

Protokolle und Interfaces für die Backend-Frontend-Kommunikation in ubiquitären Informationsumgebungen

Götz Bürkle¹

Telecooperation Office, Institut für Telematik, Universität Karlsruhe (TH)
SAP Research
goetz@buerkle.org

Zusammenfassung Mit der zunehmenden Miniaturisierung und der damit einher gehenden Verbreitung immer kleiner werdender „Computer“ kommen langsam neue Probleme zum Vorschein, an die vor einigen Jahren noch niemand dachte.

Heutzutage steckt in jedem Kleinstgerät ein „Rechner“, der immer häufiger auch mit seiner Umwelt kommuniziert. Als diese Geräte noch Träume waren wagte man noch nicht sich auszumalen, mit welchen Problemen man konfrontiert werden könnte, wenn die Zahl dieser Geräte stark ansteigen würde.

Jetzt, wo ubiquitäre Technologien eine gewisse Verbreitung gefunden haben und zum Beispiel als Sensornetze kein rein akademisches Thema mehr sind, muß man sich Gedanken machen, mit dem steigenden Kommunikationsaufkommen zwischen datengenerierenden Frontends und einem datenverarbeitenden Backend fertig zu werden.

Verschiedene Lösungsansätze dieser Problematik und einige beispielhafte Implementierungen derselben werden in dieser Arbeit vorgestellt.

1 Einleitung

Mit zunehmender Miniaturisierung und sonstiger Weiterentwicklung verschiedener Technologien werden die Möglichkeiten immer größer, ubiquitäre Informationsumgebungen zu etablieren. Mark Weisers Vision des „Ubiquitous Computing“, die er in „The computer for the 21st century“ [1] formulierte ist heute in manchen Bereichen zumindest in Labors schon Realität.

Wie Endres et al. in „A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing“ [2] beschreiben wächst das Interesse in diesem Gebiet weiter zu forschen immer noch.

Beim Begriff „ubiquitäre Informationsumgebungen“ sollte man nicht an das denken, was man heute unter vernetzten „Personal Computern“ versteht, sondern ein System von vielen geographisch verteilten und vernetzten Geräten. Überwiegt der sensorische Aspekt spricht man oft von Sensornetzwerken.

Das Thema mit dem sich diese Arbeit beschäftigt, die Backend-Frontend-Kommunikation in ubiquitären Informationsumgebungen, wirft vor allem in Sensornetzwerken Probleme auf, auf die ich im Verlauf dieser Arbeit eingehen und Lösungsansätze vorstellen werde.

Im Folgenden Abschnitt werde ich zuerst kurz einige Anwendungsbeispiele für ubiquitäre Informationsumgebungen aus verschiedenen Bereichen des alltäglichen Lebens vorstellen, um die Wichtigkeit und praktische Bedeutung des Themas aufzuzeigen.

Danach wird das Überlastungsproblem des Backends in der Backend-Frontend-Kommunikation aufgezeigt, um später im selben Abschnitt allgemeine Lösungsansätze für diese Problematik vorzustellen.

Nachdem ich allgemeine Lösungsansätze für diese Probleme, die schwerpunktmäßig in datenbankorientierte, hierarchieorientierte und ereignisorientierte Methoden aufgeteilt sind, angeführt habe will ich einige konkrete Implementierungen dieser Ansätze beispielhaft vorstellen.

Im letzten Abschnitt werde ich versuchen ein Fazit aus dieser Arbeit zu ziehen.

2 Anwendungsbeispiele

Um einen Eindruck davon zu bekommen, um welche Art von Anwendungen es hier geht will ich zuerst einige Beispiele aufzeigen, auf die ich dann später wieder verweisen kann. Als erstes Beispiel nehme ich den DigiClip (siehe Beigl et al., „DigiClip: Activating physical documents“ [3]), als zweites gehe ich kurz auf Möglichkeiten in der Lagerverwaltung (siehe Haller und Nochta, „Kooperation zwischen intelligenten Gütern“ [4]) oder der Produktion ein und als drittes skizziere ich kurz den Einsatz eines Sensornetzwerks zur Überwachung des Straßenverkehrs (siehe beispielsweise Madden et al., „Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data“ [5]).

2.1 DigiClip

Dieses System zielt darauf ab ausgedruckte Dokumente mit ihren digitalen „Originalen“ zu verknüpfen, Veränderungen anzuzeigen und durch räumliche Verfolgung der Dokumente auch festzustellen, ob die Dokumente nur an Orten bzw. in Räumen gelesen werden, an bzw. in denen dies erlaubt ist (sozusagen eine Abbildung des Rechtemanagements des Dokumentenverwaltungssystems).

2.2 Lagerverwaltung oder Produktion

Ein sehr typisches Beispiel für den Einsatz von Sensornetzen ist auch der Bereich der Lagerverwaltung oder in der Montage. In einem großen Lager werden täglich Unmengen an Waren ein- und ausgelagert. Von großem Interesse ist es möglichst effizient festzustellen, welche Waren wo eingelagert und wieder ausgelagert werden oder abzufragen, welche Waren sich überhaupt wo im Lager befinden.

Ein ähnliches Anwendungsgebiet ist die Produktion, wo es sinnvoll sein kann zu überwachen, welche Teile z. B. im Produktionsprozeß eines Automobils montiert werden um später z. B. das Produkt allein durch Integritätsbedingungen auf Vollständigkeit überprüfen zu können.

In der Literatur findet man beispielsweise in Haller und Nochta, „Kooperation zwischen intelligenten Gütern“ [4] und in Bonnet et al., „Towards Sensor Database Systems“ [6] derartige Beispiele.

2.3 Überwachung des Straßenverkehrs

Im Straßenverkehr gibt es viele Möglichkeiten mittels Sensoren Daten zu erheben. In „Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data“ [5] erwähnen Madden et al. ein System, das aus sehr vielen Induktionsschleifen besteht, womit es möglich ist den Verkehrsfluß über die Anzahl, Länge und Geschwindigkeit der Autos zu ermitteln.

Natürlich könnte man am Straßenrand auch Sensoren zur Emissionsmessung installieren oder versuchen mit Hilfe von Sensoren die Straßenverhältnisse wie Nässe, Nebel oder Glätte festzustellen. Hier sind also viele Einsatzmöglichkeiten denkbar.

Bei der Überwachung des Straßenverkehrs könnte man auch, wie Li et al. in „Event Detection Services Using Data Service Middleware in Distributed Sensor Networks“ [7] ausführen, neben den Sensoren Menschen mit weiteren mobilen Geräten wie beispielsweise PDAs mit den Sensordaten versorgen und sie so bei Ihrer Arbeit unterstützen.

3 Backend-Frontend-Kommunikation

Nachdem nun klar ist, um welche Arten von Anwendungen es überhaupt geht werde ich in diesem Abschnitt auf die grundsätzliche Problematik, die solche Anwendungen mit sich bringen können, eingehen, wozu ich zu Beginn einige Begriffe definieren werde, um dann darzustellen, wie das Problem entsteht und welche Ideen zur Lösung desselben in der Literatur diskutiert werden.

3.1 Grundsätzliche Problematik

Was sind Backend und Frontend

Im Bereich der ubiquitären Informationsumgebungen versteht man unter Backend in der Regel einen „zentraler Kern“ eines Systems. Oft ist dieser „zentrale Kern“ ein Server, der bestimmte Aufgaben übernimmt (zur Definition siehe auch „RUNES: Survey of Middleware for Networked Embedded Systems“ [8]).

Im Gegensatz dazu bezeichnet man die verteilten Komponenten eines solchen Systems als Frontend. In diesem Sinne versteht man unter „Frontend“ also nicht die Benutzerschnittstelle, wie man meinen könnte, sondern die Komponenten, die die Daten generieren.

Im vorgenannten Anwendungsbeispiel DigiClip stellen die DigiClips das Frontend dar, während der zentrale Server mit dem Dokumentenmanagementsystem das Backend darstellt.

In den Beispielen zur Lagerhaltung oder Produktion aus dem letzten Abschnitt sind die Komponenten, die die eingelagerten oder ausgelagerten Waren

erfassen bzw. die Komponenten, die den Einbau von Teilen erfassen das Frontend, während das System, das diese Daten dann weiterverarbeitet, das Backend ist.

Beim Beispiel der Überwachung des Straßenverkehrs bilden die verteilten Sensoren „auf der Straße“ das Frontend, während das Backend diese Daten sammelt und auswertet.

Was passiert zwischen Backend und Frontend

Im Fall eines Sensornetzwerkes sind also die verteilten Sensorknoten die Frontend-Komponenten. Diese erheben irgendwelche Daten, doch was passiert mit den Daten, wenn sie gemessen wurden? Sie müssen irgendwohin übermittelt werden, wo sie dann weiterverarbeitet werden können. Die Weiterverarbeitung geschieht, so zumindest der einfachste Ansatz, auf einem zentralen System, dem Backend. Und diese Datenübertragung zwischen Frontend-Komponenten und dem Backend ist die sogenannte Backend-Frontend-Kommunikation.

In kleineren Systemen kann man meist keine Probleme an dieser Architektur erkennen, doch sobald man die Zahl der Frontend-Komponenten erhöht, erhöht sich auch das Kommunikationsaufkommen zwischen dem Backend-System und den Frontend-Komponenten.

Jede Frontend-Komponente ist in der Lage eine bestimmte Bandbreite auszunutzen und je mehr Frontend-Komponenten man nun in das System einbindet, desto größer wird die gesamte Bandbreite mit der die Frontend-Komponenten kommunizieren können. Das zentrale Backend-System kann zwar bis zu einem gewissen Punkt soweit ausgebaut werden, daß es das zunehmende Datenaufkommen verarbeiten kann, doch früher oder später stößt man beim Backend-System an die Grenzen des technisch Machbaren, früher wahrscheinlich schon an die Grenzen des wirtschaftlich Vertretbaren.

Das Backend-System skaliert also nicht so gut, wie die dezentralen, verteilten Frontend-Komponenten, weswegen an der Schnittstelle zwischen dem Backend-System und den Frontend-Komponenten die Gefahr einer Überlastung besteht. Genau um dieses Überlastungsproblem soll es im Folgenden gehen.

Zuerst werde ich einige grundlegenden Lösungsansätze für dieses Problem darlegen, um danach kurz auf konkrete Implementierungen dieser Ansätze einzugehen.

3.2 Ideen für Lösungsansätze

Ein relativ weit verbreiteter Ansatz besteht in einer zunehmenden „Dezentralisierung des Backend-Systems“ in das (Sensor-)Netzwerk hinein, wodurch der zentrale Teil des Backend-Systems weniger Daten verarbeiten und mit weniger Frontend-Komponenten direkt kommunizieren muß, da bereits im Netzwerk eine Datenverarbeitung stattfindet, das sogenannte „in-network processing“.

Sensornetzwerke als Datenbanken

Solche Ansätze sehen das Sensornetz oft nicht mehr nur als einfaches Netz von

Sensoren, sondern gehen teilweise so weit, das Netz an sich als Datenbank anzusehen (siehe auch Govindan et al., „The Sensor Network as a Database“ [9] und Bonnet et al., „Towards Sensor Database Systems“ [6]). Die Benutzeranfragen gehen dann nicht mehr an einen zentralen Datenbestand, der von der Backend-Komponente verwaltet wird, sondern höchstens über das Backend in das (Sensor-)Netzwerk, dessen Knoten dann Teile der Anfrage abarbeiten und dem Backend-System bereits das Ergebnis dieses Teils mitteilt, so daß auf diese Weise das Backend-System entlastet wird.

In der Literatur findet man Begriffe wie lokalisierte Algorithmen bei Estrin et al. in „Next Century Challenges: Scalable Coordination in Sensor Networks“ [10], oft wird auch ein verteilter Ansatz einem Warehousing Ansatz gegenübergestellt, wie von Bonnet et al. in „Querying the Physical World“ [11] und „Towards Sensor Database Systems“ [6], Govindan et al. in „The Sensor Network as a Database“ [9] und Beutel et al. in „Prototyping Wireless Sensor Network Applications with BTnodes“ [12].

Der Datenbestand liegt also verteilt im Netz vor, beispielsweise auch als verteilte Tupel-Räume wie Davies et al. in „Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications“ [13] oder Costa et al. in „TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks“ [14] ausführen.

Diese Verschiebung von Funktionalität aus dem Backend in das Netzwerk bzw. in mehrere Netzwerkkomponenten ist den meisten Ansätzen ähnlich, jedoch unterscheiden diese sich in teilweise interessanten Details.

Passend zu der Sicht ein Sensornetzwerk als eine Art Datenbank aufzufassen nutzen viele Ansätze als Grundlage für die Abfrage von Informationen die Abfragesprache für relationale Datenbanken SQL (Structured Query Language) und erweitern diese dann, um sie auf die verteilte Infrastruktur anwendbar zu machen, wie es von Bonnet et al. in „Querying the Physical World“ [11] und „Towards Sensor Database Systems“ [6], Shen et al. in „Sensor Information Networking Architecture and Applications“ [15], Li et al. in „Event Detection Services Using Data Service Middleware in Distributed Sensor Networks“ [7], Madden et al. in „TinyDB: an acquisitional query processing system for sensor networks“ [16] oder Mascolo et al. in „RUNES: Survey of Middleware for Networked Embedded Systems“ [8] beschrieben wird.

Clustering - Bildung einer Hierarchie

Zentrale Bedeutung hat in den meisten Protokollen das Bilden einer, wie auch immer gearteten Hierarchie, also das Clustering. Wobei sich hier die einzelnen Ansätze unterscheiden. Einige nutzen alle oder auch nur ausgewählte Sensorknoten selbst zur Vorverarbeitung, andere wie Beutel et al. in „Prototyping Wireless Sensor Network Applications with BTnodes“ [12] schlagen vor dafür Extra-Komponenten wie PDAs als sogenannte „cluster-heads“ zu benutzen. Interessant an deren Ansatz ist insbesondere die Tatsache, daß die Software (sog. Smoblets) zur Verarbeitung nicht auf den PDAs selbst vorhanden sein muß, sondern von Sensorknoten bezogen werden kann.

In „Distributed Top-Down Hierarchy Construction“ [17] beschreiben Thaler et al. mit dem TDH (top-down hierarchy) Algorithmus eine effiziente Methode, um eine Hierarchie verfeinernd aufzubauen, die anderen aufbauenden Verfahren überlegen ist, da sie weniger Ressourcen benötigt.

Arbeiten mit Ereignissen

Weitere Ideen beschäftigen sich damit, die Kommunikation im Netz zu reduzieren und Energie zu sparen, indem man sie von Ereignissen abhängig macht oder den Kontext mit einbezieht, siehe Hill et al. in „System Architecture Directions for Networked Sensors“ [18] und Beigl et al. in „AwareCon: Situation Aware Context Communication“ [19].

Es kann sich lohnen unnötige Kommunikation zu verhindern, wenn man, wie Zhao et al. in „Information-driven dynamic sensor collaboration“ [20] vorschlagen, feststellen kann welcher Sensor zu welchem Zeitpunkt mit welchen anderen Sensoren kommunizieren soll.

In „Middleware infrastructure for context-aware ubiquitous computing systems“ [21] gehen Shehzad et al. sogar so weit die Behauptung aufzustellen, daß ereignisbasierte Middleware möglicherweise besser skaliert als andere („It is claimed that event based middleware has potentially better scaling properties for such applications than object based middleware.“).

Eine weitere Herausforderung könnte darin liegen herauszufinden, welche Daten überhaupt interessant sind, und welche überhaupt keiner weiteren Verarbeitung bedürfen. In „Event Detection Services Using Data Service Middleware in Distributed Sensor Networks“ [7] nennen Li et al. ein Beispiel, in dem man die Semantik mitnutzen kann und so bestimmte Meßdaten nicht in allen Umgebungen Alarme hervorrufen müssen.

Im Kern geht es bei allen Dezentralisierungsbestrebungen darum, daß Sensorknoten (lokal) zusammenarbeiten und so die Qualität der Daten verbessern, das Netz weniger fehleranfällig wird, die Ressourcen besser genutzt werden und vor allem die Skalierbarkeit deutlich erhöht wird, siehe Zhao et al. in „Information-driven dynamic sensor collaboration“ [20] und Hellerstein et al. in „Adaptive Query Processing: Technology in Evolution“ [22].

4 Implementierung der Ansätze/konkrete Protokolle

In diesem Abschnitt werde ich versuchen einige verschiedene Implementierungen ein wenig zu beschreiben. Die Systeme habe ich grob zeitlich angeordnet. Zuerst werde ich etwas zum COUGAR Projekt (COUGAR: The Network Is The Database, [6]) schreiben, danach zu SINA (Sensor Information Networking Architecture, [15], [8]), um dann zu Fjords (Framework in Java for Operators on Remote Data Streams, [5]) zu kommen. Als nächstes werde ich auf DSWare (Real-Time Event Detection Service using Data Service Middleware, [7], [8]) und dann auf MiLAN (Middleware Linking Applications and Networks, [23], [8]) eingehen um schließlich mit TinyDB ([16]) und RUNES (Reconfigurable Ubiquitous Networked Embedded Systems, [24]) meine Auflistung zu beenden.

4.1 COUGAR: The Network Is The Database

COUGAR wurde von Philippe Bonnet, Johannes Gehrke und Praveen Seshadri am Computer Science Department der Cornell University entwickelt.

Es setzt auf eine SQL-ähnliche Abfragesprache, wie die meisten anderen Implementierungen auch. Hervorzuheben ist bei COUGAR, daß auf jedem Knoten eine vereinfachte Version des Programms, das Anfragen abarbeitet läuft, das einfache Signalverarbeitungsfunktionen ausführt und das Ergebnis zurücksendet. Der Vorteil an dieser Architektur ist, daß sie auch in großen Sensornetzen problemlos skaliert und damit auch für den Einsatz in derartigen Umgebungen geeignet ist.

Siehe auch Bonnet et al., „Towards Sensor Database Systems“ [6].

Beispielhaft wurde COUGAR mit Hilfe von Mica Motes (siehe [25]) in einem Praxistest erprobt.

4.2 SINA (Sensor Information Network Architecture)

SINA wurde von Chien-Chung Shen an der University of Delaware entworfen.

Bei diesem System wird die Position der Sensoren in den Vordergrund gestellt. Auch das hierarchische Clustering richtet sich nach der Umgebung und der Leistung der Knoten. In jedem Cluster wird ein sogenannter „cluster head“ ausgewählt, wie der Clustering-Algorithmus genau funktioniert kann man Estrin et al. „Embedding the Internet“ [26] entnehmen.

Die „cluster heads“ dienen auch als Caches und halten beispielsweise aggregierte Daten der Knoten, die im selben Cluster sind, vor. Der Vorteil besteht darin, daß bei Anfragen die Antwortzeit dadurch verkürzt wird, daß nur noch die „cluster heads“ abgefragt werden müssen, jedoch erkauft man sich diesen Zeitvorteil durch ungenauere Ergebnisse, denn die einzelnen Knoten innerhalb eines Clusters aktualisieren die Werte des „cluster heads“ nicht kontinuierlich sondern periodisch.

Die gesammelten Daten werden bei SINA zu einem inhärenten Teil des jeweiligen Sensorknotens. Wie auch bei COUGAR und MiLAN läuft bei SINA auf jedem Sensorknoten eine Ausführungsumgebung, die ankommende Nachrichten abarbeitet, diese nennt sich abgekürzt SEE („sensor execution environment“). Die Nachrichten bestehen aus zwei Teilen. Den eigentlichen Inhalt bildet eine Anfrage, die in der Sensor Query and Tasking Language (SCTL) geschrieben wird. Dies ist eine prozedurale Skriptsprache, deren Syntax von SQL abgeleitet wurde, näheres zu SCTL wird in „Querying and Tasking in Sensor Networks“ von Jaikaeo et al. [27] beschrieben. Es ist auch möglich in SCTL mit einigen Ereignissen zu arbeiten. Dieser Inhalt wird in einen sogenannten „SCTL wrapper“ (auch „header“ genannt) eingekapselt, der ein XML-Gerüst darstellt.

Bei SINA werden auch die datenzentrierten Eigenschaften von Anfragen in Sensornetzen berücksichtigt, weswegen anstatt expliziter Adressierung ein merkmalsbasierter Ansatz gewählt wurde. Um Netzwerkbandbreite zu sparen wird auf SPIN (siehe Heinzlmann et al., „Adaptive protocols for information dissemination in wireless sensor networks“ [28]) gesetzt.

Der Begriff Frontend-Knoten wird bei SINA anders gebraucht, als ich ihn gebrauche. Bei SINA versteht man darunter einen speziellen Knoten, der direkt an das Netzwerk angeschlossen ist und über den die Anfragen in das Sensornetz verteilt und verarbeitet werden. In anderen Systemen wird diese Komponente oft „Gateway-Knoten“ genannt.

Eine Besonderheit von SINA besteht darin, wie mobile Benutzer und das stationäre Netzwerk miteinander interagieren können. Die verwendete Technik, die gewährleisten soll, daß ein mobiler Benutzer über einen Netzwerkknoten Daten anfragt, sich bewegt und das Ergebnis dann von einem anderen Netzwerkknoten geliefert bekommt nennt sich „progressive footprint chaining“. Als Beispielanwendung wird die Kursverfolgung von Fahrzeugen angeführt.

Als weitere Besonderheit werden unterschiedliche Methoden Informationen zu sammeln auf unterschiedlichen Ebenen innerhalb einer Anwendung genutzt um die Gesamtleistung zu verbessern.

In „RUNES: Survey of Middleware for Networked Embedded Systems“ [8] wird von Mascolo et al. jedoch kritisch bemerkt, daß es bislang noch keine Implementierung dieser Architektur gibt.

Siehe auch Shen et al., „Sensor Information Networking Architecture and Applications“ [15].

4.3 Fjords (Framework in Java for Operators on Remote Data Streams)

Fjords wurde von Samuel Madden und Michael J. Franklin an der University of California entwickelt.

Bei Fjords wird davon ausgegangen, daß Sensoren ununterbrochen Daten liefern, also einen Datenstrom erzeugen. Weil die Datenströme unendlich sind, laufen auch die Anfragen in der Regel immer weiter. Die Sensorknoten mit ihren beschränkten Ressourcen übernehmen hier keine Verarbeitungsaufgaben sondern sammeln lediglich Daten, fassen diese zu Paketen zusammen und verschicken sie dann, gegebenenfalls über sogenannte „sensor’s proxies“ an einen verarbeitenden Knoten, in der Regel ein gut angebundener Server. Sensor Proxies spielen bei Fjords eine wichtige Rolle, denn über sie kann die „Abtastrate“ der Sensoren an die Nachfrage nach diesen Daten angepaßt werden, was viel Energie einsparen kann. Außerdem können über die Sensor Proxies die Programme auf den Sensorknoten auch komplett ausgewechselt werden. Sensorknoten werden bei Fjords als objektrelationale Tabellen aufgefaßt, es wird die Implementierung benutzt, die auch für das Cougar Projekt in „Towards Sensor Database Systems“ von Bonnet et al. [6] vorgeschlagen wird. Wie in anderen Ansätzen (z. B. Bonnet et al., „Towards Sensor Database Systems“ [6]) ist auch hier eine Kombination von Sensordaten mit „statischen Quellen“ möglich.

Fjords kombiniert Proxies mit sich nicht gegenseitig blockierenden Operatoren auf Datenströmen und herkömmlichen Anfrageplänen. Das Anwendungsbeispiel zur Überwachung des Straßenverkehrs mit dem Berkely Highway Lab (BHL) habe ich Madden und Franklin, „Fjording the Stream: An Architecture for Queries

Over Streaming Sensor Data“ [5] entnommen, denn dort wird es als Testumgebung aufgeführt, anhand dessen gezeigt wird, daß es mit Hilfe der Fjords Architektur möglich ist viele Anfragen gleichzeitig abzuarbeiten. Im Gegensatz zu den meisten anderen Ansätzen wird bei Fjords keine spezielle Anfragesprache eingesetzt, denn die Konzepte sind auf die meisten Sprachen übertragbar. Siehe auch Madden und Franklin, „Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data“ [5].

4.4 DSWare (Real-Time Event Detection Service using Data Service Middleware)

DSWare wurde von Shuoqi Li, Ying Lin, Sang H. Son, John A. Stankovic und Yuan Wei am Department of Computer Science an der University of Virginia entwickelt.

Es handelt sich hierbei um eine flexible datenzentrierte Middleware, die die Konzepte eines Speicherdienstes aus „Data-centric storage in sensor networks“ von Shenker et al. [29] implementiert. Es besteht die Möglichkeit miteinander in Beziehung stehende Daten örtlich benachbart zu speichern. Die „Speicherhierarchie“ ist von der physischen Netzwerkierarchie entkoppelt, so können im Speicher-Overlay-Netzwerk mehrere physikalische Sensorknoten dem selben logischen Knoten zugeordnet werden. Im Gegensatz zu anderen Middlewares ist bei DSWare das Routing ausgelagert. Dadurch, daß bei DSWare Daten redundant gespeichert werden können, wirken sich Ausfälle einzelner Knoten solange nicht aus, wie irgendein Sensor die angeforderten Daten liefern kann (siehe Heinzlmann et al., „Middleware to support sensor network applications“ [23]). Die Replikation von Daten koordinieren die Knoten selbständig, so daß sie ihre Daten dann kopieren, wenn sie nicht ausgelastet sind. Der Vorteil mehrere Kopien von Daten im Sensornetz „gecached“, also redundant, vorliegen zu haben, die am häufigsten angefragt werden wird eventuell durch den Wartungsaufwand für die Kopien wieder aufgehoben. Hier muß man den Zielkonflikt zwischen schneller Antwortzeit und erhöhtem Wartungsaufwand betrachten. Für jegliche Art der örtlichen Zusammenarbeit verschiedener Knoten ist die Gruppen Management Komponente verantwortlich, die diese Funktionalität bereitstellt. Als typische Anfragen werden sogenannte „data subscription queries“ aufgeführt, deren praktische Relevanz an einem Beispiel illustriert wird, an das das Anwendungsbeispiel der Überwachung des Straßenverkehrs angelehnt ist. Außerdem bilden Ereignisse die Grundlage für DSWare. Um Ereignisse zu registrieren oder zu löschen wird auch hier eine SQL-ähnliche Syntax benutzt, wobei die Entscheidung bewußt für eine SQL-ähnliche Syntax gefällt wurde, obgleich dies einen gewissen Parsingoverhead mit sich bringt. Hervorzuheben ist bei DSWare, daß es eine Ereignis-Hierarchie gibt, so daß es neben atomaren Ereignissen auch zusammengesetzte Ereignisse gibt. Interessant ist auch der Ansatz, den ich weiter oben bei den allgemeinen Ideen schon kurz erwähnt habe, daß bei DSWare auch die Semantik von Daten mitbenutzt werden kann, um zum Beispiel Fehlalarme zu reduzieren. Eine Ereigniserkennung, die der Bedeutung von Kontexten Rechnung trägt, kann viel Energie sparen, was in Sensornetzen ein wichtiges Argument ist.

In den Abbildungen 4 und 5 in „Event Detection Services Using Data Service Middleware in Distributed Sensor Networks“ [7] zeigen Li et al. auch, daß die Reaktionszeit und die Netzwerkbelastung durch die Nutzung von DSWare relativ gering ist.

Durch seine Architektur bietet DSWare die Möglichkeit Daten verlässlich zu speichern, um die Echtzeit-Performance und die Verlässlichkeit von aggregierten Ergebnissen zu verbessern, außerdem wird der Kommunikationsoverhead verringert. So können, wie Mascolo et al. in „RUNES: Survey of Middleware for Networked Embedded Systems“ [8] beschreiben Sensornetze von Anwendungen über fast genau die selben Schnittstellen benutzt werden wie gewöhnliche Datenbanken.

Siehe auch Li et al., „Event Detection Services Using Data Service Middleware in Distributed Sensor Networks“ [7].

4.5 MiLAN (Middleware Linking Applications And Networks)

MiLAN wurde von Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho und Mark A. Perillo vom Center for Future Health, Dept. of Electrical and Computer Engr. und Computer Science Department der University of Rochester entwickelt.

Sie entwarfen eine adaptive Middleware, bei der wie auch bei COUGAR auf jedem Sensorknoten eine Version von MiLAN läuft (Mascolo et al., „RUNES: Survey of Middleware for Networked Embedded Systems“ [8]). MiLAN wird auch als proaktives Middleware-System bezeichnet, weil alle Sensorknoten in die Verarbeitung miteinbezogen werden. Wie Weis in „Adaptive Systeme“ [30] so treffend formulierte koordiniert MiLAN die empfangenen Informationen und optimiert die Abläufe beziehungsweise passt die Charakteristik des Sensornetzes entsprechend an. Dies wird dadurch begünstigt, daß MiLANs Architektur bis in den Netzwerkprotokollstack hineinreicht, was für eine Middleware eher unüblich ist.

Siehe auch Heinzelmann et al., „Middleware to support sensor network applications“ [23].

4.6 TinyDB

TinyDB wurde von Samuel Madden vom Massachusetts Institute of Technology, Michael Franklin und Joseph Hellerstein von der University of California und Wei Hong von Intel Research entworfen.

Bei TinyDB handelt es sich um eine verteilte Ausführungsumgebung für Anfragen in einem Sensornetz, die auf jedem Sensorknoten läuft. TinyDB baut auf TinyOS auf. TinyDB geht über simples Verarbeiten im Netzwerk hinaus und bietet mit ACQP („acquisitional query processing“) weitergehende Möglichkeiten für Anfragen in Sensornetzen. Anfragen gelangen über einen gut ausgestatteten PC, die sogenannte basestation, in das Sensornetz. Dieser PC parst die Anfrage, optimiert sie und sendet sie dann in das Sensornetz. Die „basestation“ bildet die Wurzel des Routing-Baumes, die Sensoren senden ihre Ergebnisse einfach auf

dem entgegengesetzten Weg zurück, auf dem sie die Anfrage erhalten haben. Hervorzuheben ist bei TinyDB die Nutzung von semantischen Routing Bäumen („semantic routing tree“, SRT), bei denen auch semantische Eigenschaften beim Aufbau des Baumes berücksichtigt werden.

Nach den Erfahrungen, die Madden et al. in „TinyDB: an acquisitional query processing system for sensor networks“ [16] beschrieben haben ist der in TinyDB gewählte geclusterte SRT-Ansatz anderen SRT-Algorithmen überlegen.

SRTs können die Anzahl der Knoten, die in die Verarbeitung einer Anfrage mit einbezogen werden reduzieren. Ein Nachteil von SRTs liegt in den hohen Kosten die entstehen, wenn Sensorknoten ihre übergeordneten Knoten wechseln. Die Kosten die entstehen, um einen SRT aufzubauen sind vergleichbar mit den Kosten anderer Verfahren.

Wie schon in anderen Systemen kommt auch bei TinyDB eine SQL-ähnliche Anfragesprache zur Anwendung.

Vier Fragestellungen spielen für die Verarbeitung von Anfragen eine Rolle:

- (1) Wann sollen Daten für eine bestimmte Anfrage gemessen und weitergeleitet werden?
- (2) Welche Sensorknoten haben überhaupt relevante Daten für eine bestimmte Anfrage?
- (3) In welcher Reihenfolge sollen verschiedene Daten gesammelt und kombiniert werden und wie sollen sie mit anderen Operationen verschachtelt werden?
- (4) Ist es überhaupt sinnvoll Rechenleistung und Bandbreite für bestimmte Daten zu „verschwenden“?

Zur Leistungssteigerung können sogenannte „materialization points“ angelegt werden, das sind kleine Puffer die Ergebnisse zwischenspeichern, die in anderen Anfragen genutzt werden können. Mit dieser Technik wird auch die Funktionalität implementiert, daß gesammelte Daten zwischengespeichert werden, wenn ein Sensorknoten keine Verbindung zum Netz hat. Bei Aggregationsanfragen werden die Daten bereits im Netz verarbeitet während sie den Routing Baum „nach oben“ fließen.

Im Bereich der Ereignisse ist TinyDB erst am Anfang der Möglichkeiten, denn bisher werden Ereignisse nur lokal gemeldet, aber das Potential ereignis-basierter Anfragen, die Energie sparen können ist erkannt.

TinyDB überwacht auch die Netzauslastung und reduziert die Anzahl der Pakete, wenn die Auslastung steigt, um das Netzwerk nicht zu überlasten und optimal zu nutzen.

Bislang gibt es leider noch keine Benchmarks für Sensornetzwerk-Datenbanken, aber TinyDB kann die Ergebnisse, die dem Benutzer geliefert werden erheblich verbessern. Bei TinyDB wird viel Wert auf die oben erwähnten Fragestellungen gelegt, wann, wo und in welcher Reihenfolge welche Daten gesammelt und welche Sensorknoten miteinbezogen werden sollen. Diese Fragestellungen wurden in der Literatur bisher nicht ausführlich behandelt.

Siehe auch Madden et al., „TinyDB: an acquisitional query processing system for sensor networks“ [16].

Wie COUGAR läuft auch TinyDB auf der Mote-Plattform (siehe [25]), die in Berkeley entwickelt wurde.

4.7 RUNES (Reconfigurable Ubiquitous Networked Embedded Systems)

RUNES wurde von Paolo Costa und Gian Pietro Picco von der Politecnico di Milano, Geoff Coulson von der Lancaster University und Cecilia Mascolo und Stefanos Zachariadis vom University College London entworfen.

Hervorzuheben an RUNES ist die Möglichkeit, daß sich das System dynamisch an sich verändernde Umstände anpassen kann, was in Sensornetzen eine wichtige Eigenschaft ist. Wie MiLAN reicht auch RUNES bis in die Netzwerkebene hinein, was auf RUNES aufbauenden Anwendungen eine weitere Abstraktionsschicht zur Verfügung stellt.

RUNES wurde als generische Software entwickelt, die auf vielen Geräten implementiert werden kann. Wichtig bei RUNES sind die Dienste, die die verteilte Koordination regeln. Wie bei den meisten anderen Systemen setzt auch RUNES auf Clustering.

Siehe auch Costa et al., „The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems“ [24] und Mascolo et al., „RUNES: Survey of Middleware for Networked Embedded Systems“ [8].

Das System wurde auf der IST2006 (Information Society Technologies) demonstriert (siehe [31]).

4.8 Zusammenfassung in einer Tabelle

	COUGAR	SINA	Fjords	DSWare	MILAN	TinyDB	RUNES
Universitäten	Cornell University Comp. Science Department	University of Delaware	University of California	University of Virginia Department of Comp. Science	University of Rochester	MIT University of California Intel Research	Politecnico di Milano Lancaster University University College London
Querying (Abfragesprache)	SQL-ähnlich	SQL-ähnlich	beliebig	SQL-ähnlich	-	SQL-ähnlich	beliebig
Clustering (Hierarchiebildung)	(keine Aussage)	basierend auf Nähe und Leistung der Sensorknoten [26]	ja	Storage Overlay Network	basierend auf Nähe und Leistung der Sensorknoten	Semantik wird berücksichtigt (semantic routing trees, SRT)	ja
Ereignisse	-	es gibt einige Ereignisse	-	Ereignishierarchie (atomare und zusammen- gesetzte Ereignisse)	-	nur lokale Ereignisse (direkt am Sensor)	flexible Event Notification Komponente
Beispiele	COUGAR Projekt, Mica Motes	Kursverfolgung („progressive footprint chaining“)	Berkely Highway Lab (BHL)	Überwachung des Straßen- verkehrs		Berkely- Mote-Plattform	Demon- stration auf IST2006 [31]

5 Fazit

Die grundsätzlichen Lösungsansätze sind vielversprechend. Richtig eingesetzt besteht die Möglichkeit, daß sie das Überlastungsproblem zwischen Frontend-Komponenten und Backend-System in ubiquitären Informationsumgebungen tatsächlich beseitigen oder zumindest deutlich abschwächen können.

Bei den konkreten Ansätzen kommt es stark auf den Anwendungsfall an, welches System besser geeignet ist die vorhandenen Ressourcen optimal zu nutzen.

Generell scheinen z. B. DSWare und TinyDB die interessantesten hier aufgelisteten Systeme zu sein.

DSWare besticht durch seine Ereignishierarchie, bei TinyDB sind die semantischen Routing Bäume noch einmal besonders hervorzuheben.

Wie man den zahlreichen Literaturverweisen entnehmen kann wird auf diesem Gebiet zur Zeit viel geforscht und es gibt viele unterschiedliche Systeme und Ansätze. Welche Ansätze sich längerfristig durchsetzen werden kann man heute kaum abschätzen, vor allem kann es kaum das eine optimale System geben, da dafür die Anwendungsfelder zu vielseitig sind.

Literatur

1. Weiser, M.: The computer for the 21st century. *Human-computer interaction: toward the year 2000* (1995) 933–940
2. Endres, C., Butz, A., MacWilliams, A.: A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems Journal* **1**(1) (2005) 41–80
3. Beigl, D., Kubach, E.: Digiclip: Activating physical documents (2004)
4. Haller, S., Nochta, Z.: Kooperation zwischen intelligenten gütern. *Industrie Management - Zeitschrift für industrielle Geschäftsprozesse* **22**(3) (2006) 39–41
5. Madden, S., Franklin, M.J.: Fjording the stream: An architecture for queries over streaming sensor data. In: *ICDE*. (2002)
6. Bonnet, P., Gehrke, J., Seshadri, P.: Towards sensor database systems. *Lecture Notes in Computer Science* **1987** (2001) 3–??
7. Li, S., Son, S., Stankovic, J.: Event detection services using data service middleware in distributed sensor networks (2003)
8. C. Mascolo, S. Hailes; L. Lymberopoulos; G. Picco; P. Costa; G. Blair; P. Okanda; T. Sivaharan, W.F.M.K.M.R.K.F., Boulis, A.: Runes: Survey of middleware for networked embedded systems (2005)
9. Govindan, R., Hellerstein, J., Hong, W., Madden, S., Franklin, M., Shenker, S.: The sensor network as a database (2002)
10. Estrin, D., Govindan, R., Heidemann, J.S., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In: *Mobile Computing and Networking*. (1999) 263–270
11. Bonnet, P., Gehrke, J., Seshadri, P.: Querying the physical world (2000)
12. Beutel, J., Kasten, O., Mattern, F., Römer, K., Siegemund, F., Thiele, L.: Prototyping wireless sensor network applications with BTnodes. In: *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*. Volume 2920 of *Lecture Notes in Computer Science*, Springer, Berlin (2004) 323–338

13. Davies, N., Wade, S., Friday, A., Blair, G.: Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications. In: Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97), Toronto, Canada (1997) 291–302
14. Costa, P., Mottola, L., Murphy, A.L., Picco, G.P.: TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In: Proceedings of the 1st International Workshop on Middleware for Wireless Sensor Networks (MidSens 2006), Melbourne, Australia, ACM Press (2006)
15. Shen, C.C., Srisathapornphat, C., Jaikaeo, C.: Sensor Information Networking Architecture and Applications. *IEEE Personal Communication Magazine* **8**(4) (2001) 52–59
16. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.* **30**(1) (2005) 122–173
17. Thaler, D., Ravishankar, C.V.: Distributed top-down hierarchy construction. In: *INFOCOM* (2). (1998) 693–701
18. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System architecture directions for networked sensors. In: *Architectural Support for Programming Languages and Operating Systems*. (2000) 93–104
19. Beigl, M., Krohn, A., Zimmer, T., Decker, C., Robinson, P.: Awarecon: Situation aware context communication (2003)
20. Zhao, F., Shin, J., Reich, J.: Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine* **19**(2) (2002) 61–72
21. Anjum Shehzad, Hung N. Q., K.A.P.M., Riaz, M., Liaquat, S., Lee, S., Lee, Y.K.: Middleware infrastructure for context-aware ubiquitous computing systems. ... (2005)
22. Hellerstein, J.M., Franklin, M.J., Chandrasekaran, S., Deshpande, A., Hildrum, K., Madden, S., Raman, V., Shah, M.A.: Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin* **23**(2) (2000) 7–18
23. Heinzelman, W., Murphy, A., Carvalho, H., Perillo, M.: Middleware to support sensor network applications (2004)
24. Costa, P., Coulson, G., Mascolo, C., Picco, G.P., Zachariadis, S.: The runes middleware: A reconfigurable component-based approach to networked embedded systems. In: Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'05), Berlin (Germany) (2005)
25. Ganapathi Kamath: Mica motes. <http://web.syr.edu/~gkamatth/topics/micamote.html> (2003)
26. Estrin, D., Govindan, R., Heidemann, J.: Embedding the Internet. *Communications of the ACM* **43**(5) (2000) 39–41 (special issue guest editors).
27. Jaikaeo, C., Srisathapornphat, C., Shen, C.C.: Querying and Tasking in Sensor Networks. In: SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V), Orlando, Florida (2000)
28. Heinzelman, W.R., Kulik, J., Balakrishnan, H.: Adaptive protocols for information dissemination in wireless sensor networks. In: *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, New York, NY, USA, ACM Press (1999) 174–185
29. Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., Estrin, D.: Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.* **33**(1) (2003) 137–142

30. Weis, C.: Adaptive systeme. In: Seminar Verteilte Informationssysteme - Datenverarbeitung in Sensornetzen, Dr. Serge Shumilov, Universität Bonn, Institut für Informatik III (2006)
31. RUNES Consortium: Runes demonstration @ ist2006. <http://www.ist-runes.org/news2006-11-23.html> (2006)