Next-Generation Middleware for Wireless Sensor Networks

Silke Rieger
Freie Universität Berlin
14195 Berlin, Deutschland
www.fu-berlin.de

***Zusammenfassung*—Mit dem zunehmenden Einsatz von drahtlosen Sensornetzwerken ist auch die Vielfalt der eingesetzten Anwendungen gestiegen. Damit diese effizient ausgeführt werden können, ist die Verwendung geeigneter Middleware erforderlich. Mittlerweile wird eine Vielzahl von Middlewarelösungen angeboten, die sich in vielerlei Hinsicht unterscheiden und teilweise signifikante Vor- und Nachteile aufweisen. In diesem Artikel werden vier verschiedene Middlewarelösungen vorgestellt. Außerdem werden Bewertungskriterien für die Middleware von drahtlosen Sensornetzwerken herausgearbeitet und anhand derer die Middlewares miteinander verglichen.**

I. Einführung

Drahtlose Sensornetze bestehen aus vielen kleinen Sensorknoten, die mittels Funk miteinander kommunizieren können. Diese Sensorknoten bestehen aus einem Speicher, einer Batterie als Energiequelle, einem Prozessor und mindestens einem Sensor. Mit Hilfe der Sensoren ist es den Sensorknoten möglich, Informationen über ihre Umgebung innerhalb einer bestimmten Reichweite zu sammeln.

Die Sensorknoten sind in der Lage selbst Berechnungen durchzuführen und nützliche Informationen zu senden. Die Knoten kommunizieren in der Regel drahtlos mit ihren Nachbarn und kennen meistens ihre eigene Position nicht. Innerhalb des Netzes können die Sensorknoten ihre Position dynamisch verändern. Aufgrund der hohen Anzahl von Sensorknoten und ihrer begrenzten Ressourcen kommt es häufig zu Ausfällen von einzelnen Sensorknoten. Das drahtlose Sensornetz muss sich also aufgrund der Mobilität der einzelnen Sensorknoten und den Ausfällen oft dynamisch den neuen Gegebenheiten anpassen. Dadurch müssen die Topologie und die Partitionierung eines drahtlosen Sensornetzwerkes häufig erneuert werden [1].

Die Möglichkeiten eines Sensorknotens werden durch seine Einschränkungen bei der Energieversorgung und seinen individuellen Ressourcen, wie etwa Speicher und CPU, begrenzt. Um eine lange Lebensdauer zu erreichen, sollten die Anwendungen eines Sensorknotens also möglichst effizient und ressourcensparsam arbeiten, weshalb die meisten drahtlosen Sensornetze ereignisbasierend organisiert sind.

Da ein Sensornetz aus sehr vielen Sensorknoten besteht und diese oft auch in schwer zugänglichen Gebieten liegen, sollte die Installation des Netzes einfach sein. Ebenso sollten sich die Sensorknoten selbstständig konfigurieren und verwalten können [2].

Ein Sensornetz kann heterogen aufgebaut sein, d.h. die einzelnen Sensorknoten können sich in ihrer Hardware unterscheiden.

II. Middleware

Die oben erwähnten Eigenschaften und vor allem die Einschränkungen drahtloser Sensornetzwerke stellen wichtige Anforderungen an die Middleware der Sensorknoten dar.

Die Middleware befindet sich zwischen dem Betriebssystem und der Anwendung. Sie soll die Komplexität und die Infrastruktur des heterogenen Systems vor den verschiedenen Anwendungen verbergen.

Aufgabe der Middleware in drahtlosen Sensornetzen ist die Unterstützung einer einfachen Entwicklung, Installation, Wartung und Ausführung von Anwendungen oder Diensten. Das beinhaltet das Erstellen komplexer Messaufgaben, die Übermittlung der Aufgaben an das drahtlose Sensornetz, die Koordination der Aufteilung des Netzes nach Teilaufgaben, das Zusammentragen der Messdaten einzelner Partitionen als Ergebnis und das Übermitteln des Ergebnisses an den Auftraggeber [1].

Ein drahtloses Sensornetzwerk kann aus hunderten Sensorknoten bestehen, die sich in schwer zugänglichen Gebieten befinden. Somit ist eine manuelle Wartung oder gar ein Austausch nicht so einfach möglich. Daher ist die Middleware in einem drahtlosen Sensornetz gezwungen einen Mechanismus anzubieten, der effizient mit den Ressourcen eines Sensorknotens umgeht. Ein Sensorknoten sollte in der Lage sein zu messen, die Daten zu verarbeiten und zu kommunizieren, ohne seine Ressourcen zu erschöpfen [3]. Klassische Middleware designt für herkömmliche Netzwerke ist dafür nicht geeignet. Hinzukommt, dass ein drahtloses Sensornetzwerk mit dynamischen Ressourcen, wie Energie, Bandbreite und Prozessorleistung, umgehen können muss, was eine Middleware für herkömmliche Netze nicht beachtet.

Ein weiterer Unterschied zu traditioneller Middleware ist die Notwendigkeit, dass die Middleware einen Mechanismus anbietet, der das Anwendungswissen in das drahtlose Sensornetzwerk einspeist [1].

Viele Anwendungen in Sensornetzen arbeiten in Echtzeit. Daher muss die Middleware einen Echtzeit-Dienst anbieten, um sich den Veränderungen anzupassen und konsistente Daten zu erhalten [3].

Wenn eine Anwendung umfangreicher wird, muss das Netzwerk ausreichend flexibel sein, um keine Performance

einzubüßen. D.h. die Middleware muss die Skalierbarkeit eines Netzwerkes unterstützen und Dienste anbieten, um die Effizienz des Netzes aufrecht zu erhalten, wenn dieses wächst [3].

Die Middleware muss alle Prinzipien und Eigenschaften eines drahtlosen Sensornetzes beachten. Die wichtigsten sind dabei häufig Energieeffizienz, Robustheit und Skalierbarkeit. Des Weiteren muss sie in der Lage sein, das drahtlose Sensornetz mit externen Komponenten zu verbinden [1].

III. Verschiedene Middleware Ansätze

Es gibt viele verschiedene Ansätze für die Entwicklung einer geeigneten Middleware für drahtlose Sensornetzwerke. Im Folgenden werden vier von ihnen vorgestellt.

A. *RUNES*

Das europäische RUNES Projekt hat als Ziel die Entwicklung einer Architektur für vernetzte eingebettete Systeme [4] [5]. RUNES steht für Reconfigurable Ubiquitous Networked Embedded System. Die RUNES Middleware soll ein heterogenes drahtloses Sensornetz ermöglichen. Hierfür wird ein sprachunabhängiges komponentenbasiertes Modell verwendet. Durch die Entkoppelung der Implementierung von der Schnittstelle ist es möglich, verschiedene Varianten derselben Komponente zu entwickeln. Außerdem ermöglicht dies die dynamische Rekonfiguration der Komponenten und ihrer Verbindungen, wodurch die dynamische Anpassung an veränderte Bedingungen gewährleistet wird. Die RUNES Middleware reicht bis in die Schichten von Netzwerk und Betriebssystem. Das erlaubt einen einheitlichen Ansatz für die Konfiguration, die Anwendung und die Rekonfiguration auf mehreren Schichten. Für verschiedene vernetzte eingebettete Systeme können Komponenten speziell angepasst werden. Das Komponentenmodell besteht aus den beiden Elementen *Component Framework* und *reflective meta-models*.

Eine Komponente kapselt bestimmte Funktionalitäten. Die Kommunikation zwischen den Komponenten erfolgt mittels Schnittstellen und Receptacles. *Schnittstellen* definieren die Datentypen und die Signatur von Operationen. Verwendet wird hierfür OMG IDL. Eine Komponente kann mehrere Schnittstellen haben. *Receptacles* sind benötigte Schnittstellen, die benutzt werden, um die Abhängigkeit einer Komponente von einer anderen zu verdeutlichen. Beim Einsetzen einer Komponente in eine Kapsel, verraten die Receptacles welche weiteren Komponenten ebenfalls geladen werden müssen. *Bindings* sind Verbindungen zwischen einer Schnittstelle und einer Receptacle. Komponenten können zu jeder Zeit geladen werden. Jede andere Komponente innerhalb der Kapsel kann eine weitere Komponente laden. In Abbildung 1 wird der Aufbau eines RUNES Systems verdeutlicht.

Jedes RUNES System hat genau eine *Kapsel*. Sie repräsentiert einen Adressraum, der alle Instanzen des Systems beinhaltet. Außerdem bietet sie eine runtime-API. Diese erlaubt Programmierern dynamisches Laden, Löschen und Binden von Komponenten. Auf verschiedenen Geräten kann sie unterschiedlich implementiert sein. Zum Beispiel als RAM

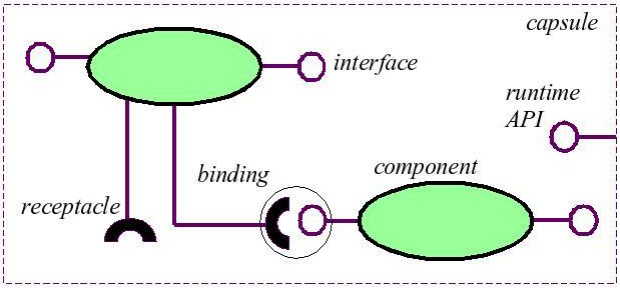

Abbildung 1. Darstellung einer RUNES Kapsel mit Komponenten und dazugehörigen Schnittstellen und Receptacles - *Abbildung aus [4] übernommen*

Chip oder als Windowsprozess auf einem PDA. Durch diese Flexibilität ist es möglich Heterogenität innerhalb des Sensornetzes zu ermöglichen.

Ein *Component Framework* ist ein gekapselter Zusammenschluss einiger Komponenten, die dieselbe Aufgabe erfüllen. Durch die Abstraktion zwischen den Komponenten und dem ganzen System soll die Verständlichkeit und die Wartbarkeit des Systems erhöht werden. Durch die Möglichkeit der Wiederverwendung und durch Dokumentationen für Programmierer wird die Entwicklung und Herstellung vereinfacht.

Während der Laufzeit ist es möglich leichte Komponenten, sogenannte Plugins, zu laden. Plugins sind eine Erweiterung oder Modifizierung der Funktionalitäten des Komponenten-Rahmenwerkes.

Die RUNES Middleware bietet einige Komponenten-Rahmenwerke standardmäßig an. Das *Local OS services* Rahmenwerk bietet Funktionalitäten an, die normalerweise vom Betriebssystem übernommen werden. Das *Network Services* Rahmenwerk unterstützt eine erweiterbare Menge von Kommunikationsdiensten und bietet eine generische API. Es wird dabei sowohl ad-hoc, als auch infrastrukturbasierte Kommunikation unterstützt. Das *Coordination Services* Rahmenwerk dient der Koordination der Knoten untereinander, zum Beispiel bietet er Stromsparfunktionen und sogenannte scheduling Dienste an. Das *Location Services* Rahmenwerk dient zur Bestimmung der physikalischen oder logischen Position der Sensorknoten. Letztlich bietet das *Interaction Services* Rahmenwerk zusätzliche Funktionen an, die auf der Netzwerkschicht aufsetzen und somit spezifische Anwendungsanforderungen erfüllen können.

B. *SwissQM*

Die Abkürzung SwissQM steht für ***S**calable **WI**reless **S**ensor network **Q**uery **M**achine.* SwissQM wurde im Rahmen des XTream-Projektes an der ETH Zürich entwickelt, mit dem Ziel eine bessere Datenerfassung und vor allem Datenunabhängigkeit in drahtlosen Sensornetzen zu erreichen [6]. Dafür sollen eine leistungsstarke und anpassungsfähige Schnittstelle und eine flexible Funktionalität für das Sensornetz geschaffen werden. Wichtig dabei ist eine Optimierung für erheblich mehr Anwendungsfälle als bisher üblich. SwissQM stellt ein generisches deklaratives Programmiermodell zur Verfügung, welches zahlreiche Optimierungen zulässt.

Mehrbenutzerbetrieb, d.h. konkurrierende Anfragen an das Sensornetz werden, wie zum Beispiel in gängigen Datenbanksystemen üblich, unterstützt.

SwissQM hat vier Anforderungen aufgestellt, um seine gesteckten Ziele zu erreichen.

1) *Trennung der Sensoren von der externen Schnittstelle*: Es sollte keine bestimmte Anfragesprache implementiert sein, damit eine Sprachunabhängigkeit besteht. Das drahtlose Sensornetz muss dynamisch anpassungsfähig sein. Die Knoten sollen keine anwendungsspezifische Funktionalität, wie zum Beispiel das Parsen von SQL besitzen. Diese Funktionalitäten stehen als dynamisch einsetzbare Erweiterungen zur Verfügung.
2) *Dynamische Multiuser und Multiprogramming Umgebung*: Von SwissQM sollten keinerlei Einschränkungen bezüglich des Mehrfachbenutzerzugriffs ausgehen.
3) *Optimierte Benutzung der Sensoren*: Erfassen, Berechnen und Senden von Daten sollten die einzigen Prozesse auf den Sensorknoten sein. Dies erhöht den Speicherplatz für Daten und ermöglicht eine höhere Anzahl von Anfragen.
4) *Erweiterbarkeit*: SwissQM legt großen Wert auf Erweiterbarkeit, dabei soll auf der einen Seite die Middleware selbst ohne großen Aufwand vom Benutzer erweitert werden können, und auf der anderen Seite sollen die mit SwissQM betriebenen Systeme ausbaufähig sein.

Ein SwissQM Sensornetz besteht aus einem Gateway-Knoten und mindestens einem Sensorknoten. Der Gateway-Knoten, den alle drahtlosen Sensornetze benötigen, dient als Schnittstelle zwischen der Außenwelt und dem Sensornetz selbst. Er muss über ausreichend Rechnerleistung verfügen und darf im Gegensatz zum Sensorknoten keiner Energie- oder Speichereinschränkung unterliegen. Die Sensorknoten dienen sozusagen nur der Datenerfassung, während das Gateway den Großteil der Verarbeitungen vornimmt. Organisiert sind die Knoten in einer Baumstruktur. Gesammelte Daten werden über die Sensorknoten zur Wurzel hin weitergeleitet, was den Umfang des Overheads durch Routingdaten in den Sensorknoten reduziert. Der Elternknoten eines Sensorknotens liegt näher an der Baumwurzel und dem Gateway, als der Sensorknoten selbst, siehe Abbildung 2. Nachteilig ist hierbei, dass bei Ausfall eines Elternknotens auch die dazugehörigen Kindknoten ausfallen. Meist wird ein einzelner Baum als Routingschema verwendet, SwissQM unterstützt aber auch die Verwendung mehrerer unabhängiger Bäume.

SwissQM setzt *In-Network Aggregation* ein, das bedeutet, dass bei einer Anfrage an das Sensornetzwerk pro Verbindung nur eine Nachricht verschickt wird und die Kindknoten ihre Daten vor dem jeweiligen Elternknoten in Richtung Wurzelknoten weitergeben. Dieses Vorgehen kann die Anzahl der verschickten Nachrichten innerhalb des Netzwerkes reduzieren, wodurch Ressourcen, wie zum Beispiel Energie, gespart werden.

Falls es nicht durch eine Optimierungsstrategie verboten ist, werden alle Daten im Gateway statt im Sensorknoten verarbeitet. Das Gateway verwaltet außerdem alle externen

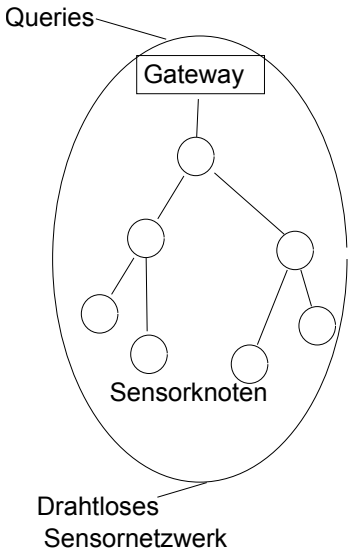

Abbildung 2. Darstellung eines SwissQM - Sensornetzwerkes

Schnittstellen, auf den Sensorknoten selbst befindet sich nur der Code, der für das Erfassen, Berechnen und Senden von Messdaten notwendig ist. Weiterer Code wird nur auf die Sensorknoten geladen, wenn dieser explizit vom Gateway weitergegeben wurde. Leitet das Gateway Programmcode an einen Sensorknoten weiter, so nennt man diesen Vorgang *push-down*.

SwissQM unterstützt eine Vielzahl von Schnittstellen, was den Vorteil bietet, dass die Schnittstellen angepasst werden können, anstatt dass eine Veränderung in den Sensorknoten vorgenommen werden muss.

Das Gateway verarbeitet die *user queries* und liefert als Antwort einen Datenstrom zurück. Dabei verarbeitet und kombiniert das Gateway user queries zu *virtual queries*. Die Anfragen werden dabei verkleinert, optimiert und in ein internes Format umgewandelt. Anschließend wird die virtuelle Anfrage in ein *network query* umgewandelt. Dieses besteht aus Bytecode, den die Sensorknoten direkt lesen können. Die Sensorknoten sind stackbasierte virtuelle Maschinen, sogenannte *Query Machines*.

Eine solche Anfragemaschine ist wie folgt aufgebaut. Die Operanden und Ergebnisse der Anweisungen werden als 16-Bit Elemente in einem *Operand Stack* gespeichert. Daten, die zu versenden sind oder empfangen, aber noch nicht verarbeitet wurden, werden in einem *Transmission Buffer* zwischen gespeichert. Für die Speicherung der Aggregationszustände wird eine Datenstruktur, genannt *Synopsis*, verwendet, hier werden die Daten von den Kindknoten mit den eigenen Daten verknüpft. Außerdem verfügt die Anfragemaschine natürlich über die notwendigen Sensorknoten und den Bytecode.

Durch den Einsatz des Gateways entsteht eine hohe Abstraktion, wodurch technische Details des Sensornetzwerkes vor dem Benutzer verborgen werden.

C. *TinyCubus*

TinyCubus beschäftigt sich mit dem Problem der steigenden Komplexität von drahtlosen Sensornetzanwendungen. TinyCubus wurde mit dem Ziel entwickelt, die verschiedenen Anwendungen eines Sensornetzes an die wechselnden Aufgaben und die unterschiedlichen Anforderungen anzupassen [7]. Die Implementierung basiert auf TinyOS [8] und besteht aus einem *Data Management Framework*, einem *Cross-Layer Framework* und einer *Configuration Engine*.

Das Datenverwaltungs-Rahmenwerk bietet eine Vielzahl von Standard-Datenmanagement- und Systemkomponenten an. Es gibt verschiedene Implementierungen für jeden Komponententyp. Das Datenverwaltungs-Rahmenwerk ist verantwortlich für die Auswahl der passenden Implementierung, die aufgrund der erhaltenen Systeminformationen ausgewählt wird. In jedem Sensorknoten sind nur die notwendigen Komponenten geladen, bei Bedarf können von anderen Sensorknoten oder dem Gateway jedoch Weitere geladen werden.

Ein Komponententyp wird nach unterschiedlichen Kriterien eingeteilt: den Anforderungen der Anwender, den Optimierungsparametern und den Systemparametern. Das Datenverwaltungs-Rahmenwerk wählt aus den zur Verfügung stehenden Komponenten diejenige aus, die die drei Kriterien am besten erfüllt. Falls sich die Anforderungen der Anwendung ändern, kann die Komponente gegen eine andere ausgetauscht werden. Diese Anpassung ist ein wichtiger Teil des Optimierungsprozesses.

Das Cross-Layer Rahmenwerk bietet eine generische Schnittstelle zur Unterstützung der Parametrisierung von Komponenten, ermöglicht die Verwaltung der Datenabhängigkeiten und steht als Vermittler zwischen den einzelnen Komponenten zur Verfügung. Es erfolgen keine direkten Datenzugriffe von einer Komponente auf eine Andere, stattdessen werden die Daten im gemeinsamen *State Repository* gespeichert. Der Austausch einer Komponente betrifft somit keine anderen Komponenten, vorausgesetzt die neue Komponente liefert dieselben Daten wie die Alte. Außerdem verwaltet der Cross-Layer die Callback-Funktionen zwischen den Komponenten.

Das Ziel der Tiny Configuration Engine ist die Unterstützung der Rekonfiguration und Installation von Komponenten. Die Configuration Engine besteht aus drei Teilen: *Topology Manager*, *Role Specification* und *Role Assignment Algorithm*. Der Topology Manager ist für die Konfiguration des Sensornetzes und die Zuweisung von Rollen an die Sensorknoten verantwortlich.

Die Abbildung 3 liefert einen guten Überblick über TinyCubus. Der in der Mitte abgebildete Würfel, genannt *Cubus*, vereint die drei Komponentenkriterien.

Die Sensorknoten innerhalb eines TinyCubus Netzwerkes sind somit in der Lage selbstständig einzelne Komponenten auszutauschen, ohne dabei von einer zentralen Stelle gesteuert zu werden. Jeder Knoten erhält dabei die zu seiner Rolle passenden Bausteine aus dem Code-Repository. Das Sensornetz baut sich sozusagen intelligent selbst auf.

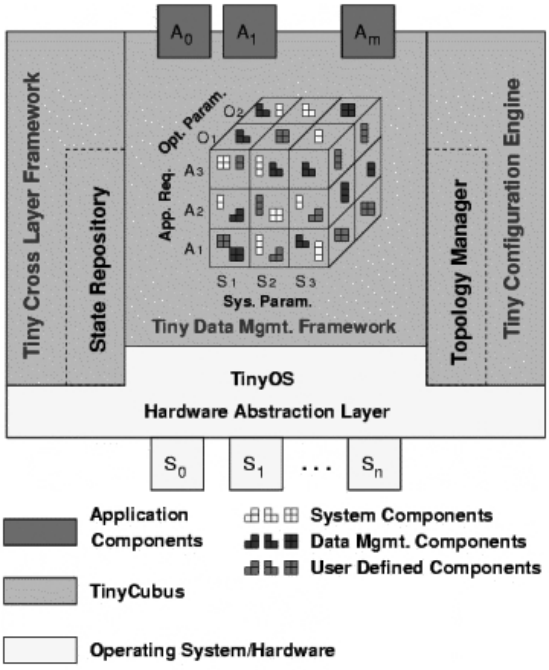

Abbildung 3. Darstellung der Komponenten von TinyCubus - *Abbildung aus [7] übernommen*

D. *TeenyLIME*

Neben den Anwendungen, die der reinen Datensammlung dienen, gibt es Anwendungen, die neben dem Sammeln der Umgebungsdaten auch auf die externen Reize reagieren können, sogenannte *sense-and-react* Anwendungen. In einem solchen System sind die Sensorknoten sogenannte Aktore, die auf externe Anregung reagieren. Ein Beispiel hierfür ist eine Anwendung zur Überwachung der Raumtemperatur. Eine *sense-and-react* Anwendung ist wie folgt aufgebaut. Es gibt die *Benutzervorgaben*, die das Ziel des Systems repräsentieren, zum Beispiel die gewünschte Raumtemperatur. Die *Sensorgeräte* sammeln die Umgebungsdaten, in diesem Fall die Raumtemperaturen. Die *Aktore* starten Aktionen aufgrund der gesammelten Daten, um die Benutzervorgaben zu erfüllen, wie etwa das Senken der Temperatur, um die gewünschte Raumtemperatur zu erreichen. Bei letzterem handelt es sich um lokale Reaktionen, die ebenso wichtig sind, wie die Fähigkeit Daten zu sammeln. Dadurch, dass die Reaktionen auf die gemessenen Daten nicht von einer zentralen Stelle gesteuert werden, sondern lokal durch die Aktore ausgelöst werden, wird die Verzögerung der Reaktion auf die Messdaten reduziert und es werden die Ressourcen eingespart, die sonst für das Weiterleiten der Messdaten verwendet werden müssten.

TeenyLIME [9] basiert auf der Idee vom geteilten Tupelraum, die ursprünglich von der Programmiersprache Linda [10] stammt. LIME steht für ***L**inda **i**n a **M**obile **E**nvironment*. Bei einem Tupelraum handelt es sich um einen Speicher, der von mehreren Prozessen genutzt werden kann. Jeder Sensorknoten in einem TeenyLIME Netzwerk hat seinen eigenen Tupelraum und kann zusätzlich auf die Tupelräume

seiner direkten Nachbarknoten zugreifen. Die Operationen finden immer im gesamten geteilten Tupelraum eines Knotens statt, ebenso stammen die Ergebnisse auch aus dem gesamten geteilten Tupelraum. Sowohl die Messdaten, als auch die Aktorbefehle werden als Tupel präsentiert. Jedes Tupel ist eine Sequenz von typisierten Feldern, wie zum Beispiel `<"Beispiel", 6, 13.4>`. Dadurch stehen jedem Knoten neben den eigenen Tupeln auch die der direkten Nachbarn zur Verfügung, wodurch die Daten nicht mehr versendet werden müssen. Dies wird meist von der Anwendungsschicht direkt ausgenutzt. Das Ziel von TeenyLIME ist eine einfache Anpassung und Erweiterung der Middleware. TeenyLIME ist auf TinyOS [8] aufgebaut und ist unabhängig von der Knotenplattform.

IV. Vergleich der Middlewares

Dieser Abschnitt der Arbeit befasst sich mit dem Vergleich der Middlewarelösungen. Zuerst werden die verschiedenen Bewertungskriterien herausgearbeitet, an denen sich die oben präsentierten Middlewarelösungen orientieren und die von den jeweiligen Autoren als besonders wichtig angesehen werden. Anschließend wird die Aussagekraft und Wichtigkeit dieser Metriken betrachtet.

Dabei gilt es *qualitative* und *quantitative* Metriken zu unterscheiden. Bei quantitativen Metriken handelt es sich um Bewertungskriterien, deren Ergebnisse sich konkret miteinander vergleichen lassen, das heißt entweder erhält man bei Vergleichen konkrete Zahlen als Ergebnisse oder Antworten in Form von Ja oder Nein. Ergebnisse qualitativer Bewertungskriterien hingegen sind nicht so eindeutig vergleichbar und auch wesentlich schwerer zu erlangen, da sie nicht direkt messbar sind.

Zuerst betrachten wir die Middleware RUNES. Der Entwicklungsschwerpunkt von RUNES ist ganz klar auf *Rekonfigurierbarkeit* und *Heterogenität* gelegt worden. Diese beiden Eigenschaften stellen natürlich auch zwei der wichtigsten Bewertungskriterien für Middlewarelösungen dar, welche für den Einsatz in drahtlosen Sensornetzwerken der Zukunft konzipiert sind. Folglich ist offensichtlich, dass RUNES hinsichtlich dieser Kriterien sehr gut abschneidet. So wurde zum Beispiel die Heterogenität in [11] exemplarisch an einem Projekt vorgeführt. In [12] wird die Middleware evaluiert, um zu zeigen, dass RUNES Heterogenität unterstützt und *energiesparsam* arbeitet. Um diese beiden Eigenschaften zu belegen, wird die Höhe des Speicherplatzbedarfes des Kernels betrachtet.

Die von SwissQM aufgestellten Anforderungen drei und vier, die bereits in Abschnitt 3B vorgestellt wurden, beinhalten die Eigenschaften *Ressourcensparsamkeit*, *Erweiterbarkeit* und *Anpassungsfähigkeit* [6], welche durch die Implementierung der Anfragemaschine bedient werden. Der eingesetzte Bytecode trägt dazu bei, dass die *Länge der* im SwissQM-Netzwerk verschickten *Nachrichten* klein bleibt.

In [7] stellen die Autoren die Middleware TinyCubus als einen rollenbasierten Algorithmus vor, wodurch die *Anzahl der versendeten Nachrichten* pro Knoten klein gehalten wird. Dies funktioniert am besten in Netzwerken, in denen die Verteilung der Rollen innerhalb des Netzes bekannt ist. Durch die Verwendung der drei Komponentenkriterien und die Möglichkeit die Komponenten während der Laufzeit zu laden, ist TinyCubus *adaptiv*, was auch ein wichtiges qualitatives Bewertungskriterium ist. Ein allgemeines Rahmenwerk ist zudem *erweiterbar*.

Middleware	*Berücksichtigte Metriken*
RUNES	Heterogenität, Rekonfigurierbarkeit, Energiesparsamkeit
SwissQM	Rekonfigurierbarkeit, Erweiterbarkeit, Ressourcensparsamkeit
TinyCubus	Rekonfigurierbarkeit, Erweiterbarkeit
TeenyLIME	Rekonfigurierbarkeit, Komplexität

Tabelle I
Qualitative Metriken

Middleware	*Berücksichtigte Metriken*
RUNES	-
SwissQM	Nachrichtenlänge
TinyCubus	Anzahl der versendeten Nachrichten
TeenyLIME	Anwendungszustände, Anzahl der Codezeilen

Tabelle II
Quantitative Metriken

In [9] wird die Middleware TeenyLIME mit einer nativen Implementierung verglichen, da den Autoren keine weitere sense-and-react Implementierung bekannt war. Die Autoren vergleichen die beiden Implementierungen im Bezug auf ihre *Komplexität*. Um die Komplexität festzustellen, betrachten die Autoren die Zustandsautomaten, sowie die daraus folgende Anzahl der *expliziten Anwendungszustände*. Je größer die Anzahl der Anwendungszustände, desto schwieriger ist die Implementierung der Zustandsübergänge, was sich auch in der *Anzahl der Codezeilen* widerspiegelt. Das qualitative Bewertungskriterium *Flexibilität* wird von der TeenyLIME Middleware durch die Entkopplung der Komponenten und dem anonymen Datenaustausch durch die Tupelräume gewährleistet.

Tabelle I bietet nochmal eine Übersicht über die betrachteten qualitativen Bewertungskriterien der vier Middlewares. Besonders wichtig für eine Middleware in einem drahtlosen Sensornetzwerk sind die Eigenschaften Ressourcensparsamkeit, Heterogenität und Rekonfigurierbarkeit. Anhand dieser drei Eigenschaften müssen sich Middlewares der nächsten Generation messen lassen, was sich nur durch umfangreiche Testläufe erreichen lässt.

In Tabelle II werden die unterschiedlichen quantitativen Eigenschaften angeführt. Besonders interessant sind hier Nachrichtenlänge und -größe, weil diese direkt mit der Energieeffizienz zu tun haben. Leider gehen nicht alle Autoren gleichermaßen auf diese Merkmale ein, so dass ein direkter Vergleich nicht möglich ist. Insgesamt sind die oben genannten qualitativen Eigenschaften ausschlaggebend, denn nur sie können die wirkliche Eignung einer Middlewarelösung auf einem bestimmten Gebiet aufzeigen. So kann zum Beispiel die Nachrichtenlänge ein Indiz für die Energieeffizienz sein, letztendlich gibt sie keinen endgültigen Aufschluss darüber.

V. Fazit

In diesem Artikel habe ich Middlewareansätze für die immer komplexer werdenden drahtlosen Sensornetzwerke vorgestellt. Diese Ansätze versuchen die Komplexität des Netzes gegenüber dem Anwendungsentwickler zu verstecken. Ganz deutlich zu erkennen ist die Fähigkeit aller Ideen möglichst heterogen, flexibel und anpassungsfähig zu sein. Die Entwicklung geht also von maßgeschneiderten speziellen Lösungen hin zu wiederverwendbaren rekonfigurierbaren Ansätzen, die mit Hilfe von Komponentensystemen relativ nah an die Hardware angepasst werden können. In Zukunft wird sich durch umfangreiche Anwendungen in echten Szenarien zeigen, ob durch Middleware entwickelte Sensornetzanwendungen einen Vorteil gegenüber denen auf das Netzwerk speziell zugeschnittenen haben werden.

12 Juni 2008

Literatur

[1] K. Roemer, O. Kasten, and F. Mattern, "Middleware challenges for wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 4, pp. 59–61, 2002.

[2] F. Golatowski, J. Blumenthal, M. Handy, M. Haase, H. Burchardt, and D. Timmermann, "Softwarearchitektur für sensornetzwerke," -, 2003. [Online]. Available: http://www-md.e-technik.uni-rostock.de/veroeff

[3] S. Hadim and N. Mohamed, "Middleware: Middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, p. 1, 2006.

[4] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis, "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), SEP 2005.

[5] C. Mascolo, S. Zachariadis, G. P. Picco, P. Costa, L. Mottola, G. Blair, and N. B. und P. Okanda, "Runes middleware architecture," 2005. [Online]. Available: http://www.ist-runes.org/docs/deliverables/D5_02_01.pdf

[6] R. Mueller, J. S. Rellermeyer, M. Duller, G. Alonso, and D. Kossmann, "A dynamic and flexible sensor network platform," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2007, pp. 1085–1087.

[7] P. J. Marrón, A. Lachenmann, D. Minder, J. Hähner, R. Sauter, and K. Rothermel, "TinyCubus: A flexible and adaptive framework for sensor networks," in *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, January 2005, pp. 278–289. [Online]. Available: citeseer.ist.psu.edu/736082.html

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, vol. 35, no. 11, pp. 93–104, 2000.

[9] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Programming wireless sensor networks with the teenylime middleware," in *Proceedings of the 8th ACM/IFIP/USENIX International Middleware Conference (Middleware 2007)*, Newport Beach (CA, USA), November 2007.

[10] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.

[11] G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, and S.Zachariadis, "Dynamic reconfiguration in the runes middleware," in *Formal research demonstration in Proc. of the 3rd IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS06)*, 2006.

[12] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis, "The runes middleware for networked embedded systems and its application in a disaster management scenario," in *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 69–78.