Entwicklung und Integration von Services und Gatewayfunktionalität auf Basis von AmICoM

Entwicklung und Integration von Services und Gatewayfunktionalität auf Basis von AmICoM

Diplomarbeit

Arbeitsgruppe Vernetzte Systeme
Fachbereich Informatik
Technische Universität Kaiserslautern

Thorsten Schmelzer

Tag der Abgabe : 26.05.2008

Betreuer : Prof. Dr. Reinhard Gotzhein und Marc Krämer

Ich erkläre hiermit, die vorliegende Diplomarbeit selbständig verfasst zu haben. Die verwendeten Quellen und Hilfsmittel sind im Text kenntlich gemacht und im Literaturverzeichnis vollständig aufgeführt.

Kaiserslautern, den 23. Mai 2008

(Thorsten Schmelzer)

Danksagung

Diese Arbeit entstand von Oktober 2007 bis Mai 2008 in der Arbeitsgruppe Vernetzte Systeme an der Technischen Universität Kaiserslautern.

Ganz besonders möchte ich dem Leiter der Arbeitsgruppe Herr Prof. Dr. Gotzhein für die Betreuung und Ausgabe der Arbeit danken. Danken möchte ich auch meinem Betreuer Marc Krämer, der mich mit großem Engagement bei der Erstellung dieser Arbeit unterstützt hat. Ingmar Fliege danke ich für seine unermüdliche Unterstützung in allen SDL-spezifischen Fragen, die Entwicklung der AmICoM, auf der diese Arbeit aufbaut, und seine Bugfixes für ConTraST. Allen anderen Arbeitsgruppenmitgliedern danke ich für die gute Arbeitsatmosphäre.

Mein weiterer Dank gilt Paul Pichota und Stephan Gornik für ihre Korrekturen und Anmerkungen zu dieser Arbeit.

Schließlich möchte ich meiner Familie, besonders meinen Eltern, die mir mein Studium ermöglicht haben, für die Unterstützung während der Diplomarbeit und des Studiums bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der heutigen Zeit gewinnt die Vernetzung von Rechnern eine immer größere Bedeutung. Ansätze wie Ambient Intelligence [For] oder Ubiquitous Computing [Wei91] zeichnen ein Bild von der Durchdringung unseres Alltagslebens mit vernetzten Rechnern. Diese vernetzten Rechner sollen dem Menschen das Leben im Alltag erleichtern, indem sie Dienste für den Menschen erbringen. Dabei sollen die Rechner weitestgehend in das Alltagsbild des Menschen integriert werden und im Hintergrund agieren. Um mit dem Menschen geeignet zu interagieren, benötigen die Knoten Sensoren und Aktuatoren sowie geeignete Kommunikationsschnittstellen, um mit anderen Knoten zu kommunizieren. In diesen Netzwerken kommen auch mobile Knoten zum Einsatz, die entweder von Menschen getragen oder in mobile Geräte eingebaut werden. Für mobile Knoten benötigt man Hardware mit geeigneten Abmessungen und sparsamem Energieverbrauch, um sie in die Umwelt des Menschen zu integrieren. Die Kommunikation zwischen mobilen Knoten erfolgt oft drahtlos, um die einzelnen Knoten nicht miteinander verkabeln zu müssen, da die Mobilität sonst aufgrund der Kabel stark eingeschränkt wäre.

Diese Arbeit baut auf der Projektarbeit [Sch07], die die Imote2-Hardwareplattform und deren Integration in das SDL Environment Framework vorstellt, auf. Der Imote2 besitzt geringe Abmessungen und einen sparsamen Energieverbrauch. Er verfügt über eine drahtlose Kommunikationsschnittstelle, sowie über zahlreiche weitere Hardwareschnittstellen. Ein Sensorboard, das über Sensoren und Aktuatoren wie Temperatursensor, Mikrofon und Leuchtdioden verfügt, kann an den Imote2 angeschlossen und von ihm genutzt werden. Die dazu notwendigen Treiber für die Hardware wurden im Verlauf dieser Arbeit entwickelt. Der Imote2 kann somit die Rolle eines Knotens in einem oben beschriebenen Netzwerk übernehmen.

Neben der entsprechenden Hardware benötigt man noch geeignete Software, die solche Netzwerke von Rechnern organisieren und koordinieren. Die in der AG entwickelte Ambient Intelligence Communication Middleware (AmICoM) [KF07] ist eine Kommunikationsmiddleware für Ambient Intelligence Netzwerke. Mit Hilfe der AmICoM können Knoten Daten, die sie mittels ihrer Sensoren ermittelt und eventuell vorverarbeitet haben, oder den Zugriff auf eine ihrer Hardwarekomponenten wie zum Beispiel Aktuatoren als Dienste dem gesamten Netzwerk zur Verfügung stellen. Andere Knoten im Netzwerk können auf diese Dienste zugreifen und sie nutzen. Die AmICoM wurde bis jetzt auf PC-Plattformen eingesetzt. Um das Einsatzspektrum der AmICoM zu erweitern, soll sie auf der Imote2-Hardwareplattform, die sich für kleine und mobile Knoten eignet, zum Einsatz kommen. Hierfür wird die PC-Version der AmICoM auf den Imote2 portiert. Um die Funktionsfähigkeit der AmICoM auf dem Imote2 zu demonstrieren, werden Dienste für den Imote2 entwickelt, die die Sensoren und Aktuatoren des Sensorboards nutzen. Damit die AmICoM

übergreifend sowohl im PC-Netzwerk als auch im Imote2-Netzwerk genutzt werden kann, wird ein Gateway entwickelt, dass beide Netzwerke miteinander verbindet.

Die AmICoM wird mittels der Specification and Description Language (SDL) [SDL] spezifiziert. SDL ist eine Spezifikationssprache, mit der sich Systeme auf einer hohen Abstraktionsebene modellieren lassen. Eine Besonderheit dieser Sprache ist, dass man SDL-Systeme in einer graphischen Repräsentation erstellen kann. Mit geeigneter Werkzeugunterstützung ist es möglich, aus einem SDL-System ausführbaren Kode zu generieren. Die Kommunikation mit der Hardware läuft über ein in der AG entwickeltes Framework, das die unterschiedlichen Treiber für die Hardware enthält und eine Schnittstelle für SDL-Systeme anbietet. Die für das Sensorboard entwickelten Treiber werden in dieses Framework integriert und sind aus einem SDL-System heraus nutzbar.

Die Arbeit ist in folgende Kapitel unterteilt: Kapitel 2 enthält die Beschreibung der Imote2-Hardwareplattform, des Sensorboards und der in dieser Arbeit verwendeten Software. Kapitel 3 handelt von der Integration des Sensorboards in das SDL Environment Framework (SEnF), der Ansteuerung der Sensoren und Aktuatoren des Sensorboards und der Kommunikation zwischen Sensorboard und Imote2. In Kapitel 4 wird die AmICoM vorgestellt und deren Portierung auf den Imote2 beschrieben. Die auf die AmICoM aufsetzenden Dienste, die auf den Sensoren und Aktuatoren des Sensorboards basieren, werden ebenfalls in Kapitel 4 vorgestellt. Kapitel 5 beschreibt das AmICoM-Gateway, das Imote2-Netzwerk und PC-Netzwerk verbindet und darüber hinaus Filterfunktionalitäten bereitstellt, um den Datenverkehr in beiden Teilnetzen gering zu halten. Kapitel 6 handelt vom Audiodienst des Sensorboards; die Ansätze der Audioabtastung und des Audiodienstes werden erläutert. Außerdem enthält das Kapitel einen Abschnitt über Audiokompression, die für den Audiodienst eingesetzt werden soll. In Kapitel 7 wird das Testsystem, das zur Überprüfung der Funktionalität der AmICoM-Implementierung verwendet wurde, vorgestellt. Im letzten Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick über die Zukunft der AmICoM gegeben.

Kapitel 2

Grundlagen

2.1 Hardware

In diesem Anschnitt wird die während dieser Arbeit verwendete Hardware genauer beschrieben. Insbesondere das in der AG entwickelte Sensorboard für den Imote2 wird vorgestellt.

2.1.1 Imote2

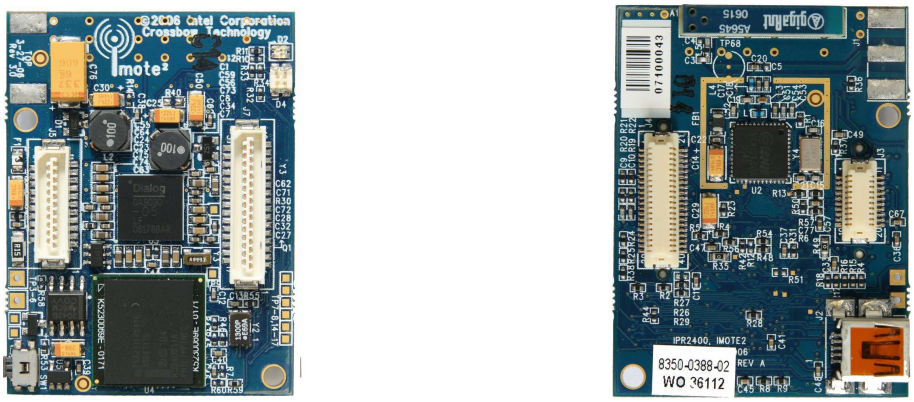


Abbildung 2.1: Die Oberseite (links) und Unterseite (rechts) der Imote2-Hardwareplattform

Der in dieser Arbeit verwendete drahtlose Sensorknoten ist der von Crossbow/Intel hergestellte Imote2 (IntelMote2). Der Imote2 besteht im Wesentlichen aus folgenden Komponenten:

- Intel PXA271: Dies ist der Mikrocontroller des Imote2 [Int06]. Die Architektur des Prozzesors ist die Intel XScale Architektur, die auf der ARM 5TE Architektur aufbaut. Der PXA271 verfügt über 256 kByte SRAM, 32 MB SDRAM und 32 MB FLASH-Speicher. Die Taktzahl des Prozessors kann auf 13, 104, 208, 312 und 416 MHz eingestellt werden.

- Texas Instruments CC2420: Der drahtlose Transceiver des Imote2 operiert im 2,4 GHz Bereich und unterstützt den ZigBee (IEEE 802.15.4) Standard [Tex07]. Er hat auf freier Fläche eine Reichweite von circa 30 Metern [Tex07] und sendet mit einer

maximalen Datenrate von 250 kbps. Der CC2420 hat je einen Sende- und Empfangspuffer von 128 Byte. Maximal können pro Frame 125 Byte Nutzdaten übertragen werden. Die Verbindung zum Hauptprozessor erfolgt über eine SPI-Schnittstelle.

- Dialog DA9030: Die Power Management Einheit der Imote2 Hardwareplattform versorgt sowohl den Prozessor als auch die übrige Peripherie mit den notwendigen Spannungen [Dia05]. Angeschlossen an den Hauptprozessor ist der DA9030 mittels eines I^2C Power Interface.

Abbildung 2.1 zeigt die Ober- und Unterseite des Imote2 und Abbildung 2.2 gibt einen Überblick über die verschiedenen Komponenten und Anschlussmöglichkeiten der Imote2 Plattform. Für nähere Informationen zu der Hardwareplattform siehe [Sch07].

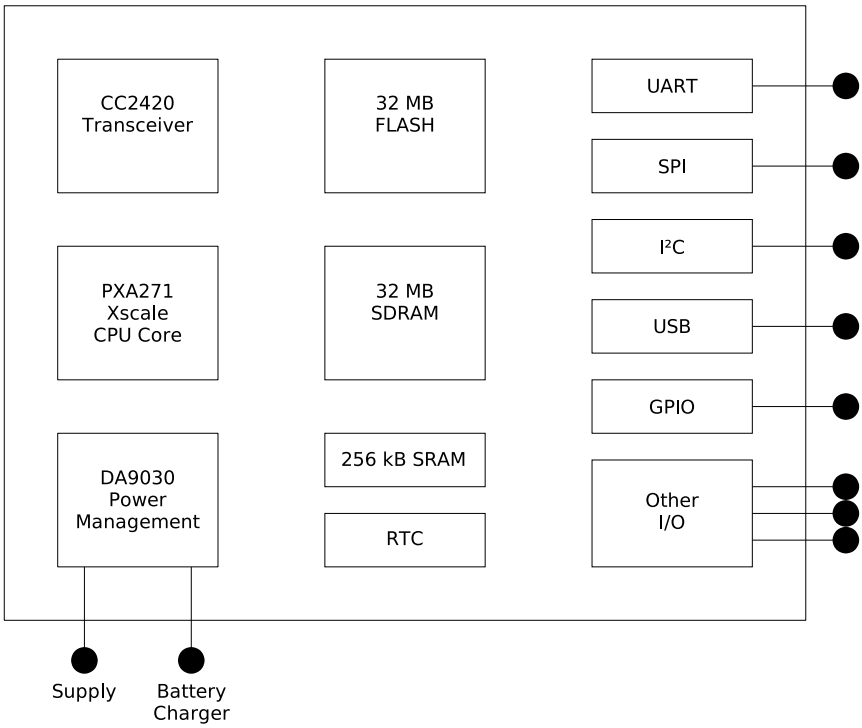


Abbildung 2.2: Die Hauptkomponenten des Imote2

Es besteht zudem die Möglichkeit, den Imote2 über einen Lithium-Polymer-Akku mit Spannung zu versorgen. Um den Akku mit dem Imote2 zu verbinden, wurden die im Lieferumfang enthaltenen Batteryboards umgebaut. Die Batteryboards ermöglichen den Betrieb des Imote2 mit drei handelsüblichen 1,5 Volt Batterien. Die Halterung für die Batterien dieser Boards wurde gegen einen 3,7 Volt Lithium-Polymer-Akku mit 1100 mAh ausgetauscht. Der Lithium-Polymer-Akku bietet eine vergleichbare Laufzeit wie die drei 1,5 Volt Batterien, hat aber deutliche kleinere Abmessungen und ist leichter.

2.1.2 Sensorboard

Das Sensorboard ist eine Eigenentwicklung der AG Vernetzte Systeme für den Imote2 (siehe Abbildung 2.3). Es stellt eine Reihe von Sensoren, Aktuatoren und weiteren Anschlussmöglichkeiten zur Verfügung. Die digitale Wandlung der Sensorergebnisse und die Steuerung

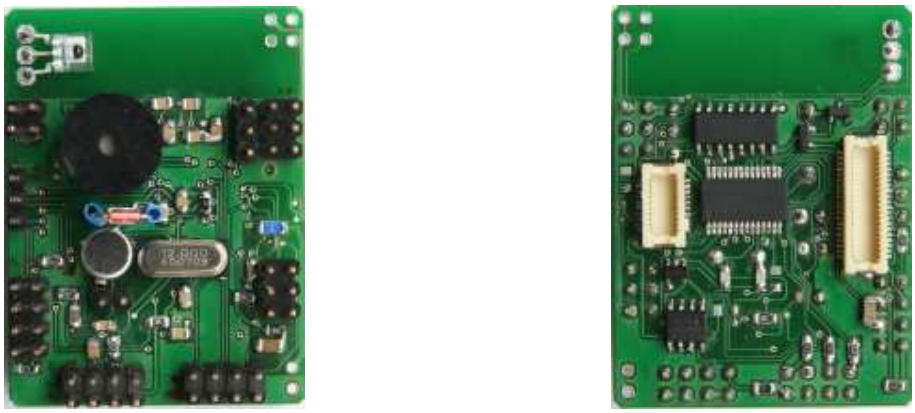


Abbildung 2.3: Die Oberseite (links) und Unterseite (rechts) des Sensorboards für die Imote2-Hardwareplattform

der Aktuatoren geschieht mit Hilfe eines zentralen Mikrocontrollers, dem PIC 18F4321 [Mic07b], der auch für die Kommunikation zwischen Sensorboard und Imote2 zuständig ist. Eine Digitalisierung von Sensordaten ist notwendig, da der Imote2 selbst nur über Digitalschnittstellen verfügt. Einen Überblick über alle Komponenten des Sensorboards zeigt Abbildung 2.4.

Die Sensoren sind im einzelnen:

- Ein Temperatursensor [ntc]: Als Temperatursensor wird ein Heißleiter (NTC-Widerstand) verwendet. Ein Heißleiter ist ein elektronisches Bauteil, dessen Widerstand mit steigender Temperatur fällt. Bildet man einen Spannungsteiler aus einem herkömmlichen Widerstand und einem NTC-Widerstand (Negative Temperature Coefficient), verändert sich die abfallende Spannung am NTC-Widerstand mit der Temperatur. Aus der Spannung kann nun die Temperatur bestimmt werden.

- Ein Passiv Infrarot Bewegungsmelder (PIR) [Hyg06]: Diese Art von Bewegungsmelder reagiert auf die Wärmestrahlung von bewegten Körpern. Wird eine Bewegung detektiert, wird ein Schaltsignal auf den Ausgang des Bewegungsmelders gelegt, das dann weiterverarbeitet werden kann.

- Ein Lichtsensor [TAO01]: Der Lichtsensor besteht aus einer Photodiode und einem Operationsverstärker. Je heller das einfallende Licht, desto mehr Strom fließt durch die Photodiode und desto höher ist somit die Ausgangsspannung des Sensors.

- Eine Mikrofonschaltung: Die Mikrofonschaltung besteht aus dem Mikrofon und einem Mikrofonverstärker. Die verstärkten Mikrofonsignale werden abgetastet, um daraus digitale Audiodaten zu gewinnen.

Die Aktuatoren des Sensorboards umfassen:

- Ein Piezoschallwandler (Piezosummer): Dieser Aktuator nutzt den umgekehrten piezoelektrischen Effekt. Wird eine Spannung an das Bauteil angelegt, so bringt eine elektronische Schaltung den Piezokristall zum Schwingen und es entsteht ein lauter Ton. Die Frequenz des Tons kann hierbei nicht verändert werden.

- LEDs und Interrupts: Es können bis zu vier LEDs an das Sensorboard angeschlossen werden. Alternativ zu den LEDs können diese Anschlüsse als Interruptpins für

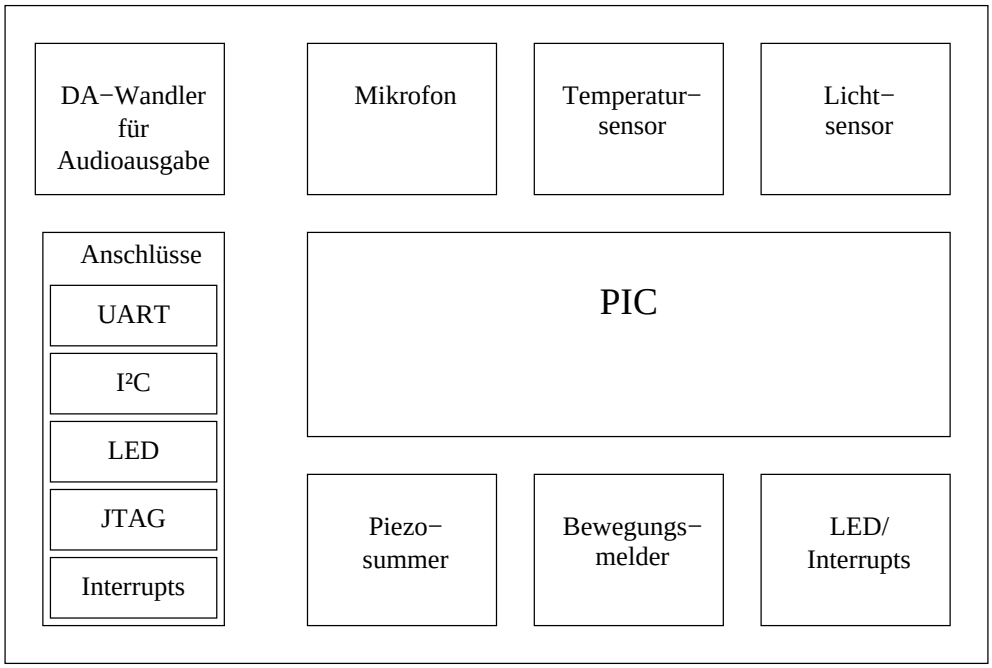


Abbildung 2.4: Überblick über die Hauptkomponenten des Sensorboards

den Mikrocontroller des Sensorboards genutzt werden. Daneben können die Pins des Programmierports des PICs, der sich ebenfalls auf dem Sensorboard befindet, mit alternativen Funktionen belegt werden und als E/A-Pins genutzt werden.

- Einem Digital-Analog Wandler [Mic07a]: Dieser 12 Bit D/A-Wandler dient dazu, Audiodaten wiederzugeben. Dafür wird ein Kopfhörer oder Lautsprecher an den Ausgang des D/A-Wandlers angeschlossen. Der D/A-Wandler ist an die SPI-Schnittstelle des Imote2 angeschlossen.

Zusätzlich besitzt das Sensorboard Anschlüsse, die vom Imote2 durch das Sensorboard geschleift werden. Dies ermöglicht einen einfachen Anschluss von Peripheriegeräten an den Imote2 mit Pfostensteckern. Folgende durchgeschleifte Anschlüsse des Imote2 stellt das Sensorboard zur Verfügung:

- Zwei UART-Schnittstellen: Die UART-Schnittstellen des Imote2 haben Spannungspegel von -3 und 3 Volt, die des PCs -12 und 12 Volt. Um den PC direkt mit den UART-Schnittstellen des Imote2 zu verbinden, haben beide Anschlüsse einen Spannungswandler, der die Signalpegel zwischen IMote2 und PC umsetzt. Darüberhinaus kann man eine der UART-Schnittstellen auch ohne vorgeschalteten Spannungswandler nutzen, dann bleiben die Signalpegel unverändert.
- Eine I^2C-Schnittstelle
- Vier LED-Anschlüsse: Die LED-Anschlüsse sind mit GPIO-Pins des Imote2 verbunden und können nicht nur für Leuchtdioden genutzt werden, sondern zum Beispiel auch für digitale Schalter.
- Vier Interrupt-Pins
- Eine JTAG-Schnittstelle: Die JTAG-Schnittstelle dient zum Beschreiben des FLASH-Speichers des Imote2 mit Hilfe eines JTAG-Kabels. Somit kann zum Beispiel ein neuer

Bootloader (siehe Abschnitt 2.2.7) auf den Imote2 übertragen werden. Mit geeigneter Software ist es auch möglich, den Imote2 über diese Schnittstelle zu debuggen.

Der Mikrocontroller des Sensorboards ist der 8-Bit Mikrocontroller PIC 18F4321 [Mic07b], der über 8 kByte Programmspeicher und 512 Byte Hauptspeicher verfügt. Getaktet wird der PIC mit einem externen Oszillator mit einer Frequenz von 12 MHz. Zu beachten ist, dass der PIC pro Befehl vier Zyklen benötigt. Er verarbeitet somit 3 Millionen Instruktionen pro Sekunde. Um mit anderen elektronischen Bauteilen zu kommunizieren, besitzt der PIC drei unterschiedliche serielle Hardware-Schnittstellen. Die SPI-Schnittstelle und die I^2C-Schnittstelle sind nur unter gegenseitigem Ausschluss nutzbar, während die UART-Schnittstelle separat nutzbar ist. Eine der Hauptaufgaben des PIC ist die Digitalisierung der Sensorwerte der an ihn angeschlossenen Sensoren. Hierzu verfügt der PIC 18F4321 über einen 10-Bit Analog-Digital-Wandler mit insgesamt 10 Kanälen. Der A/D-Wandler kann immer nur einen Wert gleichzeitig wandeln, eine parallele Wandlung mehrerer Kanäle ist nicht möglich. Mikrofon, Lichtsensor und Temperatursensor sind mit je einem Kanal verbunden. Durch Auswahl des entsprechenden Kanals kann der analoge Wert des Sensors digitalisiert werden. Zum Ansteuern von Piezoschallwandler, LEDs und Aktivieren/Deaktivieren von Sensoren besitzt der PIC eine Reihe von GPIO-Pins, die mit digitalen Werten beschaltet werden können. Der PIC kann in Assembler oder in C programmiert werden. Aus Effizienzgründen wird der PIC in dieser Arbeit mit Assembler programmiert. Für die Programmierung steht eine Entwicklungsumgebung zur Verfügung, die aus Assembler-Anweisungen Machinenkode generiert. Die entstandene Binärdatei wird mit einem Flashprogrammiergerät auf den PIC übertragen.

2.2 Software

An dieser Stelle wird die in dieser Arbeit verwendete Software vorgestellt. Es erfolgt nur eine kurze Vorstellung der einzelnen Teile, eine genauere Beschreibung findet sich in [Sch07]. Lediglich der im Vergleich zur Projektarbeit geänderte Bootloader wird genauer vorgestellt.

2.2.1 SDL

SDL (Specification and Description Language) [SDL] ist eine genormte Sprache um verteilte Systeme zu spezifizieren, sie findet vor allem im Bereich der Telekommunikation Anwendung. SDL ermöglicht die Spezifikation sowohl in graphischer (GR) als auch in textueller Repräsentation (PR). In SDL wird ein System mit Hilfe von erweiterten Zustandsautomaten beschrieben, die mittels Signalen miteinander kommunizieren. Als Umgebung eines SDL-Systems wird alles außerhalb des SDL-Systems bezeichnet, zum Beispiel andere Programme und Hardware. Mit dieser Umgebung kann ein SDL-System über eine fest definierte Schnittstelle Informationen austauschen. Das SDL Runtime Environment (siehe unten) ist eine Implementierung dieser Schnittstelle.

2.2.2 Telelogic Tau

Telelogic Tau [Tel] ist ein von der Firma Telelogic entwickeltes kommerzielles Werkzeug zur Erstellung von SDL-Systemen. Die Eingabe kann in graphischer oder in textueller Form erfolgen. Tau bietet darüberhinaus noch Analyse- und Simulationswerkzeuge für die erstellten SDL-Systeme. Tau selbst besitzt die Möglichkeit, aus den erstellten SDL-Systemen ausführbaren Kode zu erzeugen; diese Möglichkeit wird in dieser Arbeit nicht genutzt. Statt dessen wird zur Kodeerzeugung ConTraST [FGW06] verwendet.

2.2.3 ConTraST und SDLRE

ConTraST (**Co**nfigurable **Tra**nspiler for **S**DL to C++ **T**ranslation) [FGW06] ist ein Transpiler, der ein SDL-System nach C++ übersetzt. Er wurde in der AG Vernetzte Systeme entwickelt. Um lauffähigen Kode zu erzeugen, benötigt man zusätzlich zu dem übersetzten SDL-System noch eine Laufzeitumgebung (**SDL R**untime **E**nvironment). Diese ist ebenfalls eine Eigenentwicklung der AG Vernetzte Systeme und ist eine manuelle Transformation der formalen Semantik von SDL-2000 in C++. Sowohl das nach C++ übersetzte SDL-System als auch die Laufzeitumgebung werden dann von einem C++-Compiler zu einem ausführbaren System übersetzt.

2.2.4 SEnF

Das SDL Environment Framework dient zur Anbindung der Umgebung an ein SDL-System [FGJ+05]. Dazu werden mit der Umgebung Signale ausgetauscht. Signale vom SDL-System in die Umgebung werden empfangen und die entsprechende Hardware wird angesteuert. In umgekehrter Richtung werden zum Beispiel mit der Hardware empfangene Daten in ein Signal gepackt und dem SDL-System so zur Verfügung gestellt. Für jede unterstützte Hardwarekomponente gibt es ein SEnF-Modul, das sich um die Umsetzung von SDL-Signalen kümmert und verschiedene Treiberfunktionalitäten für die Hardware bereitstellt. SEnF stellt unterschiedliche Treiber für verschiedene Plattformen bereit, das bedeutet, wenn die Hardwareplattform das Modul unterstützt, wird der für die Hardwareplattform benötigte Kode automatisch hinzugefügt. Mit Hilfe von Präprozessordirektiven wird SEnF automatisch zu einem SDL-System geladen und die vom SDL-System benötigten SEnF-Module werden eingebunden.

2.2.5 Compiler

Die Wasabi Toolchain [Was04] ist ein *CrossCompiler* für die XScale-Architektur. *CrossCompiler* bedeutet, dass die Architektur des Zielsystems sich von der Architektur des compilierenden Systems unterscheidet. Im Falle des Imote2 wird für die XScale-Architektur auf einer x86-Architektur compiliert. Die Wasabi Toolchain basiert auf der GNU Complier Collection (GCC) Version 3.2.1. Mit Hilfe der Wasabi Toolchain ist es möglich, C/C++ Kode für den Imote2 zu compilieren.

Für SDL-Systeme, die für den PC compiliert werden, werden andere C/C++-Compiler verwendet. Unter Windows wird der C/C++-Compiler von Microsoft (Version 12.00.8168) eingesetzt, unter Linux der GCC (Version 4.2.3).

2.2.6 JFlashMM

JFlashMM ist eine frei verfügbare Software von Intel, die es ermöglicht, Daten in den Flash-Speicher von verschiedenen Hardwareplattformen, unter anderem auch der Imote2, zu laden. Dazu muss die Hardware über eine JTAG-Schnittstelle verfügen und man benötigt ein passendes JTAG-Kabel. Mit der Software ist man in der Lage, den Bootloader eines Imote2 wieder herzustellen, wenn dieser irrtümlich überschrieben oder gelöscht wurde. In dieser Arbeit wird eine leicht modifizierte Version von JFlashMM verwendet, die JTAG-Kabel vom Typ Wiggler unterstützt [Jfl]. Wiggler sind verhältnismäßig einfache JTAG-Kabel und es gibt eine große Anzahl an verschiedenen Schaltplänen zur freien Verfügung [Wig]. Ein Kabel dieses Typs wurde in der AG gebaut und kann zum Flashen des Imote2 verwendet werden.

2.2.7 Bootloader und USBLoader

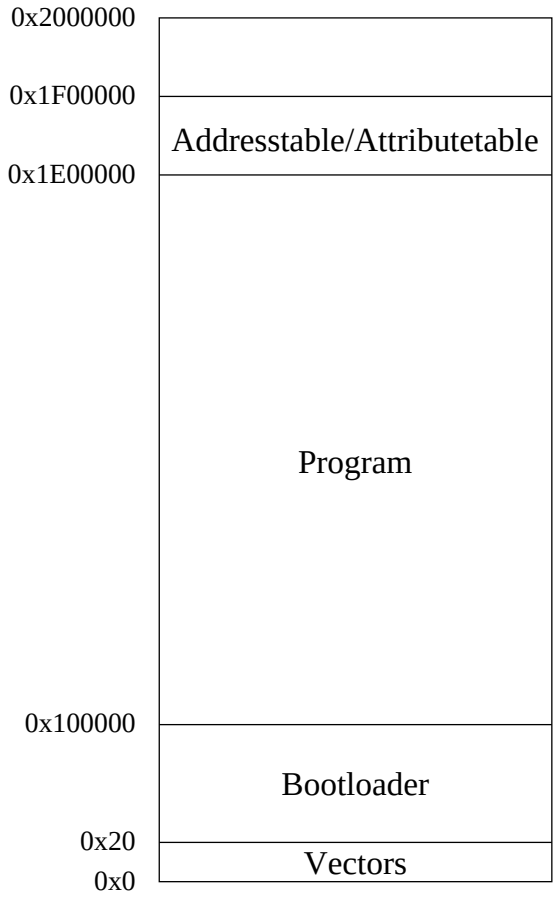


Abbildung 2.5: Die Speicheraufteilung des Imote2 laut [Sha05]

Der Bootloader, der bei dem von Crossbow gelieferten Imote2 vorinstalliert ist, dient dazu, Programme über die USB-Schnittstelle auf den Imote2 zu laden. Das Programmieren des Imote2 wäre prinzipiell nur über die JTAG-Schnittstelle möglich, jedoch ist die Kombination aus Bootloader und USB-Schnittstelle um ein vielfaches schneller. Der USBLoader ist das Programm auf PC-Seite, das mit dem Bootloader des Imote2 kommuniziert. Dem USB-Loader werden die binären Dateien übergeben, die auf den Imote2 geladen werden sollen. Der Bootloader wird nach jedem Booten des Imote2 aktiv und überprüft, ob der Imote2 über USB mit einem PC verbunden ist und ob auf dem PC eine Instanz des USBLoaders läuft. Ist das der Fall, kann nun die Binärdatei auf den Imote2 geladen werden. Detektiert der Bootloader keine Verbindung zu einem PC, so springt der Bootloader zu der geladenen Anwendung und führt diese aus.

Laut Dokumentation des Bootloaders [Sha05] sieht die Aufteilung des FLASH-Speichers wie in Abbildung 2.5 aus. Hier beginnt der Bootloader an Adresse 0x0. Die eigentliche Applikation beginnt an Adresse 0x100000 und am Ende des 32 MB großen Speichers stehen die Attributtabellen, die wichtige Konstanten wie Bootvektoren und Sprungadressen enthalten. Tests und ein Studium des Quellkodes des Bootloaders ergeben, dass diese Speicheraufteilung, wie im Dokument des Bootloaders beschrieben, nicht korrekt ist. Die tatsächliche Aufteilung des Speichers zeigt Abbildung 2.6. Bei dem Ladevorgang eines Programms über den Bootloader werden die Daten zunächst in die primäre oder sekundäre

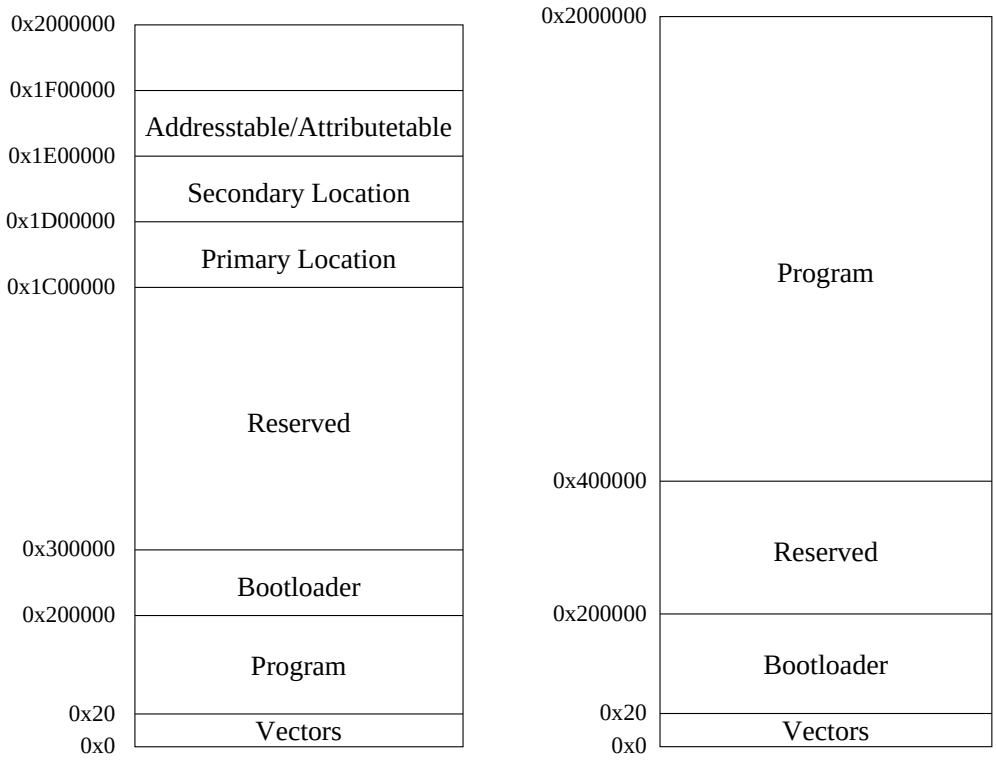


Abbildung 2.6: Die alte Speicheraufteilung (links) und neue Speicheraufteilung (rechts) des Imote2

Location geladen. Dann wird das geladene Image verifiziert und in die Programmpartition kopiert. Der Bootloader beginnt hinter der Programmpartition an Adresse 0x200000. Vor dem eigentlichen Programm liegen an Adresse 0x0 noch Sprungvektoren, die dafür sorgen, dass beim Booten zuerst der Bootloader angesprungen wird. Außerdem enthalten sie die Interruptsprungadressen. Diese Aufteilung ist denkbar ungünstig. Ab einer Größe der Imagedatei von über einem Megabyte kommt es zu einem Datenverlust im FLASH-Speicher. Da beim Laden die primäre und sekundäre Location abwechselnd genutzt werden, überschreibt man, wenn die sekundäre Location genutzt wird, die Attributtabellen, die ein Megabyte hinter der sekundären Location liegen. Die Folge davon ist, dass man keine Dateien mehr auf den Imote2 laden kann, da der Bootloader die Attributtabellen nicht mehr lesen kann. Übersteigt das geladene Image eine Größe von zwei Megabyte, wird beim Kopieren von einer der Locations in die Programmpartition der Bootloader überschrieben. Die so beschädigten Imote2 lassen sich nur mit Hilfe der JTAG-Schnittstelle wieder benutzen. Um das überschreiben von Bootloader oder Attributtabellen zu verhindern, wird die Speicheraufteilung des Imote2 verändert und der Bootloader entsprechend angepasst. Der Bootloader befindet sich am Beginn des Speichers nach den Sprungvektoren, danach folgt an Adresse 0x400000 die Programmpartition (siehe Abbildung 2.6). Auf das Verifizieren und Kopieren des Images wird verzichtet, um die Ladezeiten des Images auf den Imote2 zu verkürzen. Dafür nimmt man in Kauf, dass man einen Übertragungsfehler der USB-Schnittstelle nicht bemerkt. Das Kopieren von der primären bzw. sekundären Location in die Programmpartition kann damit entfallen. Das Image wird stattdessen über die USB-Schnittstelle direkt in die Programmpartition geladen, was ebenso die Ladezeiten des Images verkürzt. Ebenfalls eingespart werden die Attributtabellen, die in ihnen enthaltenen Informationen werden direkt in den Bootloader kodiert. Das Verhalten des Bootloaders ändert sich dadurch nicht. Der veränderte Bootloader wird mittels JTAG-Schnittstelle auf

alle Imote2 überspielt. Die maximale Größe der Imagedatei, die auf den Imote2 mit dem veränderten Bootloader geladen werden kann, beträgt 28 Megabyte.

2.2.8 Zusammenwirken der Software

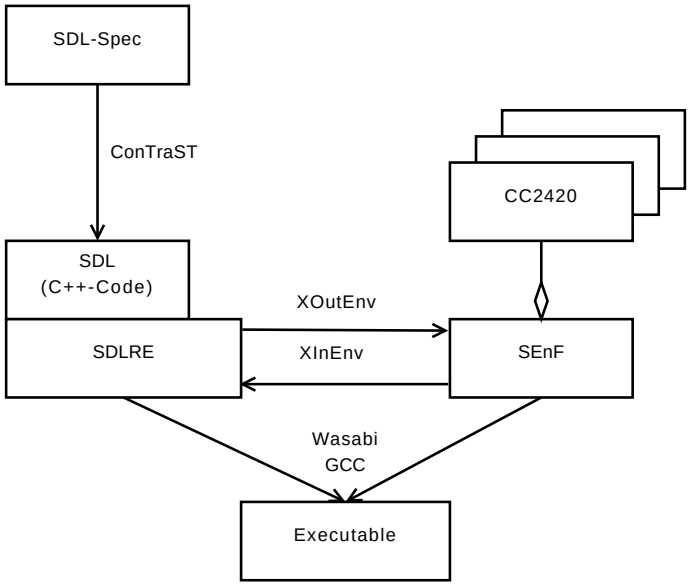


Abbildung 2.7: SDL-System mit SEnF für Imote2 [Sch07]

Abbildung 2.7 zeigt das Zusammenwirken aller verwendeter Software bis auf den Bootloader am Beispiel eines SDL-Systems für den Imote2. Zu Beginn steht die SDL-Spezifikation. Diese wird mit ConTraST nach C++ transpiliert. Die von der SDL-Spezifikation benötigten SEnF-Module werden eingebunden und zusammen mit der Laufzeitumgebung und dem transpilierten SDL-System mittels dem Wasabi Compiler zu einer ausführbaren Datei übersetzt. Diese Datei kann nun über die USB-Schnittstelle auf den Imote2 geladen und dort ausgeführt werden.

Kapitel 3

Integration des Sensorboards

In diesem Kapitel wird die Integration des Sensorboards in das SDL Environment Framework (SEnF) beschrieben, damit die Hardware des Sensorboards auf SDL-Ebene nutzbar wird. Die Integration ermöglicht das Spezifizieren von Diensten, die Sensoren oder Aktuatoren des Sensorboards nutzen, in SDL (siehe Kapitel 4). Die Integration der Imote2-Plattform wurde schon in [Sch07] besprochen. Das Kapitel unterteilt sich in drei Abschnitte. In Abschnitt 3.1 geht es um das SEnF-Modul, das dem SDL-System die Signale zur Ansteuerung des Sensorboards zur Verfügung stellt. Daneben gibt es noch ein SDL-Package, das das Interface zum Benutzer vereinfachen soll. In Abschnitt 3.2 geht es um die Kommunikation zwischen PIC und Imote2. Die Ansteuerung der einzelnen Sensoren und Aktuatoren mit dem PIC Mikrokontroller wird in Abschnitt 3.3 beschrieben.

3.1 SEnF-Modul Sensorboard

Dieses SEnF-Modul stellt dem Benutzer eine einfache Schnittstelle für die Benutzung des Sensorboards zur Verfügung. Es gibt folgende Signale von dem SDL-System in die Umgebung:

- BOARD_set(Integer,Boolean): Dieses Signal schaltet eine Komponente des Sensorboards ein oder aus. Der erste Parameter gibt die Komponente an, die möglichen Ziele sind: LEDs, der Piezoschallwandler, das Mikrofon, sowie Licht- und Temperatursensor. Der zweite Parameter des Signals gibt an, ob die Komponente an- oder ausgeschaltet werden soll. Licht- und Temperatursensor sind abschaltbar, um Energie zu sparen. Möchte man sie benutzen, müssen sie vorher aktiviert werden. Das Mikrofon kann nicht abgeschaltet werden, das Signal dient dazu, das Wandeln und Versenden von Audiodaten ein- und auszuschalten.
- BOARD_get(Integer): Das Signal fordert einmalig einen Sensorwert an. Der Typ des Sensors steht im Parameter des Signals. Mögliche Sensortypen sind der Bewegungsmelder, sowie Licht- und Temperatursensor.

Folgende Signale können aus der Umgebung zum SDL-System geschickt werden:

- BOARD_result(Integer,Integer): Dieses Signal ist die Antwort auf ein BOARD_get-Signal. Es enthält als ersten Parameter den Sensortyp, der das Ergebnis liefert, und als zweiten Parameter den gemessenen Wert. Im Falle des Temperatursensors ist

das die aktuelle Temperatur, im Falle des Lichtsensors der aktuelle Lichtwert und im Falle des Bewegungsmelders die Anzahl der registrierten Bewegungen seit dem letzten Abfragen (Polling).

- BOARD_audio(Octet_string): Wird das Mikrofon mit einem BOARD_set-Signal eingeschaltet, so beginnt die Abtastung. Eine vorher festgelegte Anzahl an Abtastwerten werden gesammelt und dann mit diesem Signal an das SDL-System geschickt. Der Parameter enthält die abgetasteten Audiodaten.

Neben der Bereitstellung der oben aufgeführten Signale enthält das SEnF-Modul die Kommunikation mit dem Sensorboard auf Imote2-Seite (siehe nächster Abschnitt). Zusätzlich zu dem SEnF-Modul gibt es ein SDL-Package, das die Benutzung des Sensorboards weiter vereinfachen soll. In diesem Package wird eine Warteschlange eingesetzt, um zu verhindern, dass der Benutzer dem Sensorboard zu viele Signale auf einmal schickt. Da der PIC die Daten im Vergleich zum Imote2 sehr viel langsamer abarbeiten kann, muss gewartet werden, bis die Übertragung eines Signals zum PIC abgeschlossen ist. Eine Warteschlange lässt sich in SDL sehr einfach mit Hilfe eines Timers und eines Save-Statements implementieren. Des Weiteren schaltet das SDL-Package Sensoren bei Bedarf an, so dass sich der Benutzer darum nicht mehr kümmern muss. Möchte der Benutzer beispielsweise die Temperatur anfordern, genügt es, ein BOARD_get-Signal für den Temperatursensor zu schicken. Das SDL-Package empfängt das Signal und schaltet den Temperatursensor mit einem BOARD_set-Signal ein. Danach muss eine kurze Zeitspanne gewartet werden, damit der Messwert des Temperatursensors zuverlässig ist. Dann wird die Temperatur mit einem BOARD_get-Signal abgefragt. Wurde die Temperatur ausgelesen, schaltet das Sensorboard den Sensor automatisch wieder ab.

3.2 Kommunikation zwischen Imote2 und Sensorboard

Um das Sensorboard nutzen zu können, müssen Imote2 und Sensorboard in geeigneter Form miteinander kommunizieren. Es gibt auf dem Sensorboard zwei Komponenten, mit denen der Imote2 direkt kommunizieren kann. Zum einen ist das der D/A-Wandler, zum anderen der PIC. Alle anderen Komponenten werden über den PIC angesprochen. Die Kommunikation zwischen den beiden Sensorboardkomponenten und dem Imote2 wird durch eine SPI-Schnittstelle realisiert. Die SPI-Schnittstelle funktioniert nach dem Master-Slave-Konzept, das bedeutet jede Kommunikation geht vom Master, in diesem Fall dem Imote2, aus. D/A-Wandler und PIC fungieren als Slaves an der SPI-Schnittstelle. Der Master wählt über die Chip-Select-Leitungen den Slave aus, mit dem er kommunizieren will. Der Austausch von Daten passiert bei einer SPI-Schnittstelle immer synchron. Dies bedeutet, wenn der Master einem seiner Slaves Daten schickt, empfängt er automatisch Daten vom Slave. Möchte der Master nur vom Slave lesen, so muss er ihm trotzdem Daten schicken. Dies können auch Fülldaten, wie zum Beispiel Nullbytes, sein. Eine SPI-Schnittstelle unterstützt verschiedene Taktraten und Framegrößen. Für die Kommunikation mit dem PIC ist die SPI-Schnittstelle mit 3,25 MHz getaktet, ein SPI-Frame ist ein Byte lang.

Die Kommunikation zwischen Imote2 und D/A-Wandler funktioniert nur in eine Richtung. Der D/A-Wandler nimmt lediglich Daten entgegen, sendet aber selbst keine Daten. Außerdem verarbeitet seine SPI-Schnittstelle nur 16-Bit Frames [Mic07a]. Damit man nicht immer zwischen eine Framegröße von 8 Bit und 16 Bit wechseln muss, werden immer nur 8-Bit Frames vom Master versendet. Der D/A-Wandler interpretiert dann zwei 8-Bit Frames als einen 16-Bit Frame. In dem 16-Bit Frame befindet sich der 12-Bit digitale Wert, der

gewandelt werden soll. Die übrigen vier Bit sind Steuerbits, mit denen beispielsweise der Ausgang des D/A-Wandlers ausgewählt wird. Empfängt der D/A-Wandler einen Frame, so beginnt er mit der Wandlung und hält den analogen Ausgangswert solange, bis er einen neuen Frame empfängt. Da die einzelnen digitalen Abtastwerte des Mikrofons nur 8 Bit breit sind, die Eingangsdaten des D/A-Wandlers aber 12 Bit breit sind, müssen die Werte um vier Bit nach links geschoben werden, um den Maximalwert und den Minimalwert des Wertebereichs des D/A-Wandlers zu erreichen. Werden Maximal- und Minimalwert des D/A-Wandlers nicht erreicht, ist das Ausgangssignal zu leise. Die Kommunikation über die SPI-Schnittstelle wird im Verlauf der Arbeit erfolgreich getestet, indem Audiodaten, die mit dem Mikrofon des Sensorboards aufgenommen wurden, zum D/A-Wandler geschickt und dort wieder in hörbare Audiosignale umgewandelt wurden.

Für die Kommunikation zwischen PIC und Imote2 wird ein einfaches Protokoll entwickelt. Die Kommunikation zwischen Imote2 und PIC gestaltet sich schwieriger als zwischen Imote2 und D/A-Wandler, da die Kommunikation in beide Richtungen erfolgt. Insbesondere muss beachtet werden, dass der PIC nur einen 1-Byte Empfangspuffer besitzt. Da die Geschwindigkeit der SPI-Schnittstelle (3,25 MHz) ungefähr gleich der Taktfrequenz des PIC (3 MIPS) ist, kann die SPI-Schnittstelle ihre volle Geschwindigkeit nicht nutzen, sondern muss zwischen den einzelnen übertragenen Bytes warten, bis der PIC das Byte aus dem Puffer gelesen hat und ein weiteres Byte empfangen kann. Folgende Anforderungen werden an das Kommunikationsprotokoll zwischen PIC und Imote2 gestellt:

- Umsetzung der empfangenen SDL-Signale BOARD_get und BOARD_set, so dass sie zum PIC gesendet werden können.
- Transfer von Sensor- und Audiodaten zum Imote2
- Mechanismus zur Benachrichtigung des Imote2, wenn dem PIC neue Sensorergebnisse vorliegen, da der Imote2 als Master die eigentliche Kommunikation initialisieren muss.
- Die Kommunikation muss so schnell sein, dass eine sinnvolle Abtastung des Audiosignals mit 8 kHz möglich ist. Diese Abastrate wird beispielsweise im Telefonnetz verwendet und stellt für Sprache einen guten Kompromiss zwischen Bandbreitenbedarf und Qualität dar.

Ein Rahmen, der zwischen Imote2 und PIC ausgetauscht wird, besteht immer aus zwei Byte. Das erste Byte enthält die Befehls- bzw. Sensorart, das zweite die Befehlsparameter bzw. das Ergebnis. Die Tabelle 3.1 enthält alle Befehle mit Parametern, die vom Imote2 zum PIC geschickt werden können.

Der erste Befehl fordert einen bestimmten Wert vom PIC an. Für die Anforderung eines Temperaturwertes ist das erste Bit gesetzt, für einen Wert des Bewegungsmelders das zweite Bit und für einen Wert des Lichtsensors das vierte Bit. Zu beachten ist, dass Temperatur- und Lichtsensor vorher eingeschaltet werden müssen. Der zweite Befehl steuert die LEDs, die an den PIC angeschlossen sind. Das zweite Byte diese Befehls wird als Bitmaske interpretiert und im Gesamten ausgewertet. Ist das Bit gesetzt, wird die entsprechende LED eingeschaltet, ist das Bit nicht gesetzt wird sie ausgeschaltet. Daher speichert sich der Imote2 den momentanen Zustand der LEDs, und setzt bei jedem BOARD_set-Signal die entsprechenden Bits. Der letzte Befehl schaltet Sensoren und Aktuatoren an oder aus. Für die jeweilige Aktion wird das entsprechende Bit gesetzt. Licht- und Temperatursensor müssen nicht gesondert ausgeschaltet werden, da sie nach Benutzung vom PIC automatisch deaktiviert werden.

Befehl	Kode (1.Byte)	Befehlsparameter (2.Byte)
Sensorwert anfordern	2	0 | 0 | 0 | 0 | Licht | 0 | Bewegung | Temperatur
LED	4	0 | 0 | 0 | 0 | 0 | LED3 | LED2 | LED1
Sensoren/Aktuatoren	6	Licht | Temperatur | Piezo | Mikrofon 0 | 0 | an | an | aus | an | aus | an

Tabelle 3.1: Rahmen vom Imote2 zum PIC

In der umgekehrten Richtung, vom PIC zum Imote2, werden die angeforderten Ergebnisse übertragen. Auch hier besteht ein Rahmen aus zwei Byte. Im ersten Byte steht, um welche Daten es sich handelt, im zweiten steht der eigentliche Wert. Tabelle 3.2 zeigt eine Übersicht über die möglichen Rahmen und den Wertebereich der Ergebnisse.

Sensor	Kode (1.Byte)	Wertebereich des 2.Bytes
Temperatursensor	1	Temperatur von 0 °C bis 60 °C
Mikrofon	2	Abtastwert zwischen -128 und 128
Bewegungsmelder	4	Anzahl der registrierten Bewegungen seit der letzten Abfrage (von 0 bis 255)
Lichtsensor	8	Lichtwert zwischen 0 und 169, je höher der Wert desto heller

Tabelle 3.2: Rahmen vom PIC zum Imote2

Als Signalisierungsmechanismus für das Vorliegen eines Sensorergebnisses, das zum Imote2 übertragen werden soll, wird eine Interruptleitung verwendet, die der PIC aktivieren kann. Der Imote2 löst einen Interrupt aus, wenn er einen Flankenwechsel von logisch Low nach logisch High entdeckt. Somit kann der PIC den Imote2 benachrichtigen, wenn er ein Wandlungsergebnis zum Imote2 übertragen möchte. Der Imote2 initialisiert dann die Kommunikation. Um die oben genannten Anforderungen zu realisieren, wird für jede beteiligte Seite je ein endlicher Automat entworfen, der das Kommunikationsprotokoll spezifiziert.

Abbildung 3.1 zeigt den Automaten für den Imote2. Anhand dieses Graphen erkennt man, dass eine Kommunikation zwischen Imote2 und PIC durch zwei verschiedene Ereignisse ausgelöst werden kann. Zum einen der Erhalt eines Signals BOARD_set oder BOARD_get vom SDL-System, dies wird im Zustandsautomaten mit dem Input *Command* dargestellt. Befindet sich der Imote2 im Zustand *Idle*, das heißt es findet gerade keine andere Übertragung statt, wird der entsprechende Rahmen anhand der Tabelle 3.1 zusammengestellt und das erste Byte an den PIC versendet. Gleichzeitig wird ein Hardware-Timer aufgezogen

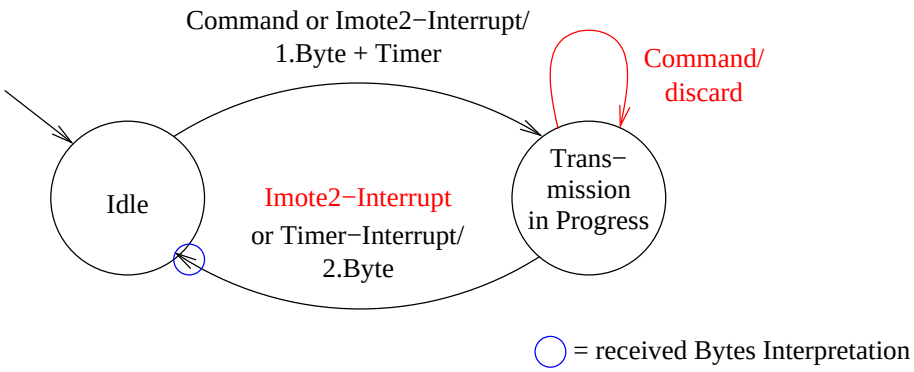


Abbildung 3.1: Zustandsautomat auf Imote2-Seite für Kommunikation zwischen Imote2 und PIC

und der Zustand gewechselt (Zustand *Transmission in Progress*). Aufgrund des begrenzten Empfangspuffers des PIC kann das zweite Byte des Rahmens nicht sofort gesendet werden, sondern es muss gewartet werden, bis der PIC das empfangene Byte aus dem Puffer gelesen und abgespeichert hat. Läuft der Timer ab, so wird ein Interrupt ausgelöst und das zweite Byte wird übertragen. Als Antwort auf einen Befehlsrahmen schickt der PIC Bytes, die nur Nullen enthalten oder ein Sensorergebnis, wenn dies dem PIC vorliegt. Ganz ähnlich läuft die Übertragung eines Ergebnisses vom PIC zum Imote2. Liegt dem PIC ein Ergebnis vor, das zum Imote2 übertragen werden soll, aktiviert der PIC die Interruptleitung zum Imote2. Die aktivierte Interruptleitung löst einen Interrupt im Imote2 aus, der als Input *Imote2-Interrupt* im Zustandsautomaten dargestellt wird, und die Übertragung beginnt. Da aufgrund der synchronen Schnittstelle Daten zum PIC übertragen werden müssen, werden Bytes versendet, die nur Nullen enthalten.

Unabhängig davon, welche Daten übertragen wurden, werden nach der Übertragung beider Bytes die empfangenen Bytes aus dem Empfangspuffer des Imote2 gelesen und anschließend ausgewertet (siehe Tabelle 3.2). Enthalten die empfangenen Bytes lediglich Nullen, so werden diese ignoriert. Werte vom Licht- und Temperatursensor, sowie vom Bewegungsmelder werden nach der Auswertung der empfangenen Bytes direkt als Signale (BOARD_result) an das SDL-System weitergegeben. Audiodaten werden bis zu einer maximalen Größe gesammelt und dann als gesammelte Bytefolge an das SDL-System geschickt (BOARD_audio). Fehler treten in diesem Zustandsdiagramm auf, wenn in dem Zustand *Transmission in Progress* Interrupts des PIC auftreten (Input *Imote2-Interrupt* im Zustandsautomaten rot markiert). Dies passiert, wenn die Übertragung des letzten Rahmens verzögert wurde (etwa durch hohe Last auf dem Knoten). Diese Art von Fehler tritt nur bei der Audiodatenübertragung auf, also genau dann, wenn in kurzer Zeit viele Rahmen zwischen PIC und Imote2 übertragen werden müssen. In diesem Fall wird das zweite Byte durch den Input *Imote2-Interrupt* gesendet, um die Übertragung zu beenden (siehe auch Zustandsautomat des PIC: Abbildung 3.2). Ein Fehler ist ebenfalls vorhanden, wenn ein neues Signal, das einen Befehl enthält, empfangen wird, wenn der momentane Übertragungsvorgang noch nicht abgeschlossen ist (rot markierter Zustandsübergang aus dem Zustand *Transmission in Progress*). Die Warteschlange im SDL-Treiber für das Sensorboard sollte dies verhindern (siehe Abschnitt 3.1), tritt der Fall trotzdem auf, so wird der neu empfangene Befehl verworfen.

Abbildung 3.2 zeigt den Zustandsautomaten auf PIC-Seite. Dieser hat mehr Zustände als der Automat des Imote2, da hier das Empfangen von Befehlen und die Übertragung von Ergebnissen gesondert behandelt werden müssen. Zustandsübergänge werden ausschließ-

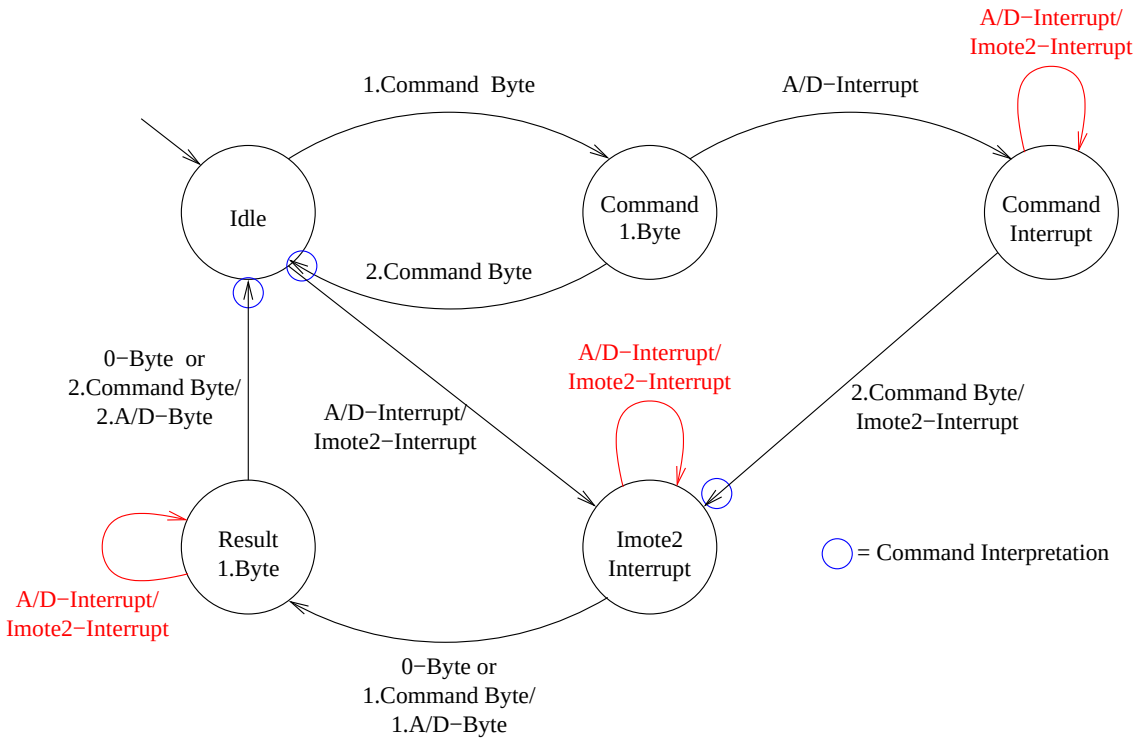


Abbildung 3.2: Zustandsautomat auf PIC-Seite für Kommunikation zwischen Imote2 und PIC

lich mit Hilfe von Interrupts durchgeführt. Für den Zustandsautomaten sind zwei Interrupts relevant. Zum einen der SPI-Interrupt des PIC, der ausgelöst wird, wenn über die SPI-Schnittstelle ein neues Datenbyte empfangen wurde (im Zustandsautomat steht als Eingabe des Zustandsübergangs immer das Byte, das mit dem SPI-Interrupt empfangen wurde). Zum anderen der A/D-Wandler Interrupt, der anzeigt, wenn der A/D-Wandler einen Wert gewandelt hat. Empfängt der PIC im Zustand *Idle* ein neues Byte über die SPI-Schnittstelle, so kann es sich dabei nur um einen Befehl vom Imote2 handeln. Der PIC liest das Byte aus dem Empfangspuffer, führt einen Zustandsübergang (Zustand *Command 1.Byte*) aus und wartet auf das zweite Byte des Befehls. Dies wird vom Imote2 erst dann gesendet, wenn der Timer des Imote2 abgelaufen ist (siehe oben). Wurde auch das zweite Byte erfolgreich empfangen, wird der Befehl ausgewertet und ausgeführt (Übersicht der Befehle siehe Tabelle 3.1).

Wird ein A/D-Wandler-Interrupt im Zustand *Idle* ausgelöst, bedeutet dies, dass das Ergebnis einer A/D-Wandlung nun vorliegt und zum Imote2 übertragen werden muss. Dies signalisiert der PIC dem Imote2 über die Interruptleitung, indem der PIC sie für einen Befehlstakt auf logisch High legt. Der Flankenwechsel löst dann den Interrupt auf dem Imote2 aus. Danach wartet der PIC im Zustand *Imote2 Interrupt* auf den Beginn der Übertragung, die nach oben beschriebenem Schema abläuft. Die zum PIC gesendeten Bytes enthalten keinen Befehl, sondern enthalten bis auf einen speziellen Fall immer Nullbytes. Der spezielle Fall tritt ein, wenn der Imote2 gerade beginnt, einen Befehl zum PIC zu senden, der PIC das erste Byte des Befehls noch nicht erhalten hat und ein A/D-Interrupt ausgelöst wird. Dann enthalten die vom Imote2 empfangenen Bytes einen Befehl und keine Nullbytes, dieser Befehl wird dann ausgewertet und ausgeführt.

Es kann passieren, dass während des Empfangs eines Befehls vom Imote2 (Zustand *Command 1.Byte*), ein A/D-Wandler-Interrupt ausgelöst wird. In diesem Fall wird die Übertra-

gung des Befehls zuerst abgeschlossen (Zustand *Command Interrupt*), und danach erst der Interrupt am Imote2 ausgelöst. Probleme treten auf wenn das Wandlungsergebnis nicht übertragen werden konnte, bevor ein neues Ergebnis vorliegt. In diesen Fall wird das alte Ergebnis verworfen und ein Interrupt beim Imote2 ausgelöst, damit das neue Ergebnis übertragen werden kann (rot markierte Zustandübergänge in den Zuständen *Command Interrupt*, *Imote2 Interrupt* und *Result 1.Byte*).

3.3 Ansteuerung der Hardware auf dem Sensorboard

Dieser Abschnitt handelt von der Anbindung der unterschiedlichen Sensoren und Aktuatoren an den PIC und davon wie diese Sensoren/Aktuatoren vom PIC aus gesteuert werden. Die LED-Anschlüsse sowie der Piezosummer sind mit E/A-Pins des PIC verbunden und können so über das Setzen und Löschen dieser E/A-Pins an- und ausgeschaltet werden. Sie benötigen keine weitere Ansteuerung und keine Auswertung.

Ebenso können der Licht- und der Temperatursensor über E/A-Pins aktiviert und deaktiviert werden.

3.3.1 A/D-Wandlung

Licht- und Temperatursensor sowie Mikrofon liefern analoge Spannungswerte. Bevor diese Werte zum Imote2 übertragen werden können, müssen sie digitalisiert werden. Dazu besitzt der PIC einen 10 Bit Analog/Digital-Wandler mit 10 verschiedenen Kanälen. Die für die Wandlung nötige Referenzspannung ist die Versorgungsspannung von 3 Volt. Dies ist auch der Spannungsbereich der Kanäle. Ein Wandlungsergebnis von 0 entspricht 0 Volt, ein Wandlungsergebnis von 1023 ($2^{10} - 1$) entspricht 3 Volt. Jeder der drei Sensoren mit analogem Ausgang ist mit einem Kanal des A/D-Wandlers verbunden. Durch Auswahl des entsprechenden Kanals und durch Setzen des Startbits wird die Wandlung gestartet. In der derzeitigen Konfiguration dauert das Wandeln eines analogen Wertes ca. 35 μs [Mic07b].

Von den 10 Bit des Wandlungsergebnisses werden nur 8 Bit genutzt, damit nur jeweils ein Byte Daten zum Imote2 übertragen werden muss. Unter dem Weglassen von zwei Bit leidet die Genauigkeit der Wandlung, jedoch reicht die restliche Genauigkeit für die Anwendungen aus. Bei einem Wertebereich von 0 bis 3 Volt erhält man mit 10 Bit eine Auflösung A pro Bit von:

$$A = 3\,\mathrm{V}/2^{10} \approx 2,9\,\mathrm{mV}$$

Wenn man nur 8 Bit verwendet, dann sinkt die Auflösung pro Bit auf:

$$A = 3\,\mathrm{V}/2^{8} \approx 11,7\,\mathrm{mV}$$

Eine Auflösung von 8 Bit reicht also immer noch aus, um kleine Spannungsveränderungen zu registrieren und so zu erkennen, dass Sensorwerte sich ändern.

3.3.2 Bewegungsmelder

Für die Auswertung des Bewegungsmelders wird ein Komparator des PIC eingesetzt. Der digitale Ausgang des Bewegungsmelders ist mit einem Pin des PIC verbunden. Der Komparator des PIC beobachtet diesen Pin und ist so konfiguriert, dass der Komparator einen

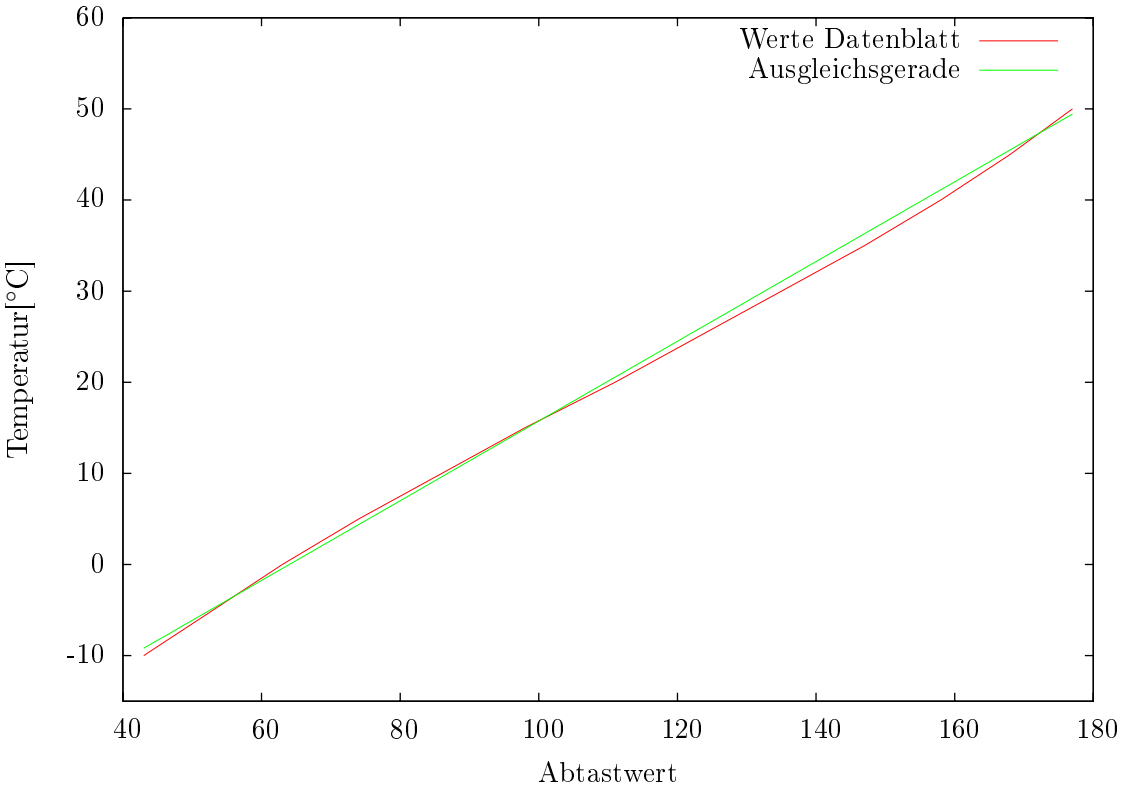


Abbildung 3.3: Temperaturkurven Temperatursensor Sensorboard

Interrupt auslöst, wenn die Spannung an diesem Pin einen bestimmten Wert übersteigt. Da der Bewegungsmelder bei jeder erkannten Bewegung einen Impuls auf seinen digitalen Ausgang anlegt [Hyg06], kann die Anzahl der erkannten Bewegungen bestimmt werden, indem man die Interrupts des Komparators zählt. Der Zählstand kann vom Imote2 aus abgefragt werden (Tabelle 3.1). Nach dem Abfragen des Zählers wird dieser wieder auf 0 gesetzt und die Zählung beginnt von Neuem. Somit wird immer die Anzahl an Bewegungen zurückgegeben, die seit dem letzten Abfragen registriert wurden. Für die Anforderung eines Wertes des Bewegungsmelders ist eigentlich keine A/D-Wandlung notwendig, aber es wäre ein zusätzlicher Zustand vonnöten, um auch im Zustand *Idle* Daten zu versenden. Um den Ablauf an den Ablauf der anderen Sensoren anzupassen, wird auch beim Bewegungsmelder eine Wandlung durchgeführt. Dann löst auch beim Bewegungsmelder der A/D-Interrupt die Datenübertragung aus. Allerdings ist das Ergebnis dieser Wandlung irrelevant, als Ergebnis dient stattdessen das Zählregister, das dann zum Imote2 übertragen wird.

3.3.3 Temperatursensor

Die Ergebnisse des Temperatursensors werden, bevor sie zum Imote2 geschickt werden, vorverarbeitet. Ziel ist es, aus dem Wandlungsergebnis eine Temperatur in Grad Celsius zu berechnen. Aus dem Datenblatt des Temperatursensors [ntc] lassen sich die Widerstandswerte des Sensors bei bestimmtem Temperaturen ablesen. Der Temperatursensor ist mit einem normalen Widerstand in Serie geschaltet und bildet somit mit diesem einen Spannungsteiler. Versorgt wird diese Schaltung mit der Versorgungsspannung von 3 Volt. Mit Hilfe der Spannungsteilerformel

$$U_{Temp} = U_{ein} \cdot \frac{R_{Temp}}{R_{Temp}+R}$$

kann der Anteil der Versorgungsspannung, der am Eingang des A/D-Wandlers anliegt, berechnet werden. Da Referenzspannung des A/D-Wandlers und Versorgungsspannung des Spannungsteilers gleich sind, kann mit diesem Anteil auch der Wert der Spannung nach der Wandlung bestimmt werden, indem man den Anteil der Spannung mit dem maximalen Ausgangswert des A/D-Wandlers multipliziert. In die Berechnung fließen nur die ersten 8 Bit des Wandlungsergebnisses mit ein. Somit lautet die vollständige Formel zur Berechnung des gewandelten Wertes W in Abhängigkeit von der Temperatur:

$$W = 255 \cdot \frac{R_{Temp}}{R_{Temp}+R}$$

Setzt man nun die Werte für R_{Temp} bei bestimmten Temperaturen aus dem Datenblatt ein, ergibt sich der Verlauf in Abbildung 3.3 (rote Kurve). Dieser Verlauf muss nun vom PIC umgesetzt werden, um einen Abtastwert in einen Temperaturwert umrechnen zu können. Das Speichern von vorberechneten Werten in einer Lookup-Tabelle auf dem PIC scheidet aufgrund der geringen Speicherkapazität des PIC aus. Also muss eine geeignete Funktion gefunden werden, die der PIC effizient berechnen kann. Dabei ist zu beachten, dass der PIC keine Gleitkommaarithmetik unterstützt und auch ganzzahlige Multiplikation nur sehr langsam ausführen kann. Die Gleichung der Regressionsgeraden für den Verlauf lautet:

$$y = 0,4328 \cdot x - 28,055$$

Ziel ist es nun, eine Gerade zu finden, die möglichst nah an der Regressionsgeraden liegt und nur mit Ganzzahloperationen berechenbar ist. Experimentelles Bestimmen liefert folgende Gerade (Abbildung 3.3 grüne Gerade):

$$y = \tfrac{1}{2} \cdot x - \tfrac{1}{16} \cdot x - 28$$

Die Abweichungen vom Originalverlauf der Temperaturkurve betragen im Messbereich im schlimmsten Fall nicht mehr als 1,5 °C (relative Abweichung nicht größer als 7 %). Der Vorteil der Geraden ist, dass sich die Multiplikationen sehr einfach mit Shiftbefehlen realisieren lassen. Eine zeitraubende Multiplikation findet nicht statt. Danach müssen lediglich zwei Subtraktionen durchgeführt werden, um das Ergebnis zu erhalten. Die Funktion wird in den PIC programmiert und getestet. Die Tests ergeben, dass bei 20 °C die gemessene Temperatur von der realen Temperatur um 5 °C abweicht. Dies könnte auf eine Nullpunktverschiebung des Sensors zurückzuführen sein. Um diese Verschiebung auszugleichen wird die folgende Funktion zur Bestimmung der Temperatur verwendet:

$$y = \tfrac{1}{2} \cdot x - \tfrac{1}{16} \cdot x - 33$$

3.3.4 Lichtsensor

Lichtwerte des Lichtsensors werden gewandelt und ohne weitere Bearbeitung an den Imote2 geschickt, da eine genaue Kalibrierung mit den vorhandenen technischen Mitteln nicht möglich ist. Das Datenblatt [TAO01] gibt die Zuordnung von Bestrahlungsstärke zu Spannung am Ausgang des Sensors an, aber es gibt momentan keine Möglichkeit die Bestrahlungsstärke zu messen. Daher werden die Rohdaten versendet. Eine spätere Umrechnung in Einheiten wie Lux oder Lumen kann mit den Rohdaten in der Anwendung erfolgen. Bei dem Lichtsensor wird das Wandlungsergebnis als positive Ganzzahl interpretiert, die Abtastwerte liegen zwischen 0 und 169. Der komplette Wertebereich des A/D-Wandlers wird nicht erreicht, da bei einer Eingangsspannung von 3 Volt die maximale Ausgangsspannung des Lichtsensors bei 2 Volt liegt [TAO01].

3.3.5 Mikrofon

Die gewandelten Werte des Mikrofons werden ebenfalls nicht weiter durch den PIC bearbeitet, sondern direkt an den Imote2 gesendet. Eine Nachbearbeitung ist nicht notwendig, da digitale Audiodaten gewöhnlich aus abgetasteten Spannungswerten bestehen. Der Null-Wert der Mikrofonschaltung liegt bei 1,5 Volt, deswegen wird ein Mikrofonabtastwert als vorzeichenbehaftete Ganzzahl interpretiert, somit liegt ein Abtastwert zwischen -127 und 128 (siehe Tabelle 3.2). Bei Stille schwanken die Werte wegen des immer vorhandenen Rauschens um ±3 um den Nullpunkt. Rauschen entsteht unter anderem durch verstärkte Aufnahme von Umgebungsgeräuschen und durch die thermische Bewegung von Ladungsträgern in elektronischen Schaltungen (sogenanntes Hintergrundrauschen) [Mül90].

Das Mikrofon unterscheidet sich von den anderen Sensoren dadurch, dass hier eine kontinuierliche Abtastung gewährleistet sein muss. Abtastwerte des Mikrofons werden nicht angefordert. Wird das Mikrofon vom Imote2 aus eingeschaltet, liefert es automatisch Werte der vorher eingestellten Abtastrate von 8 kHz. Dazu löst ein Hardwaretimer des PIC alle 125 μs einen Interrupt aus, der signalisiert, dass eine neue Wandlung gestartet werden muss.

3.3.6 Durchführung von Wandlungen

Ist das Mikrofon deaktiviert, werden alle Anfragen sofort bearbeitet, es muss nur darauf geachtet werden, dass nur eine Wandlung zur gleichen Zeit möglich ist. Dazu setzt jeder Befehl, der vom Imote2 empfangen wird und eine Anfrage enthält, ein Bit in einem Statusregister. Nach dem Setzen der Statusbits wird die nächste Wandlung ausgewählt, wenn nicht gerade eine andere Wandlung aktiv ist. Die Auswahl geschieht mit einer festen Priorität, was sich nicht nachteilig auswirkt. Die Abtastraten von Lichtsensor, Temperatursensor und Bewegungsmelder liegen nicht im Kilohertzbereich, weil es für die Anwendungen nicht sinnvoll ist, einen Wert mehrere tausendmal in der Sekunde abzufragen. Die Abtastraten für diese Sensoren liegen typischerweise um 1 Hz oder darunter. Der Lichtsensor ist aufgrund seiner technischen Eigenschaften nur für Umgebungslicht und nicht für hochfrequente Anwendungen wie zum Beispiel Fernbedienungen geeignet. Die Temperatur ändert sich sich nicht sprunghaft innerhalb einer Sekunde. Für den Bewegungsmelder ist es meist nur interessant, ob es Bewegungen in einem bestimmten Zeitraum gab. Somit genügen auch hier niedrige Abtastraten. Es bleibt genügend Zeit für alle Wandlungsvorgänge, auch wenn bestimmte Wandlungen anderen vorgezogen werden. Die ausgewählte Wandlung wird durchgeführt und ein Interrupt beim Beenden der Wandlung ausgelöst. Dann wird überprüft, ob noch weitere Wandlungen durchgeführt werden müssen, indem das Statusregister gelesen wird. Ist das der Fall, werden diese nun durchgeführt. Ist das entsprechende Statusbit für eine Wandlung gesetzt und kommt eine neue Anfrage für diesen Sensor, so wird trotzdem nur einmal ein Ergebnis bereitgestellt.

Erhöhter Koordinationsaufwand entsteht, wenn das Mikrofon aktiv ist und weitere Anfragen für andere Sensoren existieren. Angeforderte Wandlungen können nicht sofort durchgeführt werden, da sie möglicherweise mit Mikrofon-Wandlungen kollidieren. Die Wandlungen müssen zwischen den einzelnen Mikrofonwandlungen durchgeführt werden. Ein Wandlungsvorgang benötigt etwa ein Drittel der Zeit zwischen zwei Mikrofonabtastvorgängen. Es gilt:

$$3 \cdot T_{Wandlung} = 3 \cdot 35\,\mu\text{s} \leq 125\,\mu\text{s} = T_{Abtastintervall}$$

Um zusätzliche Wandlungen mit aktiviertem Mikrofon durchzuführen, schreibt eine Anforderung einer Wandlung ganz normal die entsprechenden Bits im Statusregister. Dann wird

jedoch im Gegensatz zu dem Fall, wenn das Mikrofon nicht aktiviert ist, keine Wandlung ausgeführt, sondern es wird auf die Beendigung einer Mikrofonwandlung gewartet, bevor eine andere Wandlung gestartet wird. Dies ist notwendig, um nicht mit einer eventuell auftretenden Mikrofonwandlung zu kollidieren, da Anfragen vom Imote2 nicht synchron zu den Mikrofonwandlungen sind. Zwischen zwei Mikrofonwandlungen wird maximal eine andere Wandlung ausgeführt, damit man noch genügend Zeit hat, das Ergebnis zum Imote2 zu übertragen, bevor die nächste Mikrofonwandlung gestartet werden muss. Wenn man nur die reine Wandlungsdauer eines Wertes betrachtet, so bliebe noch Zeit für eine zweite Wandlung zwischen zwei Mikrofonwandlungen. Jedoch wird noch zusätzliche Zeit benötigt, um den Zustandswechsel durchzuführen und die nächste Wandlung auszuwählen. Darüber hinaus muss bei einem Temperaturwert noch die Temperatur berechnet werden, was ebenfalls Zeit kostet.

Der gesamte Programmablauf des PIC, bis auf die Initialisierungsphase, wird über Interrupts gesteuert. Deswegen wird der PIC nur bei einem auftretenden Interrupt aktiv. Während der restlichen Zeit befindet er sich in einem Sleep-Modus. Dies hilft, den Energieverbrauch des PIC zu reduzieren.

Die Implementierung stellt dem Benutzer das Sensorboard mit allen Aktuatoren und Sensoren bis auf den D/A-Wandler zur Verfügung. Um das Sensorboard in einem SDL-System nutzen zu können, muss das entsprechende SDL-Package verwendet werden und das entsprechende SEnF-Modul geladen werden. Zusammen mit der schon ins SEnF integrierten Hardware [Sch07] besitzt die Imote2 Hardwareplattform nun die Möglichkeit aus einem SDL-System heraus als Sensorknoten genutzt zu werden.

Kapitel 4

Bereitstellung von AmICoM-Diensten auf der Imote2 Hardwareplattform

AmICoM steht für Ambient Intelligence Communication Middleware und ist eine Kommunikationsmiddleware für Ambient Intelligence Netzwerke. Ambient Intelligence bedeutet frei übersetzt „intelligente Umgebung“; sie interagiert mit dem Menschen und erbringt verschiedene Dienste für den Menschen. Die intelligente Umgebung entsteht durch die Vernetzung von Rechnern unterschiedlichen Typs. Sensoren sammeln Daten, die dann im Netzwerk verarbeitet werden und schließlich Aktuatoren ansteuern, die wieder Einfluss auf den Menschen haben. Die Recheneinheiten in einem Ambient Intelligence Netzwerk sind nicht homogen, sondern stark unterschiedlich. Ebenso unterscheiden sich die Kommunikationstechnologien der einzelnen Recheneinheiten untereinander. In diesem Netzwerk gibt es, wie oben schon erwähnt, Knoten, die für die Informationsbeschaffung verantwortlich sind und Knoten, die die vorhanden Informationen weiter verarbeiten. Nun müssen die Daten von einem Knoten auf den anderen Knoten gelangen. Dazu benötigt man neben der Kommunikationshardware eine geeignete Kommunikationsmiddleware, die, im optimalen Fall, auf jedem Knoten, unabhängig von der Rechenleistung und Kommunikationstechnologie des Knotens, eingesetzt werden kann [For].

Die AmICoM ist eine solche Kommunikationsmiddleware und verfolgt einen Ansatz, der auf verteilten Diensten basiert [KF07]. Ein Dienst stellt eine Leistung für jemand anderen bereit, der diesen Dienst dann in Anspruch nimmt, um die Leistung zu erhalten. Diese allgemeine Definition gilt auch für Dienste innerhalb der AmICoM. Im speziellen bietet ein Dienst auf dem Imote2 meist den Zugriff auf eine seiner Hardwarekomponenten an. Ein Temperaturdienst des Imote2 nutzt den Temperatursensor des Sensorboards, um den Dienstnutzern Temperaturwerte zu senden. Um die Verbindung zwischen Dienstanbieter und Dienstnutzer aufzubauen, kann eine Applikation auf einem Knoten einen Dienst in der AmICoM registrieren, die dann von einem Dienstnutzer genutzt werden kann, indem der Dienstnutzer den Dienst abonniert. Es ist dabei unerheblich, ob sich Dienstanbieter und Dienstnutzer auf ein und demselben Knoten befinden oder sich auf verschiedene Knoten im Netzwerk verteilen. Der Dienstanbieter und seine Dienstnutzer bilden nach erfolgreicher Registrierung und Abonnement zusammen eine Multicastgruppe, über die sie Daten untereinander austauschen. Wenn eine Applikation einen Dienst registriert, wird auf dem Knoten ein Dienstanbieter erstellt; wenn eine Applikation einen Dienst nutzen möchte, wird ein Dienstnutzer auf dem Knoten erstellt. Dienstanbieter und Dienstnutzer sind die Kommunikationsendpunkte für die Applikation. Die Kommunikation zwischen Dienstanbieter

und Dienstnutzer wird durch Heartbeats überwacht. Bei Heartbeats werden in regelmäßigen Abständen Nachrichten zwischen Dienstnutzer und Dienstanbieter ausgetauscht, so dass eine Unterbrechung der Kommunikation oder der Ausfall eines Knotens festgestellt werden kann. Wenn eine Wiederherstellung der Verbindung nicht möglich ist, werden die beteiligten Knoten benachrichtigt. Sollte der Knoten, der den Dienst bereitstellt, ausfallen, so kann dies durch das Ausbleiben des Heartbeats festgestellt werden und die Nutzer des Dienstes werden ebenfalls benachrichtigt.

In diesem Kapitel geht es um die Bereitstellung von AmICoM-Diensten auf dem Imote2. Das schon bestehende System für den PC wird auf den Imote2 übertragen, sodass statt eines PC auch ein Imote2 als AmICoM-Knoten verwendet werden kann. Hierzu wird die Architektur eines AmICoM-Knotens kurz vorgestellt. Danach wird die AmICoM für PC-Knoten vorgestellt und im Anschluss daran die portierte AmICoM für Imote2-Knoten. Im nächsten Abschnitt werden die während der Arbeit entwickelten AmICoM-Dienste, die die Sensoren und Aktuatoren des Imote2 Sensorboards nutzen, vorgestellt. Im letzten Abschnitt geht es um ASN.1 Datenstrukturen, die für den Datenaustausch zwischen Dienstanbieter und Dienstnutzer verwendet werden, um dem Nutzer eine einfache Schnittstelle zur Verfügung zu stellen.

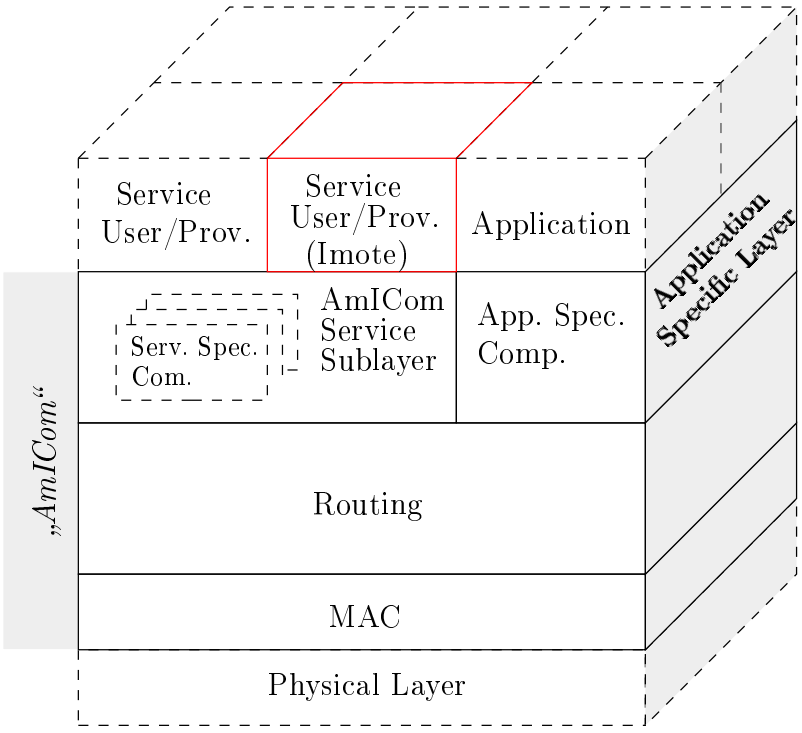


Abbildung 4.1: Architektur eines AmICoM-Knotens

4.1 Architekturüberblick AmICoM

Abbildung 4.1 zeigt die Architektur eines AmICoM-Knotens. Auf oberster Ebene befinden sich die Applikationen, sie kommunizieren mit der AmICoM und nehmen die Rolle eines Dienstanbieters und/oder Dienstnutzers an (*Service User/Provider* in der Abbildung). Die Applikationsschicht ist kein Teil der AmICoM, da die AmICoM nur die Middleware

zur Verfügung stellt und keine applikationsspezifischen Funktionalitäten. Diese Funktionalitäten befinden sich ausschließlich in der Applikationsschicht. Auf der nächsten Ebene befindet sich die AmICoM Serviceebene (*AmICoM Service Sublayer*), die für die Verwaltung der Dienstnutzer und Dienstanbieter zuständig ist. Beim Registrieren eines Dienstes oder dem Abonnieren eines Dienstes wird in dieser Ebene eine generische Repräsentation eines Dienstnutzers oder Dienstanbieters erzeugt. Diese Repräsentation übernimmt Managementaufgaben des Dienstes im AmICoM-Netzwerk. Ankommende Datenpakete werden hier den entsprechenden Diensten zugeteilt und es kann beispielsweise festgestellt werden, ob ein abonnierter Dienst noch im Netzwerk angeboten wird. Darunter befindet sich die Routingschicht, die die einzelnen Pakete im AmICoM-Netzwerk zu ihren Zielen leitet. Es ist nicht festgelegt, welches Routingprotokoll in der AmICoM verwendet wird. Die MAC-Schicht regelt den Zugriff auf das Medium, sie ist je nach verwendeter Kommunikationstechnologie in Hardware implementiert, oder es können in der AG entwickelte Protokolle [BGK07] verwendet werden. Die physikalische Schicht ist für die physikalische Übertragung der Daten auf dem Medium verantwortlich.

Neben den AmICoM Dienstanbietern und Dienstnutzern der AmICoM gibt es gibt es noch Applikationen, die nicht in das AmICoM-Netzwerk integriert sind. Sie existieren parallel neben der AmICoM und können auch gegebenenfalls Teile der Kommunikationsmiddleware wie Routing und MAC-Schicht mitbenutzen.

Die Hauptaufgabe dieser Arbeit besteht darin, die vorhandene Implementierung der AmICoM für den PC auf den Imote2 zu übertragen und für den Imote2 Basisdienste bereitzustellen, die auf den Sensoren und Aktuatoren des Sensorboards basieren. Diese Basisdienste sind in der Architektur im rot markiertem Block *Service User/Provider (Imote)* zu finden.

4.2 Benutzerschnittstelle der AmICoM

Anwendungen für die AmICoM können mit Hilfe einer einfachen Benutzerschnittstelle (API) entwickelt werden. Diese API ist für die Programmiersprachen C, C++ und Java verfügbar [FK07]. Die API besteht aus den folgenden Funktionen:

- bool: *Register*(string: name)

  Diese Funktion registriert einen Dienst, dessen Name eindeutig im gesamten Netzwerk sein muss, in der AmICoM. Wird ein Name übergeben, der schon im Netzwerk genutzt wird, schlägt die Registrierung fehl. Die Funktion gibt *false* zurück, wenn die Verbindung zur Instanz der AmICoM fehlschlägt.

- bool: *UnRegister*(string: name)

  Der Dienst mit dem übergebenen Namen wird abgemeldet und ist nicht mehr im Netzwerk verfügbar. Die Funktion gibt *false* zurück, wenn die Verbindung zur Instanz der AmICoM fehlschlägt.

- bool: *Subscribe*(string: name)

  Die Applikation versucht, den Dienst mit dem übergebenen Namen zu abonnieren. Ist dies erfolgreich, können Nutzer und Anbieter des Dienstes über das AmICoM-Netzwerk Daten austauschen. Die Funktion gibt *false* zurück, wenn die Verbindung zur Instanz der AmICoM fehlschlägt oder wenn der entsprechende Dienst nicht vorhanden ist.

- bool: *UnSubscribe*(string: name)

  Mit dieser Funktion zieht die Applikation ihr Abonnement des Dienstes mit dem übergebenen Namen zurück. Die Applikation erhält dann keine Daten mehr von dem entsprechenden Dienst. Die Funktion gibt *false* zurück, wenn die Verbindung zur Instanz der AmICoM fehlschlägt oder kein Dienst mit dem entsprechenden Namen abonniert wurde.

- bool: *Send*(string: name, bytes: data)

  Diese Funktion sendet ein Datenpaket über das Netzwerk. Jeder Dienstnutzer und der Dienstanbieter des entsprechenden Dienstes erhält das Datenpaket.

- packet: *Receive*()

  Ein mit *Send* versendetes Datenpaket kann mit dieser Funktion empfangen werden. Die Funktion gibt ein Paket zurück, das den Dienstnamen und die empfangenen Daten enthält.

- list: *QueryServices*()

  Diese Funktion liefert eine Liste mit allen im Netzwerk verfügbaren Diensten zurück.

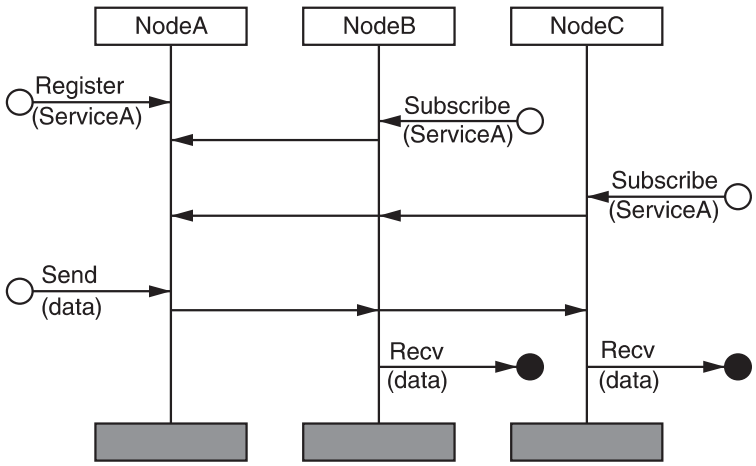


Abbildung 4.2: AmICoM-Szenario: Registrieren und Nutzen von Diensten auf verschiedenen Knoten [KF07]

Neben der Funktion *Receive* können Callback-Funktionen genutzt werden, die einen sofortigen Empfang von Daten und einen asynchronen Empfang von Ereignissen, wie das Verschwinden eines Dienstes, ermöglichen. Abbildung 4.2 zeigt ein typisches Szenario in der AmICoM. Eine Applikation registriert einen Dienst auf einem Knoten im AmICoM-Netzwerk. Andere Applikationen abonnieren diesen Dienst. In diesem Fall fungiert Knoten A als Dienstanbieter, Knoten B und C als Dienstnutzer. Daten, die der Dienst auf Knoten A versendet, werden durch das Netzwerk transportiert und von Knoten B und C empfangen.

4.3 AmICoM für PC-Plattform

Eine Realisierung der in Abschnitt 4.1 vorgestellten AmICoM-Architektur für die PC-Plattform wurde in der AG entwickelt und wird an dieser Stelle kurz vorgestellt. Große

Teile der AmICoM-Architektur werden in einem SDL-System repräsentiert. Die oberste Ebene des SDL-Systems sieht man in Abbildung 4.3. Der *AmICoM Service Sublayer* aus der Architektur wird im SDL-System durch den Block *DistributedUserProviderMiddleware* instantiiert. In diesem Block wird pro registriertem und pro abonniertem Dienst ein SDL-Prozess kreiert. Daten aus dem Netzwerk werden grundsätzlich an alle diese Prozesse verteilt. Aufgrund des Dienstnamens kann der Prozess nun entscheiden, ob die Daten zu seinem Dienst gehören und er die Daten an die entsprechende Applikation weiterleiten muss. Jeder registrierte oder abonnierte Dienst sendet ein periodisches Signal aus (Alive für Dienstanbieter, Subscribed für Dienstnutzer), um zu signalisieren, dass der Dienst noch verfügbar beziehungsweise noch abonniert ist (Heartbeat). Ein Dienstnutzer beobachtet dieses periodische Signal und weiss somit, ob der Dienst, den er abonniert hat, noch verfügbar ist. Zusätzlich gibt es in diesem Block noch einen Prozess, der das gesamte Netzwerk beobachtet, indem er alle periodischen Signale von Dienstanbietern empfängt. Aufgrund dieser Signale besitzt der Prozess eine aktuelle Liste mit allen im Netzwerk verfügbaren Diensten. Ebenfalls in diesem Block befindet sich ein CoDec (Kodierer/Dekodierer), der alle in der *DistributedUserProviderMiddleware* verwendeten Signale in eine Bytefolge übersetzt und über das Signal LAN_send an die untere Ebene weitergibt. In der umgekehrten Richtung wird eine mittels LAN_recv empfangene Bytefolge wieder in das entsprechende Signal umgewandelt.

Der Block *Flooding* enthält das Routing des AmICoM-Systems. Er spezifiziert ein Fluten von Paketen im Netzwerk. Fluten ist ein einfacher Algorithmus zur Verteilung von Paketen in einem Netzwerk, dabei wird jedes empfangene Paket auf allen verfügbaren Netzwerkschnittstellen mittels Broadcast verteilt. Somit erreicht jedes Paket jeden Knoten im Netzwerk. Um zu verhindern, dass Pakete unendlich oft innerhalb des Netzwerks versendet werden und das Netzwerk somit irgendwann zusammenbricht, darf jeder Knoten ein Paket nur einmal weiterleiten. Der Knoten muss sich Pakete, die er schon weitergeleitet hat, merken, damit er sie nicht noch einmal weiterleitet. Dazu muss jedes Paket eindeutig im Netzwerk identifizierbar sein. Dies wird durch die Verwendung von Sequenznummern und Hostnamen erreicht. Beide Informationen werden neben den Daten wiederum in eine Bytefolge kodiert und können dann versendet werden. Ebenfalls enthält der Fluter noch einen Fragmentierer und Defragmentierer, der die Pakete, falls sie die maximale Größe (Maximum Transmission Unit = MTU), die über das Netzwerk übertragen werden kann, überschreiten, fragmentieren und bei Empfang wieder zusammensetzen kann. Die MTU ist je nach verwendeter Kommunikationstechnologie unterschiedlich und kann bei Systemstart gesetzt werden.

Die MAC-Schicht ist auf dem PC vollständig in SEnF beziehungsweise in Hardware implementiert und kommt deswegen nicht im SDL-System vor. Der Fluter übergibt die gegebenenfalls fragmentierten Datenpakete dem SEnF-Modul, das sie dann auf dem jeweiligen Medium versendet. Die momentan unterstützten Kommunikationstechnologien auf der PC-Plattform sind Ethernet und WLAN.

Der Block *ApplicationAdaptation* dient zur Anbindung von Applikationen an das SDL-System. Dazu wird das in der AG entwickelte VSApplicationInterface verwendet. Diese Schnittstelle setzt aus dem SDL-System gesendete Signale in Funktionsaufrufe der Applikation um und umgekehrt. Mittels lokaler UDP-Sockets kommuniziert die Schnittstelle mit der Applikation und kann so mit dieser Daten austauschen. Zusätzlich gibt es ein AmICoM-Interface, das auf das VSApplicationInterface aufsetzt und dem Benutzer die in Abschnitt 4.2 vorgestellte API zur Verfügung stellt.

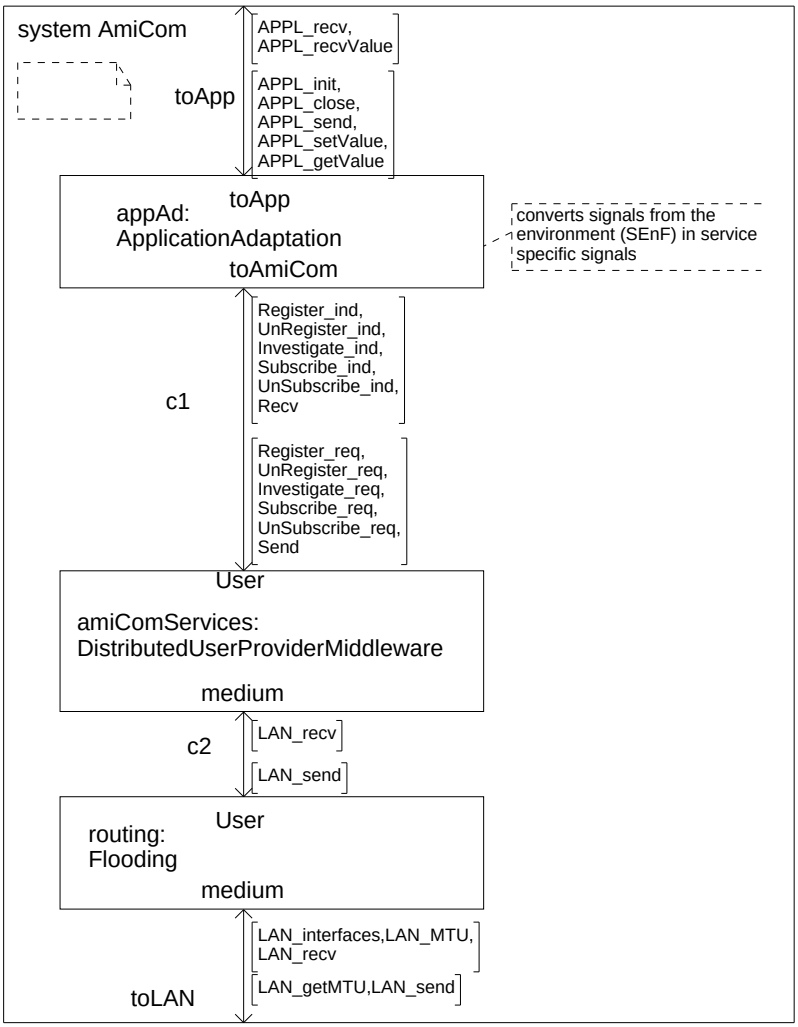


Abbildung 4.3: SDL-System AmICoM für PC-Plattform

4.4 AmICoM für Imote2-Plattform

Im Vergleich zum Entwurf für den PC ergeben sich für den Entwurf der AmICoM für den Imote2 Änderungen. Abbildung 4.4 zeigt die oberste Ebene des SDL-Systems eines Imote2-AmICoM-Knotens. Gleich geblieben sind die beiden Blöcke *DistributedUserProviderMiddleware* und *Flooding*. Das bedeutet, dass der *AmICoM Service Sublayer* bei beiden Entwürfen gleich bleibt. Änderungen ergeben sich in der MAC-Schicht, da für die Kommunikation zwischen zwei Knoten im Imote2-Netzwerk der CC2420 des Imote2 verwendet wird. Der Block *MacImote* wandelt die Signale, die mit dem Block *Flooding* ausgetauscht werden, in Signale um, die vom SEnF-Modul des CC2420 empfangen werden können. Dies ist nötig, um den Block *Flooding* aus dem PC-Entwurf übernehmen zu können. Der Block *MacImote* realisiert elementare MAC-Funktionalitäten. Ein Paket wird nur gesendet, wenn das Medium frei ist. Ist das Medium besetzt, wird das Paket im SEnF-Modul verworfen. Ob der Sendevorgang begonnen hat, wird der MAC-Schicht über ein Signal mitgeteilt. Wurde mit dem Senden nicht begonnen, kann die MAC-Schicht entscheiden, ob und wann sie versucht das Paket erneut zu übertragen. Aus Zeitgründen wurde eine Neuübertragung noch nicht in die MAC-Schicht integriert. Wurde mit dem Senden eines Pakets begonnen,

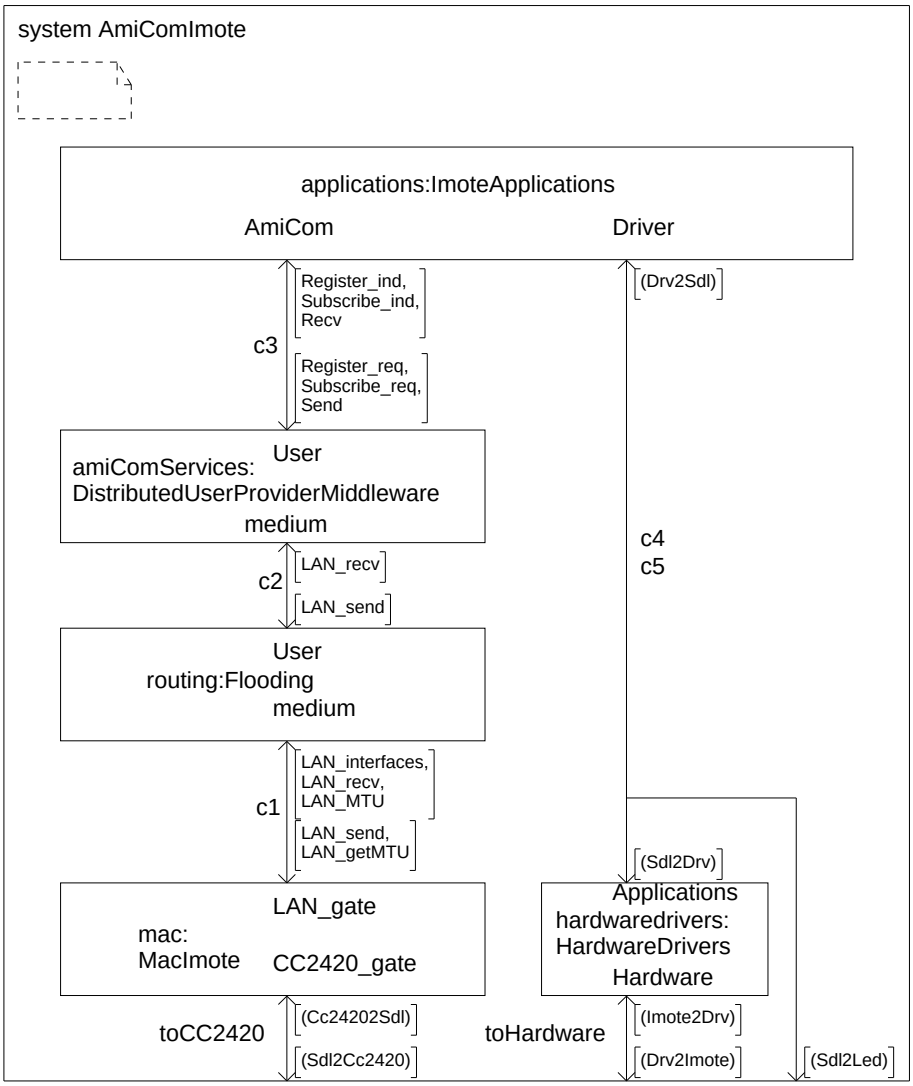


Abbildung 4.4: SDL-System AmICoM für Imote2-Plattform

sendet die MAC-Schicht nur dann ein neues Paket an den CC2420, wenn die Bestätigung, dass das Senden des letzten Pakets abgeschlossen wurde, empfangen wurde. Pakete, die während des Sendevorgangs in der MAC-Schicht eintreffen und ebenfalls gesendet werden sollen, werden in der MAC-Schicht in einer Warteschlange gespeichert. Dieses Vorgehen verhindert, dass mehr als ein Paket an das SEnF-Modul des CC2420 weitergegeben wird. Außerdem findet in der MAC-Schicht eine Auswertung des in Hardware implementierten CRC-Checks des CC2420 statt. Ist der CRC-Check fehlgeschlagen, wird das Paket in der MAC-Schicht verworfen. In Zukunft ist geplant, die jetzige Implementierung der MAC-Schicht durch MacZ, eine in der AG entwickelte MAC-Schicht [Bec06, Chr08], zu ersetzen. MacZ besitzt umfangreiche MAC-Funktionalitäten und wurde speziell für mobile Ad-Hoc-Netzwerke entwickelt.

Ebenfalls anders als im PC-Entwurf ist die Applikationsebene realisiert. Auf dem PC wird die Applikation in Java oder C/C++geschrieben, auf dem Imote2 werden auch die Applikationen als SDL-System realisiert. Die Realisierung von Applikationen in C++, die über ein ApplicationInterface ähnlich dem PC mit dem SDL-System verbunden sind, ist eben-

falls möglich, mit dem Unterschied, dass aufgrund des fehlenden Betriebssystems auf dem Imote2 keine Threads zur Verfügung stehen. Stattdessen muss der Scheduler der SDL-Laufzeitumgebung die Koordination übernehmen. Die Applikation wird zum SDL-System hinzugelinkt und mit Hilfe von C++-Funktionen innerhalb eines SDL-Prozesses aufgerufen. Die Applikation muss ihren Funktionsaufruf nach angemessener Zeit beenden, damit dem SDL-System genügend Rechenzeit zur Verfügung steht. Da bei dieser Lösung die Anwendung quasi Teil des SDL-Systems wird, liegt es nahe, die Applikation ebenfalls in SDL zu spezifizieren. Dieser Weg wurde für die Entwicklung der Basisdienste, die auf den Sensoren und Aktuatoren des Sensorboards basieren, gewählt.

Für die Ansteuerung der Hardware des Imote2 existiert der Block *HardwareDrivers*, der bis jetzt den SDL-Treiber des Sensorboards enthält. Der SDL-Treiber kommuniziert mit dem Sensorboard-Modul des SEnF und ermöglicht vollen Zugriff auf die gesamte Hardware des Sensorboards wie in Kapitel 3 beschrieben. Kommen neue Dienste mit anderen Hardwarekomponenten für den Imote2 hinzu, können weitere Treiber in diesen Block integriert werden. Daneben gibt es noch Hardware, die keinen zusätzlichen SDL-Treiber benötigt und direkt mit dem SEnF kommuniziert, wie in diesem Beispiel die LEDs.

Abbildung 4.5 zeigt den Block *ImoteApplications*, der der obersten Ebene in der AmICoM-Architektur entspricht, genauer. Dieser Block besitzt zwei Schnittstellen zu seiner Umgebung, zum einen das Gate *AmICoM*, über das Signale mit der AmICoM-Serviceebene ausgetauscht werden, zum anderen das Gate *Driver*, über das die Signale mit den Hardwaretreibern ausgetauscht werden. Der Block *ServiceMultiplexer* ordnet die Pakete der verschiedenen Dienste den entsprechenden Applikationen zu. Wenn die Applikation den jeweiligen Dienst registriert, speichert sich der *ServiceMultiplexer* die ProzessID des registrierenden Prozesses, um den Prozess dem Dienst zuordnen zu können. Diese Funktionalität wird in der PC-Implementierung vom *VSApplicationInterface* übernommen, das auf dem Imote2 nicht vorhanden ist. Im Block *SensorboardServices* finden sich alle implementierten Dienste, die das Imote2-Sensorboard nutzen. Die Dienste sind Basisdienste, die einfache Funktionalität besitzen und in den folgenden Abschnitten genauer vorgestellt werden. Der Block *SensorboardMultiplexer* ordnet die Ergebnisse der unterschiedlichen Sensoren des Sensorboards den einzelnen Diensten zu. Innerhalb des Blocks werden die BOARD_result-Signale in die dienstspezifischen Signale Temperature, Light, Movement und Audio umgewandelt, diese können dann direkt zu den entsprechenden Diensten gesendet werden. Ohne die Umbenennung müsste die Zuordnung von Sensorboardergebnissen zu den Diensten über die ProzessID erfolgen.

Die individuellen Dienste eines Imote2-Knotens, befinden sich im Block *Application*. Vorstellbar ist hier die Aggregation von Diensten, die dann wiederum als eigener Dienst angeboten wird. Beispielsweise könnte eine Aggregation daraus bestehen, von verschiedenen Imote2-Knoten den Temperaturdienst zu abonnieren, daraus eine Durchschnittstemperatur zu berechnen, und diese wiederum als eigenen Dienst im Netzwerk anzubieten. Momentan sind noch keine weiteren Dienste für den Imote2 vorhanden, deswegen enthält der Block lediglich eine Stellvertreter-Applikation, die keine Dienste bereitstellt. Möglich ist in diesem Block auch die Verwendung des oben beschriebenen ApplicationInterfaces für den Imote2. In Zukunft soll von einem Anwendungsentwickler für die AmICoM nur noch dieser Block und, bei Verwendung einer speziellen Hardware, der Block *HardwareDriver* verändert werden. Der Rest des Systems ist auf allen Imote2 gleich.

Es lässt sich zusammenfassen, dass, um die AmICoM auf dem Imote2 lauffähig zu machen, kaum Änderungen des SDL-Systems nötig waren. Dies ist der Tatsache zu verdanken, dass wir in der Lage sind, direkt aus einer SDL-Spezifikation den Kode für die entsprechende Hardwareplattform zu generieren. Änderungen waren nur nötig, weil eine andere Kommu-

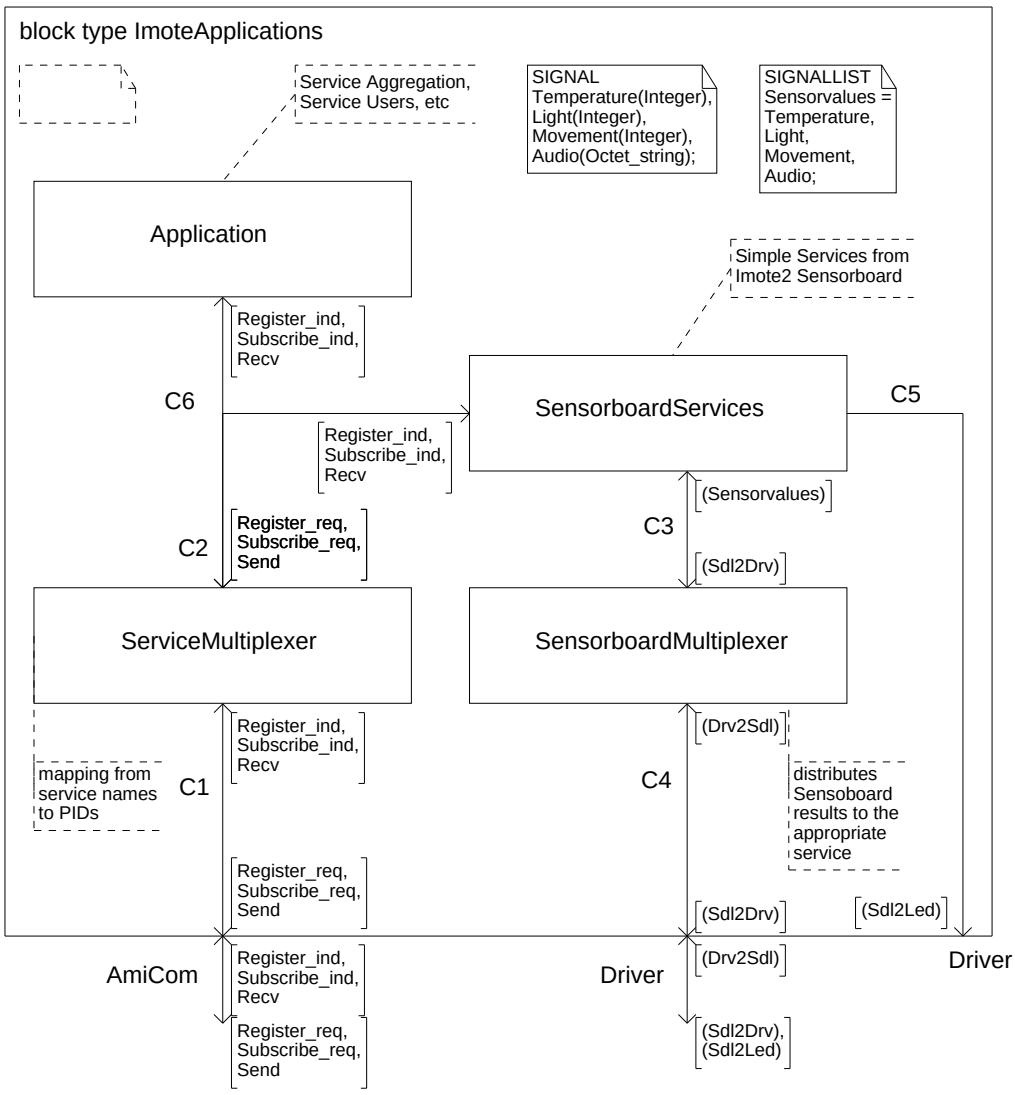


Abbildung 4.5: SDL-System Imote2-Implementierung AmICoM: Block *ImoteApplications*

nikationstechnologie benutzt wird und weil die Anbindung von Applikationen im Vergleich zur PC-Plattform unterschiedlich realisiert wird.

4.5 ASN.1 Datenstrukturen für AmICoM-Dienste

Um dem Benutzer eine plattformunabhängige Datenschnittstelle für AmICoM Dienste bereitzustellen, benötigt man eine plattformunabhängige Spezifikationssprache, mit der man diese Schnittstelle spezifizieren kann. Die Abstract Syntax Notation One (ASN.1) ist eine solche Spezifikationssprache [Lar99]. Mit ihr können abstrakte Datenstrukturen definiert und so eine einheitliche Schnittstelle geschaffen werden. ASN.1 ist eine von der ITU-T (International Telecommunication Union - Telecommunication Standardization Sector) und der ISO (International Standards Organisation) standardisierte Beschreibungssprache [ITU02a]. Eine ASN.1 Spezifikation stellt Datentypen bereit, die in einer Sprache, dies kann eine Beschreibungssprache wie SDL oder eine Programmiersprache wie C++ sein, instantiiert und mit Werten belegt werden. Diese Instantiierung ist abhängig von der verwen-

deten Sprache. Um die Werte von abstrakten Datentypen auf einem Medium zu versenden, müssen diese in ein einheitliches sprachunabhängiges Format übersetzt werden, damit die Plattformunabhängigkeit gewährleistet ist. ASN.1 definiert verschiedene *Encoding Rules*, die für jeden Datentyp in ASN.1 vorschreiben, wie diese als Bytefolge zu kodieren ist. Diese Bytefolge wird dann über ein Medium versendet und auf der Gegenseite wieder in die ursprünglichen Werte dekodiert. Die Verwendung von plattformunabhängigen Datentypen ist zum Beispiel dann von Vorteil, wenn eine der Plattformen eine andere Bytereihenfolge nutzt als die andere. Ein anderes Beispiel ist die unterschiedliche Kodierung von Zeichenketten auf verschiedenen Plattformen. All diese Kodierungsprobleme umgeht man mit der Verwendung von abstrakten Datentypen, da die Enkodierung und Dekodierung von Werten der Datentypen standardisiert ist.

Die mit ASN.1 definierten abstrakten Datenstrukturen können mit Telelogic Tau direkt in eine PR-Datei umgewandelt werden. Diese PR-Datei enthält eine Repräsentation der ASN.1 Datentypen als SDL-Datentypen. Diese SDL Datentypen können dann wie gewohnt in der SDL-Spezifikation verwendet werden. Tau generiert ebenfalls Enkodierungs- und Dekodierungsfunktionen für diese Datentypen. ConTraST generiert aus den Definition automatisch Kode und unterstützt sowohl die *Basic Encoding Rules* (BER) [ITU02b], als auch die *Packed Encoding Rules* (PER) [ITU02c]. Für andere Programmiersprachen, wie C und Java, gibt es ASN.1 Compiler [ope, asn], die eine ASN.1 Definition in die jeweilige Programmiersprache übersetzen. Mit einer Laufzeitumgebung, die auch Enkodierer und Dekodierer bereitstellt, können die ASN.1 Datentypen dann in der Programmiersprache genutzt werden. Das in dieser Arbeit verwendete ASN.1-Paket für Java wird in Abschnitt 7.2.1 vorgestellt.

Mit den abstrakten Datentypen von ASN.1 werden die Pakete, die zwischen Dienstnutzer und Dienstanbieter ausgetauscht werden, spezifiziert. Als Beispiel sei hier der LED-Dienst eines Imote2 genannt. Dieser Dienst ermöglicht eine Fernsteuerung der LEDs des Imote2. LEDs können ein- oder ausgeschaltet werden und mit einer bestimmten Frequenz blinken. Die Kommunikation erfolgt bei diesem Beispiel nur vom Dienstnutzer zum Dienstanbieter, da die LEDs keine Daten senden. Folgender ASN.1 Datentyp wird für die Ansteuerung der LEDs verwendet:

```
1 LedPacket ::= SEQUENCE {
2   color    INTEGER, -- Farbe der LED
3   on       BOOLEAN, -- LED an oder aus
4   interval INTEGER  -- Blinkfrequenz in Sekunden (0 kein blinken)
5 }
```

Der Datentyp SEQUENCE spezifiziert eine geordnete Abfolge von anderen Datentypen. In diesem Fall ist der Datentyp *LedPacket* eine Folge von einem Wert vom Typ INTEGER, einem Wert vom Typ BOOLEAN und wiederum einem INTEGER-Wert. Mit dem ersten Wert vom Typ INTEGER wird die Farbe der LED kodiert, die angesteuert werden soll. Der Wert vom Typ BOOLEAN gibt an, ob die LED an- oder ausgeschaltet werden soll. Der Wert des INTEGER-Datentyp *interval* gibt die Blinkfrequenz einer LED in Sekunden an. Ist dieser Wert von 0 verschieden, so wird der BOOLEAN-Wert nicht beachtet und die LED mit der gewünschten Frequenz umgeschaltet.

Da der LED-Dienst keine Daten sendet, sondern nur empfängt, gibt es auch nur eine Paketdefinition. Anders sieht dies bei einem Temperaturdienst aus. Hier gibt es für die Anfrage und die Antwort des Dienstes ein Paket. Die Definition lautet:

```
1 NtcPacket ::= INTEGER
2
3 NtcDataPacket ::= INTEGER
```

Der Temperaturdienst ist in der Lage, periodisch Temperaturwerte zu liefern, ohne dass der Benutzer diesen immer wieder anfordern muss. Den Zeitabstand in Sekunden zwischen zwei Temperaturwerten übergibt der Benutzer dem Dienst im *NtcPacket*. Ist der übergebene Wert 0, so wird nur einmal ein Temperaturwert geliefert. Das *NtcDataPacket* enthält die Temperaturwerte in Grad Celsius, die der Dienst an seine Dienstnutzer versendet.

Auf diese Weise lassen sich für jeden Dienst die Pakete spezifizieren, die zwischen Anbieter und Nutzer ausgetauscht werden sollen. Die vollständige ASN.1-Spezifikation für alle Sensorboard-Dienste findet sich im Anhang A.

4.6 Sensorboard-Dienste für die AmICoM

Im letzten Abschnitt wurden die Schnittstellen von AmICoM-Diensten vorgestellt. In diesem Abschnitt geht es um die implementierten Dienste, die auf den Sensoren oder Aktuatoren des Imote2 Sensorboards basieren. Jeder Dienst ist als eigenständiger Prozess, der sich im Block *SensorboardServices* (siehe Abschnitt 4.4) befindet, auf jedem Imote2 mit Sensorboard verfügbar. Folgende Dienste sind implementiert worden:

- LED Dienste: Es können mit zwei verschiedenen Diensten sowohl die LEDs des Imote2, als auch die LEDs des Sensorboards angesprochen werden. Wie oben schon erwähnt, kann man LEDs an- und ausschalten, sowie mit einer bestimmten Frequenz blinken lassen.
- Temperatur: Es ist möglich, die Temperatur nur einmal abzurufen, als auch ein Intervall anzugeben, in dem dann je ein Temperaturwert geliefert wird.
- Lichtsensor: Auch hier ist sowohl das Abfragen eines einzelnen Wertes, als auch eine Intervallangabe möglich.
- Bewegungsmelder: Es ist ebenfalls eine Intervallangabe oder eine Einzelabfrage möglich.
- Piezosummer: Der Piezosummer kann einmalig aus- und eingeschaltet werden, sowie mit einer Frequenz umgeschaltet werden.
- Mikrofon: Das Mikrofon kann ein- und ausgeschaltet werden. In der Paketdefinition ist schon vorgesehen, die Abtastfrequenz beim Einschalten mit zu übergeben, allerdings unterstützt der Dienst keine von 8 kHz verschiedenen Abtastfrequenzen. Es wird die voreingestellte Abtastrate von 8 kHz verwendet.

Anhand eines Beispiels soll gezeigt werden, wie ein Sensorboard-Dienst des Imote2 arbeitet. In den Abbildungen 4.6 und 4.7 sieht man die Spezifikation des Prozesses *Temperature*, der den Temperaturdienst enthält. Jeder Dienst im AmICoM-Netzwerk benötigt einen eindeutigen Namen. Da die Sensorboard-Dienste auf jedem Imote2 zur Verfügung stehen sollen, benötigt man noch ein zusätzliches Unterscheidungskriterium. Der Dienstname setzt sich zusammen aus der eigentlichen Beschreibung, in diesem Fall *NTC* für den Temperaturdienst, und dem Hostnamen des Imote2. Mit Hilfe des Hostnamens kann man gleiche Dienste auf unterschiedlichen Imote2 unterscheiden. Für den Hostnamen wurde eine eindeutige ID im FLASH-Speicher jedes Imote2 abgelegt, diese wird ausgelesen und daraus der Hostname bestimmt. Der Dienstname des Temperaturdienstes auf dem Imote2 mit der ID 3 lautet beispielsweise *IM3.NTC*. Weil der Dienstname in jedem Paket versendet wird,

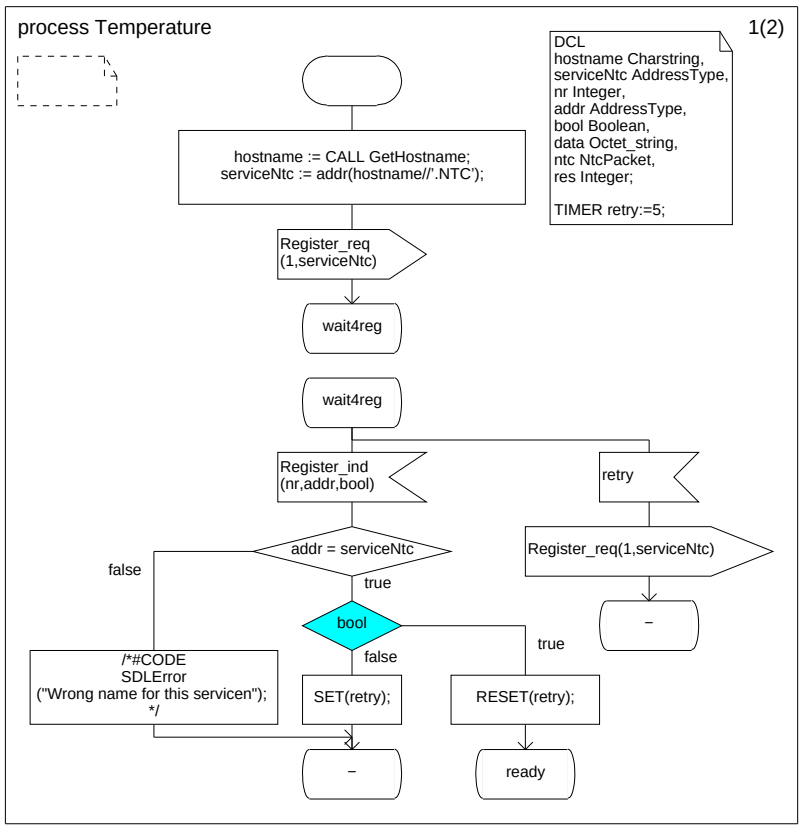


Abbildung 4.6: Prozess *Temperature* (Seite 1)

ist es sinnvoll, nicht zu lange Dienstnamen zu wählen, da der Transceiver des Imote2 nur eine stark begrenzte Bandbreite von 250 kBit/s zur Verfügung stellt.

In der Starttransition des Prozesses wird zunächst der Hostname bestimmt und dann versucht, den Dienst mit dem Dienstnamen zu registrieren (Signal Register_req). Die Antwort auf diese Anfrage ist das Signal *LedPacket* und kann positiv oder negativ ausfallen (farbig markierte Entscheidung in Abbildung 4.6). Stimmen der Dienstname der Antwort mit dem Dienstnamen des Dienstes nicht überein, wird ein Laufzeitfehler (SDLError) ausgelöst, da die Antworten zu den jeweiligen Prozessen korrekt zugeordnet werden sollten. Bekommt der Prozess eine positive Bestätigung, so wurde der Dienst erfolgreich in der AmICoM registriert und ist nun bereit zur Nutzung. Liegt eine negative Antwort vor, so wird nach einer Wartezeit versucht, den Dienst erneut zu registrieren.

Im Zustand *ready* wartet der Dienst auf Anforderungen von Dienstnutzern. Wird eine solche Anforderung mittels des Recv Signals empfangen, liegen die Daten als kodierter Wert vom Typ *NtcPacket* vor, der zunächst dekodiert werden muss, dazu wird die Funktion *decode* aufgerufen. Schlägt der Dekodierungsvorgang fehl, so wird das Paket verworfen. Bei erfolgreicher Dekodierung kann nun auf die Daten des Pakets zugegriffen werden. Das Paket enthält einen Integer-Wert (siehe Definition des Datentyps Abschnitt 4.5), das den Zeitabstand zwischen zwei Temperaturwerten angibt. Ist der Wert 0, so wird nur ein einzelner Temperaturwert geliefert und ein eventuell gesetzter Timer für einen periodischen Temperaturwert gelöscht. Mit dem Signal BOARD_get wird ein Temperaturwert vom Sensorboard angefordert. Bei einer periodischen Bereitstellung von Temperaturwerten wird ein Timer gesetzt, der auslöst, wenn der nächste Temperaturwert geliefert werden soll. Erhält

der Prozess über das Signal Temperature einen Temperaturwert, wird dieser zunächst in eine Variable eines ASN.1 Datentyps geschrieben und anschließend enkodiert. Dann wird das Paket über das Send-Signal über das Netzwerk an alle Dienstnutzer gesendet.

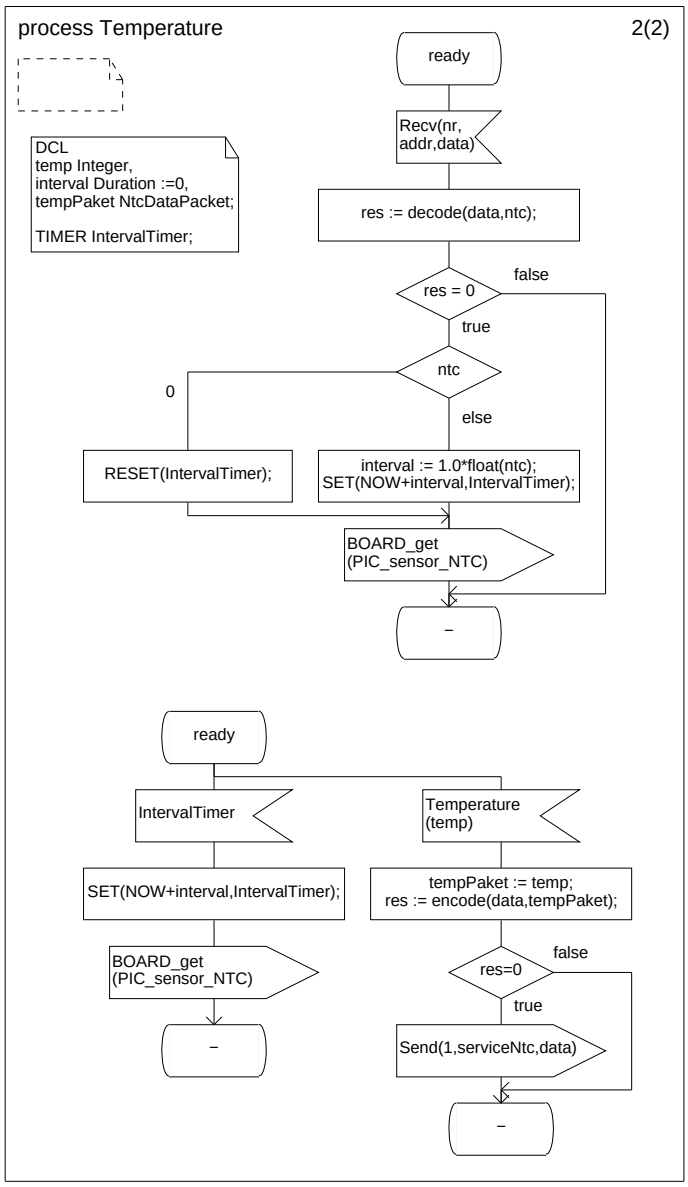


Abbildung 4.7: Prozess *Temperature* (Seite 2)

Dies ist ein Beispiel für einen einfachen Dienst für die AmICoM. Nicht berücksichtigt werden momentan mehrere Dienstnutzer, die möglicherweise unterschiedliche Zeitanforderungen an einen periodischen Temperaturdienst stellen. Während ein Knoten alle 10 Sekunden einen Temperaturwert benötigt, benötigt ein anderer nur jede Minute einen Wert. Der weitere Ausbau von Diensten hängt von den Anforderungen der Dienstnutzer ab und weitere Funktionalität wird bei Bedarf ergänzt. Die anderen am Anfang diese Abschnittes erwähnten Dienste funktionieren sehr ähnlich zu dem hier vorgestellten Temperaturdienst und werden nicht näher betrachtet.

Am Ende dieses Kapitels lässt sich zusammenfassen, dass nun sowohl für den PC als auch für den Imote2 eine Verison der AmICoM vorliegt. Es wurden eine Reihe von Beispiel-

diensten für den Imote2 entwickelt und aufgezeigt, wie sich die Datenschnittstellen von Diensten mit Hilfe von ASN.1 spezifizieren lassen. Bis jetzt können PC-Netzwerk und Imote2-Netzwerk nicht miteinander kommunizieren, da sie verschiedene Kommunikationstechnologien verwenden. Die Entwicklung eines Gateways, das zwischen den beiden Netzen vermittelt, ist Thema des nächsten Kapitels.

Kapitel 5

AmICoM-Gateway

Es wurden bis jetzt zwei Versionen der AmICoM vorgestellt. Die PC-Version nutzt als Kommunikationstechnologie Ethernet und WLAN, die Imote2-Version nutzt den 802.15.4 Standard. Damit ist zunächst die Nutzung von Diensten im selben Subnetz möglich. Zur netzübergreifenden Nutzung von Diensten ist zusätzlich eine Gatewayfunktionalität erforderlich. Ein Gateway ist laut [Tan07] ein Gerät, das zwei nicht miteinander kompatible Netze verbindet, indem es die für die Umsetzung benötigte Hardware und Software bereitstellt. Zwei über ein Gateway verbundene Netze nennt man *Internetwork*. Beispiele für Gateways sind ein E-Mail-Gateway, dass ankommende E-Mails als SMS in ein Handy-Netz versendet, oder ein Gateway, das ein Netzwerk, das TCP/IP nutzt, und ein Netzwerk, das das ATM-Transportprotokoll nutzt, verbindet.

Für den Zusammenschluss von AmICoM-Subnetzen wird solch ein Gateway benötigt. Es muss eine Umsetzung der Signale des 802.15.4-Netzwerkes in die des Ethernets/WLANs stattfinden. Um keine zusätzliche Hardware anschaffen und programmieren zu müssen, wurde eine Lösung gewählt, die mit schon vorhandenen Hardwarekomponenten auskommt. Das AmICoM-Gateway besteht aus einem PC und einem Imote2, die über eine UART-Schnittstelle miteinander verbunden sind und so die Verbindung zwischen den zwei Netzen herstellen können.

Die Verbindung von Netzwerken birgt ein potentielles Problem: Die Daten, die zwischen Dienstanbieter und Dienstnutzer ausgetauscht werden, werden mittels Broadcast im jeweiligen Netzwerk verteilt. Beide vom PC verwendeten Kommunikationstechnologien besitzen eine wesentlich höhere Übertragungsrate (Ethernet bis 1 GBit/s, WLAN bis 54 MBit/s), als der 802.15.4-Standard (bis zu 250 kBit/s), den der Imote2 verwendet. PCs können nun Dienste bereitstellen, die hohe Datenraten benötigen, wie zum Beispiel Dienste, die Videodaten über das AmICoM-Netzwerk versenden. Findet eine Datenübertragung statt, so werden aufgrund des Broadcasts die Daten auch innerhalb des Imote2-Netzwerkes verteilt. Innerhalb von kurzer Zeit wird dann das Imote2-Netzwerk überlastet, da der Transceiver des Imote2 mit den hohen Datenraten nicht zurecht kommt. Aber in den meisten Fällen ist es gar nicht nötig, bandbreitenintensive Dienste in das Imote2 Netzwerk zu leiten, da die Knoten aufgrund ihrer geringen Leistungsfähigkeit und Hardwareausstattung den Dienst nicht nutzen können. Eine bessere Lösung wäre, dass das Gateway nur die Daten in das Imote2-Netzwerk weiterleitet, die auch im Netzwerk benötigt werden. Diese Funktionalität stellt das in dieser Arbeit entwickelte AmICoM-Gateway bereit.

Das Kapitel ist in vier Abschnitte unterteilt. Der erste Abschnitt stellt den Aufbau des Gateways vor, der zweite betrachtet das Protokoll, das auf der Punkt-zu-Punkt UART-Verbindung zwischen PC und Imote2 verwendet wird. Im dritten Abschnitt geht es um die

dynamische Filtereigenschaft des Gateways, die nur die im anderen Netz benötigten Daten weiterleitet. Von dem entwickelten Filter wird mit Hilfe von ConTraST eine Dokumentation erstellt, das Ergebnis wird im letzten Abschnitt vorgestellt.

5.1 Architektur des Gateways

Abbildung 5.1 zeigt die Architektur des AmICoM-Gateways. Es besteht aus einem PC und aus einem Imote2, die über eine UART miteinander verbunden sind. Der PC kann je nach Hardwareausstattung mittels Ethernet oder WLAN mit anderen PCs im Netz verbunden sein, der Imote2 ist über den CC2420, der den 802.15.4 Standard implementiert, mit den anderen Imote2 vernetzt. Sowohl PC als auch Imote2 lassen sich zusätzlich zu ihrer Gatewayfunktionalität als normale AmICoM-Knoten verwenden, das bedeutet, sie können selbst sowohl Dienste anbieten als auch nutzen. Grundsätzlich werden alle Pakete, die PC oder Imote2 aus ihrem Netz empfangen, über die UART-Schnittstelle gesendet und im anderen Netz gesendet. Aus den oben genannten Gründen möchte man jedoch nicht, dass jedes Paket in jedes Netzwerk weitergegeben wird, sondern nur die benötigten. Der dynamische Filter, der dies bewerkstelligt, befindet sich auf dem (wesentlich leistungsfähigeren) PC. Dies bedeutet, dass der Imote2-Teil des Gateways keine Pakete filtert. Jedes Paket im Imote2-Netzwerk wird zunächst über die UART-Verbindung zum PC gesendet. Dort wird es dann vom Filter eventuell verworfen oder weitergeleitet. Dieses Vorgehen erzeugt einen höheren Bandbreitenbedarf der UART-Schnittstelle. Der Grund den Filtermechanismus nur einseitig zu verwenden ist, dass man Dienste, die nur im PC-Netzwerk verwendet werden und einen hohem Bandbreitenbedarf besitzen, daran hindert, dass ihre Daten ins Imote2-Netz gelangen und nicht umgekehrt, da das PC-Netzwerk eine wesentlich größere Bandbreite besitzt. Zum anderen werden Dienste, die im Imote2-Netzwerk angeboten werden, eher von PC genutzt, als dass ein Imote2 einen PC-Dienst nutzt. Daher müssen die Daten in der Regel sowieso über die UART-Schnittstelle übertragen werden. Schließlich verfügt der PC über signifikant mehr Leistung als der Imote2 und wird durch den dynamischen Filter kaum zusätzlich belastet. Im Folgenden werden der PC-Teil und der Imote2-Teil des Gateways genauer betrachtet.

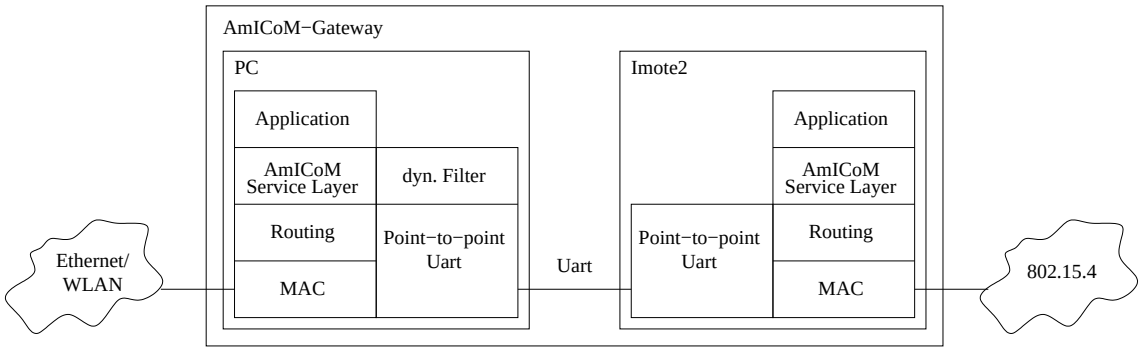


Abbildung 5.1: Schematische Übersicht des AmICoM-Gateways

Ein Unterschied zu der gewöhnlichen Arbeitsweise eines Gateways ist, dass ein Gateway normalerweise explizit adressiert wird, dies hier aber nicht der Fall ist. Da jedes Paket im AmICoM-Netzwerk an alle anderen Knoten geschickt wird, kommt auch jedes Paket am Gateway an. Das Gateway entscheidet dann selbstständig, ob es das Paket in das andere Netz leitet oder nicht.

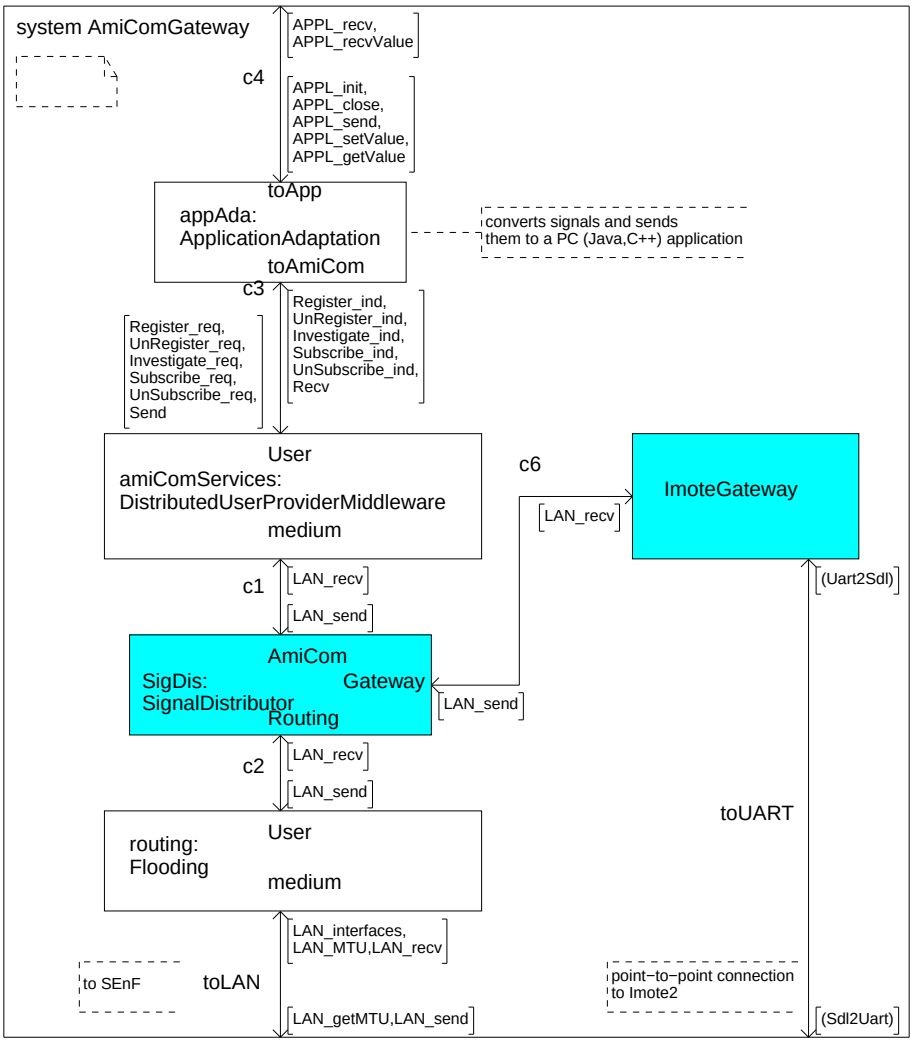


Abbildung 5.2: SDL-System des Gateways: PC-Teil

5.1.1 PC-Teil des AmICoM-Gateways

Das SDL-System für den PC-Teil des Gateways (Abbildung 5.2) ist ähnlich dem SDL-System für normale AmICoM-Knoten für den PC (Abschnitt 4.4) aufgebaut. Die Blöcke *ApplicationAdaptation*, *DistributedUserProviderMiddleware* und *Flooding* bleiben unverändert. Neu hinzugekommen sind die Blöcke *ImoteGateway* und *SignalDistributor* (in Abbildung 5.2 farbig markiert). Der Block *ImoteGateway* enthält den dynamischen Filter und das UART-Protokoll, das auf der Verbindung zwischen PC und Imote2 verwendet wird. Eine Anforderung ist, dass auch die Gatewayknoten als normale AmICoM-Knoten genutzt werden können. Deswegen enthält der PC-Teil des Gateways eine Instanz des Blockes *DistributedUserProviderMiddleware*. Nun müssen Pakete, die über die LAN-Schnittstelle den Knoten erreichen, nicht nur an die *DistributedUserProviderMiddleware* weitergegeben werden, sondern auch an den Block *ImoteGateway*, da in der AmICoM nicht bekannt ist, welcher Dienst auf welchem Knoten angeboten wird. Umgekehrt müssen Pakete vom Block *DistributedUserProviderMiddleware* sowohl im PC-Netzwerk als auch potentiell im Imote2-Netzwerk verteilt werden, gehen also sowohl an den Block *Flooding* als auch an den Block *ImoteGateway*. Dasselbe gilt für Pakete, die über die UART-Schnittstelle aus dem Imote2-Netzwerk empfangen werden. Passieren sie den Filter, werden sie sowohl an den lokalen

AmICoM-Knoten als auch an das PC-Netzwerk weitergegeben. Die Vermittlung zwischen diesen drei Blöcken leistet der Block SignalDistributor. Er besitzt drei Gates und leitet Signale, welche er über ein Gate empfängt an die beiden anderen Gates weiter. Es findet also innerhalb dieses Blockes eine Signalduplizierung statt.

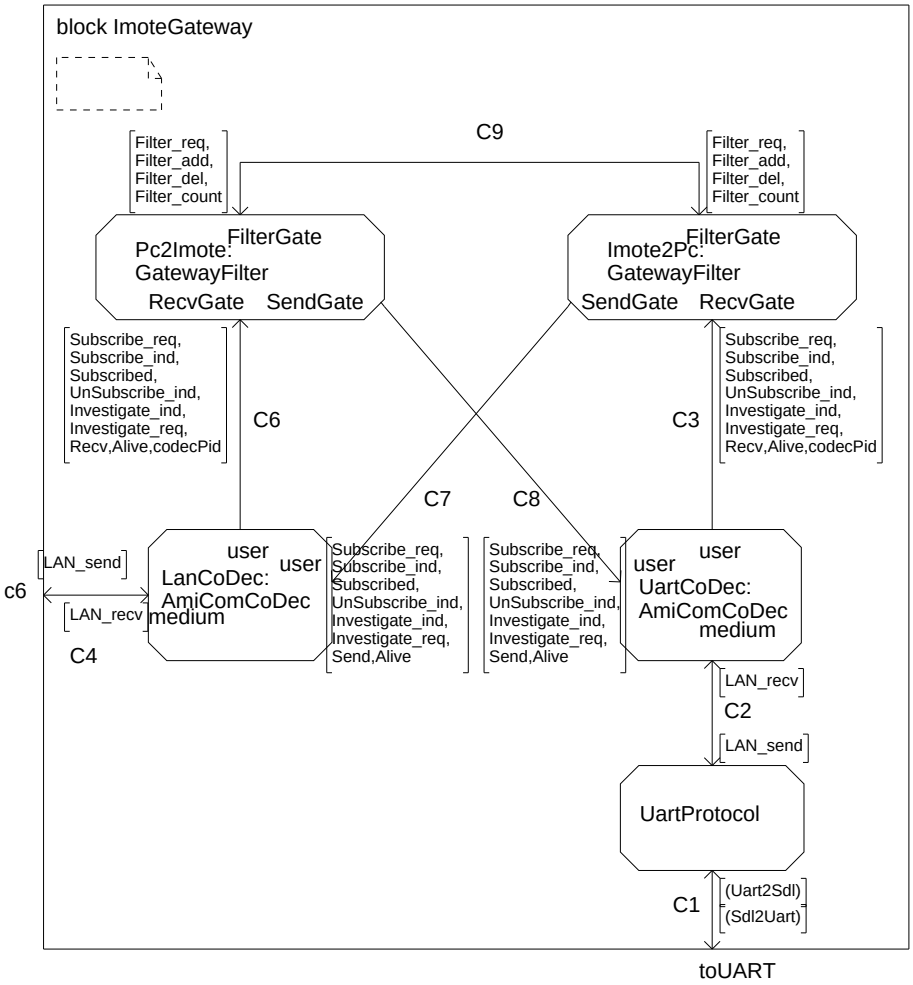


Abbildung 5.3: Block *ImoteGateway*

In Abbildung 5.3 sieht man die interne Architektur des Blocks *ImoteGateway*. Er enthält das UART-Protokoll (Block *UartProtocol*), das im nächsten Abschnitt vorgestellt wird, und den dynamischen Filter. Der Filter ist symmetrisch aufgebaut und besteht aus zwei Instanzen des Prozesstyps *GatewayFilter*. Die Funktion des dynamischen Filters wird detailliert in Abschnitt 5.3 erläutert. Zusätzlich enthält der Block noch zwei Instanzen des Prozesstyps *AmiComCoDec*. Damit eine Entscheidung im Filter über die Weiterleitung des Paketes getroffen werden kann, muss der Typ des Pakets bekannt sein. Dazu muss das Paket, das als Bytefolge mit dem Signal LAN_recv empfangen wird, in ein AmICoM-Signal umgewandelt werden. Entscheidet der Filter, dass das Paket in das andere Netzwerk weitergeleitet wird, muss das Paket wieder in eine Bytefolge encodiert werden. Diese Aufgaben übernehmen die CoDec-Prozesse. Eine Möglichkeit, um den erneuten Dekodierungsvorgang des Signals einzusparen, ist die Zwischenspeicherung der Originalbytefolge. Diese muss dann an den *GatewayFilter* weitergegeben werden; wenn der Filter das Paket durch das Gateway leitet, kann sofort die originale Bytefolge versendet werden. Dies würde eine Änderung der CoDec-Prozesse erfordern, so dass diese neben dem dekodierten Signal auch die Original-

bytefolge zum Filterblock senden. Somit könnte der CoDec-Prozess aus der AmICoM nicht wiederverwendet werden, weshalb diese Lösung nicht weiter verfolgt wird. Der zusätzliche Enkodierungsaufwand ist aufgrund der Leistungsfähigkeit des PC zu vernachlässigen.

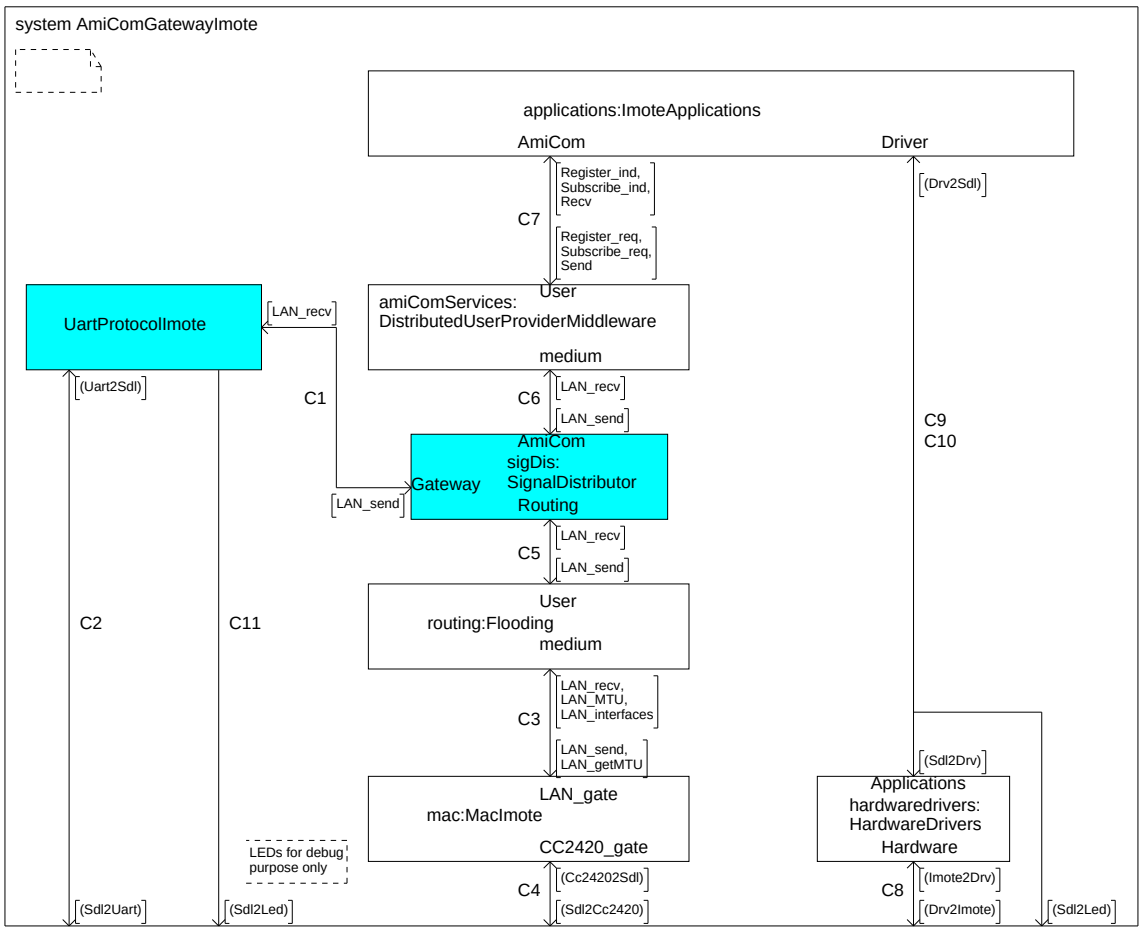


Abbildung 5.4: SDL-System des Gateways: Imote2-Teil

5.1.2 Imote2-Teil des Gateways

Auch die Imote2-Seite des Gateways (Abbildung 5.4) entspricht im wesentlichen dem normalen AmICoM-Knoten für den Imote2 (siehe Abschnitt 4.4). Alle Blöcke daraus wurden unverändert übernommen, damit kann auch der Imote2-Teil des Gateways als normaler AmICoM-Knoten genutzt werden. Um die Gatewayfunktinalität zu realisieren sind auch hier Blöcke hinzugefügt worden (farbig markiert in Abbildung 5.4). Der Block *SignalDistributor* erfüllt dieselbe Aufgabe wie auf der PC-Seite des Gateways. Er leitet ein Paket, das über einen der Blöcke *UartProtocolImote*, *DistributedUserProviderMiddleware* oder *Flooding* empfangen wurde, an die jeweils zwei anderen Blöcke weiter, um die Pakete im gesamten AmICoM-Netzwerk zu verteilen. Der Block *UartProtocolImote* enthält das UART-Protokoll auf Imote2-Seite, das mit dem PC kommuniziert und die beiden Teilnetze miteinander verbindet. Eine Filterfunktionaltät gibt es im Imote2-Teil des Gateways nicht. Die Hardwareanbindung der UART-Schnittstelle erfolgt über SEnF. Zusätzlich werden die LEDs des Imote2 nicht für einen Dienst verwendet, sondern dienen als Statusausgaben. Wenn keine Verbindung mit der PC-Seite des Gateways besteht, blinkt die rote LED des Imote2. Sobald eine Verbindung aufgebaut wurde erlischt die rote LED und die blaue

und die grüne LED leuchten dauerhaft. Beim Empfang eines Paketes über die UART-Schnittstelle erlischt die blaue LED für kurze Zeit und wird dann wieder eingeschaltet. Beim Versenden eines Pakete wird die grüne LED kurz aus- und wieder eingeschaltet. Auf diese Weise ist für den Systementwickler leicht erkennbar, wann Daten durch das Gateway fließen.

5.2 Punkt-zu-Punkt-Verbindung zwischen Imote2 und PC

Eine UART-Schnittstelle verbindet die Imote2-Seite des Gateways mit der PC-Seite des Gateways. Die Verbindung arbeitet im Voll-Duplexmodus und verfügt über eine maximale Datenrate von 115200 Bit/s in beide Richtungen. Ein UART-Zeichen besteht aus 8 Datenbits, einem Start- und einem Stoppbit. Somit ist eine Datenrate je Richtung von

$$\frac{8}{10} \cdot 115200\,\mathrm{Bit/s} = 92160\,\mathrm{Bit/s}$$

möglich. Das bedeutet, dass die UART-Verbindung der Flaschenhals des AmICoM-Netzwerks ist, da es die langsamste Verbindung ist. Die Anbindung der UART-Schnittstelle erfolgt über das SEnF mit den UART-Modulen für PC und Imote2. Auf dieser Punkt-zu-Punkt-Verbindung wird ein kleines Protokoll verwendet, das folgende Aufgaben hat:

- Beide Endpunkte müssen wissen, ob eine Verbindung erfolgreich aufgebaut wurde.
- Die Grenzen der Rahmen der AmICoM müssen erkannt werden.
- Die Daten dürfen nicht zu schnell zur UART-Schnittstelle gesendet werden, damit diese nicht überlastet wird.

Abbildung 5.5: Ein UART-Rahmen

Der PC-Teil des Protokolls, das die genannten Aufgaben erfüllt, befindet sich im Block *UartProtocol* (siehe Abb 5.3) und der Imote2-Teil des Protokolls im Block *UartProtocolImote* (siehe 5.4). Um Rahmengrenzen sicher erkennen zu können, werden die AmICoM-Rahmen in ein weiteres Rahmenformat eingepackt. Ein UART-Rahmen besteht aus einem Längenbyte am Anfang des Rahmens und einem Nullbyte am Ende des Rahmens (siehe Abbildung 5.5). Das Längenbyte enthält die Länge der Daten, wobei das Längenbyte und das Nullbyte am Ende nicht mitgezählt werden. Mit Hilfe dieser zusätzlichen Bytes lassen sich Rahmengrenzen erkennen, indem man überprüft, ob nach den Daten, deren Länge bekannt ist, an der richtigen Stelle ein Nullbyte steht. Zum Versenden eines AmICoM-Rahmens werden die zusätzlichen Bytes hinzugefügt. Beim Empfang eines Rahmens werden die ursprünglichen AmICoM-Rahmen wieder aus den Daten extrahiert und können weiterverarbeitet werden. Schlägt die Rahmenerkennung fehl, so wird die gesamte empfangene Bytefolge verworfen und erst mit der nächsten empfangenen Bytefolge wiederaufgenommen. Die Fehlerrate auf der UART-Verbindung ist gering, kommt es trotzdem zu einem Übertragungsfehler, so kann der Rahmen vom CoDec-Prozessen der AmICoM nicht korrekt

dekodiert werden und wird dort verworfen, deswegen ist eine Fehlerbehandlung im UART-Protokoll nicht unbedingt notwendig. Das Verfahren funktioniert nur unter der Annahme, das ein Signal, das empfangene Daten der UART-Schnittstelle enthält, zwar mehrere Rahmen enthalten kann, die Rahmen sich aber nicht auf mehrere Signale verteilen. Von dieser Annahme kann ausgegangen werden, da das UART-Modul des SEnF immer dann empfangene Daten an das SDL-System weitergibt, wenn eine bestimmte Zeit lang kein Byte mehr über die UART-Schnittstelle empfangen wurden. Somit enthalten Signale, die im SDL-System ankommen zwar unter Umständen mehrere Rahmen, aber diese Rahmen sind mit großer Wahrscheinlichkeit komplett.

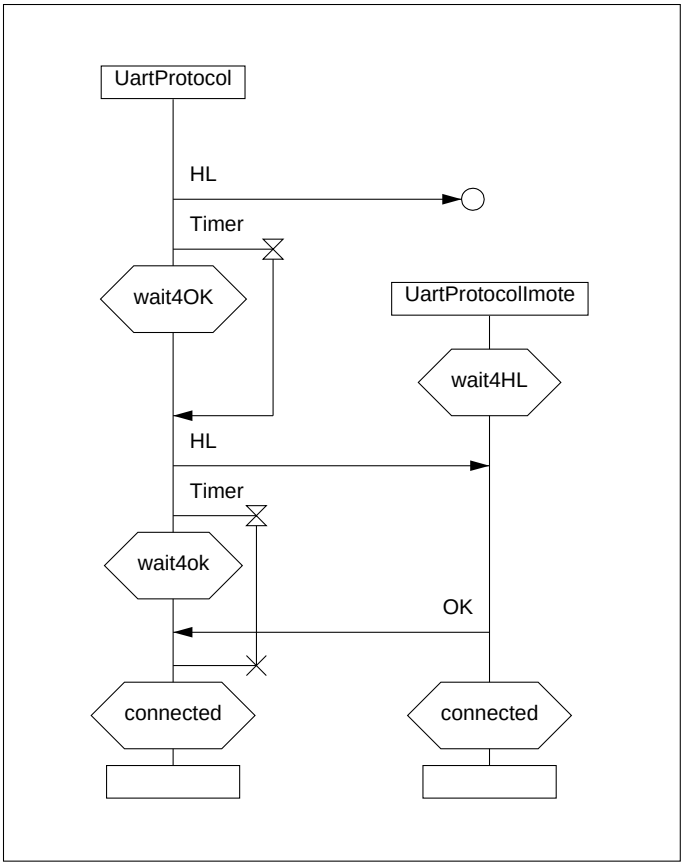


Abbildung 5.6: Handshake bei Verbindungsaufbau der UART-Schnittstelle

Ein Handshake-Protokoll wird verwendet, um zu erkennen, dass beide Seiten bereit zum Datenaustausch sind. Der Handshake geschieht durch den Austausch von 2-Byte langen Nachrichten, die keine Längen- oder Nullbytes enthalten. Abbildung 5.6 zeigt den Ablauf eines Handshakes. Der PC sendet eine Hallo-Nachricht (kodiert als HL), worauf der Imote2 mit einer OK-Nachricht antwortet. Bekommt der PC auf eine Hallo-Nachricht keine Antwort, so wiederholt er nach Ablauf eines Timers seine Anfrage. Wird sowohl die Hallo-Nachricht als auch die OK-Nachricht empfangen, gilt die Verbindung als aufgebaut und es können Daten ausgetauscht werden. Der Imote2 signalisiert eine aufgebaute Verbindung und Datenaustausch mit seinen LEDs. Fällt der PC in der Verbindung aus, so erkennt der Imote2 diesen Ausfall nicht, aber die Verbindung wird, wenn der PC wieder bereit ist, neu aufgebaut, indem der PC wiederum eine Hallo-Nachricht sendet und der Imote2 mit einer OK-Nachricht antwortet. Fällt der Imote2 aus, so wird dies ebenfalls nicht festgestellt und der PC erhält die Verbindung aufrecht. Damit sich Imote2 und PC in diesem Fall neu

synchronisieren können, sendet der Imote2, wenn er neu gestartet wird, ebenfalls Hallo-Nachrichten aus. Ist keine Verbindung aufgebaut, ignoriert der PC diese Nachrichten und unternimmt stattdessen einen eigenen Versuch, die Verbindung aufzubauen. Nimmt der PC jedoch an es besteht noch eine Verbindung zum Imote2, wird ihm durch diese Nachricht signalisiert, die Verbindung neu aufzubauen. Es wurde sich gegen eine Erkennung von Ausfällen entschieden, da diese aufwändiger zu realisieren sind. So hätte beispielsweise eine Erkennung mittels regelmäßig ausgetauschten Nachrichten für zusätzliche Belastung der Bandbreite gesorgt.

Damit die Sendepuffer der UART-Schnittstellen nicht überlaufen, wenn sie Daten in zu schneller Folge erhalten, gibt es auf beiden Seiten der Verbindung Warteschlangen. Auf PC-Seite ist die Warteschlange im SEnF-Modul der UART-Schnittstelle lokalisiert, auf Imote2-Seite im SDL. Bevor neue Daten über die UART-Schnittstelle des Imote2 gesendet werden, wird auf die Bestätigung vom SEnF-Modul gewartet, dass der vorangehende Rahmen fertig gesendet wurde. Die Benachrichtigung erfolgt über das Signal UART_sendFinished. In der Zwischenzeit werden alle weiteren Rahmen, die über die UART-Schnittstelle gesendet werden sollen, im SDL-System zwischengespeichert.

5.3 Die dynamische Filterfunktionalität des AmICoM-Gateways

In diesem Abschnitt wird die dynamische Filterfunktionalität des Gateways (siehe Abbildung 5.3) näher erläutert; für diese Funktion sind wie schon erwähnt die beiden Instanzen des Prozesses *GatewayFilter* zuständig, die sich beide auf der PC-Seite des Gateways befinden. Die Filterfunktionalität ist symmetrisch aufgebaut, das bedeutet die Funktionalität ist von beiden Netzen aus betrachtet gleich. Jede dieser Instanzen ist für eine Kommunikationsrichtung zuständig, dem entsprechend sind die Namen der Prozesse *Pc2Imote* und *Imote2Pc*. Für die Filterung unterscheidet man zwei Paketarten. Zum einen die Management-Pakete des AmICoM-Netzwerkes, die den Filter in jedem Fall passieren müssen, weil es sich hierbei um Daten handelt, die im gesamten AmICoM-Netzwerk verfügbar sein müssen. Zum anderen sind es die Pakete, die die eigentlichen Daten, die zwischen den Diensten ausgetauscht werden, beinhalten. Diese Pakete sollen nur dann in das andere Netz weitergeleitet werden, wenn sie dort gebraucht werden. In dem jeweils anderen Netz werden Datenpakete dann benötigt, wenn Dienstanbieter und Dienstnutzer sich in verschiedenen Teilnetzen befinden. Befinden sich Dienstanbieter und Dienstnutzer im selben Teilnetz, so genügt es, wenn die Datenpakete nur in diesem Teilnetz verfügbar sind. Auf diese Weise ist es möglich, die Daten der Dienste, die eine hohe Übertragungsbandbreite benötigen und nur im PC-Netzwerk genutzt werden, im Gateway zu filtern und so das Imote2-Netzwerk nicht zu überlasten.

Die grundlegende Idee ist, dass jeder Filterprozess eine Liste mit Diensten besitzt, deren Pakete den Filter passieren dürfen; alle anderen Datenpakete werden im Filter verworfen. Diese Liste wird als Durchgangsliste bezeichnet. Es muss erkannt werden, wann ein Dienstnutzer und ein Dienstanbieter sich in verschiedenen Teilnetzen befinden, denn dann muss der Dienst der Durchgangsliste hinzugefügt werden. Wenn Dienste ausfallen oder ein Dienstnutzer sein Abonnement eines Dienstes zurücknimmt, dann müssen diese Dienste wieder von der Liste entfernt werden, damit der Datenverkehr so gering wie möglich gehalten wird. Die beiden Filterinstanzen müssen ihre Listen konsistent halten, dazu kommunizieren sie mittels Signalen miteinander. Im Folgenden erläutern einige MSCs die Funktionsweise der Filterfunktionalität. Die MSCs abstrahieren von der in 5.1.1 und in 5.1.2

vorgestellten Struktur. Die Instanzen *Pc2Imote* und *Imote2Pc* in den MSCs entsprechen den jeweiligen Prozessen im SDL-System. Die Instanzen *LAN* und *UART* stehen stellvertretend für die zwei verschiedenen Netze. Die LAN-Instanz steht stellvertretend für einen Knoten im PC-Netzwerk der AmICoM; hierbei ist nicht ausgeschlossen, dass es sich um den PC-Gateway-Knoten selbst handelt. Jedes Paket, das dieser Instanz geschickt wird, wird im PC-Netzwerk entweder über den Block *SignalDistributor* oder über den Fluten-Algorithmus verteilt. Dasselbe gilt für die Instanz *Uart* und das Imote2-Netzwerk, mit dem Unterschied, dass zwischen Filter und Imote2-Netzwerk die Pakete vorher über UART-Verbindung zum Imote2 geschickt werden und somit jedes Paket aus dem Imote2-Netzwerk zunächst an den PC übermittelt wird.

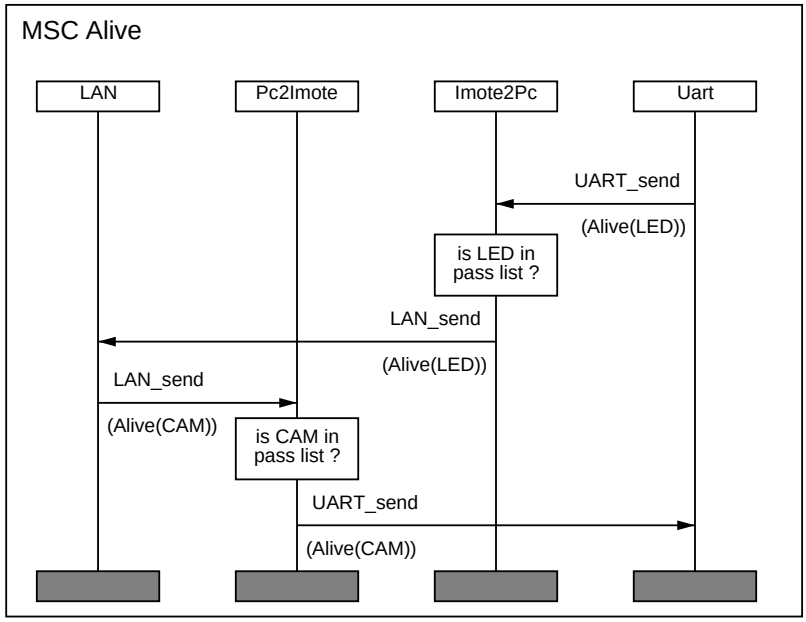


Abbildung 5.7: MSC: Weiterleitung von Alive-Nachrichten auf dem PC-Knoten

5.3.1 Verfügbare Dienste im Netzwerk

Die Nachricht Alive wird von jedem Dienstanbieter periodisch an jeden Knoten im Netzwerk versendet. Sie wird dazu verwendet, um jedem Knoten die im Netzwerk verfügbaren Dienste mitzuteilen und um den Dienstnutzern zu signalisieren, dass ihr abonnierter Dienst noch verfügbar ist. Da alle Dienste im AmICoM-Netzwerk jedem Knoten bekannt sind und Dienstnutzer darüber informiert werden, ob der Dienst noch verfügbar ist, ergibt sich für den Filter des Gateways, dass Pakete, die eine Alive-Nachricht enthalten, in jedem Fall in das andere Netz weitergeleitet werden müssen. Das MSC in Abbildung 5.7 zeigt zwei Alive-Nachrichten von zwei unterschiedlichen Diensten aus verschiedenen Netzen. Beide Nachrichten werden von den jeweiligen Filter-Instanzen nicht gefiltert und an das jeweils andere Netz weitergegeben, wo sie an alle weiteren Knoten verteilt werden.

Beim Empfang einer Alive-Nachricht wird überprüft, ob der Dienst in der Durchgangsliste steht. Ist dies der Fall, werden die Alive-Nachrichten des Dienstes beobachtet, damit der Dienst, wenn er ausfällt, wieder aus der Durchgangsliste gelöscht werden kann. Näheres hierzu siehe Abschnitt 5.3.3.

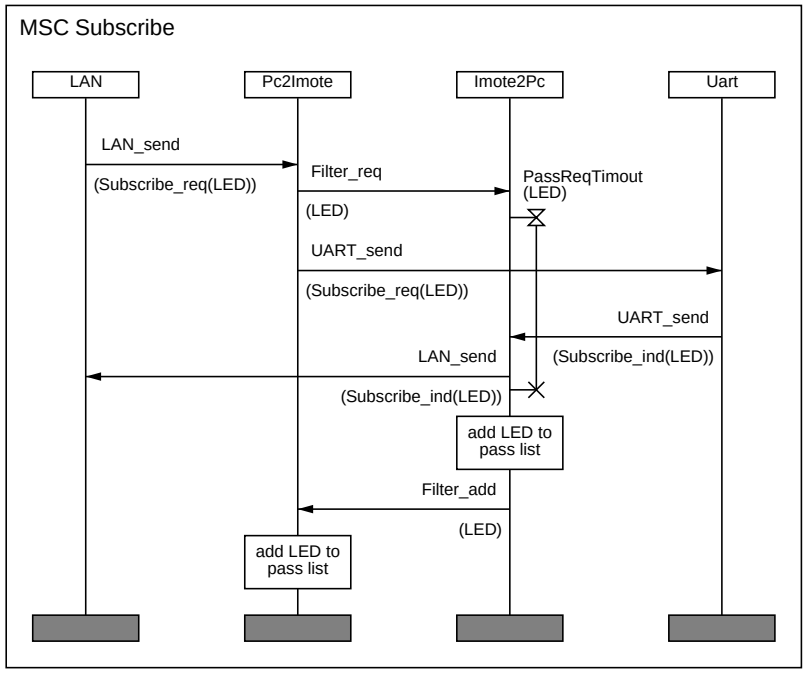


Abbildung 5.8: MSC: Hinzufügen von Diensten zur Durchgangsliste

5.3.2 Hinzufügen von Diensten in die Durchgangsliste

Bei Systemstart sind die Durchgangslisten zunächst leer und werden nach und nach aufgebaut, sobald Dienste abonniert werden. Es muss festgestellt werden, wann Dienstanbieter und Dienstnutzer sich in verschiedenen Netzen befinden, dann muss der Dienst in die Durchgangsliste von beiden Filter-Instanzen aufgenommen werden. Diese Erkennung geschieht mit den Subscribe-Nachrichten, die vom Dienstnutzer dazu benutzt werden, um einen Dienst zu abonnieren.

Den genauen Vorgang verdeutlicht das MSC in Abbildung 5.8. In diesem Szenario möchte ein Knoten aus dem PC-Netzwerk den LED-Dienst abonnieren, den ein Knoten im Imote2-Netzwerk anbietet. Der Knoten sendet dazu eine Subscribe_req-Nachricht. Diese Nachricht erreicht per Broadcast auch den Gatewayknoten und somit auch den Prozess *Pc2Imote*. Die Subscribe_req-Nachricht wird in jedem Fall in das Imote2-Netz weitergeleitet, da nicht bekannt ist, in welchem Netzteil der LED-Dienst angeboten wird. Gleichzeitig wird die andere Filter-Instanz (*Imote2Pc*) mit der Nachricht Filter_req darüber informiert, dass es im PC-Netzwerk eine Anfrage für den LED-Dienst gab. Jede Filter-Instanz hat neben der Durchgangsliste noch eine Liste, in der sie alle Dienste, für die sie eine Subscribe_ind-Nachricht erwartet, speichert. Der Prozess *Imote2Pc* wartet nun auf eine Subscribe_ind-Nachricht aus dem Imote2-Netzwerk, die anzeigt, dass der Dienst erfolgreich abonniert wurde. Wird die Nachricht vor Ablauf eines Timers empfangen, so ist bekannt, dass sich der Dienst im Imote2-Netzwerk befindet und er wird von der Liste der erwarteten Subscribe_ind-Nachrichten gelöscht, der Durchgangsliste hinzugefügt und die Nachricht wird in das PC-Netz weitergeleitet. Der Dienst wird mittels der Nachricht Filter_add noch der Durchgangsliste des Prozesses *Pc2Imote* hinzugefügt, damit Datenpakete dieses Dienstes den Filter in beiden Richtungen passieren können. Nun enthalten beide Durchgangslisten den Eintrag LED.

Angenommen der Dienst LED befindet sich nicht im Imote2- sondern im PC-Netzwerk, dann empfängt nicht der Prozess *Imote2Pc* die Nachricht Subscribe_ind, sondern der Pro-

zess *Pc2Imote*. Dieser Prozess wartet nicht auf eine Subscribe_ind-Nachricht, deswegen wird auch kein Eintrag in der Durchgangsliste erzeugt. Der im Prozess *Imote2Pc* aufgezogene Timer läuft ab und der Dienstname wird aus der Liste der erwarteten Subscribe_ind-Nachrichten gelöscht. Somit steht fest, dass sich Dienstanbieter und Dienstnutzer im selben Netzwerk befinden und deren Datenverkehr nicht durch das Gateway geleitet wird.

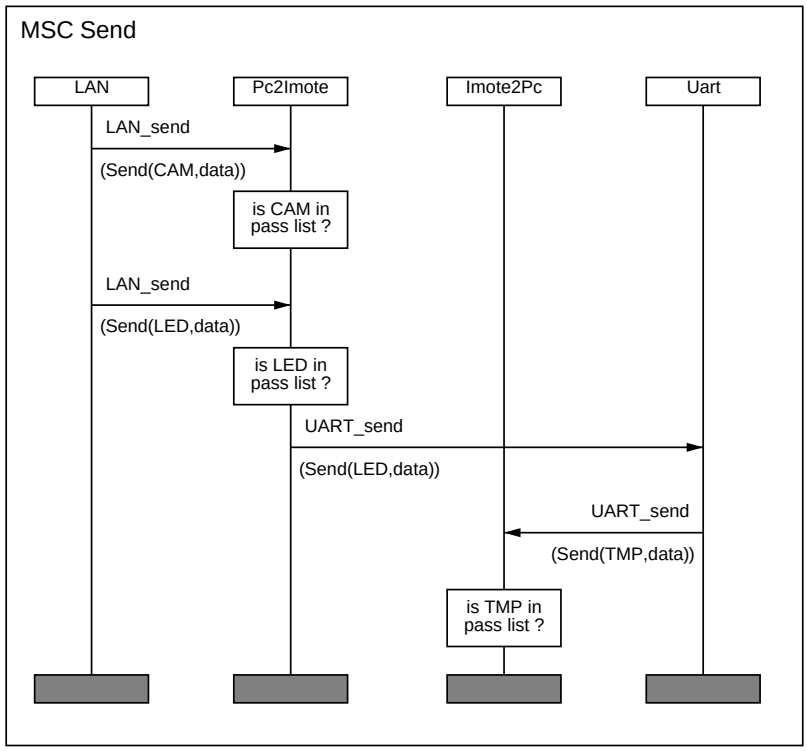


Abbildung 5.9: MSC: Datenaustausch zwischen Diensten

5.3.3 Datenaustausch zwischen Diensten

Die Filterung von Daten, die nicht im anderen Netzwerk benötigt werden, ist die Hauptaufgabe des Gateways. Wie schon erwähnt befinden sich die Dienste, bei denen sich Dienstnutzer und Dienstanbieter in verschiedenen Netzen befinden, in der Durchgangsliste. Die Entscheidung, ob ein Datenpaket durch das Gateway geleitet werden muss, ist ein Nachschlagen des Dienstnamens in der Durchgangsliste. Befindet sich der Dienstname in der Durchgangsliste, so wird das Paket weitergeleitet, ansonsten nicht. Anzahl und Größe der Managementpakete sind bei allen Diensten im AmICoM-Netzwerk in etwa gleich. Die Größe der Managementpakete ist vom Namen des Dienstes abhängig und die Anzahl von der Anzahl an Dienstnutzern. Gibt es mehrere Dienstnutzer, steigt auch die Anzahl der Pakete. Dahingegen unterscheiden sich Anzahl und Größe der Datenpakete je nach Dienst erheblich. So sind die Pakete eines Videodienstes, der Videodaten im Netzwerk verteilt, erheblich größer, als die eines Dienstes, der nur einzelne Temperaturwerte im Netzwerk verteilt. Deswegen werden auch die Datenpakete gefiltert, da sie bei bandbreitenintensiven Diensten die meiste Bandbreite benötigen.

Die Send-Nachricht enthält Datenpakete, die Dienstanbieter und Dienstnutzer untereinander austauschen. Ob diese Nachrichten durch das Gateway geleitet werden, hängt davon ab, ob sich der Dienst in der Durchgangsliste befindet. Abbildung 5.9 zeigt ein MSC, in dem mehrere Send-Nachrichten von verschiedenen Seiten am Gateway eintreffen. Nehmen

wir an, von den drei Diensten LED, CAM und TMP wurde nur der Dienst LED der Durchgangsliste hinzugefügt. Für jede Nachricht wird die Durchgangsliste abgefragt und aufgrund dieser Abfrage die Entscheidung getroffen. Dementsprechend wird nur die Send-Nachricht des LED-Dienstes durch das Gateway geleitet, die anderen beiden Send-Nachrichten des CAM- und des TMP-Dienstes werden im Filter verworfen.

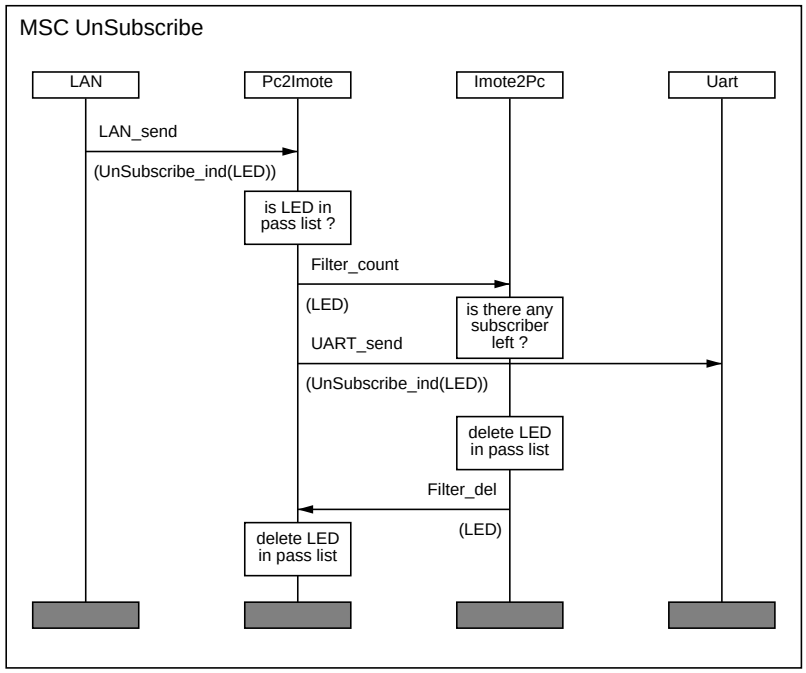


Abbildung 5.10: MSC: Löschen von Diensten aus der Durchgangsliste mittels UnSubscribe-Nachrichten

5.3.4 Löschen von Diensten aus der Durchgangsliste

Um unnötigen Datenverkehr zu vermeiden, müssen die Einträge der Durchgangsliste regelmäßig überprüft werden, ob sie noch gültig sind. Einträge der Durchgangsliste müssen in drei Fällen wieder gelöscht werden. Erstens, wenn der Dienstnutzer mittels der Nachricht UnSubscribe_ind sein Abonnement abmeldet und keine weiteren Dienstnutzer vorhanden sind, zweitens, wenn alle Dienstnutzer ausgefallen sind oder drittens, wenn der Dienstanbieter ausfällt.

Das MSC in Abbildung 5.10 beschreibt den ersten Fall. Ein Knoten im PC-Netzwerk hat den Dienst LED abonniert, der im Imote2-Netzwerk angeboten wird und nimmt nun sein Abonnement zurück. Um den Dienstanbieter darüber zu informieren, wird im PC-Netzwerk die Nachricht UnSubscribe_ind verbreitet und kommt schließlich auch am Gateway an. Der Prozess *Pc2Imote* stellt fest, dass sich der Dienst LED in seiner Durchgangsliste befindet, informiert den Prozess *Imote2Pc*, dass es einen Abonnenten dieses Dienstes weniger gibt und leitet die Nachricht in das Imote2-Netzwerk weiter. Der Prozess *Imote2Pc* hat die Anzahl der Dienstnutzer des Dienstes in seinem Imote2-Netzwerk mitgezählt und kann somit überprüfen, ob es noch einen Dienstnutzer im Imote2-Netzwerk gibt. Gibt es keinen Dienstnutzer mehr, so wird der Eintrag aus der Liste gelöscht. Dann muss noch der andere Prozess benachrichtigt werden, dass dieser den Dienst ebenfalls aus seiner Durchgangsliste löscht.

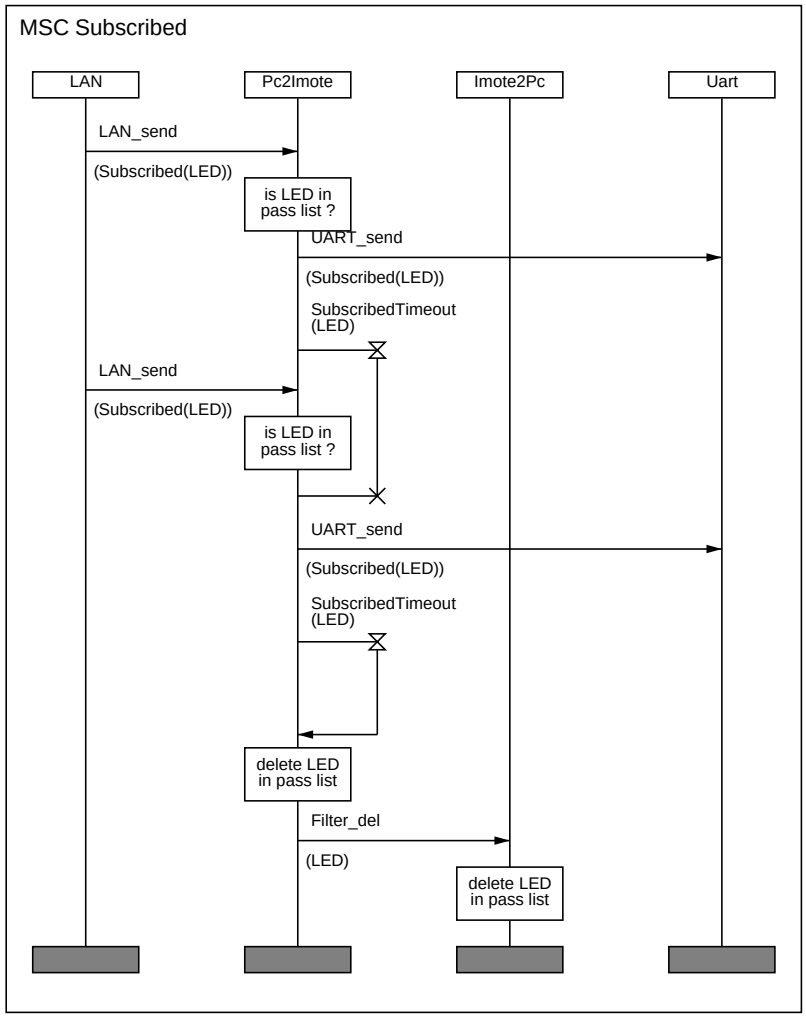


Abbildung 5.11: MSC: Löschen von Diensten aus der Durchgangsliste wegen Ausfall der Dienstnutzer

Es gibt auch Fälle, in denen sich der Dienstnutzer nicht ordnungsgemäß abmeldet, sondern einfach ausfällt. Dann kann der Eintrag aus der Durchgangsliste gelöscht werden, indem Subscribed-Nachrichten beobachtet werden, die jeder Dienstnutzer periodisch versendet. Dieses Szenario zeigt das MSC in Abbildung 5.11. Subscribed-Nachrichten werden ebenfalls nur dann durch das Gateway geleitet, wenn der Dienst in der Durchgangsliste steht. In der Abbildung wurde der LED-Dienst von einem Knoten im PC-Netzwerk abonniert und sendet in regelmäßigen Abständen Subscribed-Nachrichten. Jedes Mal wenn eine Subscribed-Nachricht vom Prozess *Pc2Imote* empfangen wird, wird ein parametrisierter Timer für diesen Dienst gestartet. Wird vor Ablauf des Timers eine neue Subscribed-Nachricht empfangen, so wird der Timer neu gestartet. Die Dauer des Timers ist so eingestellt, dass er erst auslöst, wenn drei Subscribed-Nachrichten ausgeblieben sind. Fallen alle Dienstnutzer aus, so kommen keine Subscribed-Nachrichten mehr zum Gateway und der Timer läuft ab, dann wird der Dienst aus beiden Durchgangslisten gelöscht.

Genau nach demselben Prinzip funktioniert das Löschen von Diensten aus den Durchgangslisten, wenn der Dienstanbieter ausfällt. Immer wenn eine Alive-Nachricht empfangen wird, wird ein Timer aufgezogen. Fällt der Dienst aus, läuft der Timer ab und der Dienst wird aus den Durchgangslisten gelöscht.

5.4 Dokumentation des Filterprozesses

Eine weitere Funktion des in dieser Arbeit zur Kode-Generation verwendeten Transpilers ConTraST ist, dass dieser aus einer mit Kommentaren angereichertem SDL-Komponente eine vollständige Dokumentation erstellen kann [FG07]. Diese Dokumentation enthält eine Reihe von Szenarien, die mit MSCs und ergänzendem Text beschrieben werden. Dem Leser der Dokumentation soll eine kurze Übericht über die Funktionsweise der Komponente gegeben werden. Mit Hilfe dieser Funktion wurde eine Dokumentation des Prozesses *GatewayFilter* erstellt. Die genauen Vorschriften, welche Kommentare man in eine SDL-Komponente einfügen muss, um eine Dokumetation zu erstellen, findet man in [Fli07]. Die generierte Dokumentation wurde an manchen Stellen leicht verändert, da der Dokumentationsalgorithmus noch nicht in jedem Fall fehlerfrei arbeitet. An den MSCs der Dokumentation wurden vor allem die Richtungen der Signale verändert, um die MSCs übersichtlicher zu gestalten, und die Reihenfolge der generierten Szenarien wurde verändert, um die Dokumentation verständlicher zu machen. Im Anhang B findet man die komplette Dokumentation des *GatewayFilters*.

Kapitel 6

Der Audiodienst des Imote2

Eine wichtige Funktion des Sensorboards ist die Aufnahme und Wiedergabe von Audiodaten. Es soll möglich sein, Audiodaten auf einem Knoten aufzunehmen, über die CC2420-Schnittstelle des Imote2 zu versenden und auf einem anderen Knoten wiederzugeben. Ein Dienst, der dies leistet, soll in Zukunft in die AmICoM integriert werden. Die Ansätze hierfür und die bisherigen Ergebnisse werden in diesem Kapitel beschrieben.

Das Kapitel gliedert sich in zwei Abschnitte. Abschnitt 6.1 beschäftigt sich mit der Aufnahme von Audiodaten mit dem Mikrofon des Sensorboards. Die Wahl der Abtastfrequenz und die daraus entstehende Audioqualität werden näher erläutert. In Abschnitt 6.2 geht es um die Komprimierung von Audiodaten, um bei der Übertragung Bandbreite einzusparen.

6.1 Aufnahme von Audiodaten mit dem Mikrofon des Sensorboards

Das Mikrofon des Sensorboards soll vor allem menschliche Sprache aufnehmen. Um Audiodaten mit dem Imote2 und im AmICoM-Netzwerk verarbeiten zu können, müssen die analogen Daten des Mikrofons digitalisiert werden. Dies geschieht mit dem A/D-Wandler des Sensorboards. Der Signalpegel des analogen Signals wird in periodischen Abständen durch den A/D-Wandler in einen digitalen Wert gewandelt, dies nennt man Abtastung. Für alle Abtastvorgänge gilt das Nyquist-Shannonsche Abtasttheorem. Dies besagt, dass ein bandbegrenztes Signal mit einer Maximalfrequenz f_{max} mit einer mindestens doppelt so hohen Abtastfrequenz f_{abtast} abgetastet werden muss, damit keine Informationen verloren gehen und das Signal wieder vollständig rekonstruiert werden kann. Es muss also gelten:

$$f_{abtast} \geq 2 \cdot f_{max}$$

Wird das Abtasttheorem verletzt, treten so genannte Aliasing-Effekt auf, das bedeutet, dass der ursprüngliche Signalverlauf aus den Abtastwerten nicht mehr rekonstruiert werden kann und statt dessen falsche Werte rekonstruiert werden. Kommt es bei einer Audioabtastung zu einer Verletzung des Abtasttheorems, äußert sich dies durch Störungen, wenn das abgetastet Signal wiedergegeben werden soll. Im schlimmsten Fall sind die Audiodaten nicht mehr zu verstehen. Da die menschliche Stimme eine geringere Bandbreite besitzt als Musik, benötigt man eine geringere Abtastrate als für Musikaufnahmen, um menschliche Sprache in akzeptabler Qualität aufnehmen zu können.

Ein Beispiel für den unterschiedlichen Bedarf von Bandbreite bei Sprache und Musik ist der Vergleich der Abtastraten und Auflösung der Samples zwischen dem Audio-CD Format

und dem ISDN-Telefonnetz. Während für die Aufnahme einer Audio-CD mit 44,1 kHz abgetastet wird, wird Sprache für das ISDN-Telefonnetz nur mit einer Frequenz von 8 kHz abgetastet. Neben der Abtastrate spielt für die Qualität noch die Größe der Samples, das sind die einzelnen Abtastwerte, eine Rolle. Bei der Audio-CD beträgt die Auflösung der Samples 16 Bit, bei ISDN 8 Bit.

Für das Mikrofon des Sensorboards wird eine Abtastrate von 8 kHz und eine Samplegröße von 8 Bit gewählt. Zur Unterdrückung von Störungen wird ein Bandpass, bestehend aus Hoch- und Tiefpass, zur Bandbreitenbegrenzung des Signals verwendet. Da der A/D-Wandler des Sensorboards maximal 10 Bit als Wandlungswert liefert, ist auch die maximale Größe der Samples auf 10 Bit beschränkt. Damit man einen Abtastwert bequem über die SPI-Schnittstelle übertragen und weiterverarbeiten kann, bietet sich eine Samplegröße von 8 Bit an, da der Qualitätsverlust durch das Weglassen der zwei niederwertigsten Bits gering ist (siehe Abschnitt 3.3.1). Wenn die Audiodaten im AmICoM-Netzwerk genutzt werden sollen, ist die UART-Verbindung des AmICoM-Gateways mit einer Übertragungsrate von etwa 92 kBit/s die Verbindung mit der geringsten Datenrate. Bei einer Abtastrate von 8 kHz und einer Samplegröße von 8 Bit ergibt sich ein Datenstrom von 64 kBit/s (wie bei ISDN). Somit ist eine nur eine geringe Erhöhung der Abtastrate möglich, bevor die maximale Datenrate der UART-Verbindung erreicht wird. Bei gleichbleibender Abtastrate von 8 kHz kann die Auflösung der Samples auf maximal 11 Bit erhöht werden (Datestrom von 88 kBit/s), bevor die maximale Datenrate erreicht wird.

Die Integration des Mikrofons in SEnF wird in Kapitel 3 beschrieben. Um die Funktion des Mikrofons zu testen, wurde ein SDL-System erstellt, das Daten vom Mikrofon liest und über die UART-Schnittstelle ausgibt. Die so gewonnenen Audiodaten wurden am PC wiedergegeben, so dass man deren Qualität beurteilen kann. Nach mehreren Testläufen mit Sprache und mit Musik lässt sich zusammenfassen, dass die erreichte Qualität zufriedenstellend ist. Ein Rauschen ist bei jeder Aufnahme noch festzustellen, die Sprache und die Musik sind aber verständlich. Die Qualität der Aufnahme von Sprache ist besser, als von Musik, aber dies war mit einer Abtastfrequenz von 8 kHz nicht anders zu erwarten.

6.2 Audiokompression

Eine Möglichkeit, die Übertragung von Audiodaten im AmICoM-Netzwerk zu verbessern, ist, die Audiodaten zu komprimieren und somit die Anzahl der Pakete, die versendet werden müssen, zu vermindern. Ein weiterer Vorteil der Audiokompression ist, dass die Netzwerkauslastung im Imote2 Netzwerk gesenkt wird.

Mit Hilfe von Kompression ist es möglich, Bandbreite bei der Übertragung von Daten einzusparen. Heute findet Kompression vor allem im Multimedia-Bereich Anwendung, wenn Beispielsweise Videos, Bilder und Audiodateien aus dem Internet geladen werden. Ursprünglich wurden Kompressionsverfahren entwickelt, um den damals teuren Speicherplatz effizient nutzen zu können. Aufgrund der immer weiter sinkenden Speicherpreise ist dieses Ziel in den Hintergrund gerückt. Es gibt Kompressionsverfahren, die die Art der zu komprimierenden Daten nicht berücksichtigen. Der Vorteil dieser Verfahren, ist, dass sie auf alle Arten von Daten, egal ob Texte, Audio oder Bilddaten, anwendbar sind. Der Nachteil ist, das sie spezielle Eigenschaften von bestimmten Daten nicht ausnutzen können und nur allgemeine Verfahren anwenden. Diese Verfahren erzielen bei Audiodaten keine optimalen Ergebnisse [Sal07]. Für Audiodaten gibt es spezielle Kompressionsverfahren, deren Grundlagen im folgenden vorgestellt werden.

6.2.1 Grundlagen

Die verschiedenen Audiocodecs lassen sich anhand unterschiedlicher Kriterien unterscheiden. Man unterscheidet dabei in verlustlose Kompression und verlustbehaftete Kompression [Sal07]. Bei der verlustlosen Kompression werden die vorhandenen Audiodaten ohne Informationsverlust, und somit auch ohne Qualitätsverlust, komprimiert. Bei der verlustbehafteten Kompression hingegen gehen Informationen der Originaldaten verloren. Unter der Ausnutzung von bestimmten Eigenschaften des Gehörs kann erreicht werden, dass der Hörer das Fehlen von Informationen nicht bemerkt (siehe Abschnitt 6.2.1.2).

Ein weiteres Unterscheidungskriterium für Audiocodecs ist das Verhältnis von Kompressions- und Dekompressionsaufwand. Benötigen Kompression und Dekompression die gleiche Zeit, so spricht man von symmetrischen Codecs. Ist das Verhältnis verschieden, spricht man von asymmetrischen Codecs.

Fast alle Audiocodecs sind asymmetrisch. Weil die Kompression aufwändiger ist als die Dekompression, benötigen sie meist deutlich mehr Zeit zum Komprimieren der Audiodaten, als zum Dekomprimieren der Daten. Stark asymmetrische Verfahren eigenen sich vor allem, wenn Audiodaten gespeichert werden sollen. Sie werden meist nur einmal komprimiert und danach immer wieder dekomprimiert und abgespielt. Dann ist es natürlich von Vorteil, wenn das Abspielen von komprimierten Daten mit möglichst wenig Aufwand zu realisieren ist. Man nimmt dafür in Kauf, dass der Kompressionsvorgang aufwändiger ist. Bei der Übertragung von Sprache ist die Anforderung an den Codec, dass er die Audiodaten in Echtzeit enkodieren und dekodieren kann. Abhängig ist dies von der verwendeten Hardware, auf der der Codec läuft.

Es werden nun verschiedene grundsätzliche Methoden vorgestellt, um Audiodaten zu komprimieren. Allen Methoden ist gemein, dass sie sich spezielle Eigenschaften von Audiodaten zunutze machen, um die Daten zu komprimieren.

6.2.1.1 Verlustfreie Methoden

Die einzelnen Samples einer Audioabtastung stehen nicht isoliert, sondern sind im allgemeinen korreliert, das bedeutet sie stehen in Beziehung zueinander [Sal07]. Die Werte von mehreren aufeinanderfolgenden Samples haben fast immer kleine Abstände, große Sprünge zwischen den einzelnen Samples kommen selten vor. Wenn man statt den kompletten Samples nur die Abstände der Werte kodiert, erreicht man eine Kompression der Daten, da man für das Darstellen der Abstände weniger Bits benötigt als für die kompletten Samples. Ein weiterer Punkt ist, dass in Audiodaten bestimmte Samples häufiger vorkommen als andere. Nimmt als als Samplegröße 8 Bit, so erhält man Werte zwischen 0 und 255. Wenn Stille bei dem Samplewert von 128 liegt, schwankt ein Großteil der Werte um 128, die Extremwerte 0 und 255 kommen relativ selten vor [Sal07]. Diese Verteilung von Werten kann man ausnutzen und so die Daten komprimieren, indem man Werte, die häufiger vorkommen, mit weniger Bits kodiert, als Werte, die selten vorkommen.

Eine noch bessere Kompressionsrate erhält man, wenn man beide Techniken miteinander kombiniert. Die Abstände zwischen den Samples sind ebenfalls nicht gleichverteilt und können so auch mit unterschiedlichen Längen geordnet nach ihrer Auftrittshäufigkeit kodiert werden. Diese Methoden funktionieren dann am Besten, wenn alle Samples vor dem Beginn des Enkodierungsvorgangs bekannt sind und analysiert werden können. Ist dies nicht möglich, wie zum Beispiel bei der Sprachübertragung in Echtzeit, müssen Verteilungen im Voraus angenommen werden. Wenn die Art der Audiodaten bekannt ist, kann eine speziell auf diese Art Audiodaten zugeschnittene Verteilung verwendet werden.

6.2.1.2 Verlustbehaftete Methoden

Viele der verlustbehafteten Ansätze machen sich Eigenschaften des menschlichen Gehörs zunutze um Daten zu komprimieren. Das menschliche Gehör ist je nach Alter des Menschen in der Lage Töne von 20 Hz bis 20 kHz wahrzunehmen [Sal07]. Allerdings ist nicht nur die Frequenz des Tons entscheidend, sondern auch die Lautstärke. Sehr leise Töne werden von unserem Gehör nicht wahrgenommen. Bei der Stillekompression werden alle Samples, die in der Nähe des Stillewertes liegen, so behandelt, als wären sie Stille. Die Samples werden dann auf den Stillewert gesetzt (zum Beispiel auf 128 siehe Abschnitt 6.2.1.1). So erhält man relativ lange Sequenzen von dem Stillewert, auf den man das sogenannte Run-Length-Encoding anwendet. Beim Run-Length-Encoding werden lange Folgen desselben Wertes nicht einzeln kodiert, sondern es wird zunächst einmal der Wert kodiert und dann die Länge der Folge, des Wertes. Die Schwelle, ab der Samples als Stille behandelt werden, ist variabel. Mit steigender Schwelle steigt auch die Kompressionsrate, da die Folgen von Nullen länger werden, allerdings sinkt auch die Qualität. Der Hörer bemerkt ab einer Schwelle, dass die Audiodaten nicht mehr vollständig sind.

Eine weitere Methode ist das *Companding* (Kurzform für compressing/expanding) [Sal07]. Unser Gehör benötigt eine größere Auflösung bei leisen Tönen als bei lauten Tönen. Diese Eigenschaft macht sich das *Companding* zunutze, indem es Samples nichtlinear auf andere Werte abbildet und dabei Bits einspart. Es gibt viele verschiedene nichtlinerare Abbildungen, ein einfaches Beispiel ist folgenden Formel, die 16-Bit Samples auf 15-Bit Werte abbildet.

$$Wert = 32767 \cdot (2^{\frac{Sample}{65535}} - 1)$$

Die Samples, die leise Töne darstellen, und somit niedrig sind, werden mit dieser Formel genauer abgebildet als hohe Samples. Durch die Einsparung des Bits wird eine Kompression erreicht. Mit anderen Formeln lassen sich höhere Kompressionraten erreichen, jedoch mit sinkender Qualität. Das Besondere an dieser Methode ist, dass die Kompressionsrate schon im Voraus bekannt ist. Des weiteren müssen die Werte beim Enkodieren und Dekodieren nicht berechnet werden, sondern können in Tabellen abgelegt und einfach nachgeschlagen werden. Dies sorgt für schnelle Enkodierung und Dekodierung. Bekannte Anwendungen von *Companding* sind das A-Law und das μ-Law [Sal07]. Beide Methoden nutzen unterschiedliche nichtlineare Formeln und das μ-Law wandelt 13-Bit Samples in 8-Bit Werte, das A-Law wandelt 14-Bit Samples ebenfalls in 8-Bit Werte.

Wie oben schon erwähnt ist das menschliche Gehör nicht in der Lage, sehr leise Töne zu hören. Die Empfindlichkeit des Gehörs ist auch von der Frequenz des Tons abhängig. Die größte Empfindlichkeit besitzt unser Gehör im Bereich von 2 bis 4 kHz. Diese Empfindlichkeit ist jedoch veränderlich und sorgt für eine Maskierung von Tönen.

Bei der Frequenzmaskierung erhöht ein Ton einer bestimmten Frequenz die Wahrnehmungsschwelle für Töne mit Frequenzen, die in der Nähe der Frequenz des ersten Tons liegen. Somit können dann Töne, die man normalerweise hören würde, nicht mehr wahrgenommen werden. Alles, was vom Hörer sowieso nicht gehört werden kann, benötigt man nicht zur Kodierung und erreicht eine Komprimierung der Daten. Da hierbei Informationen verloren gehen, ist dies ebenfalls eine verlustbehaftete Methode.

Bei der Temporalmaskierung geht es darum, dass laute Töne ebenfalls dafür sorgen, dass die Wahrnehmungsschwelle des Gehörs heraufgesetzt wird. Das besondere an diesem Effekt ist, dass er nicht nur kurz nach dem lauten Ton dafür sorgt, dass man leisere Töne nicht hört, sondern auch kurz davor. Kann man diese Töne, die nicht mehr wahrgenommen werden, finden und herausfiltern, erreicht man ebenfalls eine Kompression.

6.2.1.3 Sprachcodecs

Es gibt neben Audiocodecs, die für alle Arten von Audiodaten geeignet sind, noch Codecs, die speziell für Sprache ausgelegt sind. Sprachcodecs erreichen meistens eine höhere Kompressionsrate als normale Codecs, sind dafür ausschließlich für Sprache geeignet. Die höhere Kompressionsrate wird dadurch erreicht, dass Sprache nur einen Bruchteil des Frequenzbereiches benutzt, der für uns hörbar ist. Sprache findet vor allem im Bereich von 500 Hz bis 2 kHz statt [Sal07].

Eine häufig angewandte Methode, um Sprache zu komprimieren, sind die sogenannten Source Codecs, die ein mathematisches Modell der menschlichen Sprache nutzen [Sal07]. Dieses Modell ist von Parametern abhängig, die bei der Enkodierung aus den Audiosamples berechnet werden. Ein Beispiel für einen Source Codec ist der *Linear Predictive Coder* (LPC) [Sal07]. Der LPC enkodiert immer komplette Frames von Audiodaten. Ein Frame besteht aus 160 Sprachsamples, was bei einer Abtastrate von 8 kHz 20 ms Sprache entspricht. Aus jedem Frame werden 13 Parameter für das mathematische Modell des Codecs berechnet. Auf der Gegenseite beim Dekodierungsvorgang werden nur diese Parameter benötigt, um mit Hilfe des Modells ein Ausgangssignal zu erzeugen, das dem Eingangssignal ähnelt. Auf diese Weise lassen sich sehr hohe Kompressionsraten bei passabler Qualität erreichen. Diese Methode wird vor allem bei Low-Bitrate Codecs, die eine minimale Bitrate von 2 kBit/s besitzen, verwendet.

6.2.2 Bekannte Codecs

Im folgenden sollen einige bekannte Audiocodecs vorgestellt werden. Es wird vor allem begründet, warum der eine Codec für das Szenario der Audiodatenübertragung des Imote2 geeignet ist und der andere nicht. Die hier erwähnten Codecs sind nur Beispiele, da es heute eine große Anzahl an verschiedenen Codecs gibt. Eine gute Übersicht über Codecs findet man in [Sal07] und in [Ger94].

6.2.2.1 MP3

Einer der bekanntesten Audiocodecs ist der MPEG-1 Audio Layer 3 (MP3). Er wurde ab 1987 im Fraunhofer Institut für Integrierte Schaltungen (IIS) in Erlangen entwickelt und 1992 von der Moving Picture Experts Group (MPEG) zusammen mit anderen Standards für Video und Audio standardisiert [mp3]. Standardisiert wurde jedoch nur das Datenformat und die Dekodierung, deswegen gibt es heutzutage viele unterschiedliche Enkoder für den MP3-Codec. Jedoch wurden verschiedene Rahmenbedingungen für Enkoder festgelegt, so zum Beispiel die Abtastrate der Audiosamples, die 32, 44,1 oder 48 kHz beträgt. Der ursprüngliche Standard unterstützt Ausgangsbitraten von 32 bis 224 kBit/s, was Kompressionsraten von 2,7 bis 24 entspricht [Pan95]. Aktuelle Enkoder, wie zum Beispiel das Open-source Projekt LAME [Lam], unterstützen Ausgangsbitraten von 32 bis 320 kBit/s. Desweiteren gibt es drei verschiedene Enkodierungsmodi:

- Konstante Bitrate, mit wechselnder Qualität
- Konstante Qualität, mit wechselnder Bitrate
- Durchschnittliche Bitrate, der Enkoder kodiert Teile von Audiodaten mit niedrigerer Bitrate, andere mit höhere Bitrate, je nach Komplexität der Audiodaten. Im Durchschnitt wird die vorgegebene Bitrate erreicht.

Als Kompressionsverfahren nutzt MP3 ein psychoakustisches Modell, das die in Abschnitt 6.2.1.2 beschriebenen Maskierungseffekte ausnutzt, um Daten, die das menschliche Gehör nicht wahrnehmen kann, herauszufiltern. Gleichzeitig erfolgt eine Aufspaltung des Audiosignals in 32 gleich breite Subbänder, die dann nichtlinear quantisiert werden [Pan95].

Der MP3-Codec eignet sich nur bedingt für die Verwendung auf dem Imote2. Die Ausgangsbitraten sind nicht so niedrig wie bei spezialisierten Sprachcodecs, die Qualität bei niedrigen Bitraten leidet stark und die Abtastrate des Mikrofons des Sensorboards wird nicht unterstützt.

6.2.2.2 FLAC

FLAC (Free Lossless Audio Codec) ist ein Vertreter der verlustlosen Audiocodecs und ist lizenzfrei über die Projekthomepage zu beziehen [FLA]. Er ist für alle Arten von Audiodaten geeignet und verwendet die unter 6.2.1.3 vorgestellte Methode, indem er versucht, mit einer mathematischen Funktion einen Block von Eingangsdaten zu beschreiben. FLAC besitzt nicht nur ein mathematisches Modell, sondern vier, um die Originaldaten möglichst genau anzunähern. In den meisten Fällen gelingt die exakte Darstellung der Eingangsdaten nicht und da FLAC ein verlustloses Verfahren ist, müssen die Unterschiede der berechneten Funktionen zu den Originaldaten gespeichert werden. Nur so ist es möglich, beim Dekodierungsvorgang die Originaldaten wiederherzustellen. Dieses Verfahren ist asymmetrisch, FLAC benötigt länger zum Enkodieren der Daten als zum Dekodieren. Mit FLAC lassen sich Audiodaten mit einer Abtastrate von 1 Hz bis 655350 Hz und einer Samplegröße von 4 bis 32 Bit enkodieren.

Die Kompressionsrate von FLAC liegt in etwa bei 0,5 und ist damit nicht so hoch wie die von verlustbehafteten Codecs, die bessere Kompression erzielen. Dafür gibt es mit FLAC keinerlei Qualitätseinbußen. Wegen der niedrigen Kompressionsrate ist FLAC weniger für den Einsatz auf dem Imote2 geeignet.

6.2.2.3 Speex

Speex ist ein verlustbehafteter Audiocodec speziell für Sprache [Val07]. Er unterstützt die Abtastraten von 8, 16 und 32 kHz. Der enkodierte Datenstrom hat eine Bitrate von 2 bis 44 kBits/s, wobei sowohl feste Bitraten als auch variable Bitraten unterstützt werden. Die Kompressionsrate sowie die Berechnungskomplexität des Codecs kann eingestellt werden. Grundlage von Speex ist das CELP-Verfahren (Code Excited Linear Prediction), das ein hybrides Verfahren ist und sowohl das oben vorgestellte LPC als auch andere Kompressionstechniken verwendet. Es gibt eine verfahrensbedingte Verzögerung von 30 ms, die nicht von der Komplexität des Kompressionsalgorithmus beeinflußt wird, da zunächst eine bestimmte Anzahl von Audiosamples vorliegen muss, um den Enkodiervorgang zu starten. Zusätzlich kann es durch die Berechnungen des Algorithmus zu Verzögerungen kommen.

Die geringen Bitraten, die Eignung für Sprache und die Tatsache, dass Speex frei verfügbar und somit auch der Quellkode offen ist, haben uns dazu bewogen Speex auf dem Imote2 zu testen. Darüberhinaus gab es eine fertig entwickelte SDL-Komponente für Speex in der AG.

6.2.2.4 iLBC

Der internet Low Bitrate Codec (iLBC) ist ein Sprachcodec, der vor allem für Internettelefonie entwickelt wurde [iLB04]. Audiodaten für den iLBC werden mit 8 kHz abgetastet. Die

Ausgangsbitraten betragen 13,3 kBit/s bei 30 ms langen Frames und 15,2 kBit/s bei 20 ms langen Frames. Die Ausgangsbitrate ist also im Gegensatz zu anderen Codecs fest und nicht frei wählbar. Das besondere an iLBC ist, dass verlorengegangene Pakete sich nicht auf nachfolgende Pakete auswirken. Im Gegensatz zu anderen CELP basierten Codecs wie Speex benutzt iLBC eine Version des LPC, die es möglich macht jeden Frame separat zu enkodieren und dekodieren [AKH+02]. CELP benötigt hingegen auch die Daten von älteren Frames um neue Frames richtig dekodieren zu können. Somit führt der Verlust eines Frames auf einer Übertragungstrecke bei CELP dazu, dass der Fehler sich durch mehrere Frames fortsetzt auch wenn diese nicht verloren gegangen sind. Beim iLBC tritt dieser Effekt nicht auf, verlorengegangene Frames haben keine Auswirkungen auf Nachbarframes. Eine verfahrensbedingte Verzögerung von 20 bzw 30 ms tritt auf, da zunächst ein gesamter Frame vorliegen muss, um mit der Enkodierung zu starten.

iLBC scheint für die Verwendung auf dem Imote2 geeignet zu sein. Er erreicht zwar nicht so geringe Bitraten wie der Speex-Codec, aber diese Bitraten reichen immer noch aus, um den Datenverkehr erheblich zu reduzieren. Darüberhinaus ist die Robustheit des Codecs gegenüber Paketverlusten ein Vorteil. Bisher wurde iLBC noch nicht auf dem Imote2 getestet.

6.2.3 Audio-Codecs auf dem Imote2

Zum Test des Speex-Codec wurde ein einfaches SDL-System implementiert, das Audiodaten vom Sensorboard des Imote2 erhält, diese mit dem als SDL-Package vorliegenden Speex-Codec enkodiert und über die CC2420-Schnittstelle versendet. Mit einem weiteren Imote2 können die enkodierten Pakete empfangen werden und über die UART-Schnittstelle ausgegeben werden. Um den Speex-Enkoder zu benutzen, muss mittels des Signals SetQuality die Qualität eingestellt werden und mit dem Signal GetFrameSize die Framegröße abgefragt werden. Die Framegröße ist die Anzahl der Samples, die der Enkoder für jeden Enkodierungsvorgang benötigt. Mit dem Signal Encode können dem Enkoder Audiodaten, die die Länge der Framegröße haben müssen, gesendet werden, diese werden enkodiert und dann mit dem Signal Encoded wieder zurück gesendet.

Mehrere Tests ergaben, dass die Rechenleistung des Imote2 momentan nicht ausreicht, um einen Audiostrom in Echtzeit zu kodieren. Der Imote2 benötigt zum Enkodieren eines Audioframes niedrigster Qualität mehr als 10mal soviel Zeit, wie das Sensorboard benötigt, um einen neuen unkomprimierten Audioframe zu liefern. Der Enkodierungsvorgang ist im Moment mindestens um den Faktor 10 zu langsam. Allerdings wird der Imote2 neben der Enkodierung auch noch durch das SDL-System, die Kommunikation mit dem PIC des Sensorboards und dem Senden von Paketen mit dem CC2420 belastet. Durch eine Optimierung der Kommunikation mit dem PIC und dem CC2420-Modul erhoffen wir uns einen Performanzschub. Trotzdem kann nicht mit Sicherheit gesagt werden, ob der Imote2 überhaupt in der Lage ist, den Audiostrom mit diesem Codec in Echtzeit zu kodieren. Ist die Enkodierung in Echtzeit möglich, so ist es auch die Dekodierung, da diese weniger Rechenleistung benötigt [Val07].

Weiterhin sollten auch noch andere Codecs, wie zum Beispiel iLBC, auf dem Imote2 getestet werden, um festzustellen, ob diese bessere Resultate erzielen. Vorstellbar für den Einsatz auf dem Imote2 ist ein abgewandeltes A-Law bzw. μ-Law, da diese Verfahren kaum Rechenzeit benötigen, sondern mit Nachschlagen in Tabellen arbeiten.

Kapitel 7

Test der AmICoM

Die in den vergangenen Kapiteln beschriebenen Implementierungen müssen getestet werden, um die ordnungsgemäße Funktion sicherzustellen. Zu diesem Zweck wurde ein kleines AmICoM-Netzwerk aufgebaut, das in diesem Kapitel vorgestellt werden soll. Im ersten Abschnitt geht es um den Aufbau des Systems und der beteiligten Knoten, im zweiten um die Java-Anwendung, die für die PC-Knoten entwickelt wurde und die Dienste des Sensorboards nutzt. Im letzten Abschnitt werden die Ergebnisse, die aus dem Testsystem gewonnen werden konnten, kurz zusammengefasst.

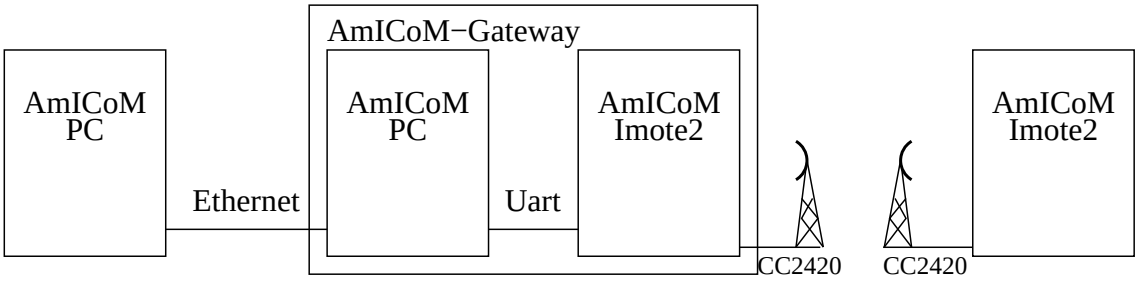


Abbildung 7.1: Testaufbau eines AmICoM-Netzwerkes

7.1 Aufbau des Testsystems

Das Testsystem besteht aus vier Knoten (siehe Abbildung 7.1). Ein PC und ein Imote2 bilden das AmICoM-Gateway und sind gleichzeitig AmICoM-Knoten (siehe Kapitel 5). Zusätzlich gibt es einen PC und einen Imote2, die normale AmICoM-Knoten sind. Das Testsystem besteht somit aus einem PC-Teilnetz und einem Imote2-Teilnetz, die über das Gateway miteinander verbunden sind. Die PCs sind untereinander mittels Ethernet vernetzt, die Imote2 verwenden den CC2420-Transceiver zur Kommunikation in ihrem Teilnetz. Auf allen Knoten werden die entsprechenden SDL-Systeme aus den Kapiteln 4 und 5 installiert. Die Imote2 registrieren ihre Sensorboard-Dienste in der AmICoM, damit diese von den PC-Knoten genutzt werden können. Um unterschiedliche Sensorboard-Dienste einzeln zu testen, kann man entscheiden, welche der implementierten Dienste auf dem Imote2 aktiv sind. Damit die PC-Knoten die Dienste nutzen können, benötigen sie eine Applikation, die die Dienste der AmICoM abonniert und mit ihnen Daten austauscht. Diese Applikation wurde in Java implementiert und wird im nächsten Abschnitt vorgestellt.

7.2 Java-Anwendung

Die entwickelte Java-Applikation setzt auf der AmICoM auf und hat die Aufgabe, die von den Imote2 registrierten Sensorboard-Dienste zu abonnieren und zu nutzen. Die Java-Applikation tauscht über das *VSApplicationInterface* mit dem SDL-System Daten aus (siehe Abschnitt 4.3). Das Java-*AmICoM-Interface* setzt auf dem *VSApplicationInterface* auf und implementiert die in Abschnitt 4.2 beschriebene API für die AmICoM. Über diese API können Dienste registriert und abonniert werden, Daten mit anderen Diensten ausgetauscht und Informationen aus dem AmICoM-Netzwerk abgerufen werden.

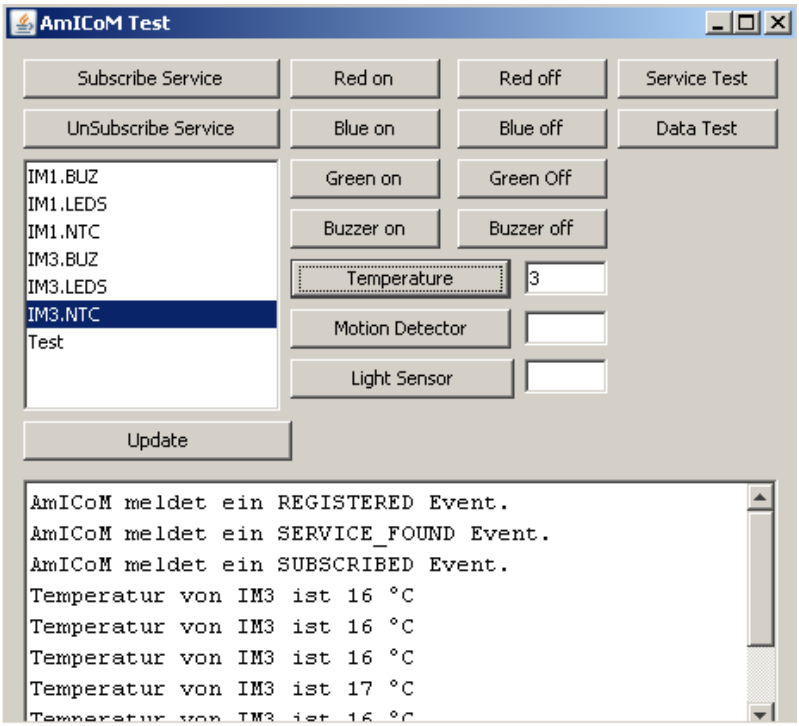


Abbildung 7.2: Benutzeroberfläche der Java-Applikation

Um die Applikation möglichst bedienen zu können, wurde eine graphische Benutzeroberfläche entwickelt, die in Abbildung 7.2 dargestellt wird. Die Benutzeroberfläche zeigt alle im AmICoM-Netzwerk verfügbaren Dienste in dem linken Textfenster an. Die Schaltfläche *Update* aktualisiert die Liste der verfügbaren Dienste. Markiert man einen Dienstnamen in der Liste und klickt auf die Schaltfläche *Subscribe Service*, so wird dieser Dienst abonniert und kann nun verwendet werden. Über das Abonnement wird der Benutzer im unteren Textfenster informiert, allerdings fehlt eine Zuordnung von Dienstnamen zu den Ereignissen die über das Java-*AmICoM-Interface* empfangen werden. Es kann nur die Art des empfangenen Ereignis ausgegeben werden. Mit der Schaltfläche *UnSubscribe Service* wird das Abonnement des Dienste wieder zurückgenommen. Dadurch, dass der Dienstname von Sensorboard-Diensten auch den Hostnamen des Imote2 enthalten, auf dem sie registriert wurden, ist eine eindeutige Zuordnung von Diensten zu Imote2-Knoten möglich.

Die Schaltfäche *Service Test* registriert einen Testdienst, der keine Funktion besitzt, auf dem PC-Knoten der AmICoM. Um die Filterfunktionalität des Gateways zu überprüfen, benötigt man registrierte Dienste in beiden Teilnetzen der AmICoM. Der Testdienst wird von dem anderen PC-Knoten im Testsystem abonniert, dann müssen Daten dieses Dienstes

im Gateway gefiltert werden, da sich Dienstanbieter und Dienstnutzer im selben Teilnetz befinden (siehe Abschnitt 5.3). Die Schaltfläche *Data Test* versendet Testdaten an alle Nutzer des Testdienstes, wenn der Testdienst registriert wurde.

Die Schaltflächen in der Mitte des Fenster dienen allesamt zur Kommunikation mit den Dienstanbietern auf den Imote2. Der Dienst, der genutzt werden soll, muss im Fenster der verfügbaren Dienste markiert sein. Möchte man beispielsweise die rote LED des Sensorboards von Imote2 Nummer 3 einschalten, so markiert man in der Liste den Eintrag *IM3.LEDS* und klickt anschließend auf die Schaltfläche *Red On*. Voraussetzung ist, dass der PC-Knoten den LED-Dienst des Imote2 Nummer 3 abonniert hat. Mit den Schaltflächen *Buzzer On* und *Buzzer Off* wird der Piezosummer des Sensorboards gesteuert.

Die Werte des Temperatursensors, des Bewegungsmelders und des Lichtsensor können mittels den Schaltflächen *Temperature*, *Motion Detection* und *Light Sensor* abgerufen werden. In die Textfelder rechts dieser Schaltflächen kann das Intervall in Sekunden für eine periodische Lieferung der Werte angegeben werden. Wird eine 0 übergeben so wird nur ein einzelner Wert geliefert und eine eventuell laufende periodische Lieferung gestoppt. Die von den Imote2 gelieferten Werte werden im unteren Textfenster ausgegeben.

Das Mikrofon des Sensorboards lässt sich im Testsystem nicht verwenden, da dieser Dienst noch nicht in die AmICoM integriert ist.

7.2.1 ASN.1 Datenstrukturen unter Java

Neben der graphischen Oberfläche wird eine ASN.1 Implementierung für Java benötigt, da die Schnittstellen der Dienste als ASN.1 Datenstrukturen realisiert sind (siehe Abschnitt 4.5). Zu diesem Zweck wird das *CODEC*-Paket verwendet [COD], das im Fraunhofer Institut für Graphische Datenverarbeitung in Darmstadt in Zusammenarbeit mit der Technischen Universität Darmstadt entwickelt wurde [SER07]. Dieses Paket kann ASN.1 Datenstrukturen auf Java-Klassen abbilden und so ASN.1 in Java integrieren. Die Abbildung von ASN.1 Datentypen in Java muss vom Entwickler selbst vorgenommen werden, das *CODEC*-Paket besitzt keinen Compiler, der aus einer ASN.1-Spezifikation automatisch Java-Kode generiert. Die in Java spezifizierten ASN.1 Datenstrukturen können mit Werten belegt werden und diese Werte können mittels den *Distinguished Encoding Rules* (DER) enkodiert und dekodiert werden. Das *CODEC*-Paket stellt BER-Enkoder und BER-Dekoder bereit. Da die *Distinguished Encoding Rules* eine Untermenge der in der AmICoM verwendeten *Basic Encoding Rules* sind, kann der BER-Dekoder auch Bytefolgen dekodieren, die mit den DER-Enkoder enkodiert wurden und umgekehrt.

```
1 LedPacket ::= SEQUENCE {
2   color    INTEGER, -- Farbe der LED
3   on       BOOLEAN, -- LED an oder aus
4   interval INTEGER  -- Blinkfrequenz in Sekunden (0 kein blinken)
5 }
```

entspricht folgendem Java-Kode:

```
1 public class LedPacket extends ASN1Sequence {
2
3     private ASN1Integer color;
4     private ASN1Boolean on;
5     private ASN1Integer interval;
6
7     public LedPacket(){
8         super(3);
9         color = new ASN1Integer(1);
```

```
10          on = new ASN1Boolean(false);
11          interval = new ASN1Integer(0);
12          add(color);
13          add(on);
14          add(interval);
15      }
16  }
```

Daneben gibt es für die Java-Klasse noch Set- und Get-Methoden für die einzelnen Parameter des *LedPacket*-Datentyps. Die anderen ASN.1 Datentypen der Spezifikation wurden auf die gleiche Weise integriert. Somit ist die Schnittstelle der Imote2 Sensorboard-Dienste unter Java im Testsystem nutzbar.

7.3 Ergebnisse

Durch den Einsatz des Testsystems konnte Folgendes gezeigt werden:

- Die AmICoM lässt sich auf den Imote2 instantiieren und funktioniert ordnungsgemäß.
- Die Funktionalität der einzelnen Sensorboard-Dienste (außer dem Mikrofon-Dienst) konnte überprüft werden. Die Dienste werden korrekt im AmICoM-Netzwerk registriert und können von einem PC-Knoten mittels der Java-Applikation abonniert und genutzt werden.
- Die Funktionalität des Gateways wurde getestet. Alle Managementpakete passieren das Gateway und sind im anderen Teilnetz verfügbar. Dies kann damit überprüft werden, dass ein Dienst, den der Imote2 anbietet, im PC-Netzwerk als vorhanden angezeigt wird (mittels *Alive*-Nachrichten). Sensorboard-Dienste, die von einer Java-Applikation im PC-Netzwerk abonniert werden, werden korrekt der Durchgangsliste hinzugefügt und bei Bedarf wieder entfernt. Dies wird durch Statusausgaben festgestellt, die anzeigen, ob ein Dienst der Durchgangsliste hinzugefügt wird und wann er wieder von ihr gelöscht wird. Um zu überprüfen, dass Dienste, die ausschließlich in einem Teilnetz angeboten und genutzt werden, vom AmICoM-Gateway gefiltert werden, wird der oben beschriebene Test-Dienst auf einem PC-Knoten registriert und auf einem anderen abonniert. Die Statusausgaben des Gateways zeigen an, dass dieser Dienst nicht der Durchgangsliste hinzugefügt wird und dass Datenpakete dieses Dienste nicht weitergeleitet werden.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit wurden verschiedene Sensorboard-Dienste für die AmICoM-Version des Imote2, die von der PC-Plattform portiert wurde, vorgestellt und ein Gateway mit Filterfunktionalität entwickelt, dass AmICoM-Subnetze miteinander verbindet. Treiber für die Sensoren und Aktuatoren des Imote2-Sensorboards wurden implementiert und in das SEnF integriert, so dass das Sensorboard aus einem SDL-System heraus nutzbar ist. Mittels dieser Treiber wurden Dienste für die Imote2-Version der AmICoM spezifiziert. Ihre Funktion konnte mit einem Testsystem, das ein kleines AmICoM-Netzwerk darstellt, überprüft werden. Das AmICoM-Gateway, das PC-Netzwerk und Imote2-Netzwerk mittels einer UART-Verbindung miteinander verbindet, konnte ebenfalls mit dem Testsystem überprüft werden. Das Gateway besitzt eine Filterfunktionalität, die die Daten von Diensten, die im jeweiligen Teilnetz nicht benötigt werden, filtert und so für eine Reduktion des Datenverkehrs sorgt. Die Ansätze des Audiodienstes für die AmICoM wurden vorgestellt und die sich ergebenden Probleme erläutert. Die Vorteile einer Audiokomprimierung für den Audiodienst und die Eigenschaften unterschiedlicher Audiokompressionsverfahren wurden ebenfalls diskutiert.

Offene Punkte

Ein Ziel ist, den Audiodienst des Sensorboards in die AmICoM zu integrieren. Um Audiodaten zu übertragen, sollen die SEnF-Module für die SPI- und die UART-Schnittstelle auf den DMA-Modus umgestellt werden, um sie effizienter zu machen. Datentransfers in und aus den Registern der Kommunikationsschnittstellen belasten dann nicht mehr den Hauptprozessor des Imote2, und man hat mehr Leistungsreserven zur Verfügung. Die Kommunikation zwischen Imote2 und PIC des Sensorboards kann ebenfalls verbessert werden. Momentan erfordert die Übertragung eines Bytes Audiodaten zwei Interrupts im Imote2. Ein großer Teil der Leistung des Imote2 wird bei Audioübertragung für die Interruptverarbeitung verwendet. Das Ziel ist es, die Anzahl der auftretenden Interrupts zu reduzieren. Sobald die entsprechenden SEnF-Module überarbeitet sind, können weitere Audiocodecs auf der Imote2 Plattform getestet werden. Sollte sich herausstellen, dass die Rechenleistung des Imote2 nicht für die Enkodierung in Echtzeit mit einem Source-Codec ausreicht, ist es möglicherweise sinnvoll, Codecs zu verwenden, die keine aufwändigen Berechnungen erfordern, wie zum Beispiel ein abgewandeltes A-Law oder μ-Law.

Ein weiterer Punkt ist, dass es bei Testläufen mit mehreren Imote2 im AmICoM-Netzwerk, die viele Dienste bereitstellen festgestellt wurde, vermehrt zu Paketverlusten kommt, da die momentane Implementierung der MAC-Ebene das Paket nicht neu sendet, wenn der

Kanal des CC2420 belegt ist, sondern verwirft. Es gilt diese Paketverluste zu minimieren und den Übertragungskanal des CC2420 besser auszunutzen.

Erweiterung der AmICoM

In die AmICoM können verschiedene in der AG entwickelten Lösungen integriert werden, um die Funktionalität der AmICoM zu erweitern und zuverbessern.

MacZ, die in der AG entwickelte MAC-Ebene, stellt umfangreiche MAC-Funktionalitäten zur Verfügung, unter anderen auch die Unterteilung des Medium in Übertragungsslots [BGK07]. MacZ enthält auch einen Quality-of-Service Ansatz und bietet Reservierungen für Übertragungsslots an. Wird MacZ in die AmICoM integriert, können damit die Paketverluste durch die Kanalbelegung (siehe oben) vermieden werden und der Kanal kann besser ausgelastet werden.

Eng verbunden mit dem Einsatz von MacZ in der AmICoM ist die geplante Integration einer Quality-of-Service Lösung in die AmICoM [Web08]. Mit ihr könnte für den Audiodienst eine feste Bandbreite garantiert werden, was die Zuverlässigkeit des Dienstes erhöht. Die QoS-Lösung verwaltet Reservierungen und kann diese in MacZ vornehmen.

Eine Portierung der AmICoM auf andere Hardware zur Erweiterung des Anwendungsspektrums ist möglich, wenn die Hardware gewisse Anforderungen erfüllt. Der Arbeitsspeicher sowie der FLASH-Speicher müssen groß genug für ein AmICoM-System sein. Auf dem Imote2 belegt eine Instanz der AmICoM etwa 1,5 MB FLASH-Speicher und 100 kB Hauptspeicher. Da der Kode-Generator ConTraST C++-Kode erzeugt, muss ein C++-Compiler für die Hardwareplattform verfügbar sein.

Einsatz der AmICoM

Der PC-Version der AmICoM ist momentan schon innerhalb des BelAmI Projektes der TU Kaiserslautern in Zusammenarbeit mit dem Fraunhofer Institut für experimentelles Softwareengineering in Kaiserslautern [bel] im Einsatz. In der Zukunft sollen weitere Dienste für die AmICoM entwickelt werden, und der Einsatz des Imote2 im AmICoM-Netzwerk sowie des Gateways innerhalb des BelAmI-Projektes ist geplant. So soll zum Beispiel ein RFID-Lesegerät an den Imote2 angeschlossen werden. Mittels eines Bodenbelags, in den RFID-Tags integriert sind, kann die Position des Imote2 in einem Raum festgestellt werden. Trägt ein Mensch den Imote2 und das RFID-Lesegrät bei sich, kann somit auf die Position des Menschen in dem Raum geschlossen werden. Diese Information kann dann innerhalb des AmICoM-Netzes als Lokalisations-Dienst anderen Knoten zur Verfügung gestellt werden. Weitere Anwendungsmöglichkeiten, wie Audioanwendungen oder die Messung von Körpertemperatur, sind innerhalb dieses Projektes für den Imote2 vorstellbar.

Anhang A

ASN.1 Datentypen für Sensorboard-Dienste

```
1 ImoteServicePackets DEFINITIONS AUTOMATIC TAGS::= BEGIN
2 -- LED-Dienst
3 LedPacket ::= SEQUENCE {
4   color     INTEGER, -- Farbe der LED
5   on        BOOLEAN, -- LED an oder aus
6   interval INTEGER  -- Blinkfrequenz in Sekunden (0 kein blinken)
7 }
8
9 -- Piezosummer-Dienst
10 BuzPacket ::= SEQUENCE {
11   on BOOLEAN,
12   interval INTEGER -- 0 bedeutet dauerhaft
13 }
14
15 -- Temperatur-Dienst
16 NtcPacket ::= INTEGER
17
18 NtcDataPacket ::= INTEGER
19
20
21 -- Lichtsensor-Dienst
22 TslPacket ::= INTEGER
23
24 TslDataPacket ::= INTEGER
25
26
27 -- Bewegungsmelder-Dienst
28 PirPacket ::= INTEGER
29
30 PirDataPacket ::= INTEGER
31
32 -- Mikrofon-Dienst
33 MicPacket ::= SEQUENCE {
34   on BOOLEAN,
35   bitrate INTEGER
36 }
37
38 MicDataPacket ::= SEQUENCE {
39   bitrate INTEGER,
40   data OCTET STRING
41 }
42 END
```

Anhang B

Mit ConTraST generierte Dokumentation des Prozesses *GatewayFilter*

Name: GatewayFilter

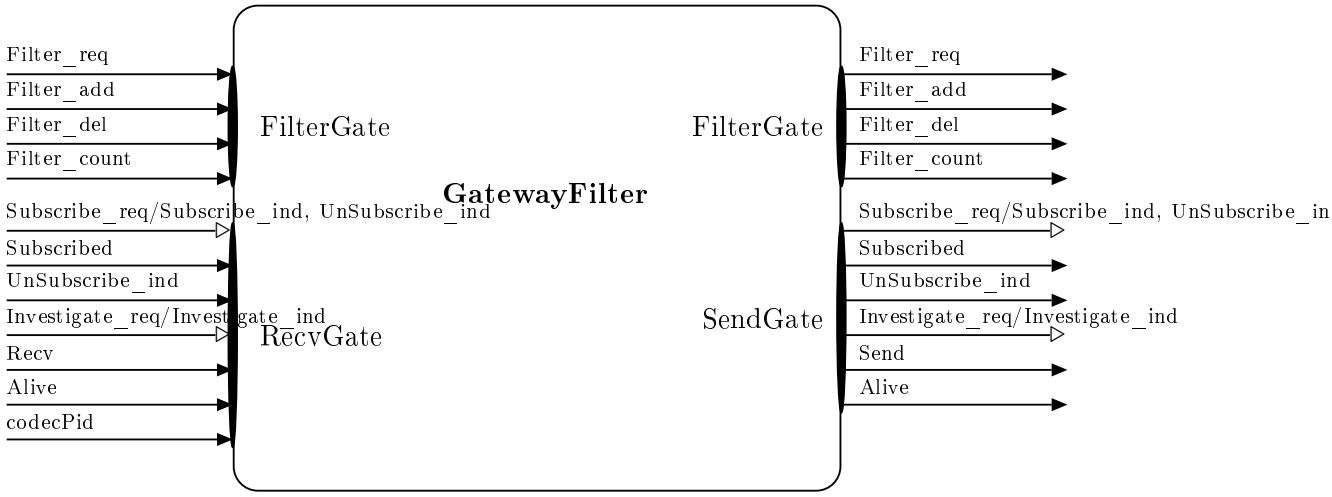
Version: 1.0

Author: Thorsten Schmelzer

Intent

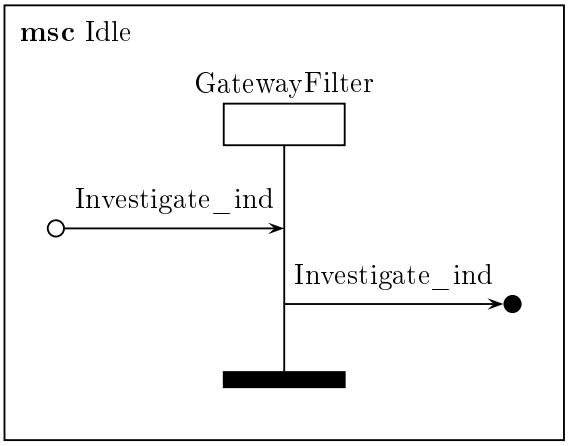
This package provides a symmetrical filter for the AmICoM-Gateway, which connects two AmICoM subnets. The goal of this filter is to reduce data traffic betweeen the two subnets. One instance of the process GatewayFilter is for one communication direction. Two instances of GatewayFilter form the complete Filter. AmICoM management packet pass the filter, whereas data packets are filtered if the service user and the service provider of a service are in the same subnet. The GatewayFilter has a list which stores the names of the services, which are allowed to pass the gateway. This list is called pass list.

Interface signature



Interface behaviour

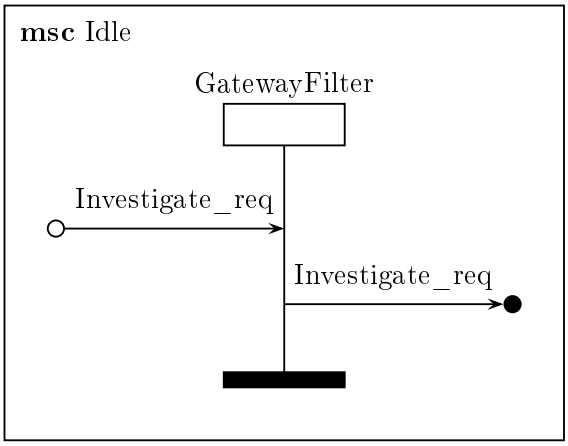
Scenario 1



Description:

The filter is ready to operate. The signal *Investigate_ind* is the answer to signal Investigate_req and includes all available services. The signal *Investigate_ind* sends the information through the Gateway to the other network.

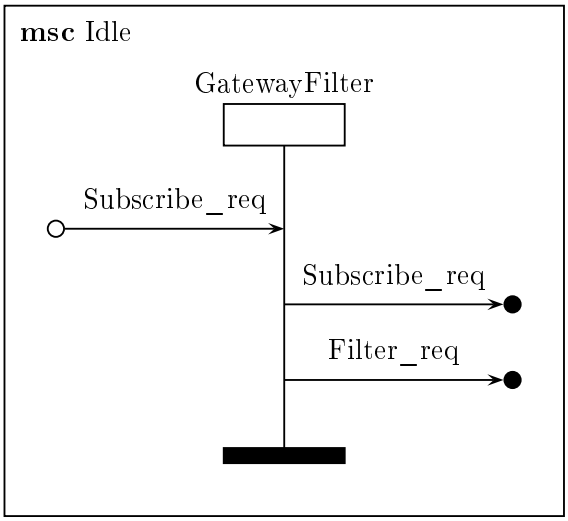
Scenario 2



Description:

The filter is ready to operate. The signal *Investigate_req* indicates a request for all available services. The signal *Investigate_req* sends the request through the Gateway to the other network.

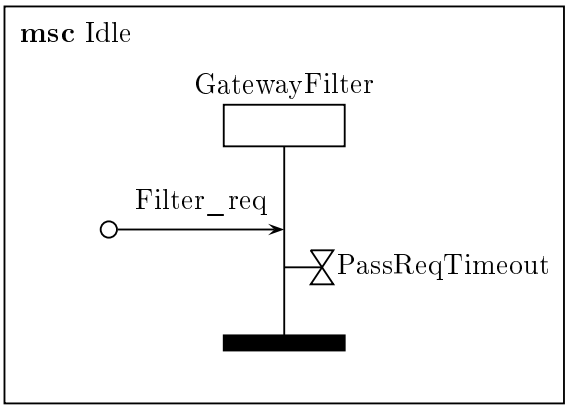
Scenario 3



Description:

The filter is ready to operate. The signal *Subscribe_req* indicates an request to subscribe for a service. The signal *Subscribe_req* sends the request through the gateway to the other network and the signal *Filter_req* informs the other GatewayFilter to wait for a *Subscribe_ind* for this service.

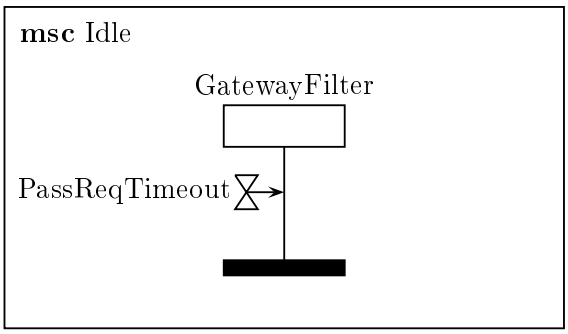
Scenario 4



Description:

The filter is ready to operate. The signal *Filter_req* tells the GatewayFilter to wait for a *Subscribe_ind* from this service. The timer *PassReqTimeout* is set for a duration of 4 time units.

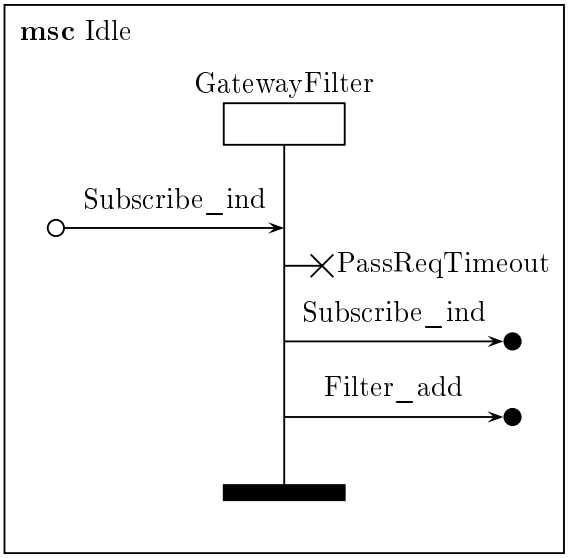
Scenario 5



Description:

The filter is ready to operate. The timer *PassReqTimeout* indicates that the GatewayFilter should no longer wait for a *Subscribe_ind* from the passed service.

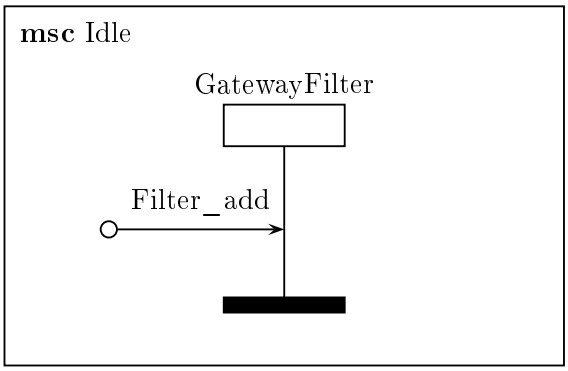
Scenario 6



Description:

The filter is ready to operate. The signal *Subscribe_ind* is the answer to the Subscribe_req signal and indicates that the service user has successfully subscribed. The timer *PassReqTimeout* is reset. Next, the signal *Subscribe_ind* sends the Indication through the gateway to the other network and next, the signal *Filter_add* tells the other Gateway filter that packets from this service are allowed to pass the gateway.

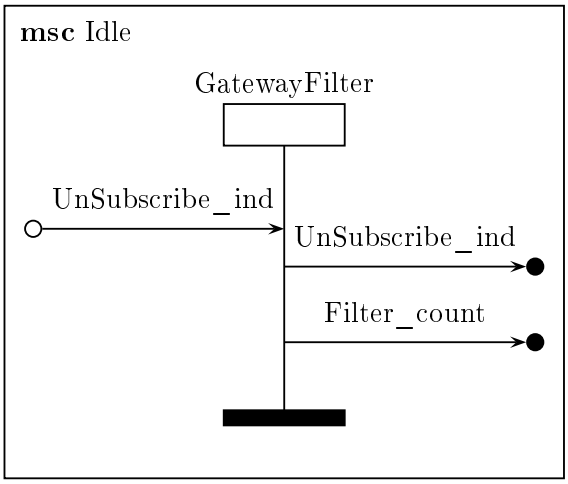
Scenario 7



Description:

The filter is ready to operate. The signal *Filter_add* adds the service with the servicename to pass list. No further operations are performed.

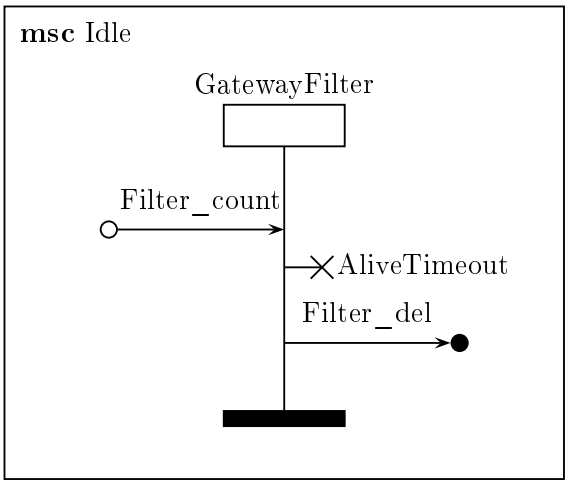
Scenario 8



Description:

The filter is ready to operate. The signal *UnSubscribe_ind* indicates that a service user has unsubscribed from a service. The signal *UnSubscribe_ind* sends the indication through the gateway to the other network and the signal *Filter_count* informs the other GatewayFilter that there is one subscriber less for this service.

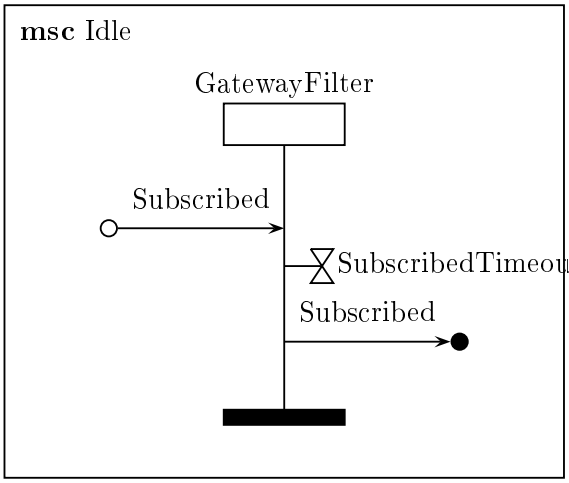
Scenario 9



Description:

The filter is ready to operate. The signal *Filter_count* indicates that there is one subscriber less for a service in the other network. If there is no more subscriber for this service in the other network , the timer *AliveTimeout* is reset. Next, the signal *Filter_del* informs the other GatewayFilter to delete the service from the pass list.

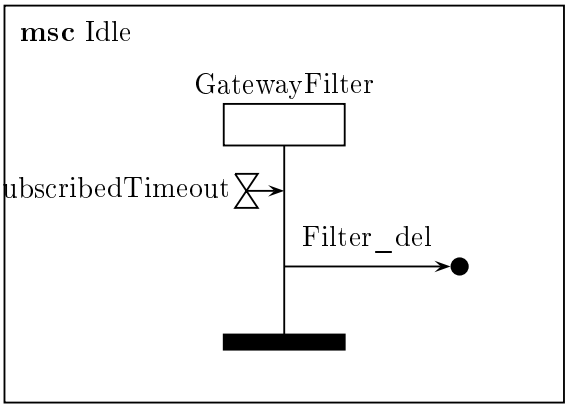
Scenario 10



Description:

The filter is ready to operate. The signal *Subscribed* indicates that there is still a service user for this service in the network. The timer *SubscribedTimeout* is set for a duration of 16 time units. Next, the signal *Subscribed* send the message through the gateway to the other network.

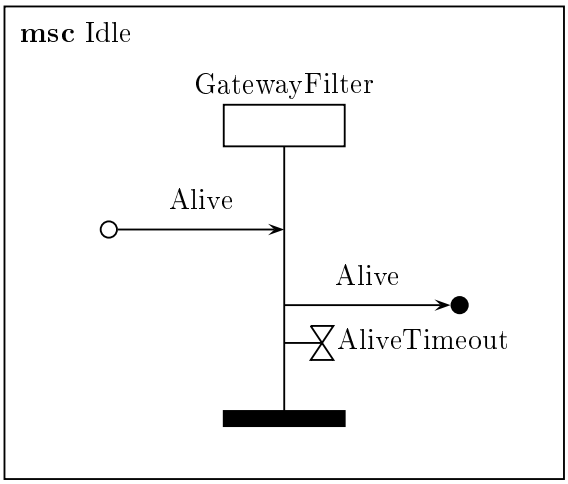
Scenario 11



Description:

The filter is ready to operate. The timer *SubscribedTimeout* indicates that the service is no longer subscribed from this network. The signal *Filter_del* informs the other GatewayFilter to delete the service from the list of allowed services.

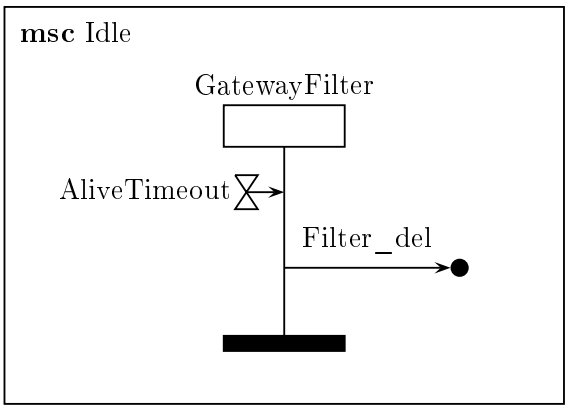
Scenario 12



Description:

The filter is ready to operate. The signal *Alive* indicates that a service is still available. The signal *Alive* sends the Alive message through the Gateway to the other network. The timer *AliveTimeout* is set for a duration of 4 time units.

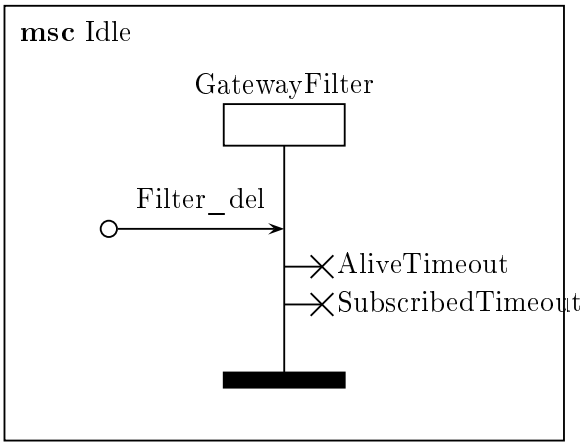
Scenario 13



Description:

The filter is ready to operate. The timer *AliveTimeout* indicates that the service with this name is no longer available and is deleted from the pass list. The signal *Filter_del* informs the other GatewayFilter to delete the service from the pass list.

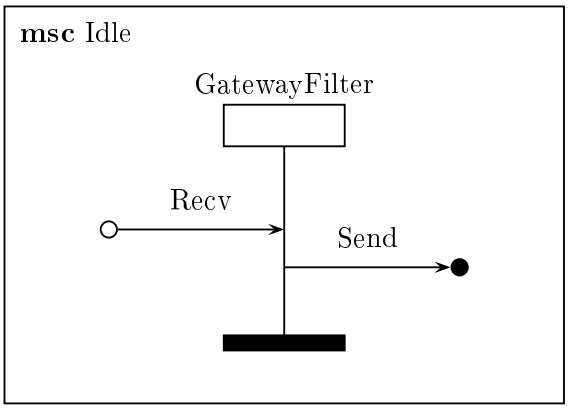
Scenario 14



Description:

The filter is ready to operate. The signal *Filter_del* deletes a service from the pass list. The timer *AliveTimeout* is reset. The timer *SubscribedTimeout* is reset.

Scenario 15



Description:

The filter is ready to operate. The signal *Recv* indicates the reception of data from a service. If the service is in the pass list, the signal *Send* sends the data through the gateway to the other network.

Imported and exported definitions

Used packages

- Signals
- AddressType (data type definition)

- LocalService (signal definition)
- ServiceUserProvider (signal definition)
- ServiceManagement
- LogIF (data type definition)
- AmiComCoDec (signal definition)

Required definitions

- Data type Logging
- Data type AddressType
- Signal Subscribe_req(Integer,AddressType)
- Signal Subscribed(AddressType,AddressType)
- Signal UnSubscribe_ind(Integer,AddressType,Boolean)
- Signal Investigate_req(Integer,AddressType)
- Signal Send(Integer,AddressType,Octet_string)
- Signal Alive(AddressType)
- Signal Recv(Integer,AddressType,Octet_string)
- Signal codecPid

Provided definitions

- Process type GatewayFilter
- Signal Filter_req, Signal Filter_add, Signal Filter_del, Signal Filter_count

Checklist

– *none* –

Anhang C

CD zur Arbeit

Zu dieser Arbeit wurde eine CD erstellt, die folgendes enthält:

- Die komplette SDL-Spezifikation der AmICoM (PC- und Imote2-Version sowie das AmICoM-Gateway).
- SDL-Spezifikation des *Sniffer* für Imote2, mit dem es möglich ist, die CC2420-Schnittstelle des Imote2 abzuhören und die empfangenen Daten über die UART-Schnittstelle auszugeben.
- Die MSCs aus Kapitel 5 (befinden sich im AmICoM-Unterordner).
- Die Dokumentation des SDL-Prozesses *GatewayFilter* (unveränderte von ConTraST generierte Version und abgeänderte Version).
- Die in dieser Arbeit verwendete Literatur, soweit sie frei als PDF-Dokument verfügbar ist.

Literaturverzeichnis

[AKH+02] Andersen, S.V., W.B. Kleijn, R. Hagen, J. Linden, M.N. Murthi und J. Skoglund: *iLBC - a linear predictive coder with robustness to packet losses.* In: *Speech Coding, 2002, IEEE Workshop Proceedings*, Seiten 23–25, Oktober 2002.

[asn] *asn1c - Open Source ASN.1 Compiler.* `http://lionet.info/asn1c/`.

[Bec06] Becker, Philipp: *Entwicklung des MacZ Service Layers.* Diplomarbeit, TU Kaiserslautern, 2006.

[bel] *Homepage BelAmI-Projekt.* `http://www.belami-project.org/index.html`.

[BGK07] Becker, Phillip, Reinhard Gotzhein und Thomas Kuhn: *MacZ - A Quality-of-Service MAC Layer for Ad-hoc Networks.* In: *Proceedings of 7th Conference on Hybrid Intelligent Systems (HIS)*, 2007.

[Chr08] Christmann, Dennis: *Mikroprotokoll-basierte Restrukturierung, toolgestützte Dokumentation und Evaluation von MacZ.* Bachelorarbeit, TU Kaiserslautern, März 2008.

[COD] *CODEC Homepage.* `http://codec.sourceforge.net/`.

[Dia05] Dialog: *DA9030 Preliminary Product Data Sheet*, 2005. `http://enaweb.eng.yale.edu/drupal/system/files/DA9030-DS07-0501.pdf`.

[FG07] Fliege, I. und R. Gotzhein: *Automated Generation of Micro Protocol Descriptions from SDL Design Specifications.* In: Gaudin, Emmanuel, Elie Najm und Rick Reed (Herausgeber): *SDL 2007: Design for Dependable Systems, 13th International SDL Forum*, Seiten 150–165, 2007.

[FGJ+05] Fliege, Ingmar, Alexander Geraldy, Simon Jung, Thomas Kuhn, Christian Webel und Christian Weber: *Konzept und Struktur des SDL Environment Frameworks (SEnF).* Technischer Bericht 341/05, TU Kaiserslautern, 2005.

[FGW06] Fliege, Ingmar, Rüdiger Grammes und Christian Weber: *ConTraST - A Configurable SDL Transpiler And Runtime Environment.* In: Gotzhein, R. und R. Reed (Herausgeber): *System Analysis and Modeling: Language Profiles, Lecture Notes in Computer Science 4320*, Seiten 216–228. Springer, 2006.

[FK07] Fliege, Ingmar und Jan Koch: *AmICom - Formally specified service platform for ambient intelligence networks.* Technischer Bericht D2.6.1, Fraunhofer IESE, Juni 2007.

[FLA] *FLAC Documentation.* http://flac.sourceforge.net/documentation.html.

[Fli07] Fliege, Ingmar: *Documentation of micro protocols.* Technischer Bericht 358/07, TU Kaiserslautern, 2007.

[For] *Forschungsschwerpunkt Ambient Intelligence (AmI) an der TU Kaiserslautern.* http://www.eit.uni-kl.de/ami/frame.html?de_inhalte.

[Ger94] Gersho, Allen: *Advances in speech and audio compression.* In: *Proceedings of the IEEE*, Band 82, Seiten 900–918, 1994.

[Hyg06] Hygrosens Instruments: *Passiv Infrarot Bewegungsmelder Low Power*, August 2006. http://www2.produktinfo.conrad.com/datenblaetter/150000-174999/172526-as-01-de-PIR_SMD_MODUL_3_5V_80_UA.pdf.

[iLB04] *iLBC White Paper.* http://www.gipscorp.com/files/english/white_papers/iLBC.WP.pdf, 2004.

[Int06] Intel: *PXA27x Processor Family - Developer's Manual*, Januar 2006. http://enaweb.eng.yale.edu/drupal/system/files/PXA27x_Developers_Manual.pdf.

[ITU02a] ITU-T Study Group 17: *X.680 - Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.* International Telecommunication Union, 2002. http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.680-200207-I!!PDF-E&type=items.

[ITU02b] ITU-T Study Group 17: *X.690 - Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).* International Telecommunication Union, 2002. http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.690-200207-I!!PDF-E&type=items.

[ITU02c] ITU-T Study Group 17: *X.691 - Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).* International Telecommunication Union, 2002. http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.691-200207-I!!PDF-E&type=items.

[Jfl] *JFlashmm for Imote2 with Wiggler support.* http://www.michaelwhyte.net/iMote2/JFlashmm-Wiggler.zip.

[KF07] Koch, Jan und Ingmar Fliege: *AmICom - Middleware Support for Ambient Communication.* In: *2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI) at the 9th International Conference on Ubiquitous Computing (UbiComp)*, 2007.

[Lam] *LAME V3.98b Program Documentation.* http://lame.cvs.sourceforge.net/*checkout*/lame/lame/doc/html/index.html.

[Lar99] Larmouth, John: *ASN.1 Complete.* Morgan Kaufmann, 1999. ISBN 0-12-233435-3, http://www.oss.com/asn1/larmouth.html.

[Mic07a] Microchip: *MCP4921/4922 Datasheet*, 2007. http://ww1.microchip.com/downloads/en/DeviceDoc/21897B.pdf.

[Mic07b] MICROCHIP: *PIC18F4321 Family Data Sheet*, 2007. http://ww1.microchip.com/downloads/en/DeviceDoc/39689E.pdf.

[Mül90] MÜLLER, RUDOLF: *Rauschen*. Springer Berlin, 2., überarbeitete u. erweiterte Auflage, 1990. ISBN 3-540-51145-8.

[mp3] *Die MP3 Geschichte*. http://www.iis.fraunhofer.de/bf/amm/mp3history/mp3history01.jsp.

[ntc] *NTC Datenblatt*. http://www.produktinfo.conrad.com/datenblaetter/450000-474999/467243-da-01-de-Heissleiter_NTC_Typ_M87.pdf.

[ope] *openASN.1 - ASN.1-Toolkit für Java*. http://www.openasn1.org/.

[Pan95] PAN, DAVIS: *A tutorial on MPEG/audio compression*. In: *Multimedia, IEEE*, Band 2, Seiten 60–74, 1995.

[Sal07] SALOMON, DAVID: *Data Compression - The Complete Reference*. Springer London, 4. Auflage, 2007. ISBN 1-8462-8602-6.

[Sch07] SCHMELZER, THORSTEN: *Integration des Imote2 in das SDL Environment Framework (SEnF)*. Projektarbeit, TU Kaiserslautern, September 2007.

[SDL] *SDL Forum*. http://www.sdl-forum.org/SDL/index.htm.

[SER07] SIERRA, ALBERTO, PETER EBINGER und VOLKER ROTH: *CODEC Tutorial*, November 2007. http://codec.sourceforge.net/CodecTutorial.pdf.

[Sha05] SHAHABDEEN, JUNAITH AHEMED: *Boot Loader Architecture*, 2005. http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-1.x/contrib/imote2/tools/src/bootloader.doc?revision=1.1.

[Tan07] TANENBAUM, ANDREW S.: *Computernetzwerke*. Pearson Education, 4., überarbeitete Auflage, 2007. ISBN 3-8273-7046-9.

[TAO01] TAOS: *TSL250R, TSL251R, TSL252R, Light To Voltage Optical Sensors*, Mai 2001. http://www2.produktinfo.conrad.com/datenblaetter/150000-174999/163970-da-01-en-TSL250R.pdf.

[Tel] *Telelogic Tau SDL Suite*. http://www.telelogic.com/products/tau/sdl/index.cfm.

[Tex07] TEXAS INSTRUMENTS: *CC2420 Datasheet Rev SWRS041b*, März 2007. http://focus.ti.com/lit/ds/symlink/cc2420.pdf.

[Val07] VALIN, JEAN-MARC: *The Speex Codec Manual Version 1.2 Beta 3*, Dezember 2007. http://speex.org/docs/manual/speex-manual.pdf.

[Was04] WASABI: *Wasabi Software Developement Tools User's Guide for Intel XScale Microarchitecture*, März 2004. http://download.intel.com/design/intelxscale/dev_tools/031121/Wasabi_XScale_Users_Guide1.pdf.

[Web08] WEBEL, CHRISTIAN: *A Tailored Communication Framework Supporting Network Quality of Service*. Ph.D. thesis to be published, 2008.

[Wei91] Weiser, Mark: *The Computer for the Twenty-First Century.* Scientific American, Seiten 94–100, September 1991. www.stanford.edu/class/cs344a/papers/computer-for-21-century.pdf.

[Wig] *Wiggler Schematic.* http://wiki.openwrt.org/OpenWrtDocs/Customizing/Hardware/JTAG_Cable.