Software-Infrastrukturen für die Verwaltung von Sensordaten

Stefan Geiger

Departement für Informatik, ETH Zurich

geigerst@ethz.ch

Zusammenfassung

Dank der stark zunehmenden Verbreitung von Mobiltelefonen, welche mit geeigneten Sensoren ausgerüstetet sind, rückt die Vision, weltweite Sensor-Web-Applikationen zu betreiben, immer näher. Die riesige Datenmenge und stark unterschiedliche Verbindungs- und Sensorqualitäten stellen allerdings eine grosse Herausforderung für das Datenmangement und die Verwaltung der Sensorknoten dar. Gemeinsam genutzte Software-Infrastrukturen sollen die Entwicklung von Sensor-Web-Anwendungen vereinfachen. In dieser Arbeit werden verschiedene Anforderungen an solche Software-Infrastrukturen vorgestellt. Ebenfalls werden einige Projekte präsentiert, welche versuchen diese Anforderungen zu erfüllen.

1 Einführung

Immer mehr Mobiltelefone sind mit einer Vielzahl von Sensoren ausgerüstet. Dies erlaubt es weltweite Sensor-Web-Applikationen zu erstellen. Jedoch stellt sich die Frage, wie man die grosse Datenmenge speichern und analysieren kann. Zudem muss man mit der unregelmässigen geographischen Verteilung der Messwerte umgehen, welche auf Grund der unregelmässigen Anordnung der Sensorknoten und Übertragungsfehlern entsteht. Ein Überblick der Anforderungen an eine Software-Infrastruktur, welche als Basis zur Implementation von Sensor-Web-Anwendungen verwendet werden kann, wird in Sektion 2 gegeben. In Sektion 3, werden die verschiedenen Probleme des Datenmangements in weltweiten Sensornetzen beschrieben und einige Lösungsvorschläge vorgestellt. Drei Projekte, welche versuchen diese Anforderungen zu erfüllen, werden dann in Sektion 4 vorgestellt.

2 Anforderungen

Viele denkbare Anwendungen weisen gemeinsame Anforderungen auf, weshalb eine Software Architektur für die Verwaltung der Sensordaten, welche für diese Anwendungen eingesetzt werden könnte, und somit die Entwicklung von Sensor-Web-Anwendungen stark erleichtern würde, nützlich wäre.

Ein Beispiel für eine solche Anwendung ist die automatische Erstellung von Staumeldungen. Dabei werden die Mobiltelefone der Autofahrer verwendet um die Positionen und Geschwindigkeiten der Autos zu messen. Die Wahrscheinlichkeit eines Staus kann damit berechnet werden und wenn nötig direkt beim interessierten Autofahrer auf dem Mobiltelefon angezeigt werden. Es kann auch eine Online-Plattform angeboten werden, wo sich die Benutzer vor der Fahrt über aktuelle Staumeldungen informieren können.

Nach [2] und [5] werden folgende Anforderungen an eine solche Software-Infrastruktur gestellt.
Die Daten müssen gespeichert bleiben, auch wenn der Träger eines Knoten den Bereich verlässt, in welchem die Daten erfasst wurden. Insbesondere soll es auch möglich sein, diese Daten einfach abzufragen und zu verarbeiten. Diese Datenzugriffe hängen meist mit dem Standort des Benutzers zusammen, welcher sich allerdings schnell ändern kann, wenn sich der Benutzer bewegt. Ein Benutzer ist beispielsweise meistens nur an den näher gelegenen Staumeldungen interessiert. Geeignete Infrastrukturen sollten also auch örtlich bezogene Abfragen unterstützen, dürfen jedoch keinen fixen Standort der Benutzer voraussetzen.

Ein Sensing-Kontext beschreibt die Eigenschaften der Umgebung des Sensorknotens. Dazu gehören zum Beispiel die Position, die Orientierung und die Kalibrierungs-Parameter des Sensors oder auch die Qualität der Verbindung zwischen der Knoten. Diese Sensing-Kontexte sollten von der Software-Infrastruktur für die Applikationen zur Verfügung gestellt werden, um die richtige Interpretation der Daten zu ermöglichen, oder vom Sensing-Kontext abhängige Aktionen zu ermöglichen.

Ausserdem muss beachtet werden, dass die benötigen Daten, um eine Anfrage zu beantworten, über mehrere Netzwerke verteilt sein können und die Verbindungsqualität derer Knoten stark variieren kann.

3 Management von Sensor-Web-Daten

Ein weltweites Sensornetz generiert zu viele Daten, um diese manuell zu bearbeiten. Allerdings können auch die verbreiteten Programme wie Matlab, Mathematica und Excel nicht mit solchen Datenmengen umgehen. Der Entwickler einer Applikation sollte zudem so weit wie möglich nur mit einer Abstraktion der verwendeten Infrastruktur arbeiten müssen.

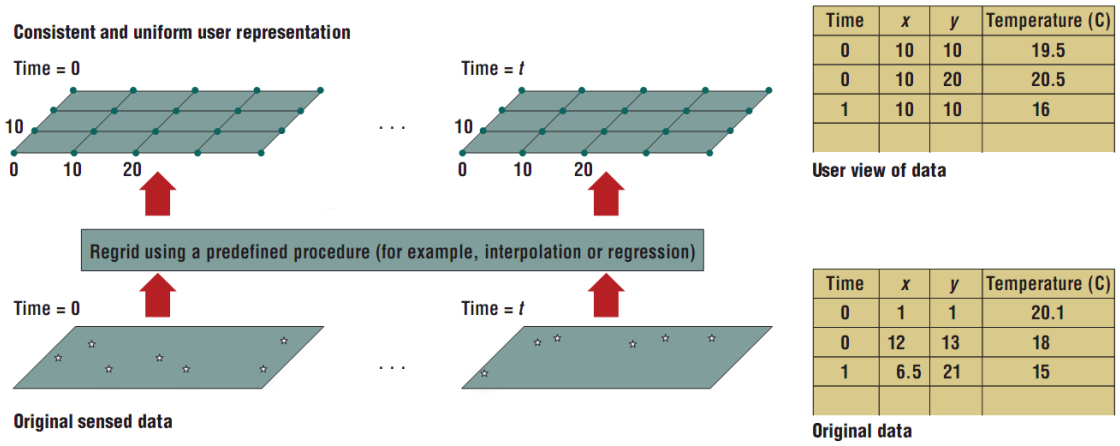
Ein guter Ausgangspunkt sind nach [3], die für normale Web-Applikationen verwendeten Datenbankmanagementsysteme. Diese sind darauf ausgelegt grosse Datenmengen zu verarbeiten und zu speichern. Zudem können sie mit parallelen und verteilten Rechnerstrukturen umgehen und bieten dazu die nötigen Abstraktionen an, um die Hardware- und Softwareplattform vor dem Benutzer zu verstecken. Deklarative Anfrageschnittstellen, welche von Datenbankmanagementsystemen angeboten werden, sind zudem intuitiver zu benutzen, als selbst geschriebene Programme, um die Daten zu verarbeiten. Datenschutz und Zuverlässigkeit werden durch Zugriffskontrollmechanismen und Replikation der Datenbank ebenfalls unterstützt.
Die bereits bestehenden Datenbanksysteme haben allerdings auch einige Nachteile, weshalb diese nicht direkt für Sensor-Web-Applikationen verwendet werden können.
Für Sensor-Web-Anwendungen irrelevante Funktionen, wie zum Beispiel Transaktionen, machen diese Datenbanksysteme zu langsam, um die sehr häufigen Schreibzugriffe der zahlreichen Sensoren zu verarbeiten.

Allerdings wurden in letzter Zeit Leichtgewichtsdatenbanken (engl. lightweight databases) entwickelt, welche diesen Anforderungen nachkommen [9]. Diese sind speziell auf eine Anwendung zugeschnitten und können dadurch zeitaufwändige Funktionen weglassen.

Die Sensordaten sind jedoch auch meistens mit Störungen behaftet, welche eine Unsicherheit der Daten verursachen, was andere Anforderungen an ein Datenbanksystem stellt, als die Daten, welche von einer Unternehmensarchitektur stammen, wofür die Datenbanksysteme normalerweise entwickelt wurden. Dise Unsicherheiten müssen beispielweise zusammen mit den Daten gespeichert werden und diese müssen bei Anfragen auch entsprechend beachtet werden. In den folgenden Abschnitten werden verschiedene Problemstellungen an das Datenmanagement in einem Sensor-Web beschrieben und einige aktuelle Lösungsvorschläge vorgestellt.

3.1 Datenaufbereitung

Die Qualität der Daten in einem Sensornetz variiert stark. Häufig fehlen Daten für einen Zeitpunkt komplett, zum Beispiel auf Grund von Übertragungsfehlern. Allerdings auch wenn die Daten übertragen und gespeichert werden können, besteht das Problem der Genauigkeit. Dies wird durch unterschiedliche Kalibrierung der Sensoren oder Rauschen, zum Beispiel durch schlechte Verbindungsstücke oder ungenaue Konvertierung von Analog- zu Digitialwerten, verursacht. Falls die Kalibrierungssparameter mitgeschickt werden, können die Daten nachträglich korrigiert werden und somit die Genauigkeit der Messwerte verbessert werden. Die Sensorknoten sind zudem meistens nicht in einer regelmässigen Anordnung angebracht, was die Analyse der Daten erschwert, auch wenn alle Knoten immer erfolgreich übertragen könnten. Um diese Probleme zu umgehen, werden die fehlenden Daten mit Hilfe von Interpolation und der Verwendung von vorhergehenden Messwerten ergänzt. Danach werden die Daten in ein gleichmässiges Gitter gebracht, um die Analyse, Visualisierung und den Vergleich mit anderen Datensätzen zu erleichern (siehe Abbildung 1).



Time	x	y	Temperature (C)
0	10	10	19.5
0	10	20	20.5
1	10	10	16

Time	x	y	Temperature (C)
0	1	1	20.1
0	12	13	18
1	6.5	21	15

Abbildung 1: Die Daten werden zur einfacheren Verwendung in ein regelmässiges Gitter gebracht [3]

Leider existieren nur wenige generisch verwendbare Programme, um die Daten zu korrigieren, Lücken aufzufüllen und danach in ein regelmässiges Gitter zu bringen. Sensorhersteller bieten zwar häufig eine

Kalibierungssoftware an, dies ist allerdings nur ein Teil des Ganzen, und die Programmierschnittstellen, falls überhaupt vorhanden, sind sehr unterschiedlich, sodass die Applikationsentwickler für jeden Hersteller speziellen Programmcode schreiben müssen.

3.2 Unsicherheit

Auf Grund der Ungenauigkeit der Sensoren, welche von Gerät zu Gerät auch stark variieren kann, sind die Daten immer mit einer gewissen Unsicherheit behaftet. Weitere Probleme entstehen falls weniger vertrauenswürdige Quellen verwendet werden, oder mehrere Werte mit einer gewissen Wahrscheinlichkeit möglich sind. Abbildung 2 zeigt einige Beispiele dazu.

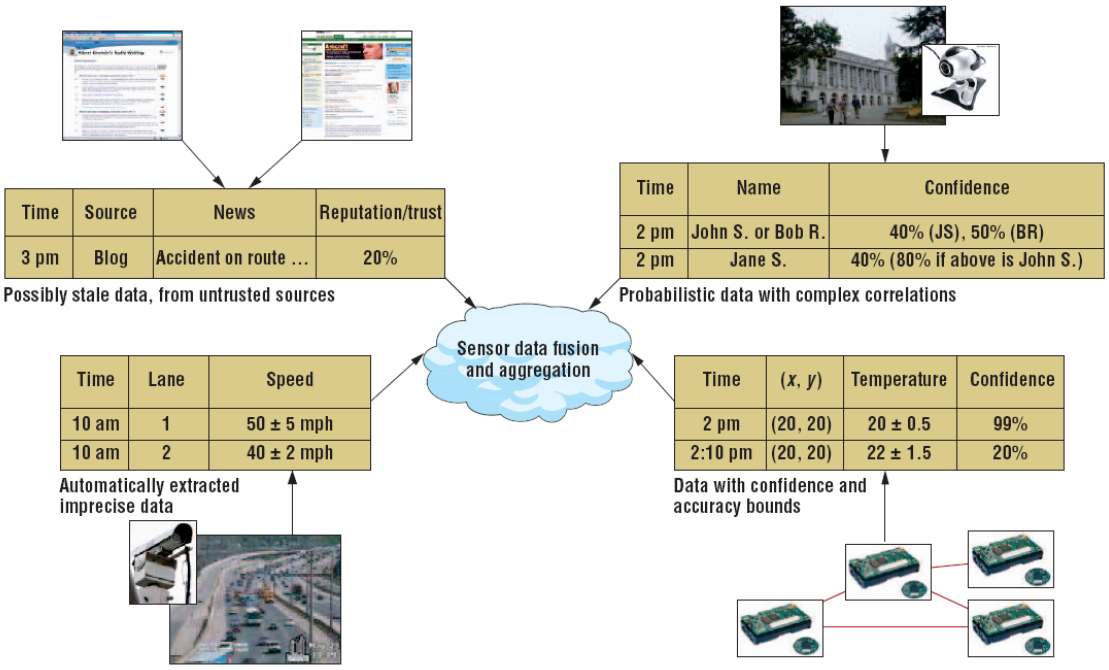


Abbildung 2: Unsicherheit und Qualitätsunterschiede [3]

Datenbanksysteme sind leider nicht geeignet, um Daten mit einer Unsicherheit zu speichern. Das Resultat einer Abfrage enthält entweder ein Datentupel, oder es enthält es nicht. Es gibt allerdings keine Möglichkeit anzugeben, wie wahrscheinlich ein Tupel in dem Resultat enthalten ist.
Abhilfe dafür bieten die probabilistischen Datenbanken [8]. Die Unsicherheit der Daten ist dabei als Wahrscheinlichkeit der Daten gespeichert. Dabei wird zwischen zwei Kategorien der Speicherung der Wahrscheinlichkeiten unterschieden:
Der *Tupel-Level*-Ansatz speichert eine Existenzwahrscheinlichkeit pro Tupel in der Datenbank. Dies kann verwendet werden, falls alle Attribute des Tupels sicher sind, die Existenz des Tupels allerdings nicht. Hingegen der *Attribut-Level*-Ansatz speichert pro Attribut eine Wahrscheinlichkeit der Präzision der Daten. Dieser Ansatz ist für Sensordaten besser geeignet, da die Messwerte meistens nicht exakt sind, die Existenz allerdings bekannt ist.

Eine verwandte Problemstellung mit der Ermittlung dieser Wahrscheinlichkeiten, ist die Bestimmung der Herkunft der Daten. Falls sich Daten als unwahr herausstellen, sollte die Möglichkeit bestehen, die Quelle als weniger vertrauenswürdig einzustufen. Die Herkunft zu bestimmen ist in einem Sensor-Web allerdings problematisch, da die Daten häufig mehrfach von verschiedenen Quellen zusammengefasst werden.

3.3 Interoperabilität

Sensor-Web-Applikationen benötigen häufig Daten von verschiedenen Quellen, welche von unterschiedlichen Organisationen verwaltet werden. Um diese Daten einfach automatisiert miteinander zu vergleichen, müssen ihre Formate genau bekannt sein. Diese Konzepte wurden bereits für das Semantische Web [1] entwickelt und können auch für Sensordaten verwendet werden. Die Daten werden in XML publiziert und werden mit einem entsprechenden RDF Dokument (Resource Description Framework) verlinkt, welches beschreibt was die Daten bedeuten. Als dritte Komponente existieren Ontologien, welche die Beziehungen zwischen einzelnen Elementen beschreibt. Zum Beispiel könnte von zwei verschiedenen Quellen die Temperatur in Celsius und Fahrenheit angegeben werden. Das RDF Dokument beschreibt dann, dass diese Werte Temperaturen sind und die Ontologie beschreibt die Beziehung zwischen den beiden Werten. Ein Programm kann dann damit automatisch die beiden Werte umrechnen und vergleichen. Leider fehlen zur Zeit noch die Programme, um Daten in diesen Formaten einfach zu veröffentlichen. Zudem müssen die entsprechenden Ontologien erarbeitet werden, welche dann gemeinsam genutzt werden müssen, denn das Konzept funktioniert nur, wenn alle die gleichen Ontologien verwenden, und nicht jede Organisation ihre eigenen.

4 Projekte

Im Folgenden werden drei Projekte vorgestellt, welche versuchen als Middleware zur Entwicklung von Sensor-Web-Applikationen zu dienen. Diese drei Projekte sind allerdings vor Allem auf eine infrastrukturlose System konzipiert und konzentrieren sich deshalb nur auf die direkte Anfrage der Daten von den Sensoren, nicht aber auf die dauerhafte Speicherung und das effiziente Auslesen der gemessenen Daten. Zudem wurden nur wenige Testapplikationen damit geschrieben, welche auch nur in einem kleinen Rahmen getestet wurden. Es ist also nicht klar, ob diese Projekte auch mit vielen Sensorknoten skalieren und ob die Schnittstellen die nötige Abstraktion und Flexibilität bietet um allen gewünschten Anwendungen gerecht zu werden.

4.1 Contory

Contory (*Con*textfact*ory*) [6] wurde an der Universität Helsinki entwickelt und bietet Unterstützung zur Programmierung von Ad-hoc-Sensornetzen mittels einer deklarativen Anfragesprache an. Der Entwickler betrachtet dabei das Sensornetz ähnlich einer Datenbank und verwendet eine SQL (Structured Query Language) ähnliche Anfragesprache. Es wird zwischen drei verschiedenen Kontexttypen unterschieden. Ein lokaler Kontext bietet Daten zu den lokal vorhandenen Sensoren an, der Ad-hoc-Kontext verwaltet dabei die Daten von anderen Netzknoten und der Infrastruktur-Kontext stellt die Verbindung zu meistens fest installierter Infrastrukutur, zum Beispiel Datenbank-Server, bereit. In einer Anfrage werden, ähnlich wie bei SQL, diejenigen Attribute, welche ausgelesen werden sollen, angegeben. Anders als bei SQL, wird statt ei-

ner Tabelle, einer der drei Kontexte angegeben von wo die Daten gelesen werden sollen. Zusätzlich werden länger dauernde Anfragen unterstützt, zum Beispiel event-basierte oder periodische Anfragen.

Das folgende Beispiel zeigt eine Anfrage, um die Geschwindigkeiten der Ad-hoc-Netzknoten abzufragen:

```
SELECT speed
FROM adHocNetwork(all, remote_region)
WHERE accurate = true
DURATION 1 hour
EVENT AVG(speed) < min_speed
```

Resultate werden zurückgegeben, sobald die durchschnittliche Geschwindigkeit der Sensoren in der angegebenen **remote_region** die spezifizierte Minimalgeschwindigkeit unterschreitet. Die **FROM** Anweisung unterstützt die drei verschiedenen Kontexte, womit man angeben kann, woher die Daten stammen sollen. Diese Angabe kann auch weggelassen werden, wodurch Contory den besten verfügbaren Kontext aussucht und von da die nötigen Informationen liefert.

Contory wurde in Java mit der Java 2 Platform Micro Edition implementiert. Dazu wurde eine Testapplikation zur Wetterbeobachtung implementiert, welche zum Beispiel einen Autofahrer vor Nebelbänken warnen kann. Dabei werden die Messwerte von Benutzern entlang der gefahrenen Strasse verwendet.

4.2 Spatial Programming

Da Contory zwar einfach zu verwenden ist, allerdings für kompliziertere Aufgaben weniger geeignet sein kann, bietet die an der Rutgers Universität entwickelte Spatial Programming [4] Middleware, abhilfe. Spatial Programming unterstützt die imperative Programmierung von Ad-hoc-Netzen und basiert auf der Abstraktion, Sensornetze als einen einzigen virtuellen Namensraum anzusehen. Ein Name in diesem Namensraum besteht aus dem Tupel $\{space, tag, index\}$. Space ist dabei ein Name, welcher die geographische Lage eines Knoten beschreibt und Tag ist der Name eines Attributes des Knotens. Verschiedene Knoten mit den selben Space- und Tag-Angaben, können mit dem Index differenziert werden. Zusätzlich muss bei einer Abfrage eine maximale Dauer angegeben werden, um korrekt zu reagieren, falls der Knoten nicht erreicht werden kann.

Das folgende Beispiel zeigt ein vereinfachtes Programm, zur Abfrage der Geschwindigkeiten der benachbarten Knoten mit Hilfe der Spatial Programming Laufzeitbibliothek:

```
int i=0;
while (NOT_DONE) {
    try {
        Neighbors[] n = {remote_region:car[i]}.neighbors;
        int speed = 0;
        for (int j=0; j<n.length; j++) {
            speed += n[j].speed;
        }
        speed = speed/n.length;
        if (speed < 50) {
            {driver_region:driver_name[0]}.slowSpeed = true;
        }
    } catch (SpaceViolationException e) {
        i++;
    }
}
```

{remote_region:car[i]} ist dabei die Referenz auf die direkten Nachbarn des Knoten. Diese bieten eine

Liste aller ihrer Nachbarn an, zusammen mit den jeweiligen Geschwindigkeiten. Sobald die durschnittliche Geschwindigkeit unter 50 fällt, wird der Fahrer benachrichtig, indem das Flag slowSpeed gesetzt wird. Dieser wird mit der Referenz **{driver_region:driver_name[0]}** identifiziert.

Mit der Hilfe von Spatial Programming können Referenzen an beliebige Attribute von Knoten in einem Sensornetz gleich verwendet werden, wie Variablen im lokalen Speicher. Die Spatial Programming Laufzeitbibliothek übernimmt dabei die Zuordnung der Namen auf die physikalischen Adressen sowie das Routing und den Zugriff auf die Sensordaten.

4.3 Migratory Services Framework

Das Migratory Services Framework [7] ist eine weitere Middleware zur Implementierung von Sensoranwendungen. Dieses wurde an der Universität Helsinki entwickelt und basiert auf dem Client-Service-Modell. Die Applikation möchte dabei eine gewisse Dienstleistung erhalten. Welche Knoten die Dienstleistung dann erbringen, wird dabei durch das Framework bestimmt. Länger andauernde Aufgaben müssen dabei gegebenenfalls an andere Knoten weitergeleitet werden, falls die dienstleistenden Knoten die Aufgaben nicht mehr übernehmen können, zum Beispiel, weil sich das Zielgebiet der Anfrage ändert oder die Knoten das Zielgebiet verlassen. Das Migratory Services Framework übernimmt das automatische Wechseln der ausführenden Knoten. Eine Anwendung muss dabei nicht erfahren, wenn der Knoten gewechselt wird, da diese nur mit einem virtuellen Knoten arbeitet. Die Abbildung 3 zeigt das transparente Wechseln der Dienstleistungen zwischen den Knoten.

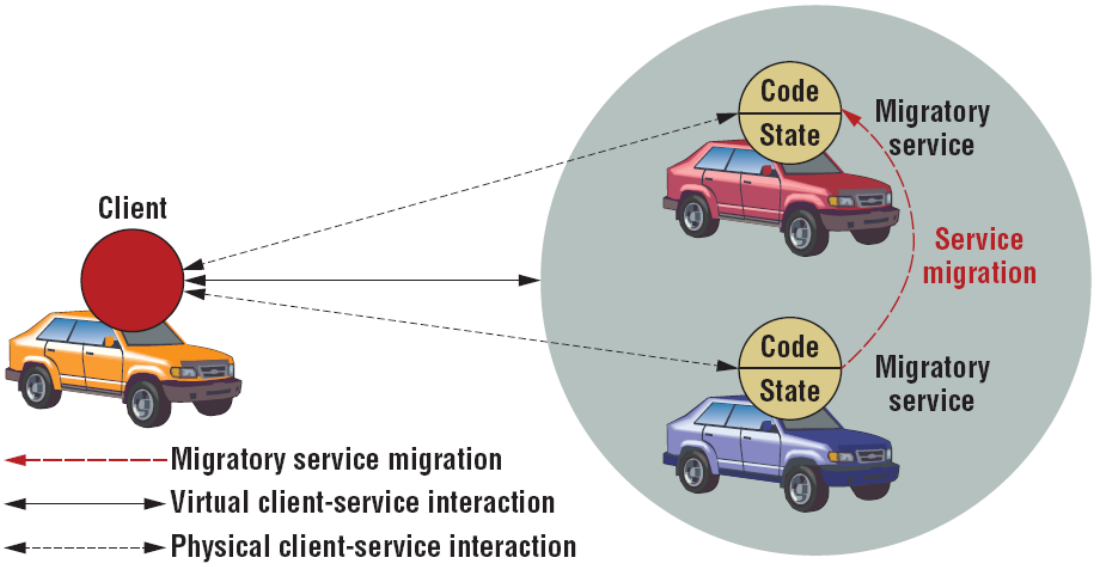


Abbildung 3: Migratory Services Framework: Automatisches Wechseln der dienstleistenden Knoten [6]

5 Schlussfolgerungen

Die Analyse der Anforderungen an weltweite Sensor-Web-Applikationen, hat gezeigt, dass eine gemeinsam genutzte Software-Infrastruktur denkbar ist, und damit die Entwicklungszeit solcher Applikationen stark verringert werden kann.

Die grosse Datenmenge in einer Sensor-Web-Applikation stellt eine Herausforderung an die Speicherung der Daten dar. Leichtgewichtsdatenbanken schaffen dafür eine Abhilfe, diese sind allerdings erst am Anfang der Entwicklungsphase. Zudem stellen die Unsicherheiten der Daten eine weitere Anforderung an die Datenbanken dar. Probabilistische Datenbanken bieten dafür einen Lösungsansatz, diese sind allerdings auch noch nicht genug weit entwickelt. Da die Daten meistens nicht immer vorhanden und auch nicht regelmässig geographisch angeordnet sind, werden Programme zur Datenaufbereitung benötigt, welche allerdings nur für spezielle Bereiche existieren.

Die vorgestellten Projekte Contory, die Spatial Programming Laufzeitbibliothek und das Migratory Services Framework bieten eine Grundlage für eine mögliche Software-Infrastruktur. Diese Implementationen wurden allerdings erst in einem kleineren Rahmen getestet worden, deshalb ist es nicht klar, ob sie auch für eine weltweite Anwendung funktionieren. Zudem wurde nicht auf das Problem des Datenmanagement eingegangen.

Literatur

[1] Semantic web. Webseite: http://www.w3.org/2001/sw.

[2] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich. Mobiscopes for human spaces. *Pervasive Computing, IEEE*, 6(2):20–29, April-June 2007.

[3] M. Balazinska, A. Deshpande, M. Franklin, P. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *Pervasive Computing, IEEE*, 6(2):30–40, April-June 2007.

[4] C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, and L. Iftode. Spatial programming using smart messages: Design and implementation. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 690–699. IEEE Computer Society, 2004.

[5] A. Campbell, S. Eisenman, N. Lane, E. Miluzzo, R. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *Internet Computing, IEEE*, 12(4):12–21, July-Aug. 2008.

[6] O. Riva and C. Borcea. The urbanet revolution: Sensor power to the people! *Pervasive Computing, IEEE*, 6(2):41–49, April-June 2007.

[7] O. Riva, T. Nadeem, C. Borcea, and L. Iftode. Context-aware migratory services in ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(12):1313–1328, 2007.

[8] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. pages 1053–1061, April 2008.

[9] J. Thomas and D. Batory. 1 p2: An extensible lightweight dbms, 1995.