Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

εSOA - SOA für eingebettete Netze

Andreas Scholz, Christian Buckl, Stephan Sommer, Alfons Kemper, Alois Knoll, Jörg Heuer und Anton Schmitt

8 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

εSOA - SOA für eingebettete Netze

Andreas Scholz[1], Christian Buckl[1], Stephan Sommer[1], Alfons Kemper[1], Alois Knoll[1], Jörg Heuer[2] und Anton Schmitt[2]

[1]Fakultät für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching, Germany
{scholza,buckl,sommerst,knoll,kemper}@in.tum.de

[2]Corporate Technology, Networks and Multimedia Communication, Siemens AG
D-81730 München, Germany
{joerg.heuer,anton.schmitt}@siemens.com

Abstract: Eingebettete Netze, bestehend aus einer Vielzahl von vernetzten Knoten mit unterschiedlichen Mess-, Steuer- und Rechenfähigkeiten, werden zunehmend in Bereichen wie der Heim- und Industrieautomatisierung, modernen Kraftfahrzeugen oder großflächigen Infrastrukturen z.B. in Energienetzen oder Verkehrsleitsystemen eingesetzt. Die Komplexität der Netze, die Heterogenität der enthaltenen Knoten und Infrastrukturänderungen durch mobile Knoten oder Knotenausfälle stellen besondere Herausforderungen während der Anwendungsentwicklung dar. Viele dieser Anforderungen können durch die aus dem Software-Bereich wohlbekannten Service orientierten Architekturen (SOAs) erfüllt werden. Um die besonderen Rahmenbedingungen von eingebetteten Netzen (Ressourcenbeschränkungen, Echtzeitanforderungen, etc) berücksichtigen zu können, sind einige Anpassungen des traditionellen SOA Paradigmas aus dem Web Service Kontext nötig. Ziel des εSOA-Projekts ist die Definition eines embedded-SOA-(εSOA)-Konzepts, das diesen Anforderungen genügt, und die Entwicklung einer Middleware für eingebettete Netze und den dazugehörigen Entwicklungswerkzeugen, welche eine effiziente Anwendungsentwicklung ermöglichen.

Keywords: SOA, eingebettete Netze, Middleware

1 Einleitung

Eingebettete Netze, bestehend aus einer Vielzahl von vernetzten Knoten mit unterschiedlichen Mess-, Steuer- und Rechenfähigkeiten, werden zunehmend in Bereichen wie der Heim- und Industrieautomatisierung, modernen Kraftfahrzeugen oder großflächigen Infrastrukturen z.B. in Energienetzen oder Verkehrsleitsystemen eingesetzt. Während der Anwendungsentwicklung treten Herausforderungen durch die besonderen Eigenschaften eingebetteter Netze auf. Die Heterogenität der verwendeten Hardware erfordert eine Abstraktionsschicht, die eine einheitliche Programmierung der Netze erlaubt, zugleich aber vorhandene Ressourcen möglichst effizient ausnutzen kann. Außerdem ist eine hohe Wiederverwendbarkeit von Softwarekomponenten erstrebenswert, um die Entwicklungskosten und -dauer zu minimieren. Änderungen zur Laufzeit können sowohl bei der verwendeten Hardware (Ausfälle, Hinzufügen neuer Knoten), als auch

der Software (Installation neuer Anwendungen, Rekonfiguration) auftreten. Um eine Programmierung durch den Endnutzer zu ermöglichen (z.B. im Bereich der Heimautomatisierung), sind Konzepte notwendig, die es erlauben Anwendungen ohne detaillierte Kenntnisse über die technischen Details der Hardware aufgrund von Domänenwissen zu entwickeln. Im Bereich der Industrieautomatisierung ist eine nahtlose Integration zwischen eingebetteten Netzen, die zur Automatisierung der Produktion verwendet werden, und den Geschäftsprozessen, die diese steuern, notwendig, um zeitnah auf Änderungen reagieren zu können. Diese Anforderungen können von einer dienstorientierten Middleware erfüllt werden, welche in den folgenden Abschnitten vorgestellt wird.

2 εSOA

Unsere εSOA-Plattform verfolgt einen dienstorientierten Ansatz bei der Anwendungsentwicklung. Die momentan am weitesten verbreitete Technik zur Realisierung von dienstorientierten Architekturen stellen Web Services dar. Viele der aus der Web-Service-Welt bekannten Vorteile dienstorientierter Architekturen lassen sich auch für eingebettete Systeme ausnutzen. Die Zerlegung in feingranulare Dienste erhöht die Wiederverwendbarkeit und erlaubt es Entwicklungskosten zu reduzieren. Da jeder dieser Dienste eine geschlossene Funktionseinheit darstellt, unterstützen SOAs implizit auch die verteilte Ausführung von Anwendungen. Diese ist gerade in Hinblick auf die Vermeidung von Überlasten auf einzelnen Knoten und Netzwerkverbindungen zwischen den Knoten entscheidend. Durch die Abstraktion der vorhandenen Hardware in Dienste kann außerdem die Integration von Komponenten verschiedener Hersteller deutlich vereinfacht werden. Bei der Anwendungsentwicklung muss nicht mehr bestimmte Hardware eines bestimmten Herstellers vorausgesetzt werden, sondern alle Produkte, welche vergleichbare Mess-, Verarbeitungs- oder Steuerdienste anbieten, können alternativ eingesetzt werden.

Die im Web-Service Umfeld verwendeten Technologien und Ansätze sind allerdings nicht direkt auf eingebettete Systeme übertragbar. Anwendungen für eingebettete Systeme sind typischerweise datengetrieben: Messdaten werden periodisch oder auf Grund von Umweltänderungen von den Sensoren erzeugt und von einer Kette nachfolgender Dienste verarbeitet. Im Gegensatz zu dem Request/Response Interaktionsmuster das aus der Web-Service-Welt bekannt ist, kann dieses Verhalten effizient durch eine Push-basierte Datenverarbeitung umgesetzt werden. Des Weiteren ist das aus der Web-Service-Welt bekannte Nachrichtenformat SOAP für leistungsschwache Geräte nicht direkt einsetzbar. Die großen Datenmengen, die durch die Verwendung von XML entstehen, stellen sowohl bei der Übertragung, insbesondere über funkbasierte Medien, als auch der Verarbeitung auf den Knoten eine oft schwer überwindbare Hürde dar. Eine wichtige Anforderung ist daher die Verwendung eines effizienten Nachrichtenformates, welches aber kompatibel zu SOAP sein muss, um eine leichte Integration des eingebetteten Netzes mit externen Diensten zu ermöglichen. Einen weiteren Unterscheid stellt die Lebensdauer von Diensten dar. Während im Web-Service-Umfeld Dienstinstanzen typischerweise kurze Lebenszyklen von wenigen Minuten bis wenigen Stunden oder Tagen haben, besitzen Anwendungen im Bereich eingebetteter Netze oft eine Lebensdauer von mehreren Jahren. Dienste, welche die besonderen Anforderungen eingebetteter Netze erfüllen, werden im Folgenden als εServices bezeichnet, die dazugehörige dienstorientierte Architektur als εSOA.

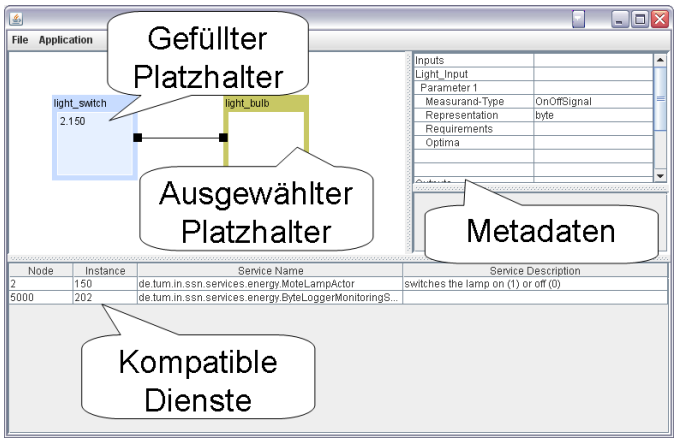

Abbildung 1: Anwendungsinstallation über Muster

Jeder εService besitzt eine Reihe von Ein- und Ausgängen, welche Datenströme konsumieren bzw. produzieren. Dienste können in zwei Klassen unterteilt werden: *Hardware-Dienste*, welche die Funktionalität vorhandener Hardware zugänglich machen, und *Software-Dienste*, welche die Anwendungslogik enthalten. Anwendungen werden durch die Verschaltung mehrerer Dienste erzeugt. Die Schnittstellen der Dienste werden durch eine Metadatenbeschreibung genauer spezifiziert. Diese erlaubt es neben den syntaktischen Eigenschaften der Ein- und Ausgänge, d.h. verwendeten Datentypen, Messeinheiten, etc., auch semantische Eigenschaften zu definieren. Darunter fällt z.B. die Angabe welche Art von Daten aus Sicht der Anwendungsdomäne von einem Ausgang produziert wird. Die Beschreibung basiert dabei auf einer domänenspezifischen Taxonomie, welche es erlaubt Daten auf verschiedenen Abstraktionsebenen zu beschreiben. So kann ein Dienst, der eine Konvertierung von Temperaturwerten zwischen Fahrenheit und Celsius vornimmt mit dem abstrakten Typ „Temperatur“ arbeiten, während Sensoren mit den konkreteren Untertypen Luft- und Wassertemperatur genauer spezifizieren können welche Art von Temperatur gemessen wird. Entscheidend ist, dass durch die bestehende Vererbungsbeziehung zwischen dem Obertyp Temperatur und den Untertypen Luft- und Wassertemperatur festgestellt werden kann, dass der Konvertierungsdienst mit den Sensoren kompatibel ist. Um gerade das Management größerer Installationen zu erleichtern, können die Metainformationen zur Laufzeit erweitert werden, z.B. um Informationen über den Ort der Installation (Raumnummer, Stockwerk, o.ä.), die Konfiguration der Sensoren, logistische Daten (Inventarnummern), etc. Diese Informationen können im Anschluss genutzt werden, um die vorhandene Vielzahl von Diensten zu filtern.

Um Dienstverschaltungen (semi-)automatisch erzeugen zu können und um ungeschulten Benutzern die Installation neuer Anwendungen zu erleichtern, werden in der εSOA-Plattform Anwendungsmuster eingesetzt. Ein Anwendungsmuster definiert die benötigten Dienste und deren Verschaltung für eine spezifische Anwendung. Die benötigten Dienste werden mit Hilfe einer Metadatenbeschreibung spezifiziert, d.h. ein Muster definiert die Eigenschaften, die ein Dienst erfüllen muss (z.B. Anzahl und Art der Ein- und Ausgänge, Datentypen, etc), um an der entsprechenden Stelle im Muster eingesetzt werden zu können. Ein Anwendungsmuster könnte also z.B. fordern, dass ein Dienst einen Ausgang mit dem Datentyp „Integer“ besitzt, der „Temperatur“-Werte misst, um einen Temperatursensor zu spezifizieren. Der Benutzer muss bei der Installa-

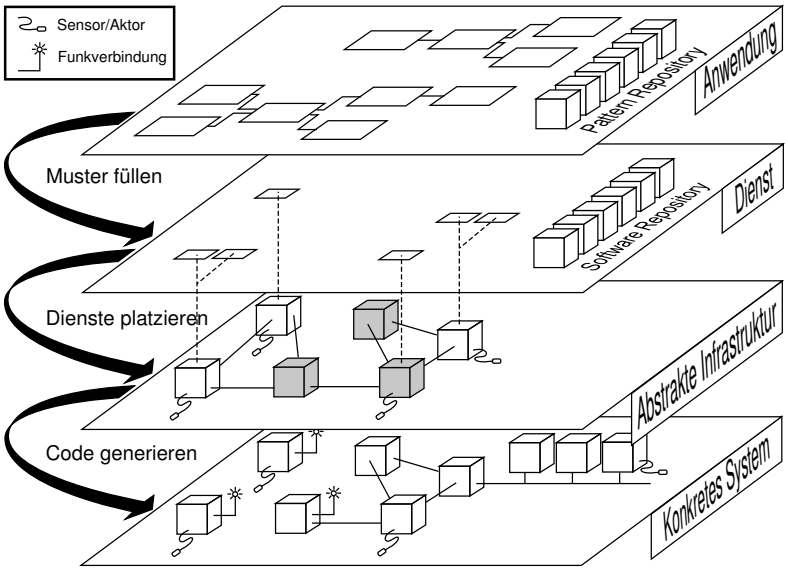

Abbildung 2: Abstraktionsebenen für eingebettete Netze

tion von Anwendungen lediglich die vorhandene Hard- und Software den Platzhaltern zuweisen, was durch Werkzeuge auch teilweise automatisiert werden kann. Bei dieser Zuweisung spielen Metadaten eine entscheidende Rolle. Durch die syntaktischen Daten wird sichergestellt, dass nur sinnvolle Verschaltungen vom Benutzer erstellt werden können, d.h. die verwendeten Dienste die versendeten Daten auch korrekt interpretieren können. Sind mehrere Dienste mit einem Platzhalter kompatibel, können die semantischen Daten herangezogen werden um eine sinnvolle Vorauswahl oder Sortierung zu treffen. So können dem Nutzer bei der Installation einer Lichtsteuerung z.B. bevorzugt Lichtschalter empfohlen werden, die sich im gleichen Raum wie die Lampe befinden, d.h. das gleiche Metadaten-Attribut „Raum“ besitzen. Eine prototypische Implementierung eines entsprechenden Tools ist in Abbildung 1 zu sehen. Durch die Abstraktion von konkreten Diensten, die in den Anwendungsmustern erzielt wird, kann eine hohe Wiederverwendbarkeit für verschiedenste Hardwarekonfigurationen erzielt werden. Ein interessanter Aspekt ist, dass diese Muster nicht vom Endnutzer selbst erstellt worden sein müssen, sondern z.B. aus dem Internet bezogen werden können. Eine mögliche Vision ist eine internetbasierte Plattform, die es Endnutzern erlaubt selbsterstellte Muster und Dienste zu hinterlegen. Nach dem Kauf neuer Hardware kann der Endanwender diese Datenbasis nach Anwendungen durchsuchen, die mit der aktuell vorhandenen Hardware realisiert werden können. Alternativ kann für den Endnutzer an Hand einer gewünschten Anwendung und einer Herstellerdatenbank eine „Einkaufsliste“ mit der benötigten Hardware erstellt werden.

3 Anwendungsentwicklung

Abbildung 2 zeigt einen Überblick über die Abstraktionsebenen bei der Anwendungsentwicklung. Die Installation einer Anwendung erfolgt in folgenden Schritten: Ein abstraktes Muster auf der *Anwendungs*-Ebene wird mit Diensten aus der *Dienst*-Ebene gefüllt. Dieser Schritt kann,

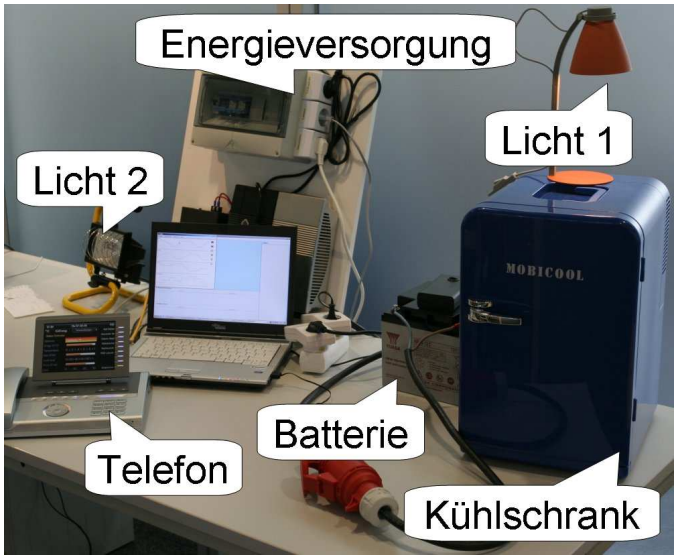

Abbildung 3: Demonstrator: Energiemanagement

wie im vorherigen Abschnitt beschrieben, teilweise automatisiert werden, erfordert im Allgemeinen aber eine Interaktion mit dem Endnutzer. Die nachfolgenden Schritte können vollautomatisch von der verwendeten Middleware durchgeführt werden. Zunächst wird mit den Informationen aus der *Infrastruktur*-Ebene eine möglichst effiziente Zuordnung von Diensten auf die verfügbaren Knoten ermittelt und die Generierung von plattform-spezifischem Code angestoßen. Die Infrastruktur-Ebene stellt ein eingebettetes Netz als eine Menge vernetzter Knoten dar. Die spezifischen Eigenschaften der verwendeten Hardware sind in Form von Knoten- oder Verbindungseigenschaften verfügbar, z.B. der verfügbare Speicher auf einem Knoten oder die Bandbreite und Latenz einer Verbindung. Nach der Installation der Dienste auf den Knoten werden diese konfiguriert und die Ausführung der Anwendung gestartet.

Durch eine geschickte Platzierung der Dienste können die entstehenden Kommunikationskosten verringert werden, z.B. durch die Platzierung der datenverarbeitenden Dienste nahe bei den datenproduzierenden Sensoren. Durch die plattform-spezifische Codegenerierung ist auch für Geräte mit eingeschränkten Ressourcen eine effiziente Implementierung gewährleistet [3]. Zum einen kann die Funktionalität der εSOA-Middleware durch das Hinzufügen oder Weglassen von Komponenten gezielt für den jeweiligen Knoten angepasst werden. Zum anderen kann die Nachrichtenverarbeitung auf den Knoten an die verwendeten Dienste angepasst werden. Basierend auf der Metadatenbeschreibung der Dienste kann bei der Codegenerierung vollautomatisch entschieden werden, ob einzelne Knoten die Verwendung von komplexen Datentypen, wie z.B. Zeichenketten, unterstützen müssen oder nicht. Diese Adaptivität erlaubt es komplexe Datentypen auf leistungsfähigen Knoten zu verwenden, ohne einen zusätzlichen Overhead auf leistungsschwachen Geräten zu erzeugen.

4 Implementierung

Basierend auf den genannten Konzepten wurde eine prototypische Implementierung der εSOA-Middleware für ein visionäres Energie/Heimautomatisierungs-Szenario entwickelt. Der Prototyp basiert auf einem Szenario, in dem der Energiepreis abhängig vom Gesamtbedarf von den Stromanbietern dynamisch angepasst wird. Durch eine geschickte Ausnutzung vorhandener Energiespeicher, wie der Batterie eines Elektroautos, und die gezielte Steuerung von Verbrauchern, wie Kühlschränken oder Klimaanlagen, können die Stromkosten für einen Haushalt gesenkt werden. Aus Sicht der Stromanbieter ist der Vorteil dieser Vorgehensweise die Reduzierung von Lastspitzen, was durch die gezielte Erhöhung des Strompreises zu bestimmten Tageszeiten erreicht werden kann. Abbildung 3 zeigt den Aufbau des Demonstrators. Die einzelnen Sensoren und Aktoren sind an ZigBee basierte Motes angeschlossen, welche entsprechende Hardware-Dienste zur Verfügung stellen. Der PC dient zur Erzeugung des aktuellen Strompreises und der Visualisierung verschiedener Messwerte, wie Kühlschranktemperatur, Ladestand der Batterie, etc. Das Telefon kann zur Konfiguration verschiedener Parameter, wie der gewünschten Zieltemperatur des Kühlschrankes oder dem minimalen Akkustand genutzt werden.

5 Verwandte Arbeiten

Im Bereich der reinen Sensornetze gibt es verwandte Projekte, welche die Entwicklung einer Middleware zur effizienten Sammlung, Verarbeitung und Weiterleitung von Daten verfolgen. Beispiele für solche Systeme sind TinyDB[9] und Cougar[11]. Charakterisierend für diese Systeme ist eine hierarchische Netzwerkarchitektur, in der Daten von den Sensoren ausgehend immer weiter verdichtet werden, bis sie einen zentralen Knoten erreichen, der sie speichern oder an externe Systeme versenden kann. Diese Netzwerkstruktur ist für Sensor/Aktor-Netze, mit einer Vielzahl parallel laufender Anwendungen, welche verschiedene Sensoren und Aktoren benötigen, ungeeignet. Die zentralen Knoten führen zu unnötigen Engpässen bei der Datenverarbeitung und können im Fehlerfall nicht einfach umgangen werden.

Ein dienstorientierter Ansatz für die Programmierung von eingebetteten Netzen wird auch von Projekten SIRENA[7] und SOCRADES[5] verfolgt. Im Gegensatz zu unserem Ansatz wird hier keine Unterscheidung zwischen den Diensten auf den eingebetteten Geräten und Web Services getroffen. Stattdessen erhalten die eingebetteten Geräten mit Hilfe eines angepassten Web Service Stacks, dem DPWS[6] Stack, direkten Zugang zur Welt der Web Services. Durch die weiter fallenden Preise und die Verfügbarkeit leistungsstärkerer Hardware ist dieser Ansatz vielversprechend. Im Gegensatz zu den Autoren sind wir aber nicht der Meinung, dass dies für alle Anwendungsfelder zutrifft. Zunehmend billigere Hardware wird auch weitere Einsatzmöglichkeiten für eingebettete Netze eröffnen, welche wiederum aus Kostengründen mit Hardwarebeschränkungen zu kämpfen haben. Wir denken deshalb, dass ein Ansatz, der auf der Generierung plattformspezifischen Codes basiert, breitere Einsatzfelder eröffnet.

Weitere Ansätze zur Entwicklung einer dienstorientierten Middleware stellen die Projekte OASiS[8], MORE[10] und RUNES[4] dar. Diese bieten jedoch keine Unterstützung für eine einfache Konfigurierbarkeit für Endnutzer oder die automatische Verschaltung von Diensten zu Anwendungen.

Für bestimmte Anwendungsgebiete sind standardisierte Middleware-Architekturen verfügbar, wie zum Beispiel KNX[1] im Bereich der Heimautomatisierung oder AUTOSAR[2] im Automotive Bereich. Diese Ansätze arbeiten auf einem sehr niedrigen Abstraktionsniveau und unterstützen dadurch weder die Programmierbarkeit durch Endnutzer, noch eine einfache und nahtlose Integration mit externen Web Services, insbesondere weil das verwendete Paradigma zur Datenverarbeitung nicht direkt kompatibel mit dienstorientierten Ansätzen ist.

6 Zusammenfassung

Die Entwicklung von Sensor-/Aktor-Netzwerken war bisher die Domäne von Experten für eingebettete Systeme. Mit der zunehmenden Verbreitung dieser Systeme steigt der Anspruch auf eine einfache Programmierbarkeit, auch durch ungeschulte Nutzer. Das dienstorientierte Konzept bietet diverse Ansätze um dieses Ziel zu erreichen. In diesem Beitrag wurde ein Ansatz zu einer weitgehenden Unterstützung bzw. teilweisen Automatisierung der Anwendunginstallation und Konfiguration vorgestellt. Basierend auf einer Beschreibung von Sensor-/Aktor-Netzwerken als eine Menge von Diensten können Verschaltungsmuster durch den Endbenutzer ausgefüllt und installiert werden. Um die Anzahl der potentiellen Komponenten für eine Position in einem solchen Muster zu minimieren, werden Metadaten benutzt. Ausgehend von verschiedenen Verschaltungen können Werkzeuge den entsprechenden Code generieren und auf das Netzwerk herunterladen. Dabei wird auch ressourceneingeschränkte Hardware unterstützt, wie in [3] aufgezeigt.

Die zukünftigen Arbeiten werden sich vor allem mit der Kopplung der Web-Service-Welt mit der εSOA-Welt, sowie mit der Entwicklung eines Laufzeitsystems zum (semi-)autonomen Betrieb der Sensornetzwerke beschäftigen.

Literatur

[1] *KNX Handbuch Version 1.1*. KNX Association, 2004.

[2] AUTOSAR – Automotive Open System Architecture. http://www.autosar.org/.

[3] C. Buckl, S. Sommer, A. Scholz, A. Knoll, and A. Kemper. Generating a Tailored Middleware for Wireless Sensor Network Applications. *SUTC*, pages 162–169, 2008.

[4] P. Costa, G. Coulson, C. Mascolo, G. P. Piccoand, and S. Zachariadis. The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems. In *PIMRC'05*, 2005.

[5] L. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. SOCRADES: A Web Service Based Shop Floor Integration Infrastructure. *IOT'08*, pages 50–67, 2008.

[6] Devices Profile for Web Services. http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf.

[7] F. Jammes and H. Smit. Service-oriented Paradigms in Industrial Automation. In *IEEE Transactions on Industrial Informatics*, volume 1, pages 62–70, 2005.

[8] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. OASiS: A Programming Framework for Service-Oriented Sensor Networks. In *COMSWARE'06*, 2007.

[9] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *TODS*, 30(1):122–173, 2005.

[10] MORE – Network-centric Middleware for Group communication and Resource Sharing across Heterogeneous Embedded Systems. http://www.ist-more.org/.

[11] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.