IMSAS

SFB 637
Autonomous Logistics

Qualitätsorientierte autonome Routen-Entscheidung in der Lebensmittel-Transportlogistik durch Imote2-Sensorknoten mit embedded Linux

Eine Dokumentation

von
Dipl.-Ing. Dirk Hentschel

22. Dezember 2009

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

AD	Analog-Digital (Wandler)
ARM	Acorn Risc Machine
ASCII	American Standard Code for Information Interchange
CDT	C/C++ Development Tools (Eclipse-Plugin)
CGI	Common Gateway Interface
CLR	Common Language Runtime (Laufzeitumgebung für C# auf Imote2)
ECP	Extended Capabilities Port (Betriebsmodus der parallelen Schnittstelle)
FEFO	First Expired First Out
FIFO	First In First Out
GPS	Global Positioning System
IP	Internet Protocol
ISM	ISM-Band: Industrial Scientific Medical (lizenzfrei)
JTAG	Joint Test Action Group
NTP	Network Time Protocol
OCD	On-Chip Debugging
OOP	Objekt-orientierte Programmierung
RISC	Reduced Instruction Set Computing
SCP	Secure Copy
SFB637	Sonderforschungsbereich 637 - Selbststeuerung logistischer Prozesse – Ein Paradigmenwechsel und seine Grenzen
SPOT	Smart Personal Objects Technology (Microsoft)
SSH	Secure Shell
STL	Standard Template Library
TAOS	Texas Advanced Optoelectronic Solutions
TCP	Transmission Control Protocol
TI	Texas-Instruments
TSP	Travelling Salesman Problem
VM	Virtual Machine
WSN	Wireless Sensor Network

1. Einleitung

Ein heutzutage neu gekaufter Imote2-Sensorknoten ist in der Regel für das *.NET Microframework* von Microsoft vorbereitet. Er lässt sich mit dem Visual Studio in der objektorientierten Programmiersprache *C#* programmieren. Das .NET-Microframework ist eine Anpassung des .NET Frameworks an Systeme mit geringen Ressourcen. Insbesondere in Hinsicht auf die Imote2-Plattform befindet sich das .NET-Microframework allerdings immer noch im Entwicklungsstadium. Die Bibliotheken zur Ansteuerung der Hardware sind unvollständig und teilweise fehlerhaft. Einige Funktionen können somit nicht genutzt werden, obwohl die Hardware sie unterstützen würde. Eine Gegenüberstellung der möglichen Funktionen ist in Abschnitt 2.4 auf Seite 7 beschrieben.

Um alle Vorteile des Sensorknotens ausnutzen zu können, ist die Verwendung eines anderen Betriebssystems eine mögliche Lösung. Hierbei kommen zwei Alternativen in Frage. Zum einen *TinyOS*, ein sehr leichtgewichtiges Betriebssystem, das auf die beschränkten Ressourcen von Sensorknoten zugeschnitten ist. Zum anderen Linux, das auch als Betriebssystem vollwertiger PC- oder Server-Systeme bekannt ist. Die beiden genannten Alternativen haben gemeinsam, dass der Quellcode frei verfügbar ist (Open Source). Dies hat den Vorteil, dass Anwender ihre eigenen Erweiterungen zu dem System beitragen können. Dadurch entwickelt sich das System ständig weiter. Es passt sich somit immer besser an die Hardware an, auf der es eingesetzt wird.

Software für TinyOS wird in der Sprache *nesC* geschrieben. Dabei handelt es sich um eine Erweiterung der Programmiersprache *C*, die speziell für drahtlose Sensornetzwerke mit geringen Ressourcen entwickelt wurde. TinyOS wird auf zahlreichen Sensor-Knoten erfolgreich eingesetzt, wie zum Beispiel auf den TelosB-Knoten (TmoteSky). Auch die Imote2-Sensorknoten wurden wurden ursprünglich mit dem Betriebssystem TinyOS ausgeliefert. Es steht daher ein umfangreiches Softwareangebot für dieses System zur Verfügung.

Die Imote2-Plattform bietet mit ihrer großzügigen Hardware-Ausstattung (siehe Abschnitt 2) jedoch deutlich mehr Möglichkeiten, als mit TinyOS realisiert werden können. Daher habe ich mich für das Betriebssystem Linux entschieden. Es stellte sich jedoch heraus, dass sich das Imote2-Linux noch in einer sehr frühen Entwicklungsphase befindet. Es gibt zwar eine große Imote2-Community[1], in der ständig über Erweiterungen diskutiert wird. Das Team der aktiv an der Umsetzung beteiligten Entwickler besteht jedoch nur aus wenigen Personen, die überwiegend in ihrer Freizeit daran arbeiten. Einige Funktionen sind somit immer noch nicht implementiert. Daher muss eventuell eine eigenständige Weiterentwicklung auf Kernel- und Treiber-Ebene vorangetrieben werden.

[1] Imote2-Community: http://tech.groups.yahoo.com/group/intel-mote2-community/

2. Imote2

Ein Imote2-Sensorknoten besteht aus drei verschiedenen Boards, die sich zusammenstecken lassen: Ein Batterieboard, ein Prozessorboard und ein Sensorboard. Abbildung 2.1 zeigt einen zusammengesteckten Imote2-Sensorknoten. In Abbildung 2.2 auf der nächsten Seite sind die Boards einzeln dargestellt. Das Prozessorboard lässt sich per USB mit dem PC verbinden. Dann kann auf den Einsatz des Batterieboards verzichtet werden. Das Sensorboard kann weggelassen werden, wenn für die jeweilige Aufgabe des Knotens keine Sensorkomponenten benötigt werden.

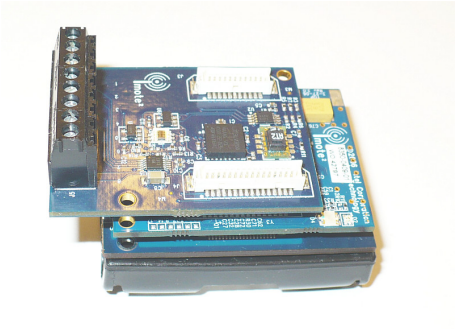

Abbildung 2.1.: Imote2-Sensorknoten

Vom Hersteller *Crossbow* ist ein Starterkit erhältlich, das aus drei Prozessorboards, zwei Sensorboards und zwei Batterieboards besteht. Das Starterkit nennt sich *Imote2.Builder Kit* und ist derzeit für ca. 1000 € über die Firma CMT zu beziehen.

Jedes der drei Boards besitzt mehrere Steckverbinder der Firma *Hirose*, mit denen die Boards miteinander verbunden werden können. Über diese Steckverbinder lässt sich auch zusätzliche Hardware anschließen. Von den insgesamt vier verschiedenen Steckverbindern wurde ein kleiner Vorrat (jeweils 2-5 Stück) beschafft, um eigene Hardware-Erweiterungen erstellen zu können. Falls dieser Vorrat nicht ausreicht, können über die Firma Digikey[1] zum Stückpreis von jeweils 1-2 € welche nachbestellt werden. In Tabelle 2.1 auf der nächsten Seite sind die Artikelnummern der Teile aufgelistet. Wegen der Versandkosten in Höhe von 18 € und der eventuell anfallenden Zollgebühren lohnt sich die Bestellung nur bei größeren Abnahmemengen.

2.1. Prozessorboard IPR2400

Das Herzstück des Imote2-Sensorknotens ist das *Prozessorboard IPR2400*. Es ist in Abbildung 2.2 in der Mitte dargestellt. Der Prozessor ist ein *ARM Xscale PXA271*. Die Taktfrequenz kann von

[1] Digikey: http://www.digikey.com/

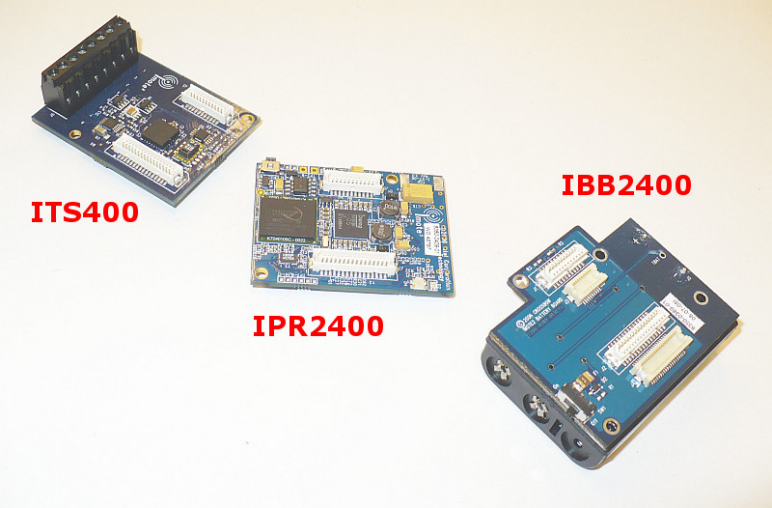

Abbildung 2.2.: Sensorboard, Prozessorboard und Batterieboard

Digikey-Nr	Teilenummer Hirose	Pins	Einsatzort
H10403CT-ND	DF9-21S-1V(32)	21	Oberseite Prozessorboard J5
H10406CT-ND	DF9-31S-1V(32)	31	Oberseite Prozessorboard J7
H10415CT-ND	DF9-21P-1V(32)	21	Unterseite Sensorboard J1
H10418CT-ND	DF9-31P-1V(32)	31	Unterseite Sensorboard J2
H10445CT-ND	DF15B(3.2)-20DP-0.65V(56)	20	Unterseite Prozessorboard J3
H10447CT-ND	DF15B(3.2)-40DP-0.65V(56)	40	Unterseite Prozessorboard J4

Tabelle 2.1.: Hirose Steckverbinder

13-416 MHz variiert werden. Der Chip ist mit 256kB SRAM, 32MB FLASH und 32MB SDRAM ausgestattet.

Auf der Board-Rückseite ist eine Antenne für das 2,4 GHz-ISM-Band angebracht. In der Abbildung ist diese jedoch nicht zu sehen. Weiter hinten auf Seite 41 befindet sich allerdings ein weiteres Foto des Prozessorboards, auf dem die Rückseite dargestellt ist. Die drahtlose Übertragung erfolgt nach dem IEEE-802.15.4-Standard unter Verwendung des weit verbreiteten Chips CC2420 von Texas Instruments (TI). Das Board besitzt außerdem eine mehrfarbige LED, die per Software angesteuert werden kann.

Das Prozessorboard verfügt über eine USB-Schnittstelle. Hierüber kann eine Verbindung mit dem PC hergestellt werden. In diesem Fall kann auf den Einsatz des *Batterieboards IBB2400* verzichtet werden, da die Spannungs-Versorgung über die USB-Schnittstelle des PC erfolgt. Es empfiehlt sich der Einsatz eines USB-Hubs um die Schnittstelle nicht zu sehr zu belasten. Es ist zu berücksichtigen, dass sich das Board nicht automatisch einschaltet. Zum Einschalten muss der kleine Taster betätigt werden, der sich in der Nähe der USB-Buchse befindet.

Über die Hirose-Steckverbinder sind einige Schnittstellen herausgeführt. Dies sind im einzelnen 3xUART, I2C, 2xSPI, SDIO, I2S, AC97, USB host, Camera I/F, GPIO. Weitere Informationen hierzu sind dem *Imote2.Builder Kit Manual* zu entnehmen[12]. Für die serielle Schnittstelle hat der Student Ishwar Lal[8] einen Adapter gebaut. Ursprünglich war der Adapter zum Anschluss eines seriellen Displays gedacht. Er lässt sich aber sehr flexibel auch für andere Zwecke einsetzen. Der Adapter ist in Abbildung 2.3 dargestellt.

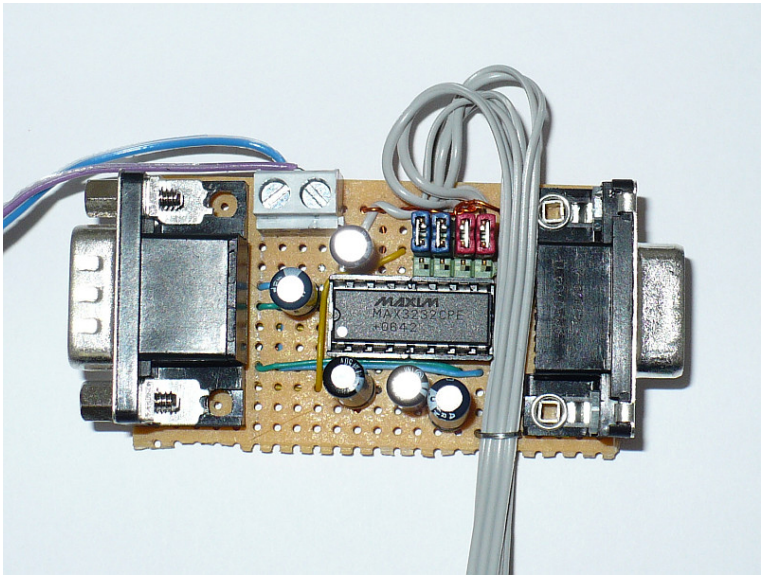

Abbildung 2.3.: Serieller Adapter zum Anschluss an Imote2[8]

2.2. Sensorboard ITS400

Ein Betrieb des Imote2-Sensorknotens ist auch ohne das *Sensorboard ITS400* möglich. Wenn der Sensorknoten beispielsweise als zentrales Gateway eingesetzt wird, kann auf die Sensorik verzichtet

werden. In einem Imote2.Builder-Kit sind daher nur zwei Sensorboards, aber drei Prozessorboards enthalten. Das Sensorboard verfügt über einen AD-Wandler, einen Temperatursensor von TI, einen kombinierten Temperatur- und Feuchtigkeitssensor von Sensirion, einen 3D-Beschleunigungssensor von STM sowie einen Lichtsensor von TAOS. Die Eigenschaften dieser Komponenten sind in Tabelle 2.2 dargestellt. Im Abschnitt 2.4 wird noch etwas näher auf den Einsatz des Sensorboards eingegangen.

Komponente	Eigenschaften	Hersteller und Typ
AD-Wandler	4 Kanäle, 12 Bit, 0-3 Volt	Maxim MAX1363
Temperatursensor	±1.5°C Genauigkeit, −25°C bis +85°C	TI TMP175
Temperatur- und Feuchtesensor	±0.3°C	Sensirion SHT15
3D-Beschleunigungs-Sensor	±2 g, 12 Bit Auflösung	STM LIS3 L02DQ
Lichtsensor	Visible Light and IR, 16 Bit	TAOS TSL2651

Tabelle 2.2.: Eigenschaften der Sensorkomponenten des Sensorboards ITS400[11]

2.3. Interface-Board IIB2400

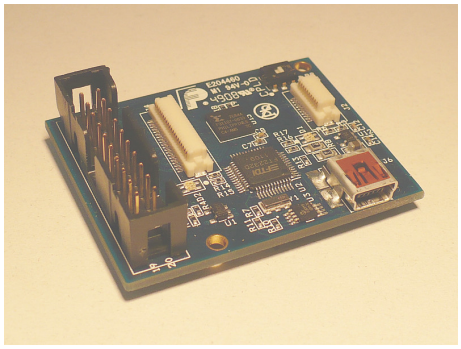

Abbildung 2.4.: Imote2 Debug-Board IIB2400

Ein weiteres Board, das mit Imote2 verbunden werden kann, ist das *Interface-Board IIB2400*. Dies ist nicht Bestandteil des Imote2.Builder-Kits und muss gesondert bestellt werden. Das Interfaceboard ist in Abbildung 2.4 dargestellt. Es verfügt über einen zusätzlichen USB-Anschluss. Wird dieser mit dem PC verbunden, erscheinen im Windows-Gerätemanager zwei neue Schnittstellen (z.B. `COM6` und `COM7`). Mit dem Hyperterminal, das Bestandteil des Windows-Betriebssystems ist, kann die Schnittstelle mit der höheren Nummer, in diesem Fall also `COM7`, geöffnet werden. Die Einstellungen sind wie im Bild 2.5 vorzunehmen. Wichtig ist, dass die Option Flusssteuerung auf „keine“ eingestellt wird.

Des weiteren verfügt das Interfaceboard über einen 20-poligen Wannenstecker, an den ein ARM-JTAG-Device angeschlossen werden kann. Hierdurch wird ein Debugging der ausgeführten Software möglich. Das Interface-Board wird daher auch als Debug-Board bezeichnet. Über die JTAG-Schnittstelle lässt sich z.B. der Bootloader austauschen. Hierauf wird in Abschnitt 3.1 genauer eingegangen.

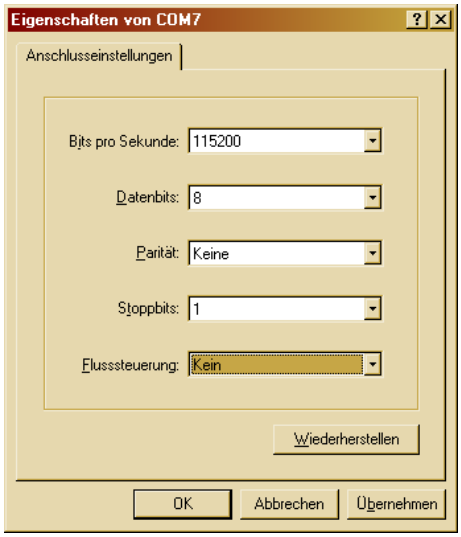

Abbildung 2.5.: Einstellungen für das Programm Hyperterminal

2.4. Vergleich der Betriebssysteme

In der nachfolgenden Tabelle 2.3 sind die Fähigkeiten der drei Betriebssysteme gegenübergestellt. Untersucht wurden bisher ausschließlich die beiden Betriebssysteme *Linux* und *.NET Microframework* (.NET MF), so dass über den Einsatz von *TinyOS* auf der Imote2-Plattform nur Vermutungen geäußert werden können. In der Tabelle sind vorhandene Eigenschaften durch einen Haken (✔), und nicht vorhandene Eigenschaften durch ein Kreuz (✗) dargestellt.

Es wurden wesentliche Unterschiede zwischen dem .NET Microframework und Linux festgestellt. So ist es unter .NET nicht möglich, während der Laufzeit Daten vom PC zu empfangen. Die USB-Verbindung ist unidirektional. Das heißt, dass nur das Senden von Debug-Messages an den PC möglich ist. Das Senden in umgekehrter Richtung ist nicht möglich. Es wurde versucht, diese Einschränkung durch Verwendung der seriellen Schnittstelle zu umgehen. Dies scheiterte jedoch an der fehlerhaften Implementierung der seriellen Schnittstelle in der Bibliothek des .NET Microframeworks.

Bei Linux hingegen wird die USB-Verbindung wie eine Netzwerkverbindung behandelt. Der Sensorknoten ist also per TCP/IP erreichbar, so dass über einen sogenannten Socket gemäß etablierter Netzwerkstandards Daten ausgetauscht werden können. Dank des vorhandenen SSH-Servers ist sogar ein interaktiver Login möglich.

Die Abfrage des Sensorboards funktioniert unter .NET sehr gut. Alle vorhandenen Sensoren für Temperatur, Feuchtigkeit, Licht, Beschleunigung sowie die AD-Ports konnten wie vorgesehen abgefragt werden. Durch Abwandlung der vorhandenen Beispielprogramme `SensorITS400` und `SensorITS400-Sleep` lassen sich leicht eigene Programme erzeugen.

Das Linux-Betriebssystem hingegen zeigt hier seine Schwächen. Für jeden einzelnen Sensor muss zunächst ein Treibermodul in den Kernel geladen werden. Anschließend lässt sich die Sensorhardware ansprechen, indem die Treiberdatei `/dev/sensor...` abgefragt wird. Die Module wurden getrennt voneinander entwickelt und befinden sich daher in unterschiedlichen Entwicklungsstadien. Bisher konnten lediglich die Treibermodule für die Temperatur-, Feuchtigkeits- und Lichtsensoren getestet werden.

Fähigkeit	Linux	.NET MF	TinyOS
Sensorboard abfragen	teilw.✔	✔	vermutl.✔
Debug Ausgaben erzeugen	✔	✔	vermutl.✔
Daten vom PC empfangen	✔	✗	vermutl.✔
rechnen	✔	✔	vermutl.✔
Multicolor LED ansteuern	✔	✔	vermutl.✔
TOSMAC-Daten senden/empfangen	✔	✔	**vermutl.✔**
TelosB kompatible TOSMAC-Pakete	✗(?)	✗	vermutl.✔
Interaktive Steuerung (ssh-Login)	✔	✗	vermutl.✗
Serielle Schnittstelle	✔	✗	vermutl.✔
Online Debugging	?	✔	?
Energiesparmodus: Sleepmode	✗	?	?
Energiesparmodus: Deep-Sleepmode	✗	✔	?
Energiesparmodus: Stand By	?	?	?
Energiesparmodus: Idle	✔	?	?
Taktfrequenz ändern	✗	?	?
Java-Programme ausführen	✔(OSGi)	✗	✗
Quelltext verfügbar	✔	✗	✔

Tabelle 2.3.: Imote2 - Gegenüberstellung der Betriebssysteme

Die analogen Eingänge, die zum Anschluss von eigenen Sensoren, wie zum Beispiel einem Gas-Sensor verwendet werden könnten, konnten bisher noch nicht erfolgreich abgefragt werden.

Die untersuchenden Arbeiten hierzu laufen jedoch noch und werden ggf. im nächsten Jahr durch den Studenten Ishwar Lal fortgesetzt. Ein wichtiges Zwischenergebnis ist jedoch, dass es zwei Versionen des Sensorboards ITS400 gibt, die zu unterschiedlichen Zeitpunkten als Bestandteil des Imote2.Builder Kits gekauft wurden. Wie in Abbildung 2.6 zu erkennen ist, ist nicht nur die Farbe unterschiedlich, sondern auch die Anordnung der Bauelemente. Unter Linux verhalten sich diese beiden Boards außerdem unterschiedlich: Der 3D-Beschleunigungs-Sensor lässt sich beispielsweise mit dem neuen, blauen Sensorboard überhaupt nicht ansprechen.

Die Imote2-Sensorknoten besitzen (hardwaremäßig) die Fähigkeit, sich in einen Energiesparmodus zu versetzen. Dadurch wird die Lebensdauer der batteriebetriebenen Einheiten deutlich erhöht. Die Untersuchungen hierzu sind noch nicht erfolgt. Für einen praxisnahen Einsatz der Sensorknoten ist der Einsatz von Energiesparmodi jedoch zwingend erforderlich, weil die Batterien im Normalbetrieb maximal einen Tag halten und weil ein Batteriewechsel im laufenden Betrieb, wie zum Beispiel während eines zu überwachenden Transportvorgangs von Lebensmitteln, nur eingeschränkt möglich ist.

Das .NET Microframework liefert das oben bereits erwähnte Beispielprogramm `SensorITS400Sleep` mit, das sich nach einem versendeten Nachrichtenpaket in den Deep-Sleep-Modus versetzt. Dies ist unter Linux bisher nicht geglückt. Durch Messungen des Energiebedarfs konnte lediglich festgestellt werden, dass sich der Sensorknoten automatisch in einen sparsameren Idle-Modus umschaltet, wenn keine Programme abgearbeitet werden oder wenn das System auf Eingaben wartet.

In diesem Zusammenhang ist auch die Veränderung der Taktfrequenz zu nennen. Laut Datenblatt lässt sich die Taktfrequenz zwischen 13 und 416 MHz variieren. Dies ist aber bisher weder unter Linux noch mit dem .NET Microframework gelungen.

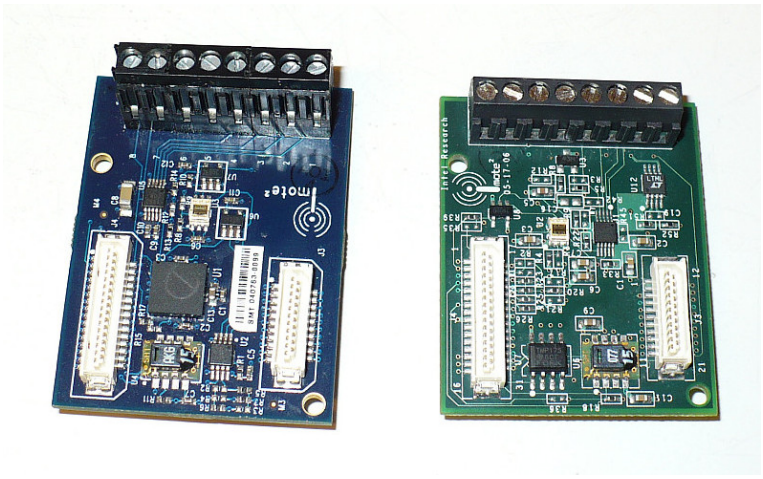

Abbildung 2.6.: Verschiedene Versionen des Imote2-Sensorboards ITS400

3. Installation von Linux auf Imote2

Linux auf dem Imote2-Knoten einzusetzen impliziert nicht, dass auch auf der PC-Seite Linux als Betriebssystem eingesetzt werden muss. Zur Zeit ist es jedoch erforderlich, zumindest eine virtuelle Linux-Maschine zu verwenden, weil ein USB-Treiber für den Bootloader *blob* derzeit unter Windows nicht verfügbar ist. Um die virtuelle Maschine verwenden zu können, muss das Programm *VMware Player*[1] installiert sein. Ein vorbereitetes VMware-Image mit Debian-Linux befindet sich im Projektverzeichnis auf Laufwerk `T`.[2] In dem vorbereiteten Image ist für den Superuser `root` das Passwort `sfb637` voreingestellt. Dieses sollte aus Sicherheitsgründen beim ersten Login geändert werden.

Nach dem Starten der virtuellen Maschine kann der Imote2-Linux-Sensorknoten per USB mit dem PC verbunden und eingeschaltet werden. Über die Menüleiste des VMware-Players muss dann bei Devices das USB-Gerät Compaq USB Device der virtuellen Maschine zugewiesen werden. Erst dann wird es möglich, eine TCP/IP-Verbindung mit dem Imote2-Sensorknoten herzustellen.

Es ist zu beachten, dass diese Vorgehensweise nur dann funktioniert, wenn auf dem Imote2-Sensorknoten bereits der Linux-Bootloader *blob* installiert ist. Bei einem neu gekauften Node aus dem *Imote2.Builder-Kit* ist jedoch der .NET-Bootloader vorinstalliert. Der Austausch des Bootloaders wird durch die Verwendung zusätzlicher Hard- und Software möglich. Zum einen wird das Imote2-Interface-Board benötigt (siehe Abschnitt 2.3). Zum anderen ist ein ARM-JTAG Adapter sowie die passende Software erforderlich. Eine Auswahl an JTAG-Adaptern wird auf den folgenden Seiten vorgestellt. Die Verwendung der passenden Software wird in Abschnitt 3.2 auf Seite 15 beschrieben.

Neben dem Bootloader-Image müssen zwei weitere Images in den Speicher des Imote2 übertragen werden. Das ist zum einen der Linux-Kernel und zum anderen das dazu passende Filesystem im JFFS2-Format. Diese beiden Images brauchen allerdings nicht über die JTAG-Schnittstelle übertragen werden. Sobald der Bootloader lauffähig ist, können die Daten per Xmodem-Protokoll vom PC auf den Imote2 geladen werden. Das Interfaceboard stellt eine serielle Schnittstelle zur Verfügung, auf die mit einem Terminalprogramm zugegriffen werden kann. In Abschnitt 3.3 wird diese Vorgehensweise näher erläutert.

In Abbildung[3] 3.1 auf der nächsten Seite ist die Unterteilung des Flash-Speichers des Imote2-Systems dargestellt. Im oberen Speicherbereich befindet sich der Bootloader. Er wird beim Einschalten automatisch gestartet. Der Bootloader lädt den Linux-Kernel, der sich an Speicheradresse 0x40000 befindet. Beim Starten des Kernels wird das JFFS2-Filesystem gemountet, das sich von Speicheradresse 0x240000 bis 0x2000000 erstreckt.

[1] VMware Player: http://www.vmware.com/de/products/player/
[2] Debian-Image für VMware: T:\Projekte\Projekt SFB_B6\Studenten\Software\VMware Image\
[3] Quelle: http://ubi.cs.washington.edu/wiki/index.php/IMote2

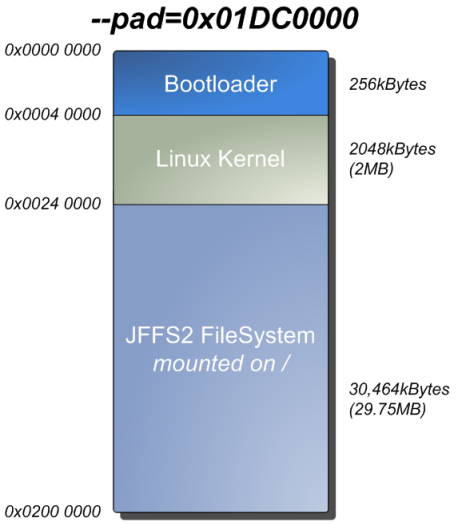

Abbildung 3.1.: Imote2-Flash-Filesystem

3.1. JTAG-Adapter

Es wurden drei verschiedene JTAG-Adapter verwendet: Macraigor OCDemon „Raven“, Olimex Arm JTAG „Wiggler“ und der Wiggler-Clone von Diygadget. Auf die Eigenschaften dieser drei Adapter werde ich in den folgenden Abschnitten eingehen. Erfolgreich einsetzen ließ sich der JTAG-Adapter von Diygadget.

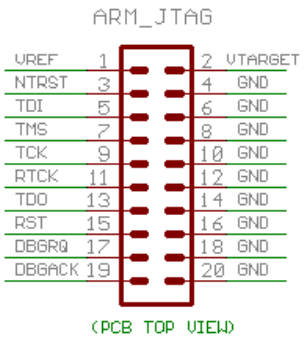

Abbildung 3.2.: Steckerbelegung nach dem JTAG Standard [Quelle: Olimex]

Allen Adaptern ist gemeinsam, dass sie einerseits über die parallele Druckerschnittstelle mit dem PC verbunden werden und andererseits über einen 20-poligen Steckverbinder an das Imote2-Debug-Board angeschlossen werden können. Die Belegung dieses Steckers entspricht dem JTAG Standard IEEE 1149.1. Sie ist in Abbildung 3.2 dargestellt.

3.1.1. Olimex JTAG-Adapter

Der ARM-JTAG-Adapter von Olimex[4] wurde beim Elektronikladen[5] bestellt. Mit 18,50 € war es der günstigste Adapter der drei getesteten. Laut Datenblatt ist er kompatibel zu dem weit verbreiteten JTAG-Adapter „Wiggler“.

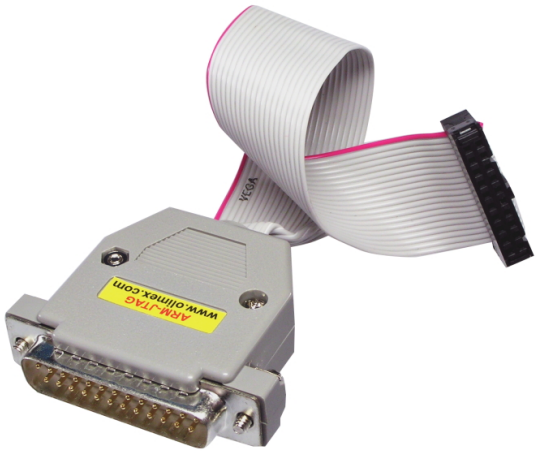

Abbildung 3.3.: Olimex ARM-JTAG-Adapter

Der in Abbildung 3.3 dargestellte Adapter ist kompakt aufgebaut. Alle Bauteile finden in dem Gehäuse des 25-poligen D-Sub-Steckers Platz. Dieser wird mit der parallelen Schnittstelle des PCs verbunden. Die Schnittstelle muss dafür in den ECP-Modus versetzt werden. Diese Einstellung wird im BIOS des PC vorgenommen. Eine externe Stromversorgung ist nicht erforderlich. Die Versorgung der Bauteile erfolgt über das Target, in diesem Fall also über das Imote2-Interface-Board.

Dieser Adapter wurde von der Software Flashprogrammer korrekt erkannt. Ein Beschreiben des Flash-Speichers war jedoch nicht möglich. Die Ursache dafür liegt aber vermutlich in der Software, da das gleiche Verhalten auch mit einem anderen JTAG-Adapter auftrat. Auf dieses Problem wird in Abschnitt 3.2.1 auf Seite 16 näher eingegangen. Die Software-Alternative Jflashmm erkannte den Olimex-Adapter hingegen nicht, so dass diese Software in Zusammenhang mit dem Olimex-Adapter nicht eingesetzt werden konnte.

3.1.2. Macraigor Raven

Der Macraigor Raven[6] ist ein sehr hochwertiger JTAG-Adapter. Mit 750 $ ist er allerdings der teuerste der drei JTAG-Adapter. Er befand sich allerdings schon am Institut und wurde nicht extra angeschafft. Wegen seiner On-Chip-Debugging-Fähigkeit wird er auch *OCDemon* genannt.

Im Gegensatz zum Wiggler kann er mit einer sehr viel höheren Geschwindigkeit betrieben werden. Während der Wiggler maximal mit 380 kHz betrieben werden kann, lässt der Raven Geschwindigkeiten bis 8 MHz zu. Es ist jedoch zu beachten, dass der Parallelport des Computers in einen anderen Modus versetzt werden muss. Im ECP-Modus darf der Raven nicht betrieben werden.

[4] Olimex Wiggler: http://www.olimex.com/dev/arm-jtag.html
[5] Elektronikladen: http://www.elektronikladen.de/armjtag.html
[6] Macraigor Raven: http://www.macraigor.com/raven.htm

Abbildung 3.4.: OC Demon „Raven“ JTAG-Adapter von Macraigor

Der Hersteller Macraigor Systems stellt auf seiner Webseite[7] eine eigene Software bereit, mit der die Programmierung eines Flash-Speichers vorgenommen werden kann. Das Programm heißt Flashprogrammer und unterstützt beide JTAG-Adapter-Typen, Raven und Wiggler. Der Flashprogrammer wird in Abschnitt 3.2.1 auf Seite 16 näher beschrieben.

3.1.3. Diygadget Buffered JTAG

In der Imote2-Community wurde der JTAG-Adapter von Diygadget[8] empfohlen. Es gibt mehrere Beiträge[9] im Nachrichten-Archiv der Mailingliste, die eine detaillierte Beschreibung liefern, wie dieser Adapter im Zusammenhang mit dem Imote2-Debug-Board und der Software Jflashmm eingesetzt wird.

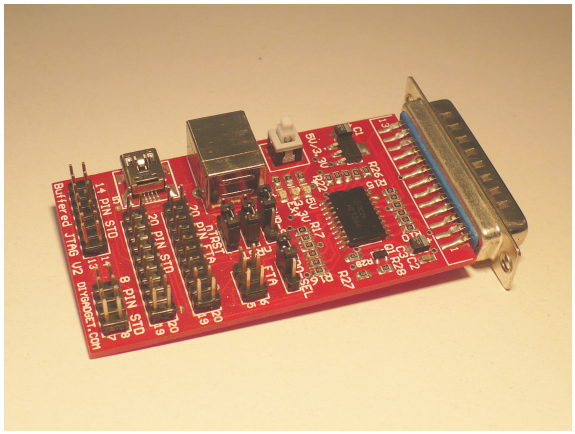

Abbildung 3.5.: Buffered JTAG V2 von Diygadget

Das Diygadget-Board wird über ein vollbelegtes Kabel mit der parallelen Schnittstelle des PCs verbunden. Für die Schnittstelle muss der ECP-Modus aktiviert sein. Das Board bietet mit Hilfe von

[7] Macraigor Systems: http://www.macraigor.com
[8] Diygadget: http://www.diygadget.com/jtag-cables/wiggler-buffered-all-in-one-jtag-programmer-version-2.html
[9] Imote2-Community, Beiträge 1497, 1524,1772, 1781, ... http://tech.groups.yahoo.com/group/intel-mote2-community

Jumpern zahlreiche Konfigurationsmöglichkeiten. Im Zusammenhang mit dem Imote2-Debugboard hat sich jedoch die folgende Kombination als einzig sinnvoll herausgestellt: Die Jumper `TDO_SEL` und `PWD_SEL` müssen nach links zeigen, die Jumper `nRST` und `nTRST` müssen nach rechts zeigen. In Abbildung 3.5 ist dies zu erkennen.

3.2. JTAG-Software

In Ergänzung zu der JTAG-Hardware ist eine Software erforderlich, mit der über die JTAG-Schnittstelle binäre Images auf die Imote2-Plattform übertragen werden können. Dabei wurden zwei verschiedene Programme untersucht. Zum einen der *Flashprogrammer* von Macraigor Systems, zum anderen das Open-Source-Programm *Jflashmm*, das ursprünglich von Intel entwickelt wurde.

Beide Programme sind universell einsetzbar und nicht speziell auf die Bedürfnisse der Imote2-Plattform abgestimmt. Es muss daher eine Konfigurationsdatei existieren, die Informationen über den Aufbau des Speichers und die möglichen Kommandos liefert. Bei Jflashmm heißt die Datei `bulbcx16.dat`. Es handelt sich dabei um eine Textdatei, die mit einem beliebigen Texteditor bei Bedarf entsprechend angepasst werden kann. Auch der Flashprogrammer besitzt eine Textdatei zur Konfiguration der Speicherarchitektur. Die Datei heißt `flash.ini`. Das Format der beiden Dateien ist allerdings grundlegend verschieden und somit nicht untereinander austauschbar.

Der Programmiervorgang über die JTAG-Schnittstelle ist fehleranfällig. Eine Fehlerkorrektur während der Übertragung findet nur bedingt statt. Daher ist es nach einem Schreibvorgang erforderlich, den Inhalt des Speichers mit der verwendeten Imagedatei Byte für Byte zu vergleichen. Eine solche Vergleichsroutine ist in beiden Programmen enthalten. Der Vergleich findet im Anschluss an eine Übertragung automatisch statt, kann aber auch gezielt separat initiiert werden.

Die beiden Programme haben neben ihren Gemeinsamkeiten auch einige Unterschiede. Der Flashprogrammer lässt sich bequem über eine GUI bedienen und bietet neben dem eigentlichen Programmiervorgang noch viele weitere Möglichkeiten. So lässt sich zum Beispiel in beliebige Speicherbereiche des Chips gezielt Einsicht nehmen. Außerdem ist es möglich, den bestehenden Speicherinhalt in eine Datei auf den PC zurück zu übertragen. Jflashmm hingegen ist deutlich einfacher aufgebaut und bietet einen geringeren Funktionsumfang. Es handelt sich um ein Konsolenprogramm, das in einem DOS-Fenster gestartet werden muss. Es gibt nur eine sehr begrenzte Anzahl an möglichen Funktionen, die mit Hilfe entsprechender Kommandozeilenparameter ausgewählt werden. Der Speicher der Targetplattform kann gelöscht, neu beschrieben und verglichen werden. Für den hier vorgesehenen Zweck reicht das allerdings aus.

Nachfolgend werden die beiden erwähnten Programme näher beschrieben. Vorweggenommen sei jedoch, dass eine erfolgreiche Übertragung des Bootloader-Images lediglich mit dem Programm Jflashmm möglich war. Nach neuesten Berichten[10] in der Imote2-Community ist inzwischen auch die Benutzung von OpenOCD möglich, um ein Flash-Image auf den Imote2 zu laden. Dies wäre eventuell eine zu untersuchende Alternative.

[10] Imote2-Community, Msg 3200: http://tech.groups.yahoo.com/group/intel-mote2-community/message/3200

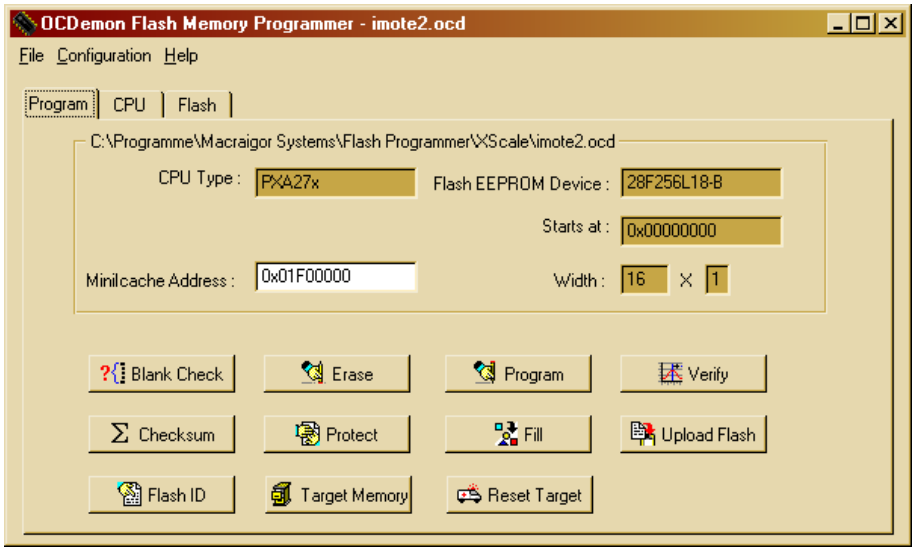

Abbildung 3.6.: Hauptfenster des Flashprogrammers

3.2.1. Macraigor Flashprogrammer

Der Flashprogrammer ist eine Software der Firma Macraigor Systems. Diese Firma ist auch Hersteller des JTAG-Adapters Raven, der in Abschnitt 3.1.2 vorgestellt wurde. Das Programm kann kostenlos von der Webseite[11] der Firma heruntergeladen werden. Es bietet die Möglichkeit, den Inhalt des Flash-Speichers zu lesen, ihn zu löschen oder neu zu beschreiben. Um den Speicher neu zu beschreiben, ist jedoch eine erweiterte Software-Lizenz erforderlich, die 500 $ kostet.

In Abbildung 3.6 ist das Programmfenster des Flashprogrammers nach dem Start dargestellt. Mit dem Button Program wird der Programmiervorgang gestartet. Es erscheint ein weiteres Fenster, in dem die Start- und End-Adresse des Speicherbereichs eingegeben, sowie die zu schreibende Imagedatei ausgewählt werden kann. Das Image muss im S19-Format vorliegen. Dieses Format wird in Abschnitt 3.2.2 auf Seite 18 beschrieben. Um das binäre Image in das S19-Format umzuwandeln, kann das Tool `BinToS19.exe` verwendet werden, das sich nach der Installation im Programmverzeichnis befindet.

Das Blob-Image ist 75592 Byte groß. Dies entspricht einem Hexwert von `0x12748`. Der Bootloader muss an Speicheradresse `0x0000` kopiert werden. Als Endadresse ergibt sich somit `0x12747`. Die Imagedatei heißt `blob.S19`. Der Programmiervorgang des Blob-Images verlief zunächst normal. Auch die automatische Verifikation im Anschluss meldete keinerlei Fehler. Dennoch war das Image *nicht* fehlerfrei übertragen worden. Der Imote2-Sensorknoten ließ sich nicht booten. Ein manueller Klick auf den Verify-Button ergab folgende Meldung:

```
Verify Mismatch at: 0x8000 Expected: 18301BE5 Actual: FFFF0000
```

Mit Hilfe des Buttons Target Memory wird ein Einblick in die Speicherbelegung möglich. Von Adresse `0x0000` bis einschließlich `0x7FFF` entsprechen die Bytes exakt dem Inhalt der Imagedatei des Bootloaders blob. Ab Adresse `0x8000` befinden sich jedoch ausschließlich Nullen und `FFFF`s im Speicher:

```
0000 FFFF  FFFF 0000  FFFF 0000 ...
```

[11] Flashprogrammer: http://www.macraigor.com/flash_prog.htm

Listing 3.1: Konfigurationsdatei des Flashprogrammers `imote2.ocd`

```
[SETUP]
CpuVendor=Intel
CpuChip=PXA27x
CpuEndian=LITTLE
FlashVendor=Intel
FlashChip=28F256L18-B
RamAddress=0xA0000000
FlashAddress=0x00000000
FlashWidth=16
FlashChipsPerSector=1
LittleEndian=0
XmonAddress=0x01F00000
SimCount=0
MemoryCount=0
TLBCount=0
ScanChainCount=0
XScaleUseRAM=0
AfterResetJTAGSpeed=8
```

In der Konfigurationsdatei für die Imote2-Plattform `imote2.ocd` (siehe Listing 3.1) steht, dass die Bezeichnung des Flash-Chips `28F256L18-B` ist. Ein Vergleich mit dem Datenblatt des Prozessors[1] bestätigte, dass diese Angabe korrekt ist. In Listing 3.2 auf der nächsten Seite ist daher der betreffende Abschnitt `28F256L18-B` dargestellt, der demnach den Speicheraufbau des Imote2-Flashs parametrisiert.

Die hexadezimale Zahl `0x8000`, die in der *Verify-Mismatch*-Meldung erwähnt wurde, entspricht dezimal 32768 (=32 kByte). Dies ist die Sektorgröße, die in der `flash.ini`-Datei angegeben wurde. Es wurde also nur der erste Sektor korrekt beschrieben. Die Wiederholung des Programmiervorgangs ergab reproduzierbare Ergebnisse, und das selbst dann, wenn der Vorgang mit einem anderen Imote2-Sensorknoten durchgeführt wurde. Das Bootloader-Image konnte auf diese Weise also nicht vollständig übertragen werden.

Macraigor bietet keinen Support für die Flashprogrammer-Software an und stellt auch keine Updates oder Fehlerkorrekturen für die `flash.ini`-Datei zur Verfügung. In dieser Datei liegt jedoch eindeutig ein Fehler vor. Auf Anregungen aus der Imote2-Community wurde bereits der `CommandType` von `WIRELESS` auf `INTEL:STRATAFLASH` geändert. Diese Bezeichnungen verweisen auf die entsprechenden Abschnitte in der `flash.ini`-Datei, in dem die JTAG-Kommandos aufgelistet sind.

Der Vergleich der beiden Abschnitte ergibt, dass sich diese beiden Kommandotypen teilweise erheblich unterscheiden. Es konnte allerdings auch durch diese Maßnahme keine wesentliche Verbesserung erzielt werden. Möglicherweise sind hier weitere Korrekturen vorzunehmen. Der genaue Aufbau der Datei sowie die Bedeutung der einzelnen JTAG-Kommandos ist jedoch nicht dokumentiert. Daher ist nicht genau bekannt, welche Kommandos wie geändert werden müssten.

Listing 3.2: Auszug aus der Datei flash.ini

```
[28F256L18-B]
ManufacturersCode=$89
DeviceID=$8810
ChipSize=32768
ChipWidth1=16
ChipWidthCount=1
FailMask=$30
SectorSize1=32768
SectorSize4=32768
SectorSize5=131072
SectorSize259=131072
SectorCount=259
CommandType=INTEL:STRATAFLASH
```

Aufgrund dieser Probleme wurde auf den weiteren Einsatz dieser Software verzichtet. Stattdessen wurde die Open-Source-Software Jflashmm verwendet. Diese Software wird im Abschnitt 3.2.3 auf der nächsten Seite vorgestellt.

3.2.2. Das S19-Format

Das S19-Format wurde bereits in den 1970er Jahren von Motorola entwickelt. Damit werden Binärdaten in ein ASCII-Format kodiert. Jedes binäre Byte wird dabei durch seine hexadezimale Repräsentation dargestellt. Eine Datei im S19-Format ist zeilenweise aufgebaut. Jede Zeile ist gleich lang, beginnt mit einem `S` und endet mit einer Prüfsumme. Dadurch können Übertragungsfehler beim Einlesen der Datei festgestellt werden.

Die ersten drei Ziffern einer jeden Zeile sind der Header. Sie geben Auskunft über den Datensatztyp sowie über die Anzahl der Datenbytes innerhalb dieser Zeile. Anschließend folgt zunächst die hexadezimale Speicheradresse und dann die Datenbytes, die an diese Adresse geschrieben werden sollen.

Listing 3.3: Anfang des blob-Images im S19-Format

```
S315 0000 0000  3f00 00ea  6f00 00ea  7000 00ea  7100 00ea 37
S315 0000 0010  7200 00ea  7300 00ea  7400 00ea  7500 00ea 86
S315 0000 0020  0004 30a0  601b 0000  4827 0100  017a a0e3 dd
S315 0000 0030  0764 a0e1  1c50 1fe5  0500 a0e1  1100 00eb 0e
S315 0000 0040  0100 30e3  0e00 000a  0750 85e0  0760 56e0 c5
```

Das Listing 3.3 zeigt als Beispiel den Anfang der „blob“-Imagedatei. Die Datei enthält normalerweise keine Leerzeichen, diese wurden lediglich aus Gründen der besseren Lesbarkeit eingefügt. Der Zeilenbeginn `S315` besagt, dass es sich bei dieser Zeile um einen „S3-Record“ handelt und dass bis zum Zeilenende 0x15 (=21) weitere Hexbytes (inkl. Adresse und Prüfsumme) folgen. Ein S3-Record wurde von Motorola als eine Datenreihe mit einer 32 Bit-Speicheradresse definiert.[12] Die Prüfsumme

[12] S-Record Types, z.B: http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf Seite 642

am Zeilenende ergibt sich aus der Summe aller Daten- und Adressbytes dieser Zeile. Dabei werden allerdings nur die 8 niederwertigen Bits der Summe berücksichtigt.

3.2.3. Jflashmm

Jflashmm ist ein Programm, das auf der Kommandozeilenebene, also in einer DOS-Eingabeaufforderung ausgeführt wird. Die Syntax für den Programmaufruf ist in Listing 3.4 dargestellt. Es werden der Reihe nach Angaben zur Target-Plattform und zur Imagedatei gefordert. Der dritte Parameter entscheidet über die Aktion, die durchgeführt werden soll, also ob programmiert, gelöscht oder verifiziert werden soll. Danach wird die Ziel-Adresse innerhalb des Flash-Speichers angegeben. Danach folgen ggf. noch Angaben zum verwendeten JTAG-Adapter und ob der Debug-Mode gewünscht ist.

Listing 3.4: Parameter des Jflashmm-Programmaufrufs

```
>JFLASHMM [*PLAT] [*IMAGE] [P, V, E] [ADDR] [INS, PAR] [NOD, DEB]

* = required parameter
Optional parameters assume the first item in the list.

Where:

[PLAT]      =The name of the platform data file without the .DAT.
[IMAGE]     =The name of the image to program in flash.
[P, V, E]   =(P)rogram and verify, (V)erify only, (E)rase device
[ADDR]      =Hex byte address to start. No leading 0X, assumes 0
[INS, PAR, WIG]  =Insight IJC cable, passive parallel, or Wiggler
   type cable
[NOD, DEB]  =NODebug (Normal) or DEBug mode
```

Bevor Jflashmm jedoch eingesetzt werden kann, muss sichergestellt werden, dass die Druckerschnittstelle (LPT1) im BIOS in den Modus *ECP* versetzt wurde. Weiterhin muss ein sogenannter *GiveIO*-Treiber installiert werden, wobei eine Datei namens `giveio.sys` in das Windows-Systemverzeichnis `C:\WINDOWS\SYSTEM32\DRIVER` kopiert und installiert werden muss. Hierfür ist wegen der nötigen Rechte die Unterstützung des Systemadministrators erforderlich. Danach muss der PC neu gestartet werden. Im Windows-Gerätemanager erscheint ein neuer Anschluss, zum Beispiel „giveio (COM5)“.

Das Imote2-Debug-Board und der Diygadget-Adapter müssen beide jeweils mit einer USB-Schnittstelle des PCs verbunden werden. Dadurch wir die Spannungsversorgung sichergestellt. Es sollte nach Möglichkeit ein aktiver USB-HUB verwendet werden, damit die Schnittstellen nicht zu sehr belastet werden. Dies könnte zu Schäden an der Hardware des PCs führen.

Zum Starten des Programmiervorgangs wird folgende Zeile eingegeben, ohne jedoch schon auf Enter zu drücken: `Jflashmm.exe bulbcx16 blob P 0 WIG`

Danach muss mit zeitlich sehr geringem Abstand zueinander zunächst der Imote2 eingeschaltet und dann Enter gedrückt werden. Hier ist etwas Übung erforderlich, um das richtige Timing zu finden. Falls es nicht klappt (wie in Listing 3.5 auf der nächsten Seite), muss die Spannungsversorgung des Imote2-Boards vollständig unterbrochen werden. Dazu ist es erforderlich, alle USB-Kabel zu entfernen

oder den aktiven HUB auszuschalten. Der Taster neben der USB-Buchse des Imote2 kann nicht zum Ausschalten verwendet werden. Die Ausgabe eines erfolgreichen Vorgangs ist in Listing 3.6 auf der nächsten Seite dargestellt.

Listing 3.5: Jflashmm-Ausgabe mit Fehler

```
JFLASH Version 5.01.003
COPYRIGHT (C) 2000 - 2003 Intel Corporation

PLATFORM SELECTION:
  Processor=              PXA27x
  Development System=     Mainstone
  Data Version=           1.00.001

error, failed to read device ID
check cables and power
ACT: 1111 1111111111111111 11111111111 1
EXP: **** 1001001001100101 00000001001 1

failed to read device ID for this Platform
```

3.3. Bootloader „blob“

In der Einleitung wurde bereits erwähnt, dass jedes Betriebssystem auf dem Imote2 einen eigenen Bootloader benötigt. Einen universellen Bootloader, der alle Betriebssysteme unterstützt, gibt es derzeit nicht. Auf einem neu gekauften Imote2-Sensorknoten ist üblicherweise der .NET-Bootloader vorinstalliert. Daher kann dieser Sensorknoten ausschließlich in C# unter Verwendung des Microframeworks programmiert werden. Um auch die Programmiersprachen C, C++ und Java nutzen zu können, muss Linux als Betriebssystem verwendet werden. Zu diesem Zweck ist es erforderlich, den Bootloader auszutauschen. Der Bootloader für Linux heißt „blob“ (Boot Loader OBject).

Mit blob ist es möglich, vor dem eigentlichen Systemstart interaktiv Befehle einzugeben. Dafür wird in Zusammenhang mit dem Interface-Board ein Terminalprogramm benutzt. Wenn hingegen innerhalb von 2 Sekunden nach dem Einschalten keine Taste gedrückt wird, setzt der Bootloader den normalen Bootvorgang fort und lädt das Kernel-Image.

Der interaktive Modus bietet die Möglichkeit, binäre Imagedateien in den Flash-Speicher zu schreiben, ohne dass ein JTAG-Adapter verwendet werden muss. Die serielle Übertragung erfolgt mit einer Geschwindigkeit von 115200 Bits pro Sekunde unter Verwendung des Xmodem-Protokolls. Es werden dabei Blockgrößen bis zu 1 KByte unterstützt. Das Xmodem-Protokoll bietet eine automatische Fehlerkorrektur, wodurch eine fehlerfreie Datenübertragung sichergestellt wird. Dies ist ein großer Vorteil gegenüber der in Abschnitt 3.1 auf Seite 12 beschriebenen Vorgehensweise mit dem JTAG-Adapter, weil dort erst im Anschluss an die Übertragung eine Verifikation durchgeführt wird.

Die Übertragung wird mit dem `xdownload`-Kommando gestartet. Dadurch wird die Imagedatei vom PC mit dem Xmodem-Protokoll des Terminalprogramms in den Arbeitsspeicher der Imote2-Plattform übertragen. Anschließend kann mit dem Kommando `fwrite` das Image vom Arbeitsspeicher in den

Listing 3.6: Jflashmm-Ausgabe eines erfolgreichen Flash-Vorgangs

```
JFLASH Version 5.01.003
COPYRIGHT (C) 2000 - 2003 Intel Corporation

PLATFORM SELECTION:
  Processor=              PXA27x
  Development System=     Mainstone
  Data Version=           1.00.001

PXA27x revision C5
Found flash type: 28F256L18B

Unlocking block at address 0
Erasing block at address 0
Unlocking block at address 8000
Erasing block at address 8000
Unlocking block at address 10000
Erasing block at address 10000
Starting programming
Using BUFFER programming mode...
Writing flash at hex address    123c0, 98.80% done
Programming done
Starting Verify
Verifying flash at hex address    126d0, 99.84% done
Verification successful!
```

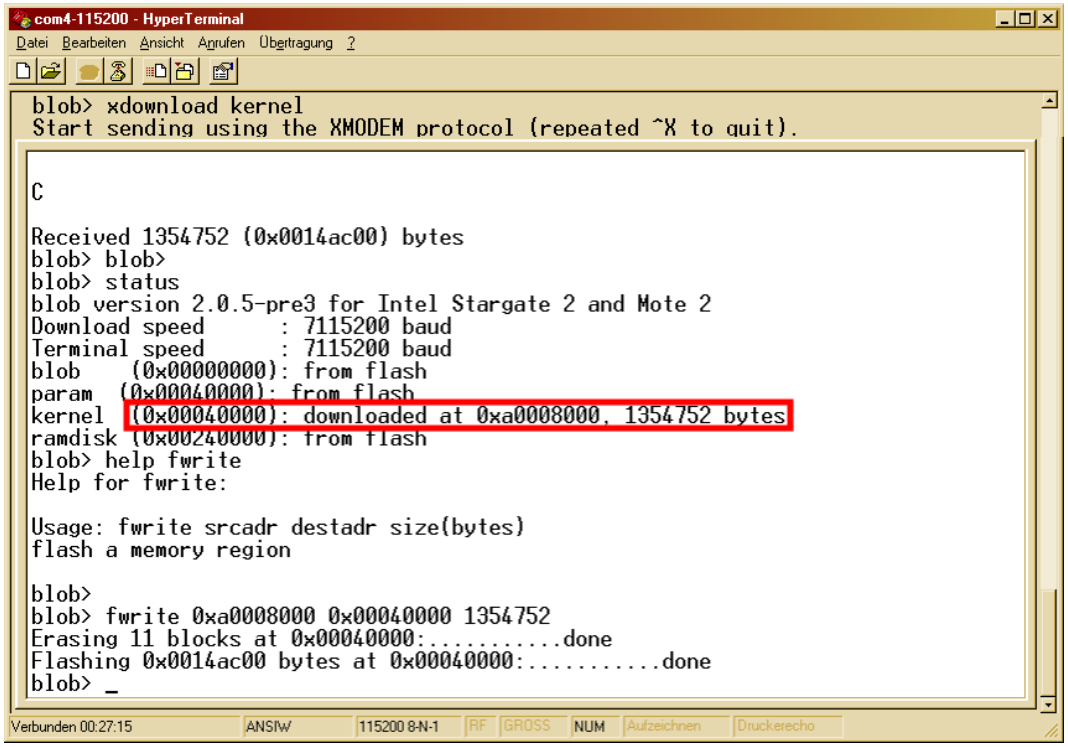

Abbildung 3.7.: Interaktive Bedienung von blob mit Hyperterminal

Flash-Speicher kopiert werden. Dem `fwrite`-Kommando müssen drei Parameter mitgegeben werden: Die Zieladresse im Flash-Speicher, die Quelladresse im Arbeitsspeicher sowie die Größe des Images in Bytes. Diese Werte lassen sich über das `status` Kommando abfragen. Die Kommandos zum Schreiben eines Kernelimages sind in Abbildung 3.7 dargestellt.

Es ist zu beachten, dass ab Kernelversion 2.6.29[13] eine modifizierte Version von blob erforderlich ist. Die jeweils aktuelle Version ist auf der Imote2-Linux Webseite[14] zu finden. Es hat zahlreiche Änderungen insbesondere in der Treiberorganisation für das Sensorboard gegeben. Wenn bereits eine ältere Blobversion installiert ist, lässt sich die neue Version per Xmodem übertragen. Die erneute Übertragung per JTAG ist dann also nicht erforderlich.

[13] Ankündigung Kernel 2.6.29 in Imote2-Community, Message 2292 vom 6.4.2009
[14] Imote2-Linux: http://sourceforge.net/projects/imote2-linux/

4. Linux-Anwendungen mit Imote2

In diesem Kapitel wird die Vorgehensweise beschrieben, wie sich der Imote2-Sensorknoten mit dem Linux-Betriebssystem anwenden lässt. Zunächst ist es erforderlich, einen Cross-Compiler auf dem PC zu installieren, mit dem der Quellcode der eigenen Programme in die Maschinensprache des Xscale PXA27x-Prozessors übersetzt wird. Der Speicherplatz auf der Imote2-Plattform reicht nicht aus, um den Compile-Vorgang direkt auf der Plattform durchzuführen. Die Installation des Crosscompilers und die Einrichtung der Programmierumgebung auf den Crosscompiler werden in dem nachfolgenden Abschnitt 4.1 beschrieben.

In Abschnitt 4.2 auf der nächsten Seite wird erklärt, wie eine TCP/IP-Verbindung mit dem Imote2-Sensorknoten aufgebaut werden kann. Eine solche Verbindung hat den Vorteil, dass mit weit verbreiteten Übertragungsprotokollen nach dem Internetstandard Daten übertragen werden können. Die mit dem Crosscompiler erstellten Programmdateien lassen sich beispielsweise mit dem *Secure-Copy-Protokoll* SCP vom PC auf die Imote2-Plattform übertragen. Das Herstellen einer Verbindung mit SSH und SCP wird in Abschnitt 4.2.1 erklärt.

Statt der Verwendung eines Passworts für den Login per SSH-Protokoll existiert die Alternative, ein kryptografisches *Keyfile* zu verwenden. Dies hat den Vorteil, das automatisierte Upload-Vorgänge ohne die Eingabe eines Passworts ausgeführt werden können. Diese Vorgehensweise wird in Abschnitt 4.2.2 auf Seite 27 beschrieben.

Ein weiteres Unterkapitel, das mit TCP/IP zu tun hat, ist Abschnitt 4.2.3 auf Seite 28. Das dort vorgestellte *Xwindow*-Protokoll wird zwar nicht direkt von Imote2 verwendet, weil es (derzeit) keine Programme mit grafischer Oberfläche für die imote2-Plattform gibt. Die Einführung ist dennoch sinnvoll, weil sich die Verwendung der erforderlichen virtuellen Maschine mit der dort beschriebenen Vorgehensweise deutlich vereinfacht. X-Anwendungen lassen sich über einen SSH-Tunnel direkt auf dem Windows-Desktop darstellen. Das Problem des doppelten Mauspfeils unter VMware wird somit umgangen.

Die Imote2-Plattform besitzt eine Echtzeituhr. Die Uhrzeit wird allerdings bei Unterbrechung der Spannungsversorgung zurückgesetzt. Bei jedem erneuten Einschalten muss die *Uhrzeit* also von Hand neu eingegeben werden. In Abschnitt 4.2.4 auf Seite 28 wird ein automatisierter Ansatz beschrieben. Der Imote2 kontaktiert einen Webserver und fragt ihn nach der Uhrzeit. Die dort beschriebene Vorgehensweise führt zugleich in die Funktionsweise von CGI ein. Das Verständnis der grundlegenden Funktionalität von CGI wird in späteren Kapiteln von Bedeutung sein. Dieser Abschnitt ist daher etwas ausführlicher.

Abschnitt 4.3 auf Seite 32 enthält einen weiteren Schwerpunkt dieses Kapitels. Dort wird die drahtlose Übertragung von TOSMAC-Nachrichten beschrieben. Auf diese Weise können Nachrichten zwischen verschiedenen Sensorknoten ausgetauscht werden. Zunächst wird auf das Übertragungsprotokoll allgemein eingegangen, anschließend folgt eine genaue Beschreibung der Vorgehensweise unter Linux. Quelltextauszüge von Beispiel-Programmen und dem Treiber runden das Kapitel ab.

4.1. Cross-Compiler

Auf dem Imote2-Linux-Sensorknoten lassen sich selbst erstellte Programme ausführen. Die Programmierung kann in *C* oder *C++* erfolgen. Aufgrund des begrenzten Speicherplatzes ist es jedoch nicht möglich, den Übersetzungsvorgang auf dem Knoten selbst durchzuführen. Daher ist ein sogenannter Cross-Compiler erforderlich, der entweder auf dem Windows-System des Arbeitsplatzrechners oder auf Linux-Ebene innerhalb der virtuellen Maschine ausgeführt wird.

Eine detaillierte Beschreibung zur Installation der Tools unter Cygwin[1] befindet sich im Netz.[2] Zur Installation unter Linux existieren ebenfalls mehrere ausführliche Beschreibungen.[3] Für reine *C*-Programme genügt es, diesen Anweisungen genau zu folgen. Für in *C++* geschriebene Programme ist jedoch noch eine zusätzliche Bibliothek `libstdc++.so.6` erforderlich, die aus dem Crosstoolverzeichnis (z.B. `/opt/crosstool/gcc-4.1.2-glibc-2.3.3/arm-xscale-linux-gnu/lib`) in das Verzeichnis `/usr/lib` auf dem Sensorknoten kopiert werden muss. Diese Datei ist jedoch mit ihren 4 MB relativ groß, wenn man bedenkt, dass insgesamt lediglich 32 MB auf dem Sensorknoten zur Verfügung stehen. Daher ist zu überlegen, ob für das vorgesehene Projekt wirklich die Objektorientierung von *C++* benötigt wird, oder ob eine Realisierung auch in *C* möglich ist.

Als Programmierumgebung habe ich Eclipse mit dem CDT-Plugin[4] verwendet. Dieses Plugin passt Eclipse an die speziellen Anforderungen der *C*-Programmierung an. Zusammen mit den oben erwähnten Crosstools lassen sich auf diese Weise Programme für die Imote2-Plattform erstellen. Dafür muss über das Menü Project/Properties bei C/C++ Build eine neue Konfiguration für Imote2 angelegt werden. Am einfachsten ist es, eine bestehende Konfiguration zu kopieren. Anschließend müssen unter Settings die Kommandos für die Tool-Chain angepasst werden, so dass statt des normalen Compilers der Cross-Compiler verwendet wird. Den voreingestellten Kommandos muss dazu jeweils `arm-xscale-gnu-` vorangestellt werden, so dass aus `g++` beispielsweise `arm-xscale-gnu-g++` wird. Das Dialogfenster zum Einstellen dieser Optionen ist in Abbildung 4.1 auf der nächsten Seite dargestellt.

Damit das fertige Ergebnis sogleich auf den Imote2-Knoten übertragen wird, kann ein sogenannter post-build step definiert werden. In Abbildung 4.2 auf der nächsten Seite ist das dafür nötige `scp`-Kommando dargestellt. Bei der Verwendung dieses Kommandos ist zu beachten, dass keine interaktive Eingabe des Passworts möglich ist. In Abschnitt 4.2.1 auf Seite 27 ist daher eine Möglichkeit beschrieben, wie auf die Eingabe des Passworts verzichtet werden kann, in dem stattdessen eine Schlüsseldatei verwendet wird.

4.2. TCP/IP mit Imote2

Die Imote2-Plattform wird durch das Betriebssystem Linux in die Lage versetzt, über das TCP/IP-Protokoll mit anderen Netzwerkteilnehmern zu kommunizieren. Die TCP/IP-Verbindung wird über die USB-Schnittstelle realisiert. Sobald die USB-Verbindung hergestellt ist, sorgt der USB-Treiber dafür, dass die virtuelle Netzwerkkarte `usb0` aktiv wird.

In der Datei `/etc/network/interfaces` lässt sich das Netzwerk konfigurieren. Diese Datei existiert auf jeder gewöhnlichen Linux-Plattform. Auf dem Imote2 ist sie ebenfalls vorhanden. In dieser Datei

[1] Cygwin: http://www.cygwin.com/
[2] Tools on Cygwin: http://sourceforge.net/apps/trac/imote2-linux/wiki/ToolsOnCygwin
[3] Imote2, Linux: http://www.eecs.harvard.edu/~konrad/projects/imote2Camera/IMote2-Installation-Instructions.html
[4] Eclipse-CDT: http://www.eclipse.org/cdt

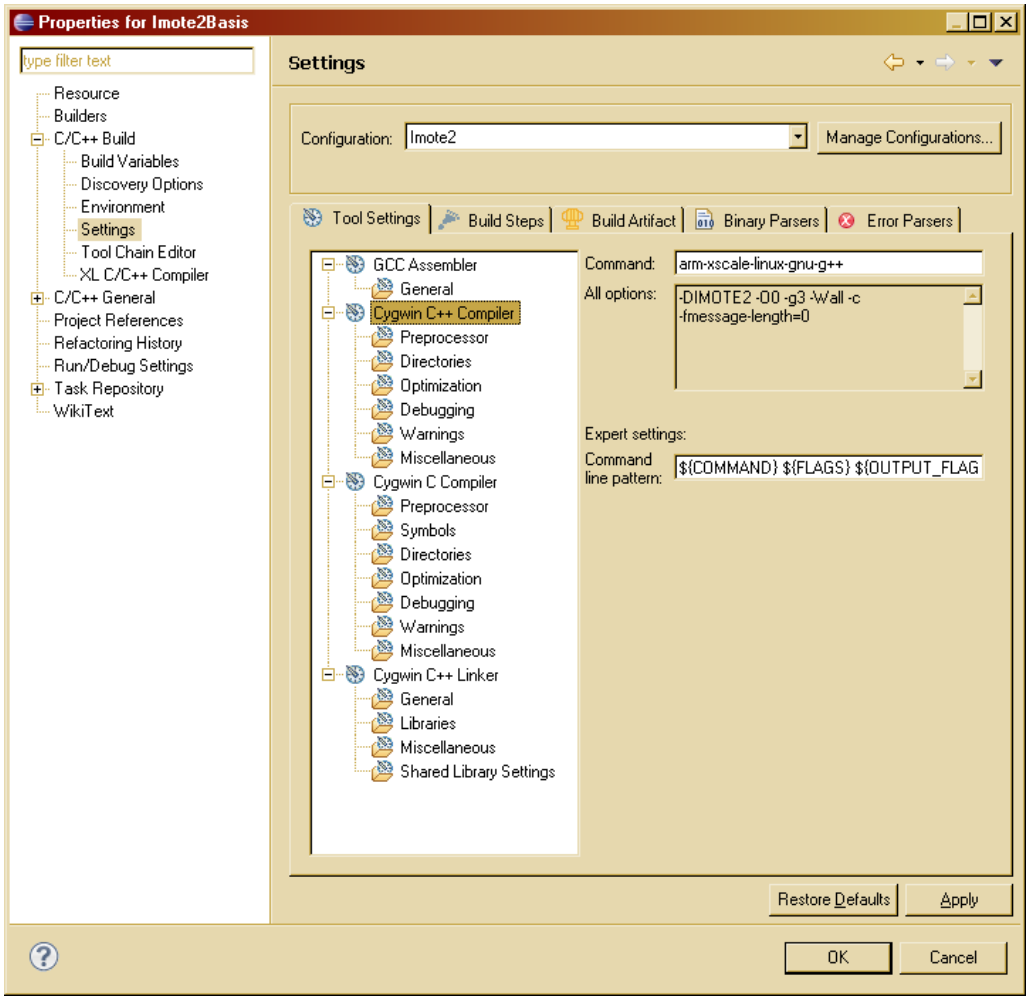

Abbildung 4.1.: Projekteinstellungen von Eclipse CDT

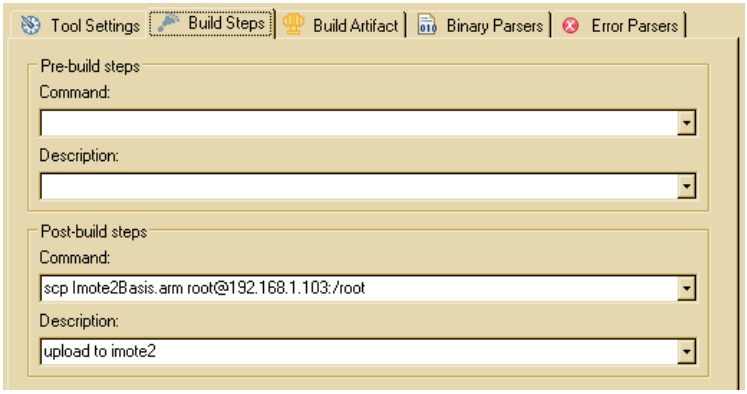

Abbildung 4.2.: Post-Build-Step-Einstellung für automatische Übertragung zum Target

wird die IP-Adresse der Plattform festgelegt. Diese IP-Adresse darf sich für Imote2 und PC nur in der letzten der vier Zahlengruppen unterscheiden. Ansonsten ist eine Kommunikation nicht möglich. Eine `interfaces`-Datei für Imote2 ist in Listing 4.1 dargestellt.

Listing 4.1: Konfiguration des Netzwerks unter Linux: `/etc/network/interfaces`

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto usb0
iface usb0 inet static
address 192.168.1.103
netmask 255.255.255.0
gateway 192.168.1.98
```

Die virtuelle Maschine auf dem PC besitzt eine ähnliche Datei, die eventuell weitere Einträge besitzt. Jede Netzwerkkarte besitzt einen eigenen Eintrag. In dem gewählten Beispiel hat der Sensorknoten die IP-Adresse `192.168.1.103` und die virtuelle Maschine hat die IP-Adresse `192.168.1.98`. Die Imote2-Plattform ist nun in der Lage, mit der virtuellen Maschine Informatinoen auszutauschen. Testen lässt sich die Verbindung mit dem `ping` Kommando. Als Parameter wird die jeweils „gegnerische“ IP-Adresse angegeben.

Die Kommunikation zwischen der Imote2-Plattform und anderen Teilnehmern außerhalb der virtuellen Maschine ist erst dann möglich, wenn weitere Maßnahmen getroffen wurden. Zunächst muss die virtuelle Maschine als Gateway angegeben werden. In dem Listing 4.1 ist das bereits durch die `gateway`-Zeile erfolgt. Der Imote2-Sensorknoten sendet dann alle TCP/IP-Pakete das Gateway. Auch diejenigen Pakete, die für einen anderen Adressaten im Netz gedacht sind. Die Aufgabe der Virtuellen Maschine ist es dann, die Pakete weiterzuleiten. Dazu muss in der virtuellen Maschine zum einen das *IP-Forwarding* aktiviert sein, zum anderen muss die *Firewall* die Weiterleitung zulassen. Mit den folgenden Anweisungen werden beide Voraussetzungen erfüllt:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A FORWARD -j ACCEPT
```

Die Windows-Ebene des PCs liegt in einem anderen Netzwerkbereich als die virtuelle Maschine. Der Adressraum 192.168.1.x ist für den Windows-Rechner nicht bekannt. Daher kann der Sensorknoten trotz IP-Forwarding von Windows nicht erreicht werden. Um den Adressraum bekannt zu machen, muss unter Windows eine Netzwerk-Route gesetzt werden. Das geht von einer Eingabeaufforderung (`cmd`) mit dem Kommando: `route add 192.168.1.0 mask 255.255.255.0 192.168.110.128`

In diesem Beispiel ist die IP-Adresse der virtuellen Maschine 192.168.110.128. Welche Adresse die virtuelle Maschine tatsächlich hat, lässt sich mit dem `ifconfig`-Kommando herausfinden. Unter Windows gibt es ein ähnliches Kommando. Dort heißt es `ipconfig`.

4.2.1. SSH und SCP

Auf dem Linux-System der Imote2-Plattform ist ein SSH-Dämon installiert. Dadurch ist es möglich, eine SSH-Verbindung aufzubauen, und sich mit dem Kommando `ssh` auf dem Sensorknoten einzuloggen. SSH steht für Secure Shell. Wenn die in Abschnitt 4.2 beschriebenen Vorkehrungen getroffen wurden, ist ein Login direkt von der Windows-Ebene möglich. Das Programm PuTTY[5] ist als SSH-Client besonders zu empfehlen.

Unter Linux oder Cygwin genügt die Eingabe des Kommandos `ssh` zuzüglich der IP-Adresse der Imote2-Plattform. Zu Beginn der SSH-Sitzung wird nach einem Benutzernamen und einem dazugehörigen Passwort gefragt. Für ein neu installiertes Imote2-Linux lautet der Username `root` und das Passwort ist leer (nur Enter drücken).

Um Dateien auf den Sensorknoten zu kopieren, wird das *Secure copy protocol* SCP verwendet. Unter Windows gibt es für diese Aufgabe das Programm WinSCP[6], das über eine grafische Oberfläche bedient wird. Das Programm lässt sich wie ein Explorer-Fenster benutzen. Dateien können mit den üblichen Methoden kopiert, verschoben und umbenannt werden.

Auf der Kommandozeilenebene kann ein Kopiervorgang durch das Kommando `scp` ausgelöst werden. Die Parameter dieses Kommandos lassen sich über den Parameter `--help` erfragen. Zum Kopieren einer Datei `helloworld.arm` auf die Targetplattform mit der IP-Adresse 192.168.1.103 als Benutzer `root` in dessen Homeverzeichnis `/root` ist zum Beispiel das folgende Kommando erforderlich:
`scp helloworld.arm root@192.168.1.103:/root`

Aus Sicherheitsgründen sollte der Root-Account stets mit einem Passwort abgesichert sein. Bei häufigen (oder automatisierten) Kopiervorgängen kann das jedoch als lästig empfunden werden. Als Alternative zur Authentifizierung per Passworteingabe kann eine Schlüsseldatei verwendet werden. Diese Vorgehensweise wird im nachfolgenden Abschnitt erklärt.

4.2.2. SSH mit Keyfile

Der SSH-Login kann auch ohne Passwort erfolgen. Das einfache Entfernen des Passworts ist jedoch nicht zu empfehlen. Stattdessen sollte ein Schlüsselpaar verwendet werden, dass aus einem öffentlichen und einem privaten Schlüssel besteht. Es handelt sich dabei um ein asymmetrisches Verschlüsselungsverfahren.

Das Schlüsselpaar wird auf dem Host erzeugt, von dem der Login erfolgen soll. Mit Host ist hier entweder die Cygwin-Umgebung auf der Windows-Ebene oder das Linux-System der virtuellen Maschine gemeint. Die Erzeugung wird mit dem Kommando `ssh-keygen` initiiert:
`ssh-keygen -t rsa`

Das Schlüsselpaar wird im Home-Unterverzeichnis `.ssh` abgelegt. Der private Schlüssel `id_rsa` muss dort verbleiben, der öffentliche Schlüssel `id_rsa.pub` hingegen muss auf den Server, bzw. in diesem Fall auf den Imote2 kopiert werden. Dort gibt es, ebenfalls im Home-Unterverzeichnis `.ssh`, eine Datei mit dem Namen `authorized_keys`. An diese Datei muss der öffentliche Schlüssel angehängt werden.
`cat ~/.ssh/id_rsa.pub | ssh root@192.168.1.103 'cat >> .ssh/authorized_keys'`

[5] Putty: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[6] WinSCP: http://winscp.net/eng/download.php

Hierbei wird ein letztes Mal das Passwort abgefragt. Bei allen zukünftigen Login-Vorgängen genügt die Existenz der privaten Schlüsseldatei auf dem Host.

4.2.3. Xwindows

Grafische Darstellung unter Linux wird mit Hilfe des sogenannten X-Window-Protokolls realisiert. Dafür wird ein Dienst gestartet, der sich X-Server nennt und die Bildschirmoberfläche verwaltet. Dieser Server kümmert sich dabei um die Treiber von Grafikkarte, Bildschirm, Maus und Tastatur. Als Clients hingegen werden die Programme bezeichnet, die etwas auf der Bildfläche darstellen möchten. Die folgenden Ausführungen beziehen sich auf das Linux-System der virtuellen Maschine. Das Imote2-Linux-System enthält keine X-Anwendungen.

Der X-Server ist bei Linux-Systemen meistens vorinstalliert. In der Regel wird er auf dem selben System ausgeführt wie seine Clients. Es ist aber auch eine logische oder räumliche Trennung von Server und Client möglich. So kann die grafische Ausgabe von Linux-Programmen beispielsweise auch auf den Windows-Desktop umgeleitet werden, sofern dort ein X-Server vorhanden ist. Diese Art der Umleitung nennt sich X-Forwarding.

Die Kommunikation zwischen Client und Server lässt sich über das SSH-Protokoll tunneln. Dafür muss dem ssh-Kommando der Parameter `-Y` mitgegeben werden, z.B. `ssh -Y root@192.168.\-110.128`. Wenn *PuTTY* als SSH-Client verwendet wird, kann im Einstellungsfenster bei Connection/SSH/X11 die Option X-Forwarding aktiviert werden. Aus der Shell können dann grafische Anwendungen, wie zum Beispiel `xterm` gestartet werden. Diese werden dann remote ausgeführt, aber lokal dargestellt.

Für die lokale Darstellung ist die Existenz eines X-Servers erforderlich. Unter Windows gibt es mehrere Programme, die einen X-Server zur Verfügung stellen. Wenn *Cygwin* eingesetzt wird, ist es am einfachsten, das Xwindow-Paket gleich mitzuinstallieren. Aber auch ohne Cygwin muss auf X-Windows nicht verzichtet werden. Ein frei verfügbarer Standalone-Xserver ist *Xming*.[7] Bestandteil von Xming ist der Xlaunch-Assistent. Dieses Tool stellt automatisch einen SSH-Tunnel mit dem Zielrechner her und führt die gewünschte Anwendung aus. Dies kann entweder ein einfaches `xterm`-Terminalfenster sein oder sogar die komplette Desktop-Umgebung von KDE oder Gnome. Für Gnome lautet das Kommando `gnome-session`.

4.2.4. Uhrzeit setzen mit CGI

Die Imote2-Sensorknoten sind mit einer Echtzeituhr ausgestattet. Diese funktioniert jedoch nur während der Laufzeit des Knotens. Sobald der Knoten einmal ausgeschaltet wurde, setzt sich die Systemuhr beim nächsten Start auf den 1. Januar 1970 zurück. Unter Linux kann die Systemuhr mit dem Kommando `date MMDDhhmm[[CC]YY][.ss]` eingestellt werden. Dabei steht `MMDD` für das Datum, `hhmm` für die Uhrzeit, `CCYY` für das Jahr und `ss` für die Sekunden. Die Jahresangabe kann zwei- oder vierstellig erfolgen, die Sekunden können weggelassen werden. Um beispielsweise das Datum auf den 28. Oktober 2009 und die Uhrzeit auf 12:32 Uhr einzustellen, kann das folgende Kommando verwendet werden: `date 102812322009`

Mit dem `date`-Kommando kann das Datum nicht nur gesetzt, sondern auch abgefragt werden. Für die Darstellung in einem bestimmten Format kann mit dem Plus-Parameter eine Vorgabe gemacht

[7] Xming: http://sourceforge.net/projects/xming/

werden. Dabei gibt es Platzhalter, die mit einem %-Zeichen eingeleitet werden, z.B. `%m` für den Monat, `%d` für den Tag usw.. Die aktuelle Uhrzeit kann also mit `date +%m%d%H%M%Y` in genau dem gewünschten Format angezeigt werden.

Dieser Vorgang lässt sich automatisieren. Dazu habe ich den folgenden Ansatz gewählt, der in Abbildung 4.3 dargestellt ist: Der Imote2-Sensorknoten kontaktiert nach dem Einschalten automatisch einen bestimmten Webserver. Dieser sendet die aktuelle Uhrzeit im oben angegebenen Format an den Imote2 zurück. Der Imote2 empfängt die Antwort und gibt sie an das System weiter.

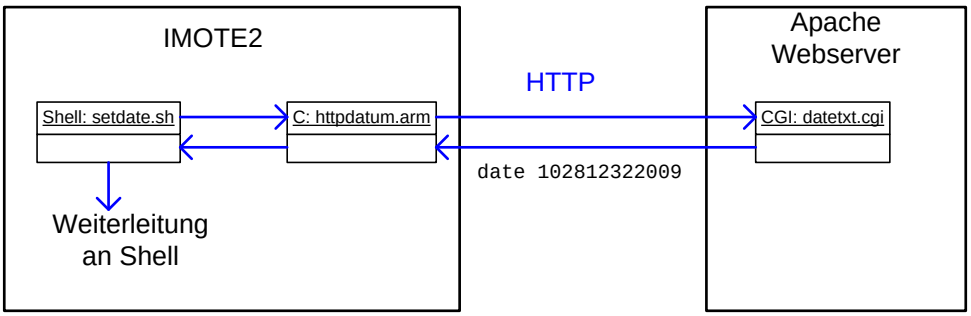

Abbildung 4.3.: Setzen des Systemdatums per HTTP und CGI-Skript

Es bietet sich an, die virtuelle Linux-Maschine auch als Webserver einzusetzen. Oft ist bereits ein Apache-Webserver vorinstalliert. Ansonsten kann er mit `apt-get` oder `aptitude` hinzugefügt werden. In der Standard-Konfiguration von Apache wird ein Verzeichnis `/usr/lib/cgi-bin` erzeugt. In diesem Verzeichnis abgelegte Dateien können als CGI-Skript (Common Gateway Interface) über das HTTP-Protokoll aufgerufen werden.

Eine Datei `test.cgi` lässt sich dann z.B. mit http://192.168.1.98/cgi-bin/test.cgi aufrufen. Die Datei muss ausführbar sein. Vor der eigentlichen Ausgabe muss eine Header-Zeile Auskunft über das Datenformat des eigentlichen Inhalts geben. Dadurch wird dem Webbrowser mitgeteilt, wie der Inhalt darzustellen ist. Zum Beispiel bedeutet `Content-type: text/plain`, dass es sich bei der Ausgabe um reinen Text handelt. Wenn diese Angabe fehlt, generiert der Webserver eine Fehlerseite: `Error 500 - Internal Server Error`. Das Listung 4.2 enthält ein Shell-Skript, das die gewünschte Aufgabe erfüllt.

Listing 4.2: CGI-Skript `datetxt.cgi` zum Senden des Systemdatums

```
#!/bin/bash
echo -e "Content-Type: text/plain\n"
date +'date %m%d%H%M%Y'
```

Die Kontaktierung des Webservers wird von einem C-Programm übernommen. Das Listing 4.3 zeigt eine mögliche Implementierung. Durch die eingebundenen Bibliotheken `socketline.h` und `tcp-query.h` werden Funktionen zum Öffnen, Lesen und Schreiben des Sockets bereitgestellt. Diese Funktionen wurden nicht selbst entwickelt, sondern es wurden Routinen von Paul Griffiths verwendet, die er auf seiner Internet-Seite bereitstellt.[8]

[8] TCP Sockets: http://www.paulgriffiths.net/program/c/sockets.php

Listing 4.3: C-Programm `httpdatum.c` zum Senden einer HTTP-Anfrage

```
#include <stdio.h>       // puts
#include <string.h>      // strlen
#include "tcpquery.h"    // int tcpquery(char *szAddress,int port)
#include "socketline.h" // Readline, Writeline(socketnr,buf,buflen)

#define LEN     80

int main()
{
        int socketnr = tcpquery("192.168.1.98", 80);
        char * query = "GET␣/cgi-bin/datetxt.cgi␣HTTP/1.1\n"\
                       "Host:␣192.168.1.98\n"\
                       "Connection:␣close\n\n";
        Writeline(socketnr, query, strlen(query));
        char buf[LEN+1];
        while (Readline(socketnr, buf, LEN) >2)
                ;       /* skip header */
        int rc=Readline(socketnr, buf, LEN);
        /* rc=number of received chars. skip line if too short */
        if(rc>0 && rc <18)
                rc=Readline(socketnr, buf, LEN);
        puts(buf);
        return 0;
}
```

Listing 4.4: Shellskript `setdate.sh` zum Setzen des Systemdatums

```
#!/bin/bash
TEMP=/tmp/datumtmp
/usr/bin/httpdatum.arm > $TEMP
. $TEMP
rm $TEMP
```

Das Programm öffnet zunächst einen Socket mit der IP-Adresse der virtuellen Maschine. Aus Sicht des Imote2-Sensorknotens ist das `192.168.1.98`. Zusammen mit der IP-Adresse wird eine Portnummer angegeben. Für das HTTP-Protokoll ist die Standardportnummer 80. In der nächsten Zeile wird die Query-Anfrage im üblichen HTTP-GET-Format formuliert. Nachdem das Query gesendet wurde, wird die erhaltene Antwort ausgewertet. Der HTTP-Header wird dabei vom Programm ignoriert. Der Content, also die generierte Ausgabe aus dem CGI-Skript, wird auf dem Bildschirm ausgegeben. Als Beispiel könnte eine von Listing 4.2 erzeugte Ausgabe so aussehen:

```
date 102812322009
```

Das ist genau das gewünschte Format. Ein weiteres Shellskript sorgt dafür, dass diese Ausgabe an das System weitergegeben und ausgeführt wird. Dazu leitet es diese Ausgabe zunächst in eine temporäre Datei `$TEMP` um. Anschließend wird diese Datei über die Shell aufgerufen (`. $TEMP`). Dieses Shellskript kann in das Verzeichnis `/etc/init.d` kopiert werden, damit es beim Start automatisch ausgeführt wird. Anschließend muss dann noch im Verzeichnis `/etc/rc2.d` ein symbolischer Link erzeugt werden:

```
ln -s ../init.d/setdate.sh S20setdate
```

Die Zahl 20 ist dabei willkürlich gewählt. Sie muss jedoch auf jeden Fall größer sein als die Zahl vor dem Networking-Skript (z.B. `S10networking`), weil die Netzwerkverbindung zum Setzen des Datums benötigt wird und weil die Skripte in der Reihenfolge des Dateinamens ausgeführt werden. Um dem Linux-Standard zu entsprechen, müsste das Skript noch etwas erweitert werden. Darauf soll aber an dieser Stelle verzichtet werden, da es hier nur auf die Kernfunktionalität ankommt.

Damit der vorgestellte Ansatz funktionieren kann, sind einige Voraussetzungen zu erfüllen:

- Auf der virtuellen Maschine muss ein Apache-Webserver installiert und erreichbar sein.
- Das CGI-Skript muss vorhanden und ausführbar sein.
- Die URL des Hosts muss bekannt (und konstant) sein.
- Die Netzwerkverbindung muss beim Booten des Imote2 hergestellt sein.

Insbesondere die zuletzt genannte Bedingung ist unter Umständen nicht erfüllt. Die Netzwerkverbindung wird nämlich erst dann hergestellt, wenn die USB-Verbindung mit der virtuellen Maschine existiert. Dafür muss aber im VMware Player das Compaq-USB-Device zugewiesen sein, was direkt nach dem Einschalten des Imote2 nicht immer der Fall ist.

Der vorgestellte Ansatz beinhaltet keinerlei Fehlerkorrektur und birgt bei Manipulation des CGI-Skripts ggf. Sicherheitsrisiken, da die Ausgabe ungeprüft an die Shell weitergeleitet wird. Vor einer Nachahmung auf anderen Linux-Systemen sei daher ausdrücklich gewarnt.

In einer zukünftigen Implementierung könnte das *Network-Time-Protocol* (NTP) verwendet werden. Damit ließe sich der Sensorknoten mit einem öffentlichen Zeitserver synchronisieren. Ein eigener Webserver mit einem CGI-Skript wäre in dem Fall nicht mehr erforderlich.

4.3. Drahtlose Übertragung mit dem TOSMAC-Protokoll

Ein wesentliches Merkmal eines drahtlosen Sensornetzwerks ist neben der Sensorfunktionalität und der Energieversorgung durch Batterien die drahtlose Übertragung von Nachrichten. Die Imote2-Plattform tauscht mit anderen Imote2-Sensorknoten Nachrichten nach dem IEEE 802.15.4 Standard aus. Es wird dafür einer von 16 möglichen Kanälen aus dem lizenzfreien 2,4 GHz-ISM-Band verwendet.

Die Imote2-Plattform ist für manche Aufgaben deutlich überdimensioniert. Hier könnten preiswertere Sensorknoten eingesetzt werden, die das Sensornetzwerk an Stellen ergänzen, wo hohe Rechenleistung und großer Speicherbedarf von untergeordneter Bedeutung sind. Lediglich die anspruchsvolleren Aufgaben wie die automatische Bewertung der Temperaturdaten und die darauf basierende Entscheidungsfindung erfolgt dann auf dem Imote2-Knoten.

Für dieses heterogene Sensornetzwerk ist es erforderlich, dass für den Datenaustausch ein gemeinsames Protokoll verwendet wird. Es bietet sich an, das weit verbreitete TOSMAC-Protokoll zu verwenden, das aus dem Betriebssystem TinyOS hervorging und sich inzwischen als Quasi-Standard zur drahtlosen Übertragung von Sensordaten etabliert hat.

4.3.1. Drahtlose Kommunikation zwischen TelosB und Imote2

Ein weiterer Sensorknoten, der von der Firma Crossbow verkauft wird, heißt TelosB. Diese Plattform ist baugleich zu dem Sensorknoten TmoteSky, der ursprünglich von MoteIV verkauft wurde. Beide Bezeichnungen werden heutzutage daher synonym verwendet.

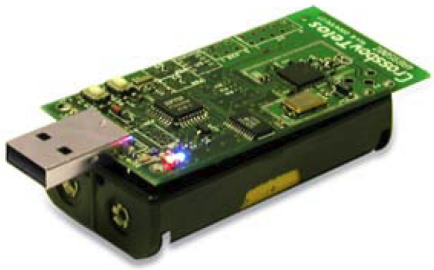

Abbildung 4.4.: Sensorknoten TelosB

Die TelosB Plattform ist mit einem 8 MHz-MSP430-Mikroprozessor und 10 KB RAM ausgestattet. Zusätzlichen stehen 1 MB Flash-Speicher zur Verfügung. Die Hardware enthält eine IEEE 802.15.4-Antenne. Als Betriebssystem wird ausschließlich TinyOS eingesetzt. Die Programmierung erfolgt in der Sprache nesC.

Der Energiebedarf von TelosB ist aufgrund der minimalen Hardware-Ausstattung sehr gering. Die Lebensdauer der Batterien ist daher dementsprechend hoch. Diese Art Sensorknoten eignet sich für langfristige Messaufgaben. Ein TelosB-Knoten kostet derzeit rund 100 € und ist damit deutlich günstiger als ein Imote2-Knoten.

Der TelosB-Knoten verwendet zum Austausch von drahtlosen Datenpaketen den gleichen Frequenzbereich wie die Imote2-Knoten. Dennoch konnten keine Informationen zwischen beiden Plattformen

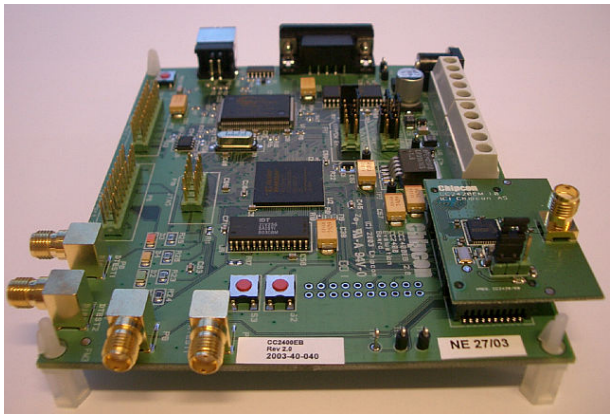

Abbildung 4.5.: Snifferboard CC2420DK von Texas Instruments

auf diese Weise ausgetauscht werden. Aufschluss über dieses Phänomen gab der Einsatz eines sogenannten Snifferboards. Es wurde das Board *CC2420DK* von Texas Instruments verwendet, das in Abbildung 4.5 dargestellt ist. Es handelt sich dabei um eine Dauerleihgabe des ITEM (Ole Bischoff). Dieses Board wird über ein USB-Kabel mit dem PC verbunden. Weitere Anschlüsse sind nicht erforderlich. Es empfiehlt sich jedoch, einen aktiven USB-Hub zu verwenden, um die Schnittstelle des PC nicht zu sehr zu belasten. Bei zu starker Belastung kann die USB-Schnittstelle beschädigt werden.

In der mitgelieferten Software wird der gewünschte, zu beobachtende Kanal innerhalb des 2,4 GHz-Bandes eingestellt. Anschließend werden alle empfangenen Datenpakete protokolliert. Das Ausgabefenster ist in Abbildung 4.6 auf der nächsten Seite dargestellt. Es ist zu erkennen, dass die Datenpakete von Imote2 und TmoteSky eine unterschiedliche Struktur haben. Das Datenfeld „Source Address“ wird von TmoteSky zusätzlich mitgesendet und bei empfangenen Paketen auch erwartet, während Imote2 ohne dieses Feld auskommt, und Pakete mit einem solchen Feld wiederum ignoriert.

Ein weiterer Unterschied ergibt sich im sogenannten *Frame Control Field*. Bei TmoteSky ist das Flag `Intra-PAN` gesetzt, bei Imote2 nicht. Auch dieses Flag könnte die Ursache für die Übertragungsprobleme sein. Ein gezieltes Setzen oder Entfernen ist jedoch in den Versuchen nicht gelungen. Das Datenfeld `Dest PAN` steht vermutlich in direktem Zusammenhang mit diesem Flag, da es sich ebenfalls unterscheidet.

Es wurden Versuche unternommen, die Struktur der Pakete zu modifzieren. Beim .NET Microframework scheiterte es daran, dass die zuständigen Bibliotheken ausschließlich im Binärformat vorliegen und kein Zugriff auf die Quellen besteht. Änderungen sind somit ausgeschlossen. Auf Linux-Ebene wären Modifikationen eventuell auf einer sehr niedrigen Kerneltreiber-Ebene möglich. Auf TinyOS-Ebene des TelosB Knotens war ein Modifikationsversuch bisher ebenfalls nicht erfolgreich.

Diskussionen in der Imote2-Community[9] ergaben hierzu keine neuen Erkenntnisse. Es wurde allerdings vereinzelt berichtet, dass eine erfolgreiche Kommunikation zwischen TelosB und Imote2 hergestellt werden konnte, wenn auf beiden Sensorknoten *TinyOS* als Betriebssystem eingesetzt wird. Dies sollte ggf. vor Ort verifiziert werden. Ein Vergleich der Source-Codes könnte weitere Erkenntnisse bringen.

[9]Message-Nr. 3021, 3029, 3030, 3032, 3033, 3034, 3035, 3046, 3049, 3051

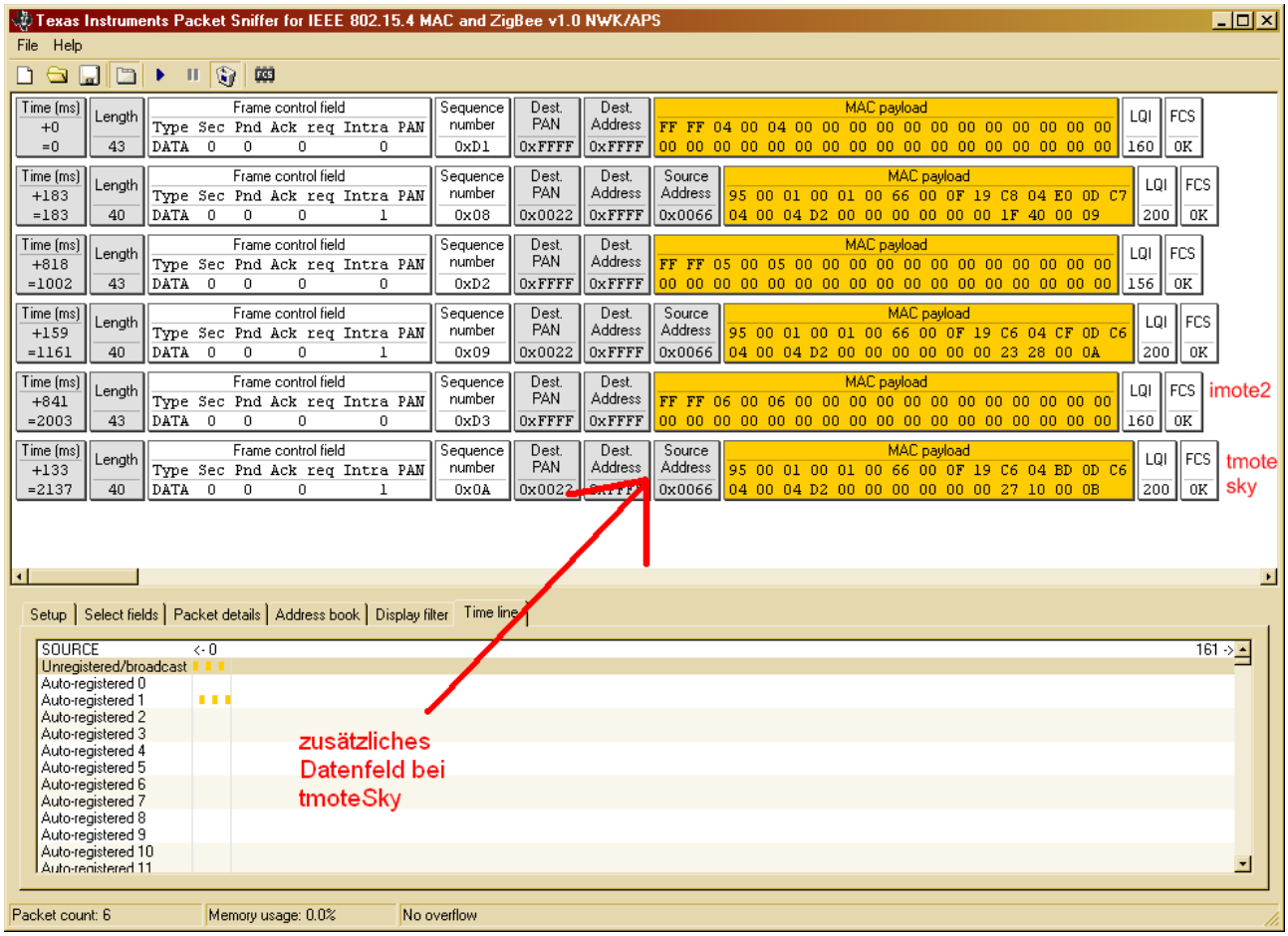

Abbildung 4.6.: Packet-Sniffer-Software für IEEE 802.15.4

Es gibt noch eine weitere Möglichkeit, mit der erreicht werden kann, dass beide Sensorknotentypen miteinander kommunizieren. Es kann ein Pärchen bestehend aus Imote2 und TelosB im Huckepackverfahren hardwaremäßig miteinander verbunden werden. Der Datenaustausch geschieht dann entweder über die serielle Schnittstelle oder über die USB-Host-Funktion des Imote2. Ein so realisiertes Gateway hat Zugriff auf beide Teilnetzwerke, wodurch der Nachrichtenaustausch möglich wird. Es stellt sich allerdings die Frage, ob dieser Aufwand gerechtfertigt ist, oder ob es nicht doch eine Software-Lösung gibt.

4.3.2. TOSMAC-Nachrichten mit Linux empfangen

Seit der Kernelversion 2.6.29 ist das Tosmac-Modul fester Bestandteil des Imote2-Linux-Systems. Der Modultreiber ist allerdings nicht besonders sauber programmiert. Der Aufbau des Treibers entspricht nicht dem Linux-Standard. Die Benutzung ist daher etwas umständlich. Das wurde schon oft innerhalb der Imote2-Community kritisiert. Man plant daher, in Zukunft diesen Treiber noch weiter zu überarbeiten.[10]

[10]Community-Beiträge z.B. 2632, 3101,...

TOSMAC-Nachrichten lassen sich durch Zugriff auf die Treiberdatei `/dev/tosmac` versenden und empfangen. Die erforderlichen Schritte zur erstmaligen Einrichtung des Modultreibers sind in [8] dokumentiert. Ein Beispielprogramm in *C* zum Empfangen eines Tosmac-Paketes auf Kanal 15 ist in Listing 4.5 dargestellt. Der Kanal wird über ein festgelegtes `ioctl()`-Kommando eingestellt.

Listing 4.5: C-Programm `empfang1.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include "tosmac.h"
#include "hilfsfunktionen.h"

int main (int argc, char **argv){
  int tos, res;
  char buf[128];
  int chn = 15; // Kanal (zwischen 11 und 26)

  tos = open ("/dev/tosmac", O_RDWR); // Treiber oeffnen
  if (tos < 0) {
      puts ("Fehler beim Oeffnen");
      return -1;
    }
  res = ioctl (tos, TOSMAC_IOSETCHAN, chn); // Kanal setzen
  res = read (tos, buf, sizeof (buf));      // Daten empfangen

  printf("%d Bytes received\n",res);
  printheader (buf);            // aus hilfsfunktionen.h
  printcontent (buf, res);      // aus hilfsfunktionen.h

  return 0;
}
```

Der eigentliche Lesevorgang wird von der `read()`-Funktion durchgeführt. Das gesamte empfangene TOSMAC-Paket wird inklusive Header in dem `char`-Array `buf[]` abgespeichert. Um die Grenzen dieses Arrays nicht zu überschreiten, wird die maximale Puffergröße mit dem `sizeof(buf)` Kommando übergeben. Die Anzahl der empfangenen Bytes wird an die Ergebnisvariable `res` zurückgegeben.

Eine Fehlerbehandlung findet zugunsten der Übersichtlichkeit in diesem Beispiel nicht statt. Das Programm wartet so lange, bis ein Paket empfangen wird. Dies wird als Blocking-Mode bezeichnet. Der Treiber lässt sich auch in den Non-Blocking Mode versetzen. Dazu muss der `open()`-Funktion die Option `O_NONBLOCK` hinzugefügt werden. Vor dem eigentlichen Lesevorgang lässt sich dann mit der `select()`-Funktion feststellen, ob Daten bereitstehen. Nur wenn das Ergebnis positiv ist, braucht der Lesevorgang ausgeführt werden. In diesem Beispiel wurde jedoch wegen der einfacheren Implementierung auf diese Abfrage verzichtet.

Listing 4.6: Ausgabe des C-Programms `empfang1.c`

```
40 Bytes received

HEADER:
 .hex=[ 1c  41  08  ce  ff  ff  ff  ff  32  7d  20  01 ]
 .num=[ 28, 65,  8,206,255,255,255,255, 50,125, 32,  1,]
 .str="__^ __A __^ __^ __^ __^ __^ __^ __2 __} __  __^ "
index:   0   1   2   3   4   5   6   7   8   9  10  11

CONTENT:
 .hex=[ 3b  01  b7  04  00  00  00  00  00  00  87  01  46  19  40 ]
 .num=[ 59,  1,183,  4,  0,  0,  0,  0,  0,  0,135,  1, 70, 25, 64 ]
 .str="__; __^ __^ __^ __^ __^ __^ __^ __^ __^ __^ __^ __F __^ __@ "
index:  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
```

Der empfangene Puffer wird nacheinander an die beiden Funktionen `printheader()` und `printcontent()` weitergegeben. Diese Funktionen sind in der separaten Datei `hilfsfunktionen.c` definiert, die per Präprozessor-Direktive `#include` eingebunden wird. Eine von dem Programm erzeugte Ausgabe ist in Listing 4.6 dargestellt. Aus Platzgründen wurde die Ausgabe nach dem 27. Byte abgeschnitten. Es folgen noch weitere Content-Bytes. Die Paketgröße ist insgesamt 40 Byte. Das dargestellte empfangene Tosmac-Paket wurde von einem Imote2.NET-Knoten mit dem Programm `XSensorITS400` erzeugt.

4.3.3. TOSMAC-Nachrichten mit Linux senden

Das Senden eines Tosmac-Paketes ist unter Linux ebenfalls möglich. In der Tosmac-Headerdatei `tosmac.h` ist die Struktur des zu sendenden Tosmac-Pakets definiert. Die vollständige Headerdatei ist im Anhang auf Seite 112 zu finden. Die Kommentare stammen von den Entwicklern des Treibers.

Zum Senden einer einfachen Tosmac-Nachricht genügt es, die Struktur `Tos_Msg` zu instanziieren, die Datenfelder zu füllen und die `length`-Variable entsprechend anzupassen. Der Treiber muss schreibend geöffnet und der Channel entsprechend gesetzt sein. Dann kann mit der `write()`-Funktion das Datenpaket gesendet werden.

In Listing 4.7 auf der nächsten Seite befindet sich ein Beispielprogramm, dass aus der aktuellen Systemzeit ein Tosmac-Paket generiert. Die in in den Listings 4.5 und 4.7 verwendeten Hilfsfunktionen zum Anzeigen eines Tosmac-Pakets sind im Anhang auf Seite 74 in Listing A.1 zu finden.

Listing 4.7: Nachricht über Tosmac senden `senden1.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <time.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include "tosmac.h"
#include "hilfsfunktionen.h"
#define IOCTL_GETVALUE 0x0001

int main (int argc, char **argv) {
  int drv, res; // Driver, result
  int chn=15;   // Kanal im ISM-Band (10..26)
  int th,tm,ts;
  char buf[128];
  TOS_Msg send_pkt;

  drv = open (TOSMAC_DEVICE, O_RDWR);   // Treiber oeffnen read+write
  if (drv < 0) {
      printf ("Error␣opening␣driver␣%s␣(%d)\n", TOSMAC_DEVICE,drv);
      return -1;
  }
  res = ioctl (drv, TOSMAC_IOSETCHAN, chn);     // Set Channel

  send_pkt.addr = 0xA1B2;       // Adresse des Paket-Absenders
  send_pkt.length = 3;

  // Daten aus aktueller Uhrzeit erzeugen
  ts=time(NULL)%86400;
  th=ts/3600;
  ts%=3600;
  tm=ts/60;
  ts%=60;
  send_pkt.data[0]= th;
  send_pkt.data[1]= tm;
  send_pkt.data[2]= ts;

  printf ("about␣to␣write:\n");
  printsendinfo (send_pkt);     // aus hilfsfunktionen.h
  write (drv, (TOS_Msg *) & send_pkt, send_pkt.length);
  printf ("\nwritten\n");

  return 0;
}
```

5. Qualitätsorientierte Autonome Routenentscheidung

Bei der qualitätsorientierten autonomen Routenentscheidung geht es darum, dass ein Transportfahrzeug selbstständig Abweichungen von optimalen Transportbedingungen feststellt und darauf reagiert. Eine mögliche Reaktion ist die Entscheidung für eine andere Route. Mit Route ist die Reihenfolge der Städte gemeint, die das Fahrzeug anfährt, um die Pakete abzuliefern. Durch diese Entscheidung soll erreicht werden, dass trotz einer aufgetretenen Störung ein möglichst großer Teil der gefährdeten Ware rechtzeitig am jeweiligen Zielort abgeliefert werden kann.

Dieses Konzept wurde bereits in einer Veröffentlichung von Reiner Jedermann[10] vorgestellt. Die Machbarkeit zeigte er mit Hilfe einer Software-Simulation. Basierend auf den original Java-Quellen dieser Simulation (SplitPlanner) habe ich dieses Konzept weiterentwickelt[2] und als Software auf Sensorknoten-Ebene implementiert.

Als Programmiersprache wurde C++ gewählt, weil dadurch eine systemnahe aber dennoch objektorientierte Realisierung möglich ist. Der Wechsel der Programmiersprache machte es erforderlich, die Software komplett neu zu schreiben. Einige Funktionen sind zwar stark an die Originalquellen angelehnt, der grundsätzliche Aufbau unterscheidet sich jedoch erheblich.

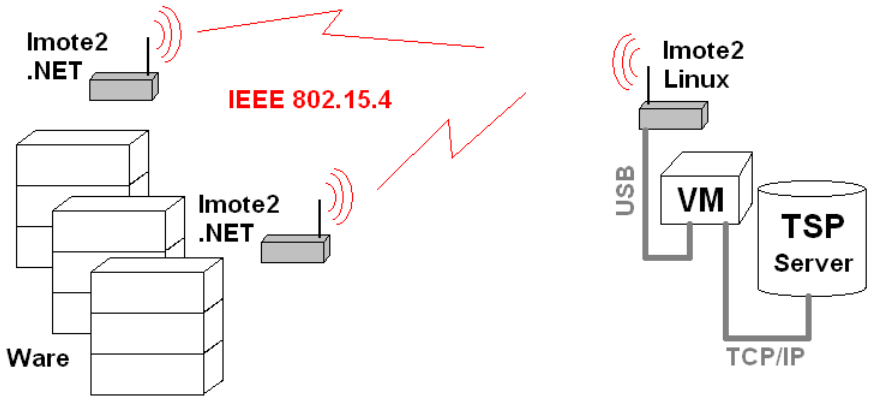

Abbildung 5.1.: Konzept für die qualitätsorientierte Routenplanung

Die Software besteht aus mehreren Ebenen, die auf räumlich voneinander getrennten Hardware-Plattformen ausgeführt werden. Die Hardwareplattformen sind Imote2 und ein Server. Die Imote2-Sensorknoten werden für zwei separate Aufgaben verwendet. Zum einen für die Messaufgabe, zum anderen für die Bewertung und Entscheidung. In Abbildung 5.1 ist das Zusammenspiel der einzelnen Einheiten dargestellt.

Für die Messaufgabe der Umweltparameter in der Umgebung der Ware kommen mehrere batteriebetriebene Imote2-Sensorknoten mit dem .NET-Microframework zum Einsatz. Diese sind drahtlos über die IEEE 802.15.4-Schnittstelle mit einem zentralen Imote2-Sensorknoten verbunden. Die Datenpakete werden mit dem TOSMAC-Protokoll übertragen. Der zentrale Sensorknoten ist mit einem embedded Linuxsystem ausgestattet. Über USB ist er mit einem PC verbunden, auf dem in einer virtuellen Maschine (VM) ebenfalls ein Linux-Betriebssystem läuft. Dadurch ist ein Austausch mittels TCP/IP-Protokoll möglich.

Auch jeder beliebige andere Rechner im Netz kann auf diese Weise kontaktiert werden. Die dritte Einheit, der TSP-Server, kann also räumlich getrennt sein. TSP steht dabei für *Travelling Salesman Problem*. Der Server empfängt Anfragen vom Sensorknoten und schlägt dann eine mögliche Lösung vor, in welcher Reihenfolge mehrere angegebene Städte am besten zu besuchen sind. Der Vorschlag wird als Antwort per TCP/IP zurückgesendet. Für die gewählte Realisierung wurde der TSP-Server auf der gleichen Hardware ausgeführt, auf der auch die virtuelle Maschine läuft.

In den nachfolgenden Abschnitten werden die einzelnen Einheiten detailliert vorgestellt. Zunächst werden die Imote2.NET-Knoten in Warennähe beschrieben. Anschließend folgt eine Beschreibung des TSP-Servers und danach die Beschreibung des zentralen Imote2.Linux-Knotens. Diese Reihenfolge wurde gewählt, weil der Linux-Knoten durch seine Gateway-Funktion auf Zulieferung von den beiden anderen Elemente angewiesen ist. Bei der Beschreibung wird auf die Besonderheiten der einzelnen Ebenen eingegangen und es werden die Ideen erläutert, die zu dieser Art der Implementierung geführt haben. Ein Demonstrationslauf mit dem Zusammenspiel aller Softwarekomponenten rundet das Kapitel ab.

5.1. Imote2.NET Sensorknoten in Waren-Nähe

Von den Imote2.NET Sensorknoten kann es mehrere geben, die im Laderaum verteilt sind. Sie sind dafür zuständig, die Umweltparameter zu überwachen. Die Sensorknoten verfügen über Sensoren für Temperatur, relative Feuchtigkeit, Licht und Beschleunigung sowie über drei analoge Eingänge, über die zusätzliche externe Sensoren kontaktiert werden können.

Ein Imote2-Sensorknoten besteht aus den drei zusammensteckbaren Boards Prozessorplattform, Batterieboard und Sensorboard. Auf der Unterseite des Prozessorboards, die im zusammengesteckten Zustand zum Batterieboard zeigt, befindet sich ein Etikett mit einem Barcode. In Abbildung 5.2 ist das Etikett links unten zu sehen. Die letzten drei Ziffern (also hier 316) sind die Sensor-NodeID. Diese wird später bei der Zuordnung der Sensordaten von Bedeutung sein.

Crossbow liefert mit dem *Imote2.NET Builder-Kit* zwei vollständige Sensorknoten, ein separates Prozessorboard sowie ein umfangreiches Software-Paket mit. Darin enthalten sind Beispiel-Anwendungen, die in C# geschrieben sind. Für den Einsatz in diesem Szenario wurde die Beispiel-Anwendung *XSensorITS400Sleep* gewählt und leicht modifiziert. Diese Anwendung fragt alle Sensorkomponenten des Sensorboards ITS400 ab. Aus den so erhaltenen Daten wird ein TOSMAC-Paket generiert, das über die IEEE 802.15.4 Schnittstelle übertragen wird. Anschließend schaltet sich der Sensorknoten in den sogenannten *Deep-Sleep-Mode*, wodurch der Energiebedarf deutlich sinkt. Nach einer definierten Zeitspanne in Sekunden (`_sleep_duration=8`) wacht der Sensorknoten wieder auf und sendet dann das nächste Paket.

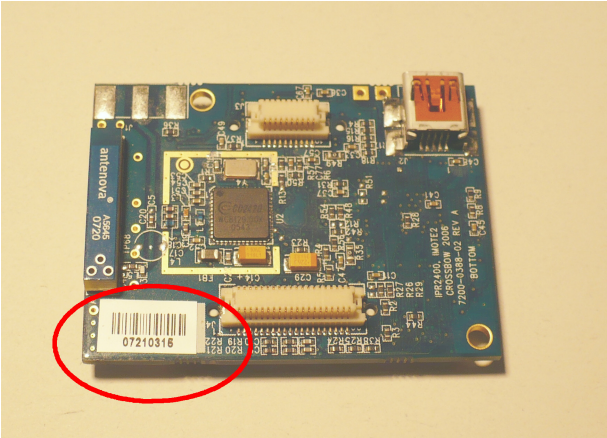

Abbildung 5.2.: Imote2 Prozessor-Board IPR2400 mit Seriennummer

Beim Einschalten sowie beim Aufwachen aus dem Deep-Sleep-Mode leuchtet beim Imote2 üblicherweise die grüne LED für 1-2 Sekunden auf und schaltet sich anschließend wieder aus. Das Programm wurde so modifiziert, dass zu Programmbeginn zusätzlich eine rote LED aufleuchtet, die beim Programmende wieder erlischt. Dieses Zeitverhalten ist in Abbildung 5.3 dargestellt. Die Laufzeit des Programms sowie die Zeitspanne zum Aufwachen lässt sich aus den zeitlichen Differenzen ermitteln.

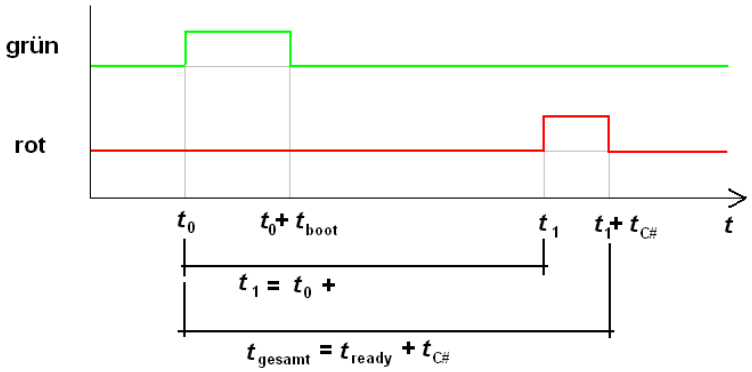

Abbildung 5.3.: Zeitdiagramm der LEDs des Programms ITS400Sleep

Es wurde eine Zeitdifferenz t_1 vom Aufleuchten der grünen bis zum Aufleuchten der roten LED von $t_1 = 7$s gemessen. Die Programmlaufzeit $t_{C\#}$ wird im wesentlichen durch das künstlich erzeugte Delay für das Warten auf die Debug-Ausgabe bestimmt. Sie kann als vernachlässigbar angesehen werden. Das Sende-Intervall zwischen zwei empfangenen Paketen erhöht sich aber um die Aufwachzeit. Bei einer Sleep-Duration von 8 Sekunden ergibt sich ein zeitlicher Abstand zweier aufeinanderfolgender Datenpakete von $7 + 8 = 15$ Sekunden.

Der Aufbau eines solchen Datenpakets ergibt sich aus Listung 5.1. Jedes Paket enthält 27 Byte Nutzdaten und einen Header, der automatisch mit den entsprechenden Angaben gefüllt wird. Bei den Nutzdaten handelt es sich um Rohdaten, die direkt von den Sensoren kommen. Um aus den Rohdaten die physikalischen Parameter zu bestimmen, sind die folgenden Umrechnungsformeln erforderlich:

Listing 5.1: Reihenfolge der Datenfelder im TOSMAC-Paket

```
///   byte          field
///    0             board_id
///    1             packet_id
///    2-3           node_id
///    4-5           count
///    6-7           accel_x
///    8-9           accel_y
///    10-11         accel_z
///    12-13         temperature; from TI sensor
///    14-15         temperature2 ; from sensirion sensor
///    16-17         humidity
///    18-19         light
///    20-21         adc0
///    22-23         adc1
///    24-25         adc2
///    26-27         adc3
```

$$T_{\text{sensirion}}[^{\circ}C] = 0{,}01 \cdot x - 39{,}6 \tag{5.1}$$

$$H_{\text{sensirion}}[\%] = (-2{,}8 \cdot 10^{-6}) \cdot x^2 + 0{,}04 \cdot x - 4 \tag{5.2}$$

$$T_{\text{TI}}[^{\circ}C] = \frac{(10 \cdot x + 8)}{160} \tag{5.3}$$

5.2. TSP-Server

In diesem Demonstrationsszenario wurde der Schwerpunkt auf die autonome Entscheidung des mobilen Systems gelegt. Der TSP-Server stellt das zentrale Gegenstück dar. Der Server ist im Gegensatz zu den Sensorknoten nicht in seinen Ressourcen beschränkt, so dass aufwendige Rechenoperationen auf einer umfangreichen Datenbasis durchgeführt werden könnten.

Das Lösen eines *Travelling Salesman Problems* ist NP-hart. Das bedeutet, dass der Rechenaufwand zum Finden der optimalen Lösung exponentiell mit der Anzahl der Städte ansteigt. Das Lösen eines solchen Problems ist jedoch *nicht* das eigentliche Ziel dieser Arbeit. Der hier eingesetzte TSP-Server wurde daher so einfach wie möglich realisiert. Es wurde ein heuristischer Ansatz gewählt, der zwar eine gute Lösung findet, die jedoch bei weitem nicht die optimale Lösung ist. Ein großer Vorteil dieses Ansatzes ist, dass die Lösung sehr schnell gefunden wird. Somit eignet sich dieses Verfahren für den hier vorgesehenen Online-Einsatz.

Um die Programmausführung etwas anschaulicher zu gestalten, werden reale Städtenamen und reale Städtekoordinaten verwendet. Der nachfolgende Abschnitt 5.2.1 beschäftigt sich mit der Entfernungsberechnung. Danach wird auf das C++-Konzept der STL-Container eingegangen. Daran schließt sich

eine Beschreibung der Modellierung der Städte innerhalb der Software an. Bevor dann auf die eigentliche TSP-Server-Software eingegangen wird, wird noch das Städte-Info-CGI vorgestellt. Dieses CGI ist im Zusammenhang mit dem TSP-Server ein hilfreiches Werkzeug. Es läuft auf dem gleichen Webserver und verwendet dieselbe Datenbasis. Dadurch lässt sich im Zweifelsfall das vom Programm errechnete Ergebnis besser nachvollziehen.

5.2.1. Entfernungsberechnung zweier Städte

Von 30 deutschen Städten wurden die geographischen Koordinaten ermittelt. In Tabelle 5.1 sind diese Städte aufgelistet. Die Koordinaten sind in eine laterale und longitudiale Komponente unterteilt. Die laterale Komponente φ wird in Grad nördlicher Breite angegeben. Die longitudinale Komponente λ wird in Grad östlicher Länge angegeben. Die Angabe ist jeweils unterteilt in Grad und Minuten.

Aus diesen Koordinaten lässt sich die Entfernung berechnen. Die Verbindung zweier Punkte auf einer Kugeloberfläche wird als Orthodrom bezeichnet. Es handelt sich dabei um die kürzest mögliche Entfernung entlang der Oberfläche. Die so erhaltene Verbindung wird auch als Luftlinie bezeichnet, weil der Verlauf der Straßenverbindungen zwischen diesen Städten ignoriert wird. Eventuelle Verkehrsstörungen, Baustellen, Staus usw. bleiben ebenfalls unberücksichtigt.

Stadt	lat (φ)	lon (λ)	Stadt	lat (φ)	lon (λ)
Hamburg	53N34	10E02	Bremen	53N05	8E49
Bremerhaven	53N33	8E35	Oldenburg	53N09	8E13
Osnabrueck	52N17	8E03	Braunschweig	52N16	10E32
Lueneburg	53N15	10E25	Celle	52N38	10E05
Hannover	52N23	9E44	Gifhorn	52N29	10E33
Verden	52N55	9E14	Aurich	53N28	7E29
Wolfsburg	52N25	10E47	Goslar	51N54	10E26
Goettingen	51N32	9E56	Muenster	51N58	7E38
Detmold	51N56	8E53	Dortmund	51N31	7E28
Duesseldorf	51N14	6E47	Aachen	50N47	6E05
Koeln	50N57	6E57	Essen	51N28	7E01
Wuppertal	51N16	7E12	Bonn	50N44	7E06
Paderborn	51N43	8E45	Bielefeld	52N01	8E32
Guetersloh	51N54	8E23	Neumuenster	54N04	9E59
Kiel	54N20	10E08	Luebeck	53N52	10E41

Tabelle 5.1.: Verwendete Städtekoordinaten für den TSP-Server

Zur Berechnung der Entfernung zweier Städte müssen die Koordinatenwerte zunächst nach Formel 5.4 in das Bogenmaß umgerechnet werden, weil die trigonometrischen Funktionen in C++ nicht mit Grad-Angaben rechnen können. Vor der Umwandlung muss der Minuten-Anteil mit dem Grad-Anteil verrechnet werden, so dass aus den beiden Ganzzahlen eine Fließkommazahl entsteht. Ein Grad besteht aus 60 Minuten. Der Minuten-Anteil muss also durch 60 geteilt werden und dann zu dem Grad-Anteil hinzuaddiert werden.

$$x_{\text{bogen}} = \pi \cdot \frac{x_{\text{grad}}}{180} \tag{5.4}$$

Die Distanz d zweier Städte A und B, mit den Koordinaten (λ_A, φ_A) und (λ_B, φ_B) errechnet sich dann nach Formel 5.5 für Orthodrome, die dem Bronstein[5] auf Seite 181 entnommen wurde. Die Konstante $r_{\text{eq}} = 6378.137$ steht für den Erdradius am Äquator in Kilometern.

$$d_{(A,B)} = \arccos\left(\sin\varphi_A \cdot \sin\varphi_B + \cos\varphi_A \cdot \cos\varphi_B \cdot \cos(\lambda_B - \lambda_A)\right) \cdot r_{\text{eq}} \tag{5.5}$$

Zur Verdeutlichung wird dieses Verfahren am Beispiel der beiden Städte Hamburg und Bremen durchgeführt. Aus Tabelle 5.1 entnimmt man die Koordinaten. Für Hamburg lautet der Eintrag `lat=53N34, lon=10E02`. Umgerechnet in Fließkommazahlen ergibt sich $\varphi = 53{,}5667°$ und $\lambda = 10{,}0333°$. Ins Bogenmaß übertragen ist $\varphi = 0{,}934915$ und $\lambda = 0{,}175115$. Für Bremen lautet der Eintrag für die Koordinaten `lat=53N05, lon=8E48`. Umgerechnet in Fließkommazahlen ergibt sich $\varphi = 53{,}0833°$ und $\lambda = 8{,}81667°$. Ins Bogenmaß übertragen ist $\varphi = 0{,}929479$ und $\lambda = 0{,}15388$.

In die Formel 5.5 werden diese Zahlenwerte nun eingesetzt, wobei A für Hamburg und B für Bremen gewählt wird. Es ergibt sich eine Entfernung von $d = 97{,}151320$km. Diese Berechnung lässt sich mit dem später in Abschnitt 5.2.5 vorgestellten Stadtinfo-CGI-Programm nachvollziehen. Es ergibt sich das gleiche Ergebnis für die Entfernung.

Der Aufruf für das CGI zur Berechnung der Entfernung zwischen Hamburg und Bremen wird durch Referenzierung der Städte erreicht. Hamburg hat Referenznummer 1 und Bremen hat Referenznummer 2. Die URL lautet: http://192.168.1.98/cgi-bin/stadt.cgi?action=distanz&stadt1=1&stadt2=2. Die IP-Adresse kann eventuell davon abweichen. Die Ausgabe ist in Abbildung 5.10 auf Seite 51 dargestellt. Erläuterungen zu den verwendeten Parametern befinden sich in Abschnitt 5.2.5, der auf Seite 48 beginnt.

5.2.2. STL-Container

Ein fester Bestandteil von ISO C++ ist die *Standard Template Library* (STL). Template bedeutet, dass ein beliebiger Datentyp verwendet werden kann. Diese Library unterteilt sich in die drei Gebiete Container, Iteratoren und Algorithmen. Container eignen sich für die Zusammenfassung verschiedener gleichartiger Objekte, Iteratoren dienen der Navigation innerhalb der Container, und Algorithmen führen Operationen mit Container-Elementen aus.

Das Konstrukt der Liste (`list`) wurde bei der Realisierung dieser Software häufig eingesetzt. Eine Liste hat gegenüber Arrays den Vorteil, dass die Anzahl der enthaltenen Objekte variable ist. Insbesondere kann die Anzahl dynamisch während der Laufzeit angepasst werden. Einfüge- und Sortier-Operationen sind bereits Bestandteil der Template-Bibliotheken, sie müssen somit nicht selbst implementiert werden.

Auf einzelne Objekte der Liste kann mit den Iteratoren zugegriffen werden. Die Listenfunktion `begin()` liefert einen Iterator auf das erste Element der Liste. Mit dem `++`-Operator lässt sich der Iterator inkrementieren. Wenn das Listenenende erreicht ist, hat der Iterator den Wert `end()`. Auf diese Weise lässt sich die gesamte Liste durchlaufen.

Dadurch, dass die Liste als Template realisiert ist, lassen sich Listen mit beliebigem Datentyp anlegen. Nachfolgend wird eine Städte-Liste eingeführt. In den späteren Abschnitten wird eine Liste mit Waren, Paketen und Sensorknoten angelegt. Oft wird statt der Objekte selbst lediglich eine Zeigerreferenz auf diese Objekte in der Liste abgespeichert. Dies hat den Vorteil, dass die Objekte nur einmal

im Speicher vorhanden sind, aber an mehreren Stellen im Programm gleichzeitig verwendet werden können.

In der nachfolgenden Stadtverwaltungsklasse wird ein anderer Ansatz gewählt. Die Liste enthält die Objekte der Städte. Wenn eine Stadt an anderer Stelle im Programm benötigt wird (z.B. als Zieladresse eines Pakets), wird ein Städte-Zeiger auf das Listenelement gesetzt. Da die Städteliste selbst sich nicht verändert, sind beide Varianten zueinander gleichwertig. In der Tsp-Klasse wird dieser Ansatz jedoch zu einem Nachteil, da dort gezielt eine andere Anordnung vorgenommen werden soll. Die Tsp-Klasse ist daher in der Ausführung nicht ganz so effizient implementiert. Es wird allerdings davon ausgegangen, dass das TSP-Problem von einem leistungsstarken externen Server bearbeitet wird. Daher fällt dieser Nachteil nicht so sehr ins Gewicht.

5.2.3. Stadt-Klasse

Die Modellierung der Städte wird in der Software durch drei Klassen realisiert, die miteinander vernetzt sind. In der Abbildung 5.4 sind die drei Klassen Stadtverwaltung, Stadt und koo dargestellt. Die als Stadtverwaltung bezeichnete Klasse verwaltet die Liste aller Städte und bietet entsprechende Funktionenen zur Abfrage von Städteinformationen. Die Stadt-Klasse beinhaltet alle Informationen, die *eine* Stadt charakterisieren. Dies sind der Name der Stadt, die Koordinaten sowie eine interne Referenznummer. Die dritte Klasse „koo“ ist eine Hilfsklasse zur einfacheren Handhabung und Umrechnung des Koordinatenpaares.

Die Stadtverwaltungs- und die Stadt-Klasse werden von den Programmen Stadtcgi, Tspserver und Imote2Basis verwendet. Diese Klassen sind daher elementar von Bedeutung. Es genügt, dass diese Klassen in dem Programm Stadtcgi als Quellcode vorliegen. Die anderen Programme können die vorkompilierten Objektdateien des Compilers verlinken, um auf die Funktionen zugreifen zu können. Die Klassen sind in sich abgeschlossen und können somit vielseitig verwendet werden.

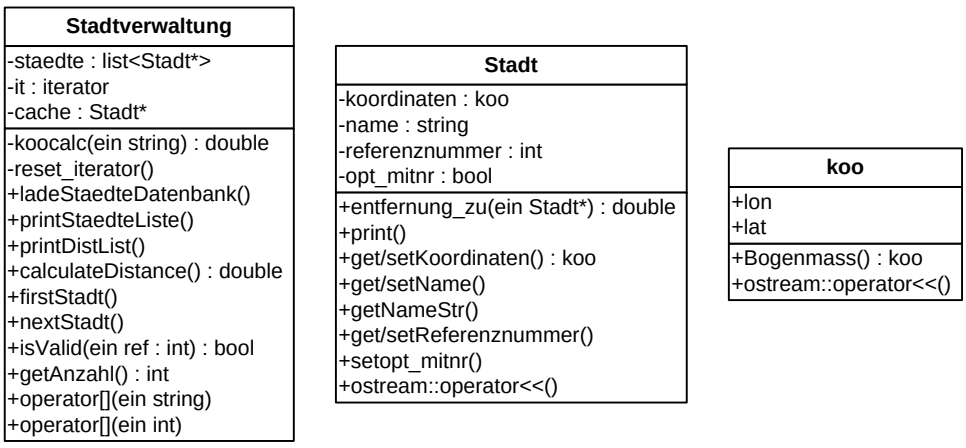

Abbildung 5.4.: Stadtverwaltung-, Stadt- und koo-Klasse

Bei der Instanziierung der Stadtverwaltung wird im Konstruktor der Klasse automatisch die Funktion `ladeStaedteDatenbank()` aufgerufen. In der Headerdatei `Stadtverwaltung.h` ist die Präprozessor-Variable `STADTVERWALTUNG_FILENAME` mit `#define` definiert, die den Namen der Datei festlegt, in der sich die Städte-Informationen befinden. Im Anhang in Listing A.25 auf Seite 116 ist die komplette Datei abgedruckt.

Die Datei wird zeilenweise eingelesen. Die in dieser Datei enthaltenen Städtenamen und Koordinaten sind durch ein Whitespacezeichen voneinander getrennt. Die Koordinatenangabe wird in dem Format „Grad-Buchstabe-Minute“ erwartet. Als Buchstabe sind dabei E und N zulässig. Beispiele für solche Angaben befinden sich in Tabelle 5.1 auf Seite 43. Diese Tabelle wurde direkt aus der Datei `staedte_koordinaten` übernommen, die im Anhang auf Seite 116 abgedruckt ist.

Mit Whitespace-Zeichen werden Leerzeichen, Tabulatoren und Zeilenumbrüche bezeichnet. Dabei ist es unerheblich, wieviele Whitespace-Zeichen hintereinander folgen. Zur besseren Lesbarkeit sind die Daten in dieser Datei allerdings tabellarisch angeordnet: Jede Zeile enthält nur einen Eintrag bestehend aus den drei Angaben Städtename, laterale und longitudinale Komponente.

Um Probleme mit verschiedenen Zeichensätzen zu vermeiden, empfiehlt es sich, auf die Umlaute ä,ö,ü und ß zu verzichten. Städtenamen dürfen nur aus einem Wort bestehen. Die Länge des Städtenamens sollte 15 Zeichen nicht überschreiten. Die Software verarbeitet zwar auch längere Städtenamen, die Darstellung auf dem Bildschirm würde sich in dem Fall aber ungleichmäßig verschieben.

Die Funktion `koocalc()` berechnet aus der Koordinaten-Angabe in der Grad-Buchstabe-Minuten-Darstellung eine Fließkommazahl nach der Art, wie sie in Abschnitt 5.2.1 auf Seite 43 erläutert wurde. Bei dieser Funktion handelt es sich um eine private Memberfunktion, das heißt, dass diese Funktion nur innerhalb der Klasse verwendet werden kann. Ein Zugriff von außen ist nicht möglich. In der Abbildung sind private Klassenbestandteile mit einem vorangestellten Minus gekennzeichnet. Öffentliche Bestandteile hingegen sind mit einem Plus gekennzeichnet. Die `koocalc()`-Funktion ist nicht zu verwechseln mit der calculateDistances()-Funktion. Jene berechnet die Distanzen zwischen zwei Städten. Die Berechnung wird jedoch nicht von der Funktion selbst durchgeführt. Vielmehr wird die Memberfunktion `entfernung_zu()` des Stadt-Objekts aufgerufen. Details dazu folgen weiter unten im Text.

Beim Einlesen der Datei wird jeder Stadt eine individuelle Referenznummer zugeordnet. Die Städte werden der Reihe nach, bei Eins beginnend, durchnummeriert. Über diese Referenznummer ist die Stadt intern sowie extern zu erreichen. Mit Hilfe des überladenen `[]`-Operators kann direkt ein Zeiger auf die Stadt mit dem in den Klammern angegebenen Index angefordert werden. Der `[]`-Operator kann auch statt einer Index-Nummer mit einem String angewendet werden. Der String muss dabei mit dem Städtenamen vollständig übereinstimmen. Wird ein ungültiger Name oder eine ungültige Nummer angegeben, so gibt diese Funktion einen `NULL`-Pointer zurück. Dieser Fall muss in der Software gegebenenfalls abgefragt werden. Der Versuch, einen Null-Pointer zu dereferenzieren würde einen `Segmentation fault`-Fehler auslösen, der zum sofortigen Abbruch des Programms führt.

Mit der `isValid()`-Funktion lässt sich im voraus feststellen, ob eine bestimmte Referenznummer vergeben wurde. Bei einer eingelesenen Anzahl von 30 Städten würde diese Funktion für jede Zahl zwischen 1 und 30 `true` zurück liefern. Mit der Funktion `getAnzahl()` lässt sich die Anzahl der Städte ermitteln. Für alle Zahlen über dreißig und für die Null würde die `isValid()`-Funktion ein `false` zurückgeben. Negative Zahlen sind für diese Referenzummern grundsätzlich nicht zulässig, weil sie als „unsigned int“ in der Stadtverwaltungsklasse festgelegt wurden.

Die Funktionen `firstStadt()` und `nextStadt()` erlauben ein Durchlaufen der Liste in der gespeicherten Reihenfolge. Dadurch kann von außen auf alle Elemente der Liste zugegriffen werden, ohne auf die Liste selbst Zugriff zu haben. Die `firstStadt()` Funktion setzt den internen Iterator `it` zurück an den Anfang der Liste und gibt anschließend einen Zeiger auf die erste Stadt zurück. Die `nextStadt()` Funktion inkrementiert den Iterator und gibt dann einen Zeiger auf die nächste Stadt zurück. Wenn das Listenende erreicht ist, wird ein `NULL`-Pointer zurückgegeben. Dadurch lässt sich die komplette Liste mit einer `while()`-Schleife durchlaufen.

Die Ausgabe der kompletten Liste ist mit der Funktion `printStaedteListe()` möglich. Es werden dann der Reihe nach alle Städtenamen mit ihren dazugehörenden Referenznummern ausgegeben. Die Funktion `printDistList()` ergänzt die Liste um jeweils eine Entfernungsangabe zu einer als Parameter übergebenen Referenzstadt. Diese beiden `print..()`-Funktionen werden von dem Stadt-Info-CGI eingesetzt. Dieses CGI ist in Abschnitt 5.2.5 auf der nächsten Seite beschrieben.

Die private Variable `opt_mitnr` der Stadt-Klasse ist ein boolscher Schalter, der für die Ausgabefunktion des überladenen ostream-`<<`-Operators benötigt wird. Mit dem Schalter wird angegeben, ob bei der Ausgabe neben dem Städtenamen auch die Referenznummer mit ausgegeben werden soll. Dieses Verhalten lässt sich über die Setter-Funktion `setopt_mitnr` ein- und ausschalten.

Die `entfernung_zu()`-Funktion berechnet die Entfernung zu einem anderen Stadt-Objekt, das als Pointer übergeben wurde. In dieser Funktion wird die Entfernung gemäß der Formel 5.5 auf Seite 44 berechnet. Um die Koordinaten wie gefordert in das Bogenmaß umzuwandeln, wird die Memberfunktion `Bogenmass()` der *koo*-Klasse verwendet. Diese Funktion gibt ein neues koo-Objekt zurück, so dass auch hier über die Attribute `lon` und `lat` auf die Komponenten des Koordinatenpaars zugegriffen werden kann. Die koo-Klasse ist bewusst einfach gehalten, weil dadurch lediglich die Handhabung der Koordinaten vereinfacht werden sollte. Von der koo-Klasse erzeugt der Compiler keine separate Objektdatei. Die Koo-Klasse kann und soll nur innerhalb der Stadtklasse verwendet werden.

5.2.4. CGI-Query Schnittstelle

Die Klasse *Cgiquery* stellt eine Sammlung von Funktionen zur Verfügung, die nützlich sind, wenn für ein C++-Programm eine CGI-Schnittstelle benötigt wird. CGI steht für Common Gateway Interface. Es handelt sich dabei um die standardisierte Schnittstelle eines Webservers zur Erzeugung dynamischer Inhalte. Der CGI-Standard enthält keine Vorgabe, welche Programmiersprache zu verwenden ist. Eine CGI-Schnittstelle kann daher als einfaches Skript (vgl. Kapitel 4.2.4 auf Seite 28) oder als komplexes Programm realisiert werden.

Für die nachfolgend beschriebenen Programme Stadtcgi und Tspserver wird der zuletzt genannte Ansatz unter Verwendung der Cgiquery-Klasse gewählt. Der Webserver setzt vor dem Aufruf des CGI-Programms eine Umgebungsvariable mit der Bezeichnung `QUERY_STRING`. In dieser Variabel sind die Parameter abgelegt, die der HTTP-Client mit der GET-Methode übergeben hat. Wenn statt der GET-Methode die PUT-Methode zum Einsatz kommt, kann die Cgiquery-Klasse die Parameter nicht verarbeiten, da die Übergabe auf andere Weise erfolgt und daher anders ausgewertet werden müsste.

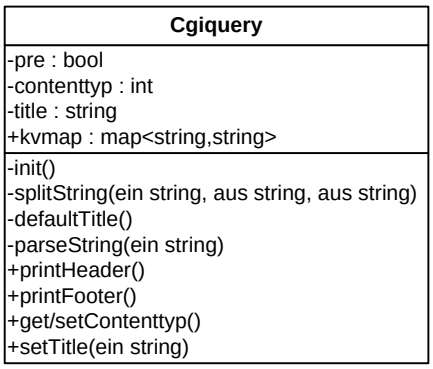

Abbildung 5.5.: CGI-Query-Schnittstelle

Bei der QUERY_STRING-Variabel handelt es sich um eine Zeichenkette mit mehreren Wert-Schlüssel-Paaren. Die Werte und Schlüssel sind jeweils mit einem =-Zeichen voneinander getrennt. Verschiedene Paare sind voneinander mit einem &-Zeichen getrennt. Sonderzeichen innerhalb des Strings werden mit einem Prozentzeichen und einem zweistelligen Hexcode kodiert, z.B. steht %20 für ein Leerzeichen (ASCII-Code 32).

Wenn die Cgiquery-Klasse instanziiert wird, ruft der Konstruktor intern die init()-Funktion der Klasse auf. Diese sorgt automatisch dafür, dass der Querystring ausgewertet wird. Dazu wird intern die parseString()-Funktion aufgerufen, die wiederum die splitString()-Funktion aufruft. Dadurch werden die Wert-Schlüssel-Paare voneinander isoliert.

Es bietet sich an, für die Wert-Schlüsselpaare einen assoziativen STL-Container zu verwenden. Dafür gibt es in C++ das map-Template. In diesem Fall wird eine Map mit zwei Strings benötigt: Ein String für den Schlüssel und ein String für den dazugehörenden Wert. Die so entstehende kvmap ist wegen der einfacheren Zugriffsmöglichkeiten als *public* gekennzeichnet. kv steht für key/value. Wenn die Instanz der Cgiquery-Klasse beispielsweise cgi genannt wird, lässt sich mit cgi.kvmap["key"] der Schlüssel mit dem Namen „key“ abfragen. Zurückgegeben wird der entsprechende Wert als String. Ein String kann mit der atoi()- Funktion aus der stdlib.h-Bibliothek in einen Integer-Zahlenwert umgewandelt werden.

Über die internen Member-Variabeln pre und contenttyp wird festgelegt, ob das Programm eine HTML-Webseite generiert, einen reinen Text oder einen vorformatierten HTML-Text. Ein vorformatierter HTML-Text hat gegenüber einer reinen Textseite den Vorteil, dass mit <a href...> Links in die Ausgabe integriert werden können. Ansonsten unterscheiden sich diese beiden Varianten nicht. Die Entscheidung für den Contenttyp muss bereits bei der Instanziierung als Parameter mit übergeben werden, weil das CGI-Interface für das korrekte Setzen des Contenttyps im HTML-Header verantwortlich ist.

Die Funktionen printHeader() und printFooter() bieten eine komfortable Möglichkeit, ein minimalistisches HTML-Gerüst zu erzeugen. Mit der Funktion setTitle() lässt sich eine Titelzeile definieren, die in der Fensterzeile des Browsers angezeigt wird. Wird kein Titel definiert, so wird intern ein Standard-Titel vergeben, der aus der aktuellen URI besteht. Die URI lässt sich über die Umgebungsvariabel REQUEST_URI abfragen, die ebenfalls vom Webserver automatisch gesetzt wird.

5.2.5. Stadt-Info-CGI

Bei dem Stadt-Info-CGI handelt es sich um ein Web-Interface für die in Abschnitt 5.2.3 auf Seite 45 vorgestellte Stadtverwaltungs-Klasse. Dieses Programm bietet vier verschiedene Funktionen, die über den action-Parameter gewählt werden können. Wenn dieser Parameter fehlt oder einen ungültigen Wert enthält, wird eine Hilfe-Seite generiert, die alle möglichen Funktionen auflistet und kurz erläutert. Die Funktion kann dann direkt über Radio-Buttons angewählt werden. Eventuell benötigte zusätzliche Parameter lassen sich über die bereitstehenden Formularfelder eingeben. Eine auf diese Art angebotene Hilfeseite erleichtert die manuelle Verwendung dieses Programms, da die Parameter nicht einzeln in die Adresszeile des Browsers eingegeben werden müssen. In Abbildung 5.6 auf der nächsten Seite ist die Hilfeseite zu sehen, wie sie im Webbrowser Firefox dargestellt wird.

Das Stadt-CGI greift auf die Städte-Datenbank zu, die bereits in Abschnitt 5.2.3 beschrieben wurde. Als Datenbank wird eine Datei mit dem Namen staedte_koordinaten geöffnet. Diese Datei muss sich im selben Verzeichnis wie das Programm befinden. Das CGI-Verzeichnis des Webbrowsers Apache ist

Abbildung 5.6.: Webbrowser beim Aufruf des Stadt-Info-CGI

unter Linux normalerweise `/usr/lib/cgi-bin`. Es ist darauf zu achten, dass die Rechte entsprechend gesetzt werden, damit der Webbrowser diese Dateien öffnen kann. Der lesende Zugriff auf die Datenbank sowie der ausführende Zugriff auf die Programmdatei muss für den Benutzer `www-data` erlaubt sein.

Mit dem Kommando `action=list` wird eine Liste aller Städte generiert. Die Liste enthält die Namen der Städte, ihre dazugehörenden Referenznummern und weitere Spalten, die eine Verlinkung zu anderen Funktionen des CGI's herstellen. Dadurch ist eine geschlossene Navigationskette innerhalb des Stadt-CGIs gegeben. Die letzte Spalte enthält weiterhin einen direkten Link zu dem freien OpenStreetMap-Projekt[1]. Die Koordinaten der aktuellen Stadt sind bereits als Parameter in dem Link enthalten. Ein Klick genügt also, um die gewählte Stadt in einer Kartendarstellung anzuzeigen. Auf diese Weise lässt sich die Korrektheit der Koordinaten überprüfen. Ein Screenshot der erzeugten Liste ist in Abbildung 5.7 auf der nächsten Seite dargestellt.

Mit dem `action=info`-Kommando lassen sich Informationen zu einer bestimmten Stadt abfragen. Die Stadt wird mit ihrer Referenznummer über den `stadt1`-Parameter festgelegt. In Abbildung 5.8 auf der nächsten Seite ist die Antwort des CGI-Programms für die Stadt Bremen dargestellt. Die Koordinaten werden in drei Darstellungsarten angeboten. Dadurch lässt sich die korrekte Umrechnung an anderer Stelle überprüfen. Die erzeugte Seite endet mit zwei Links zu den anderen CGI-Funktionen `list` und `listdist`, wobei der Parameter der aktuellen Stadt (hier 2 für Bremen) mit übergeben wird.

Das Kommando `action=listdist` bewirkt, dass eine Liste aller Städte mit ihrer Entfernung zu einer Referenzstadt angezeigt wird. Die Referenzstadt wird über den Parameter `stadt1` definiert. Die Ausgabe im Webbrowser ist in Abbildung 5.9 auf der nächsten Seite dargestellt. Als Referenzstadt wurde in diesem Fall Bremen gewählt. Daher beträgt die Entfernungsangabe hinter dem Namen der Stadt Bremen Null Kilometer. Die Ausgabe dieses Kommandos wird bis auf die Kopfzeile von der Memberfunktion `printDistList()` der Stadtverwaltungsklasse erzeugt. Die Stadtverwaltung wurde in Abschnitt 5.2.3 auf Seite 45 vorgestellt.

[1] Open-Street-Map: http://www.openstreetmap.org

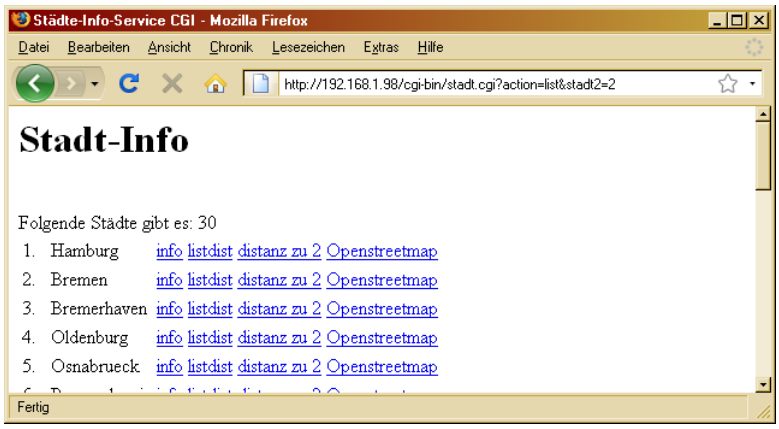

Abbildung 5.7.: Stadtcgi mit Städteliste (action=list)

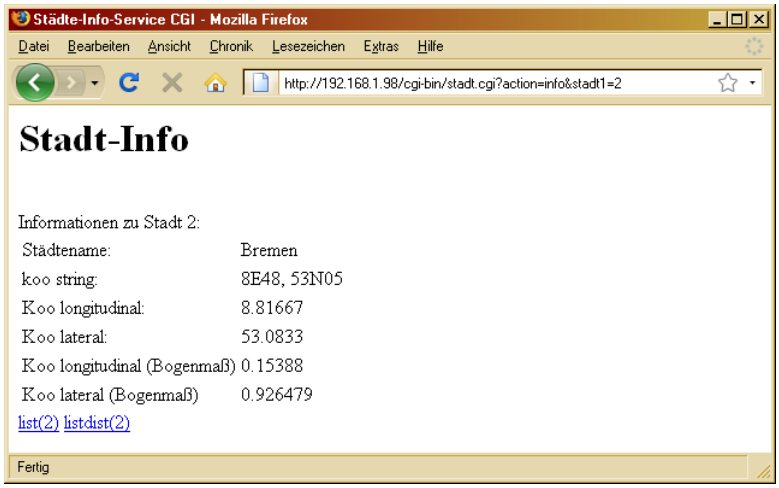

Abbildung 5.8.: Stadtcgi mit Informationen über eine Stadt (action=info)

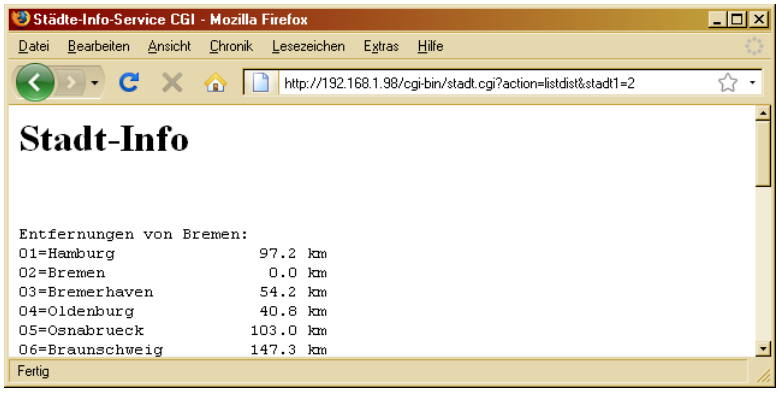

Abbildung 5.9.: Stadtcgi mit einer Entfernungsliste (action=listdist)

Das verbleibende Kommando `action=distanz` berechnet die Entfernung zweier Städte. Hierbei ist es erforderlich, dass *zwei* Städtenummern, also `stadt1` und `stadt2` als Parameter übergeben werden. Die Entfernung wird intern über die Member-Funktion `calculateDistanz()` der Stadtverwaltung berechnet. In Abbildung 5.10 ist die Ausgabe dieser Funktion für die beiden Städte Hamburg und Bremen dargestellt. Neben dem Zahlenwert werden zusätzlich die beiden Städtenamen ausgegeben. Dadurch lässt sich überprüfen, ob die richtigen Koordinaten verwendet wurden.

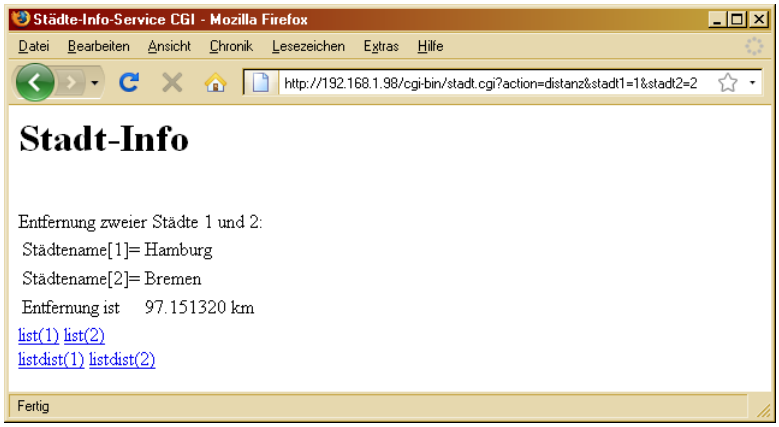

Abbildung 5.10.: Stadtcgi mit Entfernungsberechnung zweier Städte (action=distanz)

5.2.6. Tsp-Klasse

Die Klasse Tsp ist für die Lösung des *Travelling Salesman Problems* zuständig. Nach der Instanzierung werden dem Tsp-Objekt mit der `addStaedte()`-Funktion die Städte übergeben, die zu besuchen sind. Die Städte werden als String mit Referenznummern angegeben, die jeweils durch Kommata voneinander getrennt sind. Damit das Tsp-Objekt die Städte dereferenzieren kann, wird dieser Funktion zusätzlich der Zeiger auf das Stadtverwaltungsobjekt übergeben. Aus dem Nummernstring wird die Städteliste `vorgabe` rekonstruiert.

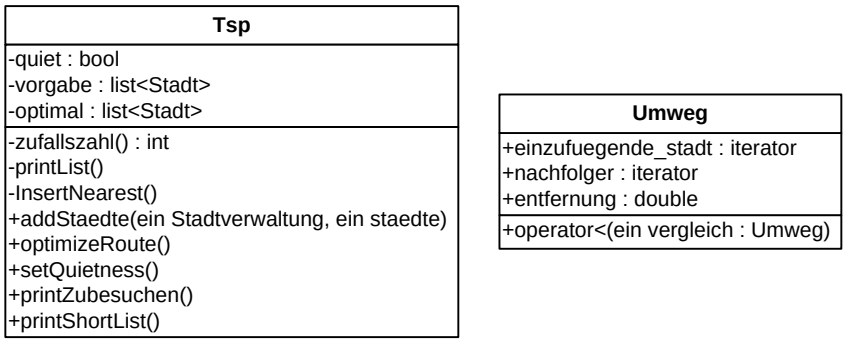

Abbildung 5.11.: Tsp- und Umweg-Klasse

Mit der `printZubesuchen()`-Funktion lässt sich die Liste der zu besuchenden Städte ausgeben. Bei der Ausgabe wird automatisch die Entfernung von einer Stadt zur nächsten berechnet. Am Ende der Liste wird die Entfernung von der letzten zur ersten Stadt berechnet. Die Summe aller Entfernungen ist die Länge der Rundreise.

Die `optimizeRoute()`-Funktion ist dafür zuständig, dass die ursprüngliche Liste `vorgabe` in die optimierte Liste `optimal` überführt wird. Beide Liste enthalten die gleichen Elemente, allerdings in einer unterschiedlichen Reihenfolge. Die Optimierung wird mit dem Verfahren des *nächsten Nachbarn* durchgeführt. Nach abgeschlossener Optimierung wird die optimierte Liste in der oben beschriebenen Form ausgegeben.

Der verwendete Code für den *Insert Nearest Neighbour* Algorithmus ist in Listing A.2 im Anhang auf Seite 76 dargestellt. Es wurden zahlreiche Kommentare eingefügt, um den Ablauf zu verdeutlichen. Die Funktion bekommt als Vorgabe eine Städte-Liste. Dabei handelt es sich um das Attribut `vorgabe` der Tsp-Klasse. Damit die ursprüngliche Liste erhalten bleibt, wird innerhalb der Funktion eine Arbeitskopie mit der Bezeichnung `zubesuchen` angelegt. Für die zu erzeugende, neue Liste mit der optimierten Reihenfolge der Städte wird eine neue Liste mit der Bezeichnung `route` angelegt. Die Variable `route` selbst ist ein Zeiger auf die neue Liste. Dieser Zeiger wird am Ende der Funktion mit `return` zurückgegeben.

Als Startwert für den Algorithmus werden 2 oder 3 Städte per Zufall ausgewählt, je nachdem wie lang die Route insgesamt ist. Die Zufallszahl ergibt sich aus einer Hilfsfunktion `zufallszahl()`, die als Parameter die Listengröße als Maximalwert übergeben bekommt. Auf die Liste lässt sich im Gegensatz zu einem Array nicht direkt mit dem `[]`-Operator auf ein Element zugreifen. Daher wird mit einer `for`-Schleife ein Iterator so lange erhöht, bis er auf das gewünschte Element zeigt. Die ausgewählte Stadt wird aus der Arbeitskopie `zubesuchen` gelöscht und an die neue `route` angehängt.

In die nun entstandene neue Route, die zu Beginn aus 2 oder 3 Städten besteht, werden nacheinander alle verbleibenden, noch zu besuchenden Städte eingefügt. Für jede Stadt wird die optimale Position zum Einfügen ermittelt. Eine in Frage kommende Position befindet sich zwischen zwei Städten der bereits bestehenden Route. Die letzte und die erste Stadt sind auf diese Weise ebenfalls miteinander verbunden, da es sich beim *Traveling Salesman-Problem* um eine Rundreise handelt. Endpunkt der Reise ist immer der Startpunkt. Es gibt also immer eine Vorgänger-Stadt und eine Nachfolger-Stadt.

Die direkte Entfernung $d_{(A,B)}$ zwischen zwei Städten A und B wird in der Variable `direkte_entfernung` abgespeichert. Wenn die einzufügende Stadt C zwischen A und B eingefügt werden soll, entstehen zwei neue Entfernungen $d_{(A,C)}$ und $d_{(C,B)}$. Statt des direkten Wegs von A nach B wird ein Umweg über C gefahren. Es ergibt sich als Entfernung für den Umweg $d_{\text{Umweg}} = d_{(A,C)} + d_{(C,B)} - d_{(A,B)}$.

Für jede mögliche Position wird die Umwegsentfernung verglichen. Zur leichteren Verwaltung der Ergebnisse wird eine Hilfsklasse Umweg eingeführt. Im Programmcode gibt es eine Instanz `umwegD`, die *diesen* Umweg, also den zur Zeit betrachteten Umweg, enthält. Das beste Ergebnis mit dem kürzesten Umweg wird in einer weiteren Instanz der Klasse `umwegK` gespeichert. Der Vergleich zweier Umweg-Objekte wird mit dem überladenen, boolschen `operator<` möglich.

Die Klasse enthält neben der Entfernungsangabe weitere Informationen. Ein Iterator auf die einzufügende Stadt (also C) ist ebenso enthalten wie ein Iterator auf die Einfüge-Position, die durch den Nachfolger (also B) charakterisiert ist. Der Vorgänger muss nicht abgespeichert werden, da sich die Anordnung in der Route erst am Ende der Schleife verändert. Für die Einfüge-Operation `insert()` der Listenklasse ist ebenfalls nur die Angabe eines Nachfolgers erforderlich.

Wenn alle möglichen Positionen der bereits bestehenden Route mit allen noch zu besuchenden Städten ausprobiert wurden, hat sich eine der Kombinationen als optimal herausgestellt. Alle nötigen Informatinen befinden sich nun in `umwegK`. Die Einfüge-Operation in die neue Liste wird durchgeführt, und die Stadt aus der ursprünglichen Liste der zu besuchenden Städte entfernt. So lange noch Städte übrig sind, wird dieser Vorgang wiederholt.

5.2.7. Tspserver-CGI

Das Tspserver-CGI-Programm ist ein CGI-Interface für die Tsp-Klasse. Es empfängt vom Client eine Liste von Städten, die besucht werden sollen. Die Liste besteht aus den Referenznummern der Städte. Voraussetzung ist, dass auf Client-Seite dieselbe Städte-Datenbank verwendet wird. Dann sind die Referenznummern der verschiedenen Programme zueinander kompatibel. Referenznummer 1 bezeichnet also immer Hamburg, 2=Bremen usw.. Die einzelnen Listenelemente sind durch Kommata voneinander getrennt. Die Liste kann beliebig lang sein. Der Server beantwortet diese Anfrage, indem er die Liste so sortiert, dass die Gesamtentfernung bei einer Rundreise möglichst kurz ist. Die sortierte Liste wird in dem gewünschten Format zurück gegeben.

Das Programm lässt sich über insgesamt vier Parameter steuern, die nachfolgend beschrieben werden. Wenn das CGI-Programm ohne jeglichen Parameter aufgerufen wird, wird eine Hilfeseite angezeigt, die die erwarteten Parameter zusammen mit einer kurzen Erläuterung auflistet. Eine solche Hilfeseite ist in Abbildung 5.12 dargestellt.

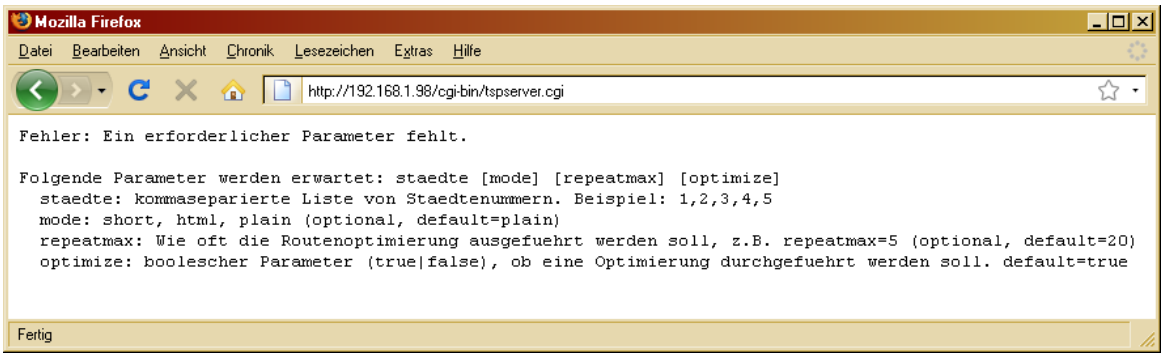

Abbildung 5.12.: Hilfeseite des Tspserver-CGI

Der Parameter `staedte` ist der einzige verpflichtende Parameter. Für alle anderen Parameter können Default-Werte angenommen werden. Dieser Parameter enthält jedoch die eigentliche Server-Anfrage, nämlich die Liste der Städte, die zu besuchen sind. Das Programm reicht diese Liste an die Tsp-Klasse weiter. Dort wird innerhalb kurzer Zeit mit einem heuristischen Verfahren eine gute Lösung für das gegebene Problem gefunden.

Mit dem `mode`-Parameter wird das Ausgabeformat des CGI-Programms festgelegt. Mögliche Werte sind `short`, `html` und `plain`. Wenn dieser Parameter fehlt, wird angenommen, dass das Format `plain` gewünscht ist. Das Format `short` bezeichnet die kürzest mögliche Darstellung. Der `Content-type` wird in diesem Fall auf `text/plain` eingestellt, und die Server-Antwort besteht nur aus einer einzelnen Zeile. Dieses Format eignet sich besonders gut für eine automatische Verarbeitung.

Für die Darstellung bei einem manuellen Aufruf durch einen Webbrowser ist dieses Format nicht optimiert. In diesem Fall sollte das Format `html` gewählt werden. Die Antwort des Servers wird dann etwas ausführlicher und zweispaltig dargestellt, wie in Abbildung 5.13 auf der nächsten Seite zu sehen ist. In der linken Spalte ist die ursprüngliche Reihenfolge der Städte dargestellt. In der rechten Spalte befindet sich die vorgeschlagene optimierte Route. Die Alternative `plain` stellt die Antwort ebenfalls ausführlich dar, verzichtet allerdings auf die mehrspaltige Darstellung. Die dafür notwendigen HTML-Tags entfallen somit.

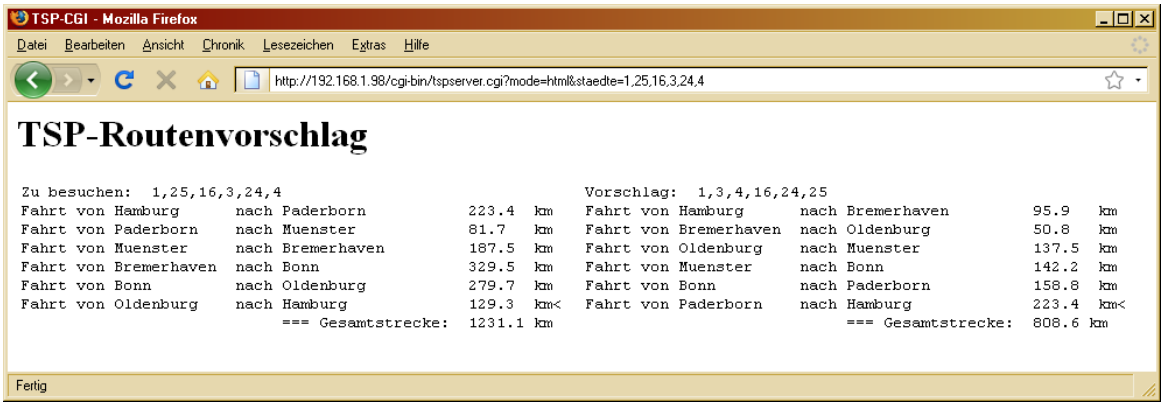

Abbildung 5.13.: Antwort des Tspserver-CGI im HTML-Format

Der `repeatmax`-Parameter wird normalerweise nicht benötigt. Im Programm wird diese Variabel verwendet, um eine Endlosschleife zu verhindern. `repeatmax=20` ist die Voreinstellung. Normalerweise wird diese Anzahl an Wiederholungen jedoch nicht erreicht. Falls die Anzahl dennoch als zu hoch empfunden wird, kann sie durch den `repeatmax`-Parameter begrenzt werden. Im Normalfall ist keine Wiederholung erforderlich. Es kann aber der Fall auftreten, dass die vorgeschlagene Route länger ist als die ursprünglich angegebene.

Das passiert zum Beispiel, wenn bereits eine optimierte Route als Anfrage gesendet wird. Um diesen Fall auszuschließen, wird der Vorgang solange wiederholt, bis die Länge kleiner oder gleich der Originallänge ist. Der Parameter ist nach obenhin nicht begrenzt. Es ergibt sich jedoch gegebenenfalls eine längere Laufzeit und somit eine größere Latenzzeit für die Serverantwort. Falls der Optimierungsvorgang wiederholt wird, ergibt sich daraus auch eine längere Server-Ausgabe. Wenn `mode=short` gesetzt ist, wird allerdings wie gewünscht nur die zuletzt bestimmte Lösung ausgegeben. Die Wiederholungen sind dann unsichtbar.

Der noch nicht erwähnte Parameter `optimize` ist mit `true` vorbelegt. Wenn dieser Parameter mit einem anderen Wert belegt wird, wird die Funktionalität des Programms deaktiviert. Die gesendete Routenanfrage wird unverändert zurück gegeben. Dies ermöglicht eine Zweckentfremdung des Programms, um im plain- oder html-mode nur die Städteliste anzuzeigen. Im short-mode bleibt die Ausgabe leer. Die Kombination `optimize=false&mode=short` ist daher nicht sinnvoll.

5.3. Zentraler Imote2.Linux Knoten

Der zentrale Sensorknoten mit dem Linux-Betriebssystem stellt das Herzstück dieses Konzepts dar. Dieser Sensorknoten verfügt einerseits über eine TCP/IP-Schnittstelle, die mit dem USB-Kabel realisiert wird, und andererseits über ein CC2420-Modul, mit dem Pakete drahtlos über IEEE 802.15.4 versendet und empfangen werden können.

Da dieser Knoten mit einem Embedded-Linux-System ausgestattet ist, können alle Vorteile dieses Betriebssystems genutzt werden. Der Sensorknoten lässt sich mit einer hardwarenahen Hochsprache programmieren. Mit Hilfe eines ARM-Cross-Compilers lässt sich beliebiger Code ausführen, der auf einem Xscale PXA 27x-Prozessor läuffähig ist. Von den 32 MB Flash-Speicher stehen auf einer neu

installierten Linux-Plattform ca. 18 MB für eigene Programme und Daten zur Verfügung. Damit lassen sich auch umfangreichere Projekte realisieren.

Das Programm, das in diesem Szenario zum Einsatz kommt, nennt sich *Imote2Basis* und ist in C++ programmiert. Dieser Name wurde gewählt, weil es sich bei diesem Sensorknoten um die Basisstation des Konzepts handelt und weil es sich dabei um eine Imote2-Plattform handelt. In der Abbildung 5.14 ist der Zusammenhang der einzelnen Softwarekomponenten der Basis-Station dargestellt. In der Mitte des Bildes befindet sich die Basis-Klasse selbst, von der alle weiteren Verzweigungen ausgehen. Die `main()`-Funktion des Hauptprogramms besteht ausschließlich aus dem Befehl zum Aufrufen der Startfunktion der Basisklasse.

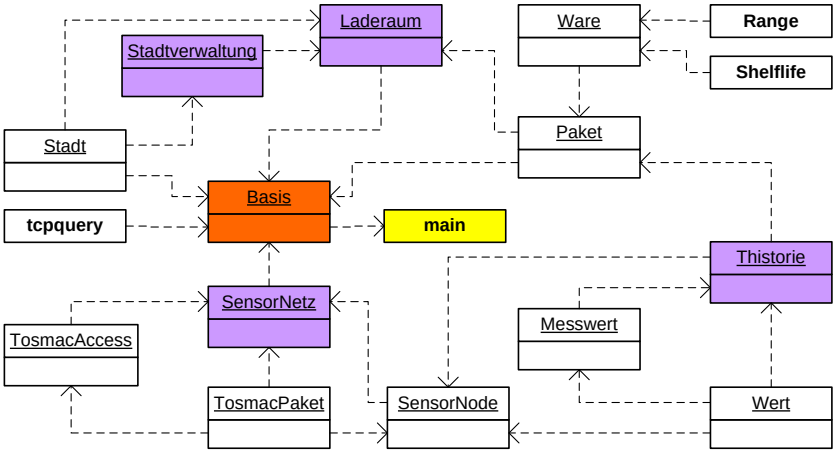

Abbildung 5.14.: Klassendiagramm von Imote2Basis

Bei der Umsetzung dieses Konzepts wurde die Sprache C++ gewählt. Einer der vielen Vorteile dieser Sprache ist es, dass eine umfangreiche Sammlung an Standard-Bibliotheken genutzt werden kann, um die Programmierung zu erleichtern. Die Kapselung in Klassen und Objekte wurde konsequent umgesetzt. Zur Verwaltung mehrerer, gleichartiger Objekte wurde oft das Konzept der STL-Container verwendet, das in Abschnitt 5.2.2 auf Seite 44 vorgestellt wurde. Die in Abbildung 5.14 dargestellten Klassen werden in den folgenden Abschnitten der Reihe nach vorgestellt. Auf den eigentlichen Programmablauf, der sich durch Funktionsaufrufe in der Basisklasse ergibt, wird in Abschnitt 5.3.6 eingegangen. Ein Testszenario mit 20 Städten wird am Ende des Kapitels vorgestellt.

5.3.1. Paket-Klasse

Jede Wareneinheit ist in einem Paket verpackt, das wiederum mit einem Zielort adressiert und von einem Messprotokoll begleitet wird. Eine Wareneinheit ist charakterisiert durch ihre Transportbedingungen. Das sind die Minimal- und Maximalwerte des optimalen Temperaturbereichs sowie der Anfangswert des Qualitätszustands.

In Abbildung 5.15 auf der nächsten Seite ist die Modellierung der vier Klassen *Paket*, *Ware*, *Shelflife* und *Range* dargestellt. Das Paket enthält jeweils einen Zeiger auf eine Ware, eine Zielstadt, einen Sensorknoten und eine Temperaturhistorie. Zeiger sind in dieser Darstellung mit einem $*$ gekennzeichnet.

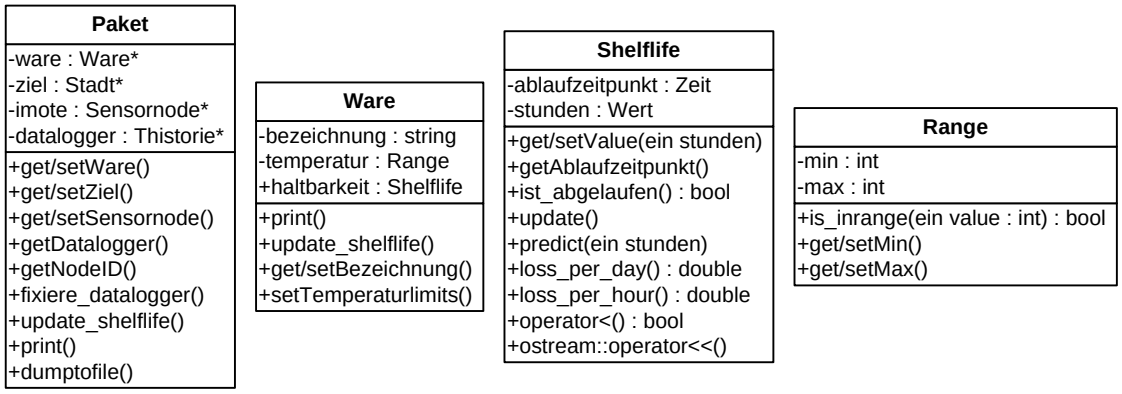

Abbildung 5.15.: Paket-, Waren-, Shelflife- und Range-Klasse

Die Ware ist in dem Paket enthalten. Sie ist durch eine Bezeichnung, einen optimalen Temperaturbereich und eine Haltbarkeit gekennzeichnet. Die Haltbarkeit wird durch die `Shelflife`-Klasse repräsentiert. Der Temperaturbereich wird durch die `Range`-Klasse beschrieben. Die Zielstadt wird durch die `Stadt`-Klasse modelliert, die in Abschnitt 5.2.3 auf Seite 45 beschrieben wird. Durch den Zeiger auf ein `Sensornode`-Objekt wird auf denjenigen Imote2 verwiesen, der für die Aufzeichnung der Sensordaten in der räumlichen Umgebung dieses Pakets zuständig ist. Die Sensornetz-Modellierung wird in Abschnitt 5.3.2 auf Seite 58 beschrieben. Die Variable `datalogger` ist ein Pointer auf ein `Thistorie`-Objekt, das durch diesen Sensorknoten erzeugt wurde. Während der Phase der Messwertaufnahme weist der Zeiger auf das aktuelle `Thistorie`-Objekt. Nachdem das Paket entladen wurde, wird mit der `fixiere_datalogger()`-Funktion eine Kopie erzeugt, auf die dann der Zeiger verweist.

Die Paketklasse verfügt jeweils über *Getter*- und *Setter*-Funktionen für die als private markierten Attribute `ware`, `ziel`, `imote` und `datalogger`. Private Attribute und Funktionen sind in dieser Darstellung mit einem vorangestellten Minus-Zeichen gekennzeichnet, öffentliche Attribute und Funktionen hingegen sind mit einem vorangestellten Plus-Zeichen gekennzeichnet. Es entsteht eine Kapselung, weil auf die Attribute der Klasse nur indirekt zugegriffen werden kann. Dadurch wird es möglich, neben der eigentlichen Zuweisung oder Abfrage der Variablen, zusätzliche Anweisungen auszuführen. Auf diese Weise kann außerdem gezielt gesteuert werden, auf welche Attribute lesend oder/und schreibend zugegriffen werden kann und auf welche nicht.

Die `update_shelflife()`-Funktion, erlaubt es, von außerhalb auf die Shelflife-Klasse zuzugreifen. Genauer gesagt wird auf die `update()`-Funktion zugegriffen. Da die Paket-Klasse jedoch keine eigene Haltbarkeit besitzt, sondern nur die Waren-Klasse, wird die `update_shelflife()`-Funktion an die Waren-Klasse weitergereicht, die wiederum den Aufruf an die Shelflife-Klasse weiterreicht. In der Shelflife-Klasse bewirkt der Funktionsaufruf, dass der in Stunden angegebene Haltbarkeitswert anhand der aktuellen Zeit an den Ablaufzeitpunkt angepasst wird. Diese Funktion kann und soll noch entsprechend erweitert werden, um zusätzlich den Ablaufzeitpunkt selbst zu modifizieren. Dazu sollen Qualitätsmodelle auf Basis der Temperaturhistorie durchgerechnet werden. Dies ist derzeit jedoch noch nicht implementiert.

Die `dumptofile()`-Funktion generiert eine Textdatei im Unterverzeichnis `lieferscheine/`. Diese Funktion wird üblicherweise am Ende eines Liefervorgangs beim Entladen des Pakets aus dem Laderaum aufgerufen. Die Laderaum-Klasse wird im Abschnitt 5.3.6 auf Seite 64 beschrieben. Die Textdatei enthält einen ausführlichen Bericht über den Transportvorgang. Neben einer genauen Beschreibung des Paket-Inhalts sind sämtliche Variablen der Paket-Klasse in eine Textform überführt worden und

dort abgespeichert. Es wird außerdem die gesamte Temperaturhistorie abgespeichert. Auch im Nachhinein ist somit noch nachvollziehbar, ob es während des Transportvorgangs zu Unregelmäßigkeiten gekommen ist, so dass eine lückenlose Qualitätsüberwachung vorliegt. Der auf diese Weise erzeugte Lieferschein ließe sich in einer Weiterentwicklung der Software zusätzlich durch eine kryptographische Signatur vor unbefugten Änderungen schützen, so dass der Logistikdienstleister den einwandfreien Transportvorgang glaubhaft nachweisen kann.

Die durch die *Range*-Klasse angegebenen Temperaturgrenzen werden im Programm derzeit nicht ausgewertet. In dem vorliegenden Stand der Software werden die Felder des Range-Objekts in allen Fällen mit Defaultwerten gefüllt. Der Vollständigkeit halber wird diese Klasse hier dennoch erwähnt. Die Existenz dieser Klasse soll die spätere Erweiterung der Software erleichtern. Die Klasse besitzt einen Minimal- und Maximalwert für die Temperatur. Die Funktion `is_inRange()` überprüft, ob ein gemessener Temperaturwert innerhalb dieser Schranken liegt. Verzweigungen innerhalb des Programms werden so vereinfacht, da nicht jede der beiden Grenzen einzeln verglichen werden muss. Jeder Ware kann eine individueller Temperaturbereich zugewiesen werden. Dies kann eine Alternative oder Ergänzung zur Shelflife-Modellierung mittels dynamischer Qualitätsmodelle sein.

Die *Shelflife*-Klasse besitzt eine Prediktionsfunktion `prediction()`. Dies ermöglicht dem Programm eine Zukunftsprognose. Als Parameter wird eine Anzahl von Stunden angegeben. Als Rückgabe wird ein neues Shelflife-Objekt erzeugt, dass entsprechend weniger Stunden Resthaltbarkeit besitzt. Derzeit ist der Verlust (`loss_per_day()`) konstant mit 24 Stunden pro Tag modelliert. Das heißt, jede Stunde des realen Zeitfortschritts wird die Resthaltbarkeit um eine Stunde reduziert. Es gibt also einen festen Zeitpunkt, an dem die Ware den Shelflife-Wert 0 erreicht. Dieser Wert lässt sich in diesem Szenario durch keine äußeren Einflüsse verändern. Diese Modellierung sollte noch erweitert werden, in dem die Qualitätsmodelle von der Universität Wageningen mit einfließen. Die Implementierung wird durch die bereits vorbereiteten Funktionen erleichtert.

Der Ablaufzeitpunkt in der Shelflife-Klasse ist durch ein Zeit-Objekt angegeben. Die Zeit-Klasse wird in Abschnitt 5.3.5 auf Seite 62 beschrieben. Für Demonstrationszwecke wird die sogenannte beschleunigte Systemzeit verwendet. Dadurch können reale Haltbarkeits- und Transportzeiten verwendet werden, während die Simulationsdauer um einen bestimmten Faktor kürzer ist. Die Wartezeiten auf das Ende des Programmdurchlaufs können dadurch reduziert werden.

Der boolsche Vergleichsoperator „`<`“ wird in der Shelflife-Klasse überladen. Dies erlaubt es, zwei Shelflife-Werte miteinander zu vergleichen, ohne den Umweg über die `getValue()` Funktion nehmen zu müssen. Dadurch lässt sich eine Liste mit mehreren Shelflife-Werten automatisch sortieren. Die Funktion des Operators „`>`“ ist damit implizit ebenfalls gegeben.

Der Operator „`<<`“ hingegen vergleicht nicht. Vielmehr handelt es sich hierbei um eine Überladung des `ostream`-Operators. Eine Ausgabe mit der `cout<<`-Anweisung wird dadurch möglich. In C++ wird diese Anweisung verwendet, um Ausgaben aller Art auf dem Bildschirm oder in eine Datei auszugeben. Durch die Überladung des `ostream`-Operators kann festgelegt werden, in welcher Form der Inhalt der Klasse ausgegeben werden soll. Bei diesem Operator handelt es sich nicht um eine Memberfunktion der Klasse. Er wird trotzdem aufgeführt, weil er in der Klassendeklaration als `friend` markiert ist und dadurch Zugriff auch auf private Attribute hat.

5.3.2. Sensornetz-Klasse

Das drahtlose Sensornetz wird durch eine übergeordnete Sammelklasse repräsentiert, die Objekte der Sensorknoten enthält. Jeder Sensornode ist dabei ein eigenes Objekt. Wie bereits im Abschnitt 5.1 auf Seite 40 beschrieben wurde, lässt sich jeder Sensornode durch seine NodeID eindeutig identifizieren. Dadurch lassen sich empfangene Datenpakete einem Sensornode-Objekt zuordnen.

Die Software ist mit einem Filter ausgestattet, der es ermöglicht, nur Pakete von bestimmten NodeIDs zu akzeptieren. Es lässt sich auch angeben, dass sämtliche Pakete akzeptiert werden und unbekannte NodeIDs automatisch dem Netzwerk hinzugefügt werden. Davon ist jedoch aus mehreren Gründen abzuraten. Dadurch, dass die Pakete im lizenzfreien ISM-Band 2,4 GHz übertragen werden, gibt es viele Störungen auf dem Kanal. Es kommt oft vor, dass ungültige Pakete (mit zufälligen NodeIDs) empfangen werden. Für jede dieser zufälligen NodeID würde dann ein zusätzliches, aber unbrauchbares Objekt angelegt werden. Ein zweiter wichtiger Grund, der für den Einsatz des Filters spricht, ist die Zuordnung von Sensorknoten und Frachtstücken. Diese Zuordnung wird nur dann möglich, wenn bei Transportbeginn bereits bekannt ist, welcher Sensorknoten für welches Frachtstück die relevanten Informationen liefert. Wenn neue Sensorknoten hinzugefügt werden, ist nicht klar, zu welchem Frachtstück sie relevante Informationen beitragen können.

Die Sammlung aller zu einem Sensorknoten gehörenden Messwerte wird als Thistorie zusammengefasst. Jedem Frachtstück ist ein Sensorknoten zugeordnet, der für dieses Frachtstück relevante Informationen zur Verfügung stellt. Dank der Pointerfunktionalität in C++ lassen sich diese Objekte bequem vernetzen, so dass nicht nur aus Sicht des Sensorknotens, sondern auch aus der Sicht des Frachtstücks auf das Messprotokoll zugegriffen werden kann. Ein Sensornode kann für mehrere Frachtstücke relevante Daten liefern. Mehrere Frachtstücke teilen sich also eine Thistorie. Erst beim Entladen eines Frachtstücks muss eine Kopie des Protokolls generiert werden, damit die individuelle Rückverfolgbarkeit im Sinne der Qualitätsüberwachung gewährleistet werden kann.

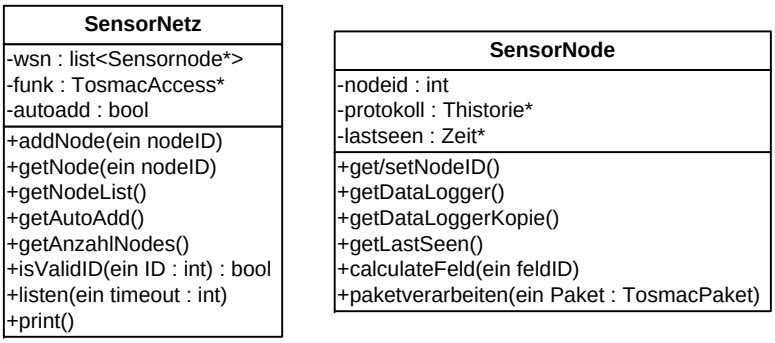

Abbildung 5.16.: Sensornetz- und Sensornode-Klasse

Die *Sensornetz*-Klasse, wie sie in Abbildung 5.16 dargestellt ist, besitzt die drei privaten Attribute `wsn`, `funk` und `autoadd`. Das Attribut `wsn` ist eine als STL-Container realisierte Liste mit Zeigern auf Sensorknoten. Das Attribut `funk` ist ein Zeiger auf die *TosmacAccess*-Klasse, die in Abschnitt 5.3.3 auf Seite 60 beschrieben wird. Die Variable `autoadd` ist ein boolscher Schalter, mit dem festgelegt wird, ob neue Sensorknoten automatisch hinzugefügt werden sollen.

Mit der Memberfunktion `addNode()` kann zur Laufzeit ein neuer Sensorknoten dem Sensornetz hinzugefügt werden. Er wird dann automatisch an die Liste angehängt. Hierbei wird einer der Vorteile bei der Verwendung von STL-Containern deutlich. Die Listengröße ist variabel und passt sich je nach

Bedarf an. Der Aufruf der `addNode()`-Funktion erfolgt im vorgestellten Programm aus der *Basis*-Klasse heraus, während der Initialisierungsphase zu Beginn des Programms. Durch diese Funktion wird zum einen das neue Objekt angelegt, zum anderen wird der anfangs erwähnte Filter spezifiziert.

Mit der `getNode()`-Funktion kann ein Zeiger auf einen der Sensorknoten angefordert werden. Als Parameter wird die NodeID angegeben. Der Sensorknoten lässt sich durch diese dynamische Referenz an mehreren Stellen im Programm parallel verwendet werden. Die Daten des Sensorknoten-Objekts sind dabei nur einmal im Speicher vorhanden. Aufgrund der in C++ vorhanden Zeigerfunktionalität muss an dieser Stelle lediglich die Speicheradresse dupliziert werden, um eine Verfügbarkeit des Objekts an mehreren Stellen zu ermöglichen.

Die Funktion `getNodeListe()` generiert ein Array mit den NodeIDs der Sensorknoten. Dies vereinfacht die dynamische Zuordnung, wie sie derzeit innerhalb der Basisklasse realisiert ist. Die Sensorknoten werden dort gleichmäßig auf die Paketanzahl verteilt, ohne die tatsächliche physikalische Position innerhalb des Laderaums zu berücksichtigen. Die Zuordnung kann in der Basisklasse mit geringem Aufwand angepasst werden, wenn sich neue Verfahren zur Lokalisierung ergeben, die sich in diesem Programm auf einfache Weise implementieren lassen.

Die `listen()`-Funktion ist eine der wichtigsten Funktionen der Sensornetzklasse. Diese Funktion gleicht durch Zugriff auf die TosmacAccess-Klasse das real existierende Sensornetzwerk mit der lokal modellierten virtuellen Repräsentation des Sensornetzwerks durch die Sensornode-Objekte ab. Als Parameter kann ein *Timeout*-Wert angegeben werden. Falls innerhalb des dadurch angegebenen Zeitraums keine Datenpakete über die Tosmac-Schnittstelle empfangen werden können, wird der Versuch abgebrochen und ein Fehlercode zurückgegeben. Falls hingegen ein Paket empfangen werden konnte, wird von der *TosmacAccess*-Klasse ein *TosmacPaket* generiert und an die `listen()`-Funktion zurückgegeben. Aus dem Tosmacpaket wird dann sofort die NodeID extrahiert, so dass eine Zuordnung zu dem entsprechenden Sensornode-Objekt möglich wird.

Die SensorNode-Klasse ist charakterisiert durch eine NodeID, ein Messprotokoll und einen Zeitstempel des letzten Funkkontakts. Die NodeID ist diejenige Nummer, die vom Imote2.NET-Knoten aus der Hardware ausgelesen wurde. Wie in Abschnitt 5.1 auf Seite 40 bereits beschrieben wurde, lässt sich dadurch jeder Sensorknoten eindeutig identifizieren. Das Protokoll ist ein Objekt der Thistorie-Klasse, die in Abschnitt 5.3.4 auf Seite 61 beschrieben wird. Auch hierbei handelt es sich nicht um das Objekt selbst, sondern lediglich um einen Zeiger, der auf das Objekt verweist. Dies erlaubt auch hier wieder eine mehrfache Verwendung des Objekts. Die Paket-Klasse profitiert von dieser Funktionalität, da sich die Zuordnung beim Entladevorgang wechseln lässt. Während des Transportvorgangs wird auf die Temperaturhistorie des Sensorknotens verlinkt. Sobald das Paket jedoch entladen ist, wird eine Kopie der Temperaturhistorie erzeugt, auf die der Zeiger dann entsprechend „umgebogen“ wird.

In der Variable `lastseen` wird der Zeitpunkt des letzten Funkkontakts mit dem Sensorknoten abgespeichert. Der Zeitpunkt wird als Objekt der Zeitklasse modelliert. Die Zuordnung zu anderen Programm-Aktivitäten, die ebenfalls durch auf die beschleunigte Systemzeit bezogene Zeitstempel markiert sind, wird auf diese Weise vereinfacht. Innerhalb des Programms ergibt sich somit eine in sich geschlossene und konsistente zeitliche Dimension, die unabhängig von der tatsächlichen Zeit der realen Welt ist. Die Zeitklasse wird in Abschnitt 5.3.5 auf Seite 62 beschrieben.

Die Memberfunktion `calculateFeld()` sorgt für die Dekodierung der Daten des Tosmacpaketes. Jeder Sensorwert wurde von dem .NET-Sensorknoten in einen 16 Bit-Wert kodiert (siehe Abschnitt 5.1 auf Seite 40). Aus diesen Daten werden in dieser Funktion die physikalischen Sensorwerte errechnet.

Dazu werden die Formeln 5.1 bis 5.3 von Seite 42 verwendet. Die jeweils nötige Feldidentifikationsnummer, die in der Abbildung 5.16 auf Seite 58 als `feldID` bezeichnet wird, entspricht der Position der Daten innerhalb des TosmacPakets. Zur leichteren Handhabung wurden einige Präprozessorkonstanten mit der `#define`-Anweisung innerhalb der `Sensornode.h`-Headerdatei festgelegt. Die Headerdatei ist im Anhang in Listing A.11 auf Seite 91 zu finden.

5.3.3. Tosmac-Klasse

Die beiden Klassen, die sich direkt auf die drahtlose Übertragung von Tosmac-Datenpaketen beziehen, sind TosmacAccess und TosmacPaket. Die Modellierung der beiden Klassen ist in Abbildung 5.17 dargestellt. Die TosmacAccess-Klasse ist für die unmittelbare Kommunikation mit dem Hardware-Treiber zuständig, während die TosmacPaket-Klasse die empfangenen Daten transportiert. Entsprechende Memberfunktionen erlauben die dezentrale Verarbeitung der Daten.

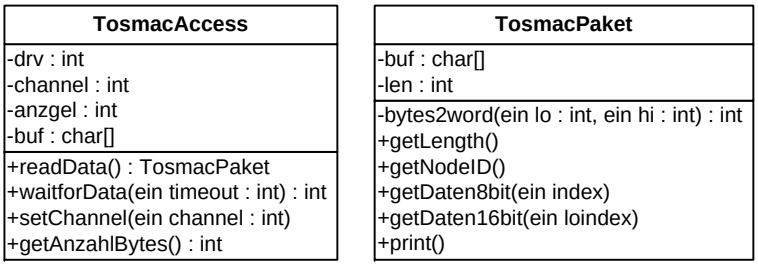

Abbildung 5.17.: TosmacAccess- und TosmacPaket-Klasse

Von der TosmacAccess-Klasse existiert nur ein einziges Objekt während der Laufzeit. In den privaten Variablen sind daher die Parameter abgespeichert, die für die Kommunikation mit dem Treiber benötigt werden. In der Variable `drv` wird die Treiber-Referenz abgelegt, die von der `open()`-Funktion des Hardware-Treibers zurückgegeben wird. Diese Referenz wird für alle weiteren Zugriffe auf den Treiber benötigt.

Die Variabel `channel` enthält den gewünschten Channel im Frequenzbereich des ISM-Bandes. Gültige Werte liegen im Bereich von 10-26. Für jeden Nutzer von drahtlosen Sensornetzwerken im NW1-Gebäude der Universität Bremen muss ein eigener Channel verwendet werden, um gegenseitige Beeinflussung zu verhindern. Die aktuelle Zuordnung von Nutzern und Channel-Nummern ist über die IMSAS-Homepage im Arbeitsgruppenbereich Logistik abrufbar.

In der `TosmacAccess.h`-Headerdatei wird über eine Präprozessorkonstante mit der Bezeichnung `DEFAULT_CHANNEL` der zu verwendende Channel festgelegt. Diese Konstante wird beim Programmstart automatisch in das TosmacAccess-Objekt übernommen. Für andere Benutzer muss sie entsprechend abgeändert werden. Während der Laufzeit lässt sich die Channel-Nummer mit der Funktion `setChannel()` verändern. Die noch nicht erwähnte Variable `buffer` ist ein `char[]`-Array mit einer bestimmten Größe, das für die temporäre Zwischenspeicherung der empfangenen Daten verwendet wird.

Durch die beiden Memberfunktionen `waitforData()` und `readData()` kann ein nicht-blockierender Zugriff auf den Tosmac-Gerätetreiber durchgeführt werden. Die `waitforData()` Funktion wartet für eine angegebenen Zeitspanne darauf, dass Daten an der Schnittstelle bereitstehen. Wenn nach Ablauf des `timeout`-Wertes immer noch keine Daten bereitstehen, wird der `return`-Wert 0 zurückgegeben.

Nur in dem Fall, dass der Rückgabewert ungleich Null ist, sollte anschließend die Funktion `readData()` ausgeführt werden. Andernfalls würde der Vorteil des nichtblockierenden Zugriffs auf den Treiber ungenutzt bleiben.

Die TosmacAccess-Klasse ist in hohem Maße von der Hardware abhängig, auf der sie ausgeführt wird. Der Tosmac-Gerätetreiber existiert ausschließlich auf der Imote2-Plattform. Zum Testen des Programmablaufs kann es allerdings von Vorteil sein, das Programm direkt auf der PC-Ebene ausführen zu können, Mit der Präprozessor-Variable `IMOTE2` kann festgelegt werden, welche Teile der Klasse kompiliert werden sollen. Wenn diese Variable nicht definiert ist, wird eine reduzierte Version dieser Klasse kompiliert, bei der die Funktionskörper der Klasse leer bleiben. In der Eclipse-Umgebung lassen sich über das Menü Project/Properties verschiedene Projektprofile definieren. Bei C/C++ Build/Settings bei Preprocessor können Symbole eingetragen, die automatisch beim Kompilationsvorgang definiert werden. Nur wenn hier das Symbol `IMOTE2` eingetragen ist, wird die komplette Klasse kompiliert.

Jedes empfangene Datenpaket wird in einem *TosmacPaket*-Objekt abgelegt. Dies verfügt über einen `buffer` aus Bytes und einer Längenvariable `len`. Das Tosmacpaket wird zum Datenaustausch zwischen der TosmacAccess-Schnittstelle und der virtuellen Sensornetzrepräsentation verwendet. Das Sensornode-Objekt wertet das das TosmacPaket aus und rechnet die Daten-Bytes in Sensor-Messwerte um. Danach wird das Tosmacpaket-Objekt nicht mehr benötigt und daher wieder aus dem Speicher entfernt. Das TosmacPaket lässt sich für Debugzwecke durch die Memberfunktion `print()` auf dem Bildschirm ausgeben.

Die private Memberfunktion `bytes2word()` fügt zwei 8 Bit-Werte zu einem 16 -Bit-Wert zusammen. Diese Hilfsfunktion wird von den beiden Memberfunktionen `getNodeID()` und `getDaten16bit()` verwendet. Die Funktion ist auf die Bytereihenfolge des sendenden Sensorknotens abgestimmt. In Abschnitt 5.1 auf Seite 40 wurde erwähnt, dass das Little-Endian-Format gewählt wurde. Dies bedeutet, dass zunächst das niederwertigere Byte übertragen wird und anschließend das höherwertige Byte.

5.3.4. Thistorie-Klasse

Von der Klasse Thistorie wird eine Temperaturhistorie verwaltet. Es handelt sich dabei um eine Sammelklasse, die eine Liste von Messwerten enthält. Jeder Messwert besteht jeweils aus einem physikalischen Wert, der durch einen Zahlenwert und eine dazugehörende Einheit gekennzeichnet ist. Ein Messwert kann mehrere physikalische Werte enthalten. Zusätzlich enthält jeder Messwert einen Zeitstempel, der den Zeitpunkt der Messung dokumentiert. Dadurch wird im Nachhinein das Temperaturverhalten nachvollziehbar. In Abbildung 5.18 sind die drei Klassen Thistorie, Messwert und Wert dargestellt. Die Klassen werden nachfolgend beschrieben.

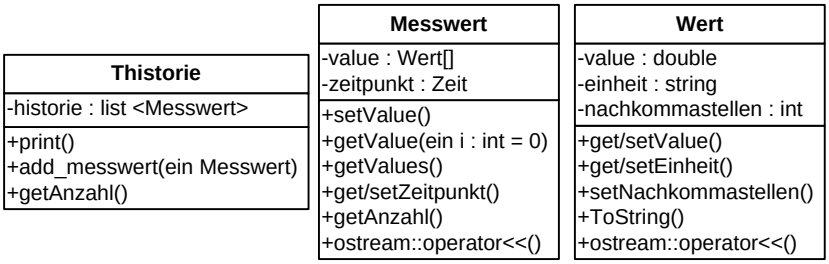

Abbildung 5.18.: Thistorie-, Messwert- und Wert-Klasse

Von der Klasse Thistorie können mehrere Objekte erzeugt werden. Jedes Objekt ist dann für einen eigenen zeitlichen Verlauf von Sensormesswerten zuständig. In dem vorliegenden Programm ist jedem Sensorknoten ein Objekt einer Thistorie Klasse zugeordnet. Beim Entladen eines Paketes aus dem Laderaum des Transportfahrzeugs wird eine Kopie dieses Objekts erzeugt und mit dem Paket verlinkt. Jedes Paket hat somit seine individuelle Temperatur-Historie, die im Lieferschein dokumentiert wird. Dadurch wird auch zu einem späteren Zeitpunkt der Einfluss der Umweltbedingungen während des Transportvorgangs nachvollziehbar, wenn etwa die Warenqualität zum Zeitpunkt der Ablieferung nicht den Anforderungen entspricht. Das ist Rahmen der Qualitätsüberwachung in der Lebensmittellogistik von Bedeutung.

Die Thistorie-Klasse besitzt neben der eigentlichen Liste mit Messwerten lediglich drei Memberfunktionen. Die Funktion `print()` gibt alle Messwerte in der Reihenfolge der Aufzeichnung auf dem Bildschirm aus. Die Funktion `add_messwert()` hängt einen neuen Messwert an die Liste an. Die Funktion `getAnzahl()` liefert die Anzahl der gespeicherten Messwerte zurück.

Die Messwertklasse war ursprünglich für die Aufnahme *eines* Messwertes konzipiert. Weil zu jedem Mess-Zeitpunkt aber stets ein ganzes Paket an Sensordaten übertragen wird, bietet es sich an, mehrere Werte in einem Objekt dieser Klasse abzulegen. Dazu muss einfach mehrmals hintereinander die `setValue()`-Funktion aufgerufen werden, der Index in dem `Wert[]`-Array wird automatisch inkrementiert, ist jedoch auf einen vorgegebenen Maximalwert begrenzt. Dieser Maximalwert ist durch eine `#define`-Anweisung in der `Messwert.h`-Header-Datei festgelegt.

Bei einem Aufruf der `getValue()`-Funktion wird, falls nicht anders angegeben, der erste gespeicherte Wert zurückgegeben. Als optionaler Parameter lässt sich sich jedoch der Index des gewünschten Elementes übergeben. Die Funktion `getValues()` hingegen gibt eine Kopie des gesamten `Wert[]`-Arrays zurück. Einmal gespeicherte Werte lassen sich im Nachhinein nicht mehr verändern. Bei der Qualitätsüberwachung stellt dies ein durchaus gewünschtes Verhalten dar, weil sich einmal protokollierte Messwerte nicht im Nachhinein manipulieren lassen sollen.

Der `operator<<` ist eine Überladung des ostream-Operators. Auf diese Weise lässt sich der Inhalt der Klasse mit der `cout<<`-Anweisung in einem vorgegebenen Format ausgeben. Im Fall der Messwert-Klasse wird eine komplette Zeile generiert. Sie beginnt mit dem Zeitstempel, der aus Datum und Uhrzeit besteht. Es folgen die Werte, die zu diesem Zeitpunkt gemessen wurden. Sie werden jeweils mit Zahlenwert und Einheit ausgegeben. Dieses Verhalten wird im `ostream`-Operator der Wert-Klasse definiert.

Die Wert-Klasse lässt sich flexibel einsetzen. Ein großer Vorteil ist, dass jeder Zahlenwert bei der Ausgabe stets von seiner dazugehörigen Einheit begleitet wird. Einer Verwechslung oder Fehlinterpretation von Messwerten lässt sich so entgegen wirken. Mit der Funktion `setNachkommastellen()` kann eingestellt werden, wieviele Nachkommastellen bei der Ausgabe dargestellt werden sollen. Bei einem ganzzahligen Messwert ist es beispielsweise nicht erforderlich, dass überhaupt Nachkommastellen angezeigt werden. In diesem Fall lässt sich mit `setNachkommastellen(0)` die Darstellung des nicht-ganzzahligen Anteils verhindern.

5.3.5. Zeit-Klasse

Die Zeit-Klasse wurde erstellt, um eine einheitliche Behandlung von Zeitstempeln innerhalb des Programms zu erreichen. Der Schwerpunkt dieser Funktion wurde daher auf zahlreiche parameterlose Funktionen gelegt, die eine Zeichenkette zurückliefern. Die Zeit-Klasse kann jedoch mehr, als nur die

Uhrzeit ausgeben. Durch die Zeit-Klasse kann eine sogenannte *beschleunigte Systemzeit* verwendet werden.

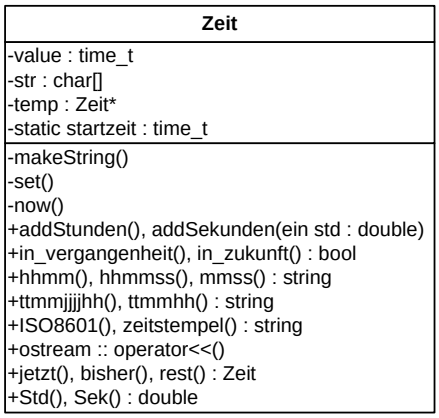

Abbildung 5.19.: Zeit-Klasse

Bei der ersten Instanziierung der Klasse wird eine statische Variable `startzeit` mit der aktuellen Systemzeit besetzt. Dafür wird die `time()`-Funktion aus der C-Bibliothek `time.h` verwendet. Der Variabel-Typ ist `time_t`. Es handelt sich dabei um die Unix-Zeit. Sie enthält die Anzahl der Sekunden seit dem 1.1.1970. Diese Variable lässt sich typecasten, so dass mit der Sekundenanzahl gerechnet werden kann.

Die Funktion `now()` berechnet die beschleunigte Systemzeit. Dafür wird zunächst wieder der aktuelle Zeitpunkt mit der `time()`-Funktion bestimmt. Von diesem Zeitpunkt wird die `startzeit` subtrahiert. Diese Differenz wird mit einem Beschleunigungsfaktor multipliziert. Das Produkt wird wieder zur `startzeit` hinzuaddiert. In der globalen Definitions-Headerdatei `defs.h` ist der Beschleunigungsfaktor durch die Präprozessor-Konstante `ACCELERATION` festgelegt. Ein Auszug aus dem Programmcode befindet sich in Listing 5.2.

Listing 5.2: Beschleunigte Systemzeit

```
time_t Zeit::now() {
        double diff = (double)(time(NULL) - startzeit);
        return startzeit + (time_t)(diff * (double) ACCELERATION);
}
```

Der Beschleunigungsfaktor lässt sich vorteilhaft zur Simulation von zeitlichen Vorgängen nutzen. Aus der Entfernung zweier Städte A und B, z.B. $d = 100\,\text{km}$ ergibt sich bei einer konstanten Geschwindigkeit eines Fahrzeugs, z.B. $v = 80\,\text{km/h}$ eine bestimmte Fahrzeit $t = \frac{d}{v} = 1,25\text{h}$. Bei normaler Geschwindigkeit müsste man also über eine Stunde warten, bis das Fahrzeug eintrifft. Statt $v = 80\,\text{km/h}$ könnte man das Fahrzeug nun mit $v = 8000\,\text{km/h}$ fahren lassen, die Zeit reduziert sich dann auf 45 Sekunden. Eine so hohe Geschwindigkeit ist jedoch nicht mehr realistisch. Eine Beschleunigung der Zeit hingegen ist anschaulicher. Daher wurde dieser Ansatz gewählt.

Der eigentliche Wert des Zeit-Objekts wird in der Variable `value` abgelegt. Auch diese Variable ist vom Typ `time_t`. Der Wert ändert sich jedoch nicht automatisch. Vielmehr stellt dieser Wert einen Bezugspunkt dar, mit dem sich die aktuelle Zeit vergleichen lässt. So lässt sich mit Zeitdifferenzen

rechnen. Die Funktionen `bisher()` und `rest()` geben ein Tochterobjekt zurück, das die Zeitdifferenz von der aktuellen Zeit und der Referenzzeit enthält. Auch diese Differenz lässt sich mit den Ausgabefunktionen darstellen. Mit den C-eigenen Funktionen wäre dieser Vorgang etwas umständlicher.

Die Funktion `bisher()` gibt die Zeitdifferenz an, wenn der Referenzzeitpunkt in der Vergangenheit liegt. Die `rest()`-Funktion hingegen gibt die Zeitdifferenz an, wenn der Referenzzeitpunkt in der Zukunft liegt. Mit den boolschen Funktionen `in_vergangenheit()` und `in_zukunft()` lassen sich diese beiden Fälle unterscheiden. Meistens ist vom Programmverlauf aber klar, welcher Fall gemeint ist. Die boolschen Funktionen werden dann eher verwendet, um festzustellen, ob ein Referenzzeitpunkt, der anfangs noch in der Zukunft lag, inzwischen erreicht ist, bzw. in der Vergangenheit liegt.

Der `value` lässt sich mit den Funktionen `addStunden()` und `addSekunden()` nachträglich verändern. Dadurch lassen sich Ablaufzeitpunkte oder Intervalle festlegen. Die Anzahl der angegebenen Stunden oder Sekunden kann auch negativ sein. Es muss nur vermieden werden, dass der absolute Zeitwert negativ wird, weil die Darstellung in dem Fall nicht mehr konsistenz ist. Die Funktion `addStunden()` wird zum Beispiel für eine Shelflife-Prognose verwendet. Die Shelflife-Klasse wird in Abschnitt 5.3.1 auf Seite 55 beschrieben.

Die Ausgabefunktionen geben den Zeitwert des Objekts in verschiedenen Formaten aus. Dies gilt auch, wenn das Objekt eine Zeit*differenz* enthält. In dem Fall machen die Datums-Ausgabefunktionen allerdings keinen Sinn, weil Differenzen von z.B. 8 Stunden intern als 1. Januar 1970, 8 Uhr morgens gespeichert werden. Die Funktion `hhmmss()` bzw. die verkürzten Varianten `hhmm()` und `mmss()` stellen die Zeit dar. Dabei steht `hh` für die zweistelligen Stunden, `mm` für die zweistelligen Minuten und `ss` für die zweistelligen Sekunden. Die Werte sind jeweils durch Doppelpunkt voneinander getrennt.

Die Funktion `ttmmjjjjhh()` gibt das Datum mit Jahreszahl aus, gefolgt von der Stunde. Dies lässt sich für Haltbarkeitsdaten verwenden. Die Funktion `ttmmhh()` liefert eine ähnliche Ausgabe, verzichtet jedoch auf die Jahreszahl. Die Funktion `ISO8601()` gibt das Datum im standardisierten ISO8601-Format aus: `jjjj-mm-ss`, das heißt, dass die vierstellige Jahreszahl an erster Stelle steht, gefolgt von dem zweistelligen Monat und dem zweistelligen Tag. Die Felder werden durch Bindestriche getrennt. Dieses Format erlaubt es, Datumsangaben mit den üblichen Sortierfunktionen chronologisch zu sortieren.

Die Funktion `zeitstempel()` schließlich gibt einen kompletten Zeitstempel bestehend aus Datum und Uhrzeit aus. Diese Funktion lässt sich für ein Messprotokoll, wie in Abschnitt 5.3.4 auf Seite 61 erwähnt, verwenden. Der überladene `ostream`-Operator`<<` verwendet die Funktion `hhmmss()` zur Ausgabe. Ein Zeitobjekt lässt sich also mit `cout<<` ausgeben. Dies erleichtert die Handhabung innerhalb des Programms.

5.3.6. Basis- und Laderaum-Klasse

Die Steuerung des Programmablaufs wird von der Basis-Klasse vorgenommen. Die Basis-Klasse ist stark vernetzt mit der Laderaumklasse, die wiederum stark mit der Route-Klasse interagiert. Der Aufbau dieser drei Klassen ist in Abbildung 5.20 dargestellt.

Die Basis-Klasse wird von der `main()`-Funktion des Programms instanziiert. Dadurch wird automatisch der Konstruktor der Klasse aufgerufen. Innerhalb des Konstruktors werden die privaten Objekte dynamisch angelegt. Die Zeiger `laderaum`, `aufenthaltsort` und `wsn` verweisen danach auf neue Objekte.

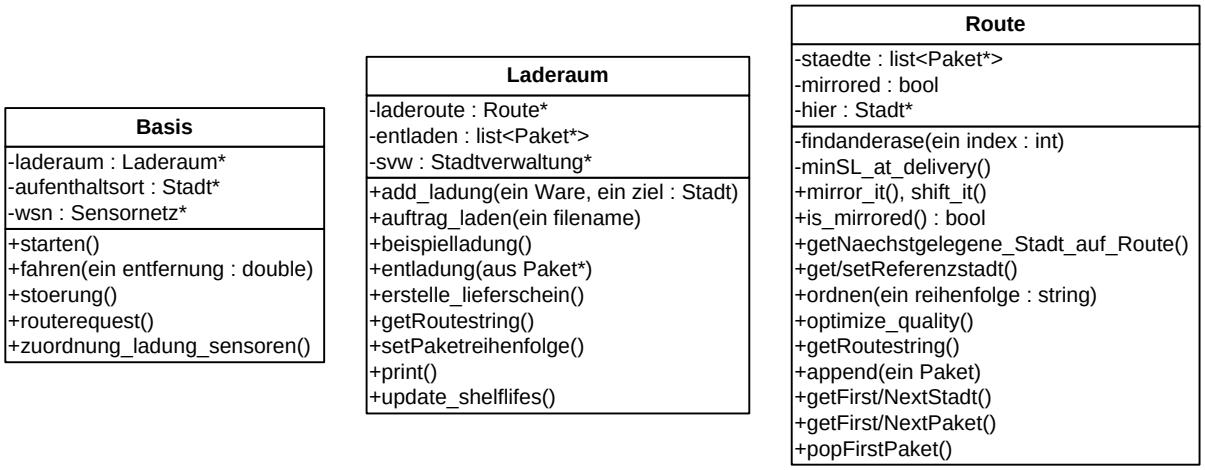

Abbildung 5.20.: Basis-, Laderaum- und Route-Klasse

Der aktuelle Aufenthaltsort ist ein Stadt-Objekt mit einer beliebigen Position, die allein durch ihre Koordinaten bestimmt ist. Es kann sich dabei, muss sich jedoch nicht, um eine Stadt aus der Stadtverwaltung handeln. Die zu verwendenden Startkoordinaten sind in der Datei `defs.h` mit der Präprozessorkonstante `STARTKOORDINATEN` festgelegt. Es handelt sich dabei um den Anfangswert zur Bestimmung der Fahrzeit. Als Vorgabe wurden die Koordinaten von Bremen verwendet. Später lässt sich das z.B. durch ein GPS-Interface ergänzen. Dafür müsste an die serielle Schnittstelle des Imote2 eine GPS-Maus angeschlossen werden. Für eine Demonstration innerhalb von Gebäuden ist das wegen des eingeschränkten GPS-Empfangs allerdings nicht immer sinnvoll.

Mit WSN wird das drahtlose Sensornetz (*wireless sensor network*) bezeichnet. Der Zeiger `wsn` zeigt daher auf ein neues Objekt der Sensornetz-Klasse, die in Abschnitt 5.3.2 eingeführt wurde. Durch die Instanziierung wird der Konstruktor der Sensornetz-Klasse aufgerufen. In dem Konstruktor wird das Sensornetz initialisiert. Dort erfolgt auch die Vorgabe für die Node-IDs, die berücksichtigt werden sollen. Im Programm sind dort zur Zeit die NodeIDs 315 und 723 fest einkodiert. Wenn andere Sensorknoten verwendet werden sollen, müssen diese Werte entsprechend geändert werden. Die Anzahl der Sensorknoten ist beliebig. Das Programm funktioniert mit *einem* Sensorknoten genauso gut wie mit fünf Sensorknoten. Es ist jedoch zu beachten, das in der aktuellen Programmversion die Sensorwerte nicht in die Qualitätsprognose mit einfließen. Die Werte werden lediglich für jedes transportierte Paket protokolliert.

Die Laderaum-Klasse enthält die drei privaten Attribute `laderoute`, `entladen` und `svw`. Mit `laderoute` wird die Reihenfolge bezeichnet, in der die Pakete ausgeliefert werden sollen. Die Liste `entladen` enthält hingegen alle Pakete, die bereits entladen worden sind. Diese sind also nicht mehr im Laderaum vorhanden. Der Zeiger `svw` zeigt auf ein Stadtverwaltungsklassenobjekt. Die Stadtverwaltung entstammt dem Stadt-CGI und wurde in Abschnitt 5.2.3 auf Seite 45 bereits vorgestellt. Die verwendete Datenbank ist dieselbe, die auch im Stadt-CGI und im TSPserver-CGI verwendet wird. Dadurch wird der Datenaustausch zwischen den Einheiten mittels Referenznummern möglich.

Die Route-Klasse enthält eine Liste mit Paketen. Von diesen Paketen wird von der Route-Klasse aber lediglich die Ziel-Information benötigt, wo also die Pakete abgeliefert werden sollen. Die Pakete selbst sind für die Route-Klasse nicht von Bedeutung. Für die Laderaum-Klasse ergibt sich jedoch ein

Vorteil bei der Verwaltung der Pakete, weil keine separate Liste geführt werden muss, die nach einer Änderung der Route anschließend synchronisiert werden müsste.

Zu den Aufgaben der Route-Klasse zählt die Verwaltung der Paketliste, deren Neu-Anordnung in einer vorgegebenen Reihenfolge sowie die Optimierung der Liste nach Qualitätskriterien. Zur Veranschaulichung der Funktionen `ordnen()` und `optimize_quality()` wird das Beispiel aus Abbildung 5.13 auf Seite 54 noch einmal herangezogen. Eine Anfrage beim TSP-Server wird von der Basis-Klasse mit der Funktion `routerequest()` ausgelöst.

Die Pakete in der ursprünglichen Liste befinden sich in der Reihenfolge `1,25,16,3,24,4`. Ein solcher Routestring kann mit der `getRoutestring()`-Funktion der Route-Klasse erzeugt werden. Da die Basis-Klasse nicht direkt auf die Route zugreifen kann, besitzt die Laderaum-Klasse ebenfalls eine Funktion mit diesem Namen, die den Aufruf allerdings unverändert weiterreicht. Die Antwort des Tspserver-CGIs ist der in Abbildung 5.13 dargestellte Routestring `1,3,4,16,24,25`. Dieser Routestring wird an die Funktion `setPaketreihenfolge()` übergeben, die ihrerseits die Funktion `ordnen()` aufruft. Bei der Neuanordnung wird zunächst eine neue, leere Paket-Liste angelegt. Anschließend wird mit Hilfe der privaten Funktion `findanderase()` der Reihe nach der Routestring abgearbeitet. Dabei wird die Referenznummer der Stadt in der ursprünglichen Liste gesucht. Das gefundene Paket wird an die neue Liste angehängt und aus der ursprünglichen Liste entfernt.

Die Optimierungsfunktion `optimize_quality()` optimiert die Route, in dem die Haltbarkeiten zum Lieferzeitpunkt ausgewertet werden. Dabei wird die Tatsache ausgenutzt, dass es sich bei der Route um eine Rundreise handelt. Die erste Stadt der Route entspricht der letzten Stadt, bzw. es wird nach der letzten Stadt zur ersten Stadt zurück gekehrt. Das Fahrzeug befindet sich zum Zeitpunkt der Anforderung nicht innerhalb der Route. Die Antwort des TSP-Servers lässt dem anfordernden Fahrzeug hier zwei Freiheitsgrade. Der erste Freiheitsgrad besteht in der Wahl der Anfangsstadt, mit welcher Stadt also begonnen wird, dem Kreis zu folgen. Die Route in der Reihenfolge `1,3,4,16,24,25` hat die gleiche Gesamtentfernung wie die Route in der Reihenfolge `3,4,16,24,25,1` oder `4,16,24,25,1,3`. Der zweite Freiheitsgrad ist die Umkehrung der Reihenfolge. Wenn dem Kreis in umgekehrter Reihenfolge gefolgt wird, z.B. `25,24,16,4,3,1`, bleibt die Gesamtentfernung ebenfalls unverändert. Für N Städte gibt es also $2 \cdot N$ verschiedene Möglichkeiten, diesen Vorschlag auszuführen.

Die verschiedenen Vorschläge unterscheiden sich in der Entfernung von der aktuellen Position des Fahrzeugs zur Anfangsstadt des Kreises. Wenn Qualitätskriterien berücksichtigt werden, gibt es weitere Unterscheidungsmerkmale. Die Optimierungsfunktion `optimize_quality()` erzeugt zur Untersuchung aller Varianten des Vorschlags für jede Variante je eine Kopie der aktuellen Route und modifiziert die Kopie der Liste mit Hilfe von Schiebe- und Spiegel-Operationen. Dafür stehen die Funktionen `shift_it()` und `mirror_it()` zur Verfügung. Die Paket-Liste ist innerhalb der Route-Klasse als `list<Paket*>`, also als Liste von Zeigern auf die Pakete, gespeichert. Die Pakete selbst müssen also nicht kopiert werden. Dieser Vorgang geht daher relativ schnell. Anschließend wird die Memberfunktion `minSL_at_delivery()` der Kopie aufgerufen.

Die Funktion `minSL_at_delivery()` durchläuft die gesamte Route, beginnend an der aktuellen Position des Fahrzeugs. Bei jeder Zwischenstation wird die Shelflife-Prognose für das dort abzuliefernde Paket berechnet. Dabei wird eine konstante Durchschnittsgeschwindigkeit von 80 $\frac{km}{h}$ angenommen. Diese Geschwindigkeitsannahme wird in der Datei `defs.h` von der Präprozessorkonstante `LKWNORMAL` festgelegt und lässt sich dort bei Bedarf verändern. Auf diese Weise findet die Funktion den niedrigsten Shelflife-Wert eines Pakets zum Zeitpunkt seiner Ablieferung.

Nachdem alle Routenvarianten auf diese Weise ausgewertet wurden, besitzt jede Routenvariante einen individuellen niedrigsten Shelflife-Wert eines Pakets. Diese Werte werden miteinander verglichen. Der höchste dieser Werte entspricht derjenigen Route, bei der das Risiko am geringsten ist, dass die Ware am Zielort bereits verdorben ist. Diese Route wird anschließend als beste Route an die Laderaum-Klasse zurückgegeben, von der der Optimierungsvorgang initiiert wurde. Alle anderen Kopien werden wieder aus dem Speicher entfernt.

Dieser Ansatz zum Finden der optimalen Route unterscheidet sich von der ursprünglichen Java-Simulation, die zu Beginn des Kapitels auf Seite 39 erwähnt wurde. Dort wurde mit einem Punktebewertungssystem gearbeitet, das Routen je nach Qualität unterschiedlich gewichtet. Für die Gewichtungsfunktion gab es dort mehrere Varianten, die mit einem als „`zielType`“ bezeichneten Buchstaben voneinander unterschiedern werden konnten. Alle Funktionen wurden in der Simulation ausgewertet, so dass sie miteinander verglichen werden konnten. Der hier verwendete Ansatz ist vergleichbar mit dem `zielType='m'`, wobei im Java-Code nicht nur die minimalen positiven Werte `minPos`, sondern auch die maximalen negativen Werte `maxNeg` mit berücksichtigt wurden. Eine getrennte Betrachtung von negativen und positiven Werten wurde bei der Implementierung von `optimize_quality()` nicht als sinnvoll erachtet.

Eine weitere Variante, die Einstiegsstadt zum Folgen der Route zu finden, besteht im Aufruf der Funktion `getNaechstgelegene_Stadt_aufRoute()`. Ausgehend vom momentanen Aufenthaltsort, der über die Funktion `setReferenzstadt()` gesetzt und in der Variable `hier` gespeichert wird, wird die jeweilge Entfernung dieser Stadt zu allen in der Route enthaltenen Städten miteinander verglichen. Diejenige Stadt mit der geringsten Entfernung wird dadurch ausgewählt. In der aktuellen Programmversion wird diese Funktion jedoch nicht mehr verwendet.

Die Laderaum-Klasse verwaltet die Pakete, die im Laderaum des Transportmittels enthalten sind. Pakete können mit der Funktion `add_ladung()` in den Laderaum eingeladen werden. Jedes Paket besteht aus einer Kombination von Ware und Lieferadresse. Statt jedes Paket einzeln hinzuzufügen, kann auch eine Auftragsdatei geladen werden, in der die benötigten Informationen enthalten sind. Die Funktion `auftrag_laden()` lädt die Datei. Die Datei besitzt ein ähnliches Format wie die Stadt-Koordinaten-Datei, die in Abschnitt 5.2.3 auf Seite 45 vorgestellt wurde. Ein Beispiel für eine Auftragsdatei ist im Anhang in Listing A.24 auf Seite 115 dargestellt.

Die Datei ist zeilenweise aufgebaut. Jede Zeile besteht aus drei Spalten und enthält die Informationen für ein Paket. Die Spalten sind durch ein oder mehrere Whitespace-Zeichen getrennt. Leerzeilen sind erlaubt. Die erste Spalte enthält die Warenbezeichnung, die zweite Spalte einen Zahlenwert für die verbleibende Haltbarkeit(Shelflife) in Stunden. Die dritte Spalte enthält den Namen oder die Referenznummer der Stadt, in der das Paket abgeliefert werden soll. Der Name muss dabei exakt mit der Angabe aus der Stadt-Koordinaten-Datei übereinstimmen. Die Warenbezeichnung darf nur aus einem Wort bestehen. Auf Sonderzeichen wie ä, ö, ü, ß sollte verzichtet werden, um Probleme mit den unterschiedlichen Zeichensätzen von Windows und Linux zu vermeiden. Die Datei kann am Anfang der Datei Kommentarzeilen enthalten. Diese werden mit einem `#`-Symbol am Zeilenanfang gekennzeichnet. Die Zeilenlänge sollte 80 Zeichen nicht überschreiten. Innerhalb der Datei sind keine weiteren Kommentare erlaubt.

Für den Fall, dass keine Auftragsdatei vorliegt, kann für einfache Testläufe auch die Funktion `beispielladung()` verwendet werden. Dadurch werden vier Beispiel-Pakete generiert. Wenn die Auftragsdatei nicht gelesen werden kann, wird automatisch von der Basis-Klasse die Beispielladung verwendet. So wird vermieden, dass der Laderaum im Fehlerfall leer ist.

Das Gegenstück zu `add_ladung()` ist die Funktion `entladung()`. Dadurch wird ein Paket aus dem Laderaum entfernt. Die Funktion ruft intern die Memberfunktion `popFirstPaket()` der Klasse Route auf. Dadurch wird das Paket aus der aktuellen Route entfernt. Die `entladung()`-Funktion hängt das entnommene Paket an die interne `entladen`-Liste an, um auch später noch auf die Daten des entladenen Pakets zugreifen zu können. Bei der Entladung wird die Funktion `fixieredatalogger()` des Paket-Objekts aufgerufen. Dadurch wird von dem Thistorie-Objekt, auf das der `datalogger`-Zeiger zeigt, eine Kopie erstellt und somit die dynamische Verlinkung mit dem Sensorknoten gelöst.

Der Entladevorgang wird von der Basis-Klasse ausgelöst. Unmittelbar nach der Entladung wird die Funktion `erstelle_lieferschein()` aufgerufen. Diese Funktion sorgt dafür, dass sämtliche Datenfelder des Pakets in einer Textdatei abgespeichert werden. Die Datei wird im Unterverzeichnis `lieferscheine/` abgelegt. Der Dateiname ergibt sich aus dem Lieferdatum, dem Paketinhalt und der Zieladresse, z.B. `2009-12-15 Kiel_Avocado.txt`. Zur Erzeugung des Lieferscheins wird die Memberfunktion `dumptofile()` des Paket-Objekts aufgerufen. Ein Beispiel für einen Lieferschein befindet sich im Anhang auf Seite 117 in Listing A.26.

Die Basis-Klasse besitzt vier noch nicht erwähnte Funktionen: `starten()`, `fahren()`, `stoerung()` und `zuordnung_ladung_sensoren()`. Die Funktion `starten()` steuert den eigentlichen Programmablauf. Dieser gliedert sich in vier Phasen: Initialisierung, Fahren, Entladen und Ende. Die Phasen Fahren und Entladen werden der Paketanzahl entsprechend oft wiederholt. Der Programmablauf wird im nächsten Abschnitt 5.3.7 an einem Beispiel erläutert. Die Funktion `fahren()` bewirkt die Bewegung des Fahrzeugs von einer Stadt zur nächsten. Die Fahrzeit ergibt sich aus der Entfernung dieser Städte zueinander und einer angenommenen konstanten Fahrgeschwindigkeit des Fahrzeugs. Während der Simulation wird lediglich auf das Verstreichen der Fahrzeit gewartet. Währenddessen können Tosmac-Nachrichten von den Sensorknoten empfangen und verarbeitet werden.

Die Funktion `stoerung()` erzeugt eine künstliche Störung während der Fahrt, die die Fahrzeit verlängert. Dies kann beispielsweise ein Stau oder ein Defekt am Fahrzeug sein. Durch diese ungeplante Verzögerung verringern sich die Resthaltbarkeitswerte der Pakete. Nach dem Ende der Störung wird automatisch ein neuer Routenvorschlag mit den nun veränderten Voraussetzungen angefordert. Das Beispiel im nächsten Abschnitt wird die Auswirkungen der Störungen verdeutlichen.

Die Funktion `zuordnung_ladung_sensoren()` führt ein Matching von Sensorknoten und Ware durch. In der aktuellen Version wird einfach der Reihe nach jedem Paket ein anderer Sensorknoten zugeordnet. An dieser Stelle sollte allerdings zukünftig noch eine bessere geeignete Strategie implementiert werden, auf welche Weise die Sensorknoten der Ware zugeordnet werden können. Ein möglicher Ansatz kann die automatische Lokalisierung der Sensorknoten sein. Oder die Sensorknotenzuordnung wird schon in der Auftragsdatei fest vorgegeben. Der Fahrer wäre dann dafür zu ständig, dafür zu sorgen, dass sich die dort angegegebenen Node-IDs tatsächlich in der Nähe der zu beobachtenden Wareneinheiten befinden.

5.3.7. Programm-Ablauf Imote2Basis

Nachdem die Basis und alle darin enthaltenen Objekte initialisiert sind, kann die Funktion `starten()` aufgerufen werden, die den eigentlichen Programmablauf steuert. Als erstes wird die Auftragsdatei geladen. Der Dateiname der Auftragsdatei ist in der Headerdatei `defs.h` definiert. Falls die Datei nicht geöffent werden kann, wird die Beispielladung geladen.

Die im Sensornetz enthaltenen Sensorknoten werden danach den Paketen zugeordnet. Dazu wird die in der Funktion `zuordnung_ladung_sensoren()` implementierte Strategie für die Zuordnung angewendet. Die Pakete befinden sich zu Beginn in der Reihenfolge, die durch die Auftragsdatei vorgegeben ist. Die Route-Funktion erzeugt mit der `getRoutestring()`-Funktion eine kommaseparierte Liste der Referenznummern der Zielstädte. Dieser Routestring wird mit der Funktion `routerequest()` als Anfrage an das Tspserver-CGI des Webservers gesendet. Die URL des Servers wird ebenfalls in der `defs.h`-Datei festgelegt. Hier ist die IP-Adresse in der Präprozessorkonstanten `TSPSERVER_IP` entsprechend anzupassen.

Die Antwort des Tspservers ist ebenfalls ein Routestring. Dieser wird nun an die Laderaum-Klasse mit der Funktion `setPaketreihenfolge()` zurückgegeben. Das bewirkt, dass die Route an den Vorschlag des Tspservers angepasst wird. Die anschließende Optimierung der Route nach Qualitätskriterien wird von der `setPaketreihenfolge()`-Funktion ebenfalls veranlasst. Die Pakete befinden sich danach in der Reihenfolge, in der sie auszuliefern sind. Mit der Funktion `laderaum->print()` lässt sich der Inhalt des Laderaums ausgeben. Hierbei werden alle Pakete in der Reihenfolge der geplanten Auslieferung ausgegeben. Jedes Paket ist zusammen mit seiner Zielstadt, seinem aktuellen Shelflife-Wert und der NodeID des überwachenden Sensorknotens in dieser Liste aufgeführt.

Aus dem aktuellen Aufenthaltsort und den Koordinaten der ersten Stadt der Route ergibt sich eine Entfernung `dist`. Die Funktion `fahren()` wird mit dieser Entfernung als Parameter aufgerufen. Bei einer angenommen konstanten Geschwindigkeit des Fahrzeugs ergibt sich daraus nach der Formel $v = \frac{s}{t}$ eine bestimmte Fahrzeit. Durch Addition zur Abfahrtzeit lässt sich die Ankunftszeit bestimmen. Die Zeit-Klasse kommt zum Einsatz, es wird also die beschleunigte Systemzeit (siehe Abschnitt 5.3.5) verwendet, so dass nicht in Echtzeit auf die Ankunft des Fahrzeugs gewartet werden muss. Während der Phase des Fahrens wird die Tosmac-Schnittstelle kontinuierlich abgefragt.

Dank des nicht blockierenden Treiberzugriffs lässt sich der Timeout-Wert auf die Fahrzeit abstimmen. Selbst wenn keine Pakete empfangen werden, wird das Ziel (annähernd) zur vorgesehenen Zeit erreicht. Die Ungenauigkeit zwischen errechneter Ankunftszeit und tatsächlicher Ankunftszeit ergibt sich daraus, dass bei einer 200-fachen Beschleunigung der Zeit 200 Sekunden der simulierten Zeit auf eine Sekunde Echtzeit abgebildet werden. Der Timeout-Wert der Tosmac-Klasse wird jedoch in ganzzahligen Sekunden angegeben. Dadurch kann nicht jeder gewünschte Zeitpunkt eingestellt werden. Dieses Verhalten ist jedoch kein gravierender Nachteil, da in echten Lieferprozessen in der Praxis ein angekündigter Zeitpunkt ebenfalls nicht auf die Sekunde genau erreicht wird.

In der Headerdatei `defs.h` lässt sich über Präprozessorkonstanten die Anzeige von zusätzlichen Statusmeldungen einschalten. Die Präprozessorkonstante `INTERVALLMELDUNGEN` bewirkt, dass während der Fahrzeit ein Zwischenstand ausgegeben wird. Es wird dann angezeigt, wieviel von der Fahrzeit bereits abgelaufen ist und wie lange noch gefahren werden muss. Mit der Präprozessorkonstante `DEBUG_TOSMAC_PAKETE` lässt sich jedes empfangene Tosmac-Paket direkt auf dem Bildschirm ausgeben. Zu gunsten der Übersichtlichkeit wurden diese beiden Konstanten jedoch für das Listing A.27 nicht gesetzt.

Nachdem die Fahrzeit abgelaufen ist, ist der Zielort erreicht. Der Stadtzeiger `aufenthaltsort` wird auf den Zielort gerichtet. Anschließend werden alle Shelflife-Werte aktualisiert, in dem die verstrichene Fahrzeit von den Resthaltbarkeiten abgezogen wird. Dann wird das Paket entladen und der Lieferschein erstellt. Im Anhang auf Seite 117 befindet sich ein Lieferschein für die Stadt Kiel als Beispiel. Danach wird das Paket gelöscht und der Speicher somit freigegeben. Das Paket wird im weiteren Verlauf des Programms nicht mehr benötigt. Die Anzahl der Pakete reduziert sich durch die Entladung um eins.

Die Schleife wird mit dem nächsten Paket auf die gleiche Weise fortgesetzt. Sie wiederholt sich so lange, bis die Anzahl der Pakete Null erreicht.

Nach der dritten Stadt wird eine künstliche Störung erzeugt, die die Fahrzeit verlängert. Dies kann beispielsweise ein Stau oder ein Defekt am Fahrzeug sein. In der Funktion `stoerung()` ist eine Verzögerung von 3 Stunden eingestellt. Eine halbe Stunde entspricht bei einer 200-fachen Beschleunigung $\frac{30 \cdot 60}{200} = 9$ Sekunden. Der Programmablauf wird also für gut eine Minute angehalten. Durch diese ungeplante Verzögerung verringern sich die Shelflife-Werte der Pakete. Zu beachten ist, dass bei dieser Art Störung die Qualitätswerte der Waren nur *indirekt*, nämlich durch die verlängerte Fahrzeit, betroffen sind. Ein Fehler des Kühlaggregats würde sich hingegen *direkt* auf die Warenqualität auswirken. Die Abhängigkeit der Qualität vom Temperaturverlauf wird jedoch in der vorliegenden Programmversion grundsätzlich vernachlässigt, weil die entsprechenden Modelle noch nicht implementiert sind. Bei einer späteren Erweiterung des Programms sollten andere, darauf abgestimmte Stör-Szenarien implementiert werden.

Nachdem sich der Stau aufgelöst hat und das Fahrzeug wieder mit seiner konstanten Geschwindigkeit weiterfahren kann, wird zunächst eine neue Route angefordert. Der Tspserver wird mit der derzeit geplanten Route konfrontiert und um einen neuen Vorschlag gebeten. Anschließend wird die lokale Optimierung durchgeführt, bei der auch der Qualitätszustand der Pakete berücksichtigt wird. Das Prinzip ist das gleiche wie schon beim Beginn der Fahrt in der Initialisierungsphase. Der Ablauf der Optimierung wurde im vorangegangenen Abschnitt 5.3.6 erläutert.

Die Ausgabe eines Programmablaufs ist im Anhang ab Seite 118 in Listing A.27 dargestellt. Es wurde die Auftragsdatei aus Listing A.24 auf Seite 115 mit 20 Paketen verwendet. Die Ausgabe wurde auf vier Seiten gekürzt. Die Laderaumauflistung wurde jeweils soweit gekürzt, dass redundante Informationen entfernt wurden. Weiterhin wurde die Formatierung leicht verändert, so dass die Programmausgabe etwas kompakter geworden ist.

Das Programm beginnt mit dem Laden des Auftrags, dem Zuordnen der Sensorknoten und dem anschließenden Neu-Anordnen der Pakete. Es sind drei Zeilen mit kommaseparierten Listen der Referenznummern der Zielstädte der Pakete angegeben. Die erste Zeile enthält die unveränderte Reihenfolge. Diese Liste wurde als Anfrage an den Tspserver geschickt. Seine Antwort ist in der folgenden Zeile dargestellt. Mit Hilfe der lokalen Optimierung unter Berücksichtigung der Qualitätskriterien wurde die Reihenfolge in der dritten Zeile als beste Lösung ausgewählt.

Es ergibt sich eine zu fahrende Gesamtstrecke von 1922 km. Um die Entfernung besser nachvollziehen zu können, kann das Tspserver-CGI mit dem Routestring und dem Parameter `optimize=false` über den Browser aufgerufen werden. In Abbildung 5.21 auf der nächsten Seite ist die Ausgabe des Tspserver-CGI-Programms für die unsortierte Original-Reihenfolge dargestellt. Die Streckenlänge der Original-Reihenfolge ist dort mit 4132 Kilometern abzulesen. Die zu fahrende Strecke reduziert sich also um einen Faktor von über 50 Prozent!

Nach Fahrtbeginn wird der Laderaum aufgelistet. Jedes der 20 Pakete ist mit seiner Bezeichnung des Inhalts, seiner Haltbarkeit in Stunden und mit Datumsangabe sowie seinem vorgesehenen Zielort aufgeführt. Die letzte Spalte enthält die Node-ID des Sensorknotens, der diesem Paket zu Beginn zugeordnet wurde. Die Zielstädte sind jeweils mit Referenznummer und Name gekennzeichnet. Dadurch lässt sich der Routestring besser nachvollziehen. Die Referenznummern in der Zielstadt-Spalte von oben nach unten gelesen entsprechen genau der kommaseparierten Liste des Routestrings. Zur besseren Orientierung ist am Ende der Liste der Routestring nochmals aufgeführt. Dadurch wird es möglich, Einzelschritte immer wieder im Browser nachzuvollziehen.

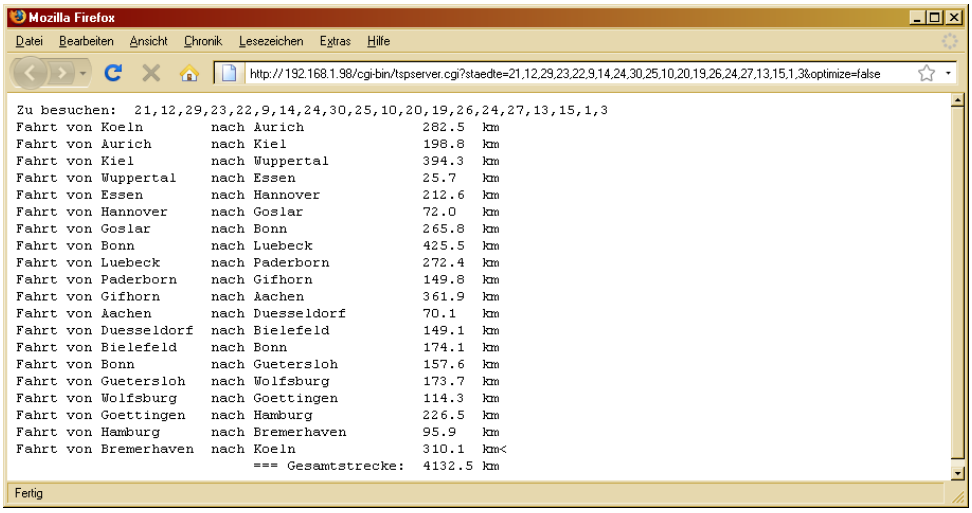

Abbildung 5.21.: Ausgabe des Tspserver-CGI für die Original-Reihenfolge der Städte

Die erste Etappe beginnt in der definierten Startstadt (Bremen) und endet in Hannover. Die Abfahrtzeit ergibt sich aus der Systemzeit. Die Ankunftzeit ergibt sich aus der Entfernung und der angenommenen Durchschnittsgeschwindigkeit. Statt um 12:08 kommt der LKW allerdings erst um 12:10 an. Die Begründung für diese Ungenauigkeit wurde bereits auf Seite 69 erwähnt. Die Entladung beginnt und der Lieferschein wird erzeugt. Der Lieferschein wird direkt in der entsprechenden Datei abgelegt und nicht auf dem Bildschirm ausgegeben. Der Laderaum enthält nun nur noch 19 Waren. Die Fahrt geht weiter nach Aurich und dann nach Bremerhaven.

Nachdem das Paket in Bremerhaven abgeliefert wurde, tritt die simulierte Störung auf. Die Weiterfahrt verzögert sich um 3 Stunden. Dies hat Auswirkungen auf die Warenqualität, wie sich aus dem Vergleich der Laderaumauflistung vor und nach der Störung ergibt. Das Transportmittel entscheidet sich, die geplante Route in Frage zu stellen und fordert einen neuen Vorschlag vom Tspserver an. Statt der ursprünglich geplanten Route „Hamburg, Kiel, Lübeck...“ lautet die neue Route nun „Kiel, Lübeck, Wolfsburg...“. Die Stadt Hamburg wird erst zum Schluss angefahren, weil die Haltbarkeit mit 74 Stunden groß genug ist. Priorität für die Auslieferung hat hingegen das Paket mit den Avocados, die für Kiel bestimmt sind.

A. Anhang

Listing A.1: Hilfsfunktionen zur Anzeige von Tosmac-Paketen, Datei `hilfsfunktionen.c`

```
#include <stdio.h>
#include "hilfsfunktionen.h"
#include "tosmac.h"

/*
 * gibt die Bytes des Buffers von Beginn bis Ende-1 aus.
 * Es werden drei verschiedene Darstellungen verwendet
 */
void printbuf (char *buf, int beginn, int ende) {
  int j;
  char c;

  printf ("␣.hex=[");
  for (j = beginn; j < ende; j++) printf ("␣%02x␣", buf[j]);
  printf("]\n␣.num=[");
  for (j = beginn; j < ende; j++) printf ("%3d,", buf[j]);
  printf ("]\n␣.str=\"");
  for (j = beginn; j < ende; j++){
      c=buf[j];
      if(c<32||c>127)c='^';
      printf ("__%c␣", c);
  }
  printf ("\"\nindex:␣");
  for (j = beginn; j < ende; j++) printf ("␣%2d␣", j);
  puts("\n");
}


/*
 * gibt den Header aus, Bytes 0-11
 */
void printheader (char *buf) {
  printf ("HEADER:␣\n");
  printbuf (buf, 0, 12);
}


/*
 * gibt den Inhalt aus, Bytes 12 bis anzahl_gelesen-1
 */
void printcontent (char *buf, int anzgel){
  printf ("CONTENT:␣\n");
  printbuf (buf, 12, anzgel);
}
```

```
/*
 * wertet die Header-Felder aus.
 * Die Bedeutung mancher Felder ist unklar
 */
void printheaderdata (char *buf) {
  printf ("-?-?-?\tSEQ\tDESTPAN/ADDR\tPAN/LocADDR\n");
  printf ("%02x%02x%02x\t", buf[0], buf[1], buf[2]);
  printf ("%02x\t", buf[3]);
  printf ("%02x%02x/%02x%02x\t", buf[5], buf[4], buf[7], buf[6]);
  printf ("%02x%02x/%02x%02x\n", buf[9], buf[8], buf[11], buf[10]);
}


/*
 * Gibt die Informationen des zu sendenden Pakets aus
 */
void printsendinfo (TOS_Msg send_pkt){
  int j;
  printf ("Length:%02d␣", send_pkt.length);
  printf ("Fcf:␣0x%02x%02x␣", send_pkt.fcfhi, send_pkt.fcflo);
  printf ("Seq#:%02x␣", send_pkt.dsn);
  printf ("DestPAN:%04x␣", send_pkt.destpan);
  printf ("DestAddr:%04x␣", send_pkt.addr);
  printf ("TypeID:%02x␣", (unsigned char) send_pkt.type);
  printf ("GroupID:%02x\n", (unsigned char) send_pkt.group);
  printf ("Data:␣\n");

  for (j = 0; j < send_pkt.length; j++)
    printf ("␣%02x␣", (unsigned char) send_pkt.data[j]);
  printf ("\n");

  for (j = 0; j < send_pkt.length; j++)
    printf ("%03d␣", (unsigned char) send_pkt.data[j]);
  printf ("\n");
}
```

Listing A.2: Insert Nearest Neighbour Algorithmus der Tsp-Klasse

```
/*
 * Insert Nearest Neighbour Approach.
 * Dieses heuristische Verfahren liefert eine gute Loesung fuer das
 * Traveling Salesman Problem. Es ist aber nicht unbedingt die
 * optimale Loesung.
 */
list<Stadt> *Tsp::InsertNearest() {
        if (vorgabe->empty())
          return vorgabe;

        /* zubesuchen: Arbeitskopie der Vorgabe
         * route: neue Anordnung der Staedte mit optimierter
         *        Reihenfolge. wird per return zurueckgegeben
         */
        list<Stadt> *zubesuchen = new list<Stadt>(*vorgabe);
        list<Stadt> *route = new list<Stadt>;
        list<Stadt>::iterator vorgaenger;

        /* 2 Städte werden per Zufall als Startwert gewählt.
         * Bei mehr als 8 Städten insgesamt werden 3 Städte
         * als Startwert gewählt */
        int anz_ecken = 2;
        if (zubesuchen->size() > 8) anz_ecken = 3;
        while (anz_ecken--) { // fuer 2 Ecken 2x ausfuehren
          int zuf = zufallszahl(zubesuchen->size());
          list<Stadt>::iterator it = zubesuchen->begin();
          for (int i = 0; i < zuf; i++) it++;
          route->push_back(*(it)); // Stadt anhaengen
          zubesuchen->erase(it); // Stadt aus Arbeitskopie entfernen
        }

        /* Hier beginnt der eigentliche Algorithmus:
         * Versuche jede der verbleibenden Städte nacheinander
         * an alle möglichen Positionen in die bereits
         * bestehende Route einzufuegen */
        while (zubesuchen->size() > 0) {
          umweg umwegD, umwegK; // dieser und kuerzester Umweg
          umwegK.entfernung = 0; // Zuruecksetzen

          /* Wenn der Nachfolger das erste Element ist, ist der
           * Vorgaenger das letzte Element.
           * route->end() zeigt jedoch hinter das letzte Element,
           * so dass der Iterator danach dekrementiert werden muss.
           */
          umwegD.nachfolger = route->begin();
          vorgaenger = route->end();
```

```
        vorgaenger--;

        /* Die gesamte Route wird durchlaufen, um die optimale
         * Kombination aus Position und einzufuegender Stadt finden:
         */
         while (umwegD.nachfolger != route->end()) {

          double direkte_entfernung =
             vorgaenger->entfernung_zu(&(*(umwegD.nachfolger)));

          /* Alle noch einzufügenden Städte werden durchlaufen */
          for (umwegD.einzufuegende_stadt = zubesuchen->begin();
               umwegD.einzufuegende_stadt != zubesuchen->end();
               umwegD.einzufuegende_stadt++) {

            double entf_zu_vorgaenger = umwegD.einzufuegende_stadt
                ->entfernung_zu(&(*vorgaenger));

            double entf_zu_nachfolger = umwegD.einzufuegende_stadt
                ->entfernung_zu(&(*(umwegD.nachfolger)));

            umwegD.entfernung = entf_zu_vorgaenger
                                + entf_zu_nachfolger
                                - direkte_entfernung;

            /* falls es noch keinen kuerzesten Umweg gibt,
             * oder der kuerzeste Umweg groesser als der
             * aktuelle ist: Definierte diesen als kuerzesten
            */
            if ((!umwegK.entfernung) || (umwegK < umwegD))
              umwegK = umwegD;
          }/*for*/
          vorgaenger = umwegD.nachfolger;
          umwegD.nachfolger++;
        }/*while: wiederholen wenn noch nicht alles ausprobiert*/

        /* Stadt wird an der besten Position eingefuegt */
        route->insert(umwegK.nachfolger,
                      *(umwegK.einzufuegende_stadt));
        zubesuchen->erase(umwegK.einzufuegende_stadt);
      }/*while: wiederholen, falls noch Städte übrig*/
      delete zubesuchen; // Arbeitskopie loeschen
      return route;
}
```

Listing A.3: Headerdatei `Basis.h`

```
/*
 * Basis.h
 *
 *      Author: dhentschel
 */

#ifndef BASIS_H_
#define BASIS_H_

#include "Laderaum.h"
#include "../../stadtcgi/src/Stadt.h"
#include <list>
#include "Paket.h"
#include "Sensornetz.h"

#include <string>

class Basis {
private:
        Laderaum* laderaum;
        Stadt* aufenthaltsort;
        Sensornetz* wsn;
public:
        Basis();
        virtual ~Basis();

        void starten();
        void fahren(double entfernung);
        void stoerung(Stadt *s);

        void zuordnung_ladung_sensoren();
        char *routerequest(string);
};

#endif /* BASIS_H_ */
```

Listing A.4: Headerdatei `Cgiquery.h`

```
/*
 * Cgiquery.h
 *
 * Die CGI-Query Klasse ist speziell für CGI-Interfaces entwickelt
 * worden. Der Query_string wird automatisch ausgewertet.
 * Die Key-Value-Paare werden in einer Map abgelegt, so dass ein
 * bequemer Zugriff auf alle Key-Value-Paare erfolgen kann
 * Außerdem wird automatisch der Content-Type gesetzt und ggf.
 * ein HTML-Header und Footer erzeugt.
 *
 *      Author: dhentschel
 */

#ifndef CGIQUERY_H_
#define CGIQUERY_H_

#include <map>
#include <string>
#include <iostream>

#define CONTENT_HTML    (0)
#define CONTENT_PLAIN   (1)
#define CONTENT_UNKNOWN (2)
#define HTML_NORMAL             false
#define HTML_PREFORMATTED       true

using namespace std;

class Cgiquery {
private:
        bool pre;       // vorformatiert ja oder nein
        int contenttyp; // Plaintext oder HTML
        string title;   // Titel der HTML-Datei

        void init(unsigned int typ = 0);
        // Initialisierungsfunktion für Konstruktoren

        string defaultTitle(); // erzeugt einen HTML-Vorgabe-Titel

        void splitString(string, string&, string&);
        // teilt Schlüssel-Werte-Paare

public:
        map<string, string> kvmap;
        // Speicherung der QueryString-Parameter
```

```
        Cgiquery(unsigned int typ); // mit vorgegebenem Dokumententyp
        Cgiquery();

        void printHeader(bool pre); // HTML-Header ausgeben
        void printFooter(); // HTML Footer ausgeben

        void parseString(string);
        // wertet den QUERY_STRING aus (eigentlich private)

        void setContenttyp(int);
        int getContenttyp();

        void setTitle(string t);
};

#endif /*CGIQUERY_H_*/
```

Listing A.5: Headerdatei `Laderaum.h`

```
/*
 * Laderaum.h
 *
 * Die Laderaumklasse verwaltet die Ladung (Pakete)
 * im Laderaum. Dabei werden zwei Listen verwaltet.
 * Eine mit den noch abzuliefernden Paketen (in
 * einer bestimmten Reihenfolge, Route). Und eine
 * mit den bereits abgelieferten Paketen.
 *
 *      Author: dhentschel
 */


#ifndef LADERAUM_H_
#define LADERAUM_H_


#include <list>
#include "Ware.h"
#include "Paket.h"
#include "Route.h"
#include "defs.h"
#include "../../stadtcgi/src/Stadt.h"
#include "../../stadtcgi/src/Stadtverwaltung.h"
using namespace std;

class Laderaum {
private:
        Route *laderoute;       // Laderaum-Inhalt
        list<Paket*> *entladen; // entladene Pakete
        Stadtverwaltung *svw;

public:
        Laderaum();             // Konstruktor
        virtual ~Laderaum();    // Destruktor

        void add_ladung(Ware*, Stadt*);
        bool auftrag_laden(string filename);
        void beispielladung();

        Paket *entladung(); // entfernt das erste Paket (pop)
        Paket *erstesPaket(); // erstes Paket (ohne entfernen)
        Paket *naechstesPaket(); // zweites, drittes, viertes...

        string erstelle_lieferschein(Paket *p); //return=filename
```

```
        string getRoutestring(); // z.B. "12,28,2,14,6"

        void setPaketreihenfolge(Stadt *hier, char* route);
        // Reihenfolge durch einen Routestring vorgeben

        void print(); // Laderaum-Inhalt + Route ausgeben

        void update_shelflifes(); // Haltbarkeitsdaten aktualisieren
};

#endif /* LADERAUM_H_ */
```

Listing A.6: Headerdatei `Messwert.h`

```
/*
 * Messwert.h
 *
 * Ein Messwert besteht aus einem (oder mehreren)
 * physikalischen Werten, (d.h. Zahlenwert+Einheit)
 * und dem Zeitpunkt der Messwertaufnahme.
 *
 *      Author: dhentschel
 */

#ifndef MESSWERT_H_
#define MESSWERT_H_

#define MAXANZWERTE 5 /* Bei Bedarf entsprechend anpassen */


#include "Zeit.h"
#include "Wert.h"


class Messwert {
private:
        Wert value[MAXANZWERTE];
        /* ein Messwert kann aus mehreren (maximal MAXANZWERTE)
         * Werten bestehen. Die Werte werden mit setValue der
         * Reihe nach hinzugefuegt. getValue gibt per default
         * nur den ersten Wert zurueck. Als Parameter kann ein
         * Index angegeben werden. Alternativ kann die getValues
         * Funktion verwendet werden, die ein Werte Array
         * zurueckgibt. Der operator<< gibt alle Werte aus.
         */
        Zeit zeitpunkt;
        int valuei;     // interner index fuer setValue

public:
        // Konstruktoren
        Messwert();                 //value=0, zeitpunkt=jetzt
        Messwert(Wert w);           //value=w, zeitpunkt=jetzt
        Messwert(Wert w, Zeit &t); //value=w, zeitpunkt=t

        Messwert* setValue(Wert w);
        Wert      getValue();
        Wert      getValue(int n); //Wert mit index n
        Wert*     getValues();     //alle Werte als Array

        Messwert* setZeitpunkt(Zeit &t);
```

```
        Zeit       getZeitpunkt();

        int        getAnzahl();      //Anzahl Werte

        friend ostream &operator<<(ostream &stream, Messwert &ob);
};

ostream &operator<<(ostream &stream, Messwert &ob);

#endif /* MESSWERT_H_ */
```

Listing A.7: Headerdatei `Paket.h`

```
/*
 * Paket.h
 *
 * Das Paket enthält eine Ware, die für einen Zielort vorgesehen ist.
 *
 *      Author: dhentschel
 */

#ifndef PAKET_H_
#define PAKET_H_
#include "Ware.h"
#include "../../stadtcgi/src/Stadt.h"
#include "Thistorie.h"
#include "Sensornode.h"
#include <string>
#include <iostream>
#include "Zeit.h"
using namespace std;

class Paket {
        Ware       *ware;
        Stadt      *ziel;
        Sensornode *imote;              // Ueberwachender Node
        Thistorie  *datalogger;         // Messprotokoll
        Zeit       *beladen, *entladen; // Zeitstempel
public:
        Paket();
        Paket(const Paket&);

        Ware* getWare();        // Getter fuer Ware
        Ware* setWare(Ware* w);

        Stadt* getZiel();       // Getter fuer Stadt
        Stadt* setZiel(Stadt* s);

        Sensornode* getSensornode();    // Getter fuer imote2
        Sensornode* setSensornode(Sensornode* sn);

        Thistorie* getDatalogger();     // Getter fuer Protokoll

        int getNodeID();                // Getter fuer Node ID

        void fixiere_datalogger();      // Kopie von Thistorie

        void update_shelflife(); // Ware->update_shelflife()
```

```
        void print();
        void print(ostream& out);
        void printDatalogger(); // Datalogger->print()

        void dumptofile(string filename);//Lieferschein erstellen

        void abreise_jetzt(); // Zeitstempel setzen
        void ankunft_jetzt();
};

#endif /* PAKET_H_ */
```

Listing A.8: Headerdatei `Range.h`

```
/*
 * Range.h
 *
 * Diese Klasse wird benoetigt, um zu ueberpruefen, ob eine Zahl
 * innerhalb gewisser Schranken ist. Dies kann zum Beispiel
 * verwendet werden, wenn eine Ware in einem bestimmten
 * Temperaturfenster transportiert werden muss. Die aktuelle
 * Temperatur kann dann einfach als Parameter der is_inrange()-
 * Funktion uebergeben werden. Falls die Temperatur innerhalb
 * der Schranken liegt, ist die Rueckgabe true, sonst false.
 *
 *      Author: dhentschel
 */

#ifndef RANGE_H_
#define RANGE_H_

class Range {
private:
        int min;
        int max;
public:
        Range(); // Konstruktor
        Range(int min, int max);

        bool is_inrange(int val); // ueberprueft, ob wert in Range

        int getMin();
        void setMin(int min);

        int getMax();
        void setMax(int max);

        void setMinMax(int min,int max); // beides setzen
};

#endif /* RANGE_H_ */
```

Listing A.9: Headerdatei `Route.h`

```
/*
 * Route.h
 *
 * Verwaltet die Reihenfolge der noch nicht abgelieferten Pakete
 *
 *      Author: dhentschel
 */

#ifndef ROUTE_H_
#define ROUTE_H_

#include "../../stadtcgi/src/Stadt.h"
#include "Paket.h"
#include "Shelflife.h"
#include <list>

class Route {
private:
        list<Paket*> *staedte;          // Staedte bzw. Pakete
        list<Paket*>::iterator gsit;    // globaler Staedte-Iterator
        bool mirrored;  // true, falls gespiegelt, sonst false
        Stadt *hier;    // Refernzstadt

        Paket* findanderase(int ref);
        // Paket mit Referenznummer finden und entfernen

        Shelflife minSL_at_delivery(); // fuer Evaluierung
public:
        Route();                // Default Konstruktor
        Route(Route &kopie);    // Copy-Konstruktor
        virtual ~Route();       // Destruktor

        bool mirror_it();       // spiegelt die Listenreihenfolge
        Paket* shift_it();      // verschiebt erste Stadt ans Ende
        bool is_mirrored();     // Getter fuer mirrored-Flag

        int anzahl_staedte();   // wieviele Staedte enthaelt Route?

        double gesamt_fahrzeit(double speed);
        double gesamt_strecke();

        /* gibt eine nach Shelflife zum Lieferzeitpunkt
         * optimierte Route zurueck. Die urspruengliche
         * Route wird dabei jedoch nicht veraendert. */
        Route* optimize_quality();
```

```
        Stadt *getNaechstgelegene_Stadt_aufRoute();
        // kuerzeste Luftlinie von Referenzstadt

        Stadt *setReferenzstadt(Stadt *hier); // ausserhalb Route
        Stadt *getReferenzstadt();

        void ordnen(char* reihenfolge);
        // sortiert Elemente gemaess Routestring

        string getRoutestring();
        // Kommaseparierte Liste der Referenznummern der Staedte

        void append(Paket*);    // haengt ein Paket ans Ende an

        Stadt* getFirstStadt(); // gibt die erste Stadt zurueck
        Stadt* getNextStadt();  // zweite, dritte, vierte... Stadt

        Paket* popFirstPaket(); // return 1.Paket & entfernen (pop)
        Paket* getFirstPaket(); // return 1.Paket ohne entfernen
        Paket* getNextPaket();  // zweites,drittes, viertes... Paket
};

/* Prototypen fuer cout<< Operatoren */
ostream &operator<<(ostream &stream, Route &ob);
ostream &operator<<(ostream &stream, Route *ob);

#endif /* ROUTE_H_ */
```

Listing A.10: Headerdatei `Sensornetz.h`

```
/*
 * Sensornetz.h
 *
 *      Author: dhentschel
 */

#ifndef SENSORNETZ_H_
#define SENSORNETZ_H_

// Return-Values fuer listen() Funktion
#define RET_LISTEN_OK            1
#define RET_LISTEN_TIMEOUT       0
#define RET_LISTEN_UNKNOWN      -1
#define RET_LISTEN_ERROR        -2

#include <list>
#include <map>
#include "TosmacAccess.h"
#include "Sensornode.h"

using namespace std;

class Sensornetz {
private:
        list<Sensornode*> wsn; // Liste der Sensorknoten
        TosmacAccess *funk;    // physikalische Tosmac-Schnittstelle
        bool autoadd;          // neue Knoten automatisch hinzufuegen?
public:
        Sensornetz();   // Konstruktor
        virtual ~Sensornetz();  // Destruktor

        Sensornode* addNode(int nodeid); // neuen Node hinzufuegen
        Sensornode* getNode(int nodeid); // Node mit ID anfordern

        int* getNodeList();   // return: Array mit NodeIDs
        bool getAutoAdd();    // Getter fuer autoadd
        int getAnzahlNodes(); // Wieviele Nodes sind im Netz?

        bool isValidID(int nodeid); // Gueltigkeitsueberpruefung

        int listen(int timeout); // return-values siehe oben, #define
        void print();   // Aufruf print() fuer jeden Node
};

#endif /* SENSORNETZ_H_ */
```

Listing A.11: Headerdatei `Sensornode.h`

```
/*
 * Sensornode.h
 *
 * Diese Klasse repraesentiert einen Sensorknoten.
 * Jeder Sensorknoten ist eindeutig durch seine Node-ID
 * gekennzeichnet. Jede Sensorknotenklasse verarbeitet
 * nur Pakete, die mit dieser ID markiert sind.
 *
 *      Author: dhentschel
 */

#ifndef SENSORNODE_H_
#define SENSORNODE_H_
#include "Thistorie.h"
#include "Tosmacpaket.h"
#include <iostream>

/* erwarteter Paketaufbau. entspricht der Reihenfolge von
 * XsensorITS400Sleep (C#.NET)
 * jeweils 16bit-Werte. Headerbytes werden mitgezaehlt */

#define FELD_ACC_X      16 /* Beschleunigung x-Komponente */
#define FELD_ACC_Y      18 /* Beschleunigung y-Komponente */
#define FELD_ACC_Z      20 /* Beschleunigung z-Komponente */
#define FELD_TEMP1      22 /* Temperatur TI-Sensor        */
#define FELD_TEMP2      24 /* Temperatur Sensirion SHT15  */
#define FELD_HUMID      26 /* Feuchtigkeit SHT15  */
#define FELD_LIGHT      28 /* TAOS-Lichtsensor    */
#define FELD_ADC_1      30 /* AD-Wandler, Kanal 1 */
#define FELD_ADC_2      32 /* AD-Wandler, Kanal 2 */
#define FELD_ADC_3      34 /* AD-Wandler, Kanal 3 */

class Sensornode {
private:
        int nodeid;
        Thistorie *protokoll;
        Zeit* lastseen; // Zeitpunkt des letzten Funkkontakts
public:
        Sensornode();
        Sensornode(int id);
        Sensornode(const Sensornode &B);  // Copy Constructor
        virtual ~Sensornode();

        int getNodeID();
        void setNodeID(int nodeid);
```

```
        Thistorie* getDataLogger();       // Zeiger auf Protokoll
        Thistorie* getDataLoggerKopie(); // Zeiger auf Protokollkopie

        Zeit* getLastseen(); // Getter fuer lastseen

        Wert calculateFeld(Tosmacpaket* tp, int feld);
        // Formel zur Umrechnung des 16 Bit Werts in z.B. Temperatur
        // je nach Feldposition wird andere Formel verwendet

        int paketverarbeiten(Tosmacpaket*);
        // wenn die NodeID nicht uebereinstimmt, wird das Paket
        // ignoriert. Ansonsten wird Messwert erzeugt und an
        // Thistorie angehaengt.

        void print(); // Node ID und Messprotokoll ausgeben
};

#endif /* SENSORNODE_H_ */
```

Listing A.12: Headerdatei `Shelflife.h`

```
/*
 * Shelflife.h
 *
 *      Diese Klasse soll das Verhalten verschiedener Fruechte
 *      bei bestimmten Temperaturen modellieren.
 *      Dynamisches Verhalten bei bestimmten Temperaturen wird
 *      hierbei allerdings noch nicht beruecksichtigt. Derzeit
 *      wird ein fester Ablaufzeitpunkt vorgegeben.
 *
 *      Author: dhentschel
 */

#ifndef SHELFLIFE_H_
#define SHELFLIFE_H_

#define SHELFLIFE_DEFAULTVALUE 0

#include <iostream>
#include "Wert.h"
#include "Zeit.h"

using namespace std;

class Shelflife {
private:
        Zeit *ablaufzeitpunkt;
        Wert stunden;
        //double value;
        //double parameter;// muss noch entsprechend ausgebaut werden
        double subtract_fahrzeit(double stunden);

public:
        Shelflife();
        Shelflife(double);
        Shelflife(const Shelflife &kopie); // Copy Constructor

        double getValue();              // in Stunden
        void   setValue(double value);  // value in Stunden

        Zeit* getAblaufzeitpunkt();

        bool ist_abgelaufen();  // ablaufzeitpunkt->inVergangenheit

        void update();  // stunden aus ablaufzeitpunkt neu berechnen

        Shelflife predict(double stunden); // Zukunfts-Prognose
```

```
        double loss_per_day();  // derzeit const 24h
        double loss_per_hour(); // derzeit const 1h

        friend ostream &operator<<(ostream &stream, Shelflife &ob);

        bool operator< (Shelflife &other); // Vergleich
};

// Ostream-Operator fuer Ausgabe mit cout <<
ostream &operator<<(ostream &stream, Shelflife &ob);
#endif /* SHELFLIFE_H_ */
```

Listing A.13: Headerdatei `Stadt.h`

```
/*
 * Stadt.h (Stadt-CGI)
 *
 * Durch diese Klasse wird eine Stadt modelliert.
 * Jede Stadt ist durch ihre Koordinaten und durch den
 * Staedtenamen charakterisiert.
 *
 *      Author: dhentschel
 */

#ifndef STADT_H_
#define STADT_H_

#include <string>
#include <iostream>

#define STADTNAMENLAENGE 40
#define PI 3.1415926536
#define EQUATORIAL_RADIUS_KM 6378.137

using namespace std;

/* Hilfs-Klasse zur Clusterung der Koordinaten */
class koo {
public:
        double lon;        // Longitudinalkoordinate
        double lat;        // Lateralkoordinate
        koo Bogenmass() { // Umrechnung ins Bogenmass
                koo bm;
                bm.lon = this->lon / 180.0 * PI;
                bm.lat = this->lat / 180.0 * PI;
                return bm;
        }
};

ostream &operator<<(ostream &stream, koo &ob);

/* Die eigentliche Stadt-Klasse */
class Stadt {
private:
        koo           koordinaten; //siehe oben
        string        name;
        unsigned int referenznummer;
        /* Die Option opt_mitnr gibt an, ob die Stadt zusammen
         * mit der Nummer ausgegeben werden soll oder ohne Nr */
        bool opt_mitnr;
```

```
public:
        Stadt();                            // Konstruktor
        Stadt(char*N);                      // Konstruktor mit Name
        Stadt(const Stadt &vorlage);        // Copy-Konstruktor

        double entfernung_zu(Stadt *B);
        // Berechnet Entfernung zu einer anderen Stadt

        void print(); // Gibt den Namen der Stadt aus

        koo& getKoordinaten() const;
        void setKoordinaten(koo koordinaten);
        void setKoordinaten(double lon, double lat);

        const char *getName();
        void        setName(const char *newname);

        string getNameStr();

        unsigned int getReferenznummer() const;
        void         setReferenznummer(unsigned int referenznummer);

        void setopt_mitnr(bool m);

        friend ostream &operator<<(ostream &stream, Stadt &ob);
};

/* fuer die Ausgabe mit cout << */
ostream &operator<<(ostream &stream, Stadt &ob);
ostream &operator<<(ostream &stream, Stadt *ob);

#endif /* STADT_H_ */
```

Listing A.14: Headerdatei `Stadtverwaltung.h`

```
/*
 * Stadtverwaltung.h (Stadt-CGI)
 *
 * Die Stadtverwaltung verwaltet die Liste aller moeglichen
 * (definierten) Staedte. Die Staedte werden aus einer
 * Textdatei eingelesen, in der die Namen und Koordinaten
 * gespeichert sind. (z.B: Bremen 53N05 8E49)
 *
 *      Author: dhentschel
 */

#ifndef STADTVERWALTUNG_H_
#define STADTVERWALTUNG_H_

#include <iostream>
#include <list>
#include <stdio.h>
#include <stddef.h>
#include <iomanip>
#include <fstream>
#include <string.h>
#include "Stadt.h"

using namespace std;

#define STADTVERWALTUNG_FILENAME "staedte_koordinaten"

class Stadtverwaltung {
private:
        list<Stadt* staedte;        // interne Liste der Staedte
        list<Stadt*>::iterator it; // Iterator fuer nextstadt()
        Stadt *cache;   // interner Cache fuer wiederholte Anfragen

        double koocalc(string); // Grad und Minute verrechnen
        void reset_iterator();  // fuer nextstadt()
public:
        Stadtverwaltung(); // Konstruktor

        void ladeStaedteDatenbank(string); // Textdatei einlesen

        void printStaedteliste();  // Liste aller Staedte ausgeben
        void printDistList(int n); // Liste aller Entf. zu Stadt n

        double calculateDistance(int stadt1, int stadt2);
        // Berechnet die Entfernung zweier Staedte
```

```
        Stadt* firststadt();  // gibt erste Stadt zurueck
        Stadt* nextstadt();   // gibt zweite,dritte...Stadt zurueck

        bool isValid(int ref);// Gueltigkeitspruefung der Ref.nr

        int getAnzahl();      // Anzahl Staedte

        Stadt* operator[](unsigned int index); //Zugriff mit [ref]
        Stadt* operator[](string match); //Zugriff mit ["name"]
};

#endif /* STADTVERWALTUNG_H_ */
```

Listing A.15: Headerdatei `Thistorie.h`

```
/*
 * Thistorie.h
 *
 * Diese Klasse verwaltet eine Liste von (Temperatur-)Messwerten.
 * Diese Liste kann sowohl einem Sensorknoten als auch
 * einer Wareneinheit zugeordnet sein.
 *
 *      Author: dhentschel
 */

#ifndef THISTORIE_H_
#define THISTORIE_H_

#include <list>
#include <iostream>
#include "Messwert.h"

using namespace std;

class Thistorie {
private:
        list<Messwert> temperaturhistorie; // Liste der Messwerte
public:
        Thistorie();                    // Konstruktor
        Thistorie(const Thistorie&); // Copy Constructor

        void print();              // Gibt alles aus, Standardausgabe
        void print(ostream& out); // dito, an eine bestimmte Ausgabe

        void add_messwert(Messwert); // fuegt einen Messwert hinzu
        int getAnzahl();             // Anzahl der Messwerte
};

#endif /* THISTORIE_H_ */
```

Listing A.16: Headerdatei TosmacAccess.h

```
/*
 * TosmacAccess.h
 *
 * Klasse für den Zugriff auf die Tosmac-Schnittstelle.
 *
 * Author: dhentschel
 */

#ifndef TOSMACACCESS_H_
#define TOSMACACCESS_H_

#include "Tosmacpaket.h"


/* IMOTE2 wird automatisch von Eclipse definiert,
 * wenn Build-Configuration "Imote2" ausgewaehlt wurde.
 * Zusaetzliches #define von Hand setzen, damit Eclipse
 * die Syntax korrekt einfaerbt. (Bug)
 */
//#define IMOTE2

#ifndef IMOTE2  /* DUMMY-Klasse, wenn fuer PC kompiliert */

class TosmacAccess {
public:
        TosmacAccess(){};       // Tu nix
        void setChannel(int){}; // Tu nix
        Tosmacpaket *readData(){return (Tosmacpaket*)0;}; // return 0
        int waitforData(int timeout){return 0;} // return 0
        int getAnzahlBytes(){return 0;} // return 0
};

#else   /* richtige Klasse, wenn fuer Imote2 kompiliert */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include "tosmac.h"

#define IOCTL_GETVALUE 0x0001
#define DEFAULT_CHANNEL 15      /* siehe IMSAS-Channel-Liste! */

using namespace std;
```

```
class TosmacAccess {
private:
        int  drv;       // Treiber-Referenznummer
        int  channel;   // Kanalnummer
        int  anzgel;    // Rueckgabewert der read-/ioctl-Funktion
        char buf[128];  // Puffer fuer Paketdaten

public:
        TosmacAccess();                 // Konstruktor

        Tosmacpaket *readData();        // Paket von Treiber lesen

        int waitforData(int timeout); // Warten auf Daten

        void setChannel(int);           // Funk-Kanal einstellen

        int getAnzahlBytes();           // Getter fuer anzgel
};

#endif /*IMOTE2*/

#endif /*TOSMACACCESS_H_*/
```

Listing A.17: Headerdatei `Tosmacpaket.h`

```
/*
 * Tosmacpaket.h
 *
 * vereinfacht den Zugriff auf ein Datenpaket, das ueber die
 * TOSMAC-Schnittstelle empfangen wurde.
 *
 *      Author: dhentschel
 */

#ifndef TOSMACPAKET_H_
#define TOSMACPAKET_H_

#define FELD_NODID      12      /* Byteposition der Node-ID */
#define FELD_COUNT      14      /* Byteposition der Sequenznummer*/

class Tosmacpaket {
private:
        char buf[128];  // Paketdaten
        int  len;       // Laenge des Pakets
        int  bytestoword(int lo,int hi); // 8bit+8bit = 16bit

public:
        Tosmacpaket(char *, int); // Konstruktor mit char[]-Array

        int getLength();          // Getter fuer len

        int getNodeID();          // Getter fuer NodeID

        int getDaten8bit(int index);    // Daten von Byteposition
        int getDaten16bit(int loindex); // bytestoword(lo,lo+1)

        void print();
};

#endif /* TOSMACPAKET_H_ */
```

Listing A.18: Headerdatei `Tsp.h`

```
/*
 * Tsp.h (Tspserver-CGI)
 *
 * Eine Klasse zum Loesen eines Traveling Salesman Problems (TSP).
 * Dabei wird ein heuristischer Algorithmus verwendet.
 *
 *      Autor: dhentschel
 */

#ifndef TSP_H_
#define TSP_H_

#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <list>
#include <stdlib.h>
#include <time.h>

#include "../../stadtcgi/src/Stadtverwaltung.h"
#include "../../stadtcgi/src/Stadt.h"
#include "../../stadtcgi/src/Cgiquery.h"

#define STALI list<Stadt>       /* Vereinfachung fuer Schreibweise*/

/*
 * Eine kleine Hilfsklasse, um die zusammengehoerenden
 * Variablen zu buendeln und zu clustern.
 */
class umweg {
public:
        STALI::iterator einzufuegende_stadt;
        STALI::iterator nachfolger;
        double entfernung;
        bool operator<(const umweg &vergleich) {
                return entfernung > vergleich.entfernung;
        }
};

/* Ab hier beginnt erst die eigentliche Tsp-Klasse */
class Tsp {
private:
        bool   quiet;   // Ausgaben unterdruecken?

        STALI* vorgabe, *optimal;       // zwei Staedtelisten
```

```
        int    zufallszahl(int obergrenze); // rand() % obergrenze

        /* printList gibt eine der beiden Staedtelisten aus.
         * Städte werden inkl. ihrer Entfernungen zum Nachfolger
         * aufgelistet. return = Summe Gesamtentfernungen */
        double printList(char *prompt, STALI *liste);

        STALI* InsertNearest(); //Insert Nearest Neighbour Approach

public:
        Tsp();          // Konstruktor
        virtual ~Tsp(); // Destruktor

        void addStaedte(Stadtverwaltung *sv, string staedte);
        // staedteString: kommaseparierte Liste von Staedtenummern

        double optimizeRoute();   // optimiert mit InsertNearest

        void setQuietness();      // set quiet=true

        double printZubesuchen(); // printList(vorgabe)

        void printShortList(STALI *liste=NULL); //kommasep.Liste
};

#endif /* TSP_H_ */
```

Listing A.19: Headerdatei `Ware.h`

```
/*
 * Ware.h
 *
 * Diese Klasse repraesentiert eine Wareneinheit.
 * Z.B. Palette, Karton, Kiste...
 *
 *      Author: dhentschel
 */

#ifndef WARE_H_
#define WARE_H_

#include <string>
#include <iostream>
#include "Range.h"
#include "Shelflife.h"
#include "defs.h"

using namespace std;

class Ware {
private:
        string bezeichnung;       // Warenbezeichnung
        Range temperaturbereich; // Wird derzeit nicht ausgewertet
public:
        Shelflife haltbarkeit;   // public ist hier einfacher

        Ware(string);          // Konstruktor mit Warenbezeichnung
        Ware(string, double); // mit Warenbezeichnung und Shelflife
        Ware(Ware*);          // Copy Konstruktor mit Pointer
        Ware(const Ware&);    // Copy Konstruktor (normal)

        void print();         // Details ausgeben an Standardausgabe
        void print(ostream&); // Ausgabe an bestimmten Ausgabe-Strom

        void update_shelflife();

        const char* getBezeichnung();
        Ware*       setBezeichnung(string);

        Ware* setTemperaturlimits(int, int); // Setter fuer Range
};
ostream &operator<<(ostream &stream, Ware &ob);
ostream &operator<<(ostream &stream, Ware *ob);
#endif /* WARE_H_ */
```

Listing A.20: Headerdatei `Wert.h`

```
/*
 * Wert.h
 *
 * Diese Klasse speichert einen Wert mit zugehoeriger Einheit.
 * Ein ostream Operator erleichtert die Ausgabe.
 *
 *      Author: dhentschel
 */

#ifndef WERT_H_
#define WERT_H_
#include <string>
#include <ostream>

using namespace std;

class Wert {
private:
        double value;           // Zahlenwert
        string einheit;         // Einheit
        int nachkommastellen;   // ob und wieviele angezeigt werden

public:
        Wert();                 // Konstruktor
        Wert(double);           // Konstruktor mit Zahlenwert
        Wert(double, string);   // Konstruktor mit Zahlenwert+Einheit
        Wert(const Wert&);      // Copy Konstruktor

        double getValue();
        void   setValue(double);

        string getEinheit();
        void   setEinheit(string);

        void set(double, string);       // Setter fuer Wert + Einheit

        void setNachkommastellen(int); // Setter fuer Nachkommastellen

        string ToString();              // Wandelt Wert in String um

        // Rechenoperationen:
        Wert  operator+(const Wert&);  // Addition von zwei Werten
        Wert  operator+(const double&);// Addition von Wert + double

        Wert& operator+=(const Wert&);  //Addition mit anderem Wert
        Wert& operator+=(const double&);//Addition mit double
```

```
        Wert& operator-=(const Wert&);  //Subtraktion mit anderem Wert
        Wert& operator-=(const double&);//Subtraktion mit double

        // Vergleichsoperationen:
        bool  operator< (const Wert&);  // Vergleich, kleiner als
        bool  operator> (const Wert&);  // Vergleich, groesser als
        bool  operator==(const Wert&);  // Vergleich, gleich
};

/* ermoeglicht die Ausgabe mit cout<<...; ruft ToString() auf */
ostream &operator<<(ostream &stream, Wert &ob);

#endif /* WERT_H_ */
```

Listing A.21: Headerdatei `Zeit.h`

```
/*
 * Zeit.h
 *
 * Zeitklasse zur einfachen und einheitlichen Handhabung von Zeit-
 * stempeln und zur Realisierung einer beschleunigten Systemzeit.
 *
 *      Author: dhentschel
 */

#ifndef ZEIT_H_
#define ZEIT_H_

#include <string>
#include <time.h>
#include <ostream>

using namespace std;

class Zeit {
private:
        static time_t startzeit; // Statisch! ab Programmstart

        time_t value;   // eigentlicher Zeitwert
        char   str[27]; // Temporaeres Array fuer Umwandlungen
        Zeit*  temp;    // Zeiger auf temporaeres Tochter-Objekt

        char*  makeString(char* format); //erzeugt String
        Zeit&  set(time_t); // interner Setter fuer value
        time_t now();   // Jetztzeit in beschleunigter Dimension

public:
        Zeit();                 // Konstruktor
        Zeit(double rel);       // now() + rel in Stunden
        Zeit(const Zeit& kopie);// Copy Constructor
        ~Zeit();

        Zeit& addStunden(double std);   // fuege Stunden hinzu
        Zeit& addSekunden(double sek);  // fuege Sekunden hinzu

        bool in_vergangenheit();// Wenn Zeitpunkt schon vergangen ist
        bool in_zukunft();      // Wenn Zeitpunkt in Zukunft liegt

        char *hhmm();           // HH:MM
        char *hhmmss();         // HH:MM:SS
        char *mmss();           // MM:SS
```

```
        char *ttmmjjjjhh();     // TT.MM.JJJJ, HH Uhr
        char *ttmmhh();         // TT.MM, HH Uhr
        char *ISO8601();        // JJJJ-MM-DD
        char *zeitstempel();    // TT.MM.JJJJ HH:MM:SS

        /* nicht-permanente Tochterobjekte */
        Zeit &rest();     // Restzeit von jetzt bis Zeitpunkt
        Zeit &restReal(); // dito, aber NICHT-beschleunigt!
        Zeit &bisher();   // vergangene Zeit seit Zeitpunkt bis jetzt
        Zeit &jetzt();    // beschleunigte Systemzeit

        double Std();     // Umwandlung in Stunden (keine Ganzzahl)
        double Sek();     // Umwandlung in Sekunden (Ganzzahl)

        friend ostream& operator<<(ostream &stream, Zeit &ob);
};

ostream& operator<<(ostream &stream, Zeit &ob);

#endif /* ZEIT_H_ */
```

Listing A.22: Headerdatei `defs.h`

```
/*
 * defs.h
 *
 * Definitionen fuer verschiedene Programmteile,
 * die hier an einem zentralen Ort geaendert werden koennen.
 *
 *      Author: dhentschel
 */



#ifndef DEFS_H_
#define DEFS_H_


//#define IMOTE2          /* wird ggf. fuer tosmac benoetigt */
#define ACCELERATION 200 /* beschleunigt den Ablauf um Faktor 200 */
#define LKWNORMAL 80     /* Geschwindigkeit des Fahrzeugs */
#define LKWSPEED (LKWNORMAL*ACCELERATION) /* 80x200=16000 km/h */


/*
 * IP-Adresse der virtuellen Maschine, auf der der Apache Server
 * mit dem CGI-Programm "tspserver.cgi" laeuft.
 * Ueblicherweise ist es dort im Pfad /usr/lib/cgi-bin zu finden.
 * Der Tspserver berechnet die kuerzeste Strecke einer Route. */
#define TSPSERVER_IP "192.168.1.98"


/*
 * Intervall fuer die Anzeige der Restfahrzeit (in Sekunden)
 * wird in Basis::starten() verwendet. */
#define DRVTIME_INTERVAL (10*ACCELERATION)


/*
 * folgende Schalter aktivieren bei Bedarf
 * zusaetzliche Textausgaben. Zur Aktivierung // entfernen
 */
//#define STATISTISCHE_AUSWERTUNG /* Basis::fahren(), listen */
//#define INTERVALLMELDUNGEN      /* Basis::fahren() */
//#define DEBUG_TOSMAC_PAKETE     /* Tosmac-Empfang */
//#define DEBUG_ENTLADUNG         /* Entladung von Paketen */
//#define DEBUG_STAUMELDUNG       /* Zwischenstand beim Stau */
```

```
/*
 * Auftragsdatei wird in Basis::starten() der Laderaumklasse
 * uebergeben, und dort in Laderaum::auftrag_laden() geladen. */
#define AUFTRAGSDATEI "auftrag.txt"


/*
 * Anfangskoordinaten zur Bestimmung der Fahrzeit.
 * Hier wurden die Koordinaten von Bremen verwendet.
 * (beliebige Vorgabe)  */
#define STARTSTADT         "Startstadt"/*beliebiger Name*/
#define STARTKOORDINATEN  /*lon*/8.81667, /*lat*/53.0833 /*=Bremen*/


/*
 * Default-Werte fuer Range in Grad Celsius  */
#define DEFAULT_TEMP_MAX 100
#define DEFAULT_TEMP_MIN -100

#endif /* DEFS_H_ */
```

Listing A.23: Headerdatei `tosmac.h`

```
/*
 * TOS-MAC user interface
 * (Original-Headerdatei des Linux-Tosmac-Treibers)
 *
 * Driver is registered as a character device,
 * so normal system calls(open/close/read/write/ioctl)
 * are used to recv/trans data and configure radio device.
 */

#ifndef __TOSMAC_H
#define __TOSMAC_H
#include <linux/types.h>

#define TOSMAC_MAJOR    240
#define TOSMAC_DEVICE   "/dev/tosmac"

/*default vlaue of data lengh in packet*/
#define TOSH_DATA_LENGTH  28

/*
 * Packet structure
 * To be compatable with TinyOS, we provide a packet data structure,
 * which is the definition of packet used in TinyOS.
 * User program need to set up a packet before making system calls
 */
/* This packet definition may be changed as developing going on */
typedef struct __TOS_Msg{
        __u8 length;    // data length include header and footer
        __u8 fcfhi;
        __u8 fcflo;
        __u8 dsn;
        __u16 destpan;  // destPAN
        __u16 addr;        // destAddr
        __u8 type;
        __u8 group;
  __s8 data[TOSH_DATA_LENGTH]; //why completely different from
} TOS_Msg;

#define MSG_ERROR_0     -1
#define MSG_ERROR_1     -2

/*
 * Important:
 * TOSMAC driver is a non-standard character device.
 * Please read descriptions below for each function for details.
 */
```

```
/*
 * Open device
 * Use open( ) system call
 * Device can be opened in blocking or non-blocking mode
 * flags supported:
 *      O_RDWR                  // read/write in blocking mode
 *      O_RDWR | O_NONBLOCK     // read/write in non-blocking mode
 */
//int open(TOSMAC_DEVICE *device_path, int flags);

/*
 * Close device
 * Use close() system call
 */
//int close(int fd);

/*
 * Read data
 * Use read( ) system call.
 * Each time, one packet will be returned if there are data received.
 * void *buf is the pointer to a packet.
 * count should always be sizeof(TOS_Msg)
 * If user give a count value not equal with sizeof(TOS_Msg), then
 * an error code will return.
 * MSG_ERR_0: count is shorter than sizeof(TOS_Msg)
 * MSG_ERR_1: count is longer than sizeof(TOS_Msg)
 */
//ssize_t read(int fd, void *buf, size_t count);

/*
 * Write data
 * Use write( ) system call.
 * Each time, one packet will be sent. Do init of packet before
 * sending.
 * So buf is the pointer to a packet.
 * count should always be sizeof(TOS_Msg)
 * If user give a count value not equal with sizeof(TOS_Msg), then
 * an error code will return.
 * MSG_ERR_0: count is shorter than sizeof(TOS_Msg)
 * MSG_ERR_1: count is longer than sizeof(TOS_Msg)
 */
//ssize_t write(int fd, const void *buf, size_t count);

/*
 * Poll device
 * Use poll() or select() system call to poll device.
```

```
 */
// int select(int n, fd_set *readfds, fd_set *writefds,
//              fd_set *exceptfds, struct timeval *timeout);
// int poll(struct pollfd *ufds, nfds_t nfds, int timeout);

/*
 * Configure device
 * Use ioctl() system call.
 * cmd: the feature needs to be set, such as channel, power
 * arg: the value will be set to that feature if necessary
 */
//int ioctl(int fd, unsigned int cmd, unsigned int arg);

/*IOMAGIC for TOSMAC driver*/
#define TOSMAC_MAGIC   0xF4
/*Commands for IOCTL, more commands will be added*/
// set radio channel
#define TOSMAC_SET_CHANNEL    _IOW(TOSMAC_MAGIC, 1, __u16)
// switch receiver on/off
#define TOSMAC_SET_TRANS_ONOFF _IO(TOSMAC_MAGIC, 2, __u8)
// change default data length
#define TOSMAC_SET_DATALENGTH _IOW(TOSMAC_MAGIC, 3, __u16)

// MODIFIED BY DIRK:
#define TOSMAC_IOCTL_MAGIC 0xF4

#define TOSMAC_IOSETADDR        _IOW(TOSMAC_IOCTL_MAGIC, 0, __u16)
#define TOSMAC_IOGETADDR        _IO(TOSMAC_IOCTL_MAGIC,  1)
#define TOSMAC_IOAUTOACK        _IO(TOSMAC_IOCTL_MAGIC,  2)
#define TOSMAC_IODISAUTOACK     _IO(TOSMAC_IOCTL_MAGIC,  3)
#define TOSMAC_IOSETCHAN        _IOW(TOSMAC_IOCTL_MAGIC, 4, __u16)
#define TOSMAC_IOGETCHAN        _IO(TOSMAC_IOCTL_MAGIC,  5)
#define TOSMAC_IOSETFREQ        _IOW(TOSMAC_IOCTL_MAGIC, 6, __u16)
#define TOSMAC_IOGETFREQ        _IO(TOSMAC_IOCTL_MAGIC,  7)
#define TOSMAC_IOSETMAXDATASIZE _IOW(TOSMAC_IOCTL_MAGIC, 8, __u16)
#define TOSMAC_IOGETMAXDATASIZE _IO(TOSMAC_IOCTL_MAGIC,  9)
#define TOSMAC_IOSETPOWER       _IOW(TOSMAC_IOCTL_MAGIC, 10, __u16)
#define TOSMAC_IOGETPOWER       _IO(TOSMAC_IOCTL_MAGIC,  11)


#endif
```

Listing A.24: Beispiel für eine Auftragsdatei `auftrag.txt`

```
# Dies ist die Auftragsdatei. Sie enthaelt die Daten fuer den Auftrag.
# Jede Zeile enthaelt ein Paket, das auszuliefern ist.
# Die Spalten sind durch Whitespaces getrennt.
# Kommentarzeilen sind nur am Dateianfang erlaubt
#
# Spalte 1: Warenbezeichnung
# Spalte 2: verbleibende Haltbarkeit(Shelflife) in Stunden
# Spalte 3: Name der Stadt (aus Staedtekoordinatendatei)
#
Aepfel          70      Koeln
Ananas          34      Aurich
Avocado         17      Kiel
Kartoffeln      90      Wuppertal
Kiwi            66      Essen

Melonen         12      Hannover
Feldsalat       16      Goslar
Moehrchen       33      Bonn
Orangen         40      Luebeck
Gurken          66      Paderborn

Tomaten         35      Gifhorn
Papaja          48      Aachen
Birnen          56      Duesseldorf
Bananen         39      Bielefeld
Pfirsiche       44      Bonn

Weintrauben     26      Guetersloh
Rhabarber       35      Wolfsburg
Bohnen          71      Goettingen
Steckruebe      82      Hamburg
Zwiebeln        42      Bremerhaven
```

Listing A.25: Städte-Koordindaten Datei `staedte_koordinaten`

```
Hamburg          53N34   10E02
Bremen           53N05   8E49
Bremerhaven      53N33   8E35
Oldenburg        53N09   8E13
Osnabrueck       52N17   8E03
Braunschweig     52N16   10E32
Lueneburg        53N15   10E25
Celle            52N38   10E05
Hannover         52N23   9E44
Gifhorn          52N29   10E33
Verden           52N55   9E14
Aurich           53N28   7E29
Wolfsburg        52N25   10E47
Goslar           51N54   10E26
Goettingen       51N32   9E56
Muenster         51N58   7E38
Detmold          51N56   8E53
Dortmund         51N31   7E28
Duesseldorf      51N14   6E47
Aachen           50N47   6E05
Koeln            50N57   6E57
Essen            51N28   7E01
Wuppertal        51N16   7E12
Bonn             50N44   7E06
Paderborn        51N43   8E45
Bielefeld        52N01   8E32
Guetersloh       51N54   8E23
Neumuenster      54N04   9E59
Kiel             54N20   10E08
Luebeck          53N52   10E41
```

Listing A.26: Beispiel für Lieferschein 2009-12-15 Kiel_Avocado.txt

```
Lieferschein
============

1 Paket abgeliefert in  Kiel

Das Paket enthaelt:     Avocado
Qualitaet/Resthaltbarkeit:      7.38889 Std
Haltbar bis             16.12, 03 Uhr
Messtechnische Ueberwachung:    Node 315

Die Temperaturhistorie enthaelt 7 Eintraege.
15.12.2009 11:37:10 24.2 °C,T   24.8 °C,S       36.8 %relHum    80
   rawLight
15.12.2009 12:30:30 24.2 °C,T   24.9 °C,S       36.7 %relHum    79
   rawLight
15.12.2009 13:23:50 24.2 °C,T   24.9 °C,S       37.2 %relHum    79
   rawLight
15.12.2009 14:17:10 24.2 °C,T   24.9 °C,S       36.8 %relHum    79
   rawLight
15.12.2009 15:10:30 24.2 °C,T   24.8 °C,S       36.7 %relHum    79
   rawLight
15.12.2009 18:43:50 24.3 °C,T   24.9 °C,S       38.5 %relHum    78
   rawLight
15.12.2009 19:37:10 24.3 °C,T   24.9 °C,S       37.0 %relHum    79
   rawLight

Beladen:                15.12.2009 10:50:30 (vor 09:36 Std)
Entladen:               15.12.2009 20:27:10
Lieferschein vom:       15.12.2009 20:27:10

<EOF>
```

Listing A.27: Gekürzte Ausgabe von Imote2Basis mit einem Auftrag aus 20 Städten

```
*** Basis() ***
starten():laden
starten():zuordnen
starten():Reihenfolge
unsortiert:             21,12,29,23,22,9,14,24,30,25,10,20,19,26,24,27,13,15,1,3
HTTP-Antwort:           23,24,15,14,13,10,30,29,1,3,12,9,25,24,21,20,19,22,26,27
optimiert:              9,12,3,1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25
zu fahrende strecke:    1922.39 km
zu fahrende zeit (80km/h):      24.0299 Std
starten():Fahrtbeginn

Laderaum enthaelt 20 Waren:
Paket Melonen     SL=12 Std (15.12, 22 Uhr) nach 09=Hannover       Node 723
Paket Ananas      SL=34 Std (16.12, 20 Uhr) nach 12=Aurich         Node 723
Paket Zwiebeln    SL=42 Std (17.12, 04 Uhr) nach 03=Bremerhaven    Node 723
Paket Steckruebe  SL=82 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Paket Avocado     SL=17 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
Paket Orangen     SL=40 Std (17.12, 02 Uhr) nach 30=Luebeck        Node 315
Paket Tomaten     SL=35 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
Paket Rhabarber   SL=35 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
Paket Feldsalat   SL=16 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
Paket Bohnen      SL=71 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
Paket Moehrchen   SL=33 Std (16.12, 19 Uhr) nach 24=Bonn           Node 723
Paket Kartoffeln  SL=90 Std (19.12, 04 Uhr) nach 23=Wuppertal      Node 723
Paket Weintrauben SL=26 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
Paket Bananen     SL=39 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
Paket Kiwi        SL=66 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
Paket Birnen      SL=56 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
Paket Papaja      SL=48 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
Paket Aepfel      SL=70 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
Paket Pfirsiche   SL=44 Std (17.12, 06 Uhr) nach 24=Bonn           Node 315
Paket Gurken      SL=66 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
Derzeitige Route:       9,12,3,1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25

Fahrt 99.4439 km nach: 09=Hannover       Ab 10:53  An 12:08  Fahrzeit: 01:14
fahren():Ankunft um 12:10:30 in 09=Hannover
Laderaum::entladung()

Laderaum enthaelt 19 Waren:
Paket Ananas      SL=33 Std (16.12, 20 Uhr) nach 12=Aurich         Node 723
Paket Zwiebeln    SL=41 Std (17.12, 04 Uhr) nach 03=Bremerhaven    Node 723
Paket Steckruebe  SL=81 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Paket Avocado     SL=16 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
...
Derzeitige Route:       12,3,1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25

Fahrt 193.229 km nach: 12=Aurich         Ab 12:10  An 14:35  Fahrzeit: 02:24

Laderaum enthaelt 18 Waren:
Paket Zwiebeln    SL=38 Std (17.12, 04 Uhr) nach 03=Bremerhaven    Node 723
Paket Steckruebe  SL=78 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Paket Avocado     SL=13 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
...
Derzeitige Route:       3,1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25

Fahrt 73.4103 km nach: 03=Bremerhaven    Ab 14:40  An 15:35  Fahrzeit: 00:55
fahren():Ankunft um 15:40:30 in 03=Bremerhaven
Laderaum::entladung()

Laderaum enthaelt 17 Waren:
Paket Steckruebe  SL=77 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Paket Avocado     SL=12 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
Paket Orangen     SL=35 Std (17.12, 02 Uhr) nach 30=Luebeck        Node 315
Paket Tomaten     SL=30 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
Paket Rhabarber   SL=30 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
Paket Feldsalat   SL=11 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
```

```
Paket Bohnen      SL=66 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
Paket Moehrchen   SL=28 Std (16.12, 19 Uhr) nach 24=Bonn           Node 723
Paket Kartoffeln  SL=85 Std (19.12, 04 Uhr) nach 23=Wuppertal      Node 723
Paket Weintrauben SL=21 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
Paket Bananen     SL=34 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
Paket Kiwi        SL=61 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
Paket Birnen      SL=51 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
Paket Papaja      SL=43 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
Paket Aepfel      SL=65 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
Paket Pfirsiche   SL=39 Std (17.12, 06 Uhr) nach 24=Bonn           Node 315
Paket Gurken      SL=61 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
Derzeitige Route:       1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25

************ nach Abfahrt in 03=Bremerhaven
**
** SIMULIERTE STOERUNG 15:40    Fahrzeug steht 3 Stunden im Stau
**
************

Shelflife-Reduktion!

Laderaum enthaelt 17 Waren:
Paket Steckruebe  SL=74 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Paket Avocado     SL= 9 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
Paket Orangen     SL=32 Std (17.12, 02 Uhr) nach 30=Luebeck        Node 315
Paket Tomaten     SL=27 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
Paket Rhabarber   SL=27 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
Paket Feldsalat   SL= 8 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
Paket Bohnen      SL=63 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
Paket Moehrchen   SL=25 Std (16.12, 19 Uhr) nach 24=Bonn           Node 723
Paket Kartoffeln  SL=82 Std (19.12, 04 Uhr) nach 23=Wuppertal      Node 723
Paket Weintrauben SL=18 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
Paket Bananen     SL=31 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
Paket Kiwi        SL=58 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
Paket Birnen      SL=48 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
Paket Papaja      SL=40 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
Paket Aepfel      SL=62 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
Paket Pfirsiche   SL=36 Std (17.12, 06 Uhr) nach 24=Bonn           Node 315
Paket Gurken      SL=58 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
Derzeitige Route:       1,29,30,10,13,14,15,24,23,27,26,22,19,20,21,24,25
starten():NEUE Reihenfolge
HTTP-Antwort:           10,26,27,22,19,20,21,24,23,24,25,15,14,13,30,29,1
optimiert:              29,30,13,14,15,25,24,23,24,21,20,19,22,27,26,10,1
zu fahrende strecke:    1518.39 km
zu fahrende zeit (80km/h):      18.9799 Std

Laderaum enthaelt 17 Waren:
Paket Avocado     SL= 9 Std (16.12, 03 Uhr) nach 29=Kiel           Node 315
Paket Orangen     SL=32 Std (17.12, 02 Uhr) nach 30=Luebeck        Node 315
Paket Rhabarber   SL=27 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
Paket Feldsalat   SL= 8 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
Paket Bohnen      SL=63 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
Paket Gurken      SL=58 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
Paket Pfirsiche   SL=36 Std (17.12, 06 Uhr) nach 24=Bonn           Node 315
Paket Kartoffeln  SL=82 Std (19.12, 04 Uhr) nach 23=Wuppertal      Node 723
Paket Moehrchen   SL=25 Std (16.12, 19 Uhr) nach 24=Bonn           Node 723
Paket Aepfel      SL=62 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
Paket Papaja      SL=40 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
Paket Birnen      SL=48 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
Paket Kiwi        SL=58 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
Paket Weintrauben SL=18 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
Paket Bananen     SL=31 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
Paket Tomaten     SL=27 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
Paket Steckruebe  SL=74 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Derzeitige Route:       29,30,13,14,15,25,24,23,24,21,20,19,22,27,26,10,1

Fahrt 133.855 km nach: 29=Kiel           Ab 18:43  An 20:24  Fahrzeit: 01:40
```

```
fahren():Ankunft um 20:27:10 in 29=Kiel

Laderaum enthaelt 16 Waren:
Paket Orangen     SL=30 Std (17.12, 02 Uhr) nach 30=Luebeck        Node 315
Paket Rhabarber   SL=25 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
...
Fahrt 63.147 km nach 30=Luebeck          Ab 20:27  An 21:14  Fahrzeit: 00:47

Laderaum enthaelt 15 Waren:
Paket Rhabarber   SL=25 Std (16.12, 21 Uhr) nach 13=Wolfsburg      Node 315
Paket Feldsalat   SL= 6 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
...
Fahrt 161.551 km nach: 13=Wolfsburg      Ab 21:17  An 23:18  Fahrzeit: 02:01

Laderaum enthaelt 14 Waren:
Paket Feldsalat   SL=4 Std (16.12, 02 Uhr) nach 14=Goslar         Node 315
Paket Bohnen      SL=58 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
...
Fahrt 62.2839 km nach: 14=Goslar         Ab 23:23  An 00:10  Fahrzeit: 00:46

Laderaum enthaelt 13 Waren:
Paket Bohnen      SL=58 Std (18.12, 09 Uhr) nach 15=Goettingen     Node 723
Paket Gurken      SL=53 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
...
Fahrt 53.4337 km nach: 15=Goettingen     Ab 00:17  An 00:57  Fahrzeit: 00:40

Laderaum enthaelt 12 Waren:
Paket Gurken      SL=52 Std (18.12, 04 Uhr) nach 25=Paderborn      Node 723
Paket Pfirsiche   SL=30 Std (17.12, 06 Uhr) nach 24=Bonn          Node 315
...
Fahrt 84.2846 km nach: 25=Paderborn      Ab 01:03  An 02:07  Fahrzeit: 01:03

Laderaum enthaelt 11 Waren:
Paket Pfirsiche   SL=29 Std (17.12, 06 Uhr) nach 24=Bonn          Node 315
Paket Kartoffeln  SL=75 Std (19.12, 04 Uhr) nach 23=Wuppertal     Node 723
...
Fahrt 158.783 km nach: 24=Bonn           Ab 02:10  An 04:09  Fahrzeit: 01:59

Laderaum enthaelt 10 Waren:
Paket Kartoffeln  SL=73 Std (19.12, 04 Uhr) nach 23=Wuppertal     Node 723
Paket Moehrchen   SL=16 Std (16.12, 19 Uhr) nach 24=Bonn          Node 723
...
Fahrt 59.7823 km nach: 23=Wuppertal      Ab 04:13  An 04:58  Fahrzeit: 00:44

Laderaum enthaelt 9 Waren:
Paket Moehrchen   SL=15 Std (16.12, 19 Uhr) nach 24=Bonn          Node 723
Paket Aepfel      SL=52 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
...
Fahrt 59.7823 km nach: 24=Bonn           Ab 05:03  An 05:48  Fahrzeit: 00:44

Laderaum enthaelt 8 Waren:
Paket Aepfel      SL=51 Std (18.12, 08 Uhr) nach 21=Koeln          Node 315
Paket Papaja      SL=29 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
...
Fahrt 26.3233 km nach: 21=Koeln          Ab 05:53  An 06:13  Fahrzeit: 00:19

Laderaum enthaelt 7 Waren:
Paket Papaja      SL=29 Std (17.12, 10 Uhr) nach 20=Aachen         Node 723
Paket Birnen      SL=37 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
...
Fahrt 63.6526 km nach: 20=Aachen         Ab 06:17  An 07:04  Fahrzeit: 00:47

Laderaum enthaelt 6 Waren:
Paket Birnen      SL=36 Std (17.12, 18 Uhr) nach 19=Duesseldorf    Node 315
Paket Kiwi        SL=46 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
...
Fahrt 70.0946 km nach: 19=Duesseldorf    Ab 07:10  An 08:03  Fahrzeit: 00:52
```

```
Laderaum enthaelt 5 Waren:
Paket Kiwi         SL=45 Std (18.12, 04 Uhr) nach 22=Essen          Node 315
Paket Weintrauben SL= 5 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
...
Fahrt 30.6243 km nach: 22=Essen          Ab 08:07  An 08:30  Fahrzeit: 00:22

Laderaum enthaelt 4 Waren:
Paket Weintrauben SL= 4 Std (16.12, 12 Uhr) nach 27=Guetersloh     Node 723
Paket Bananen     SL=17 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
...
Fahrt 105.942 km nach: 27=Guetersloh     Ab 08:30  An 09:49  Fahrzeit: 01:19

Laderaum enthaelt 3 Waren:
Paket Bananen     SL=16 Std (17.12, 01 Uhr) nach 26=Bielefeld      Node 723
Paket Tomaten     SL=12 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
...
Fahrt 16.5695 km nach: 26=Bielefeld      Ab 09:53  An 10:06  Fahrzeit: 00:12

Laderaum enthaelt 2 Waren:
Paket Tomaten     SL=12 Std (16.12, 21 Uhr) nach 10=Gifhorn        Node 315
Paket Steckruebe  SL=59 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Derzeitige Route:       10,1

Fahrt 146.923 km nach: 10=Gifhorn        Ab 10:07  An 11:57  Fahrzeit: 01:50

Laderaum enthaelt 1 Waren:
Paket Steckruebe  SL=57 Std (18.12, 20 Uhr) nach 01=Hamburg        Node 315
Derzeitige Route:       1

Fahrt 125.459 km nach: 01=Hamburg        Ab 12:03  An 13:37  Fahrzeit: 01:34

fahren():Ankunft um 13:40:30

Jetzt in 01=Hamburg
Laderaum::entladung()

starten(): alle Pakete entladen

Sensornetz:
Sensornode 315: Protokoll:
Die Temperaturhistorie enthaelt 27 Eintraege.
15.12.2009 11:37:10 24.2 °C,T   24.8 °C,S       36.8 %relHum    80 rawLight
15.12.2009 12:30:30 24.2 °C,T   24.9 °C,S       36.7 %relHum    79 rawLight
15.12.2009 13:23:50 24.2 °C,T   24.9 °C,S       37.2 %relHum    79 rawLight
...
16.12.2009 12:30:30 24.4 °C,T   25.1 °C,S       36.1 %relHum    80 rawLight
16.12.2009 13:23:50 24.4 °C,T   25.0 °C,S       36.2 %relHum    80 rawLight

Sensornode 723: Protokoll:
Die Temperaturhistorie enthaelt 27 Eintraege.
15.12.2009 11:30:30 24.6 °C,T   25.1 °C,S       37.2 %relHum    74 rawLight
15.12.2009 12:23:50 24.6 °C,T   25.1 °C,S       37.2 %relHum    74 rawLight
15.12.2009 13:17:10 24.6 °C,T   25.1 °C,S       37.6 %relHum    73 rawLight
...
16.12.2009 12:20:30 24.7 °C,T   25.2 °C,S       36.5 %relHum    74 rawLight
16.12.2009 13:13:50 24.8 °C,T   25.3 °C,S       36.5 %relHum    74 rawLight
```

Literaturverzeichnis

[1] Intel Corporation. *Intel PXA27x Processor Family - Electrical, Mechanical, and Thermal Specification*, datasheet, 2005.

[2] Dirk Hentschel, Reiner Jedermann, Walter Lang. *Autonomous Cooperating Processes for the Improvement of Food Quality*, Sensor & Test, Nürnberg 2009.

[3] Donald Thompson, Rob S.Miles. *Embedded Programming with the Microsoft .NET Micro Framework*. Microsoft Press, 2007.

[4] Frank Eller. *Visual C# 2005 - Grundlagen, Programmiertechniken, Datenbanken*. Addison-Wesley, 2006.

[5] Ilja N Bronstein, Konstantin A Semendjajew. *Taschenbuch der Mathematik*. Harri Deutsch Verlag, 2000.

[6] Jürgen Quade, Eva-Katharina Kunst. *Linux Treiber entwickeln - eine systematische Einführung in Gerätetreiber für den Kernel 2.6*. dpunkt.verlag, 2004.

[7] Jens Kühner. *Expert .NET Micro Framework - Embedded programming of microcontrollers with C# and Microsoft Visual Studio*. Apress, 2008.

[8] Ishwar Lal. *Imote2 How To under Linux OS*, Short report, IMSAS-intern, 8/4/2009.

[9] Frank Littek. *Kühltransport - Temperaturgeführte Logistik und Transporttechnik*. Agrimedia, 2005.

[10] Reiner Jedermann, Walter Lang. *The Benefits of Embedded Intelligence - Tasks and Applications for Ubiquitous Computing in Logstics*, Internet of Things 2008, LCNS 4952, Springer Verlag, Berlin/Heidelberg 2008.

[11] Crossbow Technology. *Imote2 Hardware Reference Manual*, Revision A, September 2007.

[12] Crossbow Technology. *Imote2.Builder Kit Manual*, Revision A, September 2007.