Studiengang:	Informatik
Prüfer:	Prof. Dr. K. Rothermel
Betreuer:	Dipl. Inf. G. Schiele
begonnen am:	06. März 2002
beendet am:	06. September 2002
CR-Klassifikation:	C.2.4, C.3, H.3.4

Studienarbeit Nr. 1849

Entwicklung eines Gerätes zur Erfassung des Umgebungskontextes *ContextCube*

Nikolaus Hörr

Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20–22
D–70565 Stuttgart

Abstract

Der Einsatz von mobilen Geräten in Form von Notebooks, Palmtops und Handies ist inzwischen schon stark verbreitet. Die weitere Zukunft wird von einer noch stärkeren Gegenwart von Rechnern, selbst in normalen Alltagsgegenständen, geprägt sein, so daß wir in wenigen Jahren von einer Vielzahl davon umgeben sein werden, die wir in dieser Form gar nicht mehr wahrnehmen. Viele Anwendungen in diesem Bereich benötigen Informationen über ihre Umgebung und müssen dafür bisher ihre "eigenen Sensoren" mitbringen. Dies bedeutet einen erhöhten technischen Aufwand für jedes einzelne Gerät. Zur Verbesserung können die Sensoren in die Umgebung verlagert werden.

Dazu wird in dieser Arbeit im Rahmen des CANU-Projektes ein Gerät entwickelt, welches vielfältige Kontextinformationen in seiner (zunächst stationären) Umgebung sammelt und über ein Netzwerk anderen Geräten zur Verfügung stellt.

Als Basis dieses System dient die TINI-Plattform. Es handelt sich dabei um ein eingebettetes System, das in JAVA programmierbar ist. In dieser Arbeit wird die TINI-Plattform unter anderem um Sensoren für Temperatur, Helligkeit und Lautstärke erweitert, sowie Softwarekomponenten entwickelt, um diese Sensorinformationen anderen Geräten über das Internet zur Verfügung zu stellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Aufgabenstellung

Die Aufgabe dieser Arbeit ist die Entwicklung eines Gerätes zur Erfassung des Umgebungskontextes. Dieses Gerät wird im folgenden als **ContextCube** bezeichnet.

Als Basishardware dient dabei die TINI-Platform [5]. TINI ist die Abkürzung für das in JAVA programmierbare Tini InterNet Interface, welches von Dallas Semiconductors erhältlich ist.

In dieser Arbeit wird die TINI-Platform um Sensoren für Temperatur, Helligkeit und Lautstärke sowie gegebenenfalls weitere erweitert. Dazu gehört, neben dem Aufbau und Anschluss der Sensoren, auch die Entwicklung einer Softwarekomponente um anderen Geräten die Sensorinformationen über das Internet zur Verfügung zu stellen.

1.2 Anwendungen

Eine Anwendungsmöglichkeit ist eine Heizungssteuerung oder sogar die Steuerung eines ganzen Raumklimatisierungssystems, für die der ContextCube zum Beispiel Temperatur und Luftfeuchtigkeit misst. Zusätzlich kann aus den weiteren Sensorinformationen ermittelt werden, ob der Raum gerade in Benutzung ist, wodurch die Sollwerte automatisch angepasst werden könnten. Dadurch liessen sich zusätzlich Heizkosten einsparen.

Eine andere Anwendung wäre es, die Informationen über Beleuchtung und Lautstärke — bei Räumen mit Fenstern evtl. in Verbindung mit der Information von weiteren Sensoren für die Aussenhelligkeit — zu interpretieren und daraus heuristisch zu ermitteln, ob zum Beispiel ein Hörsaal gerade belegt ist oder ob eine Veranstaltung bereits beendet ist.

Denkbar wäre auch, dass automatisch das Licht eines Raumes angeschaltet wird, wenn — durch die Sensoren des ContextCube — festgestellt wird, dass in einem zu

dunklen Raum an einem Monitor gearbeitet wird. Wenn das Licht nicht automatisch geschaltet werden kann, reicht auch eine akkustische Warnung. Dabei könnte auch die Lautstärke akkustischer Signale an die momentanen Anforderungen angepasst werden.

Kapitel 2

Anforderungsanalyse

In dieser Arbeit wird ein erweiterbares System zur Erfassung von Kontextinformationen aus der unmittelbaren Umgebung entwickelt und prototypisch realisiert.

Die Aufgaben des ContextCube lassen sich grob in drei Bereiche aufteilen:

- Sammeln von Kontextinformationen
- Bereitstellung der Daten über Netzwerk (für entfernte Benutzer)
- Anzeige der Daten (für lokale Benutzer)

2.1 Sammeln und Protokollieren vielfältiger Kontextinformationen aus der Umgebung

Für diese Implementation sind fünf Sensoren geplant. Drei davon sind durch die Aufgabenstellung vorgegeben (Temperatur, Helligkeit und Lautstärke). Um möglichst vielfältige Sensorinformationen zur Verfügung stellen zu können, wurden noch weitere Sensoren für die Luftfeuchtigkeit und die Frequenz der Beleuchtung hinzugefügt. Die Sensoren werden über den 1-Wire Bus angeschlossen, der im Kapitel 4 genauer beschrieben wird.

Temperatur Zur Messung der Temperatur steht bereits ein fertiger Sensor zur Verfügung — der Temperature iButton DS1920 [13][14]

Lautstärke Da kein fertiger 1-Wire Sensor für die Lautstärke zur Verfügung steht, wird die Lautstärkemessung mittels einer NF[1]-Verstärkerschaltung und einem A/D-Wandler mit 1-Wire Anschluss realisiert.

[1] Niederfrequenz

Intensität der Beleuchtung Für die Helligkeit steht ebenfalls kein fertiger 1-Wire Sensor zur Verfügung, daher wird die Lichtintensität mit einer Spannungsteilerschaltung — aus einem lichtabhängigen Widerstand (LDR) und einem normalen (festen) Widerstand — und einem 1-Wire A/D-Wandler gemessen.

Verbessert:

Da sich der LDR stark nichtlinear verhält, war geplant anstelle der Spannungsteilerschaltung, einen Licht-Spannungswandler (Beispielsweise der normalerweise unter anderem bei Conrad-Elektronik [27] erhältliche TSL250, 251 oder 252) zu verwenden. Da dieser nicht rechtzeitig beschafft werden konnte, wurde als "Platzhalter" ein kleines Solarzellen-Array eingesetzt, dessen Ausgangsspannung (ebenfalls nicht linear) mit der Lichtintensität ansteigt.

Frequenz der Beleuchtung Um die Frequenz der Beleuchtung zu messen, werden über einen festen Zeitraum die Helligkeitsschwankungen gezählt. Dafür werden neben einem lichtempfindlichen Bauteil ein Zähler und ein Timer benötigt. Ein Zähler mit 1-Wire Anschluss wird von Dallas Semiconductors angeboten, als Timer kann die interne Uhr des TINI genutzt werden. Alternativ können die Impulse auch direkt, beispielsweise über einen Interrupt Anschluss, vom TINI gezählt werden.

Luftfeuchtigkeit Eigentlich gibt es von Dallas Semiconductors einen Humidity iButton unter der Bezeichnung DS1910 [15][16] oder DS-ADI-401, der ebenfalls nicht erhältlich war. Es steht aber ein Plan [16] des internen Aufbaus des DS1910 zur Verfügung. Daher wird eine leicht abgeänderte Selbstbauvariante dieses Sensors verwendet.

2.2 Anbieten der Daten für entfernte Benutzer oder Geräte

Die gesammelten Kontextinformationen sollen über eine oder mehrere Schnittstellen von anderen Geräten abgerufen werden können. Dabei muss auf jeden Fall die aktuelle Information abgerufen werden können, Informationen über den Verlauf anzubieten ist sinnvoll, aber nicht zwingend notwendig.

2.3 Statusanzeige für lokale Benutzer

Um die Daten des ContextCubes auch manuell ablesen zu können sollte ein LCD[2] vorhanden sein auf welchem die Messwerte angezeigt werden. Das Display muss mindestens einen einzelnen Messwert samt, eventuell verkürztem, Bezeichner gleichzeitig

[2]Liquid Crystal Display

darstellen können, ohne dabei auf Laufschrift zurückzugreifen. Dafür ist beispielsweise ein LCD-Modul mit einer, aus 16 Zeichen bestehenden Zeile, ausreichend. Ein Display, das ohne störende Zeilenumbrüche alle (bis zu acht) Messwerte gleichzeitig anzeigen kann — dazu sind beispielsweise 4 Zeilen mit 40 Zeichen ausreichend — würde mit ca. 65 Euro den Kostenrahmen sprengen. Falls daher nur ein kleines LCD zur Verfügung steht, das nicht alle Messwerte gleichzeitig anzeigen kann, müssen die Werte abwechselnd angezeigt werden. Die Umschaltung zwischen den Werten kann in festen Zeitintervallen oder mittels eines Bedienungselementes erfolgen.

Kapitel 3

Schnittstelle des ContextCubes

Die Hauptaufgabe des ContextCubes ist, anderen Geräten Kontextinformationen zur Verfügung zu stellen, welche er aus seiner Umgebung sammelt. Dazu benötigt der ContextCube zunächst eine externe Schnittstelle, über die auf die Informationen zugegriffen werden kann.

Für diese Schnittstelle wurden die folgenden Möglichkeiten in Betracht gezogen:

- "NeXus"
- HTTP
- die Entwicklung eines eigenen, proprietären Protokolls

3.1 NeXus

Das NeXus Projekt [3] arbeitet mit einem Augmented World Model, welches in [1] genauer beschrieben wird. Es bietet den Vorteil eines — zumindest an der Universität Stuttgart — standardisierten Datenformats zum Austausch von Informationen.

Für die Kommunikation stehen verschiedene Sprachen zur Verfügung.

Da der ContextCube — ähnlich wie ein Spatial Model Server (SpaSe) [4] — Infomationen über ein bestimmtes (wenn auch im Vergleich weitaus kleineres) räumliches Gebiet anbieten soll, bietet sich ein AWQL-/AWML-SOAP- Interface, wie es auch im SpaSe eingesetzt wird, an. AWQL und AWML sind im NeXus Projekt genutze Sprachen, auf die in den folgenden (Unter-)Abschnitten noch genauer eingegangen wird. Aufgrund des sehr begrenzten Speichers des verwendeten Embedded Systems, wird bei dieser Variante nur ein Subset von AWQL implementiert.

Von einer Implementierung des ContextCubes als Client der nur in regelmässigen Zeitabständen AWQL update-Anweisungen an einen (Spatial Model) Server sendet wurde abgesehen, da der ContextCube zu beliebigen Zeitpunkten von verschiedenen

Anwendungen abfragbar sein soll, ohne die zwingende Notwendigkeit weiterer, zur Zeit nicht überall vorhandener, (NeXus-)Server. Eine zusätzliche externe Schnittstelle, die diese Funktionalität bietet indem sie nach jeder erfolgreichen Messung deren Ergebnis in einer AWML-update Anweisung an einen Spatial Server sendet wäre mit genügend Speicher recht einfach implementierbar.

Die Anfrage wird als SOAP Request [36], in den die eigentliche AWQL Anfrage eingebettet ist, an den ContextCube gesendet und die Antwort von diesem in Form eines SOAP Responses, der die eigentliche AWML Antwort enthält, zurückgesendt.

Da sowohl SOAP als auch AWQL und AWML XML-Sprachen sind, wird ein XML-Parser benötigt.

3.1.1 Augmented World Query Language

Die in [2] spezifizierte Augmented World Query Language ist, ähnlich wie SQL, eine Abfragesprache. Da AWQL recht mächtig ist (wenn auch weit weniger mächtig als SQL), sollten Einschränkungen vorgenommen werden um den kleinen Speicher nicht unnötig zu belasten. Dafür bieten sich bei Unterelemente des Restriction Elements an, da zur Auswertung einiger dieser Unterelemente (beispielsweise das Element in) Zugriff auf eine Datenbank benötigt wird.

Von einem “echten” Spatial Model Server werden die AWQL Elemente folgendermassen in SQL umgewandelt:

awql wird ignoriert.

update wird in die entsprechende SQL UPDATE Anweisung umgewandelt

generalization wird ignoriert.

aggregation wird ignoriert.

filter entspricht der SELECT-Klausel von SQL.

- includes gibt alle Attribute an, welche enthalten sein sollen.

- excludes gibt an, welche Attribute nicht enthalten sein sollen.

- attr name=“name” gibt an, welches Attribut enthalten sein soll (also SELECT Name).

- includeallother gibt an, dass nur die unter <excludes> angegebenen Attribute nicht enthalten sein sollen.

- excludeallother gibt an, dass nur die unter <includes> angegebenen Attribute enthalten sein sollen .

restriction entspricht der WHERE-Klausel in SQL.

- **Boolsche Ausdrücke (<and>, <or>, <not>)** werden zum entsprechenden Boolschen Ausdruck in SQL umgesetzt. Vor und nach dem Ausdruck steht jeweils ein Attribut mit den zugehörigen Daten. Das Attribut ist jeweils von <equal>, <in>, <inside> oder <overlaps> eingeschlossen.

- **equal** wird zu Attribut = Attribut.

- **in** wird zum Geo-Befehl IN von SQL.

- **inside** wird zum Geo-Befehl INSIDE von SQL.

- **overlaps** wird zum Geo-Befehl OVERLAPS von SQL.

- **closest** wird zum entsprechenden Geo-SQL-Befehl umgesetzt.

Da eine eigene Datenbank auf dem TINI mit einem Megabyte Speicher nicht realisierbar ist, werden die Elemente nicht in SQL umgewandelt. Stattdessen wird direkt bei der Auswertung der einzelnen Elemente ein spezieller Datentyp genutzt, der angibt, welche Objekte, in Form der Sensoren, und Attribute im Request angefordert wurden.

3.1.2 Augmented World Modeling Language

Die Augmented World Modeling Language - ebenfalls in [2] spezifiziert, kann für den benötigten Zweck recht einfach mit Hilfe von Templates erzeugt werden.

3.2 HTTP

Für HTTP-Schnittstellen gibt es mehrere Möglichkeiten:

3.2.1 Gewünschte Attributnamen im Querystring - Antwort auf Abruf aus Templates erzeugt (XML oder HTML)

Als Anfrage wird ein normaler GET-Request auf eine URL wie "http://ContextCube0001.de/sensor/?temperature&humidity" oder ähnlich gesendet. Die Antwort wird als HTML-Seite oder XML-Datei aus einem Template erzeugt. Möglich wäre auch, unterschiedliche Antwortformate anzubieten, wobei die Auswahl des Aufrufers ebenfalls im Querystring übermittelt werden kann, beispielsweise "...?temperature&XML-Format".

3.2.2 Gewünschte Attributnamen als Namen von entsprechenden Dateien

Der Aufrufer fordert mittels GET-Request eine Datei — beispielsweise "humidity.xml" — an, die die Informationen jeweils eines Sensors enthält. Falls verschiedene Formate benötigt werden, können mehrere Dateien pro Sensor existieren, z.B. zusätzlich "humidity.html". Die Dateien werden nach jeder (erfolgreichen) Messung aktualisiert.

3.2.3 Alle Informationen in einer Datei

Alle Sensorinformationen werden in einer Datei — beispielsweise "all.xml" — abgelegt, welche der Aufrufer selbst nach den für ihn relevanten Informationen durchsuchen muss. Um unterschiedliche Formate anbieten zu können, wären auch hier mehrere unterschiedlich aufgebaute Dateien möglich.

3.2.4 Mischformen

Da die drei zuvor genannten Möglichkeiten sich nicht gegenseitig ausschliessen, können sie auch in Kombination verwendet werden.

3.3 Proprietäres Protokoll

Als wahrscheinlich einfachste Möglichkeit bietet sich ein selbst entworfenes Protokoll an.

3.3.1 Keine Selektion

Der ContextCube sendet auf jede Anfrage eine Antwort, die alle verfügbaren Sensorinformationen enthält in einem vordefinierten Format.

3.3.2 Selektion von Sensoren

Um die Anfragedaten zu kodieren genügt ein Byte (bei maximal 8 Sensoren von denen mehrere gleichzeitig selektiert werden können). Alle Werte eines Sensor werden durch setzen des ihnen zugeordneten Bits im einem Byte, welches die Anfrage darstellt, in die Antwort mit eingeschlossen. Dabei ist die Formatierung der Werte für jeden Sensor identisch. Die Teilantworten werden einfach in einer besimmten Reihenfolge aneinandergehängt.

3.3.3 Selektion von Sensorattributen

Die Selektion von Sensorattributen ist nur sinnvoll, wenn die Software nicht nur einen Messwert liefert, sondern beispielsweise auch noch die Genauigkeit eines Sensors oder eine Zeitmarke seiner letzten Messung. Die Anfrage kann dabei in 5 Byte codiert werden, jeweils ein Byte für einen Sensor. Da für jeden Sensor mehrere Attibute (Messwert, Genauigkeit, Aktualität und eventuell weitere) zur verfügung stehen, geben die einzelnen Bits an, welche Attributwerte des jeweiligen Sensors in der Antwort enthalten sein sollen. Die Attributwerte haben (für jedes Attribut) eine besimmte feste Länge und werden in der angeforderten Kombination einfach aneinandergereiht zurückgesendet. Um die Anfragedaten zu kodieren genügt ein Byte pro Sensor problemlos. So können im selben Request auch für jeden einzelnen Sensor unterschiedliche Attribute angefordert werden. Prinzipiell würden auch ceil((#Sensoren * #Attribute pro Sensor)/8) Byte genügen. Diese Einsparung würde allerdings eine byteweise Verarbeitung des Requests komplizierter gestalten.

3.4 Diskussion und Auswahl

Ein selbst entworfenes Protokoll, das zunächst ausschliesslich der ContextCube beherrscht, wäre nur ein weiteres von vielen proprietären Datenformaten und hätte nur geringe Chancen sich durchzusetzen. Die Bereitstellung der Daten über eine normale HTTP-Schnittstelle wird ebenfalls als ein proprietärer Ansatz angesehen.

Ein gut erweiterbares, standardisiertes Protokoll, wie die im NeXus Projekt eingesetze Kombination aus AWQL und AWML wird am ehesten auf Akzeptanz bei Entwicklern (und sonstigen Nutzern) stossen. Ausserdem existieren bereits Komponenten, die über diese Schnittstelle auf den ContextCube zugreifen können.

Daher wurde die Variante NeXus gewählt, obwohl sie den grössten Aufwand bei Entwurf und Implementierung mit sich bringt.

Kapitel 4

Hardware

4.1 Werkzeuge

Für die Entwicklung der einzelnen Sensoren wurden folgende Geräte verwendet:

Netzgerät zur Spannungsversorgung des TINI und der Sensoren

Multimeter um Spannung, Strom und Widerstände zu messen

Oszilloskop um die Ausgangssignale der Sensoren und den korrekten Signalverlauf des 1-Wire Bus zu überprüfen

Lötkolben

PC zur Funktionsprüfung der Sensoren deren Daten der TINI anbietet.

4.2 Bauteilauswahl und Entwurf

4.2.1 TINI

Das Herzstück des ContextCubes bildet das von Dallas Semiconductors erhältliche Tini InterNet Interface (TINI) [5], ein in JAVA programmierbares Embedded System. Als Prozessor findet ein DS80C390 Verwendung. Der DS80C390 [11][12] arbeitet mit einer Taktfrequenz von 40 MHz bei vier bis fünf Takten pro Befehl und erbringt dadurch die Rechenleistung eines Standard 8051 mit 100 bis 120 MHz, der zwölf Takte pro Befehl benötigt. Der TINI besitzt neben einer Ethernet-Schnittstelle noch weitere externe Anschlüsse, für serielle (RS232-) Schnittstelle, 1-Wire- und CAN-Bus (Controller Area Network). Der 8-Bit Datenbus des Microcontrollers kann ebenfalls - zusammen mit einigen weiteren Status- und Steuerleitungen - extern abgegriffen werden.

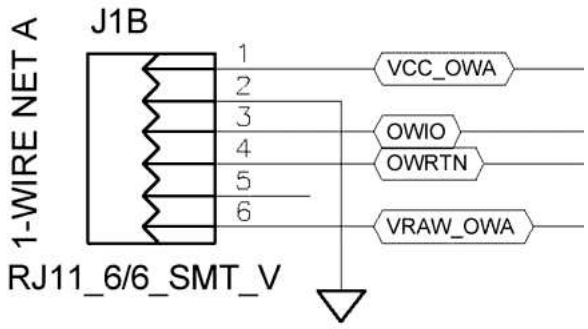

Abbildung 4.1: Anschlussbelegung der RJ11-Buchse auf dem TINI Socket (Quelle: http://www.systronix.com/tutor/tini_help/onewire/onewire480.jpg)

Die Kosten für das TINI Board mit 512kB RAM belaufen sich auf US$ 50, mit 1024kB RAM US$ 67, dazu kommt noch der Preis des verwendeten TINI Sockets US$ 35 sowie die Bearbeitungs- und Versandkosten.

Um auf dem TINI ausführbar zu sein, müssen die Class-Dateien in eine TINI-Datei umgewandelt werden. Dafür steht ein Programm namens TINIConvertor zur Verfügung.

4.2.2 1-Wire Bus für Sensoren — Bauteile

Als Schnittstelle zu den Sensoren bietet sich der 1-Wire Bus des TINI an. Er ermöglicht den Anschluss von praktisch beliebig vielen Sensoren, da jedes 1-Wire Bauteil eine einmalige 64-Bit Id besitzt, mit der es einzeln angesprochen werden kann. Der Anschluss erfolgt über eine sechspolige RJ11-Buchse auf dem TINI-Socket gemäss Abbildung 4.1.

Der 1-Wire Bus stellt eine AND-Verknüpfung in positiver Logik (bzw. OR-Verknüpfung in negativer Logik) dar. Praktisch bedeutet das eine Parallelschaltung aller 1-Wire Devices, wobei jedes Device den Bus auf low-Level ziehen kann. Dabei gibt es genau einen Bus Master und i.A. mehrere Slave devices. Die Datenleitung wird (masterseitig) über einen Widerstand (normalerweise ca. $5k\Omega$) auf eine Spannung von +5V gebracht (also Ruhezustand=high-Level).

Das bitweise Schreiben der Daten funktioniert masterseitig folgendermassen (die Numerierung der Punkte korreliert mit Abbildung 4.2):

1. der Master setzt den Bus zu Begin des Time Slots den 1-Wire-Bus auf low
2. 6μs Wartezeit
3. der Master schreibt das zu sendende Bit auf den Bus (ist dies eine 0, so wird der Bus weiter auf low gehalten, bei einer 1 kehrt der Bus auf high-Level zurück)

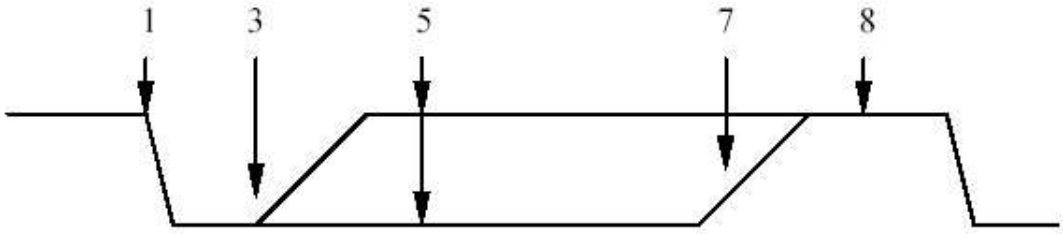

Abbildung 4.2: Signalverlauf des 1-Wire Bus (Quelle: http://pdfserv.maxim-ic.com/arpdf/AppNotes/app159.pdf)

4. 9μs Wartezeit
5. Auslesen des Bus-Levels (low level = 0, high level = 1)
6. 45μs Wartezeit
7. der Master lässt (nach einem geschriebenen 0-Bit) den Bus auf high-level zurückkehren
8. 10μs Wartezeit
9. Ausgabe des gelesenen Bits

Damit ein Slave ein Bit schreiben kann, muss der Master einen 1-Time Slot erzeugen, den der Slave - bei Bedarf - in einen 0-Time Slot umwandeln kann, indem er den Bus auf low-Pegel zieht. Mehrere Bits werden gelesen, indem entsprechend mehrere 1-Time Slots vom Master gesendet werden, die der Slave nach Bedarf in 0-Time Slots umwandeln kann.

Welches 1-Wire Device sendet wird im Normalbetrieb vom Master bestimmt, der dazu zuerst den Bus zurücksetzt, um alle 1-Wire Slaves, die potentiell in einem Wartezustand sind, zu reaktivieren. Im Laufe einer bitweisen Selektion (oder binären Suche im Fall des Temperature iButtons) werden alle Slaves, bei denen das aktuelle Bit ihrer 64-Bit ID nicht mit dem gewünschten übereinstimmt wieder in einen Wartezustand versetzt.

Der Busmaster kann durch Auslesen des Buspegels während des Schreibens, durch CRC-Checks und weitere Massnahmen in den meissten Fällen feststellen, wenn während der Kommunikation mit einem 1-Wire Bauteil Fehler auftreten. Diese Störungen können beispielsweise beim Austausch eines Sensors entstehen.

Geplante 1-Wire Devices:

DS1920 (Temperature iButton): Der DS1920 wurde nicht nur deshalb gewählt, weil er schon zu Begin der Arbeit zur Verfügung stand, sondern auch, da die vom Hersteller angegebene Toleranz bei nur 0.5°C liegt [14].

DS1910 (Hygro iButton): Der Hygro iButton ist ein anschlussfertiger 1-Wire Luftfeuchtigkeitssensor [15][16] , war aber nicht verfügbar.

DS2401 Silicon Serial Number: Der ContextCube kann anhand der Id feststellen, ob und welcher DS2401 [17][18] an den 1-Wire Bus angeschlossen ist. Dieses Bauteil ist beispielsweise zur Erkennung des Zustandes von Schaltern verwendbar.

DS2406 (Dual) Addressable Switch (mit Rückkopplung): Der DS2406 [19][20] ist beispielsweise zur Ansteuerung von optischen oder akustischen Signalen einsetzbar. Preis ca. 5 Euro

DS2423 (4 kbit 1-Wire RAM with Counter): Dieses Bauteil [21][22] besitzt vier 32-bit Counter, von denen hier nur die beiden mit externen Eingängen benutzt werden, um Schwankungen der Lichtintensität und Tastenbetätigungen zu zählen. Preis ca. 10 Euro

DS2438 (Smart Battery Monitor): Die vier DS2438 [23][24] werden als A/D-Wandler und als zusätzliche Temperaturfühler verwendet. Preis ca. 2,50 Euro

Diese Bauteile reichen für alle geplanten Sensoren als Interface zum 1-Wire Bus aus, da nur Spannungen zu messen und Impulse zu zählen sind.

4.2.3 Tini-Datenbus für LCD

Um die Daten auch ohne einen weiteren Rechner manuell auslesen zu können, besitzt der ContextCube ein LCD zur Anzeige der Messwerte. Neben jedem Messwert wird zusätzlich je ein kurzer Bezeichner angezeigt. So werden für jeden Sensormesswert maximal 16 Zeichen benötigt. Um die Bauteilkosten niedrig zu halten wurde ein Display mit nur einer Zeile bestehend aus 16 Zeichen gewählt, das zwischen den fünf (in einer erweiterten Version evtl. weiteren) möglichen Anzeigewerten umgeschaltet wird. Da beinahe alle standard LCD-Module einen Controller vom Typ KS066 oder HD44780 verwenden, die gleich anzusteuern sind, kann jedoch fast jedes andere Display angeschlossen werden. Die Umschaltung könnte automatisch in festen Zeitintervallen erfolgen, was den Bau von Bedienungselementen ersparen würde. Da die Lesegeschwindigkeit der Benutzer aber (evtl. stark) variiert, sind zwei Taster (up/down) zur Umschaltung vorgesehen. Dadurch muss ein Benutzer nicht mehr warten, bis ein bestimmter Messwert angezeigt wird, sondern kann diesen mit wenigen Tastenbetätigungen selber wählen.

Zusätzlich kann beim Start des ContextCube auf dem LCD auch für kurze Zeit die IP-Adresse des ContextCubes angezeigt werden, was den kurzfristigen Anschluss an ein Netz mit DHCP-Server zu Vorführzwecken erleichtert, indem es den Ping an die Broadcast-Adresse und die anschliessende Suche nach der MAC-IP Paarung im ARP-Cache erspart. Dieses Feature entfiel bei der Implementierung jedoch aufgrund der Speicherknappheit.

Vom direkten Anschluss eines LCD an den Datenbus wird auf verschiedenen TINI-related Websites wie [9] abgeraten, da der TINI ab einer Buslast von ungefähr $130pF$ nicht mehr zuverlässig arbeitet. Laut Datenblatt des TINI Boards [25] sinkt die maximal mögliche Betriebstemperatur bereits ab $21pF$ von 70°C auf 60°C und bei $35pF$ auf 50°C

Ausserdem besteht die Gefahr, den TINI durch Electrostatic Discharge (ESD) dauerhaft zu zerstören. Der Datenbus des TINI verträgt eine Spannung von minimal $-0.3V$ und maximal $V_{CC} + 0.3V$

Durch die Zwischenschaltung weiterer Logik-Gatter können diese Rückwirkungen zumindest in gewissen Grenzen gehalten werden.

4.3 Realisierung

4.3.1 Die Sensoren am 1-wire Bus

SMD auf Rastermaß 2,54mm

Die wichtigsten Bauelemente der Messschaltung sind ausschliesslich als Surface Mounted Device (SMD), mit Rastermaß 1,28 mm (oder teilweise auch noch kleiner) erhältlich. Da kein SMD-Werkzeug zur Verfügung stand, wurden die benötigten Anschlüsse der sechs- bis achtpoligen SMD-ICs (DS2423 und DS2438) mit angelöteten Kupferlackdrahtstücken versehen und so auf eine, für Experimentierplatinen im Rastermaß 2,54 mm, ausreichende Länge gebracht.

Temperatur

Die Temperaturmessung wird von einem Thermo iButton [14] erledigt. Zur Verbesserung des Messergebnisses können zusätzlich die Temperaturwerte der (als A/D-Wandler für andere Sensoren) angeschlossenen DS2438 [23][24] genutzt werden.

Luftfeuchtigkeit

Da der zuerst eingeplante Humidity iButton (DS1910) [15], trotz des von Dallas Semiconductors für das erste Quartal 2002 geplanten Verkaufsstarts, nicht verfügbar war, aber ein Schaltplan seines internen Aufbaus [16] verfügbar ist, wurde die Schaltung aus mehreren Bauteilen aufgebaut.

Als analoger Luftfeuchtefühler wurde der von Honeywell hergestellte HIH-3610 verwendet. Dieser Fühler ist der Nachfolge- und Austauschtyp [28] des im Originalschaltplan angegebenen — jedoch nicht mehr lieferbaren — HIH-3605. Er liefert eine lineare Ausgangsspannung die (bei $V_{DD} = 5,0V$) zwischen 0,8V und, je nach Temperatur, 3,50V bis 4,07V liegt. Die von Honeywell angegebene Genauigkeit (bei

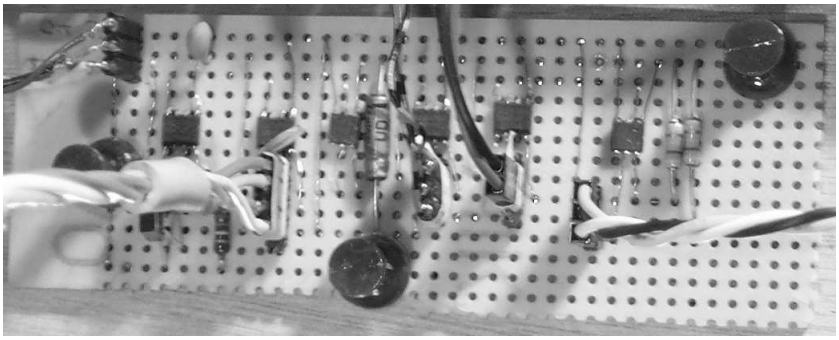

Abbildung 4.3: Der Sensor für die Luftfeuchtigkeit als Baugruppe zusammen mit allen anderen 1-Wire Bauteilen

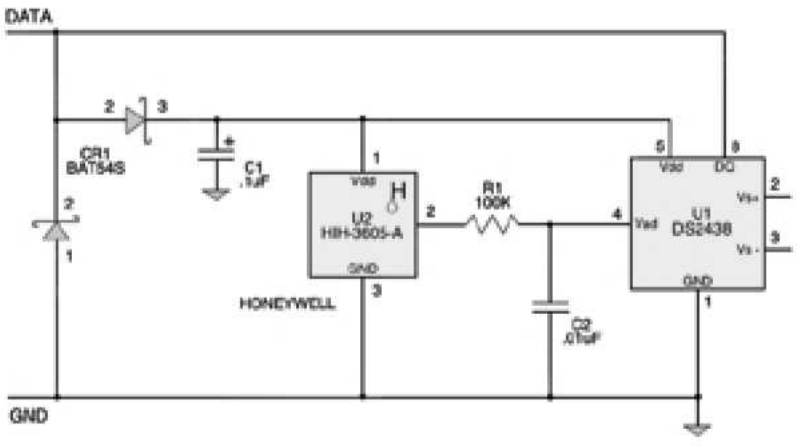

Abbildung 4.4: Originalschaltplan des DS1910 Humidity iButtons

$V_{DD} = 5,0V$) beträgt +-2%. Das analoge Signal wird von einem Tiefpass geglättet und an einen Smart Battery Monitor (DS2438) weitergeleitet. Dieser misst nicht nur die Spannung am Tiefpassausgang sondern auch die Versorgungsspannung und die Temperatur. Somit kann der Messwert temperatur- und spannungskompensiert werden.

Die vom Fühler gemessene (bereits spannungskompensierte) Luftfeuchtigkeit lässt sich wie folgt berechnen:

$rH_{Sensor} = ((V_{AD}/V_{DD}) - 0,16)/0,0062$; bei 25°C

Die rH_{Sensor} wird dann mit folgender Formel Temperaturkompensiert:

$rH_{true} = rH_{Sensor}/(1,0546 - 0,00216 * T[°C])$

Die rH_{true} wird so über den gesamten Betriebstemperaturbereich (-40°C bis 85°C) in sehr engen Toleranzgrenzen gehalten. Da der HIH-3610 Lichtempfindlich ist, sollte er vor hellem Licht geschützt montiert werden.

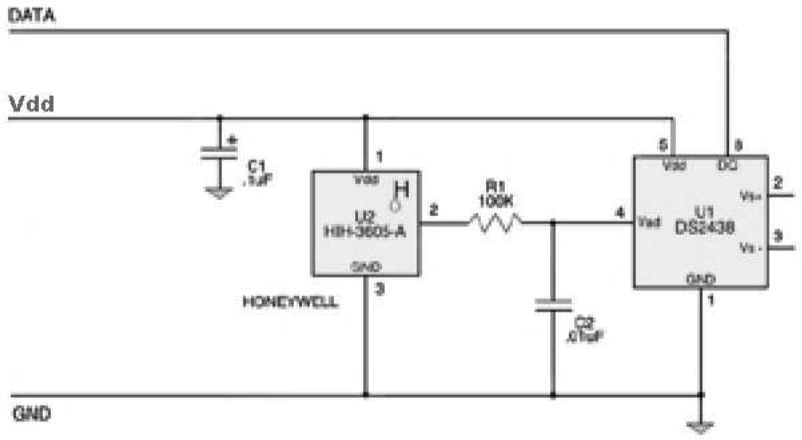

Abbildung 4.5: Schaltplan selbstgebauten 1-Wire Luftfeuchtesensors

Frequenz

Impulse intern oder extern zählen Für die Zählung der Impulse, die der analoge Teil der Schaltung liefert, kann entweder ein Interrupteingang des TINI oder ein externer Zähler verwendet werden. Die Entscheidung fiel zugunsten des externen Zählers, da die ständigen Interrupts die CPU des TINI vermutlich stärker belastet und damit noch weiter verlangsamt hätten. Ausserdem besitzt der DS80C390 [12] nur sechs externe Interrupteingänge, weshalb diese Variante nicht beliebig erweiterbar ist. Als externer Zähler wird ein DS2423 [22] verwendet. Dieser 32-Bit Zähler produziert, bei einer angenommenen durchgehenden Beleuchtung mit 100 Hz, alle 1,36 Jahre einen Überlauf, dessen Verarbeitung der Softwarekomponente überlassen weren muß.

Entwurf mit kapazitiv gekoppeltem Spannungsteiler Die zuerst geplante Schaltung zur Frequenzmessung wurde durch einen Counter, einen Fotowiderstand und dem Timer des TINI realisiert. Der Trip-Point des Counters (DS2423) ist mit $0,5 * U_{Batt} + -0,3V$angegeben, entsprechend muss der Eingang des Counters mit Spannungen zwischen $U_{Low} < 2,2V$ und $U_{High} > 2,8V$ angesteuert werden, damit er einwandfrei zählt. Um dies ueber einen weiten Helligkeits- bereich gewährleisten zu können, wurde in der ersten Schaltungsversion der Zählereingang mit einem (großen) Kondensator an den Mittelabgriff eines Spannungsteilers aus einem festen Widerstand und einem LDR (lichtabhängiger Widerstand) angekoppelt um einen Wechselspannungsanteil zu erhalten. Um die mittlere Spannung am Eingang auf $2,5V$ zu halten wurde zusätzlich der Mittelabgriff eines festen 1:1-Spannungsteilers über einem hochohmigen Widerstand damit verbunden. Da aber mit sinkender Lichtintensität auch die Intensität der Spannungsschwankungen am lichtabhängigen Spannungsteiler absinkt, zählt diese Schaltung nicht mehr richtig, wenn es zu dunkel ist, oder die Lichtintensität nur schwach schwankt.

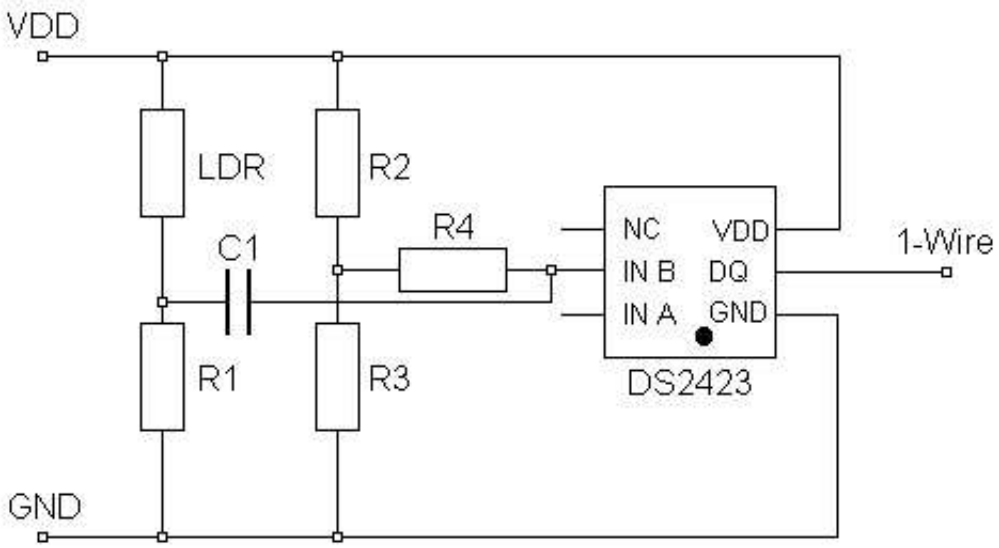

Abbildung 4.6: Schaltplan des einfachen Frequenzsensors

Entwurf mit Differenzierglied und Operationsverstärker Um auch schwache Intensitätsänderungen, beispielsweise die eines entfernt stehenden Monitors (als einzige Lichtquelle im Raum) zählen zu können, wurde die erste Schaltung leicht abgeändert und um eine Verstärkerschaltung erweitert:

Der Kondensator aus der ersten Schaltung wird im Takt mit den Helligkeitsschwankungen auf- und entladen, an einem zusätzlich in Reihe geschalteten Widerstand fällt dementsprechend eine Wechselspannung ohne Gleichspannungsoffset ab. Eine solche Schaltung wird auch als Differenzierglied bezeichnet.

Ein an beiden Anschlüssen dieses Widerstands angeschlossener Operations-Versärker liefert an seinem Ausgang im offenen (nicht rückgekoppelten) Betrieb ein Rechtecksignal, das zwischen der kleinstmöglichen und der grösstmöglichen Spannung (in diesem Fall 0V oder 5V) wechselt. Dieses Signal liegt selbst bei minimalen Helligkeitsschwankungen über 2,8V bzw. unter 2,2V und wird nun dem Zählereingang zugeführt.

Um die Frequenz zu errechnen, wird der Zähler zweimal im Abstand von einer Sekunde ausgelesen, und — sofern der Zeitabstand nicht exakt auf eine Sekunde festgelegt werden kann — die Zählerdifferenz durch die Zeitdifferenz geteilt. Dadurch liegt die Genauigkeit des Messwertes für Frequenzen bis 200Hz bei ca. +-1Hz. Dieser Frequenzbereich sollte genügen, um das Flimmern künstlicher Beleuchtung (wozu hier auch Monitore gezählt werden) zu erfassen. Bei höheren Frequenzen sinkt die Empfindlichkeit des LDR ab, was zur Folge haben könnte, dass Impulse nicht gezählt werden.

Der Widerstand R7, der einen Hysterese-Effekt bewirkt um den Signalausgang bei

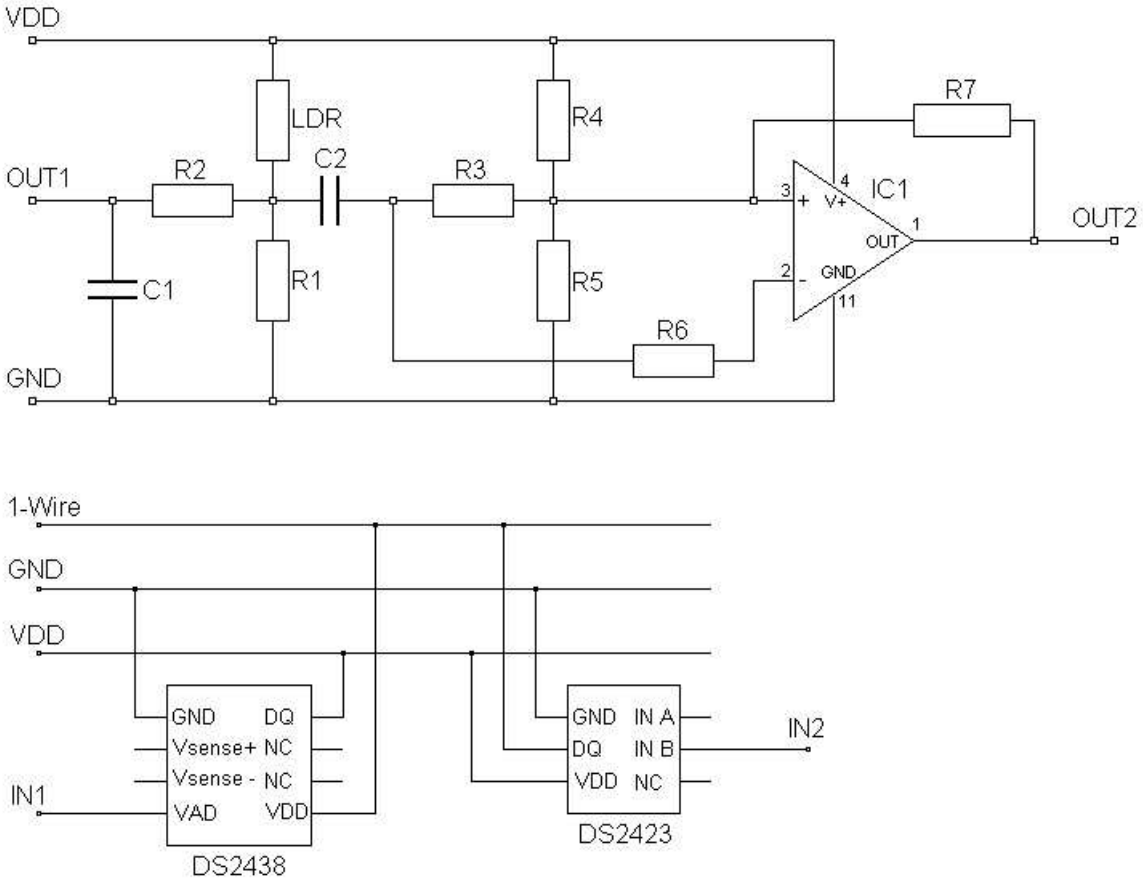

Abbildung 4.7: Schaltplan des kombinierten Lichtsensors für Intensität und Frequenz

geringen Spannungsdifferenzen der Eingänge stabiler zu halten, wurde weggelassen, um auch einen Monitor noch in grösserer Entfernung "erkennen" zu können. In Tests ergab sich keine grössere Störanfälligkeit durch die fehlende Hysterese, die Empfindlichkeitssteigerung konnte jedoch mittels eines Computermonitors bestätigt werden.

Lichtintensität

Für die Messung der Lichtintensität stehen zwei Sensoren zur Verfügung: Ein Spannungsteiler aus einem LDR und einem festen Widerstand sowie eine Solarzelle. Die Ausgangsspannungen werden von jeweils einem DS2438 gemessen.

Die Spannungsteilerschaltung wurde beibehalten, da sie gemeinsam mit einem Differenzierglied und einem Verstärker Teil des Frequenzzählers ist, wie Abbildung 4.7

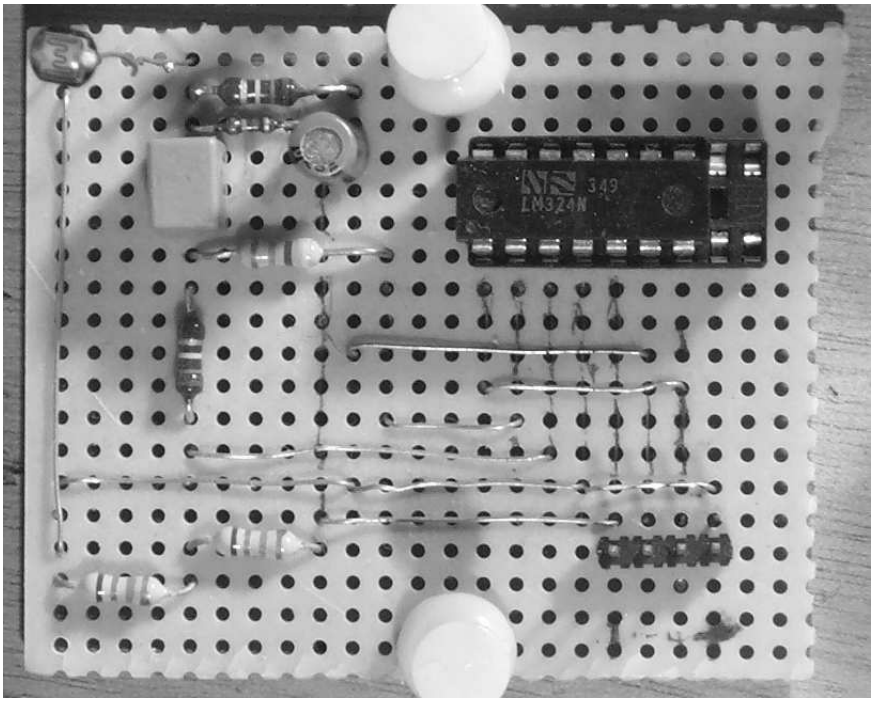

Abbildung 4.8: Der kombinierte Lichtsensor für Intensität und Frequenz

zeigt.

Die Solarzelle wurde angeschlossen, da lineare Licht-Spannungswandler nur für den Infrarot-Bereich verfügbar waren. Wenn beispielsweise ein TSL250 (Licht-Spannung-Wandler) oder ein ähnliches Bauteil mit einer linearen Ausgangskurve verfügbar ist, kann es einfach anstelle der Solarzelle angeschlossen werden, daher der 3-polige Steckverbinder mit Versorgunsspannung. Da aber beide im jetzigen Zustand verwendete Sensoren kein lineares Ausgangssignal liefern, muss eine Eichkurve angefertig werden um brauchbare Messwerte zu erhalten.

Um störenden Effekten durch eine möglicherweise flimmernde Beleuchtung vorzubeugen, wird das Signal noch von einem Tiefpass geglättet.

Lautstärke

Kondensatormikrofon mit Kopfhörerverstärker

Um die Lautstärke zu messen wurde ein Parabolmikrofon-Bausatz (allerdings ohne Parabolspiegel) der Firma Conrad Elektronik verwendet. Die Schaltung verstärkt das Signal eines angeschlossenen Kondensatormikrofons und gibt es über ihren Kopfhörerausgang ab. Die zu messende Spannung wird vor einem, als Gleichstromfilter eingestetzten, Elektrolytkondensator abgegriffen, wodurch es möglich ist, mit ihr, einen zusätzlichen Kondensator über eine Diode, entsprechend der Lautstärke, zu laden. Die Spannung am zusätzlichen Kondensator steigt dadurch fast verzöge-

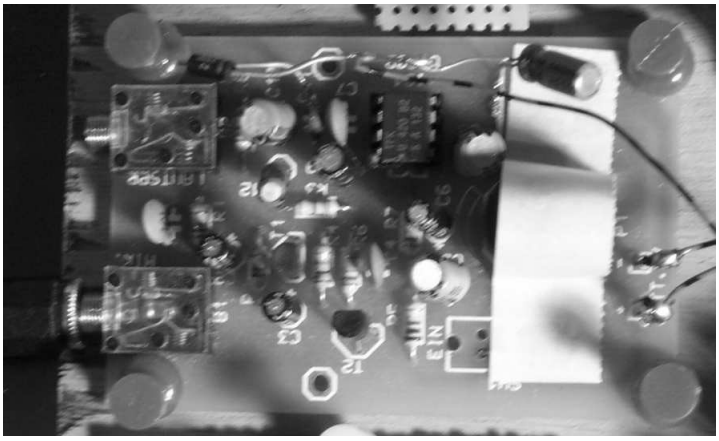

Abbildung 4.9: Die Baugruppe zur Lautstärkemessung

rungsfrei an und fällt langsam wieder ab. Dieses Verhalten der Ausgangsspannung (welche dann gemessen wird) ist nötig, da der angeschlossene A/D-Wandler nur in Intervallen von ca. 10 Sekunden ausgelesen wird.

Während des Abstimmens der Schaltung wurde festgestellt, dass ein Entladungswiderstand (in der Grössenordnung von $10G\Omega$ - NICHT handelsüblich) entfallen kann, da der verwendete Elektrolytkondensator binnen kurzer Zeit (ca. drei bis fünf Minuten) durch Leckströme entladen wird. Bei weiteren Tests wurde festgestellt, dass die Schaltung empfindlich auf elekromagnetische Interferenzen (EMI) reagiert. Ein starkes GSM900-Signal führte zur Anzeige des maximal möglichen Wertes. Daher sollten in einer späteren Version zumindest Teile des ContextCubes abgeschirmt werden. Die Schaltung reagiert weitaus empfindlicher auf hohe Töne als auf niedrige. Mit einem entsprechend angepassten Filter wäre es möglich einen linearen Frequenzgang zu erzielen. Die hierfür nötigen Geräte (z.B. Analyzer, Rauschgenerator und HiFi-Anlage) standen nicht zur Verfügung. Um die Qualität der Messwerte zu erhöhen kann statt der einfachen Verstärkerschaltung ein nur teuer im Spezialhandel erhältlicher Analog-Lautstärkesensor angeschlossen werden.

Taster um das LCD weiterzuschalten

Zur Umschaltung des LCD zwischen den verschiedenen Messwerten dienen zwei Taster (up/down).

Die Erfassung von Tastenbetätigungen wird mittels eines Zählers, in Form eines DS2423, realisiert, da ein Taster normalerweise nur sehr kurz betätigt wird und nicht garantiert werden kann, dass während des Schaltimpulses eine — mit dem Taster in Reihe geschaltete — Silcon Serial Number (DS2401) [18] abgefragt wird.

Anstelle des Counters hätte auch ein (Dual) Addressable Switch (DS2406) [20] — mit der Fähigkeit den Spannungspegel an seinem I/O-Pin festzustellen — in Kombination mit einem Kondensator verwendet werden können. Der Kondensator dient

dabei als 1-Bit Speicher, der durch Beätigung des Tasters gesetzt (+5V=high) und nach jedem Auslesen seiner Spannung durch den DS2406 von selbigem zurückgesetzt wird (0V=low). Mit dieser Lösung wären ebenfalls Tastenbetätigungen zwischen den Auslesevorgängen zu erkennen gewesen, aber maximal eine Betätigung pro Lesevorgang. Da u.a. weitere 1-Wire Devices ausgelesen werden müssen, beträgt der Abstand zwischen zwei Lesevorgängen ungefähr eine Sekunde. Dies wurde als für einen normalen Benutzer zu langsam angesehen. Daher wurde die Variante mit Counter gewählt.

So kann garantiert werden, dass jede Betätigung erfasst wird, selbst wenn der Taster mit der maximal möglichen Frequenz betätigt würde.

Die verwendeten 32-Bit-Zähler würden theoretisch nach etwas über vier Milliarden Tastenbetätigungen einen Überlauf produzieren, die für die Differenz der Zähler genutzte Integervariable bereits nach der Hälfte. Um Speicher zu sparen wird auf 64-Bit Variablen (LONG) oder eine Bereichsüberprüfung verzichtet. Die Lebensdauer von normalen Tastern liegt bei unter einer Million Schaltungen [27] und zum Austausch sollte der ContextCube ohnehin abgeschaltet sein, was den Counter zurücksetzt. Daher wird angenommen, dass es nicht zu einem Überlauf kommen wird; sollte es jemals dazu kommen, so wirkt sich der Überlauf aus, als ob der Taster mehrfach (oder gar nicht bei fünf Zuständen und einem 32-Bit grossen Wertebereich) gedrückt worden sei.

4.3.2 Display am Datenbus

LCD direkt am Datenbus

Da auf den im Zeitraum dieser Arbeit verfügbaren TINI Sockets kein LCD-Connector und auch kein Platz für die nötige kombinatorische Schaltung mehr vorgesehen ist, wurde die Schaltung als aufsteckbares Modul realisiert. Die erste Version diente der Überprüfung, ob der aktuelle TINI (Rev. D) noch immer an den selben Pins die selben LCD-Ausgangssignale führt, wie die erste Ausgabe des TINI (Rev. A).

In der ersten Version wurde nur externe Hardware verwendet, die in einem, auf einer — dem “TINI Board Web Ring” [7] angehörigen — Website [8] gefundenen Schaltplan genannt wird: vier NOR-Gatter (in Form eines 74HC02) und ein LCD-Modul. Der Widerstand kann entfallen, da das Display keine Hintergrundbeleuchtung besitzt, der Kontrast-Anschluss kann direkt auf Masse gelegt werden, sofern das Display bei Raumtemperatur betrieben wird, was zum Testen angenommen werden konnte.

Die acht Datenleitungen (D0-D7) des LCD-Modul werden direkt an die Anschlüsse des TINI Boards angeschlossen, ebenso eine weitere erforderliche Adressleitung (A3) die bei logischem Pegel high Datenempfang, sonst Steuersignalempfang des Moduls aktiviert. Das Enable-Signal ist auf high zu setzen, wenn Adressleitung A19 auf

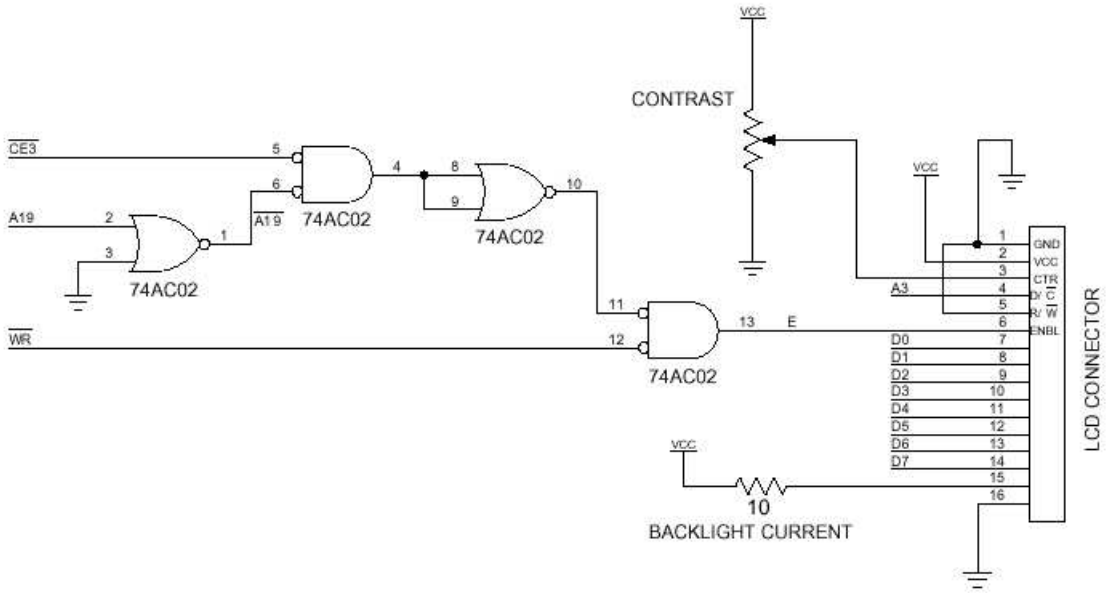

Abbildung 4.10: Schaltplan für den einfachen Anschluss eines LCD an den TINI

high und Leitungen $\overline{CE3}$ und $\overline{WR}$ auf low sind. Die kombinatorische Schaltung ist mit einem 74HC02 vierfach NOR-Gatter realisiert. Der $R/\overline{W}$-Eingang des LCD ist direkt auf Masse geschaltet, da das Modul keine Daten senden soll.

LCD-Pin		TINI-Pin	
Nummer	Name	Nummer	Name
01	GND	05	GND
02	VCC	67	VCC
03	CTR	05	GND
04	$D/\overline{C}$	57	A3
05	$R/\overline{W}$	05	GND
06	ENBL	(13∧¬31∧¬58)	A19∧¬$\overline{CE3}$ ∧ ¬$\overline{WR}$
07..14	D0..D7	53..46	D0..D7
15	BL-VCC	NC	NC
16	BL-GND	NC	NC

Tabelle 4.1: Pinbelegung LCD und TINI

Die gesamte Schaltung ist als “Fliegender Aufbau” realisiert, und besteht aus den Stiftleisten zum Anschluss an den TINI, Buchsenleisten zum Anschluss des LCD, einem gesockelten IC und der dazugehörigen Verdrahtung, wie Abbildung 4.11 zeigt.

LCD entkoppelt vom Datenbus

Da die Daten- und Adressleitungen direkt (und ungepuffert) am Prozessor des TINI angeschlossen sind, ist die maximale Länge des LCD-Anschlusskabels auf we-

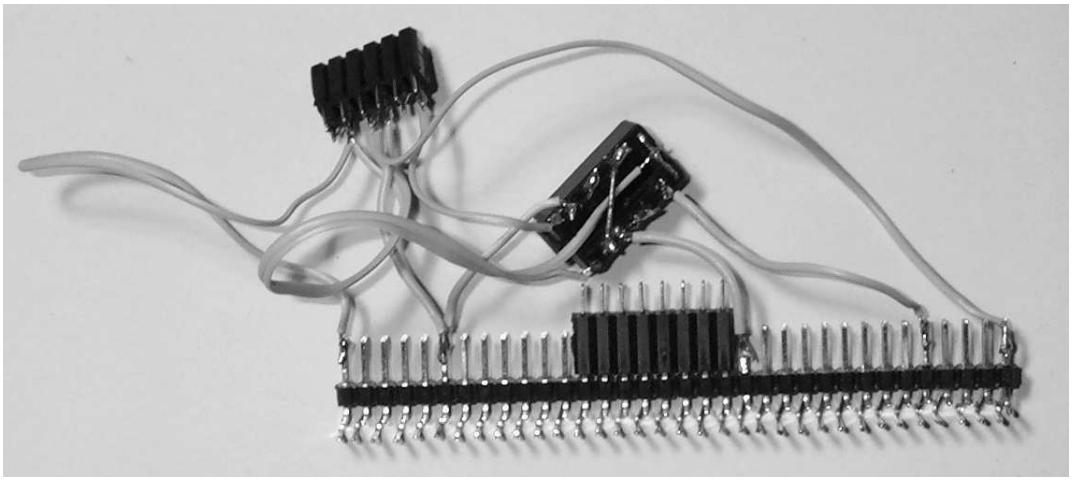

Abbildung 4.11: Minimale Schaltung zum Anschluss eines LCD

Abbildung 4.12: Schaltplan einer der doppelten Inverterstufen

nige Zentimeter begrenzt. Ausserdem besteht die Gefahr, dass der Prozessor bei Berührung des LCD durch Electrostatic Discharge (ESD) zerstört wird.

In der zweiten Version wurden deshalb alle Datenleitungen, sowie die Adressleitung A3 mit weiteren ICs der 74HCxx Familie, entsprechend Abbildung 4.12 doppelt invertiert.

Um die Störungsanfälligkeit des TINI durch verlängerte Daten- und Adressleitungen nicht unnötig zu erhöhen (der TINI hat eine Taktfrequenz von 40MHz, was die mögliche Buslänge stark begrenzt), wurden die ersten zwei 74HC04 sechsfach Inverter so nahe wie möglich an den Anschlussstiften zu den Datenleitungen plaziert. Die zweite Inverterstufe kann an beliebiger Stelle auf der Platine angebracht werden, da von ihr aus (im Normalbetrieb) keine Rückwirkungen auf den TINI erfolgen. Anstelle eines 74HC02 wurden zwei 74HC02 eingesetzt, um auch die weiteren direkt am TINI angeschlosenen Leitungen unter 25 Millimeter Länge halten zu können. Die neun einmal invertierten Signale müssen nochmals invertiert werden, wozu zwei weitere 74HC04 eingesetzt werden. Für die Regelung des Kontrastes wird ein $50k\Omega$ Potentiometer eingesetzt.

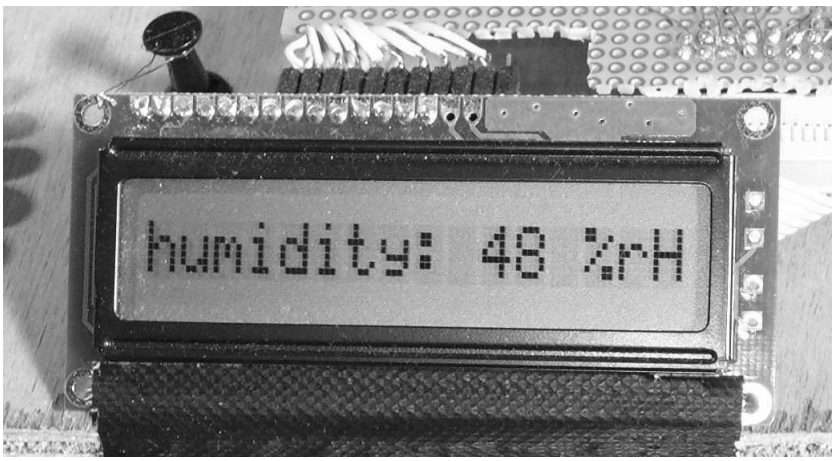

Abbildung 4.13: Anzeige der Luftfeuchtigkeit auf dem Display

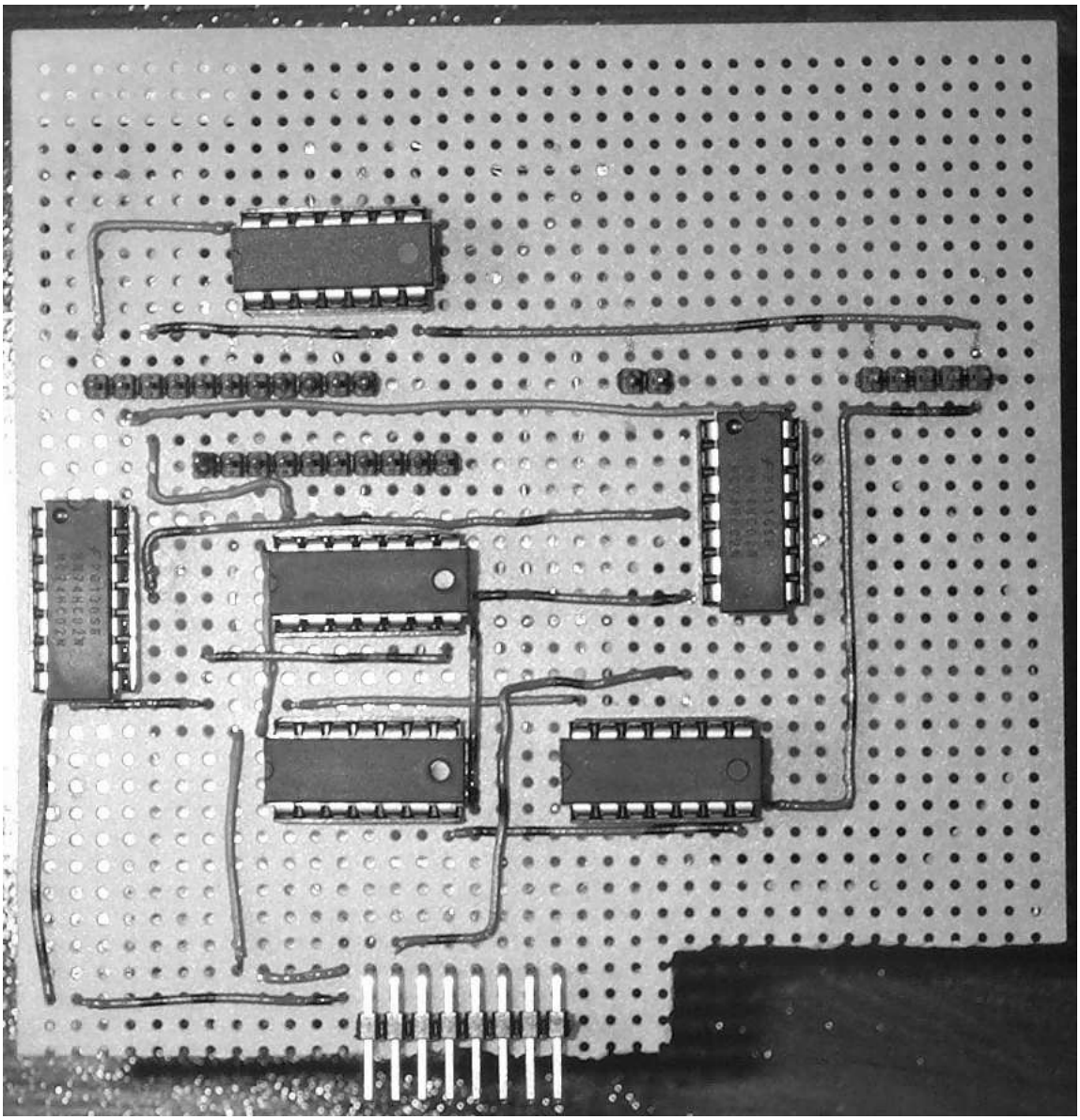

Abbildung 4.14: Verbesserte Schaltung zum Anschluss eines LCD (Bestückungsseite)

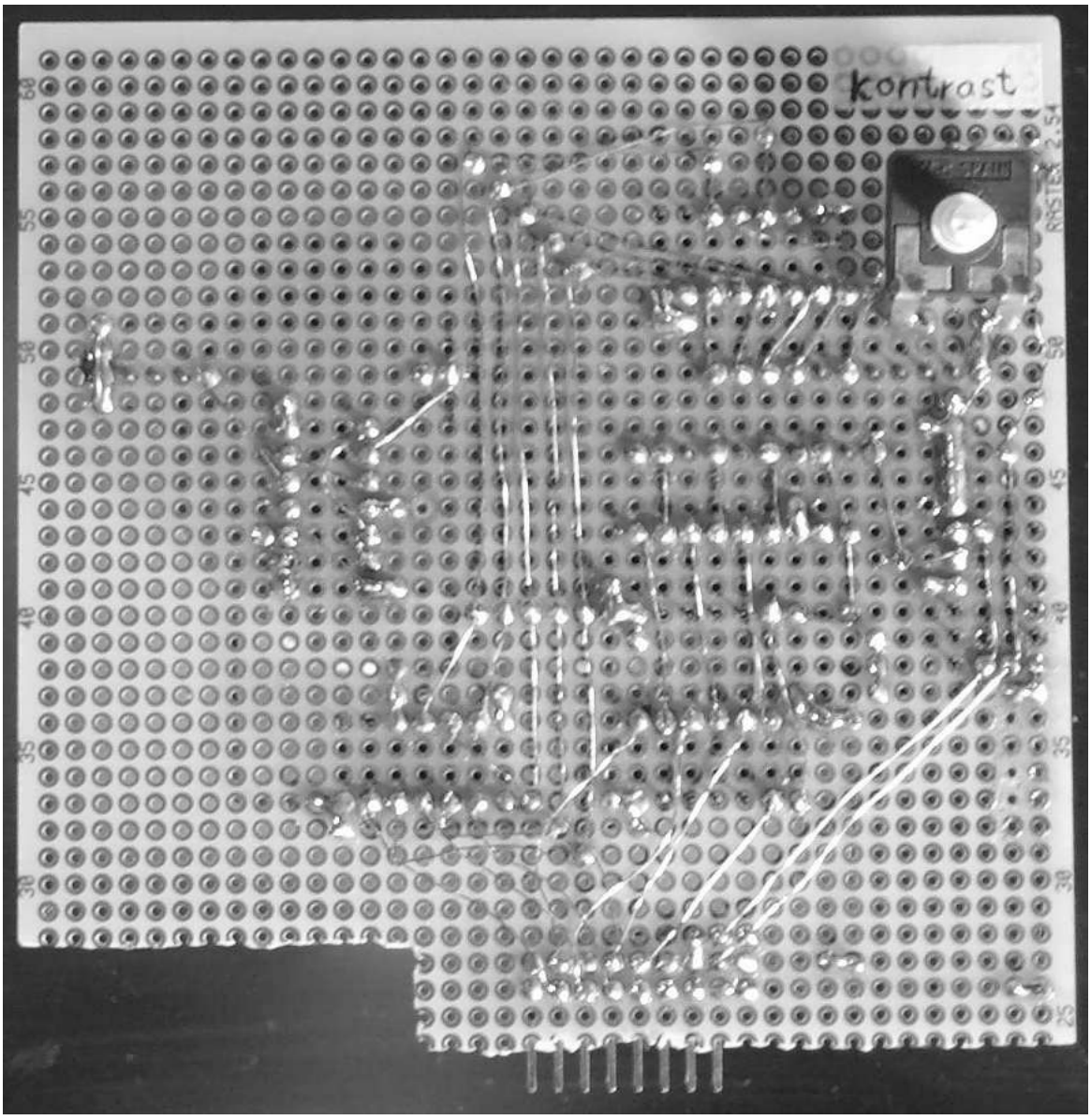

Abbildung 4.15: Verbesserte Schaltung zum Anschluss eines LCD (Lötseite)

Kapitel 5

Software

Die Software des ContextCube besteht auf mehreren Einzelkomponenten:

- HTTP-Server
- XML-Parser
- Komponente zur Auswertung der Sensorendaten

5.1 Auswahl des HttpServers

Da die externe Schnittstelle des ContextCubes auf HTTP aufsetzt ist eine HTTP-Server-Komponente erforderlich.

Um für den Einsatz im ContextCube geeigent zu sein, muß der Server in JAVA implementiert sein, außerdem klein genug, um auf einem Embedded System mit insgesamt 1 MB Speicher noch genug Platz für die Messsoftware und einen XML-Parser zu lassen. Er muß in der Lage sein, POST-Request zu verarbeiten, da diese Voraussetzung für SOAP-over-HTTP sind.

5.1.1 TiniWebServer

In der Firmware [6] des TINI ist in der — zu Beginn dieser arbeit — aktuellen Version bereits ein einfacher HTTP-Server in Form einer speziellen Klasse enthalten.

Dieser ist allerdings, wie auch der darauf basierende und als Beispielcode mitgelieferte TiniWebServer, auf die Verarbeitung von GET-Requests beschränkt. Da eigene Versuche, den HTTP-Server der Firmware um POST-Requests zu erweitern scheiterten, konnte diese Variante nicht genutzt werden.

5.1.2 Eigener HttpServer / Eigene Serverkomponente

Eine weitere Möglichkeit wäre die Implementierung eines eigenen HttpServers. Dies hätte aber entweder den Rahmen dieser Arbeit gesprengt beziehungsweise der Server hätte nur den minimalen Anforderungen genügt.

5.1.3 Servertec Internet Server TINI Edition

Der von Servertec [29] erhältliche HttpServer ist zwar servletfähig, kostet jedoch bereits als Binary US $ 50 und ist damit zu teuer.

5.1.4 TiniHttpServer

Der von Smart Software Consulting [30] angebotene TiniHttpServer [31] unterstützt nicht nur alle Standard-Http-Requests, sondern auch fast vollständig [32] den Einsatz von Servlets. Dadurch wäre der ContextCube sehr einfach zu erweitern. Wenn die ContextCube-Software als Servlet implementiert wird, ist der Server austauschbar. Ein weiterer Vorteil ist, dass dieser Server seine Logfiles bei entsprechender Konfiguration via eMail versenden kann, wenn sie eine einstellbare Größe überschreiten, was sicherstellt, dass nicht der komplette Speicher des TINI mit Loggingdaten gefüllt wird. Ein weiterer großer Vorteil ist, dass der TiniHttpServer unter der GNU General Public License (GPL) [33] kostenlos erhältlich ist, weshalb davon ausgegangen werden kann, dass der Server (auch unter Sicherheitsaspekten) weiterentwickelt wird. Allerdings ist zu bemerken, dass dieser Server in der aktuellen Version recht gross für den Einsatz auf einem Embedded System ist, ohne Servlets ca. 80kB auf dem TINI.

5.1.5 Auswahl der Serverkomponente

Die wichtigsten Kriterien für die Auswahl des Servers sind — neben den bereits genannten Grundvoraussetzungen — die Bereitstellung der benötigten Funktionen (POST-Requests) und die einfache Erweiterbarkeit. Der TiniHttpServer wurde trotz seiner Grösse von 80kB gewählt, da er Servlets ausführen kann. So können die restliche Software und auch zukünftige Erweiterungen als Servlets implementiert werden.

5.2 Auswahl des XML-Parsers

Ein XML-Parser wird benötigt, um einerseits die SOAP-Requests bearbeiten zu können, und andererseits die darin enthaltenen AWQL-Requests auswerten zu können.

5.2.1 Xerxes2

Der Parser Xerxes2 [35] wurde von der Apache Software Foundation [34] entwickelt. Dieser auch im NeXus Spatial Model Server verwendete Parser unterstützt DOM, SAX, JAXP, XML-Schema 1.0 und vieles mehr. Das Jar-File ist allerdings mit seiner Grösse von knapp 800 kB nur noch um wenig kleiner als der gesamte — ein Megabyte grosse — Speicher des TINI. Daher wurde dieser Parser — wie auch die meisten anderen verbreiteteren Parser — als zu gross für den Einsatz auf dem TINI angesehen.

5.2.2 AElfred

Der unter einer BSD-ähnlichen Lizenz [39] erhältliche Parser AElfred ist ungefähr 24 kB gross. Der Hersteller Microstar Software Ltd. wurde mittlerweile von der Open Text Corporation [38] aufgekauft, die AElfred nicht weiter entwickelt. Ausserdem wird auf [37] über Probleme mit diesem Parser auf der TINI-Platform berichtet.

5.2.3 Lark

Der von Textuality [40] für Applets entwickelte Parser Lark [41] hat eine Grösse von ca. 45 kB. Er ist ebenfalls ein nichtvalidierender Parser, der einen Kompromiss zwischen Größe, Geschwindigkeit und Vollständigkeit darstellt.

5.2.4 MinML2

MinML2 [42] wurde als Parser ausgewählt, da er mit ca. 17 kB nicht nur der kleinste zur Verfügung stehende war, sondern auch da der Autor John Wilson diesen Parser speziell auf den TINI zugeschnitten hat. Ausserdem hat der Autor auf der TINI-Mailingliste [37] bereits zu einigen Themen gute Lösungsmöglichkeiten für verschiede Probleme im Zusammenhang mit der TINI-Plattform gepostet woraus hervorgeht, dass er sich mit diesem System sehr gut auskennt.

5.3 Entwurf des AwqlServlets

Nachdem die Entscheidung über die Serverauswahl zugunsten des TiniHttpServers mit Servlet-Engine gefallen war, lag es nahe, die selbst entwickelte Software als Servlet zu entwerfen. Durch diese Trennung der Komponenten an einer wohldefinierten Schnittstelle, ist es möglich, den Server, falls nötig, gegen eine aktualisierte Version auszutauschen, indem bei der Erstellung einer neuen TINI-Datei einfach die Class-Dateien des Servers ausgewechselt werden.

5.3.1 Aufbau

Das AwqlServlet bestand im ersten Entwurf aus folgenden Teilen:

AwqlServlet: Das AwqlServlet selbst stellt die Schnittstelle zum HttpServer dar.

SoapParser: Der SoapParser wertet den SOAP-Request aus. Als Basis dient MinML2, ein SAX2-Parser, der speziell für die TINI-Plattform entwickelt wurde.

AwqlRequest: Der AwqlRequest dient als Rückgabewert des SoapParsers.

AwqlParser: Der AwqlParser, ebenfalls auf MinML2 basierend, wertet den Awql-Request aus.

RestrictionElement: Das RestrictionElement entscheidet anhand des von <restriction> eingeschlossenen Elementes, von welchen Sensoren Informationen in der Antwort übermittelt werden sollen. Dabei darf das eingeschlossene Element weitere untergeordnete Elemente besitzen.

EqualElement: EqualElement wertet ein equal-Element im Restriction-Element aus indem es seine beiden untergeordneten Elemente vergleicht.

AndElement: Das AndElement wertet ein and-Element im Restriction-Element aus indem es die Auswertungsergebnisse der eingeschlossenen Elemente konjunktiv verknüpft.

OrElement: Das OrElement wertet ein or-Element im Restriction-Element aus indem es die Auswertungsergebnisse der eingeschlossenen Elemente disjunktiv verknüpft.

NotElement: Das NotElement wertet ein not-Element im Restriction-Element aus indem es das Auswertungsergebnis des eingeschlossenen Elementes negiert.

SensorWorker: Der SensorWorker ist ein eigener Thread, der die Sensoren am 1-Wire Bus ausliest.

LCDOutput: LCDOutput steuert das Display des ContextCubes an.

AwmlComposer: Der AwmlComposer erzeugt aus Anfrage- und Sensordaten die SOAP-AWML-Antwort auf SOAP-AWQL-Requests

SensorRequest: Der SensorRequest dient als Rückgabewert des AwqlParsers

FamilyCode26Values: Die FamilyCode26Values dienen als Rückgabewert bei der Abfrage eines DS2438

SensorAttributes: Die SensorAttributes enthalten die Daten der Sensoren (Messwert, Aktualität, Toleranz, ...)

Die CCAttributes: Die CCAttributes enthalten für jeden Sensor eine Instanz der SensorAttributes und damit die gesamte Sensorinformation des ContextCube.

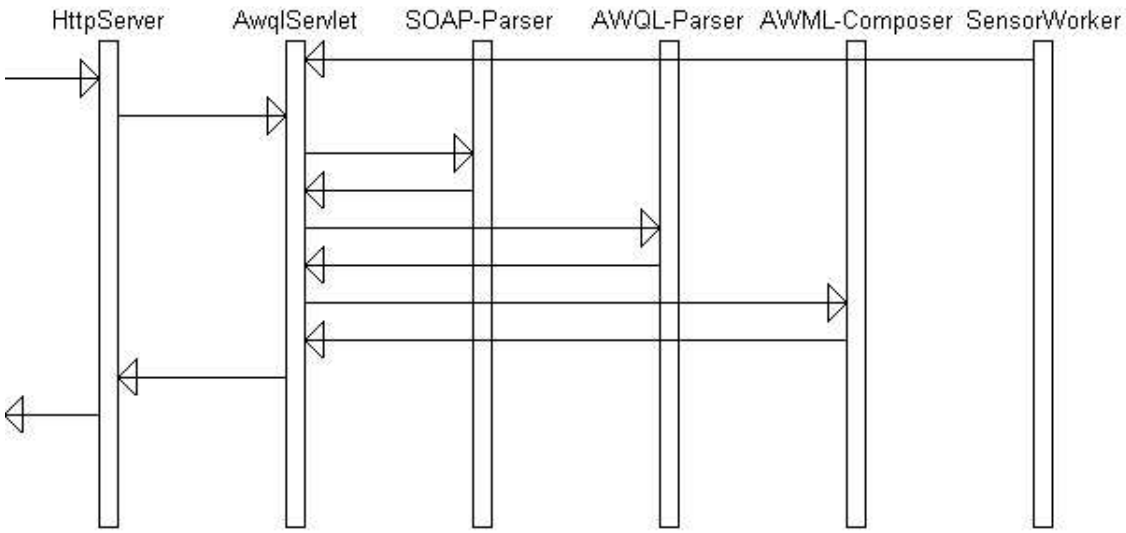

Abbildung 5.1: Geplanter Ablauf

5.3.2 Ablauf

Die Softwarekomponente hat drei Hauptaufgaben :

- Das AwqlServlet nimmt SOAP-AWQL Anfragen entgegen, filtert alles aus, was ausserhalb des SOAP-Envelopes liegt (dadurch sind zu Test- und Vorführzwecken Anfragen per Browserfileupload bequem möglich), startet den SensorWorker bei der Initialisierung des Servlets und übergibt die so "bereinigte" Anfrage an den SOAP-Parser. Dieser übersetzt sie in einen Sensor-Request.
- Der SensorWorker liesst in regelmässigen Abständen die angeschlossenen Sensoren aus und aktualisiert so die Sensordaten.
- Zuletzt erstellt der AwmlComposer aus dem SensorRequest und den Sensordaten die AWML-Antwort, die vom AwqlServlet ausgegeben wird.

Der geplante Ablauf ist in Abbildung 5.1 dargestellt.

Entgegennahme der Anfrage

Der ContextCube muss in der Lage sein, SOAP-AWQL Requests entgegenzunehmen; dafür ist der TiniHttpServer zuständig. Die SOAP-AWQL Anfrage wird von der Serverkomponente als POST-Request (SOAP-over-HTTP) entgegengenommen. Von dort wird sie als HTTP-Request an das AWQL-Servlet übergeben. Dieses leitet den Inhalt des Requests als String an den SOAP-Parser weiter, nachdem evtl.

vorhandene Teile ausserhalb des SOAP-Envelopes entfernt wurden. Der Inhalt des SOAP-Body wird dann an den AWQL-Parser weitergeleitet und von diesem in einen SensorRequest umgewandelt.

Bei einem GET-Request auf das Servlet war geplant, dem AwmlComposer, ohne weitere Analyse des Inhaltes, einen bestimmten SensorRequest übergeben, der den AwmlComposer veranlasst, anstelle der AWML-Antwort eine HTML-Antwort mit allen Sensorinformationen zu liefern. Diese Funktionalität wurde aufgrund des knappen Speichers des TINI nicht implementiert.

Ermittelung der Daten

Um immer möglichst aktuelle Daten liefern zu können, werden die Daten der Sensoren, die über den 1-Wire Bus ausgelesen werden können, ständig vom SensorWorker abgefragt, der als eigener Thread läuft. Nach jeder erfolgreichen Ablesung wird die dem jeweiligen Sensor zugeordnete Instanz der SensorAttributes aktualisiert.

Erstellen einer Antwort

Die Antwort wird vom AwmlComposer aus Templates und den Sensorinformationen erstellt. Dazu werden ihm Daten über die Sensoren und die Anfrage übergeben. Anhand der im SensorRequest übergebenen Daten entscheidet der AwmlComposer, welche Sensoren (Temperatur, Luftfeuchtigkeit, ...), in Form der Objekte, und welche ihrer Werte (Messwert, Genauigkeit, ...), in Form der Objekt-Attribute, in die von ihm erstellte Antwort mit einzuschliessen sind. Zusätzlich war geplant, auf einen speziellen SensorRequest hin die Antwort aus einem zusätzlichen HTML Template und allen Sensorinformationen zu erstellen, was sich leider mit dem verfügbaren Speicher nicht realisieren ließ.

5.4 Implementierung des AwqlServlets

Da sich während der Implementierungsphase zeigte, dass es — aufgrund der geringen Speicherkapazität des TINI — nicht möglich sein würde, die Implementierung wie geplant umzusetzen, musste der Entwurf mehrmals vereinfacht werden.

5.4.1 Keine TINI-Seitige Datenarchivierung

Da die Implementierung von weiteren Dateizugriffen die Software nochmals vergrössern würde und auch der Speicherplatz für Dateien knapp bemessen ist, wurde auf die Implementierung einer Datenarchivierung verzichtet. Um die Daten trotzdem zu archivieren, kann der ContextCube von anderen Geräten ausgelesen werden, die genügend Speicher besitzen.

5.4.2 Keine reine HTML-Schnittstelle

Die optional geplante zusätzliche HTML-Schnittstelle wurde nicht fertig implementiert, da es schon bei der Implementierung der AWQL/AWML-Schnittstelle des öfteren zu Speicherproblemen kam. Über diese Schnittstelle wäre eine HTML-Datei (oder auch XML-Datei) aus einem Template und den aktuellen Sensorinformationen abzurufen gewesen. Sollte ein kleinerer Server zum Einsatz kommen, oder ein entsprechender Microcontroller mit mehr Speicher, so ist eine HTML-Schnittstelle leicht zu implementieren: Dafür muss dem AwmlComposer nur bei jedem GET-Request auf das Servlet ein spezieller SensorRequest übergeben werden, auf den der AwmlComposer aus einem zusätzlichen HTML-Template und den Daten des SensorWorkers eine Seite mit allen Sensordaten erstellt.

5.4.3 Weitere Einschränkungen

Zuerst wurden Klassen zusammengelegt, da für jede Klasse zusätzlicher Speicher verbraucht wird. Dabei wurden der SOAP-Parser und der AWQL-Parser zunächst gemeinsam in einer Klasse implementiert, später sogar in einer gemeinsamen Methode, die pro Request nur noch einmal aufgerufen wird. Die Klasse LCDOutput wurde mit in der Klasse Sensorworker untergebracht. Diese Zusammenlegung bot sich an, da nicht nur die Anzeigewerte vom SensorWorker geliefert werden, sondern auch, die Information, welcher der Werte auf dem Display erscheinen soll. Die Klasse CCAttributes — bestehend aus einer Instanz der Klasse SensorAttributes für jeden Sensor — wurde komplett weggelassen. Stattdessen wird dem AwmlComposer nur noch eine Referenz auf den SensorWorker übergeben und darüber auf die verschiedenen SensorAttributes des SensorWorkers zugegriffen.

Als sich im Laufe der Implementierung zeigte, dass, aufgrund des knappen Speichers, weitere Vereinfachungen nötig waren, wurde auf die Implementierung der Elemente <or>, <and> und <not> verzichtet.

Sollten diese Elemente in einem Request an den ContextCube vorkommen, wird ein SOAP-Fault anstelle einer AWML-Antwort zurückgegeben.

Auf den Vergleich mit einem NeXus Object Locator (NOL) musste ebenfalls verzichtet werden. Eine NOL identifiziert die Objekte in NeXus eindeutig, und eignet sich dadurch besonders gut, einen bestimmten Sensor widerholt auszuwählen.

Daher besteht das AwqlServlet aus weniger Teilen(/Klassen) als ursprünglich geplant:

AwqlServlet: Das AwqlServlet selbst stellt die Schnittstelle zum HttpServer dar.

AwqlProcessor: Der AwqlProcessor wertet die SOAP-AWQL-Anfrage aus. Als Basis dient MinML2, ein SAX2-Parser, der speziell für embedded JAVA entwickelt wurde.

SensorWorker: Der SensorWorker ist ein eigener Thread, der die Hardware steuert.

AwmlComposer: Der AwmlComposer erzeugt aus Anfrage- und Sensordaten die Antwort.

SensorRequest: Der SensorRequest enthält Daten ueber den Request.

FamilyCode26Values: Die FamilyCode26Values dienen als Rückgabewert bei der Abfrage eines DS2438.

SensorAttributes: Die SensorAttributes enthalten die Daten der Sensoren (Messwert, Aktualität, Toleranz, ...).

5.4.4 AwqlServlet

Das AwqlServlet enthält eine Initialisierungsroutine, welche den SensorWorker startet. Es nimmt die Requests vom WebServer entgegen und entfernt alles, was ausserhalb der SOAP-Nachricht liegt. Es übergibt die SOAP-Nachricht an den Awql-Procesor und gibt den — vom AwqlProcessor erzeugten — SensorRequest und die Messdaten des SensorWorkers an den AwmlComposer weiter. Nach der Erstellung der Antwort setzt es den ContentType auf "text/xml"und den HTTP-StatusCode im Response bei Bedarf auf "500 Internal Server Error". Zuletzt übergibt es dem WebServer den HTTP-Response.

5.4.5 SensorRequest

Der SensorRequest enthält Variablen, welche angeben, welche Elemente der Response enthalten soll, wie das ECS-Präfix (ExtendedClassSchema) heissen soll, unter welchem Namen der WebServer angesprochen wurde und ob und zu welchen Fehlern es bei der Verarbeitung des AwqlRequests kam. Er wird vom AwqlProcessor mit den entsprechenden Daten versorgt und kann vom AwmlComposer einfach ausgewertet werden.

5.4.6 AwqlProcessor

Diese Klasse realisiert die Konvertierung einer gegebenen AWQL Anfrage in einen SensorRequest.

Das Parsen wird hauptsächlich vom XML-Parser MinML2 übernommen. In diesem Parser ist nicht der komplette XML-Standard implementiert, DTDs werden zum Beispiel einfach ignoriert (nichtvalidierender Parser). Nur die Callbacks in denen die einzelnen Elemente verarbeitet werden sind direkt im AwqlServlet implementiert.

Obwohl die Mächtigkeit von AWQL im Vergleich zu SQL bedeutend kleiner ist, mussten mehrere Einschränkungen vorgenommen werden um mit dem Speicher des TINI auszukommen.

Unterstütztes AWQL Subset

Kein <in>, <inside>, <overlaps> und <closest>: Da diese Geo-Befehle recht komplex sind und eine Datenbank erfordern würden, musste auf sie verzichtet werden. Um dennoch festzustellen, welcher ContextCube sich wo befindet, bleibt es dem Aufrufer überlassen, weitere NeXus-Komponenten abzufragen. In einer erweiterten Version könnte der ContextCube die Geo-Befehle auch selbstständig an einen "echten" SpaSe weiterleiten, um sie dort evaluieren zu lassen.

Nur <equal> in der Restriction: Das, der WHERE-Klausel in SQL entsprechende <restriction> Element ist nur in der Form Attribut = Attribut (<equal><attr name="type"/><nexusdata> [Messwertname]</nexusdata></equal>) möglich, da im Laufe der Arbeit Speicherplatzprobleme auf dem TINI auftraten und auf die Implementierung weiterer boolscher Ausdrücke (<and>, <or> und <not>) verzichtet werden musste.

Nur <attr name="type"/> im Equal Element Auf <equal><attr name="nol"/>...</equal> musste ebenfalls während der Implementierung verzichtet werden.

Funktionsweise/Überprüfungen/Konvertierung

Folgende Elemente können in einer AWQL-Anfrage über SOAP vorkommen:

<**SOAP-ENV:Envelope**> - muss vor <SOAP-ENV:Body> kommen

- muss korrekte XML-Namespaces haben

<**SOAP-ENV:Header**> - darf nicht vorkommen

<**SOAP-ENV:Body**> - muss vor <awql> kommen

<**awql**> - muss alle anderen AWQL Elemente einschliessen.

<**aas**> **und** <**naal**> - die naal muss der naal des ContextCube entsprechen. Bei unbekannter (nicht eigener) NAAL wird ein SOAP-Fault anstelle einer leeren Antwort ausgegeben da es zu Speicherproblemen kam und die Möglichkeit zur Ausgabe von SOAP-Faults an vielen Stellen benötigt wird.

<**scope**> **und** <**ecs**> - name muss korrekt sein
(name="nexus://nexusschemas.org/ContextCube")

- is wird vom awml-Composer übernommen und wird (als Alias für den ganzen Namen) Prefix des type-Attributes in <nexusobject>

<**restriction**> - Impelementierung beschränkt auf <equal>

<**filter**> - entspricht der SELECT-Klausel von SQL.

<**includes**> - gibt alle Attribute an, welche enthalten sein sollen

<**excludes**> - gibt an, welche Attribute nicht enthalten sein sollen

<**attr name="Name"**> - gibt an, welches Attribut enthalten sein soll (also SELECT Name)

<**includeallother**> - gibt an, dass nur die unter <excludes> angegebenen Attribute nicht enthalten sein sollen (kann also ignoriert werden)

<**excludeallother**> - gibt an, dass nur die unter <includes> angegebenen Attribute enthalten sein sollen (kann also ignoriert werden)

<**restriction**> - entspricht der WHERE-Klausel in SQL.

<**equal**> wird zu Attribut = Attribut (NOL=NOL wurde nicht implementiert)

Folgende Elemente können zwar in einer AWQL Anfrage vorkommen, wurden allerdings nicht implementiert und führen daher bei ihrem Auftreten zu einem SOAP-Fault:

<**update**>

<**generalization**> **und** <**aggregation**>

<**and**>, <**or**> **und** <**not**> (entsprechen den gleichnamigen Boolschen Operatoren)

<**in**> (entspricht dem Geo-Befehl IN von SQL)

<**inside**> (entspricht dem Geo-Befehl INSIDE von SQL)

<**overlaps**> (entspricht dem Geo-Befehl OVERLAPS von SQL)

<**closest**>

Um die Programmgrösse möglichst klein zu halten wurde in Erwägung gezogen, nur einen optimistischen Parser zu verwenden, da dies eine beachtliche Speicherersparnis mit sich bringt (im kB-Bereich). Die Anforderungen von SOAP erzwangen aber die Verwendung der pessimistischen Version.

Optimistisch:

- kleiner (Class-File 3kB) da er mit 19 Zuständen auskommt
- erkennt Elemente, die er nicht versteht
- erkennt strukturell falsches xml (durch den Parser)
- erkennt mehrere Fehler wie fehlende Elemente oder falsche XML-Namespaces (versionMismatch) nicht

Pessimistisch:

- größer (Class-File 19kB) da er weit mehr Zustände benötigt
- überprüft den Namespace
- erkennt wenn unbedingt nötige Elemente fehlen (z.B.: ecs)
- erkennt wenn Elemente in der falschen Zusammensetzung verwendet werden (z.B.: excludeallother und includeallother)

Beschreibung der nicht implementierten Hilfsklassen

Obwohl die Klassen RestrictionElement, AndElement, OrElement, NotElement und EqualElement nicht implementiert wurden, soll hier — am Beispiel der Klasse AndElement — beschrieben werden, wie sie geplant waren.

Zunächst wird im Parser ein RestrictionElement instanziiert und eine globale Variable, hier actualElementAddress genannt, auf die Adresse des RestrictionElements gesetzt.

Wenn der Parser innnerhalb des <restriction> Elements auf den Start-Tag eines <and> Elements stösst, wird eine (diesem Element zugeordnete) Instanz der Klasse AndElement erzeugt, hier mit A1 bezeichnet. Als Argument wird ihr die actualElementAddress übergeben. Der Wert wird in einer Variable, hier caller genannt, zwischengespeichert, da die actualElementAddress daraufhin mit der Adresse von A1 überschrieben wird. Dadurch wird beim Auftreten eines weiteren eingeschlossenen <and>, <or>, <not> oder <equal> Elements das Ergebnis der dabei erzeugten Instanz nicht an das RestrictionElement, sondern direkt an die übergeordnete Instanz übergeben. Ein A1 untergeordnetes Element (bzw. die zugehörige Klasseninstanz) übergibt sein Ergebnis also an A1, wobei eine (vorher initialisierte) Variable result auf das logische Produkt von sich selbst mit dem Ergebnis gesetzt wird. Stösst der Parser auf das (A1 zugeordnete) End-Tag, so wird A1 dazu veranlasst, den Wert von result an die unter der Adresse caller erreichbare Instanz — in diesem Fall an das RestrictionElement — zu übergeben und die actualElementAddress wieder auf den Wert von caller zu setzen. Ein, nicht dem RestrictionElement, sondern beispielsweise A1 untergeordnetes AndElement arbeitet genauso, nur ist die Variable caller auf eine andere Adresse gesetzt, zum Beispiel die von A1.

Die Klassen OrElement und NotElement funktionieren analog.

Die Klasse EqualElement vergleicht ihre beiden Argument-Elemente und übergibt das Ergebnis des Vergleichs ebenfalls an den Aufrufer(caller).

(Entspricht einer Baumstruktur mit ausschliesslich <equal>-Elementen an den Blättern)

Da mehrere Sensoren (=Objekte) zur Auswahl stehen, erfolgt die Auswertung — um den Inhalt(?) des <restriction> Elements nicht mehrmals parsen zu müssen — für alle Objekte auf dem ContextCube in einem Durchlauf. (Daher wird als result ein Array genutzt.)

5.4.7 SensorWorker

Ein passenderer Name wäre eigentlich HardwareWorker, nachdem die — zuerst in einer eigenen Klasse untergebrachte — Ansteuerung des LCD zur Speicherersparnis mit in dieser Klasse implementiert werden musste. Für die wiederholt benötigte Ausgabe von Messwerten steht aber weiterhin eine eigene Methode zur Verfügung: LCDOut()

Um festzustellen, welcher Messwert bei einem Aufruf dieser Methode angezeigt werden soll, werden die Zählerstände des, den zwei Tastern zugeordneten, DS2423 ausgelesen. Die Differenz dieser Werte bestimmt, welcher Wert angezeigt wird.

Dadurch wird die komplette Ansteuerung der externen Hardware von dieser Klasse realisiert. Da die Messdaten ständig zu aktualisieren sind, wurde diese Klasse als eigener Thread implementiert. Beim Start des SensorWorkers werden zuerst das Display und die fünf Instanzen der SensorAttributes initialisiert. In einer Endlosschleife werden nacheinander die Sensoren abgefragt. Die einzige Aussnahme bildet der Zähler für die Taster. Er wird durch mehrfachen Aufruf der Methode LCDOut() innerhalb eines Durchlaufs mehrfach abgefragt, um eine schnellere Reaktion des Displays auf Tastenbetätigungen zu erreichen. Daher wird diese Methode nach jeder Sensorabfrage, innerhalb der mindestens eine Sekunde dauernden Frequenzmessung der Beleuchtung und nach der Berechnung der Durchschnittstemperatur durchgeführt. Dadurch verlängert sich die Zeit zwischen je zwei Messungen des selben Sensors auf ungefähr acht Sekunden, was aber trotzdem für die meisten Anwendungen als ausreichend betrachtet werden kann. Vor jeder Sensorabfrage wird die Betriebsbereitschaft der einzelnen 1-Wire Devices (also der Sensoren) sichergestellt so weit möglich. Dies gilt auch für den Baustein der die Signale der Taster verarbeitet; die Taster können ebenfalls als Sensoren angesehen werden die eine Kraft auf ihre Taste messen und ab einem bestimmten Wert ihre Impedanz absenken. Da die Tastenbetätigungen aber nur für das lokale Display von Interesse sind, stehen sie nicht über öffentliche Methoden zur Verfügung.

Die Sensorwerte werden der Differenz der Zählerstände des — für die Taster zuständigen — DS2423 gemäss Tabelle 5.1 zugeordnet. Delta berechnet sich dabei wie folgt:

delta = ((((downKeyCountervalue-upKeyCountervalue))%5)+5)%5)

Die Addition von fünf zum Ergebnis der ersten Modulo Operation und die zweite Modulo Operation sind notwendig, da die erste Operation bei einer negativen Zählerdifferenz Werte im Bereich von minus vier bis null liefert.

delta	Messwert	Beispiel
0	Temperatur	temp: 20.1°C
1	Luftfeuchtigkeit	humidity: 64%rH
2	Beleuchtungsfrequenz	light: 100 Hz
3	Lichtintensität	light: 1234 Lux
4	Lautstärke	volume: 80 dB

Tabelle 5.1: Anzeige des LCD

Die Kommunikation über den 1-Wire Bus wird dabei für jeden Sensor in einem eigenen try-catch-Block eingebunden. Um Speicher zu sparen hätte auch nur ein grosser try-catch-Block verwendet werden können, allerdings wären dann durch den Ausfall eines Sensors möglicherweise (abhängig von der Art des Fehlers) alle Sensoren betroffen, die nach dem Auftreten des Fehlers ausgelesen würden. Daher wurde trotz der Speicherknappheit ein eigener try-catch-Block für jeden Sensor beibehalten.

Nach jedem erfolgreichen Auslesen eines Sensors wird die zugehörige Instanz von SensorAttibutes mit dem Messwert und der aktuell möglichen Genauigkeit aktualisiert. Optional kann auch die SensorId aktualisiert werden, falls mehrere (oder — wie der DS1920 — auswechselbare) Sensoren für einen besimmten Wert zur Verfügung stehen. Da die Abkürzungen der Einheiten mit in den Werten untergebracht werden sollten, werden dafür Strings genutzt. So kann, falls nötig, auch die Einheit des Messwertes vom SensorWorker, beispielsweise von Volt (V) auf Millivolt (mV), geändert werden.

Die Reihenfolge, in der die Sensoren ausgelesen werden, ist egal. Für die Kommunikation mit allen 1-Wire Bauteilen stehen bereits — im Package com.dalsemi.onewire.container [6] — fertige Klassen zur Verfügung. Da es 1-Wire Bauteile mit unterschiedlichsten Fähigkeiten gibt, steht für jede Familie von 1-Wire Bauelementen dabei eine eigene Klasse zur Verfügung. Vom ContextCube werden nur drei unterschiedliche Familien genutzt:

FamilyCode10 : Für den Thermo iButton

FamilyCode1D : Für die als Zähler eingesetzten DS2423

FamilyCode26 : Für die als A/D-Wandler und Thermometer eingesetzten DS2438

Der Thermo iButton DS1920 ist das einzige FamilyCode10-Bauteil. Da dieser Sensor — in Bauform einer Knopfzelle — nur in eine Batteriehalterung gesteckt, und daher im laufenden Betrieb austauschbar ist, wird die 64-Bit Id dieses Sensors im Gegensatz zu allen anderen angeschlossenen 1-Wire Bauteilen nicht in der Konfiguration des SensorWorkers festgelegt. Stattdessen wird der DS1920 mit der niedrigsten Id

angesprochen. Dadurch ist zu Vorführzwecken der Thermo iButton im laufenden Betrieb austauschbar. Falls weitere DS1920 an den ContextCube angeschlossen werden sollen, muss ihm natürlich bekannt sein, welcher davon für welche Anwendung zu nutzen ist.

Um die Frequenz der Beleuchtung zu messen, wird der Zähler des dafür verwendeten DS2423 zweimal im Abstand von einer Sekunde ausgelesen. Eigentlich sollte zwischen dem Ende beider Auslesevorgänge exakt eine Sekunde vergehen, was auf dem TINI — mittels Thread.sleep() — nur bedingt möglich ist. Daher wird, jeweils nach dem Lesevorgang, die aktuelle Zeit genommen und die Zählerdifferenz durch die Zeitdifferenz geteilt um die Frequenz zu errechnen.

Da die jeweils drei Messwerte der vier DS2438 alle auf die gleiche Art abgelesen werden, ist der eigentliche Lesevorgang zusammen mit dem zugehörigen try-catch Block in einer mehrfach verwendeten Methode implementiert. Diese Methode liefert den Temperaturwert und zwei Spannungswerte in Form einer Instanz der (nachfolgend beschriebenden) FamilyCode26Values als Ergebnis. Aus diesen Werten — jeweils eines DS2438 — wird die Luftfeuchtigkeit, die Helligkeit und die Lautstärke errechnet.

Um die Genauigkeit der Temperaturmessung zu erhöhen, wird die mittlere Temperatur aller (temperaturmessfähigen) Bausteine berechnet, anstatt nur den Wert des DS1920 zu betrachten. Dadurch wird es sinnvoll, den Temperaturmesswert mit einer Auflösung von einem Zehntel Grad Celsius auszugeben.

Die in der Firmware enthaltenen Methoden zur Umwandlung von Fliesskommazahlen in Strings liefern nicht den korrekten Wert, sondern beispielsweise “23.4999999” anstelle von “23.5”. Der — durch diesen Rundungsfehler stark vergrösserte — String, der den Messwert und die Einheit enthält, würde nicht mehr komplett mit Bezeichner auf das lokale Display passen. Daher musste diese Funktionalität selbst implementiert werden.

5.4.8 FamilyCode26Values

Da diese Klasse ausschliesslich intern vom SensorWorker als strukturierter Datentyp genutzt wird, besitzt sie, um Speicher zu sparen, keine Methoden. Stattdessen kann auf ihre vier — als public deklarierten — Variablen direkt zugegriffen werden. Sie enthält die drei Messwerte jeweils eines DS2438 (vom Typ double), und eine Variable (vom Typ boolean), welche angibt, ob der Sensor fehlerfrei ausgelesen werden konnte oder nicht.

5.4.9 SensorAttributes

Diese Klasse repräsentiert die Attributwerte jeweils eines Sensors. Sie besitzt mehrere Methoden, um die Attributwerte zu initialisieren, zu aktualisieren und auszulesen. Die Initialisierungs- und Aktualisierungsroutinen werden durch den SensorWorker

aufgerufen; die verschiedenen Methoden, welche die einzelnen Attributwerte auslesen, werden von dem (nachfolgend beschriebenen) AwmlComposer genutzt. Bei der Aktualisierung eines Messwertes — durch die Methode update(String value, String accuracy) — wird die aktuelle Zeit als Attribut actuality gespeichert. Dabei war ursprünglich geplant, die Genauigkeit dynamisch anzugeben, da einige Sensoren in bestimmten Messbereichen genauere oder ungenauere Werte liefern. Aufgrund des Speichermangels wird diese Funktionalität nicht genutzt, beziehungsweise bei jedem Sensor immer mit einer zugehörigen, fest gegebenen Geanuigkeit.

5.4.10 AwmlComposer

Der AwmlComposer erstellt aus den SensorAttributes des SensorWorkers und dem SensorRequest die AWML-Antwort. Falls Fehler beim Parsen auftraten, erstellt er einen entsprechenden SOAP-Fault. Wenn der Fehler innerhalb des SOAP-Body auftritt, enthält der SOAP-Fault auch ein <detail> Element. Dessen <stackTrace> Element im Fehlerfall lautet immer "<stackTrace>No other stack trace info available.</stackTrace>", da die stackTrace Methoden in der Firmware des TINI nicht vollständing implementiert sind, bzw. nur den Namen der Exception liefern, der bereits im <message> Element übermittelt wird. Der NOL[1] wird aus dem Namen, unter dem der Server (im HttpHeader) angesprochen wurde, sowie dem dabei angegebenen Port und den UUIDs[2] der zugehörigen Objekte zusammengesetzt. Durch die Verwendung des im HttpHeader angegebenen Servernamens (IP-Adresse ebenfalls möglich) und Serverports — anstelle der realen Adresse und des realen Ports — kann der ContextCube auch hinter einem NAT-Router mit Portforwarding unproblematisch angesprochen werden, ohne dass der Name (die IP-Adresse) in einer weiteren Konstanten manuell gesetzt werden muss. Diess ist allerdings aus Konsistenzgründen nur sinnvoll, wenn der ContextCube ausschliesslich über den Router erreichbar ist. Die Templates für die SOAP-AWML Antwort enthalten an einer oder mehreren Stellen jeweils eine Variable, die wiederum durch ein weiteres Template oder den Rückgabewert einer Methode aus dem SensorRequest oder dem SensorWorker ersetzt wird.

5.5 JAVA und Embedded Systems

Der TINI besitzt ein Megabyte Speicher, was — für ein in einer Hochsprache programmierbares System — nicht besonders viel ist. Dies schlägt sich auch in der Firmware nieder: Zum Beispiel haben die in der Firmware enthaltene Methode Double.toString() und die anderen damit verwandten Methoden der Float Klasse einen Rundungsfehler: System.out.println(3.05f); erzeugt manchmal einen String wie "3.0499999".

[1] Nexus Object Locator
[2] Universal Unique Identifier

Um in JAVA geschriebene Programme auf dem TINI ausführen zu können, müssen die Class-Dateien mit dem TiniConvertor — einem von Dallas Semiconductors zur Verfügung gestellten Programm — in eine Tini-Datei umgewandelt werden.

Da das Programm während der Implementierungsphase oft zu groß wurde und dadurch nicht mehr lauffähig war, mussten viele Details optimiert werden. Interessant dabei ist, dass nicht nur die absolute Programmgrösse, sondern auch innere Details für die Lauffähigkeit entscheidend sind. Da dieses Verhalten aber nur schwer durchschaubar ist, und die Programmgröße das Verhalten mitbestimmt, wurde die Software nur auf ihre absolute Größe hin optimiert. Dabei wurde ständig an der Obergrenze entwickelt, weshalb die Lauffähigkeit teilweise von Größenänderungen um weniger als fünf Byte — beispielsweise der Deklaration nur eines lokalen Bytes — abhing.

Um die Software auf den vorhandenen Speicher zu optimieren, gibt es verschiedene Möglichkeiten, von denen alle im Verlauf dieser Arbeit — teilweise sogar iteriert — angewandt wurden. Einige dieser Methoden verschlechtern allerdings die Lesbarkeit und damit auch die Wartbarkeit des Codes:

- Die Verwendung von expliziten Zahlen anstelle von Konstanten spart jeweils drei Byte. Um dem dadurch enstehenden schlechten Programmierstil entgegenzuwirken wurde neben jeder Zahl die Konstante, die sie repräsentiert, als Kommentar stehen gelassen
- Der Verzicht auf else-Zweige bei disjunkten Conditions kostet zwar etwas Rechenzeit, spart aber Speicher.
- Teilweise ist ein “Conditional And” oder ein “Conditional Or” kleiner als die normale Variante, teilweise aber auch grösser. Daher muss, um diese Anweisungen zu optimieren (sofern für deren Funktion nicht eine der Varianten zwingend erforderlich ist), für jede von ihnen einzeln ein Durchlauf des TiniConvertors erfolgen, um feststellen zu können, welche Variante sich speichersparender verhält.
- Die Nutzung von so wenigen Variablenzuweisungen wie möglich.
  So ist beispielsweise

  ```
  runs=(int)((((LCDKey.readCounter(15)-LCDKey.readCounter(14))%5)+5)%5);
  ```

  in der aktuellen Version der ContextCube Software lauffähig. Die zuerst geplante Variante mit

  ```
  runs = (int)((LCDKey.readCounter(15)-LCDKey.readCounter(14))%5);
  if (runs<0) runs=runs+5;
  ```

  machte durch die weitere if-Abfrage und die weitere Wertzuweisung das Programm so groß, daß es nicht mehr lauffähig war.
- Die Verwendung lokaler Variablen anstelle globaler Variablen spart elf Byte und ist im Gegensatz zu den anderen Optimierungsmethoden auch noch besserer Programmierstil.

Kapitel 6

Eichung der Sensoren

6.1 Genauigkeit der Sensoren

Ursprünglich war geplant, die Genauigkeit — in Form des accuracy-Attributes — dynamisch anzugeben, da einige Sensoren in bestimmten Messbereichen genauere oder ungenauere Werte liefern. Eine if-Abfrage, welche das accuracy-Attribut dynamisch setzt, war aufgrund der Speicherknappheit nicht mehr realisierbar. Daher ist die fest angegebene Genauigkeit bezogen auf den Messbereich, für den der jeweilige Sensor ausgelegt wurde.

6.2 Temperatur

Die Temperaturmesswerte wurden nur im Bereich von 20-25°C genauer untersucht. Referenzmessgeräte waren ein Multimeter mit Temperaturmessbereich und ein weiteres Digitalthermomether. Die Abweichung untereinander beträgt weniger als 0,5°C (bei 20-25°C)

Die vier Smart Battery Sensors (DS2438) liefern alle sehr gute Messwerte, die untereinander (und auch verglichen mit den Messwerten der Referenzgeräte) um weniger als 0,5°C abweichen. Die Auflösung des DS2438 beträgt 0,03125°C bei einem vom Hersteller mit maximal +-2°C angegebenen Messfehler über den Bereich von -40°C bis 85°C.

Der iButton (DS1920) liefert einen um ca. zwei bis drei Grad höheren Wert[1] als die zwei Referenzthermometer. Die Auflösung des DS1920 liegt bei 0,5°C, was zugleich der vom Hersteller angegebenen Genauigkeit im Bereich von 0°C bis 70°C entspricht.

Damit hat die vom ContextCube gelieferte Temperatur einen Messfehler von maximal +-2°C beim Betrieb nur eines Temperatursensors. Sind alle fünf Temeratursen-

[1] Eventuell ein fehlerhaftes/schlechtes Teil?

soren angeschlossen und werden deren Messwerte gemittelt, so weicht die gemessene Temperatur noch weniger von der tatsächlichen Temperatur ab.

6.3 Luftfeuchtigkeit

Die vom Hersteller angegebene Genauigkeit des — für diese Arbeit nicht erhältlichen — Hygro iButtons DS1910, liegt bei +-5%. Da die selbst nachgebaute Schaltung ihre Versorgungsspannung — im Gegensatz zum DS1910 — über eine separate Leitung und nicht vom 1-Wire Bus bezieht, sollte die Genauigkeit dieser Schaltung eher besser sein. Die Genauigkeit des analogen Luftfeuchtesensors (HIH-3610) ist bereits vom Hersteller mit +-2%rH angegeben (bei 5V und 25°C). Der Messfehler des, als Dual-A/D-Wandler und zur Temperaturkompensation eingesetzten, DS2438 liegt unter +-75mV bzw. unter +-2°C.

6.4 Lichtintensität

Der erste Lichtsensor (mit LDR) wurde nicht geeicht, da das Programm im Endzustand bereits mit einer einzigen weiteren if-Abfrage — aufgrund des knappen Speichers — nicht mehr auf dem TINI lauffähig ist und daher ein Sensor fest ausgewählt werden musste. Der zweite Sensor (mit Solarzelle) liefert ausserdem, vor allem wenn dessen Solarzelle noch durch einen Licht-Spannungs-Wandler ersetzt wird, eine besser verwertbare Ausgangskennlinie. Für die Festlegung einer Eichkurve wurden sowohl der Sensor als auch ein Multimeter mit LUX-Messbereich mit einem dimmbarem 500W Halogenstrahler beleuchtet und dabei der Messwert des Multimeters und die Spannung an der Solarzelle gegeneinander aufgetragen. Der Halogenstrahler liefert allerdings — möglicherweise aufgrund von Netzspannungsschwankungen — keine wirklich konstante Helligkeit. Die Implementierung der zur (näherungsweisen) Umrechnung benötigten Exponentialfunktion scheiterte ebenfalls an dem verfügbaren Speicherplatz

6.5 Beleuchtungsfrequenz

Um die Frequenz der Beleuchtung zu messen, werden über den Zeitraum von einer Sekunde, die Impulse gezählt. Der Messzeitraum schwankt — möglicherweise aufgrund der unterschiedlich starken Auslastung des TINI — um ungefähr +-0.09 Sekunden und wurde daher zwischen 1.00 und 1.17 Sekunden gewählt, weshalb die Impulsanzahl noch durch die Zeit geteilt wird. Da die Zeit aufgrund der Firmware nur mit einer Auflösung von 10ms zur Verfügung steht, beträgt die Genauigkeit das Maximum von +-1% des Messwertes und +-1Hz. Die Genauigkeit beträgt also +-1Hz bei 0-100Hz und +-2Hz bei 101-200Hz.

6.6 Lautstärke

Der Lautstärkesensor wurde, soweit möglich, mittels eines künstlichen Audiosignals und eines Multimeters mit dB-Bereich geeicht. Der Sensor hat einen nichtlinearen Frequenzgang. Daher wurde er nicht mit einem 1kHz Sinussignal, sondern mit einem Rauschsignal — pink Noise [2] weil dafür ein Generator zur Verfügung stand — geeicht. Besser als das rosa Rauschen wäre ein weisses Rauschen gewesen [3], für das allerdings kein Generator zur Verfügung stand. Unterhalb einer bestimmten Lautstärke wird immer ein Wert von ca. 70 bis 75 dB angezeigt. Dies ist mit der aktuellen Sensorschaltung nicht anders zu realisieren, da die gemessene Ausgangsspannung des Verstärkers aufgrund des unerwartet starken Grundrauschens nicht unter den Wert sinken kann, den sie bei 70 bis 75 dB hat. Dieses Verhalten wurde erst so spät entdeckt, weil der Aufbau vorher immer im selben Raum mit einem Lüftungsgeräusche abstrahlenden Rechner stand. Da bei der Lautstärke normalerweise nur das Maximum eines (kurzen) Zeitintervalles von Bedeutung ist und die Schaltung für diesen Zweck ausgelegt wurde, fällt der angegebene Messwert nach einem lauten Geräusch nur langsam ab.

[2] Rauschen mit etwas stärkeren tieffrequenten Anteilen

[3] Das Audiofrequenzspektrum kann der ContextCube nicht messen.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Kombination aus JAVA, XML und Embedded System aufgrund der aktuellen verfügbaren Hardware mit vielen Kompromissen verbunden, da nicht genügend Speicher zur Verfügung steht. Trotzdem konnte die prinzipielle Aufgabenstellung umgesetzt werden, allerdings waren nicht alle Ideen/Entwürfe implementierbar.

Die komplett zusammengebaute Schaltung ist zu Vorführungszwecken auf einer Holzplatte montiert. Dabei wurden die einzelnen Baugruppen nebeneinander gesetzt, um gut sichtbar zu sein wie Abbildung 7.1 zeigt. Daher ist die Grundfläche des ContextCube in dieser Anordnung weit grösser als für seine Funktion nötig.

Die Kosten für die Hardware des Gesamtsystems liegen bei ungefähr 220 Euro.

7.2 Ausblick

7.2.1 Mehr Speicher

Aufgrund der fortschreitenden technischen Entwicklung ist davon auszugehen, dass in absehbarer Zeit embedded Systeme mit mehr Resourcen zur Verfügung stehen.

Der Speichermangel könnte bereits durch die Aufrüstung des externen Speichers eines (aktuellen) TINI auf die maximal möglichen vier MB behoben werden. Die Aufrüstung erfordert allerdings eine SMD-Lötanlage.

Dallas Semiconductors hat zudem mittlerweile für Herbst 2002 den Nachfolger des auf dem aktuellen TINI verwendeten Prozessors angekündigt [26]. Dieser DS80C400 bezeichnete Prozessor hat eine um 25% höhere Standardtaktfrequenz von 50 MHz, die sich noch auf 75 MHz erhöhen lässt. Der DS80C400 kann maximal 16 MB Speicher verwalten.

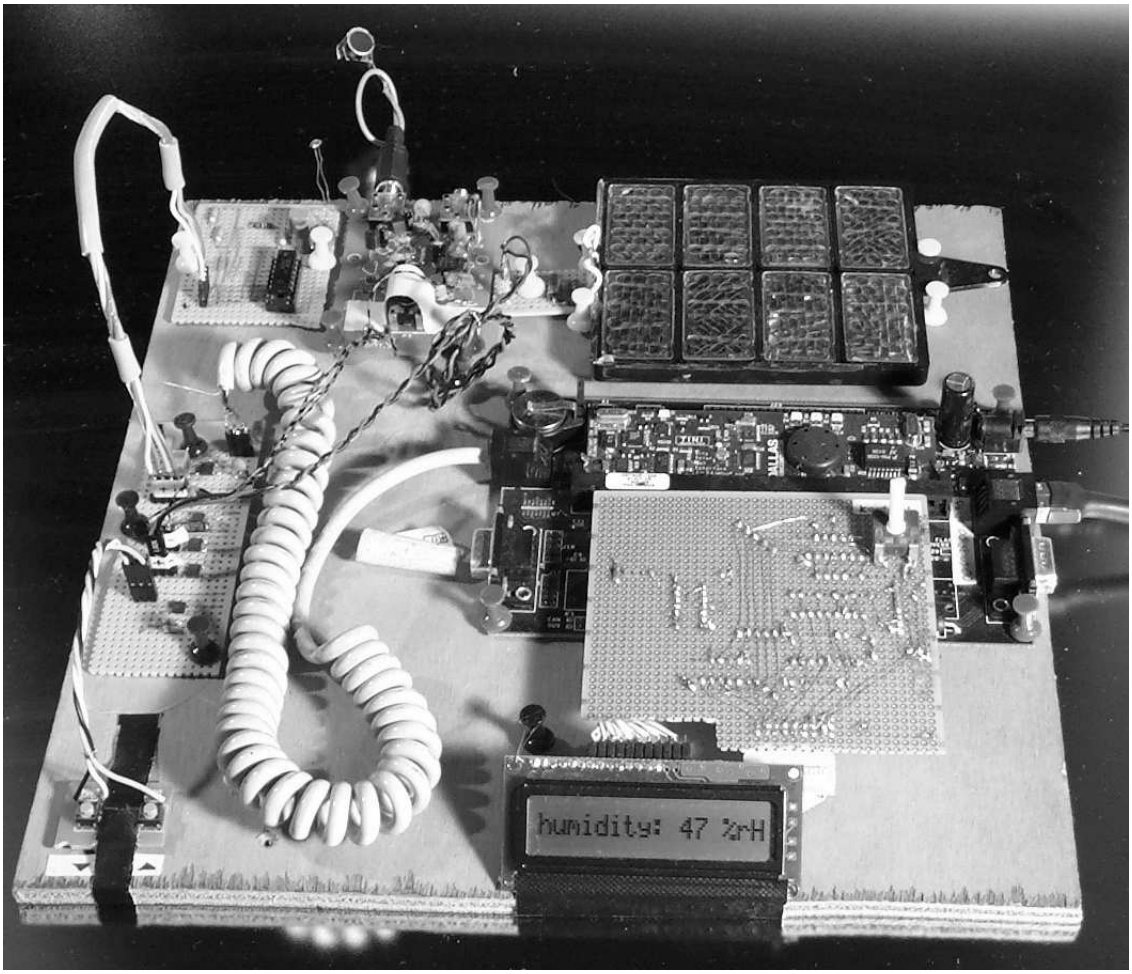

Abbildung 7.1: Der fertige ContextCube in Betrieb

Auch die Verwendung eines kleineren Servlet-fähigen HTTP-Servers wäre eine Möglichkeit mehr Resourcen für Erweiterungen freizugeben.

7.2.2 Weitere Sensoren

Durch mehr Speicher wäre es nicht nur möglich, die geplanten, aber nicht implementierten Teile nachzurüsten, sondern auch, den ContextCube um Sensoren für weitere Messwerte (beispielsweise Kohlenstoffdioxidgehalt der Luft) zu erweitern. Der Anschluss zusätzlicher Sensorik ist durch den 1-Wire Bus als Interface ohne großen Aufwand zu erledigen.

Da sich die TINI-Hardware auch um eine IrDA-Schnittstelle erweitern lässt, könnten die Sensordaten zusätzlich auch über diese angeboten werden. Die IrDA-Schnittstelle kann auch als eine Art Sensor zur Erkennung anderer IrDA-fähiger Geräte wie Laptops und PDAs genutzt werden. Die Ids und die Anzahl der — in der Umgebung des ContextCubes betriebenen Geräte können dann, ähnlich wie die anderen Senso-

rinformationen, abgerufen werden. Anhand der Ids kann — zumindest, wenn es sich um personengebundene Geräte handelt — auch festgestellt werden, welche Personen sich in einem Raum befinden. Daher sollte — aus Gründen des Datenschutzes — der Zugriff auf diese Erweiterung besonders restriktiv gehandhabt werden.

7.2.3 Erweiterung des ContextCube zum Aktor

Der ContextCube kann auch zum Aktor erweitert werden. Beispielsweise könnte die Beleuchtung oder eine Kaffeemaschine in einem Raum über den 1-Wire Bus des TINI (oder auch über IrDA) auf einen AWQL update-Request hin an- oder ausgeschaltet werden.

Sofern die IrDA-Schnittstelle auch auf niedriger Ebene angesprochen werden kann ist es auch möglich, fernsteuerbare Geräte wie einen Fernseher oder eine Stereoanlage über den ContextCube zu steuern. Da allerdings, je nach Hersteller und Gerät, einer von mehr als 500 verschiedenen Fernsteuercodes benutzt wird, sollte der ContextCube dann zusätzlich einen Lernmodus für unterschiedliche Gerätecodes besitzen. Alternativ könnten die Codes auch über das angeschlossene Netzwerk bezogen werden.

Literaturverzeichnis

[1] Messmer, Jens: Modellierung der Augmented World in Nexus (Diplomarbeit Nr.1870 2001)

[2] Schwarz, Thomas; Nicklas, Daniela; Großmann, Matthias; Volz, Steffen: Information Management and Exchange in Nexus, Research Group Nexus (Technical Report, Universität 2001)

[3] http://www.nexus.uni-stuttgart.de/, Research Group NeXus (Projektseiten)

[4] Spatial Model Server für NeXus, Research Group NeXus

[5] http://www.ibutton.com/TINI/index.html

[6]ftp://ftp.dalsemi.com/pub/tini/old/tini1_02d.tgz

[7]http://E.webring.com/hub?ring=tiniring

[8] http://www.dreamfabric.com/tini/

[9] http://www.systronix.com/tutor/tini_help/busload.htm

[10] http://www.systronix.com/tutor/tini_help/onewire/onewire480.jpg

[11] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2956

[12] http://pdfserv.maxim-ic.com/arpdf/DS80C390.pdf

[13] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2818

[14] http://pdfserv.maxim-ic.com/arpdf/DS1920.pdf

[15] http://www.ibutton.com/weather/humidity.html

[16] Awtrey, Dan: The 1-Wire Humidity Sensor (Sensors magazine, August 2000) (http://www.ibutton.com/weather/humsensor.pdf)

[17] http://www.maxim-ic.com/quick_view2.cfm?qv_pk=2903

[18] http://pdfserv.maxim-ic.com/arpdf/DS2401.pdf

[19] http://www.maxim-ic.com/quick_view2.cfm?qv_pk=2907

[20]http://pdfserv.maxim-ic.com/arpdf/DS2406.pdf

[21] http://www.maxim-ic.com/quick_view2.cfm?qv_pk=2912

[22] http://pdfserv.maxim-ic.com/arpdf/DS2422-DS2423.pdf

[23] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2919

[24] http://pdfserv.maxim-ic.com/arpdf/DS2438.pdf

[25] http://pdfserv.maxim-ic.com/arpdf/AppNotes/app195.pdf (Application Note)

[26]http://www.ibutton.com/TINI/ds80c400.pdf

[27] http://www.conrad.de/

[28] http://content.honeywell.com/sensing/prodinfo/humiditymoisture/technical/faq_hummoisture.stm

[29] http://www.servertec.com/products/tini_iws/tini_iws.html

[30] http://www.smartsc.com/

[31] http://www.smartsc.com/tini/TiniHttpServer/index.html

[32] http://www.smartsc.com/tini/TiniHttpServer/docs/Limitations.html

[33] http://www.gnu.org/copyleft/gpl.html

[34] http://www.apache.org/

[35] http://xml.apache.org/xerces2-j/index.html

[36] http://www.w3.org/TR/SOAP/

[37] http://lists.dalsemi.com/maillists/tini/

[38] http://www.opentext.com

[39] http://www.opensource.org/licenses/bsd-license.php

[40] http://www.textuality.com/

[41] http://www.textuality.com/Lark/

[42] http://www.wilson.co.uk/xml/minml2.htm