Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2323

Integration eines Public Sensing Systems in ein Pub/Sub System

Stefan Schweizer

Studiengang:	Informatik
Prüfer:	Prof. Dr. K. Rothermel
Betreuer:	Harald Weinschrott
begonnen am:	1.3.2011
beendet am:	31.8.2011
CR-Klassifikation:	H.3.3, C.2.1

Zusammenfassung

Die Verbreitung von Smartphones hat in den letzten Jahren stark zugenommen. Da diese Kleincomputer mit einer Menge von Sensoren und einer ständigen Datenverbindung ausgestattet sind, stellen sie große Sensornetzwerke dar, die neue Anwendungsszenarien erlauben.

Gemein ist diesen Anwendungen, dass sie in der Lagen sein müssen, eine hohe Anzahl an Sensordaten zu verarbeiten, und an mögliche Konsumenten weiterzuleiten.

Ein gut erforschtes Konzept, strukturierte Daten zu interessierten Konsumenten zu transportieren, sind Publish/Subscribe Systeme. In dieser Arbeit wird daher untersucht, wie Sensornetzwerke an solche Publish/Subscribe Systeme angebunden werden können.

Dazu wird ein Gateway, als Schnittstelle zwischen Sensornetzwerken und Publish/Subscribe Systemen, entwickelt und implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

Zu Beginn der 90er Jahre formulierte Mark Weiser in "The Computer for the 21st Century" [Wei02] die Vision des *ubiquitous computing*. Durch fortschreitende Miniaturisierung werden Computer in Zukunft in allen möglichen Alltagsgegenständen integriert sein. Diese Computer werden einerseits allgegenwärtig sein, werden sich jedoch so in unsere Umwelt einfügen, dass wir sie nicht mehr bewusst wahrnehmen. Schaut man sich die Entwicklung in den letzten 20 Jahren an, stellt man fest, dass diese Vision bis heute nichts an ihrer Aktualität verloren hat.

Einen besonderen Beitrag zu dieser Entwicklung leistete dabei die Weiterentwicklung von Mobiltelefonen hin zu Smartphones. Diese mobilen Kleincomputer sind heute ständige Begleiter vieler Menschen und übernehmen die Aufgabe einer Kamera oder spielen Multimedia-Inhalte ab. Durch die hohe Netzabdeckung in den westlichen Industrieländern und sinkende Preise ist es heute möglich, eine ständige Datenverbindung aufrecht zu halten.

Diese Smartphones sind außerdem mit einer Menge von Sensoren ausgestattet und bilden damit ein großes Netzwerk aus mobilen, mit einer ständigen Datenverbindung ausgestattete Sensorknoten. Diese vernetzen Sensoren in den Händen von Millionen von Menschen ermöglichen dabei ganz neu Anwendungsszenarien, die in [CEL+08] unter dem Begriff "People Centric Sensing" zusammengefasst werden.

Mit der wohl weiter steigenden Verbreitung von Smartphones ist davon auszugehen, dass diese Anwendungen in der Lage sein müssen, mit einer sehr hohen Anzahl an Sensordaten umzugehen. Es besteht daher Bedarf nach einem skalierenden System, das die Möglichkeit bietet, große Menge von Sensordaten zu verwalten und an mögliche Konsumenten weiterzuleiten.

Ein gut erforschtes Konzept, strukturierte Daten zu interessierten Konsumenten zu transportieren, sind Publish/Subscribe Systeme. In der folgenden Arbeit wird daher untersucht, wie Sensornetzwerke an solche Publish/Subscribe Systeme angebunden werden können.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden grundlegende Konzepte von Sensornetzwerken und Publish/Subscribe Systemen vorgestellt.

Kapitel 3 – Entwurf: In diesem Kapitel wird ein Protokoll zur Anbindung von Sensornetzwerken entwickelt. Es wird außerdem gezeigt, wie sich Sensorinformationen auf ein Publish/Subscribe System abbilden lassen.

Kapitel 4 – Implementierung: In diesem Kapitel wird eine Implementation des im vorherigen Kapitel entwickelten Entwurfs vorgestellt.

Kapitel 5 – Evaluation: In diesem Kapitel wird die Implementierung mit Hilfe von simulierten Sensornetzen auf ihre Leistungsfähigkeit überprüft.

Kapitel 6 – Zusammenfassung und Ausblick Dieses Kapitel liefert eine Zusammenfassung der Arbeit und gibt eine Ausblick auf mögliche Weiterentwicklungen.

2 Grundlagen

2.1 Sensornetze

Die Aufgabe von Sensornetzwerken ist die möglichst effiziente Übermittlung von Sensordaten. Dabei ist für diese Arbeit vor allem die Interaktion mit den Kosumenten der Sensordaten interessant. Dazu wird im Folgenden auf die Rahmenbedingungen dieser Sensornetze, deren charakteristischen Eigenschaften und die möglichen Interaktionsmodelle eingegangen.

2.1.1 Urban Sensing

Drahtlose Sensornetzwerke sind schon seit längerem Gegenstand der Forschung. Dabei lag der Fokus zu Beginn vor allem auf relativ statischen Sensoren zur Überwachung von Umweltveränderungen oder Industrieanlagen. Diese Sensornetzwerke wurden für einen spezifischen Zweck entwickelt und mit Hilfe von spezieller, stromsparender Hard- und Software auf einem definierten Gebiet ausgebracht.

Mit dem Aufkommen von Smartphones rückte in den letzten Jahren eine weitere Art von Sensornetzen in den Vordergrund, die unter dem Begriff *urban sensing* [ATC05] zusammengefasst werden können. Darunter werden Sensornetzwerke verstanden, bei denen der Mensch und die ihn umgebenden Informationen im Vordergrund stehen. Menschen bewegen sich mit ihren Smartphones durch ein urbanes Gebiet und sind dabei in der Lage, Sensorinformationen über ihre Umgebung zu sammeln. Die teilnehmenden Personen müssen dabei bereit sein, die knappen Ressourcen ihres Smartphones zum Sammeln von Sensordaten zur Verfügung zu stellen. So stehen beim *urban sensing* Anwendungen im Vordergrund, bei denen die beteiligten Parteien sowohl Produzenten als auch Konsumenten sind.

2.1.2 Charakteristische Eigenschaften

Von Konsumentenseite sind die besonderen Eigenschaften dieser dynamischen Sensornetze zu beachten [HMCP04].

Sensoren sind inherent verteilt und hoch dynamisch. Drahtlose Datenverbindungen sind vor allem in dicht bebauten Gebieten unzuverlässig. Dies gilt sowohl für die Positionsbestimmung per GPS als auch für die Datenübertragung per GSM oder Wifi. Dadurch bleibt oft nur ein kurzes Zeitfenster zur Messung oder Übertragung der Messdaten.

Dazu stellen Anwendungen oft Anforderungen an die Dienstgüte (Quality of Service) der gelieferten Daten. Durch die dynamische Natur der hier betrachteten Sensornetze ist es nötig, diese Anforderungen auf die realen Gegebenheiten abzubilden. Dazu ist eine Koordinierung innerhalb des Sensornetzwerkes erforderlich. In [HW10] wird mit StreamShaper ein möglicher Ansatz dazu vorgestellt.

Der limitierende Faktor von mobilen Embedded-Systeme ist die Energieversorgung. Es ist daher die Aufgabe eines Sensornetzwerkes mit dieser Resource möglichst sparsam umzugehen. Ziel muß es sein, durch Kooperation unnötige Mess- und Übertragungsvorgänge zu vermeiden.

Um den oben beschriebenen charakteristischen Eigenschaften dieser Sensornetzwerke Rechnung zu tragen, ist es nötig, von Konsumentenseite proaktiv auf das Netzwerk einzuwirken. Es müssen also nicht nur Sensordaten zu den Konsumenten gelangen, sondern die Konsumenten müssen das Sensornetzwerk auch über ihre Anforderungen und Zustände informieren. So ist es, um den Energiebedarf zu minimieren, für das Sensornetz wichtig zu wissen, ob es zu einem bestimmten Zeitpunkt überhaupt einen Konsumenten für die gemessenen Daten gibt und wenn ja, in welcher Dienstgüte die Daten erhoben werden müssen. Dieses Wissen ermöglicht dem Sensorsnetz, sich dynamisch an die Anforderungen anzupassen. Aber auch für den Konsument ist es wichtig, Zustandsinformationen über das Sensornetzwerk zu erfahren. Welche Daten können überhaupt geliefert werden? Sind Daten für eine bestimmtes Gebiet erhältlich, zu welchem Zeitpunkt oder Intervall. In welcher Dienstgüte sind die Daten abrufbar.

2.1.3 Interaktionsmodelle

In einem drahtlosen Sensornetzwerk sind unterschiedliche Akteure beteiligt [TAGH02].

Auf der einen Seite befinden sich die Sensoren. Diese messen physikalische Phänomene und leiten diese Messdaten über drahtlose Datenverbindung weiter. Auf der anderen Seite befinden sich die Konsumenten, die diese Messdaten nutzen.

Das Interaktionsmodell des Sensornetzwerkes wird einerseits durch die Anforderungen der Konsumenten als auch durch die Eigenschaften der Sensoren bestimmt. So wird man bei zufällig ausgebrachten Sensoren in einem unzugänglichen Waldgebiet, zum Messen von Umweltdaten ein Mesh-Netzwerkstruktur, wie in [SGAP00] beschrieben, wählen. Will man hingegen Verkehrsdaten in einem urbanisierten Gebiet sammeln, wird man die Sensoren eher mit einem GSM-Modul ausstatten und mit einer Client-Server Architektur die Sensoren an zentrale Knoten anbinden. Will man Konsumergeräte wie Smartphones als Sensorknoten nutzen, sind die Möglichkeiten der Kommunikation durch die Geräte vorgegeben und können von Gerät zu Gerät stark variieren. So verwenden zum Beispiel, viele Mobilfunkprovider keine öffentlichen IP Adressen. Damit ist es über das Mobilfunknetz unmöglich, Daten direkt an diese Mobiltelefone zu schicken.

Allgemein können folgende Interaktionsmodelle, wie in [ATC05] beschrieben, unterschieden werden.

Mobiles Peering

Sensoren können beim Zusammentreffen direkt miteinander interagieren. Dieses Modell ist vor allem für einen kurzen Datenaustausch und geringe Distanzen geeignet. So können zum Beispiel beim zusammentreffen zweier Personen automatisch Visitenkarten ausgetauscht werden. Dieses Modell kann auch im Straßenverkehr interessant sein, um Verkehrsinformationen zwischen Verkehrsteilnehmern auszutauschen.

Statische Sensoren ↔ Mobil Sensoren

Eine Kombination von statischen mit mobilen Sensoren kann in mehreren Fällen Sinn ergeben. So kann der statische Sensor besser an die Konsumenten angebunden sein und so als Datensenke für vorbeikommende mobile Sensoren dienen. Ein statischer Sensor kann auch in einem Gebiet ausgebracht werden, um Daten über vorbeikommende mobilen Sensoren zu sammeln.

Mobile Sensoren ↔ Gateway

Das Gateway übernimmt hier zwei Aufgaben. Erstens gibt es Steuerinformationen an die Sensornetzwerk weiter. Das Gateway leitet dazu Anfragen der Konsumenten an das Sensornetzwerk weiter. So können bestimmte Sensorinformationen beim Sensornetzwerk bestellt oder abbestellt werden. Zweitens nimmt das Gateway Sensorinformationen aus dem Sensornetzwerk entgegen und leitet diese an interessierte Konsumenten weiter.

2.2 Publish/Subscribe

Schon heute sind sehr viele Informationen im Internet abrufbar. Es ist ein leichtes den günstigsten Preis für ein bestimmtes Produkt oder eine Dienstleistung abzurufen. So kann man zum Beispiel leicht auf verschiedenen Vergleichsportalen den günstigsten Preis für eine Flugreise erfahren. Anders sieht es jedoch aus, wenn man nur einen bestimmten Betrag für eine Flugreise zur Verfügung hat. In diesem Fall wird besonders im World Wide Web bisher vor allem Polling eingesetzt. Dabei ruft der Benutzer oder ein in die Webseite eingebettetes Skript die Daten in einem kurzen Intervall immer wieder ab, bis eine bestimmte Bedingung, in diesem Beispiel ein Flugpreis unter einem definierten Betrag, erfüllt ist.

Diese Methode hat gravierende Nachteile. Es entsteht eine unnötig hohe Last sowohl im Netzwerk, als auch bei der Datenquelle. Es werden viele Anfragen generiert, obwohl sich die Information nicht geändert hat. Sollte sich die Information doch ändern, also im Beispiel der Preis der Flugreise unter den gewünschten Betrag fallen, ist die Wahrscheinlichkeit hoch, dass dieses Ereignis erst mit einer gewissen Zeitverzögerung beim Benutzer ankommt, denn dieser erfährt davon erst beim nächsten Abruf. Verringert man das Zeitintervall, um schneller von einer Zustandsänderung zu erfahren, erhöht man die Last auf Seitens des Dienstleistungsanbieters enorm. Diese Zeitverzögerung erhöht sich, wenn die Information nicht direkt vom Konsumenten abgerufen wird, sondern noch über mehrere Zwischensysteme geleitet werden muß. Es besteht also ein Trade-off zwischen Genauigkeit der Anfragen und der Skalierbarkeit des Systems. Da Anfragen von Benutzerseite initiiert werden, muss die Adresse des Diensteanbieters bekannt sein. Dadurch entsteht eine enge Kopplung zwischen Benutzer und Anbieter. Diese ist jedoch gar nicht immer gewünscht. So sollte ein günstiger Flug am Besten anbieterübergreifend gefunden werden.

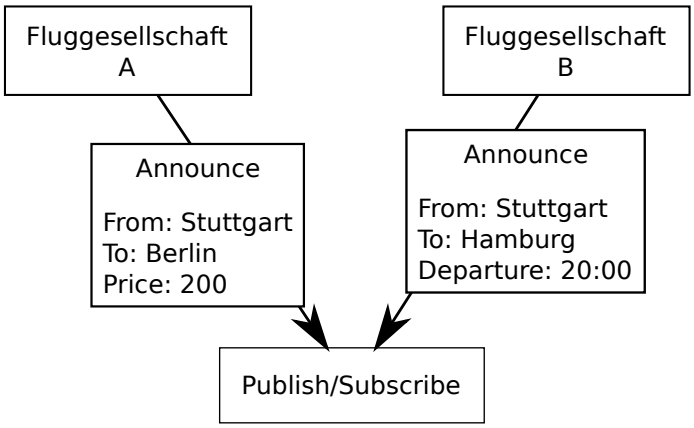

Abbildung 2.1: Beispiel für zwei unterschiedliche Announcements von zwei Fluggesellschaften

Es besteht also der Bedarf nach einem Ansatz, der dem Trend nach informationsgetriebenen Anwendungen Rechnung trägt.

Unter Publish/Subscribe versteht man ein Entwurfsmuster bei dem Konsumenten, Interesse für bestimmte Ereignisse bekunden (”subscribe”) und es Produzenten gibt, die diese Ereignisse generieren ("publish"). Die Aufgabe eines Publish/Subscribe Systems ist dabei, die auf die Bekundungen passenden Ereignisse an die Konsumenten weiterzuleiten.

Publish/Subscribe ist im Gegensatz zum oben beschriebenen Polling ein ereignisgetriebener Ansatz. Konsumenten subskribieren dabei eine Menge von Nachrichten, die an sie weitergeleitet werden sollen. Dafür werden vom System die zwei Methoden `subscribe()` und `unsubscribe()` angeboten. Ein Konsument könnte zum Beispiel mit `subscribe(From=Stuttgart, To=Berlin, Price<200)` dem Publish/Subscribe System mitteilen, nur Benachrichtigungen über Flüge von Stuttgart nach Berlin mit einem Preis von unter 200 Euro zu erhalten. Sobald nun ein Fluggesellschaft eine Notifikation wie zum Beispiel `publish(From=Stuttgart, To=Berlin, Price=123, Date=10.10.2011, Departure=14:12)` erzeugt, wird diese an den Konsumenten weitergeleitet. Dadurch wird nur kommuniziert, wenn es für den Konsumenten interessant ist und es werden unnötige Nachfragen vermieden.

Es ist für den Konsumenten auch unerheblich, von welchen Produzenten das Ereignis generiert wurde. Die Adressierung der Nachrichten erfolgt anhand deren Inhalt und nicht wer sie produziert hat oder wen sie letztendlich erreicht. Dadurch sind Produzent und Konsument nur sehr lose miteinander

verbunden. Beiden Akteuren muss nur das Publish/Subscribe System bekannt sein. Dies ermöglicht, dynamisch Akteure hinzuzufügen, wie auch zu entfernen. So sind auch subscriptions auf Ereignisse möglich, für die gerade gar kein Produzent vorhanden ist. Eine weitere Folge der losen Kopplung ist die Verlagerung der Last auf das Publish/Subscribe System. Für die Produzenten ist es unerheblich an wie viele Konsumenten die Benachrichtigungen weitergeleitet werden. Auch die Konsumenten können ihre Last, durch die Art ihrer Subskriptionen, selbst regeln.

Da die Konsumenten für bestimmte Nachrichten im System bekannt sind, können diese Nachrichten direkt weitergeleitet werden, was zu einer sehr geringen Latenz führt. Dies ist vor allem für sehr zeitkritische Anwendungen wie Börsensysteme wichtig. Haben mehrere Konsumenten die selben Menge von Nachrichten abonniert, bieten Publish/Subscribe Systeme die Möglichkeit, den Nachrichtenfluss zu den Konsumenten zu optimieren. Sollte zum Beispiel das Netzwerk die Möglichkeit von Multicast bieten, wäre es möglich, Konsumenten je nach ihrer Schnittmenge in verschiedene Multicast-Gruppen einzuteilen und so die Anzahl der zu verschickenden Nachrichten stark zu reduzieren.

2.2.1 Notification Filter

Einen besonderer Stellenwert bei Publish/Subscribe Systemen nimmt die Nachrichtenfilterung ein. Für die Konsumenten ist es wichtig, Filter so einrichten zu können, dass sie nur Nachrichten erreichen, die sie auch interessieren. Ausdrucksstarke Filter entlasten auch das Publish/Subscribe System, da es die Anzahl der zu verschickenden Nachrichten reduziert und so Last vom System als auch vom Netzwerk nimmt. Doch auch hier findet ein Trade-off zwischen Ausdrucksstärke und Skalierbarkeit statt. Sehr ausdrucksstarke Filter nehmen zwar Last vom Netzwerk und den Konsumenten, erzeugen jedoch eine höhere Last im Publish/Subscribe System. Im Folgenden werden drei mögliche Methoden zur Filterung in Publish/Subscribe Systemen vorgestellt [Eug01].

Kanäle

Eine einfache Filtermöglichkeit ist, Nachrichten in bestimmte Kanäle einzuteilen. Produzenten stellen dabei ihrer Nachrichten in vordefinierte Kanäle ein, die von den Konsumenten dann abonniert werden können. So könnte man sich bei einem Aktienkurs-Dienst je einen Kanal pro Aktie vorstellen. Diese Filter lassen sich einfach implementieren und können auch gut auf Multicast-Gruppen abgebildet

werden. Die Kanäle müssen jedoch im vorher definiert werden und erlauben daher den Konsumenten nur eine grobe Filterung. Diese Filter werden zum Beispiel vom CORBA Event Service eingesetzt.

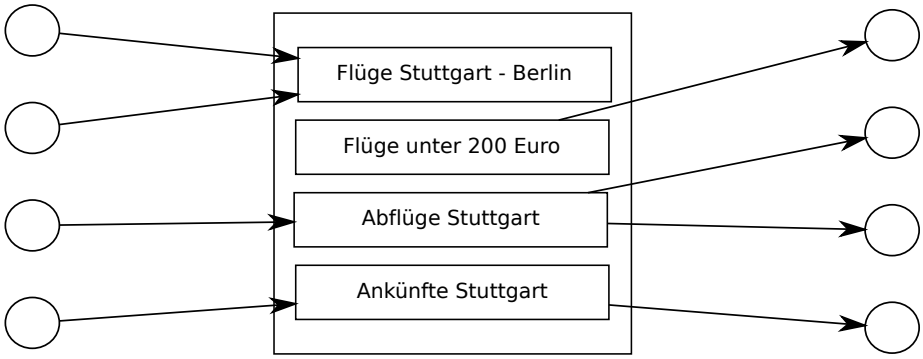

Abbildung 2.2: Beispiel für ein Publish/Subscribe System mit Kanälen

Subjects

Eine weitere Möglichkeit ist die Schaffung von Subjects. Dabei werden Nachrichten in einen Baum einsortiert, in dem Teilbäume jeweils Unterkategorien bilden. So würde ein Abonnement des Teilbaumes `Fluege.Stuttgart`, alle Flüge von und nach Stuttgart abdecken. Auch hier werden die Subjects vorgegeben und müssen den Produzenten und Konsumenten bekannt sein.

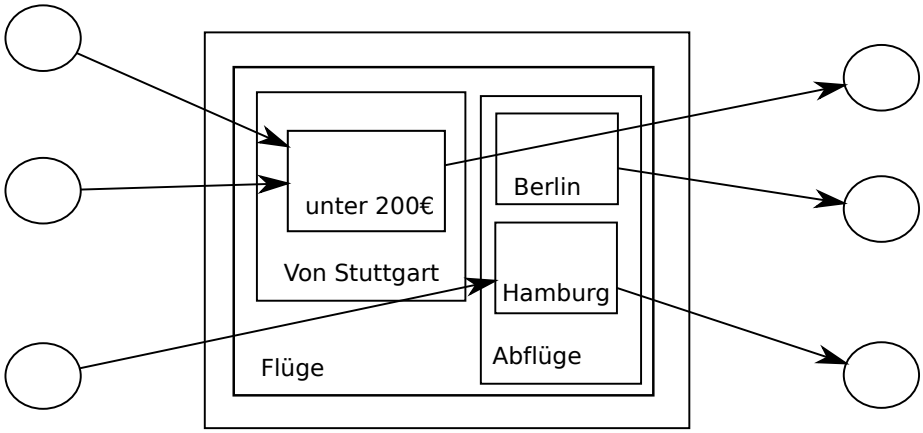

Abbildung 2.3: Beispiel für ein Publish/Subscribe System mit Subjects

Content Filter

Eine ausdrucksstärkere Methode ist es, anhand des Inhaltes der Nachrichten zu filtern. Dafür benötigt man sowohl ein Daten- als auch ein Filtermodell. Ein einfaches Datenmodell besteht dabei aus einem Schlüssel/Wert Paar. Filter schränken dabei den Wertebereich eines oder mehrerer Schlüssel ein. So bietet zum Beispiel SIENA die Vergleich- und Ordnungsoperationen $(=, \neq, <, >)$ sowie Teilzeichenfolge (*), Prefix (> *) und Suffix (* <) für Zeichenketten. So würde `{Departure=Stuttgart, Destination=Berlin, Price<200}` ein Filter für Flüge von Stuttgart nach Berlin unter 200 Euro einrichten. Bestehen mehrere Filter für den selben Schlüssel, müssen alle Filter erfüllt sein. Diese Filter auf den Inhalt, stellen eine ausdrucksstarke Möglichkeit der Filterung dar und ermöglichen eine sehr lose Kopplung von Produzenten und Konsumenten. Leider lassen sich diese Filter nur schwer auf Multicast-Gruppen abbilden, da die Anzahl der Gruppen exponentiell mit der Menge der Filterattribute anwachsen kann. Da diese Art von Filter hohe Anforderungen an die Infrastruktur bezüglich Rechenleistung und Netzwerklast stellen, ist besonders bei großen Wide-Area Netzwerken eine skalierbare und verteilte Infrastruktur nötig. Ein möglicher Ansatz am Beispiel von SIENA wird in [CRW03] beschrieben.

Publish/Subscribe Systeme sind schon seit längerem Gegenstand der Forschung. So sind an mehreren Universitäten unterschiedliche Systeme wie SIENA (Univ. of Colorado at Boulder), Jedi (Politecnico di Milano, Italy) oder Elvin (DSTC, Australia) entstanden. Auch in der Industrie werden Publish/Subscribe Systeme als Teil der Messaging Middleware eingesetzt. Beispiele hierfür sind TIBCO Rendezvous, IBM WebSphere MQ oder Implementierungen auf dem Java Message Service (JMS) wie der SUN ONE Message Queue.

3 Entwurf

In diesem Kapitel wird ein Entwurf für ein Gateway zwischen Android basierten Sensornetzwerken und einem Publish/Subscribe System vorgestellt. Das Gateway übernimmt dabei die Funktion eines Message Brokers [GH03]. Es empfängt Sensorinformationen von verschiedenen Sensornetzwerken und leitet diese an das Publish/Subscribe System weiter. Außerdem nimmt es Subskriptionen vom Publish/Subscribe System entgegen und abonniert die gewünschten Sensordaten beim Sensornetzwerk. Diese Entkoppelung von Sensornetzwerken und Publish/Subscribe System hat mehrere Vorteile.

Beide Systeme werden unabhängig von der jeweiligen Implementierung. Man stelle sich mehrere tausend Sensorknoten vor, die ihre Sensorinformationen direkt an ein Publish/Subscribe System liefern. Nun wird nach einiger Zeit das Publish/Subscribe System durch ein neues System ausgetauscht, das eine etwas andere Semantik bietet. Dies hat zur Folge, dass nun tausende Sensorknoten angepasst werden müssen.

Durch die Entkopplung entsteht die Möglichkeit der Filterung und Aggregation im Gateway. Bestehen zum Beispiel mehrere Abonnements für die selben Sensordaten, wird das Gateway diese beim Sensornetzwerk nur einmal anfordern. Damit wird Last vom Sensornetzwerk genommen und die Sensorknoten können stromsparender arbeiten.

Um eine besser Skalierbarkeit zu erreichen ist es möglich, das Gateway zu partitionieren und zu replizieren. Eine Partitionierung kann dabei zum Beispiel anhand der geographischen Position der Sensorknoten erfolgen. Dabei wird das betrachtete Gebiet mit einem Gitternetz überzogen und die Knoten im selben Quadranten werden einem Gateway zugeordnet. In [ALM11] wird ein solcher Ansatz beschrieben. Für eine Replikation ist eine Koordination zwischen den Gateway Knoten erforderlich. Dazu muss ein geeignetes Commit-Protokoll (z.B. Zwei-Phasen-Commit) implementiert werden. Diese Koordinierung kann auch von einem Datenbanksystem bereitgestellt werden.

Im Folgenden wird nun ein möglicher Entwurf für ein solches Gateway vorgestellt.

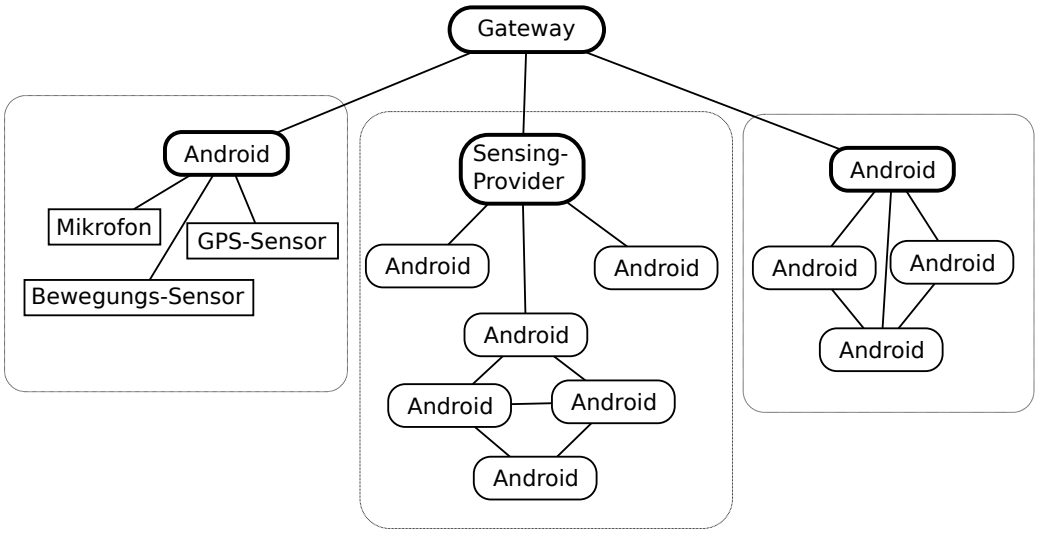

Abbildung 3.1: 3 Sensornetzwerke an einem Gateway

3.1 Schnittstelle zum Sensornetzwerk

Die im Folgenden beschriebene Schnittstelle dient der Integration von Sensornetzen. Dabei standen Sensornetzwerke aus mobilen Endgeräten mit Android Betriebssystem im Vordergrund, wie sie bereits in einer vorherigen Studienarbeit vorgestellt wurden. Es werden jedoch keine Android spezifischen Funktionalitäten benutzt, wodurch die Schnittstelle auch von anderen internetfähigen Geräten benutzt werden kann.

Unter einem Sensornetzwerk wird im Folgenden eine Menge von Sensoren verstanden, die über einen Sensorknoten am Gateway bekannt gemacht werden. Besteht ein Sensornetzwerk nur aus einem Android Gerät kann dieses, wenn gewünscht, direkt mit dem Gateway kommunizieren. Sollen mehrere Android Geräte zu einem Sensornetzwerk zusammengefasst werden, muss eine Instanz ausgewählt werden, die die Kommunikation mit dem Gateway übernimmt. Die Architektur hinter dieser Instanz kann vom Sensornetzwerk frei gewählt werden. So ist im Sensornetzwerk, zum Beispiel, sowohl eine Peer-to-Peer Struktur vorstellbar, die einen besonders langlebigen Knoten als Instanz auswählt, als auch eine Client-Server Architektur, in der ein dezidierter Knoten die Kommunikation mit dem Gateway übernimmt.

In Abbildung 3.1 sind exemplarisch drei mögliche Architekturen abgebildet. Das linke Sensornetz besteht dabei nur aus einem Knoten mit drei verschiedenen Sensoren. Hier übernimmt der Knoten direkt die Kommunikation mit dem Gateway. Das rechte Sensornetzwerk besteht aus einem Mesh-Netzwerk aus mehreren Android Knoten. Ein vom Mesh-Netzwerk ausgewählter Knoten übernimmt hier die Kommunikation mit dem Gateway und stellt diesem, die in den Android Knoten vorhandenen Sensoren, zur Verfügung. Das mittlere Sensornetz enthält einen dezidierten Knoten, der für die dahinterliegenden Sensoren die Kommunikation mit dem Gateway übernimmt. Dieser dezidierte Knoten hat dabei Kenntnis über alle in seinem Netzwerk vorhandenen Sensoren.

3.1.1 Protokoll

Im Folgenden wird das Protokoll zur Kommunikation zwischen Sensornetzwerken und dem Gateway beschrieben. Beim Entwurf des Protokolls standen folgende Punkte besonders im Vordergrund:

- Möglichst guter Kompromiss zwischen Wartbarkeit und Effizienz

  Binärprotokolle bieten meist eine sehr hohe Effizienz, da Daten ohne hohen Overhead und maschinennah übertragen werden können. Sie haben jedoch auch entscheidenden Nachteile. Sie sind meist schwierig zu Implementieren und es werden spezielle Werkzeuge benötigt, um sie auf Protokollebene zu analysieren. So haben sich im Internet meist textbasierte Protokolle durchgesetzt.

- Erweiterbarkeit

  Es sollte möglich sein, das Protokoll und die verwendeten Datenstrukturen im Nachhinein zu erweitern. Dabei sollte eine spätere Erweiterung möglichst rückwärtskompatibel möglich sein.

- Transportprotokoll

  In den letzten Jahren zeigte sich ein klarer Trend hin zu Web basierten Lösungen. In der Wahl des Transportprotokolls wurde daher versucht, diesem Rechnung zu tragen. Die Kommunikation sollte sowohl über Schicht 3 Protokolle, wie TCP und UDP, als auch über Schicht 7 Protokolle, wie HTTP oder SMTP, möglich sein.

- Geringe Anforderungen an das Sensornetzwerk

  Zwischen Sensornetzwerk und Gateway besteht ein asymmetrisches Ressourcen-Verhältnis. Sensorknoten sind sehr begrenzt in den ihnen zur Verfügung stehenden Ressourcen wie Energie, CPU oder Netzwerkbandbreite. Beim Gateway hingegen, handelt es sich um ein stationäres System, das leicht an die angeforderte Leistung angepasst werden kann. Es ist daher sinnvoll durch ein möglichst einfaches Protokoll geringe Anforderungen an das Sensornetzwerk zu stellen und rechenintensive Aufgabe auf das dahinterliegende System zu übertragen.

JSON

Nach Beachtung der oben genannten Punkte, fiel die Wahl auf ein auf JSON aufbauendes Nachrichtenformat. Die JavaScript Object Notation (JSON) [jso] ist ein textbasiertes Datenformat zum Serialisieren von strukturierten Daten. Es ermöglicht eine recht kompakte Formatierung und ist das native Datenformat der von den meisten Browsern unterstützten JavaScript Programmiersprache. Damit können auf HTML5 basierende Anwendungen direkt angebunden werden.

JSON ist aus Objekten und Arrays aufgebaut (siehe Listing 3.1).

Objekte sind dabei Schlüssel/Wert Paare der Form `Schlüssel:Wert`, die mit `{` beginnen und mit `}` enden. Der Schlüssel ist eine Zeichenkette und der Wert ein Objekt, ein Array, eine Zeichenkette, eine Zahl oder einer der Ausdrücke `true`, `false` oder null. Mit diesen Objekten lassen sich verschachtelte Strukturen aufbauen.

Arrays beginnt mit `[` und endet mit `]`. Sie enthalten eine durch Kommata geteilte, geordnete Liste von Werten, gleichen oder verschiedenen Typs. Leere Arrays sind zulässig.

Nachrichtenformat

Nachrichten werden in JSON formatierten Zeichenketten ausgetauscht. Eine Nachricht besteht dabei aus einem Objekt der Form

```
{ string: { members } }
```

wobei der `string` den Nachrichtentyp bestimmt und `members` die dazugehörigen Daten.

Listing 3.1 JSON Aufbau

```
object
        { }
        { members }

members
        pair
        pair, members

pair
        { string: value }

value
        string
        number
        object
        array

array
        [ ]
        [ elements ]

elements
        value
        value, elements

value
        string
        number
        object
        array
        true
        false
        null
```

Nachrichtenfluss

Der Nachrichtenfluss zwischen Gateway und Sensornetzwerk ist Beispielhaft in in Abbildung3.2 dargestellt.

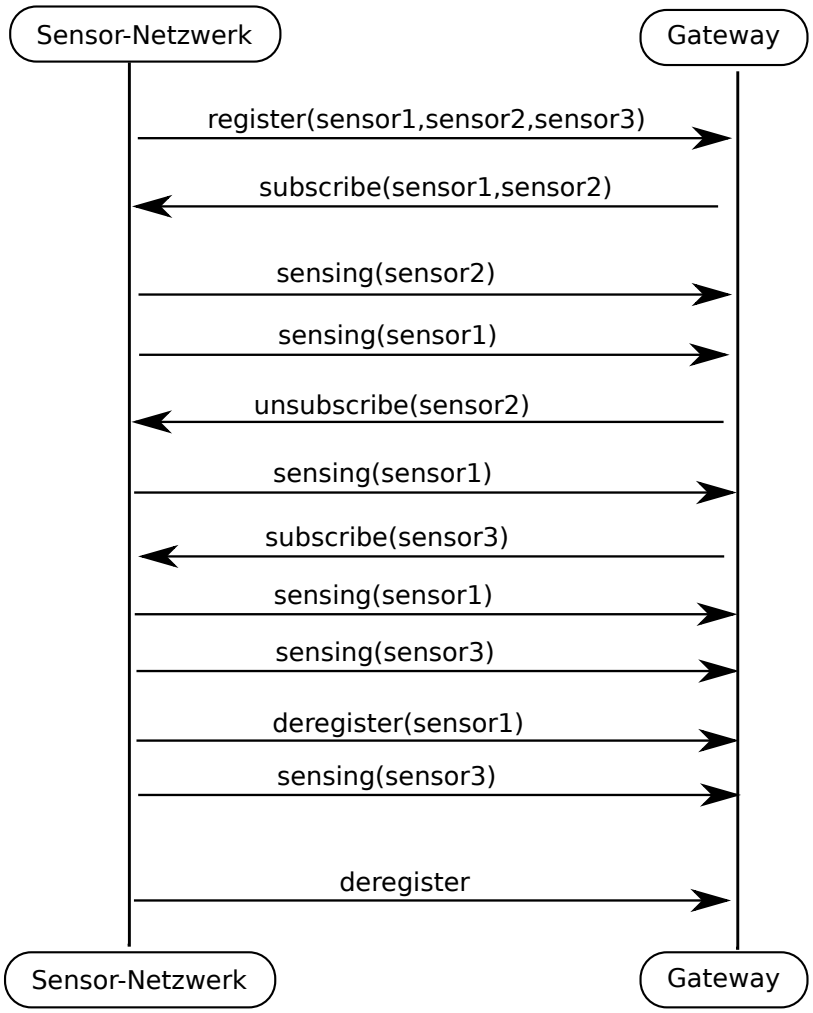

Abbildung 3.2: Kommunikation zwischen Sensornetzwerk und Gateway

In einem ersten Schritt registriert sich das Sensornetzwerk am Gateway und bietet diesem Sensordaten von drei verschiedenen Sensoren an. Das Gateway abonniert darauf hin Sensor 1 und Sensor 2. Danach überträgt das Sensornetzwerk 2 Sensordaten. Das Gateway hat nun kein Interesse mehr an Sensordaten von Sensor 2 und kündigt das Abonnement für diesen Sensor und abonniert Sensordaten von Sensor 3. Nach ein paar übertragenen Sensordaten, kann das Sensornetzwerk Sensor 1 nicht mehr anbieten. Es folgt noch eine weitere Sensormessung, bevor sich das Sensornetzwerk am Gateway abmeldet.

Nach dieser informellen Beschreibung werden nun die einzelnen Nachrichten definiert.

3.1.2 Nachrichtentypen

Registrierung

Zur Registrierung sendet das Sensornetzwerk eine *register* Nachricht. Mit dieser Nachricht teilt das Sensornetzwerk mit, welche Sensoren es anbieten kann.

Listing 3.2 register Nachricht

```
{ register: {
       sensing-id: string,
       address: string,
       area: [ { x: float, y: float } ],
       sensors: [ sensorobjects ]
}}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *sensing-id*: Ein eindeutige Bezeichnung des Sensornetzwerkes
- *address* (optional): Die Adresse, unter der das Gateway das Sensornetzwerk erreichen kann. Diese Adresse hängt vom darunterliegenden Transportprotokoll ab. Sollte dieser Schlüssel fehlen, ist das Sensornetzwerk unter der Adresse erreichbar, von der auch diese Registrierung erhalten hat.
- *area* (optional): Eine Liste von x,y Koordinaten, die als Float kodiert werden. Zwei Punkte definieren eine Gerade. Werden hier mehr als zwei Punkte angegeben, schließt der letzte und der erste Punkt ein Fläche. Fehlt dieser Schlüssel, wird das gesamte Gebiet abgedeckt.
- *sensors*: Eine Liste von Sensoren, die vom Sensornetzwerk angeboten werden.

  sensorobjects sind dabei wie folgt definiert:

Listing 3.3 sensorobject Struktur

```
{
        name: string,
        type: string,
        min-value: integer,
        max-value: integer
}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *name*: Ein in diesem Sensornetz eindeutiger Bezeichner des Sensor
- *type*: Gibt Auskunft über den vom Sensor angebotenen Typ. Dieser wird hier nicht weiter spezifiziert. Es wird angenommen, dass sich die einzelnen Parteien auf eine gemeinsame Taxonomie einigen.
- *min-value* (optional): Sollte dieser Sensor nur Daten über einem Wert liefern, kann hier ein Minimum definiert werden
- *max-value* (optional): Sollte dieser Sensor nur Daten unter einem Wert liefern, kann hier ein Maximum definiert werden

Abmelden

Mit einer *deregister* Nachricht kann sich das Sensornetzwerk abmelden. Werden zusätzlich Sensoren angegeben, werden nur diese abgemeldet. Wird der letzte registrierte Sensor abgemeldet, ist auch das Sensornetzwerk abgemeldet. Die darunterliegende Datenverbindung kann jedoch offen gehalten werden, um eine neue Registrierung zu ermöglichen.

Listing 3.4 deregister Nachricht

```
{ deregister: {
        sensing-id: string,
        sensors: [ sensorobjects ]
}}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *sensing-id*: Ein eindeutiger Bezeichner des abzumeldenden Sensornetzwerkes
- *sensors*: Eine Liste von Sensoren, die nicht mehr verfügbar sind. Die Liste besteht dabei aus *sensorobjects*, wie in Listing 3.3 definiert.

Abonnieren von Sensoren

Sensoren können mit einer *subscribe* Nachricht beim Sensornetzwerk abonniert werden. Nach dem Abonnieren werden vom Sensornetzwerk Sensordaten mit Hilfe von sensing Nachrichten übertragen. Sensoren die vorher nicht registriert wurden sind zu ignorieren.

Listing 3.5 subscribe Nachricht

```
{ subscribe: {
       sensing-id: string,
       sensors: [ sensorobjects ]
}}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *sensing-id*: Eine bei der Registrierung übermittelte *sensing-id*
- *sensors*: Eine Liste, bestehend aus *sensorobjects*, wie in Listing 3.3 definiert.

Abbestellen von Sensordaten

Sensordaten können mit einer *unsubscribe* Nachricht beim Sensornetzwerk abbestellt werden. Somit werden keine Sensordaten mehr für die angegebenen Sensoren übertragen. Wurden die übermittelten Sensoren vorher nicht abonniert, kann diese Nachricht ignoriert werden.

Listing 3.6 unsubscribe Nachricht

```
{ unsubscribe: {
       sensing-id: string,
       sensors [ sensorobjects ]
}}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *sensing-id*: Eine bei der Registrierung übermittelte *sensing-id*
- *sensors*: Eine Liste von Sensoren, für die Sensordaten abbestellt werden sollen. Die Liste besteht dabei aus *sensorobjects*, wie in Listing 3.3 definiert.

Übertragen von Sensordaten

Sensordaten werden mit einer *sensing* Nachricht an das Gateway übermittelt. Diese Nachricht kann auch ohne vorherige *register* oder *subscribe* Nachricht versendet werden. Dies ist vor allem für sehr kurzlebige Sensornetze nützlich.

Listing 3.7 sensing Nachricht

```
{ sensing: {
       sensing-id: string,
       sensors [ sensingobject ]
}}
```

Dabei haben die einzelnen Schlüssel/Wert Paare folgende Bedeutung:

- *sensing-id*: Eine bei der Registrierung übermittelte *sensing-id*
- *sensors*: Eine Liste von Sensordaten. *sensingobject* ist dabei wie folgt definiert:

Listing 3.8 sensing Struktur

```
{
       name: string,
       type: string,
       value: string,
       timestamp: string,
       area: [ { x: float, y: float } ]
}
```

Dabei haben die einzelnen Key-Value Paare folgende Bedeutung:

- *name*: Ein Bezeichner des Sensors im Sensornetz
- *type*: Gibt Auskunft über den vom Sensor angebotenen Typ. Dieser wird hier nicht weiter spezifiziert. Es wird angenommen dass sich die einzelnen Parteien auf eine gemeinsame Taxonomie einigen.
- *value*: Der gemessene Sensorwert
- *timestamp* (optional): Zeitstempel zum Sensorwert
- *area* (optional): Eine Liste von x,y Koordinaten, die als Float kodiert werden. Zwei Punkte definieren eine Gerade. Werden hier mehr als zwei Punkte angegeben, schließt der letzte und der erste Punkt ein Fläche. Fehlt dieser Schlüssel ist die Abdeckung dieses Sensorwertes nicht spezifiziert.

3.2 Schnittstelle zum Publish/Subscribe System

Wie schon in den Grundlagen beschrieben, bieten Publish/Subscribe Systeme Schnittstellen zum Veröffentlichen (Publish) und zum Subskribieren (Subscribe) von Benachrichtigungen (Notifications). In dem hier betrachteten Fall, werden von Sensornetzwerken Sensordaten veröffentlicht, die an Knoten mit passender Subskription weitergeleitet werden. Die Aufgabe der hier vorgestellten Schnittstelle besteht nun darin, eine mögliche Abbildung von Sensordaten auf ein Publish/Subscribe System zu schaffen. Dabei sind die Merkmale der Einzelsysteme zu beachten.

Nachfolgend werden content-based Publish/Subscribe Systeme betrachtet. Eine Subskription ist dabei ein Filter auf Benachrichtigungen und wird typischerweise als Prädikat auf den Inhalt der Subskriptionen definiert. Die Ausdrucksstärke, der zur Verfügung gestellten Filtermöglichkeiten, bestimmt dabei wesentlich die Flexibilität aber auch die Komplexität des Systems. Intern werden diese Systeme zur besseren Skalierbarkeit meist als verteilte Systeme implementiert. Dabei wird durch geeignete Routing- und Filterregeln sichergestellt, dass Benachrichtigungen so durch das verteilte System geleitet werden, dass sie den passenden Subskribenten erreichen. Wie dieses content-based Routing innerhalb eines verteilten Publish/Subscribe Systems aussehen kann, wird in [ACoo] am Beispiel von SIENA beschrieben. Dabei ist es von Vorteil, diese Routing- und Filterentscheidungen möglichst nahe am Produzenten von Benachrichtigungen, in diesem Fall dem Sensornetz, zu treffen.

So kann der Routing-Baum im verteilten System schon früh beschnitten werden und es wird unnötige Nachrichtenkommunikation vermieden. Folgend werden dazu zwei Ansatzpunkte aufgezeigt.

Einige Filterentscheidungen können schon im Gateway getroffen werden. So ist es zum Beispiel unnötig, Sensorinformationen in das Publish/Subscribe System einzubringen, für die es keinen Subskribenten gibt. Diese Sensorinformationen müssen, wenn möglich, erst gar nicht im Sensornetzwerk erhoben werden. Werden Sensorinformationen in das Publish/Subscribe System eingebracht, sollten diese, dessen Filtersemantik unterstützen. Eine *sensing* Nachricht, wie in Listing 3.8 definiert, würde von den meisten Publish/Subscribe Systemen nur als uninterpretierbare Zeichenkette verstanden werden, was ein effizientes Routing und Filtern unnötig erschwert.

Das Erheben von Sensordaten und deren Übermittlung ist, wie schon in Kapitel 2.1 beschrieben, eine teure Operation. Energieverbrauch, Rechenleistung und Übertragungsbandbreite sind in mobilen Geräten meist sehr limitierte Ressourcen, mit denen möglichst schonend umgegangen werden sollte. Es sollten daher vom Sensornetz möglichst nur Sensorinformationen generiert werden, die auch im Publish/Subscribe System subskribiert sind. Daher ist eine möglichst hohe Abdeckung von Subskriptionen und generierten Sensordaten wünschenswert.

3.2.1 Benachrichtigungen

Benachrichtigungen bestehen aus einer Menge von Schlüssel/Wert Paaren der Form $\alpha = (key_\alpha, value_\alpha)$. Dabei bezeichnet key_α eine beliebige Zeichenkette und $value_\alpha$ einen dazugehörigen Wert.

Dabei soll hier weder eine Taxonomie für key_α noch ein dazugehöriger Datentyp für $value_\alpha$ vorgegeben werden. Die Kommunikationspartner haben hier die Wahl, entweder durch vorherige Absprache oder durch das Kontaktieren eines geeigneten Dienstes, sich auf eine gemeinsame Taxonomie zu einigen.

Abbildung von Sensorinformationen auf Benachrichtigungen

Es ist nicht davon auszugehen, dass das Publish/Subscribe System in der Lage ist, JSON Nachrichten direkt zu interpretieren. Um dem Publish/Subscribe System trotzdem die Möglichkeit zu geben,

`{`
`  name: Temeraturknoten 1,`
`  type: Temperatur,`
`  area: [ { x: 10, y: 10 } ]`
`  value: 5`
`}`
	type = Temperatur
	name = Temperaturknoten 1
	value = 5
	posx = 10
	posy = 10

Tabelle 3.1: Beispiel für eine Benachrichtigungstransformation

sein Routing zu optimieren, sollte, wenn möglich, ein Abbildung von sensing Nachrichten auf Benachrichtigungen im Publish/Subscribe, erfolgen.

Tabelle 3.1 zeigt eine mögliche Benachrichtigung, die aus einer vom Sensornetz gelieferten sensing-Nachricht generiert wurde. Links ist dabei eine Sensing-Nachricht zu sehen und rechts die daraus generierte Benachrichtigung im Publish/Subscribe System. In diesem Beispiel wird angenommen, dass das Publish/Subscribe System in der Lage ist, Geoinformationen für das Routing zu nutzen. Dazu muss die Geoinformation aber in den Schlüsseln *posx* und *posy* übermittelt werden. Der Schlüssel *area* wird also für die Benachrichtigung in zwei Schlüssel transformiert.

Für komplexe Datentypen, wie in diesem Beispiel das abgedeckte Gebiet, muss je nach Publish/Subscribe System, eine Abbildung definiert werden. Es wird daher die folgende Abbildung vorgeschlagen:

- Einfache Datentypen

  Ist der Wert, in der vom Sensornetz gelieferten Sensornachricht vom Typ Integer, Float oder eine Zeichenkette, wird dieses Schlüssel/Wert Paar direkt in die Benachrichtigung übernommen.

- Komplexer Datentyp

  Handelt es sich nicht um einen einfachen Datentyp, wird dieser, wenn möglich, auf einen vom Publish/Subscribe System unterstützten Typ abgebildet. Sollte dies nicht möglich sein, wird der Wert als JSON formatierte Zeichenkette in die Benachrichtigung übernommen.

3.2.2 Subskriptionen

Subskriptionen sind eine Menge von Attributen der Form $\theta = (key_\theta, value_\theta, operator_\theta)$. Dabei sei key_θ eine beliebige Zeichenkette, $value_\theta$, der zu diesem key_θ gehörende Wert und $operator_\theta$, die auf den key_θ anzuwendende Bedingung. Bei der Wahl der unterstützten Operation wurden hier Operationen gewählt, die auch in den gängigen Publish/Subscribe Systemen vorhanden sind. Unterstützt werden die Operationen $=, \neq, <, >$. Dadurch ist das Publish/Subscribe System in der Lage, diese Operationen zur Optimierung seines Routing zu benutzen.

type	=	Temperatur
value	>	0
value	<	10

Tabelle 3.2: Beispiel für eine Subskription

type	=	Temperatur
value	=	5
posx	=	3
posy	=	5

Tabelle 3.3: Beispiel für eine überdeckte Benachrichtigung

type	=	Temperatur
value	=	20

Tabelle 3.4: Beispiel für eine nicht überdeckte Benachrichtigung

In Tabelle 3.2 ist ein Beispiel für eine Subskription zu sehen. Dabei ist der Absender dieser Subskription an Benachrichtigungen interessiert, die vom Typ *Temperatur* sind und einen Wert im Bereich von $0..10$ liefern. Die Benachrichtigung in Tabelle 3.3 wird von der Subskription überdeckt, da *value = 5* in den Wertebereich von $0..10$ fällt und auch der Schlüssel *type* die selbe Zeichenkette enthält. Dass in der Benachrichtigung noch weitere Attribute angegeben sind, spielt dabei keine Rolle. Die Benachrichtigung in Tabelle 3.4 überdeckt die Subskription jedoch nicht. Der gelieferte Wert von 20 fällt nicht in den von der Subskription gewünschten Bereich.

Enthält also eine Subskription mehrere Bedingungen für einen Schlüssel, werden diese Bedingungen als Konjunktion aufgefasst. Es müssen alle Bedingungen erfüllt sein. Eine Benachrichtigung b erfüllt einen Filter f oder f überdeckt b wenn

$$f \text{ überdeckt } b \iff \forall \theta \in f : \exists \alpha \in n : \theta \text{ überdeckt } \alpha$$

Abbildung von Sensorinformationen auf Subskriptionen

Die Aufgabe des Gateways besteht nun darin, die richtigen Sensordaten beim Sensornetzwerk zu abonnieren.

Dies soll an einem Beispiel erläutert werden. In Abbildung 3.3 erreichen das Gateway drei Subskriptionen von drei verschiedenen Knoten im Pub/Sub System. Knoten 1 interessiert sich für Sensordaten vom Typ *temp* in einem Wertebereich von 1 bis 10, Knoten 2 für einen Wertebereich von 5 bis 7 und Knoten 3 für Sensordaten mit dem Wert 37. Das Sensornetz bietet Sensordaten von Typ *temp* an.

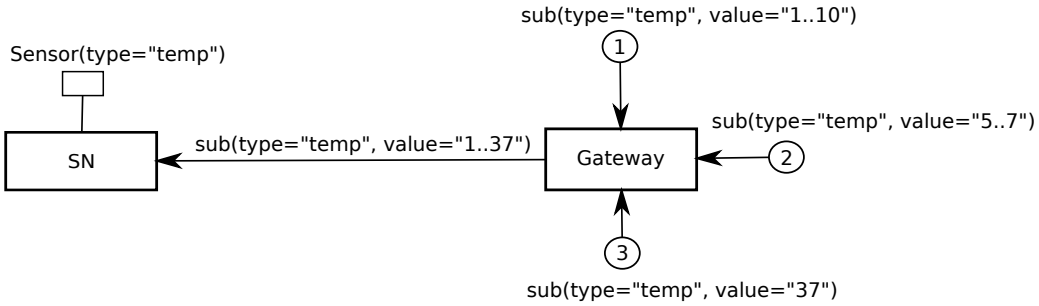

Abbildung 3.3: Beispiel zur Abbildung von Pub/Sub auf Sensornetz

Sollte das Sensornetzwerk kein Abonnement von Wertebereichen unterstützen, werden alle Sensordaten vom Typ *temp* abonniert.

Werden jedoch Wertebereiche vom Sensornetzwerk unterstützt wird iterativ ein Filterbaum aufgebaut. Dabei wird folgendermaßen vorgegangen:

- Empfang der Subskription von Knoten 1. Knoten *type = temp* und Knoten *value = 1..10* wird zum Filterbaum hinzugefügt.

Abbildung 3.4: Filterbaum nach Subskription von Knoten 1

- Empfang der Subskription von Knoten 2. Es wird der Knoten *value = 5..7* unter den Knoten *value = 1..10* eingefügt, da *5..7* von 1..10 überdeckt wird.

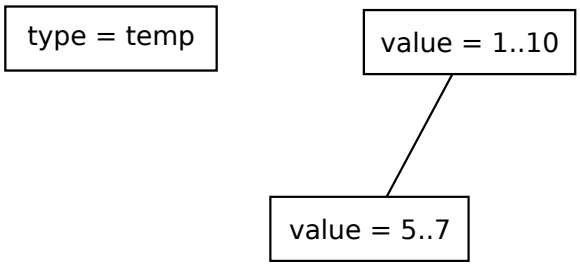

Abbildung 3.5: Filterbaum nach Subskription von Knoten 2

- Empfang der Subskription von Knoten 3. Es wird der Knoten *value = 37* hinzugefügt. Da der Wurzelknoten jetzt nicht mehr den gesamten Bereich des Baumes abdeckt, wird dieser auf *value = 1..37* erweitert.

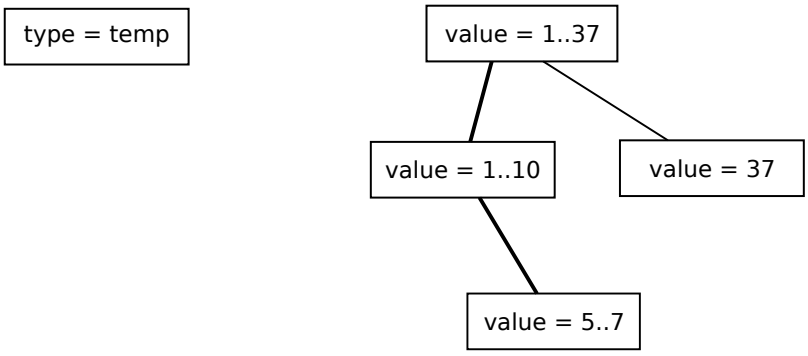

Abbildung 3.6: Filterbaum nach Subskription von Knoten 3

In dem hier vorgestellten Beispiel liefert das Sensornetzwerk Daten mit *type = temp*. Der *type* Baum wird überdeckt. Da ein Wertebereich von Seiten des Sensornetzwerkes nicht spezifiziert ist, überdeckt der vom Sensornetzwerk gelieferten Werte für *value* auch den Wurzelknoten aus dem

value Baum. Da in beiden Bäumen der Wurzelknoten überdeckt wird, können alle Subskriptionen vom Sensornetzwerk mit Sensordaten versorgt werden.

Würde das Sensornetzwerk für den Schlüssel *value* nur Daten im Bereich 4..8 liefern, könnte nur der Knoten mit *value = 5..7* überdeckt werden und somit nur für die Subskription von Knoten 2 Sensordaten beim Sensornetzwerk abonniert werden.

Für jede neue Subskription n sind also folgende Schritte zu erledigen:

1. Suche eine Subskription s welche Bäume für alle Schlüssel von n enthält. Kann s nicht gefunden werden, dann füge die neue Subskription n hinzu und aktualisiere die Abonnements beim Sensornetz.
2. Durchlaufe zu jedem Attribut n_a in n den passenden Baum s_b in s
   a) Wird der Wurzelknoten a von s_b nicht von n_a überdeckt, dann füge in b_n einen neuen Wurzelknoten k ein der den alten Wurzelknoten a und n_a überdeckt. Danach wird a und n_a an k angehängt und die Abonnements beim Sensornetz aktualisieren.
   b) Wird der Wurzelknoten a von n_a überdeckt, wird mit einer Tiefensuche bei überdeckenden Knoten so lange abgestiegen bis ein Blatt Knoten erreicht ist. n_a wird dann an dieses Blatt angehängt. Kann kein Blatt erreicht werden wird n_a an den Wurzelknoten a angehängt.

3.2.3 Advertisements

Advertisements werden genutzt, um den Willen oder die Möglichkeit, eine bestimmte Klasse von Benachrichtigungen zu publizieren, im Publish/Subscribe System bekannt zu machen.

Advertisements dienen dabei zwei Aspekten:

- Durch Advertisements erlangt das System Kenntnis über die Quelle und die Art, der von ihr verbreiteten Benachrichtigungen. Das System kann diese Information zur Optimierung seines Routings nutzen.
- Konsumenten im Publish/Subscribe System erfahren über mögliche Benachrichtigungen und können ihre Subskriptionen darauf abstimmen.

Advertisements sind dabei wie Subskriptionen aufgebaut. Subskriptionen werden jedoch vom Konsumenten generiert, bestimmen also das Ziel einer Bekanntmachung, wohingegen Advertisements vom Produzenten generiert werden, also die Quelle von Bekanntmachungen bestimmen.

Eine Menge von Benachrichtigungen n werden dann von Advertisements α überdeckt, wenn

$$\alpha \text{ überdeckt } n \iff \forall \alpha_n \in n : \exists \theta_\alpha \in \alpha : \theta_\alpha \text{ überdeckt } \alpha_n$$

type	=	Temperatur Innen
type	=	Temperatur Aussen
value	>	20

Tabelle 3.5: Beispiel für eine Advertisement

In Tabelle 3.5 ist ein Beispiel für ein Advertisement abgebildet. In diesem Beispiel werden Werte über 20 sowohl für *Temperatur Innen* als auch für *Temperatur Aussen* bekannt gemacht. Es ist zu beachten, dass im Gegensatz zu Subskriptionen, bei denen mehrere Attribute mit dem selben Schlüssel als Konjunktion aufgefasst werden, in Advertisements mehrere Attribute mit dem selben Schlüssel als Disjunktion interpretiert werden.

Abbildung von Sensorinformationen auf Advertisements

Advertisements dienen der Optimierung und werden nicht von allen Publish/Subscribe Systemen unterstützt. Sollte das Publish/Subscribe System Advertisements unterstützen, können *register* Nachrichten (siehe Kapitel 3.1.2) benutzt werden, um daraus Advertisements zu generieren. Zur Konvertierung werden die selben Regeln wie bei Benachrichtigungen angewendet.

Dabei ist abzuwägen wie häufig Advertisements versendet werden. Es ist davon auszugehen, dass von den Sensornetzwerken viele *register* und *unregister* Nachrichten generiert werden. Dies ist der dynamischen Natur, der hier betrachteten Sensornetzwerken, geschuldet. Sensoren können nur für eine kurze Zeit zur Verfügung stehen oder nur in kurzen Intervallen in der Lage sein, Sensordaten zu liefern. Im schlechtesten Fall wird zwischen einer *register* und *unregister* Nachricht nur eine *sensing* Nachricht versendet. Wie oben beschrieben, sollen Advertisements dem Publish/Subscribe System

helfen sein internes Routing zu optimieren. Für nur kurzzeitig verfügbare Sensordaten ist jedoch eine Optimierung nicht sinnvoll.

Es wird daher folgendes Vorgehen beim Empfang einer register Nachrichten r aus den Sensornetzwerken vorgeschlagen:

1. Prüfe für jeden Sensor s in r:
    a) Prüfe ob bereits ein Advertisement a für alle Attribute s_a in s versendet wurde
        i. Existiert ein solches a und alle s_a werden überdeckt, ist nichts zu tun
        ii. Existiert ein solches a und nicht alle s_a werden überdeckt, dann generiere ein neues Advertisement das alle Attribute in a und s_a überdeckt
        iii. Existiert kein solches a dann generiere ein neues Advertisement mit s_a

Dieses Vorgehen hat zur Folge, dass nur Advertisements mit wachsender Überdeckung versendet werden. Dadurch ist Anzahl der zu versendende Advertisements begrenzt.

4 Implementierung

In diesem Kapitel wird eine Implementierung der im vorhergehenden Kapitel beschriebenen Konzepte vorgestellt. Die Implementierung wurde in C programmiert und unter GNU/Linux entwickelt.

Dabei wurde ein Server entwickelt, der als Gateway zwischen den Sensornetzwerken und dem Publish/Subscribe System dient. Zudem wurde ein Client entwickelt, der die Sensornetzwerk Schnittstelle implementiert und zum Testen des Gateways verwendet wurde.

4.1 Designentscheidungen

4.1.1 Netzwerk

Die Kommunikation mit den Sensornetzwerk erfolgt über TCP/IP Verbindungen. Dabei sollte die Implementierung in der Lage sein, möglichst viele Verbindungen gleichzeitig zu verarbeiten.

UNIX Betriebssysteme generieren für jede TCP/IP Verbindung einen *file descriptor* über den Lese- und Schreibzugriffe abgewickelt werden. Mit Hilfe dieser *file descriptors* werden über Lese- und Schreib-Methoden Daten von den Verbindungen gelesen bzw. geschrieben.

Diese Lese- und Schreibzugriffe sind standardmäßig blockierend. Dies bedeutet, dass die Methodenaufrufe erst zurückkehren, wenn Daten erfolgreich gelesen bzw. geschrieben wurden oder ein Fehler aufgetreten ist. So blockiert zum Beispiel ein read() Aufruf den Prozess so lange bis Daten zum Lesen vorliegen.

Während der Prozess durch eine Verbindung blockiert ist, können keine anderen Verbindungen bearbeitet werden.

Es gibt zwei verbreitete Ansätze um dieses Problem zu lösen [Keg]:

- Ein Prozess/Thread für jeden Client und blockierende Methodenaufrufe

  Hierbei wird zu Beginn ein Prozess/Thread gestartet, der sich um die Annahme von neuen Verbindungen kümmert. Für jede einkommende Verbindung startet dieser Prozess/Thread einen eigenen Prozess/Thread, der dann die weitere Kommunikation für diese Verbindung übernimmt. Somit kann jeder Prozess/Thread blockieren.

- Ein Prozess für mehrere Clients und nicht blockierende Methodenaufrufe

  Bei diesem Ansatz werden *file descriptors* in einen nicht blockierenden Zustand geschaltet. Dies hat zur Folge, dass Lese- und Schreiboperationen immer sofort zurückkehren. Dabei wird durch einen geeigneten Rückgabewert signalisiert, ob der Funktionsaufruf erfolgreich war bzw. Daten zum Lesen oder Schreiben zur Verfügung standen. Somit können mehrere Verbindungen in einem Prozess/Thread verarbeitet werden. Das Betriebssystem signalisiert dabei dem Prozess/Thread, wann von einem *file descriptor* Daten gelesen oder geschrieben werden können.

In der hier beschriebenen Implementierung wurde der zweite Ansatz verwendet. Die Praxis hat gezeigt, dass es sich dabei um einen leistungsfähigen Ansatz handelt, der auch von leistungsfähigen Web Servern wie Lighttpd [lig] oder nginx [ngi] verwendet wird.

4.1.2 Verwendete Bibliotheken

Die Implementierung benutzt zusätzlich drei externe Bibliotheken. Alle stehen unter freien Lizenzen, welche eine Modifikation und Weiterverbreitung erlauben. Diese werden daher mit der Implementierung mitgeliefert.

libevent

Betriebssysteme stellen unterschiedliche Schnittstellen zum Signalisieren von lese- und schreibbereiten *file descriptors* zur Verfügung. So unterstützen alle UNIX Betriebssysteme die Funktionen select() und poll(). Diese wurden jedoch in den letzten Jahren durch effizientere Schnittstellen ergänzt. So wurde zum Beispiel unter Linux die epoll() und unter BSD die kqueue() Schnittstelle geschaffen.

Libevent [lib] abstrahiert dabei von diesen betriebssystemspezifischen Schnittstellen. Es bietet die Möglichkeit Callback Methoden für Ereignisse zu definieren. Dabei werden als Ereignisse sowohl Lese- und Schreibmöglichkeiten auf *file descriptors* als auch Betriebssystemsignale und timeouts unterstützt.

Durch die Verwendung von libevent, wird die Implementierung unabhängig von der betriebssystemspezifischen Schnittstelle und kann so die jeweils optimale Schnittstelle nutzen.

yajl

Bei yajl [yaj] handelt es sich eine Bibliothek zum Parsen und Generieren der JSON. yajl wird bei der Kommunikation mit den Sensor-Netzwerken verwendet.

list.h

Diese Header-Datei stellt eine doppelt verkettete Liste zur Verfügung. Es handelt sich dabei um eine Implementierung, die auch im Linux Kernel verwendet wird. Diese Listen werden in der Implementierung als Datenstruktur verwendet.

4.2 Programme

Die beiden Programme können mit verschiedenen Parametern von der Kommandozeile aus aufgerufen werden.

4.2.1 ./daemon

Das daemon Programm implementiert das Gateway zwischen Sensornetzwerken und einem Publish/-Subscribe System.

Parameter

- `-verbose|-v` : Schaltet eine ausführliche Ausgabe ein
- `-server|-s <ip:port>` : Gibt die Adresse und den Port auf dem neue Verbindungen von den Sensornetzen angenommen werden sollen
- `-usage|-help|-h` : Gibt eine Hilfe aus

4.2.2 ./sensing-client

Das sensing-client Programm implementiert das in Kapitel 3.1 beschriebene Protokoll auf Seiten des Sensornetzwerkes. Es wurde zum Testen der Gateway-Implementierung verwendet.

Parameter

- `-verbose|-v` : Schaltet eine ausführliche Ausgabe ein
- `-server|-s <arg>` : Adresse des Gateways
- `-port|-p <arg>` : Port des Gateways (default 4242)
- `-sensing_id|-i <arg>` : Eine eindeutige id für dieses Netzwerk
- `-sensing_type|-t <arg>` : Typ der vom Sensornetzwerk generierten Daten
- `-sensor_count|-c <arg>` : Anzahl der zu simulierenden Sensoren (default 10)
- `-sensings_per_second|-g <arg>` : Anzahl der simulierten sensing Nachrichten pro Sekunde (default 2)
- `-sensings_per_message|-m <arg>`: Anzahl der sensings pro Nachricht (default 1)
- `-usage|-help|-h` : Gibt eine Hilfe aus

4.3 Module

Die Implementierung verteilt sich auf verschiedene Module, die jeweils in eigenen c-Dateien implementiert sind und ihre Schnittstellen über Header Dateien exportieren. In Abbildung 4.1 sind schematisch die drei Module und die von diesen Modulen exportierten Methoden abgebildet.

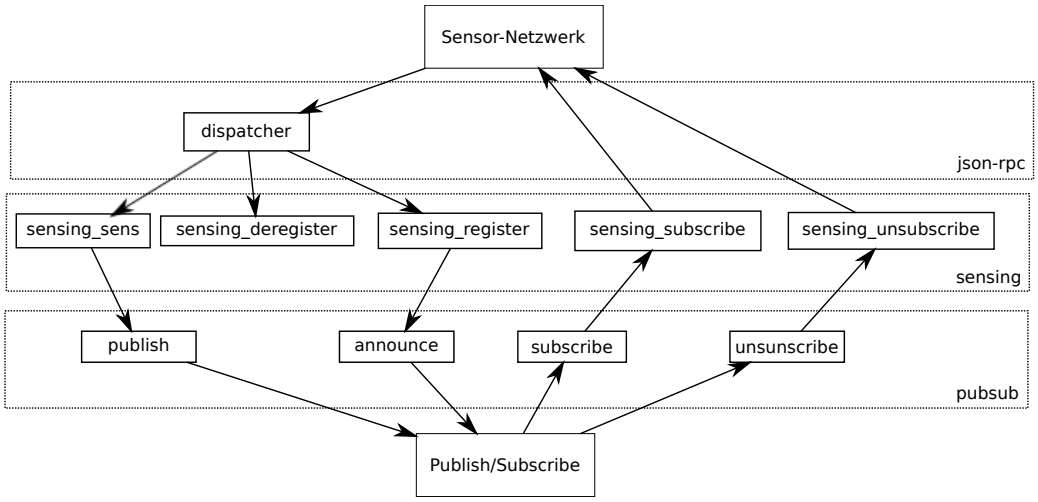

Abbildung 4.1: Schematischer Aufrufgraph der einzelnen Module

4.3.1 json-rpc

In diesem Modul ist die Netzwerkschnittstelle zum Sensornetzwerk implementiert. Es stellt zudem einen Callback Mechanismus zur Verfügung, um die unterschiedlichen Nachrichtentypen (siehe Kapitel 3.1.2) auf Methodenaufrufe abzubilden.

Zum Verwalten der registrierten Callback Methoden wird die in Listing 4.1 abgebildete Datenstruktur verwendet. Für jede registrierte Methode wird ein Objekt vom Typ `struct jsonrpc_opt` allokiert. Dabei wird einer Zeichenkette in `method` ein Funktionszeiger in `cb` zugewiesen. Dieses Objekt wird schließlich zur Liste `jsonrpc_opts` hinzugefügt. Zum Finden von Callbacks wird diese Liste linear durchlaufen.

Listing 4.1 Callback Datenstruktur

```
struct jsonrpc_opt {
        const char *method;
        jsonrpc_cb cb;
        struct list_head list;
};

LIST_HEAD(jsonrpc_opts);
```

Die Netzwerkkommunikation erfolgt über von der libevent Bibliothek bereitgestellten Methoden. Zu jeder TCP/IP Verbindung wird hierfür von der libevent eine Object vom Typ `struct bufferevent` generiert. Dieses Object enthält jeweils einen Zwischenspeicher für empfangene und versandte Daten. Mithilfe von der libevent bereitgestellte Methoden kann in diese Zwischenspeicher geschrieben oder davon gelesen werden. Liegen Daten im Sendespeicher, versucht libevent diese an die Gegenstelle zu versenden. Werden Daten auf der Schnittstelle empfangen, werden diese von der libevent in den Lesespeicher geschrieben.

Schnittstelle

Es werden folgende Methoden exportiert:

int jsonrpc_server(const char *addr)

Parameter

- `addr` : Adresse in der Form `ip:port` auf der neue Verbindungen vom Sensornetz angenommen werden sollen.

Diese Methode öffnet einen Port auf der in `addr` übergebenen Adresse. Dazu wird ein Socket generiert, dieser an die Adresse gebunden und mit `listen()` auf neue Verbindungen gewartet. Danach wird der event-loop aus der libevent gestartet.

void jsonrpc_register_cb(jsonrpc_cb cb, const char *method)

Parameter

- `cb` : Die aufzurufende Methode. `jsonrpc_cb` ist dabei von Typ

  ```
  typedef void (*jsonrpc_cb)(struct bufferevent *bev, yajl_val params)
  ```

- `method` : Eine Zeichenkette die mit der Methode verbunden wird.

Mit dieser Methode verbindet man die Methode `cb` mit einem Nachrichtentyp aus Kapitel 3.1.2.

So wird ein `jsonrpc_register_cb(sensing_message, ''sensing'')` Aufruf für jede eintreffende sensing Nachricht die Methode `sensing_message` aufgerufen. Die Nachricht wird dabei in `params` übergeben.

void jsonrpc_readcb(struct bufferevent *bev, void *rpc)

Parameter

- `bev` : Ein Zeiger auf den bufferevent für diese Verbindung
- `rpc` : Einen Zeiger auf eine `struct jsonrpc`

Diese Methode wird als Lese-Callback an die libevent übergeben. Sie wird von der libevent aufgerufen, wenn Daten zum Lesen zur Verfügung stehen.

4.3.2 sensing

Dieses Modul verarbeitet die von den Sensornetzwerken eintreffenden Nachrichten wie sie in Kapitel 3.1.2 definiert sind. Dazu wird mit Hilfe der Methode `jsonrpc_register_cb()` die in Tabelle 4.1 aufgelistete Verknüpfung hergestellt.

Außerdem stellt es Methoden bereit, um Nachrichten an das Sensornetzwerk zu versenden.

Zum Speichern von Sensornetzwerken und deren Sensoren wird die in Abbildung 4.2 abgebildete Datenstruktur verwendet. Ein Objekt vom Typ `struct sens_net` beschreibt dabei eine Sensornetzwerk. Es beinhaltet Daten über das Sensornetzwerk wie die sensing_id, die Netzwerkadresse und

Nachrichten aus den Sensornetz	Callback Methode
register	sensing_register()
unregister	sensing_unregister()
sensing	sensing_sens()

Tabelle 4.1: Callback-Methoden zu den jeweiligen Nachrichten aus den Sensornetzwerken

das vom Sensornetzwerk abgedeckte Gebiet. Zusätzlich wird die vom Sensornetzwerk empfangene register Nachricht in `register_json` referenziert. Besteht eine Verbindung zum Sensornetzwerk, beinhaltet `bev` einen Verweis auf den zu dieser Verbindung gehörenden bufferevent.

Die vom Sensornetzwerk registrierten Sensoren werden in der Liste `sensors` gespeichert. Diese Liste enthält Objekte vom Typ `struct sensor`. Diese Objekte enthalten Informationen über jeweils einen vom Sensornetzwerk angebotenen Sensor. Außerdem ist in `subscribed` vermerkt, wie viele Subskriptionen Interesse an Sensordaten von diesem Sensor haben. Wird dieser Wert 0 können Daten von diesem Sensor abbestellt werden.

Listing 4.2 Sensing Datenstruktur

```
struct sensor {
        char *name;
        char *type;
        int min_val;
        int max_val;
        yajl_val val;
        int subscribed;
        struct list_head list;
};

struct sens_net {
        char *sensing_id;
        char *address;
        yajl_val area;
        yajl_val register_json;
        struct bufferevent *bev;
        struct list_head sensors;
        struct list_head list;
};

static LIST_HEAD(sens_nets);
```

Schnittstelle

Es werden folgende Methoden exportiert:

void sensing_sens(struct bufferevent *bev, yajl_val sensing)

Parameter

- `bev` : Verbindung über die die Sensordaten empfangen wurden
- `sensing` : Die empfangenen sensing Nachricht

Diese Methode wird aufgerufen wenn Sensordaten, wie in Kapitel **??** definiert, vom einem Sensornetzwerk empfangen wurden. Nach dem parsen der Sensordaten in `sensing` werden die Daten an die publish() Methode des pubsub Moduls übergeben.

void sensing_register(struct bufferevent *bev, yajl_val deregister)

Parameter

- `bev` : Verbindung, über die die Nachricht empfangen wurde
- `deregister` : Die empfangene register Nachricht

Diese Methode wird aufgerufen wenn eine *register* Nachricht, wie in Listing 3.2 definiert, von einem Sensornetzwerk empfangen wurde. Nach dem Parsen der Nachricht wird das Sensornetzwerk in die Datenstruktur eingefügt. Dazu wird zunächst überprüft ob bereits ein Sensornetz mit der selben `sensing_id` vorhanden ist. Sollte dies der Fall sein, werden werden noch unbekannte Sensoren zum Sensornetz hinzugefügt. Ist dies nicht der Fall, wird ein neues Sensornetz allokiert.

void sensing_deregister(struct bufferevent *bev, yajl_val deregister)

Parameter

- `bev` : Verbindung über die eine deregister Nachricht empfangen wurde
- `deregister` : Die empfangenen deregister Nachricht

Diese Methode wird aufgerufen wenn eine *deregister* Nachricht, wie in Kapitel 3.1.2 definiert, von einem Sensornetzwerk empfangen wurde. Nach dem Parsen der Nachricht werden die in der *deregister* Nachricht aufgeführten Sensoren aus der Datenstruktur entfernt. Sollten danach keine Sensoren zu dieser `sensing_id` vorhanden sein, wird auch die Datenstruktur für das Sensornetz entfernt.

void sensing_subscribe(char *name, char *type, char *area, int min_val, int max_val)

Parameter

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet

- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Wird für ein Parameter ein NULL-Zeiger übergeben, entspricht das einem beliebigen Wert.

Diese Methode subskribiert Sensoren bei den Sensornetzwerken und wird durch subscribe Aufrufe aus dem Publish/Subscribe aufgerufen. Dazu wird überprüft, ob bereits überdeckende Subskriptionen in den Sensornetzwerken vorhanden sind. Sollte dies nicht der Fall sein, wird eine subscribe Nachricht aus Kapitel 3.1.2 generiert und an das Sensornetzwerk übermittelt. Für alle überdeckten Sensoren wird der `subscribed` Wert in `struct sensor` um eins erhöht.

sensing_unsubscribe(char *name, char *type, char *area, int min_val, int max_val)

Parameter

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet
- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Wird für ein Parameter ein NULL-Zeiger übergeben, entspricht das einem beliebigen Wert.

Diese Methode hebt Subskriptionen bei den Sensornetzwerken auf. Dazu werden überdeckte Sensoren in der Datenstruktur gesucht. Für jeden überdeckten Sensor wird dessen `subscribed` Wert um eins verringert. Wird dieser Wert 0, wird eine oder mehrere *unsubscribe* Nachrichten, wie in Kapitel 3.1.2 definiert, an die Sensornetzwerke übermittelt.

4.3.3 pubsub

Dieses Modul stellt eine Schnittstelle zum Publish/Subscribe System zur Verfügung.

Zum Verwalten der Subskriptionen aus dem Publish/Subscribe System, wird die in der Abbildung 4.3 abgebildete Datenstruktur verwendet. Für jede Subskription wird von der Methode `subscribe()` ein Objekt vom Typ `struct subscription` allokiert.

Listing 4.3 pubsub Datenstruktur

```
struct subscription {
        char *name;
        char *type;
        char *area;
        int min_val;
        int max_val;
        struct list_head list;
};
```

Schnittstelle

Es werden folgende Methoden exportiert:

struct subscription *find_covering_subscription(const char *name, char *type, char *area, int min_val, int max_val)

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet
- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Wird für ein Parameter ein NULL-Zeiger übergeben, entspricht das einem beliebigen Wert.

Diese Methode prüft, ob die übergebenen Werte schon von einer anderen Subskription aus dem Publish/Subscribe System überdeckt wird. Sollte dies der Fall sein, wird die überdeckende Subskription zurückgeliefert. Wenn das nicht der Fall ist, wird eine neue Subskription in der Datenstruktur angelegt.

void subscribe(char *name, char *type, char *area, int min_val, int max_val)

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet
- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Wird für ein Parameter ein NULL-Zeiger übergeben, entspricht das einem beliebigen Wert.

Diese Methode wird vom Publish/Subscribe System aufgerufen. Es wird mit Hilfe der Methode `find_covering_subscription()` überprüft, ob bereits eine überdeckende Subskription vorhanden ist. Sollte dies nicht der Fall sein, werden mit Hilfe von `sensing_subscribe()` aus dem sensing Modul passende Sensordaten abonniert.

void unsubscribe(char *name, char *type, char *area, int min_val, int max_val)

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet
- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Wird für ein Parameter ein NULL-Zeiger übergeben, entspricht das einem beliebigen Wert.

Diese Methode wird vom Publish/Subscribe System aufgerufen. Sie löscht die Subskription in der Datenstruktur und übergibt die Parameter an die Methode `sensing_unsubscribe()`.

void publish(char *name, char *type, char *area, char *timestamp, char *value)

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abgedeckte Gebiet
- `timestamp` : Zeitstempel des Sensorwertes
- `value` : Gemessene Sensorwert

Diese Methode wandelt die übergebenen Parameter in ein für das Publish/Subscribe System passendes Format um und ruft dann die `publish()` Methode des Publish/Subscribe Systems auf.

void announce(char *name, char *type, char *area, int min_val, int max_val)

- `name` : Der Name des Sensors
- `type` : Typ der Sensoren
- `area` : Das abzudeckende Gebiet
- `min_val` : Minimalwert der gewünschten Messwerte
- `max_val` : Maximalwert der gewünschten Messwerte

Diese Methode ruft die `announce()` Methode des Publish/Subscribe Systems auf.

5 Evaluation

Dieses Kapitel untersucht die in Kapitel 4 vorstellte Implementierung auf ihre Leistungsfähigkeit untersucht. Dazu wird zunächst die Testumgebung vorgestellt.

5.1 Testumgebung

Die Evaluation fand auf einem Core 2 Duo P8400 mit 4GB RAM unter Debian Linux statt. Als Kernel kam dabei die Version 3.0.2 zum Einsatz. Die Implementierung wurde mit der GNU Compiler Collection in der Version 4.6.1 mit dem Parametern

```
-g -O3 -Wundef -Wall -Wextra -Wcast-align -Wpointer-arith -Wredundant-decls
-Wstrict-prototypes -Wmissing-prototypes -Wno-pointer-sign
-Wno-unused-parameter -Wwrite-strings -Wformat=2 -march=native
```

übersetzt.

Die Sensornetzwerke wurden mit dem in Kapitel 4.2.2 vorgestellten sensing-client simuliert. Dabei kommunizierten das Gateway und der sensing-client über die loopback Schnittstelle. Als Transportprotokoll wurde TCP verwendet.

5.2 Testläufe

Es wurden mehrere Testläufe durchgeführt, um zu evaluieren, wie viele Sensordaten von Sensornetzwerken an das Publish/Subscribe System weitergereicht werden können. Dabei sollte im Besonderen untersucht werden, wie gut die Implementierung mit der Anzahl der weiterzuleitenden Sensordaten skaliert.

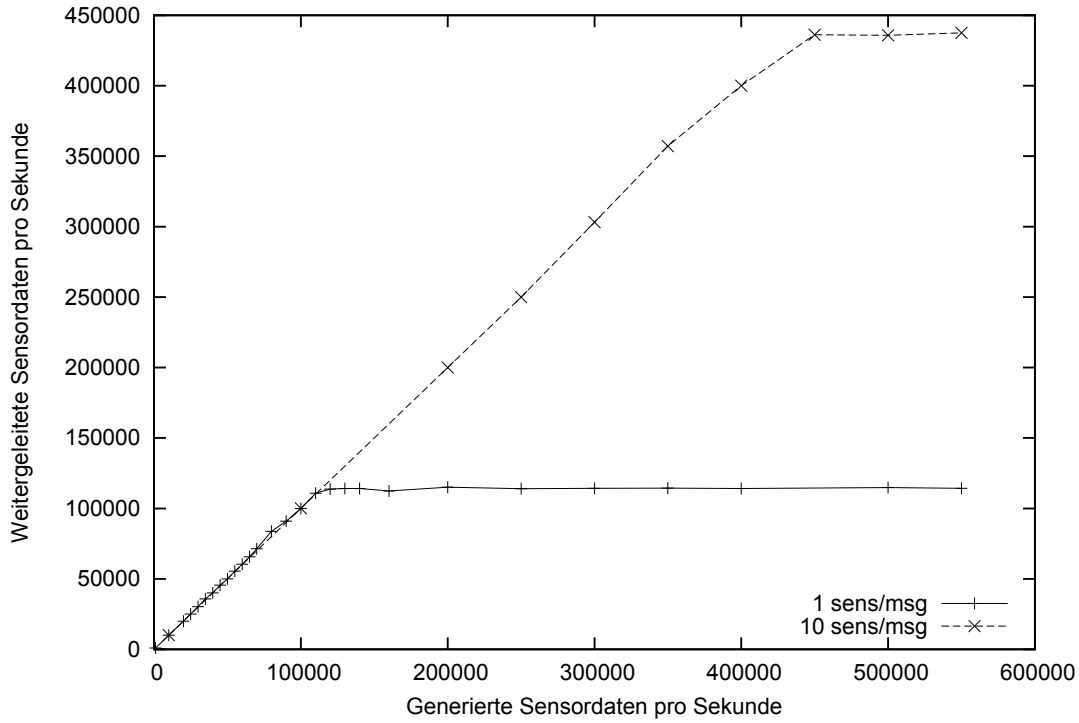

Abbildung 5.1: Testläufe mit einem Sensornetz

5.2.1 Ein Sensornetzwerk

In einem ersten Test wurde ein Sensornetz mit dem Gateway verbunden. Im Gateway wurde eine alles überdeckende Subskription eingerichtet. Es wurden zwei Testläufe durchgeführt, deren Ergebnisse in Abbildung 5.1 zu sehen sind.

Im ersten Testlauf wurde vom Sensornetzwerk für jedes Sensordatum eine sensing Nachricht generiert. Bis zu einer Rate von 110000 Nachrichten pro Sekunde war das Gateway in der Lage, diese Nachrichten zu verarbeiten und in der selben Rate publish Aufrufe für das Publish/Subscribe System zu generieren. Mit 110000 Nachrichten pro Sekunde war die CPU vollständig ausgelastet.

Im zweiten Testlauf wurden jeweils 10 Sensordaten in einer sensing Nachricht zusammengefasst. Damit konnten bis zu 430000 Sensorwerte pro Sekunde vom Gateway verarbeitet und in publish Aufrufe für das Publish/Subscribe System umgesetzt werden. Mit 430000 Sensorwerten und 43000 Nachrichten pro Sekunde war die CPU vollständig ausgelastet.

5.2.2 Mehrere Sensornetzwerke

In einem zweiten Test wurden mehrere Sensornetzwerke gleichzeitig mit dem Gateway verbunden. Jedes Sensornetzwerk baute dabei eine eigene TCP Verbindung zum Gateway auf. Im Gateway wurde wieder eine alles überdeckende Subskription eingerichtet. Jedes Sensornetzwerk generiert in diesem Test jeweils 1000 sensing Nachrichten pro Sekunde. Es wurden zwei Testläufe durchgeführt deren Ergebnisse in Abbildung 5.2 zu sehen sind.

Im ersten Testlauf wurde eine Sensorwert pro sensing Nachricht generiert. Bei 110000 Nachrichten pro Sekunde von 110 Sensornetzwerken war die CPU vollständig ausgelastet und es konnten nicht mehr Sensorinformationen an das Publish/Subscribe System weitergeleitet werden.

Im zweiten Testlauf wurden 10 Sensorwerte pro sensing Nachrichten, insgesamt also 10000 Sensorwerte pro Sekunde, generiert. Hier war die CPU bei knapp über 43 Verbindungen und 430000 Sensorwerten pro Sekunde voll ausgelastet. Es konnten maximal 43000 Nachrichten angenommen werden.

5.3 Analyse

Die Implementierung skalierte linear in allen untersuchten Fällen. Die Leistung des Gateways wurde nur durch die Rechenkapazität des Testsystems begrenzt. Dabei machte es keinen Unterschied, ob die sensing Nachrichten von einem oder mehreren Sensornetzwerken versendet wurden.

Durch das Zusammenfassen von mehreren Sensordaten in einer sensing Nachricht konnten im Gateway deutlich mehr Sensorwerte verarbeitet werden. Es ist also von Vorteil, wenn im Sensornetzwerk Sensordaten zusammengefasst werden können.

Die Anzahl der Verbindungen scheint weit weniger ins Gewicht zu fallen als die Anzahl der zu verarbeitenden Nachrichten. So konnten sowohl mit einem als auch mit mehreren Sensornetzen jeweils die gleiche Übertragungsleistung erreicht werden.

Es wurden in den Testläufe sehr viele Sensordaten pro Sekunde generiert. Es ist anzunehmen, dass in realen Sensornetzwerken eine weitaus geringe Anzahl an Sensordaten und sensing Nachrichten generiert und an das Gateway verschickt werden. Da die Anzahl der Verbindungen keinen großen Einfluss auf die Leistungsfähigkeit des Gateways hat, sollte bei ein oder zwei sensing Nachrichten pro

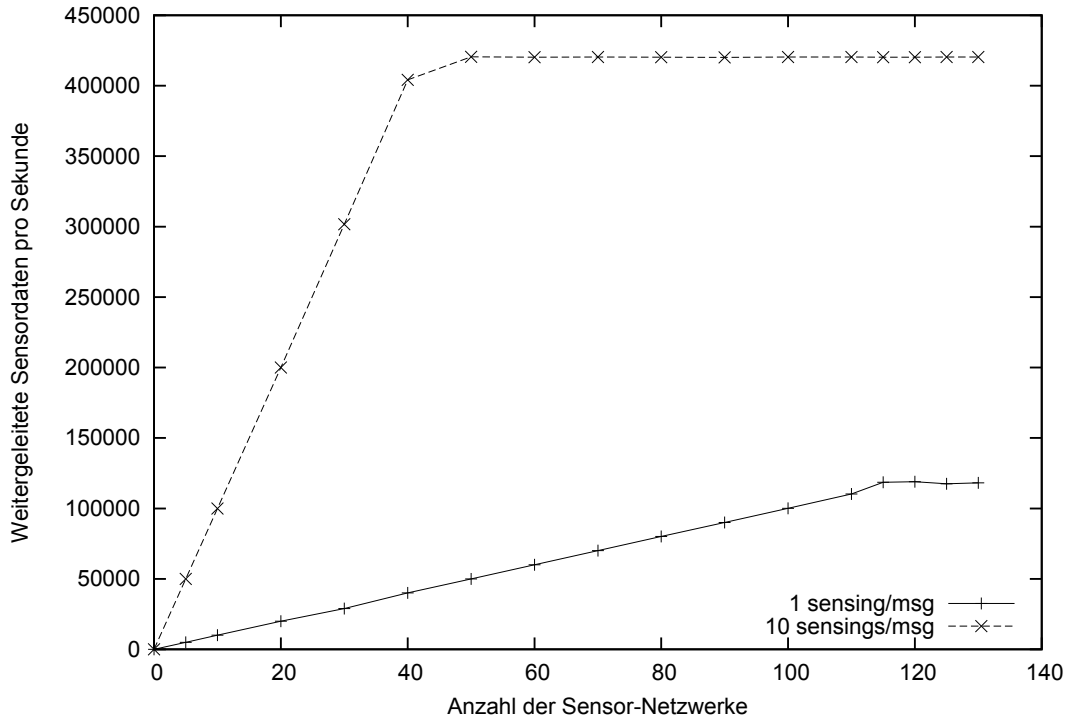

Abbildung 5.2: Testläufe mit mehreren Sensornetzen

Sekunden und Sensornetz, auch mehrere 10000 Sensornetzwerke angebunden werden können. Durch Vorgaben im Betriebssystem sind jedoch so viele TCP Verbindungen nur mit speziellen Einstellungen möglich. Hier bietet sich der Wechsel auf ein verbindungsloses Protokoll, z.B. UDP, an.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, wie man Sensornetzwerke an ein Publish/Subscribe System anbinden kann.

Dazu wurde zunächst ein Netzwerkprotokoll zur Anbindung von Sensornetzwerken entwickelt und eine Abbildung von Sensornachrichten auf Publish/Subscribe Systeme definiert. Die entwickelten Konzepte wurden in einer Implementierung realisiert und auf ihre Leistungsfähigkeit untersucht.

Es konnte gezeigt werden, dass sich Sensornetzwerke gut an Publish/Subscribe Systeme anbinden lassen. Die entworfenen Konzepte zeigten in der Implementierung eine gute Leistungsfähigkeit.

Trotzdem lässt die Implementierung noch Raum für Verbesserungen. So wurde die Datenstrukur möglichst einfach gehalten. Hier wäre es interessant, diese Datenstrukur durch eine leistungsfähige Datenbank zu ersetzen. Dabei bietet sich auch die Möglichkeit einer verteilten Datenspeicherung, zum Beispiel durch ein Peer-to-Peer System, an. Dadurch könnte leicht die Last auf mehrere Knoten verteilen.

Des Weiteren wäre es interessant das Transportprotokoll zu den Sensornetzen durch UDP zu ersetzen. Die Evaluation hat gezeigt dass dadurch eine größere Menge an Sensornetzwerken gleichzeitig angebunden werden kann.

Literaturverzeichnis

[ACo00] A. L. W. Antonio Carzaniga, David S. Rosenblum. Content-Based Addressing and Routing: A General Model and its Application. 2000. (Zitiert auf Seite 28)

[ALM11] J. H. Ahnn, U. Lee, H. J. Moon. GeoServ: A Distributed Urban Sensing Platform. In *Proc. 11th IEEE/ACM Int Cluster, Cloud and Grid Computing (CCGrid) Symp*, pp. 164–173. 2011. doi:10.1109/CCGrid.2011.10. (Zitiert auf Seite 18)

[ATC05] N. D. L. E. M. R. A. P. Andrew T. Campbell, Shane B. Eisenman. People-Centric Urban Sensing. 2005. (Zitiert auf den Seiten 10 und 12)

[CEL+08] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, G.-S. Ahn. The Rise of People-Centric Sensing. 12(4):12–21, 2008. doi:10.1109/MIC.2008.90. (Zitiert auf Seite 8)

[CRW03] A. Carzaniga, D. S. Rosenblum, A. L. Wolf. Design and evaluation of a wide-area event notification service. In *Proc. [Organically Assured and Survivable Information Systems] Foundations of Intrusion Tolerant Systems*, p. 283–334. 2003. doi:10.1109/FITS.2003.1264940. (Zitiert auf Seite 17)

[Eug01] P. T. Eugster. Type-Based Publish/Subscribe. 2001. (Zitiert auf Seite 15)

[GH03] B. W. Gregor Hohpe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc. Boston, 2003. (Zitiert auf Seite 18)

[HMCP04] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, M. A. Perillo. Middleware to support sensor network applications. 18(1):6–14, 2004. doi:10.1109/MNET.2004.1265828. (Zitiert auf Seite 11)

[HW10] K. R. Harald Weinschrott, Frank Dürr. StreamShaper: Coordination Algorithms for Participatory Mobile Urban Sensing. 2010. (Zitiert auf Seite 11)

[jso] JSON RFC 4627. URL http://www.ietf.org/rfc/rfc4627.txt. (Zitiert auf Seite 21)

[Keg] D. Kegel. The C10K problem. URL http://www.kegel.com/c10k.html. (Zitiert auf Seite 37)

[lib] libevent. URL http://monkey.org/~provos/libevent/. (Zitiert auf Seite 39)

[lig] lighttpd HTTP Server. URL http://www.lighttpd.net/. (Zitiert auf Seite 38)

[ngi] nginx HTTP server. URL http://nginx.org/en/. (Zitiert auf Seite 38)

[SGAP00] K. Sohrabi, J. Gao, V. Ailawadhi, G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000. doi:10.1109/98.878532. (Zitiert auf Seite 12)

[TAGH02] S. Tilak, N. Abu-Ghazaleh, W. Heinzelman. A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2):28–36, 2002. (Zitiert auf Seite 11)

[Wei02] M. Weiser. The computer for the 21st Century. 99(1):19–25, 2002. doi:10.1109/MPRV.2002.993141. (Zitiert auf Seite 8)

[yaj] yajl. URL http://lloyd.github.com/yajl/. (Zitiert auf Seite 39)

Alle URLs wurden zuletzt am 29.08.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Stefan Schweizer)