetterdienstes[14] mit stündlichen Außentemperaturwerten der Stadt Hamburg verwendet. Es werden Daten aus einem Zeitraum von zwei Mo-

[4] `http://www.google-melange.com/gsoc/homepage/google/gsoc2013` [Zugriff am 21. März 2014]
[5] `http://coalesenses.de/` [Zugriff am 21. März 2014]
[6] `http://contiki-os.org/` [Zugriff am 21. März 2014]
[7] `http://www.tinyos.net/` [Zugriff am 21. März 2014]
[8] `https://github.com/itm/shawn/` [Zugriff am 21. März 2014]
[9] `https://www.apple.com/de/ios/` [Zugriff am 21. März 2014]
[10] `http://www.cosa.fh-luebeck.de/de/projects/trisos/` [Zugriff am 21. März 2014]
[11] `http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/software/sof_ScatterWeb2.html` [Zugriff am 21. März 2014]
[12] `http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/software/sof_FeuerWare.html` [Zugriff am 21. März 2014]
[13] `http://db.csail.mit.edu/labdata/labdata.html` [Zugriff am 21. März 2014]
[14] `https://werdis.dwd.de/` [Zugriff am 21. März 2014]

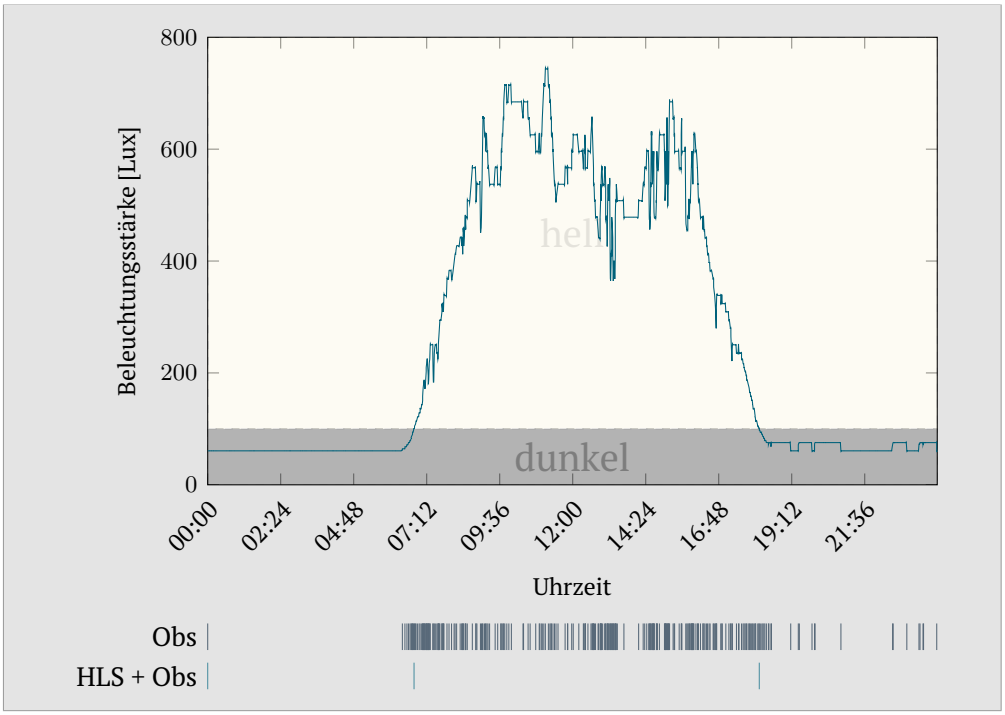


Abbildung 4.24.: Diagramm der rohen Lichtwerte eines Tages. Die farbigen Flächen markieren die Zustände *dunkel* und *hell*. Unter dem Diagramm repräsentiert jede Markierung eine Nachricht, die durch Observe (Obs) bzw. die HLS-Option in Kombination mit Observe (HLS+Obs) für einen Schwellwert von 100 Lux verschickt wird.

Methode	Anzahl Nachrichten
High-Level State + Observe	105
Conditional Observe (CO_0)	64
Conditional Observe (CO_{10})	37
Conditional Observe (CO_{20})	4

Tabelle 4.5.: Anzahl Nachrichten für die in Abbildung 4.26 gezeigten Methode und deren Schwellwerte.

naten gewählt. Abbildung 4.25 zeigt den Temperaturverlauf sowie die zwei Zustände *kalt* und *warm*. Wie in Abbildung 4.24 ist unter dem Diagramm das Verschicken von Benachrichtigungen aufgetragen. Diesmal wird HLS in Kombination mit Observe und Conditional Observe mit dem Condition Type *Value*<> gegenübergestellt. Für beide Methoden wird der Schwellwert auf 10 °C gesetzt. Es ist zu erkennen, dass exakt die gleiche Anzahl an Nachrichten, insgesamt 37, zu jeweils den gleichen Zeitpunkten, verschickt werden. Bei der Nutzung von zwei Zuständen ist somit keine Ersparnis in der Anzahl von Nachrichten möglich. Allerdings hat die Nutzung von HLS den Vorteil, dass man Zustände zurückgeliefert bekommt und somit der Nutzer eine nützliche Semantik erhält.

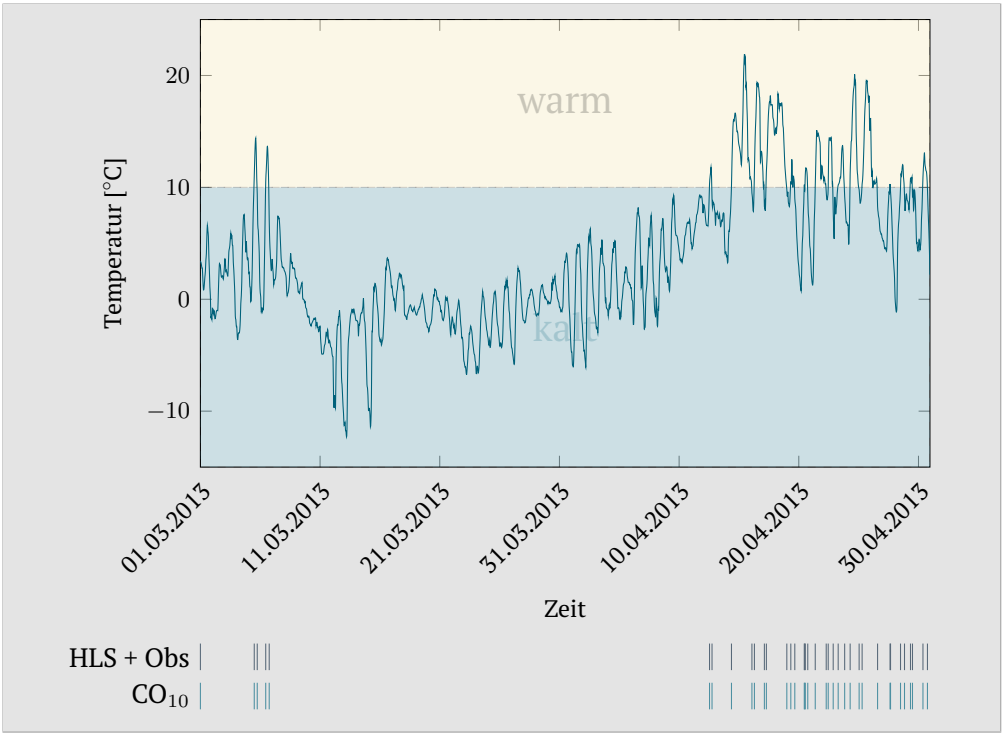


Abbildung 4.25.: Diagramm der rohen Temperaturwerte von zwei Monaten. Die farbigen Flächen markieren die Zustände *kalt* und *warm*. Unter dem Diagramm repräsentiert jede Markierung eine Nachricht, die durch die HLS-Option in Kombination mit Observe bzw. Conditional Observe (CO) für einen Schwellwert von $10\,°C$ (CO_{10}) verschickt wird.

Der nächste Test wird mit denselben Daten jedoch mit einem Zustand mehr durchgeführt. Der zusätzliche Zustand *extrem* ist dabei definiert für Werte unter $0\,°C$ und über $20\,°C$. Abbildung 4.26 zeigt den bekannten Temperaturverlauf sowie die Zustände und die Zeitpunkte der verschickten Nachrichten. Für die Definition der Zustände mittels HLS ist eine Nachricht vom Client erforderlich. Eine weitere Nachricht muss geschickt werden, um das Observe auf die neu erzeugte Ressource zu starten. Bei der Nutzung von Conditional Observe ist dagegen für jeden Zustandsübergang eine separate Nachricht notwendig, in diesem Fall somit insgesamt drei Nachrichten. Zudem müssen diese Nachrichten von unterschiedlichen Adressen bzw. Ports verschickt werden. In Tabelle 4.5 ist die jeweilige Anzahl an verschickten Nachrichten pro (Conditional-)Observe-Beziehung aufgeführt. Die Anzahl von HLS + Observe ist wiederum gleich der Summe aller durch CO verschickten Nachrichten. Allerdings muss der Client die Nachrichten der unterschiedlichen CO-Beziehungen wieder zusammenführen, um so den aktuellen Zustand daraus abzuleiten. Ein weiterer Nachteil wird deutlich, wenn mehrere Clients dieselben Zustände beobachten wollen. Während die HLS-Ressource von mehreren Clients observierbar ist, muss bei CO jeder Client für jeden Schwellwert eine eigene CO-Beziehung initiieren. Dies spiegelt sich auf dem Server in höherem Ressourcenverbrauch wider, da

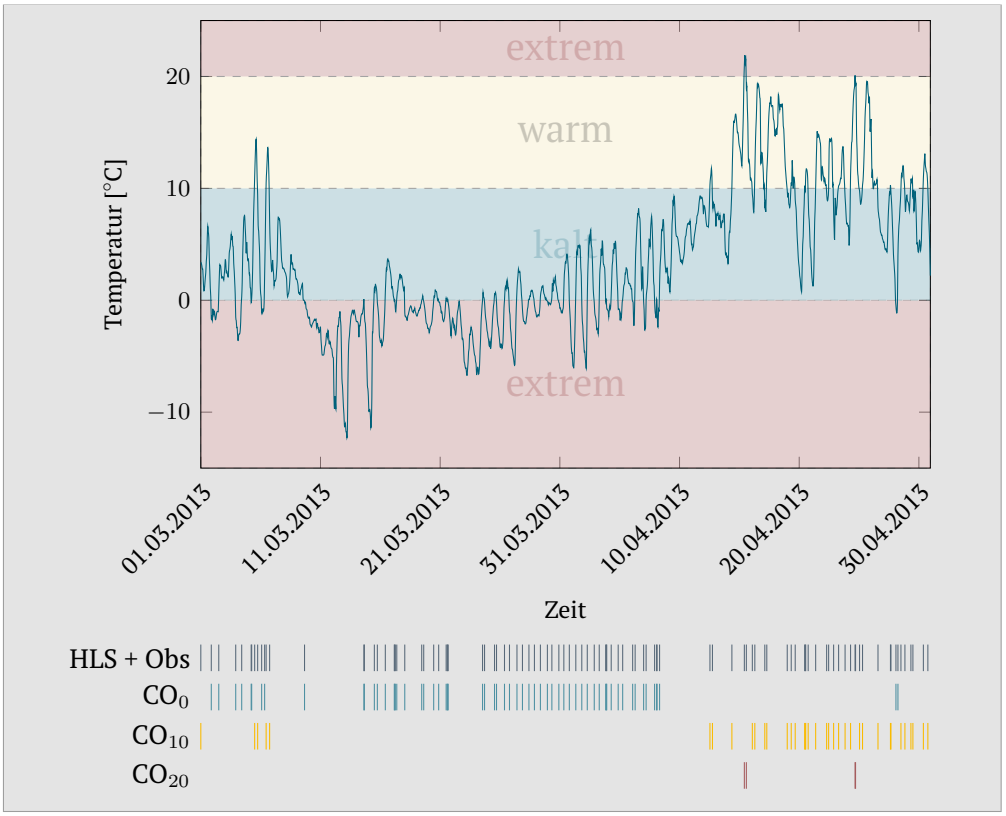


Abbildung 4.26.: Diagramm wie in Abbildung 4.25 mit denselben Daten aber drei Zuständen. Die Schwellwerte der Zustände sind 0 °C (CO_0), 10 °C (CO_{10}) und 20 °C (CO_{20}).

Methode	Anzahl Nachrichten
High-Level State + Observe	65
Conditional Observe (CO_{-10})	5
Conditional Observe (CO_8)	38
Conditional Observe (CO_{10})	4
Conditional Observe (CO_{15})	18
Conditional Observe (CO_{20})	4

Tabelle 4.6.: Anzahl Nachrichten für die in Abbildung 4.27 gezeigten Methoden und deren Schwellwerte.

für n Clients bei HLS + Observe n Observationsbeziehungen verwaltet werden müssen, während für CO $3 \times n$ nötig sind.

Der letzte Test wird wiederum mit denselben Daten durchgeführt. Diesmal werden sechs Zustände definiert. Die Grenzen der Zustände *extrem kalt*, *kalt*, *relativ kalt*, *relativ warm*, *warm* und *heiß* liegen bei 0 °C, 8 °C, 10 °C, 15 °C und 20 °C. In Abbildung 4.27 und Tabelle 4.6 sind die Ergebnisse dieses Versuchs dargestellt. Die Anzahl aller durch CO-Beziehungen verschickten Nachrich-

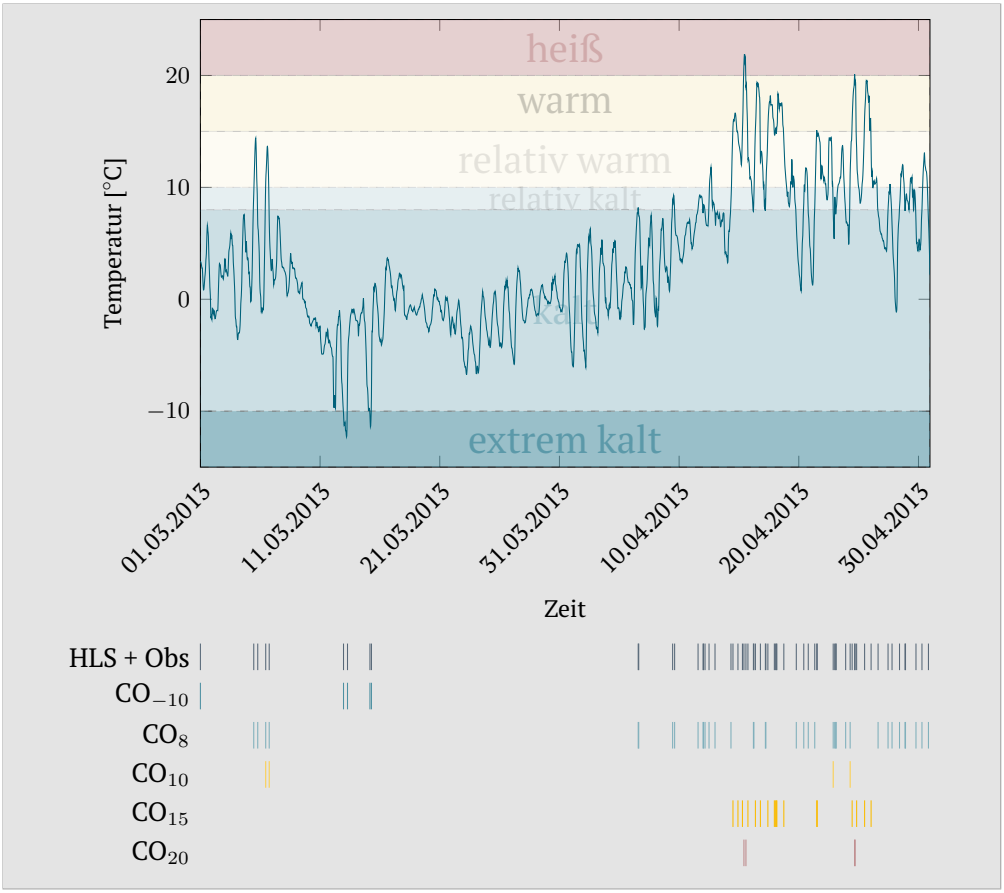


Abbildung 4.27.: Diagramm wie in Abbildung 4.25 mit denselben Daten aber sechs Zuständen. Die Schwellwerte der Zustände sind $-10\,°C$ (CO_{-10}), $8\,°C$ (CO_8), $10\,°C$ (CO_{10}), $15\,°C$ (CO_{15}) und $20\,°C$ (CO_{20}).

ten liegt um vier höher als bei HLS + Observe. Dies lässt sich damit erklären, dass bei einigen aufeinanderfolgenden Werten ein Zustand übersprungen wird (z. B. direkt von *kalt* zu *relativ warm*). Dadurch schickt HLS in Kombination mit Observe nur eine Nachricht, während CO zwei Schwellwert-Nachrichten verschickt, da zwei Schwellwerte überschritten werden und die CO-Beziehungen unabhängig voneinander sind. Somit kann die Anzahl der von HLS + Observe versendeten Nachrichten sogar geringer sein als die aller CO-Beziehungen zusammengenommen.

Fazit

Die HLS-Option ist eine Erweiterung für CoAP, die ein einfaches und flexibles Erstellen neuer Zustandsressourcen erlaubt. Diese Ressourcen, welche die Werte von Sensorressourcen nutzen, um daraus, mittels Abbildung, Zustände zu ermitteln, lassen sich anschließend durch Clients abfragen oder mit Hilfe von CoAP Observe beobachten. Die Evaluation hat gezeigt, dass im Gegensatz zu CoAP Observe eine große Zahl an Nachrichten gespart werden kann. Zusätz-

lich wird den Sensorwerten eine Semantik gegeben. Im Gegensatz zu CoAP Conditional Observe bietet die HLS-Option den Vorteil, dass die Erzeugung und Verwaltung mehrerer Zustände sowohl für den Client als auch den Server weniger Aufwand bedeutet. Unter bestimmten Voraussetzungen lassen sich mit HLS in Kombination mit CoAP Observe sogar Nachrichten gegenüber CoAP Conditional Observe sparen.

4.2.3. Skriptbasierte Zustandsermittlung

CoAP und das darunter liegende Kommunikationsprotokoll 6LoWPAN setzen sich erst langsam im Bereich von Sensorknoten durch. Dies liegt z. T. an den geringen verfügbaren Ressourcen und andererseits an den über lange Zeit entwickelten, hochspezialisierten und effizienten proprietären Protokollen, die eine lange Batterielaufzeit ermöglichen.

Um auch auf Sensorknoten, die CoAP nicht unterstützen, eine Zustandsermittlung zu ermöglichen, wird ein skriptbasiertes System entwickelt und für die iSense-Plattform implementiert. Dabei schreibt ein Nutzer ein individuelles Skript, welches die Datenerfassung und die Datenreduktion beschreibt. Dieses wird von einer Serverkomponente optimiert, serialisiert und in das WSN geschickt. Die Sensorknoten deserialisieren das Skript und ein Interpreter führt die entsprechenden Anweisungen aus. Teile dieser Arbeit wurden vorab in [27] veröffentlicht.

Skriptaufbau

Das Skript wird in XML geschrieben und besteht aus zwei Hauptelementen. Der erste Teil definiert, wann, wie und welche Daten gemessen und gespeichert werden sollen. Der zweite Part beschreibt die Reduzierung der Detailstufe von Daten. Dieser Teil lässt sich nicht nur zur Verringerung von Übertragungsvolumen nutzen, sondern auch zur Ermittlung von Zuständen und weiterer Statistiken über die gemessenen Sensorwerte. Beide Teile werden in den nächsten Abschnitten genauer vorgestellt.

Datenerfassung Der erste Teil des Skriptes beinhaltet verschiedene Bestandteile, die nachfolgend erläutert werden. Durch die Definition dieser Elemente wird beschrieben, wann, wie oft, wie und welche Sensordaten erfasst und ggf. gespeichert bzw. verschickt werden sollen:

Scope Ein *scope* definiert eine Menge von Sensorknoten. Dies geschieht entweder über die explizite Aufzählung von Knotenadressen oder über Aliase. Es existieren die Aliase *broadcast*, *server* sowie jeweils einer für jeden vorhandenen Sensortyp des WSNs. Der Bezeichner *broadcast* ist die Menge aller Sensorknoten, während *server* die Senke bezeichnet, welche die Nachricht in das Backend weiterleitet. Gibt man einen Sensortyp als Alias

an, so umfasst die Menge der Sensorknoten all diejenigen, die mit einem Sensor diesen Typs ausgestattet sind.

Event Ein *event* ist ein Ereignis, das innerhalb des Sensornetzes detektierbar ist oder ein externes Ereignis, welches vom Backend in das Sensornetz gemeldet wird. Es lassen sich beliebige externe Ereignisse definieren, deren Auftreten dem Backend von außen mitgeteilt werden kann. Dieses übersetzt das Ereignis in einen Identifikator, der dafür im WSN genutzt wird. Interne Ereignisse sind bestimmte Messungen an Sensoren, wie *Temperatursensor misst mehr als* $20\,°$C.

Behavior Ein *behavior* beschreibt das Verhalten eines Sensors. Das Standardverhalten, das Messen eines Wertes mit dem Sensor, wird immer ausgeführt, wenn ein *behavior* gestartet wird. Optional lässt sich auch das Loggen in den Flashspeicher und das Versenden der Messwerte starten. Die Zeitintervalle für die drei Aktionen sind individuell festlegbar. Beispielsweise kann alle zehn Sekunden ein Messwert genommen und gespeichert werden, aber nur alle fünf Minuten werden die Sensorwerte gebündelt verschickt.

Record Ein *record* beschreibt die Größe eines Speicherbereichs, der für das Loggen von Sensordaten zur Verfügung steht. Häufen sich mehr Daten an als Speicher zur Verfügung steht, so werden die ältesten Daten wie bei einem Ringpuffer überschrieben . Über einen Identifikator wird ein *record* in einem *behavior* referenziert. Weiterhin können Daten über den Identifikator abgerufen werden.

Rule Eine *rule* ist eine Regel, die eine Anfangs- und Endbedingung haben kann. Solange eine Regel aktiv ist, d. h. ihre Bedingungen zutreffen, wird das von der Regel referenzierte *behavior* ausgeführt. Bedingungen sind boolesche Formel über *events*. Erlaubt sind die booleschen Operatoren *und*, *oder* und *Ereignis* x_i *folgt auf* x_j. Da die definierbaren Ereignisse diskrete Zeitpunkte besitzen, muss für die Operatoren *und* und *Ereignis* x_i *folgt auf Ereignis* x_j eine Zeitspanne angegeben werden, in der alle *events* eintreten müssen, damit die Bedingung erfüllt ist. Weiterhin lässt sich spezifizieren, ob eine Regel nur zu bestimmten Zeiten am Tag aktiv sein soll. Sie wird dann nur in diesem Zeitraum ausgewertet.

Ruleset Ein *ruleset* ist eine Menge von *rules*, die für einen *scope* ausgewertet wird, d. h. alle Ereignisse, die innerhalb dieses *scopes* auftreten, werden bei der Auswertung von Regel berücksichtigt. Somit kann die Regel x_i *und* x_j erfüllt sein, wenn ein *scope* die Knoten 1 und 2 beinhaltet und Ereignis x_i am Knoten 1 und Ereignis x_j am Knoten 2 eintritt.

Datendetailierungsgradanpassung Daten, die durch ein *behavior* in den Flashspeicher geloggt werden, haben die Form von Tupeln, die den Messzeitpunkt und den gemessenen Sensorwert beinhalten. Eine Menge dieser Tupel lässt sich als Tabelle mit zwei Spalten betrachten, wobei die erste Spalte alle Mess-

zeitpunkte beinhaltet und die zweite die Messwerte. Jedes Tupel wird somit durch eine Zeile repräsentiert.

Auf diese Tabelle lassen sich verschiedene Funktionen anwenden, welche die Struktur bzw. den Inhalt modifizieren. Dadurch kann u. a. auch der Detailierungsgrad (LoD) angepasst werden. Durch eine Reduzierung des LoD wird die Datenmenge reduziert, so dass Ressourcen bei der Übertragung dieser Daten ins Backend eingespart werden. Allerdings wird diese Ersparnis mit geringerer Aussagekraft der Daten erkauft. Andererseits gibt es Situationen in denen dieser Trade-Off Vorteile bietet. Wie in den vorherigen Abschnitten beschrieben, ist eine Zustandsermittlung immer mit dem Verlust von Informationen verbunden, was aber kein Nachteil ist. Die Zustandsermittlung mit der Anpassung des LoD ist somit ein Anwendungsfall, bei dem Ressourcen gespart werden, der Verlust an Informationen jedoch keinen negativen Effekt hat.

Nachfolgend werden die verfügbaren Modifikationsfunktionen für die Tabelle beschrieben. Anschließend wird exemplarisch gezeigt, wie diese Funktionen, u. a. für die Ermittlung von Zuständen, nutzbar sind.

Original Die *Original*-Funktion modifiziert die Daten der Tabelle nicht. Die Funktion muss genutzt werden, wenn die Daten unverändert angefordert werden sollen.

Delete Die *Delete*-Funktion hat als Parameter einen Spaltennamen. Diese Spalte, sofern vorhanden, wird aus der Tabelle gelöscht.

Count Die *Count*-Funktion ist nur auf eine Tabelle mit einer Spalte anwendbar. Sie zählt die Anzahl an Zeilen, löscht die Tabelle und erzeugt eine 1×1 Tabelle, die nur die Anzahl der Zeilen beinhaltet.

Add Die *Add*-Funktion ist ebenfalls nur auf Tabellen mit einer Spalte anwendbar. Sie addiert die Werte dieser Spalte auf, löscht die Tabelle und erzeugt eine 1×1 Tabelle mit der Summe der Werte.

Histogram Auch die *Histogram*-Funktion ist nur mit einspaltigen Tabellen nutzbar. Es entsteht eine neue Tabelle mit zwei Spalten. Die erste Spalte enthält alle Werte, die in der ursprünglichen Spalte der Tabelle vorkamen, während die zweite Spalte die Häufigkeit des Vorkommens des Wertes in der ursprünglichen Tabelle auflistet.

Precision Die *Precision*-Funktion besitzt drei Parameter: einen Spaltennamen, eine Zahl und eine Richtung. Die Zahl ist die Anzahl der Bits, die für die Werte in der angegebenen Spalte zur Speicherung genutzt werden sollen. Dies geschieht nur, sofern diese kleiner ist als die aktuelle verwendete Anzahl an Bits. Andernfalls wird nichts geändert. Dabei werden, je nach angegebener Richtung, beginnend vom höherwertigsten bzw. niederwertigsten Bit, die Bits auf 0 gesetzt. Ist von Beginn an bekannt, dass nur kleine Zahlen in der Tabelle entstehen, lässt sich durch Löschen der höherwertigen Bits die Anzahl benötigter Bits reduzieren, ohne einen Informationsverlust in Kauf nehmen zu müssen. Auch das Löschen, und

somit Nichtübertragen, niederwertiger Bits reduziert Kommunikationsaufwand. Durch Auffüllen der niederwertigen Bits mit Nullen im Backend wird die ursprüngliche Größenordnung der Zahl wiederhergestellt. Dies geht aber mit einem Informationsverlust einher, welcher durch einen Fehlerintervall charakterisierbar ist.

Sort Die *Sort*-Funktion hat als Parameter einen Spaltennamen und die Sortierreihenfolge (aufsteigend/absteigend). Die angegebene Spalte wird gemäß des zweiten Parameters sortiert. Die Werte der anderen Spalten werden entsprechend mitsortiert.

Aggregation Die *Aggregation*-Funktion ist nur auf Tabellen mit einer Spalte anwendbar. Sie erlaubt es eine Aggregation über die Spalte auszuführen. Dazu stehen verschiedene Aggregationen zur Verfügung: Bei Anwendung der Minimums-, Medians-, Durchschnitts- und Maximumsbildung wird eine 1×1 Tabelle mit dem entsprechenden Wert erzeugt. Die *Range*-Aggregation erzeugt ebenfalls eine 1×1 Tabelle. Als Wert enthält diese die Differenz des Maximums und Minimums. Bei *MinMedianMax* entsteht eine 1×3 Tabelle mit dem Minimum, Median und Maximum. Weiterhin kann ein Perzentil angegeben werden, welches eine 1×1 Tabelle zurückgibt.

Filter Die *Filter*-Funktion hat als Parameter einen Spaltennamen und die zu verwendende Filterfunktion. Alle Zeilen, deren Werte in der angegeben Spalte die Filterfunktion erfüllen, bleiben erhalten. Die restlichen werden gelöscht. Als Filterfunktionen stehen Vergleichsoperatoren ($<, >, =, !=$) sowie die Funktionen *topK*, *bottomK* und *everyK*, welche die ersten K, letzten K bzw. jede $K.$ Zeile erhalten, zur Verfügung.

Map Die *Map*-Funktion erlaubt das Abbilden von Wertintervallen auf andere Werte und setzt damit die Idee der Intervall-Abbildung um, auch wenn zunächst kein Zustand, sondern nur ein neuer Zahlenwert entsteht. Dieser kann an der Basisstation als Zustand interpretiert werden. Für die Abbildung werden der Funktion eine Spalte, eine Menge von Intervallen, definiert durch *u* und *o*, und ein neuer Wert für jedes Intervall übergeben.

Ein *Level* wird definiert durch eine oder mehrere Funktionen, die in der angegebenen Reihenfolge auf die Tabelle angewendet werden. Dabei ist die Ausgabe einer Funktion die Eingabe für die nächste. Es lassen sich mehrere *Level* definieren. Auch hier gilt, dass die Ausgabe eines *Levels* die Eingabe für das nächste *Level* ist. Jedes *Level* entspricht dabei einem LoD. So können für einen Datensatz verschiedene LoDs definiert und die Ergebnisse der Anwendungen dieser LoDs auf die Daten durch einen Client abgerufen werden.

Zustandsermittlung und Statistikgenerierung Mit Hilfe von LoD-Skripten lassen sich auch, aus im Flash gespeicherten Sensorwerten, Zustände ermitteln.

Ein Beispiel für ein LoD-Skript ist in Quelltext 4.3 gegeben. Es besteht aus vier *Leveln* (Zeilen 3–5, 6–22, 23–27 und 28–30). Wird der Datensatz aus dem *record*

Zeitstempel (32)	Sensorwert (32)
1388534400	10
1388538000	10
1388541600	11
...	...
1388563200	18
1388566800	20
...	...
1388588400	22
1388592000	22
...	...
1388613600	11
1388617200	11

Tabelle 4.7.: Stündliche Temperaturwerte für einen Tag. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 4 aus Quelltext 4.3.

mit der Id 1 (siehe Zeile 2) auf LoD 4 abgerufen, so wird der originale Datensatz versendet. *Level* 3 wendet auf die Sensorwerte die *map*-Funktion an. Alle Werte < 20 werden auf eine 0 und alle anderen Werte auf eine 1 abgebildet. Angenommen die Sensorwerte werden von einem Temperatursensor gemessen. Da der Client das LoD-Skript kennt, kann er rekonstruieren, zu welchen Zeitpunkten der Sensor den Zustand *kalt* (Werte < 20) und *warm* (Werte ≥ 20) gemessen hat. Nach Anwendung der *map*-Funktion wird noch die Anzahl der Bits zum Speichern, beginnend mit dem höchstwertigen, auf 1 reduziert. Dies ist möglich, da bekannt ist, dass nur die Zahlen 0 und 1 entstehen können und diese mit einem Bit kodierbar sind. Dies spart Ressourcen bei der Übertragung der Zustände. Das nächste *Level*, Nummer 2, löscht die Zeitstempel, so dass zwar nicht mehr die Zeitpunkte der Zustände bekannt sind, aber die Reihenfolge in der diese gemessen wurden. Auf dem letzten definierten *Level* wird nur noch ein Histogramm an den Client geschickt, d. h. die Häufigkeit der Zustände *kalt* und *warm*.

Mit den zur Verfügung stehenden Funktionen lassen sich somit nicht nur Zustände aus den Sensorwerten ermitteln, sondern auch viele weitere Statistiken erzeugen. In Abschnitt 4.3 wird diskutiert, wie Wahrscheinlichkeitsmodelle zur Ressourcenersparnis und Effizienzsteigerung eingesetzt werden können. Das Beispiel aus Quelltext 4.3 zeigt, wie man einige dieser Informationen mit einfachen LoD-Skripten erzeugen kann, um diese dann vom Sensor abzurufen. Eine detailliertere Diskussion zur Erzeugung von Wahrscheinlichkeitsmodellen mittels Skript ist in Abschnitt 4.3.3 enthalten.

```
<compression>
 <levels recordId="1">
  <level number="4">
   <original/>
  </level>
  <level number="3">
   <map>
    <column>value</column>
    <mappings>
     <map>
      <lowerBound>-inf</lowerBound>
      <upperBound>20</upperBound>
      <newValue>0</newValue>
     </map>
     <map>
      <lowerBound>20</lowerBound>
      <upperBound>inf</upperBound>
      <newValue>1</newValue>
     </map>
    </mappings>
   </map>
   <precision>
    <column>value</column>
    <bits>1</bits>
    <startWithMsb>true</startWithMsb>
   </precision>
  </level>
  <level number="2">
   <delete>
    <column>timestamp</column>
   </delete>
  </level>
  <level number="1">
   <histogram/>
  </level>
 </levels>
</compression>
```

Quelltext 4.3: LoD-Skriptfragment.

Evaluation

Die Evaluation soll zeigen, wie viele Daten bei der Übertragung nach Anwendung verschiedener *Levels* gespart werden können. In Tabelle 4.7 sind exemplarisch die stündlich geloggten Werte eines Temperatursensors über einen Tag dargestellt. Die Tabellen 4.8, 4.9 und 4.10 zeigen, wie sich die Daten durch Anwendung der Funktionen der *Levels* aus Quelltext 4.3 verändern. In Tabelle 4.11 ist jeweils die Anzahl an Bytes angegeben, die zur Serialisierung benötigt wird. Es ist zu erkennen, dass die Anzahl an Bytes stark reduziert werden kann. Vergleicht man *Level* 2 und 1 ist aber ebenso zu sehen, dass sich die Datenmenge nicht zwingend mit jedem *Level* verringert, sondern, im Gegenteil, sogar wieder steigen kann.

Zeitstempel (32)	Sensorwert (1)
1388534400	0
1388538000	0
1388541600	0
...	...
1388563200	0
1388566800	1
...	...
1388588400	1
1388592000	1
...	...
1388613600	0
1388617200	0

Tabelle 4.8.: Stündliche Temperaturzustände (0 $\widehat{=}$ *kalt*, 1 $\widehat{=}$ *warm*) für einen Tag. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 3 aus Quelltext 4.3.

Sensorwert (1)
0
0
0
...
0
1
...
1
1
...
0
0

Tabelle 4.9.: 24 Zustände (0 $\widehat{=}$ *kalt*, 1 $\widehat{=}$ *warm*) ohne zeitlichen Bezug. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahl in Klammern hinter dem Spaltennamen gibt die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 2 aus Quelltext 4.3.

Man nehme an, dass b_{ts} und b_v die Anzahl Bits sind, die für einen Wert der Spalte für den Zeitstempel bzw. den Sensorwert benötigt werden. Die Variablen c, $c_{distinct}$ und $c_{filtered}$ sind die Anzahl der Zeilen, die Anzahl der verschiedenen Zeilen bzw. die Anzahl der durch Filterung übrigbleibenden Zeilen der Tabelle. Dann ist die Zahl der Bits T_{pre}, die zur Serialisierung der Tabelle benötigt werden, definiert als:

Sensorwert (1)	Anzahl (32)
0	16
1	8

Tabelle 4.10.: Histogramm für Zustände (0 $\widehat{=}$ *kalt*, 1 $\widehat{=}$ *warm*). Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 1 aus Quelltext 4.3.

Level	Bits	Bytes	Ersparnis
4	$24 \times 32 + 24 \times 32 = 1536$	192	–
3	$24 \times 32 + 24 \times 1 = 792$	99	48,44 %
2	$24 \times 1 = 24$	3	98,44 %
1	$2 \times 1 + 2 \times 32 = 66$	8,25	95,70 %

Tabelle 4.11.: Anzahl Bits, die zur Serialisierung der reinen Werte notwendig sind (ohne Overhead). Die Ersparnis bezieht sich auf die Originaldaten, d. h. *Level* 4.

$$T_{pre} = (b_{ts} + b_v) * c$$

$T_{Funktionsname}$ bezeichnet die Anzahl an Bits, die nach der Anwendung der Funktion zur Serialisierung benötigt werden. Somit gelten folgende Gleichungen:

$$T_{original} = T_{sort} = T_{map} = T_{pre}$$

$$T_{count} = T_{add} = 32$$

$$T_{histogram} = (b_v + 32) * c_{distinct}$$

$$T_{delete_{ts}} = b_v * c \text{ und } T_{delete_v} = b_{ts} * c$$

$$T_{precision_{ts}} = (b_{ts}^{new} + b_v) * c \text{ und } T_{precision_v} = (b_{ts} + b_v^{new}) * c$$

$$T_{aggregation_{MinMaxMedian}} = 96 \text{ und } T_{aggregation_{other}} = 32$$

$$T_{filter} = (b_{ts} + b_v) * c_{filtered}$$

Bei der Serialisierung fällt kein zusätzlicher Overhead an. Der Client fordert Daten eines bestimmten *records* auf einem bestimmten *Level* an. Diese Anfrage

hat eine Nachrichten-Id, die in der Antwort wiederverwendet wird. Dadurch weiß der Client, welche Daten in der Antwort zu erwarten sind. Da der Client ebenfalls das LoD-Skript und das *Level* kennt, weiß dieser automatisch, wie die Daten aufgebaut sind, d. h. wie viele Spalten enthalten sind und mit wie vielen Bits diese serialisiert wurden. Die Anzahl der Zeilen ist entweder explizit durch das LoD-Skript bekannt (z. B. wenn als letztes eine Minimumsbildung stattgefunden hat) oder sie kann implizit berechnet werden, da die Bits pro Spalte und die Gesamtzahl an Bits bekannt sind.

Fazit

Die skriptbasierte Erfassung, Speicherung, Datenreduktion und Abfrage von Sensordaten ist ein flexibler Mechanismus, der für die Realisierung verschiedener Aufgaben einsetzbar ist. Es wurde gezeigt, wie aus rohen Sensorwerten Zustände ermittelt werden können. Weiterhin ist es möglich, verschiedene statistische Werte, wie die Häufigkeit, Extremwerte, Mittelwert usw. zu erzeugen. Die Evaluation zeigt, dass der Nutzer durch ein Skript feingranular den Trade-Off zwischen Datenersparnis und Informationsgehalt steuern kann. Dies erlaubt somit, Energie bei der Übertragung zu sparen, falls der Nutzer dafür bereit ist die Daten nicht in voller Auflösung zu erhalten.

4.2.4. Browser-Erweiterung zur Erzeugung virtueller Sensoren

Eine Vielzahl von Sensordaten ist entweder gar nicht oder nicht direkt zugänglich. Stattdessen senden die Sensorknoten ihre Messwerte an einen Server, der diese für interne oder auch öffentliche Zwecke speichert und bereitstellt. Dies hat für den Betreiber vor allem den Vorteil, dass Sensorknoten nicht von zahlreichen Clients abgefragt werden bzw. ihre Daten an diese schicken müssen, sondern dass sie nur mit dem eigenen Server kommunizieren. Dies kann den Ressourcenverbrauch gering halten. Allerdings können für solche Sensorknoten nicht die vorgestellten Verfahren aus Abschnitt 4.2.2 und Abschnitt 4.2.3 genutzt werden, sondern es müssen andere Mechanismen zum Zugriff und zur Zustandsermittlung entwickelt werden.

Eine häufig genutzte Methode zum Bereitstellen der Sensordaten ist das Veröffentlichen auf Webseiten. Beispiele dafür sind Wetterportale[15] und Parkplatzinformationsseiten[16]. Im Kontrast dazu gibt es Webseiten, die dynamische Informationen enthalten, welche aber nicht von Sensoren stammen, wie z. B. der Kantinenspeiseplan[17] oder die Lagerbestände auf einer Shoppingwebseite. Aus diesen können sich ebenfalls Zustände wie *Lieblingsessen* oder *Artikel wieder verfügbar* ableiten lassen.

[15] z. B. `http://www.wetter.de/` [Zugriff am 21. März 2014]
[16] z. B. `http://www.parken-luebeck.de/` [Zugriff am 21. März 2014]
[17] z. B. `http://www.studentenwerk-s-h.de/seiten_essen/plan_mensa_luebeck.html` [Zugriff am 21. März 2014]

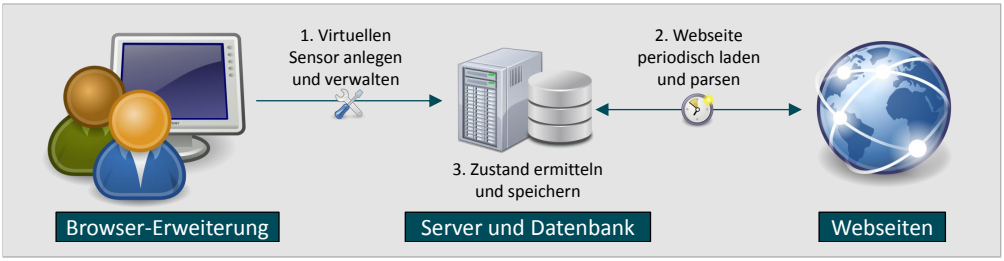


Abbildung 4.28.: Architektur zur Erzeugung virtueller Sensoren.

In diesem Abschnitt wird ein Verfahren vorgestellt, das es erlaubt *virtuelle Sensoren* zu erzeugen. Dies sind keine realen Sensoren, sondern ein Konzept, das ähnlich wie reale Sensoren Messwerte generiert. Diese Werte sind durch periodische Extraktion von Textfragmenten aus Webseiten erhebbar. Daraus wird in einem Nachverarbeitungsschritt ein Zustand ermittelt, der gespeichert und weiterverwendet werden kann.

Architektur

Die Architektur für virtuelle Sensoren ist in Abbildung 4.28 zu sehen. Sie besteht aus zwei Komponenten. Auf Nutzerseite wird eine Browser-Erweiterung installiert und genutzt. Diese erlaubt das Erstellen und Löschen von virtuellen Sensoren. Da Nutzer üblicherweise ihren Browser nicht dauerhaft geöffnet haben, gibt es eine Server-Komponente, welche die virtuellen Sensoren verwaltet und die eigentliche Datenextraktion, Zustandsermittlung und die Speicherung der Zustände übernimmt.

Browser-Erweiterung Die Browser-Erweiterung erlaubt die Erzeugung virtueller Sensoren. Dafür sind verschiedene Informationen erforderlich, die teils manuell, teils automatisch erhoben werden.

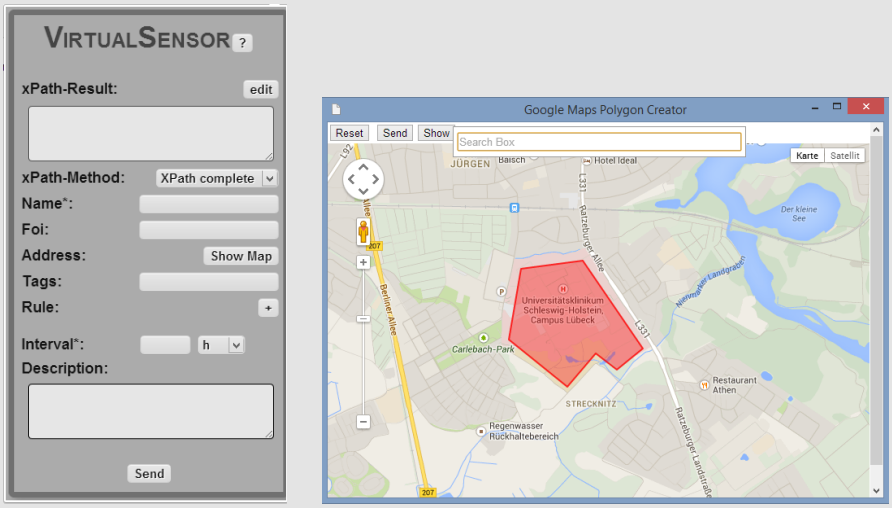
Sobald der Nutzer eine Webseite aufgerufen hat und für diese einen virtuellen Sensor erzeugen möchte, markiert er den zu extrahierenden Textteil und startet die Erweiterung. Die GUI der Erweiterung, die in Abbildung 4.29a zu sehen ist, erlaubt dem Nutzer folgende Metainformationen über den virtuellen Sensor einzugeben:

Name Der Name des virtuellen Sensors.

Feature of Interest Die beobachtete Entität, z. B. das Wetter, ein bestimmter Raum oder ein Parkplatz.

Location Eine textuelle Beschreibung des Ortes des Sensors. Bei einer Wetterseite kann dies z. B. die Stadt sein, für die das Wetter gilt.

Tags Beliebig viele beschreibende Schlüsselwörter, die für eine spätere Suche nutzbar sind.



(a) Hauptmaske. **(b)** Kartenmaske.

Abbildung 4.29.: Ausschnitte der GUI zur Erstellung eines virtuellen Sensors.

Interval Das Zeitintervall, nach welchem ein Zustand für den virtuellen Sensor ermittelt werden soll. In diesem Intervall wird periodisch die Textstelle extrahiert, daraus ein Zustand ermittelt und gespeichert. Es kann in Stunden, Minuten und Sekunden definiert werden.

Description Eine textuelle Beschreibung des Sensors.

Auf einer Karte des Dienstes Google Maps (siehe Abbildung 4.29b) kann der Nutzer den Ort des virtuellen Sensors noch präziser mittels eines einzuzeichnenden Polygons spezifizieren.

Zum Extrahieren der markierten Textstelle stehen die Methoden *XPath complete*, *XPath relative* und *Position* zur Verfügung. Bei *XPath complete* und *XPath relative* wird durch die Erweiterung automatisch ein kompletter bzw. relativer XML Path Language-Ausdruck (XPath-Ausdruck) generiert, der zum Wiederfinden der Textstelle dient. Die Methode *Position* nutzt die Position im HTML-Quelltext, d. h. es werden die Stellen des Start- und Endzeichens der Markierung verwendet. Standardmäßig wird zur Extraktion die Methode *XPath complete* genutzt, aber der Nutzer hat die Möglichkeit manuell eine andere Methode auszuwählen.

Der Nutzer kann ferner eine beliebige Anzahl an Regeln zur Zustandsermittlung definieren. Dabei stehen wieder die zwei bekannten Regeltypen aus Abschnitt 4.2 zur Verfügung. Die Syntax der Intervall-Abbildung ist *map:u:o->z*, wobei *map* die genutzte Funktion ist. Die übrigen Parameter sind wie zuvor belegt. Der zweite Regeltyp lehnt sich an die String-Abbildung an. Statt aber nur Strings auf einen Zustand abzubilden, kann der Nutzer auch reguläre Aus-

```
map:-inf:10->kalt
map:10:20->moderat
map:20:+inf->warm
```

Quelltext 4.4: Beispiel von drei *map*-Regeln.

```
regex:.*(Pommes|Schnitzel).*->frittiert
regex:.*(Salat).*->Rohkost
```

Quelltext 4.5: Beispiel von zwei *regex*-Regeln.

drücke, wie sie in Java verwendet werden[18], nutzen. Die Syntax dieses Regeltyps ist *regex:reg_expr->z*. *regex* signalisiert den Regeltyp, *reg_expr* ist der Reguläre Ausdruck und *z* wiederum der Zustand der angenommen wird, wenn der extrahierte Text das Muster des Regulären Ausdrucks erfüllt. Es können dennoch weiterhin einfache Strings zur Abbildung genutzt werden. Reguläre Ausdrücke erlauben eine wesentlich flexiblere und mächtigere Zustandsermittlung im Vergleich zur einfachen String-Abbildung. Die Verwendung und Auswertung von Regulären Ausdrücken ist möglich, weil der Serverkomponente im Vergleich zu Sensorknoten erheblich mehr Ressourcen zur Verfügung stehen. Wie in Abschnitt 4.2.1 werden die Regeln nacheinander evaluiert, bis die erste Regel zutrifft. Der für die Regel angegebene Zustand wird dann als momentaner Zustand des virtuellen Sensors genutzt.

In Quelltext 4.4 ist ein Beispiel für *map*-Regeln gegeben, die extrahierte Temperaturwerte einer Wetterseite in einen Zustand umwandeln. Die Werte *-inf* und *+inf* sind Aliase für $-\infty$ und $+\infty$. Der Quelltext 4.5 zeigt ein Beispiel der Anwendung von *regex*-Regeln. Dabei wird der Text eines Speiseplans auf Schlüsselwörter untersucht und ein Zustand ermittelt, der etwas über die Zubereitungsart aussagt. Sobald die notwendigen Angaben eingetragen sind, kann der Nutzer den virtuellen Sensor speichern. Die Parameter für den Sensor werden dadurch an die Serverkomponente geschickt, die alle weiteren Operationen ausführt. Das System erlaubt weiterhin, die bereits angelegten virtuellen Sensoren wieder zu löschen, falls diese nicht mehr benötigt werden.

Serverkomponente Die Serverkomponente verwaltet die virtuellen Sensoren aller Nutzer. Dazu werden die von der Erweiterung gesendeten Daten gespeichert. Nach Ablauf des für den virtuellen Sensor definierten Zeitintervalls wird die dazugehörige Webseite heruntergeladen. Anschließend wird die gewählte Extraktionsmethode ausgeführt, um den Textabschnitt auszulesen. Auf diesen werden die Zustandsregeln angewendet, um einen Zustand zu ermitteln. Ist dies erfolgreich, wird der Zustand zusammen mit der aktuellen Zeit gespeichert. Tritt ein Fehler auf, wird eine entsprechende Fehlermeldung gespeichert und bei der nächsten Gelegenheit dem Nutzer präsentiert, so dass dieser ggf.

[18] `http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html` [Zugriff am 21. März 2014]

Abbildung 4.30.: Evaluationseinstellungen für einen virtuellen Sensor für die Webseite `http://www.wetter.com/`.

Gegenmaßnahmen wie eine Anpassung der Extraktionsmethode vornehmen kann. Optional kann die Serverkomponente aus den vorhandenen Metadaten RDF-Daten nach der Struktur aus Kapitel 5 erzeugen und in einem Triplestore, wie demjenigen aus Kapitel 6 speichern.

Evaluation

Um die korrekte Funktionsweise der Erweiterung zu prüfen, wird über einen Zeitraum von etwa acht Stunden ein virtueller Sensor für das Wetter in Lübeck für die Webseite von `http://www.wetter.com` angelegt. Die Konfiguration ist Abbildung 4.30 zu entnehmen. Es ist zu erkennen, dass ein relativer XPath verwendet wird, um periodisch alle zehn Minuten aus der Temperatur, die auf der Webseite veröffentlicht wird, einen von vier Zuständen zu ermitteln.

Abbildung 4.31 zeigt die von der Webseite extrahierten Temperaturen, welche realistische Daten wiedergeben, auch wenn es keine manuelle Verifikation der Daten durch Protokollieren der Webseite gab.

Weitere Tests zeigen allerdings auch Nachteile der Implementierung auf. So kann die Serverkomponente selber keine Inhalte, die per Asynchronous JavaScript and XML (Ajax) nachgeladen werden, verarbeiten. Somit lassen sich auch keine Informationen, die aus diesen nachgeladenen Daten extrahiert werden sollen, nutzen.

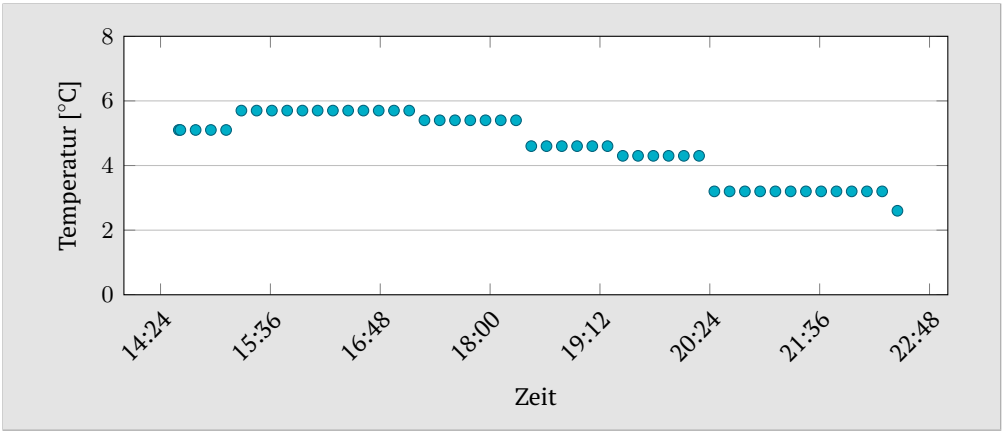


Abbildung 4.31.: Diagramm der durch die Erweiterung extrahierten Temperaturwerte.

Sobald sich die Struktur oder Bezeichner der Elemente einer Webseite ändern, kann es passieren, dass virtuelle Sensoren für diese Seite nicht mehr funktionieren, da der XPath nicht mehr korrekt funktioniert. Bei der Extraktionsmethode *Position* kann dies schon passieren, sobald sich nur der Inhalt der Seite ändert. Durch das Präsentieren einer Fehlermeldung erfährt der Nutzer von der Problematik und kann Gegenmaßnahmen ergreifen.

Fazit

Die vorgestellte Erweiterung erlaubt, durch Markieren eines Textfragments auf einer Webseite, der Eingabe einiger Metainformationen sowie Regeln zur Zustandsermittlung und der Markierung der geografischen Region das Erstellen eines virtuellen Sensors aus dem Textfragment. Der Zustand des Sensors wird durch die Serveranwendung periodisch durch Herunterladen der Webseite, Extrahieren des Textfragments und Anwenden der Regeln ermittelt und gespeichert. XPath hat sich dabei als stabilste Methode der drei unterstützten bewährt, da es nicht nur die absolute Position im Dokument betrachtet, sondern zusätzlich die Struktur mit einbezieht und relativ zu dieser das Fragment sucht. Mit der Erweiterung lassen sich somit auf einfache und flexible Art Informationen aus Webseiten, die den Zustand der realen Welt repräsentieren, so verarbeiten, dass man Daten erhält, die wie Daten von realen Sensoren aussehen.

4.2.5. Fazit

Abbildung 4.32 veranschaulicht die Unterschiede und Gemeinsamkeiten der vier vorgestellten Technologien zur Zustandsermittlung. Diese erlauben es, mit kamerabestückten Sensorknoten, Sensorknoten mit Sensoren, die skalare Werte oder Strings ausgeben und aus Webseiten Zustände der realen Welt zu ermitteln. Boolesche Werte lassen sich ebenso nutzen, da diese entweder

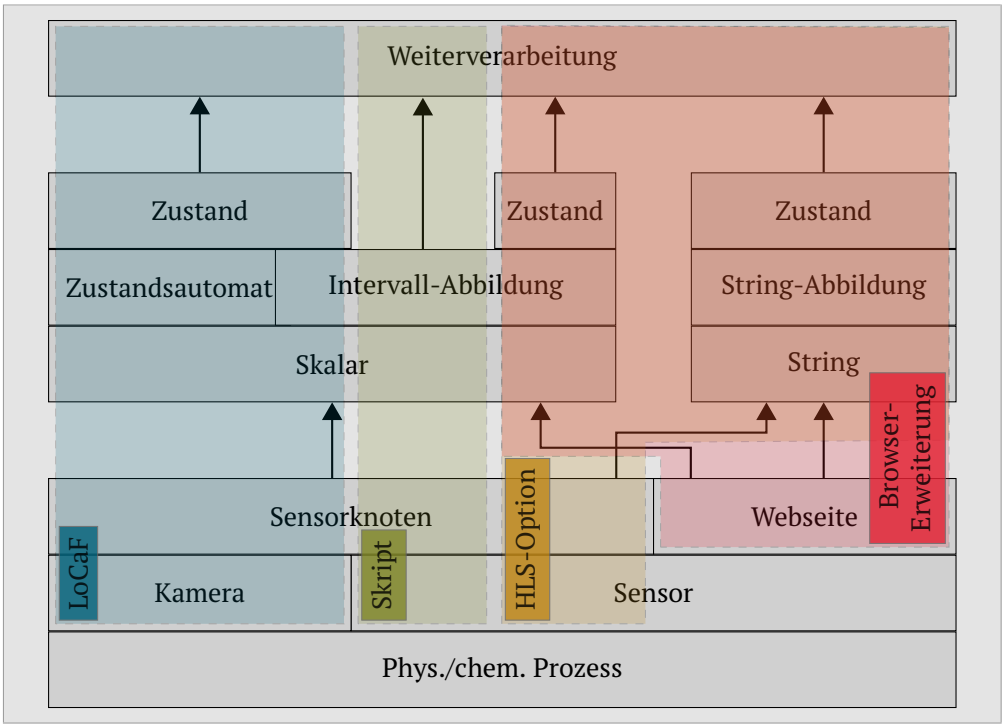


Abbildung 4.32.: Veranschaulichung des Zusammenhangs der vorgestellten Technologien zur Zustandsermittlung.

durch Zahlen oder Strings dargestellt werden können. Die Regelsprachen, mit der in den jeweiligen Systemen der Zustand aus den Sensorausgaben erzeugt wird, basiert dabei auf zwei gemeinsamen Regeltypen. Der erste erlaubt die Abbildung von Zahlen und Zahlintervallen auf einen Zustand, während der zweite einen Zustand aus Strings ermittelt. Diese Regeln lassen sich nicht auf Bilder anwenden. Allerdings wurde bei LoCaF gezeigt, wie sich durch eine Vorverarbeitung aus einem Foto die Anzahl der bewegten Objekte im Bild und der von diesen eingenommenen Fläche berechnen lässt. Diese beiden Werte können dann als Eingabe für die Regeln genutzt werden, um einen Zustand zu ermitteln. Sobald der Ressourcenverbrauch keine große Rolle spielt, kann die Komplexität der Regeln auch erhöht werden. So sind beim letzten System, der Browser-Erweiterung, statt einfacher Strings auch reguläre Ausdrücke nutzbar. Eine komplexere Regelsprache hat aber andererseits den Nachteil der schwierigeren Erlernbarkeit. Die verschiedenen Technologien haben exemplarisch gezeigt, wie auf einfache, aber dennoch flexible Art Zustände ermittelt werden können. Die vorgestellten Ideen sind mit nur wenig Aufwand auch für andere Systeme, wie Webcams oder Smartphones, umsetzbar. Ebenso ist es denkbar die Ideen von LoCaF und der Browser-Erweiterung zu verknüpfen, so dass Nutzer nicht nur aus Text von Webseiten, sondern auch aus eingebetteten Bildern periodisch und automatisiert Zustände ermitteln können.

4.3. Zustandswahrscheinlichkeiten

Die Berechnung von Zustandswahrscheinlichkeiten erlaubt das Erstellung von Wahrscheinlichkeitsmodellen für Sensorzustände. Diese Modelle können wie bei Sensor Ranking dazu genutzt werden, die Beantwortung von Suchanfragen nach Sensoren effizienter abzuwickeln. Nachfolgend wird eine formale Definition von Sensoren, Zuständen, Wahrscheinlichkeitsmodellen und deren Abfrage gegeben. Darauf aufbauend werden verschiedene Modelltypen und Verfahren zur Erzeugung der Modelle vorgestellt. Die wesentlichen Inhalte aus Abschnitt 4.3.2 wurden vorab in [162] veröffentlicht.

4.3.1. Definition

Ein Sensor s aus einer Menge von Sensoren $\mathcal{S}$, der den Zustand der realen Welt überwacht, wird durch die Funktion

$$s : \mathcal{T} \mapsto \mathcal{V}$$

beschrieben. Dabei ist $\mathcal{T}$ eine diskrete Menge an Zeitpunkten. $\mathcal{V}$ ist eine endliche, diskrete, geordnete Menge von Zuständen des Sensors. Dies können z. B. die Menge der Temperaturzustände $\mathcal{V} = \{\textit{kalt}, \textit{moderat}, \textit{warm}\}$ sein. Sensoren geben also einen von endlich vielen, diskreten Zuständen aus, wie sie im vorherigen Abschnitt aus rohen Sensorwerten ermittelt wurden.

Ein Wahrscheinlichkeitsmodell wird durch

$$P : \mathcal{S} \times \mathcal{T} \times \mathcal{V} \times \mathcal{A}^n \mapsto [0, 1]$$

definiert. Das Modell berechnet die Wahrscheinlichkeit, dass ein Sensor $s \in \mathcal{S}$ den Zustand $v \in \mathcal{V}$ zum Zeitpunkt $t \in \mathcal{T}$ misst. $\mathcal{A}^n$ mit $n \in \mathbb{Z}_+$ ist die Menge weiterer Parameter eines Modells.

Diese Modelle können mit der Sensor-Ranking-Technik genutzt werden, um effizient eine Menge von Sensoren zu finden, die einen gewissen Zustand messen. Eine Anfrage an das System ist dabei wie folgt definiert:

$$Q : \mathcal{T} \times \mathcal{V} \times k \mapsto \mathcal{R}.$$

Durch die Anfrage sollen $k \in \mathbb{Z}_+^*$ Sensoren gefunden werden, die zum Zeitpunkt $t \in \mathcal{T}$ den Zustand $v \in \mathcal{V}$ messen. Das Ergebnis ist die Menge $\mathcal{R} \subseteq \mathcal{S}$.

Quelltext 4.6 zeigt den Sensor-Ranking-Algorithmus im Pseudocode. Dabei ist $t = t_{now}$ gesetzt, d. h. es soll stets nach dem aktuellen Zustand von Sensoren gesucht werden. Die Zeilen 1–6 beinhalten bereits definierte Variablen und Funktionen. Die Menge $\mathcal{C}$ (Zeile 7) beinhaltet alle Sensoren, die für die Beantwortung der Suchanfrage bereits kontaktiert wurden. In Zeile 9 wird die Menge der Sensoren anhand der Wahrscheinlichkeit für den gesuchten Zustand sortiert. In den Zeilen 10–16 wird die Suche ausgeführt. Die Zeilen 11–15 werden solange wiederholt bis k Sensoren gefunden oder alle Sensoren angefragt

```
1  Eingabe: v ∈ 𝒱 : Zu suchender Zustand
2  Eingabe: k : Anzahl zu findender Sensoren
3  Ausgabe: ℛ : Menge Sensoren mit gesuchtem Zustand v
4  𝒮 : Menge von Sensoren
5  P(s,t,v) : Wahrscheinlichkeit von Sensor s für Zustand v zum
       Zeitpunkt t
6  s(t) : Zustand von Sensor s zur Zeit t
7  𝒞 : Menge bereits kontaktierter Sensoren
8
9  𝒮_𝒫 ← 𝒮 absteigend nach Wahrscheinlichkeit von P(s,t_now,v) sortiert
10 while |ℛ| < k and |𝒞| < |𝒮| do
11  s_i ← pop(𝒮_𝒫)
12  if s_i(t_now) == v then
13   ℛ ← ℛ ∪ s_i
14  end if
15  𝒞 ← 𝒞 ∪ s_i
16 end while
17 return ℛ
```

Quelltext 4.6: Sensor Ranking im Pseudocode.

wurden. Zuerst wird der erste Sensor der sortierten Menge selektiert und aus der Menge entfernt (Zeile 11), sein aktueller Zustand angefragt (Zeile 12) und, sofern der aktuelle Zustand dem gesuchten entspricht, der Ergebnismenge hinzugefügt (Zeile 13). Anschließend wird er noch der Menge kontaktierter Sensoren hinzugefügt (Zeile 15).

Im optimalen Fall berechnet ein Wahrscheinlichkeitsmodell eine hohe Wahrscheinlichkeit für einen Sensor, der den gesuchten Zustand misst und eine niedrige Wahrscheinlichkeit für unpassende Sensoren, d. h. Sensoren die einen anderen Zustand messen. Somit werden im idealen Fall nur k Sensoren angefragt, um die gewünschte Menge an passenden Sensoren zu finden. Dies funktioniert natürlich nur, sofern mindestens k Sensoren derzeit den Zustand messen. Zusätzlich müssen diese Sensoren durch die Modelle am höchsten gerankt werden, d. h. die Wahrscheinlichkeiten müssen unter allen Sensoren die höchsten Werte aufweisen. Dagegen müssen mehr als k Sensoren angefragt werden, falls das Ranking nicht optimal ist. Im schlimmsten Fall sind alle Sensoren abzufragen. Dies passiert entweder, wenn nicht ausreichend Sensoren den Zustand messen, selbst falls das Ranking optimal ist, oder wenn k Sensoren den Zustand messen, diese jedoch durch die Modelle am niedrigsten gerankt werden. Die Anzahl der notwendigen Anfragen zum Finden von k Treffern ist somit die Metrik zur Bewertung der Güte der Wahrscheinlichkeitsmodelle.

Das naive Modell ist definiert durch

$$P(s,t,v) = \frac{1}{|\mathcal{V}|}. \tag{4.1}$$

In diesem Modell ist die Wahrscheinlichkeit, unabhängig von der Eingabe, für alle Zustände identisch. Es dient als Grundwahrheit zur Evaluation und

entspricht einer zufälligen Auswahl des als nächsten abzufragenden Sensors beim Sensor Ranking.

Ein aggregiertes Wahrscheinlichkeitsmodell (APM) betrachtet die Häufigkeit des Vorkommens eines jeden Zustandes zu den vergangenen Zeitpunkten t' aus der Menge der diskreten, zurückliegenden Zeitpunkte $\mathcal{T}$. Es ist definiert durch

$$P(s,t,v) = \frac{\sum\limits_{\forall t' \in \mathcal{T}} \begin{cases} 1, & \text{falls } s(t') = v \\ 0, & \text{sonst} \end{cases}}{|\mathcal{T}|}. \tag{4.2}$$

Die Annahme für das APM ist, dass je öfter ein Zustand in der Vergangenheit vorkam, desto höher ist die Wahrscheinlichkeit, dass er auch in Zukunft vorkommt. Dieses Modell ist somit unabhängig vom Parameter t, d. h. unabhängig von der Zeit.

Durch Hinzunahme der Zeit als Parameter entsteht ein zeitabhängiges Wahrscheinlichkeitsmodell (TPM), welches durch

$$P(s,t,v) = \frac{\sum\limits_{\forall t' \in \mathcal{T}'} \begin{cases} 1, & \text{falls } s(t') = v \\ 0, & \text{sonst} \end{cases}}{|\mathcal{T}'|} \text{ mit } t \in \mathcal{T}' \subseteq \mathcal{T} \tag{4.3}$$

definiert wird. Die Menge $\mathcal{T}'$ umfasst vergangene, diskrete Zeitpunkte, die für das TPM einbezogen werden sollen. Wenn $\mathcal{T}' = \mathcal{T}$ betrachtet wird, ergibt sich das APM. Das TPM ist somit ein APM, welches nur für einen bestimmten Zeitraum gilt. Möchte man alle Zeitpunkte abdecken, so muss die Vereinigung aller Mengen $\mathcal{T}'$ der Menge $\mathcal{T}$ entsprechen. Eine Annahme, die dem TPM zugrunde liegt, ist, dass einige von Sensoren gemessene Zustände einem periodischen Muster folgen und somit von der Zeit beeinflusst werden. Möchte man dies mit dem TPM abbilden, so muss die Menge $\mathcal{T}'$ jeweils die entsprechenden diskreten Zeitpunkte der Periodizität beinhalten. Ein Modell, das z. B. periodische Wochenmuster beschreiben soll, kann eine Menge an Zeitpunkten für die Wochentage und eine Menge für die Wochenenden nutzen.

Es ist eine Vielzahl weiterer Modelle denkbar. Die für die jeweiligen Modelle nötigen Parameter können durch $\mathcal{A}^n$ angegeben werden. Durch Nutzung der Position als Parameter ergibt sich z. B. ein ortabhängiges Wahrscheinlichkeitsmodell (PPM). Zusätzlich kann die Zeit als Parameter einfließen, so dass ein Modell entsteht, welches eine Wahrscheinlichkeit abhängig von Standort und Zeit berechnet. Darüber hinaus sind Modelle für Korrelationen von Zuständen eines Sensors (intra-Sensor-korrelationsbasiertes Wahrscheinlichkeitsmodell (intra-CPM)) oder zwischen unterschiedlichen Sensoren (inter-Sensor-korrelationsbasiertes Wahrscheinlichkeitsmodell (inter-CPM)) umsetzbar, wenn dem Modell als Parameter die nötigen Informationen übergeben werden.

Eine Realisierung eines inter-CPM wird im nächsten Abschnitt vorgestellt und evaluiert. Auf die Erzeugung verschiedener Modelltypen gehen die Abschnitte 4.3.3, 4.3.4 und 5.3 ein. Die Repräsentation der Modelle wird in Kapitel 5 behandelt.

4.3.2. Korrelationsbasierte Modelle mit Bayes'schen Netzen

Im vorherigen Abschnitt wurde das inter-CPM eingeführt. Eine mögliche Umsetzung für dieses ist die Modellierung von Korrelationen zwischen Sensoren durch ein Bayes'sches Netz (BN).

Ein BN [163] ist ein gerichteter, azyklischer Graph zur Repräsentation von abhängen Wahrscheinlichkeiten. Dabei sind die Knoten Zufallsvariablen und die Kanten die Abhängigkeiten zwischen diesen. Ein Knoten x, der eine Kante zu Knoten y besitzt, wird Vaterknoten von y genannt. Die Wahrscheinlichkeitsverteilung eines Knoten wird aus den abhängigen Wahrscheinlichkeiten seiner Vaterknoten berechnet. Die Verteilungen werden durch Wahrscheinlichkeitstabellen beschrieben. Sobald der Wert, also der momentane Zustand, einer Variable bekannt ist, d. h. Evidenz vorliegt, lässt sich diese im Netz für den Knoten, der die Variable repräsentiert, eintragen. Anschließend kann diese Evidenz im Netz verbreitet und die Wahrscheinlichkeitsverteilungen der Knoten, für die noch keine Evidenz vorliegt, neu berechnet werden. Dieses Verfahren nennt sich Inferenz. Sowohl die exakte als auch approximative Lösung ist $\mathcal{NP}$-hart. Allerdings existieren verschiedene approximative Verfahren mit polynomieller Laufzeit für unterschiedliche Typen von Netzen.

Nachfolgend wird ein Verfahren zur Konstruktion eines BNs zur Umsetzung eines inter-CPMs beschrieben. Die Korrelation zwischen zwei Sensoren s_i und s_j wird definiert als

$$C_{i,j} = \frac{\sum\limits_{\forall t \in \mathcal{T}} 1 - \frac{|s_i(t) - s_j(t)|}{|\mathcal{V}|-1}}{|\mathcal{T}|},$$

wobei die Differenz von zwei Zuständen als die Differenz ihrer Indizes in der geordneten Menge $\mathcal{V}$ berechnet wird. Die Menge $\mathcal{T}$ beinhaltet diskrete, vergangene Zeitpunkte, an denen die Zustände beider Sensoren bekannt sind. $C_{i,j}$ kann Werte zwischen 0 und 1 annehmen. Umso höher der Wert von $C_{i,j}$, desto größer ist die Korrelation der beiden Sensoren. Jeder Sensor s ist ein Knoten im BN. Da die Konstruktion von Kreisen nicht möglich ist, kann kein vollständiges Netz mit allen Abhängigkeiten zwischen den Sensoren erzeugt werden. Da zusätzlich die Größe der Wahrscheinlichkeitstabellen exponentiell mit der Anzahl der Eltern und deren Zuständen wächst, ist ein niedriger Eingangsgrad für jeden Knoten erstrebenswert. Deswegen wird eine einfache Netzstruktur in Form einer Liste gewählt. Somit hat jeder Knoten nur einen Elternknoten bis auf die Wurzel des Netzes. Damit sind alle Knoten verbunden und dennoch entstehen nur kleine Tabellen. Es wird trotzdem die Korrelation zwischen jedem Paar

	t_1	t_2	t_3	t_4
s_1	k	k	w	w
s_2	w	k	w	k
s_3	w	k	w	w
s_4	w	w	k	k

Tabelle 4.12.: Vier Sensoren mit jeweils vier Zustandsmessungen.

i	j	$C_{i,j}$
1	2	$\frac{1}{2}$
1	3	$\frac{3}{4}$
1	4	0
2	3	$\frac{3}{4}$
2	4	$\frac{1}{2}$
3	4	$\frac{1}{4}$

Tabelle 4.13.: Korrelationen für jedes Sensorpaar.

von Knoten modelliert, weil die Abhängigkeitsbeziehungen in BNs transitiv sind. Da das Propagieren von Evidenz aber mehr Auswirkung auf näherliegende Knoten hat, werden bei der Konstruktion immer stark korrelierende Knoten, d. h. mit großem $C_{i,j}$-Wert, miteinander verbunden.

Der Aufbau des Netzes wird an einem Beispiel illustriert. Das BN wird für die vier Sensoren $s_1 - s_4$ erzeugt, welche die Zustände *kalt* (k) und *warm* (w) messen. In Tabelle 4.12 sind die Sensoren und jeweils vier Zustandsmessungen für die Zeitpunkte t_1–t_4 aufgelistet. Die berechneten $C_{i,j}$ für jedes Sensorpaar finden sich in Tabelle 4.13. Um die Listenstruktur zu erzeugen, werden die Sensorpaare zuerst absteigend nach dem Wert $C_{i,j}$ sortiert. Nun wird das Paar mit der höchsten Korrelation gewählt und verbunden. Im Beispiel sind das die Sensoren s_1 und s_3 sowie s_2 und s_3. Es wird zufällig das 1. Paar gewählt. Der Sensor s_1 ist damit Vaterknoten von s_3. Weil s_1 keinen weiteren Kindknoten erhalten soll, werden alle Sensorpaare, die s_1 enthalten aus der Tabelle entfernt. Im nächsten Schritt wird ein Kindknoten für s_3 gesucht, indem aus allen verbleibenden Paaren, die s_3 enthalten, dasjenige mit dem höchsten Korrelationswert gewählt wird. Für das Beispiel ist dies s_2. Dieser Prozess wird solange wiederholt, bis die Tabelle leer ist und somit alle Knoten im BN verbunden sind. Die entstehende Liste hat somit s_1 als Wurzelknoten und die Struktur $s_1 \rightarrow s_3 \rightarrow s_2 \rightarrow s_4$, wobei $\rightarrow$ die Vaterbeziehung repräsentiert.

Nach der Konstruktion der Struktur werden die Wahrscheinlichkeitstabellen der Knoten gefüllt. Diese enthalten die Wahrscheinlichkeit für jeden Zustand $v \in \mathcal{V}$. Da die Wurzel des Netzes keine Abhängigkeiten durch einen Vaterknoten besitzt, sind die Wahrscheinlichkeiten für jeden Zustand definiert durch Gleichung 4.2. Für alle anderen Knoten gilt die Gleichung

```
1  Eingabe: v ∈ 𝒱 : Zu suchender Zustand
2  Eingabe: k : Anzahl zu findender Sensoren
3  Ausgabe: ℛ : Menge Sensoren mit gesuchtem Zustand v
4  𝒮 : Menge von Sensoren
5  P(s,t,v) : Wahrscheinlichkeit von Sensor s für Zustand v zum
       Zeitpunkt t
6  s(t) : Zustand von Sensor s zur Zeit t
7  𝒞 : Menge bereits kontaktierter Sensoren
8
9  while |ℛ| < k and |𝒞| < |𝒮| do
10  𝒮_𝒫 ← 𝒮 absteigend nach Wahrscheinlichkeit von P(s,t_now,v)
       sortiert
11  s_i ← pop(𝒮_𝒫)
12  𝒮 ← 𝒮\s_i
13  if s_i(t_now) == v then
14   ℛ ← ℛ ∪ s_i
15  end if
16  𝒞 ← 𝒞 ∪ s_i
17  Füge Evidenz ein und aktualisiere Bayes'sches Netz
18 end while
19 return ℛ
```

Quelltext 4.7: Abfragealgorithmus im Pseudocode.

$$P(s,t,v) = \sum_{\forall v,w \in \mathcal{V}} P(s,v,w)$$

mit

$$P(s,v,w) = \frac{\sum\limits_{\forall t' \in \mathcal{T}} \begin{cases} 1, & \text{falls } s(t') = v \wedge par(s,t') = w \\ 0, & \text{sonst} \end{cases}}{|\mathcal{T}|}.$$

Die Funktion $par(s,t)$ gibt den Zustand des Vaterknotens von s zum Zeitpunkt t zurück.

Der in Quelltext 4.6 beschrieben Abfragealgorithmus wird an drei Stellen für die Nutzung mit dem BN angepasst. Der veränderte Algorithmus ist in Quelltext 4.7 zu sehen. Die Evidenz, die durch das Abfragen von Sensoren zur Verfügung steht, wird in das BN eingetragen, d. h. die Wahrscheinlichkeit für den gemessenen Zustand wird auf 1 und für alle anderen Zustände auf 0 gesetzt. Anschließend wird das Netz aktualisiert (Zeile 17). Aufgrund der Änderung der Wahrscheinlichkeiten nach jedem Durchlauf, wird das Sortieren der Menge $\mathcal{S}$ in die while-Schleife verschoben (Zeile 10). Da $\mathcal{S}$ in jeder Iteration neu sortiert und ein Sensor ausgewählt wird, muss der zu kontaktierende Sensor aus der Menge entfernt werden, damit er nicht wiederholt angefragt wird (Zeile 12).

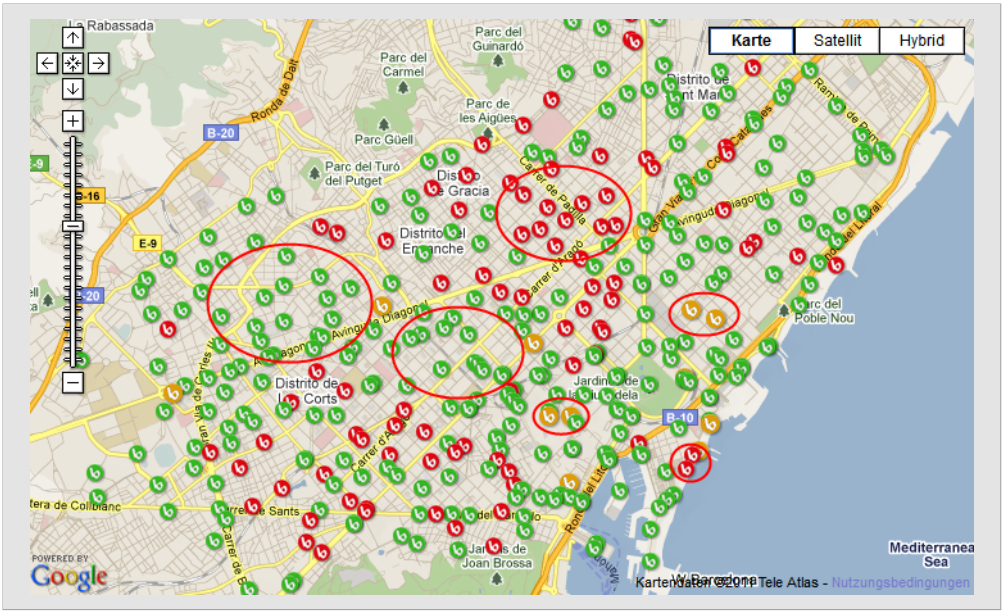


Abbildung 4.33.: Karte von Barcelonas Bicing Mietstationen. Kreise markieren Mengen von Stationen mit gleichem Zustand, d. h. einer ähnlichen Anzahl an verfügbaren Fahrrädern.

Evaluation

Zur Evaluation des Ansatzes wird ein Datensatz von 384 Stationen des *Bicing* Fahrradmietsystems in Barcelona[19] genutzt. Jede Station besitzt einen Sensor, der die Anzahl verfügbarer Fahrräder misst. Die Information kann online auf einer Webseite nachgeschaut werden. Abbildung 4.33 zeigt eine Karte mit eingezeichneten Stationen. Dabei spiegelt die Farbkodierung die Anzahl an Rädern wider. Da Barcelona mehrere Berge im Stadtgebiet besitzt, weisen naheliegende Stationen ähnliche Nutzungsmuster auf, weil die Nutzer die Räder einsetzen, um die Anhöhen hinabzufahren, jedoch seltener, um wieder hinaufzufahren. Somit sind die Stationen auf Bergen öfter leer, während im Tal ausreichend Fahrräder verfügbar sind. Die roten Kreise in Abbildung 4.33 zeigen jeweils Mengen von Stationen, die derzeit ähnliche Werte messen, was durch die Färbung signalisiert wird.

Der Suchalgorithmus aus Quelltext 4.7 ist in Java implementiert und nutzt das SMILE-Framework[20] des Decision Systems Laboratory der University of Pittsburgh für die Inferenz in BNs. Zur Inferenz wird das EPIS-Sampling [164] genutzt. Der Datensatz des Fahrradmietsystems liegt für den schnellen Zugriff in einer SQL-Datenbank vor. Für den Versuch werden statt der Anzahl an Rädern fünf Zustände von *leer* bis *ausreichend Räder* genutzt. Der vorgestellte Algorithmus wird verglichen mit der zufälligen Auswahl von Sensoren, welches dem naiven Model aus Gleichung 4.1 entspricht. Es werden jeweils

[19] `https://www.bicing.cat/` [Zugriff am 21. März 2014]
[20] `http://genie.sis.pitt.edu/` [Zugriff am 21. März 2014]

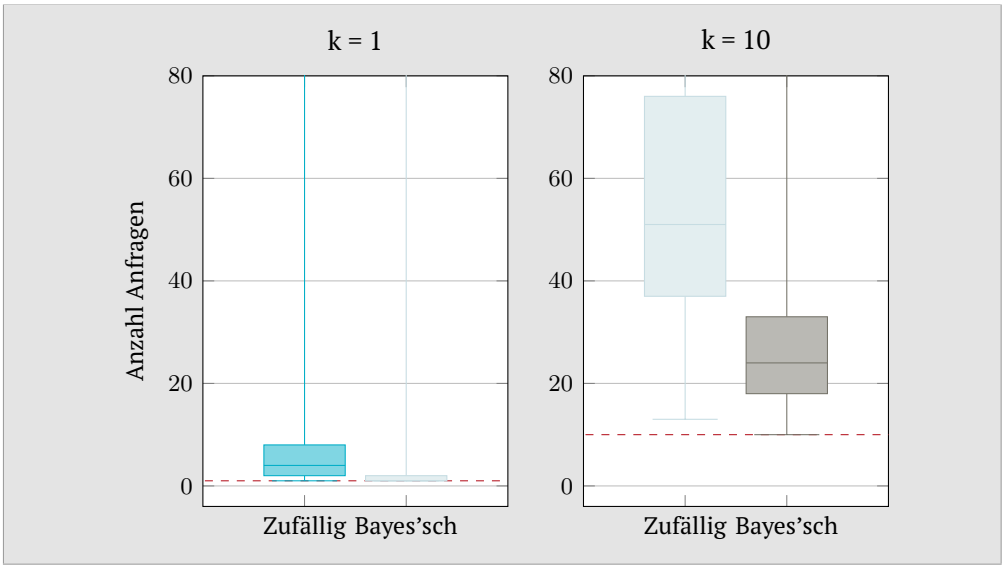


Abbildung 4.34.: Boxplots der Anzahl an Sensoranfragen für 384 Sensoren, um k = 1 und k = 10 Sensoren mit gesuchtem Zustand zu finden. Vergleicht werden jeweils ein zufälliges Anfragemuster und die Verwendung von Sensor Ranking mit Wahrscheinlichkeitsmodellen, die auf Bayes'schen Netzen basieren. Die Maxima aller Plots wurden zur besseren Darstellung der Boxen nicht geplottet und liegen bei 384. Die rote Linie gibt das erreichbare Optimum an.

1.000 Suchanfragen durchgeführt, um eine bzw. zehn passende Mietstationen zu finden. Der zu suchende Zustand wird dabei jeweils zufällig gewählt.

Abbildung 4.34 zeigt Boxplots für die Suche nach $k = 1$ und $k = 10$ Treffern für die zufällige Sensorauswahl und den BN-Ansatz. Die gestrichelte rote Linie ist das erreichbare Optimum. Zur besseren Übersicht ist die y-Achse bei 80 abgeschnitten. Die Maximalwerte liegen für alle Plots bei 384. Im schlechtesten Fall werden somit alle Sensoren abgefragt, da sich nicht ausreichend passende Treffer finden lassen. Für beide Werte von k erzielt der korrelationsbasierte Algorithmus bessere Ergebnisse, sowohl für den besten Fall als auch im Mittel. Für $k = 1$ ist für den BN-Ansatz der Median aller Läufe optimal, d. h. bei mindestens der Hälfte aller Suchanfragen wird mit der ersten Anfrage ein passender Sensor gefunden.

Fazit

Das vorgestellte korrelationsbasierte Wahrscheinlichkeitsmodell auf Basis von BNs benötigt signifikant weniger Anfragen an Sensoren, um eine vorgegebene Menge von Ergebnissen mit einem bestimmten Zustand zu finden. Der Nachteil des Ansatzes ist, dass von allen Sensoren Daten an zentraler Stelle, z. B. im Backend, zu sammeln sind, um das BN erzeugen zu können. Nachfolgend werden deshalb zwei alternative Lösungen vorgestellt, die das Erzeugen von Modellen auf Sensorknoten ermöglichen.

4.3.3. Modellgenerierung mittels Skriptsprache

In Abschnitt 4.2.3 wird beschrieben, wie sich mittels Skriptsprache Zustände und Statistiken aus den Sensorwerten auf einem Sensorknoten erzeugen lassen. Das dort genutzte Beispiel aus Quelltext 4.3 hat vier verschiedene Detailstufen. Nach dem Mapping auf die Werte 0 und 1, die den Zuständen *kalt* und *warm* entsprechen, wird der Zeitstempel entfernt und ein Histogramm erzeugt. Am Ende enthält die Tabelle somit in der ersten Spalte die beiden möglichen Werte 0 und 1 und in der zweiten deren Häufigkeit. Diese Information kann im Backend zur Erzeugung eines APM genutzt werden, da sowohl die Gesamtzahl an Zuständen, die Summe aus beiden Häufigkeiten, als auch die Häufigkeit der einzelnen Zustände bekannt ist.

TPMs lassen sich ebenfalls auf einfache Weise erzeugen. Dazu ist lediglich, bevor die Zeitstempel gelöscht werden, die Filterfunktion auf die Zeitstempel-Spalte anzuwenden, um die Zeilen des Zeitraumes zu behalten, für welche die Wahrscheinlichkeit der Zustände ermittelt werden soll. Um somit die Wahrscheinlichkeiten für unterschiedliche Zeiträume zu ermitteln, müssen LoD-Skripte für die jeweiligen Zeitintervalle auf dem Knoten vorhanden sein.

Erweitert man das System aus Abschnitt 4.2.3 in dem Sinne, dass neben dem Zeitstempel und Sensorwert auch die Position in Form von Koordinaten mitgespeichert wird, ist es auch möglich, PPMs auf dem Knoten zu erzeugen. Die Tabelle hat entsprechend noch Spalten für die Komponenten von Koordinaten. Somit lässt sich statt nach der Zeit, wie beim TPM, nach der Position filtern.

Modelle wie das inter-CPM lassen sich dagegen nicht durch ein Skript erzeugen, da sie Informationen von verschiedenen Sensoren benötigen. Dafür müssten die Zustände der Sensoren zentral, z. B. im Backend, gesammelt und ausgewertet werden, wie es beim Ansatz mit BNs im vorherigen Abschnitt durchgeführt wurde.

4.3.4. Modellgenerierung mittels CoAP

Neben der HLS-Option aus Abschnitt 4.2.2 wurde eine weitere Option für CoAP in der Wiselib implementiert. Diese erlaubt gleichermaßen wie die HLS-Option ein dynamisches Erstellen von Subressourcen unterhalb von Sensor- und Zustandsressourcen durch Senden einer Option an einen Server. Die so erstellte Ressource, eine Wahrscheinlichkeitsmodellressource, bietet ein Wahrscheinlichkeitsmodell für die Sensorwerte bzw. Zustände an. Dies ist in Abbildung 4.35 visualisiert.

Das Format der Option ist in Abbildung 4.36 zu sehen. Das T-Feld gibt dabei den Typ des zu nutzenden Wahrscheinlichkeitsmodells an. Zur Erstellung des Modells werden historische Werte des Sensors benötigt. Deshalb sind als weitere gemeinsame Parameter für jedes Modell das Intervall, welches die Periode zum Loggen von Werten definiert, und die Anzahl Werte (*Werteexponent*), die zur Erstellung des Modells genutzt werden sollen, nötig. Die Periode wird dabei

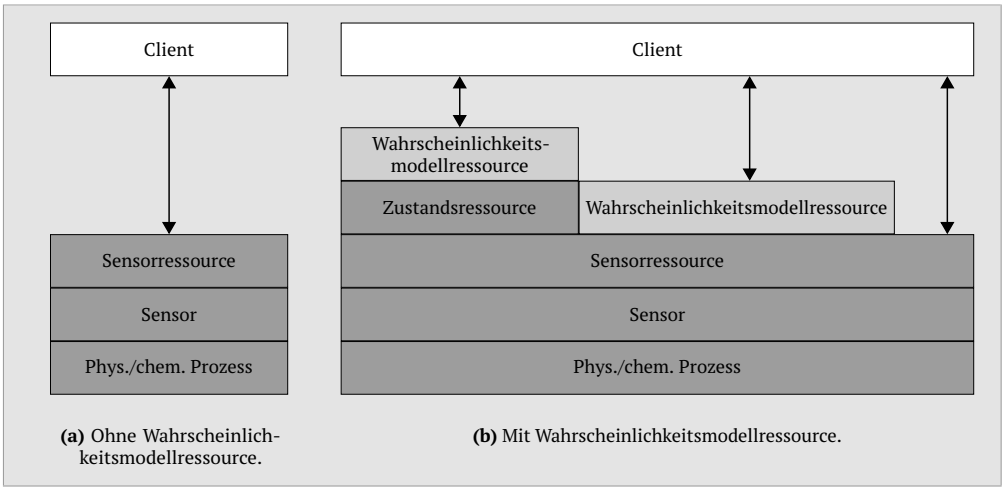


Abbildung 4.35.: Veranschaulichung der Ressourcenabfrage durch einen Client. Die Sensorressource greift auf den Sensor zu der wiederum den phys./chem. Prozess überwacht. Die Wahrscheinlichkeitsmodellressourcen erzeugen das Wahrscheinlichkeitsmodell aus Sensorwerten oder Zuständen der darunterliegenden Ressource.

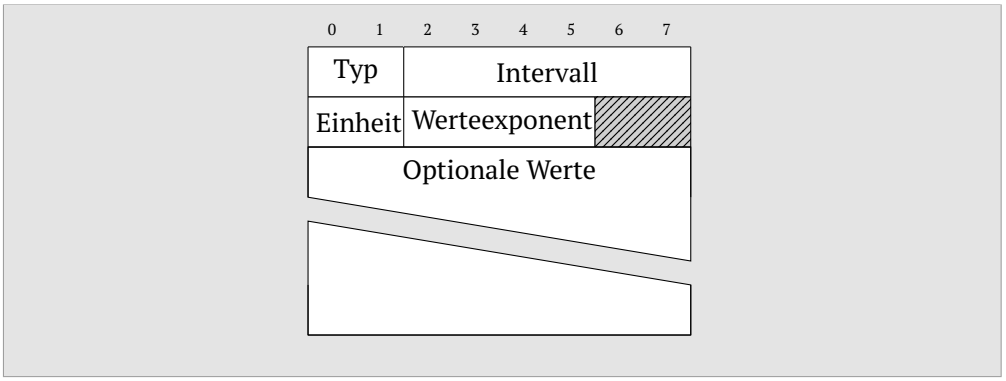


Abbildung 4.36.: Format der Probability-Model-Option.

durch einen *Intervall* (1–60) und eine *Einheit* (Sekunde, Minute oder Stunde) definiert. Die Anzahl zu speichernder Werte ergibt sich aus 2^x, wobei x der in der Option gesendete Werteexponent ist, der die Werte 1–16 annehmen kann. Die *optionalen Werte* enthalten modellspezifische Parameter, die nachfolgend beschrieben werden.

Als Modelle können theoretisch alle diejenigen genutzt werden, die lokal auf einem Sensorknoten berechenbar sind, d. h. APM, TPM, PPM und intra-CPM. In der Wiselib ist exemplarisch die Erzeugung von APM- und TPM-Ressourcen umgesetzt. Der Pfad der Ressource wird entsprechend des Modellnamens gewählt, d. h. */apmx* für ein APM und */tpmx* für ein TPM, wobei x eine fortlaufende Zahl ist. So können auch mehrere Modelle, z. B. mit unterschiedlichen Logging-Intervallen oder verschiedenen Zeitschlitzdefinitionen, für eine Ressource parallel existieren.

Während für das APM keine weiteren Parameter notwendig sind, wird für ein TPM neben den bereits genannten Parametern noch eine Beschreibung der Zeitschlitze benötigt. Zuerst wird durch 2 Bit die Zeiteinheit in Minuten, Stunden oder Tagen definiert. Mittels eines nachfolgenden Bytes wird die Anzahl an Schlitzen angegeben. Danach folgen entsprechend viele Felder (jeweils 5 Bit), welche die Länge eines Schlitzes in der definierten Einheit angeben. Die Zeitschlitze starten immer am Beginn der Woche um 0 Uhr und werden fortlaufend aufaddiert. Werden z. B. drei Schlitze mit jeweils acht Stunden definiert, so gehen die Schlitze von 0–8, 8–16 und 16–24 Uhr. Da die 3×8 Stunden aufaddiert 24 Stunden ergeben, wird beim nächsten Tag genauso vorgegangen. Ist der Wert < 24, so werden die restlichen Stunden des Tages als eigener Schlitz angelegt und die Schlitzeinteilung ebenfalls beim neuen Tag neu begonnen. Tritt der Fall auf, dass die Stundenzeitschlitze aufaddiert > 24 sind, so wird erst bei der nächsten Woche neu mit der Zeitschlitzeinteilung begonnen. Ein ähnlicher Mechanismus wird für die Zeiteinheit Minuten bzw. Tage verwendet. Hier werden die Schlitze an den Anfängen der nächsten Stunde oder des nächsten Tages bzw. der nächsten Woche oder des nächsten Monats wiederholt. PPMs ließen sich analog durch Angabe von Regionen als Parameter realisieren.

Sobald die Ressource erstellt wird, speichert diese in der spezifizierten Periode neue Werte bzw. Zustände der zugrundeliegenden Ressource. Sobald ein Client die Ressource anfragt, werden die Wahrscheinlichkeiten für die gespeicherten Werte bzw. Zustände berechnet und an den Client gesendet. Die Rückgabe besteht dabei aus einer Liste von Sensorwerten bzw. den Zuständen und den zugehörigen Wahrscheinlichkeiten. Während die Antwort für das APM eine solche Liste beinhaltet, wird für das TPM für jeden Zeitslot eine Liste angegeben. Zusätzlich beinhaltet die Antwort beim TPM den Beginn und das Ende für jeden Slot.

4.3.5. Fazit

Nach einer formalen Definition, wurde in Abschnitt 4.3.2 eine Möglichkeit zur Erzeugung von inter-CPMs präsentiert. Die Evaluation des auf BNs basierenden Ansatzes hat das Potential von Wahrscheinlichkeitsmodellen gezeigt. Die Auswahl anzufragender Sensoren zur Beantwortung einer Suchanfrage kann mit Hilfe von Modellen im Vergleich zur zufälligen Selektion Ressourcen einsparen. Es werden signifikant weniger Sensoren kontaktiert, um ausreichend passende Sensoren zu ermitteln. Somit sind weniger Nachrichten erforderlich, was auch zu einer Reduktion des Energieverbrauchs führt und damit die Lebenszeit von Sensorknoten erhöht.

Da die Generierung von Modellen im Backend auch Nachteile mit sich bringt, wurden alternativ zwei Verfahren vorgestellt, die es erlauben, Wahrscheinlichkeitsmodelle auf Sensorknoten zu erzeugen und abzurufen. Das erste baut auf der in Abschnitt 4.2.3 vorgestellten Skriptsprache auf. Die zweite Variante ist eine weitere CoAP-Option, die ähnlich der in Abschnitt 4.2.2 präsentierten HLS-Option dynamisch neue Ressourcen erstellt. Diese nutzen als Datenlieferant

entweder Sensorressourcen oder von der HLS-Option erzeugte Zustandsressourcen. Daraus können die erzeugten Ressourcen verschiedene Typen von Modellen erzeugen, die durch Clients abgerufen werden können.

Der Vorteil der Erzeugung von Wahrscheinlichkeitsmodellen auf Sensorknoten ist, dass keine Daten per Funk an andere Geräte übertragen werden müssen. Nur die Parameter der Modelle müssen nach der Generierung verschickt werden. Die Datenmenge der Modellparameter ist im Allgemeinen wesentlich kleiner als die zugrundeliegenden Daten. Aber auch die Generierung auf Sensorknoten mit lokal vorliegenden Daten hat Nachteile. Es können z. B. keine Daten von anderen Sensoren in die Modelle einfließen. Außerdem ist die Speicher- und Rechenkraft auf Sensorknoten begrenzt, weswegen nur eine begrenzte Anzahl an Daten einfließen kann und nur einfache Modelle berechenbar sind. In Abschnitt 5.3 wird deshalb ein zweites Verfahren vorgestellt, das die Generierung im Backend ermöglicht. Dieses basiert auf der im nächsten Kapitel vorgestellten RDF-Modellierung.

5 Informationsmodellierung

> The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.
>
> — *Tim Berners-Lee*

In Kapitel 4 wurde beschrieben, wie aus rohen Sensorwerten höherwertige Zustände ermittelt werden können. Anschließend sind Verfahren zur Generierung von Wahrscheinlichkeitsmodellen für diese Zustände vorgestellt worden. Damit Anwendungen, wie z. B. eine Suchmaschine für das IoT, aus den Zuständen und Modellen einen Nutzen ziehen können, müssen weitere Informationen zur Verfügung stehen. So kann es notwendig sein, z. B. den Sensortyp, die Adresse des Sensors bzw. Sensorknotens, den Ort und die beobachtete Entität zu kennen. Diese oftmals statischen Informationen müssen deshalb mit den dynamischen Informationen zusammengeführt und einheitlich abgespeichert werden. Relationale Datenbanken oder Klartext, wie etwa XML oder Comma-separated values (CSV), eignen sich für diesen Zweck. Um allerdings bereits bestehendes Wissen zu integrieren und mit Hilfe dieses Wissens auch neue Fakten zu generieren, bietet es sich an, Techniken des Semantic Web (vgl. Abschnitt 2.3) einzusetzen.

Nach einer Diskussion verwandter Arbeiten wird im Folgenden dargestellt, wie Sensorknoten, die angeschlossenen Sensoren, die Zustände der Sensoren, die Historie von Sensorzuständen und die Wahrscheinlichkeitsmodelle mittels RDF

beschreibbar sind. Dabei werden existierende Konzepte, d. h. Ontologien, aus dem Semantic-Web-Bereich wiederverwendet. Das vorgestellte Modell basiert auf der Ontologie von Compton et al. [165], welches auch schon in anderen Projekten Verwendung findet [166, 167]. Die wesentlichen Inhalte wurden vorab in [168] veröffentlicht.

5.1. Verwandte Arbeiten

Die W3C Semantic Sensor Network Incubator Group hat die Semantic-Sensor-Network-Ontologie (SSN-Ontologie) [165] zur Beschreibung von Sensoren und deren Beobachtungen erstellt. Dazu wurden bestehende Ontologien untersucht, um die wichtigen Aspekte zu extrahieren und Nachteile zu identifizieren. Die Ontologie betrachtet dabei drei verschiedene Perspektiven eines sensorischen Systems: den Sensor, die Beobachtung und das Gesamtsystem. Die Sensorperspektive dient zur Beschreibung von Sensoren, deren Messfähigkeiten, wie Genauigkeit und Auflösung, und unter welchen Bedingungen diese erreicht werden. Die beobachtete Entität, die verwendete Messmethode und der Messzeitpunkt sowie die Repräsentation des Ergebnisses werden durch die Beobachtungsperspektive modelliert. Schlussendlich bietet die Systemperspektive u. a. Möglichkeiten zur Modellierung der Plattformen, der Subsysteme und deren Arbeitsbereiche. Bestimmte Aspekte wie Messeinheiten, geografischen Positionen und Zeitangaben wurden absichtlich nicht modelliert, um dem Nutzer die freie Wahl zu geben, eigene oder bestehende Ontologien zu nutzen, die seinen Anwendungsfall optimal unterstützen.

Die im Rahmen des SPITFIRE-Projektes[1] erstellte SPITFIRE-Ontologie[2] [167] ist in Teilen eine Erweiterung bzw. Spezialisierung der SSN-Ontologie des W3C. Die Ontologie setzt die gemessenen Sensordaten in einen Kontext mit dem zugrundeliegenden Ereignis. Sie erlaubt die Struktur und den Auslöser des gemessenen Ereignisses sowie die Beziehung zu anderen Aktivitäten zu modellieren. Dazu wird u. a. die *friend of a friend*-Ontologie zur Modellierung von sozialen Beziehungen eingebunden.

Die beschriebenen Ontologien erlauben zwar Sensoren, die Messwerte und zugrundeliegenden Ereignisse zu modellieren, bieten aber keine Möglichkeit der Modellierung von höherwertigen Zuständen und Wahrscheinlichkeitsmodellen für diese.

5.2. RDF-Modellierung

Das in dieser Arbeit entwickelte RDF-Modell wird inkrementell aufgebaut und es wird erklärt, welche Elemente optional bzw. verpflichtend sind. Neue Modellteile sind dabei eingefärbt, während bereits bekannte mit weißem Hintergrund

[1] `http://spitfire-project.eu/` [Zugriff am 21. März 2014]
[2] `http://spitfire-project.eu/ontology.owl` [Zugriff am 21. März 2014]

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX muo: <http://purl.oclc.org/NET/muo/muo#>
PREFIX spf: <http://spitfire-project.eu/ontology/ns/>
PREFIX spfu: <http://spitfire-project.eu/uom/>
PREFIX spfp: <http://spitfire-project.eu/property/>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX s: <http://www.iti.uni-luebeck.de/ontology/state#>
PREFIX h: <http://www.iti.uni-luebeck.de/ontology/history#>
PREFIX m: <http://www.iti.uni-luebeck.de/ontology/model#>
PREFIX q: <http://www.iti.uni-luebeck.de/ontology/query#>
PREFIX r: <http://www.iti.uni-luebeck.de/ontology/ranking#>
```

Quelltext 5.1: Verwendete Namensräume und deren Präfixe.

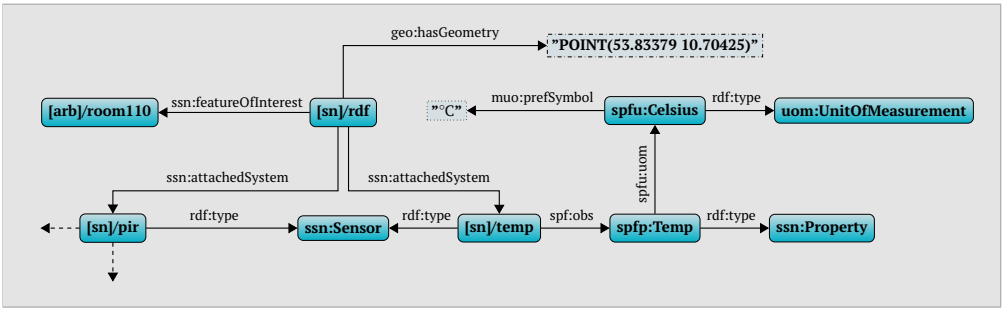


Abbildung 5.1.: RDF zur Beschreibung eines Sensorknotens und der angeschlossenen Sensoren.

dargestellt werden. Für eine bessere Übersichtlichkeit der Modelle sind die URIs abgekürzt. Die Präfixe und zugehörigen Namensräume sind in Quelltext 5.1 aufgeführt. Dies ist die Schreibweise zur Definition von Präfixen im ersten Teil von SPARQL-Anfragen. In den folgenden Anfragen werden diese der Redundanz wegen ebenfalls weggelassen. Wie in Abschnitt 2.3.1 beschrieben, sind URIs als abgerundete Rechtecke dargestellt, während Literale als normale Rechtecke visualisiert werden. Der Rahmen von Literalrechtecken dient als Kodierung des Datentyps, ebenfalls um eine bessere Übersichtlichkeit zu gewährleisten. Alle numerischen Datentypen haben einen durchgezogenen, Strings einen gepunkteten, Datums- und Zeittypen einen gestrichelten und WKT-Typen einen strichpunktierten Rahmen (vgl. Tabelle 2.3 in Abschnitt 2.3.1).

5.2.1. Metadaten

Die SSN-Ontologie [165], die als Grundlage für das RDF-Modell dient, erlaubt u. a. Konzepte, wie Sensorknoten, Sensoren und deren Messwerte, zu definieren. Abbildung 5.1 zeigt den für diese Arbeit wichtigen Teil zur Modellierung von Sensorknoten und deren Sensoren. Das Element *[sn]/rdf* beschreibt den

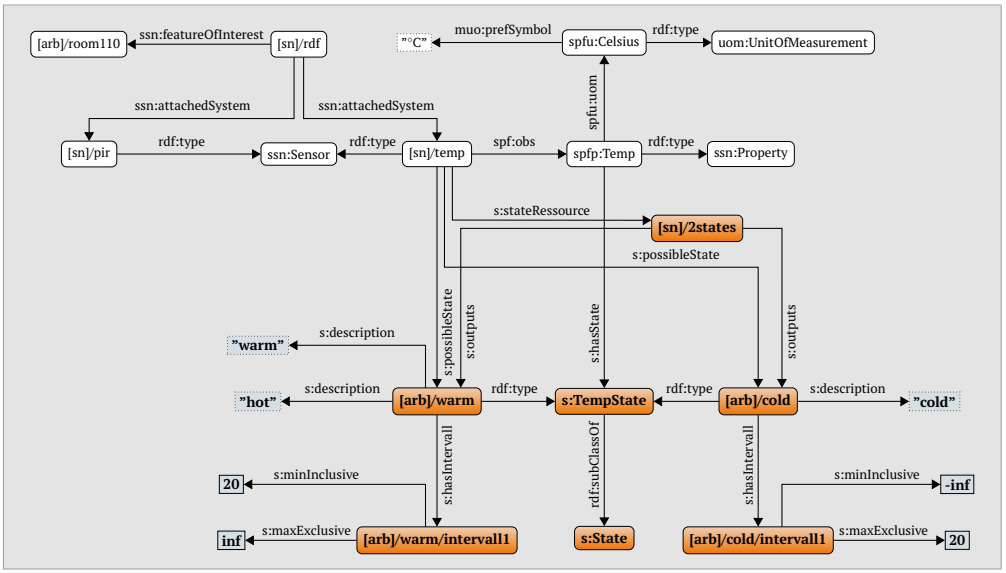


Abbildung 5.2.: RDF zur Beschreibung von Sensorzuständen.

Sensorknoten, wobei *[sn]* ein Platzhalter für die IPv6-Adresse ist, unter der dieser erreichbar ist. Das Element */rdf* ist die Hauptressource, welche Informationen über den Sensorknoten liefert. Das Prädikat *ssn:featureOfIntererst* gibt an, welche Entität der Sensorknoten überwacht. In diesem Fall ist dies die Entität mit der URI *[arb]/room110*, wobei *[arb]* stellvertretend für eine beliebige URI steht. Dem Namen nach zu urteilen repräsentiert die URI einen Raum. An einen Sensorknoten können beliebig viele Sensoren angeschlossen sein. Der Sensorknoten im Beispiel besitzt zwei Sensoren, die jeweils über das Prädikat *ssn:attachedSystem* angebunden sind. Das Element *[sn]/pir* stellt einen PIR-Sensor zur Detektion von Bewegungen dar, während *[sn]/temp* ein Temperatursensor ist, der in der Maßeinheit Celsius mit dem Symbol °C misst. Über das Prädikat *geo:hasGeometry*, welches an den Sensorknoten gebunden wird, kann die geografische Ausdehnung bzw. Lage, in diesem Fall eine Latitude-Longitude-Koordinate, als GeoSPARQL-Ausdruck angegeben werden. Die Region ist die einzige Ergänzung zur Ontologie. Zwar kann die überwachte Entität ebenfalls eine Position enthalten, dies ist aber nicht zwingend vorgeschrieben. Auch das Format kann beliebig sein. Für das spätere Finden anhand einer Position wird deshalb ein GeoSPARQL-Ausdruck verwendet.

5.2.2. Zustände

Die Umfrageergebnisse aus Abschnitt 3.2.2 haben gezeigt, dass Nutzer eine unterschiedliche Genauigkeit für Zustände wünschen. Weiterhin wurden bei den Technologien aus Kapitel 4 verschiedene Varianten vorgestellt, wie Zustände aus Sensorrohwerten ermittelt werden können. Obwohl diese Technologien bereits Zustände als Ausgabe haben, kann es von Interessen sein, die dahinter liegenden Zustandsgrenzen zu kennen und abrufen zu können, um z. B.

aus anderen Sensorwerten ebenfalls diese Zustände ermitteln zu können. Diesen Umständen muss bei der Modellierung von Zuständen Rechnung getragen werden.

Abbildung 5.2 zeigt ein Beispiel für die Modellerweiterung zur Definition von Zuständen eines Sensors. Dabei benötigt ein Sensor zwei oder mehr Zustände. Diese werden über das Prädikat *s:possibleState* angebunden. Im Beispiel besitzt der Sensor die zwei Zustände *warm* und *cold*. Jeder Zustand ist durch ein oder mehrere Intervalle definierbar, die mittels des Prädikats *s:hasIntervall* angegeben werden. Ein oder beliebig viele Begriffe dienen zur Beschreibung des Zustandes sowie als Hilfe zur textuellen Suche und werden mit *s:description* verknüpft. Ein Intervall wiederum besteht aus einer unteren und einer oberen numerischen Grenze, die mit *s:minInclusive* und *s:maxExclusive* angegeben werden. Während die untere Grenze inklusiv ist, ist die obere exklusiv. Der Zustand *warm* wird daher durch die Begriffe „warm“ und „hot“ beschrieben und gilt für Sensorwerte, die größer oder gleich $20\,°$C sind. Alle Werte darunter sind dem Zustand *cold* zugeordnet. Wären die Grenzen beide inklusiv, gäbe es für $20\,°$C zwei Zustände. Bei exklusiven Grenzen gäbe es dagegen keinen Zustand für diesen Wert. Die beiden Zustände sind vom Typ *s:TempState*, welcher eine Unterklasse des Typs *s:State* ist. Das Element *s:TempState* wird von der physikalischen Einheit *spfp:Temp* mit *s:hasState* referenziert. Dies zeigt, dass alle Kinder von *s:TempState* mögliche Zustände der physikalischen Einheit sind, d. h. der *warm*-Zustand ist sinnvoll für Temperatur, aber nicht für Lichtstärke. Dies erlaubt zudem die Weiterverwendung bereits definierter Zustände für Temperatur.

Die HLS-Option aus Abschnitt 4.2.2 erlaubt das standardisierte Abfragen von Zuständen über eine spezielle Ressource. In der Modellierung kann diese Ressource über das Prädikat *s:stateRessource* angegeben werden. Von diesem Element lassen sich mit dem Prädikat *s:outputs* wiederum die von der Ressource ausgegebenen Zustände referenzieren.

Für Sensoren, die eine String-Abbildung statt der Intervall-Abbildung nutzen, können mit dem Prädikat *s:fromString* die Zeichenketten, von denen auf den Zustand abgebildet wird, spezifiziert werden. Ein Beispiel dazu findet sich in Abbildung 5.8.

Die Freiheit bei der Definition von Zuständen kann zu zwei vermeintlichen Problemen führen. Zum einen gehen Informationen verloren, da nicht mehr Rohwerte, sondern nur noch höherwertige Zustände genutzt werden, zum anderen kann eine Unmenge von Zuständen durch die Nutzer definiert werden, was zu einer Heterogenität führt. Allerdings ist für die Suche im IoT explizit gewollt, Zustände der Sensoren statt Rohwerte zu speichern. Dies kann u. a. zur Einsparung von Ressourcen führen. Dennoch ist es weiterhin realisierbar, auch die Rohwerte mit einer eigenen Ontologie abzuspeichern. Des Weiteren ist es zwar möglich nur zwei Zustände, wie im Beispiel, zu definieren, aber ebenso gut kann eine sehr feine Zustandsgranularität genutzt werden. Dadurch lässt sich z. B. bei Temperaturzuständen eine Genauigkeit von ganzzahligen Grad oder

```
SELECT ?state WHERE
{
 <[sn]/temp> s:possibleState ?state .
}
```

Quelltext 5.2: Anfrage, um alle Zustände eines Sensors abzurufen.

?state
[arb]/warm
[arb]/cold

Tabelle 5.1.: Ergebnis der Anfrage aus Quelltext 5.2.

sogar Gradangaben mit Nachkommastelle definieren. Auch die Bedenken einer Heterogenität von Zuständen bewahrheitet sich nicht zwingend. Ein Grundkonzept des Semantic Web ist die Wiederverwendung von Konzepten. Bevor also neue Zustände definiert werden, ist davon auszugehen, dass Entwickler bereits existierende Zustände, die ihren Anforderungen entsprechen, verwenden.

Basierend auf dem Modell für Zustände können folgende SPARQL-Anfragen formuliert werden, die es erlauben

1. alle möglichen Zustände eines Sensors abzurufen (siehe Quelltext 5.2 und Tabelle 5.1),
2. den Zustand eines Sensors für einen bestimmten Rohwert abzurufen (siehe Quelltext 5.3 und Tabelle 5.2) und
3. den Zustand eines Sensors für einen bestimmten Suchbegriff abzurufen (siehe Quelltext 5.3 und Tabelle 5.2).

5.2.3. Historie von Zuständen

Um den aktuellen Zustand eines Sensors zu erfahren, kann direkt mit diesem kommuniziert werden. Möchte man dagegen die Zustandshistorie eines Sensors in Erfahrung bringen, um z. B. Wahrscheinlichkeitsmodelle daraus zu berech-

```
SELECT ?state WHERE
{
 <[sn]/temp> s:possibleState ?state .
 ?state s:hasIntervall ?int1 .
 ?state s:hasIntervall ?int2 .
 ?int1 s:minInclusive ?min .
 ?int2 s:maxExclusive ?max .
 FILTER(?min <= 21 && 21 < ?max && ?int1 = ?int2)
}
```

Quelltext 5.3: Anfrage, um den Zustand für 21 °C abzurufen.

?state
[arb]/warm

Tabelle 5.2.: Ergebnis der Anfrage aus Quelltext 5.3.

```
SELECT ?state WHERE
{
 <[sn]/temp> s:possibleState ?state .
 ?state s:description ?desc .
 FILTER(?desc = "warm")
}
```

Quelltext 5.4: Anfrage, um den Zustand für den Begriff „warm“ abzurufen.

nen, muss er eine Möglichkeit dafür bieten. Der Sensor muss dementsprechend seine vergangenen Zustände archivieren und zur Verfügung stellen. Aufgrund des begrenzten Speichers ist dies allerdings nur eingeschränkt möglich. Außerdem soll die Kommunikation mit dem Sensor aus Energiegründen minimiert werden. Deshalb ist es hilfreich, vergangene Zustände nicht im Sensor selbst, sondern auf Systemen mit mehr Ressourcen zu speichern.

Abbildung 5.3 zeigt, wie die Zustandshistorie für einen Sensor in RDF modelliert wird. Jeder Zustand besitzt eine eigene Historie. Diese ist über das Prädikat *h:state* mit dem Zustand verbunden. Außerdem wird sie für die Eindeutigkeit noch vom Sensor mittels *h:has* spezifiziert. Die Historie speichert für den jeweiligen Zustand, zu welchem Zeitpunkt dieser vom Sensor eingenommen wurde. Dazu werden voll spezifizierte Datums- und Zeitliterale vom Typ *xsd:dateTime* über *h:time* angegeben. Der Zustand gilt als maximal gültig vom Zeitstempel (inklusive) bis zu dem nächsten chronologischen Zeitstempel (exklusive) einer der Zustände des Sensors. Es kann allerdings passieren, dass der Sensor zwischenzeitlich einen anderen Zustand gemessen hat, dieser jedoch nicht in der Historie gespeichert ist, da er z. B. nicht vom Sensor kommuniziert wurde. Obwohl es Ontologien für Zeit, wie die *Time Ontology in OWL* [169] gibt, werden einfache XML-Schema-Datentypen für das Modell verwendet. Dadurch werden weniger Tripel für die gleiche Information benötigt, was im gleichen Schritt zu einfacheren SPARQL-Anfragen führt. Außerdem lassen sich einige Konstrukte in der *Time Ontology in OWL* nicht mit einfachen SPARQL-Anfragen nutzen, sondern es wird ein spezieller temporaler Reasoner gebraucht. Dies ist eine Komponente, welche die zeitlichen Angaben auflöst und daraus SPARQL-abfragefähige Tripel generiert.

?state
[arb]/warm

Tabelle 5.3.: Ergebnis der Anfrage aus Quelltext 5.4.

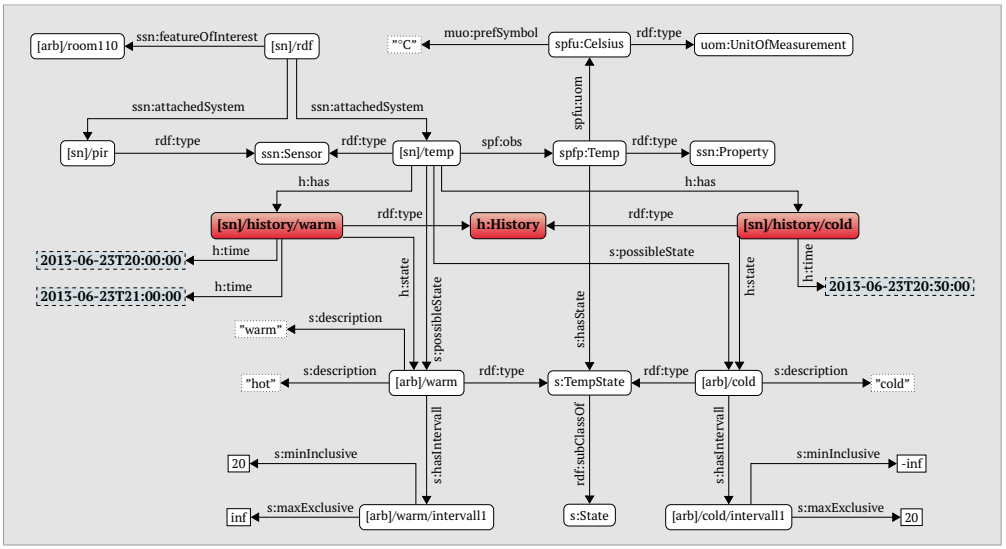


Abbildung 5.3.: RDF zur Beschreibung der Historie von Sensorzuständen.

```
1 SELECT ?datetime ?state WHERE
2 {
3  <[sn]/temp> h:has ?history .
4  ?history h:time ?datetime .
5  ?history h:state ?state .
6 }
7 ORDER BY DESC(?datetime)
8 LIMIT 1
```

Quelltext 5.5: Anfrage, um den letzten Zustand des Sensors abzurufen.

Der Aufbau der Historie erlaubt ein effizientes Hinzufügen neuer Einträge, da keine *UPDATE*- oder *DELETE*-Operationen ausgeführt werden müssen. Stattdessen ist es nur erforderlich, ein neues Tripel über eine *INSERT*-Operation einzufügen. Eine Beispielanfrage zum Abrufen des aktuellsten Historieneintrages ist in Quelltext 5.5 gezeigt. Das Ergebnis für die Anfrage auf den Beispielgraphen findet sich in Tabelle 5.4. Die Anfrage wählt alle Zeitpunkte der Historie und zugehörige Zustände aus (Zeile 1–6), sortiert diese nach der Zeit (Zeile 7) und gibt als Ergebnis nur die erste Zeile des Zwischenergebnisses zurück (Zeile 8).

?datetime	?state
2013-06-23T21:00:00	[arb]/warm

Tabelle 5.4.: Ergebnis der Anfrage aus Quelltext 5.5.

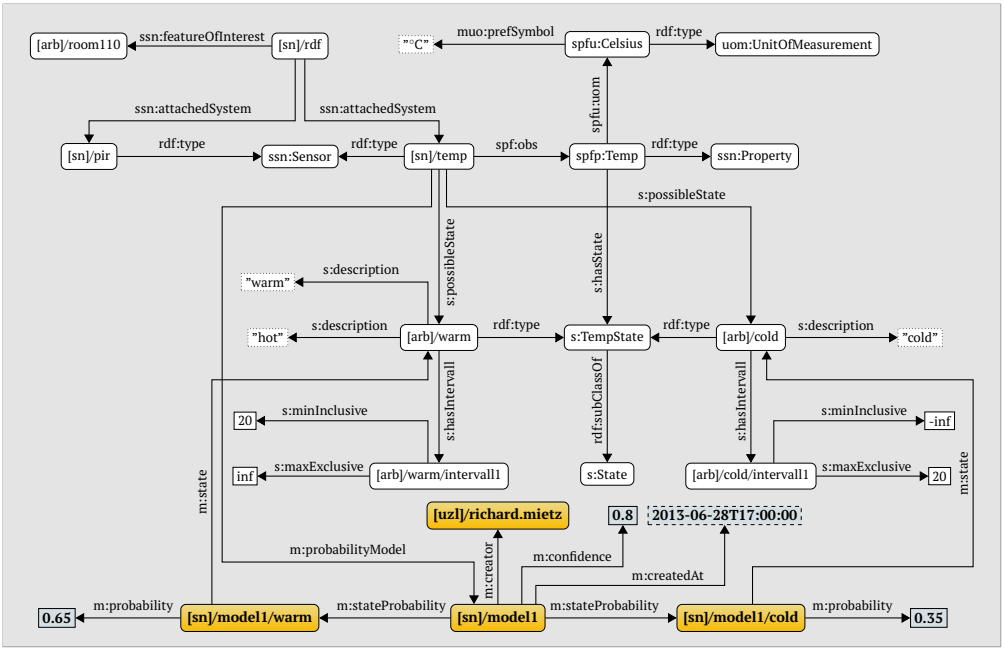


Abbildung 5.4.: RDF zur Beschreibung eines APMs.

```
SELECT ?prob WHERE
{
 <[sn]/temp> m:probabilityModel ?model .
 ?model m:stateProbability ?modelstate .
 ?modelstate m:state ?state .
 ?state s:description ?desc .
 ?modelstate m:probability ?prob .
 FILTER(?desc = "warm")
}
```

Quelltext 5.6: Anfrage, um die Wahrscheinlichkeit für den Zustand mit dem beschreibendem Begriff „warm“ von einem gegebenen Sensor abzurufen.

5.2.4. Basiswahrscheinlichkeitsmodell

In Abschnitt 4.3.1 wurde das APM eingeführt. Dieses dient als Basis, welches in den nächsten Abschnitten erweitert wird, um auch die anderen vorgestellten Modelltypen abbilden zu können.

Die Modellierung des APM ist in Abbildung 5.4 dargestellt. Der Übersichtlichkeit halber werden einige Elemente aus vorherigen Graphen nicht wiederholt. Das zentrale Element ist *[sn]/model1*. Durch *m:creator* und *m:createdAt* ist es möglich, den Ersteller und die Erstell- bzw. Aktualisierungszeit anzugeben. Des Weiteren erlaubt das Prädikat *m:confidence* einen Konfidenzwert im Intervall von 0 bis 1 anzugeben, der die Verlässlichkeit des Modells darstellt. So kann z. B. eine geringere Konfidenz angegeben werden, falls nicht alle historischen Daten bei der Berechnung der Wahrscheinlichkeit eingeflossen sind. Die Kon-

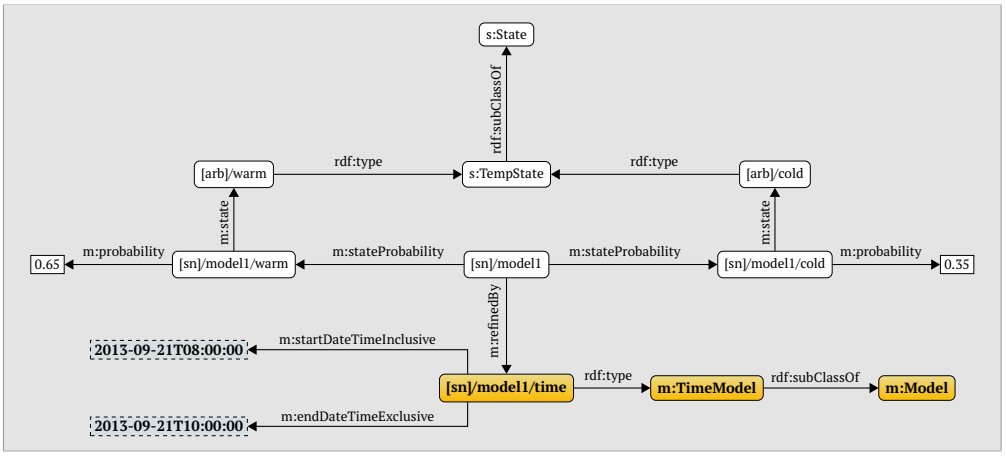


Abbildung 5.5.: RDF zur Beschreibung eines TPMs.

fidenz kann u. a. dazu genutzt werden, um nur Modelle mit einer Konfidenz über einem Schwellwert zu betrachten oder um die Wahrscheinlichkeiten durch Multiplikation mit dem Konfidenzwert zu skalieren. Die Wahrscheinlichkeiten sind Fließkommawerte, die per *m:probability* an ein Element angebunden sind, welches die Verknüpfung von Zustand und Modell widerspiegelt. Dieses zeigt per *m:state* auf den zugehörigen Zustand und wird vom Modell mit dem Prädikat *m:stateProbability* verlinkt. Das APM erlaubt eine Wahrscheinlichkeit pro Zustand. Es ist somit ein simples, jedoch ebenso eingeschränktes Modell. Dafür ist die Komplexität einer Anfrage nach einer Zustandswahrscheinlichkeit sehr gering. Quelltext 5.6 zeigt eine Anfrage, die für den Zustand *warm* als Ergebnis die Wahrscheinlichkeit zurückgibt.

5.2.5. Temporale Erweiterungen

Das vorgestellte APM ist nur hilfreich, sofern die Wahrscheinlichkeiten für die Zustände dauerhaft nahezu gleichverteilt sind. Oftmals sind die Wahrscheinlichkeiten aber abhängig von der Zeit, sei es von der Tageszeit oder vom Wochentag. So sind Parkplätze und Hörsäle an der Uni tagsüber an Wochentagen üblicherweise besetzt und an Wochenenden frei. Ebenso hat die Messkurve eines Temperatursensors im Außeneinsatz einen charakteristischen Tagesverlauf.

Daher gibt es zur Modellierung von zeitabhängigen Wahrscheinlichkeiten das TPM welches in Abschnitt 4.3.1 eingeführt wurde und in Abbildung 5.5 gezeigt ist. Das TPM verfeinert dabei die Aussage eines APMs, indem die Wahrscheinlichkeiten nur für einen gegebenen Zeitraum Gültigkeit besitzen. Im Beispiel ist dies zwischen 8 Uhr (inklusiv) und 10 Uhr morgens (exklusiv) am 21. September 2013. Diese Zeitpunkte werden über *m:startDateTimeInclusive* und *m:endDateTimeExclusive* angegeben. Diese gehen von einem Element aus, das mittels des Prädikats *m:refinedBy* an das APM gebunden wird. Dieses Prädikat erlaubt das APM auch durch andere Erweiterungen, von denen ein paar im wei-

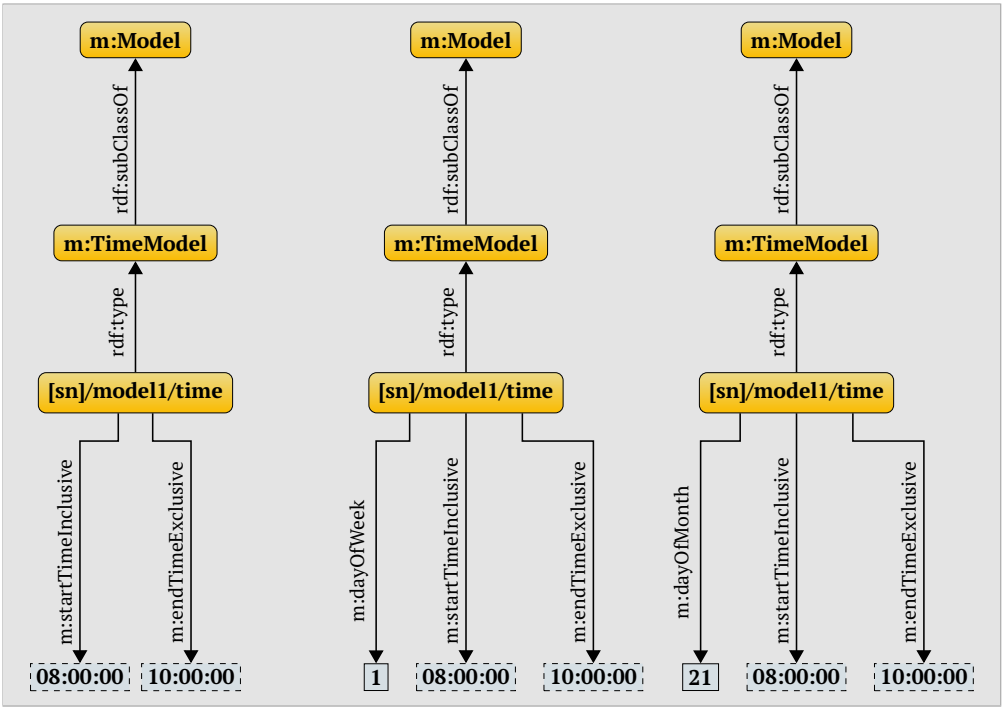


Abbildung 5.6.: Varianten von TPMs.

teren Verlauf vorgestellt werden, zu verfeinern. Das Element für eine zeitliche Verfeinerung ist vom Typ *m:TimeModel*, das wiederum eine Unterklasse von *m:Model* ist.

Das Modell in Abbildung 5.5 ist auf eine Zeit und ein Datum festgelegt. Um flexibler bei der Definition von Gültigkeitszeiträumen zu sein, existieren drei weitere Varianten, die in Abbildung 5.6 dargestellt sind. Diese erlauben

1. Zeiträume ohne Angabe eines bestimmten Datums,
2. Zeiträume mit Angabe für einen wiederkehrenden Wochentag und
3. Zeiträume mit Angabe für einen wiederkehrenden Monatstag

zu spezifizieren.

Durch die Einschränkung der Gültigkeit der Wahrscheinlichkeiten des TPMs in Abbildung 5.5 auf einen spezifischen Zeitraum, sind für andere Zeitpunkte keine Wahrscheinlichkeiten mehr definiert. Deshalb muss für jeden Zeitraum für den Wahrscheinlichkeiten verfügbar sein sollen, ein weiteres Modell zum Sensor hinzugefügt werden. Ein Beispiel ist in Abbildung 5.7 gegeben, bei dem der Sensor zwei Modelle mit unterschiedlichen Zeiträumen und Wahrscheinlichkeiten besitzt.

Zur Abfrage der gültigen Wahrscheinlichkeit benötigt man beim TPM zusätzlich die Suchzeit. Die Wahrscheinlichkeit für den Zustand *warm* um 11 Uhr kann mit der Anfrage aus Quelltext 5.7 abgerufen werden. Statt der Wahrscheinlichkeit

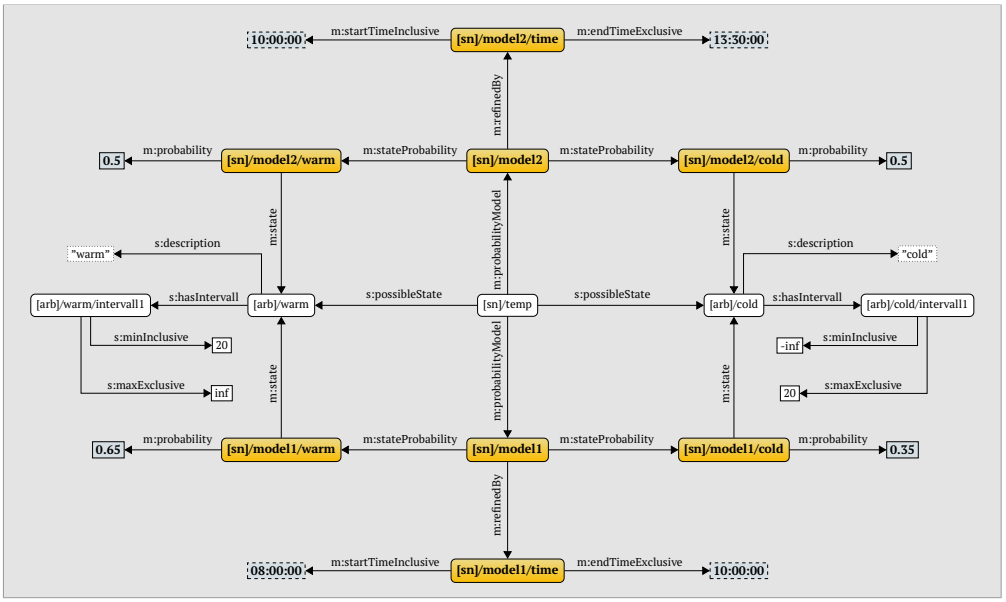


Abbildung 5.7.: Sensor mit zwei TPMs.

```
SELECT ?prob WHERE
{
 <[sn]/temp> m:probabilityModel ?model .
 ?model m:stateProbability ?modelstate .
 ?modelstate m:state ?state .
 ?state s:description ?desc .
 ?modelstate m:probability ?prob .
 ?model m:refinedBy ?refinement .
 ?refinement rdf:type m:TimeModel .
 ?refinement m:startTimeInclusive ?start .
 ?refinement m:endTimeExclusive ?end .
 FILTER(?desc = "warm" && ?start <= "11:00:00"^^xsd:time && "
     11:00:00"^^xsd:time < ?end)
}
```

Quelltext 5.7: Anfrage, um die Wahrscheinlichkeit für den Zustand mit dem Begriff „warm“ um 11 Uhr morgens von einem gegebenen Sensor abzurufen.

für einen gegebenen Sensor ist jedoch eine Liste von Sensoren mit den zugehörigen Wahrscheinlichkeiten das Ziel. Dafür muss die Anfrage entsprechend Quelltext 5.8 abgeändert werden.

Während die Zustandswahrscheinlichkeiten von einigen Sensoren gut durch APMs abgebildet werden können, kann es für andere Sensoren sinnvoll sein TPMs oder eine der nachfolgenden Modellvarianten zu nutzen. Auf der anderen Seite soll eine Suchanfrage in der Lage sein jeglichen Sensor und seine Zustandswahrscheinlichkeit zu selektieren, unabhängig davon, welches Modell er nutzt. Nutzt man allerdings die Abfrage für APMs, bezieht man nicht die temporalen Eigenschaften ein und selektiert sogar mehrere Wahrscheinlichkeiten für Sensoren mit TPM, da sie Tripel mehrfach enthalten. Nutzt man dagegen

```
SELECT ?sensor ?prob WHERE
{
 ?sensor m:probabilityModel ?model .
 ...
}
ORDER BY DESC(?prob)
```

Quelltext 5.8: Notwendige Änderung für die Anfrage aus Quelltext 5.7.

```
SELECT ?sensors ?prob WHERE
{
 ?sensor m:probabilityModel ?model .
 ?model m:stateProbability ?modelstate .
 ?modelstate m:state ?state .
 ?state s:description ?desc .
 ?modelstate m:probability ?prob .
 OPTIONAL { ?model m:refinedBy ?refinement . }
 OPTIONAL { ?refinement rdf:type m:TimeModel .
   ?refinement m:startTimeInclusive ?start .
   ?refinement m:endTimeExclusive ?end .
      FILTER (?start <= "11:00:00"^^xsd:time && "11:00:00"^^xsd:
          time < ?end) }
 FILTER(?desc = "warm")
}
ORDER BY DESC(?prob)
```

Quelltext 5.9: Anfrage, um alle Sensoren zu selektieren, die ein TPM oder APM besitzen.

die Anfrage für TPMs, werden Sensoren mit APM nicht berücksichtigt, da sie die entsprechenden Tripel nicht enthalten. Aufgrund dessen müssen alle Tripel, die durch Modellerweiterungen eingeführt und durch Tripelmuster abgefragt werden, in *OPTIONAL*-Blöcke integriert werden. Damit ergibt sich bis hierher die Anfrage aus Quelltext 5.9, um sowohl APMs als auch TPMs einzubeziehen.

5.2.6. Räumliche Erweiterungen

TPMs erlauben die Wahrscheinlichkeiten zeitlich einzuschränken, d. h. sie besitzen die Zeit als weiteren Parameter. Allerdings kann es auch Sensoren bzw. deren beobachtete Entitäten geben, deren Zustandswahrscheinlichkeiten von der Örtlichkeit abhängen. Als Beispiel kann man sich ein Taxi vorstellen, das seine Belegung, d. h. ob es frei oder besetzt ist, im Internet bereitstellt. Das Taxi kann als Sensor aufgefasst werden. Wenn es Personen zu einem Konzert bringt, ist es nach dem Absetzen an der Konzerthalle frei, während es auf dem Weg dorthin belegt ist.

Abbildung 5.8 zeigt die räumliche Verfeinerung, das PPM, eines Modells. PPMs besitzen somit den Ort als Parameter. Das Hauptelement wird, wie beim TPM, mit *m:refinedBy* an das Modell gebunden und ist vom Typ *m:PlaceModel*. Außer-

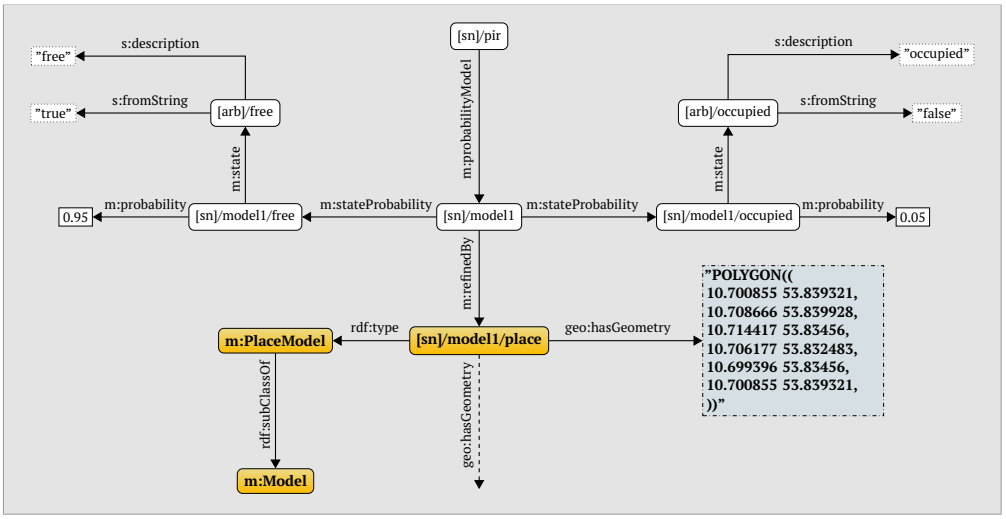


Abbildung 5.8.: RDF zur Beschreibung eines PPMs.

dem können eine oder mehrere Geometrien mit Hilfe des GeoSPARQL-Prädikats *geo:hasGeometry* spezifiziert werden. Die Wahrscheinlichkeiten gelten dann nur innerhalb dieser Geometrien. Im Beispiel ist die Geometrie ein Polygon, welches die Universität zu Lübeck überdeckt.

Wenn man das Beispiel des Taxis nicht zum Zeitpunkt des Bringens, sondern des Abholens vom Konzert betrachtet, ist zu vermuten, dass sich die Wahrscheinlichkeiten für die Belegung vom ersten Beispiel unterscheiden, da nun viele Leute nach Hause wollen und ein Taxi daher schnell belegt ist. Es wäre daher sinnvoll, wenn Modelltypen miteinander kombinierbar sind, um solche Eigenschaften abzubilden. Dies ist mit den vorliegenden Modellen ohne Modifikation möglich. Verschiedene Modelltypen können parallel an das Standardmodell mittels *m:refinedBy* angebunden werden.

Der Quelltext 5.10 zeigt, wie das PPM abgefragt werden kann, die Abfrage jedoch weiterhin kompatibel mit dem APM, dem TPM oder Verknüpfungen davon bleibt. Dazu muss zum einen ein *OPTIONAL*-Block der Anfrage hinzugefügt werden, welcher die PPM-typischen Tripel abfragt (Zeile 13–14) und zum anderen muss der Suchort integriert werden. Ein *FILTER* (Zeile 15), der mittels der GeoSPARQL-Funktion *geof:within* prüft, ob der gegebene Suchort in Form einer Geokoordinate innerhalb einer der Geometrien des Modells liegt, selektiert nur örtlich zutreffende PPMs.

5.2.7. Erweiterung um Korrelationen

Die beiden letzten Erweiterungen behandeln Korrelationen. Die Abbildungen 5.9 und 5.10 zeigen Beispiele für ein intra-CPM und ein inter-CPM. Bei ersterem wird die Wahrscheinlichkeit für einen Zustand mit einem vorherigen Zustand desselben Sensors verknüpft. Beim zweiten dagegen hängt sie vom Zustand

```
SELECT ?sensors ?prob WHERE
{
 ?sensor m:probabilityModel ?model .
 ?model m:stateProbability ?modelstate .
 ?modelstate m:state ?state .
 ?state s:description ?desc .
 ?modelstate m:probability ?prob .
 OPTIONAL { ?model m:refinedBy ?refinement . }
 OPTIONAL { ?refinement rdf:type m:TimeModel .
   ?refinement m:startTimeInclusive ?start .
   ?refinement m:endTimeExclusive ?end .
      FILTER (?start <= "11:00:00"^^xsd:time && "11:00:00"^^xsd:
          time < ?end) }
  OPTIONAL { ?refinement rdf:type m:PlaceModel .
   ?refinement geo:hasGeometry ?place .
      FILTER (geof:within("POINT(53.83379␣10.70425)"^^geo:
          wktLiteral,?place)) }
 FILTER(?desc = "warm")
}
ORDER BY DESC(?prob)
```

Quelltext 5.10: Anfrage, um alle Sensoren zu selektieren, die PPMs, TPMs oder ein APM besitzen.

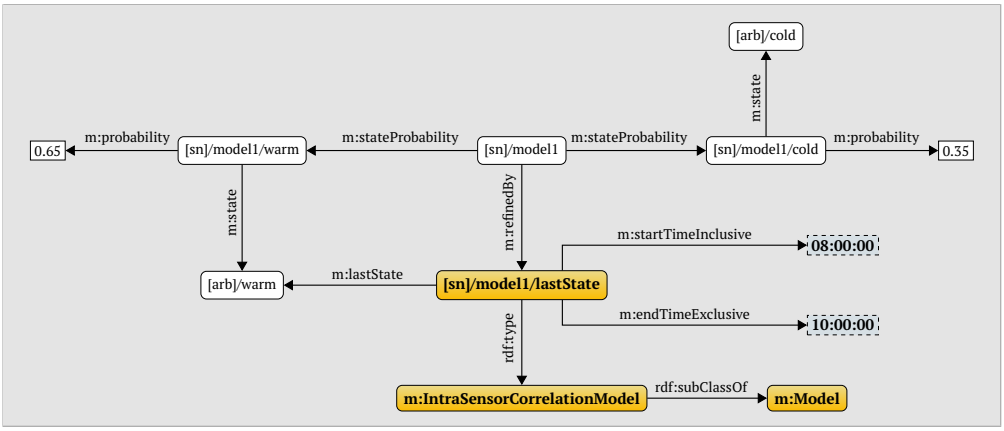


Abbildung 5.9.: RDF zur Beschreibung von intra-Sensor-Korrelationen.

eines Sensors vom gleichen Typ auf einem anderen Sensorknoten zur selben Zeit ab. Eine Beispielanfrage für das intra-CPM ist in Quelltext 5.11 gegeben. Die Zeilen 9–13 selektieren die modellspezifischen Tripel. Die *FILTER*-Anweisung in Zeile 14 ermittelt die Modelle, bei denen bekannt ist, dass der Zustand um 9:05 Uhr *warm* war.

5.2.8. Generelle Erweiterbarkeit

Ein wichtiger Aspekt des Semantic Web ist, dass jeder Nutzer eigene Ontologien und Erweiterungen beitragen kann. Best Practice ist dabei auf bestehenden

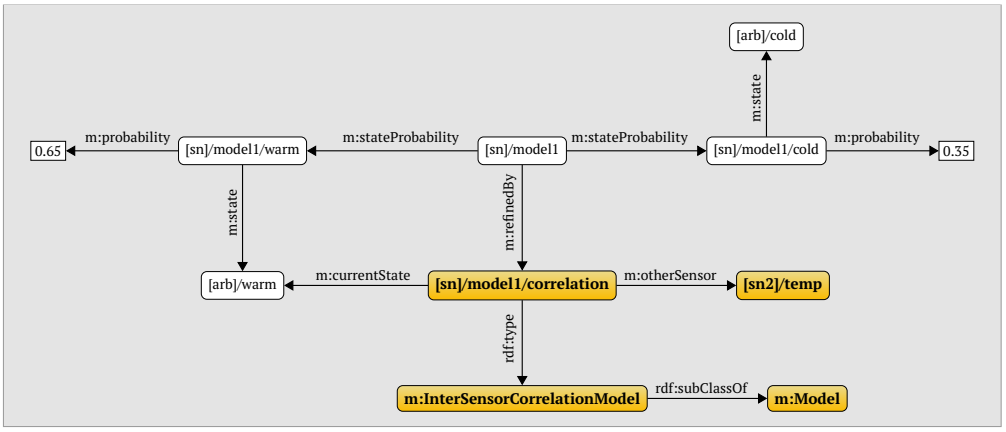


Abbildung 5.10.: RDF zur Beschreibung von inter-Sensor-Korrelationen.

```
SELECT ?sensors ?prob WHERE
{
 ?sensor m:probabilityModel ?model .
 ?model m:stateProbability ?modelstate .
 ?modelstate m:state ?state .
 ?state s:description ?desc .
 ?modelstate m:probability ?prob .
 OPTIONAL { ?model m:refinedBy ?refinement . }
 OPTIONAL { ?refinement rdf:type m:IntraCorrelationModel .
   ?refinement m:startTimeInclusive ?lastStateStart .
   ?refinement m:endTimeExclusive ?lastStateEnd .
   ?refinement m:lastState ?lastState .
   ?lastState s:description ?lastStateDesc .
      FILTER (?lastStateDesc = "warm" && ?lastStateStart <= "
          09:05:00"^^xsd:time && "09:05:00"^^xsd:time < ?
          lastStateEnd) }
 FILTER( ?desc = "warm")
}
ORDER BY DESC(?prob)
```

Quelltext 5.11: Anfrage für das intra-CPM.

Konzepten aufzubauen bzw. diese zu integrieren, anstatt ein eigenes Konzept zu entwickeln, das viele Ähnlichkeiten hat, aber nicht kompatibel ist. Deshalb ist das vorgestellte Modell so konzipiert, dass es einfach erweiterbar ist.

Die Zustandswahrscheinlichkeiten von Temperatur-, Luftfeuchte- und Windsensoren können z. B. vom Wetter abhängen, welches durch eine Erweiterung modellierbar ist. Solche neuen Verfeinerungen für das Wahrscheinlichkeitsmodell lassen sich in ein paar Schritten hinzufügen:

1. Das Hauptelement der Verfeinerung muss per *m:refinedBy* angebunden werden.
2. Das Hauptelement muss von einem Typ sein (*rdf:type*), der eine Unterklasse von *m:Model* ist.

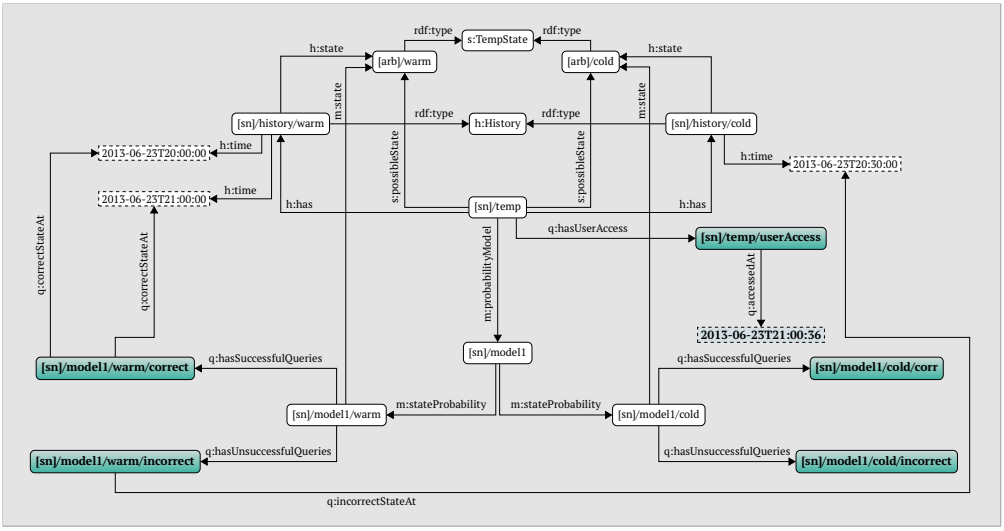


Abbildung 5.11.: RDF zur Beschreibung von Zustandsabfragen.

3. Beliebige Tripel, welche die Verfeinerung beschreiben, müssen an das Hauptelement angefügt werden.
4. SPARQL-Anfragen müssen um einen *OPTIONAL*-Block ergänzt werden, der die Tripel der Verfeinerung abfragt.
5. Dem *OPTIONAL*-Block muss ein *FILTER* hinzugefügt werden, der nur Modelle selektiert, die aufgrund der Eigenschaft der Verfeinerung gerade Gültigkeit besitzen.

5.2.9. Zustandsabfrage

Nachdem die Sensoren und deren Zustandswahrscheinlichkeiten mit Hilfe einer SPARQL-Abfrage selektiert und nach absteigender Wahrscheinlichkeit sortiert sind, können diese als Ergebnis dem Nutzer der Suchmaschine präsentiert werden. Allerdings ist es für diesen nützlicher, wenn er den tatsächlichen aktuellen Zustand statt einer Wahrscheinlichkeit als Ergebnis erhält. Deshalb soll die Suchmaschine als nächsten Schritt die Sensoren in absteigender Reihenfolge der Zustandswahrscheinlichkeit für deren aktuellen Zustand kontaktieren (vgl. Sensor Ranking [18]).

Es wird als Beispiel ein Temperatursensor mit den möglichen Zuständen *warm* und *kalt* betrachtet. Eine Suchmaschine kontaktiert diesen, weil sie nach Sensoren sucht, die *warm* messen. Als Ergebnis kann der Sensor einen Sensorwert liefern, der in das Intervall des Zustandes *warm* fällt oder direkt den Zustand. Dies ist somit eine erfolgreiche Anfrage. Liefert der Sensor dagegen *kalt* zurück, so ist die Anfrage erfolglos.

Die Modellerweiterung in Abbildung 5.11 zeigt, wie sich diese erfolgreichen und erfolglosen Anfragen an einen Sensor in RDF abbilden lassen. Jedes Modell

hat ein Element für einen der möglichen Zustände. An dieses werden mittels *q:hasSuccessfulQueries* und *q:hasUnsuccessfulQueries* Elemente angebunden, die erfolgreiche und erfolglose Anfragen speichern. Dies geschieht über die Prädikate *q:correctStateAt* und *q:incorrectStateAt*, die auf Zeitstempel-Literale verweisen. Diese Zeitstempel können gleichzeitig für die Historie genutzt werden, da der Sensor zu diesem Zeitpunkt diesen Zustand gemessen hat. In dem Beispiel gab es um 20:00 und 21:00 Uhr erfolgreiche Anfragen und um 20:30 Uhr eine erfolglose Anfrage für den Zustand *warm*. Anfragen für den Zustand *cold* wurden nicht durchgeführt.

Die Angaben aus Abbildung 5.11 lassen sich z. B. dazu nutzen mit der Gleichung 5.1 einen Konfidenzwert zu berechnen. Dabei wird das Verhältnis von erfolgreichen Anfragen cq_i zur Gesamtzahl an Anfragen (erfolgreiche Anfragen cq_i und erfolglose Anfragen iq_i) über die Menge aller Zustände $\mathcal{Z}$ aufaddiert und normiert, so dass ein Wert zwischen 0 (nur erfolglose Anfrage) und 1 (nur erfolgreiche Anfragen) ermittelt wird.

$$\text{confidence} = \frac{1}{|\mathcal{Z}|} \sum_{i=1}^{|\mathcal{Z}|} \frac{|cq_i|}{|cq_i| + |iq_i|} \tag{5.1}$$

Abbildung 5.11 zeigt weiterhin, wie der Zugriff durch einen Nutzer auf einen Sensor durch das Prädikat *q:accessedAt* modelliert wird. Dies ist vergleichbar mit dem Aufruf eines Ergebnisses, analog zum Besuch einer Webseite bei der Suche mit einer heutigen Suchmaschine. Diese protokollieren, welche Webseiten angeschaut werden, um die Qualität der präsentierten Ergebnisse zu analysieren, d. h. ob dem Nutzer nützliche Resultate angezeigt werden. Auch in einer IoT-Suchmaschine kann diese Information von Interesse sein. Zum einen als Rankingfaktor, um oft zugegriffene Sensoren höher zu bewerten. Zum anderen auch für das Gegenteil, wenn z. B. ein Sensor, der einen freien Parkplatz anzeigt, zuletzt oft angefragt wurde. Dies kann bedeuten, dass der Parkplatz bald besetzt ist. Entsprechend sollten die Zugriffe als Abwertungskriterium genutzt werden. Die Verwendung dieser Information hängt somit individuell vom Szenario ab.

5.2.10. Rankingfaktoren

Sobald eine Menge von Sensoren als Ergebnis auf eine Nutzeranfrage feststeht, muss diese dem Nutzer präsentiert werden. Dabei ist die Reihenfolge auf der Ergebnisseite, das sogenannte Ranking, von entscheidender Bedeutung. Der Nutzer möchte auf der ersten Seite und am besten in den ersten Treffern ein Ergebnis finden, das seinen Ansprüchen genügt. Diese Zufriedenstellung des Benutzers ist wiederum wichtig für Suchmaschinenbetreiber, denn nur bei Sucherfolg wird eine Suchmaschine dauerhaft verwendet.

Ergebnisse können nach unterschiedlichen Kriterien sortiert werden. Heutige Suchmaschinen beziehen zum Teil hunderte Faktoren in die Berechnung der

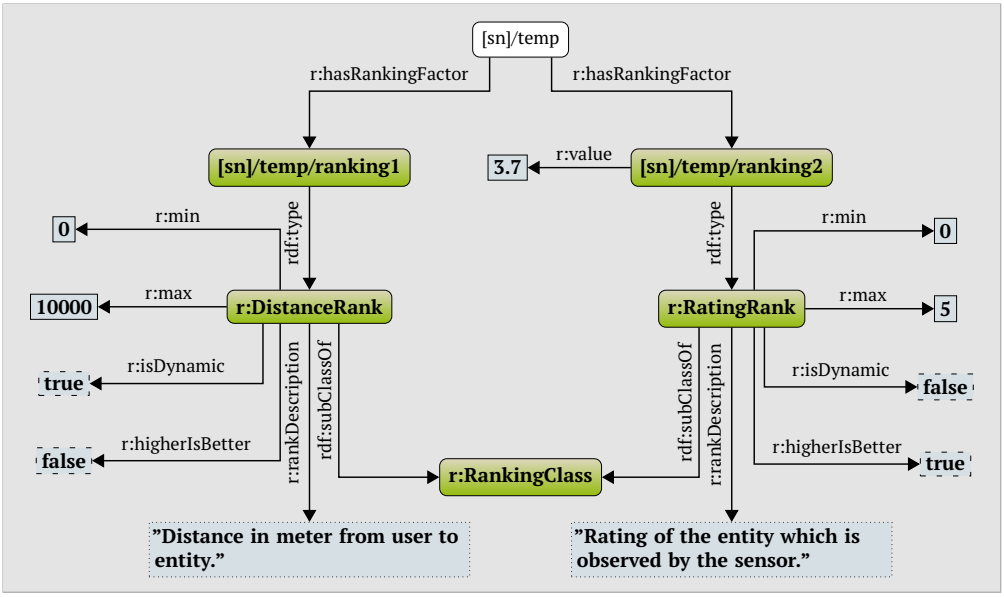


Abbildung 5.12.: RDF zur Beschreibung von Rankingfaktoren für einen Sensor.

Seitenebene	Domainebene
Gesamtvorkommen des Schlüsselwortes	Schlüsselwort ist Domainname
Schlüsselwort in Titel	Schlüsselwort ist Teil des Domainnamens
Schlüsselwort in Überschrift	Schlüsselwort ist erstes Wort des Domainnamens
Schlüsselwort in Metadaten	Schlüsselwort ist Subdomainname
Flexion von Schlüsselwort	Schlüsselwort ist Teil des Subdomainnamens
Nähe von Schlüsselwörtern	

Tabelle 5.5.: Keyword-basierte Rankingfaktoren [nach 170].

Ergebnisreihenfolge ein. Dabei hat jede Suchmaschine ihren eigenen, geheimen Algorithmus, der unterschiedliche Faktoren einbezieht und gewichtet. Einige der bekannten Faktoren sind Tabelle 5.5, 5.6 und 5.7 zu entnehmen. Dies ist nur ein kleiner Ausschnitt der genutzten Informationen. Sensoren dagegen besitzen meist nur wenige Informationen. Die dem Modell zugrundeliegende Ontologie speichert für einen Sensor vor allem technische Daten (z. B. Arbeitsbereich, Genauigkeit und Zuverlässigkeit), die für den Nutzer meist von geringem Interesse sind. Die Antworten zur Frage über die Sortierkriterien aus Abbildung 3.13 in Abschnitt 3.2.2 zeigen, dass es bei der Echtzeitsuche nach dynamischen Daten in mobilen Kontexten vor allem auf Faktoren wie die Aktualität der Daten und die geografische Position bzw. Entfernung vom Suchenden ankommt. Andererseits ist das Ergebnis nicht eindeutig. Je nach Szenario schwankt die Wichtigkeit der Rankingfaktoren. Deshalb ist es unverzichtbar, die Faktoren flexibel gewichten zu können.

Seitenebene	Domainebene
Anzahl von Links in Webseite	Anzahl unterschiedlicher Domains, die auf die Webseite verweisen
Anzahl von IP-Adressen, die auf die Webseite verlinken	Anzahl unterschiedlicher Uniform Resource Locators (URLs) die zu der Domain verlinken
Anzahl externer Links	Thematische Relevanz der Domain die zu der Domain verlinkt
% externer Links	„Distanz“ der Domain von einer Menge vertrauenswürdiger Domains in Hops
Anzahl interner Links zu der Webseite	

Tabelle 5.6.: Link-basierte Rankingfaktoren [nach 170].

Seitenebene	Domainebene
Länge der Webseite	Länge des Domainnamen
Einzigartigkeit des Webseiteninhalts	Einzigartigkeit des Domaininhalts
Alter der Webseite	Zeit seit dem die Domain registriert wurde
Aktualität der Webseite	Zeit seit die Registrierungsdetails der Domain sich geändert haben
Ladegeschwindigkeit	Ladegeschwindigkeit über die gesamte Domain
Nutzung von Werbung	Anzahl an Links zu Fehlerseiten von der Domain
Validierungsfehler	Anzahl an Fehlerseiten unter dieser Domain

Tabelle 5.7.: Weitere Rankingfaktoren [nach 170].

Abbildung 5.12 zeigt das Modell für Rankingfaktoren. Ein Sensor kann beliebig viele Faktoren besitzen, die über das Prädikat *r:hasRankingFactor* angegeben werden. Jeder Rankingfaktor ist von einem bestimmten Typ, der eine Subklasse von *r:RankingClass* ist. Sein möglicher Minimal- und Maximalwert werden mit *r:min* und *r:max* angegeben. Ist das Objekt, das mit dem Prädikat *r:isDynamic* angebunden wird, *false*, so bedeutet dies, dass der Wert bereits im RDF-Graphen vorliegt. Bei *true* dagegen ist der Wert erst zur Suchzeit bekannt. Im Beispielgraphen gibt es zwei Faktoren. Der eine gibt einen Wert an, der eine Bewertung der überwachten Entität zwischen 0 und 5 darstellt. Derzeit hat die Entität eine Bewertung von 3,7. Der zweite Faktor ist die Distanz in Metern, die hier beispielhaft zwischen 0 und 10.000 liegen kann. Erst zur Suchzeit ist der Ort des Suchenden bekannt. Somit lässt sich auch die Distanz erst zu diesem Zeitpunkt auswerten. Wird mit dem Prädikat *r:higherIsBetter* ein *true* angegeben, so sind höhere Werte wie beim Rating besser. Dagegen sind bei einem *false* niedrigere Werte wie bei der Distanz vorteilhafter. Die Gleichung 5.2 gibt an, wie der Rankingwert rv_s für den Sensor s aus der Menge aller Sensoren der Ergebnismenge $\mathcal{S}$ aus den Angaben berechnet wird. Jeder Rankingfaktor r_i aus der Menge aller Rankingfaktoren $\mathcal{R}_s$ des Sensors s wird mit einem für den Faktor gegebenen Wert w_i gewichtet. Diese Gewichte können entweder durch den Nutzer gegeben

$$rv_s = \frac{1}{\mathcal{N}} \sum_{i=1}^{|\mathcal{R}_s|} w_i * r_i \tag{5.2}$$

$$\text{mit } r_i = \begin{cases} \dfrac{s(v_i) - r^{min}}{r_i^{max} - r_i^{min}} & \text{für } \xrightarrow{\text{r:higherIsBetter}} \boxed{\text{true}} \\ 1 - \left(\dfrac{s(v_i) - r^{min}}{r_i^{max} - r_i^{min}} \right) & \text{für } \xrightarrow{\text{r:higherIsBetter}} \boxed{\text{false}} \end{cases}$$

$$\text{mit } s(v) = max(r_i^{min}, min(r_i^{max}, v))$$

oder durch die Suchmaschine definiert werden. Durch $\frac{1}{\mathcal{N}}$ wird die Summe der Werte normiert. Diese Normierung kann mit der Anzahl von für den Sensor vorhandenen Rankingfaktoren geschehen. Dafür ist $\mathcal{N} = |\mathcal{R}_s|$ zu setzen. Alternativ lässt sich das Maximum der Anzahl an Faktoren, die ein Sensor der Ergebnismenge besitzt, verwenden. Dies ist durch $\mathcal{N} = \arg\max_{s \in \mathcal{S}} |\mathcal{R}_s|$ möglich. Je weniger Rankingfaktoren ein Sensor besitzt, desto schlechter würde dadurch sein Rankingwert. Der Hintergrund ist eine mögliche Manipulation, d. h. ein Verbessern des Rankings für einen Sensor durch Löschen bzw. Nicht-Hinzufügen von negativen Rankingfaktoren. Die Berechnung der Werte r_i ist abhängig vom booleschen Wert, der mit dem Prädikat *r:higherIsBetter* angegeben wird. Die für den Rankingfaktor spezifizierten Intervallgrenzen sind durch r_i^{min} und r_i^{max} gegeben. Der Wert r_i kann die Zahlen von 0 bis 1 annehmen, je nachdem, ob v_i den schlechtesten bzw. besten Wert des Faktors annimmt. Durch die Anwendung der Funktion $s(v)$ werden die Werte die außerhalb des für den Rankingfaktor angegebenen Intervalls liegen, auf den minimalen bzw. maximal gültigen Wert gesetzt.

5.2.11. Evaluation

Durch die Nutzung von RDF ist eine gute strukturelle Skalierbarkeit gewährleistet, da das Hinzufügen neuer Tripel ohne Probleme möglich ist. Bei der Abbildung von Konzepten in RDF ist auf der einen Seite eine hohe Flexibilität für Erweiterungen gewünscht. Auf der anderen Seite sollten möglichst wenige Tripel benötigt werden. Eine geringere Anzahl Tripel pro Konzept bedeutet kompaktere SPARQL-Anfragen, d. h. weniger Tripelmuster, für dieses Konzept. Dies ist vor allem im Kontext von verteilten Datenbanken, bei der zur Bearbeitung von SPARQL-Anfragen Tripel von verschiedenen Systemen abgerufen werden müssen, wichtig, um eine effektive Antwort gewährleisten zu können.

Tabelle 5.8 listet die verschiedenen vorgestellten Konzepte, die Anzahl der notwendigen Tripel und die Variablen, von denen diese Anzahl abhängig ist, auf. Die Menge an Tripeln ist entweder konstant oder hängt linear von den

Konzept	Abhängige Variabel	Anzahl Tripel
Ein global definierter Zustand (Intervalle)	# beschreibender Terme *nr_terms*, # Intervalle *nr_ints*	$1 + nr_terms + 3 \times nr_ints$
Ein global definierter Zustand (Strings)	# beschreibender Terme *nr_terms*, # String *nr_strings*	$1 + nr_terms + nr_strings$
Zustände der Sensoren	# Sensorzustände *nr_states*	$2 \times nr_states$
Historie für Zustand	# Zeitpunkte *nr_times* in welcher der Sensor im Zustand der Historie war	$3 + nr_times$
APM	*nr_states*	$nr_apm = 4 + 3 \times nr_states$
TPM	# Gültigkeitsperioden *nr_periods*	$nr_apm + nr_periods \times (2 + [2; 3])$
PPM	# Regionen *nr_regions*	$nr_apm + 2 + nr_regions$
intra-CPM	–	$nr_apm + 5$
inter-CPM	–	$nr_apm + 4$
Zustandsabfrage	# Sensorzustände *nr_states*, # Anfragen *nr_queries*	$2 \times nr_states + nr_queries$
Ein global definierter Rankingfaktor	–	6
Rankingfaktoren der Sensoren	# dynamischer Rankingfaktoren *nr_dyn_factors*, # statischer Rankingfaktoren *nr_static_factor*	$2 \times nr_dyn_factor + 3 \times nr_static_factor$

Tabelle 5.8.: Anzahl benötigter RDF-Tripel für die verschiedenen Modellkonzepte.

Variablen ab. Der höchste lineare Faktor ist 5, während die meisten im Bereich zwischen 1 und 3 liegen. Ebenso ist der höchste konstante Faktor mit 6 relativ klein. Somit ist eine gute räumliche Skalierbarkeit garantiert, da durch das Hinzufügen von neuen Elementen nur wenige neue Tripel benötigt werden.

5.2.12. Fazit

Mit der vorgestellten semantischen Modellierung lassen sich Metadaten, Zustände, Zustandshistorie, Wahrscheinlichkeitsmodelle, Rankingfaktoren und Zugriffsmuster von Sensoren mit einer geringen Anzahl von Tripeln abspeichern. Durch die Verwendung von RDF und existierenden Ontologien ist eine einfache Erweiterbarkeit gewährleistet.

5.3. Wahrscheinlichkeitsmodellgenerierung mittels SPARQL

In Abschnitt 4.3 wurden bereits zwei Verfahren zur Generierung von Wahrscheinlichkeitsmodellen auf Sensorknoten anhand lokal verfügbarer Daten vorgestellt. Es wurden allerdings auch Nachteile dieses Vorgehens genannt. Diese sind sowohl die geringen verfügbaren Speicher- und Rechenkapazitäten als auch die Beschränkung auf die Verwendung der lokalen Daten. Deswegen

```
1  CONSTRUCT
2  {
3     [sn]/temp m:probabilityModel [sn]/model1 .
4     [sn]/model1 m:creator [uzl]/richard.mietz .
5     [sn]/model1 m:confidence "1.0"^^xsd:double .
6     [sn]/model1 m:stateProbability [sn]/model1/warm .
7     [sn]/model1 m:stateProbability [sn]/model1/cold .
8     [sn]/model1/warm m:state [arb]/warm .
9     [sn]/model1/cold m:state [arb]/cold .
10    [sn]/model1/warm m:probability ?percWarm .
11    [sn]/model1/cold m:probability ?percCold .
12 }
13 WHERE
14 {
15    SELECT (COUNT(?warm) AS ?warmCount)
16           (COUNT(?cold) AS ?coldCount)
17           (COUNT(?state) AS ?stateCount)
18           (100/xsd:double(?stateCount)*xsd:double(?warmCount) AS
                ?percWarm)
19           (100/xsd:double(?stateCount)*xsd:double(?coldCount) AS
                ?percCold)
20    {
21       [sn]/history/warm h:time ?warm .
22       [sn]/history/cold h:time ?cold .
23       [sn]/temp h:has ?hist .
24       ?hist h:time ?state .
25    }
26 }
```

Quelltext 5.12: SPARQL-Anfrage zur Erzeugung eines APMs.

wurde als Alternative die Generierung im Backend diskutiert und anhand eines inter-CPM demonstriert (vgl. Abschnitt 4.3.2). Anknüpfend folgt ein zweites Verfahren für diese Art der Erstellung.

Angenommen, Sensordaten haben das in Kapitel 5 beschriebenem Format, dann können daraus mit SPARQL-CONSTRUCT-Anfragen Wahrscheinlichkeitsmodelle im Backend generiert werden. Der Quelltext 5.12 zeigt eine SPARQL-Anfrage, die ein APM für einen Sensor mit den Zuständen *warm* und *kalt* generiert. Die Zeilen 21–24 selektieren die Zeitpunkte, zu denen der Zustand *warm*, *kalt* und allgemein einer der Zustände gemessen wurde. Die Häufigkeiten der Vorkommnisse werden in den Zeilen 15–17 gezählt und an die Variablen *?warmCount*, *?coldCount* und *?stateCount* gebunden. Daraus werden wiederum in den Zeilen 18 und 19 die prozentualen Häufigkeiten für die beiden Zustände berechnet und an die Variablen *?percWarm* und *?percCold* gebunden. Dies sind die Wahrscheinlichkeiten des APM für den Sensor mit zwei Zuständen. Die Zeilen 1–12 erzeugen mittels *CONSTRUCT* die RDF-Tripel, die ein APM beschreiben. Diese müssen nur noch in den Triplestore eingefügt werden.

Die SPARQL-Anfrage für ein TPM ist etwas aufwendiger. In Quelltext 5.13 ist die Berechnung der Wahrscheinlichkeiten für den Zeitslot von 15 bis 20 Uhr zu sehen. Die ersten Zeilen sind der Übersichtlichkeit halber weggelassen. Hier

```
1  [...]
2  WHERE
3  {
4   SELECT (COUNT(?warmTime) AS ?warmCount)
5       (COUNT(?coldTime) AS ?coldCount)
6       (COUNT(?stateTime) AS ?stateCount)
7       (100/xsd:double(?stateCount)*xsd:double(?warmCount) AS ?
            percWarm)
8       (100/xsd:double(?stateCount)*xsd:double(?coldCount) AS ?
            percCold)
9   {
10     [sn]/history/warm h:time ?warm .
11     [sn]/history/cold h:time ?cold .
12     [sn]/temp h:has ?hist .
13     ?hist h:time ?state .
14     BIND(strdt(str(substr(?warm,12)),xsd:time) AS ?warmTime) .
15     BIND(strdt(str(substr(?cold,12)),xsd:time) AS ?coldTime) .
16     BIND(strdt(str(substr(?state,12)),xsd:time) AS ?stateTime) .
17     FILTER ("15:00:00"^^xsd:time <= ?warmTime && "20:00:00"^^xsd
           :time > ?warmTime) .
18     FILTER ("15:00:00"^^xsd:time <= ?coldTime && "20:00:00"^^xsd
           :time > ?coldTime) .
19     FILTER ("15:00:00"^^xsd:time <= ?stateTime && "20:00:00"^^
           xsd:time > ?stateTime) .
20  }
21 }
```

Quelltext 5.13: Teil einer SPARQL-Anfrage zur Erzeugung eines Teils eines TPMs.

werden ähnlich wie in Quelltext 5.12 die RDF-Tripel für ein TPM erzeugt. Die Wahrscheinlichkeiten lassen sich ebenfalls ähnlich berechnen. Zeile 10–12 selektiert die Zeiten, zu denen die Zustände gemessen wurden. In den Zeilen 14–16 wird jeweils aus den Zeitstempeln, die aus Datum und Uhrzeit bestehen, die Uhrzeit extrahiert und an Variablen gebunden. Anschließend werden in Zeile 17–19 alle Vorkommen, die außerhalb des Zeitslots liegen, ausgefiltert. Eine Anfrage kann somit nur die Tripel für einen Zeitslot erzeugen. Für jeden weiteren Zeitslot wird eine entsprechend der Zeit angepasste Anfrage benötigt. Alternativ können die einzelnen Teile per *UNION* in einer Anfrage zusammengefasst werden.

Anfragen für die anderen in Kapitel 5 besprochenen Wahrscheinlichkeitsmodelle sind ebenfalls möglich.

5.3.1. Fazit

Es wurde, aufbauend auf der in Abschnitt 5.2 präsentierten Modellierung, dargelegt, wie mit einfachen SPARQL-Anfragen Wahrscheinlichkeitsmodelle erzeugt werden können. Dies wurde exemplarisch am Beispiel von APMs und TPMs gezeigt. Anfragen für andere Modelltypen sind ebenso leicht umsetzbar. Die Generierung im Backend hat im Gegensatz zur Erzeugung auf einem Sensor-

knoten den Vorteil, dass nicht nur lokale Sensordaten genutzt werden können. Stattdessen ist es möglich, auch Daten anderer Sensoren und weitere Informationsquellen, wie z. B. Wetter- oder Stauinformationen zu verwenden. Auf der anderen Seite müssen ausreichend Daten vom Sensor vorliegen, um ein Modell mit hoher Güte generieren zu können.

Mit Bezug auf Abschnitt 4.3 lässt sich somit sagen, dass sowohl die Generierung auf dem Sensorknoten als auch im Backend ihre Vor- und Nachteile besitzen. Welches Vorgehen weniger Ressourcen verbraucht und Modelle mit höherer Güte erzeugt, ist individuell von verschiedenen Faktoren abhängig. Zu diesen Kriterien zählen u. a. die Vorhersagbarkeit der Zustände, die Häufigkeit von Zustandsänderungen und die nötigen Ressourcen zur Übertragung und Berechnung der Modelle. Es muss daher situativ untersucht werden, welches Verfahren besser geeignet ist.

6 Informationsspeicherung und -präsentation

> Getting information off the Internet is like taking a drink from a fire hydrant.
>
> — *Mitchell Kapor*

In den vorherigen Kapiteln wurde gezeigt, wie aus Sensorwerten Zustände ermittelt, für diese Zustände Wahrscheinlichkeitsmodelle generiert und wie die Zustände und Modelle in RDF modelliert werden können. Dieses Kapitel behandelt zwei noch offene Fragen:

1. Wie können die RDF-Daten effizient gespeichert und abgefragt werden?
2. Wie können die gespeicherten Daten für Anwendungsentwickler und Anwender zur Verfügung gestellt werden?

Zur Speicherung von semantischen Daten in Form von RDF-Tripeln werden Datenbanken genutzt, die speziell auf diesen Zweck zugeschnitten sind. Diese semantischen Datenbanken, auch Triplestores genannt, sind darauf optimiert, Tripel zu indizieren, zu speichern und über Abfragesprachen aufzufinden.

Es existiert eine Vielzahl von Triplestores, die in der Lage sind bis zu mehrere hundert Millionen Tripel zu verarbeiten [171]. Allerdings brauchen die Systeme mehrere Stunden, um Anfragen auf dieser Menge von Daten zu beantworten. Geht man von mehr als zehn Milliarden Sensoren im zukünftigen IoT aus, bei dem jeder Sensor mit dem Modell aus Kapitel 5 durch etwa 100

Tripel beschrieben wird, ergeben sich Billionen von Tripeln. Eine Suche, die mit geringer Antwortzeit Ergebnisse liefert und somit Nutzer zufrieden stellt, ist mit diesen Systemen nicht möglich.

Deshalb ist es notwendig Triplestores skalierbar zu gestalten, damit diese auch mit einer wachsenden Menge an Daten umgehen können. Ein Ansatz ist die Verteilung der Daten auf eine Vielzahl von Systemen und damit auch die Verarbeitung von (Teil-)Anfragen auf diesen Systemen.

In diesem Kapitel wird DecentSPARQL, ein Ansatz zum effizienten und verteilten Speichern von semantischen Daten auf Basis von strukturierten Peer-to-Peer-Netzwerken (P2P-Netzwerken) beschrieben. Die Idee hinter DecentSPARQL soll die Nachteile bestehender Systeme bewältigen.

In einem strukturierten P2P-Netz kann jeder Teilnehmer, auch Peer genannt, gleichzeitig Client und Server sein, d. h. dass jeder Teilnehmer sowohl Daten speichern als auch Anfragen beantworten kann. Die meisten strukturierten P2P-Netze bauen ihre Struktur durch Nutzung von Distributed Hash Tables (DHTs) auf. Dabei ist jeder Peer für ein Intervall aus dem Wertebereich einer konsistenten Hashfunktion zuständig. Für ein neues Datum, welches im Netz gespeichert werden soll, wird der Hashwert berechnet und zu dem Peer geroutet, welcher für diesen zuständig ist. Ebenso wird ein Datum aus einer Anfrage gehasht und an den entsprechenden Peer weitergeleitet, um zu prüfen, ob es dort gespeichert ist. Dabei ergeben sich zwei große Herausforderungen. Zum einen soll eine bestmögliche Gleichverteilung der Daten und Anfragen auf die Peers gewährleistet werden, damit keine sogenannten Hotspots entstehen, an denen sich Daten oder Anfragen häufen. Zum anderen wird eine effiziente Anfrageverarbeitung benötigt, da es bei komplexeren Anfragen vorkommen kann, dass Daten von verschiedenen Peers kombiniert werden müssen. Um die Menge an zu verschickenden Daten zu minimieren, sollte die Lösungsmenge so früh wie möglich, z. B. durch Ausfiltern, verkleinert werden. Ein Vorteil von P2P-Systemen ist die Möglichkeit des dynamischen Beitretens und Verlassens des Netzes durch Peers. So können jederzeit neue Systeme integriert werden, um auf wachsende Datenmengen zu reagieren. Sobald ein Peer dem Netz beitritt, werden ihm ein Hashintervall und die Daten des Intervalls zugeteilt. Mittels Replikation über mehrere Peers kann ein Netz zudem gegen plötzliches Verschwinden von Peers, z. B. durch Ausfall, abgesichert werden.

Die gespeicherten Daten können per SPARQL-Anfrage von einem Triplestore abgefragt werden (vgl. Abschnitt 2.3.2). Für Anwendungsentwickler, welche die Daten für eine Suche im IoT nutzen wollen, ist diese Abfragesprache keine komfortable Möglichkeit, da sie die Struktur der Daten kennen und ggf. SPARQL erlernen müssen. Somit ist es für Endnutzer erst recht nicht möglich, mittels Abfragesprache die Suche erfolgreich zu nutzen. Deswegen werden nach der Präsentation der Speicherschicht Schnittstellen für Entwickler und Nutzer vorgestellt. Diese verbergen die Details der Speicherung, so dass auch andere Verfahren zum Speichern nutzbar sind. Damit müssen Anwendungen, welche auf den Daten aufbauen, nicht geändert werden, wenn die Speicher-

Name	P2P-Architektur	Datenabfrage
RdfPeers [173, 174]	Multi-Attribute Addressable Network [175]	RDF Data Query Language (RDQL) [176]
RdfCube [177]	Content Addressable Network (CAN) [178]	Nicht näher spezifiziert
BabelPeers [179]	Nicht näher spezifiziert	Nicht näher spezifiziert
3rdf [180]	3nuts [181]	Untermenge von SPARQL
Atlas [182]	Bamboo [183]	Untermenge von SPARQL
GridVine [184]	P-Grid [185]	RDQL [176]
Page [186]	Bamboo [183]	Nicht näher spezifiziert
DSS [187]	Chord [188]	RDQL [176]
Unistore [189]	P-Grid [185]	Vertical Query Language [189], eine modifizierte Untermenge von SPARQL
Edutella [190]	HyperCuP superpeer topology [191]	RDF Query Exchange Language [190]

Tabelle 6.1.: Liste von P2P-Triplestores.

schicht ausgetauscht wird. Zusätzlich lassen sich zwischen Schnittstelle und Speicherschicht Verfahren zur Steigerung der Effizienz wie Sensor Ranking implementieren.

In Abschnitt 6.1 werden zuerst verwandte Arbeiten beschrieben. Anschließend wird in Abschnitt 6.2 die P2P-basierte Speicherschicht DecentSPARQL präsentiert. Abschnitt 6.3 behandelt abschließend die Schnittstellen zur Suche in den gespeicherten Daten. Für Entwickler wird ein Zugriff per REST vorgestellt, während für die Endnutzer eine GUI präsentiert wird. Das Kapitel schließt mit einer Nutzerstudie unter 20 Teilnehmern zur Gebrauchstauglichkeit der vorgestellten GUI. Die Inhalte zu DecentSPARQL wurden vorab in [28] veröffentlicht.

6.1. Verwandte Arbeiten

Im Bereich der Datenspeicherung existiert bereits eine Vielzahl von Triplestores, die auf unstrukturierten und strukturierten P2P-Netzen aufsetzt. Im Weiteren wird nicht auf unstrukturierte Netze eingegangen, da für die Suche nach Daten ohne Hilfsmechanismen wie Ontologien oder spezialisierte Peers das ganze Netz geflutet werden muss. Diese Vorgehensweise skaliert im IoT nicht. Eine Übersicht von Triplestores, die auf strukturierten P2P-Netzen basieren, ist in Tabelle 6.1 gegeben. Eine weiterführende Behandlung existierender Systeme ist in der Arbeit von Filali et al. [172] zu finden.

Tabelle 6.1 können zwei Nachteile existierender Implementierungen entnommen werden. Erstens basiert jedes System auf einer spezifischen P2P-Struktur und ist daher an Vor- und Nachteile dieser gebunden und zweitens verwenden die Systeme veraltete, nicht standardisierte oder nicht näher spezifizierte Abfragesprachen. Nur 3rdf und Atlas nutzen eine Untermenge des standardisierten

SPARQL. Zudem besitzt keines der vorgestellten Systeme eine Unterstützung für GeoSPARQL.

Die ersten sechs Systeme in Tabelle 6.1 nutzen die gleiche Datenverteilungsstrategie, bei der Hashwerte für das Subjekt *S*, das Prädikat *P* und das Objekt *O*, also *hash(S)*, *hash(P)* und *hash(O)* gebildet werden. An den für die Hashwerte zuständigen Peers wird jeweils das Tripel gespeichert. Die restlichen vier Systeme nutzen zusätzliche Informationen, wie Schemata, Ontologien oder ein viertes Element, um die Daten zu verteilen. Damit können diese Systeme Daten zwar gleichmäßiger verteilen und teilweise sinnvoll gruppieren, benötigen aber Vorwissen über die Daten. Die anderen Systeme nutzen ausschließlich Hashwerte der Daten zum Verteilen. Die in DHTs genutzten Hashfunktionen besitzen üblicherweise zwar die Eigenschaft gleichverteilte Hashwerte zu generieren, aber nur sofern die Eingabemenge ebenfalls gleichverteilt ist. Dies ist bei RDF-Daten oftmals nicht der Fall. Standardisierte Prädikate wie *rdf:type* und *rdf:subClassOf* zur Beschreibung von Eigenschaften und Relationen sind überproportional häufig vorhanden. Peers, die für die Hashwerte dieser Prädikate zuständig sind, werden zu sogenannten Hotspots bei der Speicherung und Beantwortung von Anfragen. Dies ist ein Nachteil bei der Verteilung in diesen Systemen.

In Abschnitt 1.2 wurden verschiedene Suchsysteme vorgestellt. Nur in den Arbeiten [5–7, 16, 17] werden die GUIs der Systeme gezeigt. Von diesen wiederum sind nur in [5, 16] Nutzerevaluation durchgeführt worden, bei der aber zum Teil nur die Funktionalität und nicht die Gebrauchstauglichkeit untersucht wurde (vgl. Abschnitt 1.2.1). In den in Abschnitt 3.1 vorgestellten Nutzerstudien wurde ebenfalls vorrangig die Funktionalität der Systeme und nicht die Gebrauchstauglichkeit der bereitgestellten GUIs untersucht.

6.2. DecentSPARQL

DecentSPARQL[1] ist ein Backend für die SPARQL-Engine LUPOSDATE[2] [192], ein java-basiertes, quelloffenes semantisches Datenbankmanagementsystem LUPOSDATE unterstützt die etablierten Semantic-Web-Standards SPARQL in Version 1.1, RDFS [193], OWL [194] und RIF BLD [195]. Es stellt bereits verschiedene zentralisierte Backends zur Speicherung von RDF-Daten zur Verfügung. So existieren u. a. ein In-Memory-Store, d. h. die Daten werden komplett im Arbeitsspeicher gehalten, und eine Implementierung, welche die Daten auf der lokalen Festplatte speichert.

DecentSPARQL setzt an dieser Schnittstelle an und ist ein Backend, das eine dezentrale Speicherung ermöglicht. Dazu verteilt DecentSPARQL die Daten in einem strukturierten P2P-Netz. Dies ist in Abbildung 6.1 dargestellt. Auf jedem Peer des Netzes läuft ein LUPOSDATE-Client. Über DecentSPARQL kommunizieren die Peers miteinander. Dabei erlaubt DecentSPARQL beliebige P2P-

[1] `https://github.com/itm/decentsparql/` [Zugriff am 21. März 2014]
[2] `https://github.com/luposdate/luposdate/` [Zugriff am 21. März 2014]

Netze zu verwenden. Dazu müssen lediglich die elementaren Funktionen für P2P-Systeme implementiert werden. Dies sind *put(key, value)*, *get(key)* und *sendMessage(key, message)* zum Einfügen und Abfragen von Daten sowie zum Verschicken von Nachrichten an andere Peers. Zurzeit existieren Implementierungen, um Chordless[3] (eine Implementierung von Chord [188]) und TOM P2P[4] als unterliegendes P2P-Netz zu nutzen.

Durch die Verwendung von etablierten P2P-Technologien und die dadurch einhergehende Dezentralisierung und Unabhängigkeit von einer zentralen Komponente, werden eine gute Skalierbarkeit und Robustheit ermöglicht.

Wie oben erwähnt, ist eine weitere Herausforderung neben der Skalierbarkeit die gleichmäßige Verteilung von Daten und Anfragen, um Hotspots zu vermeiden. Um dies zu ermöglichen, erlaubt DecentSPARQL die Implementierung beliebiger Datenverteilungsstrategien. Dazu müssen nur drei Methoden zum Hinzufügen, Entfernen und Abrufen von Tripeln implementiert werden. Die ersten beiden Methoden erhalten als Argument jeweils ein vollständiges Tripel aus Subjekt, Prädikat und Objekt. Die Methode zum Abrufen erhält ebenfalls Subjekt, Prädikat und Objekt, wobei eine oder mehrere Komponenten auch Variablen sein können. Sie liefert alle Tripel zurück, die dem Anfragemuster entsprechen, sofern die Verteilungsstrategie dieses Muster erlaubt. So stellt die Abfrage mit Variablen für Subjekt, Prädikat und Objekt einen Datenbank-Dump dar, da alle Tripel auf dieses Muster passen. Solch eine Anfrage sollte aus Effizienzgründen nicht implementiert und mit einer Fehlermeldung quittiert werden.

Die Datenverteilungsstrategien können auch dazu genutzt werden, Replikation und dadurch Robustheit gegenüber Ausfällen zu integrieren. Angenommen, es wird die am häufigsten genutzte Strategie, das Hashen der drei Elemente eines Tripels (*hash(S)*, *hash(P)* und *hash(O)*), verwendet. Außerdem nehme man an, dass die Hashfunktion auf das Intervall $[0, 2^x - 1]$ abbildet (üblicherweise mit $x = 128$ oder $x = 160$). Dann könnte man das Tripel zusätzlich auf Peers verteilen, die für *hash(S)*+*y*, *hash(P)*+*y* und *hash(O)*+*y* mit $y = \frac{2^x-1}{2}$ zuständig sind. Falls das Netz ausreichend Teilnehmer besitzt, liegen die beiden berechneten Punkte wahrscheinlich auf unterschiedlichen Peers. Würde der Peer für *hash(S)* ausfallen, gäbe es immer noch eine Kopie auf dem Peer, der für *hash(S)*+*y* zuständig ist. Es ist durch entsprechende Verteilung natürlich auch möglich mehr als ein Replikat anzulegen, um so mehrere Ausfälle kompensieren zu können.

Den Entwicklern bietet die Implementierung eigener Strategien einige Vorteile. Zum einen ist es möglich, neue Ansätze zu erproben und evaluieren. Zum anderen können bekannte Methoden, wie das Hashen und Verteilen nach Subjekt, Prädikat und Objekt, verwendet werden oder es lassen sich Methoden einsetzen, die spezifisch auf die zu erwartenden Daten zugeschnitten sind.

[3] `https://github.com/rwglab/p2p-triplestore/tree/master/chordless/` [Zugriff am 21. März 2014]

[4] `http://tomp2p.net/` [Zugriff am 21. März 2014]

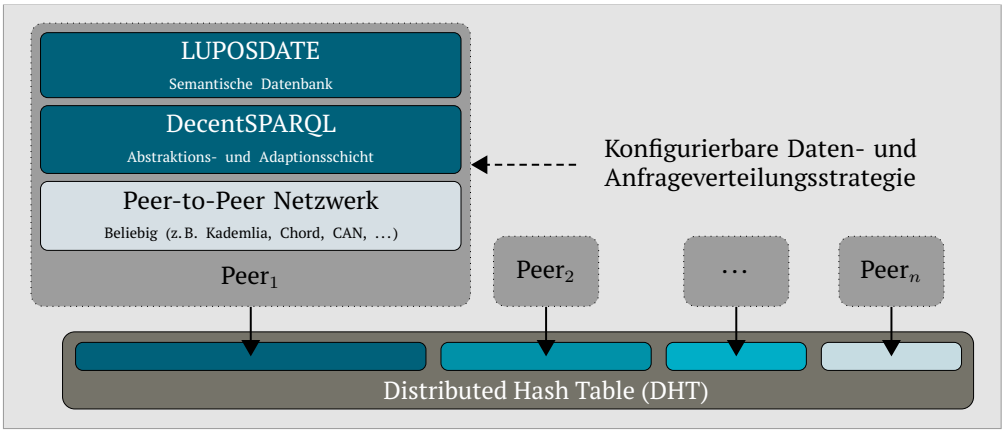


Abbildung 6.1.: DecentSPARQL-Architektur.

Bei komplexen Anfragen ist zu erwarten, dass Daten von vielen Peers abgerufen und aggregiert werden müssen. Die einfachste, aber ineffizienteste Methode ist, dass der Peer, der die Anfrage erhält, alle Daten von den für die jeweiligen Daten zuständigen Peers sammelt und anschließend die weiteren Operatoren des Abfragebaums wie Filter und Joins lokal ausführt. Neben der Implementierung eigener Datenverteilungsstrategien lassen sich auch eigene Anfrageverteilungsstrategien implementieren. Dies ist möglich, weil DecentSPARQL als Backend vollen Zugriff auf die Datenstrukturen von LUPOSDATE hat. LUPOSDATE generiert aus SPARQL-Anfragen einen Operatorgraph. Dieser spiegelt die Abfolge der Ausführung von Operatoren auf den Daten wider. Allerdings ist er optimiert auf eine Ausführung, bei der alle Daten lokal vorliegen. Bei einer verteilten Beantwortung kann es vorkommen, dass dieser nicht mehr optimal ist. Darum ist es möglich, den Operatorgraph umzusortieren und Teile des Baums an Peers zu verschicken. Die Peers beantworten diesen Teilbaum und schicken ihre Ergebnisse an den aufrufenden Peer zurück, der diese weiterverwendet. Dies kann auch rekursiv geschehen. Somit ist es z. B. möglich, FILTER-Anweisungen aus SPARQL-Anfragen möglichst weit in Richtung der Blätter des Operatorgraphen zu schieben, um dadurch die Menge an Tripeln möglichst früh reduziert wird. Durch eine effiziente Verteilung von Teilanfragen kann somit die Latenz beim Beantworten und die zu verschickende Datenmenge reduziert werden.

6.2.1. GeoSPARQL-Erweiterung für LUPOSDATE

GeoSPARQL [88, 89] besitzt neben eigenen Datentypen zur Definition von Geometrien auch Funktionen, die auf diesen Geometrien arbeiten, um Eigenschaften dieser oder Relationen zwischen diesen zu prüfen (vgl. Abschnitt 2.3.3). Da diese Datentypen und Funktionen nicht in den Standards von RDF und SPARQL definiert sind, sondern durch einen späteren Standard beschrieben werden, sind sie in den meisten Triplestores nicht implementiert und somit nicht nutzbar.

Die Modellierung aus Kapitel 5 nutzt GeoSPARQL-Datentypen, um die Position von Sensoren und die Regionen von PPMs zu beschreiben. Im IoT werden zudem viele Dienste Ortsinformationen der teilnehmenden Geräte und Nutzer benötigen. Um diese geografischen Daten speichern und abrufen zu können, muss DecentSPARQL den GeoSPARQL-Standard unterstützen. Deswegen wird das zugrundeliegende Datenbankmanagementsystem, LUPOSDATE, um die Speicherung, Serialisierung und Deserialisierung von GeoSPARQL-Datentypen erweitert. Außerdem werden die meisten der im Standard beschriebenen Funktionen implementiert. An DecentSPARQL selbst sind keine weiteren Änderungen notwendig, um GeoSPARQL verteilt im P2P-Netz nutzen zu können.

Ein weiterer Standard zur Beschreibung von Geometrien und Funktionen auf Geometrien ist SPARQL-ST [196]. Da dieser sehr viele Gemeinsamkeiten mit GeoSPARQL aufweist, wird er ebenfalls in LUPOSDATE integriert.

6.2.2. Evaluation

Der GeoSPARQL-Standard beschreibt insgesamt 30 Anforderungen, die in sechs Konformitätsklassen (siehe [88, Tabelle 1]) eingeteilt sind. Unterstützt ein System nur eine Teilmenge der Anforderungen, so ist es nicht voll kompatibel mit dem Standard, sondern nur für die entsprechenden Konformitätsklassen, die es umsetzt. Mit Hilfe von definierten Testfällen kann die Konformität getestet werden. Die GeoSPARQL-Erweiterung von LUPOSDATE unterstützt die Klassen *Core*, *Topology Vocabulary Extension*, *Geometry Topology Extension* und *RDFS Entailment Extension*. Die Klasse *Query Rewrite Extension* wird nicht unterstützt, während für die Klasse *Geometry Extension* 10 von 14 Anforderungen umgesetzt sind. Es fehlen dort die Anforderungen zur Serialisierung ins GML-Format.

In Abschnitt 1.3 wurde die Skalierbarkeit als eine Herausforderung für eine IoT-Suchmaschine motiviert. Dabei wurden vor allem eine gute horizontale Skalierbarkeit und die Berücksichtigung der vier Arten der Skalierbarkeit gefordert.

Eine horizontale Skalierbarkeit ist bei DHT-basierten P2P-Systemen, wie den in DecentSPARQL verwendeten, direkt gegeben. Das Gesamtsystem kann dynamisch um weitere Einzelsysteme (Peers) erweitert werden. P2P-Netze weisen weiterhin positive Eigenschaften für drei der vier Arten von Skalierbarkeit auf. Die strukturelle Skalierbarkeit wird durch die Grundidee des dynamischen Beitretens neuer Peers zum Netz erreicht. Der Prozess des Beitretens beeinflusst die zeitlich-räumliche Skalierbarkeit nur kurzfristig und geringfügig. Nach dem Beitritt muss der neue Peer die ihm zugeteilten Daten gesendet bekommen und die verschickenden Peers müssen diese Daten löschen. Außerdem müssen Anfragen nach Daten ggf. über mehr Hops als vorher geroutet werden. Da übliche Operationen, wie das Einfügen und Abfragen von Daten, $\mathcal{O}(\log n)$ Schritte bei n Peers benötigen, erhöht sich die Anzahl Hops bei Verdopplung der Peers nur um 1. Die räumliche Skalierbarkeit ist bei der Verwaltung von P2P-Netzen ebenfalls gut. Ein Peer muss bei n Peers eine Nachbarschaftsliste mit $\mathcal{O}(\log n)$

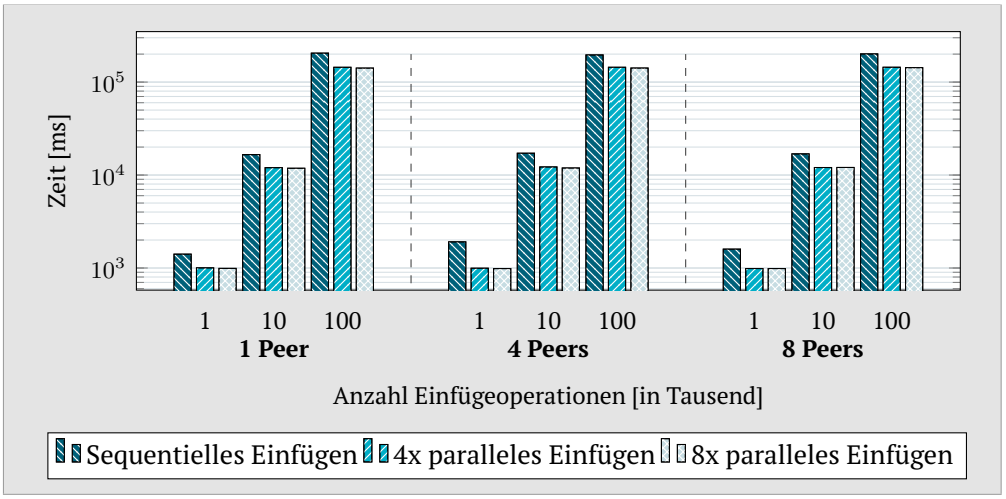


Abbildung 6.2.: Benötigte Zeit in Millisekunden für 1.000, 10.000 und 100.000 Einfügeoperationen mit Netzgrößen von einem, vier und acht Peers mit jeweils einem, vier oder acht genutzten Threads zum Einfügen.

Adressen verwalten. Somit erhöht sich auch hier die Liste nur um 1 bei der Verdopplung der Netzteilnehmer. Die Speicherlast im Bezug auf die Daten nimmt beim Hinzufügen von Peers sogar ab, da die Daten umverteilt werden.

Über die Lastskalierbarkeit kann ohne weitere Versuche keine Aussage getroffen werden, da diese sehr von der Implementierung abhängig ist. Deshalb werden die Einfüge- und Abfragezeiten für eine unterschiedliche Anzahl an Peers und Mengen an RDF-Daten untersucht.

Der erste Versuch untersucht die benötigte Zeit für das Einfügen von 1.000, 10.000 und 100.000 Elementen durch einen Peer. Es werden Versuchsreihen mit einer Netzgröße von einem, vier und acht Peers in Kombination mit einem, vier und acht Threads zum Hinzufügen auf dem zuständigen Peer durchgeführt. Eine Operation repräsentiert das Einfügen eines Datums und nicht das Einfügen eines Tripels. Wie viele Tripel mit 1.000 Operationen dem Netz hinzugefügt werden, hängt von der genutzten Verteilungsstrategie ab. Bei der häufig verwendeten Strategie des Hashens von Subjekt, Prädikat und Objekt, muss die Anzahl an Operationen somit durch 3 geteilt werden, um die Anzahl an Tripeln zu erhalten.

Abbildung 6.2 zeigt die Ergebnisse des Experiments. Die angegebenen Zeiten sind die Mittelwerte von jeweils zehn Durchgängen, wobei alle Peers lokal auf einem Rechner mit einem 4-Kern-Prozessor laufen. Für ein besseres Größenverhältnis ist die y-Achse logarithmisch skaliert.

Das Einfügen von mehr Elementen skaliert nicht linear, sondern etwa mit einem Faktor von 1,1 bis 1,2. Bei Versuchen, bei denen die Einfügeoperationen an verschiedenen Peers anfallen und nicht immer beim selben, skaliert es dagegen anfangs linear und erst später mit einem ähnlichen Faktor wie bei den gezeigten Ergebnissen. Dies spricht dafür, dass die genutzte Speicherschicht

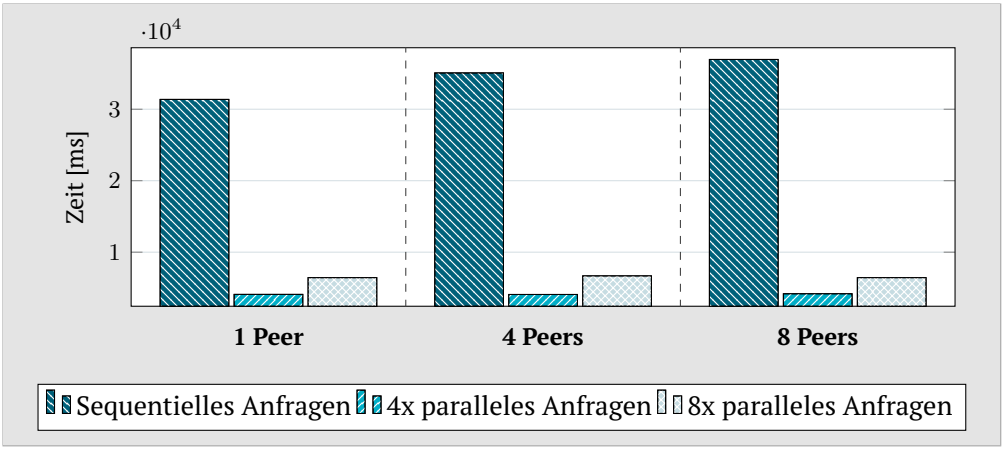


Abbildung 6.3.: Benötigte Zeit in Millisekunden für 1.000 SPARQL-Anfragen mit Netzgrößen von einem, vier und acht Peers mit jeweils einem, vier oder acht genutzten Threads für die Anfragen.

bei zu vielen Elementen die Verzögerung verursacht. Die Nutzung mehrerer Threads zum parallelen Einfügen skaliert in der Anzahl der zur Verfügung stehenden Prozessorkerne. Im konkreten Beispiel mit vier genutzten Kernen gibt es noch eine Verbesserung, während die Erhöhung auf acht Threads nur noch einen sehr geringen Gewinn liefert. Bei der Erhöhung der teilnehmenden Peers steigt die benötigte Zeit im Mittel für die verschiedenen Szenarien nicht. Vereinzelt liegt sie etwas höher bzw. niedriger als bei nur einem Peer. Dabei ist zu beachten, dass in der Betrachtung die Netzwerklatenzen fast komplett fehlen, da alle Peers auf einem Rechner laufen.

Der Test zur Abfragezeit basiert auf ähnlichen Bedingungen wie der vorherige. Es werden jeweils zehn Läufe mit einem, vier und acht Peers und einem, vier und acht Threads durchgeführt, um 1.000 SPARQL-Anfragen durch einen Peer beantworten zu lassen. Dies geschieht mit durch den Berlin SPARQL Benchmark[5] generierten Daten und Anfragen. Es wird nur eine Teilmenge der Anfragen genutzt, da aufgrund der verwendeten Strategie Anfragen mit Tripelmustern, die nur aus Variablen bestehen, nicht beantwortbar sind.

Abbildung 6.3 zeigt die Mittelwerte über die jeweils zehn durchgeführten Läufe der Versuchsreihe. Diesmal benötigt die y-Achse keine logarithmische Skalierung. Es ist zu erkennen, dass das parallele Anfragen wesentlich weniger Zeit als das sequentielle benötigt. Dies ist darauf zurückzuführen, dass Daten von verschiedenen Peers zur Beantwortung gesammelt werden müssen. Bei der sequentiellen Verantwortung muss immer auf alle Daten gewartet werden, bevor die nächste Anfrage gestartet werden kann, während bei der parallelen Verarbeitung schon die nächsten Anfragen verschickt werden können und parallel die Ergebnisse gesammelt werden. Auch in dieser Betrachtung fehlen wieder die Netzwerklatenzen. Die benötigte Zeit ist auch abhängig von der Anzahl

[5] `http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/` [Zugriff am 21. März 2014]

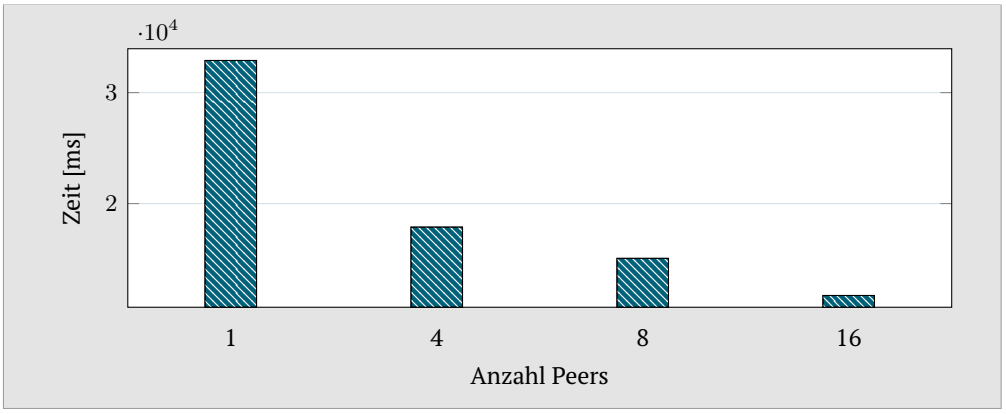


Abbildung 6.4.: Benötigte Zeit in Millisekunden für 1.000 SPARQL-Anfragen mit Netzgrößen von einem, vier, acht Peers und sechzehn Peers. Die Anfragen sind zufällig auf die Peers verteilt.

im Netz gespeicherter Tripel. Bei mehr Tripeln wird zum einen mehr Zeit für Joins und weitere lokale Operationen benötigt und zum anderen mehr Zeit und Bandbreite bei der Übertragung durch größere Datenmengen gebraucht. Dieser Ressourcenaufwand kann andererseits durch bessere Beantwortungsstrategien wie das Verteilen von Subanfragegraphen vermindert werden.

Abbildung 6.4 zeigt die Ergebnisse des letzten Versuchs. Statt alle Anfragen von einem Peer beantworten zu lassen, werden die Anfragen diesmal zufällig an die Peers verteilt. Dies ist ein realistischeres Szenario für das IoT, bei dem jeder Peer Anfragen beantworten wird, statt dass dies nur durch einen Peer geschieht. Die Zeit zur Beantwortung aller Anfragen fällt wie zu erwarten mit der Steigerung der Peers, da die Last besser aufgeteilt wird.

6.2.3. Fazit

Bei der durch das IoT zu erwartenden produzierten Datenmenge skaliert eine zentrale Serverstruktur nicht. P2P-Technologien zum verteilten Speichern und Beantworten von Anfragen können eine Möglichkeit sein, Skalierbarkeit zu gewährleisten. Mit DecentSPARQL, einer Abstraktionsschicht für den Triplestore LUPOSDATE, wurde ein System präsentiert, das semantische Daten in Form von RDF in einer P2P-Struktur speichert. Die Daten können mittels SPARQL abgefragt werden. Durch die Kombination von DecentSPARQL mit der Erweiterung von LUPOSDATE um den offenen Standard GeoSPARQL, der zum Speichern und Abfragen von geografischen Daten dient, entsteht der erste P2P-basierte Triplestore, der GeoSPARQL und SPARQL-ST unterstützt. Die Evaluation hat gezeigt, dass, obwohl die absolute Performance eher gering ist, das Konzept eines auf Grundlage von P2P verteilten Triplestores eine Skalierbarkeit sowohl beim Speichern als auch bei der Abfrage ermöglicht. Durch parallele Verarbeitung kann die benötigte Zeit für Operationen im Allgemeinen verringert

werden. Optimierungspotential entsteht außerdem durch die Nutzung besserer Verteilungs- und Abfragestrategien.

6.3. Suchschnittstelle

In Abschnitt 1.3 wurde als eine Herausforderung für die Nutzung einer Suche im IoT die Akzeptanz durch Entwickler und Nutzer genannt. Diese spiegelt sich für den Entwickler u. a. in offenen, flexiblen Schnittstellen einer Technologie wider. Für den Nutzer ist dagegen die Gebrauchstauglichkeit der GUI von großer Bedeutung. Er muss mit der Schnittstelle möglichst effizient, effektiv und für ihn zufriedenstellend arbeiten können. Das bedeutet, dass er in der Lage sein muss die Aufgabe, in diesem Fall eine Suche, zu bewältigen und dies mit möglichst geringem Zeitaufwand. Außerdem soll er sich dabei wohlfühlen. Nachfolgend werden eine REST-Schnittstelle für Entwickler und eine GUI für Endnutzer, die wiederum die REST-Schnittstelle verwendet, präsentiert.

6.3.1. REST-Schnittstelle

Die REST-Schnittstelle für Entwickler bietet eine Ressource, die es ermöglicht eine Suchanfrage an das Backend zu schicken. Dieses beantwortet die Anfrage und liefert ein Ergebnis zurück. Die Suchanfrage muss dabei mindestens aus

- einem Suchstring, der eine Entität und einen Zustand beinhaltet,

bestehen. Optional kann noch

- die Anzahl zu findender Ergebnisse,
- die Suchregion und
- die einzubeziehenden Rankingfaktoren

angegeben werden. Wird keine Ergebnisanzahl mitgeschickt, so wird der in der Umfrage in Kapitel 3 ermittelte Standardwert von zehn Ergebnissen verwendet. Die Region kann auf drei verschiedene Arten angegeben werden: als Kreis, der durch eine Mittelpunkt-GPS-Koordinate und einen Radius definiert wird, als Rechteck, der durch die rechte obere und linke untere Ecke als GPS-Koordinaten spezifiziert wird, oder in Form von GPS-Koordinaten, die ein Polygon beschreiben. Ohne Angabe einer Region wird weltweit nach passenden Entitäten gesucht. Für die Rankingfaktoren wird eine Liste der Namen der zu nutzenden Faktoren inklusive der Parameter übergeben. Dies sind die Gewichtung und ein Filterintervall. Alle Entitäten, deren Rankingwert für den Faktor außerhalb des Filterintervalls liegen, werden nicht mit in die Ergebnismenge aufgenommen. Damit der Entwickler weiß, welche Rankingfaktoren existieren, gibt es eine weitere Ressource, welche die zur Verfügung stehenden Faktoren inkl. Beschreibung und weiterer Parameter, wie Minimal- und Maximalwert zurückliefert.

Als Antwort auf eine Suchanfrage liefert der Server Titel, Beschreibung, Position, URI, den Zustand, die Zeit der letzten Zustandsmessung, d. h. die Aktualität und die Rankingwerte der gefunden Entitäten zurück.

6.3.2. Grafische Benutzungsschnittstelle

Die GUI ist in Abbildung 6.5 zu sehen. Sie besteht aus vier Bereichen, die zum Teil Unterbereiche enthalten. Diese sind in der Abbildung durch Zahlen und Buchstaben gekennzeichnet.

Der Bereich 1 beinhaltet das Suchfeld und den Button zum Abschicken einer Suchanfrage. Das Suchfeld bietet eine Autovervollständigung (siehe Abbildung 6.6a), um den Eingabeprozess für den Nutzer durch Vorschläge zu vereinfachen und zu beschleunigen.

Im Bereich 3 können die Sucheinstellungen verändert werden, sofern die Standardeinstellungen für die Suchanfrage nicht den Nutzerwünschen entsprechen. Dieser Bereich kann auf- und zugeklappt werden, so dass der Nutzer weniger scrollen muss, falls er keine Einstellungen ändern möchte. Der Zustand (auf-/zugeklappt) wird über den Browserneustart hinweg gespeichert, so dass der Nutzer die Aktion nicht jedes Mal wiederholen muss. Im Unterbereich 3a kann der Nutzer die Anzahl Ergebnisse, die er erhalten möchte, bestimmen. Im Unterbereich 3b sind die verschiedenen Rankingfaktoren aufgelistet. Für jeden Faktor kann eine Gewichtung (vgl. w_i in Gleichung 5.2 in Abschnitt 5.2.10) und ein Filterintervall angegeben werden. Entitäten, deren Rankingwert außerhalb des Intervalls liegen, werden nicht in die Ergebnismenge mit aufgenommen. Außerdem lässt sich jeder Rankingfaktor deaktivieren, damit er nicht in die Berechnung des Rankingwertes einbezogen wird. Zuletzt, kann im Unterbereich 3c die Suchregion angegeben werden. Die Karte stellt die Position des Nutzers mit einem Marker dar. Die Position wird über die W3C Geolocation API [197] ermittelt. Ein Button erlaubt jederzeit den Positionsmarker erneut zu zentrieren, sollte der Kartenausschnitt zuvor bewegt worden sein. Der Nutzer kann ein Rechteck, einen Kreis oder ein Polygon beliebiger Größe auf die Karte zeichnen, dieses nachträglich verändern oder löschen. Das Anlegen einer Suchregion und eines Rankings kann einige Zeit in Anspruch nehmen. Möchte ein Nutzer eine Suche regelmäßig ausführen, z. B. jeden Morgen einen freien Parkplatz suchen, so stört dieser Prozess die Effizienz. Deshalb können die Einstellungen für eine Suche benannt und gespeichert werden, damit sie jederzeit durch einen Mausklick nochmals ausgeführt werden kann. Diese gespeicherten Suchen sind auch nach einem Browserneustart verfügbar. Um nach Tätigung der Einstellungen direkt suchen zu können, gibt es einen zweiten Suchbutton am Ende des Sucheinstellungsdialoges. Somit muss der Nutzer nicht erst wieder nach oben scrollen.

Die gespeicherten Suchen werden im Bereich 2 aufgelistet. Ein Klick auf den Button 2a führt die Suche erneut aus und lädt die Einstellungen. Somit können diese auch nachträglich verändert und z. B. unter einem anderen Namen

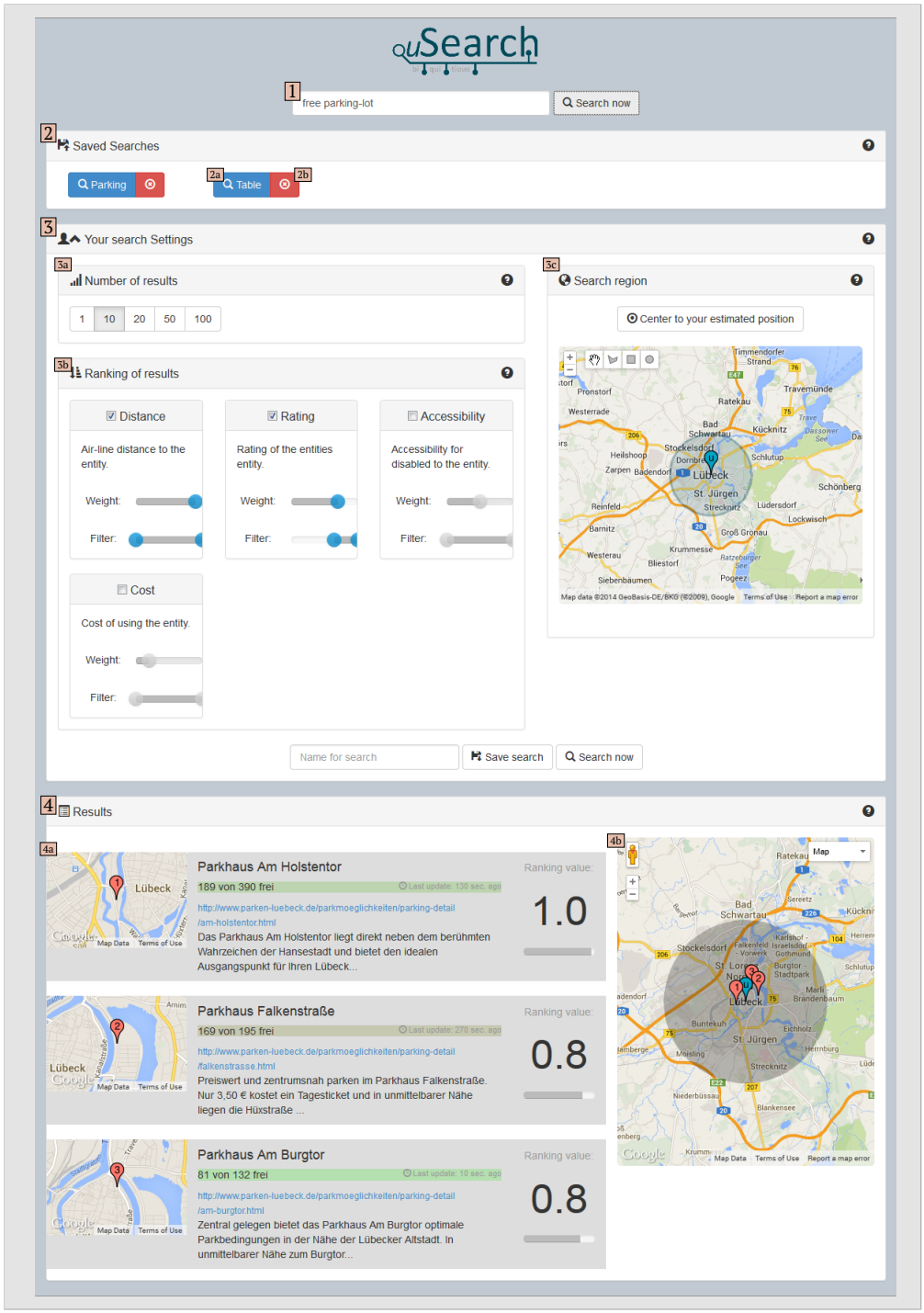
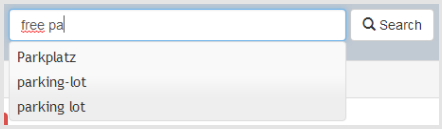
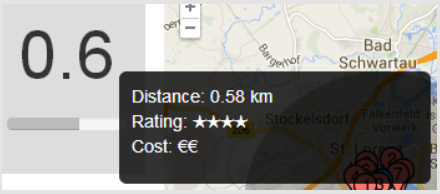


Abbildung 6.5.: Die GUI zur Suche im IoT. Die Zahlen und Buchstaben dienen zur Kennzeichnung von Bereichen zur Referenzierung aus dem Text.

abgespeichert werden. Wird eine Suche nicht mehr benötigt, so lässt sie sich über den roten Button 2b löschen.



(a) Autovervollständigung des Suchfeldes



(b) Tooltip mit Aufschlüsselung des Rankingwertes

Abbildung 6.6.: Ausschnitte der GUI.

Ist eine Suche erfolgreich, so werden die Ergebnisse in Bereich 4 präsentiert. In der Karte in 4b sind die Nutzerposition, die Suchregion und alle Ergebnisse mit jeweils einem Marker verortet. Der Unterbereich 4a beinhaltet eine Listenansicht der Ergebnisse. Diese stellt pro Ergebnis diejenigen Informationen, die von der REST-Schnittstelle zurückgeliefert werden, dar. Die Ergebnisse sind anhand ihres Rankingwertes sortiert, der mit Hilfe der in den Sucheinstellungen gemachten Angaben berechnet wird. Fährt der Nutzer über den Wert, erhält er ein Tooltip (siehe Abbildung 6.6b), das eine Aufschlüsselung der zugrundeliegenden Werte auflistet. Für jedes Ergebnis wird eine kleine statische Karte mit der Position angezeigt. Ein Klick auf diese Karte öffnet ein Overlay, in welchem die Route von der Nutzerposition zu der Entität angezeigt wird (siehe Abbildung 6.7).

Viele der Elemente besitzen als Hilfestellung ein Fragezeichensymbol, das jeweils ein Tooltip mit Erklärungen anzeigt, sobald mit der Maus darübergefahren wird.

Die GUI ist unter den Aspekten des responsiven Webdesigns entwickelt. Bei diesem gestalterischen Ansatz handelt es sich um eine Technik zur Erstellung von Webseiten, welche die Charakteristiken verschiedener Geräteklassen bei der Darstellung einbeziehen. Dadurch muss nicht für jede Auflösung eine eigene Version entwickelt werden, sondern die gleiche Webseite wird auf Geräten wie Tablets, Smartphones und Desktop-Computern korrekt angezeigt. Für die GUI ist dies mit Hilfe der Frameworks Bootstrap[6] und jQuery[7] umgesetzt.

Evaluation

Zum Test der Kriterien der Gebrauchstauglichkeit, d. h. der Effektivität, Effizienz und Zufriedenheit des Nutzers, gibt es eine Vielzahl von Methoden. Eine Übersicht findet sich in [93].

Wie schon in Kapitel 3 beschrieben, soll der Nutzer im Mittelpunkt stehen. Deshalb wird zur Evaluation der GUI eine Studie durchgeführt, bei der die Teilnehmer zunächst sechs Aufgaben lösen sollen, die typische Funktionen der

[6] `http://getbootstrap.com/` [Zugriff am 21. März 2014]
[7] `http://jquery.com/` [Zugriff am 21. März 2014]

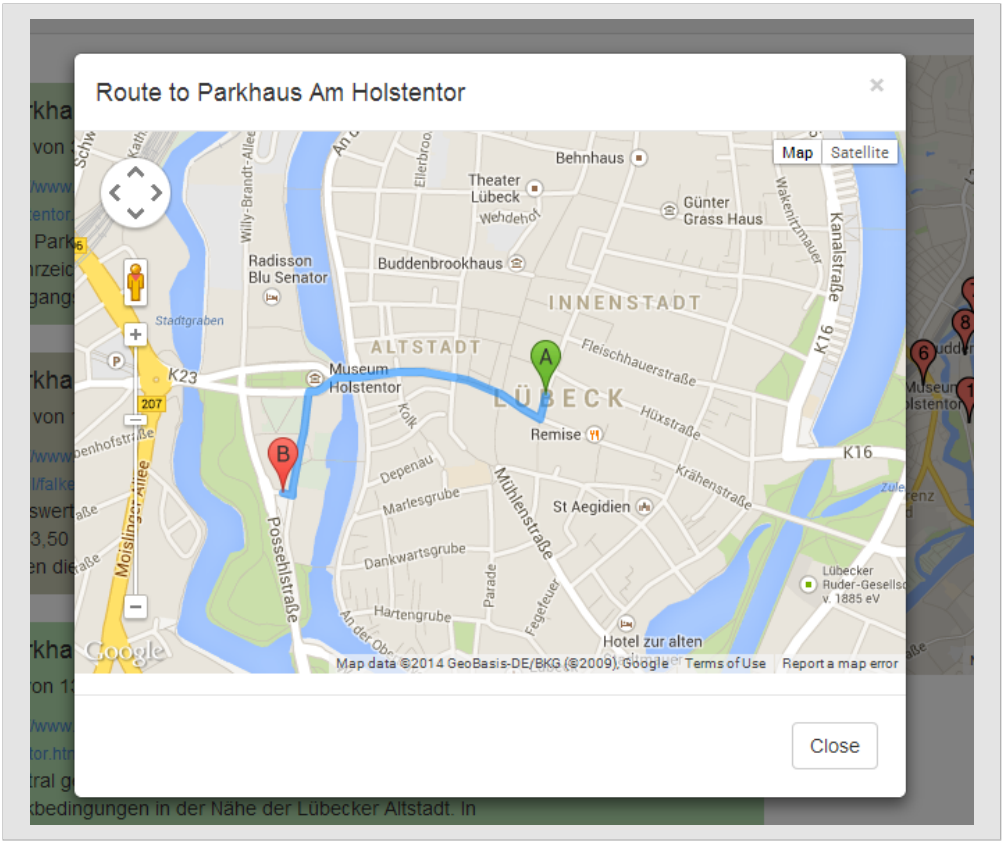


Abbildung 6.7.: Routingfunktion der GUI.

GUI abprüfen. Die Aufgaben sind in Anhang A.3 aufgeführt. Das erfolgreiche Absolvieren der Aufgaben testet die Effektivität der GUI. Die Dauer, die zur erfolgreichen Bewältigung jeder Aufgabe benötigt wird, lässt Rückschlüsse auf die Effizienz zu. Der letzte Punkt, die Zufriedenheit, wird durch einen Fragebogen bestimmt, den jeder Teilnehmer nach der Versuchsreihe ausfüllen muss.

Als Fragebogen wird die von Brooke [198] entwickelte System Usability Scale (SUS) verwendet. Dabei handelt es sich um zehn Aussagen, welche die Stimmung und Meinung des Teilnehmers zu der Nutzung eines Systems abfragen. Zur Beantwortung jeder Aussagen steht eine fünfstufige Likert-Skala zur Verfügung, auf der die Aussage von „starke Ablehnung“ mit dem Wert 1 bis „starke Zustimmung“ mit dem Wert 5 bewertet werden kann. Die Hälfte der Aussagen ist negativ formuliert, so dass bei dieser Hälfte eine niedrige und bei der anderen Hälfte eine hohe Punktzahl das bessere Ergebnis darstellt. Aus den Antworten einer Versuchsperson lässt sich ein Wert berechnen, der zwischen 0 und 100 liegt und die Zufriedenheit bei der Arbeit mit dem System widerspiegelt. Werte unter 50 stehen dabei für eine sehr schlechte Usability. Ab 70 gilt die Usability dagegen als gut. Eine Punktzahl von 100 bedeutet eine perfekte Usability. Bangor et al. [199] haben mit über 1.000 erweiterten SUS-Bögen untersucht, wie der SUS-Wert alternativ zum Punktewert als Note und mit umgangssprachlichen

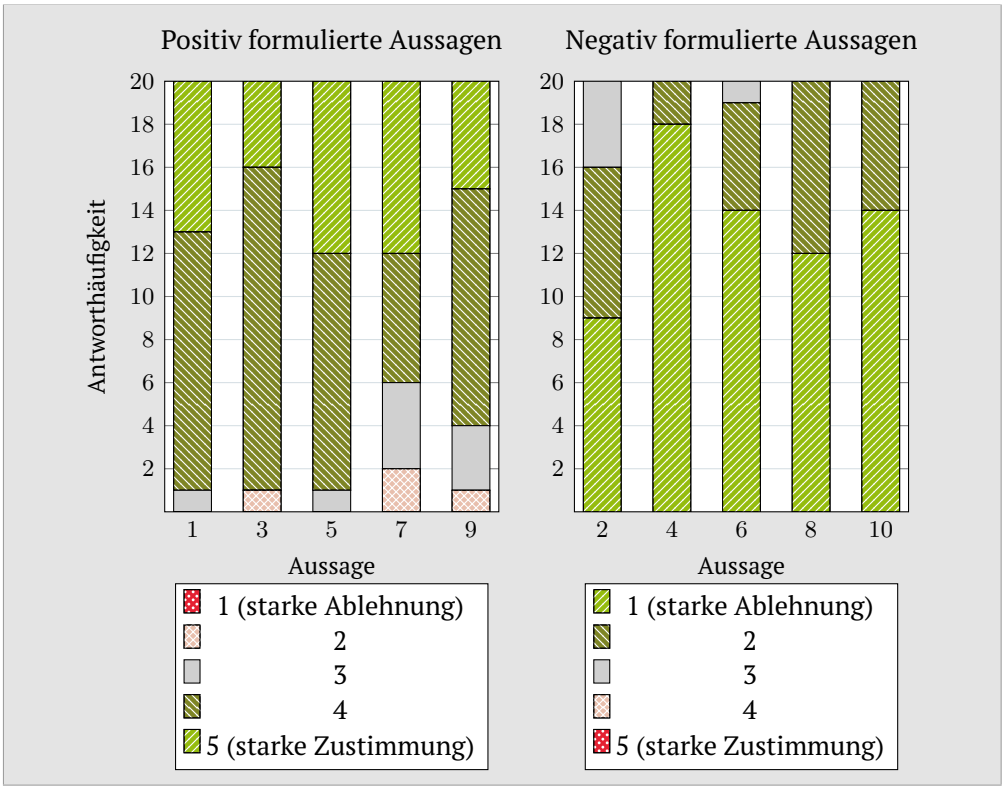


Abbildung 6.8.: Antworthäufigkeiten für die Antworten 1 (starke Ablehnung) bis 5 (starke Zustimmung) der zehn Fragen des SUS-Fragebogens. Für Fragen mit ungerader Fragenummer ist 5 die beste Antwort. Für Fragen mit gerader Fragenummer ist 1 die beste Antwort.

Begriffen, wie *OK* oder *Gut*, ausgedrückt werden kann. In dem Bogen, der in Anhang A.4 abgebildet ist, werden zusätzlich noch einige generelle Fragen zum Teilnehmer und allgemeine Anregungen zum System mit Freitextantworten gestellt.

SUS besitzt zwar nur zehn Aussagen und ist somit sehr einfach, kann aber dennoch gute Ergebnisse liefern. Untersuchungen haben gezeigt, dass die Tendenz, welche durch die Auswertung ermittelt wird, mit anderen Usability-Tests übereingestimmt. Dabei reichen schon ca. 12–14 Personen aus, um ein aussagekräftiges Ergebnis zu erhalten [200].

Die Versuchsreihe wurde mit insgesamt 20 Teilnehmern durchgeführt. Die häufigste Frage während des Versuches war, ob der Suchbegriff in Englisch oder Deutsch eingegeben werden müsse. Es waren beide Varianten möglich. Außerdem konnte beobachtet werden, dass anfangs die Suchregion in den Suchbegriff integriert wurde, anstatt die Funktion in den Sucheinstellungen zu nutzen.

Insgesamt lösten die Teilnehmer 77,5 % der Aufgaben ohne Probleme. Die restlichen wurden mit Einschränkungen bewerkstelligt, d. h. der Proband hat erst mit einer Hilfestellung durch den Versuchsleiter die Aufgabe bewältigt

bekommen. Für die Aufgaben 1, 3, und 6 musste nie eine Hilfestellung gegeben werden. Die häufigste Unterstützung bei den anderen beiden Aufgaben war der Hinweis, dass die Suchregion nicht über den Suchbegriff angegeben wird. Sobald der Nutzer eine Funktion entdeckt hatte, wurde diese schnell verstanden und anschließend korrekt genutzt. Die Dauer zur Bewältigung der Aufgaben nahm zwar mit der Komplexität zu, aber war dennoch sehr gering und betrug im Großteil weit unter einer Minute. Der Hauptteil der Aufgaben wurde innerhalb von Sekunden gelöst. Zudem war zu beobachten, dass die Nutzer mit zunehmender Nutzung gleicher Funktionen diese flüssiger bedienen konnten.

Abbildung 6.8 zeigt die Antworthäufigkeiten aller Teilnehmer auf der Likert-Skala für die zehn Aussagen des SUS-Bogens. Bei jeder ungerade nummerierten Aussage ist eine starke Zustimmung (grün) besser, während bei jeder gerade nummerierten eine starke Ablehnung (rot) besser ist. Bis auf bei Aussage 3, 7 und 9 wurden für alle Aussagen neutrale oder positive Antworten gegeben. Bei insgesamt 200 Bewertungen wurden nur viermal negative, also 2 %, und 14 mal, d. h. 7 %, neutrale Antworten angekreuzt.

Insgesamt erreicht die GUI auf der SUS-Punkteskala eine Durchschnittspunktzahl von 84,625. Dies entspricht nach den Untersuchungen von Bangor et al. [199] der Schulnote 2 bzw. dem Begriff *exzellent*. Der Median liegt bei 87,5, was derselben alternativen Bewertung gleichkommt. Die schlechteste und beste Bewertung durch einen Nutzer sind 67,5 und 95 Punkte, was den Schulnoten 4 und 1 entspricht.

Die Evaluation zeigt, dass die GUI eine gute Gebrauchstauglichkeit aufweist. Das Feedback der Teilnehmer ist durchweg positiv. Ein paar Verbesserungen sind aber dennoch sinnvoll. Das Angeben einer Suchregion wurde vornehmlich durch Eingabe ins Suchfeld vorgenommen. Im optimalen Fall sollten Ortsbegriffe erkannt und die Suchregion automatisch für den Suchort eingestellt werden. Dies kann sich allerdings als schwierig erweisen. Deswegen wäre zu mindestens ein auffälligerer Hinweis auf das Einstellen in den Sucheinstellungen nützlich als der derzeit durch die Hilfe-Tooltips gegebene. Bei den Ergebnissen sollte das Abrufen der Route besser gekennzeichnet werden, damit es einfacher gefunden wird. Der Rankingwert sollte nicht als Zahl zwischen 0 und 1, sondern als Prozentzahl zwischen 0 und 100 ausgegeben werden. Ein Prozentwert ist für Nutzer intuitiver verständlich. Zudem sollte die Gewichtung der einzelnen Rankingfaktoren prozentual geschehen und alle Gewichtungen stets 100 % ergeben, d. h. sobald eine Gewichtung erhöht bzw. verringert wird, sollten die anderen entsprechend erniedrigt bzw. angehoben werden.

6.3.3. Fazit

Die REST-Schnittstelle und die darauf aufbauende GUI bieten Entwicklern und Nutzern die Möglichkeit, die Suche für das IoT zu nutzen. Die REST-Schnittstelle enthält dabei die Grundfunktonalität zum Suchen, während die GUI eine mögliche Visualisierung der Sucheingabe, Sucheinstellungen und Ergebnisse

zeigt. Eine Evaluation der Gebrauchstauglichkeit der GUI unter Verwendung des SUS-Bogens mit 20 Teilnehmern hat mit einer Durchschnittspunktzahl von 84,625 von 100 möglichen Punkten eine gute Bewertung ergeben. Über die Freitextantworten wurden außerdem einige Verbesserungsvorschläge durch die Nutzer gegeben, die vermutlich zu einer noch besseren Punktzahl bei einem erneuten Test führen würden.

7 Zusammenfassung und Ausblick

> And one more thing.
>
> — *Steve Jobs*

In den letzten Jahrzehnten ist ein enormer Fortschritt in vielen Bereichen der Informationstechnologie zu beobachten. Die Integrationsdichte von integrierten Schaltkreisen verdoppelt sich ca. alle zwei Jahre und ermöglicht so immer schnellere Prozessoren bzw. die gleiche Leistung auf weniger Platz unterzubringen. Innovative Materialien, Sensoren, Funktechnologien und Architekturkonzepte erlauben neue Geräteklassen, die es ermöglichen die Umwelt detaillierter wahrzunehmen und mit ihr zu interagieren.

Das IoT als Erweiterung des Internets ist die Vision, dass unsere Alltagsgegenstände mit Sensorik, Aktuatorik und Informationstechnologie ausgerüstet und in das Internet integriert werden, um dort eine virtuelle Repräsentation bereitzustellen. Über diese Repräsentation kann der Zustand der realen Welt abgefragt werden und teilweise sogar mit der Entität interagiert werden. Dies ermöglicht viele neue Anwendungen, die den Menschen in seinem Alltag unterstützen können. Ein Basisdienst für das IoT ist die Suche nach Entitäten, die für die Erbringung eines Dienstes benötigt werden. Dabei ist es von Interesse, Entitäten nicht nur anhand von statischen Metadaten zu suchen, sondern auch anhand ihres aktuellen Zustandes. Beispielsweise ist ein Autofahrer nicht an allen Parkplätzen in seiner Umgebung interessiert, sondern nur an denen, die gerade frei sind.

Zwei große Herausforderungen bei der Umsetzung einer Suchmaschine für das IoT sind die Skalierbarkeit und Akzeptanz. Die durch die große Anzahl an Sensoren produzierte Datenmenge muss effizient modelliert, gespeichert und abrufbar sein, damit Betreiber und Nutzer bereit sind, die Technologien dauerhaft zu nutzen.

7.1. Beiträge der Arbeit

Diese Arbeit behandelt verschiedene Aspekte einer zustandsbasierten Suche für das IoT und stellt unterschiedliche Technologien zur Realisierung einer solchen Suche vor:

- Eine Nutzerbefragung zur Anforderungsanalyse,
- Mechanismen für verschiedene Geräteklassen zur Ermittlung von höherwertigen Zuständen aus Kamerabildern bzw. rohen Sensorwerten,
- verschiedene Verfahren zur Modellierung sowie Generierung von Wahrscheinlichkeitsmodellen für die Sensorzustände,
- eine semantische Modellierung von Sensoren, deren Zuständen, Wahrscheinlichkeitsmodellen und weiteren Informationen,
- DecentSPARQL, eine Lösung zum skalierbaren Speichern von semantischen Daten und
- eine REST-Schnittstelle für Entwickler sowie eine GUI für Endanwender zur Nutzung der Suche im IoT, inklusive einer Nutzerstudie mit 20 Teilnehmern, welche die Gebrauchstauglichkeit der GUI aufzeigt.

Die Befragung unter 184 Personen hat gezeigt, dass alle Bereiche einer zustandsbasierten Suche für das IoT flexibel gestaltet und erweiterbar sein müssen, um dem Nutzerverhalten und der Akzeptanz durch die Nutzer gerecht zu werden. Ein weiteres Resultat ist, dass im Allgemeinen nicht rohe Sensorwerte für die Nutzer von Interesse sind, sondern höherwertige Zustände. Die Ergebnisse der Umfrage wurden bei der Entwicklung der Technologien für eine zustandsbasierte Suche im IoT berücksichtigt. Es wurde insbesondere Wert auf eine hohe Flexibilität und eine gute Erweiterbarkeit, vor allem durch die Nutzung offener Standards, gelegt.

Um die von Nutzern gewünschten Zustände aus rohen Werten zu ermitteln, wurden LoCaF, die HLS-Option für CoAP, eine skriptbasierte Variante und eine Browser-Erweiterung für Chrome entwickelt. LoCaF ermöglicht, Bilder von einem kamerabestückten Sensorknoten mit Hilfe verschiedener Bildverarbeitungsmethoden zu verarbeiten, in den Bildern Objekte zu erkennen, diese zu zählen und daraus mit einer Regelsprache einen Zustand zu ermitteln. Dieser kann anschließend im Internet auf verschiedene Art veröffentlicht werden. Die HLS-Option erlaubt das Erzeugen von Zustandsressourcen auf CoAP-basierten Sensorknoten, die durch Abbildung aus Sensorwerten Zustände ermittelt. Über die Ressource sind die Zustände für Clients abrufbar. Für Sensorknoten, die kein

CoAP unterstützen, wurde eine Variante präsentiert, bei der durch ein Skript die Datenerfassung und -verarbeitung beschrieben wird. Zuletzt ist durch die vorgestellte Browser-Erweiterung eine Möglichkeit gegeben, die das Ermitteln von Zuständen für Informationen erlaubt, die auf Webseiten publiziert werden. Dies kann nützlich sein, falls Sensorknoten ihre Ausgaben nicht frei zugreifbar machen, sondern nur über Internetseiten.

Anschließend ist motiviert worden, wie Wahrscheinlichkeitsmodelle für eine effiziente Abfrage von Sensorzuständen nutzbar sind. Es wurden mehrere, neue oder auf den Vorarbeiten basierende Verfahren zur Generierung von Wahrscheinlichkeitsmodellen auf Sensorknoten bzw. im Backend präsentiert. Auf den Sensorknoten ist dies mit den Skripten möglich, die auch zur Zustandsermittlung genutzt werden. Für CoAP-basierte Knoten wurde eine weitere Option entwickelt, die das dynamische Erstellen von Wahrscheinlichkeitsmodellressourcen ermöglicht. Von diesen Ressourcen können daraufhin Modelle von Clients abgerufen werden. Im Backend lassen sich Modelle mit Hilfe von SPARQL-Anfragen über historische Daten erstellen.

Das Semantic Web ist die Idee, Wissen semantisch in IT-Systemen zu repräsentieren, so dass sowohl Menschen als auch Computer die Daten verstehen und interpretieren können. Die Integration verschiedener Datenquellen soll zur Erzeugung neuer Fakten beitragen. Um auch die Daten von Sensoren des IoT für das Semantic Web nutzbar zu machen, wurde eine Modellierung von Sensorzuständen, der Historie von Zuständen, von Wahrscheinlichkeitsmodellen und weiteren Konzepten erarbeitet. Dabei wurden die zuvor eingeführten Wahrscheinlichkeitsmodelltypen modelliert und erörtert, wie diese leicht zu kombinieren sind. Auch das Erstellen neuer Modelltypen ist mit wenigen Arbeitsschritten möglich.

Eine zentrale Speicherung der semantischen Daten von Millionen oder Milliarden von Sensoren eines zukünftigen IoT skaliert nicht. Deswegen bietet DecentSPARQL eine verteilte Lösung. Diese Abstraktionsschicht für die semantische Datenbank LUPOSDATE erlaubt das verteilte Speichern und Abfragen in P2P-Netzen. Dabei kann der Entwickler eine der existierenden Datenverteilungsstrategien nutzen oder eigene entwickeln und evaluieren. Ebenso kann die Abfrageverarbeitung durch Verteilung von Teilanfragen angepasst und somit optimiert werden. Durch die Erweiterung von LUPOSDATE um GeoSPARQL, ist DecentSPARQL der erste verteilte Triplestore, der diesen Standard unterstützt.

Abschließend wurden Suchschnittstellen für Entwickler und Nutzer vorgestellt. Eine Nutzerstudie mit 20 Personen hat die Gebrauchstauglichkeit der GUI für Endnutzer gezeigt. Dabei erreichte die GUI von 100 möglichen Punkten eine Durchschnittspunktzahl von 84,625, was der Schulnote 2 entspricht. Dies zeigt, dass die entwickelte GUI eine gute Gebrauchstauglichkeit aufweist und nur an wenigen Stellen optimiert werden muss.

Zusammenfassend zeigt Abbildung 7.1 zwei Netzdiagramme. Im ersten Diagramm ist in jeder Kategorie das Maximum der in Abschnitt 1.2.1 beschriebenen und in Abbildung 1.1 bewerteten existierenden Systeme aufgetragen. Das zweite

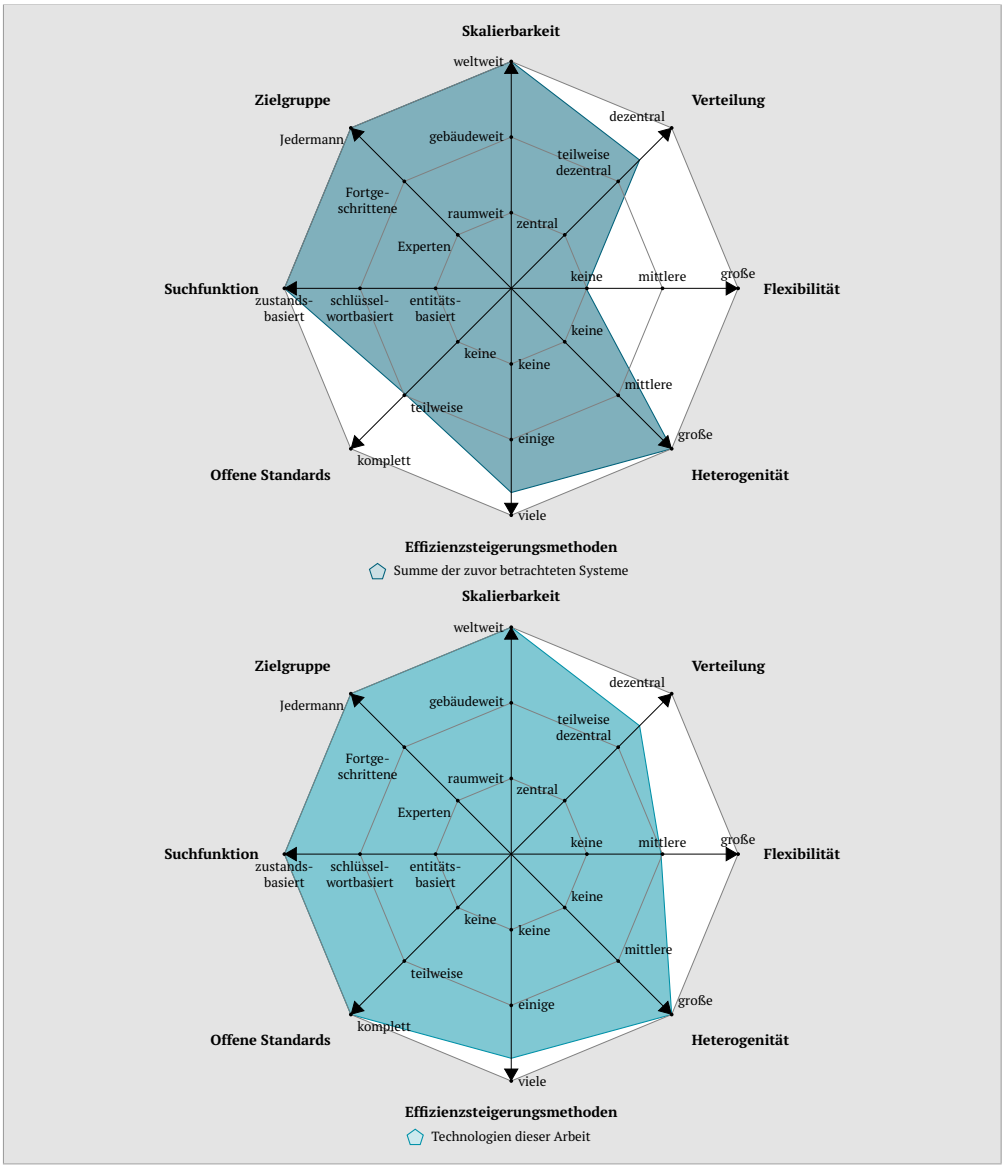


Abbildung 7.1.: Qualitativer Vergleich der Summe der in Kapitel 1 betrachteten Systeme mit den Technologien dieser Arbeit.

Diagramm zeigt die Bewertungen der Technologien dieser Arbeit. Der Vergleich zeigt, dass das zweite Diagramm pareto-optimal zum Ersten ist, d. h. in jeder Kategorie erreicht es einen mindestens genauso guten Wert wie der aus dem ersten Diagramm. Die Abbildung und die Nutzerstudie zur GUI stützen somit die These, dass *auf offenen Standards basierende Technologien entwickelt werden können, die eine zustandsbasierte Suche für das IoT realisierbar machen, die aus technischer Sicht skalierbar und ressourceneffizient arbeitet und zugleich eine hohe Benutzerakzeptanz durch eine gute Gebrauchstauglichkeit erzielt*.

7.2. Ausblick

Obgleich viele Nachteile existierender Systeme mit den in dieser Arbeit vorgestellten Technologien überwunden oder verbessert wurden, bleiben einige offene Fragen und mögliche Verbesserungen bestehen.

Auf der Ebene des Einsatzes von Sensoren zur Überwachung der Umwelt, in der wir leben, müssen zum einen die folgenden rechtlichen Fragen geklärt werden:

- Wer darf Sensoren einsetzen?
- Wo dürfen Sensoren eingesetzt werden?
- Wer oder was darf mit den Sensoren überwacht werden?
- Wie viele und was für Daten dürfen erhoben werden?
- Wie müssen die Daten gespeichert/weiterverarbeitet werden (z. B. Anonymisieren durch Unkenntlichmachung von Gesichtern)?
- Wer hat die Kontrolle über Sensoren und Daten?

Auf der anderen Seite müssen ebenso gesellschaftliche Fragen geklärt werden:

- Welche Daten sind Personen bereit von sich preiszugeben?
- Welchen Nutzen erwarten die Menschen bei Bereitstellung ihrer Daten?
- Wie wirkt sich eine dauerhafte, oft nicht sichtbare/wahrnehmbare, Überwachung durch Sensortechnologie gesundheitlich und gesellschaftlich auf die Menschen aus?

Die Behandlung oder gar Beantwortung dieser Fragen war zu keiner Zeit Fokus, da diese Arbeit die technologischen Aspekte behandelt. Diese Fragestellungen müssen von Personen der jeweiligen Fachgebiete wie Jura, Psychologie, Soziologie und Medizin erörtert werden.

Auch auf technologischer Ebene gibt es Verbesserungsmöglichkeiten und Richtungen, in die weitere Forschungen getätigt werden können.

Eine Portierung der in Kapitel 4 vorgestellten Möglichkeiten zur Zustandsermittlung auf andere Plattformen, wie z. B. Webcams, und die Integration in weitere Protokolle, würde zusätzliche Nutzungsmöglichkeiten eröffnen.

Im Kontext der Wahrscheinlichkeitsmodelle lässt sich untersuchen unter welchen Umständen der Einsatz der Modelle eine Ressourceneinsparung bringt, d. h. wie hoch die Güte der Modelle sein muss, damit sich der Einsatz lohnt. Dazu müssen Faktoren, wie u. a. die Häufigkeit und Muster der Zustandsänderungen, die Anfragehäufigkeit und -muster durch Nutzer und der Ressourcenaufwand zur Generierung der Modelle einbezogen werden. In diesem Zusammenhang ist auch eine isolierte Gegenüberstellung der Generierung von Modellen auf den Geräten und im Backend in Bezug auf den Ressourcenverbrauch sinnvoll. Aufbauend auf den Erkenntnissen könnte eine Komponente realisiert werden, die adaptiv entscheidet, wie die Daten von den Sensoren über-

mittelt werden, ob die Nutzung von Wahrscheinlichkeitsmodellen Ressourcen spart und wo diese Modelle generiert werden sollen.

Der in Kapitel 6 vorgestellte, verteilte Triplestore DecentSPARQL kann um neue Verteilungs- und Anfragestrategien erweitert werden, die eine effizientere Speicherung und Abfrage der RDF-Daten ermöglicht. Losgelöst von semantischen Technologien, aber im Kontext des verteilten Speicherns, stellt sich die Frage, welches die optimale Abfolge beim Filtern der Daten ist, um die Ergebnismenge für das Sensor Ranking zu erhalten. So lässt sich u. a. nach Position, Filterkriterien im Ranking, Zustandswahrscheinlichkeit oder Entität filtern.

Beispielhaft wird in Kapitel 6 eine GUI für Browser präsentiert. Diese ist zwar auch auf Smartphones nutzbar, dennoch könnte die Entwicklung einer eigenständigen Applikation das volle Potential des jeweiligen Smartphones ausnutzen. So könnten Sensorinformationen des Smartphones zur Bestimmung des Suchkontextes genutzt werden, um die Anfrage besser verstehen und zufriedenstellendere Ergebnisse liefern zu können. Eine Verbesserung bei der Eingabe der Suchanfrage wäre, wenn diese in natürlicher Sprache formuliert werden kann. Die Maschine müsste dann selbstständig, z. B. anhand des aktuellen Kontextes und des Gelernten aus früheren Suchanfragen, extrahieren, was der statische und der dynamische Anteil der Anfrage ist. Der nächste Schritt wäre die Nutzung von Sprach- und Gestenerkennung zur Benutzung der Suche. Dies würde die Verwendung in Fahrzeugen erleichtern, da der Fahrer nicht so stark abgelenkt wäre. Um schnell eine Suche erneut ausführen zu können, gibt es die Möglichkeit die Einstellungen einer einmal durchgeführten Suche zu speichern. Dies geschieht zurzeit lokal im Browser. Durch Synchronisierung mit einem Cloud-Dienst könnten diese Suchen einerseits über verschiedene Geräte eines Nutzers synchronisiert und andererseits mit anderen Nutzern geteilt werden.

Um den Kontext des Anwenders nutzen und bessere Ergebnisse auf Suchanfragen liefern zu können, müssten weitere Informationen, wie Wetter-, Stau- oder Kalenderdaten, in die Verarbeitung mit einbezogen werden. Durch Verwendung von Semantic-Web-Technologien zur Modellierung und -speicherung von Daten sind dafür bereits die Grundlagen gelegt, da viele weitere Datenquellen ebenfalls auf diese Techniken setzen und somit leicht integrierbar sind.

A
Anhang

A.1. Umfrage

Suche im Internet der Dinge

Willkommen bei der Umfrage mit dem Titel **"Suche im Internet der Dinge"**. Sollte Ihnen der Begriff Internet der Dinge nichts sagen, so können Sie dennoch an der Umfrage teilnehmen. Dies ist sogar explizit gewünscht, da wir auch Umfrageteilnehmer ohne Vorwissen benötigen. Bevor wir Fragen mit Bezug auf das Internet der Dinge stellen werden, werden wir die Idee dahinter kurz erläutern, damit Sie ein Verständnis dafür bekommen.

Die Umfrage umfasst maximal 44 Fragen (Davon 23 allgemeine Fragen und 21 spezifische zum Thema "Suche im Internet der Dinge) und dauert ca. 20-25 Minuten. Sie können die Beantwortung jederzeit unterbrechen und später fortsetzen. Sollten Sie eine Frage nicht verstehen oder beantworten möchten, so überspringen Sie diese einfach. Zu einigen Fragen gibt es auch Hilfestellungen, gekennzeichnet durch ein kleines Fragezeichen, am Ende der Frage (nach den Antworten).

Sie können jetzt mit der Umfrage beginnen oder weiterlesen, falls Sie noch etwas über den Umfrageleiter oder die Verwertung der Ergebnisse erfahren möchten.

Über den Umfrageleiter:

Mein Name ist Richard Mietz und ich arbeite als Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Kay Römer am Institut für Technische Informatik der Universität zu Lübeck. Im Rahmen der Projekte Real-World G-Lab und iBAST beschäftige ich mich u.a. mit dem Internet der Dinge und dort im Speziellen mit der Suche im Internet der Dinge.

Über die Verwertung der Umfrageergebnisse:

Sollten ausreichend Teilnehmer an der Umfrage teilnehmen, möchten wir die Umfrageergebnisse nutzen, um das Suchverhalten in einem zukünftigen Internet der Dinge zu untersuchen und zu modellieren. Wir möchten die Erkenntnisse in Form von Veröffentlichungen anderen interessierten Gruppen zugänglich machen. Weiterhin wollen wir die gewonnenen Informationen für weitere Forschung nutzen. Wir denken bei Erfolg der Umfrage außerdem über weitere Umfragen nach, um weitere Fragestellungen zu untersuchen.

Diese Umfrage enthält 44 Fragen.

Persönliche Informationen

Hier werden einige Fragen zu persönlichen Informationen gestellt, die bei der späteren Auswertung dazu dienen sollen zu analysieren wie repräsentativ die Umfrage war.

1 [alter]Bitte geben sie Ihr Alter in Jahren an:

Jede Antwort muss zwischen 0 und 115 sein

Bitte geben Sie Ihre Antwort hier ein:

[ ]

2 [geschlecht]Bitte geben Sie Ihr Geschlecht an:

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ weiblich
- ○ männlich

3 [bildung]Bitte geben Sie Ihre höchste abgeschlossene Bildung an:

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ kein Abschluss
- ○ noch Schüler
- ○ Volksschule / Hauptschule
- ○ Mittlere Reife / POS
- ○ Fachschulabschluss
- ○ Fachabitur / Fachhochschulreife
- ○ Allgemeine Hochschulreife / Abitur / EOS
- ○ Hochschulabschluss
- ○ Promotion
- ○ Sonstiges [ ]

4 [beruf]Bitte geben Sie Ihr aktuelles Berufsfeld an:

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Arbeitssuchend
- ☐ Schüler
- ☐ Student
- ☐ Wissenschaftlicher Mitarbeiter / Forscher / Doktorand
- ☐ Professor oder andere lehrende Tätigkeit
- ☐ Medizin (z.B. Arzt, Pflegepersonal)
- ☐ IT (z.B. Programmierer, Entwickler)
- ☐ Telekommunikation
- ☐ Medien (z.B. Journalist, Mediengestalter)
- ☐ Ingenieur
- ☐ Dienstleistungen (z.B. Friseur, Kellner)
- ☐ Finanzen (z.B. Bankkaufmann, Versicherungsmakler)
- ☐ Recht (z.B. Jurist)
- ☐ Vertrieb / Verkauf (z.B. Kassierer)
- ☐ Marketing (z.B. Werbekaufmann)
- ☐ Sonstiges: [ ]

Nutzung von Hardware

In diese Gruppe von Fragen möchte ich Informationen über das Nutzungsverhalten von bestimmter Hardware erhalten.

5 [pc_ex]

Benutzen Sie regelmäßig (mindestens 1x die Woche) einen oder mehrere stationäre PCs (Tower)?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Ja
- ◯ Nein

Als stationärer Computer ist ein PC gemeint, der nur an einem Ort genutzt wird. Notebooks sind explizit nicht gemeint, auch wenn sie nur an einem Ort genutzt werden.

6 [pc_os]

Welches Betriebssystem nutzen Sie hauptsächlich?

Beantworten Sie diese Frage nur, wenn folgende Bedingungen erfüllt sind:
* ((**pc_ex.NAOK** == "Y"))

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Windows
- ◯ Linux
- ◯ Mac
- ◯ Sonstiges [ ]

7 [notebook_ex]

Benutzen Sie regelmäßig (mindestens 1x die Woche) ein oder mehrere Notebook(s)/Netbook(s)?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Ja
- ◯ Nein

8 [notebook_os]

Welches Betriebssystem nutzen Sie hauptsächlich?

Beantworten Sie diese Frage nur, wenn folgende Bedingungen erfüllt sind:
* ((**notebook_ex.NAOK** == "Y"))

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Windows
- ◯ Linux
- ◯ Mac
- ◯ Sonstiges [ ]

9 [tablet_ex]

Benutzen Sie regelmäßig (mindestens 1x die Woche) ein oder mehrere Tablet-PC(s)?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Ja
- ◯ Nein

Ein Tablet-PC ist ein tragbarer Rechner, der anstatt mit Tastatur und Maus mit Hilfe des berührungsempfindlichen Bildschirms bedient wird (entweder per Stift oder per Finger).

10 [tablet_os]

Welches Betriebssystem nutzen Sie hauptsächlich?

Beantworten Sie diese Frage nur, wenn folgende Bedingungen erfüllt sind:
* ((**tablet_ex.NAOK** == "Y"))

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Android
- ◯ iOS (Apple)
- ◯ webOS
- ◯ Sonstiges [ ]

11 [handy_ex]

Benutzen Sie regelmäßig (mindestens 1x die Woche) ein oder mehrere Handy(s)?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Ja
- ◯ Nein

12 [handy_os]

Welches Betriebssystem nutzen Sie hauptsächlich?

Beantworten Sie diese Frage nur, wenn folgende Bedingungen erfüllt sind:
* ((**handy_ex.NAOK** == "Y"))

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Android
- ◯ iOS (Apple)
- ◯ Symbian
- ◯ Windows Phone / Windows Mobile
- ◯ BlackBerry OS (RIM)
- ◯ Bada (vor allem Samsung Wave Handys)
- ◯ Mobilinux
- ◯ Sonstiges [ ]

13 [navi]

Benutzen Sie regelmäßig (mindestens 1x im Monat) ein Navigationsgerät im Auto?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ◯ Ja
- ◯ Nein

Gemeint sind explizit Navigationsgeräte, die nur diesem Zweck dienen und nicht Navigationsanwendungen auf Handys o.ä.

Nutzung von Webdiensten

In dieser Kategorie soll das Nutzungsverhalten im Internet untersucht werden. Dabei beziehen sich die Fragen auf die meistgenutzte Suchmaschine Google und weitere Dienste dieses Unternehmens. Sollten Sie eine andere Suchmaschine bevorzugen, so beantworten Sie die Frage entsprechend für diese Suchmaschine und entsprechende Dienste, falls diese von dieser angeboten werden.

14 [google_service_know]Welchen dieser Google-Dienste kennen Sie?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Google Websuche
- ☐ Google Earth
- ☐ Google Maps
- ☐ Routing-Funktion in Google Maps
- ☐ Google Places
- ☐ Google Latitude
- ☐ Google News
- ☐ Google Kalender
- ☐ Google+

Eine Beschreibung der Dienste, falls Sie deren Namen nicht kennen sollten:

- **Google Websuche**: Der Dienst, der es erlaubt nach Webseiten zu suchen
- **Google Earth**: Darstellung eines virtuellen Globus, auf dem auch weitere Informationen wie Grenzen, Wikipedia-Artikel etc. eingeblendet werden könne
- **Google Maps**: Digitale Landkarte, die Straßen, Flüsse, Berge, aber auch Geschäfte und Hotels eingezeichnet hat.
- **Routing-Funktion in Google Maps**: Die Möglichkeit die kürzeste/schnellste Route von Ort A zu Ort B berechnen zu lassen
- **Google Places**: Erlaubt das Bewerten bzw. das Lesen von Bewertungen von Dingen wie Restaurants, Kneipen, Kinos und Hotels und stellt diese Bewertungen auch innerhalt von Google Maps dar
- **Google Latitude**: Erlaubt die Position von freigeschalteten Freunden auf Google Maps zu sehen. Die Position wird mittels Mobiltelefon o.ä. ermittelt
- **Google News**: Eine Nachrichtenseite von Google, die ihre Nachrichten aus den Nachrichten anderer Quellen zusammenstellt
- **Google Kalender**: Eine Kalender-Anwendung, die online genutzt werden kann
- **Google+**: Ein soziales Netzwerk von Google, ähnlich wie Facebook

15 [google_service_use]Welchen dieser Google-Dienste nutzen Sie?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Google Websuche
- ☐ Google Earth
- ☐ Google Maps
- ☐ Routing-Funktion in Google Maps
- ☐ Google Places
- ☐ Google Latitude
- ☐ Google News
- ☐ Google Kalender
- ☐ Google+

16 [google_search_know]Welchen dieser Google-Suche-Funktionen kennen Sie?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Mehrere Begriffe auf einmal suchen (sog. UND-Verknüpfung)
- ☐ Alternative Begriffe suchen (sog. ODER-Verknüpfung mit dem Schlüsselwortes OR)
- ☐ Phrasen/Textblöcke suchen (mit " ")
- ☐ Nahe beieinander liegende Begriffe suchen (mit ADJ / NEAR)
- ☐ Beliebigen Begriff suchen (mit *)
- ☐ Ausschließen von Begriffen (mit -)
- ☐ Auf bestimmten Seiten suchen (mit site:)
- ☐ Nach bestimmten Dateitypen suchen (mit filetype:)
- ☐ Zeitliche Einschränkung der Ergebnisse (mit daterange:)

17 [google_search_use]Welche dieser Google-Suche-Funktionen nutzen Sie regelmäßig?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Mehrere Begriffe auf einmal suchen (sog. UND-Verknüpfung)
- ☐ Alternative Begriffe suchen (sog. ODER-Verknüpfung mit dem Schlüsselwortes OR)
- ☐ Phrasen/Textblöcke suchen (mit " ")
- ☐ Nahe beieinander liegende Begriffe suchen (mit ADJ / NEAR)
- ☐ Beliebigen Begriff suchen (mit *)
- ☐ Ausschließen von Begriffen (mit -)
- ☐ Auf bestimmten Seiten suchen (mit site:)
- ☐ Nach bestimmten Dateitypen suchen (mit filetype:)
- ☐ Zeitliche Einschränkung der Ergebnisse (mit daterange:)

18 [google_search_use_1]Welche dieser Google-Suche-Funktionen haben Sie überhaupt schon einmal genutzt?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Mehrere Begriffe auf einmal suchen (sog. UND-Verknüpfung)
- ☐ Alternative Begriffe suchen (sog. ODER-Verknüpfung mit dem Schlüsselwortes OR)
- ☐ Phrasen/Textblöcke suchen (mit " ")
- ☐ Nahe beieinander liegende Begriffe suchen (mit ADJ / NEAR)
- ☐ Beliebigen Begriff suchen (mit *)
- ☐ Ausschließen von Begriffen (mit -)
- ☐ Auf bestimmten Seiten suchen (mit site:)
- ☐ Nach bestimmten Dateitypen suchen (mit filetype:)
- ☐ Zeitliche Einschränkung der Ergebnisse (mit daterange:)

IT-Kenntnisse

Mit diesen Fragen soll Ihr Wissen im Bereich IT getestet werden.

19 [iot_know]Haben Sie den Begriff "Internet der Dinge"/"Internet of Things" schon einmal gehört?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ Ja
- ○ Nein

20 [iot_explain]Beschreiben Sie kurz Ihr Verständnis vom Internet der Dinge.

Beantworten Sie diese Frage nur, wenn folgende Bedingungen erfüllt sind:
" ((**iot_know.NAOK** == "\"))

Bitte geben Sie Ihre Antwort hier ein:

21 [progs]Wie stimmen Sie folgenden Aussagen zu?

Bitte wählen Sie die zutreffende Antwort für jeden Punkt aus:

	Stimme ich zu	Stimme ich eher zu	Stimme ich eher nicht zu	Stimme ich nicht zu
Ich komme schnell mit neuen mir unbekannten Programmen zurecht	○	○	○	○
Ich nutze auch komplexe Funktionen in Programmen	○	○	○	○
Ich verstehe/erlene komplexe Funktionen in Programmen selbstständig	○	○	○	○

22 [it_news]Aus welchen Quellen beziehen Sie hauptsächlich Neuigkeiten in Bezug auf IT?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Fach-IT-News Seiten (z.B. Heise, Golem)
- ☐ Andere IT-News Seiten (z.B. Computerbild.de)
- ☐ Andere News Seiten (z.B. spiegel.de, zeit.de)
- ☐ Fach-IT-Zeitschriften (z.B. iX, C't)
- ☐ Andere IT-Zeitschriften (z.B. ComputerBild, PC Welt)
- ☐ Andere-Zeitschriften/Zeitungen (z.B. Spiegel, Welt)
- ☐ Ich interessiere mich nicht explizit für Neuigkeiten aus dem Bereich IT
- ☐ Sonstiges:

Erklärung des Internet der Dinge

Für die nachfolgenden Fragen benötigen Sie ein gewisses Verständnis über das Internet der Dinge, welches unten kurz erläutert wird.

23 [what_iot]Das Internet der Dinge

Das Internet der Dinge (engl.: Internet of Things) ist die Idee das Alltagsgegenstände in das Internet integriert werden und dort somit eine virtuelle Repräsentation erhalten.

Ein Beispiel ist das der Zustand von Dingen im Internet abrufbar ist. Stellen Sie sich vor Ihr Lieblingsrestaurant hat nicht nur eine normale Homepage mit der Speisekarte und den Öffnungszeiten sondern Sie können auch live sehen, ob Tische frei sind. Sie müssten nicht reservieren und könnten sich spontan für einen Besuch entscheiden. Würden nun alle Restaurants solch einen Service anbieten, könnten Sie nach Restaurants mit freien Tischen in Ihrer Umgebung suchen (falls z.B. Ihr Lieblingsrestaurant gerade voll ist).

Um so etwas zu realisieren würden Sensoren im Restaurant/an den Tischen angebracht werden, die die Tischbelegung erfassen können und dann im Internet für jeden zugänglich veröffentlichen.

Diese Idee wird heute schon teilweise umgesetzt, insbesondere in der Logistik. Die nachfolgenden Fragen, sollen das Suchverhalten untersuchen, falls solche Szenarien auch im privaten Bereich umgesetzt werden würden.

Bei der Suche im Internet der Dinge geht es also grundsätzlich darum **Dinge** anhand ihres **momentanen Zustandes** zu suchen und zu finden.

Suchverhalten

In dieser Sektion geht es um das allgemeine Suchverhalten im Internet der Dinge, sofern eine solche Suche existieren würde. Nehmen sie dies einfach hypothetisch an und überlegen Sie, wie sie eine solche Suche nutzen würden.

24 [which_searches]Welche Suchen würden Sie ausführen wollen, also nach welchen Dingen mit welchen Zuständen würden Sie suchen wollen (z.B. freier Tisch in einem Restaurant)?

Bitte geben Sie Ihre Antwort hier ein:

25 [which_device]Mit welchen Geräten würden Sie eine solche Suche durchführen?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Stationärer PC
- ☐ Notebook/Netbook
- ☐ Tablet-PC
- ☐ Handy
- ☐ Navigationsgerät
- ☐ Sonstiges:

26 [where]Wo würden Sie solch eine Suche nutzen?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Zu Hause
- ☐ Bei täglichen Besorgungen (z.B. Einkaufen, Amtsbesuchen)
- ☐ Unterwegs als Fußgänger
- ☐ Unterwegs mit dem Fahrrad
- ☐ Unterwegs im öffentlichen Nahverkehr
- ☐ Unterwegs im öffentlichen Fernverkehr
- ☐ Unterwegs mit dem Auto
- ☐ Im Urlaub / auf Reisen
- ☐ Bei der Arbeit
- ☐ Sonstiges:

27 [which_info]Welche Informationen würden Sie für solch eine Suche preisgeben, um z.B. Ihr Suchergebnis zu verbessern?

Bitte wählen Sie alle zutreffenden Antworten aus:

- ☐ Ihre Position
- ☐ Alter
- ☐ Geschlecht
- ☐ Frühere Suchanfragen
- ☐ Vorlieben (z.B. Vorliebe für Pizza zum Bevorzugen italienischer Restaurant bei einer Restaurantsuche)
- ☐ Mitgliedschaften in Gruppierungen (z.B. Bevorzugen von Restaurants, die Rabattaktionen für ADAC-Mitglieder bieten)
- ☐ Bewertungen, die Sie zu Dingen gemacht haben
- ☐ Freundesliste aus sozialen Netzwerken
- ☐ Religion (z.B. Bevorzugung koscherer Restaurant bei jüdischem Glauben)
- ☐ Sonstiges: _____

Eigene Suchanfragen

Bei den folgenden Fragen sollen Sie ihre Kreativität einsetzen und Suchanfragen formulieren.

28 [how_free_table]Wenn Sie bei Google nach einem Restaurant suchen, lautet ihre Suchanfrage "Restaurant". Wie würde Ihre Suchanfrage aussehen, wenn Sie nach einem freien Tisch in einem Restaurant suchen würden?

Bitte geben Sie Ihre Antwort hier ein:

_____

Sie dürfen jede Syntax nutzen, die Sie möchten, auch eine selbst ausgedachte.

29 [how_free_table_temp]Wie würde Ihre Suchanfrage aussehen, wenn Sie nach einem freien Tisch suchen würden, bei der das Restaurant zusätzlich mindestens 15 Grad warm ist?

Bitte geben Sie Ihre Antwort hier ein:

_____

Sie dürfen jede Syntax nutzen, die Sie möchten, auch eine selbst ausgedachte.

30 [how_free_tabl_parkin]Wie würde Ihre Suchanfrage aussehen, wenn Sie nach einem freien Tisch in einem Restaurant suchen würden und zusätzlich nach einen freien Parkplatz in der Nähe?

Bitte geben Sie Ihre Antwort hier ein:

_____

Sie dürfen jede Syntax nutzen, die Sie möchten, auch eine selbst ausgedachte.

Konkrete Suchen nach Dingen

Im vorletzten Frageblock möchte ich einige Fragen zu konkreten "Suchszenarien" stellen, um zu untersuchen ob und wie solche Suchanfragen genutzt werden würden.

31 [search_often]Wie oft würden Sie durchschnittlich aufgeführte Suchen ausführen? Suche nach ...

Bitte wählen Sie die zutreffende Antwort für jeden Punkt aus:

	Mehrmals täglich	Täglich	Mehrmals Wöchentlich	Wöchentlich	Mehrmals Monatlich	Monatlich	Zumindestens ein Mal im Jahr	Nie
freien Taxen	○	○	○	○	○	○	○	○
freien Autos beim Car-Sharing	○	○	○	○	○	○	○	○
freien Parkplätzen	○	○	○	○	○	○	○	○
staufreien Autostrecken	○	○	○	○	○	○	○	○
freien Fahrrädern bei Fahrradverleihern	○	○	○	○	○	○	○	○
Position eines Zuges/Busses auf den Sie warten	○	○	○	○	○	○	○	○
freien Tische in Restaurants	○	○	○	○	○	○	○	○
kurzen Warteschlangen vor Imbissen	○	○	○	○	○	○	○	○
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	○	○	○	○	○	○	○	○
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	○	○	○	○	○	○	○	○
sonnigen, ruhigen Orten	○	○	○	○	○	○	○	○

32 [search_when]Zu welcher Tageszeit würden Sie folgende Suchanfragen hauptsächlich nutzen? Suche nach ...

	Morgens (6-10 Uhr)	Vormittags (10-12 Uhr)	Mittags (12-14 Uhr)	Nachmittags (14-18 Uhr)	Abends (18-22 Uhr)	Nachts (22- 6 Uhr)
freien Taxen	☐	☐	☐	☐	☐	☐
freien Autos beim Car-Sharing	☐	☐	☐	☐	☐	☐
freien Parkplätzen	☐	☐	☐	☐	☐	☐
staufreien Autostrecken	☐	☐	☐	☐	☐	☐
freien Fahrrädern bei Fahrradverleihern	☐	☐	☐	☐	☐	☐
Position eines Zuges/Busses auf den Sie warten	☐	☐	☐	☐	☐	☐
freien Tische in Restaurants	☐	☐	☐	☐	☐	☐
kurzen Warteschlangen vor Imbissen	☐	☐	☐	☐	☐	☐
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	☐	☐	☐	☐	☐	☐
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	☐	☐	☐	☐	☐	☐
sonnigen, ruhigen Orten	☐	☐	☐	☐	☐	☐

Mehrfachnennung möglich.

33 [search_device]Mit welchen Geräten würden Sie die folgenden Suchanfragen hauptsächlich durchführen? Suche nach ...

	Stationärer PC	Notebook/Netbook	Tablet-PC	Handy	Navigationsgerät
freien Taxen	☐	☐	☐	☐	☐
freien Autos beim Car-Sharing	☐	☐	☐	☐	☐
freien Parkplätzen	☐	☐	☐	☐	☐
staufreien Autostrecken	☐	☐	☐	☐	☐
freien Fahrrädern bei Fahrradverleihern	☐	☐	☐	☐	☐
Position eines Zuges/Busses auf den Sie warten	☐	☐	☐	☐	☐
freien Tische in Restaurants	☐	☐	☐	☐	☐
kurzen Warteschlangen vor Imbissen	☐	☐	☐	☐	☐
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	☐	☐	☐	☐	☐
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	☐	☐	☐	☐	☐
sonnigen, ruhigen Orten	☐	☐	☐	☐	☐

Mehrfachnennung möglich.

34 [search_where]Wo würden Sie folgende Suchanfragen hauptsächlich durchführen? Suche nach ...

	Zu Hause	Unterwegs	Bei der Arbeit
freien Taxen	☐	☐	☐
freien Autos beim Car-Sharing	☐	☐	☐
freien Parkplätzen	☐	☐	☐
staufreien Autostrecken	☐	☐	☐
freien Fahrrädern bei Fahrradverleihern	☐	☐	☐
Position eines Zuges/Busses auf den Sie warten	☐	☐	☐
freien Tische in Restaurants	☐	☐	☐
kurzen Warteschlangen vor Imbissen	☐	☐	☐
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	☐	☐	☐
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	☐	☐	☐
sonnigen, ruhigen Orten	☐	☐	☐

Mehrfachnennung möglich.

35 [search_context]In welchem Kontext würden Sie die folgenden Suchen hauptsächlich durchführen? Suche nach ...

	Freizeit	Sport	Tägliche Besorgungen	Reise / Ausflug	Arbeit
freien Taxen	☐	☐	☐	☐	☐
freien Autos beim Car-Sharing	☐	☐	☐	☐	☐
freien Parkplätzen	☐	☐	☐	☐	☐
staufreien Autostrecken	☐	☐	☐	☐	☐
freien Fahrrädern bei Fahrradverleihern	☐	☐	☐	☐	☐
Position eines Zuges/Busses auf den Sie warten	☐	☐	☐	☐	☐
freien Tische in Restaurants	☐	☐	☐	☐	☐
kurzen Warteschlangen vor Imbissen	☐	☐	☐	☐	☐
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	☐	☐	☐	☐	☐
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	☐	☐	☐	☐	☐
sonnigen, ruhigen Orten	☐	☐	☐	☐	☐

Mehrfachnennung möglich.

Suchergebnisse

Diese abschließenden Fragen behandeln die Darstellung der Ergebnisse.

36 [search_result]Welche Informationen sollen auf der Ergebnisseite für gefundene Dinge für folgende Suchanfragen angezeigt werden? Suche nach ...

	Beschreibung des Dings	Ort	Entfernung in Metern	Entfernung in Minuten und Sekunden	Wert des Zustandes des Dings	Aktualität des Zustandes (Zeit seit letzter Messung)	Weiß Nicht
freien Taxen	☐	☐	☐	☐	☐	☐	☐
freien Autos beim Car-Sharing	☐	☐	☐	☐	☐	☐	☐
freien Parkplätzen	☐	☐	☐	☐	☐	☐	☐
staufreien Autostrecken	☐	☐	☐	☐	☐	☐	☐
freien Fahrrädern bei Fahrradverleihern	☐	☐	☐	☐	☐	☐	☐
Position eines Zuges/Busses auf den Sie warten	☐	☐	☐	☐	☐	☐	☐
freien Tische in Restaurants	☐	☐	☐	☐	☐	☐	☐
kurzen Warteschlangen vor Imbissen	☐	☐	☐	☐	☐	☐	☐
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	☐	☐	☐	☐	☐	☐	☐
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	☐	☐	☐	☐	☐	☐	☐
sonnigen, ruhigen Orten	☐	☐	☐	☐	☐	☐	☐

Mehrfachnennung möglich.

Genauere Beschreibung der Antwortmöglichkeiten:

- **Beschreibung des Dings:** Ein beschreibender Text des Dings, z.B. "Italienisches Restaurant auf mittlerem Preisniveau"
- **Ort:** Der Ort des Dings
- **Entfernung in Metern:** Die Entfernung von der eigenen Position zu der des Dings, z.B. "Die Entfernung beträgt 1300 Meter"
- **Entfernung in Minuten und Sekunden:** Wie viel Zeit man zu dem Ding benötigt, z.B. "Sie benötigen 7 Minuten zu Fuß"
- **Wert des Zustandes des Dings:** Der momentane Zustand des Dings, z.B. "es sind 3 Tische frei"
- **Aktualität des Zustandes (Zeit seit letzter Messung):** Die vergangene Zeit seit dem der Zustand zum letzten Mal gemessen wurde, z.B. "vor 5 Minuten waren 3 Tische frei"

37 [search_ranking]Wonach sollen Ergebnisse auf der Ergebnisseite für folgende Suchanfragen sortiert sein? Suche nach ...

Bitte wählen Sie die zutreffende Antwort für jeden Punkt aus:

	Beliebtheit des Ortes (Stadtmitte ist beliebter als Stadtrand)	Entfernung in Metern	Entfernung in Minuten und Sekunden	Aktualität des Zustandes	Bewertungen anderer des Dings
freien Taxen	○	○	○	○	○
freien Autos beim Car-Sharing	○	○	○	○	○
freien Parkplätzen	○	○	○	○	○
staufreien Autostrecken	○	○	○	○	○
freien Fahrrädern bei Fahrradverleihern	○	○	○	○	○
Position eines Zuges/Busses auf den Sie warten	○	○	○	○	○
freien Tische in Restaurants	○	○	○	○	○
kurzen Warteschlangen vor Imbissen	○	○	○	○	○
einem verlorenen Gegenstand (der einen Positionsensor integriert hat)	○	○	○	○	○
Wassertemperaturen von Schwimmmöglichkeiten (See, Meer, Schwimmbad)	○	○	○	○	○
sonnigen, ruhigen Orten	○	○	○	○	○

Bitte geben Sie in jeder Zeile das für Sie wichtigste Merkmal an, nach welchen die Ergebnisse sortiert werden sollen.

Genauere Beschreibung der Antwortmöglichkeiten:

- **Beliebtheit des Ortes (Stadtmitte ist beliebter als Stadtrand)**: Dinge an beliebten Orten wie der Stadtmitte werden bei der Sortierung bevorzugt, d.h. weiter oben in der Liste angezeigt.
- **Entfernung in Metern:** Die gefundenen Dinge werden entsprechend ihrer Entfernung in Metern vom eigenen Standort zu ihnen sortiert. Kürzere Strecken werden weiter oben in der Liste angezeigt.
- **Entfernung in Minuten und Sekunden:** Die gefundenen Dinge werden entsprechend der Zeit sortiert, die benötigt wird um vom eigenen Standort zu ihnen zu gelangen. Kürzere Zeiten werden weiter oben in der Liste angezeigt.
- **Aktualität des Zustandes:** Die gefundenen Dinge werden anhand der vergangene Zeit seit dem der Zustand zum letzten Mal gemessen wurde sortiert. Aktuellere Zustände werden weiter oben in der Liste angezeigt.
- **Bewertungen anderer des Dings**: Dinge werden anhand von Bewertungen (wie heutzutage auf Qype oder Google Places möglich) sortiert. Dinge mit besserer Bewertung werden weiter oben in der Liste angezeigt.

38 [granularity_light]Wie (mit welcher Genauigkeit) sollte die Helligkeit angezeigt werden wenn Sie z.B. nach einem sonnigen Ort suchen?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ 90.000 Lux
- ○ 90% der maximalen Sonneneinstrahlung
- ○ Sonnig

39 [granularity_pos]Wie (mit welcher Genauigkeit) sollte die Position eines Dings zusätzlich zu einer Kartenansicht angezeigt werden?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ Geokoordinaten (Longitude und Latitude)
- ○ Straße, Stadt
- ○ Straße, Hausnummer, Stadt
- ○ Gar nicht (Kartenansicht reicht)

40 [granularity_occ]Wie (mit welcher Genauigkeit) sollte die Belegung eines Restaurants angezeigt werden?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ 3 von 10 Tischen frei (8 von 50 Plätzen frei)
- ○ 3 von 10 Tischen frei
- ○ 3 Tische frei (8 Plätze frei)
- ○ 3 Tische frei
- ○ Noch Tische frei

41 [granularity_temp]Wie (mit welcher Genauigkeit) sollte die Temperatur eines Dings angezeigt werden?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ 23,21 Grad
- ○ 23,1 Grad
- ○ 23 Grad
- ○ Mehr als 20 Grad
- ○ Warm

42 [granularity_time]Wie (in welcher Genauigkeit) sollten Wartezeiten angezeigt werden?

Bitte wählen Sie nur eine der folgenden Antworten aus:

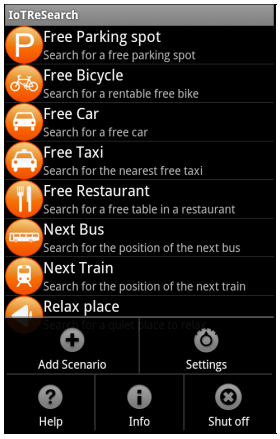
- ○ 7 Minuten und 20 Sekunden
- ○ ca. 5 Minuten
- ○ ca. 5-10 Minuten
- ○ kurze Wartezeit

43 [result_number]Wie viele Ergebnisse sollen für eine Suchanfrage auf der ersten Ergebnisseite angezeigt werden?

Bitte wählen Sie nur eine der folgenden Antworten aus:

- ○ 1
- ○ 3
- ○ 10
- ○ 100
- ○ Alle

Abschluss



Gegebenenfalls planen wir weitere Umfragen zum Thema "Internet der Dinge" durchzuführen. Falls Sie auch für weitere Umfragen zur Verfügung stehen würden, wäre es nett, wenn Sie ihre Mailadresse hinterlassen. Falls Sie dies nicht möchten, lassen Sie das Feld einfach leer. Die Bereitschaft kann jederzeit per Mail an mietz@iti.uni-luebeck.de widerrufen werden. Auch in jeder weiteren Mail wird darauf hingewiesen, so dass Sie die Mailadresse nicht jetzt aufschreiben müssen.

Hinweis: Die Mailadresse wird nicht genutzt, um Umfrageergebnisse zu personalisieren. Sie wird nur zum Zwecke der Mitteilung über neue Umfragen genutzt.

Des Weiteren haben wir eine Android-Anwendung entwickelt mit der wir das Suchverhalten im zukünftigen Internet der Dinge untersuchen möchten. Diese Anwendung hat einige vordefinierte Suchszenarien (u.a Suche nach freiem Parkplatz und freiem Taxi). Außerdem können neue Szenarien nach eigenen Bedürfnissen hinzugefügt werden. Die Idee ist, dass der Nutzer über die Anwendung oder das Widget ein Suchszenario "ausführt" sobald er denkt, dass er eine solche Information gebrauchen könnte (z.B. morgens in der Nähe der Arbeitsstelle einen freien Parkplatz finden). Dabei wird die Zeit, Position, das gesuchte Szenario sowie ein paar Daten zur Identifizierung des Geräts (u.a. Android-Version) gespeichert und an einen Server übermittelt. Die Anwendung hat keinen Zugriff auf persönliche Informationen und kann diese somit auch nicht übermitteln. Da die Suchen Zukunftsszenarien darstellen kann die Anwendung leider keine Daten zurückliefern. Sie könnten mit der Nutzung der Anwendung einen Beitrag zu unserer aktuellen Forschung leisten. Die Anwendung kann unter http://www.iti.uni-luebeck.de/mitarbeiter/dipl-inf-richard-mietz/iotresearch heruntergeladen werden. (Am besten direkt mit dem Android-Handy besuchen für eine leichtere Installation).

44 [mail]Bitte geben Sie hier Ihre Mailadresse ein, falls Sie über weitere Umfragen informiert werden möchten:

Bitte geben Sie Ihre Antwort hier ein:

A.2. XML Schemata

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example.com/state-option"
           targetNamespace="http://www.example.com/state-option"
     xmlns:so="http://www.example.com/state-option"
     elementFormDefault="qualified"
           attributeFormDefault="unqualified">

 <!-- To describe a state resources -->
    <xs:element name="r" type="resource"/>

 <!-- To describe the mappings of a state resource along with the uri-path
     -->
    <xs:complexType name="resource">
        <xs:sequence>
            <xs:choice>
                <xs:element name="num" type="num_map" maxOccurs="unbounded"
                    />
                <xs:element name="str" type="string_map" maxOccurs="
                    unbounded"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>

 <!-- To describe a numerical mapping with lower bound,
        upper bound and state -->
    <xs:complexType name="num_map">
        <xs:sequence>
            <xs:sequence maxOccurs="unbounded">
                    <xs:element name="l" type="xs:double"/>
                    <xs:element name="h" type="xs:double"/>
                </xs:sequence>
            <xs:element name="s" type="xs:string" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

 <!-- To describe a string mapping with one or more strings and state -->
    <xs:complexType name="string_map">
        <xs:sequence>
            <xs:element name="str" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="s" type="xs:string" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Quelltext A.1: XML Schema für die Beschreibung einer Zustandsressource in der CoAP-HLS-Option.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example.com/state-option"
           targetNamespace="http://www.example.com/state-option"
     xmlns:so="http://www.example.com/state-option"
     elementFormDefault="qualified"
           attributeFormDefault="unqualified">

    <xs:element name="res" type="res"/>

 <!-- To describe one or more state resources -->
    <xs:complexType name="res">
        <xs:sequence>
            <xs:element name="r" type="resource" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

 <!-- To describe the mappings of a state resource along with the uri-path
     -->
    <xs:complexType name="resource">
        <xs:sequence>
            <xs:element name="p" type="xs:normalizedString" maxOccurs="1"/>
            <xs:choice>
                <xs:element name="num" type="num_map" maxOccurs="unbounded"
                    />
                <xs:element name="str" type="string_map" maxOccurs="
                    unbounded"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>

 <!-- To describe a numerical mapping with lower bound,
        upper bound and state -->
    <xs:complexType name="num_map">
        <xs:sequence>
            <xs:sequence maxOccurs="unbounded">
                    <xs:element name="l" type="xs:double"/>
                    <xs:element name="h" type="xs:double"/>
                </xs:sequence>
            <xs:element name="s" type="xs:string" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

 <!-- To describe a string mapping with one or more strings and state -->
    <xs:complexType name="string_map">
        <xs:sequence>
            <xs:element name="str" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="s" type="xs:string" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Quelltext A.2: XML Schema für die Beschreibung aller bestehenden Zustandsressourcen in der CoAP-HLS-Option.

A.3. Aufgabenstellungen der Usability-Evaluation

Aufgabe 1 Du möchtest mit ein paar Freunden etwas essen gehen. Deshalb möchtest du nach Restaurants mit freien Tischen suchen. Sobald Ergebnisse angezeigt werden und du das Fenster wieder minimiert hast, ist die Aufgabe abgeschlossen.

Aufgabe 2 Die Ergebnisse haben dich nicht zufriedengestellt. Deshalb möchtest du die Sortierung der Ergebnisse, also das Ranking anpassen. Du möchtest, dass die Ergebnisse hauptsächlich nach der Distanz und der Bewertung des Restaurants, d. h. dem Rating sortiert werden. Du möchtest außerdem, dass nur Restaurants mit 4 Sternen oder besserer Bewertung einbezogen werden. Die anderen Rankingkriterien sollen nicht einbezogen werden. Suche nun wieder nach Tischen. Sobald du Ergebnisse siehst und das Fenster wieder minimiert hast, ist die Aufgabe abgeschlossen.

Aufgabe 3 Die Suche entspricht nun deinen Wünschen. Da du evtl. öfter nach Tischen suchen möchtest, speichere die Suche unter einem beliebigen Namen. Sobald dies erledigt ist und du das Fenster wieder minimiert hast, ist die Aufgabe abgeschlossen.

Aufgabe 4 Du möchtest nun mit dem Auto in die Stadt und suchst deshalb einen freien Parkplatz auf der Innenstadtinsel. Das Ranking soll nur anhand der Kosten geschehen. Sobald du die nötigen Einstellungen gemacht, Ergebnisse bekommen und du das Fenster wieder minimiert hast, ist die Aufgabe abgeschlossen.

Aufgabe 5 Du möchtest nun nochmal die vorher gespeicherte Suche ausführen. Dabei fällt dir auf, dass du auch nach Restaurants außerhalb der Innenstadtinsel gesucht hast. Du möchtest aber, wie bei den Parkplätzen, auf der Innenstadtinsel suchen. Passe die Suche an, speichere diese und führe sie aus. Lösche außerdem die alte Suche. Sobald du dies erledigt hast, Ergebnisse bekommen und du das Fenster wieder minimiert hast, ist die Aufgabe abgeschlossen.

Aufgabe 6 Lasse dir die Route zum besten Ergebnis anzeigen. Sobald du die Route angezeigt bekommen hast, ist die Aufgabe abgeschlossen. Du bist dann fertig.

A.4. Fragebogen der Usability-Evaluation

Fragebogen zur Nutzung der GUI

Geschlecht: ☐ Männlich ☐ Weiblich **Alter**:

Bewerte deine PC-Kenntnisse (Schulnote von 1 bis 6):

Beruf:

	Starke Ablehnung				Starke Zustimmung
1. Ich denke, dass ich dieses System gerne regelmäßig nutzen würde.	1	2	3	4	5
2. Ich fand das System unnötig komplex.	1	2	3	4	5
3. Ich denke, das System war leicht zu benutzen.	1	2	3	4	5
4. Ich denke, ich würde die Unterstützung einer fachkundigen Person benötigen, um das System benutzen zu können.	1	2	3	4	5
5. Ich fand, die verschiedenen Funktionen des Systems waren gut integriert.	1	2	3	4	5
6. Ich halte das System für zu inkonsistent.	1	2	3	4	5
7. Ich glaube, dass die meisten Menschen sehr schnell lernen würden, mit dem System umzugehen.	1	2	3	4	5
8. Ich fand das System sehr umständlich zu benutzen.	1	2	3	4	5
9. Ich fühlte mich bei der Nutzung des Systems sehr sicher.	1	2	3	4	5
10. Ich musste viele Dinge lernen, bevor ich mit dem System arbeiten konnte.	1	2	3	4	5

Für folgende Fragen kannst Du auch die Rückseite nutzen:

Was gefiel dir besonders:

Was gefiel dir gar nicht:

Was war die unklar:

B
Verzeichnisse

Tabellenverzeichnis

1.1. Fragestellungen und mögliche Werte der Kategorien der Netzdiagramme. 12

2.1. Verschiedene Versionen der iSense-Plattform [37] der Firma Coalesenses [Daten aus 38–40]. 24
2.2. Condition-Typen der CoAP-Conditional-Observe-Option [nach 74, Abschnitt 4, Tabelle 3]. 33
2.3. Literaltypen und deren Visualisierung durch verschiedene Rahmentypen. 37
2.4. Relationen von Simple Feature, Egenhofer und RCC8 [nach 89]. . 40

3.1. Für die Umfrage verwendete Szenarien. 45
3.2. Von den Teilnehmern genannte Szenarien. 48

4.1. Sensortypen und von diesen produzierte Ausgabetypen. 66
4.2. Schlüsselwörter für die Vorlagen und entsprechende Ersetzungen. 74
4.3. Datentypen und deren Länge für die HLS-Option. 84

4.4. Antworttypen. 85
4.5. Anzahl Nachrichten für die in Abbildung 4.26 gezeigten Methode und deren Schwellwerte. 92
4.6. Anzahl Nachrichten für die in Abbildung 4.27 gezeigten Methoden und deren Schwellwerte. 94
4.7. Stündliche Temperaturwerte für einen Tag. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 4 aus Quelltext 4.3. 100
4.8. Stündliche Temperaturzustände (0 ≙ *kalt*, 1 ≙ *warm*) für einen Tag. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 3 aus Quelltext 4.3. 102
4.9. 24 Zustände (0 ≙ *kalt*, 1 ≙ *warm*) ohne zeitlichen Bezug. „...“ repräsentiert jeweils fünf ausgelassene Zeilen. Die Zahl in Klammern hinter dem Spaltennamen gibt die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 2 aus Quelltext 4.3. 102
4.10. Histogramm für Zustände (0 ≙ *kalt*, 1 ≙ *warm*). Die Zahlen in Klammern hinter den Spaltennamen geben die Anzahl Bits zur Speicherung an. Das Format der Daten entspricht *Level* 1 aus Quelltext 4.3. 103
4.11. Anzahl Bits, die zur Serialisierung der reinen Werte notwendig sind (ohne Overhead). Die Ersparnis bezieht sich auf die Originaldaten, d. h. *Level* 4. 103
4.12. Vier Sensoren mit jeweils vier Zustandsmessungen. 115
4.13. Korrelationen für jedes Sensorpaar. 115

5.1. Ergebnis der Anfrage aus Quelltext 5.2. 128
5.2. Ergebnis der Anfrage aus Quelltext 5.3. 129
5.3. Ergebnis der Anfrage aus Quelltext 5.4. 129
5.4. Ergebnis der Anfrage aus Quelltext 5.5. 130
5.5. Keyword-basierte Rankingfaktoren [nach 170]. 141
5.6. Link-basierte Rankingfaktoren [nach 170]. 142
5.7. Weitere Rankingfaktoren [nach 170]. 142
5.8. Anzahl benötigter RDF-Tripel für die verschiedenen Modellkonzepte. 144

6.1. Liste von P2P-Triplestores. 151

Abbildungsverzeichnis

1.1. Qualitativer Vergleich der vorgestellten Systeme. 13
1.1. Qualitativer Vergleich der vorgestellten Systeme. 14
1.1. Qualitativer Vergleich der vorgestellten Systeme. 15
1.2. Grafische Darstellung der Beiträge dieser Arbeit und deren Zusammenhänge. 18

2.1. Mooresches Gesetz und Anzahl Transistoren verschiedener Mikrochips, aufgetragen nach ihrer Markteinführung [Daten von 30]. 22
2.2. Unterschiedliche Dimensionen bei der Definition des Begriffes *Internet der Dinge* [Nach 69, Abb. 1]. 28
2.3. Zwei Aussagen in Form von Tripeln dargestellt als gerichtete Graphen. 35
2.4. XML Schema Datentypen [nach 81]. 36
2.5. Ein exemplarischer RDF-Graph für das Beispiel aus Abbildung 2.3. URIs werden als Elemente mit runden Kanten repräsentiert, während Literale als Rechtecke dargestellt werden. 37

3.1. Gesamtverteilung und nach Alter gruppierte Verteilung des Geschlechts. 46
3.2. Ergebnisse der Frage über den derzeitigen Beruf. 47
3.3. Ergebnisse der Frage zur höchsten abgeschlossenen Ausbildung. 47
3.4. Antworten auf die Fragen nach dem generellen Suchverhalten. . 50
3.5. Antworten auf die Frage über das Bereitstellen persönlicher Daten. 50
3.6. Antworten auf die Frage zum Nutzungsort. 52
3.7. Antworten auf die Frage zur Gerätenutzung. 53
3.8. Antworten auf die Frage zum Suchkontext. 54
3.9. Antworten auf die Fragen zum Nutzungszeitpunkt. 54
3.10. Antworten auf die Frage zur Nutzungshäufigkeit. 55
3.11. Antworten auf die Frage zur Informationsdarstellung bei Ergebnissen. 56
3.12. Antworten auf die Frage zur Zustandsgenauigkeit. 57
3.13. Antworten auf die Frage zur Sortierung der Ergebnisse. 58

3.14. Boxplots der Anzahl von Änderungen der Sensorrohwerte ( ) bzw. Anzahl von Zustandsänderungen mit zwei Zuständen. Die Zustandsgrenzen liegen bei 10 °C ( ), 15 °C ( ), 20 °C ( ) und der Mediantemperatur ( ). 58

4.1. LoCaF-Architektur. 68
4.2. Dialoge zur Verwaltung der Verbindung zum Sensorknoten. . . . 69
4.3. Dialoge zur Konfiguration der Bildverarbeitungsschritte. 70
4.4. Einstellungsdialoge für die Objekterkennung und die Veröffentlichung von Sensorzuständen. 73
4.5. Screenshot einiger Tweets durch LoCaF. 76
4.6. Originalbilder (links) und nach der Blob-Erkennung (rechts). Die Anzahl an tatsächlichen (links) und erkannten (rechts) Studenten beim Betreten des Hörsaals ist in der unteren rechten Ecke angegeben. 77
4.7. Geschätzte und tatsächliche Anzahl Personen beim Betreten des Hörsaals. 78
4.8. Geschätzte und tatsächliche Anzahl während der Vorlesung. . . 78
4.9. Geschätzte und tatsächliche Anzahl Personen beim Verlassen des Hörsaals. 79
4.10. Vergleich der tatsächlichen Anzahl an Personen mit der geschätzten Anzahl ohne und mit Anwendung von Filtern. 79
4.11. Vergleich der tatsächlichen Anzahl an Personen mit der geschätzten Anzahl ohne und mit Anwendung von CLAHE. 80
4.12. Originalbilder (links) und nach der Blob-Erkennung (rechts) eines einparkenden Autos auf einem Parkplatz. Das letzte Bild stellt eine mögliche Szenenauswahl eines einzelnen Parkplatzes dar. 81
4.13. Veranschaulichung der Ressourcenabfrage durch einen Client. Die Sensorressource greift auf den Sensor zu, der wiederum den phys./chem. Prozess überwacht. Die Zustandsressourcen erzeugen ihren Zustand aus dem Wert der Sensorressource. 83
4.14. Format der HLS-Option. 83
4.15. Formate für die Zustandsermittlung. 84
4.16. Kommunikation zwischen Client 1 und dem Server zur Erzeugung einer Zustandsressource mit zwei Zuständen. 87
4.17. Kommunikation zwischen Client 2 und dem Server. Client 2 erzeugt eine Zustandsressource mit vier Zuständen. 87
4.18. Kommunikation zwischen Client 3 und dem Server. Client 3 möchte eine Zustandsressource mit gleicher Semantik wie eine bereits existierende Ressource erzeugen. Die Antwort beinhaltet die Adresse der bestehenden Ressource. 88
4.19. Kommunikation zwischen Client 4 und dem Server. Die angeforderte Zustandsressource kann durch den Server nicht erstellt werden, da zu wenig Ressourcen vorhanden sind. 88
4.20. Kommunikation zwischen Client 4 und dem Server zur Abfrage aller existierenden Zustandsressourcen. 89

4.21. Kommunikation zwischen Client 1 und dem Server zur Erzeugung einer Zustandsressource mit vier Zuständen unter Nutzung der String-Abbildung. 89
4.22. Kommunikation zwischen Client 1 und dem Server zur Abfrage des aktuellen Zustandes. 90
4.23. Kommunikation zwischen Client 1 und dem Server zum Löschen der Zustandsressource. 90
4.24. Diagramm der rohen Lichtwerte eines Tages. Die farbigen Flächen markieren die Zustände *dunkel* und *hell*. Unter dem Diagramm repräsentiert jede Markierung eine Nachricht, die durch Observe (Obs) bzw. die HLS-Option in Kombination mit Observe (HLS+Obs) für einen Schwellwert von 100 Lux verschickt wird. . 92
4.25. Diagramm der rohen Temperaturwerte von zwei Monaten. Die farbigen Flächen markieren die Zustände *kalt* und *warm*. Unter dem Diagramm repräsentiert jede Markierung eine Nachricht, die durch die HLS-Option in Kombination mit Observe bzw. Conditional Observe (CO) für einen Schwellwert von $10\,°$C (CO_{10}) verschickt wird. 93
4.26. Diagramm wie in Abbildung 4.25 mit denselben Daten aber drei Zuständen. Die Schwellwerte der Zustände sind $0\,°$C (CO_0), $10\,°$C (CO_{10}) und $20\,°$C (CO_{20}). 94
4.27. Diagramm wie in Abbildung 4.25 mit denselben Daten aber sechs Zuständen. Die Schwellwerte der Zustände sind $-10\,°$C (CO_{-10}), $8\,°$C (CO_8), $10\,°$C (CO_{10}), $15\,°$C (CO_{15}) und $20\,°$C (CO_{20}). 95
4.28. Architektur zur Erzeugung virtueller Sensoren. 105
4.29. Ausschnitte der GUI zur Erstellung eines virtuellen Sensors. . . . 106
4.30. Evaluationseinstellungen für einen virtuellen Sensor für die Webseite `http://www.wetter.com/`. 108
4.31. Diagramm der durch die Erweiterung extrahierten Temperaturwerte. 109
4.32. Veranschaulichung des Zusammenhangs der vorgestellten Technologien zur Zustandsermittlung. 110
4.33. Karte von Barcelonas Bicing Mietstationen. Kreise markieren Mengen von Stationen mit gleichem Zustand, d. h. einer ähnlichen Anzahl an verfügbaren Fahrrädern. 117
4.34. Boxplots der Anzahl an Sensoranfragen für 384 Sensoren, um k = 1 und k = 10 Sensoren mit gesuchtem Zustand zu finden. Vergleicht werden jeweils ein zufälliges Anfragemuster und die Verwendung von Sensor Ranking mit Wahrscheinlichkeitsmodellen, die auf Bayes'schen Netzen basieren. Die Maxima aller Plots wurden zur besseren Darstellung der Boxen nicht geplottet und liegen bei 384. Die rote Linie gibt das erreichbare Optimum an. 118

4.35. Veranschaulichung der Ressourcenabfrage durch einen Client. Die Sensorressource greift auf den Sensor zu der wiederum den phys./chem. Prozess überwacht. Die Wahrscheinlichkeitsmodellressourcen erzeugen das Wahrscheinlichkeitsmodell aus Sensorwerten oder Zuständen der darunterliegenden Ressource. . . 120
4.36. Format der Probability-Model-Option. 120

5.1. RDF zur Beschreibung eines Sensorknotens und der angeschlossenen Sensoren. 125
5.2. RDF zur Beschreibung von Sensorzuständen. 126
5.3. RDF zur Beschreibung der Historie von Sensorzuständen. 130
5.4. RDF zur Beschreibung eines APMs. 131
5.5. RDF zur Beschreibung eines TPMs. 132
5.6. Varianten von TPMs. 133
5.7. Sensor mit zwei TPMs. 134
5.8. RDF zur Beschreibung eines PPMs. 136
5.9. RDF zur Beschreibung von intra-Sensor-Korrelationen. 137
5.10. RDF zur Beschreibung von inter-Sensor-Korrelationen. 138
5.11. RDF zur Beschreibung von Zustandsabfragen. 139
5.12. RDF zur Beschreibung von Rankingfaktoren für einen Sensor. . . 141

6.1. DecentSPARQL-Architektur. 154
6.2. Benötigte Zeit in Millisekunden für 1.000, 10.000 und 100.000 Einfügeoperationen mit Netzgrößen von einem, vier und acht Peers mit jeweils einem, vier oder acht genutzten Threads zum Einfügen. 156
6.3. Benötigte Zeit in Millisekunden für 1.000 SPARQL-Anfragen mit Netzgrößen von einem, vier und acht Peers mit jeweils einem, vier oder acht genutzten Threads für die Anfragen. 157
6.4. Benötigte Zeit in Millisekunden für 1.000 SPARQL-Anfragen mit Netzgrößen von einem, vier, acht Peers und sechzehn Peers. Die Anfragen sind zufällig auf die Peers verteilt. 158
6.5. Die GUI zur Suche im IoT. Die Zahlen und Buchstaben dienen zur Kennzeichnung von Bereichen zur Referenzierung aus dem Text. 161
6.6. Ausschnitte der GUI. 162
6.7. Routingfunktion der GUI. 163
6.8. Antworthäufigkeiten für die Antworten 1 (starke Ablehnung) bis 5 (starke Zustimmung) der zehn Fragen des SUS-Fragebogens. Für Fragen mit ungerader Fragenummer ist 5 die beste Antwort. Für Fragen mit gerader Fragenummer ist 1 die beste Antwort. . . 164

7.1. Qualitativer Vergleich der Summe der in Kapitel 1 betrachteten Systeme mit den Technologien dieser Arbeit. 170

Quelltextverzeichnis

2.1. Turtle-Notation der Tripel aus dem Beispiel aus Abbildung 2.5. . 38
2.2. Turtle-Notation der Tripel mit Präfixen aus dem Beispiel in Abbildung 2.5. 38
2.3. SPARQL-Anfrage, um die 20 größten Städte mit mehr als 5.000 Einwohnern sowie die Bundesländer in denen diese liegen zu selektieren. 39

4.1. Beispiel von zwei *map*-Regeln für einen Tagungsraum. 73
4.2. Beispiel von zwei *switch*-Regeln für einen Parkplatz. 73
4.3. LoD-Skriptfragment. 101
4.4. Beispiel von drei *map*-Regeln. 107
4.5. Beispiel von zwei *regex*-Regeln. 107
4.6. Sensor Ranking im Pseudocode. 112
4.7. Abfragealgorithmus im Pseudocode. 116

5.1. Verwendete Namensräume und deren Präfixe. 125
5.2. Anfrage, um alle Zustände eines Sensors abzurufen. 128
5.3. Anfrage, um den Zustand für 21 °C abzurufen. 128
5.4. Anfrage, um den Zustand für den Begriff „warm“ abzurufen. . . 129
5.5. Anfrage, um den letzten Zustand des Sensors abzurufen. 130
5.6. Anfrage, um die Wahrscheinlichkeit für den Zustand mit dem beschreibendem Begriff „warm“ von einem gegebenen Sensor abzurufen. 131
5.7. Anfrage, um die Wahrscheinlichkeit für den Zustand mit dem Begriff „warm“ um 11 Uhr morgens von einem gegebenen Sensor abzurufen. 134
5.8. Notwendige Änderung für die Anfrage aus Quelltext 5.7. 135
5.9. Anfrage, um alle Sensoren zu selektieren, die ein TPM oder APM besitzen. 135
5.10. Anfrage, um alle Sensoren zu selektieren, die PPMs, TPMs oder ein APM besitzen. 137
5.11. Anfrage für das intra-CPM. 138
5.12. SPARQL-Anfrage zur Erzeugung eines APMs. 145
5.13. Teil einer SPARQL-Anfrage zur Erzeugung eines Teils eines TPMs. 146

A.1. XML Schema für die Beschreibung einer Zustandsressource in der CoAP-HLS-Option. 179
A.2. XML Schema für die Beschreibung aller bestehenden Zustandsressourcen in der CoAP-HLS-Option. 180

Abkürzungsverzeichnis

6LoWPAN IPv6 over Low power Wireless Personal Area Network 27
AHE Adaptive Histogram Equalization 70
Ajax Asynchronous JavaScript and XML 108
API Application Programming Interface 8
APM aggregiertes Wahrscheinlichkeitsmodell (engl.: aggregated probability model) 113
BN Bayes'sches Netz 114
CAN Content Addressable Network 151
CLAHE Contrast Limited Adaptive Histogram Equalization 71
CoAP Constrained Application Protocol 17
CoRE Constrained RESTful Environments 31
CSV Comma-separated values 123
DHT Distributed Hash Table 150
GML Geography Markup Language 40
GPS Global Positioning System 25
GUI grafische Benutzungsoberfläche (engl.: graphical user interface) 10
HLS High-Level-State 82
HTML Hypertext Markup Language 30
HTTP Hypertext Transfer Protocol 29
IETF Internet Engineering Task Force 31
inter-CPM inter-Sensor-korrelationsbasiertes Wahrscheinlichkeitsmodell (engl.: inter-sensor correlation-based probability model) 113
intra-CPM intra-Sensor-korrelationsbasiertes Wahrscheinlichkeitsmodell (engl.: intra-sensor correlation-based probability model) 113
IoT Internet of Things(dt.: Internet der Dinge) 2
IP Internet Protocol 27
IPv6 Internet Protocol Version 6 27
IRI Internationalized Resource Identifier 35
JPEG Joint Picture Expert Group 68
JSON Javascript Object Notation 30
kB Kilobyte 7
LoCaF Lousy Camera Framework 68
LoD Detailierungsgrad (engl.: Level of Detail) 98
OGC Open Geospatial Consortium 39
P2P Peer-to-Peer 150
PIR Passive Infrared 25

PNetS Personalized Networked Space 10
PPM ortabhängiges Wahrscheinlichkeitsmodell (engl.: place-dependent probability model) 113
RCC Region Connection Calculus 40
RDF Resource Description Framework 17
RDQL RDF Data Query Language 151
REST Representational State Transfer 30
RFID Radio Frequency Identification 4
SPARQL SPARQL Protocol and RDF Query Language 37
SPARUL SPARQL/Update 39
SQL Structured Query Language 38
SSN Semantic Sensor Network 124
SUS System Usability Scale 163
TCP Transmission Control Protocol 31
TPM zeitabhängiges Wahrscheinlichkeitsmodell (engl.: time-dependent probability model) 113
Turtle Terse RDF Triple Language 37
UDP User Datagram Protocol 31
URI Uniform Resource Identifier 30
URL Uniform Resource Locator 142
W3C World Wide Web Consortium 30
WKT Well-known Text 40
WoT Web of Things 29
WSN Wireless Sensor Network 25
WWW World Wide Web 34
XHTML Extensible Hypertext Markup Language 34
XML Extensible Markup Language 30
XPath XML Path Language 106

Eigene Publikationen

[BFF+10] Daniel Bimschas, Sándor Fekete, Stefan Fischer, Mesut Günes, Oliver Hahm, Horst Hellbrück, Alexander Kröller, Richard Mietz, Max Pagel, Dennis Pfisterer, Kay Römer, Kaspar Schleiser und Torsten Teubler. Real-World G-Lab: Goals, Current Status and Future Plans. *G-Lab Status Seminar*, 2010.

[BFF+11] Daniel Bimschas, Sándor Fekete, Stefan Fischer, Horst Hellbrück, Alexander Kröller, Richard Mietz, Max Pagel, Dennis Pfisterer, Kay Römer und Torsten Teubler. Real-World G-Lab: Integrating Wireless Sensor Networks with the Future Internet. In Thomas Magedanz, Anastasius Gavras, Nguyen Huu Thanh und Jeffry S. Chase, Herausgeber, *Testbeds and Research Infrastructures. Development of Networks and Communities*, Band 46 von *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Seiten 577–579. Springer Berlin Heidelberg, 2011. URL `http://dx.doi.org/10.1007/978-3-642-17851-1_46`.

[BHH+11] Daniel Bimschas, Henning Hasemann, Manfred Hauswirth, Marcel Karnstedt, Oliver Kleine, Alexander Kröller, Myriam Leggieri, Richard Mietz, Alexandre Passant, Dennis Pfisterer et al. Semantic-Service Provisioning for the Internet of Things. *Electronic Communications of the EASST*, Band 37, 2011. URL `http://journal.ub.tu-berlin.de/eceasst/article/view/473`.

[BHM+10] Daniel Bimschas, Horst Hellbrück, Richard Mietz, Dennis Pfisterer, Kay Römer und Torsten Teubler. Middleware for Smart Gateways Connecting Sensornets to the Internet. In *Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, Seiten 8–14. ACM, 2010. URL `http://dx.doi.org/10.1145/1890784.1890787`.

[GMR+10] Nils Glombitza, Richard Mietz, Kay Römer, Stefan Fischer und Dennis Pfisterer. Self-Description and Protocol Conversion for a Web of Things. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, SUTC, Seiten 229–236. IEEE, 2010. URL `http://dx.doi.org/10.1109/SUTC.2010.13`.

[MAR14] Richard Mietz, Philipp Abraham und Kay Römer. High-level States with CoAP: Giving Meaning to Raw Sensor Values to Support IoT Applications. In *Proceedings of the Ninth IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, ISSNIP. Singapore, April 2014.

[MBB+14] Richard Mietz, Carsten Buschmann, Dennis Boldt, Kay Römer und Stefan Fischer. Poster Abstract: iBAST–Instantaneous Bridge Assessment Based on Sensor Network Technology. In Koen Langendoen, Wen Hu, Federico Ferrari, Marco Zimmerling und Luca Mottola, Herausgeber, *Real-World Wireless Sensor Networks*, Band 281 von *Lecture Notes in Electrical Engineering*, Seiten 85–89. Springer International Publishing, 2014. ISBN 978-3-319-03070-8. URL http://dx.doi.org/10.1007/978-3-319-03071-5_10.

[MGK+13] Richard Mietz, Sven Groppe, Oliver Kleine, Daniel Bimschas, Stefan Fischer, Kay Römer und Dennis Pfisterer. A P2P Semantic Query Framework for the Internet of Things. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, Seiten 1–7, 2013. URL http://dx.doi.org/10.1515/pik-2013-0006.

[MGRP13] Richard Mietz, Sven Groppe, Kay Römer und Dennis Pfisterer. Semantic Models for Scalable Search in the Internet of Things. *Journal of Sensor and Actuator Networks*, Band 2(2):Seiten 172–195, 2013. URL http://dx.doi.org/10.3390/jsan2020172.

[Mie13] Richard Mietz. CoAP High-Level State Option Extension. IETF Internet Draft – work in progress 01, individual, Dezember 2013. URL https://datatracker.ietf.org/doc/draft-mietz-core-coap-state-option/.

[MMR12] Benjamin Meyer, Richard Mietz und Kay Römer. LoCaF: Detecting Real-World States with Lousy Wireless Cameras. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, DCOSS, Seiten 58–66. IEEE, 2012. URL http://dx.doi.org/10.1109/DCOSS.2012.9.

[MR11] Richard Mietz und Kay Römer. Exploiting correlations for efficient content-based sensor search. In *2011 IEEE Sensors*, Seiten 187–190. IEEE, 2011. URL http://dx.doi.org/10.1109/ICSENS.2011.6127082.

[MR12] Richard Mietz und Kay Römer. Work in Progress: Resourse-Aware Fault Localization in Large Sensor Networks. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, DCOSS, Seiten 281–283. IEEE, 2012. URL http://dx.doi.org/10.1109/DCOSS.2012.49.

[MR13] Richard Mietz und Kay Römer. Smart City Search: A User Survey. In Juan A. Botía und Dimitris Charitos, Herausgeber, *Intelligent*

Environments (Workshops), Band 17, Seiten 635–646. IOS Press, 2013. ISBN 978-1-61499-286-8. URL http://dx.doi.org/10.3233/978-1-61499-286-8-635.

[PRB+11] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth et al. SPITFIRE: Toward a Semantic Web of Things. *IEEE Communications Magazine*, Band 49(11):Seiten 40–48, 2011. URL http://dx.doi.org/10.1109/MCOM.2011.6069708.

Literaturverzeichnis

[1] Dave Evans. The Internet of Things – How the Next Evolution of the Internet Is Changing Everything. whitepaper, Cisco Internet Business Solutions Group (IBSG), 2011.

[2] Jon Iwata. IBM Investor Briefing – Making Markets: Smarter Planet. 2012. http://www.ibm.com/investor/events/investor0512/presentation/05_Smarter_Planet.pdf [Zugriff am 21. März 2014].

[3] Inc. Gartner. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. 2013. https://www.gartner.com/newsroom/id/2636073 [Zugriff am 21. März 2014].

[4] Alexa Internet. Alexa Top 500 Global Sites. http://www.alexa.com/topsites, 2014. [Zugriff am 21. März 2014].

[5] Kok-Kiong Yap, Vikram Srinivasan und Mehul Motani. MAX: Human-Centric Search of the Physical World. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, Seiten 166–179. ACM, New York, NY, USA, 2005. ISBN 1-59593-054-X. URL http://dx.doi.org/10.1145/1098918.1098937.

[6] Christian Frank, Philipp Bolliger, Christof Roduner und Wolfgang Kellerer. Objects Calling Home: Locating Objects Using Mobile Phones. In *Proceedings of the 5th international conference on Pervasive computing*, PERVASIVE'07, Seiten 351–368. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72036-2. URL http://dx.doi.org/10.1007/978-3-540-72037-9_21.

[7] Haodong Wang, C.C. Tan und Qun Li. Snoogle: A Search Engine for the Physical World. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Seiten –. 2008. ISSN 0743-166X. URL http://dx.doi.org/10.1109/INFOCOM.2008.196.

[8] Haodong Wang, Chiu C. Tan und Qun Li. Snoogle: A Search Engine for Pervasive Environments. In *IEEE Trans. Parallel Distrib. Syst.*, Band 21:Seiten 1188–1202, 2010. ISSN 1045-9219. URL http://dx.doi.org/10.1109/TPDS.2009.145.

[9] Chiu C. Tan, Bo Sheng, Haodong Wang und Qun Li. Microsearch: When Search Engines Meet Small Devices. In *Pervasive Computing* (herausgegeben von Jadwiga Indulska, DonaldJ. Patterson, Tom Rodden und Max Ott), Band 5013 von *Lecture Notes in Computer Science*, Seiten 93–110. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79575-9. URL http://dx.doi.org/10.1007/978-3-540-79576-6_6.

[10] Chiu C. Tan, Bo Sheng, Haodong Wang und Qun Li. Microsearch: A Search Engine for Embedded Devices Used in Pervasive Computing. In *ACM Trans. Embed. Comput. Syst.*, Band 9:Seiten 43:1–43:29, 2010. ISSN 1539-9087. URL http://dx.doi.org/10.1145/1721695.1721709.

[11] Tingxin Yan, Deepak Ganesan und Manmatha R. Distributed Image Search in Camera Sensor Networks. In *Proceedings ACM Sensys*, Seiten 155–168. 2008. URL http://dx.doi.org/10.1145/1460412.1460428. http://lass.cs.umass.edu/~yan/pubs/SenSys08-imgSearch.pdf.

[12] Xively. Xively – Public Cloud for the Internet of Things. 2014. http://www.xively.com/ [Zugriff am 21. März 2014].

[13] Ming Li, Deepak Ganesan und Prashant Shenoy. PRESTO: Feedback-driven Data Management in Sensor Networks. In *IEEE/ACM Trans. Netw.*, Band 17:Seiten 1256–1269, 2009. ISSN 1063-6692. URL http://dx.doi.org/10.1109/TNET.2008.2006818.

[14] Benedikt Ostermaier, B. Maryam Elahi, Kay Römer, Michael Fahrmair und Wolfgang Kellerer. Dyser: Towards a Real-Time Search Engine for the Web of Things. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, Seiten 429–430. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-990-6. URL http://dx.doi.org/10.1145/1460412.1460487.

[15] Jonas Michel, Christine Julien, Jamie Payton und Gruia-Catalin Roman. The Gander Search Engine for Personalized Networked Spaces. Technischer Bericht, Technical Report TR-ARiSE-2011-008, The University of Texas at Austin, 2011. http://mpc.ece.utexas.edu/Papers/TR-UTEDGE-2011-008.pdf.

[16] Jonas Reinhardt Michel. *The Gander Search Engine for Personalized Networked Spaces*. Diplomarbeit, The University of Texas at Austin, 2012. https://repositories.lib.utexas.edu/bitstream/handle/2152/19714/michel_thesis_2012912.pdf?sequence=1.

[17] Cuong Truong. *Routing and Sensor Search in the Internet of Things*. Dissertation, University of Lübeck, 2013.

[18] B. Maryam Elahi, Kay Römer, Benedikt Ostermaier, Michael Fahrmair und Wolfgang Kellerer. Sensor Ranking: A Primitive for Efficient Content-based Sensor Search. In *Proceedings of the 2009 Intl. Conference on Information Processing in Sensor Networks*, Seiten 217–228. 2009. http://vs.inf.ethz.ch/publ/papers/sensorranking.pdf.

[19] Rudolf Kruse, Jörg Gebhardt und Frank Klawonn. *Fuzzy-Systeme (2. Aufl.)*. Leitfäden der Informatik. Teubner, 1995. ISBN 978-3-519-12130-5.

[20] André B. Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, Seiten 195–203. ACM, New York, NY, USA, 2000. ISBN 1-58113-195-X. URL `http://dx.doi.org/10.1145/350391.350432`.

[21] Deutsches Institut für Normung DIN. *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten Teil 11: Anforderungen an die Gebrauchstauglichkeit - Leitsätze*. DIN EN ISO 9241-11. Beuth, Berlin, 1998.

[22] George W. Furnas, Thomas K. Landauer, Louis M. Gomez und Susan T. Dumais. The Vocabulary Problem in Human-system Communication. In *Commun. ACM*, Band 30(11):Seiten 964–971, 1987. ISSN 0001-0782. URL `http://dx.doi.org/10.1145/32206.32212`.

[23] Richard Mietz und Kay Römer. Smart City Search: A User Survey. In *Intelligent Environments (Workshops)* (herausgegeben von Juan A. Botía und Dimitris Charitos), Band 17, Seiten 635–646. IOS Press, 2013. ISBN 978-1-61499-286-8. URL `http://dx.doi.org/10.3233/978-1-61499-286-8-635`.

[24] Benjamin Meyer, Richard Mietz und Kay Römer. LoCaF: Detecting Real-World States with Lousy Wireless Cameras. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, DCOSS, Seiten 58–66. IEEE, 2012. URL `http://dx.doi.org/10.1109/DCOSS.2012.9`.

[25] Richard Mietz. CoAP High-Level State Option Extension. IETF Internet Draft – work in progress 01, individual, 2013. URL `https://datatracker.ietf.org/doc/draft-mietz-core-coap-state-option/`.

[26] Richard Mietz, Philipp Abraham und Kay Römer. High-level States with CoAP: Giving Meaning to Raw Sensor Values to Support IoT Applications. In *Proceedings of the Ninth IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, ISSNIP 2014. Singapore, 2014.

[27] Richard Mietz und Kay Römer. Work in Progress: Resourse-Aware Fault Localization in Large Sensor Networks. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, DCOSS, Seiten 281–283. IEEE, 2012. URL `http://dx.doi.org/10.1109/DCOSS.2012.49`.

[28] Richard Mietz, Sven Groppe, Oliver Kleine, Daniel Bimschas, Stefan Fischer, Kay Römer und Dennis Pfisterer. A P2P Semantic Query Framework for the Internet of Things. In *PIK-Praxis der Informationsverarbeitung und Kommunikation*, Seiten 1–7, 2013. URL `http://dx.doi.org/10.1515/pik-2013-0006`.

[29] Gordon E. Moore et al. Cramming more components onto integrated circuits. 1965.

[30] Wikipedia. Transistor count. https://en.wikipedia.org/w/index.php?title=Transistor_count&oldid=590730642, 2014. [Zugriff am 17. Januar 2014].

[31] Coalesenses GmbH. Moore's Law: „We See No End in Sight" Says Intel's Pat Gelsinger | JAVA Developer's Journal. 2014. http://java.sys-con.com/node/557154 [Zugriff am 21. März 2014].

[32] Jonathan G. Koomey, Stephen Berard, Marla Sanchez und Henry Wong. Implications of Historical Trends in the Electrical Efficiency of Computing. In *IEEE Annals of the History of Computing*, Band 33(3):Seiten 46–54, 2011. ISSN 1058-6180. URL http://dx.doi.org/10.1109/MAHC.2010.28.

[33] Mark Weiser. The Computer for the 21st Century. In *Scientific american*, Band 265(3):Seiten 94–104, 1991.

[34] Gyuhae Park, Tajana Rosing, Michael Todd, Charles R. Farrar und William Hodgkiss. Energy Harvesting for Structural Health Monitoring Sensor Networks. In *Journal of Infrastructure Systems*, Band 14(1):Seiten 64–79, 2008. URL http://dx.doi.org/10.1061/(ASCE)1076-0342(2008)14:1(64).

[35] Jason L. Hill und David E. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. In *IEEE Micro*, Band 22(6):Seiten 12–24, 2002. ISSN 0272-1732. URL http://dx.doi.org/10.1109/MM.2002.1134340.

[36] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler und Kristofer Pister. System Architecture Directions for Networked Sensors. In *SIGARCH Comput. Archit. News*, Band 28(5):Seiten 93–104, 2000. ISSN 0163-5964. URL http://dx.doi.org/10.1145/378995.379006.

[37] Carsten Buschmann und Dennis Pfisterer. iSense: A Modular Hardware and Software Platform for Wireless Sensor Networks. Technischer Bericht, 6. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, 2007. URL http://ds.informatik.rwth-aachen.de/events/fgsn07/fgsn07proc.pdf.

[38] Coalesenses GmbH. Core Module CM10C, CM10S. 2014. http://coalesenses.de/download/data_sheets/DS_CM10X_1v1.pdf [Zugriff am 21. März 2014].

[39] Coalesenses GmbH. iSense Core Module Datasheet CM20I, CM20U, CM20HP. 2014. http://coalesenses.de/download/data_sheets/DS_CM20X_1v0.pdf [Zugriff am 21. März 2014].

[40] Coalesenses GmbH. iSense Core Module 3 Datasheet CM30I, CM30U, CM30HP. 2014. http://coalesenses.de/download/data_sheets/DS_CM30X_1v3.pdf [Zugriff am 21. März 2014].

[41] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk und John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, Seiten 88–97. ACM, New York, NY, USA, 2002. ISBN 1-58113-589-0. URL http://dx.doi.org/10.1145/570738.570751.

[42] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson und David Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, Seiten 214–226. ACM, New York, NY, USA, 2004. ISBN 1-58113-879-2. URL http://dx.doi.org/10.1145/1031495.1031521.

[43] Edoardo S. Biagioni und Kent W. Bridges. The Application of Remote Sensor Technology To Assist the Recovery of Rare and Endangered Species. In *International Journal of High Performance Computing Applications*, Band 16(3):Seiten 315–324, 2002. URL http://dx.doi.org/10.1177/109434200201600310 01.

[44] David E. Culler. Toward the Sensor Network Macroscope. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '05, Seite 1. ACM, New York, NY, USA, 2005. ISBN 1-59593-004-3. URL http://dx.doi.org/10.1145/1062689.1062690.

[45] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay und Wei Hong. A Macroscope in the Redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, Seiten 51–63. ACM, New York, NY, USA, 2005. ISBN 1-59593-054-X. URL http://dx.doi.org/10.1145/1098918.1098925.

[46] Kirk Martinez, Royan Ong und Jane Hart. Glacsweb: a sensor network for hostile environments. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, IEEE SECON 2004, Seiten 81–87. 2004. URL http://dx.doi.org/10.1109/SAHCN.2004.1381905.

[47] Christian Skalka und Jeff Frolik. Snowcloud: A Complete System for Snow Hydrology Research. In *ACM Workshop on Real-World Wireless Sensor Networks*, RealWSN. 2013. URL http://dx.doi.org/10.1007/978-3-319-03071-5_1.

[48] David M. Doolin und Nicholas Sitar. Wireless sensors for wildfire monitoring. In *Smart Structures and Materials*, Band 5765, Seiten 477–484. 2005. URL http://dx.doi.org/10.1117/12.605655.

[49] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh und Daniel Rubenstein. Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In

SIGARCH Comput. Archit. News, Band 30(5):Seiten 96–107, 2002. ISSN 0163-5964. URL http://dx.doi.org/10.1145/635506.605408.

[50] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon und Margaret Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, Seiten 227–238. ACM, New York, NY, USA, 2004. ISBN 1-58113-879-2. URL http://dx.doi.org/10.1145/1031495.1031522.

[51] Philipp Sommer, Branislav Kusy, Adam Mckeown und Raja Jurdak. The Big Night Out: Experiences from Tracking Flying Foxes with Delay-Tolerant Wireless Networking. In *Proceedings of the Fifth Workshop on Real-World Wireless Sensor Networks*, RealWSN. 2013. URL http://dx.doi.org/10.1007/978-3-319-03071-5_2.

[52] Koen Langendoen, Aline Baggio und Otto Visser. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *20th International Parallel and Distributed Processing Symposium*, IPDPS 2006, Seite 8. 2006. URL http://dx.doi.org/10.1109/IPDPS.2006.1639412.

[53] Richard Beckwith, Dan Teibel und Pat Bowen. Unwired Wine: Sensor Networks in Vineyards. In *Proceedings of IEEE Sensors, 2004*, Seiten 561–564. 2004. URL http://dx.doi.org/10.1109/ICSENS.2004.1426227.

[54] Jenna Burrell, Tim Brooke und Richard Beckwith. Vineyard Computing: Sensor Networks in Agricultural Production. In *IEEE Pervasive Computing*, Band 3(1):Seiten 38–45, 2004. ISSN 1536-1268. URL http://dx.doi.org/10.1109/MPRV.2004.1269130.

[55] Zack Butler, Peter Corke, Ron Peterson und Daniela Rus. Networked Cows: Virtual Fences for Controlling Cows. In *In WAMES 2004*. 2004. URL http://dx.doi.org/10.1109/ROBOT.2004.1302415.

[56] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman und Mark Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, Seiten 64–75. ACM, New York, NY, USA, 2005. ISBN 1-59593-054-X. URL http://dx.doi.org/10.1145/1098918.1098926.

[57] Dong Li, Wei Liu, Ze Zhao und Li Cui. Demonstration of a WSN Application in Relic Protection and an Optimized System Deployment Tool. In *International Conference on Information Processing in Sensor Networks*, Seiten 541–542. 2008. URL http://dx.doi.org/10.1109/IPSN.2008.18.

[58] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser und Martin Turon. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *6th International Sym-*

posium on Information Processing in Sensor Networks, IPSN 2007, Seiten 254–263. 2007. URL http://dx.doi.org/10.1109/IPSN.2007.4379685.

[59] Richard Mietz, Carsten Buschmann, Dennis Boldt, Kay Römer und Stefan Fischer. Poster Abstract: iBAST–Instantaneous Bridge Assessment Based on Sensor Network Technology. In *Real-World Wireless Sensor Networks* (herausgegeben von Koen Langendoen, Wen Hu, Federico Ferrari, Marco Zimmerling und Luca Mottola), Band 281 von *Lecture Notes in Electrical Engineering*, Seiten 85–89. Springer International Publishing, 2014. ISBN 978-3-319-03070-8. URL http://dx.doi.org/10.1007/978-3-319-03071-5_10.

[60] Matteo Ceriotti, Luca Mottola, Gian Pietro Picco, Amy L. Murphy, Stefan Guna, Michele Corra, Matteo Pozzi, Daniele Zonta und Paolo Zanon. Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, Seiten 277–288. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-1-4244-5108-1. URL http://dx.doi.org/10.1145/1602165.1602191.

[61] Octav Chipara, Chenyang Lu, Thomas C. Bailey und Gruia-Catalin Roman. Reliable Clinical Monitoring Using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, Seiten 155–168. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0344-6. URL http://dx.doi.org/10.1145/1869983.1869999.

[62] Dennis Pfisterer, Martin Lipphardt, Carsten Buschmann, Horst Hellbrueck, Stefan Fischer und Jan Hendrik Sauselin. MarathonNet: Adding Value to Large Scale Sport Events - a Connectivity Analysis. In *Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*, InterSense '06. ACM, New York, NY, USA, 2006. ISBN 1-59593-427-8. URL http://dx.doi.org/10.1145/1142680.1142696.

[63] Carlo Alberto Boano, Matteo Lasagni und Kay Römer. Non-Invasive Measurement of Core Body Temperature in Marathon Runners. In *2013 IEEE International Conference on Body Sensor Networks*, BSN, Seiten 1–6. 2013. ISSN 2325-1425. URL http://dx.doi.org/10.1109/BSN.2013.6575484.

[64] Amol Deshpande, Carlos Guestrin und Samuel R. Madden. Resource-Aware Wireless Sensor-Actuator Networks. In *IEEE Data Engineering*, Band 28:Seite 2005, 2005.

[65] Felix Jonathan Oppermann, Carlo Alberto Boano und Kay Römer. A Decade of Wireless Sensing Applications: Survey and Taxonomy. In *The Art of Wireless Sensor Networks* (herausgegeben von Habib M. Ammari), Signals and Communication Technology. Springer, 2014. URL http://dx.doi.org/10.1007/978-3-642-40009-4_2.

[66] Nils Glombitza, Richard Mietz, Kay Römer, Stefan Fischer und Dennis Pfisterer. Self-Description and Protocol Conversion for a Web of Things. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, SUTC, Seiten 229–236. IEEE, 2010. URL http://dx.doi.org/10.1109/SUTC.2010.13.

[67] Daniel Bimschas, Horst Hellbrück, Richard Mietz, Dennis Pfisterer, Kay Römer und Torsten Teubler. Middleware for Smart Gateways Connecting Sensornets to the Internet. In *Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, Seiten 8–14. ACM, 2010. URL http://dx.doi.org/10.1145/1890784.1890787.

[68] Kevin Ashton. That 'Internet of Things' Thing. In *RFiD Journal*, Band 22:Seiten 97–114, 2009.

[69] Luigi Atzori, Antonio Iera und Giacomo Morabito. The Internet of Things: A survey. In *Computer Networks*, Band 54(15):Seiten 2787–2805, 2010. ISSN 1389-1286. URL http://dx.doi.org/10.1016/j.comnet.2010.05.010.

[70] Ioan Toma, Elena Simperl und Graham Hench. A joint roadmap for Semantic technologies and the Internet of Things. In *Proceedings of the Third STI Roadmapping Workshop*, Band 1. Crete, Greece, 2009.

[71] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, 2000.

[72] Zach Shelby, Klaus Hartke und Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), 2014. URL http://www.ietf.org/rfc/rfc7252.txt.

[73] Klaus Hartke. Observing Resources in CoAP. Internet-Draft draft-ietf-core-observe-14, Internet Engineering Task Force, 2014. URL https://www.ietf.org/id/draft-ietf-core-observe-14.txt. Work in progress.

[74] Shi. Li, Jeroen Hoebeke, Floris Van den Abeele und Antonio J. Jara. Conditional observe in CoAP. Internet-Draft draft-li-core-conditional-observe-04, Internet Engineering Task Force, 2013. URL https://tools.ietf.org/id/draft-li-core-conditional-observe-04.txt. Work in progress.

[75] Ian Jacobs, David Raggett und Arnaud Le Hors. HTML 4.01 Specification. W3C recommendation, W3C, 1999. http://www.w3.org/TR/1999/REC-html401-19991224/.

[76] Steven Pemberton. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). W3C recommendation, W3C, 2002. http://www.w3.org/TR/2002/REC-xhtml1-20020801/.

[77] Tim Bray, Jean Paoli, Eve Maler, François Yergeau und C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation, W3C, 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[78] Eric Miller und Frank Manola. RDF Primer. W3C recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[79] Guus Schreiber und Yves Raimond. RDF 1.1 Primer. W3C recommendation, W3C, 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/.

[80] Priscilla Walmsley und David C. Fallside. XML Schema Part 0: Primer Second Edition. W3C recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[81] Ashok Malhotra und Paul V. Biron. XML Schema Part 2: Datatypes Second Edition. W3C recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[82] Gavin Carothers und Eric Prud'hommeaux. Turtle. W3C candidate recommendation, W3C, 2013. http://www.w3.org/TR/2013/CR-turtle-20130219/.

[83] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[84] Gavin Carothers. RDF 1.1 N-Triples. Candidate recommendation, W3C, 2013. http://www.w3.org/TR/2013/CR-n-triples-20131105/.

[85] Eric Prud'hommeaux und Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation, W3C, 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[86] Andy Seaborne und Steve Harris. SPARQL 1.1 Query. W3C working draft, W3C, 2009. http://www.w3.org/TR/2009/WD-sparql11-query-20091022/.

[87] Paul Gearon und Simon Schenk. SPARQL 1.1 Update. W3C working draft, W3C, 2009. http://www.w3.org/TR/2009/WD-sparql11-update-20091022/.

[88] Open Geospatial Consortium et al. OGC GeoSPARQL-A Geographic Query Language for RDF Data. In *OGC Candidate Implementation Standard*, 2012. URL https://portal.opengeospatial.org/files/?artifact_id=47664.

[89] Robert Battle und Dave Kolas. Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL. In *Semantic Web*, Band 3(4):Seiten 355–370, 2012. URL http://dx.doi.org/10.3233/SW-2012-0065.

[90] David A. Randell, Zhan Cui und Anthony G. Cohn. A Spatial Logic based on Regions and Connection. In *Proceedings 3rd International Conference on Knowledge Representation and Reasoning*. 1992.

[91] Max J. Egenhofer. A Formal Definition of Binary topological Relationships. In *Foundations of Data Organization and Algorithms* (herausgegeben von Witold Litwin und Hans-Jörg Schek), Band 367 von *Lecture Notes in Computer Science*, Seiten 457–472. Springer Berlin Heidelberg, 1989. ISBN 978-3-540-51295-0. URL `http://dx.doi.org/10.1007/3-540-51295-0_148`.

[92] OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture. 2006. URL `http://www.opengeospatial.org/standards/sfa/`.

[93] Martin Maguire. Methods to support human-centered design. In *International Journal of Human-Computer Studies*, Band 55(4):Seiten 587–634, 2001. URL `http://dx.doi.org/10.1006/ijhc.2001.0503`.

[94] Shravan Gaonkar, Jack Li, Romit Roy Choudhury, Landon Cox und Al Schmidt. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, Seiten 174–186. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-139-2. URL `http://dx.doi.org/10.1145/1378600.1378620`.

[95] Dominique Guinard, Iulia Ion und Simon Mayer. In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services* (herausgegeben von Alessandro Puiatti und Tao Gu), Band 104 von *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Seiten 326–337. Springer Berlin Heidelberg, 2012. URL `http://dx.doi.org/10.1007/978-3-642-30973-1_32`.

[96] Lan S. Bai, Robert P. Dick und Peter A. Dinda. Archetype-based Design: Sensor Network Programming for Application Experts, Not Just Programming Experts. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, Seiten 85–96. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-1-4244-5108-1. URL `http://dl.acm.org/citation.cfm?id=1602165.1602175`.

[97] J. Scott Miller, Peter A. Dinda und Robert P. Dick. Evaluating a BASIC Approach to Sensor Network Node Programming. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, Seiten 155–168. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-519-2. URL `http://dx.doi.org/10.1145/1644038.1644054`.

[98] Joakim Eriksson, Niclas Finne, Luca Mottola, Thiemo Voigt, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Stefano Tranquillini, Bengt-Ove Holländer, Felix Jonathan Opper-

mann, Kay Römer, Sebastian Doeweling, Nina Oertel, Patrik Spieß Florian Probst und Patricio Moreno Montero. D-3.6 & D-5.4 – Final application implementations and evaluation of system & Final evaluation of the programming model. 2013. Deliverable D-3.6 & D-5.4 URL http://www.project-makesense.eu/images/makeSense/docs/d-5.4.pdf [Zugriff am 21. März 2014].

[99] Mirko Presser, Herausgeber. *Inspiring the Internet of Things!* Alexandra Institute, 2011. http://www.alexandra.dk/uk/services/publications/documents/iot_comic_book.pdf.

[100] Danny Sullivan. Google Still World's Most Popular Search Engine By Far, But Share Of Unique Searchers Dips Slightly. http://searchengineland.com/google-worlds-most-popular-search-engine-148089, 2013. [Zugriff am 21. März 2014].

[101] Samuel D. Gosling, Simine Vazire, Sanjay Srivastava und Oliver P. John. Should We Trust Web-Based Studies? A Comparative Analysis of Six Preconceptions About Internet Questionnaires. In *The American psychologist*, Band 59(2):Seiten 93–104, 2004. ISSN 0003-066X. URL http://dx.doi.org/10.1037/0003-066x.59.2.93.

[102] Felicity Bishop, Graham Lewis, Scott Harris, Naomi McKay, Philippa Prentice, Haymo Thiel und George Lewith. A within-subjects trial to test the equivalence of online and paper outcome measures: the Roland Morris Disability Questionnaire. In *BMC Musculoskeletal Disorders*, Band 11(1):Seite 113, 2010. ISSN 1471-2474. URL http://dx.doi.org/10.1186/1471-2474-11-113.

[103] Martyn Denscombe. Web-Based Questionnaires and the Mode Effect: An Evaluation Based on Completion Rates and Data Contents of Near-Identical Questionnaires Delivered in Different Modes. In *Social Science Computer Review*, Band 24(2):Seiten 246–254, 2006. URL http://dx.doi.org/10.1177/0894439305284522.

[104] Sundar Pichai. Sharing what's up our sleeve: Android coming to wearables. http://googleblog.blogspot.de/2014/03/sharing-whats-up-our-sleeve-android.html, 2014. [Zugriff am 21. März 2014].

[105] Google Inc. Google Glass. http://www.google.com/glass/start/, 2014. [Zugriff am 21. März 2014].

[106] Roy Williams. You've got better things to do than wait in traffic. http://googleblog.blogspot.de/2011/03/youve-got-better-things-to-do-than-wait.html, 2013. [Zugriff am 21. März 2014].

[107] Bernard J. Jansen und Amanda Spink. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. In *Information Processing & Management*, Band 42(1):Seiten 248–263, 2006. ISSN 0306-4573. URL http://dx.doi.org/10.1016/j.ipm.2004.10.007. Formal Methods for Information Retrieval.

[108] MYI Idris, EM Tamil, Z Razak, NM Noor und LW Kin. Smart Parking System using Image Processing Techniques in Wireless Sensor Network Environment. In *Information Technology Journal*, Band 8(2):Seiten 114–127, 2009. URL http://dx.doi.org/10.3923/itj.2009.114.127.

[109] Qi Wu, Chingchun Huang, Shih yu Wang, Wei-Chen Chiu und Tsuhan Chen. Robust Parking Space Detection Considering Inter-Space Correlation. In *2007 IEEE International Conference on Multimedia and Expo*, Seiten 659–662. 2007. URL http://dx.doi.org/10.1109/ICME.2007.4284736.

[110] Rahayu Yusnita, Norbaya Fariza und Basharuddin Norazwinawati. Intelligent Parking Space Detection System Based on Image Processing. In *International Journal of Innovation, Management and Technology*, Band 3(3), 2012. URL http://dx.doi.org/10.7763/IJIMT.2012.V3.228.

[111] Katy Blumer, Hala R. Halaseh, MianUmair Ahsan, Haiwei Dong und Nikolaos Mavridis. Cost-Effective Single-Camera Multi-Car Parking Monitoring and Vacancy Detection towards Real-World Parking Statistics and Real-Time Reporting. In *Neural Information Processing* (herausgegeben von Tingwen Huang, Zhigang Zeng, Chuandong Li und ChiSing Leung), Band 7667 von *Lecture Notes in Computer Science*, Seiten 506–515. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-34499-2. URL http://dx.doi.org/10.1007/978-3-642-34500-5_60.

[112] Nicholas True. Vacant Parking Space Detection in Static Images. In *University of California, San Diego*, 2007.

[113] Suhas Mathur, Tong Jin, Nikhil Kasturirangan, Janani Chandrasekaran, Wenzhi Xue, Marco Gruteser und Wade Trappe. ParkNet: Drive-by Sensing of Road-Side Parking Statistics. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, Seiten 123–136. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-985-5. URL http://dx.doi.org/10.1145/1814433.1814448.

[114] Ravi Bagree, Vishwas Raj Jain, Aman Kumar und Prabhat Ranjan. TigerCENSE: Wireless Image Sensor Network to Monitor Tiger Movement. In *Proceedings of the 4th international conference on Real-world wireless sensor networks*, REALWSN '10, Seiten 13–24. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 3-642-17519-8, 978-3-642-17519-0. URL http://dx.doi.org/10.1007/978-3-642-17520-6_2.

[115] Grigore Stamatescu, Kay Römer, Ralf Ludwig, Saleh M. Ibrahim und Valentin Sgarciu. Work in Progress: MiceNet: Monitoring Behaviour of Laboratory Mice with Sensor Networks. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops*, DCOSS, Seiten 1–3. 2011. URL http://dx.doi.org/10.1109/DCOSS.2011.5982197.

[116] Philipp Bolliger, Moritz Köhler und Kay Römer. Facet: Towards a Smart Camera Network of Mobile Phones. In *Proceedings of the 1st international conference on Autonomic computing and communication sys-*

tems, Autonomics '07, Seiten 17:1–17:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2007. ISBN 978-963-9799-09-7. URL http://dl.acm.org/citation.cfm?id=1365562.1365585.

[117] Lukas Ruge, Bashar Altakrouri und Andreas Schrader. SoundOfTheCity - Continuous Noise Monitoring for a Healthy City. In *5th International Workshop on Smart Environments and Ambient Intelligence (SENAmI 2013) at IEEE International Conference on Pervasive Computing and Communication (PerCom 2013)*. San Diego, California, USA, 2013. URL http://dx.doi.org/10.1109/PerComW.2013.6529577.

[118] Martin Azizyan, Ionut Constandache und Romit Roy Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, Seiten 261–272. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-702-8. URL http://dx.doi.org/10.1145/1614320.1614350.

[119] Rana El Kaliouby und Peter Robinson. Real-Time Inference of Complex Mental States from Facial Expressions and Head Gestures. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, Band 10 von *CVPRW '04*, Seiten 154–. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2158-4. URL http://dx.doi.org/10.1109/CVPR.2004.153.

[120] Curtis Padgett, Garrison W. Cottrell und Ralph Adolphs. Categorical Perception in Facial Emotion Classification. In *Proceedings of the Cognitive Science Conference*, Band 18, Seiten 249–253. 1996.

[121] Maja Pantic und Ioannis Patras. Dynamics of Facial Expression: Recognition of Facial Actions and Their Temporal Segments From Face Profile Image Sequences. In *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Band 36(2):Seiten 433–449, 2006. ISSN 1083-4419. URL http://dx.doi.org/10.1109/TSMCB.2005.859075.

[122] Rosalind Wright Picard, Elias Vyzas und Jennifer Healey. Toward Machine Emotional Intelligence: Analysis of Affective Physiological State. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Band 23(10):Seiten 1175–1191, 2001. ISSN 0162-8828. URL http://dx.doi.org/10.1109/34.954607.

[123] Valery A. Petrushin. Emotion Recognition in Speech Signal: Experimental Study, Development and Applications. In *Proceedings ICSLP 2000*, Seiten 222–225. 2000.

[124] Tin Lay Nwe, Foo Say Wei und Liyanage Chandratilake De Silva. Speech based emotion classification. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology*, Band 1 von *TENCON*

2001, Seiten 297–301. 2001. URL http://dx.doi.org/10.1109/TENCON.2001.949600.

[125] Zhihong Zeng, Yuxiao Hu, Yun Fu, Thomas S. Huang, Glenn I. Roisman und Zhen Wen. Audio-visual Emotion Recognition in Adult Attachment Interview. In *Proceedings of the 8th International Conference on Multimodal Interfaces*, ICMI '06, Seiten 139–145. ACM, New York, NY, USA, 2006. ISBN 1-59593-541-X. URL http://dx.doi.org/10.1145/1180995.1181028.

[126] Yasunari Yoshitomi, Sung-Ill Kim, Takako Kawano und Tetsuro Kilazoe. Effect of Sensor Fusion for Recognition of Emotional States Using Voice, Face Image and Thermal Image of Face. In *9th IEEE International Workshop on Robot and Human Interactive Communication, 2000. RO-MAN 2000. Proceedings.*, Seiten 178–183. 2000. URL http://dx.doi.org/10.1109/ROMAN.2000.892491.

[127] Paul Ekman und Wallace V. Friesen. Facial action coding system: A technique for the measurement of facial movement. Palo Alto. 1978.

[128] Beat Fasel und Juergen Luettin. Automatic facial expression analysis: a survey. In *Pattern Recognition*, Band 36(1):Seiten 259–275, 2003. URL http://dx.doi.org/10.1016/S0031-3203(02)00052-3.

[129] Maja Pantic und Léon J.M. Rothkrantz. Toward an Affect-Sensitive Multimodal Human–Computer Interaction. In *Proceedings of the IEEE*, Band 91(9):Seiten 1370–1390, 2003. URL http://dx.doi.org/10.1109/JPROC.2003.817122.

[130] Osama Masoud und Nikos Papanikolopoulos. A Method For Human Action Recognition. In *Image and Vision Computing*, Band 21(8):Seiten 729–743, 2003. ISSN 0262-8856. URL http://dx.doi.org/http://dx.doi.org/10.1016/S0262-8856(03)00068-4.

[131] Vinay D. Shet, David Harwood und Larry S. Davis. VidMAP: Video Monitoring of Activity with Prolog. In *IEEE Conference on Advanced Video and Signal Based Surveillance, 2005. AVSS 2005.*, Seiten 224–229. 2005. URL http://dx.doi.org/10.1109/AVSS.2005.1577271.

[132] Nishkam Ravi, D. Nikhil, Preetham Mysore und Michael L. Littman. Activity Recognition from Accelerometer Data. In *In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI*, IAAI '05, Seiten 1541–1546. AAAI Press, 2005. ISBN 1-57735-236-x.

[133] Jennifer R. Kwapisz, Gary M. Weiss und Samuel A. Moore. Activity Recognition Using Cell Phone Accelerometers. In *SIGKDD Explor. Newsl.*, Band 12(2):Seiten 74–82, 2011. ISSN 1931-0145. URL http://dx.doi.org/10.1145/1964897.1964918.

[134] Martin Berchtold, Matthias Budde, Dawud Gordon, Hedda Rahel Schmidtke und Michael Beigl. ActiServ: Activity Recognition Service for mobile

phones. In *2010 International Symposium on Wearable Computers*, ISWC, Seiten 1–8. 2010. ISSN 1550-4816. URL http://dx.doi.org/10.1109/ISWC.2010.5665868.

[135] Norbert Győrbíró, Ákos Fábián und Gergely Hományi. An Activity Recognition System For Mobile Phones. In *Mobile Networks and Applications*, Band 14(1):Seiten 82–91, 2009. ISSN 1383-469X. URL http://dx.doi.org/10.1007/s11036-008-0112-y.

[136] Juha Pärkkä, Miikka Ermes, Panu Korpipää, Jani Mäntyjärvi, Johannes Peltola und Ilkka Korhonen. Activity Classification Using Realistic Data from Wearable Sensors. In *IEEE Transactions on Information Technology in Biomedicine*, Band 10(1):Seiten 119–128, 2006. ISSN 1089-7771. URL http://dx.doi.org/10.1109/TITB.2005.856863.

[137] Amarnag Subramanya, Alvin Raj, Jeff Bilmes und Dieter Fox. Recognizing Activities and Spatial Context Using Wearable Sensors. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence*, UAI '06. 2006.

[138] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng und Andrew T. Campbell. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, Seiten 337–350. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-990-6. URL http://dx.doi.org/10.1145/1460412.1460445.

[139] Athanasios Bamis, Dimitrios Lymberopoulos, Thiago Teixeira und Andreas Savvides. The BehaviorScope framework for enabling ambient assisted living. In *Personal Ubiquitous Comput.*, Band 14:Seiten 473–487, 2010. ISSN 1617-4909. URL http://dx.doi.org/10.1007/s00779-010-0282-z.

[140] Ronald Poppe. A survey on vision-based human action recognition. In *Image and Vision Computing*, Band 28(6):Seiten 976–990, 2010. ISSN 0262-8856. URL http://dx.doi.org/10.1016/j.imavis.2009.11.014.

[141] Akin Avci, Stephan Bosch, Mihai Marin-Perianu, Raluca Marin-Perianu und Paul Havinga. Activity Recognition Using Inertial Sensing for Healthcare, Wellbeing and Sports Applications: A Survey. In *Proceedings of the 23th International Conference on Architecture of Computing Systems, ARCS 2010*, Seiten 167–176. VDE Verlag, Berlin, 2010. URL http://doc.utwente.nl/70138/.

[142] D. Carlson und A. Schrader. Dynamix: An Open Plug-and-Play Context Framework for Android. In *2012 3rd International Conference on the Internet of Things (IOT)*, Seiten 151–158. 2012. URL http://dx.doi.org/10.1109/IOT.2012.6402317.

[143] Thorsten Caus, Stefan Christmann und Svenja Hagenhoff. Hydra – An Application Framework for the Development of Context-Aware Mobile Services. In *Business Information Systems* (herausgegeben von Witold Abramowicz und Dieter Fensel), Band 7 von *Lecture Notes in Business Information Processing*, Seiten 471–481. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79395-3. URL http://dx.doi.org/10.1007/978-3-540-79396-0_41.

[144] Anuja Meetoo-Appavoo. SmartSense: A Novel Smart and Intelligent Context-Aware Framework. In *IJCSNS*, Band 11(8):Seiten 214–221, 2011.

[145] David Bannach, Paul Lukowicz und Oliver Amft. Rapid Prototyping of Activity Recognition Applications. In *IEEE Pervasive Computing*, Band 7(2):Seiten 22–31, 2008. ISSN 1536-1268. URL http://dx.doi.org/10.1109/MPRV.2008.36.

[146] Stefan Pietschmann, Annett Mitschick, Ronny Winkler und Klaus Meissner. CroCo: Ontology-Based, Cross-Application Context Management. In *Third International Workshop on Semantic Media Adaptation and Personalization, 2008*, SMAP '08, Seiten 88–93. 2008. URL http://dx.doi.org/10.1109/SMAP.2008.10.

[147] Charith Perera, Arkady Zaslavsky, Peter Christen und Dimitrios Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. In *Communications Surveys Tutorials, IEEE*, Band 16(1):Seiten 414–454, 2014. ISSN 1553-877X. URL http://dx.doi.org/10.1109/SURV.2013.042313.00197.

[148] David Chu, Amol Deshpande, Joseph M. Hellerstein und Wei Hong. Approximate Data Collection in Sensor Networks using Probabilistic Models. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, Seite 48. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2570-9. URL http://dx.doi.org/10.1109/ICDE.2006.21.

[149] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein und Wei Hong. Model-Driven Data Acquisition in Sensor Networks. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, Seiten 588–599. VLDB Endowment, 2004. ISBN 0-12-088469-0. URL http://dl.acm.org/citation.cfm?id=1316689.1316741. http://www.vldb.org/conf/2004/RS16P1.PDF.

[150] Buğra Gedik, Ling Liu und Philip S. Yu. ASAP: An Adaptive Sampling Approach to Data Collection in Sensor Networks. In *IEEE Trans. Parallel Distrib. Syst.*, Band 18:Seiten 1766–1783, 2007. ISSN 1045-9219. URL http://dx.doi.org/10.1109/TPDS.2007.1110.

[151] Qi Han, Sharad Mehrotra und Nalini Venkatasubramanian. Energy Efficient Data Collection in Distributed Sensor Environments. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, ICDCS '04, Seiten 590–597. IEEE Computer Society, Washington, DC,

USA, 2004. ISBN 0-7695-2086-3. URL http://dx.doi.org/10.1109/ICDCS.2004.1281626.

[152] Daniela Tulone und Samuel Madden. PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In *Wireless Sensor Networks* (herausgegeben von Kay Römer, Holger Karl und Friedemann Mattern), Band 3868 von *Lecture Notes in Computer Science*, Kapitel 5, Seiten 21–37. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2006. URL http://dx.doi.org/10.1007/11669463_5.

[153] Daniela Tulone und Samuel Madden. An Energy-efficient Querying Framework In Sensor Networks For Detecting Node Similarities. In *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, MSWiM '06, Seiten 191–300. ACM, New York, NY, USA, 2006. ISBN 1-59593-477-4. URL http://dx.doi.org/10.1145/1164717.1164768.

[154] Rafael C. Gonzalez und Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 Auflage, 1992. ISBN 0201508036.

[155] Klaus Hildebrandt und Konrad Polthier. Anisotropic Filtering of Non-Linear Surface Features. In *Computer Graphics Forum*, Band 23:Seiten 391–400, 2004. URL http://dx.doi.org/10.1111/j.1467-8659.2004.00770.x.

[156] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart Ter Haar Romeny und John B. Zimmerman. Adaptive Histogram Equalization and Its Variations. In *Comput. Vision Graph. Image Process.*, Band 39(3):Seiten 355–368, 1987. ISSN 0734-189X. URL http://dx.doi.org/10.1016/S0734-189X(87)80186-X.

[157] Ali M. Reza. Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement. In *J. VLSI Signal Process. Syst.*, Band 38(1):Seiten 35–44, 2004. ISSN 0922-5773. URL http://dx.doi.org/10.1023/B:VLSI.0000028532.53893.82.

[158] Paul Viola und Michael Jones. Robust Real-Time Face Detection. In *International Journal of Computer Vision*, Band 57:Seiten 137–154, 2004. URL http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb.

[159] Chris Stauffer und William Eric Leifur Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, Band 2, Seite 252. 1999. ISSN 1063-6919. URL http://dx.doi.org/10.1109/CVPR.1999.784637.

[160] George Stockman und Linda G. Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1. Auflage, 2001. ISBN 0130307963.

[161] Tobias Baumgartner, Ioannis Chatzigiannakis, Sándor P. Fekete, Christos Koninis, Alexander Kröller und Apostolos Pyrgelis. Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks. In *CoRR*, Band abs/1101.3067, 2011. URL http://dx.doi.org/10.1007/978-3-642-11917-0_11.

[162] Richard Mietz und Kay Römer. Exploiting correlations for efficient content-based sensor search. In *2011 IEEE Sensors*, Seiten 187–190. IEEE, 2011. URL http://dx.doi.org/10.1109/ICSENS.2011.6127082.

[163] Judea Pearl. Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning. In *Proceedings of the Cognitive Science Society*, Seiten 329–334. 1985.

[164] Changhe Yuan und Marek J. Druzdzel. An Importance Sampling Algorithm Based on Evidence Pre-propagation. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI '03, Seiten 624–631. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. ISBN 0-127-05664-5. URL http://dl.acm.org/citation.cfm?id=2100584.2100660.

[165] Michael Compton, Payam Barnaghi, Luis Bermudez, Raul Garcia-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth und Kerry Taylor. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. In *Web Semantics: Science, Services and Agents on the World Wide Web*, Band 17(0), 2012. ISSN 1570-8268. URL http://dx.doi.org/10.1016/j.websem.2012.05.003.

[166] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth et al. SPITFIRE: Toward a Semantic Web of Things. In *IEEE Communications Magazine*, Band 49(11):Seiten 40–48, 2011. URL http://dx.doi.org/10.1109/MCOM.2011.6069708.

[167] Myriam Leggieri, Alexandre Passant und Manfred Hauswirth. A Contextualised Cognitive Perspective for Linked Sensor Data. In *3rd Intl. Workshop on Semantic Sensor Network*, 2010. http://ceur-ws.org/Vol-668/paper11.pdf.

[168] Richard Mietz, Sven Groppe, Kay Römer und Dennis Pfisterer. Semantic Models for Scalable Search in the Internet of Things. In *Journal of Sensor and Actuator Networks*, Band 2(2):Seiten 172–195, 2013. URL http://dx.doi.org/10.3390/jsan2020172.

[169] Feng Pan und Jerry R. Hobbs. Time Ontology in OWL. W3C working draft, W3C, 2006. http://www.w3.org/TR/2006/WD-owl-time-20060927/.

[170] Moz. 2011 Search Engine Ranking Factors - Moz. http://moz.com/article/search-ranking-factors/, 2013. [Zugriff am 21. März 2014].

[171] Kurt Rohloff, Mike Dean, Ian Emmons, Dorene Ryder und John Sumner. An Evaluation of Triple-Store Technologies for Large Data Stores. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops* (herausgegeben von Robert Meersman, Zahir Tari und Pilar Herrero), Band 4806 von *Lecture Notes in Computer Science*, Seiten 1105–1114. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-76889-0. URL http://dx.doi.org/10.1007/978-3-540-76890-6_38.

[172] Imen Filali, Francesco Bongiovanni, Fabrice Huet und Françoise Baude. RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions. Rapport de recherche RR-7457, INRIA, 2010. URL http://hal.inria.fr/inria-00540314/.

[173] Min Cai und Martin Frank. RDFPeers: A Scalable Distributed RDF Repository Based on a Structured Peer-to-Peer Network. 2004. URL http://dx.doi.org/10.1145/988672.988760.

[174] Min Cai, Martin Frank, Baoshi Yan und Robert Macgregor. A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. In *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, Seite 2005. 2004. URL http://dx.doi.org/10.1016/j.websem.2004.10.003.

[175] Min Cai, Martin Frank, Jinbo Chen und Pedro Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *Journal of Grid Computing*, Band 2(1):Seiten 3–14, 2004. ISSN 1570-7873. URL http://dx.doi.org/10.1007/s10723-004-1184-y.

[176] Andy Seaborne. RDQL - A Query Language for RDF. W3C member submission, W3C, 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.

[177] Akiyoshi Matono, SaidMirza Pahlevi und Isao Kojima. RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores. In *Databases, Information Systems, and Peer-to-Peer Computing* (herausgegeben von Gianluca Moro, Sonia Bergamaschi, Sam Joseph, Jean-Henry Morin und ArisM. Ouksel), Band 4125 von *Lecture Notes in Computer Science*, Seiten 323–330. Springer, 2007. ISBN 978-3-540-71660-0. URL http://dx.doi.org/10.1007/978-3-540-71661-7_31.

[178] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp und Scott Shenker. A Scalable Content-Addressable Network. In *SIGCOMM Comput. Commun. Rev.*, Band 31(4):Seiten 161–172, 2001. ISSN 0146-4833. URL http://dx.doi.org/10.1145/964723.383072.

[179] Dominic Battré, Felix Heine, André Höing und Odej Kao. On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT Based RDF

Stores. In *Proceedings of the 2006 international conference on Databases, information systems, and peer-to-peer computing*, Seiten 343–354. Springer-Verlag, 2007. ISBN 978-3-540-71660-0. URL http://dx.doi.org/10.1007/978-3-540-71661-7_33.

[180] Liaquat Ali, Thomas Janson und Georg Lausen. 3rdf: Storing and Querying RDF Data on Top of the 3nuts Overlay Network. In *2012 23rd International Workshop on Database and Expert Systems Applications*, Seiten 257–261, 2011. ISSN 1529-4188. URL http://dx.doi.org/10.1109/DEXA.2011.1.

[181] Thomas Janson, Peter Mahlmann und Christian Schindelhauer. A Self-Stabilizing Locality-Aware Peer-to-Peer Network Combining Random Networks, Search Trees, and DHTs. In *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ICPADS '10, Seiten 123–130. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-0-7695-4307-9. URL http://dx.doi.org/10.1109/ICPADS.2010.42.

[182] Zoi Kaoudi, Manolis Koubarakis, Kostis Kyzirakos, Iris Miliaraki, Matoula Magiridou und Antonios Papadakis-Pesaresi. Atlas: Storing, Updating and Querying RDF(S) Data on Top of DHTs. In *Web Semantics: Science, Services and Agents on the World Wide Web*, Band 8(4):Seiten 271–277, 2010. ISSN 1570-8268. URL http://dx.doi.org/10.1016/j.websem.2010.07.001.

[183] Sean Rhea, Dennis Geels, Timothy Roscoe und John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, Seite 10. USENIX Association, Berkeley, CA, USA, 2004. URL http://dl.acm.org/citation.cfm?id=1247415.1247425.

[184] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth und Tim Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *The Semantic Web – ISWC 2004* (herausgegeben von SheilaA. McIlraith, Dimitris Plexousakis und Frank Harmelen), Band 3298 von *Lecture Notes in Computer Science*, Seiten 107–121. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23798-3. URL http://dx.doi.org/10.1007/978-3-540-30475-3_9.

[185] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva und Roman Schmidt. P-Grid: A Self-organizing Structured P2P System. In *SIGMOD Rec.*, Band 32(3):Seiten 29–33, 2003. ISSN 0163-5808. URL http://dx.doi.org/10.1145/945721.945729.

[186] Emanuele Valle, Andrea Turati und Alessandro Ghioni. PAGE: A Distributed Infrastructure for Fostering RDF-Based Interoperability. In *Distributed Applications and Interoperable Systems* (herausgegeben von Frank Eliassen und Alberto Montresor), Band 4025 von *Lecture Notes in Computer Science*, Seiten 347–353. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35126-9. URL http://dx.doi.org/10.1007/11773887_27.

[187] Tao Gu, Hung Keng Pung und Daqing Zhang. Information retrieval in schema-based P2P systems using one-dimensional semantic space. In *Comput. Netw.*, Band 51(16):Seiten 4543–4560, 2007. ISSN 1389-1286. URL http://dx.doi.org/10.1016/j.comnet.2007.06.019.

[188] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek und Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM Comput. Commun. Rev.*, Band 31(4):Seiten 149–160, 2001. ISSN 0146-4833. URL http://dx.doi.org/10.1145/964723.383071.

[189] Marcel Karnstedt, Kai-Uwe. Sattler, Martin Richtarsky, Jessica Muller, Manfred Hauswirth, Roman Schmidt und Renault John. UniStore: Querying a DHT-based Universal Storage. In *23rd International Conference on Data Engineering*, Seiten 1503–1504. 2007. URL http://dx.doi.org/10.1109/ICDE.2007.369054.

[190] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér und Tore Risch. EDUTELLA: a P2P networking Infrastructure Based on RDF. In *Proceedings of the 11th international conference on World Wide Web*, Seiten 604–615. ACM, New York, NY, USA, 2002. ISBN 1-58113-449-5. URL http://dx.doi.org/10.1145/511446.511525.

[191] Mario Schlosser, Michael Sintek, Stefan Decker und Wolfgang Nejdl. HyperCuP — Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In *Agents and Peer-to-Peer Computing* (herausgegeben von Gianluca Moro und Manolis Koubarakis), Band 2530 von *Lecture Notes in Computer Science*, Seiten 112–124. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40538-2. URL http://dx.doi.org/10.1007/3-540-45074-2_11.

[192] Sven Groppe. *Data Management and Query Processing in Semantic Web Databases*. Springer Verlag, Heidelberg, 2011. ISBN 978-3-642-19356-9. URL http://www.ifis.uni-luebeck.de/~groppe/SemWebDBBook/.

[193] Dan Brickley und Ramanathan Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[194] Sebastian Rudolph, Pascal Hitzler, Bijan Parsia, Markus Krötzsch und Peter Patel-Schneider. OWL 2 Web Ontology Language Primer (Second Edition). Technischer Bericht, W3C, 2012. http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[195] Harold Boley und Michael Kifer. RIF Basic Logic Dialect (Second Edition). W3C recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-rif-bld-20130205/.

[196] Matthew Perry, Prateek Jain und Amit P. Sheth. SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries. In *Geospatial Semantics and the Semantic Web*, Seiten 61–86. 2011. URL http://dx.doi.org/10.1007/978-1-4419-9446-2_3.

[197] Andrei Popescu. Geolocation API Specification. W3C recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-geolocation-API-20131024/.

[198] John Brooke. SUS-A quick and dirty usability scale. In *Usability evaluation in industry*, Band 189:Seite 194, 1996.

[199] Aaron Bangor, Philip Kortum und James Miller. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. In *Journal of usability studies*, Band 4(3):Seiten 114–123, 2009.

[200] Thomas S. Tullis und Jacqueline N. Stetson. A Comparison of Questionnaires for Assessing Website Usability. In *Proceedings of the Usability Professionals Association (UPA) 2004 Conference*, Seiten 7–11. 2004. URL http://usabilityprofessionals.org/usability_resources/conference/2004/UPA-2004-TullisStetson.pdf.