Schließen aus Kontexten

Ausarbeitung zum Seminar
"Ubiquitous Computing"
von
Martin Berchtold

Betreuer: Tobias Zimmer

1 Einführung

1.1 Was ist Kontext? Verschiedene Definitionen in der Literatur

In einem Papier von Dey und Abowed [DA00] fanden sich folgende Zitate aus anderen Arbeiten zusammen getragen.
Schilit und Theimer [ST94] definieren den Kontext als Orte, Identität naher Personen und Objekte, und Änderungen bezüglich dieser Objekte und Personen. In ähnlicher Weise definiert Brown [BBC97] Kontext als Lokation, Identität von Personen in der Umgebung des Benutzers, Tageszeit, Jahreszeit, Temperatur usw. Ryan u.a. [RPM97] beschränken sich auf Lokation, Umgebung, Identität und Zeit. Dey [Dey98] zählt als Kontext den emotionalen Zustand des Benutzers, seinen Grad an Aufmerksamkeit, Lokation und Orientierung, Datum und Zeit, Objekte und Personen in seiner Umgebung. Diese Einschränkungen, der Vielfalt des Begriffes Kontext, sind schwer zu Implementieren. Wird ein weiterer Begriff in die Definition aufgenommen, so passt dies meist nicht in die aktuelle Definition. Es mssen also globalere realitätsnähere Beschreibungen definiert werden.
Nun wurden viele Definitionen veröffentlicht, die zwischen Umgebungen von Benutzer und Applikation unterscheiden. Brown [BBC97] meint Kontext seien Elemente der Umgebung des Benutzers, von denen der Computer Kenntnis besitzt. Franklin und Flaschbart [FJ98] sehen mehr die Situation des Benutzers als Kontext.
Zusammenfassend kann jedoch die Untergliederung von Schilit u.a. [BAW94], Dey u.a. [DAW99] und Pascoe [Pas98] als Definition genannt werden. Kontext besteht in ihrem Sinne aus:

- *Berechenbare Umgebung:* verfügbare Prozessoren, Geräte zugänglich für die Benutzereingabe, Kapazität des Netzwerkes, Konektivität und Kosten der Berechnung.

– *Benutzerumgebung:* Lokation, Menge der nahen Personen und soziale Situation.
– *Physikalische Umgebung:* sensorisch erfassbare Parameter.

1.2 Zwei konkrete Modelle / Rahmenwerke

In diesem Unterkapitel möchte ich die zwei verbreitetsten Modelle darstellen, die einen allgemeinen Zugang zur Aneignung von kontextuellem Wissen bieten. Ein weiteres Rahmenwerk findet sich in [KMK+03], welches kontextuelle Informationen in Form von Ereignissen dem System bereitstellt. Dies ist ein sehr neues Rahmenwerk. Beschreiben und gegenüberstellen möchte ich deshalb nur das Schichten-Modell von Schmidt u.a. [SL01] und das Rahmenwerk von Dey u.a. [DA99].

Das "Sensor-Cue-Kontext-Anwendung"-Schichtenmodell [SL01] ist in vier Schichten aufgeteilt (Fig.1). Die Unterste ist die Hardware, die Sensoren. Hier werden, je nach Sensor, gewisse Aspekte der Umwelt aufgegriffen. Die nächste Schicht (Cue) ist eine Abstraktion der Sensorwerte in die Richtung der Daten, die später mit Algorithmen einheitlich verarbeitet werden können. Diese Algorithmen, zur Erzeugung einer maschinell verwertbaren Kontextrepräsentation, finden sich auf der dritten Schicht, der Ebene des Kontextes. Zuletzt kommt nun die Applikation, die das gewonnene Wissen verwertet. Mit Schnittstellen zwischen den einzelnen Schichten, bietet dieses Modell Möglichkeiten zur Verteilung, von Sensoren, Cue-Extraktion, Erstellung des Kontextes und Applikation, auf verschiedene Systeme.

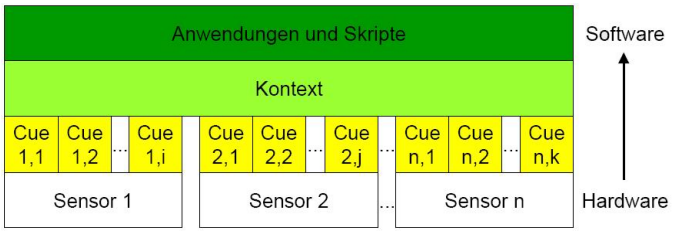

Fig. 1. "Sensor-Cue-Kontext-Anwendung"-Schichtenmodell

Das "Kontext-Toolkit"-Rahmenwerk
Informationen über den Kontext sind anders zu verwerten als herkömmliche Eingabemethoden wie Maus und Tastatur. Aus diesen werden Ereignisse für das Rechnersystem generiert, wann kann jedoch aus bestimmten Kontexten ein Ereignis werden. Was für eine Applikation ereignisreich an der Umgebung sein

kann, ist für eine andere Anwendung völlig belanglos oder in der bereitgestellten Form des Ereignisses nicht zu verwerten. Desshalb sind die meisten Applikationen, die kontextuelles Wissen nutzen, genau für die Verwendung eines Aspektes des Kontextes konstruiert, was es schwierig macht neue Applikationen zu entwickeln oder bereits existierende nochmals zu verwenden.
Um diese Nachteile zu eliminieren wurde das "context-toolkit" [DA99] erstellt. Dieses Werkzeug ist eine in Java geschriebene *Middelware*, das Abstraktionen und Hilfen im Bereich des kontextuellen Bewustseins bereitstellt.
Die Aufteilung der Elemente der Verarbeitung (Fig.2) zeigt die thematische Einordnung in dieses Kapitel, denn schon die Abstraktion, die u.A in [MKVA03]) Verwendung findet, einzelner Aufgabengebiete - *abgesehen won der realen Implementierung und Verwendung dieser Java Software* - bringt große Vorteile gegenüber herkömmlichen Anwendungen. Bisherige Systeme hängen stark von der verwendeten Hardware ab, was die Möglichkeiten der Generalisierung stark beeinträchtigt oder gar verhindert. Die Verwendung einer ersten Abstraktionsebene, die der *Widgets* (s.u.) macht das ganze System schon unabhängig von der verwendeten Hardware. Die Verwendung von XML, zur Kommunikation zwischen den einzelnen Elementen des *toolkit*'s, ermöglicht eine Verteilung der Komponenten auf verschidene Rechnersysteme. Das "context toolkit" setzt sich aus

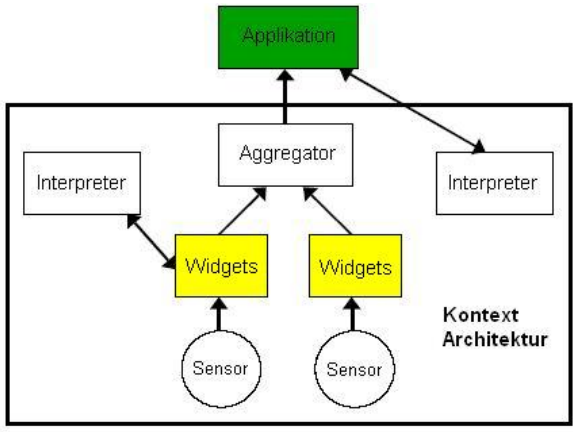

Fig. 2. "Kontext-Toolkit"-Rahmenwek

den folgenden Komponenten zusammen:

– Widgets:
Sie vermitteln zwischen Umwelt und Benutzer. Diese Interaktionsobjekte schachteln Informationen in kleine *Stücke* des Kontexts. Sie stellen eine standartisierte Schnittstelle für die Applikationen bereit. Diese Schnittstelle verbirgt die Mechanismen zur Generierung dieser Informationen.

– Aggregators:
  Die *Sammler* können als meta-"widgets" angesehen werden. Sie stellen die Möglichkeit zur Anhäufung von Kontexten bereit.
– Interpreters:
  Der Interpreter leitet, aus den schon zur Verfügung stehenden Kontexten, eine Aussage auf einer höheren Ebene ab.

Weitere Informationen findet sich in dem Papier [DA99]. Die Seite zum herunter laden des *toolkit*'s findet sich unter [].

Zusammenfassung - Gegenüberstellung
Die Verwendung von zwei Abstraktionsebenen des "context-toolkits" auf der Kontextebene (Widgets und Aggregators) ermöglicht es verschiedene Algorithmen, mit verschiedenen Aufgabengebieten zur Generierung von Kontext anzuwenden, ohne daß Zwischeninformationen verloren gehen. Das Modell von Schmidt u.a. [SL01] macht keine Angaben darüber, wie sich die Kontextschicht intern darstellt.
Da schon erstellte Kontextinformationen nicht immer in einem für die Anwendung zugänglichem Format vorhanden sind, stellen die Interpreter [DA99] eine sehr mächtige Komponente dar. Auch hier wird in [Sch02] keine Aussage gemacht, wie dies in der Schnittstelle, zwischen Applikation und Kontext, bewerkstelligt weren soll.
Nun ist aber das "context-toolkit" ein sehr rechenintensives Rahmenwerk - *gerade die Implementierung in Java und die Algorithmen des Interpreters* - wogegen die Architektur des "Sensor-Cue-Kontext-Anwendung"-Schichtenmodells extra für flexible effizient Systeme mit wenig Rechenleistung erstellt wurde.
In einem System [MKVA03] das auf "Smart-Its" (s.u.) implementiert ist, wird eine Mischung beider Modelle vorgeschlagen. In diesem Report befindet sich auch die hier geführte Argumentation.

2 Die Sensoren

Sensoren können als *Sinnesorgane* der ubiquitären Kontexterfassung angesehen werden. Je nachdem welche Aspekte des Umgebungskontextes erkannt werden soll, muß der Sensor gewählt werden. Hierbei kann ein Sensor, je nach Interpretation seiner Messwerte, verschiedene kontextuelle Informationen liefern. In dieser Arbeit werden Verfahren und Anwendungen analysiert, die verschiedene Sensoren benutzen.

So werden im Kapitel 4.1 zwei Verfahren dargestellt, die Messwerten eines Beschleunigungssensors Bezeichnungen menschlicher Bewegungenmuster zuordnen. Gerade Sensoren der Beschleunigung sind reich an Informationen, um alle Arten von Bewegungen zu erkennen.

Eine weitere Arbeit, die in diesem Kapitel genannt wird, benutzt den Boden

als kognitives Werkzeug. Mit Hilfe von Drucksensoren im Boden eingebaut, werden Laufmuster, einzelne Personen anhand dieser Laufmuster und Gegenstände auf diesem Boden erkannt.

Um Bewegungen und Aufenthaltsorte von Personen feststellen zu können, werden in einer weiteren Arbeit Kameras benutzt. Hier gilt, daß in Bildern mehr Informationen liegen als das *Bild an sich.*

Eine Komination verschiedenster Sensoren (Fig.3: Beschleunigung x-y-z 4a/b, Temperatur 5, Licht des normalen und infraroten Spektrums 1/2, Druck 3 und Ton) findet sich in einem Artefakt kleinster Bauart, den "Smart-Its". Diese Artefakte besitzen einen Mikrokontroller als Recheneinheit und ein RF-Modul zur Kommunikation. Durch sie und andere Geräte dieser Art sind Informationen zugänglich die durch einzelne Sensoren nicht erfassbar wären. Diese Artefakte werden in Kapitel 4.1 benutzt, um kontextuelles Wissen über die Umgebung zu sammeln.
Zu einem ähnlichem Gerät, daß auch viele Sensoren in einem Gerät vereint, wird vieles in Kapitel 6.1 berichtet.

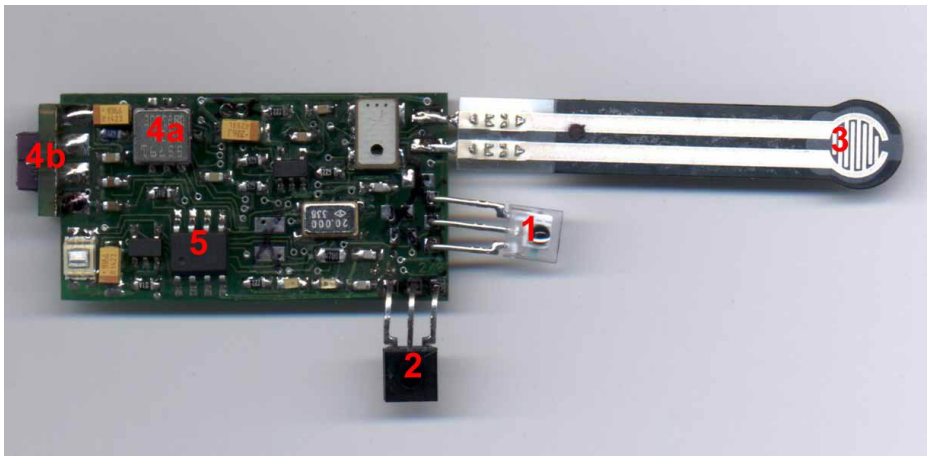

Fig. 3. Smart-Its Sensor Board (Vers.2/02)

3 Erste Abstraktion der Sensormesswerte in Richtung der Daten

Sensoren liefern einen kontinuierlichen Strom an Messwerten. Damit dieser Strom an Werten in ein verarbeitbares Paket umgewandelt werden kann, wird immer ein

bestimmter Ausschnitt des Stromes gesondert verarbeitet. Dieses Verfahren des *gleitenden Fensters* (sliding window) arbeitet mit einer gewissen Breite des Fensters, also eine betimmte Anzahl der Messwerte die betrachtet wird. Ist ein Teil des Datenstromes angeschaut worden, so wird der Ausschnitt auf den nächsten Teil des Stromes weiter bewegt, das *Fenster gleitet* weiter. Diese Ausschnitte können sich natürlich auch überschneiden. Verwendet und detailliert beschrieben wird dies in [AJLS97](s.u.).

Wird eine hohe "Polling"-Rate der Sensoren gewählt, so kann bei vielen Sensoren eine Überlastung des Speichers schnell erreicht werden. Dies tritt natürlich am schnellsten bei begrenzten Speichern eines Mikrokontrollers ein. In diesem Bereich sind zwei Verfahren zu nennen, eines für hohe und eines für sehr hohe "Polling"-Raten.
Für hohe Raten eignet sich eine Berechnung eines Mittelwertes. Hier wird für eine Reihe von n Messwerten x_i ein Mittelwert $\overline{x}$ berechnet, mit

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Diese Methode ist aber nur für eine geringe Abweichungen der Messwerte voneinander geeignet, da sonst die Aussagekraft der erhaltenen Daten gering oder gar falsch ist. Die mittlere absolute Abweichung - *in [KSSF03] mit "sum of absolute differenz" bezeichnet* - wird mit

$$\frac{1}{n} \sum_{i=1}^{n} |x_i - \widetilde{x}|$$

berechnet, wobei

$$\widetilde{x} := \begin{cases} x_{(\frac{n+1}{2})} & \text{,falls } n \text{ ungerade,} \\ \frac{1}{2}(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}) & \text{,falls n gerade.} \end{cases}$$

den Zentralwert darstellt. Bei n Messwerten wird nur ein Datum gespeichert, was den Speicherplatz, gerade bei einem großen Wert für n, stark reduziert.
Als Methode für sehr hohe "Polling"-Raten kann eine Fast Fourier Transformation (FFT) eingesetzt werden. Bei einer Fourier-Transformation wird der Abtastwert (Messwert) in den Frequenzanteil überführt. Die FFT macht dies sogar in einer akzeptablen Zeitkomplexität $O(n \log(n))$. Vorgeschlagen werden beide Verfahren in der Arbeit von Krause u.a. [KSSF03](s.u.).

Sensoren tendieren auch zu großen Abweichungen und verrauschten Signalen, wesshalb die Signale oft gefiltert werden müssen. Hierzu ist auch die Methode der absoluten Abweichung und die FFT, die schon genannt wurden, einsetzbar.

Um Daten in einem Algorithmus verarbeiten zu können muss eine standartisierte Eingabeform produziert werden. Diese Eingabe erfolgt meist in Form

eines Vektors. Da häufig mehrere verschiedene Sensoren in den Vektor eingebunden werden, deren Werte in verschiedenen Wertebereichen liegen, muss der Vektor normalisiert werden. Eine Normalisierung eines Vektors $X = (x_1, .., x_n)$, wird durch teilen dieses Vektors durch seine Norm erreicht, also

$$E := \frac{X}{\|X\|}.$$

Ein Beispiel hierfür findet sich wieder in [KSSF03](s.u.).

4 Projektion von Sensordaten auf kontextuelle Aussagen

Nachdem die Messwerte der Sensoren in verarbeitbare Daten umgewandelt wurden, können diese auf kontextuelle Aussagen projiziert werden. In zwei Themengebieten sind insgesamt drei verschiedene Verfahren dargestellt, die zu den gängigsten gehören.
Das erste Themengebiet ist das der neuronalen Netze, die detailliert beschrieben werden, da sie oft Anwendung in ubiquitären Systemen finden. Mit dem zweiten Gebiet, das der "Hidden Markov Models", wird ein Verfahren der Probabilistik beschrieben. Gerade Aussagen über Wahrscheinlichkeiten sind in der ubiquitären Kontext-Erfassung ein mächtiges Mittel, da ja auch menschliche Sinnesorgane zum großen Teil der Interpretation unterliegen.

4.1 Neuronale Netze [Zel94]

Ein neuronales Netz ist ein gerichteter Graph (Fig.4), wobei die Knoten die Neuronen (Zellen) darstellen. Die Knoten sind hirarchisch angeordnet. Auf unterster Ebene befindet sich die Eingabeschicht (input layer) des Netzes. Diesen ersten Neuronen werden die zu verarbeitenden Daten eingegeben. Auf der obersten Ebene befinden sich die Neuronen der Ausgabeschicht (output layer). Jedes Neuron dieser Schicht wird mit einem *Ergebnis* der Verarbeitung des Netzes assoziiert. Zwischen Ein- und Ausgabe befinden sich die Schichten der Verarbeitung der Daten. Diese Ebenen von Neuronen werden verdeckte Schichten (hidden layers) genannt. Sie können aus 0 bis n Ebenen Neuronen bestehen.
Die einzelnen Ebenen Neuronen sind über Kanten miteinander verbunden. Meist ist jedes Neuron der Schicht k mit jedem Neuron der Schicht $k + 1$ über eine Kante verbunden. Jede Kante wird mit einem Gewicht versehen, das für eine Kante von Neuron i zum Neuron j mit w_{ij} bezeichnet wird. Die Ausgabe o_i die eine Zelle i produziert, wird mit dem Gewicht w_{ij} der Kante multipliziert und ergibt die Eingabe der Nachfolgerzelle j. Somit ergibt sich die Eingabe des Neurons j aus der Summe der Ausgaben der Vorgängerzellen, multipliziert mit den Gewichten der Verbindungen zwischen den Neuronen. Hieraus folgt - *in der mathemetischen Schreibweise* - also die Eingabe

$$net_j(t) = \sum_i o_i(t) w_{ij}$$

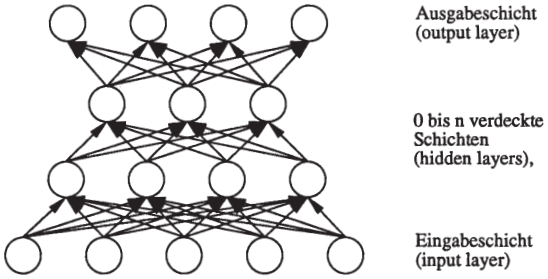

Fig. 4. Einfache Darstellung eines "feed forward" neuronalen Netzes

des Neurons j (der Schicht k) aus den Neuronen i der Schicht $(k-1)$. Eine graphische Beschreibung finden sie in der Fig.5.

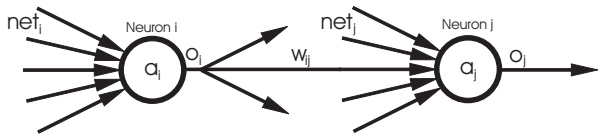

Fig. 5. Idealisiertes Neuron

Die Ausgabe die ein Neuron produziert hängt mit dessen Eingabe zusammen. Überschreitet die Eingabe net_i einen gewissen Wert, so produziert die Zelle i eine Ausgabe, sie *feuert*. In den gängigen Netzen sind dies binäre Werte, wobei 0 (auch -1) keine Ausgabe darstellt und die Zelle bei 1 "feuert". Dieser Zusammenhang zwischen Ein- und Ausgabe der Zelle wird Aktivierungsfunktion - *hier wurde der Sachverhalt vereinfacht, da die Acktivierungsfunktion nicht unbedingt das Ausgabeverhalten wiedergibt (siehe [Zel94])* - genannt. Mathematisch kann die Aktivierungsfunktion wie folgt beschrieben werden:

$$a_j(t) = \begin{cases} 1 & \text{falls } net_j(t) \geq \theta_j, \\ 0 & \text{sonst.} \end{cases}$$

Ein neuronales Netz kann nun durch Lernen so modifiziert werden, daß es auf eine Eingabe eine bestimmte Ausgabe produziert. Dieses Training des Netzes wird meist durch Änderung der Gewichte - *es gibt auch noch andere Möglichkeiten (siehe [Zel94]), die aber selten verwendet werden* - erreicht. Gewichte können in ihrem Wert erhöht oder erniedrigt werden. Sollte eine Verbindung, zwischen zwei Neuronen, getrennt werden, so wird das Gewicht auf Null gesetzt. Es existieren verschiedene Kategorien von Lernverfahren:

- Überwachtes Lernen (supervised learning)
  Bei diesen Lernverfahren wird auf eine Eingabe eine gewünschte Ausgabe dem Netz als Information zugeführt.
- Bestärkendes Lernen(reinforcement learning)
  Diese Verfahren arbeiten mit *belohnen* und *bestrafen* des Netzes. Wird eine Ausgabe produziert, so bekommt das Netz als Information nur, ob diese Ausgabe richtig oder falsch ist.
- Unüberwachtes Lernen (unsupervised learning)
  Hier bekommt das Netz keinerlei Informationen über die Richtigkeit seiner Ausgabe. Diese Verfahren abeiten nach der Methode der Clusterbildung, wobei ähnliche Daten auf gleiche Cluster abgebildet werden.

Neuronale Netze unterscheiden sich nicht nur durch die verschiedenen Lernverfahren, sondern auch durch ihre Topologie. Sind die Schichten durch die Kanten nur in vorwärts Richtung (Eingabeschicht in Richtung Ausgabeschicht) verbunden, so nennt man dies ein "feed forward" Netz. Existieren auch Kanten die zurück führend sind, so kann zwischen mehreren Arten von rückkoppelnden Netzen unterschieden werden:

- direct feedback
- indirect feedback
- lateral feedback

Für Erklärungen wird wieder auf das Buch von Andreas Zell [Zel94] verwiesen. Jetzt aber genug der Theorie, nun zu Netzen die zur Kontexterfassung in ubiquitären Systemen Einsatz finden.

Backpropagation ist ein Lehrnverfahren zur Modifikation der Gewichte eines neuronalen Netzwerks. Dieses Gradientenabstiegsverfahren arbeitet nach der Methode des steilsten Abstiegs der Fehlerfunktion. Die Fehlerfunktion $E(w) = E(w_1, .., w_n)$ gibt den Fehler an, den das Netzwerk bei gegebenen Gewichten $w_1, .., w_n$ über alle Trainingsmuster aufsummiert besitzt. Der Fehler beschreibt die Differenz von Ausgabe und gewünschter Ausgabe des neuronalen Netzes, bei einer bestimmten Eingabe. Mit Backpropagation wird nun versucht, ein globales Minimum der Fehlerfunktion (Fig.6)zu finden.

Ein erstes Beispiel, für die Verwendung eines Backpropagation Lehrnverfahrens, findet sich in der Arbeit von Cliff Randell und Henk L. Muller [RM02]. Sie benutzten ein einschichtiges Netz (eine Schicht modifizierbarer Gewichte) um Werte von Beschleunigungssensoren (x-y-Achse) auf Körperaktivitäten (Rennen, Sitzen, Laufen, treppauf und treppab Laufen) abbilden zu können. Das Netz wurde im ersten Schritt mit eindeutig klassifizierbaren Daten - *Mengen, der auf Aktivitäten abgebildeten Messdaten, besitzen keine Überschneidung* - trainiert. Nach dieser anfänglichen Modifikation, wurden Daten des normalen Ablaufs der Bewegung, zur weiteren Verbesserung der Klassifikation benutzt. Das Resultat des Versuchs, nach Abschluss des Trainings und Benutzung im alltäglichen Gebrauch, war sehr gut. Die Abbildung war in $85 - 90\%$ der Fälle richtig. Fehler

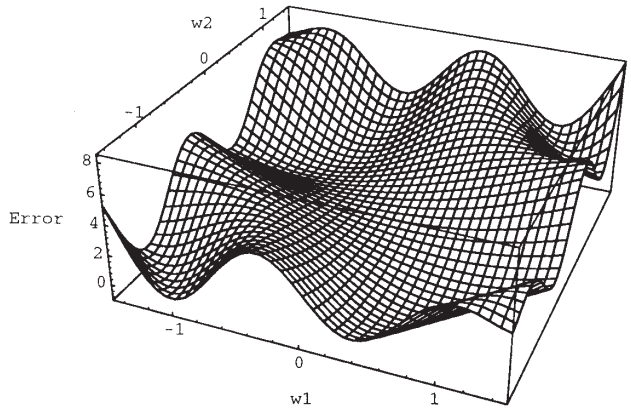

Fig. 6. Fehlerfläche eines Neuronalen Netzes als Funktion der Gewichte w_1 und w_2

der Zuordnung ($10 - 15\%$), von Daten auf Kontext, konnten durch temporale Filterung auf 5% reduziert werden.

Eine weitere Implementierung des Lernverfahrens konnte in einem Rahmenwerk für mobile Agenten [RF03] gefunden werden. In dieser Arbeit wird ein neuronales Netz, trainiert durch Backpropagation, zur Bestimmung der Position von mobilen Geräten benutzt. Dieser Pocket-PC verfügt über eine drahtlose Netzwerkkarte nach dem IEEE 802.11b Standard. Nun wird mittels des neuronalen Netzes die Feldstärke der Zugangspunkte ("Accsesspoint" des "Wireless-Networks") auf die Benutzerposition abgebildet. Leider wurden in dem referenzierten Papier keine Aussagen darüber gemacht, wie gut dies funktioniert.

Als negatives Beispiel, für die Wahl des Backpropagations Lehrnverfahrens, kann das jetzt zuletzt vorgestellte Papier angesehen werden. Das an der Universität Koblenz entwickelte Multi-Agenten-Netz-Informationssystem [BTW00] benutzt das Lernverfahren um Seiten im Internet zu finden, die Angaben über Adressen enthalten. Sie benutzen 100 Neuronen für die Eingabe, 50 in den verdeckten Schichten und zwei als Ausgabe. Jedes Eingabeneuron ist mit einem zu erkennenden Wort assoziiert, wobei die Zelle "feuert", wenn das Wort auf der zu testenden Seite gefunden wird. Die Ausgabeneuronen sind mit den Aussagen (Kontext: Addresse) "Seite besitzt Addresse" und "Seite besitzt keine Addresse" verbunden. Zuerst erfolgt ein "offline"-Training mit 3000 vorklassifizierten Seiten. Danach ein weiteres Test-Training mit weiteren 3000 Seiten.
Die Seiten mit keiner Information bezüglich einer Addresse machen 98% aller Seiten im Netz aus. Von den 2% aller Seiten mit Addressen werden nur 60% als solche erkannt.

Im Gegensatz zu anderen Verfahren, zur gänzlichen Abbildung von Sensordaten auf "Kontext", ist Backpropagation selten anzutreffen. Häufig wird ein neuronales Netz, das nach diesem Lernverfahren trainiert wird, für Subprobleme wie in [RF03] und [BTW00] eingesetzt.

Kohonen Selforganizing Maps (KSOM, Selbstorganisierende Karten) sind eine Weiterentwicklung des Verfahrens der lernenden Vektorquantisierung (LVQ), wesshalb hier zuerst auf die LVQ eingegangen wird. In beiden Fällen handelt es sich um einschichtige neuronale Netze, also um Netze mit einer Schicht modifizierbarer Gewichte. Jedes Neuron der Ausgabeschicht ist mit jedem der Eingabeschicht verbunden. Für ein Neuron j ergibt sich somit ein Gewichtsvektor $W_j = (w_{1j}, w_{2j}, ..., w_{nj})$ (n Neuronen der Eingabeschicht, Fig.7) der "Codebook"-Vektor genannt wird. Ein neuer Eingabevektor wird nun mit

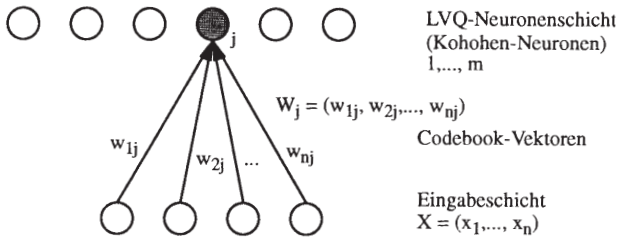

Fig. 7. Netzstruktur der lernenden Vektorquantisierung (LVQ)

allen "Codebook"-Vektoren (CV) verglichen, wobei derjenige CV der *Gewinner* ist, der dem Eingabevektor am ähnlichsten ist. Als *Ähnlichkeitsmaß* wird meist der Euklidische Abstand, zwischen dem Coodebook und dem Eingabe-Vektor, benutzt. Der CV der den geringsten Abstand besitzt gibt nun die Klasse (in unserem Fall einen speziellen Kontext) des Eingabevektors an. Der CV des *Gewinner*-Neurons wird in der Phase des Trainings dem Eingabe-Vektor ähnlicher gemacht (Fig.8).
Der Unterschied der KSOM (Fig.9) zu der LVQ besteht nun darin, daß die Neuronen der Ausgabeschicht zueinander in Beziehung stehen. Diese Beziehung zwischen Nachbarn wird durch ein meist quadratisches oder hexagonales zweidimensionales Gitter, der Ausgabezellen, ausgedrückt. Nun wird beim Training des Netzes nicht mehr nur der ähnlichste CV verändert, sondern auch seine Nachbarn. Mit abnehmender Stärke werden auch der Nachbarn Nachbarn dem Eingabe-Vektor ähnlicher gemacht.
Im Gegensatz zur LVQ, die ein überwachtes Lernverfahren darstellt, sind die KSOM's ein Beispiel für ein unüberwachtes Lernverfahren. Bei der LVQ muss

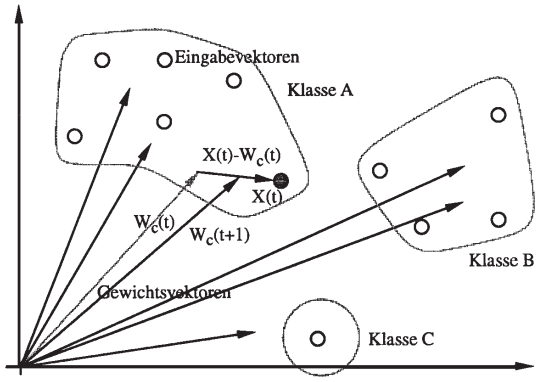

Fig. 8. Adaption der Gewichte der lernenden Vektorquantisierung (LVQ)

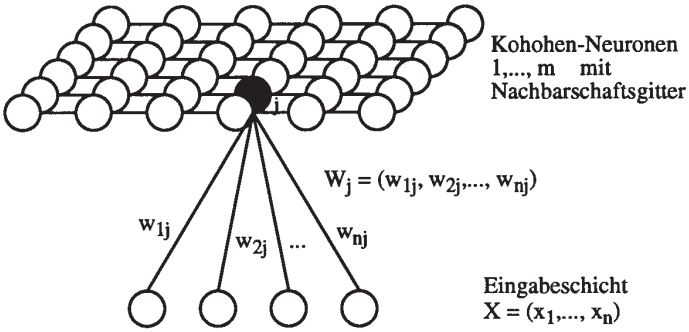

Fig. 9. Netzstruktur der selbstorganisierenden Karten (SOM)

zu jedem Eingabevektor bekannt sein, zu welcher Klasse er gehört (Fig.8). Die KSOM's bilden die Zugehörigkeit zu den Klassen selbst aus.

Als ersten Bericht, in dem eine selbstorganisierende Karte benutzt wird, möchte ich ein Papier von Laerhoven [Lae01] behandeln. Er hat in seiner Arbeit einen Beschleunigungssensor benutzt, der zwei Werte (x-y-Achse) liefert. Ziel des Versuchs war es mit Hilfe eines KSOM's, aus diesen Sensordaten, typische menschliche Bewegungsmuster (sitzen, laufen, rennen, usw.) zu erkennen. Hierfür hat er eine 10×10 -Karte (100 Neuronen im Gitter) benutzt. Wurde die SOM mit den gewonnenen Daten trainiert, so wurde eine sehr langsame Konvergenz des Verfahrens, der Zuordnung von Daten zu Kontext, festgestellt. Auch stellt sich eine Konvergenz nur ein, wenn sich die *Gewinner-Neuronen* für die verschiedenen Kontexte weit genug entfernt voneinander auf der Karte befinden.

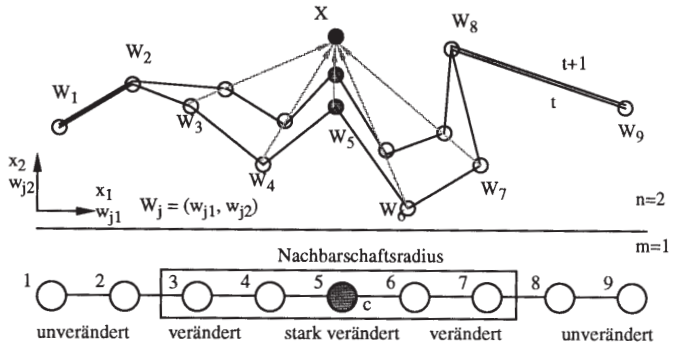

Fig. 10. Bewegung der Gewichte einer eindimensionalen selbstorganisierenden Karten

Aus diesen Gründen hatte er das Netz mit einem weiteren neuronalen Netz, das auf das KSOM aufgesetzt wird, kombiniert. Das Verfahren des k-means Clustering, das er benutzte, filtert die anfänglichen KSOM Cluster und sichert den Erhalt der Daten. Somit wurde also das Problem gelöst, daß ein Wert für einen CV, nach Training auf einen Kontext, durch Eingabe von Daten für einen neuen Kontext, überschrieben wird.

Eine sehr interressante Entwicklung der SOM fand in einem Projekt [CLS02] des Computing Department der Universität Lancaster statt. Das Verfahren der SOM findet sich normalerweise auf einem Einprozessorsystem zentral implementiert. In diesem Fall wurde der Algorithmus so abgeändert, daß er auf mehreren "Smart-Its" - einzelne Microcontroller mit Sensoren und RF-Modul kominiert - verteilt läuft. Ein "Smart-It" verkörpert ein Neuron und hält seinen Coodbookvektor im eigenen Speicher. Die physische Distanz der "Smart-Its" bestimmt den Abstand der Neuronen im Gitter. Neuronen können ad-hoc entfernt, hinzugefügt und bewegt werden.
Die Sensordaten, die die einzelnen Neuronen erfassen sind nur geringfügig anders, da sich alle "Smart-Its" im gleichen Kontext befinden. Wurden nun neue Sensordaten abgefragt, so berechnet jedes Neuron die euklidische Distanz zwischen erfassten Daten und CV und versendet diese Information gemeinsam mit der eigenen ID an die anderen Zellen. Nun kann jedes Neuron den Gewinner feststellen und wie weit dieser von ihm entfernt ist. Diese Informationen reichen nun aus, um jedem Neuron die Möglichkeit zu geben, seinen CV zu verändern. Mit der richtigen Lernrate [Zel94] konnte für jeden trainierten Kontext ('Licht an', 'Leute die sich unterhalten bei Licht', 'Leute die sich in der Dunkelheit unterhalten' und 'Heizung an') eine eindeutige Zuordnug zu einem Neuron erzielt werden.

Zuletzt kann auf eine weitere Implementierung eines KSOM's verwiesen werden, unter vielen in der Literatur, die aber in Kapitel 6.1 behandelt wird.

Desweiteren zeigt die Beliebtheit der KSOM, daß dieses Verfahren wohl sehr gute Ergebnisse in der derzeitigen Kontextaquisition erbringt.

4.2 Hidden Markov Models (HMM)

HMM's [RJ93] können als spezielle Form probabilistischer endlicher Automaten angesehen werden. Sie besitzen eine endliche Menge von Zuständen Q mit N Elementen. Die Übergänge zwischen den Zuständen sind über Übergangswahrscheinlichkeiten $\Lambda = a_{ij}$ definiert, wobei

$$a_{ij} = p\{q_{t+1} = j | q_t = i\} \text{ mit } a_{ij} \geq 0 \text{ für } 1 \leq i, j \leq N,$$

unter der Nebenbedingung

$$\sum_{j=1}^{N} a_{ij} = 1, 1 \leq i \leq N,$$

die Wahrscheinlichkeit eines Überganges von Zustand q_t in einen Zustand q_{t+1} ist. Jeder Zustand geht nur aus dem vorherigen hervor. Wie schon der Begriff *hidden* (versteckt) ausdrückt, ist dies von aussen nicht zu sehen. Nach *Aussen* wird nur eine Ausgabe weitergegeben. Dies ist mit der Wahrscheinlichkeitsverteilung, in jedem der Zustände, von $B = \{b_j(k)\}$ beschrieben. Hierbei ist mit

$$b_j(k) = p\{o_t = \nu_k | q_t = j\} \text{ mit } 1 \leq j \leq N, 1 \leq k \leq M$$

die Wahrscheinlichkeit einer Aussgabe eines Symbols ν_k gemeint, in einem Zustand q_t. Das Element ν_k ist k-tes Symbol im Alphabeth mit M Symbolen und o_t der derzeitige Parameter-Vektor. Es gilt wieder die Nebenbedingung

$$\sum_{j=1}^{M} b_j(k) = 1,\ 1 \leq j \leq N \text{ und } b_j(k) \geq 0.$$

Nun muss noch die initiale Zustandsverteilung $\pi = \{\pi_i\}$ beschrieben werden:

$$\pi_i = p\{q_1 = i\} \text{ mit } 1 \leq i \leq N$$

Also setzt sich das HMM aus $\lambda = (\Lambda, B, \pi)$ zusammen.

Ein sehr interresante Implementierung [SSP98] eines HMM's rechnet aus einem Video, aufgennomen über zwei Kameras, die Position des Benutzers zurück. Die beiden Kameras befinden sich an einer Mütze befestigt auf dem Kopf, wobei eine den Boden mit Teilen des Gesichts und die andere in Laufrichtung filmt. Um die Position bestimmen und weiteren Intensionen des Benutzers vorhersagen zu können, muss die Architektur des Gebeäudes (es handelt sich hier um ein Indoor-Positionierungssystem), in dem sich der Benutzer bewegt, bekannt sein. Aus den zwei Kameras werden drei Bildausschnitte augewählt, aus denen durchschnittliche Werte über die Pixel der Rot-, Grün-, Blau- und Helligkeitsgrade berechnet

werden. Diese Werte ergeben den Merkmalsvektor. Ein Bildausschnitt wird aus der vorwärtsgerichteten Kamera entnommen, die anderen bilden den Boden und die Nase ab. Der Boden gibt Informationen in welchem Raum sich der Benutzer befindet, die Nase in welchen Lichtverhältnissen die Person steht. Trainiert wird das HMM mit einem 24,5min Video, das 87 Ortsübergänge enthält. Im Training wird eine statistische Grammatik erzeugt, die später in einem Test (weitere 19.3min Video, 55 Ortsübergänge) benutzt wird um die Räume zu gewichten, die als nächstes in der Bewegung vorhergesagt werden. Hierzu muss eine Annahme des aktuellen Aufenthaltsortes vorhanden sein. Der Rechner muss die Räume an den Übergängen Segmentieren und benennen. Die Exaktheit der Aussagen, wird mit $Acc = \frac{N-D-S-I}{N}$ betimmt, wobei N die Gesamtzahl der Räume ist, D die Anzahl der Raumwechsel die nicht erkannt wurden, S die Anzahl der Räume die falsch klassifiziert wurden und I die Menge der Raumübergänge die nicht stimmten. Das beste Resultat wurde mit einem HMM erzielt, daß über drei Zustände verfügt. Dieses HMM besaß für die Trainingsdaten eine Exaktheit von $Acc = 68,97\%$ und für den Test $Acc = 81,82\%$.
Es wurde noch versucht mit der vorwärts gerichteten Kamera Bewegungen mit den Armen zu erkennen, was in [SSP98] beschrieben ist.

Ein System das Anwendung im "Context Aware Home" findet, ist ein Boden, der mit Gewichtssensoren ausgestattet ist [AJLS97], der zur Erkennung verschiedener Laufmuster und Gegenständen auf dem Boden dient. Diese Sensoren sind unter Teppichkacheln (die mit Sperrholz- und Metallplatten unterlegt werden) jeweils an den Ecken angebracht. So entsteht ein Gitter, wobei die Sensoren an den Schnittpunkten angebracht sind. Um die HMM's zu erstellen, wurde das Hidden Markov Modell Toolkit Vers.1.3 (HTK) benutzt. Entwickelt wurde das HTK von Young [You93].
Die Sensordaten wurden in Signaturen von jeweils 20 pro Person, für 15 verschiedene Personen, in einer Datenbank gespeichert. Es wurden 150 Signaturen zum Training und 150 für den Test verwendet. Eine Signatur ist ein diskretes Signal mit Z Messpunkten, aus der eine Beobachtungssequenz $o_1, .., o_T$ erzeugt wird. Es wird immer ein Ausschnitt (Fenster) der Signatur betrachtet, der L Messwerte beinhaltet. Aufeinander folgende Fenster dürfen sich maximal um M Messpunkte überlappen. Diese L Messwerte werden in einem Vektor o_t zusammen gefasst. Die Länge der Beobachtungssequenz T ist somit:

$$T = \lceil \frac{Z - L}{L - M} \rceil + 1$$

Die Ergebnisse der HMM's finden sie in Fig.11, wobei der Fehler in Abbhängigkeit von der Anzahl der Zustände des jeweiligen HMM's dargestellt ist.
Zuletzt möchte ich wieder auf das Kapitel 6.1 verwisen, in dem eine Anwendung beschrieben wird, die ein Markov Modell erster Ordnung verwendet.

Fig. 11. Durchschnittliche Fehlerrate in Abängigkeit von der Anzahl der Zustände N des HMM's

5 Repräsentationen von Wissen und Schließen aus vorhandenem Wissen

5.1 Fuzzy Logic (Unscharfe Mengen)

Nicht wie in der klassischen Mengenlehre, wird bei unscharfen Mengen klare Grenze definiert, ob ein Element zu der Menge gehört oder nicht. Es wird viel mehr ein Zugehörigkeitsgrad $\mu_A(x)$ definiert, der aussagt zu welchem Grad ein Element x zu einer Menge A gehört [Tra94]. Dies ist auch nicht im Sinne der Wahrscheinlichkeitstheorie zu verstehen, in welcher diese Funktion die Wahrscheinlichkeit einer Zugehörigkeit ausdrückt. Das Element mit einem Zugehörigkeitsgrad von z.B. $\mu_A(x) = 0.7$ gehört zu der Menge A zu 70%. Hierzu werden die klassischen Venn-Diagramme (Fig.12) um einen Bereich um die Menge A mit $\mu_A(x) = 1$ erweitert, der die unscharfe Menge $(0 < \mu_A(x) < 1)$ beschreibt. Über diese

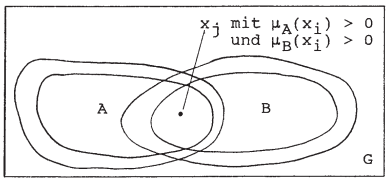

Fig. 12. Ein Venn-Diagramm mit zwei unscharfen Mengen $A \bigcap B$

Zugehörigkeitsfunktion kann dies nun im zweidimensionalen, mit z.B. zwei Mengen A und B (Fig.13), sehr anschaulich visualisiert werden.
Nun können die klassischen Logikoperatoren definiert werden. Im Gegensatz zu der Boolschen Algebra, sind die Operatoren in der Fuzzy Logic nicht eindeutig festgelegt, vielmehr gibt es verschiedene Definitionen für UND und ODER.

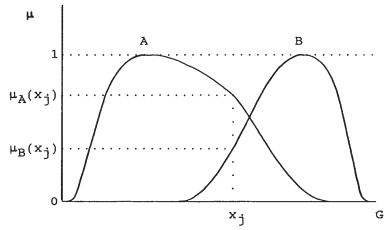

Fig. 13. Schnitt zweier unscharfen Mengengraphen

UND:

$$\mu_{AundB}(x) = \min(\mu_A(x); \mu_B(x)) \tag{1}$$

$$\mu_{AundB}(x) = \mu_A(x)\mu_B(x) \tag{2}$$

$$\mu_{AundB}(x) = \max(0; [\mu_A(x) + \mu_B(x) - 1]) \tag{3}$$

ODER:

$$\mu_{AoderB}(x) = \max(\mu_A(x); \mu_B(x)) \tag{4}$$

$$\mu_{AoderB}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \tag{5}$$

$$\mu_{AoderB}(x) = \min(1; [\mu_B(x) + \mu_B(x)]) \tag{6}$$

NEGATION:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{7}$$

Eine Arbeit die sich nur mit Fuzzy Logic beschäftigt [KK02], wurde an einem europäischen Institut erstellt. Es handelt sich bei dieser Veröffentlichung um ein Client / Server Modell, das dem Client Informationen über seine Umgebung bereitstellt. Jeder Server deckt eine bestimmte Region (Domäne) mit Daten ab. Die Empfänger der Daten sind mobile Geräte die über ein Satelliten Positionierungs System und eine mobile Kommunikationsverbindung verfügen.
Auf dem Server befindet sich nun eine Datenbank, die Daten der abdeckbaren - *Sendereichweite der mobilen Kommunikationsverbindung* - Umgebung enthält. Jeder Punkt (Location Data Point, LDP) der Umgebung, zum Beispiel Museen, Geschäfte, Attraktionen, etc., der von Interesse für einen mobilen Benutzer sein könnte, wird in Form eines Vektors in der Datenbank gespeichert. Dieser Vektor enthält die geographische Lage, Namen und Attribute des Punktes (LDP). Die Attribute sind für alle Punkte der Serverdomäne die gleichen, wobei die Zugehörigkeitsfunktion $\mu_i(x)$ die Zugehörigkeit des LDP i zu dem jeweiligen Attribut x ausdrückt. So bedeutet z.B. eine Zugehörigkeit zu dem Attribut $x = \textit{Parkplätze vorhanden}$ von $\mu_i(x) = 0,2$, daß wohl wenige bis keine Parkplätze an einem Punkt i (*Kunstmuseum*) vorhanden sind.
Ein Benutzer möchte jetzt eine Information (*was gibt es hier?*) von seiner Umgebung erhalten. Hierfür muss er dem System (zuständiger Server) einen, ihm

zuvor zur Verfügung gestellten, ausgefüllten *Fragebogen* zusenden. Dieser *Fragebogen* enthält die Interessen des Benutzers zu den einzelnen Attributen. Nun werden in einer Matrix die LDP-Vektoren kominiert mit den Benutzerinteressen gespeichert. Um den optimalen Punkt der Umgebung des Benutzers, heraus zu finden, der das Interesse am besten deckt, werden fuzzy Operationen durchgeführt. Zuerst werden alle zuvor mit den Benutzerwünschen kombinierten Attribute der LDP's "UND"-Verknüpft (Minimumoperator), danach das Maximum (ODER) unter allen LDP's emittelt. Dieser so gefundene LDP müsste die Benutzerwünsche best möglich abdecken.
Somit wurde mit Hilfe einer unscharfen Logik ein System geschaffen, das Aufgrund von Benutzerinformationen eine Umgebungskarte erstellt, die sich für jede Person, die das System benutzt, anders darstellt.

Eine Infrastruktur, die u.a. Fuzzy Logic benutzt, wird in Kapitel 6.2 besprochen.

5.2 Ontologien

Formale Ontologien [Gru93] werden derzeit darauf untersucht, ob sie ein Weg darstellen, um Inhalts spezifische Vereinbarungen, für das Bereitstellen und Wiederbenutzen von Wissen unter verschiedenen Softwareeinheiten, definieren zu können.
In die Umgangssprache Übersetzt bedeutet Ontologie: die Wissenschaft des Seienden. Hieraus lässt sich für die Wissensrepräsentation ableiten, daß nur repräsentiert werden kann was existiert.
Eine Menge von Objekten wird in einem repräsentativen Vokabular formuliert, das Wissen repräsentiert. Formell ist Ontologie die Aussage einer logischen Theorie. Jede Ontologie definiert eine Menge von Klassen, Relationen, Funktionen und Objektkonstanten für eine Domäne eines Diskurs. Sie beinhaltet weiterhin eine Axiomatisierung um die Möglichkeiten der Interpretationen zu beschr̈anken.
Will eine Task einen Diskurs in einer bestimmten Ontologie führen, so muss sie sich ihr *verpflichten*. Eine Task kann sich bezüglich einer Ontologie *verpflichten*, wenn ihre beobachtbaren Aktionen konsistent mit den Definitionen der Ontologie sind. Ontologische *Verpflichtungen* sind Vereinbarungen, das geteilte Vokabular in einer kohärenten und konsistenten Weise zu benutzen. Eine Task die einer Ontologie *verpflichtet* ist, weiß Dinge der Wissensbasis, die andere Tasks nicht wissen. Dies bedeutet auch, daß nicht alle Anfragen beantwortet werden müssen die in dem Vokabular formuliert werden können.
Nicht wie Logiken sind Ontologien standartisiert, vielmehr gibt es Prinzipien des Designs. Die Prinzipien lauten wie folgt:

1. **Klarheit:** Eine Ontologie sollte effizient die beabsichtigte Bedeutung von definierten Bestimungen kommunizieren können. definitionen sollten effektiv sein. Wenn Definitionen in axiomen formuliert werden kann, dann sollte dies geschehen.
2. **Kohärenz:** Eine Ontologie sollte Folgerungen bedingen, die konsistent mit den Definitionen sind. Es sollten wenigstens die Axiome logisch konsistent

sein. Wenn ein Satz, der von Axiomen abgeleitet werden kann, mit den Definitionen oder Beispielen die informell gemacht wurden im Konflikt steht, so ist die Ontologie inkohärent.
3. **Ausdehnungsfähigkeit:** Eine Ontologie sollte von einer Task, die sich zu ihr verpflichtet hat, eigenständig erweitert oder spezialisiert werden können. Das muss in Konsistenz mit den Definitionen geschehen.
4. **Minimale Kodierungs-Tendenz:** Das Konzept sollte auf einer Wissensebene spezifiziert werden, ohne von einer bestimmten Symbolik abhängig zu sein. Kodierungs-Tendenzen sollten minimiert werden, weil Tasks die Wissen austauschen in verschieden repräsentierten Systemen implementiert sein können.
5. **Minimale ontologische Verpflichtungen:** Eine Ontologie sollte so wenig Behauptungen aufstellen, bezüglich der Welt die modelliert wird, als möglich. Dies erlaubt den Parteien, die sich der Ontologie vepflichtet haben, die Ontologie zu spezialisieren und zu realisieren wie dies gebraucht wird. Erreicht wird dies mit der schwächsten Theorie, die die essentiellen Therme benutzt.

Da meist nicht alle Prinzipien des Designs im gleichen Maße eingehalten werden können, müssen zum Teil Kompromisse eingegangen werden.

6 Zwei konkrete Implementierungen

6.1 Unüberwachte, dynamische Identifikation von Physiologischem- und Aktivitäts-Kontext in tragbaren Rechnern [KSSF03]

Ein tragbares System, welches bestimmte Benutzer-Kontexte und kontextuelle Übergangswahrscheinlichkeiten online, ohne Überwachung von Aussen, erkennen kann, wird in [KSSF03] beschrieben. Um dem Benutzer ein möglichst benutzbares System zu bieten, wurde ein gut tragbares Sensor-Armband (Fig.14) in Verbindung mit einem "Tablet-PC" als Hardware gewählt. Das Armband verfügt über Sensorik der Beschleunigung, der galvanischen Hautresonanz, der Hautemperatur, der körpernahen Temperatur und eine drahtlose Kommunikationsverbindung. Als Detektor für die Herzfrequenz, dient ein drathloser Sensor der auf der Brust angebracht wird. Die Sensordaten werden im Armband gespeichert oder gleich über Funk an einen PC weiter gegeben. Ein Knopf am Arband gibt die Möglichkeit Zeitstempel zu sammeln. Nun wurden zwei Verfahren entwickelt um die Sensordaten auf kontextuelle Aussagen abzubilden, ein "online"- und ein "offline"-Algorithmus. Der "online"-Algorithmus geht aus dem "offline" hervor, mit der Anpassung an einen kontinuierlichen Datenstrom gegenüber gespeicherten Daten. Aus diesem Grund wird hier im Folgenden nur der "offline"-Algorithmus dargestellt.

Der "offline" Algorithmus kombiniert verschiedene schon zuvor ganannte Verfahren.

Zuerst werden die Messwerte der Sensoren auf verarbeitbare Datenvektoren abgebildet. Die Sensoren die einen geringen Wertestrom liefern, wobei hier alle

Fig. 14. Das "SenseWear" - Armband

Sensoren bis auf den Beschleunigungs-Sensor gemeint sind, werden auf dem Armband durch Mittelwert oder absolute Differenz in Datenvektoren umgewandelt und gespeichert. Diese Vektoren ergeben dann einen 8 dimensionalen Merkmalsraum. Die Beschleunigungs-Werte, die acht mal pro Sekunde gemessen werden, werden durch eine FFT (Fast Fourier Transformation) in einen 128 dimensionalen Merkmalsraum projiziert. Um jeder Dimension der Merkmalsvektoren gleiche Bedeutung zukommen zu lassen, werden die Daten normalisiert. Da das nachher verwendete neuronale Netze schlecht mit hoch dimensionalen Datenräumen arbeitet, wurde eine Dimensionsreduktion mittels PCA (Principal Component Analysis) durchgeführt.

Die nun erhaltenen Merkmalsvektoren bilden die Eingabe für eine KSOM. Es wurde eine hexagonale Topologie der SOM gewählt, die eine Kartenstruktur von 20×23 Zellen. Das KSOM ist bereits ein clusterisierender Prozess, jedoch die CV müssen noch gruppiert werden. Wie schon in [Lae01] vorgeschlagen, wird auf das KSOM ein k-means Clustering angewendet. Hierbei wurde für eine niedrige Datenrate - *womit alle Sensoren außer dem Beschleunigungssensor gemeint sind* - eine Anzahl von 14 Clustern gefunden.

Da abrupte Kontextwechsel (z.B. Laufen $\longrightarrow$ Rennen) den lernenden Charakter des Systems negativ beeinflussen, wird ein Markov Modell erster Ordnung trainiert, das Cluster der SOM zusammenfasst. Somit werden die Cluster die zu *ähnlich* sind zu einem vereinigt. Jeder Zustand, der in dem erstellten Markov-Graphen eine zu geringe Wahrscheinlichkeit hat, eine Schleife auszubilden, wird aus dem Graphen entfernt. Nach dieser Reduktion wurden die Cluster in ihrer

Anzahl auf 5 reduziert.

Mit weiteren Iterationen des k-means Clustering, werden nun die Vektoren von weggefallenen Clustern anderen Clustern zugeordnet.

Mit dieser Vorgehensweise wurde nun mit verschiedenen Testdaten an Studien herangegangen.

Resultate und Bewertung der Studien

1. **Die erste Studie** wurde auf Daten aufgebaut, die im normalen Tagesablauf von zwei Testpersonen gesammelt wurden. Durch sammeln von Zeitstempeln bei subjektiv wahrgenommenen Kontextwechseln, wurde eine Vergleichsmenge zu den Ergebnissen des Systems geschaffen. Das System wurde mit beiden Algorihmen getestet, dem "online"- und dem "offline"-Algorithmus. Es wurde festgestellt, daß sich die Methode bis zu einem gewissen Grad mit der subjektiven Klassifikation messen kann. Ein Fehler der Methode trat immer bei der Klassifikation des Kontextes *Pendeln* auf, was immer als *arbeiten im Büro* falsch klassifiziert wurde. Weitere Fehler konnten mit einer zu strikten Eliminierung von Clustern im Markov Graphen begründet werden. In Figure 15 sehen sie die Resultate zussamengefasst, wobei oben die subjektiven Zeitstempel mit Bennenung, in der Mitte die Sensordaten und unten die Cluster nach Ausführung der Algorithmen abgebildet sind.

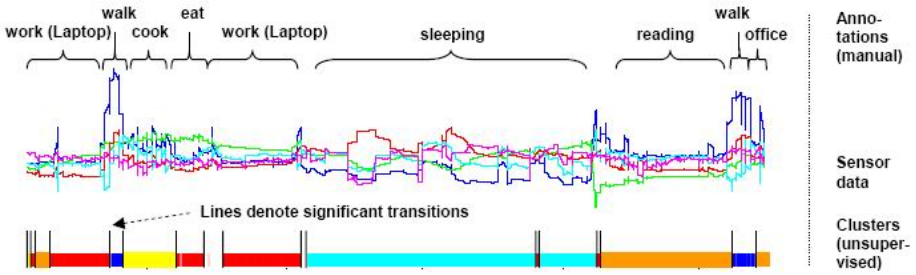

Fig. 15. Resultate des Clusterings

2. **Studie der Bewegungen**
   Es wurden Daten, in einem Zeitraum von drei Minuten, typischer menschlicher Bewegungen einer Testperson aufgezeichnet. Folgenge Bewegungen wurden in gegebener Reihenfolge ausgeführt: Laufen(1), Rennen(2), Laufen(1), Sitzen(3), Kniebeugen(4), Laufen(1), Wedeln mit den Armen(5), Laufen(1) und Treppen auf- und ablaufen(6). Es wurden sechs Cluster erkannt, also genau den Arten der Bewegungen entsprechend.
   Nun wurde das trainierte Netz auf Daten zum Vergleich angewendet, wobei

sehr gute Resultate fest zu stellen waren. Auch kleine Bewegungsunterschiede wurden innerhalb von vier Sekunden erkannt.

3. **Vergleichsprüfung der Sensoren**
   Auf eine 20 stündige Datenreihe angewendet, wurde nun die Bedeutung der einzelnen Sensoren getestet. Es wurde nacheinander jeweils ein Sensor abgeschaltet und die Resultate des Algorithmusses betrachtet. Die meisten unstabilen Cluster wurden von den Beschleunigungs-Sensoren produziert, deren Summe der absoluten Differenz berechnet wurde. Speziell der Zustand *Schlafen* wurde mit verwendeten Beschleunigungs-Sensoren schlecht erkannt.
4. **Resultate der Rechenleistung**
   Da das System in Matlab erstellt wurde, kann es auf verschiedene Prozessoren übersetzt werden. Der "online"-Algorithmus müsste für einen StrongArm mit 400MHz ausführbar sein, da der Algorithmus nur einen Puffer von 1,2MByte benötigt.
   Es wird leider nur die Hypothese aufgestellt und nicht gesagt, ob dies wirklich getestet wurde.

6.2 Eine "Middelware" für Kontext bewusste Agenten in ubiquitären Systemen [RC03]

7 Literatur

References

[AJLS97] M.D. Addlesee, A.H. Jones, F. Livesey, and F.S. Samaria. *The ORL Active Floor.* The Olivetti and Oracle Research Laboratory, 1997.

[BAW94] B.Schilit, N. Adams, and R. Want. Context-aware computing applications. *1st International Workshop on Mobile Computing Systems and Applications*, 1994.

[BBC97] P.J. Brown, J.D. Bovey, and X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*, 1997.

[BTW00] Gerd Beuster, Bernd Thomas, and Christian Wolff. *MIA An Ubiquitous Multi-Agent Web Information System.* Institut für Informatik, Universität Koblenz, Rheinau 1, 56075 Koblenz, Germany, 2000.

[CLS02] Elaine Catterall, Kristof Van Laerhoven, and Martin Strohbach. *Self-Organization in Ad Hoc Sensor Networks: An Empirical Study.* Computing Department Lancaster University, Lancaster LA1 4YR, United Kingdom, 2002.

[DA99] Anind K. Dey and Gregory D. Abowd. The context toolkit: Aiding the development of context-aware applications. In *Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing*, 1999.

[DA00] Anind K. Dey and Gregory D. Abowd. *Towards a Better Understanding of Context and Context-Awareness.* Graphics, Visualization and Usability Center and College of Computing, Georgia Institute of Technology, Atlanta, GA, USA 30332-0280, 2000.

[DAW99] A.K. Dey, G.D. Abowd, and A. Wood. Cyberdesk: A framework for providing self-integrating context-aware services. *Knowledge-Based Systems*, 1999.

[Dey98] Anind K. Dey. Context-aware computing: The cyberdesk project. Technical report, AAAI 1998 Spring Symposium on Intelligent Environments, 1998.

[FJ98] D. Franklin and Flaschbart J. All gadget and no representation makes jack a dull environment. Technical report, 1998.

[Gru93] Thomas R. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing.* Stanford Knowledge Systems Laboratory, 701 Welch Road, Building C, Palo Alto, CA 94304, 1993.

[KK02] EVANGELOS KOTSAKIS and MICHALIS KETSELIDIS. *Modeling Personalized and Context Sensitive Behavior for Location Aware Services by Employing Fuzzy Logic.* Institute for the Protection and the Security of the Citizen (IPSC), Joint Research Center, European Commission, Via Fermi 1, TP261, I-21020 Ispra (VA), ITALY, 2002.

[KMK+03] P. Korpipäa, J. Mäntyjärvi, J. Kela, H. Keränen, and E-J. Malm. Managing context information in mobile devices. Technical report, 2003.

[KSSF03] Andreas Krause, Daniel P. Siewiorek, Asim Smailagic, and Jonny Farringdon. *Unsupervised, Dynamic Identification of Physiological and Activity Context in Wearable Computing.* Munich University of Technology and Carnegie Mellon University and BodyMedia Inc., 2003.

[Lae01] Kristof Van Laerhoven. *Combining the Self-Organizing Map and K-Means Clustering for On-line Classification of Sensor Data.* Starlab Research Laboratories, Englandstraat 555, B-1180 Brussels, Belgium, 2001.

[MKVA03] Esko-Juhani Malm, Jouni Kaartinen, Elena Vildjiounaite, and Petteri Alahuhta. Smart-it context architecture. Technical report, Project group of Smart-Its, 2003.

[Pas98] J. Pascoe. Adding generic contextual capabilities to wearable computerse. *2nd International Symposium*, 1998.

[RC03] Anand Ranganathan and Roy H. Campbell. *A Middleware for Context-Aware Agents in Ubiquitous Computing Environments.* Department of Computer Science University of Illinois, Urbana-Champaign, USA, 2003.

[RF03] Marcela Rodriguez and Jesus Favela. *A Framework for Supporting Autonomous Agents in Ubiquitous Computing Environments.* Departamento de Ciencias de la Computación, CICESE, Ensenada, México, 2003.

[RJ93] Lawrence R. Rabiner and B-H. Juang. *Fundamentals of Speech Recognition*, chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Prentice-Hall, 1993.

[RM02] Cliff Randell and Henk L. Muller. *The Well Mannered Wearable Computer.* Department of Computer Science, University of Bristol, Bristol, U.K., 2002.

[RPM97] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological. *Computer Applications in Archaeology*, 1997.

[Sch02] Albrecht Schmidt. *Ubiquitous Computing Computing in Context.* PhD thesis, Lancaster University, 2002.

[SL01] A. Schmidt and K. Laerhoven. How to build smart appliances? *IEEE Personal Communications*, 2001.

[SSP98] Thad Starner, Bernt Schiele, and Alex Pentland. *Visual Context Awareness via Wearable Computing.* Media Laboratory, Massachusetts Institute of Technology, 1998.

[ST94] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 1994.

[Tra94] Dirk H. Traeger. *Einführung in die Fuzzy-Logik.* B.G.Teubner Stuttgart, 1994.

[You93] S.J. Young. The htk hidden markov model toolkit: Design and philosophy. Technical report, Department of Engineering, Cambridge University (UK), 1993.

[Zel94] Andreas Zell. *Simulation Neuronaler Netze.* Oldenbourg, 1994.