Softwarearchitektur für Sensornetzwerke

Frank Golatowski, Jan Blumenthal, Matthias Handy, Marc Haase,
Hagen Burchardt, Dirk Timmermann
Institut f. Angewandte Mikroelektronik und Datentechnik
Universität Rostock
Richard-Wagner-Str. 31, 18119 Rostock, Deutschland
{frank.golatowski,jan.blumenthal,matthias.handy,marc.haase,hagen.burchardt,
dirk.timmermann}@etechnik.uni-rostock.de

Abstrakt

Dieser Artikel beschreibt das neuartige Konzept eine service-orientierten Softwarearchitektur für mobile Sensornetzwerke. Diese Architektur ermöglicht eine flexible, skalierbare Programmierung von Anwendungen basierend auf einer adaptiven Middleware. Die Middleware bietet Mechanismen zur kooperativen Datenerfassung, Selbstorganisation, Vernetzung und Energieoptimierung, wodurch übergeordnete Service-Strukturen aufgebaut werden können. Die Entwicklung dieser Anwendungen wird durch optionale Administrierungskomponenten vereinfacht und deutlich beschleunigt.

1 Einleitung

Die zunehmende Miniaturisierung elektronischer Komponenten und die Fortschritte in der Entwicklung moderner Übertragungstechnologien ermöglichen künftig die Entwicklung leistungsstarker spontan vernetzbarer und mobiler Systeme. Insbesondere *drahtlosen Sensornetzwerken* wird ein wirtschaftlich enormes Entwicklungspotenzial in den nächsten 15 Jahren vorausgesagt, wenn die in dieser Arbeit angeführten Forschungsfragen zufrieden stellend gelöst werden können.

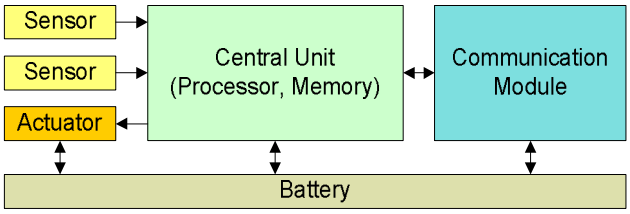

Abbildung 1: Aufbau eines Sensorknotens

Sensornetzwerke bestehen aus einer Vielzahl kleiner *Sensorknoten*, die drahtlos miteinander kommunizieren. Dadurch ist es möglich, sie auch in unzugänglichen Gebieten auszubringen, wodurch sich neue Anwendungsfelder erschließen. Sensorknoten vereinen die Fähigkeiten zur Berechnung, Kommunikation und Datenaufnahme durch *Sensoren* auf engstem Raum möglichst in einem Chip (Computation, Communication, Sensing). Prinzipiell ist das Steuern eines *Aktors* ebenfalls möglich. Abbildung 1 zeigt den Aufbau eines Sensorknotens. Die Entwicklung von Sensorknoten wird durch

- steigende Integrationsdichte auf Mikrochips,
- leistungsfähige und drahtlosen Netzwerktechnologien,
- Kombination von Signalverarbeitung und Sensordatenaufnahme sowie den
- Fortschritten im Bereich der mikroelektromechanischen Systeme (MEMS)

beeinflusst. Außerdem ist die Langlebigkeit der Netzwerkknoten durch die Verwendung energie-effizienter Bauteile und Verfahren von entscheidender Bedeutung.

Die Einsatzmöglichkeiten für Sensorknoten werden physikalisch durch Prozessorleistung, Sendereichweite, Empfangsempfindlichkeit, Stromverbrauch, Gewicht und Größe bestimmt. Denkbare Anwendungen sind medizinisches Monitoring (Monitoring over increased range) verschiedenster Gesundheitsparameter (intra- und extrakorporal), Umweltmonitoring (Wetter, Gewässer, Kontamination, Geologie), Geräte- und Maschinenüberwachung und –steuerung (Stress-, Ermüdungsmessung).
Dieser Artikel ist wie folgt organisiert: In Abschnitt 2 werden Anforderungen an Sensornetzwerke herausgearbeitet. Abschnitt 3 betrachtet Aspekte bei der Softwareentwicklung und zeigt bereits vorhandene Lösungsansätze auf. In Abschnitt 4 wird ein neuartiges Konzept einer service-orientierten Softwarearchitektur für Sensornetzwerke präsentiert. Abschnitt 5 beschließt diesen Beitrag mit einem Ausblick auf zukünftige Forschungsaktivitäten.

2 Anforderungen an Sensornetzwerke

Durch die Vernetzung der Sensorknoten ergeben sich neben anwendungsspezifischen Anforderungen eines Knotens auch dynamische Systemanforderungen an das Netzwerk. Der Fokus bei der Programmierung verschiebt sich von den Einzelergebnissen der Knoten zum Gesamtergebnis des Netzwerkes. Daraus ergeben sich folgende Anforderungen an den Entwurfs- und Implementierungsprozess von Sensornetzwerksoftware:

1. Sensornetzwerke müssen **selbstorganisierend** sein.
2. Sensorknoten müssen Funktionen zur **Aufrechterhaltung des Netzwerkes** übernehmen.
3. Durch eine **kooperative Bearbeitung** von Aufgaben soll eine höhere Ergebnisqualität erreicht und neue Anwendungsgebiete erschlossen werden.
4. Sensornetzwerke erfordern **adaptive Sicherheitsmechanismen**, die an die Aufgabe und die Umgebung, in denen sie verteilt sind, ausgerichtet sind.
5. In Sensornetzwerkknoten müssen Mechanismen, Algorithmen und Protokolle zum Einsatz kommen, die einen **sparsamen Umgang mit Energie** ermöglichen.

Auf diese Anforderungen wird nachfolgend im Einzelnen eingegangen.

2.1 Selbstorganisation

Die hohe Anzahl von Knoten in einem Sensornetzwerk erfordert durchdachte Lösungen zur automatischen Organisation des Netzwerkes. Ein manueller Bootvorgang durch den Administrator ist nahezu unmöglich. Es ist Aufgabe der Software, durch Interaktion mit den Nachbarknoten selbständig eine funktionierende Netzwerk-Infrastruktur aufzubauen. Für selbst-organisierende Netzwerke ist die Kenntnis des aktuellen Kontextes (*Context Awareness*) entscheidend. In der Organisationsphase sind vor allem der Infrastrukturkontext (Wahrnehmung der Netzwerkbreite und -verlässlichkeit) und der Domänenkontext (Beziehungen zwischen den Netzwerkteilnehmern) von Bedeutung. Für die Betriebsfähigkeit eines Sensors ist der aktuelle Systemkontext, z.B. das Wissen über die Datensenken des Systems, wichtig. Dieser Kontext kann sich z.B. durch hohe Mobilität des Systems ständig ändern, so dass Update-Mechanismen implementiert werden müssen.

2.2 Netzwerkfunktionalität

Die Softwareentwicklung für Sensornetzwerke erfordert in vielen Bereichen einen Paradigmenwechsel. Viele herkömmliche Kommunikationsprinzipien und -strukturen sind aufgrund der dynamischen Topologie und der kooperativen Aufgabenbewältigung in Sensornetzwerken ungeeignet.
Konventionelle drahtgebundene Netzwerke basieren in der Regel auf einem Client-Server-Ansatz, bei dem der Client eine Anfrage sendet und der Server darauf antwortet (*request-*

reply). Die Übertragung in Sensornetzwerken sollte dagegen ereignisbasiert sein: Das Überschreiten eines Grenzwertes an einem Knoten führt zur Auslösung eines Ereignisses, das den Datensenken zugeleitet wird. Im Gegensatz zur herkömmlichen Kommunikation via request-reply entfallen somit energiezehrende Pollingabfragen des Clients[5].
Die Adressierung von Knoten unterscheidet sich in Sensornetzwerken ebenfalls von drahtgebundenen Netzwerken. Für Sensornetzwerke ist die explizite Adressierung eines Knotens via ID bzw. IP unvorteilhaft, da durch die zufällige Verteilung und Mobilität der Sensorknoten eine Zuordnung der Knotenadresse zum Ort der Messung nicht möglich ist. Da das Ansprechen eines bestimmten Knotens nachrangig ist, wird die ortsspezifische Adressierung (*attribute-based naming)* verwendet[6]. Für die zufrieden stellende Bearbeitung von Anfragen sind Kenntnisse über die Position von Knoten und deren aktueller Zustand (Context awareness) notwendig. Eine Anfrage an das Netzwerk könnte z.B. sein: „Wie ist die aktuelle Temperatur an Ort(x,y)?“.
In großen Sensornetzwerken müssen die Datenpakete zwischen zwei Kommunikationspartnern über dazwischen liegende, nicht beteiligte, Knoten geroutet werden, d.h. alle Knoten haben neben der Messaufgabe weitere Aufgaben zu erfüllen, um den Betrieb des Netzwerkes als Ganzes aufrecht zu erhalten. Diese Hilfsaufgaben können die Lebensdauer bzw. die originäre Funktionsfähigkeit des Sensorknotens erheblich beeinträchtigen. Für die Implementierung gilt es Verfahren zu nutzen, die einen Kompromiss zwischen Netzwerknutzen, Sensoraufgabe und Überlebensdauer des Knotens bieten.
Ein möglicher Ansatz dafür ist die Strategie der *Kommunikationsvermeidung* bzw. *-verminderung*. Eine Möglichkeit zur Verringerung des Übertragungsaufwands ist das kontext-abhängige Routing. Für statische Netzwerke empfiehlt sich eine proaktive Routingstrategie, d.h. die Routen, die ein Paket zurücklegt, werden beim Booten des Netzwerkes angelegt und bleiben konstant. In einem Netzwerk mit mobilen Knoten ist dieser Ansatz nicht praktikabel, da die Routen bei einer Neuordnung des Netzwerkes ihre Gültigkeit verlieren. Daher wird in dynamischen Netzwerken das reaktive Routing benutzt, d.h. die Wegsuche erfolgt unmittelbar vor dem Paketversand. Reaktives Routing führt durch den ständigen Routenaufbau zu erhöhter Netzlast und höherem Energieverbrauch. Ein optimal angepasster Routingalgorithmus zeichnet sich somit durch eine Anpassung der Strategie an das Mobilitätsverhalten des Netzwerkes aus.

2.3 Kooperative Algorithmen

Ein entscheidender Vorteil von Sensornetzwerken gegenüber statisch vernetzten Makrosensoren ist die Möglichkeit der Implementierung von kooperativen Algorithmen. Ein vorstellbares Anwendungsgebiet derartiger Algorithmen ist die Minimierung des Netzwerkverkehrs durch Datenvorverarbeitung und -aggregation. Für eine Sensoranwendung ist es unerheblich, ob eine Datenaggregation innerhalb des eigenen oder des benachbarten Knotens erfolgt. Wichtig ist, dass die Kommunikation durch das Netzwerk zur Datensenke minimiert wird, da diese in der Regel relativ weit von den Sensorknoten entfernt ist. Ein Beispiel für ein kooperatives Verfahren ist die Positionsbestimmung (*Location awareness*) durch Triangulation. Hierzu werden Messungen aus mindestens drei verschiedenen Knoten benötigt. Die berechneten Positionen können im Anschluss für die Adressierung bzw. das Routing benutzt werden.

2.4 Auswahl geeigneter Sicherheitsmechanismen zum Schutz des Sensornetzwerkes

Die Auswahl geeigneter Sicherheitsmechanismen für drahtlose Sensornetzwerke wird vom jeweiligen Einsatzzweck und von der Umgebung, in die die Sensornetzwerke ausgebracht

werden, bestimmt. Zusätzlich müssen die Ressourcen der Sensorknoten (Prozessorleistung, Speicher und Energie) bei dieser Auswahl berücksichtigt werden.
Neben den Standard-Sicherheitsanforderungen, wie Verfügbarkeit, Vertraulichkeit, Integrität, Authentifizierung und Beweisbarkeit besitzen drahtlose Sensornetzwerke spezielle Sicherheitsanforderungen, wie Aktualität von Daten (*message freshness*), Selbsterkennung von Angriffen (*intrusion detection*), Toleranz gegenüber Angriffen und deren Eindämmung, die für den Schutz des Sensornetzwerkes erforderlich sind.
Zur Gewährleistung der genannten Sicherheitsanforderungen wurden speziell für drahtlose Sensornetzwerke angepasste Sicherheitsmechanismen von verschiedenen Forschungsgruppen entwickelt und getestet [13]. Der Einsatz und das Management dieser Sicherheitsmechanismen ist Aufgabe der Middleware. Sie entscheidet anhand von zuvor von dem Betreiber des Sensornetzwerkes definierten Sicherheitsrichtlinien (policies), welche Mechanismen eingesetzt werden können.

2.5 Low-Power-Ansatz

Sensorknoten sind gewöhnlich batteriebetrieben; da sie außerdem sehr klein und zahlreich sind, ist ein Austausch oder ein Nachladen der Batterie oft unmöglich. Außerdem werden Sensornetzwerke häufig in gefährlicher Umgebung ausgebracht, in der sie aus der Ferne gewartet und gesteuert werden. Folglich ist eine lange Batterielaufzeit ein entscheidendes Evaluierungskriterium für Sensorknoten.
Stromsparfunktionen sind umsetzbar auf verschiedenen Ebenen. Die in Sensorknoten eingesetzte Mikrocontroller-Hardware unterstützt vielfältige Stromsparfunktionen. Dazu gehört z.B. das gezielte Abschalten unbenötigter Hardwarekomponenten (Dynamic Power Management) und die Taktreduzierung des Sensorknotens.
Auf Betriebssystemebene können durch ein Low-Power Task-Scheduling, welche sich beispielsweise nichtlineare Batterieeffekte zu Nutze macht, erhebliche Einsparungen erzielt werden [15].
Durch eine energieoptimierte Ansteuerung der Kommunikationskomponente kann der Energieverbrauch eines Sensorknotens verringert werden. Besondere Einsparpotenziale bieten dabei zum einen die MAC-Schicht (TDMA vs. CDMA) als auch die Netzwerkschicht (Low-Power-Routing).

3 Software Engineering

Für die Entwicklung von Anwendungssoftware für Sensornetzwerke ist die Nutzung eines komponentenbasierten Frameworks wünschenswert. Die Komponenten dieses Frameworks stellen die Funktionalität einzelner Sensoren, Sensorknoten und des gesamten Sensornetzwerkes bereit. Zur weiteren Differenzierung der Anwendungen wird zwischen der *Sensoranwendung*, der *Knotenanwendung* und der *Netzwerkanwendung* unterschieden.
Die *Sensoranwendung* beinhaltet den kompletten Mess- und Auslesevorgang des Sensors sowie die lokale Speicherung der Daten. Sie hat vollständigen Hardwarezugriff und kann auf das Betriebssystem direkt zugreifen. Netzwerkzugriffe sind nicht möglich. Die Sensoranwendung stellt wesentliche Basisfunktionen des lokalen Sensorknotens bereit, die von der Sensorknotenanwendung genutzt werden können.
Die *Knotenanwendung* beinhaltet alle anwendungsspezifischen Aufgaben und Middleware-Funktionen zum Aufbau des Netzwerkes sowie zur Aufrechterhaltung des Netzbetriebes (Routing, Suche von Knoten und Diensten, Selbstlokalisation, etc.).
Die *Netzwerkanwendung* beschreibt sämtliche Aufgaben und benötigten Dienste des gesamten Netzwerkes, wobei keine Zuordnung zu einzelnen Knoten erfolgt. Sie stellt dem Administrator eine Schnittstelle zur Auswertung des Netzwerkes zur Verfügung.

Das Ziel unserer Forschungsaktivitäten ist die Entwicklung eines Frameworks, das die Entwicklung von Software für Sensor-, Sensorknoten- und Sensornetzwerkanwendungen grundlegend vereinfacht und eine Verteilung, Konfigurierung, Skalierbarkeit und Portierbarkeit unterstützt. Im Framework soll eine Middleware zum Einsatz kommen, die eine Anwendungsentwicklung ermöglicht, die von einer hardwarenahen Programmierung eingebetteter Systeme mit Ressourceneinschränkungen losgelöst ist. Dadurch sollen der Programmieraufwand sowie die Wartung und Pflege der Software maßgeblich minimiert werden.

3.1 Aktuelle Softwarearchitekturen kleiner verteilter Geräte mit drahtloser Netzwerkanbindung

Für eingebettete Systeme gibt es bereits Lösungen, die Servicearchitekturen und Context Awareness unterstützen. In [11] wird die distributed middleware infrastructure GAIA vorgestellt. Sie zeichnet sich durch die Koordination von Softwareinheiten und heterogene Netzwerkeinheiten aus. Das Netzwerk reagiert nach außen wie eine einzige, geschlossene Einheit. Durch die Verwendung von CORBA, XML, SQL und JAVA wird GAIA allerdings für Sensor-Netzwerke aufgrund des Ressourcenbedarfs unattraktiv.
Einen weiteren Ansatz stellt Tiny-OS dar, das bereits ein weit fortgeschrittenes Framework für Sensor-Netzwerke bietet [3]. Es ist optimiert auf Speicherverbrauch und Energieeffizienz. Tiny-OS [12] bietet Mechanismen (Events & Components), um die Kopplung der einzelnen Schichten statisch im Vorfeld zu definieren. Die Festlegung der benötigten Instanzen zur Kompilierungszeit verhindert die dynamische Speicherallozierung zur Laufzeit, wodurch zusätzlich Geschwindigkeitsvorteile erzielt werden. Tiny-OS unterstützt die Ausführung verschiedener Threads und bietet verschiedene zusätzliche Erweiterungen wie die virtuelle Maschine Maté [2] und die Datenbank TinyDB zur kooperativen Datenaggregation an. Services werden derzeit nicht unterstützt. Programme für Maté sind vorkompiliert und durch den minimalen Befehlssatz extrem kurz. Sie können in einem Tiny-OS Datenpaket von max. 24 Bytes übertragen werden. Maté erlaubt somit eine dynamische Anpassung der Knotenanwendung zur Laufzeit.
Die offene und plattformunabhängige Architektur OSGI wird in [10] für Services auf eingebetteten Systemen vorgeschlagen. OSGI ist allerdings für Sensornetzwerke aufgrund der sehr hohen Ressourcenanforderungen, u.a. ist eine virtuelle Java Maschine erforderlich (JVM), ungeeignet. Es ist demnach zwingend erforderlich, eine Servicearchitektur unter dem Kriterium des geringsten Ressourcenverbrauchs zu entwickeln.
In der Regel müssen Dienste, zumindest teilweise, lokal ausgeführt werden, um die Kommunikation mit dem Service Provider aufzunehmen. Die Ausführung kann plattformspezifisch (native), aber auch in einer speziell angepassten virtuellen Maschine erfolgen.

3.2 Eigenschaften einer Middleware für Sensornetzwerke

Die Bezeichnung Middleware bezieht sich auf die Softwareschicht zwischen Betriebssystem und Sensoranwendung einerseits und der verteilten Anwendung andererseits, die über das Netzwerk interagiert. Die Middleware-Schicht versucht vorrangig die Komplexität der Netzwerkumgebung durch die Isolierung der Anwendung vor Protokollbehandlung, Speicherentkoppelung, Netzwerkfehlern und Parallelität zu verbergen [14]. Eine Middleware für Sensornetzwerke muss folgenden Eigenschaften genügen:

- skalierbar
- generisch
- adaptiv
- reflektiv

Die eingeschränkten Ressourcen (Speicher, Verarbeitungsgeschwindigkeit, Bandbreite) verfügbarer Knotenhardware erfordern eine Optimierung jeder Knotenanwendung. Die Optimierung erfolgt zur Kompilierungszeit. Dabei wird die Anwendung auf alle wesentlichen Komponenten reduziert und die verwendeten Datentypen und Schnittstellen angepasst (*skalierbare Middleware*).
Die Middlewarekomponenten benötigen ein generisches Interface, um den Anpassungsaufwand an andere Anwendungen bzw. Knoten zu minimieren. Das Verwenden von gleichen Middleware-Komponenten in verschiedenen Anwendungen führt zu einer höheren Anzahl von komplexen Schnittstellen. Die Reduzierung dieses Overheads ist das Ziel einer *generischen Middleware*, d.h. nicht die Anpassung der Anwendung an die Schnittstellen, sondern die Anpassung der Schnittstellen an die Anwendung und deren Umgebung ist entscheidend. Eine Middleware-Funktion SetBaudrate(int transmitter, long baudrate) identifiziert beispielsweise über den ersten Parameter das Netzwerkinterface. Ein Knoten, in dem nur ein Interface eingebaut ist, benötigt diesen Parameter allerdings nicht. Die Kenntnis dieser Information zur Compilierungszeit kann folglich zur Optimierung benutzt werden.
Eine weitere Möglichkeit ist, die Semantik von Datentypen zu verändern. Eine mögliche Anwendung ist die Definition der Positionsgenauigkeit von Adressen, die zur Änderung der Datentypbreite führt. Die Typbreite hat entscheidenden Einfluss auf den Netzwerkverkehr. Neben der hardware-bedingten gibt es demnach auch eine anwendungsbezogene Datentypoptimierung.
Die Mobilität von Knoten und Änderungen in der Infrastruktur erfordern je nach Sensornetzwerkanwendung auch Anpassungen der Middleware zur Laufzeit. Diese Middleware muss in der Lage sein, Komponenten dynamisch auszutauschen und auszuführen (*adaptive Middleware*).
Reflektion bezieht sich auf die Fähigkeit eines Systems, sich selbst zu verstehen und selbst auf sich zu wirken. Ein reflektierendes System kann sein eigenes Verhalten darstellen. Dabei werden zwei wesentliche Mechanismen unterschieden - die Inspektion und die Adaptierung des eigenen Verhaltens [1][4]. Die Inspektion umfasst Möglichkeiten der Analyse des Verhaltens, z.B. durch Debugging oder Logging. Die Adaptierung gestattet die Änderung der Interna der Schicht, sodass sich das Verhalten zur Anwendung hin ändert. Im Gegensatz zur adaptiven Middleware tauscht eine *reflektive Middleware* keine Komponenten aus sondern ändert lediglich das Verhalten einzelner Komponenten. Ein Beispiel für reflektives Verhalten ist die Änderung der Routingstrategie in Abhängigkeit von der Mobilität. Die Schnittstellen zwischen den Softwareschichten bleiben konstant.

3.3 Dienste in Sensornetzwerken

Neben den ursprünglichen Netzwerk-Funktionen wie Routing werden zukünftig Servicearchitekturen benötigt, die das Aufsuchen und Ausführen von Diensten ermöglichen. Ein Dienst ist ein Programm auf das über standardisierte Funktionen über ein Netzwerk zugegriffen werden kann. Dienste gestatten eine Kaskadierung ohne vorherige Kenntnis voneinander, wodurch eine Lösung komplexer Aufgaben möglich ist.
Ein typischer Dienst wird während der Initialisierung eines Knotens genutzt, der in dem Netzwerk nach einer Senke für seine Daten sucht (*Service Discovery*). Das können Gateways aber auch benachbarte Knoten sein.
Im Desktop-Bereich ist JINI eine weit verbreitete Lösung, aber für Sensor-Netzwerke aufgrund der Ressourcenanforderungen ungeeignet. Sun hat für eingebettete Systeme die Surrogate-Architektur vorgeschlagen [9]. Diese bietet sich vor allem für Systeme an, die von einem IP-basierten Netzwerk aus gesteuert werden. Der Client hat die Möglichkeit, über eine Anfrage an den Proxy-Server auf einen nicht-standardisierten Dienst zuzugreifen. Das Surrogate übernimmt die Anpassungen von der proprietären zur standardisierten Service-

Architektur. Er agiert als Service-Provider. Eine Servicearchitektur für Sensornetzwerke ist Teil der Sensornetzwerkanwendung und arbeitet im Gegensatz zur ereignisbasierten Knotenanwendung auf dem Client-Server-Prinzip.

4 Konzept einer service-orientierten Software-Architektur für Sensornetzwerke

In Abbildung 2 ist ein Beispiel für eine einfache Service-Architektur in einem Sensor-Netzwerk gezeigt. Die Aufgabe besteht in der Erfassung des Oberflächenprofils in dem Ausbringungsgebiet. Der Client fordert das Oberflächenprofil eines begrenzten Gebietes an und stellt dazu eine Anfrage an den Proxy. Dieser Proxy kommuniziert über ein proprietäres Protokoll mit dem Sensornetzwerk. Die in dem spezifizierten Gebiet liegenden Sensoren ermitteln über kooperative Verfahren das Profil und übertragen es an den Proxy, der es an den Client weiterleitet.

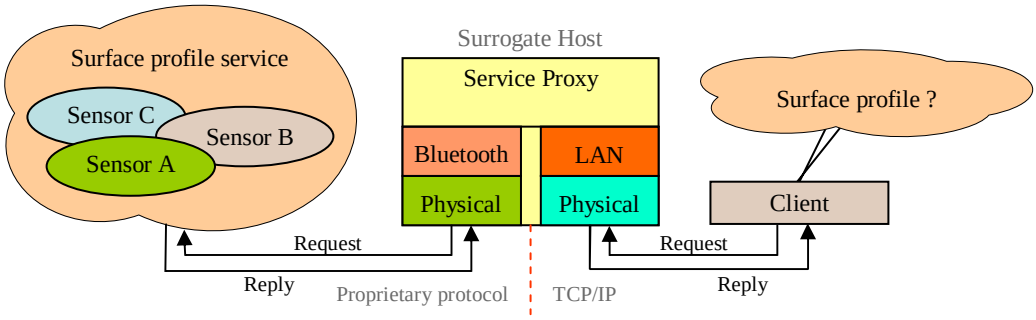

Abbildung 2: Beispiel für eine Surrogate-Architektur in Sensornetzwerken

Die in den Abschnitten 2 und 3 aufgeführten Anforderungen bzw. Ziele erfordern eine flexible Architektur der Software. Für den Knoten ergibt sich somit die in Abbildung 3 dargestellte Struktur. Sie unterteilt sich in drei Funktionsblöcke.
Die gerätespezifischen Aufgaben realisiert das *Betriebssystem*. Es umfasst den Bootvorgang, die Initialisierung der Hardware, das Scheduling sowie das Speicher- und Prozessmanagement. Es besteht lediglich aus maßgeschneiderten Komponenten, die für diesen Knoten notwendig sind.
Die Ansteuerung der Sensoren und den Messvorgang übernimmt der *Sensortreiber*. Er hat direkten Zugriff auf die Hardware des Sensors und nutzt Funktionen des Betriebssystems zur Einbindung der Sensor-Hardware.

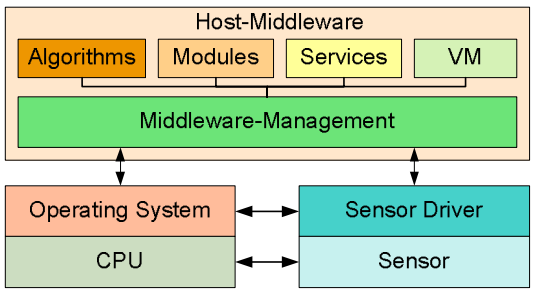

Abbildung 3: Struktur der Knotenanwendung

Die *Host-Middleware* ist die übergeordnete Softwareinstanz, die das Zusammenwirken der Knoten im Netzwerk organisiert. Das *Middleware-Management* verwaltet vier optionale Komponenten, die je nach Umfang der Knotenanwendung implementiert werden. Module sind zusätzliche Komponenten, die die Funktionalität des Knotens erhöhen. Typische Module

sind z.B. das Routing- oder das Securitymodul. Die Algorithmen beschreiben das Verhalten der Module. Beispielsweise kann sich das Verhalten eines Security-Moduls verändern, indem der Verschlüsselungsalgorithmus ausgewechselt wird. Services enthalten die Softwareteile, die zur Erfüllung von lokalen oder kooperativen Diensten notwendig sind. Sie können auf die Messdaten des Sensortreibers zugreifen und entsprechende Vorauswertungen durchführen. Eine Ausführung plattformunabhängiger Programme kann durch Hinzufügen von speziell angepassten *virtuellen Maschinen* (VM) erreicht werden.
Die Komponenten eines Knotens können statisch bzw. dynamisch zusammengebunden werden. Die statische Bindung ermöglicht die Anpassung der Schnittstellen zwischen den einzelnen Komponenten innerhalb eines Knotens, wodurch sich deutliche Optimierungspotenziale ergeben (Softwarescaling). Die dynamische Bindung kann für Komponenten definiert werden, die zur Laufzeit erweitert bzw. ausgetauscht werden. Die damit verbundene systemweit einheitliche Schnittstelle führt allerdings zu erheblichem Overhead.

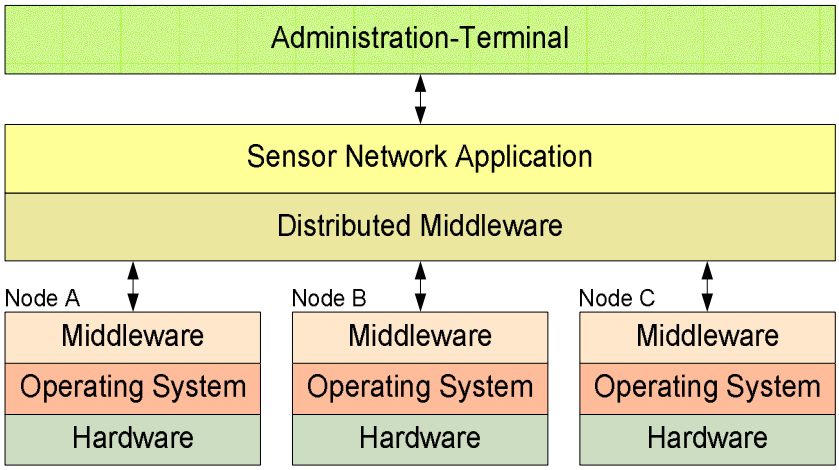

Abbildung 4: Struktur einer Sensor-Netzwerkes

Aus Sicht des Netzwerkes ergibt sich die in Abbildung 4 dargestellte logische Struktur. Die einzelnen Knoten sind nur über Middleware Services erreichbar. Die *Distributed Middleware* koordiniert die Zusammenarbeit der Dienste innerhalb des Netzwerks. Sie ist logisch auf der Netzwerkebene angesiedelt. Physikalisch existiert sie auf den einzelnen Knoten. Die Ergebnisse dieser Zusammenarbeit bilden die *Sensornetzwerk-Anwendung*.
Das *Administration-Terminal* ist eine externe Einheit, die zur Konfiguration und Auswertung des Netzwerkes an beliebiger Position im Netzwerk angeschlossen werden kann.

5 Zusammenfassung

Ausgehend von den Anforderungen an Sensornetzwerke wurden in diesem Artikel Aspekte des Softwareengineerings beschrieben. Zentrales Ziel ist eine möglichst einfache Anwendungserstellung, die unabhängig vom Aufbau der zugrunde liegenden Sensornetzwerkknoten ist. Es wurde ein Konzept einer service-orientierten Softwarearchitektur vorgestellt, die die Funktionalität des Gesamtnetzes und die Programmierung im Ganzen auf hohem Abstraktionsniveau ermöglicht.
Unsere aktuellen Arbeiten konzentrieren sich auf die Umsetzung dieser Architektur, die eingebettet in einem Framework ist. Dieses Framework soll die Entwicklung von Sensor-, Sensorknoten- und Sensornetzwerk-Anwendung grundlegend vereinfachen. Es stellt Funktionen zum Management und zur Konfiguration bereit. Es verbessert die Skalierbarkeit und die Portierbarkeit der Anwendungen.

6 Literatur

[1] G. Coulson, „What is reflective middleware?“, URL: http://dsonline.computer.org/middleware/RMarticle1.htm, DS Online, 2003.

[2] P. Levis, D. Culler, „Maté: a tiny virtual machine for sensor networks“, *Proc. of ACM Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, Oktober 2002.

[3] J. Hill et al., „System architecture directions for networked sensors“, *Proc. of the Ninth International Conference on Architectural Support for Porgramming Languages and Operating Systems*, November 2000.

[4] P. Meas, „Concepts and experiments in computational reflection“, PhD Thesis, Vrije University Brüssel, 1987.

[5] K. Römer, O. Kasten, F. Mattern, „Middleware challenges for wireless sensor networks“, *Mobile Computing and Communications Review*, Volume 6, Number 2, 2002.

[6] P. Rentala, R. Musunuri, S. Gandham, U. Saxena, „Survey on sensor networks“, *Proc. of International Conference on Mobile Computing and Networking*, 2001.

[7] D. Chess, C. Harrison, A. Kershenbaum, „Mobile agents : are they a good idea ?“, *IBM Research Report*, 1995.

[8] J. Waldo, „*The Jini™ specification*“, 2nd Edition, Addison-Wesley, 2001.

[9] „*The Jini™ technology surrogate architecture overview*“, Sun Microsystems, 2001.

[10] Open Service Gateway Initiative, URL: http://www.osgi.org/about/mission.asp

[11] M. Román et al., „*Gaia : a middleware infrastructure to enable actives spaces*“, Digital Computer Labs, University of Illinois, 2002.

[12] D. Culler, „Tiny OS – a component-based OS for the networked sensor regime“, URL: http://webs.cs.berkeley.edu/tos/, 2003.

[13] A. Perrig, D. Culler et al., “SPINS: Security protocols for sensor networks.”, *Proc. of the of the Seventh Annual International Conference on Mobile Computing and Networking*, Juli 2001.

[14] K. Geihs, “Middleware Challenges Ahead”, IEEE Computer, Juni/2001, S. 24-31.

[15] D. Rakhmatov, S. Vrudhula, C. Chakrabarti, "Battery-concious task sequencing for portable devices including voltage/clock scaling", *Proc. of the 39th Design and Automation Conference*, New Orleans, 2002.