Universität Karlsruhe (TH)
Fakultät für Informatik
Telecooperation Office

Mobile und verteilte Systeme

Ubiquitous Computing

Teil III

Seminar WS 2005/06

Herausgeber:

Tobias Zimmer
Albert Krohn
Christian Decker
Till Riedel
Dr. Michael Beigl

Universität Karlsruhe
Telecooperation Office, TecO

Interner Bericht 03/2006
ISSN 1432-7864

Inhaltsverzeichnis

Vorwort

Das Seminar „Mobile und verteilte Systeme – Ubiquitous Computing“ am Telecooperation Office (TecO) erfreut sich großer Beliebtheit. Der vorliegende interne Bericht enthält die studentischen Beiträge zu diesem Seminar, das im WS 2005/06 in dieser Form zum vierten Mal stattgefunden hat.

Die Themenauswahl für das Seminar orientiert sich im wesentlichen an aktuellen wissenschaftlichen Fragestellungen in den Bereichen:

- Kontexterfassung und Verarbeitung
- Kommunikation in ubiquitären Informationssystemen
- Sicherheitsaspekte in ubiquitären Systemen
- Technologien für das Ubiquitous Computing
- Auswirkungen ubiquitärer Technologien
- Softwaretechnik für ubiquitären Informationssystemen

Auf grund des stetig wachsenden Interesses der Studenten an diesen Themenbreichen haben wir uns entschlossen einen Seminarband mit den Beiträgen unserer Studenten als internen Bericht zu veröffentlichen. Durch die engagierte Mitarbeit der beteiligten Studenten wird ein Ausschnitt aus diesem komplexen und umfassenden Themengebiet klar und übersichtlich präsentiert. Für den Fleiß und das Engagement unserer Seminarteilnehmer wollen wir uns an dieser Stelle daher herzlich bedanken.

Bestärkt durch die gute Resonanz der Studenten, werden wir dieses Seminar auch im nächsten Wintersemester – dann natürlich mit aktualisierten Themen – wieder anbieten.

Minimalistische Kerne für Sensorknoten

Alex Günter

Telecooperation Office, Institut für Telematik, Universität Karlsruhe (TH)
alex.guenter@gmx.de

Zusammenfassung. Mikrocontroller und Sensorknoten müssen eine Reihe von Aufgaben bewältigen. Hierbei wird der Einsatz von Betriebssystemen immer wichtiger. Die eingeschränkten Ressourcen solcher Geräte führen zu kompakten Betriebssystemen in Form von minimalistischen Kernen.

1 Einleitung

Betriebssysteme haben die Aufgabe, eine einfache Schnittstelle für den Programmierer zu bieten und die Ressourcen eines Systems zu verwalten. Hierbei werden an Betriebssysteme für Mikrocontroller und Sensorknoten spezielle Anforderungen gestellt, da in solchen Systemen bestimmte Einschränkungen vorherrschen. Allein durch den vergleichsweise gering vorhandenen Speicher ergibt sich die Notwendigkeit für minimalistische Kerne, welche nur die wichtigsten Aufgaben übernehmen.

Als erstes werde ich die verschiedenen Plattformen vergleichen, danach analysiere ich die Anforderungen an Betriebssysteme für Mikrocontroller und Sensorknoten. Hierauf folgt die Vorstellung der untersuchten Betriebssysteme. Anschließend folgen ein Ausblick und eine tabellarische Übersicht der wichtigsten Funktionen der einzelnen Systeme.

2 Plattformvergleich

Mainframes stehen in großen Rechenzentren von Firmen oder Universitäten. Ihre Aufgabe besteht in der Bewältigung einer großen Anzahl von Ein- und Ausgaben. Dazu besitzen sie teilweise tausende von Festplatten mit Terrabyte-großen Speichern. Sie dienen heutzutage als Web-Server, Server für E-Commerce und Server für Business-to-Business Anwendungen. Die Betriebssysteme, welche auf Mainframes laufen, sind für die gleichzeitige Ausführung vieler Prozesse konzipiert. Diese Prozesse benötigen schnelle Ein- und Ausgabe. Beispielsweise

haben Banken oder Flugbuchungen relativ kleine Datenmengen bei Transaktionen, aber pro Sekunde werden davon Tausende ausgeführt.

Server in kleinen und mittleren Unternehmen dienen dazu, vielen Benutzern Software- und Hardware-Ressourcen über Netzwerke anzubieten. Beispiele hierfür sind Druckdienste, Dateifreigaben und Webdienste. Hierzu werden häufig Multiprozessorsysteme eingesetzt. Eine der Hauptaufgaben des Server-Betriebssystems ist es, die Kommunikation und Koordination der Prozesse optimal einzuteilen.

PCs sind multifunktional. Sie haben genügend Festplattenspeicher, um Multimediadienste bereitzustellen. Der Hauptspeicher ist groß genug um mehrere großen Anwendungen zugleich ablaufen zu lassen und die Prozessoren sind schnell genug um dem Nutzer in komfortabler Weise, am besten über GUI Systeme, die Ergebnisse zu liefern. Somit müssen auch die eingesetzten Betriebssysteme für viele unterschiedliche Anforderungen und Aufgaben ausgelegt sein. Da Personalcomputer die Interaktion mit dem Benutzer in den Vordergrund stellen, ist die Reaktivität des Systems von großer Bedeutung. Es muss schnell auf Benutzereingaben reagieren, denn der Anwender ist nicht bereit, Wartezeit bei Dialogsystemen in Kauf zu nehmen.

Bei den ersten drei hier vorgestellten Plattformen sind Hardware und Betriebssystem miteinander verzahnt. Die eingesetzten Prozessoren unterstützen das Betriebssystem mit Schutzmechanismen, indem sie einen privilegierten Modus für Betriebssystembefehle bereithalten und im Benutzermodus keine Befehle direkt an die Hardware zulassen, vermeiden sie beispielsweise Abstürze durch schlecht programmierte Anwendungen. Die Hardware stellt auch eine Speicherverwaltungseinheit bereit, welche vom Betriebssystem genutzt wird, um virtuelle Speicherseiten effizient ansprechen zu können. Die Zusammenarbeit von Hard- und Software erhöht die Sicherheit und die Leistung des Gesamtsystems.

Mikrocontroller sind im Prinzip Mikrorechner auf einem Chip. Hierbei sind Prozessorkern, Speicher und Ein-/ Ausgabeschnittstellen auf einem Baustein vereint. Eine große Anzahl an Peripheriegeräten kann über verschiedenste Kommunikationskanäle verbunden werden (l2c, can-bus, pci, scsi, profibus, usb). Zur Messung von analogen Eingangssignalen stehen Analog-Digitalwandler zur Verfügung. Es gibt Mikrocontroller in diversen Ausführungen, eingeteilt in die so genannten Mikrocontrollerfamilien. Da ihr Einsatzgebiet meist das Steuern und Regeln von technischen Anlagen ist, spielt die Rechenleistung eine eher untergeordnete Rolle. Die Palette reicht von Prozessoren mit wenigen MHz und 8 Bit breiten Registern bis hin zu einigen hundert MHz mit 32 Bit oder mehr1. Die Vielfalt übersteigt die Anzahl der verfügbaren Prozessoren für PCs, Server und Mainframes zusammen um ein Weites. Der Speicher ist bei den meisten Mikrocontrollern als Harvard-Architektur aufgebaut. Hierbei liegen Programme und Daten in getrennten Speichern. Bei den oben genannten Systemen hingegen

[1] siehe Brinkschulte, Ungerer, Mikrocontroller und Mirkoprozessoren, Springer-Verlag 2002

wird die Von-Neumann-Architektur benutzt, in der Programme und Daten im gleichen Speicher liegen.

Eingebettete Systeme dienen der Steuerung und Regelung. Hier werden Mikrocontroller oder Mikroprozessoren in ein technisches Umfeld eingebettet. Sie sind manchmal nicht auf den ersten Blick als Computer zu erkennen, z.B. in der Kaffeemaschine oder als Leitrechner eines Fließbandes. Die Anzahl der zur Verfügung gestellten Schnittstellen ist meist höher, als die eines reinen Rechnersystems, da eine Vielzahl an Sensoren und Aktuatoren angeschlossen können werden muss. Es gibt spezielle Anforderungen an die mechanische Belastbarkeit. Die Bauweise wird durch das Einsatzgebiet diktiert, sei es mechanisch stabil als Industrie-PC oder ein extrem kleines Gehäuse eines mobilen Endgerätes. Bei mobilen Geräten muss auch der Energieverbrauch minimiert werden, da die Batterie einen limitierenden Faktor darstellt. Die Prozessorleistung muss limitiert werden, wenn die Abwärme eine Rolle spielt. Bestimmte Maßnahmen zur Gewährleistung der Zuverlässigkeit, z.B. ein Notbetrieb beim Einsatz in einem Atomkraftwerk, sind unabdingbar. Eine weitere Anforderung stellt die Einhaltung von Zeitbedingungen dar, d.h. der vom Betriebssystem unterstützte Echtzeitbetrieb, bei dem garantierte Zeitschranken eingehalten werden müssen. Hierfür werden Mikrocontroller ohne Pipelines oder Cache eingesetzt, um die zeitliche Vorhersagbarkeit gewährleisten zu können.

Auf Sensorknoten sind zumeist Mikrocontroller im Bereich von ein bis 20 MHz verbaut mit 8 Bit breiten Registern. Sensorknoten sind für die Überwachung von Umweltdaten oder die Lokation von Gegenständen konzipiert. Der Speicher ist aufgeteilt in nichtflüchtigen Festwertspeicher und flüchtigen Schreiblesespeicher (RAM), wobei der Speicher insgesamt im hunderter-Kbyte Bereich liegt. Sensorknoten haben zusätzlich zum Mikrocontroller Sensoren und Kommunikationserweiterungen. Sensorknoten sind batteriebetrieben, daher wurden spezielle Vorkehrungen getroffen, um Energie zu sparen. Das Betriebssystem unterstützt hierbei die Hardware, indem es den Prozessor schlafen legt, einzelne nichtbenötigte Sensoren vorübergehend oder den Knoten komplett abschaltet. Aufgrund dieser und weiteren Einschränkungen werden spezielle Anforderungen an das Betriebssystem gestellt.

Eine Übersicht über die typischen Eigenschaften der Betriebssysteme findet sich in Tabelle 1.

Tabelle 1. Übersicht Platformen

Platform	Typische Eigenschaften
Mainframes	Hohes IO Aufkommen Von-Neumann-Architektur
Server	Kommunikation und Koordination der Prozesse Von-Neumann-Architektur
PCs	Interaktion mit Benutzer Von-Neumann-Architektur
Mikrocontroller	Peripherieanbindung Kommunikation Analog Digital Wandler Steuerung und Regelung Harvardarchitektur
Eingebettete Systeme	Engerhieverbrauch Harvardarchitektur
Sensorknoten	Hohe Anzahl Sensoren Energieverbrauch Harvardarchitektur

3 Anforderungsanalyse

Ich möchte an dieser Stelle eine Anforderungsanalyse für Mikrocontroller und Sensorknoten durchführen. Bedingt durch die Einschränkungen von Mikrocontrollern und Sensorknoten ergeben sich bestimmte Anforderungen an darauf laufende Betriebssysteme.

Prinzipiell hat ein Betriebssystem folgende Anforderungen zu erfüllen:

- Prozessormanagment
- Prozessmanagment
- Speichermanagment
- Gerätemanagment

Bei Mikrocontrollern, welche z.B. bei der Fahrstuhlsteuerung oder im Auto verbaut werden, werden sehr hohe Anforderungen an die Rechtzeitigkeit der Systeme gestellt, welche durch das Prozessmanagment sichergestellt werden müssen. Bei solchen Einsatzgebieten spielt der Energieverbrauch nur eine untergeordnete Rolle, meist sind Energiequellen in geeignetem Umfang vorhanden. Das Prozessormanagment ist hierbei nicht zwingend auf Standby oder

Schlafmodus ausgerichtet. Meist wird ein Betriebssystem für eine Aufgabe entwickelt, bei der nur spezielle Geräte verbaut werden. Das Gerätemanagment muss an die Hardware angepasst werden. Ein modularer Aufbau ist nicht notwendig.

Sensorknoten hingegen werden mit Batterien betrieben. Die Stromaufnahme stellt hier eine limitierende Größe des Systems dar. Auch die höhere Anzahl an Sensoren und Aktuatoren zwingen dem Betriebssystem eine andere Gliederung auf. Es kommt wahrscheinlich häufiger vor, bei einem Sensorknoten verschiedenste Arten von Sensoren zu testen oder bei einer neuen Generation von Sensoren den alten Kern beibehalten zu wollen. Auch kann sich im Laufe eines Sensorknotenlebens die Anwendung ändern. Hier muss das Betriebssystem eine geeignete Modularität aufweisen können, um Treiber oder Programme laden zu können. Sensorknoten werden in großer Stückzahl zur Erfassung von Umweltdaten, über große Areale verteilt, eingesetzt. Hierfür besitzen Sensorknoten die Möglichkeit, drahtlos zu kommunizieren. Dies kann auch dazu benutzt werden, um Updates Over-the-Air zu verteilen.

Das Betriebssystem muss in geeigneter Weise ein Prozessormanagment bereitstellen, um den Prozessor schlafen zu legen, wenn dieser keine Aufgaben zu bewältigen hat. Hier muss ein geeignetes Scheduling-Verfahren dafür Sorge tragen, dass dies so oft wie möglich geschieht. Andererseits müssen eingehende Ereignisse über Unterbrechungen behandelt werden. Da mehrere Sensoren für Datenströme sorgen müssen diese entsprechend behandelt werden können. Da es in Sensorknoten keine oder nur geringe Möglichkeit zur Pufferung der anfallenden Daten gibt, müssen diese sofort behandelt werden, ein Datenverlust wäre andernfalls die Folge. Hierbei kann es durch das gleichzeitige Eintreffen von Unterbrechungen zu konkurrierenden Ereignissen kommen. Durch die Vergabe von Prioritäten oder das entsprechende Scheduling müssen diese behandelt werden.

4 Vorstellung der untersuchten Betriebssysteme

Untersuchungsgegenstand waren vier Betriebssysteme für Mikrocontroller und drei Betriebssysteme für Sensorknoten. Bei Mikrocontrollern handelt es sich im speziellen um FreeRTOS, Pico]OS, XMK und PicOS. Im Umfeld der Sensorknoten waren es SOS, Contiki und AmbientRT. Im Folgenden werde ich auf die Eigenschaften der einzelnen Systeme näher eingehen und, wenn möglich, den Aufbau des Betriebssystems darstellen.

4.1 Betriebssysteme für Mikrocontroller

4.1.1 FreeRTOS (http://www.freertos.org/)

FreeRTOS ist ein Echtzeitbetriebssystem für eingebettete Systeme, es existieren Portierungen für viele Mikrocontroller (siehe Anhang). FreeRTOS ist ein freies Betriebssystem. Es kann kostenlos von der Homepage heruntergeladen und für die eigenen Bedürfnisse angepasst werden. Der Scheduler wurde unter dem Gesichtspunkt der Minimalität entwickelt, er ermöglicht preemptives oder kooperatives Scheduling. Bei Echtzeit-Task (siehe Abbildung 1) besteht die Möglichkeit der Prioritätenvergabe, wobei Null den Idle-Task bezeichnet, welcher immer dann aktiv wird, wenn keine Rechenzeit benötigt wird. Umso niedriger der vergebene Wert des Tasks, desto niedriger seine Priorität. Tasks können auch dieselbe Priorität haben, dann teilen sie sich die CPU Zeit. Dies wird über volle Verdrängung mit Round-Robin Zeitscheiben realisiert. Wird kooperatives Scheduling verwendet, müssen die Tasks freiwillig über einen Yield() Aufruf ihre Rechenzeit abgeben oder sie blockieren und es wird hierdurch eine Kontextwechsel erreicht. Der Idle-Task wird automatisch beim Erzeugen des ersten Tasks per yTASKCreate() mit erzeugt. Seine Aufgabe besteht darin, den Speicher, welcher vom Betriebssystem für eine Task reserviert wurde, beim Löschen einer Task aufzuräumen.

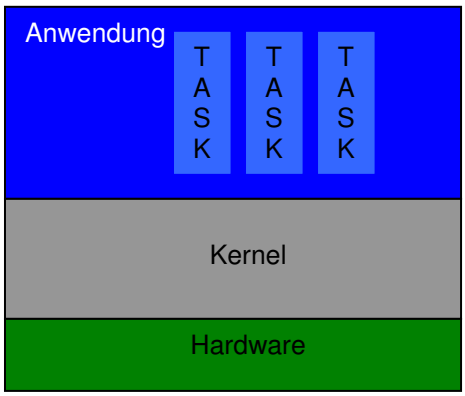


Abbildung 1: Aufbau FreeRTOS

Der Kernel stellt Werkzeuge zur Erleichterung der Programmierung zur Verfügung. Es gibt Warteschlange, in die Elemente als Kopie, nicht als Referenz gespeichert werden können. Bei großen Elementen ist es daher sinnvoll, nur einen Zeiger in die Warteschlange zu legen. Des Weitern gibt es Semaphoren, welche durch Makros generiert werden und auf Warteschlangen aufbauen. Die Speicherverwaltung ist in drei Schemata aufgeteilt. Der erste Fall ist am besten

geeignet für Anwendungen, welche niemals ihre Tasks oder Warteschlangen löschen. Hierbei wird der einmalig alloziierte Speicher nicht mehr freigegeben. Das zweite Schema sollte angewandt werden, falls mehrmals Tasks bzw. Warteschlangen erzeugt und gelöscht werden sollen. Hierbei darf der zu alloziierende Speicher aber keine zufällige Größe haben. Dieses Schema ist nicht deterministisch. Das dritte Schema wird für gewöhnliche Speicheranforderungen und Freigaben benötigt. Es wird beispielsweise bei Anwendungen benutzt, welche auf der PC-Portierung von FreeRTOS laufen.

Eine Beispielanwendung ist die Implementierung eines eingebetteten Webservers. Dieser kann über TCP/IP kommunizieren. Es wurde beispielsweise aufgezeigt, wie ein eingebettetes System über seine Netzwerkschnittstelle mit einem PC oder Router verbunden werden kann und durch die Eingabe der IP im Browser des PCs Seiten angezeigt werden können, welche die aktuellen Stadien der Tasks des eingebetteten Systems in Form einer Tabelle darstellt. Hierbei lief der Webserver und 31 weitere Demo-Tasks auf einem SAM7x ARM Board. Es ist sogar möglich, eigene CGI Skripte auf dem Server laufen zu lassen. Hierdurch wird dynamischer HTML Code generiert.

4.1.2 Pico]OS (http://picoos.sourceforge.net)

Pic]OS wurde von Dennis Kuschel, Bremen, Deutschland entwickelt. Auch Pico]OS ist ein freies Echtzeitbetriebssystem. Es ist, wie FreeRTOS, für verschiedene Plattformen und Prozessorfamilien erhältlich. Es unterstützt Low-End 8-Bit Mikroprozessoren, 16-Bit und 32-Bit. Der Aufbau ist schichtenbasiert (siehe Abbildung 2), auf der untersten Ebene, pico-layer genannt, läuft der Kern welcher alle grundlegenden Betriebssystemfunktionen wie Taskverwaltung, Ressourcenverwaltung, Interprozessokommunikation und Zeitgeber enthält. Komplexere Funktionen sind im nano-layer angesiedelt. Darauf aufbauend soll der micro-layer implementiert werden, welcher Treiber und ein Dateisystem enthalten wird, aber zum jetzigen Zeitpunkt noch nicht fertig gestellt ist. Durch den Aufbau als Schichten können, je nach darunter liegender Hardware, z.B. bei eingeschränkten Ressourcen nur der pico-layer genutzt werden. Der Scheduler unterstützt zwei verschiedene Betriebsmodi. Zum einen Standard prioritätenbasiertes oder Round-Robin Scheduling. Hierbei ist die Anzahl der Tasks durch den eingesetzten Prozessor limitiert. Bei 8-Bit Prozessoren sind maximal 64 Tasks erlaubt, bei 32-Bit Prozessoren sind es hingegen 1024 Tasks. Im nano-layer kann ein Speichermanager eingebunden werden, hierdurch kann dynamisch Speicher alloziert werden. Des Weiteren werden zur Interprozesskommunikation sogenannte „Message Boxes“ verwendet. Hierbei besitzt jede Task eine Message Box, worin die Nachrichten gesammelt werden. Die Nachrichten können entweder Puffer oder Zeiger sein. Listen können dazu benutzt werden, Daten, entweder als einfache Elemente oder in einer Queue (Warteschlange), zu speichern. Hiervon es gibt blockierende und nicht-blockierende Varianten. Es können Flags gesetzt werden, um auf Ereignisse zu warten. Ein Thread hat besitzt die Möglichkeit, auf mehrere Flags gleichzeitig zu warten. Die Anzahl der möglichen Flags ist nur durch die darunter liegende

Architektur beschränkt. Zur Task-Synchronisation können Mutexe eingesetzt werden. Teile eines Codes, welche mit einem Mutex geschützt werden, können jeweils nur von einem Thread gleichzeitig ausgeführt werden. Wollen andere Threads auf diesen Teil zugreifen und den Mutex nutzen werden sie entweder blockiert oder sie können in einer Schleife den Zustand des Mutex abfragen bis dieser frei wird. Des Weiteren besteht die Möglichkeit, Semaphoren zur Synchronisation einzusetzen. Um Hardware Interrupts für Threads zugänglich zu machen, besteht die Möglichkeit, diese mit Software Interrupts zu koppeln. Diese werden aufgerufen, wenn die Hardware Interrupts aktiviert werden. Alle Software Interrupts werden in einer globalen Liste geführt.

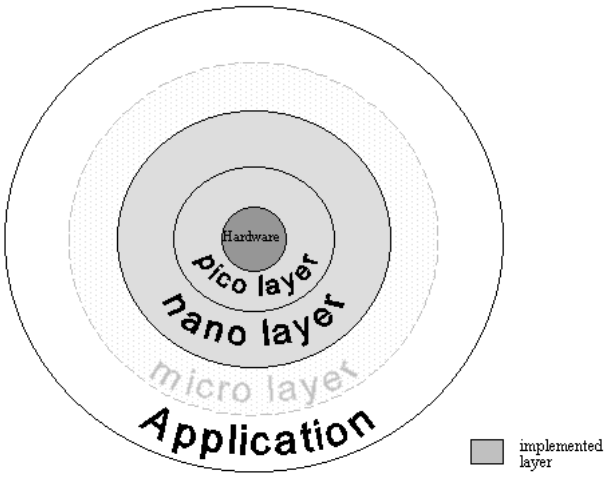


Fig.: pico]OS Layer Scheme

Abbildung 2: pico]OS Schichtenmodell

4.1.3 XMK (http://www.shift-right.com/xmk/)

Genau wie Pico]OS und FreeRTOS ist XMK ein Echtzeitbetriebssystem für 8, 16 oder 32-Bit Mikrocontroller. Auch dieses Betriebssystem für Mikrocontroller ist ein freies Betriebssystem, diesmal unter BSD Lizenz. Es besitzt eine hohe Skalierbarkeit. Es ist von einem reinen Thread-Scheduler bis hin zu einem kompletten Echtzeitbetriebssystem mit TCP/IP Netzwerkkommunikation einsetzbar. Bei XMK existiert eine Trennung in zwei Schichten (siehe Abbildung 3). Die untere Schicht ist der sogenannte XMK Scheduler, welcher nur den Kern-Scheduler und die Kernel-Schnittstellen beinhaltet. Der darauf aufsetzende Teil ist der „Application Programming Layer (APL)“. Dies ist eine Bibliothek, welche kernel- und plattformunabhängig ist und Schnittstellen zur Kommunikation zwischen Threads, Heap-Speicher, Speicherbereiche, Dateidiskreptoren usw. bereitstellt. Eine Schicht über der APL sitzen die eigentlichen Anwendungen, welche über die Schnittstellen des APL laufen. Der XMK-Scheduler ist ein

statischer Scheduler, insofern als dass er statisch mit der Anwendung zur Kompilierzeit gelinkt wird. Hieraus wird eine einzige Binärdatei (ausführbares Image), welche auf das Ziel aufgespielt wird. Der Scheduler geht davon aus, dass der Kernel und die Anwendung sich einen gemeinsamen Adressraum teilen. Es gibt in XMK zwei Arten von Threads: statische und dynamische. Statische Threads werden vor dem Start des Schedulers erzeugt und niemals beendet. Dynamische Threads hingegen werden nur nach dem Start des Schedulers erzeugt und können beendet werden. Es können beide Arten von Threads in einer Anwendung benutzt werden. Das Scheduling unterstützt zurzeit nur preemptives Scheduling mit Prioritäten. Unterbrechungen werden in einen Interrupt-Vektor eingehängt. Threads werden mit Hilfe von Semaphoren synchronisiert. Um Konflikte beim gleichzeitigen Zugriff auf dieselbe Ressource zu vermeiden existieren Mutexe. Das „Highlight“ bei XMK ist der APL Layer. Dieser ist prinzipiell plattformunabhänging, auch wenn er ursprünglich für den XMK-Scheduler konzipiert war. Hieraus folgt, dass alle Programme, welche auf Basis der APL geschrieben wurden, auf jedem System lauffähig sind, auf welches APL portiert wurde. Die Programme sind zwar nicht binär-kompatibel, können aber ohne Veränderung mit der entsprechenden APL Bibliothek gelinkt und kompiliert werden. Die Schicht zwischen APL und XMK-Scheduler ist der Hardware Abstraction Layer. Nur dieser muss prinzipiell portiert werden, wenn die APL auf anderen Mikrocontroller eingesetzt werden soll. Hier werden alle direkten Zugriffe auf die Hardware realisiert, diese werden als Schnittstelle der APL zur Verfügung gestellt. Eine korrekte Implementierung auf dem neuen Mikrocontroller ist die einzige Vorraussetzung für den korrekten Betrieb der Schichten darüber.

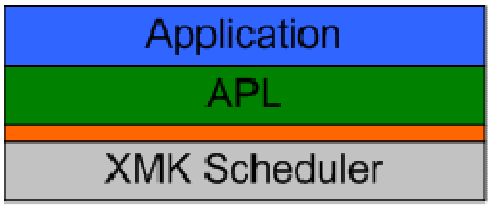


Abbildung 3: Schichten von XMK

4.1.4 PicOS

PicoOS ist ein Betriebssystem für Mikrocontroller und eingebettete Systeme. Erdacht wurde es am Departement of Computing Science an der University of Alberta2, mittlerweile wird es durch die Olsonet Communications Corporation 3unter GPL angeboten. Es wurde zuerst auf einer Platine von Cyan Technologie 4eCOG1 mit 4Kbyte RAM implementiert. Auch weitere Mikrocontroller dieser Firma werden mit dem PicOS Betriebssystem betrieben.

[2] Pawel Gburzynski, http://www.cs.ualberta.ca/~pawel/
[3] http://www.olsonet.com/
[4] http://www.cyantechnology.com/solutions/PicOS.php

Der Kernel von PicOS ist für die Speichervergabe zuständig. Er verwaltet die Prozess-Kontroll-Blöcke (engl. PCB) und bietet Schnittstellen für den Zugriff auf die Gerätetreiber (siehe Abbildung 4). Hier sind unter anderem LCD-Anzeige, LEDs, Netzwerkkarte, Funknetzwerk, serielle Schnittstellen und AD Wandler zu nennen.

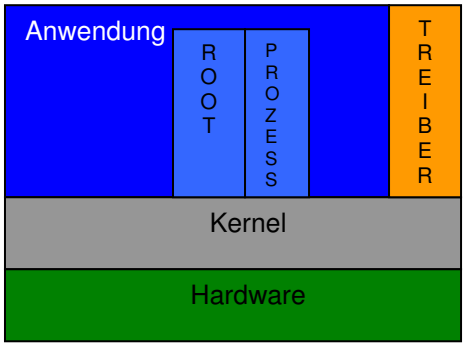


Abbildung 4: Aufbau PicOS

Der Speicher (Heap) wird in getrennte Pools aufgeteilt. Diese können von der Anwengung mit Daten belegt werden, falls ein malloc-Aufrauf erfolgreich war. Anderfalls besteht die Möglichkeit, einen Prozess solange schlafen zu legen, bis wieder Speicher im Pool zur Verfügung steht.

Tasks werden in PicOS Prozesse genannt. Sie wurden als endliche Automaten designt. Jeder Prozess hat Zustände, in denen das jeweilige Teilprogramm abläuft, bis es fertig ist. Danach kann der Scheduler einen anderen Prozess aufrufen und dort in einen Zustand einspringen. Dieser Zustand ist anfangs der Zustand Null, falls schon einmal der Prozessor abgegeben wurde ist es grundsätzlich der letzte Zustand, in dem man sich befand. Es können aber auch Ereignisse, an denen ein Zustand eines Prozesses aufgerufen werden soll, festgelegt werden, indem ein sogenannter „Wait Request“ festgelegt wird. Der Prozess wird dann blockiert bis das Ereignis eintritt und andere Prozesse werden ausgeführt. Ein Prozess kann auch auf mehr als ein Ereigniss warten. Tritt eines dieser Ereignisse ein, wird der Prozess fortgeführt, in dem Zustand, welcher mit dem Ereignis verknüpft war. Nun werden alle anderen Ereignisse, auf die gewartet wurde, gelöscht. Falls sich der Prozess dennoch für die Ereignisse interessiert, muss er wieder Wait Requests anlegen.

Prozesse werden durch den Scheduler nicht verdrängt, ein Prozess kann nur an definierten Stellen, nämlich an der Grenzen der Zustände, verlassen werden. Eine Ausnahme hierbei bilden die Interrupts im Kernel, welche den Prozess auch während der Abarbeitung eines Zusandes verdrängen können.

Eine Anwendung in PicOS hat immer einen Root Prozess, welcher automatisch nach einem Reset geladen wird. Davor können nur Gerätetreiber geladen werde. Der Root Prozess ist verantwortlich für das Erzeugen der weiteren Prozesse, in dem er über fork-Aufrufe neue Prozesse explizit erzeugt. Diese können sich dann selbst beenden, nachdem sie fertig sind, oder durch andere Prozesse zerstört werden (kill). Alle Prozesse und der Kernel teilen sich einen globalen Stack Speicher.

Für die Zusammenarbeit der einzelnen Prozesse steht eine Sammlung von einfachen Werkzeugen zur Verfügung. Wird ein neuer Prozess über eine fork-Anweisung erzeugt, wird im als Parameter auch eine Datenstruktur übergeben, auf der er arbeiten kann. Wird diese Datenstruktur von mehreren Prozessen benutzt, ist hierdurch die Interprozesskommunikation möglich. Synchronisationsmechanismen wie Semaphoren oder kritische Breiche sind nicht vorhanden, können aber durch wait/signal Mechanismen nachgebildet werden. Da aber Prozesse in PicOS nur an eindeutigen Stellen verdrängt werden können, sind alle problematischen Operationen atomar, solange keine Zustandsübergänge dazwischen stattfinden.

4.2 Betriebssysteme für Sensorknoten

4.2.1 SOS

Der Hauptpunkt in SOS ist die Wiederkonfigurierbarkeit der individuellen Sensorknoten, nachdem sie entwickelt und gestartet wurden. Module können während der Laufzeit eingefügt oder entfernt werden, somit können Updates auf Knoten zur Laufzeit ausgeführt werden.

Module ins SOS sind unabhängige Binärdateien welche eine spezielle Funktion oder einen Task implementieren. Die Module sind über dem Kernel angesiedelt (siehe Abbildung 5), in der sogenannten Modulschicht. Hierin werden die Anwendungen, Treiber und Protokolle entwickelt. Eine Veränderung des SOS Kernels ist nicht notwendig, nur wenn sich etwas an der darunterliegenden Hardware ändert. Eine Anwendung besteht aus einem oder mehreren Modulen, welche miteinander interagieren. Um Modularität zu gewährleisten, können Module untereinander Nachrichten austauschen und benutzen Funktionsschnittstellen.

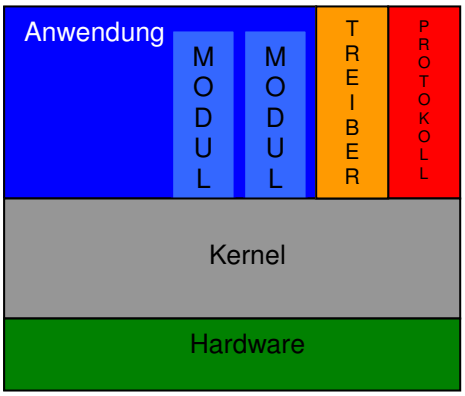


Abbildung 5: Aufbau SOS

Module besitzten genau definierte Einsprung- und Aussprungspunkte. Es gibt zwei Arten, wie der Programmablauf ein Modul aufrufen kann: erstens durch Nachrichten, welche durch den Scheduler verteilt werden oder zweitens durch Aufrufe der Funktionen eines Moduls, welche zum externen Gebrauch registriert werden müssen.

Der Nachrichtenverkehr wird durch ein Nachrichten Steuerungsprogramm[5] gehandhabt. Diese Funktion hat als Parameter die Nachricht ansich und den Status des Moduls. Jedes Steuerprogramm eines Moduls muss mindestens zwei Funktionen bereitstellen, um init und final Nachrichten verarbeiten zu können. Diese werden durch den Kernel erzeugt, init beim Einfügen eines neuen Moduls und final beim Entfernen. Bei der Behandlung der init-Nachricht werden in einem Modul die grundlegenden Dinge eingerichtet, wie periodische Timer, das Registrieren der Funktionen oder das Einschreiben für Funktionen. Eine ähnliche Aufgabe hat Behandlungsroutine der final-Nachricht, hierbei werden alle reservierten Ressourcen wieder gelöst, dazu gehören Timer, Speicher und registrierte Funktionen. Weitere Aufgaben des Steuerprogramms ist die Auswertung von Nachrichten für Auslöser von Timern, Signale von Sensoren und Nachrichten von anderen Modulen. Der Nachrichten-Scheduler im Kernel nimmt Nachrichten aus einer Prioritätswarteschlange und verteilt sie an die einzelnen Module. Hierdurch findet eine asynchrone Kommunikation statt. Für synchrone Operationen werden die Funktionsaufrufe genutzt. Hierfür stehen Verfahren zum eintragen und abonnieren von Funktionen bereit. Registriert sich ein Modul beim Kernel, so werden diesem die genauen Positionen in der Binärdatei der eigenen Funktionen mitgeteilt. Hierfür führt der Kernel eine Liste, den Funktonskotrollblock[6]. Hier werden die Tupel {module ID, function ID} eingtragen.
Hat ein Modul andererseits vor, eine Funktion zu benutzen, so muss er diese abonnieren. Hierfür macht es einen Kernelaufruf. Ist dieser erfolgreich, so übergibt der Kernel einen Zeiger auf einen Funktionszeiger auf die gewünschte Funktion.

[5] engl.: handler
[6] Function Control Block, FCB

Der Zugriff auf die Funktion sollte nur über den dereferenzierten Zeiger erfolgen, da hierdurch fehlgeleitete Aufrufe nicht ins Nirvana laufen. Das gewährleistet einen reibungslosen Austausch von Modulen, da bei einer Aktualisierung von Funktionen kein erneutes abonnieren notwendig ist.
Funktionsaufrufe direkt in den Kernel werden über sogenannte Einsprungtabellen gemacht, da diese gleich bleiben, auch wenn verschiedene Versionen des Kernels vorliegen sollten. Hierdurch können auch unterschiedlichen Sensorknoten verschiedene Versionen des Kernels vorhanden sein, aber die gleichen Module laufen. Hierdurch wird ein Aufbau von Sensornetzen mit heterogenen Kernel möglich.

Die Verteilung von neuen Modulen über das Netz geschieht durch spezielle Protokolle. Diese lauschen am Netzwerk, ob Ankündigungen für neue Module vorhanden sind. Falls dies der Fall ist, wird untersucht, ob es sich um aktuellere Versionen als die Vorhandenen handelt. Falls genügend Platz auf dem Sensorknoten vorhanden ist wird dann das Update heruntergeladen. Ist das Modul dann ordnungsgemäß installiert schickt der Kernel die init-Nachricht an das Modul.

Um ein Modul zu enternen schickt der Kernel eine final-Nachricht an ein Modul. Dieses hat dann die Möglichkeit, alle benutzten Ressourcen freizugeben und alle abhängigien Module zu informieren. Danach führt der Kernel eine Reinigung durch (Müll einsammeln7) indem er den dynamisch zugeordeten Speicher freigibt, ebenso Timer, Sensortreiber und andere Ressourcen.

Das Scheduling in SOS ist kooperativ, Nachrichten werden in Warteschlangen eingetragen und sind mit Prioritäten versehen. Es gibt Warteschlangen für Nachrichten mit hoher Priorität, z.B. eingehende oder ausgehende Funkdaten.

Die Speicherverwaltung ist dynamisch. Es wird ein best-fit Algorithmus mit drei verschiedenen Blockgrößen ausgeführt, wenn Speicher benötigt wird. Um die Effizienz zu steigern wird über die Einträge im Speicher Buch geführt. Die Größe der Blöcke ist entsprechend den typischen Anwendungsfällen angepasst. Die Kleinsten für Nachrichtenköpfe, die anderen Größen sind z.B für das kopieren von Modulen gedacht. Für jede der Blockgrößen gibt es Listen über den vorhandenen freien Speicher. So kann immer in der gleichen Zeit Speicher angefordert oder freigegeben werden, eine Beeinträchtigung des Scheduling findet hierdurch nicht statt.

Anwendungen haben als grundlegende Struktur einen großen Switch Block, welcher je nach eingehender Nachricht den entsprechenden Zustand aufruft, beispielsweise init oder final. Anwendungen sind in der Programmiersprache C geschrieben, da hier der Aufwand des Einlernens niedriger ist und es genügend Compiler und Debugger gibt.

[7] engl.: garbage collection

4.2.2 Contiki

Das Hauptziel von Contikit ist Portierbarkeit. Contiki wurde speziell für eingeschränkte Geräte wie Sensorknoten entwickelt, es existieren aber auch Portierungen auf viele andere Geräte. Es unterstützt dynamisches laden und entfernen von Programmen. Der Kernel ist Ereignisbasiert und unterstützt zugleich verdrängendes Multithreading, welches als Bibliothek bei Bedarf eingebunden werden kann. Contiki wurde in C geschrieben und auf mehrere Mikrocontroller portiert.

Da Sensorknoten in hoher Anzahl benutzt werden und teilweise weit verstreut positioniert werden, sollte die Möglichkeite bestehen, Programme dynamisch über das Netzwerk einspielen zu können, ohne jeden Knoten einzeln einsammeln zu müssen. Aber die hierfür benötigte Energie muss so gering wie möglich gehalten werden. Andere Betriebssysteme laden immer das komplette Abbild, welches den Kernel, Bibliotheken und die Anwendungen enthält, auf den Knoten. Contiki kann auch nur individuelle Anwendungen laden und verringert hierdurch das Übertragungsvolumen und somit die Übertragungsdauer und den Energieverbrauch.

Um möglichst viele Plattformen unterstützen zu können, muss ein Betriebssystem portabel sein. Das Problem bei Sensorknoten besteht in der großen Anzahl an verfügbaren und eingesetzten Sensoren und Kommunikationsgeräten, welche sich je nach Hersteller oder Mikrocontrollerfamilie stark unterscheiden. Die einzige Gemeinsamkeit ist die Prozessorarchitektur, welche ein Speichermodell ohne Segmentierung oder Speicherschutzmechanismen besitzt. Programmcode wird für gewöhnlich in wiederbeschreibbaren ROM Modulen gespeichert und Daten in RAM. Aus diesem Grund wurde Contiki so konstruiert, dass die Abstraktion des grundlegenden Systems nur auf dem multiplexen der CPU und dem nachladen von Programmen und Diensten besteht. Andere Abstraktionen wurden in die Bibliotheken verlagert.

Contiki vereinigt einen Ereignisbasierten Kernel und verdrängbare Threads, welche durch Bibliotheken zu den Programmen gebunden werden, welche Threads benötigen (siehe Abbildung 6). Hierbei werden die Vorteile von ereignisbasierten Systemen, welche als endliche Automaten dargestellt werden können, voll ausgenutzt. Aber gleichzeitig die gravierenden Nachteile, wie z.B. die vollständige Abarbeitung eines Ereignisses durch einen nichtverdrängbaren Eventhandler, vermieden. Würde beispielsweise eine Verschlüsselung zur Datenübertragung eingesetzt, so würde die Schlüsselberechnung mehrere Sekunden die CPU komplett für sich beanspruchen, ohne die Möglichkeit, verdrängt werden zu können. Wird die Verschlüsselung jedoch als Thread programmiert, so kann dieser vom Scheduler in der Bibliothek verdrängt werden, andere Aufgaben können bearbeitet werden. Es kann somit auch auf externe Ereignisse reagiert werden, ohne dass diese verloren gehen würden.

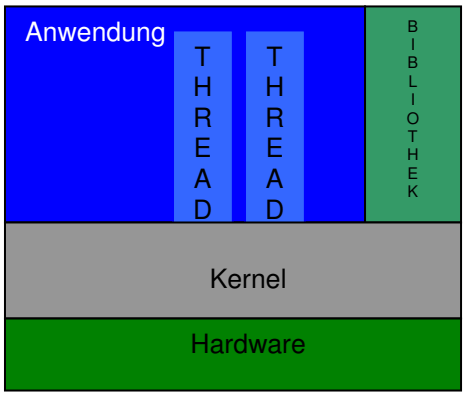


Abbildung 6: Aufbau Contiki

4.2.3 Ambient RT

Ambient RT ist ein Echtzeitbetriebssystem für eingebettete Geräte. Es wurde mit Blick auf die geringe Rechenleistung, wenig Speicher und begrenzten Energievorrat entwickelt. Der Echtzeitanteil an Ambient RT steht unter der GPL Lizenz, darf also frei benutzt und angepasst werden.

Obwohl AmbientRT auf Hardware mit eingeschränkten Ressourcen läuft, bietet es einen Echtzeit-Scheduler mit Earliest-Deadline-First (EDF) Scheduling. Somit ist der Prozessor besser ausgelastet als in Systemen mit kooperativem oder keinem Scheduling. Diese Art von Scheduling wurde theoretisch als verklemmungsfrei bewiesen. Eine Task (Prozess) ist auch hier Teil eines Programms. In AmbientRT gibt es nur Echtzeit-Tasks, diese sind an Zeitbeschränkungen gebunden, d.h. sie müssen zu einer bestimmten oder nach einer bestimmten Zeit beendet sein. Hierbei hat der Anwendungsdesigner die Aufgabe, die benötigten Eigenschaften festzulegen. Dazu gehören:

- Deadline
- Periode
- CPU Kosten
- Ressourcenverbrauch

Während die Eigenschaften ReleaseTime und Absolute Deadline der Task beim ersten aktiv-werden vom System dynamisch zugeordnet werden.
Der Kernel von AmbientRT führt Tasks aus, als liefen sie parallel. Dieses Multitasking wird durch den Scheduler erledigt, welcher zu jedem Zeitpunkt festlegt, welche Task als nächstes laufen darf. Diese Entscheidung wird anhand der Zeitkriterien der einzelnen Tasks entschieden, um eine optimale Zeiteinteilung zu erhalten. Der Scheduler vergibt dynamische Prioritäten, welche den Tasks

zugeordnet werden und sich im Laufe der Zeit ändern. Die Prioritätenvergabe erfolgt anhand der Deadlines, je näher ein Task seiner Deadline kommt, desto höher wird er eingestuft. Allgemein führt der Gebrauch der absoluten Deadline als Priorität zu einer besseren Auslastung des Prozessors als es feste Prioritäten tun würden.

Jede Task besitzt einen Kontext, welcher aus dem aktuellen Inhalt seiner Register und des Stacks besteht. Beim Kontextwechsel, also wenn ein anderer Task durch den Scheduler ausgewählt wurde, muss normalerweise der Kontext gesichert werden. Da sich bei AmbientRT alle Task einen einzigen Stack teilen, muss der Kontext nicht gesichert werden, sonder kann auf dem Stack bleiben. Wenn eine Task verdrängt und ein neuer Kontext erstellt wird, dann geschieht dies einfach auf dem alten Task darüber. Die Wiederherstellung eines Kontexts geschieht nur, falls der laufende Task beendet und der verdrängte Task fortgeführt wird. Und da unter der beendeten Task der Kontext der verdrängten Task liegt, ist eine Wiederherstellung der Kontextinformationen nichts anderes, als das Entfernen des Kontext der laufenden Task.
Bei der gemeinsamen Verwendung einer Ressource kann es in einem Multitaskingbetriebssytem zu Konflikten kommen. Um diese zu vermeiden gibt es die Möglichkeit sich eine Ressource durch Mutex zu schützen. Mit AmbientRT werden Mutexe durch den Scheduler durchgeführt. Dieser stellt die automatische Synchronisation von geteilten Ressourcen zur Verfügung. Jeder Task, welcher eine Ressource verwenden will, wird in eine Liste eingetragen. Aus dieser werden die Informationen für den Vergleich durch den Scheduler bezogen. Wollen zwei Task auf die selbe Ressource zugreifen, wird einer blockiert. Hierbei wird der schon laufende Task weiterlaufen gelassen bis dieser fertig ist. Dies gilt auch, falls der wartende Task eine kürzere Deadline besitzt. Diese Verzögerung wird beim blockierten Task dazu addiert und ergibt somit eine sogenannte Blockierzeit, welche vom Scheduler mit beachtet werden muss.

Die Echtzeitfähigkeiten von AmbientRT ist hartes Echtzeitscheduling. Das bedeutet, dass alle Zeitschranken eingehalten werden müssen. Durch eine mathematische Analyse wird ein Set von Tasks berechnet, welche diese Schranken einhalten. Wenn ein solches Set durchführbar ist, dann wird garantiert, dass zu jedem Zeitpunkt jeder Task vor seiner Deadline beendet ist.

4.2.3.1 Data Centric Architecture

Eine weitere Besonderheit bei AmbientRT ist der Aufbau bzw das Konzept des Nachrichtenaustauschs und das Anzeigen von Unterbrechungen bzw Ereignissen. Hier wird eine andere Form der Interprozesskommunikation angewandt, welche sich Data-Centric Architecture nennt. Im Mittelpunkt steht hierbei der Datenstrom, welcher entweder von Daten-produzierenden oder -konsumierenden Komponenten erzeugt wird. Diese Komponenten werden Data Centric Entities (DCE) genannt. Datenströme sind entweder Speicherobjekte oder Ereignisse, wobei Ereignisse z.B. das Auftreten von Hardwareunterbrechungen bedeutet. Jedes dieser Objekte oder

Ereignisse wird Data Type (DT) genannt. Die DCEs werden in einem System mit einschreiben / veröffentlichen benutzt, welche ihnen erlauben, auf DTs von anderen DCEs zu reagieren.

Der Kernel von AmbientRT unterstützt die Data-Centric Architektur, welche es den Anwendungen ermöglicht, sich dynamisch neu zu konfigurieren. Anwendungen bestehen aus funktionalen Blöcken, welche, im Gegensatz zu statischen Anwendungen, nur lose Verbindungen untereinander eingehen. Diese werden zentral von einem Manager (siehe Abbildung 7) koordiniert. Dieser kann virtuelle Verbindungen zwischen den einzelnen Blöcken herstellen oder lösen. Wenn diese Verbindungen neu arrangiert werden entstehen neue Konfigurationen, somit wird das System funktional anpassbar.

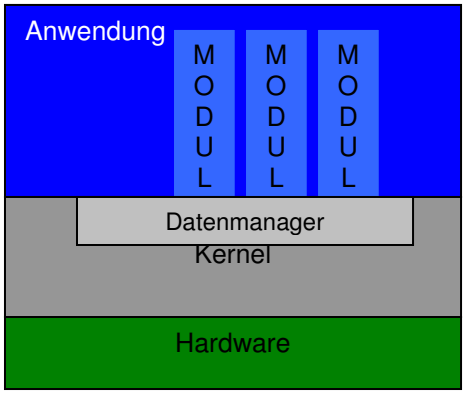


Abbildung 7: Aufbau AmbientRT

Der Kernel enthält den Datenmanager, welcher eine Liste unterhält, in der die DTs eingetragen sind. Somit können die Datenströme nach verfolgt werden. Hierbei enthält jeder Eintrag in der Liste eines DTs seine Daten und welche DCEs bei dieser eingeschrieben sind. Der Manager kümmert sich auch um die Aktivierung und Deaktivierung von DCEs. Wenn ein DCE seine Daten über ein DT verteilt werden alle DCEs, welche sich für diese DT eingeschrieben haben, aktiviert. Zwischen dem Data Manager und dem Scheduler herrscht eine enge Verzahnung, so dass DCEs auch durch den Scheduler gesteuert werden.

4.2.3.2 Module

Anwendungen in AmbientRT können aus Modulen bestehen. Diese sind separat kompilierbare Tasks, welche unabhängig vom Kernel übersetzt werden und nachträglich in das System eingefügt werden können. Das Einfügen während der Laufzeit kann wegen der Data-Centric Architektur durchgeführt werden. Der Kernel kann einzelne Module dynamisch laden und ausführen. Auf diese Art und

Weise kann AmbientRT auch nachträglich mit neuer Funktionalität ausgestattet werden. Module können in einer Anwendung getauscht, eingefügt oder entfernt werden, somit muss nicht die komplette Anwendung neu auf den Sensorknoten oder das eingebettete System übertragen werden. Dies spart Energie, indem weniger Daten über das Netz transportiert werden müssen. AmbientRT enthält ein Protokoll zur Übertragung von Binärdaten, so können neue Module über Funk oder einen seriellen Port geladen werden. Diese werden Stückchenweise in den RAM Speicher geladen, bis ein koplettes Paket übertragen worden ist. Danach wird das gesamte Paket in den sekundären Speicher geschrieben, wo es von einem einfachen Dateisystem verwaltet wird.

5 Fazit

Die vorgestellten Betriebssysteme geben einen guten Überblick, welche Eigenschaften Betriebssysteme auf Mikrocontrollern und Sensorknoten haben können. Auch in Zukunft werden Sensorknoten von der knappen Energie abhängig sein. Nach Moore's Gesetz nimmt die Anzahl der Transistoren stetig zu, oder andersherum: Prozessoren mit gleichbleibender Anzahl an Transistoren werden immer kleiner. Somit werden auch zukünftig die hier aufgezeigten Einschränkungen gelten, auch wenn die Geräte immer kleiner werden.

Die verschiedenen Gruppen, welche für die Entwicklung der einzelnen Systeme verantwortlich sind, planen weitere Verbesserungen. Contiki soll eine Unterstützung durch das Betriebssystem für sichere Code Updates erhalten. Bei PicOS wird an einem Plugin für erweiterbare Netzwerkmodule gearbeitet. Dieses soll anpassbare drahtlose Verbindungen herstellen, beispielsweise in Verbindung mit TCP/IP. Des weiteren soll eine Entwicklungsumgebung geschaffen werden, welche Zustandsautomaten in die Programmiersprache integriert, anstelle der Emulation durch *switch* und *case*. In SOS sollen Mechanismen zur Fehlerbehandlung und Vermeidung verstärkt Einzug halten. Hierbei soll das System gegen falsche Operationen der Module geschützt werden. Dies soll dazu beitragen, die Konsistenz des Systems zu vereinfachen.

6 Literaturverzeichnis

[1] Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren, Springer-Verlag 2002

[2] Tanenbaum, Moderne Betriebssysteme, Prentice Hall, Pearson Studium 2003

[3] FreeRTOS, http://www.freertos.org/ (Januar 2006)

[4] Pico]OS, http://picoos.sourceforge.net (Januar 2006)

[5] XMK, http://www.shift-right.com/xmk/ (Januar 2006)

[6] Akhmetshina, Gburzynski, Vizeacoumar. "PicOS: A Tiny Operating System for Extremly Small Embedded Platforms", *Proceedings of ESA'03*, Las Vegas, June 23-26, 2003, pp. 116-122.

[7] Chih-Chieh Han, Ram Kumar Rengaswamy, Roy Shea, Eddie Kohler and Mani Srivastava. **"**SOS: A dynamic operating system for sensor networks", *Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, 2005

[8] Roy Shea, Chih-Chieh Han, and Ram Kumar Rengaswamy. **"**Motivations Behind SOS**",** *NESL Technical Report.*, 2004.

[9] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors". In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors 2004* (IEEE EmNetS-I), Tampa, Florida, USA, November 2004.

[10] "AmbientRT, real-time OS for embedded systems", Ambient Systems BV, Enschede, the Netherlands, 19th January 2005

[11] T.J. Hofmeijer, S.O. Dulman, P.G. Jansen, P.J.M. Havinga. "AmbientRT - Real Time System Software Support for Data Centric Sensor Networks", ISSNIP 2004, Australia, December 2004

7 Anhang : FactSheets

7.1 FactSheet FreeRTOS

Kernel	Realtime Betriebssystem
Scheduling	Preemptives Scheduling, kooperatives Scheduling
Speicherverwaltung	Statisch bei Kompilierung, dynamisch mit festen Größen, dynamisch mit beliebigen Größen
Portierungen	– ST Microelectronics STR71x (ARM7) (STR711F, STR712F, etc.). – LPC2106, LPC2124 and LPC2129 (ARM7). Includes I2C driver source code. Demo's for Olimex and Keil hardware. – Renesas H8S2329 (Hitachi H8/S) with an EDK2329 demo. – Atmel AT91SAM7 family (AT91SAM7X256, AT91SAM7X128, AT91SAM7S32, AT91SAM7S64, AT91SAM7S128, AT91SAM7S256). Includes USB driver source code for the IAR Kickstart, plus uIP and lwIP embedded TCP/IP stack demos. – AT91FR40008 with an Embest ATEB40X demo. – MSP430 with a Softbaugh demo including LCD driver. MSPGCC and Rowley CrossWorks tools are supported. – HCS12 (MC9S12C32 small memory model and MC9S12DP256B banked memory model) – Cygnal 8051 / 8052 – Microchip PICMicro (PIC18) – Atmel AVR (MegaAVR) with an STK500 demo. – RDC8822 Microcontroller (AMD embedded 186 clone) with demo for the Flashlite 186 SBC.

	– PC [running on top of FreeDOS or other DOS] – ColdFire - note this port is unsupported – Zilog Z80 (eZ80 Acclaim!) - note this port is unsupported – Xilinx Microblaze soft processor core running on a Virtex4 FPGA.

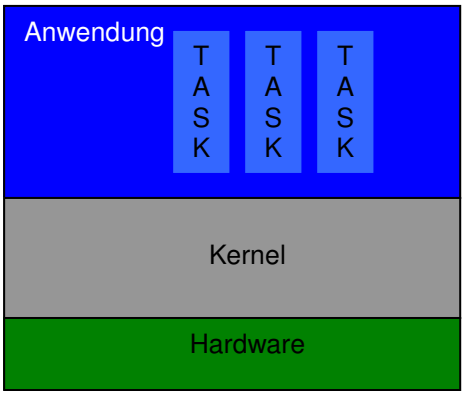


Abbildung: Aufbau FreeRTOS

7.2 FactSheet Pic]OS

Kernel	Echtzeitbetriebssystem
Scheduling	Prioritätsbasierend oder RoundRobin
Speicherverwaltung	Dynamisch
Interprozesskommunikation	Über Message Boxes
Ressourcenverwaltung	Semaphoren, Mutexe
Portierungen	– 6502 CPU and compatible series, especially Commodore 64. – 80x86 in real mode, the RTOS is loadable from DOS. – 80x86 WIN32 enables pico]OS to run on top of MS Windows. – ARM CPU: SAMSUNG S3C2510A (ARM940T core) and compatible series. – MegaAVR: Atmel(tm) RISC core processor. – PowerPC: IBM PPC440 and compatible w/o floating point.

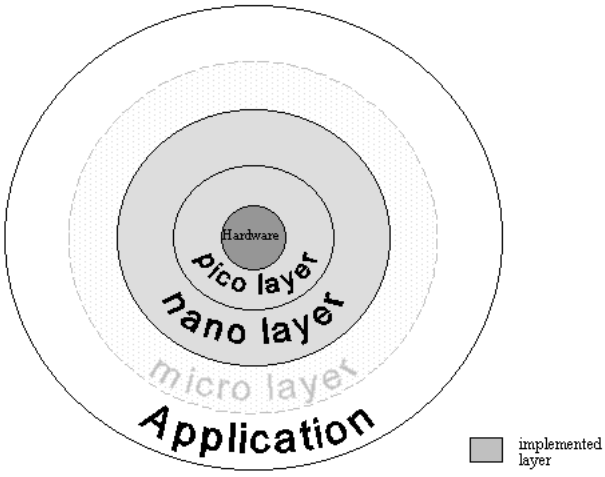


Fig.: pico]OS Layer Scheme

Abbildung: Aufbau Pic]OS

7.3 FactSheet XMK

Kernel	Echtzeitbetriebssystem
Scheduling	Preemptives Scheduling mit Prioritäten
Speicherverwaltung	Durch APL verwaltet
Interprozesskommunikation	Über APL
Ressourcenverwaltung	Semaphoren, Mutexe
Portierungen	– Atmel AVR – Hitachi / Renesas : H8/300, H8/300L,H8/300SLP – H8/300HN, H8/300 Tiny – H8s/Advance Mode – M16C and R8C/Tiny

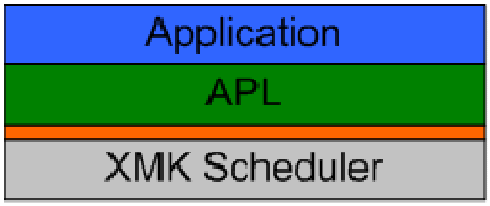


Abbildung: Aufbau XMK

7.4 FactSheet PicOS

Kernel	Ereignisbasiert
Scheduling	Nur an Zustandsgrenzen ist Prozesswechsel möglich. Ausnahme : Interrupts
Speicherverwaltung	Speicher in getrennte Pools aufgeteilt
Interprozesskommunikation	Über gemeinsame Datenstrukturen
Ressourcenverwaltung	Synchronisationsmechanismen nicht vorhanden, können nachgebildet werden
Portierungen	Cyan Technology eCOG1

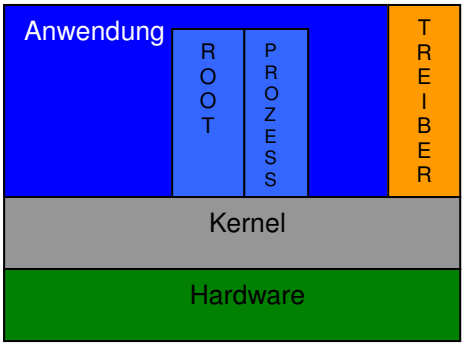


Abbildung: Aufbau PicOS

7.5 FactSheet SOS

Kernel	Ereignisbasiert
Scheduling	Kooperativ Nachrichten in Warteschlangen mit Prioritäten
Speicherverwaltung	Dynamisch, Best-Fit Algorithmus
Interprozesskommunikation	Über Nachrichten
Ressourcenverwaltung	-
Portierungen	– AVR – Crossbow Mica Mote – Yale's XYZ Node

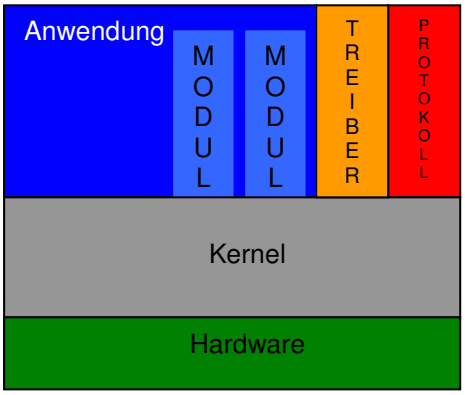


Abbildung: Aufbau SOS

7.6 FactSheet Contiki

Kernel	Ereignisbasiert
Scheduling	Verdrängendes Multitasking über Bibliothek
Speicherverwaltung	Blockweise
Interprozesskommunikation	Über Ereignisse
Ressourcenverwaltung	-
Portierungen	– Raw X11 – TI MSP430 / FU Berlin sensor board – x86, Atari Portfolio, Atari ST – Atmel AVR 8-bit microcontoller – Commodore 64 – Atari 8-bit – Casio PocketViewer – Game Boy – GP32 – Nintendo Entertainment System – Atari Jaguar – Z80 based PC-6001 – VIC 20 – Apple][– Tandy CoCo – PCEngine – Plus/4 – CBM PET – Commodore 128 – Geplant : Sharp Wizard PDA, PlayStation, Sega DreamCast

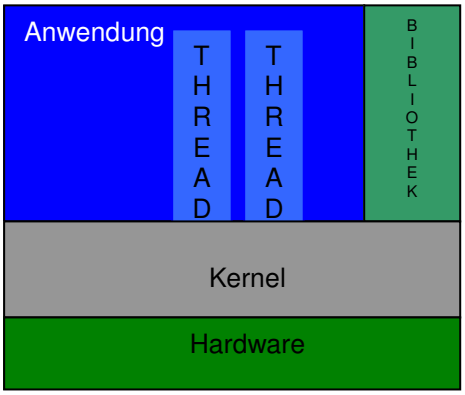


Abbildung: Aufbau Contiki

7.7 FactSheet AmbientRT

Kernel	Echtzeitbetriebssystem
Scheduling	Verdrängendes Echtzeitscheduling mit EDF
Speicherverwaltung	dynamisch
Interprozesskommunikation	Nicht benötigt, da Data Centric Architecture
Ressourcenverwaltung	Automatische Mutexe
Portierungen	MSP430 processor

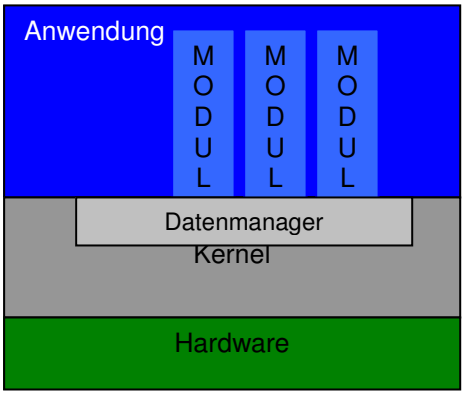


Abbildung: Aufbau Contiki

Makroprogrammierung für Sensornetze

Seminararbeit von **Stephan Oehlert**

Betreuer Till Riedel

2. Februar 2006

1 Einleitung

Sensornetze sind komplexe Gebilde, die vielfältige Aufgaben übernehmen können. Sie bestehen aus einer beliebig großen Anzahl Knoten, die, je nach Bedarf, Sensoren auslesen, mit ihrer Umgebung interagieren, untereinander Daten austauschen und Berechnungen ausführen können. Die steigende Komplexität der Aufgaben, die Sensornetze übernehmen sollen, macht die Programmierung der Sensorknoten aufwendiger und erfordert neue Programmieransätze; der Energieverbrauch der Knoten muss auf ein Minimum reduziert werden, damit lange Laufzeiten ohne manuelle Eingriffe realisiert werden können. Die Bandbreite, die zwischen den Knoten für die Datenübertragung zur Verfügung steht, ist durch den hohen Energieverbrauch der Kommunikation stark eingeschränkt, und Übertragungsfehler und der Verlust von Nachrichten oder ganzer Knoten kann zu jeder Zeit eintreten. Es muss deshalb mit den zur Verfügung stehenden Ressourcen sparsam umgegangen werden und die von Programmen genutzten Schnittstellen müssen robust gegenüber möglichen Fehlern sein, damit sie zuverlässig arbeiten können.
Die große Mehrheit der Makroprogrammiersprachen nutzt als Plattform für Hardwarezugriffe TinyOS. Es besteht aus Komponenten, mit denen die Hardware angesprochen werden kann, und einer C-ähnlichen Programmiersprache namens nesC. TinyOS ist kein Betriebssystem im Sinne von Windows oder Linux, da es keinen Kern mitbringt; voneinander isolierte Prozesse und Multithreading sind nicht möglich, Speicherverwaltung gibt es nicht, und es kann nur eine Anwendung zur Zeit ablaufen. Dafür haben Programme volle Kontrolle über die Hardware und können ressourcenschonend arbeiten.
Es kommen oft 8Bit Mica-, Mica2- oder Mica2Dot-Plattformen zum Einsatz. Einen Beispielaufbau eines Sensorknoten mit Mica2Dot-Chip zeigt Abbildung 1. Stromversorgung, Controller und Sensorik bilden einen Computer mit geringem Platzbedarf.
Makroprogrammierung soll das Programmieren von komplexen Anwendungen erleichtern. Das Programm soll *im großen* geschrieben werden können und damit die Komplexität der dafür auf den einzelnen Knoten nötigen Rechenschritte verborgen werden. Es soll dem Programmierer eine andere Sichtweise auf das System als ganzes ermöglichen. Ziel ist es, damit eine deutliche Vereinfachung der Programmierung zu bewirken; letztlich sollen gegebenenfalls auch Nicht-Programmierer, also Nutzer aus anderen Wissenschaftsbereichen oder Endanwender, in der Lage sein, Anwendungen zu entwickeln oder bestehende Anwendungen ihren Bedürfnissen anzupassen, gegebenenfalls auch zur Laufzeit. So existieren viele Algorithmen für komplexe Graphenstrukturen, die theoretisch auch auf Sensornetztopologien anwendbar sind, dafür aber eine Sicht auf das Netz als ganzes erfordern.
Diese Anwendungen können durch Makroprogrammiersprachen verwirklicht werden. Sie bilden Plattformen, auf denen Programme aufgesetzt werden, die bestimmte Aspekte der Technik der Sensornetze übernehmen und vor dem Programmierer verbergen. Bei niedriger Abstraktionsebene kann sich das auf die Wiederverwendbarkeit Hardwarespezifischen Codes beziehen, bestimmte Funktionalität zusammenfassende Komponenten einschließen, oder, bei hoher Abstraktionsebene, die vollständige Verdeckung von Netzeigenschaften wie Topologie und Größe bedeuten; dann kann das Sensornetz als reine Datenbank oder frei programmierbare Mehrprozessormaschine betrachtet werden.
Zentrales Problem der Makroprogrammierung von potenziell sehr großen Sensornetzen ist die Umsetzung des Programms, welches das globale Verhalten des Netzes beschreibt, in komplexe Befehle, die dann lokal auf den Knoten ausgeführt werden [9].

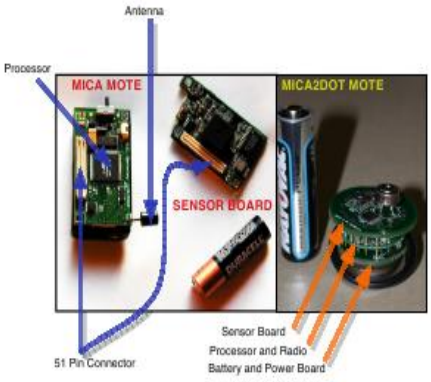


Abbildung 1: Ein Sensorknoten mit Mica2Dot-Plattform [1]

Während das Programm ein verhältnismäßig einfach verständliches Verhalten des Netzes beschreiben kann, ist die Ausführung auf Knotenebene möglicherweise aufwendig und von den aktuellen Netzeigenschaften wie den Abständen der Knoten voneinander abhängig. Die auf den Sensorknoten ablaufenden Programme können diesem Fall in ihrem Umfang anwachsen und hohe Anforderungen an das System auf den Knoten stellen. Beim Einsatz der Makroprogrammiersprache muss dies bedacht werden und gegebenenfalls ein Kompromiss bei der Funktionalität zugunsten der Effizienz eingegangen werden.
Da oft eine lange Laufzeit der Programme erwünscht ist, müssen die Berechnungen auf den Knoten und die Kommunikation zwischen ihnen ressourcenschonend ablaufen. Datenaustausch muss so sparsam wie möglich eingesetzt werden, da der Energieverbrauch hier besonders hoch ist. Dies ist in der Regel Aufgabe der Makroprogrammiersprache, die die direkte Kommunikation zwischen Sensorknoten vor dem Programmierer verdeckt.
Anwendungen, die auf Sensornetzen ausgeführt werden, nutzen die Eigenschaft, dass die Prozessoren auf den Sensorknoten parallel arbeiten können und nicht auf gemeinsamen Speicher angewiesen sind. Damit gibt es Gemeinsamkeiten in der Programmierung von Sensornetzen und klassichen Multiprozessorsystemen; Code, der parallel auf mehreren Prozessoren laufen soll, aber auf Synchronisation der Prozessoren untereinander achten muss, kann leicht Verklemmungen (deadlocks) oder Wettlaufsituationen (race conditions) enthalten; in solchen Fällen wird zeitlich abweichende Ausführung des Programms auf den Prozessoren nicht beachtet, so dass Zugriffe auf Bertiebsmittel nicht in der gewünschten Reihenfolge erfolgen und das Programm unerwartet stehen bleibt oder anderes Fehlverhalten zeigt.
Fehler und Probleme können an vielen Stellen und zu jeder Zeit auftreten können. Sensornetze basieren auf drahtloser Kommunikation zwischen den Knoten; die Funkverbindungen können durch zu große Entfernung der Knoten voneinander, durch Abschirmung voneinander oder Funkstörung von außen unterbrochen werden oder ganze Knoten ausfallen. Bei Makroprogrammiersprachen, die das dynamische Verteilen von Code innerhalb des Sensornetzes unterstützen und so die Reprogrammierung erleichtern, kann ausserdem von außen beabsichtigt oder versehentlich bösartiger oder fehlerhafter Code eingeführt werden, wenn nicht Maßnahmen zur Netzsicherheit das verhindern. Diese Aspekte müssen bei der Entwicklung berücksichtigt werden.

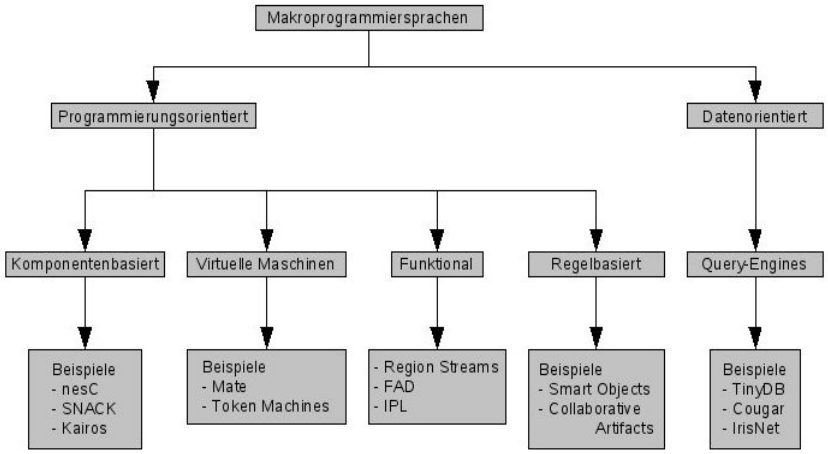


Abbildung 2: Verschiedene Programmiermodelle für Sensornetze

2 Ansätze zur Makroprogrammierung

Herkömmliche Plattformen für die Programmierung von Sensornetzen waren in ihrem Einsatzzweck stark eingeschränkt. Die geringe Rechenleistung und der begrenzte Speicher der Sensorknoten machte eine auf extrem sparsamen Umgang mit Ressourcen ausgelegte Programmierung notwendig. Vor allem in der Funkkommunikation führen verringern den Energieverbrauch, weshalb die Programme besonders darauf ausgelegt waren. Da jede Abstraktion der Hardware Komplexität mit sich bringt und damit nach leistungsfähigerer Hardware bzw. höheren Energieverbrauch erfordert, wurde darauf fast vollständig verzichtet. Mit leistungsfähigeren Sensorknoten können jetzt Makroprogrammiersprachen für die Programmierung der Sensorknoten genutzt werden, ohne lange Laufzeiten der Systeme unmöglich zu machen.
Der Begriff Makroprogrammierung ist nicht fest definiert; die Auffassung, wie umfassend die Anwendungsmöglichkeiten der Programmiersprache sein müssen, ist sehr unteschiedlich. Hier wird die Definition nicht sehr einschränkend genommen, damit auch sehr verschiedene Ansätze beschrieben werden können.
Es gibt viele Ansätze, die Netzstruktur eines Sensornetzes zu abstrahieren, und einige grundlegende sollen hier vorgestellt werden. Die Methoden werden danach unterteilt, auf welche Weise der Anwendungscode in der Makroprogrammiersprache von der Plattform in Maschinencode übersetzt wird, zu welchem Zeitpunkt dies geschieht und damit wie flexibel das System verändert werden kann. Manche Ansätze lassen auch Funktionalität gezielt weg und konzentrieren sich auf Teilaspekte, weil in vielen Sensornetzanwendungen große Flexibilität bei der Implementierung gar nicht gewünscht ist und so Entwicklungszeit gespart werden kann.

2.1 Komponentenbasierte Systeme

Besonders die hardwaresnahe Programmierung macht die Entwicklung schwierig und erhöht die Wahrscheinlichkeit schwer zu lokalisierender Programmfehler. Alle Makroprogrammiersprachen abstrahieren deshalb Systemzugriffe und damit die Hardware. Die Plattform implementiert eine Abstraktionsschicht über dem System und lässt durch

die Makroprogrammiersprache keine direkten Zugriffe mehr zu.
Wie auf gewöhnlichen Computersystemen stellt man bei der Programmierung von Sensornetzen fest, dass sich Teile der benötigten Funktionen überschneiden; lassen sich diese so zusammenfassen, dass sie mehrfach genutzt werden können, ergeben sich erhebliche Einsparungen bei der Entwicklungszeit neuer Anwendungen. Auf allen bekannten Rechnerarchitekturen wird Programmcode in Bibliotheken zusammengefasst, so dass Anwendungen auf diese zugreifen und die benötigten Funktionen nutzen können. Dies wird auf die Programmierung von Sensorknoten übertragen; Komponentenbasierte Ansätze kapseln wiederverwendbare Teile und stellen diese an Schnittstellen bereit. In der Makroprogrammiersprache ist die Hardware abstrahiert und andere Aspekte wie die Zusammenarbeit der Knoten steht im Vordergrund. Das Programm wird durch den Compiler der Programmiersprache in Maschinencode übersetzt und mit den Bibliotheken gebunden. Sind die Funktionen in den bestehenden Bibliotheken nicht hinreichend für den Einsatzzweck, können Komponenten auch erweitert oder neue hinzugefügt werden. Diese sind dann aber im gegensatz zur Makroprogrammiersprache an das System gebunden, für dass sie entwickelt werden.
Die Makroprogramme müssen vor dem Einsatz kompiliert und auf allen Knoten installiert werden. Zur Laufzeit können sie untereinander Daten austauschen. Soll das Programm zur Laufzeit noch veränderbar sein, dann muss es einen Mechanismus geben, mit dem kompilierter Programmcode als Datenpakete von einem zentralen Rechner im Netz verschickt und auf den Knoten installiert werden kann. Der Code muss dann auf den Knoten dynamisch an die Bibliotheken der Komponenten gebunden werden.

2.2 Virtuelle Maschinen

Um die Anwendungen flexibler gestalten zu können, sollen die Sensorknoten reprogrammierbar sein. Es soll auch zur Laufzeit möglich sein, Programme anzupassen, ohne auf jedem Knoten neue Versionen installieren zu müssen. Dafür muss Programmcode der Anwendungen zur Laufzeit drahtlos zu den Knoten übertragen und dort automatisch ausgeführt werden können. Ein Mechnismus dafür sind Virtuelle Maschinen; dies sind Programme, die auf allen Knoten installiert sind und kontinuierlich laufen. Sie enthalten Funktionen, die die Kommunikation mit anderen Knoten bewerkstelligen und darüber Daten und Programmcode übertragen können. Die Anwendung, die das Sensornetz steuern soll, wird von der VM ausgeführt und kann durch sie überwacht werden. Nur die Funktionen, die von der VM an Schnittstellen dem Programm verfügbar gemacht werden, können von diesem auch genutzt werden. Das Programm ist so vollständig von der Komplexität der Hardware getrennt.
Der Programmcode kann für die Architektur der Knoten, auf denen er laufen soll, kompiliert sein; die Virtuelle Maschine kann auch ein eigenes plattformunabhängige Format implementieren, so dass das Anwendungsprogramm in Bytecode kompiliert wird, der von der VM zur Laufzeit zu nativem Maschinencode übersetzt wird. Unterschiede ergeben sich dabei in der größe der kompilierten Programme, da diejenigen in Form von Bytecode gegebenenfalls kleiner sind. Die VM ergänzt dann die nötigen Befehle in Maschinencode.
Solche Virtuellen Maschinen gibt es auch für gewöhnliche Rechnerarchitekturen; Java oder .NET bieten diese Funktionalität. Programme werden in Bytecode übersetzt; jede Installation der VM umfasst festgelegte Schnittstellen, wodurch garantiert werden kann, dass der Bytecode auf jeder der Unterstützten Plattformen ausgeführt werden kann. Sie eignen sich aber in ihrer jetzigen Form nicht für den Einsatz in Sensornetzen, da ihre Laufzeitumgebungen größeren Speicherbedarf haben, als gängige Hardware

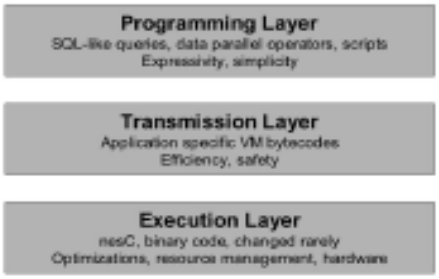


Abbildung 3: Schema VM-basierter Makroprogrammiersprachen [4]

bietet. Theoretisch wären Implementationen aber möglich. Stattdessen kommen spezielle Plattformen zum Einsatz, deren Bedarf an Speicher und Rechenleistung stark reduziert ist und damit auf den Sensorknoten installiert werden können.

2.3 Funktionale Makroprogrammiersprachen

In rein Funktionalen Programmiersprachen bestehen Programmen ausschließlich aus Funktionen, die bei der Ausführung ausgewertet werden. Die Funktionen sind frei von Nebeneffekten, so dass, werden sie mehrfach ausgewertet, sich jedes Mal das gleiche Ergebnis ergibt. Die direkte Veränderung von Programmzuständen wie Variablen ist nicht möglich [6]; sich ändernde Zustände können aber als 'Monaden' implementiert werden.
Diese Festlegung kann für die Sensornetzprogrammierung genutzt werden. Die Ausführung von Programmen in Sensornetzen ist vielen Einschränkungen unterworfen, Fehler können an vielen Stellen aufteten und das System kann sehr unzuverlässig sein. Wird ein funktionales Programm auf einem Sensornetz ausgeführt, hat der Programmierer keinen Einfluss darauf, welche Funktion auf welchem Knoten ausgewertet wird; da auf Knotenzustände nicht zugegriffen werden kann, stellt das kein Hindernis dar. Auf welchem Knoten eine Funktion oder ein Teil davon berechnet wird, ist für das Ergebnis völlig unerheblich; die Funktion kann vorher optimiert, redundant berechnet oder semantisch transformiert werden, ohne dass es für den Programmierer Veränderungen am Funktionsergebnis gibt.
Funktionsparameter können so gegebenenfalls vollständig parallel berechnet werden; dies muss vollständig von der Plattform durchgeführt werden, kann aber so die Effizienz des Netzes steigern.

2.4 Regelbasierte Systeme

Sensorknoten können in alltäglich benutzte Gegenstände eingesetzt werden und auf verschiedenste Arten den Nutzer unterstützen. Über Sensoren können Umgebungseigenschaften registriert und im Zusammenspiel mit anderen Sensorknoten Informationen über den Kontext bestimmt werden. Die Anwendung interpretiert diese Informationen und kann daraus ableiten dass bestimmte Bedingungen erfüllt sind oder ein Ereignis eingetreten ist, auf das mit festgelegten Aktionen reagiert werden soll.
Die Vielzahl der Möglichkeiten der Dienste, die solche *Smart Objects* erfüllen können, ergibt sich schon aus der Vielfalt der Gegenstände, in die sie integriert werden können.

Dazu gehören nicht nur im alltägliche technische Geräte wie Kühlschränke, Mikrowellen usw., sondern praktisch alle vom Menschen genutzten Gegenstände wie z.B. Möbel, Beleuchtung, Verpackungen und vieles anderes. Vorstellbar sind sicherheitsrelevante Installationen, bei denen Menschen unter bestimmten Vorraussetzungen in Gefahr geraten können und dies vom System vorher erkannt und darauf reagiert werden soll [7].
Die meisten dieser Gegenstände werden für triviale Aktivitäten genutzt; die darin eingebetteten Sensorknoten müssen daher einfache Ereignisse erkennen und leicht verständliche Aktionen durchführen, damit sie dem Nutzer einen Mehrwert bieten und ihn nicht durch Übermäßige Komplexität des Systems behindern. Es muss dabei berücksichtigt werden, dass viele Objekte für verschiedene Zwecke eingesetzt werden können und sich darin eingebettete Sensorknoten ihre Aktionen trotz beschränktem Energieverbrauch flexibel an die unterschiedlichen Nutzungsweisen anpassen können müssen [8].
Da die Reaktionen des Sensornetzes auf Ereignisse in der Umgebung nicht übermäßig komplex sein dürfen, kann auch die Makroprogrammiersprache darauf ausgelegt werden. Programme können aus Regeln bestehen, die das Verhalten der Sensorknoten bestimmen. Aus Daten der Sensoren verschiedener Knoten werden Kontextinformationen abgeleitet und durch die Regeln unter festgelegten Bedingungen Aktionen ausgeführt. Vorteil solcher Programmierung ist, dass sich die Programm zwischen Knoten auf unkomplizierte Weise verschicken und in den Knoten speichern lassen. Damit ist das Sensornetz rerogrammierbar.
Für ein solches System muss auf allen Knoten ein Programm laufen, das diese Regeln interpretieren kann und zur Laufzeit ständig auswertet. Der Austausch der Sensordaten muss durch die Plattform erfolgen.

2.5 Query-Engines

Häufig werden Sensorknoten zur reinen Datengewinnung genutzt. Beispielsweise bei Beobachtungen natürlicher Vorgänge wie der Erforschung von Erdbeben werden Sensoren zur Messdatenerfassung eingesetzt; in gewissen Abständen sollen diese Daten oder Teile davon erfasst und weiterverarbeitet werden. In solchen Fällen ist nicht die Programmierung des Sensornetzes mit neuer Funktionalität Ziel des Entwicklers, sondern es sollen vom Sensornetz ausschließlich Daten an eine zentrale Station gesendet werden. Bei dieser Betrachtungsweise auf das Sensornetz sind Topologie und Größe nicht von Interesse und sollen hinter einer Schnittstelle verdeckt sein. *Query-Engines* stellen die Sensordaten dem Entwickler gegenüber wie eine Datenbank mit Tabellen dar, aus denen er die Daten mit einer deklarativen Anfragesprache, meist ähnlich SQL, beliebig auslesen kann. Abbildung 4 verdeutlich dies an einem Beispiel.
Damit ein solcher Ansatz effizient umgesetzt werden kann, muss auf jedem Sensorknoten ein 'Query-Prozessor' laufen, der die Anfragen auswertet und entsprechend weiterleitet. Es soll vermieden werden, dass Sensordaten im Netz verschickt werden, die für die Beantwortung der Anfrage nicht nötig sind und damit unnötige Kommunikation erzeugen.
Die Möglichkeiten solcher Systeme sind dadurch begrenzt, dass sich ihre Flexibilität auf die Einsatzmöglichkeiten der Abfragesprache beschränkt. Teilweise wird diese so erweitert, dass nicht nur Daten aggregiert werden können, sondern auch Kriterien gegeben werden können, unter denen bestimmte Aktionen auszuführen sind. Inwieweit dadurch Programmierung des Netzes möglich ist, hängt von der Leistungsfähigkeit der Spracherweiterungen ab. Nötig ist immer ein zentraler Rechner, von dem aus die Be-

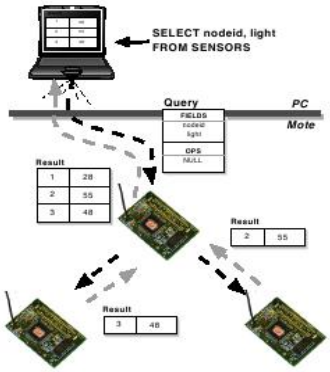


Abbildung 4: Ablauf einer Abfrage in TinyDB [1]

fehle bzw. Abfragen ins Netz gesendet werden.

3 Beispiele für Makroprogrammierspachen

Die große Bandbreite der Einsatzzwecke von Makroprogrammiersprachen erfordert entsprechend unterschiedliche Ansätze und oft sind Neu- oder Weiterentwicklungen notwendig, um unter den gegebenen technischen Rahmenbedingen die Aufgabe zu bewältigen. Manche bauen auf den Entwicklungen anderer Projekte auf und fügen der Makroprogrammiersprache eine Abstraktionsschicht hinzu, so dass Funktionen leichter oder flexibler genutzt werden können. Einige nutzen bestehende Programmiersprachen und adaptieren und erweitern sie. Sensornetze haben andere Eigenschaften als herkömmliche Architekturen, so dass die Sprachen entsprechend angepasst werden müssen. Wird die Syntax und Nutzungsweise dabei größtenteils beibehalten, ist weniger Einarbeitungszeit in das neue System nötig. Gegebenenfalls kann auch Programmcode von der ursprüngilchen Plattform ohne viele Anpassungen Übernommen und so die Entwicklungszeit verkürzt werden.
Es sollen hier aus der Vielzahl der Makroprogrammiersprachen in der Entwicklung oder bereits im Einsatz einige ausgewählt und vorgestellt werden. Die Beispiele sollen verdeutlichen, welcher Aspekt der Makroprogrammierung jeweils im Vordergrund steht und worin die Vorteile für Entwickler bestehen.

3.1 Sensor Network Application Cunstruction Kit (SNACK)

Werden Systemfunktionen einer Plattform zu Modulen zusammengefasst und abstrahiert, ergeben sich Abhängigkeiten zwischen den von den Modulen bereitgestellten Diensten und den darunterliegenden Funktionen. In Sensornetzen, in denen effiziente Datenübertragung wichtig für einen geringen Energieverbrauch ist, müssen Daten möglichst zusammengefasst übertragen werden. Module erschweren dies, in dem der Programmierer weniger Kontrolle über die letztlich ausgeführten Systemfunktionen hat; dies kann die Wiederverwendbarkeit von Programmcode unmöglich machen [2]. Die Komponenten in SNACK, genannt *smart libraries*, sollen deshalb so konstruiert sein, dass sie sich beliebig zu Diensten verknüpfen lassen, damit besonders viel Code

```
[...]
service SenseLight (period: max unint32_t $p = 1000) {
  sense :: my Sense(period = $p)
  sensor :: Photo;
  sense [NodePut16] -> out;
  sense [StdControl] -> sensor;
  sense [ADC] -> sensor;
};
[...]
```

Abbildung 5: Codebeispiel für SNACK

gemeinsam genutzt werden kann. Für die Zusammensetzung der Komponenten zu Applikationen wird die Sprache SNACK verwendet, die von Anwendungsprogrammierern genutzt werden soll, um das Verhalten der Dienste in der Applikation zu bestimmen.
Besonderes Augenmerk wird dabei auf die Flexibilität der Komponenten und auf Sparsamen Umgang mit Speicher gelegt. Da Speicherplatz in Sensorknoten besonders knapp ist, werden Daten meherer Komponenten nach Möglichkeit gemeinsam aufbewahrt. Dies erfordert bei der Entwicklung der Komponenten einen größeren Programmieraufwand, da diese dafür angepasst sein müssen. Um die Komponenten leichter wiederverwendbar zu machen, können ihnen bei der Compilierung des SNACK-Programms Parameter übergeben werden, die bestimmte Eigenschaften des Funktionsverhaltens festlegen.
Der Code in Abbildung 5 ist ein Ausschnitt aus einem Programm, welches Daten der Lichtsensoren in einem Sensornetz aggregiert [2]. Es beschreibt den Dienst *SenseLight*, eine Spezialisierung von *Sense*, der die Daten in der als Parameter `$p` angegebenen Häufigkeit ausliest. In Zeile 2 wird `sense` als Instanz von `Sense` deklariert. Das Schlüsselwort `my` teilt SNACK mit, dass `sense` nicht von verschiedenen Instanzen gemeinsam genutzt werden soll; es kann mehrere unabhängige `Sense`-Instanzen geben. `sensor` wird als Instanz der Komponente `Photo` deklariert (die Komponente `Photo` wird für den Dienst `SenseLight` benutzt); in den Zeilen 5-7 werden die Ausgänge einiger Infrastruktur-Schnittstellen mit den Eingängen benötigter Komponenten verbunden.

3.2 Kairos

Kairos ist eine Programmierplattform, bei der ein zentrales Makroprogramm das gesamte Sensornetz steuern können soll. Kairos besteht aus einer Laufzeitbibliothek, die auf allen Knoten installiert sein muss, und einem Präprozessor, der in den Compiliervorgang der Programmiersprache integriert wird. Es ist als Erweiterung der Programmiersprache Python implementiert, aber auch andere Programmiersprachen wären hier möglich.
Programme in Kairos können Variablen auf Knoten lesen und schreiben, durch die jeweils nächsten Nachbarn iterieren und beliebige Knoten addressieren [3]. Das Verhalten der zur Laufzeit aufgerufenen Funktionen hat die Eigenschaft, dass Zugriffe auf entfernte Knoten das Programm nur so lange blockieren, bis das Objekt selbst initialisiert und verfügbar ist: damit müssen Knoten nicht ständig Daten zur Synchronisation austauschen, was zu Energieeinsparungen bei der Kommunikation führt. Programme

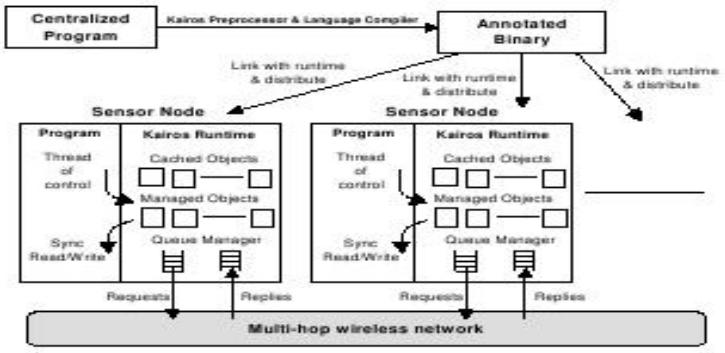


Abbildung 6: Kairos-Architektur [3]

sollen sich trotzdem durch *eventual consitency* dem gewünschten Ergebnis annähern; die verteilten Algorithmen sollen darauf ausgerichtet sein. Viele bereits bekannte Algorithmen für Graphenstrukturen sollen sich nahezu eins-zu-eins implementieren lassen. Die Abstraktionen, um die Kairos die Makroprogrammiersprache erweitert, betrifft den Zugriff auf Knoten, Nachbarknoten und entfernten Datenzugriff. Die Programme können Knoten und Listen von Knoten direkt verändern; zur Identifizierung werden diese automatisch mit Nummern belegt. Auf eine Liste der direkten Nachbarn im Sensornetz, die von der Laufzeitumgebung ständig aktualisiert wird, kann über die Bibliotheken von Kairos jederzeit zugegriffen werden; Daten auf entfernten Knoten können von der Anwendung beliebig manipuliert werden, der Zugriff erfolgt dabei über den Variablenname und Knotennummer, die von der Laufzeitumgebung ermittelt werden. Operationen auf den direkten Nachbarn sind oft das zentrale Element von Kairos-Anwendungen.
Der Vorgang der Übersetzung, Installation und Ausführung eines Kairos-Programms ist in Abbildung 6. Der Präprozessor von Kairos ersetzt den Programmcode (hier in Python) durch neuen annotierten Code, der dann durch den Compiler in ein natives Binärprogramm übersetzt [3] wird. Das Originalprogramm beschreibt den Daten- und Kontrollfluss der Anwendung *im großen* (d.h. für das gesamte Sensornetz), während das durch den Präprozessor veränderte Programm die auf den Knoten lokal ausgeführte Version darstellt. Das Binärprogramm wird gegen die Laufzeitbibliotheken gebunden und auf den Knoten installiert und gestartet. Zur Laufzeit werden lokale und entfernte, auf dem Knoten referenzierte, Objekte durch die Kairos-Umgebung verwaltet. Damit nicht bei jedem Zugriff auf entfernte Objekte die Daten erneut angefragt werden müssen, werden diese im Cache vorgehalten und unabhängig vom Hauptprogramm aktualisiert; eine Warteschlange steuert die Kommunikation.
Das Codebeispiel in Abbildung 7 dient der Initialisierung der Entfernungs-Variablen und der Listen von Nachbarknoten zum späteren Aufbau eines Routing-Baumes. Der Aufruf `get_available_nodes()` sorgt dafür, dass der Code auf jedem der iterierten Knoten ausgeführt wird; die Schleife in Zeile 3 lässt einen Iterator `temp` über jeden Knoten laufen aus. Es wird überprüft, ob `temp` in diesem Durchlauf den eigenen Knoten referenziert (Zeile 7); wenn nicht, wird die Entfernung zu dem Knoten auf 'unendlich' gesetzt, ansonsten auf 0. Mit `create_node_list(get_neighbors(temp))` in Zeile 9 wird auf jedem Knoten eine Liste aller direkten Nachbarknoten erzeugt.

```
1  [...]
2  full_node_set=get_available_nodes();
3  for (node temp=get_first(full_node_set); temp!=NULL;
4        temp=get_next(full_node_set))
5    self=get_local_node_id();
6    if (temp==root)
7      dist_from_root=0; parent=self;
8    else dist_from_root=INF;
9    neighboring_nodes=create_node_list(get_neighbors(temp));
10 [...]
```

Abbildung 7: Codebeispiel für Kairos [3]

3.3 Maté

Die von einer VM festgelegte Funktionalität darf einerseits nicht zu zu einschränkend sein, damit vielseitige Anwendungen möglich sind, muss aber andererseits sparsam mit den Systemressourcen umgehen und effiziente Programme zulassen. Maté [4] unterscheidet sich deshalb von bekannten VMs wie der Java VM dadurch, dass der Programmierer selbst festlegen kann, welcher Funktionsumfang auf den Sensorknoten mitgeliefert wird. Das Ziel ist, durch eine flexible Architektur die Programmumgebung applikationsspezifisch zu machen. Der Programmierer wählt die benötigten Umgebungserweiterungen aus; die Umgebung wird erstellt und zu Beginn auf den Knoten installiert. Es gibt dabei einige Basiskomponenten wie Scheduler und Nebenläufigkeits-Manager, die in jeder Installation vorhanden sein müssen, und beliebige weitere Komponenten, die für die Anwendung bzw. Makroprogrammiersprache benötigt werden. Das Makroprogramm wird zu Bytecode kompiliert und nach bedarf dur ch die VM im Sensornetz verteilt.
Abbildung 8 gibt eine Übersicht über die Struktur einer Maté-VM ohne optionale Erweiterungen. Programmcode wird in Kapseln (*capsules*) gespeichert, in denen er mittels Funkkommunikation auf andere Sensorknoten Übertragen werden kann. Die Aufbewahrung des Codes findet durch den Kapselspeicher (*Capsule Store*) statt. Der Nebenläufigkeitsmanager (*Concurrency-Manager*) verwaltet die Prozesse und verwaltet den Zugriff auf gemeinsam genutzte Ressourcen, um Synchronisationsprobleme zu verhindern. Soll ein Programmkontext ausgeführt werden, wird dieser an den Scheduler übergeben; dieser führt die Codeteile nacheinander aus.
Als Makroprogrammiersprachen werden zur Zeit TinyScript und motlle unterstützt. TinyScript ist eine auf das nötigste begrenzte imperative Sprache mit dynamischer Typisierung. Programme für Maté können dadurch präzise und verständlich formuliert werden. Motlle ist 'C'-ähnlich in der Syntax und deutlich umfangreicher in der Funktionalität [4]. Es ermöglich weit komplexere Programme.
Die Nutzung von Bytecode statt Binärcode für Programme bringt einige Vorteile mit sich, die andere, auf rein compiliertem Code aufbauende Systeme nicht bieten können. Durch Analyse des Bytecodes während der Installation können gegebenenfalls Fehler in der Parallelverarbeitung entdeckt werden. Programmierfehler, die sonst zur Laufzeit schwierig zu entdeckende Fehler hervorrufen, können so frühzeitig entdeckt und beho-

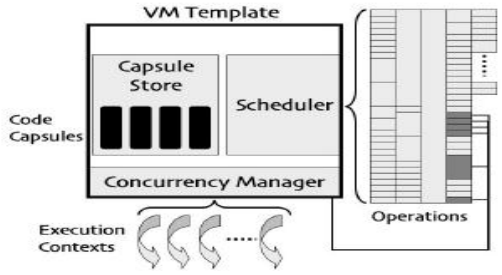


Abbildung 8: Mate-Architektur [4]

ben werden.
Durch die flexible Verteilung des Codes im Sensornetz durch die Laufzeitumgebung ergibt sich das Problem, dass von außen gezielt schädlicher Code in das Sensornetz eingeführt werden könnte. Bei Sicherheitskritischen Anwendungen oder um die Sensordaten vor unbefugtem Zugriff zu schützen, können in Maté Sicherheitsmechanismen genutzt werden. In Maté haben Programme endliche Versionsnummern, so dass nach einer Installation mit der höchstmöglichen Versionsnummer keine Updates möglich sind, also auch nicht von außen. Desweiteren können die Knoten durch Prüfsummen vor unberechtigtem Zugriff geschützt werden.

3.4 TinyDB

In manchen Sensornetzinstallationen ist flexible programmierbarkeit nicht von Interesse. Wenn ausschließlich Sensordaten vieler Knoten nach Bedarf ausgelesen werden können müssen, steht ausschließlich die Datenaggregation im Vordergrund, und TinyDB kann diese deutlich vereinfachen.
Von herkömmlichen Systemen zur Datenhaltung erwartet die Anwendungssoftware, dass von ihr benötigten Daten bereits zum Zeitpunkt des Zugriffs in der Datenbank vorliegen und abgegriffen werden können. Bei Zugriff auf Daten, die von Sensoren in einem Sensornetz mit möglicherweise hunderten oder tausenden Knoten ist diese Voraussetzung nicht gegeben, da sich die von den Sensoren bereitgestellten Daten kontinuierlich ändern können und gegebenenfalls über viele Hops durch das Sensornetz angefragt und wieder zurückgereicht werden müssen.
Dafür ergibt sich in Sensornetzen ein hohes Einsparungspotenzial an Energie, wenn für die Anfrage nötige Kommunikation reduziert werden kann. Anfragen müssen auf den Knoten optimiert werden können, um die Anzahl der einzubeziehenden Sensoren möglichst gering zu halten; die Aufnahme und Verarbeitung der Sensordaten muss aus Effizienzgründen an verschiedenen Stellen stattfindenden können und die Anfragen müssen auf den Knoten hinreichend schnell ausgeführt werden.
TinyDB ist in der Entwicklung bereits weit fortgeschritten. Es wird in zahlreichen Projekten eingesetzt, vor allem zur Beobachtung und Erforschung von Naturereignissen [1]. Auf jedem Knoten im Sensoretz läuft ein Query-Prozessor, der Anfragen in der *Acquisitional Query Language* verarbeiten und sie entsprechend umsetzen kann. Die Struktur der Anfragen ähnelt gewöhnlichen SQL-Statements als 'select-from-where-groupby'-Befehlen. Über die Funktionen herkömmlicher Anfragesprachen hinaus werden Techniken zur Anfrageoptimierung eingesetzt, was den Energieverbrauch deutlich

```
SELECT COUNT(*)
FROM sensors s
WHERE s.light > threshold
GROUP BY s.floor
SAMPLE PERIOD 60s
```

Abbildung 9: Codebeispiel für TinyDB

reduziert. So werden Aspekte berücksichtigt wie der Richtige Zeitpunkt für die Weiterleitung der Anfragen, welche Sensoren konkret benötigt werden, oder in welcher Reihenfolge die Daten am effizientesten ausgelesen werden können.
Das Codebeispiel in Abbildung 9 erzeugt jede 60 Sekunden eine Ausgabe mit der Anzahl der Stockwerke, auf denen Licht brennt.

3.5 Regiment

Ein wichtiger Aspekt größerer Sensornetze ist, dass die für eine Berechnung relevanten Sensordaten oft auf auf einem begrenzten Teil des gesamten Netzes anfallen. So kann sich bei einem Sensornetz, das ein bestimmtes Objekt im Raum verfolgen können soll, das Objekt selbst immer nur in der Nähe eines Teils der Sensorknoten befinden; die Abschätzung, wo sich das Objekt zu einem bestimmten Zeitpunkt befindet, lässt sich also in diesem Fall von dem Teil der Sensorknoten bestimmen, der sich in unmittelbarer Nähe befindet. Es kann daher sinnvoll sein, räumliche Abstände der Knoten voneinander zu nutzen, Datenberechnungen nur auf gerade dafür relevanten Knoten ausführen zu lassen. Die Daten lassen sich dann lokal so weiterverarbeiten und bei Bedarf weiterleiten, dass sich insgesamt erhebliche Einsparungen bei Kommunikation und Energieverbrauch ergeben. *Abstract Regions* ist ein Ansatz, bei der Anwendungsentwicklung räumliche Eigenschaften der Knoten zueinander zu nutzen und dabei die Details von Routing, Datenverteilung und Knotenstatus-Management außer acht zu lassen [9]. Es werden dafür eine Reihe Funktionen zur verfügung gestellt, die jederzeit den Status von Nachbarknoten wiedergeben können und bei der effizienten Verarbeitung der lokalen Daten mit denen anderer Knoten helfen. Welche Eigenschaften für die Gruppierung der Knoten dabei genutzt werden, lässt sich unabhängig festlegen. So können das die k nächsten Knoten sein, alle Knoten innerhalb N *Hops*, oder ganze Baumstrukturen.
Diese Eigenschaften werden von *Regiment* genutzt, einer funktionalen Makroprogrammiersprache. Regiment basiert auf örtlich verteilten und zeitlich variierenden Knotenzuständen [6], sogenannten *Region Streams*. Rein funktionale Programmiersprachen wie Regiment verdecken den aktuellen Status eines Programmes vor dem Programmierer und lassen keine direkte Manipulation von Programmdaten zu. Stattdessen bilden Daten und Programm zu jedem Zeitpunkt Funktionen, deren Ergebnis deterministisch ist, da sich die Daten nicht beliebig ändern können. Diese Festlegung macht die Ausführung von Funktionen sehr flexibel; um die Zuverlässigkeit einer Berechnung zu erhöhen, ist es möglich, die gleiche Operation mehrfach ausführen zu lassen. Da auch keine Ausgaben des Programms geben kann, ist es unerheblich, wo es konkret ausgeführt wird.
Funktionale Makroprogramme sollen sich besonders für die Programmierung von Sensornetzen eignen, da die Eigenschaften funktionaler Sprchen denen von Sensornetzen entgegenkommen. Für diese Anwendungen bedeutet die Abstraktion von direktem Zugriff auf Speicher, dass der Compiler die Entscheidung treffen kann, wo und wann

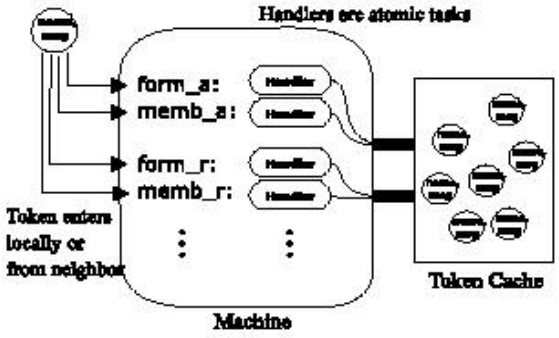


Abbildung 10: Das *Token Machine*-Modell von Regiment [6]

```
let aboveThresh (p,x) = p > threshold
    read node = (readsensor PROXIMITY node,
                    get_location node)
in afilter aboveThresh (amap read world)
```

Abbildung 11: Ein Codebeispiel für Regiment

Programmteile auf den Knoten ausgeführt werden [6]. Ausserdem könne so die Auswertung mehererer Argumente einer Funktion wirklich parallel erfolgen, da der Compiler solche Situationen erkennen könne.
Programme in Regiment werden vom Compiler in eine von einer *Token Machine* verstandenen Zwischensprache (*intermediate language*) übersetzt. Wie in Abbildung 10 dargestellt, besteht diese aus Steuerungsprogrammen (*token handlers*), die durch gleichnamige Token ausgeführt werden. Token können intern oder bei Empfang von Nachrichten mit der entsprechenden Identifikation erzeugt werden. Damit können lokale und entfernte Funktionsaufrufe ausgelöst werden.
Das Codebeispiel in Abbildung 11 gibt ein *Gebiet (area)* zurück mit allen Knoten, deren Sensoren einen Wert über dem Schwellwert (*threshold*) anzeigen. Ausgeführt wird die Funktion *afilter*, die auf alle Knoten in *world* die Funktion *aboveThresh* anwendet und nur die Knoten zurückgibt, für die *aboveThresh* 'wahr' ergibt. *aboveThresh* prüft, ob der jeweilige Sensorwert über einer gegebenen Schwelle liegt.

3.6 Smart Object Systems

Sensorknoten sind inzwischen leistungsfähig genug, um in Alltagsgegenstände eingebettet zu werden. Durch sehr geringen Energieverbrauch können sie dort eine lange Zeit ohne äußeren Eingriff bleiben; bei vielen eingebetteten Sensorknoten entsteht so ein Sensornetz, mit dem der Nutzer interagieren kann und das eine Reihe denkbarer Dienste erfüllen kann. Diese Dienste müssen einfach strukturiert sein, damit sie einen Nutzen darstellen.
Als wichtigste Eigenschaften solcher *Smart Objects* sollen gelten, dass die Knoten vollständig autonom handeln können, sie flexibel auch bei unvorhergesehener Benutzung des Gegenstandes agieren, und sich das Verhalten des Knoten aus dem Zusammenspiel mit anderen Knoten ergibt [8]. Der Mensch nimmt Aktionen in der Regel als

```
Aufgabe: Ein Handy vibriert nur, wenn der Besitzer in der Nähe ist.
E: Ein Anruf kommt.
C: Besitzer in der Nähe?
A: Vibrationsalarm

Aufgabe: Ein Kühlschrank erkennt, wenn die Haltbarkeit
  eines Lebensmittels abgelaufen ist
E: Kühlschrank wird geöffnet
C: Lebensmittel ist abgelaufen
A: Abgelaufene Lebensmittel erscheinen auf Display
```

Abbildung 12: ECA-Regeln am Beispiel

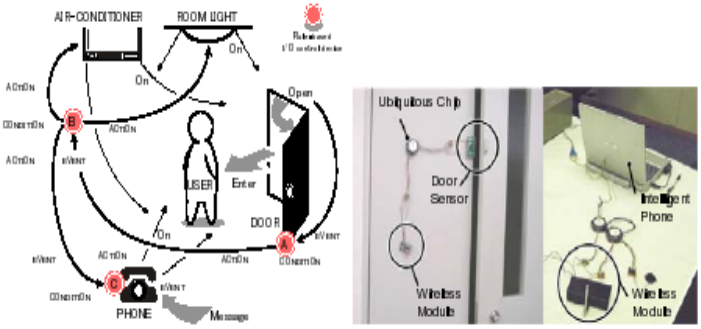


Abbildung 13: Eine Beispielanwendung für Smart Objects [8]

kausal abhängig von Ereignissen an, es bietet sich also an, diese Denkweise auf die Programmierung der Sensorknoten zu übertragen.
Hierfür werden bei *Event-driven Rules* sogenannte ECA-Regeln erstellt und in einer Datenbank gespeichert. Ein *Event* (E) ist dabei ein beliebiges beobachtetes Ereignis, *Conditions* (C) frei festlegbare Bedingungen, die für auszuführende Aktionen gelten müssen, und *Actions* (A) die bei Auslösung des Ereignisses und Erfüllung der Bedingungen auszuührenden Aktionen. Die spezielle Datenbank speichert die Regeln und wertet diese bei Auslösen eines Ereignisses aus und führt ggf. die zugehörigen Aktionen aus.
Diese einfache Beschreibung von Zusammenhängen zwischen Ereignissen und Aktionen lässt einfach bedienbare Funktionalität in Alltagsgegenständen zu, die sich auch im Nachhinein noch flexibel ändern lässt, da nur die Regeln, die die Verhaltensweise des Knotens bei Ereignissen beschreiben, angepasst werden müssen. Die Regeln selbst werden als Strings formuliert, so dass diese sich als kurze Nachrichten innerhalb des Netzwerkes verschicken lassen. Einige Beispiele für ECA-Regeln sind in Abbildung 12 genannt.

4 Evaluation

Es gibt inzwischen eine große Zahl unterschiedlicher Makroprogrammierungsansätze für Sensornetze. In vielen Fällen werden neue Werkzeuge benötigt, um die speziellen Anforderungen der Anwendungen erfüllen zu können. Oft werden dabei Ansätze ähnlich den schon bestehenden gewählt, so dass sich die Zielsetzungen zwischen den Projekten dann überschneiden. Da die Anwendungen in der Regel ein geschlossenes System darstellen, bei dem keine Kommunikation mit anderen Sensornetzen nötig ist, kann dieser Aspekt bei der Entwicklung vernachlässigt werden und es müssen keine Kompromisse bei der Interoperabilität gemacht werden. Wichtiger sind viel mehr geringer Ressourcenverbrauch und damit lange Laufzeiten und zuverlässige Funktion auch beim Auftreten von Störungen.

4.1 Makroprogrammiersprachen

Die beschriebenen Makroprogrammiersprachen unterscheiden sich deutlich in der Abstraktionsebene. Komponentenbasierte Systeme sind erweiterbar und verbergen weniger, erfordern dafür aber eine intensivere Auseinandersetzung mit der zugrunde liegenden Technik. Inwiefern die bestehenden Komponenten dabei übernommen werden können hängt vom Einzelfall ab, im Idealfall sind die Komponenten hinreichend flexibel implementiert und lassen sich unverändert weiterverwenden. Sind aber die Komponenten zu starr an einen Nutzungszweck gebunden, vergrößert sich die Effizienzproblem eher noch, da dann ähnliche Funktionen mehrfach implementiert werden müssen und entsprechend mehr Speicher benötigt wird. Ist der Grad der Abstraktion dagegen sehr niedrig gewählt, ist der Effizienzgewinn durch Wiederverwendbarkeit sehr niedrig. Komponentenbasierte Systeme laufen nach dem Start nativ ohne eine Programmumgebung ab. Teile des Programms lassen sich zur Laufzeit nur austauschen, wenn in der Anwendung dafür Funktionen vorgesehen sind, die den Programmcode installieren und evtl. dynamisch gegen die Komponenten binden können.
Der Vorteil Virtueller Maschinen ist hier, dass deren Programme in kontrollierten Umgebungen ablaufen und die VM in der Lage ist, den Programmcode selbst als Daten im Sensornetz zu übertragen. Damit können von einem zentralen Rechner neue Programme in das Netz eingeführt, verteilt und installiert werden. Bytecode hat hier gegenüber Maschinencode den Vorteil, dass die Programme dank Einsparungen plattformabhängiger Programmteile kleiner sein können als fertig kompilierte Programme. Auch sind die bereits übersetzten Programme nicht an die Hardware gebunden, sondern könnten bei Bedarf auf unterschiedlichen Architekturen eingesetzt werden. Dafür entfällt, verzichtet man auf die Platzeinsparungen und plattformunabhängigkeit durch Bytecode, die Umsetzung von Bytecode auf Maschinencode auf Kosten größerer Programme, was aber aus Effizienzgründen sinnvoll sein kann [5], da die Umsetzung zu Lasten des Prozessors geht.
Gerade für virtuelle Maschinen gilt, dass die Vielfalt möglicher Anwendungen durch die bereitgestellten Schnittstellen begrenzt ist. Anwendungsentwickler sind beschränkt auf die Funktionen der VM, da das System selbst vollständig vor ihnen verdeckt ist. Verlangt man eine hohe Flexibilität, muss die VM entsprechend viele Funktionen bereitstellen. Da sich dies ressourcenschonend nur schwer implementieren lässt, kann die VM in ihrem Umfang selbst auch variabel sein [4]; dann muss der Umfang der VM zum Zeitpunkt der Installation festgelegt werden. Soll dann allerdings eine Anwendung auf unterschiedlichen Sensornetzen laufen, muss sichergestellt sein, dass mindestens der genutzte Funktionsumfang auch auf beiden eingesetzten VMs integriert ist, was dem

System einen Teil seiner Flexibilität wieder nimmt.
Vorzug einer Bytecode verarbeitenden VM ist die vollständige Kontrolle über den Code selbst. Bestimmte Programmierfehler können bereits während der Installation erkannt werden, so dass das System robuster gegenüber Deadlocks und Race-conditions ist und Typsicherheit in den Programmen gewährleistet ist. Solche Fehler sind sonst schwer zu erkennen und können zu unerwartetem Systemverhalten führen.
Da Berechnungen in Sensornetzen verteilt ablaufen, soll der Programmierer möglichst eine Sicht auf das Netz als ganzes haben; das kommt den dynamischen Eigenschaften der Sensornetze entgegen, da sich Routen zwischen Knoten häufig ändern können und Kommunikation sehr unzuverlässig ist. Funktionale Programme können die räumliche Anordnung der Knoten zueinander und deren Sensordaten nutzen, um u.a. die Position von Objekten im Sensornetz festzustellen oder Graphenalgorithmen zu implementieren [3, 6]. Region Streams ist allerdings noch weniger weit forgeschritten als viele andere Systeme, da Regiment viele komplexe Sprachkonstrukte zulässt; Prototypen existieren aber bereits.
Sollen Sensornetze in in Alltagsgegenstände eingebettet werden, bieten sich regelbasierte Systeme wie Smart Objects an. Eine Implementation zeigt, wie auf einem Chip Regeln gespeichert werden, der mit alltäglichen Geräten in einer Wohnung interagiert und so u.a. die Alarmanlage zu steuern hilft, oder beim erneuten betreten eines Raumes die Beleuchtung wieder in den Zustand bei Verlassen bringt [8]. Denkbar sind auch Anwendungen, in denen die Benutzer selbst Regeln festlegen oder verändern; dann müssen diese allerdings auch ohne Programmierkenntnisse verständlich sein.
Weiterhin auf Regeln basiert ist eine Anwendung zur Erhöhung der Sicherheit von Gefahrengut [7]. Sensorknoten werden an Behälter angebracht und sollen dafür sorgen, dass bestimmte Bedingungen nicht ohne Warnung eintreten können; so sollen bestimmte chemische Substanzen, deren Reaktion miteinander Giftstoffe freisetzen kann, nicht nah beieinander gelagert werden. Im Programm, das solche Situationen erkennen können soll, steht die Ableitung von Wissen über Nähe und Ort der anderen Knoten und den Informationen über das Gut im Vordergrund, um rechtzeitig Gefahrensutuationen erkennen zu können. Die Verarbeitung der Daten erfolgt dabei nicht auf einer zentralen Steuerungseinheit sondern in den Sensorknoten selbst, da bei Transport der Güter sonst keine durchgehende Überwachung realistisch wäre.
Solche Systeme haben gegenüber anderen Makroprogrammiersprachen den Vorteil, dass sie leichter reprogrammierbar sind und für die gewünschten Einsatzzwecke völlig ausreichen. Sie sind aber gegenüber frei programmierbaren Systemen auch weit weniger flexibel und die Details der Netzeigenschaften werden vollständig verdeckt.

4.2 Datenorientierte Ansätze

Sind nur die Daten der Sensoren in den Knoten von Interesse und sollen diese zentral gesammelt werden, können rein datenorientierte Ansätze wie TinyDB die Komplexität des Senornetzes verbergen. Die Daten können an einer zentralen Schnittstelle aus dem Sensornetz angefragt und ausgelesen werden. Genutzt wird dies bereits in Installationen zur Erforschung und Überwachung der Umwelt [1]: Auf Great Duck Island wurden 2003 die Höhlen eines vom aussterben bedrohten Seevogels beobachtet [Mainwaring et al. 2002]. Weitere Sensornetze soll zur Erdbebenerforschung [UC Berkeley 2001] und anderem eingesetzt werden
Vorteil eines solchen Ansatzes ist vor allem die Einfachheit, mit der so Daten beliebig vieler Knoten ausgelesen werden können. Die Netzstruktur ist vollständig abstrahiert

und die Programmierung entfällt. Da bei vielen Experimenten Daten nur selten angefordert werden, können die Systeme sparsam mit den Ressourcen umgehen und lange Laufzeiten ohne manuelle Eingriffe realisiert werden [1].
Das gesamte Spektrum der Einsatzmöglichkeiten ist durch die Abfragesprache beschränkt. Programmieren der Sensorknoten ist nicht möglich, zudem muss die Programmumgebung auf jedem Knoten zu Beginn fest installiert sein. Der Entwickler hat auch keinen Einfluss auf die verwendeten Algorithmen für Datenaggregation und Routing, da die Sicht auf die Kommunikation verdeckt bleibt. Für viele Aufgaben sind solche Netze also zu unflexibel.

5 Fazit

Diese Arbeit soll eine Übersicht geben, in der die Herausforderungen, die Sensornetzte an die Programmierung stellen, dargestellt werden, und wie Ansätze zur Makroprogrammierung aussehen können. An einigen Beispielen realer Projekte lässt sich erkennen, wie einzelne Aspekte handhabbar und Sensornetze für verschiedenste Anwendungen genutzt werden. Bei der Vielzahl der Projekte, die sich mit neuen Anwendungen und unterschiedlichen Herangehensweisen daran beschäftigen, ist diese Übersicht bei weitem nicht vollständig, sondern soll die grundlegenden Aspekte beschreiben, die Programmierung von Sensornetzen von der Programmierung anderer Systeme unterscheiden.
Das Forschungsgebiet Sensornetze ist nach wie vor recht jung, und es sind noch viele Einsatzzwecke vorstellbar, die von den jetzigen Ansätzen noch nicht abgedeckt werden. Wann sich welche Herangehenweise besser als andere eignet, muss in Prototypen getestet und ausführlich untersucht werden; vieles ist noch nicht ausgereift, und manche Probleme wie Sicherheitsaspekte lassen sich mit sparsamen Umgang mit Ressourcen schwer vereinbaren.
Trotzdem lässt sich erkennen, wie Aspekte wie Abstraktion der Hardware, Energieeffizienz, Leistung und einfache Programmierung gegeneinander abgewogen und wie in Zukunft Makroprogrammiersprachen die Entwicklung komplexer Anwendungen erleichtern können werden.

Literatur

[1] Samuel R. Madden et al. Tinydb: An acquisiional query processing system for sensor networks.

[2] Ben Greenstein, Eddie Kohler, and Deborah Estrin. A sensor network application construction kit (snack).

[3] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macroprogramming wireless sensor networks using kairos.

[4] Philip Lewis, David Gay, and David Culler. Bridging the gap: Programming sensor networks with application specific virtual machines. *OSDI*, 2004.

[5] Ryan Newton, Arwind, and Matt Welsh. Building up macroprogramming: An intermediate language for sensor networks.

[6] Ryan Newton and Matt Welsh. Region streams: Functional macroprogramming for sensor networks. *DMSN*, 2004.

[7] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology.

[8] Tsutomu Terada and Masahiko Tsukamoto. Smart object systems by event-driven rules.

[9] Matt Welsh and Geoff Mainland. Programming sensor network using abstract regions. *Proceedings of the First USENIX Symposium on Networked Systems Design and Implementation*, March 2004.

A Übersicht über einige Makroprogrammmiersprachen für Sensornetze

Name	*Typ*	*Ausgeführter Code*	*Sprache*	*Host-System*	*Implementation*	*Beispielanwendung*
SNACK	komponentenbasiert	Binärcode	verschiedene möglich, v.a. Makroprogrammiersprachen	TinyOS	existiert	
Kairos	High-Level Framework	Binärcode	Kairos (Python-ähnlich, mit speziellem Präprozessor)	TinyOS	Prototyp	Object-Tracking
Maté	Virtuelle Maschine	Bytecode	Diverse möglich, zur Zeit TinyScript, Motlle	TinyOS	existiert	
TinyDB	Query Engine	interpretierte Anfragen	TinySQL	TinyOS / Mate-VM	Prototyp	Erforschung der Natur
Region Streams	Middleware für funktionale Makroprogramme	Regiment-Quellcode	Regiment	Token-Machine	Prototyp	
Smart Objects System	regelbasiert	ECA-Regeln	ECA (Event/ Condition/ Action)-Regeln		existiert	Smart Objects
Cooperative Artefacts	regelbasiert		Prolog-ähnlich		Prototyp	Sicherheit bei Gefahrengut

Kontext-Modelle für Ubiquitäre Systeme

Autor
Claus Krekeler

Claus.Krekeler@web.de

Seminarbetreuer
Tobias Zimmer

Telecooperation Office (TecO) – Institut für Telematik **an der Fakultät für Informatik** der Universität Karlsruhe
Vincenz-Prießnitz-Str. 3, D-76131 Karlsruhe
Zimmer@TecO.edu

Abriss: Ein wichtiger Bestandteil des *Ubiquitous Computing* ist das Kontext-Bewusstsein des ubiquitären Systems, *Context-Awareness*. Um einem Rechensystem ein solches Bewusstsein zu ermöglichen, benötigt man einen Mechanismus, der die reale Welt – die Welt, wie der Benutzer sie wahrnimmt – in eine „computerverstehbare" Form bringt. Diese Brücke zwischen realer und virtueller Welt wird von Kontext-Modellen geschlagen.
In dieser Seminararbeit sollen einige Ansätze für die Kontext-Modellierung in UbiComp-Systemen verglichen und deren Vor- und Nachteile betrachtet werden. Hierzu möchte ich 6 Hauptkategorien von Modellierungsansätzen, anhand von einigen Anforderungen, die allgemein an Kontext-Modelle in ubiquitären Umgebungen gestellt werden können, bewerten.

1 Einführung

Der Begriff ***Ubiquitous Computing*** wurde von Mark Weiser Anfang der 1990-er Jahre in seinem Artikel »The Computer for the 21st Century« [36] geprägt.
Die Vision des Ubiquitous Computing (UbiComp) ist die Vorstellung, dass uns die Hilfsmittel der elektronischen Datenverarbeitung in Zukunft an jedem Ort und zu jeder Zeit zur Verfügung stehen können. Deren Anwendung soll so einfach werden, dass sie jeder – ohne Vorkenntnisse – wie selbstverständlich nutzen kann. Der Computer wird so zum Alltagsgegenstand und der Benutzer nimmt ihn nicht mehr bewusst als Maschine, d.h. Fremdkörper, war. Die Integration in unsere Umwelt soll so nahtlos geschehen, dass Norman von »Disappearing Interfaces« spricht [23].
Um dies zu erreichen ersetzt man einerseits den monolithischen „Alleskönner" PC durch verteilte Systeme, deren Komponenten hoch spezialisiert und intelligent vernetzt sind, und andererseits trennt man sich weitgehend von den herkömmlichen Ein-/ Ausgabegeräten, wie

Tastatur, Maus und Bildschirm, und gibt dem Computersystem ein „Kontext-Bewusstsein". – Durch die Verteilung der Systemkomponenten erhält man mehr Flexibilität und eine maximale Leistung zur richtigen Zeit am richtigen Ort, während durch die Veränderung der Schnittstellen die Bedienung der Systeme einfacher und intuitiver gestaltet werden kann, denn die Maschine kann auf diese Art nicht nur ihre eigene virtuelle, sondern auch unsere reale Welt erkennen und diese Informationen als Input für ein situationsabhängiges Operieren nutzen.

Context-Awareness

Bevor ich auf Kontext-Modelle eingehe, möchte ich zunächst noch festhalten, wie die Begriffe *Kontext (Context)* und *Kontext-Bewusstsein (Context-Awareness)* in dieser Arbeit definiert sind. Diese Definitionen werden auch von Dey verwendet [9].

Context: Als Kontext wird jegliche Information bezeichnet, die die Situation einer Entität beschreibt. Entitäten sind Personen, Objekte oder Lokalitäten, die relevant für die Interaktion zwischen einem Benutzer und einer Anwendung sind. Der Benutzer und die Anwendung selbst gehören hierbei ebenfalls zur Menge der Entitäten.

Context-Awareness: Ein System ist „context-aware", wenn es Kontext verwendet, um dem Benutzer relevante Informationen und / oder Dienste anzubieten. Dabei hängt die Relevanz vom Auftrag des Benutzers ab.

Kontext-Modelle

Die Brücke zwischen realer und virtueller Welt kann, wie bereits gesagt, durch Kontext-Modelle gebildet werden, die unsere Umwelt und die, darin enthaltenen, Entitäten – sowie deren Beziehungen, Eigenschaften und Veränderungen – in elektronischer Form so darstellen, dass Applikationen diese Informationen für ihre Zwecke nutzen können.

Je weiter ein Kontext-Modell seine Beschreibung des Kontextes von der eigentlichen Kontext-Erfassung durch Sensoren abstrahiert und je besser es gelingt, die Kontext-Beschreibung von der eigentlichen Aufgabe einer Applikation zu entkoppeln, desto einfacher wird die Verwendung des Modells für den Benutzer [6]. Durch die Abstraktion bleibt es dem Programmierer / Benutzer erspart, sich mit technischen Einzelheiten der Kontext-Erfassung auseinandersetzen zu müssen, und

durch eine gute Entkopplung wird es für den Applikationsprogrammierer einfacher, sich auf die eigentliche Aufgabe seiner Anwendung zu konzentrieren – unabhängig von der Art der Eingabeschnittstelle.
Meiner Meinung nach, sind diese beiden Forderungen auch wichtig, wenn man ein Kontext-Modell in einer – bereits existierenden – UbiComp-Umgebung (s. Abschnitt 2, Anforderung 6) einsetzen möchte.

Aufbau dieser Arbeit
In Abschnitt 2 nenne ich zunächst einige Anforderungen, die Kontext-Modelle für ubiquitäre Systeme erfüllen sollten. In Abschnitt 3 stelle ich 6 Kategorien von Kontext-Modellen und einige ihrer Vertreter kurz vor.
In Abschnitt 4 versuche ich dann, die, in Abschnitt 3 vorgestellten, Modelle mit Hilfe der, in Abschnitt 2 genannten, Anforderungen zu bewerten. Schließlich gibt es in Abschnitt 5 noch eine tabellarische Zusammenfassung der Ergebnisse.
Meine Arbeit stützt sich – natürlich nicht allein, aber doch hauptsächlich – auf das Paper von Strang und Linnhoff-Popien [30].

2 Anforderungen ubiquitärer Systeme an Kontext-Modelle

Durch den – bereits in der Einführung beschriebenen – verteilten und sich ständig verändernden Aufbau des Systems und die Komplexität der Kontext-Informationen, werden an UbiComp-Systeme äußerst hohe Anforderungen gestellt. Strang und Linnhoff-Popien nennen hierzu folgende [30]:

1. *Fließender Verteilter Aufbau (FVA)*: Jedes ubiquitäre System leitet sich von einem verteilten System ab. Es gibt im UbiComp jedoch – speziell bei der Kontext-Erkennung – keine zentrale Ebene, die für die Erzeugung, den Einsatz und die Verwaltung der Daten und Dienste verantwortlich ist. Durch die, sich ständig im Fluss befindlichen, Kontext-Daten der Umgebung ändern sich stattdessen Zusammensetzung und Administration eines Kontext-Modells und seiner Daten mit einer sehr hohen Dynamik im Bezug auf Zeit, Netztopologie und Ursprung.

Ein Kontext-Modell sollte daher Mechanismen haben, die ein gutes Handling dieser Dynamik unterstützen.

2. *Partielle Validierung (PV)*: Es ist wichtig, dass man Kontext-Wissen auf der Struktur- und der Instanz-Ebene in einem Modell nachbilden und vor allem zur Laufzeit partiell validieren kann – auch wenn dieses Kontext-Wissen, auf Grund seines fließenden verteilten Aufbaus, an keinem speziellen Ort und zu keinem Zeitpunkt als eindeutiges und umfassendes Ergebnis zur Verfügung steht.

   Das ist besonders wichtig wegen der komplexen Kontext-Beziehungen, die jeden Modellierungsversuch fehleranfällig machen.
3. *Informationsqualität und -quantität (Qua)*: Die Informationsqualität der Sensorwerte und die Informationsvielfalt verschiedener Sensorarten, die eine Entität in einer UbiComp-Umgebung charakterisieren, variieren mit der Zeit.

   Daher sollte ein, für den Einsatz in UbiComp-Systemen gedachtes, Kontext-Modell grundsätzlich Erkennungsmechanismen für Qualität und Quantität besitzen, um relevante Daten herausfiltern zu können.
4. *Unvollständigkeit und Mehrdeutigkeit (UuM)*: Die verfügbaren und relevanten Kontext-Informationen, die eine Entität über die Zeit charakterisieren, sind normalerweise unvollständig und / oder nicht eindeutig – besonders, wenn diese Informationen aus Sensornetzen stammen.

   Das sollte durch das Kontext-Modell aufgefangen werden, z.B. durch Interpolation der fehlenden Daten.
5. *Grad der Formalität (GdF)*: Es ist schwierig kontextbezogene Gegebenheiten und Beziehungen präzise und verständlich darzustellen. Dennoch ist es notwendig, dass jede beteiligte Partei in einer UbiComp-Umgebung die Bedeutung der Informationen gleich Interpretiert und dieselben Begriffe verwendet (*shared understanding*).
6. *Anwendbarkeit innerhalb bereits existierender UbiComp-Umgebungen (Anw)*: Aus der Implementierungsperspektive ist es wichtig, dass ein Kontext-Modell auch in der Infrastruktur einer existierenden UbiComp-Umgebung lauffähig ist, z.B. durch ein Framework wie Web-Services.

3 Kategorien von Kontext-Modellen

In diesem Abschnitt werden 6 Kategorien von Kontext-Modellen vorgestellt. Die Auflistung dieser Kategorien, und die der darin enthaltenen Modelle, ist nicht vollständig, aber es handelt sich um die wichtigsten Vertreter für die Modellierung von kontextbezogenem Wissen. Einige der Modelle lassen sich in mehrere Kategorien einordnen. In diesem Fall werden sie in der Kategorie angeführt, in die sie am besten hinein passen.

3.1 Key-Value Modelle

Dies ist der simpelste und wahrscheinlich älteste Ansatz zur Kontext-Modellierung. Man gibt dem System ein Schlüsselwort, das von Interesse ist, und noch einen oder mehrer Werte, die den Kontext wiedergeben, und man erhält ein entsprechendes Ergebnis.
Ein Vertreter dieser Kategorie kommt im PARCTab-Projekt zur Anwendung [35]. Mit dem System lassen sich Personen, Objekte oder Dienste in einer Büroumgebung orten bzw. das Verhalten von Applikationen, entsprechend des Aufenthaltsorts des PARCTabs beeinflussen. Der Kontext ist hier das Büro, in dem sich ein PARCTab aufhält – also der Ort. Applikationen können diesen über einen *Nameservice* abfragen.
Bekannt ist auch JINI. Bei einem „Gelbe Seiten"-Mechanismus können sich hier Ressourcen- und Dienstanbieter anmelden. Interessierte Clients können sie dann in der Liste finden, z.B. suche
$\underbrace{\text{Tintenstrahldrucker}}_{\text{key(-word)}}$ mit $\underbrace{\text{300 dpi}}_{\text{value}}$.
Dieses Modell ist, auf Grund seiner simplen Listenstruktur, einfach zu implementieren und zu verwalten, eignet sich aber nicht für die Modellierung komplexerer Kontext-Beschreibungen.

3.2 Markup Scheme Modelle

Diese Art von Modellierungsansätzen hat eine hierarchische Datenstruktur, bestehend aus so genannten *Markup Tags* mit Attributen und Inhalten. Dabei sind insbesondere die Inhalte der Markup Tags meist rekursiv durch andere Tags definiert.

Typische Repräsentanten hierfür sind *Profiles*. Sie gehen normalerweise auf die *Standard Generic Markup Language (SGML)* zurück, der Oberklasse aller Markup-Sprachen, wie dem bekannten XML.
Einige dieser Modelle sind als Erweiterungen der Standards *Composite Capabilities / Preferences Profile (CC/PP)* [31] und *User Agent Profile (UAProf)* [34] definiert. Diese Ansätze erweitern normalerweise das Basisvokabular und die Prozeduren von CC/PP und UAProf, um die höhere Dynamik und Komplexität von Kontext-Informationen, gegenüber statischen Profiles, in den Griff zu bekommen.
Ein Beispiel dieser Ansätze sind die *Comprehensive Structured Context Profiles (CSCP)* von Held et al. [13]. Anders als CC/PP, definiert CSCP keine feste Hierarchie. Vielmehr bietet es die volle Flexibilität von RDF/S, um die natürliche Struktur von Profil-Informationen, wie sie von Kontext-Informationen benötigt werden, auszudrücken. Die Namen von Attributen werden, entsprechend ihrer Position in der Profilstruktur, kontextsensitiv interpretiert. Dadurch ist eine einheitliche Namensgebung durch das ganze Profil hindurch, wie es bei CC/PP notwendig ist, nicht erforderlich. Ein weiterer Nachteil von CC/PP, wurde durch einen flexibleren `override`- und `merge`-Mechanismus behoben, der z.B. das Überspringen und / oder Zusammenführen eines ganzen Profilzweigs erlaubt.
Ein ähnlicher Ansatz, wie CSCP, ist *CC/PP Context Extension* von Indulska et al.. [20]. Die Autoren erweiterten das Basisvokabular von CC/PP und UAProf um eine Anzahl von Komponentenattributen bezüglich einiger Kontext-Aspekte, z.B. Ort, Netzwerkcharakterisierung und Applikationsanforderungen sowie spezielle Typen von Beziehungen und Abhängigkeiten. Sie fassten zusammen, dass ihr Ansatz in der Lage sei, Applikationen und anderen Bestandteilen einer ubiquitären Infrastruktur Context-Awareness zu ermöglichen. Allerdings sei es schwierig, komplexe Kontext-Beziehungen und -Randbedingungen festzulegen, da der Ansatz auf CC/PP basiere.
Ein anderer Vorschlag, der nicht auf CC/PP basiert, ist *Pervasive Profile Description Language (PPDL)* [7]. Diese XML-basierte Sprache erlaubt es, Kontext-Informationen und -Abhängigkeiten zu erklären, wenn sie Interaktionsmuster in einem begrenzten Maßstab definieren. Die Anzahl beschreibbarer Kontext-Aspekte und die Verständlichkeit der Sprache selbst, scheinen relativ begrenzt zu sein.

Da keine Design-Kriterien und nur Teile der Sprache der Öffentlichkeit zugänglich sind, bleibt der eigentliche Wert des Kontext-Modells unbekannt.

3.3 Grafische Modelle

Ein bekanntes Allzweck-Modellierungsinstrument ist die *Unified Modeling Language (UML),* die starke grafische Elemente (UML-Diagramme) hat. Wegen ihrer allgemeinen Struktur ist UML auch geeignet, um Kontext zu modellieren. Das wird z.B. bei Bauer [4] gezeigt, wo Kontext-Aspekte für den Flugverkehr als UML-Erweiterungen modelliert werden.
Ein anderes Beispiel ist das Modell von Henricksen et al. [17], welches eine Erweiterung zu *Object-Role Modeling (ORM)* [11] bezüglich einiger Kontext-Klassifikationen und -Beschreibungen ist [19]. Eine Domäne wird in ORM modelliert, indem man Sachverhalte (*facts*) und die Rollen, die Entitäten in diesem Zusammenhang spielen, benennt. Die Erweiterungen von Henricksen et al. ermöglichen es, die *facts* – je nach ihrer Persistenz und Bezugsquelle – kategorisieren zu können (s. Anhang). Hiernach sind Sachverhalte entweder *statisch* (facts, die unverändert bleiben, solange die zugehörigen Entitäten bestehen) oder *dynamisch*. Letztere werden entsprechend der Bestimmungsquelle noch weiter unterteilt. Es gibt zugewiesene (*profiled)*, gemessene (*sensed*) und hergeleitete (*derived*) *fact-Typen*. Um auch einen Zeitaspekt des Kontextes darstellen zu können, wurde bei Henricksen et al. ein *history-fact-type* eingeführt. Die letzte Erweiterung der Autoren zu ORM ist die Möglichkeit, Abhängigkeiten zwischen *facts* darzustellen. Die Änderung des einen Faktums induziert hier automatisch die Änderung eines anderen (*DependsOn-Relation*).

3.4 Objektorientierte Modelle

Alle objektorientierten Kontext-Modelle nutzen *Abgeschlossenheit* („Black Box"-Prinzip) und *Wiederverwendbarkeit*, um die Dynamik der Kontext-Informationen in ubiquitären Umgebungen in den Griff zu bekommen. Die Einzelheiten der Kontext-Verarbeitung sind auf der Objekt-Ebene gekapselt und daher für andere Komponenten nicht

sichtbar. Zugang zur Kontext-Information kann nur über spezifizierte Schnittstellen genommen werden.
Ein Beispiel für diese Ansätze sind die *cues* [25], die im TEA-Projekt [12, 26] entwickelt wurden. Das cue-Konzept sieht eine Abstraktion von physischen und logischen Sensoren vor. Ein *cue* ist eine Funktion, die einen Sensorwert in einen symbolischen oder sub-symbolischen Wert umwandelt. Eine endliche oder unendliche Menge von möglichen Werten wird für jeden cue definiert. Die Ausgabe von jedem cue hängt von einem einzelnen Sensor ab, aber verschiedene cues können auf denselben Sensoren basieren. Der Kontext ist als eine Abstraktionsebene an der Spitze aller verfügbaren cues modelliert. Somit sind die cues Objekte, die Kontext-Informationen durch ihre Schnittstellen zur Verfügung stellen, ohne deren Herkunft preiszugeben.

3.5 Logik-basierte Modelle

Eine Logik definiert die Regeln, durch die ein zusammengesetzter Ausdruck oder Fakt von einer Menge anderer Ausdrücke oder Fakten abgeleitet werden kann. Zur Beschreibung der Bedingungen in einer Menge von Regeln wird ein formales System verwendet.
In einem Logik-basierten Kontext-Modell ist folglich der Kontext durch Fakten, Ausdrücke und Regeln definiert. kontextbezogene Informationen werden in einem solchen System gewöhnlich durch Fakten bzw. durch Schlussfolgerungen aus den Regeln des Systems geändert, d.h. hinzugefügt, aktualisiert oder gelöscht. Alle Logik-basierten Modelle haben einen hohen Grad an Formalität.
Einer der ersten Logik-basierten Ansätze wurde als *Formalizing Context* Anfang 1993 von McCarthy und seiner Gruppe in Stanford [21, 22] veröffentlicht. McCarthy vermied es nachdrücklich Kontext explizit zu definieren. Stattdessen versuchte er ein Formalisierungsrezept anzugeben, das es erlaubte, einfache, allgemein geltende, statische Phänomene als Axiome darzustellen, die sich zu einem komplexeren und sich verändernden Kontext abstrahieren lassen.
Der Schwerpunkt von Giunchiglias Ansatz, der manchmal als *Multicontext Systems* bezeichnet wird, liegt weniger auf der Kontext-Modellierung als auf der Kontext-Begründung [15, 14]. Seiner Meinung nach ist Kontext die spezifische Teilmenge des Gesamtstatus einer individuellen Entität, die für die Begründung eines gegebenen

Ziels verwendet wird; Kontext kann als (Teil-)Theorie der Welt gesehen werden, die eine individuelle subjektive Perspektive von ihr kodiert.
Ein anderer früher Ansatz ist die *Extended Situation Theory* von Akman und Surav [1], der eine Erweiterung der, von Barwise und Perry stammenden *Situation Theory* ist [3]. Barwise und Perry versuchten, die modell-theoretische Semantik von natürlicher Sprache in ein formal logisches System zu fassen. Akman und Surav verwendeten und erweiterten das System, um den Kontext mit Situationstypen zu modellieren, die einfache Situationen sind und daher erstklassige Objekte der Situation Theory. Die Vielfalt verschiedener Kontexte wird in Form von Regeln und Vorraussetzungen, bezogen auf einen bestimmten Gesichtspunkt, angesprochen. Sie repräsentieren die Fakten, bezogen auf einen speziellen Kontext mit parameterfreien Ausdrücken, unterstützt von dem Situationstyp, der zum Kontext korrespondiert.
Ein ähnlicher Vorschlag ist das *Sensed Context Model*, das von Gray und Salber vorgestellt wurde [10]. Sie verwenden die Prädikaten-Logik erster Ordnung als formale Darstellung von kontextbezogenen Sätzen und Relationen.
Ein weiterer Ansatz ist das *Multimedia System* von Bacon et al. [2]. In diesem System ist der Ort, als ein Kontext-Aspekt, als Fakten in einem regelbasierten System ausgedrückt. Das System selbst ist in PROLOG implementiert.

3.6 Ontologie-basierte Modelle

Nach Gruber [16] ist eine Ontologie eine explizite Spezifikation für die Erstellung eines Konzepts. Der Begriff stammt aus der Philosophie, wo die Ontologie eine systematische Beschreibung dessen ist, was existiert.
Im Bereich der Informatik ist das „Existierende“ alles das, was verkörpert werden kann, d.h. beschreibbar ist. Wenn Wissen über eine Domäne durch einen deklarativen Formalismus repräsentiert wird, so nennt man die Menge der beschreibbaren Objekte „Argumentationsraum“ (*universe of discourse*). Diese Menge von Objekten, und deren beschreibbare Relationen, spiegeln sich wider im Beschreibungsvokabular eines wissensbasierten Programms, das Wissen repräsentiert.

In der Informatik kann man deshalb die Ontologie eines Programms beschreiben, indem man eine Menge von „figürlich“ darstellbaren Begriffen definiert. In einer solchen Ontologie assoziieren Definitionen die Namen der Entitäten im „Argumentationsraum“ (z.B. Klassen, Relationen, Funktionen usw.) mit natürlichsprachlichem Text, der die Bedeutung der Namen beschreibt. Formale Axiome bestimmen die Interpretation und die richtige Verwendung dieser Begriffe.
Ontologien spezifizieren also, wie man die Welt beschreiben kann – sowohl die reale als auch die virtuelle Welt. Sie sind deshalb besonders geeignet, um Teile der Informationen, die in unserem täglichen Leben repräsentiert und verwendet werden, auf computerlesbare Datenstrukturen abzubilden.
Ein Versuch, Kontext mit Hilfe von Ontologien zu modellieren wurde erstmals von Ötztürk und Aamodt unternommen [24]. Sie untersuchten psychologische Studien auf die Unterschiede zwischen „sich erinnern“ und „wiedererkennen“ bezüglich verschiedener Ergebnis kombiniert mit kontextbezogener Information. Von dieser Untersuchung leiteten sie die Notwendigkeit ab, das Wissen von unterschiedlichen Bereichen zu normalisieren und zu kombinieren. Sie stellten ein, auf Ontologie basierendes, Kontext-Modell vor, das sich auf diese Untersuchungen stützte.
Ein weiterer Ansatz wurde als *Aspect-Scale-Contextinformation (ASC)* vorgestellt [27]. Die Verwendung von Ontologien stellt einen einheitlichen Weg dar, sowohl die Kernkonzepte des Modells, als auch eine beliebige Anzahl von Sub-Konzepten und Sachverhalten zu spezifizieren, die es gemeinsam ermöglichen, Kontext-Wissen in einem ubiquitären Computersystem zu sharen und wieder zu verwenden [8]. Das Modell wurde mit ausgewählten Ontologiesprachen implementiert. Diese Implementationen bilden den Kern von einer nicht monolithischen *Context Ontology Language (CoOL)*, welche durch Integrationselemente, wie Erweiterungen für Web-Services und andere ergänzt wurden [28, 29]. Abgesehen davon, dass die Sprache die Interoperabilität von Services bezüglich ihrer kontextbezogenen Kompatibilität und Ersetzbarkeit bestimmt, wird sie auch verwendet, um in verteilten Service-Frameworks verschiedenartigen Applikationen *Context-Awareness* anzubieten.
Das Modell *CONON* von Wang et. al. [32, 33] beruht auf der gleichen Idee wie ASC/CoOL. Das Modell wurde entwickelt, weil es Knowledge Sharing, logische Schlussfolgerungen und

Wiederverwendbarkeit von Wissen ermöglicht. Wang et. al. entwarf eine übergeordnete Ontologie, die grundlegende kontextbezogene Charakteristika von Entitäten und eine Sammlung von bereichsspezifischen Ontologien mit deren Besonderheiten in jeder Sub-Domäne einschließt. Die CANON-Ontologien sind in OWL-DL serialisiert, wobei diese Sprache eine semantische Äquivalenz zu gut erforschten Logiken besitzt. Das erlaubt den Einsatz von Inferenzmaschinen, die zur Konsistenzprüfung und für kontextbezogene Schlussfolgerungen bei Beschreibungssprachen entwickelt wurden.
Ein weiteres viel versprechendes Kontext-Modell ist das *CoBrA*-System [5]. Dieses System stellt eine Menge von ontologischen Konzepten zur Verfügung, die zur Charakterisierung von Entitäten, wie Personen, Orten oder einigen anderen Arten von Objekten, innerhalb ihrer Kontexte dient. CoBrA verwendet eine Broker-zentrierte Agenten-Architektur, um Laufzeitunterstützung für Systeme mit Kontext-Bewusstsein zur Verfügung zu stellen (z.B. intelligente Meetingräume).

4 Bewertung der Modelle

Im Folgenden werden die, in Abschnitt 3 vorgestellten, Modelle geprüft, inwieweit sie die, in Abschnitt 2 genannten, Anforderungen erfüllen. Leider sind bei einzelnen Modellen nicht alle Details der Öffentlichkeit zugänglich. In diesen Fällen können die entsprechenden Ansätze mit den Eigenschaften anderer Ansätze in der gleichen Kategorie verglichen werden.

4.1 Key-Value Modelle

Wie in 3.1 bereits erwähnt, ist diese Kategorie allgemein ungeeignet für die Beschreibung komplexer Kontext-Informationen. Abgesehen davon entsprechen die, darin enthaltenen, Kontext-Modelle aber auch den Anforderungen 1 bis 5 nur schwach.
Der fließende verteilte Aufbau (Anforderung 1) sowie die Unvollständigkeit (s. Anforderung 4) werden nur auf der Instanz-Ebene erfüllt. Es gibt keine Hilfsmittel zur Fehlererkennung und -bewertung

während der Laufzeit, was die partielle Validierung (Anforderung 2) schwierig und Such-Algorithmen fehleranfällig macht.
Die simple Bauweise der Key-Value Modelle – Schlüsselwort+(exakter) Wert – ist ein Vorteil im Bezug auf Management und geringes Fehlerrisiko. Diese Einfachheit ist aber dann ein Nachteil, wenn Mehrdeutigkeiten bzw. qualitative Meta-Informationen berücksichtigt werden sollen (s. Anforderungen 4 und 3).
Eine Stärke dieser Modelle ist es allerdings, dass der Einsatz in, bereits existierenden, UbiComp-Umgebungen (Anforderung 6) gut unterstützt wird.

4.2 Markup Scheme Modelle

Markup Scheme Modelle unterstützen die partielle Validierung (Anforderung 2) sehr stark. Normalerweise existieren, auch für komplexe Typen, *scheme*-Definitionen und Validierungswerkzeuge, die für die Typ-Prüfung verwendet werden können. Eine Bereichsprüfung für numerische Werte ist, bis zu einem gewissen Grad, ebenfalls möglich. – Die Unvollständigkeit und Mehrdeutigkeit (Anforderung 4) werden hingegen nicht von den Modellen unterstützt, sondern müssen auf der Applikationsebene behandelt werden.
Ob und inwieweit der fließende verteilte Aufbau eines ubiquitären Systems (Anforderung 1) berücksichtigt wird, hängt von dem einzelnen Modellierungsansatz ab. Die `override`- und `merge`-Mechanismen, die im Bezug auf diese Anforderung erforderlich sind, sind bei den Standards CC/PP und UAProf nur sehr eingeschränkt vorhanden (s. Abschnitt 3.2). Diese Schwäche wird in CSCP behoben, indem man flexiblere Mechanismen zur Verfügung stellt.
Meta-Informationen zur Qualitätskontrolle (Anforderung 3) können den Kontext-Informationen auf jeder Ebene den Markup-Daten hinzugefügt werden. Offenbar, ist dies bei den Ansätzen CSCP, CC/PP Context Extension und PPDL bis zu einer gewissen Grenze der Fall.
Ein höherer Grad an Formalität (Anforderung 5) kann durch eine allgemeine Definition des Entwurfs erreicht werden. Der Einsatz in bereits existierenden Markup- Infrastrukturen (Anforderung 6), z.B. bei Web-Services, ist eine Stärke dieser Art von Kontext-Modellen.

4.3 Grafische Modelle

Die Stärke der grafischen Modelle liegt eindeutig auf der Strukturebene. Sie werden hauptsächlich verwendet, um die Struktur von Kontext-Wissen zu beschreiben und um Code (Bauers Ansatz) oder ein ER-Modell (Henricksens Ansatz) von dem Modell abzuleiten, was die Anwendbarkeit in existierenden Umgebungen (Anforderung 6) begünstigt.
Für den fließenden verteilten Aufbau (Anforderung 1) bestehen Einschränkungen auf der Strukturebene, weil das Sharing von Modell-Fragmenten weniger effizient ist, als das Teilen von Instanz-Daten. Die partielle Validierung (Anforderung 2) ist möglich, die Unvollständigkeit und Mehrdeutigkeit (Anforderung 4) scheint bei Bauer unberücksichtigt zu sein; wird aber bei Henricksen in einer überarbeiteten Version [18] behandelt.
Die meisten Erweiterungen zu ORM, die bei Henricksen hinzugefügt wurden, sind Qualitätsmerkmale, so dass Meta-Informationen zur Qualitätsbestimmung in diesem Ansatz verwendet werden können. Die Formalität (Anforderung 5) wird bei allen grafischen Modellen nur schwach unterstützt.
Grafische Modelle werden hauptsächlich als Strukturierungshilfe für den Menschen genutzt.

4.4 Objektorientierte Modelle

Objektorientierte Kontext-Modelle sind stark auf den fließenden verteilten Aufbau (Anforderung 1) zugeschnitten. Neue Typen von Kontext-Informationen (Klassen) sowie neue oder aktualisierte Instanzen (Objekte) lassen sich flexibel handhaben. Die partielle Validierung (Anforderung 2) ist möglich. Dies geschieht normalerweise durch einen Compiler auf der Struktur-Ebene oder zur Laufzeit auf der Instanz-Ebene.
Beim TEA-Ansatz werden Meta-Informationen für die Qualitätskontrolle (Anforderung 3) eingebunden, da das cue-Konzept einen Parameter enthält, der die Qualität der Ausgabesymbole der cues beschreibt. Das ist zugleich hilfreich für die Handhabung der Unvollständigkeit und Mehrdeutigkeit (Anforderung 4).
Ein höherer Grad an Formalität (Anforderung 5) wird durch die Verwendung wohldefinierter Schnittstellen erreicht, die den Zugang

zum Inhalt des Objekts herstellen. Allerdings, ist die *Unsichtbarkeit*, als Konsequenz des „Black Box"-Prinzips, ein gewisser Nachteil im Bezug auf diese Anforderung.
Die Anwendbarkeit in existierenden UbiComp-Umgebungen (Anforderung 6) ist bei dem Modell gegeben. Es werden aber zusätzliche Anforderungen an die Ressourcen der Geräte gestellt – Anforderungen, die oft in ubiquitären Systemen nicht erfüllt werden können.

4.5 Logik-basierte Modelle

Logik-basierte Modelle können verteilt Aufgebaut sein (Anforderung 1), die partielle Validierung (Anforderung 2) ist jedoch nur mit Schwierigkeiten möglich.
Ihr Grad an Formalität ist sehr hoch, sodass Anforderung 5 gut abgedeckt wird. Aber ohne die partielle Validierung ist die Spezifikation von Kontext-Wissen in diesen Modellen äußerst fehleranfällig. Keines dieser Modelle scheint eine Qualitätsprüfung (Anforderung 3) zuzulassen – auch, wenn es einfach sein sollte, entsprechende Meta-Informationen hinzuzufügen.
Unvollständigkeit und Mehrdeutigkeit (Anforderung 4) werden ebenfalls nicht berücksichtigt, und die Anwendbarkeit (Anforderung 6) scheint hier ein Hauptproblem zu sein, da rein logische Inferenzmaschinen in UbiComp-Geräten meist nicht verfügbar sind.

4.6 Ontologie-basierte Modelle

Wegen der Ähnlichkeiten zwischen den Ontologie-basierten Modellen (Konzepte und Fakten) und den objektorientierten Modellen (Klassen und Instanzen), haben erstere ihre Stärke ebenfalls im Bereich des verteilten Aufbaus (Anforderung 1). Eine partielle Validierung (Anforderung 2) ist möglich. Es existiert eine umfassende Anzahl von Validierungswerkzeugen.
Das ASC-Modell jedoch, scheint das einzige der aufgeführten Modelle zu sein, das nicht nur eine Validierung des Datentyps erlaubt, sondern auch den vollen Dateninhalt überprüfbar macht. Dies geschieht durch spezifizieren von Bewertungsstufen für die Kontext-Information – die so genannten *scales*.

Alle Ontologie-basierten Modellierungsansätze übernehmen ihre Stärke im Bezug auf Normalisierung und Formalität von den Ontologien.
Das ASC-Modell und das CONON-Modell unterstützen von Haus aus die Mehrdeutigkeit (s. Anforderung 4), und die Verwendung von Meta-Information für die Qualitätskontrolle (s. Anforderung 3) ist möglich.
Der CoBrA-Ansatz scheint diese Anforderungen dagegen nicht zu unterstützen, kann aber leicht in diese Richtung erweitert werden.
Die Unvollständigkeit (s. Anforderung 4) wird bei allen Vorschlägen in ähnlicher Weise abgedeckt.
Die Anwendbarkeit in bestehenden Systemen (Anforderung 6) wird im ASC-Modell dadurch berücksichtigt, dass Integrationselemente von CoOL übernommen werden. Die Anwendbarkeit von CONON-Ontologien ist – ohne weitere Integrationselemente – begrenzt auf Umgebungen, die zur Wissensrepräsentation OWL-DL handhaben können. Wegen seiner Broker-zentrierten Agenten-Architektur, ist CoBrA besonders geeignet für Agenten-Systeme.

5 Tabellarische Zusammenfassung

Der Übersichtlichkeit halber fasse ich hier die, aus Abschnitt 4 resultierenden, Ergebnisse noch einmal in einer Tabelle zusammen.

Modellierungsansätze	Anforderungen					
	FVA	PV	Qua	UuM	GdF	Anw
Key-Value Modelle	-	-	-	-	-	+
Markup Modelle	+	++	-	-	+	++
Grafische Modelle	-	-	+	-	+	+
Objektorientierte Modelle	++	+	+	+	+	+
Logik-basierte Modelle	++	-	-	-	++	-
Ontologie-basierte Modelle	++	++	+	+	++	+

Nach dieser Tabelle entsprechen die Ontologie-basierten Modellierungsansätze den, in Abschnitt 2 genannten, Anforderungen am besten, was nicht heißt, dass die anderen Ansätze schlechter wären.

Je nach Gewichtung der Anforderung oder mit einem ganz anderen Anforderungskatalog, kann sich die Tabelle verändern – meist kommen ja auch noch Randbedingung wie entstehende Kosten, vorhandene Technologie, verwendetes Kontext-Wissen usw. hinzu.

Anhang

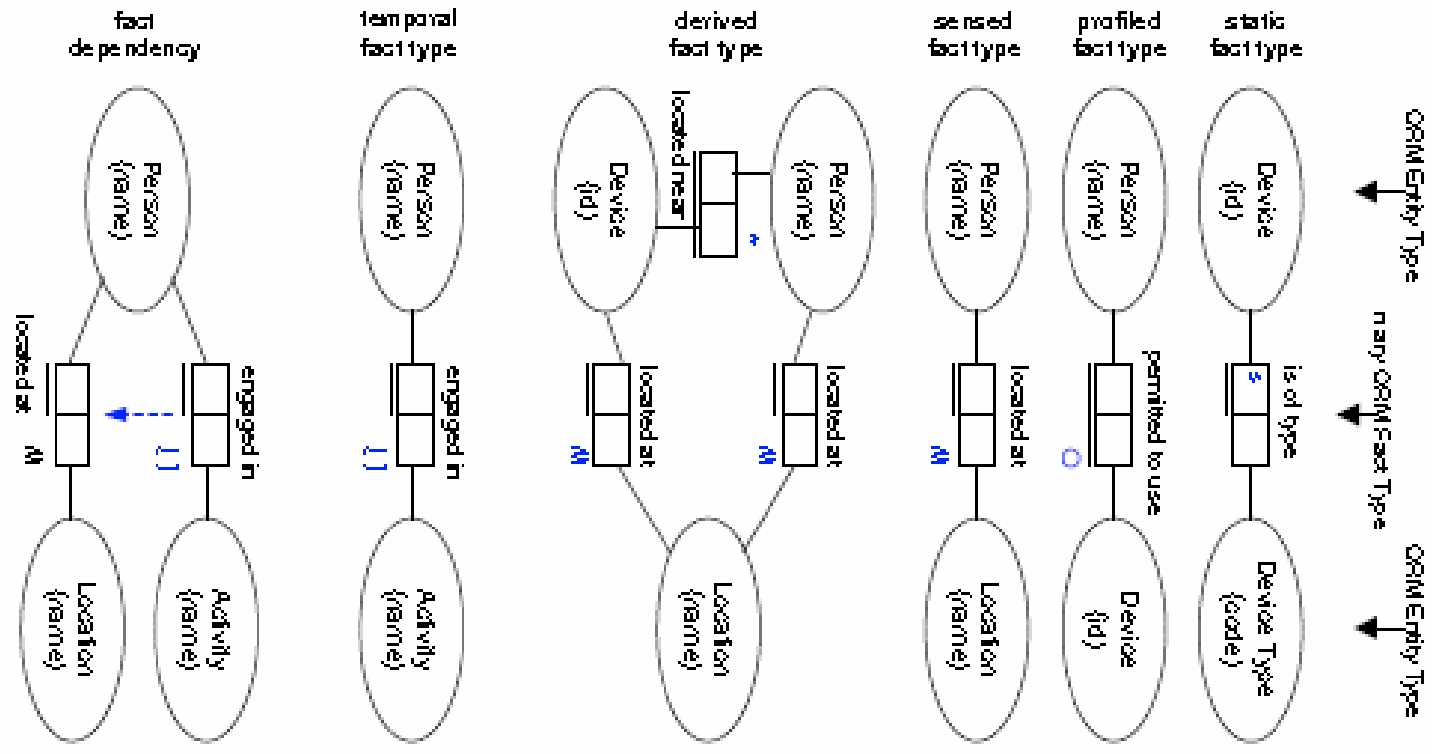


Abbildung: kontextbezogene Erweiterungen zu ORM

Referenzen

[1] AKMAN, V. UND SURAV, M. The use of situation theory in context modeling. *Computational Intelligence 13*, 3 (1997), 427–438.

[2] BACON, J., BATES, J. UND HALLS, D. Location-oriented multimedia. *IEEE Personal Communications 4*, 5 (1997).

[3] BARWISE, J. UND PERRY, J. *Situations and Attitudes*. MIT Press, 1983.

[4] BAUER, J. Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic, Mar. 2003. Diplomarbeit.

[5] CHEN, H., FININ, T. UND JOSHI, A. Using OWL in a Pervasive Computing Broker. In *Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003)* (2003).

[6] Christopoulou, E., GOUMOPOULOS, C. UND KAMEAS, A. An ontology-based context management and reasoning process for UbiComp applications. *ACM International Conference Proceeding Series; Vol. 121 Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologie*, Grenoble, France

[7] CHTCHERBINA, E. UND FRANZ, M. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)* (L'Aquila/Italy, January 2003).

[8] DE BRUIJN, J. Using Ontologies – Enabling Knowledge Sharing and Reuse on the Semantic Web. Tech. Rep. Technical Report DERI-2003-10-29, Digital Enterprise Research Institute (DERI), Austria, October 2003.

[9] DEY, A. K. (2001) Understanding and using context. Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing 5, 1.

[10] GRAY, P. UND SALBER, D. Modeling and Using Sensed Context Information in the design of Interactive Applications. In *LNCS 2254: Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001)* (Toronto/Canada, May 2001), M. R. Little and L. Nigay, Eds., Lecture Notes in Computer Science (LNCS), Springer, p. 317 ff.

[11] HALPIN, T. A. Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. Morgan Kaufman Publishers, San Francisco, 2001.

[12] Esprit project 26900: Technology for enabled awareness (tea), 1998.

[13] HELD, A., BUCHHOLZ, S. UND SCHILL, A. Modeling of context information for pervasive computing applications. In *Proceedings of SCI 2002/ISAS 2002* (2002).

[14] GHIDINI, C. UND GIUNCHIGLIA, F. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence 127*, 2 (2001), 221–259.

[15] GIUNCHIGLIA, F. Contextual reasoning. *Epistemologica - Special Issue on I Linguaggi e le Macchine 16* (1993), 345–364. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.

[16] GRUBER T. R.. Toward principles of the design of ontologies used for knowledge sharing. In *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, Padova, Italy, 1993.

[17] HENRICKSEN, K., I, J. UND RAKOTONIRAINY, A. Generating Context Management Infrastructure from High-Level Context Models. In *Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), pp. 1–6.

[18] HENRICKSEN, K. UND INDULSKA, J. Modeling and Using Imperfect Context Information. In *Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004)* (Orlando, FL, USA, March 2004), pp. 33–37

[19] HENRICKSEN, K., INDULSKA, J. UND RAKOTONIRAINY, A. Modeling context information in pervasive computing systems. In *LNCS 2414: Proceedings of 1st International Conference on Pervasive Computing* (Zurich, Switzerland, 2002), F. Mattern and M. Naghshineh, Eds., Lecture Notes in Computer Science (LNCS), Springer, pp. 167–180.

[20] INDULSKA, J., ROBINSONA, R., RAKOTONIRAINY, A. UND HENRICKSEN, K. Experiences in using cc/pp in context-aware systems. In *LNCS 2574: Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), M.-S. Chen, P. K. Chrysanthis, M. Sloman, and A. Zaslavsky, Eds., Lecture Notes in Computer Science (LNCS), Springer, pp. 247–261.

[21] MCCARTHY, J. Notes on formalizing contexts. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (San Mateo, California, 1993), R. Bajcsy, Ed., Morgan Kaufmann, pp. 555–560.

[22] MCCARTHY, J. UND BUVAČ. Formalizing context (expanded notes). In *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language* (Menlo Park, California, 1997), S. Buvač und Ł. Iwańska, Eds., American Association for Artificial Intelligence, pp. 99–135.

[23] NORMAN, D. A. Why Interfaces Don't Work. The Art of Human-Computer Interface Design. Brenda Laurel (editor). Addision-Wesley. 1992.

[24] ÖTZÜRK, P. UND AAMODT, A. Towards a model of context for case-based diagnostic problem solving. In *Context-97; Proceedings of the interdisciplinary conference on modeling and using context* (Rio de Janeiro, February 1997), pp. 198–208.

[25] SCHMIDT, A., BEIGL, M. UND GELLERSEN, H.-W. There is more to context than location. *Computers and Graphics 23*, 6 (1999), 893–901.

[26] SCHMIDT, A. UND LAERHOVEN, K. V. How to Build Smart Appliances. *IEEE Personal Communications* (August 2001).

[27] STRANG, T. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, Ludwig-Maximilians-University Munich, Oct. 2003.

[28] STRANG, T., LINNHOFF-POPIEN, C. UND FRANK, K. Applications of a Context Ontology Language. In *Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003)* (Split/Croatia, Venice/Italy, Ancona/Italy, Dubrovnik/Croatia, October 2003), D. Begusic and N. Rozic, Eds., Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Croatia, pp. 14–18.

[29] Strang, T., LINNHOFF-POPIEN, C. UND FRANK, K. CoOL: A Context Ontology Language to enable Contextual Interoperability. In *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)* (Paris/France, November 2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 236–247.

[30] STRANG, T. UND LINNHOFF-POPIEN, L. (2004) A Context Modeling Survey. in 1st International Workshop on Advanced Context Modeling, Reasoning And Management, Nottingham, *6th International Conference on Ubiquitous Computing*. UK. pp. 33-40.

[31] W3C. Composite Capabilities / Preferences Profile (CC/PP). http://www.w3.org/Mobile/CCPP.

[32] WANG, X. H., GU, T., PUNG, H. K. UND ZHANG, D. Q. ONTOLOGY Based Context Modeling and Reasoning using OWL. In *Proceedings of the 2004 Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS2004)* (San Diego, CA, USA, January 2004).

[33] WANG, X. H., ZHANG, D. Q., GU, T. UND PUNG, H. K. Ontology Based Context Modeling and Reasoning using OWL. In *Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004)* (Orlando, FL, USA, March 2004), pp. 18–22.

[34] WAPForum. User Agent Profile (UAProf). http://www.wapforum.org.
[35] WANT R., SCHILIT B. N., ADAMS N. I., GOLD R., PETERSEN K., GOLDBERG D., ELLIS J. R. UND WEISER M. (1995) The PARCTAB Ubiquitous Computing Experiment. Technical Report CSL-95-1, Xerox Palo Alto Research Center
[36] WEISER, M. (1991) The Computer for the 21st Century. *Scientific American*. 265, pp. 94-104

Kontextmanagement: Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte

Anno v. Heimburg
Tobias Zimmer

26. Januar 2006

Zusammenfassung

Wir geben einen Überblick über die Welt der Kontextmanagementsysteme, untersuchen sie im Bezug auf ihren Umgang mit Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte und geben einen kurzen Ausblick auf die erwartete Entwicklung.

1 Einleitung

Mark Weiser entwickelte 1996 die Vision der „calm technology“ [37]: Computer sollten nicht mehr Geräte sein, mit denen man sich explizit auseinandersetzt, vielmehr sollten sie Teil der Umgebung werden, unser Leben und unsere Arbeit mit den Objekten unserer Umwelt erleichtern. Seien es elektronische Museumsführer, die automatisch herausfinden, vor welchem Bild man gerade steht und entsprechende Informationen anbieten, das Mobiltelefon, das sich in einer Besprechung von selbst auf „lautlos“ schaltet, das Auto, das den Gemütszustand des Fahrers überwacht und bei Aggression beruhigende Musik und Beleuchtung auswählt oder auch der Laptop, der in der Umgebung stehende Drucker selbstständig erkennt und dem Anwender zur Verfügung stellt: All dies sind Beispiele für Systeme, die ihre Umwelt wahrnehmen und auf Änderungen reagieren.

Der Entwickler solcher kontextsensitiver Systeme steht vor einer Vielzahl von Herausforderungen. Oftmals müssen zahlreiche Sensoren und eingebettete Systeme koordiniert werden, zur Informationsbearbeitung stehen nur begrenzte, eventuell auch verteilte Ressourcen zur Verfügung, Geräte sind mobil und nicht unbedingt ständig verfügbar. Darüber hinaus sind Sensordaten nicht immer genau oder zuverlässig, veralten, der Zustand der Umwelt - der Kontext - kann mehrdeutig sein ... Kontextmanagementsysteme versuchen, den Entwickler bei seiner Arbeit zu unterstützen, indem sie ihm Funktionen zum Umgang mit diesen Problemen zur Verfügung stellen oder gar ganz von ihnen abstrahieren.

Im Folgenden diskutieren wir den Begriff des Kontexts genauer, gehen auf die Bedeutung von Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte für Kontextmanagementsysteme ein und stellen verschiedene Kontextmanagementsysteme unter diesem Aspekt vor.

2 Kontext

2.1 Motivation

Mensch-Maschine-Kommunikation ist trotz aller Fortschritte bei Benutzeroberflächen langsam, umständlich und von Missverständnissen geprägt. Menschen können sich unter Einbeziehung von Kontextinformationen untereinander wesentlich effektiver austauschen [19].

Unterhalten sich zum Beispiel zwei Personen über das Wetter, so geht es im Allgemeinen um das aktuelle Wetter zum gegenwärtigen Zeitpunkt, dessen sind sich die Gesprächspartner bewusst, ohne dass es angesprochen werden müsste. Nur, wenn Umstände einer Situation vom normalen abweichen, wenn etwas außergewöhnliches passiert, bedarf es der expliziten Erwähnung. Beobachtet man zum Beispiel auf dem Weg zur Bank einen Unfall und erzählt später davon, so ist das außergewöhnliche der Unfall; dass man auf dem Weg zur Bank war, wird eben noch erwähnt, und nicht explizit erwähnt werden muss, dass Menschen zu Banken gehen, dass Unfälle passieren, und dass man auf dem Weg zur Bank Zeuge eines Unfalls werden kann [22]. Man kann in diesem Zusammenhang beobachten, dass der implizit vorhandene Kontext auch von der Lebensgeschichte und Ausbildung der Gesprächspartner abhängt. Erzählt ein Chirurg eine Gruselgeschichte aus einer kürzlich vorgenommenen Operation, so wird er gegenüber einem anderen Chirurgen gleich in medias res gehen, während er gegenüber einem Informatiker zuerst die allgemeinen Umstände erläutern wird.

Menschen verwenden Kontext also zur Kompression bei der Kommunikation. Nur das Ungewöhnliche, das die Norm sprengende, wird explizit angesprochen, über die Begleitumstände wird kaum ein Wort verloren. Erst das macht Unterhaltungen zwischen Menschen überhaupt möglich, den gesamten Kontext vorher explizit zu etablieren, ist zeitlich gar nicht praktikabel. Darüber hinaus gehen Menschen davon aus, dass dem Gegenüber der aktuelle Kontext bekannt ist und er sich entsprechend verhält. Technische Gegenstände enttäuschen in dieser Hinsicht aber, zahlreiche in Besprechungen klingelnde Mobiltelefone sind Zeugen. Wäre einem technischen Gegenstand ebenfalls der Kontext seiner Interaktion mit dem Menschen bekannt, so ließe er sich wesentlich effizienter Einsetzen, er wäre angenehmer zu bedienen, und sein Nutzwert würde steigen.

2.2 Definition

Den Begriff „Kontext“ zu definieren erweist sich als recht schwierig, und die genaue Definition hat mit der Zeit Änderungen erfahren. Schilit u.a. [32] schreiben 1994:

> Three important aspects of context are: where you are, who you are with, and what resources are nearby. Context encompasses more than just the user's location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation; e.g., whether you are with your manager or with a co-worker.

Diese Auslegung ist breit und eng zugleich. Auf der einen Seite werden alle äußeren Umstände zu Kontext erklärt, die sich im näheren Umfeld befin-

den, auf der anderen Seite wird der Kontext beschränkt auf den aktuellen Ort und Zeitpunkt. Geht man von einem normalen Informationsaustausch aus, so sind nicht selbstverständlich alle Ressourcen in der Umgebung für diesen Informationsaustausch relevant: Unterhält man sich während einer Kaffeepause mit einem Kollegen über den Stau auf der Autobahn am Morgen, so ist der eigentliche Kontext der Unterhaltung im Sinne der Kompression nicht die Tatsache, dass im Pausenraum Kühlschrank und Kaffeemaschine zur Verfügung stehen, sondern die Autobahn, das eigene Auto und die Verkehrssituation an diesem Morgen. Sowohl örtlich als auch zeitlich weicht also der Kontext des Gesprächs vom hier und jetzt ab. Außerdem berücksichtigt die Auslegung von Schilit nicht die Geschichte einer Situation - eine Bemerkung zur Zuverlässigkeit der Kaffeemaschine macht nur dann Sinn, wenn sich alle Gesprächspartner der häufigen Ausfälle in letzter Zeit bewusst sind.

Fünf Jahre später geht Dey [19] zumindest auf das erste Problem ein:

> Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Diese Definition beschränkt Kontext zum einen nicht mehr auf die aktuell vorhandenen Personen und Ressourcen, sondern schließt alle Personen, Orte und Objekte ein, die für die Interaktion von Belang sind, und beschränkt damit auch gleichzeitig im Vergleich zu Schilit u.a. den Kontext auf die Aspekte, die tatsächlich für die Interaktion relevant sind. Das bedeutet auch, dass Kontext nicht im vorhinein bestimmt werden kann, sondern von der spezifischen Interaktion abhängt. Unbehandelt bleibt hier allerdings noch die Geschichte als Teil des Kontexts, die Dourish in seiner Definition 2003 [22] aufnimmt:

> This alternative view [of context] takes a different stance of each of the four assumptions mentioned above:
>
> – Firstly, rather than considering context to be information, it instead argues that contextuality is a relational property that holds between objects or activities. It is not simply the case that something is or is not context; rather, it may or may not be contextually relevant to some particular activity.
> – Secondly, rather than considering that context can be delineated and defined in advance, the alternative view argues that the scope of contextual features is defined dynamically.
> – Thirdly, rather than considering that context is stable, it instead argues that context is particular to each occasion of activity or action. Context is an occasioned property, relevant to particular settings, particular instances of action and particular parties to that action.
> – Fourthly, rather than taking context and content to be two separable entities, it instead argues that context arises from the activity. Context isn't just „there", but is actively produced, maintained and enacted in the course of the activity at hand.

Wie Dey sieht Dourish Kontext als von der Interaktion abhängig und damit als dynamisch an. Hinzu kommt die Einbeziehung der Geschichte, Dourish argumentiert, dass die vergangenen Aktionen der Teilnehmer an einer Interaktion wiederum Teil des Kontextes für die nächste Interaktion sind. Kürzer könnte man auch formulieren: Kontext sind alle für die aktuelle Interaktion relevanten Informationen bezüglich Personen, Orten, Objekten und Aktionen.

2.3 Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte

Kontextmanagementsysteme sammeln mit Hilfe von Sensoren Kontextdaten, abstrahieren von ihnen und stellen sie Anwendungen zur Verfügung. So könnte ein hypothetisches Kontextmanagementsystem aus GPS-Daten auf den Aufenthalt in einem Besprechungsraum und aus Umgebungsgeräuschen auf eine Unterhaltung schließen, eine entsprechende Anwendung könnte daraus auf eine Besprechung schließen und ein Mobiltelefon auf „lautlos“ stellen. Damit eine solche Deduktion zuverlässig funktionieren kann, muss das Kontextmanagementsystem auf Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte der Kontextdaten eingehen. Im folgenden werden diese Aspekte näher erläutert.

Geschichte Auf die Geschichte wurde bereits in Abschnitt 2.2 eingegangen. Menschen bewerten eine Situation nicht nur aus der aktuellen Situation heraus, sondern beziehen auch die vorangegangenen Interaktionen in den Kontext ein. Um sich so zu verhalten, wie ein Mensch es erwartet, müssen Informationssysteme ebenfalls die Geschichte in die Bewertung einer Situation mit einfließen lassen [22].

Zugehörigkeit Werden Kontextdaten aus der Umgebung durch Sensoren an einer bestimmten Position erfasst, so sind diese Daten an die Position gebunden, an der sie erfasst wurden [33]. Beispielsweise kann ein Thermometer nur die Temperatur an seinem Standpunkt messen, je größer die Distanz zum Thermometer, desto größer kann die Abweichung zwischen tatsächlicher und gemessener Temperatur werden. Die Relevanz von Kontextdaten sinkt also mit der steigenden Distanz zum Ort ihrer Erhebung. Bei der Verarbeitung von Kontextdaten muss dieses Phänomen berücksichtigt werden.

Zeitnähe Analog zur Zugehörigkeit gilt, dass Kontextdaten zu einer bestimmten Zeit erfasst worden und an diesen Zeitpunkt gebunden sind. Je größer der zeitliche Abstand zum Zeitpunkt der Erhebung, desto geringer die Relevanz. Analog zum Problem der Zugehörigkeit gilt hier, dass das Alter der Kontextdaten bei ihrer Verarbeitung einbezogen werden muss [33].

Qualität und Zuverlässigkeit Der Begriff Qualität bezieht sich auf Aspekte der Informationsgewinnung und fasst Eigenschaften wie Auflösung und Genauigkeit von Messungen zusammen. Da Messdaten nicht beliebig genau sind und sich Daten mehrerer Sensoren widersprechen oder ergänzen können, sind daraus abgeleitete Informationen ebenfalls nicht völlig sicher. Will man auf einer hohen Abstraktionsebene Entscheidungen treffen, muss die Zuverlässigkeit der abgeleiteten Kontextdaten mitgeführt werden.

3 Systeme

Im Folgenden werden verschiedene Kontextmanagementsysteme kurz vorgestellt und es wird dargelegt, inwieweit sie sich mit den erwähnten Problematiken beschäftigen und wie sie gegebenenfalls mit ihnen umgehen. Dabei wurde auf die Untersuchung von sehr frühen Kontextmanagementsystemen wie etwa CALAIS [29] und Stick-e-Notes [5] verzichtet, da diese Systeme sich noch in erster Linie mit den Problemen der Kontextgewinnung und der allgemeinen Herausforderung des Entwurfs eines Frameworks zur Kontextverarbeitung beschäftigen und sich nicht mit Themen wie Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte befassen.

3.1 Ontologiebasierte Ansätze

Den Kontextmanagementsystemen in diesem Abschnitt ist gemeinsam, dass sie Kontextinformationen mit Hilfe von Ontologien darstellen, verarbeiten und speichern.

3.1.1 GAS

Das Ziel bei der Entwicklung des Gadgetware-Architectural-Style-Systems war die Schaffung eines Frameworks zur Vereinfachung der Kommunikation und Zusammenarbeit zwischen den einzelnen Knoten in einer ubiquitären Informationssystemumgebung. Die drei Hauptelemente sind hierbei die GAS-Ontologie [15], der auf jedem GAS-Konten (eGt) laufenden GAS Ontology Manager und die eGadgetWorld[14] als zentrale Verwaltungsstelle aller Ontology Manager.

Die Aufgabe des Ontology Managers besteht darin, einen uniformen Zugriff auf die Ontologien jedes einzelnen eGts zu gewährleisten und Anfragen der Anwendungen sowohl vom lokalen als auch von entfernten eGts zu bearbeiten. Er ist insofern zwar ein zentraler Bestandteil des GAS-Systems, trägt aber nicht zur Lösung der hier betrachteten Problemstellungen bei.

Die GAS-Ontologie, geschrieben in DAML+OIL, ist in zwei Ebenen unterteilt: Die GAS Core Ontology (GAS-CO), welche das gemeinsame Vokabular des ubiquitären Informationssystems enthält, und die GAS Higher Ontology (GAS-HO), die zur Beschreibung des spezifischen Wissens der einzelnen eGts dient. Da die GAS-CO auf allen am GAS-System beteiligten Knoten laufen muss, lag das Hauptaugenmerk der Entwicklung auf der Umsetzbarkeit auch auf wenig leistungsfähigen Geräten. Entsprechend simpel ist die Ontologie ausgefallen, auf die untersuchten Fragestellungen wurde nicht explizit eingegangen. Nichtsdestotrotz ließe sich zumindest ein Teilbereich abdecken. Den einzelnen eGts und ihrem Wissen können nicht näher spezifizierte Eigenschaften zugeordnet werden; fielen darunter auch Dinge wie Ort und Zeit der Datenerfassung, könnte man zumindest diese Informationen vorhalten. Schwieriger würde es bei Zuverlässigkeit, Qualität und Geschichte, da der GAS Ontology Manager, der für die Kommunikation zwischen den eGts verantwortlich ist, nicht darauf vorbereitet ist.

Entscheidend für die Betrachtungen hier ist jedoch, dass diese Aspekte offenbar bei der Entwicklung von GAS keine Rolle gespielt haben; es ging darum, ein System zu schaffen, das bezüglich der Leistungsfähigkeit der beteiligten Geräte möglichst anspruchslos ist.

3.1.2 CoBrA

Bei der Context Broker Architecture (CoBra) geht es um die Entwicklung eines Frameworks, welches seinerseits die Entwicklung von ubiquitären Informationssystem softwareseitig möglichst effektiv unterstützt. Im Gegensatz zu GAS standen geringe Anforderungen an die Hardware eher im Hintergrund, vielmehr wurde großer Wert auf eine einfache Anwendungsentwicklung gelegt. CoBrA soll außerdem Konzepte aus dem Semantic Web in die Welt der ubiquitären Informationssysteme tragen [11, 12].

Im Zentrum CoBrAs steht der Context Broker, der Kontextinformationen aus allen möglichen Quellen wie dem Semantic Web, mobilen und stationären Sensoren und anderen Agenten und Geräten sammelt und Anwendungen zur Verfügung stellt. Dabei macht CoBrA auf der Architekturebene keine Unterschiede zwischen Kontextquellen und -senken, im Prinzip kann jedes angebundene System Anfragen stellen und Kontext auf verschiedenen Abstraktionsebenen liefern. Über die Einbindung von Datenbanken erlaubt CoBrA die Speicherung und den späteren Zugriff auf Kontextdaten, somit werden auch Applikationen, die sich für die Kontextgeschichte interessieren, unterstützt. Die Darstellung der Kontextdaten erfolgt mit den in OWL definierten Ontologien COBRA-ONT bzw. SOUPA.

Die Grundidee bei der Entwicklung einer Ontologie für CoBrA war die Wiederverwendung von bereits existierenden Technologien des Semantic Web. Bereits in der ersten Version, COBRA-ONT [12], unterstützt die Ontologie die Zuordnung von Orten zu Kontextdaten mithilfe der `locatedIn`-Eigenschaft. Der Nachfolger SOUPA [13] erlaubt darüber hinaus die Zuordnung von Zeitpunkten unter Verwendung von DAML-Time [27] und ermöglicht Aussagen der Art `at-time(locatedIn(harry,room201), clock_time(12:57))`. Zeitnähe und Zugehörigkeit lassen sich so zumindest durch die Anwendung berücksichtigen, explizit unterstützt werden sie aber nicht.

Zuverlässigkeit und Qualität werden nach dem jetzigen Stand ebenfalls nicht durch CoBrA oder SOUPA unterstützt, allerdings hat sich die gleiche Forschungsgruppe mit der Zuverlässigkeit im Semantic Web auseinandergesetzt und BayesOWL zur OWL-Modellierung für Unsicherheiten entwickelt [21]. Es ist zumindest denkbar, diese Ergebnisse auch in SOUPA einzuführen.

3.1.3 SOCAM

SOCAM steht für Service-Oriented Context Aware Middleware. Sie verfolgt wie CoBrA das Ziel, die Entwicklung von kontextsensitiven Anwendungen zu unterstützen, und wurde unter anderem von CoBrA und Context Toolkit, Cooltown, Context Fabric und Gaia inspiriert, auf die später noch eingegangen wird [41, 35]. Die Architektur von SOCAM ähnelt der CoBrAs: Ein Context Interpreter im Mittelpunkt sammelt Kontextdaten und verarbeitet Anfragen von Anwendungen, auch ihm steht eine Datenbank zur Speicherung vergangener Kontextdaten zur Verfügung. Dabei unterscheidet SOCAM die drei Ebenen Sensoren/Aktuatoren (Datenquellen), Context Interpreter (Abstraktion der Kontextdaten) und Anwendungen (Kontextdatensenken). Wie bei CoBrA werden Kontextdaten jedes Abstraktionsniveaus in Form einer in OWL definierten Ontologie dargestellt.

Die SOCAM-Ontologie [35, 40] ermöglicht zu einem Datum die Zuordnung

von vier Qualitätsparametern: Genauigkeit – die Messgenauigkeit eines Sensors, Auflösung – kleinste unterscheidbare Einheit, Sicherheit – Zuverlässigkeit von abgeleiteten Daten und Frische – Messzeitpunkt und mittlere Lebenszeit einer Messung. Diese Parameter werden bei der Weiterverarbeitung berücksichtigt und auf allen Abstraktionsebenen der Anwendung zur Verfügung gestellt. Das kommt den hier geforderten Kriterien sehr nahe, einzig die Frage der örtlichen Zugehörigkeit eines Datums wird von SOCAM nicht abgebildet.

3.1.4 Gaia

Der Context Service von Gaia [31] stellte den Versuch dar, ein Kontextmanagementsystem auf Basis von Prädikatenlogik zu entwerfen, um komplexe Schlussfolgerungen ziehen zu können. Kontextdaten sowie zugehörige Metainformationen werden dabei in OWL als Prädikate dargestellt, die mit den üblichen Operatoren der Prädikatenlogik verknüpft werden können. Anwendungen können entweder Anfragen an dieses System stellen oder sich bei Änderungen gewisser Kontextdaten benachrichtigen lassen [30]. Auch Gaia ist in drei Ebenen aufgebaut: Kontextquellen liefern Daten an „Context Synthesizer“, welche von den Rohdaten abstrahieren und Anfragen der Anwendungen in der obersten Ebene bearbeiten. Ein Kontextdatenspeicher erlaubt die Einbeziehung der Geschichte in die Anfragen.

Es stellte sich heraus, dass die Prädikatenlogik alleine nicht ausreicht, um ungenaue oder unzuverlässige Kontextdaten verarbeiten zu können. Gaia wurde erweitert, um es Anwendungen zu ermöglichen, Anfragen und Regeln in probabilistischer Logik zu stellen, darüber hinaus können Anwendungen Kontextdaten durch Bayes'sche Netzwerke verarbeiten lassen [1]. Die sich daraus ergebenden Wahrscheinlichkeitswerte werden wiederum über Prädikate mit den Kontextdaten verknüpft. Gaia wird aktiv weiterentwickelt, wenn der Fokus im Moment auch bei Sicherheit und Privatsphäre liegt.

Auf unsere fünf Kriterien bezogen unterstützt Gaia die Verwaltung von Geschichte, Zuverlässigkeit und Qualität der Kontextdaten, allerdings weder Zugehörigkeit noch Zeitnähe.

3.2 Objektbasierte Ansätze

Die folgenden Systeme verwalten Kontext in Form von Objekten. Viele dieser Frameworks sind im Unterschied zu den ontologiebasierten Ansätzen an sich nicht lauffähig, sondern unterstützen den Anwendungsentwickler beim Aufbau eines eigenen Kontextmanagementsystems.

3.2.1 Context Toolkit / Context Fabric

Das Context Toolkit [20] ist eines der frühen Kontextmanagementsysteme und ist nach dem Vorbild klassischer GUI-Toolkits aufgebaut. Widgets abstrahieren Sensoren und Aktuatoren, Interpretierer erhöhen das Abstraktionsniveau der Kontextdaten, und Aggregatoren fassen Kontextdaten zusammen und stellen sie als Einheit der Anwendung zur Verfügung. Dem frühen Entstehungsdatum entsprechend befasst sich das Toolkit vor allem mit dem generellen Problem der Entwicklung eines Kontext-Frameworks. Das prinzipielle Problem der Zuverlässigkeit und Qualität wurde auch beim Context Toolkit erkannt, die Lösung

aber den Anwendungen überlassen.

Das Context Fabric [28] möchte die Weiterentwicklung des Context Toolkits sein. Der grundsätzliche Aufbau ist entsprechend ähnlich, allerdings wird die Funktionalität in Form von Diensten angeboten und ist somit anders als beim Context Toolkit nicht mehr Teil der Anwendung selbst. Darüber hinaus bietet das Context Toolkit den Anwendungen Daten zur Qualität der Sensoren und zur Zuverlässigkeit des abstrahierten Kontexts an, sie werden als Eigenschaften der jeweiligen Objekte modelliert. Die Kontextgeschichte wird vom Context Toolkit nicht vorgehalten, ebensowenig Bewertungen der Zeitnähe und Zugehörigkeit.

3.2.2 PACE

PACE [25] geht von einem 5-Schichten-Modell bei kontextverarbeitenden Applikationen aus: Auf Schicht null sind die Sensoren und Aktuatoren, auf Schicht eins die Kontextverarbeitung, auf Schicht zwei die Datenspeicher, auf Schicht drei die Entscheidungsfindung auf Basis der Kontextdaten und auf Schicht vier schließlich die Anwendung selbst. PACE stellt dabei Funktionen zur Verfügung, die Anwendungsentwicklern die Realisierung der Schichten 1-3 erheblich vereinfachen sollen. PACE ist dabei kein alleine lauffähiges System, sondern eine Sammlung von Funktionen zur Implementierung einer kontextsensitiven Anwendung.

PACE modelliert Kontext als Objekte und erlaubt das Versehen von Kontextdaten mit beliebigen Zusatzinformationen, die von entsprechend ausgelegten Diensten höhere Schichten auch verwendet werden können [26]. Über die Datenspeicher der Schicht zwei kann zudem die Kontextgeschichte vorgehalten werden. Inwieweit ein PACE-System also Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte unterstützt, liegt in den Händen des Anwendungsentwicklers.

3.2.3 JCAF

JCAFs Grundvorstellung ist die eines ereignisbasierten, sicheren, verteilten Systems, in dem verschiedene Kontextdienste auf diversen Knoten zusammenarbeiten. Die Context Clients als höchste Ebene bedienen sich der Dienste der Context Server (mittlere Ebene), die die Daten der Sensoren und Aktuatoren der untersten Ebene verarbeite n und von ihnen abstrahieren [2]. Die Kommunikation zwischen Clients und Servern sowie zwischen den Servern erfolgt via Java RMI. Kontextdaten und -Ereignisse sind als Java-Objekte dargestellt. Dabei stellt JCAF nur Funktionsbibliotheken und Java-Erweiterungen zur Verfügung, es will bei der Entwicklung eines Kontextmanagementsystems helfen, nicht eines sein. In dieser Hinsicht ähnelt es PACE.

Die Modellierung der Kontextdaten als Java-Objekte erlaubt bei entsprechender Programmierung der Services und Clients das Vorhalten der notwendigen Daten zur Verarbeitung von Qualität, Zuverlässigkeit, Zeitnähe und Zugehörigkeit als Objekteigenschaften. Anders als PACE unterstützt JCAF Datenspeicher nicht explizit, sie ließen sich aber als Context Services im Rahmen von JCAF realisieren. Wie bei PACE gilt auch hier, dass der Grad, indem Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte unterstützt werden, vom Anwendungsentwickler abhängt.

3.2.4 RCSM

Bei RCSM werden Kontextdaten und Kontextverarbeitungsregeln mithilfe einer Interface Definition Language als Objekte [39] modelliert. Ein Object Request Broker stellt das Kontextwissen und die Verarbeitungsmöglichkeiten des jeweiligen Gerätes anderen Geräten zur Verfügung, ein sogenannter Situation-Aware-(SA)-Processor bearbeitet Anfragen anhand der verfügbaren Kontextdaten und -Regeln. Die Fähigkeiten von RCSM hängen also maßgeblich von der Mächtigkeit der IDL und des SA-Processors ab.

Der SA-Processor speichert vergangene Kontextdaten und ausgeführte Aktionen und kann bei der Beantwortung von Anfragen so die Geschichte berücksichtigen. Bezüglich der anderen interessanten Parameter erlaubt die IDL zwar prinzipiell die Verbindung von Kontextdaten mit Angaben zu Qualität, Zuverlässigkeit und Alter, es ist aber nicht möglich, diese Parameter bei der Kontextverarbeitung über mehr als eine Abstraktionsebene hinweg zu verwenden. In [38] wird die Erweiterung der IDL und für den SA-Processor Funktionalitäten zum Umgang mit ungenauen Kontextdaten und -Regeln angekündigt, die Entwicklung von RCSM scheint aber seit Ende 2003 zu ruhen.

3.3 Andere Ansätze

In diesem letzten Abschnitt werden Systeme vorgestellt, die eigene Wege bei der Kontextdarstellung und -Verarbeitung gehen.

3.3.1 Cooltown

HP's Cooltown hatte als Entwicklungsziel, Objekten der wirklichen Welt URLs zuzuordnen [18] und so den Menschen bei seiner Interaktion mit diesen Objekten unterstützen zu können [3]. Dies wird erreicht, indem ein Endbenutzergerät seinen Standort zum Beispiel anhand von Funketiketten oder GPS-Daten ausfindig macht, damit die URL der in der Nähe befindlichen Ressourcen durch einen Server bestimmen lässt und dann die Ressourcen anspricht. Der Fokus von Cooltown liegt bei der zuverlässigen Bestimmung des aktuellen Standorts und der darauffolgenden Zuordnung von Ressourcen, die hier relevanten Aspekte haben bei der Entwicklung keine Rolle gespielt.

3.3.2 MUSE

Ziel bei MUSE (Multi-Use Sensor Environment) war die Entwicklung einer „Black Box“, der auf der einen Seite Sensordaten zur Verfügung stehen und die auf der anderen Seite Anfragen nach Zuständen mit Aussagen bezüglich ihrer Wahrscheinlichkeit beantwortet. Die Zuverlässigkeit der abgeleiteten Kontextdaten war also zentraler Aspekt bei der Entwicklung. Die Black Box ist als Bayes'sches Netzwerk realisiert, das auf die Erkennung der relevanten Umgebungszustände trainiert wird.

Bis auf die Zuverlässigkeit der abgeleiteten Kontextdaten werden die hier behandelten Problematiken nicht explizit berücksichtigt, allerdings fließen Qualität und Zuverlässigkeit der Sensoren beim Lernprozess in das Bayes'sche Netzwerk ein [6]. Es ist darüber hinaus vielleicht möglich, MUSE mit Daten zu trainieren, die die Behandlung von Zeitnähe und Zugehörigkeit erlauben. Was die Frage der Geschichte angeht, so hat MUSE zwar keinen expliziten Speicher für

System	Qualität	Zuverlässigkt.	Zeitnähe	Zugehörigkt.	Geschichte
Gas	-	-	-	-	-
CoBra	-	-	o	o	x
SOCAM	x	x	x	-	x
Gaia	x	x	-	-	x
Context Toolkit	-	-	-	-	-
Context Fabric	x	x	-	-	-
PACE	x	x	x	x	x
JCAF	x	x	x	x	-
RCSM	-	-	-	-	x
Cooltown	-	-	-	-	-
MUSE	x	x	-	-	-
SOLAR	x	x	x	x	o

Tabelle 1: Übersicht der Kontextmanagementsysteme, -: Nicht unterstützt, x: Unterstützt, o: Angekündigt/teilweise

alte Daten, ist allerdings ein lernendes System: Die vergangenen Kontextdaten aus den Lernphasen sind implizit im Ergebnis einer Anfrage berücksichtigt.

3.3.3 SOLAR

SOLAR [9] ist die Implementierung eines sogenannten Context Fusion Networks [10]. Ziel ist die Erleichterung der Anwendungsentwicklung durch eine gemeinsame Schnittstelle zu den Kontextdaten und Geräten, darüber hinaus soll von der dynamischen und weit verteilten Natur eines großen ubiquitären Informationssystems abstrahiert werden. Bei der Entwicklung stand die Berücksichtigung von Skalierbarkeit, Mobilität und Informationsqualität im Vordergrund.

Sensoren liefern Kontextdaten in Form von Ereignissen. Jedes Ereignis besteht aus einer beliebigen Menge von Attribut-Wert-Paaren, neben den eigentlichen Kontextdaten können so auch Qualitätsmerkmale dargestellt werden. Anwendungen spezifizieren eine Reihe an Filterketten, in denen Fusionsoperatoren schrittweise von den Sensordaten abstrahieren und dabei die Datenmenge reduzieren. Ziel ist es hierbei, möglichst viel Kontextbearbeitung im Netzwerk selbst durchzuführen und so die Last möglichst gut auf die Knoten zu verteilen. Stehen für die einzelnen Sensoren Daten bezüglich deren Qualität und Ort zur Verfügung, so können Anwendungen selbst bestimmen, wie SOLAR mit Qualität, Zuverlässigkeit, Zeitnähe und Zugehörigkeit umgehen soll, geeignete Fusionsoperatoren stehen zur Verfügung. Den Umgang mit vergangenem Kontextdaten beherrscht SOLAR noch nicht, die Entwicklung entsprechender Ergänzungen wird aber in [8] in Aussicht gestellt.

4 Fazit und Ausblick

Der Umgang mit Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte hat für Kontextmanagementsysteme zunehmend an Bedeutung gewonnen. Die frühen Entwicklungen beschäftigten sich hauptsächlich mit dem Problem der Kontextakquisition und dem Entwurf von Toolkits für kontextsensitive Anwendungen und überließen weitergehende Maßnahmen, sofern überhaupt vorgesehen, den Anwendungsentwicklern. Man war sich der Probleme zum Teil zwar bereits Ende der neunziger Jahre bewusst, erste Kontextmanagementsysteme, die den Anwendungsentwickler aktiv bei deren Lösung unterstützen soll-

ten, wurden aber nicht vor 2003 veröffentlicht. Am wenigsten wird der Umgang mit der örtlichen Zugehörigkeit von Kontextdaten betrachtet, was daran liegen mag, dass diese Problematik nur für Systeme größerer räumlicher Ausdehnung von Bedeutung ist. SOLAR, das explizit entwickelt wurde, um mit großen ubiquitären Informationssystemen umgehen zu können, unterstützt beispielsweise die räumliche Zuordnung von Kontexten.

Insgesamt lassen sich zwei Klassen an Systemen ausmachen: Systeme wie PACE und JCAF unterstützen den Entwickler bei der Implementierung von Anwendungen und stellen dabei in verschiedenem Maße Werkzeuge zum Umgang mit den hier betrachteten Eigenschaften, deren Verarbeitung bleibt aber der Anwendung überlassen. Bei dieser Klasse ist die Frage, inwieweit der Entwickler unterstützt wird, die entscheidende. Von den hier betrachteten Systemen geht PACE mit seinen beliebigen Zusatzeigenschaften der Kontextdaten und seiner expliziten Unterstützung der Kontextgeschichte am weitesten.

Im Gegensatz dazu spezifizieren Anwendungen bei System wie SOLAR, CoBrA und Gaia Regeln, nach denen das Kontextmanagementsystems die Kontextdaten für die Anwendung aufbereiten soll, dementsprechend müssen diese Systeme mit Qualität, Zuverlässigkeit, Zeitnähe, Zugehörigkeit und Geschichte umgehen können. Am weitesten sind hierbei SOCAM und SOLAR fortgeschritten, SOCAM fehlt einzig die Berücksichtigung der örtlichen Zugehörigkeit, SOLAR soll noch um den Umgang mit Geschichte erweitert werden.

Von den hier vorgestellten Systemen werden SOCAM, SOLAR und Gaia erkennbar aktiv weiterentwickelt, inwieweit die CoBrA-Forschungsgruppe ihre Ergebnisse aus dem Semantic Web weiterhin in CoBrA integrieren wird, ist unklar. Es fällt auf, dass die Aktivität eindeutig in Richtung ontologiebasierter Systeme mit Bayes'schen Erweiterungen zum Umgang mit unsicheren Kontexten geht. Qualität, Zuverlässigkeit, und Geschichte werden bei allen neueren Ansätzen berücksichtigt, auf Zeitnähe wird ebenfalls zunehmend Wert gelegt. Die Frage der örtlichen Zugehörigkeit wird an Bedeutung gewinnen, je mehr Kontextmanagementsysteme in räumlich großen Umgebungen zum Einsatz kommen. Von SOLAR darf man erwarten, als erstes Kontextmanagementsystems mit allen fünf Merkmalen umgehen werden zu können.

Literatur

[1] Anand Ranganathan, Jalal Al-Muhtadi und Roy H. Campbell: *Reasoning about Uncertain Contexts in Pervasive Computing Environments.* IEEE Pervasive Computing, 3(2):62–70, April-Juni 2004.

[2] Bardram, Jakob E.: *The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications.* In: *Pervasive*, Seiten 98–115, 2005.

[3] Barton, John und Tim Kindberg: *The Cooltown User Experience.* Technischer Bericht HPL-2001-22, Hewlett Packard Laboratories, Februar 2001.

[4] Biegel, Gregory und Vinny Cahill: *A Framework for Developing Mobile, Context-aware Applications.* In: *Proceedings of the* 2^{nd} *IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, Seiten 361–365. IEEE Computer Society, März 2004.

[5] Brown, Peter J.: *The Stick-e Document: A Framework for Creating Context-aware Applications.* Electronic Publishing – Origination, Dissemination, and Design, 8(2/3):259–272, Juni/September 1995.

[6] Castro, Paul, Patrick Chiu, Ted Kremenek und Richard R. Muntz: *A Probabilistic Room Location Service for Wireless Networked Environments.* In: *Ubicomp*, Seiten 18–34, 2001.

[7] Castro, Paul und Richard R. Muntz: *Managing context data for smart spaces.* IEEE Personal Communications, 7(5):44–46, Oktober 2000.

[8] Chen, Guanling: *Solar: Building a Context Fusion Network for Pervasive Computing.* Doktorarbeit, Dartmouth College, Hanover, NH, USA, August 04 2004.

[9] Chen, Guanling und David Kotz: *Supporting Adaptive Ubiquitous Applications with the SOLAR System.* Technischer Bericht TR2001-397, Dartmouth College, Computer Science, Hanover, NH, Mai 2001.

[10] Chen, Guanling, Ming Li und David Kotz: *Design and Implementation of a Large-Scale Context Fusion Network.* In: *MobiQuitous*, Seiten 246–255, 2004.

[11] Chen, Harry, Tim Finin und Anupam Joshi: *An Intelligent Broker for Context-Aware Systems.* In: *Adjunct Proceedings of Ubicomp 2003*, Seiten 183–184, Oktober 2003.

[12] Chen, Harry, Tim Finin und Anupam Joshi: *An Ontology for Context-Aware Pervasive Computing Environments*, Mai 2003.

[13] Chen, Harry, Tim Finin und Anupam Joshi: *The SOUPA Ontology for Pervasive Computing.* In: *Ontologies for Agents: Theory and Experiences*, Whitestein Series in Software Agent Technologies. Springer, Juli 2005.

[14] Christopoulou, Eleni und Achilles Kameas: *Using Ontologies to Address Key Issues in Ubiquitous Computing Systems.* In: *EUSAI*, Seiten 13–24, 2004.

[15] Christopoulou, Eleni und Achilles Kameas: *GAS Ontology: An ontology for collaboration among ubiquitous computing devices.* International Journal of Human-Computer Studies, 62(5):664–685, 2005.

[16] Clarke, Siobhán und Cormac Driver: *Context-Aware Trails.* IEEE Computer, 37(8):97–99, August 2004.

[17] Cohen, Norman H., Apratim Purakayastha, Luke Wong und Danny L. Yeh: *iQueue: A Pervasive Data Composition Framework.* In: *Mobile Data Management*, Seiten 146–153, 2002.

[18] Debaty, Philippe und Debbie Caswell: *Uniform Web Presence Architecture for People, Places, and Things.* Technischer Bericht HPL-2000-67, Hewlett Packard Laboratories, Juni 2000.

[19] Dey, Anind K.: *Understanding and Using Context.* Personal Ubiquitous Computing, 5(1):4–7, 2001.

[20] Dey, Anind K., Gregory D. Abowd und Daniel Salber: *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.* Human-Computer Interaction, 16(2/4):97–166, 2001.

[21] Ding, Zhongli, Yun Peng und Rong Pan: *BayesOWL: Uncertainty Modeling in Semantic Web Ontologies.* In: *Soft Computing in Ontologies and Semantic Web*, Studies in Fuzziness and Soft Computing. Springer, Oktober 2005.

[22] Dourish, Paul: *What we talk about when we talk about context.* Personal and Ubiquitous Computing, 8(1):19–30, 2004.

[23] Garlan, David, Daniel P. Siewiorek, Asim Smailagic und Peter Steenkiste: *Project Aura: Toward Distraction-Free Pervasive Computing*, Mai 24 2002.

[24] Garlan, David und João Pedro Sousa: *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments.* In: *WICSA*, Seiten 29–43, 2002.

[25] Henricksen, Karen, Jadwiga Indulska, Ted McFadden und Sasitharan Balasubramaniam: *Middleware for Distributed Context-Aware Systems.* In: *OTM Conferences (1)*, Seiten 846–863, 2005.

[26] Henricksen, Karen, Jadwiga Indulska und Andry Rakotonirainy: *Modeling Context Information in Pervasive Computing Systems.* Januar 01 2002.

[27] Hobbs, Jerry A.: *A DAML Ontology of Time.* http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt, November 2002.

[28] Hong, Jason I. und James A. Landay: *An Infrastructure Approach to Context-Aware Computing.* Human-Computer Interaction, 16(2/4):287–303, 2001.

[29] Nelson, Giles J.: *Context-aware and Location Systems*. Doktorarbeit, University of Cambridge, Cambridge, UK, Januar 1998.

[30] Ranganathan, Anand und Roy H. Campbell: *An infrastructure for context-awareness based on first order logic*. Personal and Ubiquitous Computing, 7(6):353–364, 2003.

[31] Román, Manuel, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell und Klara Nahrstedt: *Gaia: A Middleware Infrastructure to Enable Active Spaces*. IEEE Pervasive Computing, Seiten 74–83, Oktober-Dezember 2002.

[32] Schilit, Bill, Norman Adams und Roy Want: *Context-Aware Computing Applications*. In: *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.

[33] Schmidt, Albrecht: *Ubiquitous Computing - Computing in Context*. Doktorarbeit, Lancaster University, Lancaster, UK, November 2002.

[34] Schmidt, Albrecht, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven und Walter Van de Velde: *Advanced Interaction in Context*. In: *HUC*, Seiten 89–101, 1999.

[35] Wang, Xiao Hang, Da Qing Zhang, Tao Gu und Hung Keng Pung: *Ontology Based Context Modeling and Reasoning using OWL*. In: *PerCom Workshops*, Seiten 18–22, 2004.

[36] Weiser, Mark: *The Computer for the 21st Century*. Scientific American, 265(3):94–104, 1991.

[37] Weiser, Mark und John Brown: *The Coming Age of Calm Technology*. In: Denning, Peter J. und Robert M. Metcalfe (Herausgeber): *Beyond Calculation: The Next Fifty Years of Computing*. Copernicus, 1997.

[38] Yau, Stephen S., Dazhi Huang, Haishan Gong und Siddharth Seth: *Development and Runtime Support for Situation-Aware Application Software in Ubiquitous Computing Environments*. In: *COMPSAC*, Seiten 452–457, 2004.

[39] Yau, Stephen S., Yu Wang und Fariaz Karim: *Development of Situation-Aware Application Software for Ubiquitous Computing Environment*. In: *COMPSAC*, Seiten 233–238, 2002.

[40] Zhang, Da Qing, Hung Keng Pung und Tao Gu: *A Bayesian Approach For Dealing With Uncertain Contexts*, April 16 2004.

[41] Zhang, Da Qing, Hung Keng Pung, Tao Gu und Xiao Hang Wang: *A Middleware for Building Context-Aware Mobile Services*, April 13 2004.

Algorithmen für die Kontextgewinnung

Seminararbeit
Ubiquitäre Informationstechnologien
WS 05/06

Tobias Zimmer
Universität Karlsruhe
Institut für Telematik

Jochen Ring
email: ringos@web.de
Matr.Nr.: 1056180

Inhaltsverzeichnis

1. Einführung

Um eine intelligente Umwelt verwirklichen zu können, müssen ubiquitäre Informationssysteme in diese Umwelt integriert werden. Dazu zählen allgegenwärtige Rechen-, Kommunikations- und Sensoreinheiten, die die Aufgabe haben einzelne Kontexte erkennen zu können. Mit Erkennung ist dabei ein kognitiver Prozess gemeint, wobei zunächst einzelne Ereignisse eines bestimmten Kontextes wahrgenommen und diese dann verschiedenen Klassen zugeordnet werden. Dies geschieht auf Basis von Vorschriften, denen der Prozess der Erkennung und Klassenzuordnung oder auch Klassifikation unterliegt. Solche Vorschriften, die zu diesem Zweck eingesetzt werden, nennt man Kontextalgorithmen. Es gibt eine Vielzahl von Ansätzen zur Verwirklichung von Kontextalgorithmen, die auf unterschiedliche Bereiche der Kontexterkennung und Klassifikation zugeschnitten sind. Allgemein ist es nicht möglich einen Algorithmus zu nennen, der universell für die Aufgabe der Kontextklassifikation eingesetzt werden kann.
Zur Lösung der Aufgabe der Kontextklassifikation müssen Sensordaten zunächst zu einzelnen Klassen zusammengefasst werden können, wofür Verfahren der statistischen Kontextmodellierung zum Zuge kommen können, von denen ein Ausschnitt in Abschnitt 2 erläutert wird. Es können zu diesem Zweck aber auch Pattern-Matching Algorithmen eingesetzt werden, von denen einige repräsentativ in Abschnitt 3 vorgestellt werden. Dabei setzt sich die Erkennung von Mustern aus einzelnen Entscheidungsprozessen zusammen. Da aufgrund verrauschter oder uneindeutiger Sensordaten ein gewisser Grad an Unsicherheit herrscht, kommen die in Abschnitt 4 vorgestellt Entscheidungsmechanismen bei nicht eindeutig identifizierbaren Kontexten zum tragen.

2. Statistische Kontextmodellierung

Ziel der statistischen Kontextmodellierung ist es, durch Sensoren wahrgenommene Merkmale der Umgebung gewissen Kontextklassen zuzuordnen. Dazu ist es nötig, den Merkmalsraum durch die Bildung von Dichtefunktionen in die verschiedenen Klassen zu zerlegen. Eine Dichtefunktion ist eine Funktion, die Ereignissen Wahrscheinlichkeiten zuordnet. Sie wird im diskreten Fall auch Zähldichte genannt [4]. Der Mittelwert wird dazu benutzt, den Durchschnittswert vorliegender Werte anzugeben. Er wird meist mit μ bezeichnet [5]. Die Kovarianzmatrix ist eine Matrix, die alle paarweisen Kovarianzen eines Zufallsvektors enthält. Die Kovarianz stellt den Zusammenhang zwischen zwei statistischen Merkmalen X und Y dar. Fällt die Kovarianz positiv aus, bedeutet dies, dass X und Y einen gleichsinnigen linearen Zusammenhang besitzen. D.h. wenn X hohe Werte hat, dann hat auch Y hohe Werte und wenn X niedrige Werte hat, dann hat auch Y niedrige Werte. Eine negative Kovarianz drückt aus, ob X und Y gegensinnigen linearen Zusammenhang aufweisen, ob also niedrige Werte von X mit hohen Werten von Y einhergehen und hohe Werte von X mit niedrigen Werten von Y [6].
Den Mittelwert oder auch Erwartungswert μ einer Zufallsvariablen $X = (x_1, ..., x_n)$ mit den Wahrscheinlichkeiten p_i für das Auftreten des Ereignisses x_i berechnet man mit:

$$\mu = \sum_{i=1}^{n} x_i \cdot p_i \quad .$$

Die Kovarianz zweier Zufallsvariablen X und Y mit den Erwartungswerten μ_X und μ_Y der jeweiligen Zufallsvariablen wird definiert als:

$$\sigma(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_X) \cdot (y_i - \mu_Y) \quad .$$

Wie man der Formel für die Kovarianzen leicht entnehmen kann, gilt σ(X, Y) = σ(Y, X). Außerdem ist die Kovarianz σ(X, X) gleich der Varianz der Zufallsvariablen X, die die Abweichung von X von ihrem Erwartungswert anzeigt.
Die Kovarianzmatrix Σ zweier Zufallsvariablen X und Y hat dann folgende Gestalt:

$$\Sigma = \begin{pmatrix} \sigma(X, X) \sigma(X, Y) \\ \sigma(Y, X) \sigma(Y, Y) \end{pmatrix} \quad ,$$

wobei diese Matrix auf der Hauptdiagonalen die Varianzen enthält. Darüber hinaus ist sie, aufgrund der oben genannten Eigenschaft (σ(X, Y) = σ(Y, X)) symmetrisch [6]. Hierbei lassen sich grundlegend zwei Verfahrenstypen unterscheiden:
Überwachte und unüberwachte Verfahren. Bei überwachten Verfahren lässt sich die Lage der einzelnen Klassen durch Schätzung von Dichtefunktionen aus einzelnen Stichproben oder der Messung der Abstände einzelner Merkmale abschätzen und im Voraus bestimmen. Bei unüberwachten Verfahren kommt es zu einer offenen Klassenzuordnung, wobei nicht bekannt ist, ob Klassen überhaupt existieren [1]. In diesem Abschnitt wird lediglich auf die überwachten Verfahren eingegangen, die sich wiederum in parametrische und nicht-parametrische Verfahren aufteilen. Zu den parametrischen gehören Klassifikatoren wie Bayes und Gauß, bei denen eine Dichtefunktion geschätzt wird. Bei nicht-parametrischen Verfahren werden keine Dichtefunktionen geschätzt. Zu ihnen zählen Methoden wie Parzen Window und K Nearest Neighbors.

2.1 Bayes

Der Bayes Klassifikator benutzt zur Abschätzung einer Dichtefunktion das Bayes Theorem. Ziel dieses Klassifikationsverfahrens ist es abzuschätzen, zu welcher Klasse ein Sensordatum am wahrscheinlichsten gehört [3]. Bei der Bayes Klassifikation wird davon ausgegangen, dass jeder Kontext mit normaler Dichte durch den Mittelwert μ und die Kovarianzmatrix Σ charakterisiert werden kann [2].
Voraussetzung für dieses Klassifikationsverfahren ist, dass die Wahrscheinlichkeit *P(context)* der Kontextklasse *context* a priori bekannt ist oder geschätzt werden kann. Des weiteren muss die Wahrscheinlichkeit *P(sensordata|context)* für das Auftreten eines Sensordatums *sensordata* unter der Bedingung, dass *sensordata* in der Kontextklasse *context* ist, a priori bekannt sein oder geschätzt werden können [1].
Die Wahrscheinlichkeit, sich bei gegebenen Sensordaten in einem gewissen Kontext zu befinden, berechnet sich nach Bayes wie folgt:

$$P(context|sensordata)=\frac{P(sensordata|context)\cdot P(context)}{P(sensordata)} \quad ,$$

wobei

$$P(sensordata)=\sum_{i=1}^{n} P(sensordata|context)\cdot P(context)$$

die Wahrscheinlichkeit eines Sensordatums unabhängig von der Kontextklasse ist [2].
Vorteilhaft bei dieser Definition der bedingten Wahrscheinlichkeit ist die Wechselseitigkeit der bedingten Wahrscheinlichkeiten.

2.2 Gauß-Klassifizierung

Bei der Gauß-Klassifizierung wird eine zugrunde liegende Wahrscheinlichkeitsverteilung auf Basis der Normalverteilung unterstellt. Die Parameter für die Klassifikation können geschätzt werden.
Da die Normalverteilung nicht immer sehr gut geeignet ist, um Sensordaten zu klassifizieren, werden oft eher Verfahren wie Parzen Window oder K Nearest Neighbors angewandt [7].

2.3 Parzen Window

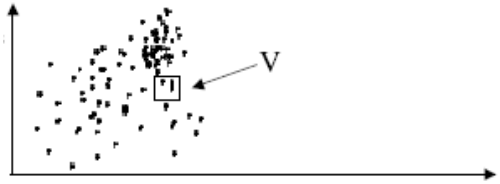
Da bei diesem Verfahren keine Dichtefunktion geschätzt werden, wird hier ein grundlegend anderes Vorgehen an den Tag gelegt. Es sei k die Anzahl der Sensordaten, die innerhalb eines Fensters vom Volumen V liegen und n die Anzahl aller abgetasteten Sensorwerte. Allgemein kann so die Wahrscheinlichkeit P(x), dass sich ein Vektor x in einem solchen Fenster befindet, durch

$$P(x)\approx\frac{k/n}{V}$$

geschätzt werden. Um Problemen durch zu kleine oder zu große Volumina vorzubeugen, wird auch

$$V=\frac{1}{\sqrt{n}}$$

benutzt [7].
Die Wahrscheinlichkeitsverteilung P(x) kann dann direkt aus den Daten gewonnen werden [8].



2.4 K Nearest Neighbors

Die Funktionsweise von K Nearest Neighbors ist der von Parzen Window sehr ähnlich. Es wird ebenfalls ein Fenster vom Volumen V gewählt, mit dem Unterschied, dass hier V eine Funktion über den Daten ist. Es werden immer die k nächsten Nachbarn eines Sensordatums betrachtet, wobei $k=\sqrt{n}$ und n die Anzahl der Sensordaten insgesamt ist. Der Algorithmus läuft nach dem Schema ab, dass zunächst die k nächsten Nachbarn von Sensordatum x gesucht werden. Man ordnet x derjenigen Klasse zu, die von diesen k nächsten Nachbarn am häufigsten repräsentiert wird [7].

2.5 Distanzen

Um für einen K Nearest Neighbors Algorithmus herausfinden zu können, welches die k nächsten Nachbarn sind, müssen die Abstände der einzelnen Vektoren gemessen werden können. Seien im folgenden x und y d-dimensionale Vektoren, wobei x und y die Sensordaten von d verschiedenen Sensoren sind. Dann kann, mittels nachfolgender Distanzen, der Abstand bestimmt werden [9]. Es kann außerdem eine direkte Klassifizierung durchgeführt werden, indem man die Distanzen zwischen Klassenmittelpunkten und Merkmalsvektor misst [1].

2.5.1 Manhattan Distanz [9]

Es werden bei der Manhattan Distanz vom jeweils betrachteten Sensor (mittels Bildung der Differenzen) immer zwei Werte miteinander verglichen, indem der kleinere vom größeren Wert abgezogen wird. Dies wird für alle der d Sensoren durchgeführt und dann alle Differenzen miteinander addiert.
Die Manhattan Distanz berechnet man formal folgendermaßen:

$$Manh(x, y) = \sum_{i=1}^{d} |x_i - y_i| \quad .$$

Die Umrisslinien für die Manhattan Distanz sehen im 2-dimensionalen Fall wie folgt aus:

2.5.2 Euklid Distanz [9]

Die Euklid Distanz wird ganz ähnlich wie die Manhattan Distanz berechnet. Wieder werden zwei Merkmalsvektoren miteinander verglichen, indem die Werte der jeweils gleichen Sensoren miteinander verglichen werden. Dabei werden wieder zunächst die Differenzen ihrer jeweiligen Werte gebildet und diese dann jedoch quadriert. Dabei kann die Nutzung der Betragsfunktion wegfallen, da die einzelnen Differenzen durch die Quadratbildung ohnehin positiv sind. Diese Quadrate werden anschließend für alle d Sensoren addiert und aus dieser Summe die Wurzel berechnet.
Die Euklid Distanz wird formal nach diesem Schema berechnet:

$$Eukl(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \quad .$$

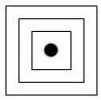
Die Umrisslinien für die Euklid Distanz haben im 2-dimensionalen Fall folgende Gestalt:

2.5.3 Chebychev Distanz [9]

Bei der Chebychev Distanz findet der Vergleich beziehungsweise die Abstandsmessung zweier Vektoren dadurch statt, dass nur der Sensor von Interesse ist, bei dem der Unterschied zwischen den beiden Vektoren am größten ist. Ergebnis dieser Distanzbildung ist also die maximale Abweichung zwischen zwei Sensordaten bei Betrachtung der d Sensoren.
Die Chebychev Distanz berechnet man formal folgendermaßen:

$$Cheb(x, y) = max_i |x_i - y_i|, i = 1, \ldots, d$$

und sie hat im 2-dimensionalen Fall folgende Umrisslinien:



2.5.4 Mahalanobis Distanz [9]

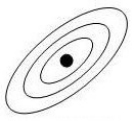
Die drei oben genannten Verfahren zur Distanzberechnung ließen sich jeweils im Ergebnis verbessern, wenn an die Differenzen von x_i und y_i noch ein Gewicht g_i hinzu multipliziert würde. Da die Auswahl solcher Gewichte aber eher subjektiven Kriterien unterliegt und es somit dennoch zu Verzerrungen kommen kann, empfiehlt sich die Verwendung der Mahalanobis Distanz, da diese die inverse Kovarianzmatrix der Sensordaten sozusagen als Gewichtung verwendet.
Die Mahalanobis Distanz ist eine Art gewichtete Euklid Distanz, da diese beiden Distanzen übereinstimmen, sofern D die Einheitsmatrix ist.
Die Mahalanobis Distanz wird berechnet wie folgt:

$$Mahalanobis(x, y) \equiv \sqrt{(x - y)' \cdot Cov(D)^{-1} \cdot (x - y)} \quad ,$$

wobei D eine Datensatzmatrix verschiedener Sensoren über eine gewisse Zeitdauer ist. D hat folgende Gestalt:

$$D = \begin{pmatrix} x_1(t_1) x_1(t_2) \cdots x_1(t_n) \\ x_2(t_1) x_2(t_2) \cdots x_2(t_n) \\ \vdots \\ x_d(t_1) x_d(t_2) \cdots x_d(t_n) \end{pmatrix} \quad .$$

Die Mahalanobis Distanz weist im 2-dimensionlen Fall folgende Umrisslinien auf:

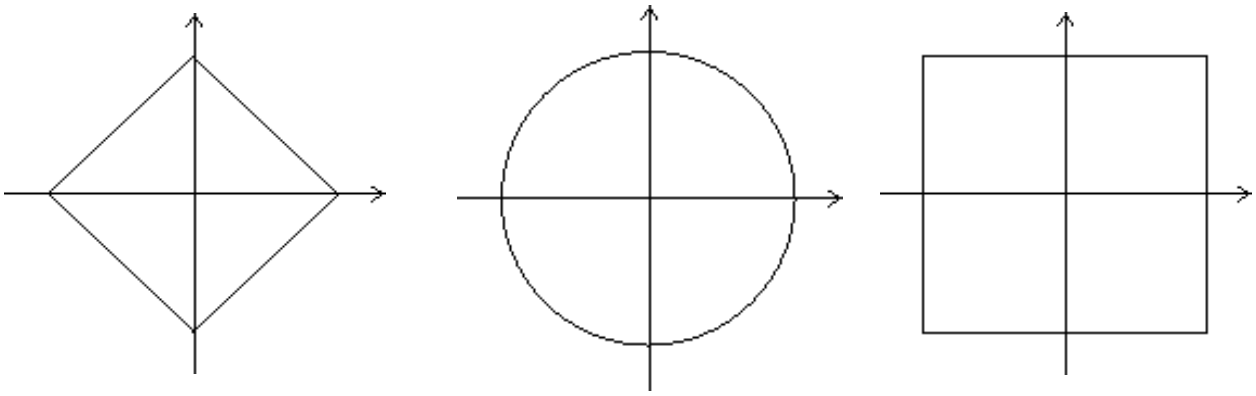


2.5.5 Fazit und Vergleich der einzelnen Distanzen

Der Begriff Norm ist die Verallgemeinerung des Begriffs der Länge eines Vektors. Mittels der Distanzbildung wird im Grunde nichts anderes getan als gemessen wie lange der Vektor zwischen zwei betrachteten Datenvektoren ist.
Allgemein können oben genannte Distanzen mit der Minkowski Distanz dargestellt werden
Die Minkowski Distanz wird für ein beliebiges λ definiert als:

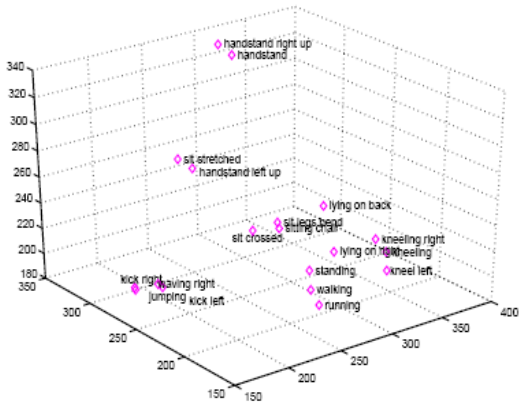
$$Mink(x, y) = \sqrt[\lambda]{\sum_{i=1}^{d} |x_i - y_i|^{\lambda}} \quad .$$

Es fällt auf, dass für λ = 1 Minkowski und Manhattan Distanz gleich sind. Dagegen gilt für Minkowski und Chebychev Distanz, dass sie für λ = ∞ übereinstimmen. Für λ = 2 sind Minkowski und Euklid Distanz identisch [9].
Betrachtet man die Minkowski Distanz, erklärt sich auch das Zustandekommen der Umrisslinien der einzelnen Distanzen. Im zweidimensionalen Fall erhält man mittels obiger Distanzen durch Bildung der Menge aller auf 1 normierten Vektoren, die folgenden Einheitskreise [10].



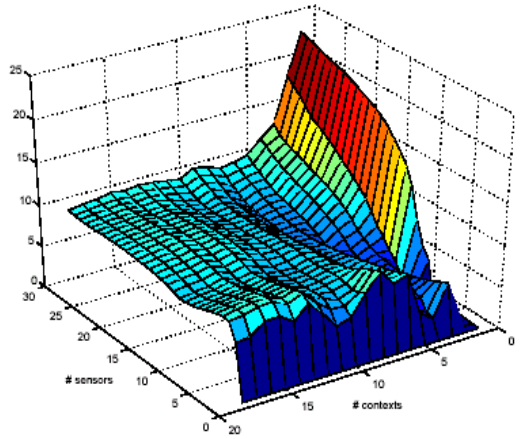
Einheitskreis für λ = 1 | Einheitskreis für λ = 2 | Einheitskreis für λ = ∞

Der Nachteil einer Klassifikation über die Distanzbildung mittels Manhattan, Euklid und Chebychev Distanz ist sicherlich, dass lediglich die Mittelpunkte einzelner Klassen einbezogen werden. Hier liegt eindeutig der Vorteil der Mahalanobis Distanz, die auch die Kovarianzen berücksichtigt [10].
In [11] wurde versucht, mittels einer Anordnung von 30 Beschleunigungsmessern die Bewegungen verschiedener Testpersonen zu klassifizieren. Ziel dieser Studie war es zu demonstrieren, wie sich Kontextalgorithmen unter der Bedingung verhalten, dass sich die Anzahl der Sensoren und die Anzahl der Kontexte verändert. Zum einen sollte das Klassifikationsverhalten als eine Funktion über der Anzahl der Sensoren geprüft werden und zum anderen das Verhalten der Erkennungsrate bei zunehmender Anzahl der Kontexte. Es wurden 20 Kontexte und 30 Beschleunigungsmesser benutzt. Bei den Kontexten handelte es sich um Bewegungskontexte, wie knien, gehen, stehen (auch Handstand), laufen, sitzen, liegen, kicken, winken, springen und andere mehr (siehe untere Abbildung), die mit Hilfe einer Anordnung der 30 Beschleunigungsmesser, die am Körper angebracht wurden, wahrgenommen werden sollten.

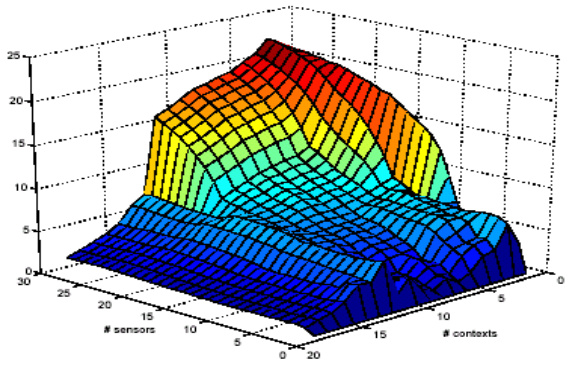


Mittelwertsverteilung der einzelnen Kontexte;
3D Darstellung

Dies führte zu folgender dreidimensionaler Darstellung:



3D Darstellung der Erkennungsrate unter Benutzung der Klassenmittelpunkte.



3D Darstellung der Erkennungsrate unter zusätzlicher Einbeziehung der Varianzen.

Die x-Achse repräsentiert die Anzahl der eingesetzten Sensoren, wobei mit dem besten Sensor begonnen und immer der Nächstbeste hinzugefügt wurde. Die y-Achse zeigt die Anzahl der einbezogenen Kontexte. Im Experiment wurde nacheinander der jeweils am besten von den restlichen zu unterscheidende Kontext hinzugefügt.
Die Erkennungsrate für diese Kontexte wurde als Standardabweichung zwischen den Mittelwerten der Kontextklassen implementiert. Die obigen 3D Darstellungen sollen zeigen, dass sich die Erkennungsrate immens verschlechtert, sobald die Anzahl der unterschiedlichen Kontexte zunimmt. Es ist auch feststellbar, dass die Erkennungsrate mit zunehmender Sensoranzahl steigt. Beide Abbildungen sind jedoch stark von der Art der hinzugefügten Sensoren und Kontexte abhängig.

3. Pattern-Matching Algorithmen

Pattern-Matching Algorithmen lassen sich grundsätzlich in die Bereiche Klassifikation und Mustergruppierung oder Clustering unterteilen. Bei der Klassifikation werden Muster vordefinierten Musterklassen zugeordnet. Beim Clustering bestehen keine vordefinierten Klassen, d.h. es dreht sich hier, im Gegensatz zu Abschnitt 2, um unüberwachte Verfahren. Neuronale Netze und Hidden Markov Modelle können als ein statistisches Klassifikationsverfahren betrachtet werden. Außer statistischen Verfahren gibt es noch Template Matching und syntaktische Verfahren, auf die im Folgenden jedoch nicht weiter eingegangen wird [12].

3.1 Hidden Markov Model (HMM)

Hidden Markov Modelle sind stochastische Modelle, die eingesetzt werden, um Signale zu analysieren und zu beschreiben [13]. Sie sind eine wichtige Methode zur Klassifizierung von dynamischen Mustern und haben sich in der Sprach-, Handschrift- und Gestikerkennung durchgesetzt [12]. Bei diesen Modellen handelt es sich um zweistufige stochastische Prozesse, die zum einen aus einer Markovkette und zum anderen aus der Ausgabe des Hidden Markov Modells bestehen. Eine Markovkette ist eine Kette von Zuständen, wobei den einzelnen Zuständen Übergangswahrscheinlichkeiten zugeordnet werden. Die Markovkette ermöglicht eine Beschreibung des zeitlichen Verlaufs eines Signals, indem den Zuständen der Markovkette Übergangswahrscheinlichkeiten zugeordnet werden [13]. Ein Hidden Markov Modell kann eingesetzt werden, wenn der Prozess der Kontexterkennung als Abfolge von Zuständen betrachtet wird. Ein HMM λ kann dargestellt werden als $\lambda = (\Pi, A, B)$. Π stellt die Anfangswahrscheinlichkeitsverteilung mit der Wahrscheinlichkeit $\pi(i)$ dar, dass der Zustand S_i der Startzustand ist. A

ist eine Zustandsübergangsmatrix mit $A = \{a_{ij}\}$, wobei a_{ij} die Wahrscheinlichkeit angibt, dass ein Zustandsübergang von S_i nach S_j stattfindet. $B = \{b_1, ..., b_n\}$ ist die Menge der Emissionswahrscheinlichkeiten, wobei $b_i(o)$ die Wahrscheinlichkeit ist, dass im Zustand S_i die Beobachtung o gemacht wird [14]. Abhängig vom aktuellen Zustand wird mittels der Emissionswahrscheinlichkeiten die Ausgabe $O = o_1, ..., o_T$ erzeugt. Diese $b_j(\cdot)$ berechnet man mittels der Summe über gewichtete c_{kj} multivariate Gaußverteilungen $N(\cdot)$ mit Erwartungswert μ_{kj} und der Kovarianzmatrix Σ_{kj} für den Zustand j und die Gaußverteilung k:

$$b_j(o_t) = \sum_{k=1}^{K} c_{kj} \cdot N(o_t | \mu_{kj}, \Sigma_{kj}) \quad [12].$$

Setzt man Hidden Markov Modelle zur Mustererkennung ein, so wird zu jeder Klasse ω_i ein Modell λ_k verwendet. Aufgrund dieser Vorgehensweise handelt es sich bei diesen Modellen wieder um ein überwachtes Verfahren, da Kenntnis über die einzelnen Klassen ω_i bestehen muss, um ein Modell λ_k bilden zu können. Wenn also eine Merkmalsfolge O klassifiziert werden soll, wird für jedes λ_k die Wahrscheinlichkeit $P(O|\lambda_k)$ berechnet. Nach Bayes errechnet man $P(\lambda_k|O)$ mittels oben angegebenem Schema. Einer Merkmalsfolge wird dann die Klasse mit der höchsten Wahrscheinlichkeit zugewiesen.
Die Wahrscheinlichkeit $P(O|\lambda_k)$ lässt sich effizient mit dem Forward-Backward Algorithmus berechnen. Es seien $\alpha_t(j) = P(o_1\ o_2\ ...\ o_t, q_t = j \mid \lambda)$ die Vorwärtswahrscheinlichkeiten und $\beta_t(i) = P(o_{t+1}\ o_{t+2}\ ...\ o_T \mid q_t = i, \lambda)$ die Rückwärtswahrscheinlichkeiten, wobei T die Länge der Observationssequenzen ist. Sowohl Vorwärts- als auch Rückwärtswahrscheinlichkeiten sind T x N Matrizen, wenn das Hidden Markov Modell aus N Zuständen besteht. Diese Wahrscheinlichkeiten lassen sich dann mittels Forward- bzw. Backward Algorithmus rekursiv berechnen.
Die Vorwärtswahrscheinlichkeiten beim Forward Algorithmus werden so initialisiert, dass zunächst für jeden Zustand die Wahrscheinlichkeit, dass dieser Zustand Startzustand ist, mit der Emissionswahrscheinlichkeit für das erste Merkmal der Observationssequenz multipliziert wird. Formal sieht diese Initialisierung wie folgt aus:

$$\alpha_1(j) = \pi(j) \cdot b_j(o_1), \forall\, 1 \le j \le N \quad .$$

Die Vorwärtswahrscheinlichkeiten werden dann rekursiv unter Betrachtung des vorherigen Zustands berechnet. Im einzelnen werden für einen Zustand j zum Zeitpunkt t die Vorwärtswahrscheinlichkeiten $\alpha_{t-1}(i)$ für einen Zustand i zum Zeitpunkt t – 1 mit den Übergangswahrscheinlichkeiten a_{ij} für einen Zustandsübergang von i nach j multipliziert. Dies geschieht für alle direkten Vorgänger von i im Hidden Markov Modell. Danach werden diese Produkte aus Übergangswahrscheinlichkeiten und Vorwärtswahrscheinlichkeiten addiert. Diese Summe wird dann mit den Emissionswahrscheinlichkeiten für den jeweiligen Zustand j für das Auftreten eines Merkmals zum Zeitpunkt t multipliziert. Formal geschieht dies nach folgender Vorgehensweise für t = 2, 3, ..., T:

$$\alpha_t(j) = b_j(o_t) \sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij}, \forall\, 1 \le j \le N \quad .$$

Wenn man dieses solange durchführt, bis man sämtliche Vorwärtswahrscheinlichkeiten zu allen Beobachtungsmerkmalen berechnet hat, lässt sich $P(O|\lambda)$ bestimmen als:

$$P(O|\lambda) = \alpha_T(N) \quad .$$

Der Backward Algorithmus berechnet zeitlich gesehen $P(O|\lambda)$ aus der anderen Richtung. Die Rückwärtswahrscheinlichkeiten β werden zum Zeitpunkt T initialisiert mit:

$$\beta_T(N) = 1, \forall\, 1 \le j \le N \quad .$$

Mit dieser Initialisierung wird aus $\beta_{t+1}(j)$ $\beta_t(i)$ berechnet, indem die Übergangswahrscheinlichkeit a_{ij}, dass ein Zustandsübergang von i nach j stattfindet, mit der Emissionswahrscheinlichkeit $b_j(o_{t+1})$ des Observationsmerkmals zum Zeitpunkt t + 1 und der Rückwärtswahrscheinlichkeit $\beta_{t+1}(j)$ (ebenfalls zum Zeitpunkt t + 1) multipliziert wird. Dies wird für alle Zustände des Hidden Markov Modells durchgeführt und die Ergebnisse miteinander addiert. Diese Summe ergibt dann die Rückwärtswahrscheinlichkeit zum Zeitpunkt t. Formal gilt für alle t = T – 1, T – 2, ..., 1:

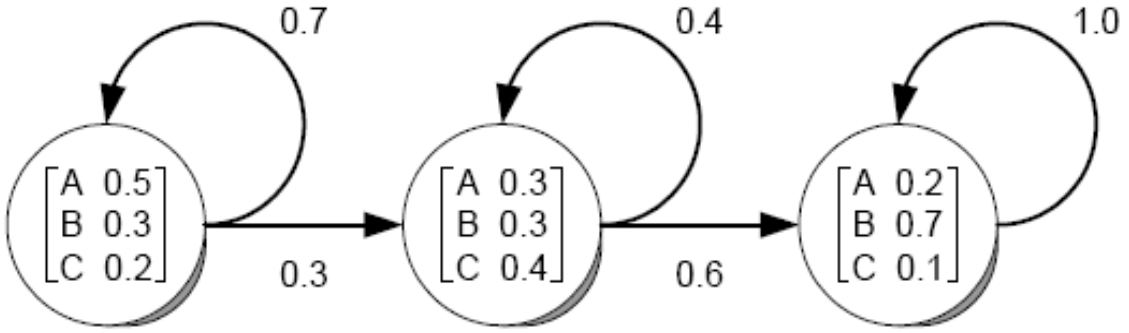
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j), \forall\, 1 \le j \le N \quad .$$

$P(O|\lambda)$ ergibt sich dann aus dem Produkt über den Anfangswahrscheinlichkeiten $\pi(j)$, dass der Zustand j der Startzustand ist, der Emissionswahrscheinlichkeit für den Zustand j für das erste Beobachtungsmerkmal und der Rückwärtswahrscheinlichkeit zum Zeitpunkt t = 1, ebenfalls für den Zustand j. Dies wird für alle Zustände des Hidden Markov Modells durchgeführt und die Ergebnisse miteinander addiert:

$$P(O|\lambda) = \sum_{j=1}^{N} \pi(j) \cdot b_j(o_1) \cdot \beta_1(j) \quad [12].$$

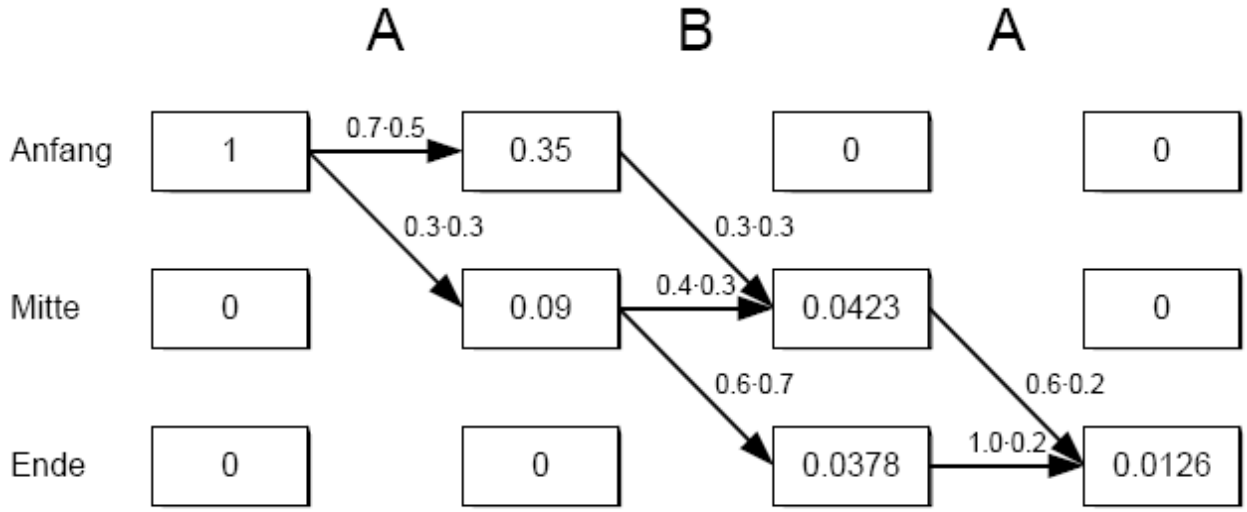
Da sich die Vorgehensweise von Forward und Backward Algorithmus sehr stark ähnelt, wird hier lediglich der Forward Algorithmus an einem Beispiel verdeutlicht. Für den Backward Algorithmus ist analog vorzugehen.

Gegeben sei folgendes Hidden Markov Modell λ:



Hidden Markov Modell

Für die Wahrscheinlichkeit der Zeichenkette O = ABA ergibt sich mit dem Forward Algorithmus dann folgendes:



Forward Algorithmus

Die Wahrscheinlichkeit von P(O|λ) ist also 0.0126.
Hidden Markov Modelle finden in unterschiedlichen Bereichen der Mustererkennung Verwendung. Sie werden zum Beispiel in der Sprach- oder Schrifterkennung eingesetzt, weil hier ein Wort als Ganzes erfasst werden kann und nicht Buchstabe für Buchstabe betrachtet wird.
Hidden Markov Modelle sind gut geeignet, um zeitdiskrete Ereignisse zu klassifizieren und zu modellieren. Außerdem ist die einfache graphische Darstellbarkeit der Ergebnisse von Vorteil. Ein großer Nachteil aber ist die Markov'sche Bedingung selbst, die auch als Gedächtnislosigkeit eines Prozesses bezeichnet wird. Es ist zwar von Vorteil, dass lediglich Kenntnis über die Gegenwart bestehen muss, um eine Prognose für die Zukunft aufstellen zu können, es werden jedoch keine Ereignisse berücksichtigt, die weiter zurückliegen (ein Zustand zum Zeitpunkt t hängt allein vom Zeitpunkt t – 1 ab).

3.2 Neuronale Netze

Wie bereits eingangs erwähnt können Neuronale Netze ebenfalls als statistische Klassifikationsmethode bezeichnet werden. Ein neuronales Netz ist ein Graph, dessen Knoten als Neuronen bezeichnet werden. Sie dienen der Informationsaufnahme und Weiterleitung. Dabei ist der Einfluss auf ein empfangendes Neuron um so größer, je höher der Aktivitätslevel und je höher das Gewicht zwischen zwei Neuronen sind. Es sei im Folgenden a_j der Aktivitätslevel des Neurons j und w_{ij} das Gewicht zwischen sendendem Neuron j und empfangendem Neuron i. Formal kann der Input eines Neurons damit wie folgt dargestellt werden:

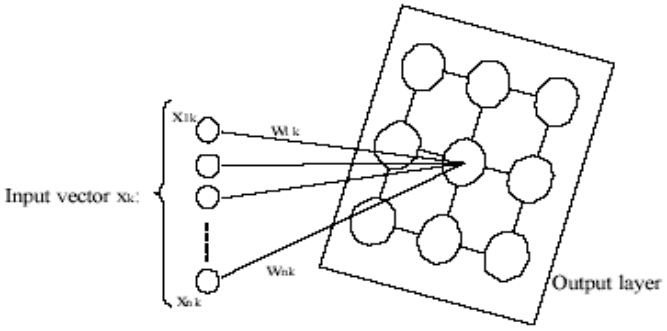
$$input_{ij} = a_j \cdot w_{ij}$$

Das Wissen eines Netzes wird durch seine Gewichtung ausgedrückt, wobei man Lernen bei einem neuronalen Netz als die Gewichtsveränderung in den interagierenden Neuronen definieren kann. Die Lernphase wird auch als Trainingsphase bezeichnet. In dieser Phase werden die Gewichte einzelner Neuronen im Netz verändert. Lernregeln

geben dabei vor, wie diese Modifikationen vorzunehmen sind. Man unterscheidet zwei Typen von Lernregeln. Zum einen überwachtes Lernen und zum andern unüberwachtes Lernen. Bei überwachtem Lernen oder supervised Learning wird die korrekte Ausgabe vorgegeben, woraufhin die Gewichte anhand dieser Vorgaben verändert werden. Dazu wird meist ein Fehlermaß eingeführt, das minimiert werden soll. Um den beobachteten Fehler zwischen erwünschtem Wert und der tatsächlichen Ausgabe eines Neuronalen Netzes zu verringern, werden dann die Gewichte im Netz verändert. Nach Abschluss dieses Lernprozesses sollte ein Netz dann die Fähigkeit besitzen, bei unbekannter Eingabe eine korrekte Ausgabe zu erzeugen. Bei unüberwachtem Lernen oder unsupervised Learning gibt es keine derartigen Vorgaben. Hier wird sozusagen direkt versucht, Daten bestimmten Klassen zuzuweisen, deren Klassenattribute der Eingabe am ähnlichsten sind.

3.2.1 Kohonen Self-Organizing Map (SOM)

Eine spezielle Art von Neuronalen Netzen sind die Kohonen Self-Organizing Maps. Sie bilden selbstständig Karten (Maps) über ihre Eingabe. Dies kann auch als clustern der Eingabe aufgefasst werden, womit dieses Verfahren als ein unüberwachtes Klassifikationsverfahren betrachtet werden kann. Kohonen Netze bestehen aus einer Eingabe- und einer Ausgabeschicht, wobei die Ausgabeschicht oft zweidimensionaler Struktur ist [15].



Kohonen Self-Organizing Map [17]

Dabei werden Einheiten im Verbund mit anderen Einheiten, in Abhängigkeit des Sensorinputs, für Tasks rekrutiert. Die oben dargestellte Karte veranschaulicht, dass ähnliche Eingabedaten auf Einheiten einer bestimmten Region der Karte dargestellt und benachbarte Einheiten bei ähnlicher Eingabe aktiv werden [16]. Um ein solches Netz zu trainieren wird ein konkurrierender Algorithmus den man als Winner-Takes-All Algorithmus bezeichnet oder auch als competetive learning. Er gehört zur Klasse der unüberwachten Lernalgorithmen [15]. Voraussetzung für diesen Algorithmus ist, dass jedes Neuron i einen eigenen Prototypvektor w_i hat, um Speicherplatz zu reservieren und um bestimmte Eingabevektoren zu speichern. Initiiert werden diese Prototypvektoren mit kleinen Zufallswerten und der Dimension n (gleich der Dimension der Eingabe). Der Lernalgorithmus der SOM für die Prototypvektoren hat dabei folgende Gestalt:

$$w_i = w_i + \alpha \cdot \eta(winner) \cdot (x_i - w_i), \forall i \in \{1, \ldots, n\} \quad .$$

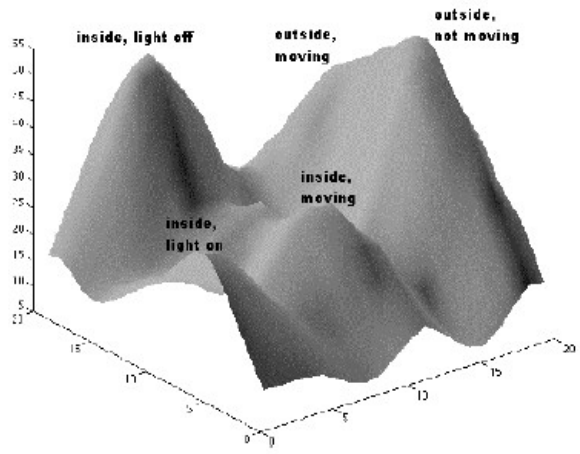
Die Lernrate α und die Nachbarfunktion η(winner) nehmen Werte zwischen 0 und 1 an [16]. Dieser Algorithmus ordnet die Eingabe, indem jeder Eingabe eine bestimmte Einheit der Karte zugewiesen wird. Nach mehreren Iterationen ist die resultierende Karte topologisch geordnet, d.h. ähnliche Eingaben aktivieren benachbarte Einheiten[17]. Winner ist immer die Einheit deren Prototypvektor die kleinste Euklid'sche Distanz zum Eingabevektor hat:

$$winner = arg\ \min_j \sqrt{\sum_{j=1}^{n} (x_j - w_j)^2} \quad .$$

Die Nachbarfunktion wird traditionell als Gauß'sche Glockenkurve implementiert:

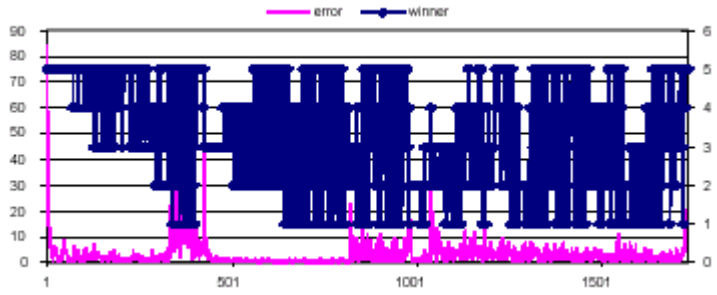
$$\eta(winner) = \frac{1}{\sqrt{2\pi}\, nb} \cdot e^{\frac{-0{,}5 \cdot (winner - current)^2}{nb^2}} \quad .$$

Die Nachbarn von Winner, die ihren Prototypvektor verbessern dürfen, liegen hierbei in einem Radius der Größe nb. Selbstorganisierte Karten sind auch gut geeignet, um verrauschte Daten zu verarbeiten [17].
Nach ein paar Iterationen beginnen die Neuronen sich selbst strukturiert und topologisch zu organisieren, so dass unterschiedliche Sensoreingaben unterschiedliche Neuronen aktivieren. Die Aktivität einzelner Neuronen kann überwacht und hinterher dargestellt werden.

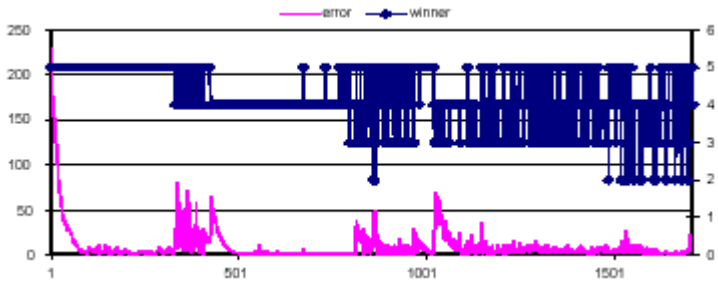


Darstellung der Aktivitäten einer SOM für die Sensorwerte fünf einfacher Kontexte [17]

Diese Aktivitätendarstellung zeigt, welche Neuronen bei unterschiedlichen Sensoreingaben aktiviert werden. Dabei werden durch x- und y-Achse die zweidimensionale SOM repräsentiert, wobei die z-Achse anzeigt, wie oft ein bestimmtes Neuron gewonnen hat [17].
In [16] werden die Einheiten der SOM durch Smart-Its verkörpert. Die Topologie der SOM orientiert sich dabei an den physikalischen Distanzen der Smart-Its. Die Eigenschaft eines Ad Hoc Netzwerks ist dadurch gewährleistet, dass Einheiten versetzt, eingefügt oder entfernt werden können.
Die Selbstorganisation findet hierbei dadurch statt, dass zunächst die Eingabe gelesen wird. Daraufhin berechnen die Einheiten die Euklid Distanz zwischen der Eingabe und ihrem eigenen Prototypvektor. Um den Winner zu ermitteln, wird ein Datenpaket bestehend aus Unit ID, Timestamp und Error mittels Broadcast verschickt. Dabei ist die Einheit mit dem kleinsten Error Winner. Wobei die Unit ID der eindeutigen Identifizierung und der Timestamp der Eliminierung alter Datenpakete dient. Der Error enthält die Euklid Distanz zwischen Eingabe und Prototypvektor. Danach ermittelt jede Einheit die physikalische Distanz zum Winner, um nach obiger Update-Regel seinen Prototypvektor verändern zu dürfen. Resultat der SOM ist die eigenständige Aktivierung der Winner-Einheit. Dabei hängt der Erfolg der Kohonen SOM stark von der Wahl der Lernrate und des Nachbarradiusparameters ab.



Benutzt wurde eine hohe Lernrate. Dadurch vergessen die Einheiten leicht ihren Prototypvektor.



Benutzt wurde eine normale Lernrate. Einheit 5 spezialisiert sich auf den ersten Kontext.
Einheit 4 spezialisiert sich auf den dritten Kontext.

Die Abbildungen zur Aktivitätendarstellung der einzelnen Einheiten zeigen, dass bei hoher Lernrate der Prototypvektor mit beliebigen Inputs leicht überschrieben werden kann. Dies ist auch bekannt als “Catastrophic Forgetting”. Daher sind kleinere Lernraten vorzuziehen, da sie den Prototypvektor schützen, obwohl durch Fluktuationen in den Sensordaten verschiedene Einheiten für denselben Kontext rekrutiert werden.

Dies zeigt, dass Sensornetzwerke lernen können, sich auf die Erkennung bestimmter Zustände der Umgebung oder des Kontexts zu spezialisieren. Außerdem können eingehende Sensordaten durch Selbstorganisation verteilter Sensormodule mit begrenzter Verarbeitungskapazität gebündelt werden [16].
Kohonen SOMs können benutzt werden, um Eingabedaten zu bündeln oder zu clustern. Resultat dieser Clustering-Schicht könnte ein einfacher Identifizierer sein, der mit einem Label verbunden ist. Da eine der Systemanforderungen aus [17] vorschreibt, dass vorschriftswidriges User-Feedback auftreten kann, ist es möglich, dass dieses Label noch nicht gegeben wurde. In diesem Fall ist ein distanzgewichteter K Nearest Neighbors Algorithmus für die Suche nach einer topologisch geordneten Cluster-Map verantwortlich, d.h. das wahrscheinlichste Label wird dadurch bestimmt, dass die K nähesten Labels der Karte mit den Distanzen zu der Einheit ohne Label multipliziert werden.
Das in [17] beschriebene System basiert auf Sensoren, die sehr verrauschte Signale weitergeben können. Aus diesem Grund ist eine Überwachungsschicht empfehlenswert. Sie soll hauptsächlich Übergänge von bekannten Kontexten zu anderen überwachen. Jeder Kontext wird dazu durch einen Zustand repräsentiert, wobei Zustandsübergänge durch Kanten dargestellt werden. Das Modell erhält einen Wahrscheinlichkeitswert für jede Kontextveränderung. Folglich überprüft das Überwachungsmodell, ob ein Übergang wahrscheinlich ist. Falls ein Übergang nicht sehr wahrscheinlich ist, wird er auch nicht durchgeführt. Es wird aber ein Mechanismus initiiert, so dass der Übergang nach mehreren Versuchen hintereinander wahrscheinlicher wird. Es sei $P(x \rightarrow y)$ die Wahrscheinlichkeit, dass ein Zustandsübergang von x nach y auftritt, k sei die Größe des Pufferspeichers und μ ein Zähler zwischen 0 und k. Ein Übergang findet also statt, wenn:

$$P(x \rightarrow y) > \frac{k-\mu}{k} \quad .$$

Jeder Zustandsübergang ist somit einzig und allein vom vorherigen Zustand abhängig, wodurch dieses Modell zu einer Markovkette wird. In jedem Zustand wird auch gespeichert, wie lange in dem jeweiligen Kontext verharrt wurde. Dies kontrolliert die Flexibilität der SOM. Je neuer ein Kontext ist, desto flexibler sollte die Karte sein. Diese Methode schützt das System davor, von Ausreißern in den Sensorsignalen durcheinander gebracht zu werden. Das Resultat dieses Modells ist ein gerichteter Graph, der das Nutzerverhalten in Bezug auf die durchlaufenen Kontexte widergespiegelt. Wenn ein Benutzer eher dazu tendiert von Kontext A nach Kontext B zu wechseln, wird das in der Verbindungsstärke des Graphen widergespiegelt.
Im Experiment zu [17] wurde eine Hose mit Beschleunigungsmessern ausgestattet. Das Resultat ist Tabelle 1 zu entnehmen. Hierbei wurde der erste Zyklus benutzt, um die Kohonen SOM anzuordnen, die Zyklen 2 und 4 wurden zum Training benutzt und die Zyklen 3 und 5 zum Testen.

Aktivität	Erkennung in Zyklus	
	3	5
sitzen	96%	96%
stehen	94%	94%
gehen	75%	75%
laufen	74%	78%
treppauf steigen	45%	42%
treppab steigen	48%	64%
Fahrrad fahren	89%	91%

Tabelle 1: Ergebnisse eines Tests mit Beschleunigungsmessern in einer Hose

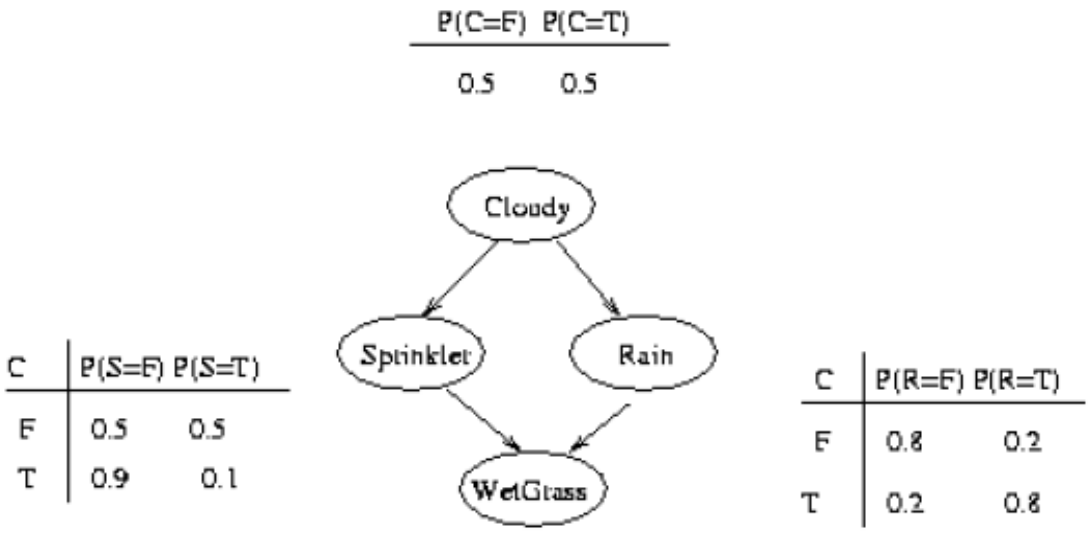
Tabelle 1 kann entnommen werden, dass gehen, laufen, trepp aufsteigen und trepp absteigen weniger erfolgreich erkannt wurden als sitzen, stehen und Fahrrad fahren. Verbesserungen können durch vorkonfigurierte KSOMs oder durch überschreiben des Prototypvektors erzielt werden.
In der Markovkette wurde die Verbindung von sitzen zu stehen sehr stark, wobei andere schwach bis nicht existent waren. Hier können allerdings durch Verlängerung der Trainingsphase Verbesserungen vorgenommen werden.
Wie in Abbildung 10 zu sehen ist, sind Kohonen Self-Organizing Maps gut zu visualisieren und dadurch leicht nachzuvollziehen.
Ein Nachteil einer Kohonen SOM ist jedoch, dass der traditionelle Algorithmus mit einer hohen Lernrate und einem großen Nachbarradius initialisiert wird. Beide werden dann nach und nach fixiert. Nach dieser Phase ist der Algorithmus kaum noch lernfähig, was für Systeme, die auch weiterhin anpassbar bleiben sollen sehr hinderlich ist. Damit der Algorithmus flexibel bleibt, müssten zuvor gespeicherte Prototypvektoren überschrieben werden können. Im Feld des Machine Learning ist dies bekannt als “Stability-Plasticity-Dilemma” oder auch als “Catastrophic Forgetting”.
Ein weiterer Nachteil ist, dass sich bei steigender Anzahl der Eingabedaten der Lernprozess verlangsamt. Hierbei spricht man von dem “Fluch der Dimensionalität”. Dem wurde versucht durch eine Hierarchie von Kohonen Maps zu entgehen. Dies führte jedoch zu höheren Designproblemen und ist bei Hunderten von Eingaben auch keine Lösung [17].

4. Entscheidungsmechanismen bei nicht eindeutig identifizierbaren Kontexten

Wie das oben angeführte Beispiel zur Erkennung der Bewegungskontexte und die dazu angeführten Abbildungen 5 und 6 zeigen, sind Systeme, die zur Kontexterkennung eingesetzt werden, nicht immer in der Lage, den aktuellen Kontext exakt zu bestimmen. Aus diesem Grund werden nachfolgende Mechanismen, die bei Unsicherheit unterstützend eingesetzt werden könnnen, benötigt [18].

4.1 Bayes'sche Netze

Ein Bayes'sches Netz ist ein azyklischer gerichteter Graph, bestehend aus Knoten und Kanten und kann zur Entscheidungsfindung bei Unsicherheit unterstützend eingesetzt werden. Knoten stehen für Variablen und Kanten für bedingte Abhängigkeiten zwischen Variablen. Dabei beziehen sich die Abhängigkeiten auf die tatschächlichen Wahrscheinlichkeiten [19]. Ein wichtiger Vorteil gerichteter Graphen ist die gute Darstellbarkeit kausaler Zusammenhänge. Eine Kante, die von einem Knoten a zu einem Knoten b führt, kann so interpretiert werden, dass der Knoten a Verursacher des Knotens b ist [21]. Alle Knoten, die eine Kante haben, die zu einem Knoten k führt, heißen Elternknoten von k. Für Knoten, die keine Elternknoten haben, müssen a priori die Wahrscheinlichkeiten bekannt sein. Mit diesen Wahrscheinlichkeiten lassen sich dann mittels Inferenz die Wahrscheinlichkeiten unbekannter Variablen berechnen [20].
Bayes'sche Netze tragen ihren Namen, da sie Bayes'sche Regeln zur probabilistischen Inferenz benutzen [21].



Bayes'sches Netz

In obigem Bayes'schen Netz stehen F und T für die Wahrheitswerte falsch und wahr. Man sieht in diesem Beispiel, dass das Ereignis „Gras ist nass“ (W = T) auf zwei verschiedene Arten zustande kommen kann. Entweder ist der Rasensprenger an (S = T) oder es regnet (R = T). Die Stärke der Beziehung wird in den Tabellen dargestellt. Man sieht beispielsweise, dass die Wahrscheinlichkeit P(W = T | S = T, R = F) = 0,9 ist (zweite Zeile in der unteren Tabelle). Es ist
P(W = F | S = T, R = F) = 1 – 0,9 = 0,1, da in jeder Zeile die Summe der Wahrscheinlichkeiten 1 ergeben muss.
Auf Grund der Kettenregel der Wahrscheinlichkeiten ist die vereinte Wahrscheinlichkeit aller Knoten:

$$P(C=c, S=s, R=r, W=w) = P(C=c) \cdot P(S=s|C=c) \cdot P(R=r|C=c, S=s) \cdot P(W|C=c, S=s, R=r) \quad .$$

Wenn man die bedingten Unabhängigkeiten berücksichtigt, kann dies geschrieben werden als:

$$P(C=c, S=s, R=r, W=w) = P(C=c) \cdot P(S=s|C=c) \cdot P(R=r|C=c) \cdot P(W=w|S=s, R=r) \quad .$$

Dies ist möglich, da im dritten Term R, bei Kenntnis von C, von S unabhängig ist und da im vierten Term bei Kenntnis von S und R, W unabhängig von C ist.

Die am häufigsten durchgeführte Aufgabe eines Bayes'schen Netzes ist die Inferenz der Wahrscheinlichkeiten. Betrachtet man das Beispiel und dort das Ereignis „Gras ist nass“, dann sieht man, dass es zwei mögliche Gründe dafür gibt. Zum einen „es regnet“ und zum anderen „der Rasensprenger ist an“. Um herauszufinden was wahrscheinlicher ist, kann die Bayes'sche Regel zur Berechnung der a posteriori Wahrscheinlichkeit benutzt werden:

$$P(S=1|W=1) = \frac{P(S=1,W=1)}{P(W=1)} = \frac{\sum_{c=0}^{1}\sum_{r=0}^{1} P(C=c, S=1, R=r, W=1)}{\sum_{c=0}^{1}\sum_{s=0}^{1}\sum_{r=0}^{1} P(C=c, S=1, R=r, W=1)} = \frac{0{,}2781}{0{,}6471} \approx 0{,}430 \quad ,$$

$$P(R=1|W=1) = \frac{P(R=1,W=1)}{P(W=1)} = \frac{\sum_{c=0}^{1}\sum_{s=0}^{1} P(C=c, S=s, R=1, W=1)}{\sum_{c=0}^{1}\sum_{s=0}^{1}\sum_{r=0}^{1} P(C=c, S=s, R=1, W=1)} = \frac{0{,}4581}{0{,}6471} \approx 0{,}708 \quad .$$

Daraus kann man ablesen, dass die Wahrscheinlichkeit höher ist, dass das Gras aufgrund von Regen nass ist als aufgrund des Rasensprengers [21].

Da man mit Bayes'schen Netzen eine vereinte Wahrscheinlichkeitsverteilung aller Variablen berechnet, gibt es zwei Vorgehensweisen: top-down oder bottom-up [19].

In diesem Beispiel konnte durch Inferenz aus der Tatsache, dass das Gras nass war, die wahrscheinlichste Ursache dafür ermittelt werden. Dieses Verfahren nennt man bottom-up, da man von Auswirkungen auf Ursachen schließt. Die Vorgehensweise in umgekehrter Richtung nennt man top-down, mit der man zum Beispiel aus der Tatsache, dass es wolkig ist, berechnen kann mit welcher Wahrscheinlichkeit das Gras nass ist.

Bayes'sche Netze ermöglichen eine ökonomische Darstellung von abhängigen Variablen und ihrer Beziehungen untereinander. Ein Knoten steht mit diversen anderen in Beziehung und es ist sonst keine Information von Nöten, um die gewünschten Werte zu berechnen. Mit Bayes'schen Netzen können auch unvollständige Datensätze ohne Schwierigkeit behandelt werden. Dies ist vor allem in unzuverlässigen Systemen oder in Systemen, in denen die Anzahl der Sensoren a priori nicht bekannt ist, wichtig. Um fehlende Variablen aus dem Rest zu berechnen, könnte z.B. ein Monte Carlo Algorithmus benutzt werden. Außerdem sind Bayes'sche Netze lernfähig bezüglich zufälliger oder unregelmäßiger Beziehungen. Viele andere Mechanismen aus dem Sektor der künstlichen Intelligenz, wie z.B. Kohonen Netze, sind nicht so flexibel und können einmal erlerntes Verhalten wieder anpassen [19]. Um Entscheidungen im Bayes'schen Netz treffen zu können, müssen Regeln für die Wahrscheinlichkeiten der einzelnen Ereignisse erfüllt werden. Solche Regeln können beispielsweise in Probabilistischer oder Fuzzy Logik verfasst werden.

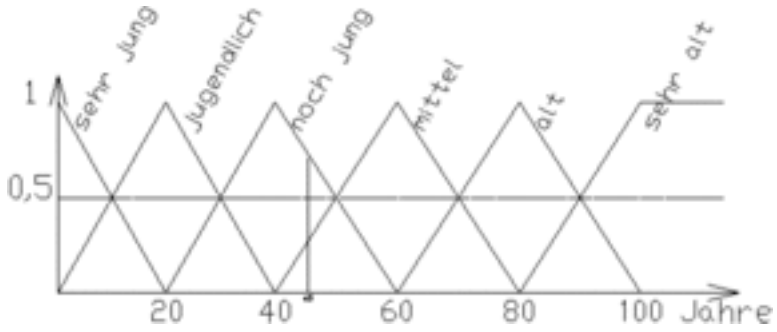
4.2 Probabilistische Logik

Probabilistische Logiken bauen auf der klassischen Bool'schen zweiwertigen Logik auf, d.h. es soll ermöglicht werden, bei Bestehen zweier Aussagen A und B, die beide die Wahrheitswerte „wahr“ oder „falsch“ annehmen können, den kombinierten Wahrheitswert von einem Ausdruck wie A ∧ B bestimmen zu können [22]. Bei Unsicherheit lassen sich mittels Probabilistischer Logik Aussagen treffen wie: „Die Wahrscheinlichkeit von A ist kleiner als ein Drittel.“, oder „Die Wahrscheinlichkeit von A ist mindestens zwei mal so hoch wie die von B.“, wobei A und B willkürliche Ereignisse sind. Mit Probabilistischer Logik können Regeln aufgestellt werden, mit denen es möglich ist, Entscheidungen über die Wahrscheinlichkeiten von Ereignissen in Form von Wahrscheinlichkeiten anderer verwandter Ereignisse zu treffen [18]. Sie werden dazu eingesetzt, Expertensysteme zu definieren. Da es aber bei mehreren Experten zu Widersprüchen kommen kann, gibt es Regeln, mit denen die einzelnen Experten bewertet werden. Es ist den Experten allerdings meistens nicht möglich, exakte Wahrscheinlichkeiten anzugeben. Aus diesem Grund geben sie ein Intervall der Form an [untere Schranke; obere Schranke], innerhalb dessen die Wahrscheinlichkeit für ein gewisses Ereignis liegt. Die Verarbeitung solcher Intervalle nennt man Worst-Case Analyse [22]. Expertensysteme werden eingesetzt, um bei Ungewissheit Entscheidungen treffen zu können, d.h. wenn weder der aktuelle noch der zukünftige Zustand gewiss sind. Expertensysteme sind regelbasiert. Regelbasiert bedeutet, dass an Hand, meist manuell eingepflegter Regeln, Entscheidungen für bestimmte Ereignisse getroffen werden [23]. Mittels dieser Regeln können auch Entscheidungen über die Wahrscheinlichkeit eines Ereignisses, in Form von Wahrscheinlichkeiten verwandter Ereignisse getroffen werden [18]. Gegeben sei ein Experte, der unvollständiges Wissen hat. Folglich besteht Ungewissheit darüber, ob die beiden Aussagen A und B „wahr“ oder „falsch“ sind. Es ist jedoch möglich eine Aussage darüber zu treffen, mit welcher Wahrscheinlichkeit diese Aussagen „wahr“ oder „falsch“ sind. Es soll folglich mittels der Probabilistischen Logik eine Entscheidung darüber getroffen werden, wie wahrscheinlich der Ausdruck A ∧ B ist. Es fällt auf, dass der Unterschied zwischen Probabilistischer Logik und klassischer Logik darin besteht, dass, falls bei der Probabilistischen Logik alle Wahrscheinlichkeiten entweder 0 oder 1 sind, wieder eine klassische Logik vorliegt.

Probabilistische Logiken sind folglich eine Erweiterung der klassischen Logik [22].

4.3 Fuzzy Logik

Die Fuzzy Logik ist eine Verallgemeinerung der klassischen zweiwertigen Bool'schen Logik. Wie das Wort Fuzzy schon impliziert (fuzzy = ungenau, verschwommen, unscharf), können mittels der Fuzzy Logik auch „schwammige“ Begriffe behandelt werden. Sie basiert auf den so genannten Fuzzy Mengen. Kernaussage dieser Fuzzy Mengen ist, dass es Elemente gibt, bei denen es schwer zu sagen ist, ob sie zu einer Menge gehören oder nicht, d.h. es können auch Aussagen getroffen werden, wie: „Ein Element gehört *ein wenig* zu einer bestimmten Menge.“ Der Grad der Zugehörigkeit wird durch die Zugehörigkeitsfunktion beschrieben, die den Elementen einen Wert zwischen 0 und 1 zuweist. Die Zugehörigkeitsfunktion wird am Beispiel des Alters einer Person verdeutlicht. Die Zugehörigkeitsfunktion bestehe hier aus mehreren Dreiecksfunktionen für die einzelnen Altersbereiche.



Zugehörigkeitsfunktion für einzelne Altersbereiche

Eine Person mit 45 Jahren wäre dann, wie man der Zugehörigkeitsfunktion entnehmen kann, zu 75% „noch jung“ und zu 25% „mittleren Alters“. Solche Zugehörigkeitsfunktionen ergeben sich meist empirisch [24].

4.4 Vergleich der Entscheidungsmechanismen

Der Vergleich zwischen Bayes'schen Netzen, Probabilistischer Logik und Fuzzy Logik, soll an Hand von Gaia, einem Richtmodell für eine Pervasive Computing Infrastruktur erbracht werden. Gaia ermöglicht es Applikationen und Diensten bei Unsicherheit Entscheidungen zu treffen, indem es Mechanismen wie Bayes'sche Netze, Probabilistische Logik und Fuzzy Logik zur Verfügung stellt. Dieses Modell basiert auf Prädikaten verknüpft mit Confidence Werten, die die Kontexte repräsentieren sollen. Dabei stellt Gaia jede Information, deren Wahrheitswert potentiell unsicher ist, als ein Prädikat dar. Die Prädikate werden in diesem Zusammenhang so definiert, dass ihre Argumente eine Subjekt-Objekt Form (ContextType(<Subjekt>, <Objekt>)) oder eine Subjekt-Verb-Objekt Form (ContextType(<Subjekt>, <Verb>, >Objekt>)) haben.

Beispiele:

location(jeff, in, room3105)
activity(room3102, meeting)
light(room3220, dim)

Manche Kontexte, wie „location“ und „activity“, werden als unsicher betrachtet. Unsicherheit ist hier so modelliert, dass ein Confidence Wert zwischen 0 und 1 an die Prädikate angehängt wird. Dieser Wert repräsentiert die Wahrscheinlichkeit (bei einem probabilistischen Ansatz) oder den Zugehörigkeitswert (im Falle von Fuzzy Logik) eines Ereignisses gemäß eines Kontext-Prädikats. Zum Beispiel wird eine 50 prozentige Wahrscheinlichkeit, dass sich Carol im Zimmer 3233 befindet, dargestellt als prob(location(carol, in, room3233)) = 0,5. Mittels Belief Network von Microsoft (MSBN), einem Toolkit zum Erstellen von Bayes'schen Netzen, können die Gaia-Entwickler Bayes'sche Netze aufstellen, die kausale Abhängigkeiten zwischen den Ereignissen repräsentieren. Jeder Knoten des Netzes ist mit einem bestimmten Kontextprädikat verbunden. Bei Gaia werden Kontexte höherer Stufe von Kontexten niederer Stufe abgeleitet, d.h. wenn auf Sensorebene schon Unsicherheit besteht, pflanzt sich diese bis auf höhere Stufen fort. Es müssten also Wahrscheinlichkeiten oder Confidence-Werte verschiedener Sensoren vereint werden, um das Maß der Unsicherheit des abgeleiteten Kontexts ermitteln zu können. Dazu können Probabilistische Logik oder Bayes'sche Ansätze verwendet werden. In [18] wurde ein Bayes'sches Netz trainiert und mittels dieses Netzes konnte in 84% der Fälle aus den Sensordaten (Lichtstatus oder Anzahl anwesender Leute) die gerade durchgeführte Aktivität (Meeting oder Präsentation), die in einem Raum stattfand, korrekt ermittelt werden. Ein Grund für eine derart gute Trefferquote könnte allerdings sein, dass in dem beobachteten Raum nur eine eingeschränkte Anzahl gut unterscheidbarer Aktivitäten stattfand.
Applikationen in Gaia benutzen die Wahrscheinlichkeiten verschiedener Kontexte, um ihr Verhalten anzupassen. Die

Applikationen erhalten Kontextprädikate mit ihren Wahrscheinlichkeiten von so genannten Kontextprovidern und Kontextsynthesizern, und sie entscheiden über unsichere Kontextinformationen, indem sie Fuzzy Logik und Bayes'sche Inferenz Mechanismen benutzen. Falls sie nicht direkt auf die Wahrscheinlichkeiten der einzelnen Kontextprädikate Bezug nehmen wollen, können sie annehmen, dass das Prädikat mit der größten Wahrscheinlichkeit wahr ist und die restlichen falsch sind.
Access Control Entscheidungen in Gaia werden mittels Fuzzy Reasoning abgewickelt. Access Control Verfahren für verschiedenste Applikationen und Dienste basieren auf linearen Abweichungen der Confidence Werte von Kontextinformationen. Gaia gewährt einem Benutzer Zugang zu einer Resource, wenn der Benutzer hinreichend genau identifiziert wurde und wenn der Kontext mit einer bestimmten Genauigkeit abgetastet wurde. Es folgt ein Beispiel für eine Zugangskontrollregel, die in Prolog geschrieben wurde:

```
canAccess(P, display):-
      confidenceLevel(authenticated(P),C), C > 0.7,
      Prob(activity(2401, cs 101 presentation), Y), Y > 0.8,
      possessRole(P, presenter)
```

Dieses Beispiel sagt aus, dass ein Benutzer P Zugang auf das Display hat, falls P mit einem Confidence Wert von mindestens 0.7 identifiziert wurde und falls die Aktivität im Raum mit einer Wahrscheinlichkeit von mindestens 0.8 ein Vortrag und P der Vortragende ist. Bei Gaia wird auch zur Problembehandlung ein Bayes'sches Netz benutzt. Vorausgesetzt es existieren Anzeichen, dass das Netz ableitet, dass die Wahrscheinlichkeiten verschiedener Gaiadienste, Applikationen, Geräte oder anderer Ressourcen falsch sind. Lernen mit Bayes'schen Netzen und Regeln, geschrieben in Probabilistischer oder Fuzzy Logik, sind nützlich für unterschiedliche Szenarien. Bayes'sche Netze sind nützlich, um die Wahrscheinlichkeitsverteilungen von Ereignissen zu erlernen und um Entscheidungen über kausale Zusammenhänge zwischen Beobachtungen und Systemzustand treffen zu können. Sie müssen selbstverständlich trainiert werden bevor sie eingesetzt werden können. Da sie aber flexibel und leicht zu trainieren sind, können sie sich leicht an sich ändernde Umstände anpassen. Probabilistische Logik ist nützlich, wenn man detailiertes Wissen über die Wahrscheinlichkeiten einzelner Ereignisse besitzt. Fuzzy Logik kann dagegen eingesetzt werden, wenn man schwammige Begriffe darstellen möchte. Sowohl Probabilistische als auch Fuzzy Logik eignen sich, wenn es schwer ist Daten zu beschaffen, um ein Bayes'sches Netz zu trainieren.

5. Allgemeines Fazit

Es gibt eine große Anzahl an Algorithmen, die für verschiedene Situationen verwendet werden können. Mit dem Bayes Klassifikator können, unter Zuhilfenahme des Bayes Theorems, Klassenzugehörigkeiten gut bestimmt werden. Mit dieser Klassifikationsmethode erzielt man gute Ergebnisse, solange die einzelnen Attribute nicht zu stark korrelieren. Der Gauß Klassifikator ist aufgrund der Normalverteilungsannahme nicht besonders gut geeignet zur Klassifikation von Sensordaten. Bei Parzen Window und K Nearest Neighbors handelt es sich um einfache Klassifikatoren. Sie sind allerdings nicht besonders effizient. Diese Klassifikatoren kommen dennoch recht häufig zum Einsatz, da keine Trainingsphase benötigt wird. Die Klassifikation mittels Distanzen ist ebenfalls ein recht einfaches Verfahren. Da jedoch meist nur die Klassenmittelpunkte in Betracht gezogen werden, sind diese Verfahren für stark variierende Merkmalsklassen weniger geeignet. In diesem Fall wäre der Einsatz der Mahalanobis Distanzbildung vorzuziehen, da dort auch Kovarianzen berücksichtigt werden. Hidden Markov Modelle werden benutzt, um beispielsweise bei der Spracherkennung für ein Eingabesignal aus einer Datenbank die passenden Phoneme zu finden. Dazu wird das akkustische Modell eines Phonems in verschiedene Teilstücke zerlegt. Zur Spracherkennung wird dann das Eingabesignal mit den gespeicherten Teilstücken verglichen. Es wurde auch versucht für die Problematik der Spracherkennung neuronale Netze einzusetzen. Es gab zwar durchaus erfolgversprechende Ergebnisse, die Entwicklung wurde jedoch zugunsten der Hidden Markov Modelle wieder aufgegeben. Es wird allerdings die Möglichkeit genutzt, die Ausgabe eines neuronalen Netzes als Eingabe für ein Hidden Markov Modell, so zu sagen als eine Art Vorklassifikator, zu benutzen. Der Vorteil dieser Methode ist, dass auch Daten, die kurz vor oder kurz nach dem gerade bearbeiteten Zeitraum behandelt wurden, genutzt werden können. So können Datenklassifikation und kontextsensitive Zusammensetzung voneinander getrennt werden. Neuronale Netze sind vor allem dann sinnvoll einsetzbar, wenn a priori kein Wissen über ein zu lösendes Problem vorliegt. Dagegen haben Kohonen Self-Organizing Maps den Vorteil, dass sie zwar selbstorganisiert arbeiten, jedoch ist einmal erlerntes Verhalten kaum noch änderbar. Sie bieten aber eine gute Darstellbarkeit ihrer Vorgehensweise. Bayes'sche Netze, Probabilistische Logik und Fuzzy Logik können gut eingesetzt werden, um bei Unsicherheit Klassifikationsentscheidungen zu treffen. Dabei können Probabilistische Logik und Fuzzy Logik benutzt werden, um Regeln aufzustellen oder, wenn es schwer ist Daten zu beschaffen, um ein Bayes'sches Netz zu trainieren.
Abschließend lässt sich demnach feststellen, dass es keinen Kontextalgorithmus gibt, der in allen möglichen Szenarien gleich gut einsetzbar ist.

Literaturverzeichnis:

[1] Henryk Plötz. „SV2 – Mustererkennung, Vorlesungsnotiz“ http://www.informatik.hu-berlin.de/~ploetz/SV2/VL.pdf 09.10.2004

[2] Ozan Cakmakci, Joelle Coutaz, Kristof Van Laerhoven, Hans-Werner Gellersen . „Context Awareness in Systems with Limited Resources“. In Proceedings of AIMS-2002. http://citeseer.ist.psu.edu/cakmakci02context.html

[3] Thorben Maier, Stefan Schmidt, Marco Priebe. „Pro-Seminar Neuronale Netze“. http://informatik.hu-berlin.de/~priebe/backprop.pdf 10.02.2004

[4] http://de.wikipedia.org/wiki/Dichtefunktion. Januar 2006

[5] http://de.wikipedia.org/wiki/Mittelwert. Januar 2006

[6] http://de.wikipedia.org/wiki/Kovarianzmatrix. Januar 2006

[7] Vorlesung Kognitive Systeme Sommersemester 2003. Prof. Waibel und Prof. Dillmann

[8] http://en.wikipedia.org/wiki/Parzen_window. Januar 2006

[9] Kristof Van Laerhoven. http://www.comp.lancs.ac.uk/~kristof/. Januar 2006

[10] http://de.wikipedia.org/wiki/Normierter_Raum. Januar 2006

[11] Kristof Van Laerhoven, Albrecht Schmidt, Hans-Werner Gellersen. „Multi-Sensor Context Aware Clothing“. http://www.comp.lancs.ac.uk/~kristof/old/papers/iswc_2002.pdf

[12] Stefan Eickeler. „Automatische Bildfolgenanalyse mit statistischen Mustererkennungsverfahren“. 05.11.2001

[13] http://www.nue.tu-berlin.de/forschung/projekte/mpeg7/Vorlesung/VL9.pdf. Januar 2006

[14] Ozan Cakmakci, Joelle Coutaz, Kristof Van Laerhoven, Hans-Werner Gellersen. „Context Awareness in Systems with Limited Resources“. In Proceedings of AIMS-2002. http://citeseer.ist.psu.edu/cakmakci02context.html, http://www.equator.ac.uk/var/uploads/aims_2002.pdf

[15] Günter Daniel Rey, Fabian Beck. „Neuronale Netze, Eine Einführung“ http://neuronalesnetz.de/. Januar 2006

[16] Elaine Catterall, Kristof Van Laeroven, Martin Strohbach. „Self-Organization in Ad Hoc Sensor Networks: An Empirical Study“. Artificial Life VIII, Abbass, Bedau (eds) (MIT Press) 2002. pp 260-263

[17] Kristof Van Laerhoven, Ozan Cakmakci. „What Shall We Teach Our Pants?“. IEEE Press, 2000

[18] Anand Ranganathan, Jalal Al-Muhtadi, Roy H. Campbell. „Reasoning about Uncertain Contexts in Pervasive Computing Environments“. IEEE Pervasive Computing Magazine, volume 3, no. 2, April-June 2004.

[19] Michal Rój. „Data Mining Techniques in Ubiquitous Computing“. IWCIT 2003

[20] Volker Tresp. „Bayes'sche Netze: Konstruktion, Inferenz, Lernen und Kausalität“ http://www.dbs.ifi.lmu.de/Lehre/MaschLernen/Vorlesung24Juni05_V2.pdf. Vorlesung 24 Juni 2005

[21] K. Murphy. „A Brief Introdution to to Graphical Models and Baysian Networks“. http://www.ai.mit.edu/~murphyk/bayes/bnintro.html. Februar 2003

[22] Björn Winterberg. „Probabilistische- und Fuzzy Logiken“. Seminar Nichtklassische Logiken von Bernd Farwer, http://homepage.mac.com/farwer/nkl0506/Folien_attachments/Probabilistische-_und_Fuzzy-Logiken.pdf

[23] http://de.wikipedia.org/wiki/Expertensystem. Januar 2006

[24] http://de.wikipedia.org/wiki/Fuzzy-Logik Januar 2006

Seminar

Ubiquiräre Informationssysteme
WS 05/06

Thema:
Performanceanalyse kontextsensitiver Anwendungen

Erstellt von:

Thomas Ring

Matrikelnummer: 1032055

Bertreuer:

Tobias Zimmer

1. Einführung

Kontextsensitive Anwendungen können die Mensch-Computer-Interaktion verbessern, in dem sie Informationen über den Benutzer und dessen Umwelt auswerten und eigenständig darauf reagieren. Ein Bereich des Benutzerkontextes ist die physische Aktivität, zu dem z.B. alltägliche Bewegungen wie Gehen, Stehen und Sitzen usw. zählen. Um diese Aktivitäten zu erkennen werden Sensorsysteme eingesetzt, die Daten sammeln. Diese Daten werden dann ausgewertet und mittels Klassifikatoren beurteilt. Letztere werden zur Initialisierung mit Daten der zu erkennenden Bewegungen trainiert. Die Trainingsdaten werden dazu von einer Person oder mehreren Personen über verschiedene Verfahren, auf die in Kapitel 2 eingegangen wird, aufgezeichnet.
Auf dem Gebiet der Aktivitätenerkennung gibt es einen Reihe von Studien. In Kapitel 4 stelle ich einige vor und gehe dabei auf die Vorgehensweise solcher Studien ein. Anhand der Ergebnisse der Studien wird der Leistungsstand des Bereichs Aktivitätenerkennung aufgezeigt und verglichen. Um die Leistung zu bewerten werden Standardformeln und sogenannte Confusion Matrizen eingesetzt, die Rückschlüsse anhand der Erkennungsraten zulassen. Diese werden in Kapitel 3 vorgestellt. In Kapitel 5 sind die Ergebnisse zusammengefasst und sofern möglich vergleichend bzgl. der Ansätze und Ergebnisse gegenübergestellt.

2. Verfahren der Trainings- bzw Testdatengewinnung

Um Trainings- bzw. Testdaten zu erhalten wird auf verschiedene Methoden zurückgegriffen. Mit diesen Daten kann dann das System trainiert und getestet werden.
Dies kann durch eine direkte Überwachung der Testperson durch ausgebildetes Personal geschehen. Obwohl die direkte Überwachung gute Ergebnisse der Messungen liefert, ist sie sehr kosten- und zeitintensiv. Außerdem wirkt sich diese Methode nicht selten auf das Verhalten einer Testperson aus, wodurch Messungen verfälscht werden, weil durch den psychischen Druck der Beobachtung, aber auch durch die übermäßige Konzentration auf sich und die gerade ausgeführte Bewegung, die Körperhaltung und -bewegungen nicht den natürlichen entsprechen. Deswegen sind direkte Überwachungen für viele Aufgaben nicht geeignet.
Eine zweite Methode sind die sogenannten self-reports, wobei hier zwischen recall surveys und time diaries unterschieden wird. Die Versuchsperson muss hier selbständig die erforderlichen Aufzeichnungen vornehmen.
Bei den recall surveys füllt die Versuchsperson einen Fragekatalog mit Fragen zu einem zeitlich zurückliegenden Ereignis aus. Hierbei treten häufig Fehler auf, da sich die Versuchsperson an einen vergangenen Zeitpunkt zurückerinnern muss. Um diese Fehler zu minimieren können time-diaries verwendet werden, denn die Versuchsperson ist angehalten in bestimmten Zeitintervallen oder Zeitpunkten Notizen zu machen. Zur Vereinfachung des self-reporting werden auch Diktiergeräte, Foto- und Videokameras eingesetzt.
Als letztes sei noch die experience sampling method (ESM), die auch als ecological momentary assessment (EMA) bekannt ist, erwähnt. Für das ESM/EMA wird die Versuchsperson zu bestimmten Zeitpunkten daran erinnert Aufzeichnungen zu machen. Bei hohen Aufzeichnungsraten kann das die Versuchsperson empfindlich stören und kontraproduktiv auf das Ergebnis wirken.

3. Bewertung der Leistung eines Klassifikationssystems

Die Leistung eines Klassifikationsalgorithmus, kann aus einer sogenannten Confusion Matrix abgelesen werden, die bei unüberwachtem Lernen auch Matching Matrix genannt wird. Sie enthält Informationen über die wirkliche und die vorhergesagte Klassifikation eines Klassifikatorsystems. So ein Klassifikatorsystem versucht aus bestimmten Merkmalen auf nicht direkt beobachtbare Eigenschaften zu schließen. Dabei kann das System richtig liegen, aber auch eine falsche Prognose abgeben, d.h. dass die Klassifizierung des Systems nicht der tatsächlichen entspricht oder erst gar keine Klassifizierung möglich war. Genau dieser Sachverhalt wird in der Confusion Matrix dargestellt.
Allgemein ist die Confusion Matrix für n Klassen eine n$\times$n-Matrix Z, $Z = (z_{ij})$ mit i, j aus $N = \{1, \ldots, n\}$. Die Matrixelemente z_{ij} enthalten die Anzahl der vorhergesagten Klassifizierungen zur Klasse j bei tatsächlicher Klassifizierung zur Klasse i. Also entsprechen die Hauptdiagonalelemente der Matrix z_{ij}, mit i = j, den richtigen Klassifizierungen des Systems.
Für zwei Klassen ist die Confusion Matrix in Tabelle 3.1 beispielhaft dargestellt. Häufig werden die Einträge der Matrix auch mit englischen Namen bezeichnet. TP (*true positive*), für das Element z_{11}, gibt die Anzahl der richtigen Vorhersagen zur Klasse 1 an, TN (*true negative*), für das Element z_{22}, die zu Klasse 2. FP (*false positive*), für das Element z_{21}, und FN (*false negative*), für das Element z_{12}, gibt die Anzahl der falschen Vorhersagen zur Klasse 1 bzw Klasse 2 an.

		Vorhersage	
		positiv	negativ
Wirklichkeit	positiv	z_{11}	z_{12}
	negativ	z_{21}	z_{22}

Tabelle 3.1. Beispielhafte Confusion Matrix mit zwei Klassen.

Um die Leistung zu messen und zu vergleichen werden definierte Standardformeln herangezogen. Diese sind u.a. die Genauigkeit (AC von *accuracy*), die TPR (*true positive rate*), die FPR (*false positive rate*) und die Präzission PPV (*positive predicted value*).
Die TPR gibt das Verhältnis der richtigen Klassifizierungen zu allen Klassifizierungen der Klasse i an und ist auch als *recall* oder *sensitivity* bekannt.

$$TPR_i = \frac{TP_i}{TP_i + FN_i},$$

wobei

$$TP_i = z_{ii}$$

die Anzahl der TP der Klasse i und

$$FN_i = \sum_{\substack{j \in N \\ j \neq i}} z_{ij}$$

ist die Anzahl der FN der Klasse i.
Das Mittel über die TPR aller Klassen ist allgemein als Genauigkeit bekannt (AC):

$$AC = \frac{\sum_{i=1}^{n} TPR_i}{n},$$

Die FPR gibt den Anteil an, der nicht zu einer Klasse gehört, aber fälschlicher Weise als diese Klassifiziert wurde und wird auch als *false alarm rate* bezeichnet. Für eine Klasse i ist das die Anzahl der falschen Klassifizierungen zu dieser Klasse geteilt durch die Anzahl aller Vorkommnisse, die nicht zu der Klasse gehören.

$$FPR_i = \frac{FP_i}{FP_i + TN_i},$$

wobei

$$FP_i = \sum_{\substack{j \in N \\ j \neq i}} z_{ji}$$

die Anzahl der FP der Klasse i und

$$TN_i = \sum_{\substack{(k,j) \in N \times N \\ k \neq i \wedge j \neq i}} z_{kj}$$

die Anzahl der TN der Klasse i ist.
Die Gesamt-FPR, bei n Klassen, ist gegeben durch:

$$FPR_N = \frac{\sum_{i \in N} (FPR_i \cdot \sum_{j \in N} z_{ij})}{\sum_{(i,j) \in N \times N} z_{ij}}$$

Der PPV gibt den prozentualen Anteil der korrekt klassifizierten zu den falsch klassifizierten Vorkommnissen einer Klasse i an und ist definiert als:

$$PPV_i = \frac{TP_i}{TP_i + FP_i},$$

wobei TP_i und FP_i wie oben definiert sind.
Der Gesamt-PPV über alle Klassen berechnet sich dann aus:

$$PPV_N = \frac{\sum_{i \in N} (PPV_i \cdot \sum_{j \in N} z_{ij})}{\sum_{(i,j) \in N \times N} z_{ij}}.$$

4. Analyse und Übersicht

4.1 Aktivitätenerkennung aus Beschleunigungsdaten

Ein Bestandteil des Benutzerkontexts ist die physische Bewegung. Um diese Bewegungen zu beschreiben werden Beschleunigungsdaten bzw. eine Kombination aus diesen und anderen Daten verwendet. Hierbei ist jetzt von Interesse welche Erkennungsraten bei der Erkennung der physischen Bewegung eines Menschen erreicht werden können. Diese hängen davon ab, wie und wo die Daten gesammelt wurden. Auch die Frage wie diese Ergebnisse auf reale Bedingungen übertragbar sind, ist von Interesse, da die Forschungsergebnisse aus zum Teil eingeschränkten Laborversuchen stammen. Manche aber auch in einer natürlichen Umgebung, außerhalb eines Labors, doch mit der Einschränkung weniger Datensätzen einer Versuchsperson, ihre Forschung betreiben [17]. Als letztes gibt es dann noch Arbeiten, die natürliche Daten von mehreren Versuchspersonen sammeln, sich aber auf einen kleinen Ausschnitt von alltäglichen Bewegungsabläufen beschränken [10, 7].

4.1.1 Studie von L. Bao, S.Intille [4]

Beispielhaft für diese Forschung stelle ich kurz die Studie von Ling Bao und Stephen S. Intille [4] vor. Anschließend führe ich Ergebnisse von anderen, früheren Forschungen an.
In dieser Studie wurden Algorithmen entwickelt und getestet, die physische Bewegung anhand der Daten von fünf biachsialen Beschleunigungsmesser erkennen. Die Beschleunigungsmesser werden gleichzeitig an verschiedenen Köperpartien getragen. Siehe auch Abbildung 4.1.

4.1.1.1 Einleitung

Bei der Studie trugen die Probanden fünf biachsiale Beschleunigungsmesser während sie eine Reihe von alltäglichen Bewegungen ausführten, wie zum Beispiel Gehen, Fahrrad fahren usw. Insgesamt wurden zwanzig Aktivitäten untersucht. Siehe dazu auch Tabelle 4.2.
Um die Daten zu sammeln wurden zwei verschiedene Protokolle angewandt: Eine semi-natürliche, benutzergetriebene Datensammlung und eine spezifische Aktivitätendatensammlung.
Realistische Trainingsdaten erhält man von Versuchspersonen, die ihren normalen, alltäglichen Tätigkeiten nachgehen. Dies bedarf aber der direkten Überwachung durch einen Forscher, Aufzeichnung durch den Probanden selbst oder durch die Benutzung der ESM um die Aktivitäten zu Kennzeichnen, damit anschließend ein Algorithmus trainiert und getestet werden kann.
Bei der semi-natürlichen, benutzergetriebenen Datensammlung mußte die Versuchsperson, die auf einer Liste aufgeführten Aktivitäten ausführen. Die einzelnen Aufgaben wurden in einer Art Hindernisparcour bewältigt, damit die

Versuchspersonen möglichst nicht zu sehr auf die Datensammlung und dadurch auf die momentane Aktivität achteten. Deswegen wurden die Versuchspersonen auch nicht beobachtet. Alle Versuchspersonen mußten die Anfangs- und Endzeiten der jeweiligen Aufgabe notieren, um später die notwendigen Daten zu extrahieren. Ein Ausschnitt aus dem Datensatz mit den sogenannten self-report-Marken ist in Abbildung 4.2 exemplarisch zu sehen.
Mit der spezifischen Aktivitätendatensammlung wurden die Aktivitäten dann ein zweites Mal aufgezeichnet. Nun unter kontrollierteren Bedingungen. Die Bewegungen und Aktivitäten wurden klar definiert. Unter [3] sind alle 20 Aktivitäten beschrieben.

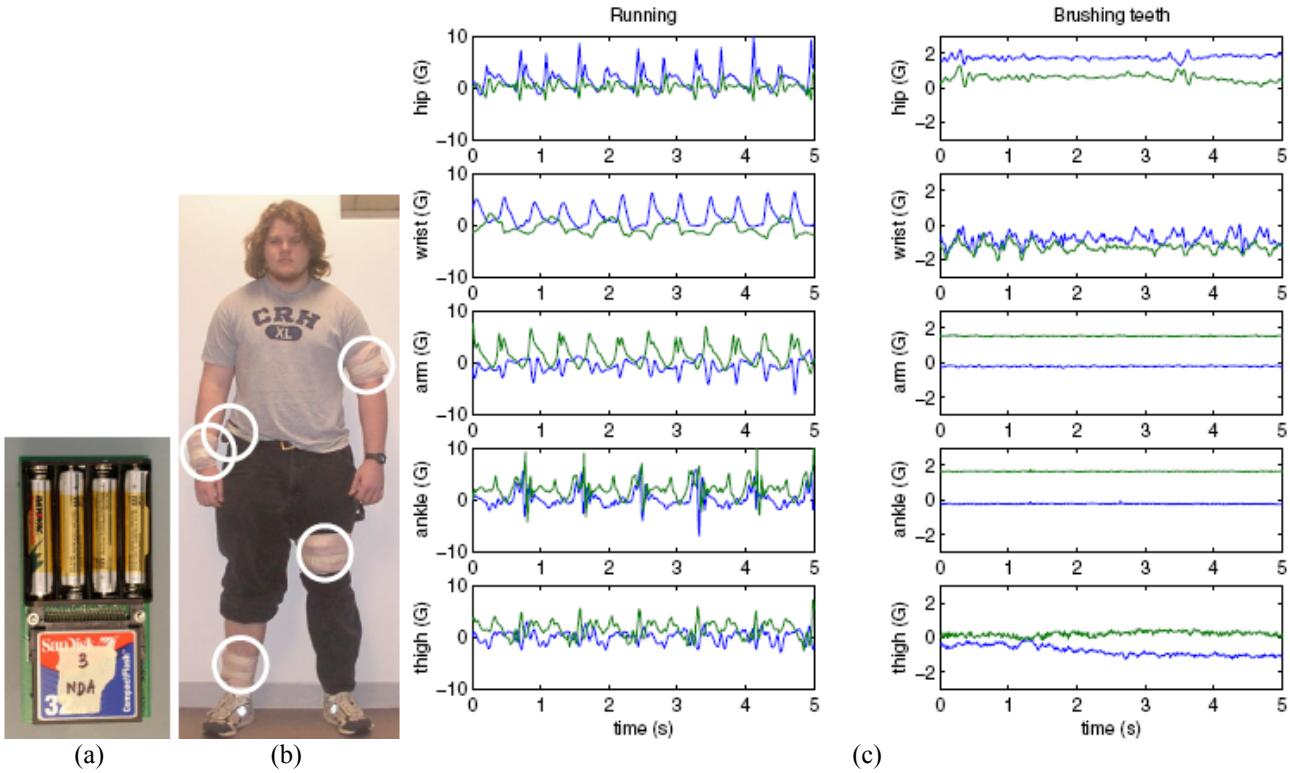


Abb. 4.1. (a) Hoarder data collection board. (b) Hoarder boards wurden an den vier Gliedmaßen wie hier gezeigt angebracht und eines an der rechten Hüfte. (c) Beschleunigungssignale der fünf biachsialen Accelerometer für Rennen und Zähne putzen. [4]

4.1.1.2 Merkmalberechnung

Die Merkmale wurden mit 512 Stichprobenfenstern der Beschleunigungsdaten, mit 256 überlappenden Stichproben bei aufeinanderfolgenden Fenstern extrahiert. Jedes Fenster war 6,7 Sekunden groß. Das Mittel, die Energie, Entropy und die Korrelation wurden aus den Signalen der Aktivitätenerkennung, innerhalb der Fenster, extrahiert. Der Gleichstromanteil wurde für den durchschnittlichen Beschleunigungswert des Signals innerhalb eines Fensters herangezogen. Die Energie wurde aus der Summe der Wurzel der diskreten FFT des Signals berechnet. Diese Summe wurde zur Normalisierung durch die Fensterlänge dividiert.
Die Entropy des Frequenzbereichs berechnet sich aus der normalisierten Entropy der Information der diskreten FFT des Signals. Dieses Merkmal unterstützt die Unterscheidungsfähigkeit von Aktivitäten mit ähnlichen Energiewerten.
Die Korrelation wird aus zwei verschiedenen Beschleunigungsachsen berechnet. Genauer gesagt ist die Korellation das Skalarprodukt von zwei Beschleunigungsachsen geteilt durch die Fenstergröße von 512. Merkmale, die die Korrelation oder die Beschleunigung zweier Achsen messen, können die Aktivitätenerkennung bei Bewegungen mit mehreren Körperteilen verbessern.
Die Abbildung 4.3 zeigt den Vorteil der oben beschriebenen Merkmalextrahierung. Dennoch sei gesagt, dass bei Aktivitäten, die zwar sehr ähnliche Beschleunigungswerte aufweisen, aber nur unwesentlich von ihnen charakterisiert werden, eine klare Erkennung nur sehr ungenau möglich ist; z.B. Fernsehen und Sitzen, die sehr ähnliche, wenn nicht gleiche, Beschleunigungswerte aufweisen. Also ist es nicht möglich sie nur anhand der Beschleunigungsdaten und der daraus extrahierten Merkmale zu unterscheiden.

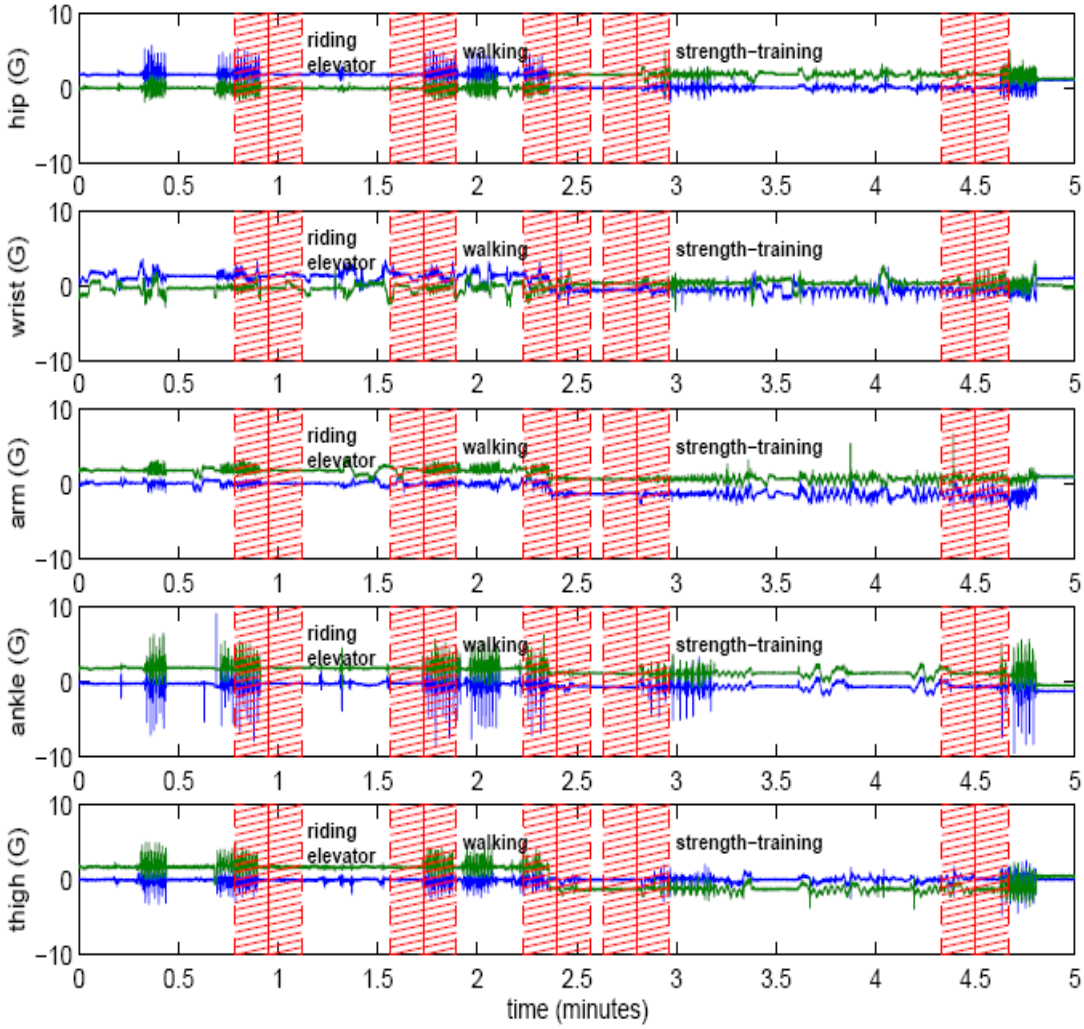


Abb. 4.2. Beschleunigungsdaten einer Versuchsperson mit Self-Report-Marken [3]

4.1.1.3 Ergebnisse

Aus den Beschleunigungsdaten wurden das Mittel, die Energie, Entropy und die Korrelation extrahiert. Eine Reihe von Klassifikatoren wurden verwendet, um die Aktivitätenerkennung anhand der extrahierten Merkmale durchzuführen.
Zum einen wurden Entscheidungstabellen eingesetzt. Aber auch instance-based learning (IBL oder nearest neighbor) kam zum Einsatz. Dieser Algorithmus gehört zu den Methoden zur Klassifizierung von Phänomenen, die beobachtbare Merkmale besitzen. Mit den Merkmalen wird ein Multidimensionaler Merkmalsraum aufgespannt. Die Trainingssätze werden in den Merkmalsraum abgebildet und anhand a priori bekannter Klassen bewertet.
Der C4.5 Algorithmus, der der Entscheidungsfindung dient, wurde auch verwendet und wird bei Entscheidungsbäumen eingesetzt. Es können eine beliebige Anzahl von Verzweigungen eingebaut werden. Das hat zur Folge, dass nach der ersten Klassifizierung die nachfolgenden Aufsplittungen weniger bedeutungsvoll sind.
Der naive Bayes Klassifikator ist ein auf der statischen Entscheidungstheorie basierendes Klassifikationsverfahren. Mittels des naiven Bayes-Klassifikators ist es möglich, die Zugehörigkeit eines Objekts zu einer Klasse zu bestimmen.
Diese Klassifikatoren wurden auf zwei verschiedene Arten trainiert und getestet.
Erst wurden die Klassifikatoren mit der spezifischen Aktivitätsdatensammlung einer Versuchsperson trainiert und anschließend die Erkennung mit der Versuchsperson auf dem Hindernisparcour getestet. Dies wurde für jede Versuchsperson wiederholt. Danach wurden die Klassifikatoren mit den speziellen Aktivitätsdatensammlung und den Daten des Hindernisparcours von jeder Versuchsperson bis auf eine trainiert und mit der nicht berücksichtigten Versuchperson auf dem Hindernisparcour getestet. Das sogenannte Leave-one-subject-out Training wurde wieder für alle Versuchspersonen wiederholt. Die durchschnittliche Abweichung und die Standardabweichung der korrekten Erkennung der Klassifikatoren sind in Tabelle 4.1 zusammengefasst.
Bei dieser Studie waren die Erkennungsraten bei den Entscheidungstabellen am höchsten, was durch vorangehnde Arbeiten bestätigt wird bei den entscheidungsbasierte Algorithmen Sitzen, Liegen, Stehen und andere Bewegungen mit

einer Genauigkeit von 89,30% erkennen [1]. Nearest neighbor war der zweit beste Algorithmus in dieser Studie. Dies wird auch wieder durch andere Sutdien belegt, die eine Genauigkeit von über 90% bei der Erkennug von Körperbewegung und Körperhaltung hatten [14, 10]. Bei dem C4.5 Klassifiaktor war zu sehen, dass hier der Vergleich der Hauptwerte zu einer guten Erkennungsrate führte. Wobei für den naiven Bayes Klassifikator genau wegen der Annaheme der Unabhängigkeit der Werte eine eher schwache Leistung resultierte. Desweiteren ist zu sagen, dass Bayes Algorithmen anscheinend mehr Daten benötigen um eine genau Werteverteilung zu berechnen, die zu einem ausreichenden Ergebnis führen.
Außerdem waren die Erkennungsraten wesentlich höher bei dem Leave-one-subject-out Prozess und zwar für alle Algorithmen. Dies kommt zum einen daher, dass die Klassifikatoren mit mehr Datensätzen trainiert wurden. Das führte zu allgmeineren und robusteren Klassifikatoren für die Aktivitäten. Aber es läßt auch den Schluss zu dass trotz der individuellen Beschleunigungsdaten eine große Übereinstimmung in den Bewegungsmustern besteht.

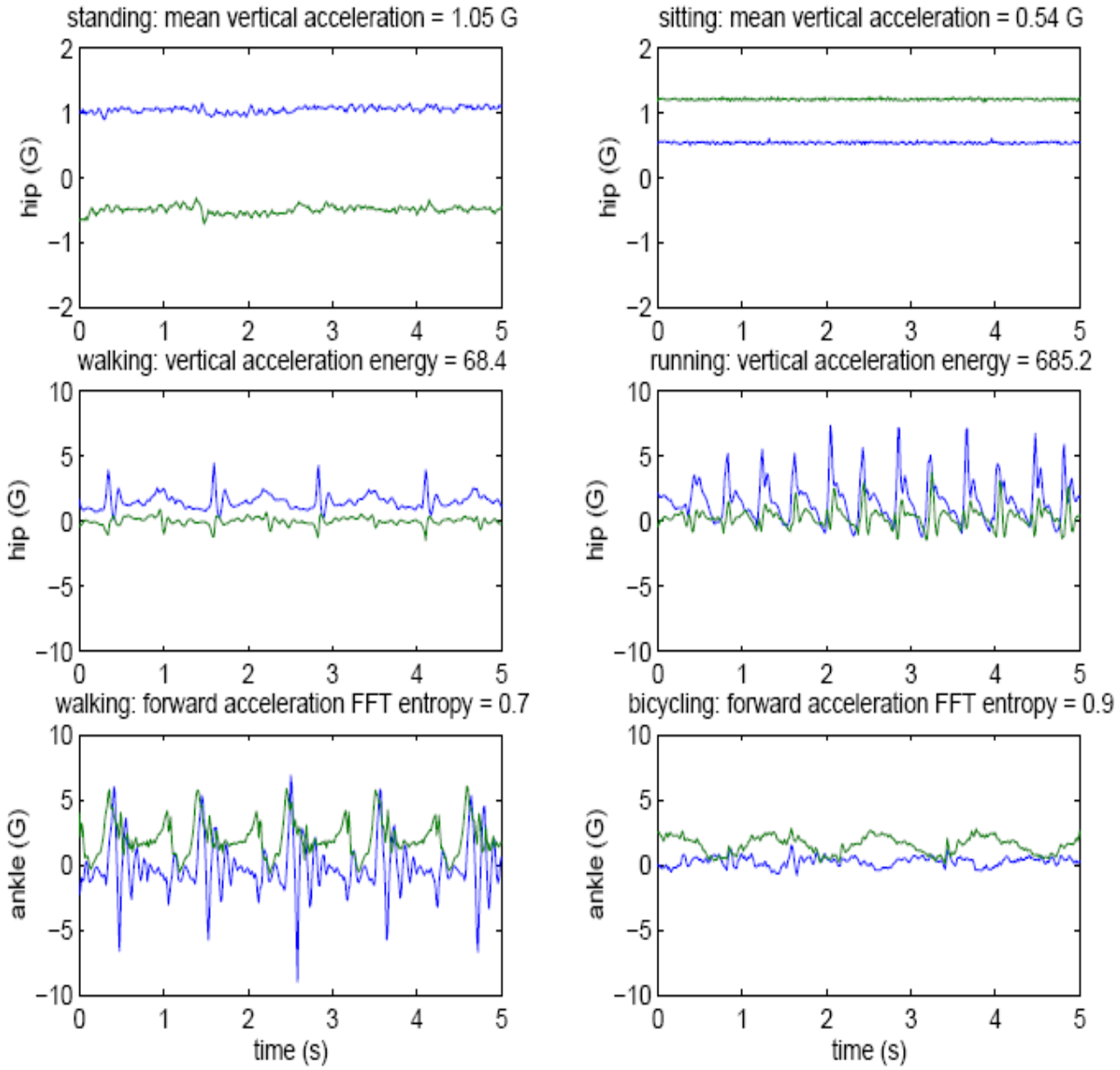


Abb. 4.3. Unterschiede der Merkmalwerte unterstützen die Unterscheidung verschiedener Aktivitäten. [3]

Klassifikator	Benutzerspezifisches Training	Leave-one-subject-out Training
Entscheidungstabelle	36,32 ± 14,501	46,75 ± 9,296
IBL	69,21 ± 6,822	82,70 ± 6,416
C4.5	71,58 ± 7,438	84,26 ± 5,178
naive Bayes	34,94 ± 5,818	52,35 ± 1,690

Tabelle 4.1. Zusammenfassung der Resultate der einzelnen Klassifikatoren (durchschnittliche Abweichung ± Standardabweichung). Die Klassifikatoren wurden mit Labordaten trainiert und auf dem Hindernisparcour getestet. [4]

Aktivität	Erkennungsrate	Aktivität	Erkennungsrate
Gehen	89,71	Gehen und etwas tragen	82,10
Sitzen und relaxen	94,78	Am Computer arbeiten	97,49
Still stehen	95,67	Essen oder trinken	88,67
Fernsehen	77,29	Lesen	91,79
Rennen	87,68	Fahrradfahren	96,29
Dehnen	41,42	Krafttraining	82,51
Schrubben	81,09	Staubsaugen	96,41
Wäsche zusammenlegen	95,14	Hinliegen und relaxen	94,96
Zähne putzen	85,27	Treppen steigen	85,61
Fahrstuhl fahren	43,58	Rolltreppe fahren	70,56

Tabelle 4.2. Aggregierte Erkennungsraten (in %) der Aktivitäten für die getesteten Versuchspersonen mit dem Leave-one-subject-out Training mit dem C4.5 Klassifikator. [4]

4.1.1.4 Fazit

Diese Studie hat gezeigt, dass ein benutzerspezifisches Training nicht nötig ist, um Erkennungsraten von über 80% für diese 20 alltäglchen Aktivitäten zu erreichen. Sie hat außerdem gezeigt, dass mit einer Gesamterkennungsrate von 84,26% in einer semi-natürlichen Umgebung Versuchspersonen sich frei außerhalb eines Labors, ohne von einem Forscher beobachtet zu werden, bewegen können, um die notwendigen Daten zu sammeln. Es sind also keine Labordaten erforderlich, um eine gute Erkennungsrate zu erzielen. Bei komplexen Bewegungen oder Bewegungen, die nicht eindeutig anhand der Beschleunigungsdaten erkennbar sind, gehen die Erkennungsraten zurück.
Der Einsatz von anderen Sensordaten könnte dem entgegenwirken und eine Verbesserung der Erkennungsrate ergeben, z.B. der Einsatz von Herzfrequenzen, die die Intensität der physischen Bewegung anzeigen könnten. Aber auch GPS-Ortungsdaten könnten einer Applikation Informationen liefern, in welcher Umgebung sich die Person im Momenet befindet. Ob man sich zu Hause oder in der Arbeit befindet. Um dann zu identifizieren ob man gerade am Computer arbeitet oder einfach vor einem Fernseher sitzt.

4.1.2 Weitere Studien zur Aktivitätenerkennung

4.1.2.1 Activity and Location Recognition Using Wearable Sensors [13]

Eine Studie von Seon-Woo Lee und Kenji Mase benutzt Beschleunigungs- und Winkelgeschwindigkeitdaten, um Aktivitäten und den Ort einer Person zu erkennen. Bei der Aktivitätenerkennung beschränkte man sich auf Gehen, Treppauf- und Treppabgehen.
Zwei Module mit Sensoren wurden eingesetzt, siehe dazu auch Abbildung 4.4. Das eine Modul wurde in der rechten oder linken Hosentasche getragen. Es enthielt einen biachsialen Beschleunigungsmesser und ein Gyroskop. Damit wurden die Beschleunigung und der Winkel der Hüfte des Benutzers gemessen. Das zweite Modul wurde in der Mitte der Taille angebracht und diente der Erkennung der Bewegungsrichtung, mithilfe eines digitalen Kompass. Beide Module waren mit einem PDA verbunden.
Von über 50 Testdatensammlungen wurden die Haupt- und Standardabweichungen der Horizontal- und Vertikal-beschleunigung, sowie die Integrale über die vier ersten Nulldurchgänge der Hüftwinkeldatenfunktion, berechnet. Mit den Abweichungen konnten Fuzzy-Sets, die auf einer Gaußschen Verteilungsfunktionen beruhen, beschrieben werden. Dies wurde für alle Aktivitäten durchgeführt.
Als Klassifikator kam dann eine einfache fuzzy-logic-reasoning Methode zum Einsatz, die Aktivitäten anhand der wahrscheinlichsten Aktivitätenverteilungsfunktion klassifiziert. Die Ergebnisse des Versuchs sind in der Tabelle 4.3 angegeben.

	Gehen	Treppauf	Treppab	Fehler	Gesamtanzahl der Schritte
Gehen	95,91	0,51	0,67	2,92	978
Treppauf	0	94,35	0	5,65	195
Treppab	0,51	0	92,85	6,63	199

Tabelle 4.3. Erkennungsraten in % für alle acht Teilnehmer der Studie [13]

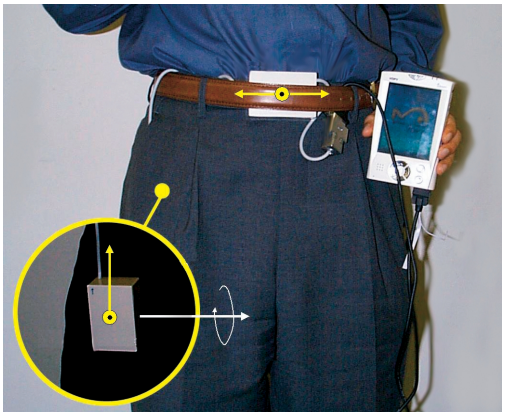


Abb 4.4. Benutzung der Versuchsmodule und Richtung der Messungen, der Module [13]

4.1.2.2 Hierarchical Recognition of International Human Gestures for Sports Video Annotation[9]

Bei dieser Forschungsarbeit war das Ziel die Erkennung komplexer, menschlicher Gesten anhand von Beschleunigungsdaten. Unter der Annahme, dass sich komplexe Gesten in Subgesten zerlegen lassen, wurde hier ein Ansatz gewählt, der auf Hidden Markov Modellen (HMMs) aufbaut. HMMs gehören zu den dynamischen Bayesschen Netzwerken. Es wurden sogenannte multi-layer HMMs eingestzt, die ein Spezialtyp der abstrakten HMMs[6] sind, aber auch mit anderen Erweiterungen der Standard-HMMs, wie gekoppelte HMMs[5] oder factorial HMMs[11], in Beziehung stehen. Faktorial HMMs stellen eine Zerlegung in Unterprozesse dar, die alle zu derselben Beobachtung beitragen, gekoppelte HMMs hingegen eine Zerlegung in Unterprozesse, die alle ihre eigene Beobachtung haben. Die Erkennung ist in hohem Maße von der Komplexität der Gesten abhängig. Da sehr unklar ist wann eine Subgeste beginnt und wann sie endet, weisen Messungen menschlicher Gesten einen hohen Rauschpegel auf. Wegen der unklaren Definition der Subgestengrenzen und das hohe Maß an Unterschieden der Gesten selbst eignen sich zur Erkennung HMMs. Abbildung 4.6 zeigt wie Gesten und Subgesten mit einander interagieren. Die unterste Ebene enthält einige Standard-HMMs, die einer individuellen Bewegung entsprechen.
Es sollten drei Kung Fu Schläge erkannt werden. Als Sensoren kamen hier zwei orthogonale uniachsiale Beschleunigungsmesser zum Einsatz, die am Handgelenk befestigt waren. Die Daten wurden von einem Kung Fu Lehrer aufgenommen. Als Merkmale, mit denen die HMMs trainiert wurden, extrahierte man die Nulldurchgangsrate der ersten und zweiten Ableitung, den quadratischen Mittelwert und den Durchschnitt aus den Beschleunigungsdaten über sliding windows von 32 Samples mit 8 überlappenden Samples. Da die Subgesten unabhängig voneinander sind, konnte jede separat trainiert werden. Aus dem Training resultieren Zustandsübergangswahrscheinlichkeiten, Beobachtungswahrscheinlichkeiten und eine Anfangszustandswahrscheinlichkeit. Neun HMMs wurden für neun Subgesten benutzt. Dann wurden drei HMMs mit Sequenzen der Subgesten, für die drei Kung Fu Schläge, trainiert.
Für die Trainingsdaten wurden zehn Schläge aufgezeichnet und für den Test je Schlag zehn Bewegungen geprüft. Die Ergebnisse stehen in Tabelle 4.4. Daraus ergibt sich ein Erkennungsrate von 96,67%, bei einer falschen Klassifizierung aus den 30 Testdurchläufen.

	Cuts	Elbows	Punch Blocks
Cuts	10	0	0
Elbows	1	9	0
Punch Block	0	0	10

Tabelle 4.4. Confusion Matrix für die drei Kung Fu Bewegungen die erkannt wurden. [9]

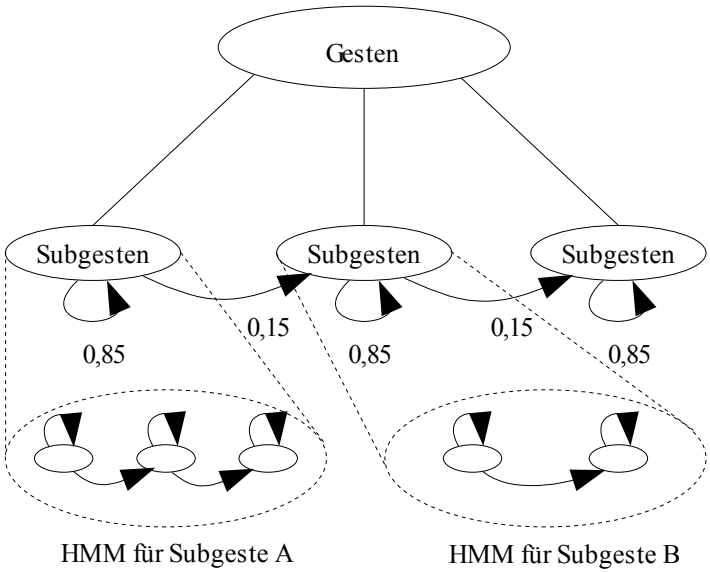


Abb 4.6. Gestenhierarchie [9]

4.1.2.5 Recognition of Walking Behaviors for Pedestrian Navigation[14]

Das Paper stellt eine Methode für die Erkennung von Gehbewegungen, dazu gehören hier Gehen, Treppauf- und Treppab gehen, vor. Mittels eines biachsialen Beschleunigungsmessers am unteren Rücken, einem digitalen Kompass und einem Infrarotlichtsensor wurden die entsprechenden Daten gewonnen, aus denen die Merkmale zur Klassifizierung berechnet wurden. Dazu durchliefen sechs Versuchspersonen, vorgeschriebene Sequenzen mit den oben genannten Laufbewegungen, um die Daten zu sammeln.
Als ein Merkmal wurde die Wechselstromanteile der Horizontal- und Vertikalbeschleunigung durch die Subtraktion des Mittels über 50 Samples von den einzelnen Datensammlungen und dann mit einem Lowpassfilter mit 5 Hz Cut-Off-Frequenz geglättet. Die Eliminierung der Gleichstromanteile sollte die Richtungsfehler durch die Bewegung des Sensormoduls verhindern. Als weitere Merkmale zog man die letzten positiven und negativen Scheitelwerte der gefilterten Beschleunigungsdaten über 0,5 Sekunden breite sliding windows heran. Und um die Erkennungsrate zu erhöhen wurde noch die Kreuzkorrelation der beiden Beschleunigungsfunktionen berechnet, da sich ohne dieses Merkmale das normale Gehen und Treppauf-/Treppab gehen nur schwer unterscheiden ließen.
Zur Klassifizierung der Bewegungen wurde ein nearest-neighbour-Klassifikator gewählt. Der die Aktivitäten mittels der euklidischen Distanz zwischen dem Merkmalsvektor und den personenspezifischen Referenzmerkmalsvektoren klassifiziert. Die Referenzmerkmalsvektoren wurden aus dem Mittel der Trainingsmerkmalsvektoren für jede Versuchsperson berechnet.
Bei den Testdurchläufen ergab sich eine Erkennungsrate von 83,3% - 96,3%. Siehe dazu auch Tabelle 4.5.

	Gehen	Treppauf	Treppab	nicht Klassifiziert
Gehen	96,3	1,7	0	2,0
Treppauf	11,1	83,3	0	5,6
Treppab	0	2,8	95,8	1,4

Tabelle 4.5. Erkennungsraten in % für Gehen, Treppauf- und Treppab gehen. [14]

4.1.2.4 Context Awareness by Analysing Accelerometer Data [16]

In der Studie von Randell und Muller wurden ein System entwickelt, das mit zwei orthogonalen, uniachsialen Beschleunigungsmessern in einer Hosentasche Bewegungen einer Person erkennen sollte. Mit den Sensoren wurden die Horizontal- und Vertikalbeschleunigung gemessen, wobei die Daten hier mit einer relativ geringen Samplefrequenz von 5 Hz aufgezeichnet wurden, da bei dieser Studie auch auf die Energieeffizienz des Systems geachtet wurde. Die Bewegungen, die erkannt werden sollten waren Gehen, Rennen, Treppauf-, Treppabgehen, Sitzen und Stehen. Aus den gewonnenen Daten wurden vier Merkmale extrahiert. Zum einen der quadratische Mittelwert und zum anderen das Integral über beide Achsen der Beschleunigungsmessungen. Diese Merkmale sind aber sehr von der Person, bei der die Daten aufgezeichnet wurden und von der getragenen Kleidung abhängig, d.h. bei der Erkennung der Aktivitäten muss das System von einer Person trainiert und kann auch nur von dieser getestet werden. Insgesamt wurden von 10 Personen Testdaten aufgezeichnet. Zur Klassifizierung der Aktivitäten wurde ein Clustering Algorithmus, genauer ein neuronales Netz, eingesetzt. Dieses Netzwerk wurde dann mit einer gesammelten Grundwahrheit über die 10 Personen trainiert, und nach weiterem personenspezifischen Training analysiert. Dabei ergaben sich Erkennungsraten der Aktivitäten von 85-90%. Die weiter in diesem Paper nicht dokumentiert und aufgeschlüsselt sind.

4.1.2.5 What Shall we Teach Our Pants? [18]

Hier sollte auch wieder die Erkennung eines kleinen Ausschnitts von alltäglichen Bewegungen untersucht werden. Im Einzelnen waren das Gehen, Sitzen, Rennen, Springen, Treppauf-, Treppab gehen und Fahrrad fahren. Man kam zu dem Schluss, dass die beste Position für den Sensor knapp über dem Knie liegt. Man verwendete zwei orthogonale, uniachsiale Beschleunigungsmesser, mit denen die Horizontal- und Vertikalbeschleunigung gemessen wurden. In Abbildung 4.7 sind Beschleunigungsdaten eines Sensors abgebildet, die beim Sitzen, Aufstehen und Gehen, in dieser Reihenfolge, aufgezeichnet wurden. Als Merkmalswerte wurden die maximale Beschleunigung und die Standardabweichung der Beschleunigung extrahiert. Aber, um das System flexibel für die Erkennung weiterer

Aktivitäten zu halten, auch die Nulldurchgangsrate der Beschleunigung und das Mittel über die Standardabweichungen der Beschleunigungsdaten von 50 Samples. Das System benutzt zur Erkennung der Bewegungen eine Hierarchie von Kohonen Maps (SOM). Kohonen Maps gehören zu den Self-Organizing Maps (SOM), sind eine Art künstlich neuronaler Netze bzw. eine Methode des unüberwachten Lernens.
Die Trainingsdaten stammen von einem Forscher, der die Bewegungen fünfmal nacheinander ausführte. Jede Bewegungsart dauerte ungefähr eine Minute. Die Resultate des Experiments sind aus Abbildung 4.6 zu entnehmen.

	Erkennungsrate des Testzyklus	
	3	5
Sitzen	96%	96%
Stehen	94%	94%
Gehen	75%	75%
Rennen	74%	78%
Treppauf	45%	42%
Treppab	48%	64%
Fahrrad fahren	89%	91%

Tabelle 4.6. Ergebnisse des Experiments. Zyklus 2 und 4 wurden für das Training, 3 und 5 zum Testen des Systems verwendet. Zyklus 1 wurde zur Initialisierung der Kohonen SOM benutzt. [18]

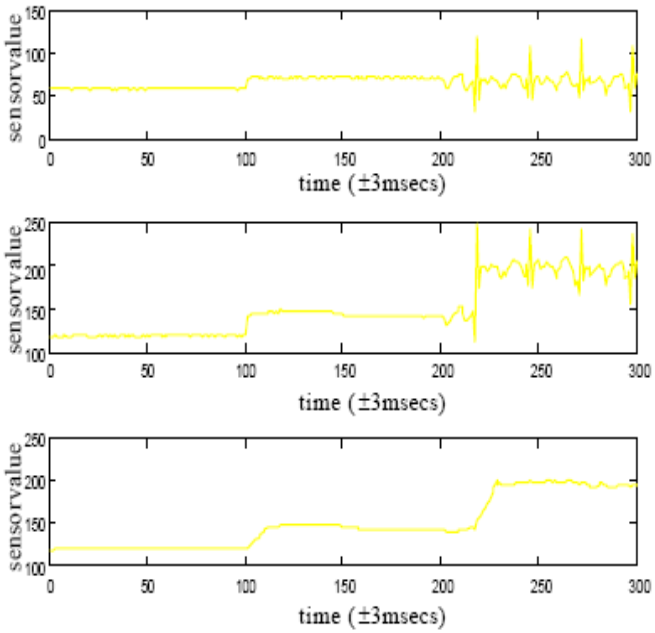


Abb. 4.7. Aufgezeichnete Sensordaten eines uniachsialen Beschleunigungsmessers (oben) während eine Testperson saß, aufstand und ging, jeweils 100 Testwerte. Die Summe über das Maximum und die Standardabweichung (Mitte). Das Mittel (unten) ist ausreichend, um diese Bewegungen zu unterscheiden. [18]

Ref.	Erkennungs-rate	Aktivitätenerkennung	Anzahl der Teilnehmer	Datentyp	Anzahl der Sensoren	Sensoren-anordnung
[13]	92,85% bis 95,91%	Körperbewegung	8	L	2	2 Oberschenkel
[15]	83% bis 90%	Körperbewegung, Körperhaltung	6	L	6	3 linke Hüfte, 3 rechete Hüfte
[10]	95,8%	Körperbewegung, Körperhaltung, Tippen, Sprechen, Fahrrad fahren	24	L	4	Brust, Oberschenkel, Handgelenk, Unterarm
[10]	66.7%	Körperbewegung, Körperhaltung, Tippen, Sprechen, Fahrrad fahren	24	N	4	Brust, Oberschenkel, Handgelenk, Unterarm
[1]	89.30%	Körperbewegung, Körperhaltung	5	L	2	Brust, Oberschenkel
[17]	86% bis 93%	Körperbewegung, Körperhaltung, Spielen	1	N	3	2 Taille 1 Oberschenkel
[12]	≈65% bis ≈95%	Körperbewegung, Tippen, Treppen steigen, Hände schütteln, Schreiben	1	L	bis zu 36	alle Hauptgelenke
[9]	96.67%	3 Kung Fu Armbewegungen	1	L	2	2 Handgelenk
[18]	42% bis 96%	Körperbewegung, Körperhaltung, Fahrrad fahren	1	L	2	2 unterer Rücken
[16]	85% bis 90%	Körperbewegung, Körperhaltung	10	L	2	2 Knie

Tabelle 5.1. Zusammenfassung von einigen, reprästentativen Studien der Aktivitätenerkennung, die Beschleungiungs-daten verwenden. In der Spalte “Datentyp” wird angegeben wo die Daten gesammelt wurden. Dabei bezeichnet (L) unter Laborbedingungen und (N) unter naturlichen Bedingungen. Die Spalte “Anzahl der Sensoren” enthält die Anzahl der uniachsialen Beschleunigungsmesser pro Versuchsperson. [4]

5. Schlussfolgerung

Die Auswertung der Papers hat gezeigt, dass die Leistung eines kontextsensitiven Systems, im Speziellen hier die Aktivitätenerkennung, alle ähnlich gut Leistungen bei der Erkennung von einfach zu definierenden Bewegungen, wie z.B. Gehen mit Erkennungsraten von bis zu über 95%, haben. Aber die Erkennung bei sich kaum unterscheidenden Sensordaten zum Teil stark absinken [18, 12]. Dies könnte mit anderen Sensordaten z.B. GPS-Informationen, aber auch Informationen über die Herzfrequenz oder der Extraktion weiterer Merkmale, die die Unterscheidung verbessern, erreicht werden. Zusätzliche Sensoren könnten auch hier ein bessere Erkennung unterstützen. Bei Vergleichen verschiedener Klassifikatoren konnten Entscheidungsbäume als Klassifikator bessere Ergebnisse als andere liefern [4].
Entscheidungsbäume haben zwar den Nachteil, dass sie nur langsam trainiert werden können, aber eine schnelle Reaktionszeit haben. Deshalb ist eine Echtzeiterkennung mit diesem Klassifikator möglich.
Dennoch liefern andere Klassifikatoren, für verschiedene Ansätze in der Erkennung der Aktivitäten, gute Ergebnisse wie z.B. die Verwendung von abstrakten HMMs für die Gestenerkennung mit einer Genauigkeit von insgesamt AC=96.67% [9].
Da viele Ergebnisse aus Laborversuchen stammen[13, 15, 10, 1, 12, 9, 18, 16], sind Vergleiche mit denen die unter „realen“ Bedingungen gesammelt wurden nur schlecht möglich. Festzustellen ist, dass je größer der Trainingsdatensatz für den Klassifikator ist und je besser, sprich natürlicher, die Umgebung ist, in der die Daten gesammelt werden, desto besser läßt sich ein System auch bei einem großen Personenkreis einsetzen [4]. Es werden also vergleichbare Ergebnisse mit den Laborversuchen erreicht, deren Systeme aber nicht so universell einsetzbar sind, da das Systeme oft nur mit einer Person trainiert und getestet wurde [17, 12, 9, 1]. Dadurch ist die Übertragbarkeit auf ein reales Szenario nur sehr eingeschränkt möglich.
Mit der richtigen Platzierung von nur wenigen Sensoren ist eine gute Erkennung möglich, wie die meisten der Versuche zeigen. Nur so ist auch ein System für den realen Einsatz vorstellbar. Eine Zusammenfassung der Versuchsergebnisse ist in Tabelle 5.1 zu sehen.

Referenzen

1. K. Aminian, P. Robert, E.E. Buchser, B. Rutschmann, D. Hayoz, and M. Depairon. Physical activity monitoring based on accelerometry: validation and comparison with video observation. *Medical & Biological Engineering & Computing*, 37(3):304–8, 1999.

2. K. Aminian, P. Robert, E. Jequier, and Y. Schutz. Estimation of speed and incline of walking using neural network. *IEEE Transactions on Instrumentation and Measurement*, 44(3):743–746, 1995.

3. L. Bao. *Physical Activity Recognition from Acceleration Data under Semi-Naturalistic Conditions*. M.Eng. Thesis, Massachusetts Institute of Technology, 2003.

4. Ling Bao, Stephen S. Intille. Activity Recognition from User-Annotated Acceleration Data. *In Proceedings of Pervasive 2004: the Second International Conference on Pervasive Computing. Springer, 2004.*

5. M. Brand, N. Oliver, and A. Pentland. Coupled hidden Markov models for complex action recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 994–999, San Juan, Puerto Rico, 1997.

6. H. H. Bui, S. Venkatesh, and G. West. Tracking and surveillance in wide–area spatial environments using the abstract hidden Markov model. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):177–195, 2001.

7. J.B. Bussmann, W.L. Martens, J.H. Tulen, F.C. Schasfoort, H.J. van den Berg-Emons, and H.J. Stam. Measuring daily behavior using ambulatory accelerometry: the Activity Monitor. *Behavior Research Methods, Instruments, & Computers*, 33(3):349–56, 2001.

8. O. Cakmakci, J. Coutaz, K. Van Laerhoven, and H.-W. Gellersen. Context Awareness in Systems with Limited Resources. *In Proceedings of the third workshop on Artificial Intelligence in Mobile Systems (AIMS), ECAI 2002, Lyon, France.* 2002. pp. 21-29.

9. G.S. Chambers, S. Venkatesh, G.A.W. West, and H.H. Bui. Hierarchical recognition of intentional human gestures for sports video annotation. In *International Conference on Pattern Recognition*, pages 1082–5, Quebec City, 2002.

10. F. Foerster, M. Smeja, and J. Fahrenberg. Detection of posture and motion by accelerometry: a validation in ambulatory monitoring. *Computers in Human Behavior*,15:571–583, 1999.

11. Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

12. N. Kern, B. Schiele, and A. Schmidt. Multi-sensor activity context detection for wearable computing. In *European Symposium on Ambient Intelligence (EUSAI)*. 2003.

13. Seon-Woo Lee and Kenji Mase. Activity and location recognition using wearable sensors. *IEEE Pervasive Computing*, 1(3):24–32, 2002.

14. S.-W. Lee and K. Mase. Recognition of walking behaviors for pedestrian navigation. *In Proceedings of 2001 IEEE Conference on Control Applications (CCA01)*, pages 1152–5. IEEE Press, 2001.

15. J. Mantyjarvi, J. Himberg, and T. Seppanen. Recognizing human motion with multiple acceleration sensors. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 747–52. IEEE Press, 2001.

16. C. Randell and H. Muller. Context awareness by analysing accelerometer data. In B. MacIntyre and B. Iannucci, editors, *The Fourth International Symposium on Wearable Computers*, pages 175–176. IEEE Press, 2000.

17. M. Uiterwaal, E.B. Glerum, H.J. Busser, and R.C. van Lummel. Ambulatory monitoring of physical activity in working situations, a validation study. *Journal of Medical Engineering & Technology.*, 22(4):168–72, 1998.

18. K. Van Laerhoven and O. Cakmakci. What shall we teach our pants? In *The Fourth International Symposium on Wearable Computers*, pages 77–83. IEEE Press, 2000.