Ein Vergleich von Betriebssystemen für Sensorknoten: Mantis OS, cocoOS und .NET Micro Framework

Maximilian Weber
Betreuer: Christoph Söllner
Hauptseminar Sensorknoten - Betrieb, Netze & Anwendungen
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: webermax@in.tum.de

KURZFASSUNG

In diesem Papier wird ein Vergleich der Betriebssysteme Mantis OS, cocoOS und .NET Micro Framework vorgenommen. Dabei wird der Einsatz auf Sensorknoten hinsichtlich Effizienz, Sicherheit und Benutzungskomfort beleuchtet. Die jeweiligen Vor- und Nachteile werden diskutiert und eine abschließende Bewertung vorgenommen. Für die optimale Wahl eines der behandelten Betriebssysteme ist der konkrete Anwendungsfall ausschlaggebend.

Schlüsselworte

Sensorknoten, Sensornetz, Betriebssystem, Mantis OS, cocoOS, .NET Micro Framework

1. EINLEITUNG

Die systematische Erfassung großer Datenvolumen innerhalb technischer und biologischer Systeme findet zunehmend Verbreitung. So werden nicht nur Fußballspieler während eines gesamten Turniers möglichst ganzheitlich überwacht, auch Smartphones generieren Positions- und Verbindungsdaten in erheblichem Ausmaß an zentraler Stelle. Ebenso werden mittels Sensornetzen Frühwarnsysteme für Erdbeben realisiert. Doch damit sind die Anwendungsmöglichkeiten vernetzter Sensorknoten bei Weitem nicht ausgeschöpft. Beispielsweise kann das bereits 1999 von Kahn [4] beschriebene Smart Dust dank Innovationen aus der Nanotechnologie heute längst realisiert werden. Dabei kommen winzige Sensorknoten zum Einsatz, die nicht größer als ein Kubikmillimeter sind. Lee [7] verwendet hierzu optische Kommunikationswege und Solarzellen zur Energiegewinnung.

Die vom gewöhnlichen Rechnerbetrieb abweichenden, äußeren Bedingungen bei derartigem Einsatz von IT Systemen erfordern besondere Softwarelösungen. Denn die Beschränkungen der Leistungsfähigkeit durch den extrem verkleinerten Formfaktor eines Sensorknotens von meist wenigen Zentimetern erzwingt eine wesentlich sparsamere Art der Ressourcenverwaltung und damit der Programmausführung. Für die Lösung der hieraus resultierenden Problemstellungen wie der geringeren Bandbreite bei der Datenübertragung oder der niedrigen Taktraten von Mikrocontrollern gibt es grundsätzlich verschieden etablierte Ansätze für die Schaffung von Rahmenbedingungen für Anwendungsprogramme.

Die zahlreich vorhandenen Umsetzungen werden unterschiedlichen Sicherheitsanforderungen bei Kommunikation und Programmausführung gerecht und unterstützen den Entwickler mehr oder weniger ausführlich mit Dokumentationsmaterial, was im Folgenden anhand der konkreten Ausprägungen Mantis OS, cocoOS und .NET Micro Framework erläutert wird.

2. BEGRIFFSKLÄRUNGEN

Ein Betriebbsystem für Sensorknoten weist grundlegende Unterschiede zu konventionellen Betriebssystemen, wie sie in Desktopumgebungen verwendet werden, auf. Diesem neuen Kontext wird der Begriff des Betriebssystems entsprechend angepasst. Dieser Definition wird außerdem die Klärung der benötigten Begriffe Sensorknoten und Sensornetz vorweggenommen.

2.1 Definition eines Sensorknotens

Sensorknoten sind auf das Wesentliche reduzierte Rechenmaschinen. Sie verfügen, wie zum Beispiel das Modell TELOSB von Crossbow [12], in der Regel mindestens über einen Mikrocontroller, eine Programmierschnittstelle, eine (Funk-)Netzwerkeinheit und die Möglichkeit, Sensoren für Messungen anzubringen, beispielsweise um die Umgebungstemperatur, die Lichtstärke oder Beschleunigungen zu erfassen. Üblicherweise werden diese mittels einer mobilen Stromquelle betrieben. Dabei kann es sich um eine Batterie handeln oder sogar um eine dauerhafte Energieversorung wie sie beispielsweise mittels photovoltaischer Zellen realisiert werden kann.

2.2 Definition eines Sensornetzes

Die Zusammenschaltung einer Anzahl von Sensorknoten erfolgt nur bedingt analog der Vernetzung konventioneller Rechner. Besonders den Einschränkungen hinsichtlich Reichweite und Bandbreite muss Rechnung getragen werden. Dabei kann zwischen zentral vernetzten Netzwerken, beispielsweise mit Sterntopologie, und vermaschten Sensornetzen ohne feste Infrastruktur unterschieden werden. Die logische Netzwerkstruktur kann abhängig sein von den physikalischen Bedingungen für die Kommunikationswege innerhalb des Netzwerks (siehe Abbildung 1) [9]. So sind Sensornetze mit mobilen Knoten, welche sogar den Kommunikationspartner wechseln können, in ihrer physikalischen Topologie flexibel, was aber nicht zwingend eine Veränderung der logischen Topologie nach sich ziehen muss.

2.3 Definition eines Betriebssystems für Sensorknoten

Im Gegensatz zu Desktopsystemen wird auf einem Sensorknoten in der Regel keine separate Installation für das Be-

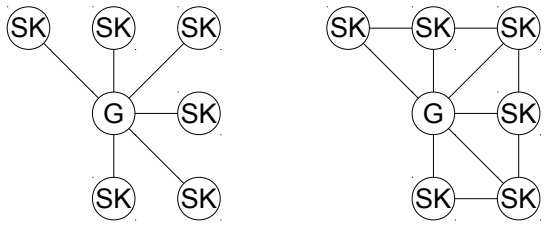

Abbildung 1: Physikalische Sterntopologie (links) und logische Vermaschung (rechts) von Sensorknoten (SK) und Gateway (G)

triebssystem durchgeführt. Stattdessen wird ein Rahmenprogramm eingesetzt, welches Anwendungsprogrammen eine Hardwareabstraktion liefert und die Zuteilung von Resourcen wie Rechenzeit und Sendezeit übernimmt. Eingebettet in diesen Rahmen wird somit das Programm gemeinsam mit dem Betriebssystem auf den Sensorknoten übertragen.

3. SPEZIELLE ANFORDERUNGEN

Um trotz der Einschränkungen auf Seiten der Hardware möglichst flexibel zu bleiben, werden die speziellen Betriebssysteme, beziehungsweise Frameworks, den restriktiven Anforderungen angepasst, um einen komfortableren Umgang mit den nachfolgenden Problemstellungen zu ermöglichen. Damit müssen häufig auftretende Probleme für verschiedene Programme nicht mehrfach gelöst werden.

3.1 Energieverbrauch

Der wichtigste Aspekt für den Produktivbetrieb eines Sensorknotens ist die Leistungsaufnahme. Einerseits sollen Daten möglichst großflächig erfasst werden, was eine relativ hohe Übertragungsleistung der verwendeten Funkeinheiten erfordert. Andererseits sollen Daten üblicherweise über einen möglichst langen Zeitraum hinweg erfasst werden, um die Aussagekraft zu erhöhen. Diesen Widerspruch gilt es durch einen effektiven Bereitschaftszustand aufzulösen, welchem sich das Gesamtkonzept für den Softwarebetrieb unterordnet.

3.2 Rechenleistung

Die Rechenleistung verbreiterter Mikrocontroller für Sensorknoten liegt weit unterhalb derjeniger gängiger Desktop CPUs. Dies ist sowohl der lediglich knapp verfügbaren Energieressource geschuldet als auch der Wirtschaftlichkeit. Damit werden beispielsweise Kontextwechsel beim Übergang zu einem anderen Prozess teuer, was ein geeignetes Scheduling Verfahren erfordert.

3.3 Sicherheit

Nicht nur hinsichtlich des Datenschutzes ist die Anfälligkeit eines Sensornetzes für Manipulationen, Ausspähungen oder Überlastung von Bedeutung. So gilt es ebenso, sicherheitskritische Systeme hinreichend vor Ausfällen zu schützen und diese zu erkennen.

3.4 Multitasking

Ein sinnvoller Ansatz für den Betrieb eines Sensorknotens in einem Netzwerk ist die parallele Ausführung mehrerer Aufgaben. Dies kann auf unterschiedliche Art, zum Beispiel mittels der Nutzung von Semaphoren realisiert werden.
Vor allen Dingen in mobilen Sensornetzen mit einer Vielzahl gleichartiger Knoten kann die Idee des Multitaskings gut ausgenutzt werden. Krüger [5] hat dies mit der funktionalen und geographischen Gruppierung von Sensoren gezeigt. Vor allen Dingen wenn mehrere Sensoren je Knoten verwendet werden ist es notwendig, die Messwerte des einen Sensors zu erfassen, während die Daten des anderen Sensors bereits ausgewertet und übertragen werden.

4. MANTIS OS

Nach Aufgaben abgegrenzte Schichten bilden das Modell für das Mantis OS-Design.
Mantis OS bietet weitreichende Möglichkeiten für Multitasking und ist speziell für den Einsatz in Sensornetzen ausgelegt. Das Betriebssystem ist unter einer GNU General Public License (GPL) kompatiblen Lizenz frei verfügbar. Weiterhin ist die gcc Toolchain für den Übersetzungsvorgang ausreichend.

4.1 Dokumentation

Bhatti [1] legt in einer ausführlichen Veröffentlichung die Grundprinzipien des Frameworks dar und die Projekthomepage [20] sammelt weitere Publikationen über Mantis OS. Eine anschauliche Dokumentation ist dort ebenfalls zu finden. Diese stellt eine gelungene Ergänzung zur ausführlichen Online-Beschreibung der Mantis API dar und beschreibt den Installationsvorgang des Systems auf der verwendeten Workstation sowie den schematischen Aufbau der vorhandenen Schichten.

4.2 Hardwareunterstützung

Durch die Orientierung am POSIX-Standard erzielt Mantis OS eine hohe Hardware-Kompatibilität. Zur weiteren Vereinfachung wird von Bhatti eine Referenzimplementierung, der *MANTIS hardware nymph sensor node* [1], vorgestellt. Diese besteht aus einem single-board Design mit einem Atmel Atmega128(L) Mikrocontroller sowie einer analogen Sensorschnittstelle. Der Nymph Sensorknoten ist als Ausgangspunkt für individuelle Umsetzungen erhältlich.
Weiterhin kann Mantis OS auf verbreiteten Plattformen wie MICA und TELOS Motes eingesetzt werden [20].

4.3 Architektur

Das Mantis System ist geprägt von einer nicht-strikten Schichtenarchitektur (siehe Abbildung 2) [21]. Den Kern bilden die Geräteschicht, die Netzwerk- und Protokollschicht sowie die Kommunikationsschicht.

4.3.1 COM Layer

Kommunikationsaufgaben werden von der entsprechenden Schicht abgewickelt. Sämtliche Kommunikationsgeräte werden mit ihrer Hilfe angesprochen. Dazu zählen sowohl Drahtlosverbindungen als auch die Datenübetragung über weitere, serielle Schnittstellen.
Dabei wird nicht nur das Senden und Empfangen organisiert, sondern auch die konkrete Behandlung der Zwischenspeicher für Lese- und Schreibvorgänge. Damit kann auch das Mantis Terminal angesprochen werden.

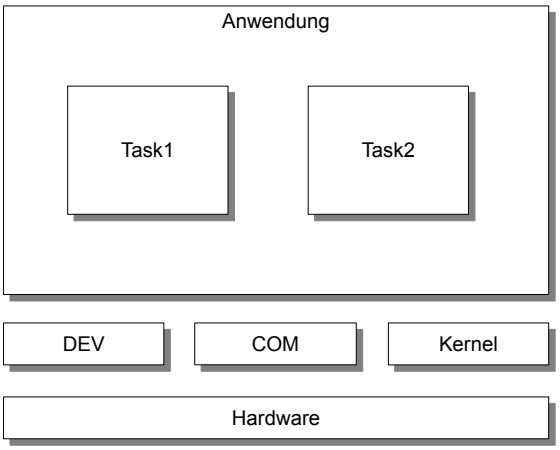

Abbildung 2: Mantis OS Architektur

4.3.2 DEV Layer

Eine weitere Hardwareabstraktion stellt die Geräteschicht dar. Sie ist aber nicht für die Nutzung aller Geräte notwendig, wie die Bezeichnung vermuten lässt, sondern bildet Ein- und Ausgabegeräte wie Sensoren oder den Speicher ab.
Bevor auf ein solches Gerät zugegriffen werden kann, muss es für den entsprechenden Vorgang reserviert werden und anschließend wieder freigegeben werden.
Auf Basis dieser Funktionalität können Gerätetreiber implementiert werden. Dies geschieht durch Programmierung der Schreib- und Lesevorgänge für das jeweilige Gerät innerhalb einer Funktion, deren Bezeichnung einem festgelegten Schema folgt.

4.3.3 NET Layer

Die Protokollschicht wird im NET Layer verankert und greift auf die Kommunikationsschicht zurück, um benutzerdefinierte Protokolle umzusetzen. So können beispielsweise verschiedene Routing Protokolle implementiert werden.

4.3.4 System Management

Auf Systemressourcen kann mittels weiterer Schnittstellen zugegriffen werden. Das Mantis System stellt auf diese Weise die Funktionalität zur Verwaltung von Threads und deren Synchronisation bereit. Für letzteres kann neben einem Mutex auch ein Semaphor verwendet werden.
Es stehen weiterhin Timer zur Festlegung von Standby-Zeiten zur Verfügung. Auch die zeitliche Planung der einmaligen Ausführung kleinerer Aufgaben ist hiermit möglich ohne größere Blöcke wie Threads benutzen zu müssen.

4.4 Programmierung

Das Mantis Betriebssystem kommt der Idee eines konventionellen Betriebssystems sehr nahe indem es dem C Programmierer eine Unix-ähnliche Umgebung bereitstellt.
Die wichtigsten Schnittstellen [8] für die Programmierung von Sensornetzwerken sind Aufrufe an den Task-Scheduler zur Erstellung (*mos_thread_new*) und Pausierung (*mos_thread_sleep*) von Threads, Methoden zum Senden (*com_send*) und Empfangen (*com_recv*) von Nachrichten, die Ansteuerung vorhandener LEDs (*mos_led_toggle*) sowie das Schreiben (*dev_write*) und Lesen (*dev_read*) von Sensorwerten.

4.5 Multitasking

Mantis OS ermöglicht von Haus aus keine ereignisbasierte Programmierung. Das ausschließlich verwendete, präemptive Multitasking stützt sich bei der Wahl für einen aktiven Thread auf das Round Robin verfahren.

4.6 Energiesparen

Mantis OS verwendet einen separaten Scheduler zur Einsparung von Energie. Dieser erkennt den Zustand, in welchem sich alle Threads in Ruhe befinden. Basierend auf den jeweiligen Zeiten, welche die einzelnen Threads in diesem Zustand verweilen werden, wird der Mikrocontroller ebenfalls für einen gewissen Zeitraum abgeschaltet. Das ermöglicht eine zufriedenstellende Energieeffizienz trotz Multitaskings.

4.7 Remote Shell

Die entfernte Verwaltung ist eine weitere Schlüsselfunktion von Mantis OS [1]. Gegenwärtig noch nicht realisiertes Ziel dabei ist die Remote-Programmierung von Sensorknoten, wobei die entfernte Anmeldung zur Manipulation von Variablen bereits implementiert ist.

5. COCOOS

Das wichtigste Merkmal der erst zwei Jahre jungen cocoOS Umgebung sind Koroutinen. Diese werden in Tasks abgebildet, welche in der main Funktion des verwendeten C Programms initalisiert werden.
Die Schlüsselidee der cocoOS Tasks besteht darin, die Programmausführung gezielt zu unterbrechen, um auf Events reagieren zu können. Diese Ereignisse bilden die Schnittstelle zwischen den einzelnen Tasks.
Das System ist unter einer BSD Lizenz frei verfügbar.

5.1 Dokumentation

Die offizielle Projekthomepage [18], liefert lediglich eine rudimentäre Übersicht über das Gesamtkonzept der Plattform. Weitere Quellen wie das offizielle Supportforum [19] bieten kaum eine nennenswerte Ergänzung hierzu.

5.2 Hardwareunterstützung

cocoOS kann auf eingebetteten Mikrocontrollern wie AVR und MSP430 Architekturen eingesetzt werden [18] . Es hat den Anspruch, für den Einsatz in heterogenen Netzwerken prädestiniert zu sein.

5.3 Architektur

Die wesentlichen Elemente des cocoOS Frameworks sind in Abbildung 3 zusammengefasst. Die relativ lose Struktur bietet dabei keine Hardwareabstraktion an. Ein Schichtenmodell, wie es das Mantis OS verwendet, wird nicht angewendet.

5.3.1 Tasks

Als Kernbestandteil von cocoOS verfügen Prozeduren über die wichtigsten Elemente zur Steuerung des Programmablaufs. Zu den Steuerungsmechanismen zählen das Erstellen, Löschen, Pausieren, Fortführen und Warten auf Tasks.

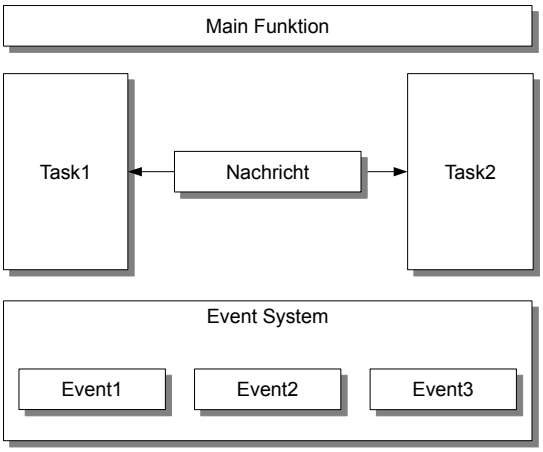

Abbildung 3: cocoOS Architektur

5.3.2 Events

Zur Synchronisation der einzelnen Tasks dienen Events. Sie können innerhalb eines Task-Rumpfes ausgelöst werden und werden unmittelbar danach automatisch wieder deaktiviert. Neben der Möglichkeit, das Auftreten eines einzelnen Events abzuwarten, kann dies ebenso für mehrere Events gleichzeitig geschehen.

5.3.3 Nachrichten

Zur Erweiterung der Events können Nachrichten verwendet werden, um zwischen Prozeduren zu kommunizieren. Solche Nachrichten werden von einem vordefinierten C Struct abgeleitet. Sie verfügen mindestens über einen Identifier und können vom cocoOS Kernel verzögert gesendet werden. Darüber hinaus können sie um beliebige Nutzdaten erweitert werden.

5.4 Programmierung

Die cocoOS Plattform wird in C programmiert. In der Main Funktion des Rahmenprogramms werden einzelne Prozeduren initialisiert. Deren Rumpf wird mittels weiterer C Funktionen implementiert. Weiterhin können eigene Formate für Nachrichten festgelegt werden. Die korrekte Benutzung von Semaphoren, zum Beispiel der Nachrichtenwarteschlange mittels *msg_q_get()* und *msg_q_give()*, bleibt dabei dem Programmierer überlassen.
Listing 1 veranschaulicht dies an einem Beispiel. Das Programm startet zwei Prozeduren welche sich innerhalb von Endlosschleifen im Wechsel Nachrichten und Events zukommen lassen.

Listing 1: Hello cocoOS

```
Evt_t event;

static Msg_t msg[16];

void task1(void) {
    task_open();
    for(;;) {
        /* Setze Semaphor und Empfange
            Nachricht */
        msg_q_get(task2);
        msg_receive(task2, &msg);
        msg_q_give(task2);
        /* Löse Event aus */
        event_signal(event);
    }
    task_close();
}

void task2(void) {
    task_open();
    for(;;) {
        /* Sende Nachricht */
        msg_q_get(task1);
        msg_post_in(task1, &msg, 1000);
        msg_q_give(task1);
        /* Warte auf Event */
        event_wait(event);
    }
    task_close();
}

int main(void)
{
    task_create(task2, 1, msg, 16, sizeof(
        Msg_t));
    task_create(task2, 2, NULL, 0, 0);
    os_start();
    return 0;
}
```

5.5 Semaphore

cocoOS stellt binäre Semaphore zur Verfügung und solche, welche Zählwerte annehmen können. Das verwendete Modell entspricht der Lösung des Consumer-Producer Problems. Bei der Programmierung ist auf die korrekte Verwendung der Semaphore zur Vermeidung von Konflikten zu achten.

5.6 Scheduling

Die cocoOS Tasks werden standardmäßig entsprechend ihrer Priorität ausgeführt. Dabei beeinflussen sich die einzelnen Tasks maßgeblich gegenseitig in ihrer Priorität. So können sie einerseits Events auslösen und andererseits Nachrichten direkt an andere Tasks senden.
Weiterhin besteht die Möglichkeit, diesen Vorgang explizit um das Round Robin Verfahren für die Zuteilung von Prozessorzeit zu erweitern.

5.7 Timer

cocoOS stellt mehrere Timer zur Verfügung. Neben dem Zugriff auf die Systemuhr besteht die Möglichkeit, weitere Timer zu Verwenden. Diese können individuell gesteuert werden, beispielsweise um die Länge einer übertragenen Nachricht zu überwachen.
Der primäre Timer wird per *os_tick()* anhand der benötigten Hardware Uhr aufgerufen und dekrementiert dabei die System Timer. Eigene Timer können mittels per *os_sub_tick(id)* verwendet werden. Dies bietet auch die Möglichkeit, mittels *os_sub_nTick(id, nTicks)* mehr als einen Schritt zu erhöhen.

doi: 10.2313/NET-2012-08-2_01

6. .NET MICRO FRAMEWORK

Das .NET Micro Framework [10] ist Teil der Microsoft Windows Embedded Familie [13] zum Einsatz in Hardwareumgebungen mit begrenzten Ressourcen. Es ist die offizielle Fortführung der Windows CE Familie. Das Framework interpretiert auszuführenden Programmcode anstatt ihn zu kompilieren. Damit handelt es sich nicht um ein Echtzeit–Betriebssystem.
Der Fokus liegt auf MIPS-, ARM- und x86-Prozessoren [3], womit auch die Möglichkeit zur gleichzeitigen Abhandlung mehrerer Ausführungsstränge gegeben ist [6].

6.1 Dokumentation

Der Hersteller ist sichtlich darum bemüht, Entwicklern eine breite Unterstützung für das Micro Framework anzubieten. Der zur Verfügung gestellte Internetauftritt [15] legt starken Fokus auf die Vernetzung der Entwickler untereinander. Er ist mit ausführlichen Materialien ausgestattet und bietet neben Verweisen auf Blogeinträge, Foren und einem obligatorischen Wiki auch Videomaterial an. Dies ermöglicht einen

6.2 Hardwareunterstützung

Das .NET Micro Framework benötigt zur Ausführung mindestens 64KB RAM und 256KB Flash Speicher [14].
Damit die Umgebung direkt auf Hardware eingesetzt werden kann, müssen der Hardwareabstraktionsschicht die entsprechenden Gerätetreiber einzeln beigefügt werden. Aktuell sind keine Treiber für WE Compact 7.0 online aufgeführt [16]. Alternativ sind vorgefertigte Pakete für eine Vielzahl an Zielplattformen vorhanden (siehe Abschnitt 6.3.1).

6.3 Architektur

Der Aufbau des .NET Micro Framework gliedert sich in klar abgegrenzte Schichten. Durch diese Struktur sind Programmierer bei der Umsetzung von Anwendungen im Programmaufbau wenig eingeschränkt. Jedoch ist zur Entwicklung Microsoft's Visual Studio für .NET Anwendungen erforderlich.

6.3.1 Board Support Packages

Der Hersteller bietet sogenannte Board Support Packages [3], welche für ein breites Hardwarespektrum verfügbar sind [17], an. Diese Pakete enthalten bereits, wie in Abbildung 4 gezeigt, die benötigten Gerätetreiber für die jeweilige Zielplattform sowie eine Abstraktion des verwendeten Kernels und eine Hilfseinrichtung für die Fehlersuche während des Entwicklungsprozesses. Für die Erstellung eigener Pakete werden vom Hersteller Vorlagen zur weiteren Anpassung zur Verfügung gestellt.

6.3.2 OEM Adaption Layer

Die hardwareabhängigen Funktionen werden dem Anwendungsprogrammierer mittels dem OEM Adaption Layer bereitgestellt. Dieser bildet eine Schnittstelle zum verwendeten Kernel abhängig vom verwendeten Mikrokontroller. Dabei wird zur vereinfachten Organisation dieser äußerst komplexen Komponente die Verwendung einheitlicher Bibliotheken innerhalb derselben Chipfamilie vorgeschlagen.

6.3.3 Kernel-Independent Transport Layer

Während des Programmiervorgangs kann zu Debugging Zwecken der Kernel-Independent Transport Layer verwendet werden. Diese Schicht wird vom OEM Adaption Layer (OAL)

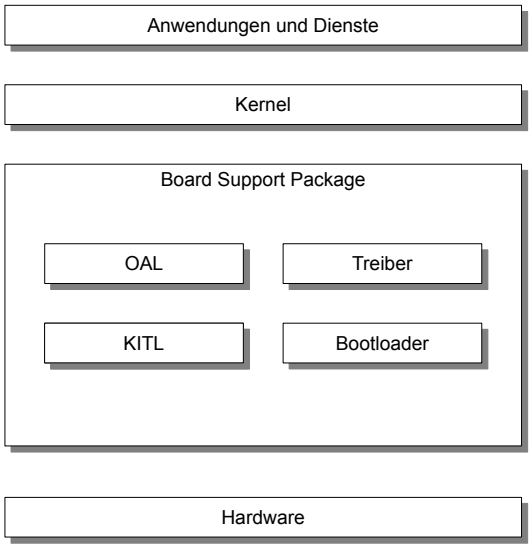

Abbildung 4: .NET Micro Framework Architektur

aufgerufen und ermöglicht das registrieren eigener Kommunikationsdienste. Sie stellt eine Schnittstelle zwischen der Arbeitsumgebung des Entwicklers und der Zielplattform dar.

6.3.4 Bootloader

Für die Zielplattform wird ein Bootloader zur Initialisierung des Betriebssystems benötigt. Der Bootloader stellt einen weiteren Bestandteil der Board Support Packages dar.
Das Speicherlayout kann über Konfigurationsdateien festgelegt werden, welche ebenfalls Bestandteil der Board Support Packages sind und zusätzlich zur Konfiguration der Speicherabbilder von Bootloader und Laufzeitumgebung verwendet werden.

7. ZUSAMMENFASSUNG

Zunächst fällt auf, dass cocoOS ein Nischendasein führt. Im Vergleich zu populäreren Systemen wie TinyOS gibt es lediglich eine geringe Anzahl an Fachpublikationen. Das cocoOS Support Forum [11] ist noch leer und die Dokumentation spärlich. Die ausführlichen Beschreibungen des .NET Micro Frameworks bilden dazu einen starken Kontrast.
Die Sicherheitsmechanismen der betrachteten Systeme sind rudimentärer Natur. Dies ist prinzipbedingt den knappen Ressourcen der Zielplattformen geschuldet.
Hinsichtlich des Arbeitsflusses wärend der Programmierung ist das .NET Micro Framework erste Wahl, sofern neben einer geeignete Zielplattform auch eine geeignete Entwicklungsplattform aus dem Hause Microsoft verfügbar ist. Dabei werden Kenntnisse im Umgang mit .NET vorausgesetzt. Andernfalls steht Mantis OS mit breiter Hardwareunterstützung zur Verfügung. Dieses hinterlässt insgesamt den besten Gesamteindruck im Hinblick auf den Einsatz in Sensornetzwerken. Für kleine Sensornetze empfiehlt sich cocoOS, wobei hier die Auswahl an Zielplattformen eingeschränkt ist.

Die Sicherheit der beiden Betriebssysteme Mantis OS und

Tabelle 1: Vergleichsmatrix

	MantisOS	cocoOS	.NET
Dokumentation	⊕⊕	⊖	⊕
Energiesparen	⊕		
Hardwareunterstützung	⊕⊕	⊕	⊕⊕
Sicherheit			
Programmierung	⊕⊕	⊕	⊖
Multitasking	⊕	⊕⊕	
Footprint	⊕	⊕	
Flache Lernkurve	⊕	⊕⊕	⊖⊖

cocoOS muss vom Anwender protokollseitig bei der Datenübertragung umgesetzt werden. Dabei geht man davon aus, dass dem unberechtigten, physischen Zugriff auf einen Sensorknoten beim aktuellen Stand der Technik kaum etwas entgegenzusetzen ist.

Eine Zusammenstellung der jeweiligen Stärken und Schwächen der betrachteten Betriebssysteme wurde in Tabelle 1 als Vergleichsmatrix erstellt. Dabei dürfen die angegebenen Gewichte als relative Metrik verstanden werden.

8. AUSBLICK

Die voranschreitende Weiterentwicklung von Hardwaremodulen, welche in Sensorknoten eingesetzt werden können, wird langfristig zu komplexeren Lösungen für Bestriebssysteme in Sensornetzen führen.
Damit wird nicht nur die Leistungsfähigkeit hinsichtlich der Erfassung wachsender Datenvolumen zunehmen, sondern auch die Sicherheitsanforderungen. Abhörsicherheit und Vertraulichkeit werden in demselben Maße an Bedeutung gewinnen. Schließlich ist zu erwarten, dass die Handhabung der Software für Sensorknoten weiter vereinfacht werden kann. Sensornetzwerke werden damit Einzug in Privatwohnungen halten und dort den technischen Fortgang, beispielsweise in der Heimautomatisierung, entscheidend prägen. Diese Entwicklung wird zusätzlich befeuert durch die Idee des *Internet of Things* [2] und der Etablierung von IPv6 Netzwerken zur globalen Adressierung kleinster elektrischer Geräte.
Damit ist zu erwarten, dass der Sicherheitsaspekt bei der Entwicklung von Betriebssystemen für Sensorknoten in den Fokus rückt ebenso wie die Leistungsfähigkeit hinsichtlich automatischer Venetzung. Dabei wird die wichtigste Herausforderung weiterhin im Ressourcenmanagement bestehen. Denn einerseits werden zwar die Hardwarekomponenten stetig leistungsfähiger. Doch andererseits werden gleichzeitig auch die Sensorknoten stetig kleiner.

9. LITERATUR

[1] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10(4):563–579, Aug. 2005.

[2] L. Coetzee and J. Eksteen. The internet of things - promise for the future? an introduction. In *IST-Africa Conference Proceedings, 2011*, pages 1 –9, May 2011.

[3] W. Giberson and J. Chan. *The Basics of Bringing up a Hardware Platform*. Microsoft, March 2011.

[4] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 271–278, New York, NY, USA, 1999. ACM.

[5] M. Krüger, R. Karnapke, and J. Nolte. Controlling sensors and actuators collectively using the cocos-framework. In *Proceedings of the First ACM workshop on Sensor and actor networks*, SANET '07, pages 53–54, New York, NY, USA, 2007. ACM.

[6] G. Langer and J. Chan. *Symmetric Multiprocessing Guide for Windows Embedded Compact 7*. Microsoft, November 2011.

[7] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1mm3 die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 402 –404, February 2012.

[8] M. M. R. Mozumdar, L. Lavagno, and L. Vanzago. A comparison of software platforms for wireless sensor networks: Mantis, tinyos, and zigbee. *ACM Trans. Embed. Comput. Syst.*, 8(2):12:1–12:23, Feb. 2009.

[9] X. Wang, F. Silva, and J. Heidemann. Infrastructureless location aware configuration for sensor networks. In *Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on*, pages 174 – 183, December 2004.

[10] *.NET Micro Framework Homepage*, `http://www.microsoft.com/en-us/netmf/about/default.aspx`

[11] *cocoOS Support Forum*, `http://www.cocoos.net/forum/`

[12] *TelosB Datasheet*, `http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`

[13] *Windows Embedded Homepage*, `http://msdn.microsoft.com/de-DE/windowsembedded/`

[14] *.NET Micro Framework - Getting started*, `http://www.microsoft.com/en-us/netmf/about/gettingstarted.aspx`

[15] *Windows Compact*, `http://www.microsoft.com/windowsembedded/en-us/develop/windows-embedded-compact-7-for-developers.aspx`

[16] *Windows Compact Device Drivers*, `http://www.microsoft.com/windowsembedded/en-us/downloads/windows-embedded-driver-chooser.aspx?fsr=1&wince=1&WEItemsPerPage=10&sort=ReleaseDateConverted_desc`

[17] *Windows Compact Board Support Packages*, `http://www.microsoft.com/windowsembedded/en-us/downloads/board-support-packages-for-windows-embedded.aspx`

[18] *cocoOS Homepage*, `http://www.cocoos.net/`

[19] *cocoOS Support Forum*, `http://www.cocoos.net/forum/`

[20] *Mantis OS Homepage*, `http://mantisos.org/`

[21] *MANTIS: Programming Guides*, `http://mantisos.org/index/tiki-index.php%3Fpage=Programming+Guides.html`