

Freie Universität



Berlin

Diplomarbeit

Autonomous Wildlife Monitoring

Design, implementation, and evaluation of a wireless solution for nightingale song recording

Frank Beier – beier@inf.fu-berlin.de
06.10.2009

Betreuer: Prof. Dr.-Ing. Jochen Schiller
Dipl.-Inform. Tomasz Naumowicz

Danksagung

Ich möchte Herrn Dipl.-Inform. Tomasz Naumowicz meinen Dank für seine stets motivierende und produktive Betreuung meiner Diplomarbeit aussprechen. Insbesondere möchte ich für die zahlreichen Stunden danken, in der wir über den Aufbau und die Umsetzung des entstandenen Systems diskutiert haben. Weiterer Dank geht an das Team der Biologen. Hier wären zu nennen (in alphabetischer Reihenfolge): Sarah Kiefer, Professor Dr. Silke Kipper, Kim Geraldine Mortega und Michael Weiss.

Ganz besonderer Dank geht an meine Eltern und meine Freundin, die mich während meines gesamten Studiums immer tatkräftig unterstützt haben.

Kurzfassung

Der Gesang der Nachtigall ist aktueller Forschungsgegenstand in der Verhaltensbiologie aufgrund des enormen Variantenreichtums und der Komplexität. Um diesen Gesang analysieren zu können werden qualitativ hochwertige Audioaufzeichnungen benötigt, die allerdings nur sehr aufwendig erstellt werden können. Üblich ist die manuelle Aufzeichnung vor Ort mit einem Audiorekorder.

In dieser Arbeit wird ein System vorgestellt, was diese Aufgabe vereinfacht und die benötigten Aufnahmen selbstständig erstellt. Um die Aufgabe umzusetzen wird dabei aus mehreren Klein notebooks (Netbooks) ein Mesh-Netzwerk erstellt, das Audioaufzeichnungen durchführt und über eine WLAN Schnittstelle gesteuert werden kann. Zur komfortablen Steuerung wird dem Benutzer des Systems eine Software mit grafischer Benutzeroberfläche zur Verfügung gestellt. Fertiggestellte Aufnahmen sollen mit hoher Geschwindigkeit vom Benutzer aus dem System heruntergeladen werden können. Da das System im freien Feld eingesetzt werden soll, muss es verschiedenen Witterungsbedingungen widerstehen und über eine lokale Stromversorgung mehrere Tage lauffähig sein.

Das System wurde umfangreich getestet. Die Zuverlässigkeit bezüglich korrekt erstellter Audioaufnahmen betrug 100%. Mit einer Autobatterie erreichte das System bei einem definierten 24-Zyklus eine Laufzeit ca. 3 Zyklen. Ein Zyklus umfasst dabei zwei gleichzeitige sechsstündige Aufnahmen, deren Komprimierung und Download. Die übrigen Stunden verbrachte das System vollautomatisch im Standby-Modus.

Die Downloadgeschwindigkeit ist stark von der Qualität der Funkverbindung abhängig. Unter idealen Voraussetzungen sind 2MB/s möglich. Werden die Daten nicht über eine Direktverbindung, sondern über das Mesh-Netzwerk über eine oder mehrere Zwischenstationen weitergeleitet, sinkt diese Geschwindigkeit beträchtlich. Bei einer Zwischenstation betrug diese ca. 170kB/s.

Inhalt

Inhalt	I
1 Einleitung	1
1.1 Anforderungsanalyse.....	3
1.2 Verwandte Arbeiten	6
1.2.1 Autonomous Monitoring of Vulnerable Habitats.....	6
1.2.2 MSB-430H: Implementierung eines Audiolinks.....	9
1.2.3 Automated wildlife monitoring using self-configuring sensor networks deployed in natural habitats	10
1.2.4 The ZebraNet Wildlife Tracker	11
2 Systementwurf	15
2.1 Plattform.....	15
2.1.1 ScatterWeb Plattform	15
2.1.2 Alix2c2	16
2.1.3 Asus Eee PC	18
2.2 Aufbau des Systems	20
2.2.1 Netbook-Komponente	21
2.2.2 Benutzer-Komponente.....	28
2.2.3 Erweiterungs-Komponente.....	28
3 Grundlagen	30
3.1 Windows Communication Foundation.....	30
3.2 DirectSound.....	31
3.3 Mesh Connectivity Layer	32
3.4 WavPack.....	33
4 Softwarearchitektur	34
4.1 Netbook-Komponente	34
4.1.1 XML-Dokumente	34
4.1.2 WCF-Service	43
4.1.3 Standby Service	45
4.1.4 Record-Service	51
4.1.5 Battery-Service	57

4.1.6	ARM-Service.....	57
4.1.7	FTP-Service.....	58
4.2	Benutzer-Komponente	59
4.2.1	Architektur.....	60
5	Evaluation	70
5.1	Leistungsaufnahme.....	70
5.1.1	Versuchsaufbau	70
5.1.2	Messwerte.....	70
5.2	Download Performance.....	71
5.2.1	Versuchsaufbau	71
5.2.2	Messwerte.....	71
5.3	WavPack Geschwindigkeit	72
5.4	WavPack Komprimierungsfaktor.....	72
5.5	Laufzeit.....	72
5.5.1	Versuchsaufbau	72
5.5.2	Theoretische Überlegungen.....	73
5.5.3	Messwerte.....	73
5.6	Zuverlässigkeit	74
6	Zusammenfassung	75
6.1	Zukünftige Arbeiten	77
6.1.1	Verbesserungen	77
6.1.2	Erweiterungen.....	77
7	Anhang	79
7.1	WCF Service API.....	79
7.1.1	Binding	79
7.1.2	Operations Contracts	79
7.1.3	Data Contracts	81
7.2	Management Console API	83
7.2.1	ClientBackend Kontruktor.....	83
7.2.2	OnKeepAliveTimer	83
7.2.3	saveSchedule	83
7.2.4	saveMaxID	83
7.2.5	loadSchedule.....	83

7.2.6	loadMaxID.....	83
7.2.7	saveServiceTime.....	84
7.2.8	loadServiceTime.....	84
7.2.9	getSchedule.....	84
7.2.10	updateSchedule.....	84
7.2.11	GetMicroName.....	84
7.2.12	DeleteTimer.....	85
7.2.13	SendTimer.....	85
7.2.14	sendSchedule.....	85
7.2.15	GetStatus.....	85
7.2.16	Getmaxid.....	86
7.2.17	Scan.....	86
7.2.18	checkIP.....	86
7.2.19	OnServiceFound.....	86
7.2.20	Download.....	86
7.2.21	ftp_DownloadProgressChanged.....	87
7.2.22	ftp_DownloadFileCompleted.....	87
7.2.23	OnDownloadStatusUpdated.....	87
7.2.24	OnDownloadComplete.....	87
7.2.25	OnDownloadCheck.....	87
7.2.26	Download2.....	87
7.2.27	getDownloadList.....	88
7.2.28	refreshDownloadList.....	88
7.2.29	DeleteDownload.....	88
7.2.30	SetStandby.....	88
7.2.31	WriteMicXML.....	88
7.2.32	ReadMicXML.....	88
7.2.33	RenameMic.....	89
7.2.34	getMicros.....	89
7.2.35	Decompress.....	89
7.2.36	restoreMicName.....	89
8	Literaturverzeichnis.....	90

1 Einleitung

Die Nachtigall (*Luscinia megarhynchos*) ist eine Vogelart aus der Ordnung der Sperlingsvögel (Passeriformes) und ist in Asien, Europa und Nordafrika heimisch. Durch ihren Gesang hat sie die Menschen immer wieder auf sich aufmerksam gemacht, so dass sie in der Literatur sehr häufig erwähnt wird (z.B. William Shakespeare: „Romeo and Juliet“).

Ab April beginnen die Männchen ab ca. 23 Uhr bis zur Morgendämmerung mit ihrem Gesang um eine Brutpartnerin anzulocken; sind sie dabei erfolgreich, stellen sie ihren Gesang ein, so dass ab Mitte Mai nur noch unverpaarte Nachtigallenmännchen zu hören sind. Während der Brutzeit singen die Männchen auch tagsüber, was u.a. zur Revierverteidigung dient.

Dabei beherrschen die Nachtigallen über 200 Strophenotypen mit denen sie in einer einzigen Nacht über 1000 Strophen produzieren können. Den Gesang, der eine Reihe von Einzel- und Doppeltönen darstellt, lernen die Jungvögel von erwachsenen Tieren. Aufgrund des enormen Variantenreichtums und der Komplexität ist der Gesang Inhalt verschiedener Fragestellungen, u. a. zur Funktion des Gedächtnisses, im Bereich der Verhaltensbiologie.

Die Arbeitsgruppe von Professor Dr. Silke Kipper (Institut für Biologie, Lehrstuhl für Verhaltensbiologie, Freie Universität Berlin) untersucht in verschiedenen Projekten die Lernstrategien und den adaptiven Wert des Gesangs von Nachtigallen [1].

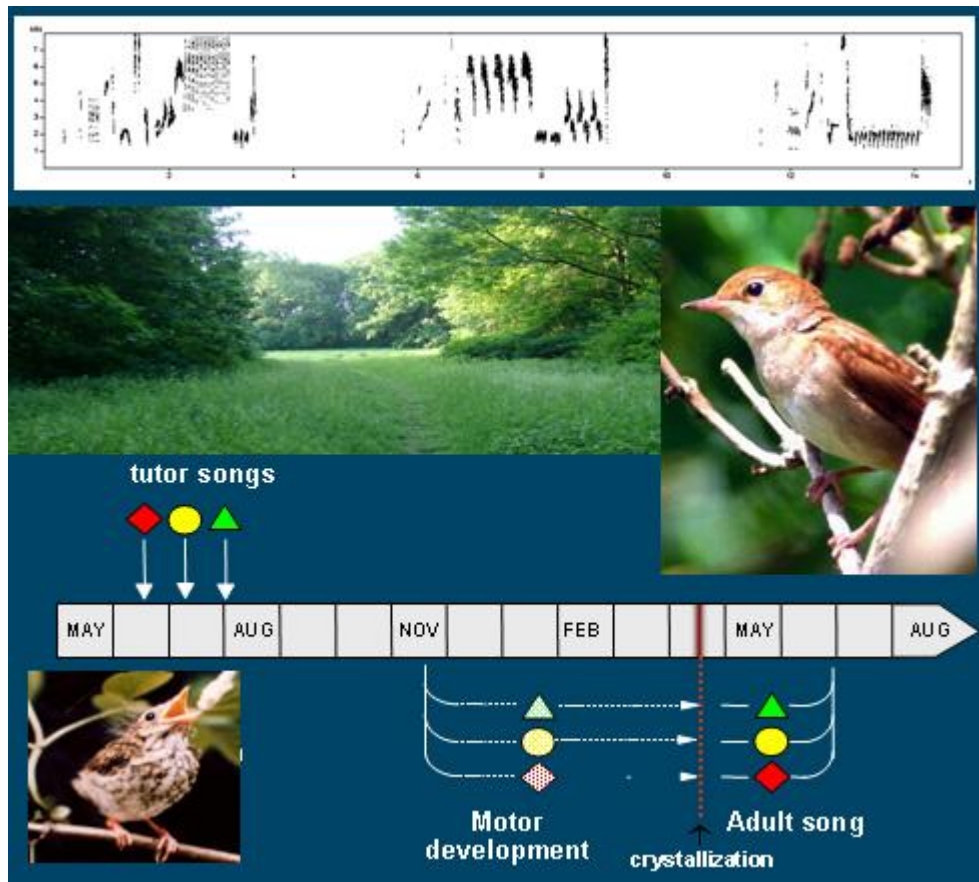


Abbildung 1: Lernstrategien und adaptiver Wert des Gesangs bei Nachtigallen [1]

Um den Gesang zu analysieren, müssen vor Ort an den Nistplätzen Audioaufnahmen des Gesangs angefertigt werden. An diesem Punkt setzt diese Diplomarbeit an. Ziel ist es ein System zu entwickeln, mit dem man zuverlässig diese Audioaufzeichnungen erstellen kann. Daraus ergeben sich eine Vielzahl von Anforderungen, die im folgenden Kapitel näher beschrieben werden.

1.1 Anforderungsanalyse

Die Anforderungen wurden zusammen mit der Arbeitsgruppe von Professor Dr. Silke Kipper analysiert. Die folgende Tabelle zeigt das Ergebnis der Analyse.

Anforderung	Beschreibung
Erstellung von Audioaufnahmen	Primäre Anforderung an das System ist die Erstellung von digitalen Audioaufnahmen.
Qualitätskriterien	Die Audioaufnahmen sollen eine Auflösung von 16bit und eine Samplingrate von 44100Hz besitzen. Das gewählte Audioformat muss eine verlustfreie Weiterverarbeitung gewährleisten.
Skalierbarkeit der Aufnahmegeräte	Das System soll mit einer hohen Anzahl von Aufnahmegeräten betrieben werden können und trotzdem performant bleiben. Geplant ist ein Ausbau bis auf 100 Aufnahmegeräte.
Komfortabel in der Bedienung	Der Benutzer soll die Möglichkeit haben, das System einfach und komfortabel bedienen zu können. Eine geeignete Software zur Verwaltung wird dafür benötigt, welche auf einem gängigen Notebook mit Windows-Betriebssystem lauffähig ist.

Zuverlässigkeit	Das System soll eine hohe Zuverlässigkeit in Bezug auf korrekt erstellte Audioaufnahmen aufweisen.
Erweiterbarkeit	Das System soll für zukünftige Erweiterungen, wie z.B. Videoaufzeichnungen, offen sein. Eine Erweiterung soll durch Hinzufügen von neuen Komponenten problemlos möglich sein ohne das bestehende System ändern zu müssen.
Geringer Preis pro Aufnahmegerät	Der Preis pro Aufnahmegerät soll gering gehalten werden.
Nachkaufbare Hardware	Die benötigte Hardware soll auch in einigen Jahren noch auf dem Markt verfügbar sein. Da die Hardwareentwicklung stetig voranschreitet, soll das System aus Standardkomponenten erstellt werden. Es muss gewährleistet werden, dass die erstellte Software ohne Einschränkungen auf kompatibler Hardware lauffähig ist.
Robustheit im Feld	Die Audioaufnahmen sollen im freien Feld erstellt werden. Dazu ist es nötig, dass das System ausreichend gegen Umwelteinflüsse geschützt ist.
Laufzeit	Das System soll im freien Feld ohne Infrastruktur lauffähig sein. Es muss

	<p>also über eine eigene Stromversorgung in Form einer Batterie verfügen. Diese soll eine Laufzeit von mehreren Tagen haben.</p>
<p>Drahtloses System</p>	<p>Verkabelung im freien Feld ist aufwendig und nicht immer möglich. Daher soll das System auf Basis eines drahtlosen Netzwerks konstruiert werden. Darüber hinaus soll der Benutzer ebenfalls drahtlosen Zugang zu diesem Netzwerk erhalten.</p>
<p>Downloadgeschwindigkeit</p>	<p>Die Übertragung der fertiggestellten Audioaufnahmen vom System auf den Rechner des Benutzers soll mit einer angemessenen Geschwindigkeit realisierbar sein. Für eine Aufnahme von beispielsweise einer Stunde sollte der Benutzer nicht länger als einige Minuten Zeit benötigen.</p>
<p>Autonomer Betrieb</p>	<p>Das System soll an den Nistplätzen der Nachtigallen installiert und eingeschaltet werden. Danach soll das System ohne Eingriff autonom lauffähig sein. Sollte es zu Fehlern im Programmablauf kommen, soll das System dies selbstständig korrigieren.</p>

1.2 Verwandte Arbeiten

1.2.1 Autonomous Monitoring of Vulnerable Habitats

Autonomous Monitoring of Vulnerable Habitats [2] ist ein Projekt von Microsoft Research in Kooperation mit der Oxford University und der Freien Universität Berlin. Ziel des Projekts ist es Daten über das Verhalten des Schwarzschnabel-Sturmtaucher (*Puffinus puffinus*), der einen großen Teil seines Lebens auf dem Meer verbringt und in Erdhöhlen nistet, auf Skomer Island (UK) zu sammeln.

Bisher haben die Forscher das Verhalten der Seevögel mit Hilfe von Miniatur-GPS-Loggern analysiert, allerdings sind dazu manuelle Inspektionen der Erdhöhlen alle 20-30 min nötig um die bereits erfassten Vögel wieder einzufangen. Da diese Technik sich aber als eher unpraktikabel erwiesen hat, wird nun ein drahtloses Sensornetzwerk (Wireless Sensor Network, WSN) eingesetzt.

Damit können die markierten Vögel identifiziert und deren Ankunft und Abflug fast augenblicklich festgestellt werden, die Überwachung einer größeren Menge an Vögeln ist somit möglich. Zudem werden Umweltdaten zu Temperatur und Luftfeuchtigkeit inner- und außerhalb der Erdhöhlen ausgegeben. Dieses integrierte System ermöglicht die sofortige Rückmeldung von dem Netzwerk, redundante Aufnahmen von Daten und die Übermittlung der Daten zum Server auf dem Festland für spätere Analysen und Weiterverarbeitung.

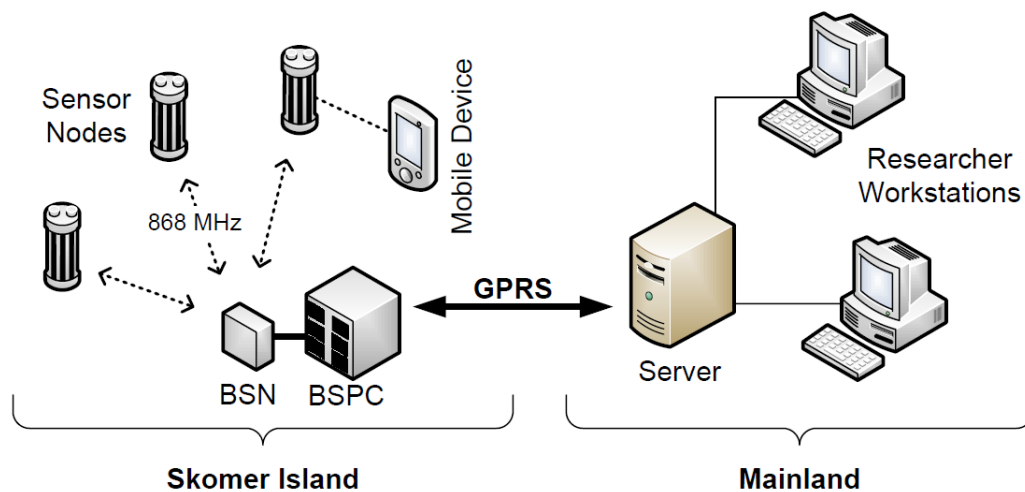


Abbildung 2: Systemarchitektur Autonomous Monitoring of Vulnerable Habitats [2]

Das System, welches im Deployment 2007 eingesetzt wurde, besteht aus mehreren Komponenten (Abbildung 2). Zum einen werden Sensorknoten (Abbildung 3) an den Nestern angebracht, welche mittels RFID anwesende Vögel identifizieren. Die Sensorknoten bestehen aus einem MSB430 Board, das mit einem Mikrocontroller des Typs TI MSP430F1612, Sensirion SHT11 Temperatur- und Feuchtigkeitssensoren und einem Chipcon CC1020 Funktransceiver ausgestattet ist. Zusätzlich wird ein Erweiterungsboard mit einem RFID-Leser und zwei passiven Infrarotsensoren verwendet. Die erfassten Daten werden per Funk an eine Basisstation übermittelt.

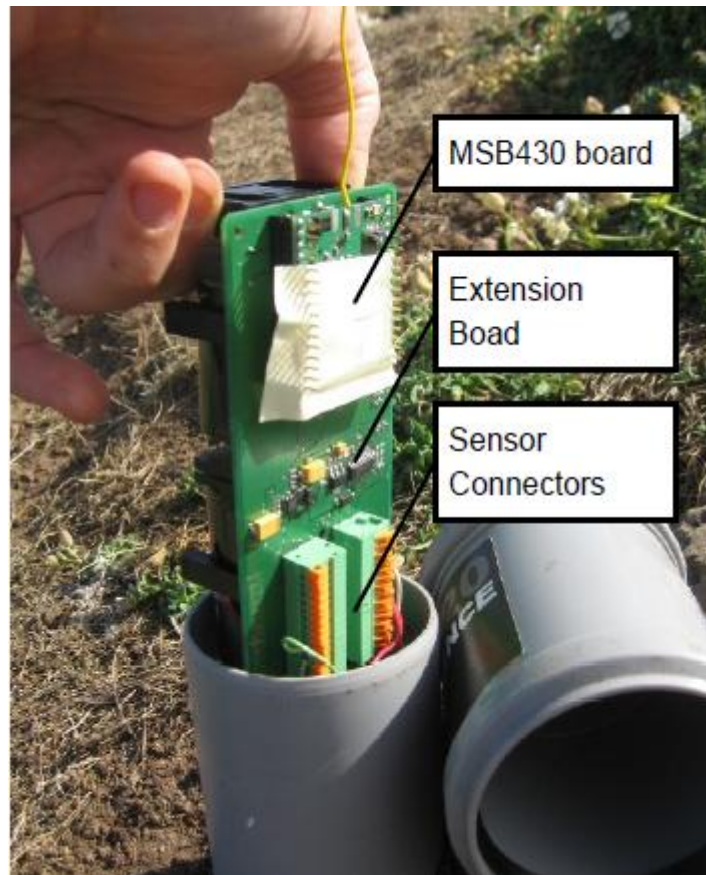


Abbildung 3: Sensorknoten [2]

Die Basisstation bestand aus einem Sensorknoten (BSN) und einem Computer auf Basis eines EPIA VIA Boards (BSPC), als Betriebssystem wird Windows XP eingesetzt. Die ankommenden Daten werden vom BSN empfangen und an den BSPC weitergeleitet und dort gespeichert. Einmal täglich werden alle gespeicherten Daten über eine GPRS Funkverbindung an einen Server auf dem Festland übermittelt, welcher diese Daten zur Weiterverarbeitung den Wissenschaftlern bereitstellt.

Weiterhin werden den Wissenschaftlern vor Ort mobile Geräte auf Basis von Windows Mobile zur Verfügung gestellt, mit denen sie aktuelle Daten direkt von den Sensorknoten empfangen können.

Das Projekt läuft seit 2007 und wird stetig weiterentwickelt. Publikationen zu Ergebnissen wurden bisher nicht veröffentlicht.

1.2.2 MSB-430H: Implementierung eines Audiolinks

In der Studienarbeit von Frank Beier und Marco Ziegert [3] wird eine Audioübertragung auf Basis eines MSB-430H Sensorknotens realisiert. Dabei werden die Audiodaten über den AD-Wandler eines Sensorknotens gemessen und direkt über die vorhandene Funkschnittstelle an einen Empfängersensorknoten gesendet. Der Empfänger gibt diese Daten in Echtzeit an seinem integrierten DA-Wandler aus. Optional können die Audiodaten mit dem verlustbehafteten Komprimierungsverfahren ADPCM en- und decodiert werden.

Ohne Komprimierung werden Abtastraten von 16kHz bei einer Auflösung von 8bit, mit ADPCM-Komprimierung 8kHz bei 12bit erreicht. Die Auflösung ist dabei auf maximal 12bit durch den integrierten AD-Wandler beschränkt.

Die Variante ohne Komprimierung wird von der Geschwindigkeit der Funkverbindung beschränkt. Wird hier die Abtastrate oder die Auflösung erhöht, so können die Daten nicht schnell genug verschickt und empfangen werden, was erst zu Verzögerungen in der Übertragung und schließlich zu Sprüngen durch Überlauf des Datenpuffers führt.

Die Variante mit Komprimierung wird hingegen von der Geschwindigkeit des Mikrocontrollers limitiert. Eine höhere Abtastrate ist nicht möglich, da die Komprimierung bei höheren Abtastraten nicht in Echtzeit ausgeführt werden kann.

Durch die beschränkte Abtastrate und Auflösung des Audiosignals ist das System nur zur Übertragung von niedrig-frequenten Audiodaten wie z.B. menschliche Sprache geeignet.

1.2.3 Automated wildlife monitoring using self-configuring sensor networks deployed in natural habitats

Das Ziel dieses Projekts der University of California ist es bestimmte Tierstimmen aus einer geräuschvollen Umgebung zu filtern und damit deren Aufenthaltsort in freier Wildbahn zu ermitteln.

Das System baut auf einem drahtlosen Netzwerk aus Sensorknoten auf, welche jeweils mit vier Kondensator-Mikrofonen ausgestattet sind. Als Betriebssystem wird Linux eingesetzt. Die Sensorknoten, Acoustic Embedded Networked Sensing Boxes (Acoustic ENSBox) genannt, kommunizieren über den WLAN Standard 802.11b. Abbildung 1 zeigt einen solchen Sensorknoten. Die Acoustic ENSBox verfügt über Datenverarbeitungs-, Speicher- und Netzwerkmöglichkeiten um Audiodaten in Echtzeit zu verarbeiten und um verteilte Algorithmen zu implementieren, einschließlich einer hochpräzisen selbstkalibrierten Lokalisation und Orientierung.



Abbildung 4: Acoustic Embedded Networked Sensing Box [4]

Die Sensorknoten bestimmen selbstständig ihre relative Lage zu den anderen Sensorknoten durch Messung von Richtung und Zeitunterschieden von akustischen Signalen (Time-of-Flight). Um die Tierstimmen auch bei Hintergrundgeräuschen besser erkennen zu können, wird eine Technik auf Basis der Constant False Alarm Rate (CFAR) eingesetzt. Damit war es möglich sieben verschiedene Vogelgesänge in einer Aufnahme aus dem Regenwald im Monte Azules Biosphäre Naturschutzgebiet in Mexiko nachzuweisen. Mit einer modifizierten Version dieses Algorithmus konnten auch Warnrufe von Gelbbäuchigen Murmelieren in Echtzeit nachgewiesen werden.

Die Lokalisierung der einzelnen Vögel basiert auf einem Algorithmus von Kung Yao et al. [5]. Er ermöglicht es in Echtzeit mehrere Quellen verfolgen zu können.

Über Laufzeit und Energieversorgung der Sensorknoten werden keine Angaben gemacht. Der Fokus der Arbeit liegt mehr auf den Erkennungsalgorithmen um Tierstimmen korrekt erkennen und zuzuordnen zu können.

1.2.4 The ZebraNet Wildlife Tracker

Die Princeton University (USA) entwickelte 2002 im Rahmen des ZebraNet-Projekts [6] ein System zur Überwachung von Zebras, mit welchem herausgefunden werden sollte, wie, warum und wann Zebras ihre Wanderungen unternehmen. Dazu sollten Halsbänder mit Sensorknoten, die als Peer-to-Peer-Netzwerk agieren und Daten zur Überwachung des Standortes an die Forscher übertragen, an den Zebras angebracht werden.

Daraus ergaben sich folgende Ziele für das Projekt:

- Die Position der Zebras sollte alle drei Minuten per GPS bestimmt und aufgezeichnet werden.

- Jede Stunde sollten drei Minuten lang die Aktivitäten der Zebras detailliert (Position, Temperatur, Wetter, Umwelt, Körperbewegungen) aufgezeichnet werden.
- Es war geplant, dass das System mindestens 1 Jahr lang ohne Einfluss der Forscher läuft.
- Die Datenübertragung musste über ein großes Gebiet über mehrere hundert oder tausend km² möglich sein. Das System sollte im Mpala Forschungszentrum in Zentralkenia aufgestellt werden.
- Das System sollte ohne feste Basisstationen, Antennen oder Mobilfunknetze agieren und zuverlässig die Daten übertragen.
- Es musste ein Halsband-Design entwickelt werden, was gut an die Zebras angepasst ist und nicht schwerer wie 3-5 lbs ist



Abbildung 5: Zebra mit ZebraNet Halsband [7]

Daraus ergab sich das Hauptziel, dass eine große Datenmenge, die über Monate und Jahre aufgenommen wurde, zu den Forscher zurück geliefert werden muss. Dazu musste das System energieeffizient arbeiten, die Daten adäquat aufzeichnen und speichern und robust sein um seine Zuverlässigkeit unter den rauen Bedingungen der Wildnis zu garantieren.

Die Knoten beinhalteten ein Low-Power Miniatur-GPS-System mit einer programmierbaren CPU, einen nicht-flüchtigen Speicher für die Datenaufzeichnung und einen Funk-Sendeempfangsgerät um mit den anderen Knoten oder der Basisstation zu kommunizieren. Eine der Besonderheiten in diesem Projekt war, dass die Basisstation mobil sein sollte, so dass die Forscher beim Durchfahren oder Überfliegen der Region die Daten übertragen konnten.

Ein wichtiger Aspekt war auch, dass die Daten auch über andere Knoten weitergeleitet und Positionsaufzeichnungen von anderen Knoten gespeichert werden konnten, da nicht alle Knoten immer in Reichweite der Basisstation sind.

Der Prototyp bestand aus einem Board des Typs GPS-MS1E, als CPU wurde ein 20MHz Hitachi SH1 32-bit Mikroprozessor eingesetzt. Desweiteren war ein GPS-Modul und sowohl ein Kurz- (hauptsächlich für die Kommunikation mit anderen Knoten) als auch ein Langreichweitenfunkmodul (für die Kommunikation mit der Basisstation) vorhanden. Die Energieversorgung wurde mit einem Solarmodul mit einer Leistungsabgabe von 5W, gepuffert von 14 Lithium-Ionen Zellen, gewährleistet. Die maximale Leistungsaufnahme betrug 1622mA bei 3,6V.

Die Forschergruppe der Princeton Universität stand dabei vor dem Problem, dass sie nicht im Voraus wissen konnten, wie sich die Zebras bewegen werden, was die Entwicklung eines optimalen Routing-Protokolls erschwerte. Sie verwendeten dazu bisher gesammelte Daten, welche von Wissenschaftlern durch Beobachtung vor Ort zusammengetragen wurden, und entwarfen eine Simulation der Zebrawanderungen und der Umgebung (ZNetSim).

Dabei wurden drei verschiedenen Protokolle verglichen: zwei Peer-to-Peer Protokolle (Flooding and History Based) und eines, welches keine Peer-to-Peer Transfers erlaubt, sondern nur die Übertragung zwischen Knoten und Basisstation. Mit beiden Peer-to-Peer-Protokollen konnte in einer Funkreichweite von 6 Km eine 100% Erfolgsrate bei der Übertragung erreicht werden, verglichen mit der direkten Übertragung bei 11 km Reichweite. Allerdings hat das Flooding Protokoll eine bessere Performance in Peer-to-Peer-Netzwerken ohne Beschränkungen bei Speicher und Bandbreite. Der Vorteil des History-Based-Protokolls ist, dass es intelligenter die Knoten auswählt, mit denen es Daten austauscht und daher mehr brauchbare Daten zur Basis sendet. Bei größeren Funkreichweiten Jedoch verbraucht dieses Protokoll sehr viel Energie und auch die Bandbreitenanforderungen sind extrem hoch, zudem werden bei diesem Protokoll die Daten nur zu einem Empfangsgerät gesendet.

Aus diesen Gründen wurde zunächst für das ZebraNet ein Flooding-Protokoll für kurze Reichweiten und ein direktes Protokoll für größere Reichweiten eingesetzt, mit denen man eine Erfolgsrate von 83% erreicht.

Im Januar 2004 fand das erste Deployment mit einem Prototyp statt. Ergebnisse dazu sind zu finden bei [8].

2 Systementwurf

2.1 Plattform

Es wurden mehrere Hard- und Softwareplattformen evaluiert um die Anforderungen, die in Kapitel 1.1 bereits beschrieben wurden, bestmöglich erfüllen zu können. Diese werden im folgenden Kapitel vorgestellt.

2.1.1 ScatterWeb Plattform

Die ScatterWeb Plattform wurde bereits in [3] zur Übertragung von Audiosignalen genutzt. Auf Basis des MSP430F1612 von Texas Instruments [9] bietet sie eine stromsparende Plattform für verteilte Anwendungen mit geringem Rechenaufwand. Ebenso ist ein Funkmodul für eine drahtlose Kommunikation zwischen ScatterWeb Sensorknoten und anderen Systemen bereits vorhanden. Nähere Informationen zur ScatterWeb Plattform sind verfügbar bei [10].

Beier und Ziegert zeigen in [3] zeigt jedoch, dass die Bandbreite der Funkanbindung der einzelnen Sensorknoten maximal für eine Abtastfrequenz von 16kHz bei einer Auflösung von 8-Bit ohne Komprimierungsverfahren ausreicht. Verwendet man ein einfaches Komprimierungsverfahren (z.B. ADPCM) um die Datenmenge zu reduzieren, so steigt die Rechenlast und limitiert wiederum die Abtastfrequenz. Eine Zwischenspeicherung von Aufnahmen und späteres Versenden mit einer langsameren Geschwindigkeit ist nicht möglich, da der interne Speicher der Sensorknoten mit 60 KB zu gering ist.

Aufgrund der geringen Abtastrate und der damit verbundenen, schlechten Audioqualität der Aufnahmen wurde diese Plattform als ungeeignet eingestuft und verworfen.



Abbildung 6: MSB-430 Sensorknoten [10]

2.1.2 Alix2c2

Auf der Suche nach einer leistungsfähigeren Plattform wurde ein Embedded PC auf Basis eines Alix2c2 Boards von PC Engines [11] evaluiert. Diese Boards werden am Fachbereich Informatik der Freien Universität Berlin im Projekt DES-Testbed [12] eingesetzt und standen zum Test zur Verfügung.

Das Board verfügt über eine 500MHz AMD Geode LX800 CPU, 256MB DRAM, einem CompactFlash Steckplatz, zwei miniPCI Steckplätze, zwei Ethernet Schnittstellen und einem dual USB 2.0 Port.

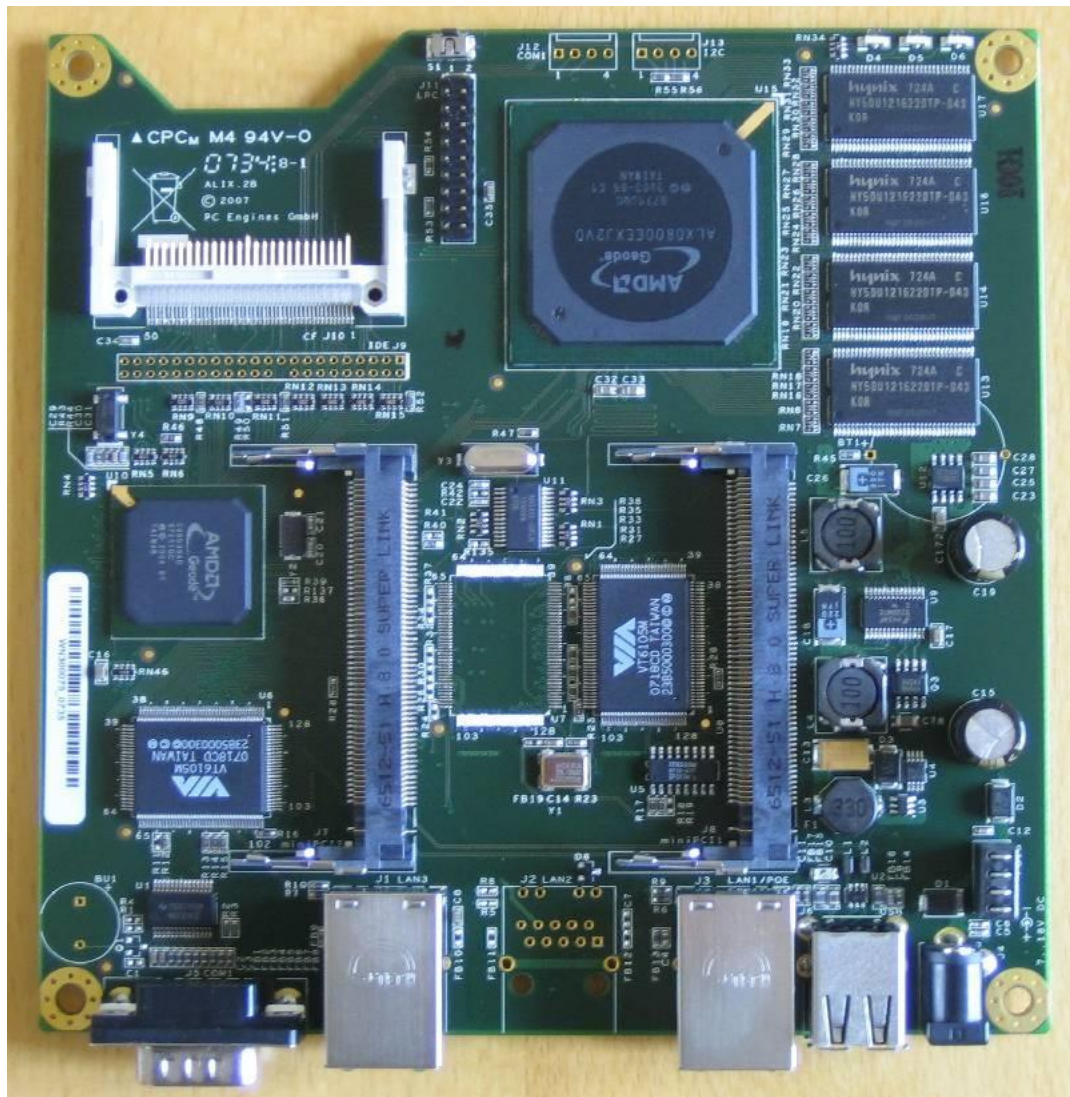


Abbildung 7: Alix 2c2 [11]

Um Audioaufzeichnungen mit diesem Board anfertigen zu können wird zusätzlich eine oder mehrere Soundkarten benötigt. Diese sind in vielen Varianten als USB Soundkarten und vereinzelt als miniPCI Soundkarten verfügbar. Für die drahtlose Vernetzung können Wireless LAN Adapter auf miniPCI oder USB Basis verwendet werden. Aufnahmen können auf einer CompactFlash Speicherkarte zwischengespeichert werden.

Als Betriebssystem standen Windows CE und Linux zur Wahl. Linux wurde evaluiert und aufgrund fehlender Kenntnisse und der damit verbundenen aufwendigen Einarbeitungszeit verworfen. Die Evaluation von Windows CE ergab, dass spezielle Treiber zum Betrieb von Soundkarten benötigt werden, die allerdings nicht verfügbar waren. Eine Entwicklung der benötigten Treiber war aus Zeitgründen nicht möglich. Daher wurde auch diese Plattform verworfen.

2.1.3 Asus Eee PC

Als dritte Plattform wurde ein Netbook der Marke Asus Eee PC 701 4G evaluiert. Es verfügt über eine Celeron-M 353 900MHz ULV CPU, 512MB RAM, 4GB Solid State Disk (SSD), interne Soundkarte, drei USB 2.0 Ports und ein integriertes WLAN Modul (802.11b/g). Als Betriebssystem wird Linux mitgeliefert. Aus den bereits oben genannten Gründen bezüglich der Einarbeitungszeit in Linux wurde allerdings nachträglich Windows XP installiert, für welches eine Vielzahl an Treibern für alle gängigen Hardwarekomponenten wie zum Beispiel USB-Soundkarten zur Verfügung stehen.



Abbildung 8: Asus Eee PC 701 4G

Das Netbook ist keine speziell angepasste Hardware, sondern besteht aus Standardkomponenten, was mehrere Vorteile in sich birgt:

Der Preis ist gering trotz sehr guter Ausstattung. Er lag zu Beginn dieser Arbeit bei 179€ im örtlichen Fachhandel.

Da Windows XP als Betriebssystem verwendet wird, ist die entwickelte Software generell auch auf anderen Computern lauffähig, sofern auf diesen Windows XP lauffähig ist. Dadurch kann ohne Anpassung der Software in Zukunft modernere Hardware verwendet werden, was z.B. zu Kostenersparnissen in der Anschaffung, höherer Leistungsfähigkeit oder geringerem Energieverbrauch führen kann.

Es können über den USB Anschluss weitere Geräteklassen wie z.B. Webcams für zukünftige Anwendungen angeschlossen werden, ohne dass für diese Geräte Treiber entwickelt werden müssen, wenn es sich um Standardkomponenten handelt.

Ferner hat das System bereits einen internen Akku zur Stromversorgung und eine entsprechende Ladeelektronik. Debuginformationen können über das vorhandene Display ausgegeben und Betriebssystem sowie Daten auf der internen SSD gespeichert werden. Daher wurde der Asus Eee PC als Plattform ausgewählt.

2.2 Aufbau des Systems

Das System auf Basis der Asus Eee PCs wurde nun folgendermaßen entworfen um die Anforderungen möglichst gut zu erfüllen.

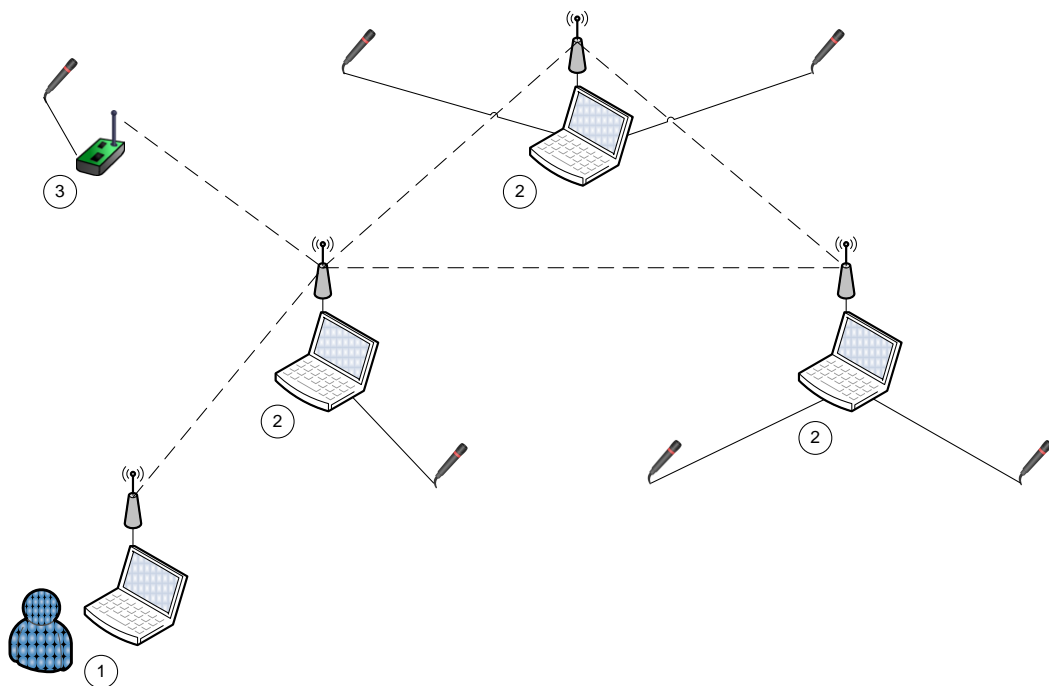


Abbildung 9: Systemaufbau

1: Benutzer-Komponente, 2: Netbook-Komponente, 3: Mikrocontroller-Komponente

Die Netbooks bilden ein Mesh-Netzwerk, welches Audioaufnahmen durchführt (siehe Abbildung 9, Komponente 2). Ein Mesh-Netzwerk bietet den Vorteil, dass anders als bei einem klassischen Ad-Hoc Netzwerk Verbindungen nicht nur zwischen Komponenten in direkter Funkreichweite möglich sind, sondern zusätzlich die Möglichkeit besteht, dass Daten von anderen Komponenten im Netzwerk weitergeleitet werden können. Dadurch sind Verbindungen zwischen beliebigen Teilnehmern im Mesh-Netzwerk möglich, sofern das Netzwerk zusammenhängend ist.

Das System kann zusätzlich von mehreren externen Erweiterungskomponenten ergänzt werden, die direkt mit jeweils einem Netbook über WLAN kommunizieren und ebenfalls Aufnahmen durchführen (siehe Abbildung 9, Komponente 3). Diese Erweiterungskomponenten werden im Rahmen der Diplomarbeit von Marco Ziegert entwickelt [13]. Gesteuert werden alle Komponenten durch eine Software auf einem weiteren Computer, der über WLAN mit dem Mesh-Netzwerk kommuniziert (siehe Abbildung 9, Komponente 1). Netbooks aus dem Mesh-Netzwerk können dabei direkt angesprochen werden, falls sie in Funkreichweite sind. Andernfalls werden Anfragen über andere Netbooks aus dem Mesh-Netzwerk an das gewünschte Netbook weitergeleitet. Die Mikrocontroller werden indirekt über das Mesh-Netzwerk angesteuert. Daten werden hierfür nur in das Mesh-Netzwerk gesendet und die Mikrocontroller verbinden sich selbständig mit einem beliebigen Netbook um die für sie bestimmten Daten zu erhalten. Diese Unterscheidung in der Kommunikation intern im System soll für den Benutzer vollständig transparent verlaufen.

Die Aufgaben der einzelnen Komponenten und deren Soft- und Hardwarebasis werden im folgendem genauer erörtert.

2.2.1 Netbook-Komponente

2.2.1.1 Aufgaben

Aufgabe der Netbook-Komponente ist es einen Zeitplan für Audioaufnahmen an angeschlossenen Aufnahmegegeräten zu erstellen, diese Audioaufnahmen zur gewünschten Zeit durchzuführen und sie schließlich als komprimierte Datei zum Download bereitzustellen. Dazu werden mehrere Windows Services implementiert, welche über XML-Dateien gekoppelt sind.

Die Schnittstelle zur Benutzer-Komponente, mit welcher der Benutzer das System steuern kann, wird durch einen Windows Communication Foundation (WCF) Service realisiert.

Audioaufnahmen auf lokal verfügbaren Aufnahmegegeräten zu erstellen und als Download bereitzustellen ist Aufgabe des Record Service. Der Zugriff auf Aufnahmegegeräte erfolgt dabei über DirectSound, der Download über einen FTP-Server.

Für das Energiemanagement ist der Standby Service zuständig. Er sorgt dafür, dass die Netbooks automatisch in den energiesparenden Standby-Modus versetzt und aus diesem in den normalen Betriebsmodus erweckt werden.

Der Battery Service liest über ein externes USB-Voltmeter die aktuelle Spannung der Energieversorgung. Dadurch kann der Benutzer rechtzeitig erkennen, wenn diese ausgetauscht werden muss.

Es besteht die Möglichkeit das System um weitere Services zu erweitern. Ein Beispiel dafür ist die Erweiterung von Marco Ziegert [13].

Abbildung 10 zeigt eine Übersicht über die Software Komponenten.

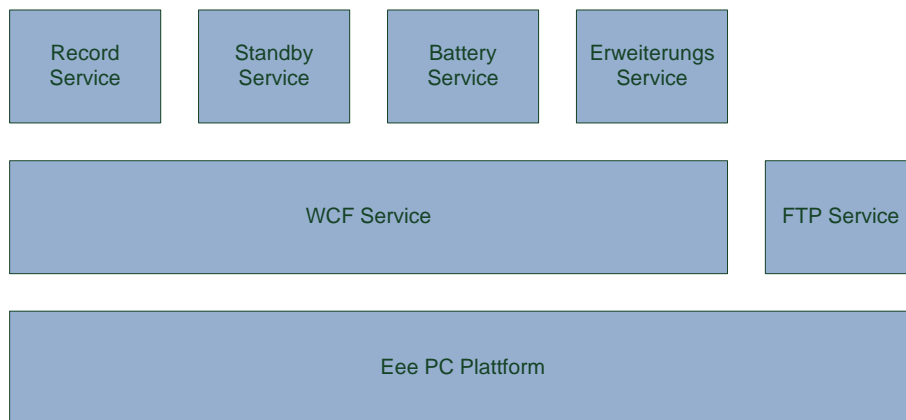


Abbildung 10: Software Komponenten der Netbook-Komponente

2.2.1.2 Hardware

Als Hardware werden beim Prototyp mehrere Netbooks vom Typ Asus Eee PC 701 4G verwendet. Funkverbindungen können über den eingebauten WLAN Adapter realisiert werden. Die vorhandene interne Soundkarte des Netbooks wird hierbei als Aufnahmegerät verwendet. Zusätzlich kann es durch mehrere USB-Soundkarten erweitert werden, beispielsweise vom Typ SpeedLink SL-8850, welche sich als geeignet erwiesen haben. Da die maximale Kabellänge bei einer USB Verbindung bei 5m liegt, kann ein USB-Extender verwendet werden um diese Beschränkung zu umgehen. Getestet wurde ein Digitus DA-70139-1, welcher aus zwei Adaptern besteht, die durch ein Ethernetkabel verbunden werden. Laut Hersteller beträgt die maximale Kabellänge hier 60m. Getestet wurde eine Länge von 10m, hier war ein problemloser Betrieb möglich. Eine dedizierte Stromversorgung ist dabei nicht nötig.



Abbildung 11: USB Soundkarte SpeedLink SL-8850

Das USB-Voltmeter, welches für den Battery Service nötig ist um Spannungswerte der externen Energieversorgung liefern zu können, konnte aufgrund von Liefer-schwierigkeiten nicht beschafft werden. Dieser Teil der Komponente ist daher nicht funktionsfähig, der Service liefert stets eine Spannung von 12V.

Als Energieversorgung werden Starterbatterien verwendet, wie sie in gängigen Au-tomobilen eingesetzt werden. Die Spannungswandlung übernimmt dabei ein KFZ-Netzteil, welches im Fachhandel für viele Netbooktypen erhältlich ist.



Abbildung 12: KFZ-Netzteil, Autobatterie und Anschlussadapter

Um das Netbook vor Wettereinflüssen zu schützen, wurde ein wasserdichtes Gehäuse gesucht. Als erstes Gehäuse wurde eine Explorer Koffer (Modell 10) der Firma Hofbauer evaluiert. Er lässt sich luft- und wasserdicht verschließen und bietet genügend Platz für einen Asus Eee PC. Allerdings ist hierbei die Kühlung des Systems problematisch, da es bei der Leistungsaufnahme des Asus Eee PC zu einer beträchtlichen Wärmeabgabe kommt. Dieses Gehäuse wurde daher verworfen.

Es wurde eine Anfrage an die Feinwerktechnik des Fachbereichs Physik an der Freien Universität Berlin bezüglich eines passenden Gehäuses gestellt. Vorge schlagen wurde ein gegen Regenwasser geschütztes Gehäuse mit einer aktiven Kühlung. Die Konstruktion eines solchen Gehäuses hätte allerdings mehrere Monate gedauert, so dass diese Möglichkeit vorerst aus Zeitmangel verworfen wurde.

Um das System dennoch im Feld testen zu können wurde provisorisch für den Prototyp ein offenes Gehäuse aus einer einfachen Plastikbox mit einem übergroßen Deckel gewählt. Diese Eigenkonstruktion ist gut gegen Regenwasser geschützt und gewährleistet durch die offene Konstruktion eine gute Abfuhr der Wärme.



Abbildung 13: Netbook-Komponente im Feld, Gehäuse offen



Abbildung 14: Netbook-Komponente im Feld, Gehäuse geschlossen

2.2.1.3 Software

Als Betriebssystem der Netbooks wird wie oben bereits beschrieben Windows XP eingesetzt. Die Windows Services werden auf Basis des .NET 3.5 Frameworks implementiert.

Damit die Benutzer-Komponente auch auf Netbooks zugreifen kann, welche sich nicht in direkter Funkreichweite befinden, bildet die Netbook-Komponente ein Mesh-Netzwerk mit Multihop-Unterstützung. Dazu wird die Netzwerkstackerweiterung MCL (Mesh Connectivity Layer) von Microsoft Research [14] genutzt.

2.2.2 Benutzer-Komponente

2.2.2.1 Aufgaben

Aufgabe der Benutzerkomponente ist es dem Benutzer eine einfache Steuerung des Systems zu ermöglichen. So können beispielsweise Aufnahmen programmiert und vollendete Aufnahmen als komprimierte Audiodateien heruntergeladen werden. Dieses wird durch eine grafische Benutzeroberfläche realisiert.

2.2.2.2 Hardware

Die Hardwarevoraussetzung für die Benutzerkomponente ist ein gängiger Computer mit einem WLAN-Adapter, vorzugsweise ein mobiler Rechner, welcher leicht ins Feld mitgenommen werden kann.

2.2.2.3 Software

Als Betriebssystem wird Windows XP und die Laufzeitumgebung Microsoft .NET 3.5 vorausgesetzt. Um die Multihop-Eigenschaften des Systems nutzen zu können wird ferner die Netzwerkstackerweiterung MCL benötigt, welche auch auf den Rechnern der Netbook-Komponente zum Einsatz kommt.

2.2.3 Erweiterungs-Komponente

Eine Erweiterungs-Komponente wird von Marco Ziegert im Rahmen seiner Diplomarbeit erstellt [13]. Ihre Aufgabe ist es auf Basis eines Mikrocontrollers zu vorher vom Benutzer festgelegten Zeiten Audioaufnahmen durchzuführen und diese auf die Netbook-Komponente zum späteren Download durch den Benutzer zu übertragen. Die Kommunikation mit der Netbook-Komponente wird mittels WLAN realisiert. Sie stellt eine Alternative zur Aufnahmefunktion der Netbook-Komponente da und kann vor allem überall dort eingesetzt werden, wo eine kabelgebundene Verbindung der Netbook-Komponente zum Aufnahmegerät aufgrund von physischen Gegebenheiten schwierig oder unmöglich ist.

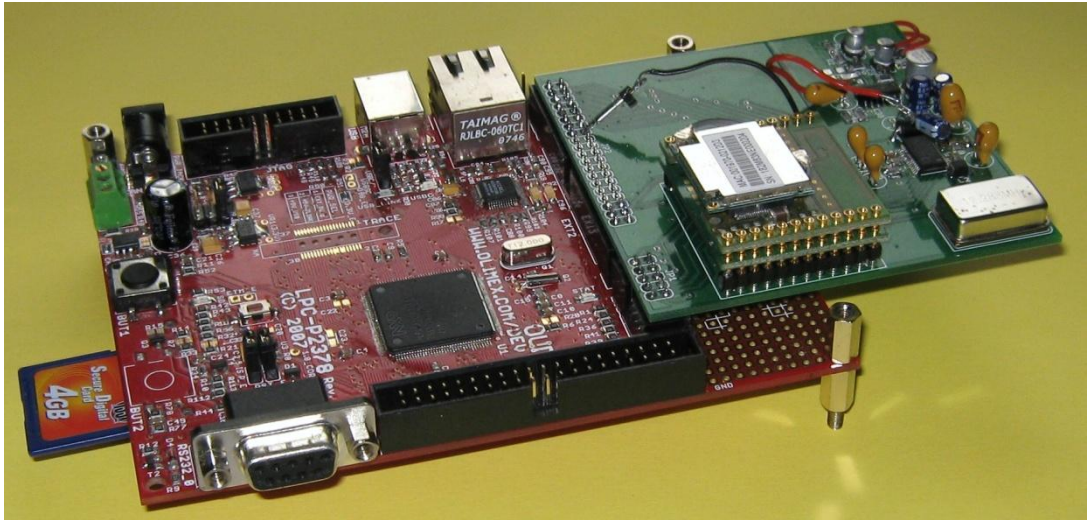


Abbildung 15: Erweiterungs-Komponente

3 Grundlagen

3.1 Windows Communication Foundation

Folgendes Kapitel beschreibt die Windows Communication Foundation (WCF). Die Informationen entstammen aus [15].

WCF ist ein Framework für verteilte Anwendungen. WCF ist Bestandteil von Microsoft .NET seit Version 3.0 und löst bisher konkurrierende Technologien wie .NET Sockets und .NET Remoting ab.

Die Kommunikation zwischen einem WCF-Service und einem Client geschieht über Punkt-zu-Punkt-Verbindungen. Der Zugangspunkt des Services basiert dabei auf dem ABC-Konzept. A steht dabei für Address (Adresse) und beschreibt die Adresse, wo der Service zu finden ist. B steht für Binding (Bindung) und beschreibt wie mit dem Service zu kommunizieren ist. C steht für Contract (Vertrag) und beschreibt, welche Dienstleistungen der Service anbietet.

3.1.1.1 Adresse

Adressen zu Services bestehen aus einem Uniform Resource Identifier (URI) und optionalen Erweiterungen. Ein URI besteht aus einem Protokoll gefolgt vom Namen des Rechners, auf welchem der Service gehostet ist. Weiterhin folgen Port und Verzeichnis des Services.

Beispiel:

<http://MeinServer:8080/Service1>

3.1.1.2 Binding

Das Binding beschreibt, wie auf einen Service zugegriffen werden kann. Es beschreibt das Protokoll, die Kodierung, Sicherheit und Zustellungsgarantien. Zu

jedem Binding eines Services gehört genau ein Contract, wobei ein Contract mehrere Bindings besitzen kann. Zu den Standardbindings in WCF gehören: BasicHttpBinding, WSHttpBinding, WSDualHttpBinding, WSFederationHttpBinding, NetTcpBinding, NetPeerTcpBinding, NetNamedPipeBinding, NetMsmqBinding, MsmqIntegrationBinding und selbst definierte CustomBindings. Jedes Binding besitzt spezielle Vor- und Nachteile den anderen Bindings gegenüber. Dazu gehören die Unterstützung für Interoperabilität, Security, Sessions, Transaktionen und Duplexkommunikation.

3.1.1.3 Contract

Contracts definieren Schnittstellen, Daten und Nachrichten, die ein Service selbst anbietet oder von anderen annimmt. Man unterscheidet zwischen vier Arten von Contracts in WCF: Dienstschnittstelle, Transferdaten, Nachrichten und Fehlerbehandlung. Diese Contract-Arten werden über ihre jeweiligen Attribute definiert.

Weitere Informationen zu WCF sind zu finden in [15] und [16].

3.2 DirectSound

DirectSound ist Bestandteil von Microsoft DirectX, welches eine Sammlung von Application Programming Interfaces (API) für verschiedene Windows Plattformen ist. DirectSound bietet Applikationen eine einfache Möglichkeit direkt auf Treiber von Soundkarten zuzugreifen, wobei sowohl Soundausgabe als auch Soundaufnahme unterstützt werden.

Um die DirectX API unter .NET nutzen zu können, wird Managed DirectX (MDX) verwendet. MDX stellt für alle .NET Programmiersprachen ein einfaches Interface auf die Komponenten von DirectX bereit und vereinfacht so die Entwicklung von Applikationen, welche direkten Hardwarezugriff benötigen.

Die für die Arbeit relevante Soundaufnahme wird dabei über sogenannte Capture Devices realisiert. Jedes Aufnahmegerät stellt dabei ein eigenes Capture Device bereit, so dass Aufnahmen von verschiedenen Audioquellen gleichzeitig aufgezeichnet werden können.

Näheres zu DirectX, DirectSound und Managed DirectX sind zu finden unter [17] und [18].

3.3 Mesh Connectivity Layer

Der Mesh Connectivity Layer (MCL) ist eine von Microsoft Research entwickelte Implementierung des Mesh-Routing Protokolls Link Quality Source Routing (LQSR), entwickelt von Drayes, Padhye und Zill [19]. Dabei ist LQSR eine Weiterentwicklung von Dynamic Source Routing (DSR), beschrieben in [20].

LQSR wurde als Routing Protokoll gewählt, da es optimiert ist für Szenarien, in welchen die Funkstationen ihren Aufenthaltsort nicht oder sehr selten wechseln und ein hoher Datendurchsatz erforderlich ist. Dazu wurde DSR um mehrere Metriken ergänzt: Hop Count, Per-hop Round Trip Time, Per-Hop Paket Pair Delay und Expected Transmission Count. Ausführliche Beschreibungen der einzelnen Metriken und die genaue Funktionsweise von LQSR ist zu finden in [19].

Microsoft Research stellt das Treiberpaket MCL zur Verfügung. Es implementiert LQSR und stellt einen virtuellen Netzwerkadapter bereit, welcher auf Schicht 2,5 des ISO-OSI Modells arbeitet. Diese Architektur hat zwei Vorteile. Zum einen können Applikationen, die auf höheren Schichten aufsetzen, ohne Veränderungen betrieben werden. Zum anderen erscheint MCL nur als weiteres Protokoll für Software, die auf Schichten unterhalb von MCL arbeitet, was dazu führt, dass unterschiedliche, physikalische Netzwerkadapter benutzt und kombiniert werden können, z.B. Ethernet und WLAN. Näheres zu MCL ist zu finden bei [14].

3.4 WavPack

WavPack ist ein freies Audioformat unter BSD-Lizenz (Berkeley Software Distribution). Es unterstützt verlustfreie und verlustbehaftete Kompression von Audio-daten. Darüber hinaus beherrscht es einen hybriden Modus, in welchem eine verlustbehaftete Audiodatei und eine dazugehörige Korrekturdatei erzeugt werden. Zusammengenommen lässt sich aus diesen beiden Dateien wiederum eine verlustfreie Audiodatei erzeugen. Die durchschnittliche Kompressionsrate bei verlustfreier Kompression wird von den Entwicklern mit 30% - 70% angegeben, abhängig von der Beschaffenheit des Ausgangsmaterials.

WavPack nutzt nur freie Technologien zur Kompression. Geschützte Technologien wie LZW-Kompression (Lempel-Ziv-Welch Kompression) oder Arithmetisches Kodieren werden bewusst nicht eingesetzt um WavPack auch in Zukunft unter BSD-Lizenz anbieten zu können.

WavPack unterstützt alle gängigen PCM (Pulse-Code-Modulation) Formate: Mono, Stereo, Multichannel, ein großes Spektrum an Samplingraten und Auflösungen. Es ist für viele Plattformen verfügbar, wie z.B. Windows, Linux, OS X und ARM-Mikrocontroller. Da die Kompression ausschließlich mit Ganzzahloperationen operiert, ist WavPack besonders für Mikrocontroller-Plattformen geeignet.

Weitere Informationen, Quellcode und Plugins für weit verbreitete Standardprogramme können auf [21] bezogen werden.

4 Softwarearchitektur

4.1 Netbook-Komponente

Wie bereits in Kapitel 2.2 beschrieben, gliedert sich die Netbook-Komponente in mehrere Services, die über XML-Dateien lose gekoppelt sind. Abbildung 16 zeigt die Kopplung der Services.

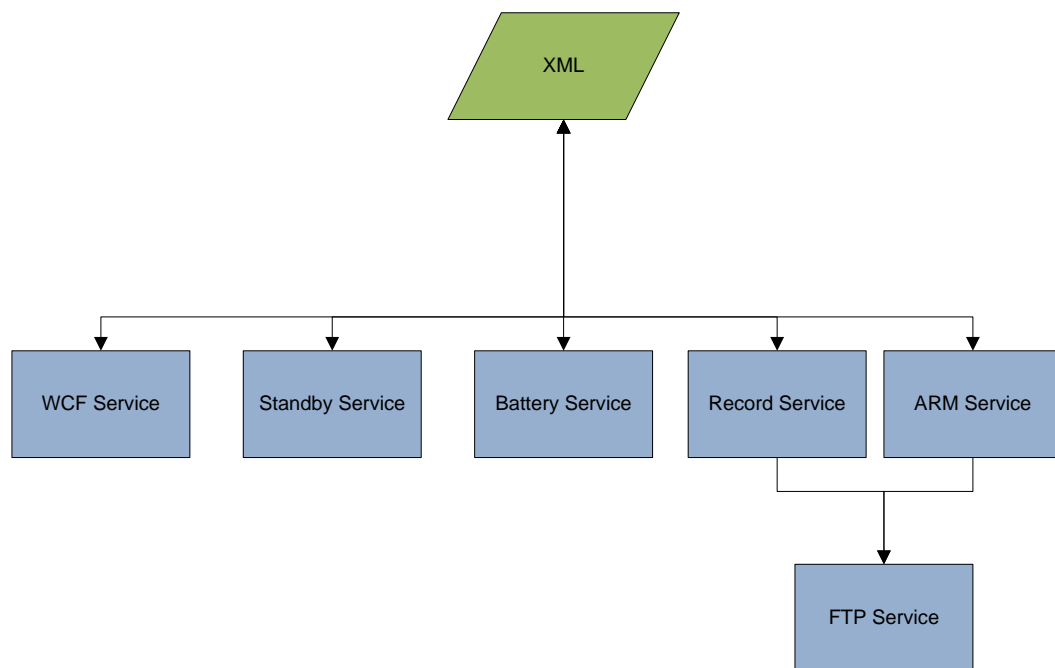


Abbildung 16: Übersicht Netbook-Komponente

4.1.1 XML-Dokumente

Die einzelnen Windows-Services kommunizieren untereinander über mehrere XML Dateien. Dabei ist zu beachten, dass Windows-Services bei einem Schreibzugriff die jeweilige Datei für den Zugriff anderer Windows-Services sperren. Sollte eine Datei gesperrt sein, so wartet der anfragende Windows-Service eine zufällige Zeit und versucht es erneut. Nach einer festgelegten Anzahl von Versuchen beendet der er den Zugriffsversuch um das System nicht über unbestimmte Zeit zu blockieren.

Folgende XML-Dateien werden zur Verfügung gestellt:

- Schedule.xml
- Config.xml
- Status.xml
- Power.xml
- Compress.xml
- Download.xml
- Standby.xml

4.1.1.1 Schedule.xml

In der Schedule.xml werden die programmierten Aufnahmetimer gespeichert. Ein Aufnahmetimer-Element (<timer>) besteht aus einer eindeutigen ID (<timer_no>), Startzeitpunkt (<start>) im Format „DD.MM.YYYY hh:mm:ss“, analog dazu der Endzeitpunkt (<end>), einer Aufnahmegeräteerkennung (<node>) und einem Freitextfeld (<text>) für Notizen zum jeweiligen Aufnahmetimer des Benutzers. Die Aufnahmegeräteerkennung setzt sich aus einer GUID (Globally Unique Identifier) und einer ID des lokalen Rechners (StationID) zusammen. Die GUID wird von Windows vergeben und identifiziert eindeutig ein Aufnahmegerät am lokalen Rechner. Sie ist eine 128bit große Zahl im Format „XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX“, wobei X für eine Hexadezimalziffern steht. Auf System mit dem Betriebssystem Windows XP sind diese GUIDs für Aufnahmegeräte immer gleich, bis auf die letzten Stellen, welche die Aufnahmegeräte durchnummerieren. Beispielsweise ist dem ersten Aufnahmegerät unter Windows XP immer die GUID „bd6dd71b-3deb-11d1-b171-00c04fc20000“ zugeordnet.

Neben diesen Aufnahmetimern wird noch eine Timestamp und die maximale bisher vergebene Aufnahmetimer-ID gespeichert. Die Timestamp gibt dabei den Zeitpunkt an, zu dem die Schedule.xml zuletzt geändert wurde.

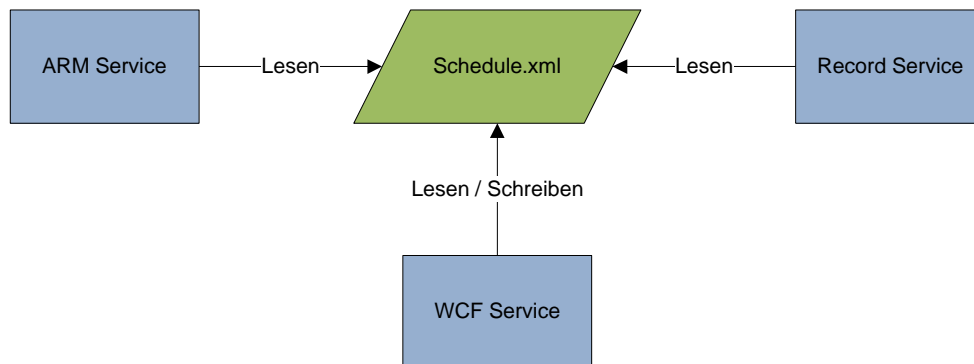


Abbildung 17: Kopplung Schedule.xml

Beispiel Schedule.xml:

```

<?xml version="1.0" encoding="utf-8" ?>
<schedule>
  <timestamp>11.06.2009 17:05:04</timestamp>
  <maxid>229</maxid>
  <timer>
    <timer_no>226</timer_no>
    <start>09.06.2009 23:30:59</start>
    <end>10.06.2009 00:10:59</end>
    <node>bd6dd71b-3deb-11d1-b171-00c04fc20002:1</node>
    <text>
    </text>
  </timer>
  <timer>
    <timer_no>227</timer_no>
    <start>09.06.2009 23:30:59</start>
    <end>10.06.2009 00:10:59</end>
    <node>bd6dd71b-3deb-11d1-b171-00c04fc20000:1</node>
    <text>
    </text>
  </timer>
  <timer>
    <timer_no>228</timer_no>
    <start>09.06.2009 23:30:59</start>
    <end>10.06.2009 00:10:59</end>
    <node>bd6dd71b-3deb-11d1-b171-00c04fc20003:1</node>
    <text>
    </text>
  </timer>
</schedule>

```

```

<timer>
  <timer_no>229</timer_no>
  <start>10.06.2009 00:30:18</start>
  <end>10.06.2009 02:30:18</end>
  <node>bd6dd71b-3deb-11d1-b171-00c04fc20002:2</node>
  <text>
  </text>
</timer>
</schedule>

```

4.1.1.2 Config.xml

In der Config.xml werden Einstellungen für diesen Rechner gespeichert. Zum einen sind dies drei Pfade für verschiedene Speicherorte auf dem lokalen Rechner (<downloadpath>, <schedulepath>, <temppath>), zum anderen die IP-Adresse des lokalen Mesh-Adapters (<ip>) und eine im gesamten System eindeutige ID, die diesen Rechner identifiziert (<station>).

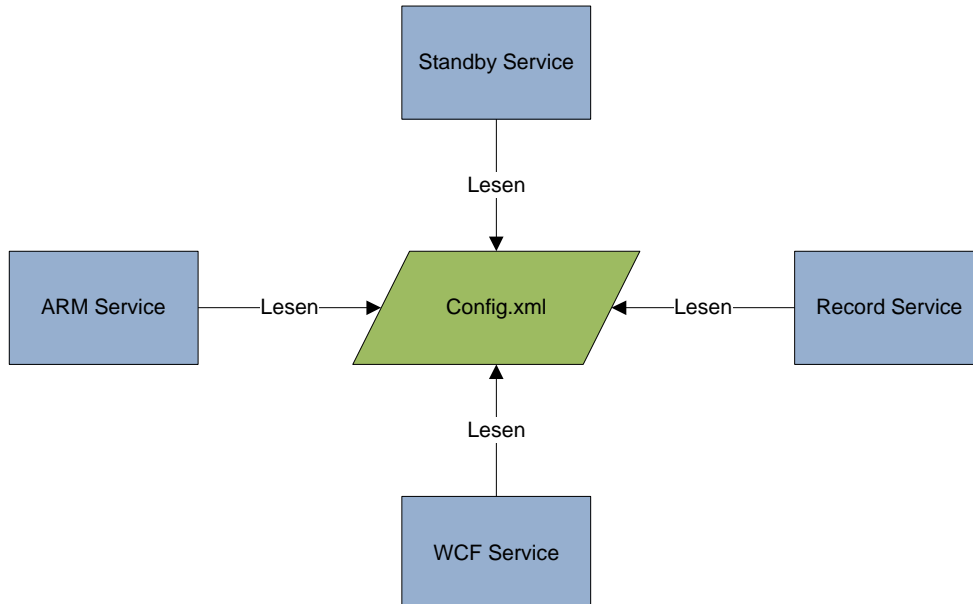


Abbildung 18: Kopplung Config.xml

Beispiel Config.xml:

```

<?xml version="1.0" encoding="utf-8" ?>
<config>
  <downloadpath>C:\data\</downloadpath>
  <schedulepath>C:\service\</schedulepath>
  <temppath>C:\temp\</temppath>
  <ip>192.168.1.10</ip>
  <station>1</station>
</config>

```

4.1.1.3 Status.xml

In der Status.xml werden die verfügbaren Aufnahmegeräte gespeichert. Das Tag „<id>“ ist hierbei die oben bereits erwähnte GUID des Aufnahmegerätes und die Station ID, getrennt durch einen Doppelpunkt. Durch dieses Tag lässt sich das Aufnahmegerät eindeutig identifizieren. Im Tag „<type>“ wird die Art des Aufnahmegerätes angegeben. Eine „1“ steht hierbei für ein lokales Soundaufnahmegerät, wie wir es verwenden. Bei einer „2“ ist das Aufnahmegerät ein durch die ARM-Erweiterung von Marco Ziegert bereitgestelltes Aufnahmegerät. Ist das Aufnahmegerät nicht ein lokales Aufnahmegerät vom Typ 1, so kann die ID im Tag „<id>“ auch einem anderen Schema entsprechen als das oben beschriebene. Im Tag „<optional>“ können weitere Informationen über das Aufnahmegerät gespeichert werden. Dieses Tag wird jedoch in dieser Version noch nicht benutzt und ist für Erweiterungen gedacht.

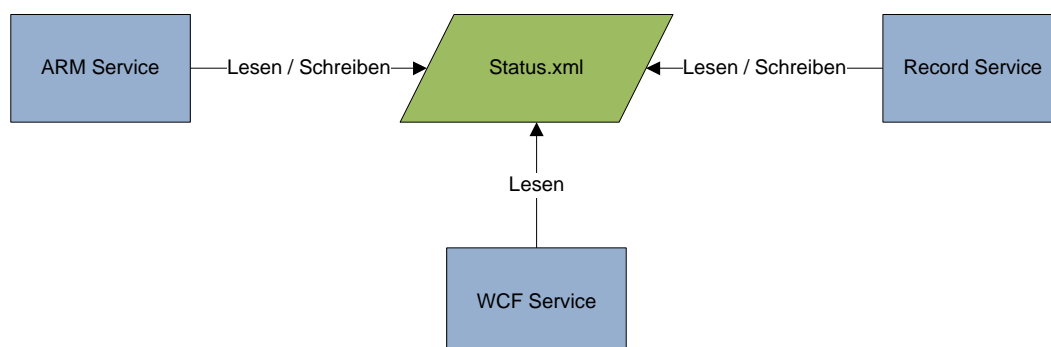


Abbildung 19: Kopplung Status.xml

Beispiel Status.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<status_set>
  <status>
    <id>ARM1</id>
    <type>2</type>
    <optional>
      </optional>
    </status>
  <status>
    <id>bd6dd71b-3deb-11d1-b171-00c04fc20000:1</id>
    <type>1</type>
    <optional>
      </optional>
    </status>
  <status>
    <id>bd6dd71b-3deb-11d1-b171-00c04fc20002:1</id>
    <type>1</type>
    <optional>
      </optional>
    </status>
  <status>
    <id>bd6dd71b-3deb-11d1-b171-00c04fc20003:1</id>
    <type>1</type>
    <optional>
      </optional>
    </status>
</status_set>
```

4.1.1.4 Power.xml

In der Power.xml wird die aktuelle Spannung der Stromversorgung gespeichert (Tag „<power>“).

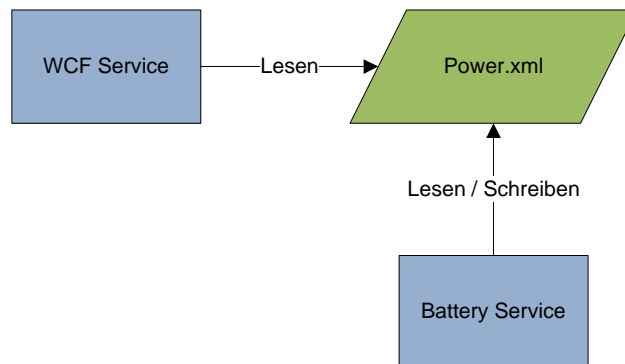


Abbildung 20: Kopplung Power.xml

Beispiel Power.xml:

```

<?xml version="1.0" encoding="utf-8" ?>
<power>12.0</power>
  
```

4.1.1.5 Compress.xml

In der Compress.xml wird der Pfad gespeichert, an dem sich das Komprimierungstool befindet. In dieser Version ist das der Pfad des WavPack Kommandozeilentools im Tag „<wavpack_path>“.

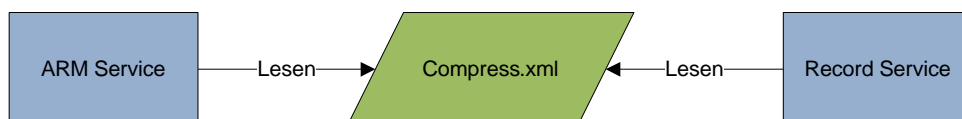


Abbildung 21: Kopplung Compress.xml

Beispiel Compress.xml:

```

<?xml version="1.0" encoding="utf-8" ?>
<Properties>
  <wavpack_path>C:\\Program
Files\\Wavpack\\wavpack.exe</wavpack_path>
</Properties>
  
```

4.1.1.6 Download.xml

In der Download.xml werden die Pfade zu fertiggestellten Aufnahmen gespeichert. Dabei besteht die Möglichkeit, dass eine Aufnahme auf mehrere Dateien, sogenannte Chunks, aufgeteilt ist. Ein Chunk (Tag „<chunk>“) hat hierbei eine ID (Tag „<chunkID>“), die für eine bestimmte Aufnahme eindeutig ist. Diese ID ist eine fortlaufende Nummerierung der Chunks zu dieser Aufnahme, beginnend bei eins. Das Tag „<timerID>“ ordnet diesen Chunk dem zugehörigen Timer der Aufnahme eindeutig zu. Durch die Chunk-ID und die Timer-ID ist ein Chunk eindeutig identifiziert. „<url>“ gibt den Pfad des Chunks an, relativ zum Homeverzeichnis des FTP-Servers, im allgemeinen also nur den Dateinamen.

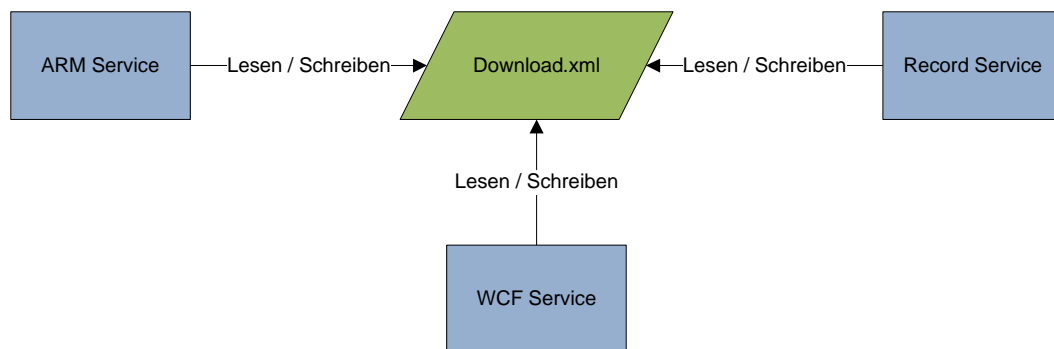


Abbildung 22: Kopplung Download.xml

Beispiel Download.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<download>
  <chunk>
    <chunkID>1</chunkID>
    <timerID>230</timerID>
    <url>timerID230 station1 micIDbd6dd71b-3deb-11d1-b171-
00c04fc20000 start 11.06.2009 17_05_59.wv</url>
  </chunk>
  <chunk>
    <chunkID>1</chunkID>
```

```

    <timerID>231</timerID>
    <url>timerID231 station1 micIDbd6dd71b-3deb-11d1-b171-
00c04fc20002 start 11.06.2009 17_05_59.wv</url>
  </chunk>
<chunk>
  <chunkID>1</chunkID>
  <timerID>235</timerID>
  <url>timerID235 station1 micIDbd6dd71b-3deb-11d1-b171-
00c04fc20003 start 11.06.2009 17_05_59.wv</url>
</chunk>
<chunk>
  <chunkID>1</chunkID>
  <timerID>236</timerID>
  <url>timerID236 station1 micIDbd6dd71b-3deb-11d1-b171-
00c04fc20000 start 11.06.2009 17_05_59.wv</url>
</chunk>
</download>

```

4.1.1.7 Standby.xml

In der Standby.xml werden verschiedene Werte gespeichert, die das Energiemanagement des Systems steuern. Die Tags „<from>“ und „<to>“ definieren die Start- bzw. Endzeiten der Service Time. Die Service Time ist eine Zeitspanne, die der Benutzer frei festlegen kann zu der das System verfügbar sein soll, es wird also während dieser Zeitspanne kein Wechsel in den Standby-Modus stattfinden. Dies kann dafür genutzt werden, dass der Benutzer im Feld Aufnahmen downloaden kann, wofür das System natürlich verfügbar sein muss, ohne dass dieser manuell das System aus dem Standby-Modus erwecken muss. Das Tag „<keepalive>“ ist ein Zeitstempel, der angibt, dass das System nicht in den Standby-Modus wechseln solle, falls er nur eine geringe Zeit in der Vergangenheit liegt. Andere Windows-Services können hier einen aktuellen Zeitstempel eintragen um zu verhindern, dass das System in den Standby-Modus wechselt, obwohl der jeweilige Windows-Service noch Aufgaben zu erledigen hat. Das Tag „<next>“ gibt den Zeitpunkt an, zu welchem der nächste Aufnahmetimer ansteht, damit das System rechtzeitig aus dem Standby-Modus reaktiviert werden kann.

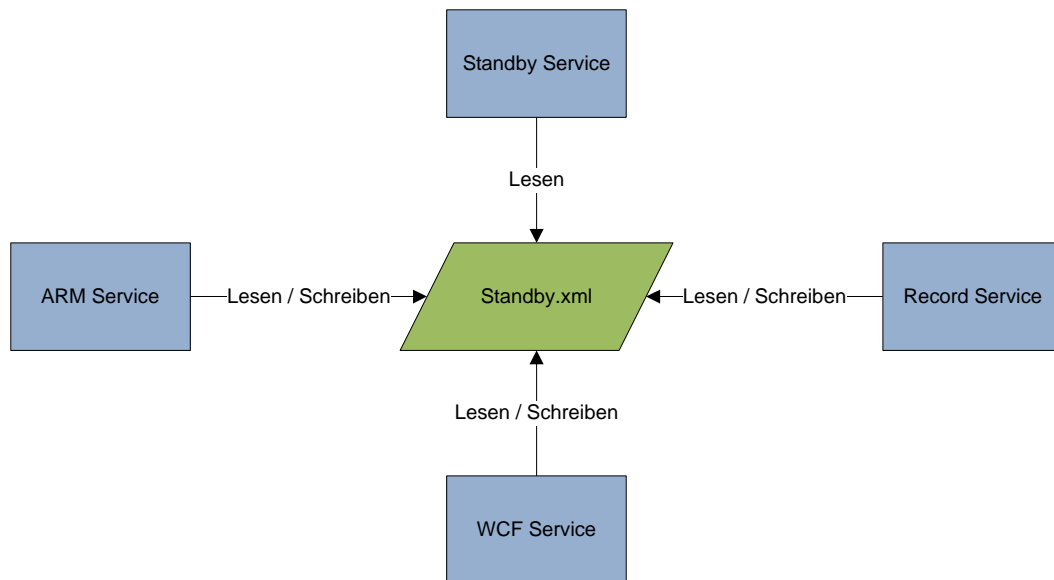


Abbildung 23: Kopplung Standby.xml

Beispiel Standby.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<dutytime>
  <from>08:00</from>
  <to>09:00</to>
  <keepalive>11.06.2009 17:05:58</keepalive>
  <next>12.06.2009 10:00:00</next>
</dutytime>
  
```

4.1.2 WCF-Service

Der WCF Service stellt einen WCF-Dienst bereit, über welchen die Netbook-Komponente gesteuert werden kann. Er greift dazu auf die XML-Dateien zu, die ihn mit den restlichen Services des lokalen Rechners koppelt. Abbildung 24 zeigt diese Kopplung.

Aufgaben des WCF-Dienstes sind:

- Empfangen und Verarbeiten eines neuen Zeitplans
- Senden des aktuellen Zeitplans
- Senden von Informationen über verfügbare Aufnahmegeräte
- Senden des Zeitstempels des aktuellen Zeitplans
- Senden einer Liste der verfügbaren Downloads
- Löschen von Downloads
- Setzen einer Service Time
- Setzen eines „Keep Alive“ Signals
- Senden von Informationen über das System (lokale Zeit, Spannungsversorgung)

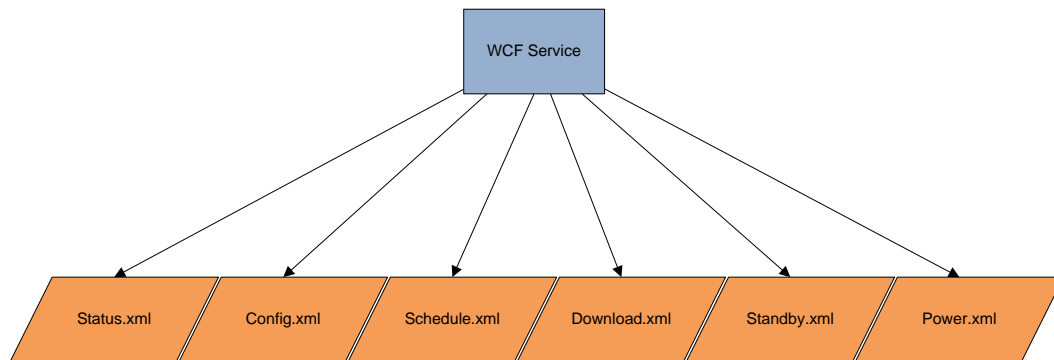


Abbildung 24: Kopplung WCF-Service

Weiterführende Beschreibungen zu den Aufgaben und API des WCF-Service sind zu finden im Anhang 7.1.

4.1.3 Standby Service

Der Standby Service hat die Aufgabe, den lokalen Rechner bei Aufnahmen oder zu einer vorher vom Benutzer bestimmten Zeit aus dem Standby-Modus aufzuwecken und in den übrigen Zeiten ihn in den Standby-Modus zu versetzen um Energie zu sparen. Diese Funktionalität ist von entscheidender Bedeutung, da das System im freien Feld ohne Infrastrukturstromversorgung betrieben wird und die autonome Laufzeit direkt vom Energiemanagement des Systems abhängig ist. Desweiteren ist es Aufgabe des Standby Services in regelmäßigen Abständen bei anderen, erreichbaren Rechner der Netbook-Komponente nachzufragen, ob es eine neuere Version als die ihm aktuell vorliegende der Schedule.xml gibt. Sollte dies der Fall sein, so aktualisiert der Standby Service die lokale Schedule.xml auf den neueren Stand des kontaktierten Rechners. Dadurch wird sichergestellt, dass auch ein Rechner, der bei einer Aktualisierung der Schedule.xml durch den Benutzer nicht verfügbar war, die aktuellen Daten erhalten kann und die Datenbasis des Systems seine Konsistenz bewahrt.

Die Zugriffe einer .NET Anwendung auf das Betriebssystem sind gewissen Beschränkungen unterworfen. Es ist möglich den Rechner in den Standby-Modus zu versetzen, jedoch nicht ihn aus diesem wieder zu erwecken. Das setzen des Standby-Modus geschieht durch den Aufruf der Funktion `SetSuspendState`, genauer mit dem Aufruf: `System.Windows.Forms.Application.SetSuspendState(System.Windows.Forms.PowerState.Suspend, true, false);`. Der erste Parameter definiert die Art des Systemstatus, in unserem Fall der Standby-Modus (suspend). Der zweite Parameter gibt an, ob ein Wechsel in diesen Modus erzwungen werden soll. Der dritte Parameter verhindert oder ermöglicht das Aufwecken des Rechners aus diesem Modus, wobei false ein Aufwecken ermöglicht.

Um den Rechner nun aber wirklich aus dem Standby-Modus aufwecken zu können, benötigen wir einige Funktionen, die nicht Teil von .NET sind. Diese Funktionen befinden sich in der kernel32.dll unseres Windows-Systems. Es werden drei

Funktionen aus dieser kernel32.dll importiert: CreateWaitableTimer, SetThreadExecutionState und SetWaitableTimer.

SetThreadExecutionState signalisiert Windows nach einem beendeten Standby, dass das System nicht wieder in den Standby-Modus wechseln soll, was es sonst nach einer gewissen Zeit automatisch tun würde. Ebenso kann hier angegeben werden, ob der Monitor eingeschaltet werden soll oder nicht. In unserem Fall wird kein Monitor benötigt und daher kein „ES_DISPLAY_REQUIRED“ gesetzt. CreateWaitableTimer erzeugt einen Timer, der es ermöglicht, das System aus einem Standby zu erwecken und SetWaitableTimer aktiviert diesen Timer und setzt die gewünschte Zeitspanne, die vergehen soll, bis der Timer ausgelöst wird.

4.1.3.1 Programmablauf

Der Service startet zwei Timer vom Typ System.Threading.Timer. Der eine Timer (Timer1) wird alle fünf Minuten aufgerufen und ist für den Wechsel in den Standby-Modus zuständig. Der andere Timer (Timer2) löst nur alle 30 Minuten aus und aktualisiert die Schedule.xml, falls erreichbare Rechner eine neue Version verfügbar haben.

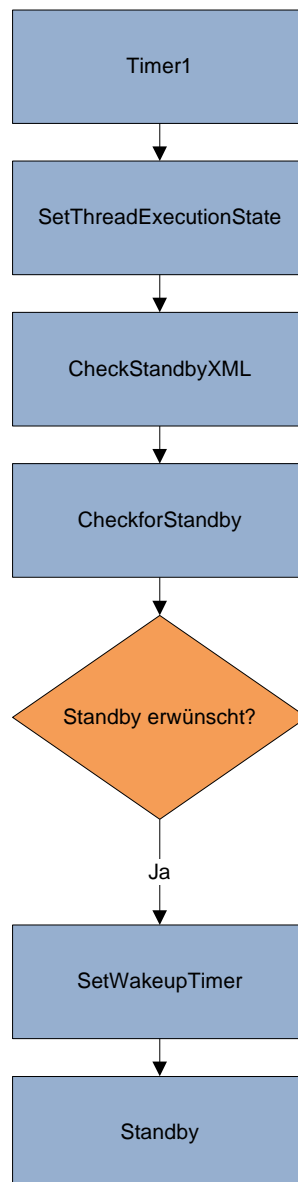


Abbildung 25: Programmfluss Timer1

Löst Timer1 aus, so wird die Funktion CheckStandbyXML aufgerufen. Diese liest die Standby.xml und aktualisiert die Variablen, welche Zeitstempel zur Standby-Steuerung verwalten. Daraufhin wird die Funktion CheckforStandby aufgerufen, die überprüft, ob ein Wechsel in den Standby-Modus durchgeführt werden soll. Ein Wechsel wird nur dann durchgeführt, wenn folgende Bedingungen erfüllt sind:

- Die vom Benutzer definierte Service-Time ist zu diesem Zeitpunkt nicht aktiv. Diese Service-Time wird in den Variablen „from“ und „to“, welche den Start- und Endzeitpunkt definieren, verwaltet.
- Der letzte Zeitstempel „keepalive“ aus der Standby.xml liegt länger als sechs Minuten in der Vergangenheit. Dieses wird in der Variable „keepalive“ verwaltet.
- Der nächste Aufnahmetimer liegt mehr als zehn Minuten in der Zukunft. Dieses wird in der Variable „next“ verwaltet.
- Der letzte Aufruf der CheckforStandby Methode darf nicht länger als sechs Minuten in der Vergangenheit liegen. Dann wird davon ausgegangen, dass der Benutzer manuell den Rechner reaktiviert hat und nicht wünscht, dass er sofort wieder in den Standby-Modus wechselt. Die Information, wann der letzte Aufruf der CheckforStandby Methode stattgefunden hat, wird in der Variable „last“ verwaltet.

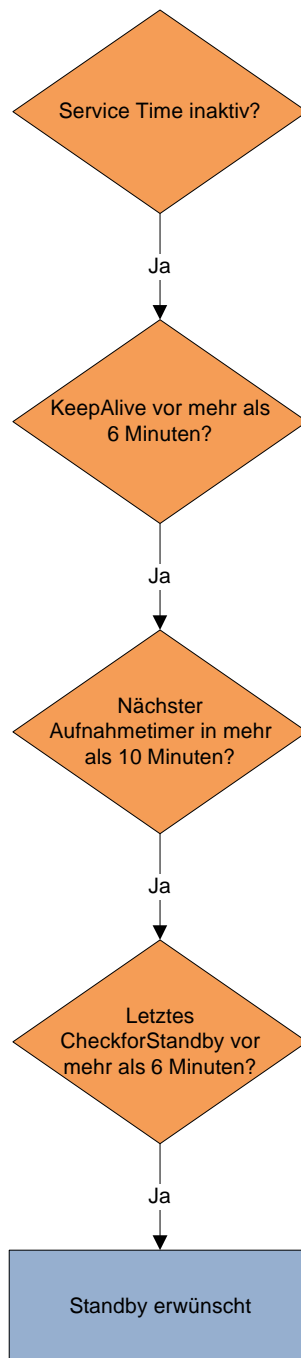


Abbildung 26: Bedingungen für Standby

Sollten alle vier Bedingungen erfüllt sein, so kann das System in den Standby-Modus gesetzt werden. Zuvor ist jedoch ein Timer zu setzen, der das System zur richtigen Zeit wieder aus dem Standby-Modus in den normalen Betriebsmodus

versetzt. Es wird also erst einmal ermittelt, ob als nächstes Ereignis ein Aufnahmetimer oder der Start der Service-Time ansteht. Die Zeitspanne zu diesem nächsten Ereignis abzüglich einer Pufferzeit von zwei Minuten wird berechnet und der Timer damit gestartet. Dabei ist als Timer der oben beschriebene WaitableTimer zu benutzen, da nur dieser das System aus dem Standby-Modus reaktivieren kann. Das System wird nun in den Standby-Modus versetzt und der WaitableTimer reaktiviert es zwei Minuten vor dem nächsten Ereignis. Nach Reaktivierung wird ein „KeepAlive“ mit einem aktuellen Zeitstempel in der Standby.xml gesetzt um eine erneute, sofortige Aktivierung des Standby-Modus zu verhindern. Die SetThreadExecutionState muss periodisch aufgerufen werden um Windows zu hindern, ungewollt in den Standby-Modus zu wechseln. Dies geschieht allerdings bereits in Timer1 alle zwei Minuten. Zusätzlich wird noch die Methode Replication aufgerufen, die die oben beschriebene Aktualisierung der Schedule.xml mit verfügbaren Rechnern durchführt.

Die einzige Aufgabe von Timer2 besteht darin in regelmäßigen Abständen die Replication Methode aufzurufen. Die Replication Methode startet mit Hilfe des Threadpools mehrere Threads, die jeweils die Aufgabe haben den WCF-Service eines bestimmten anderen Rechners zu kontaktieren und den Zeitstempel dessen Schedule.xml abzurufen. Ist der Zeitstempel neuer als der Zeitstempel der lokalen Schedule.xml, so wird die lokale Schedule.xml mit den Daten aus der aktuelleren Version des kontaktierten Rechners ersetzt.

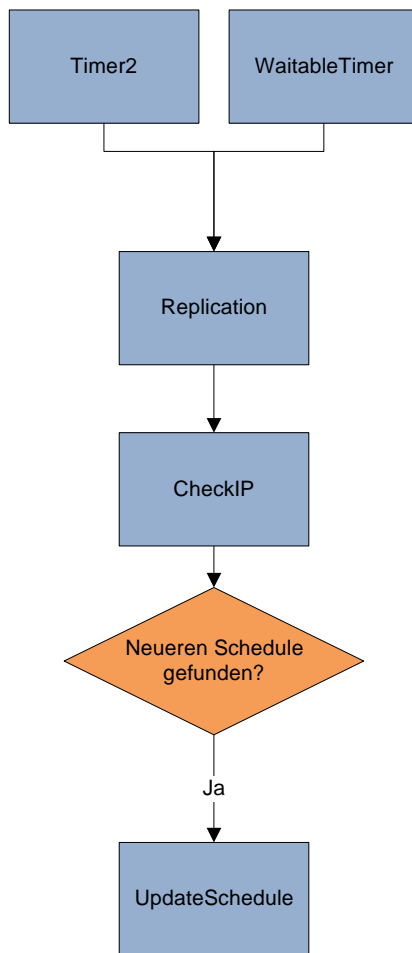


Abbildung 27: Replication

4.1.4 Record-Service

4.1.4.1 Kurzbeschreibung

Der Record Service ist ein Windows Service mit der Aufgabe lokal verfügbare Aufnahmegерäte in die Status.xml Datei einzutragen, Aufnahmen pünktlich auszuführen, vollendete Aufnahmen zu komprimieren, vollendete Aufnahmen in die Download.xml einzutragen und durch Modifizierung der Standby.xml dafür zu sorgen, dass der Rechner zur Aufnahmezeit durchgehend betriebsbereit ist ohne in den Standby-Modus zu wechseln.

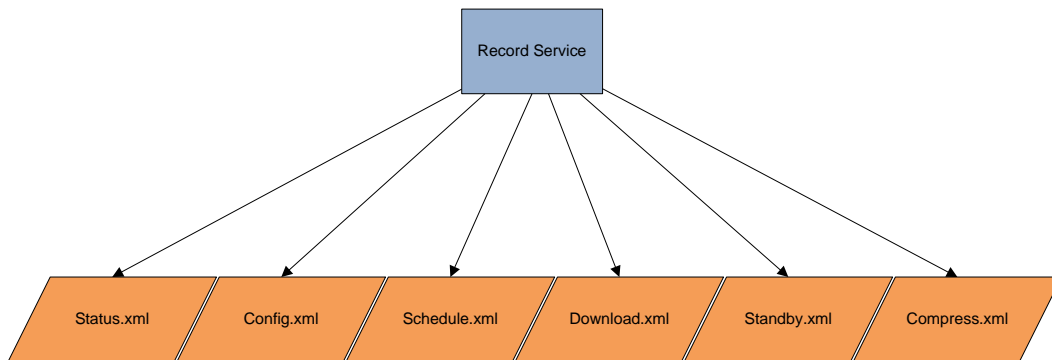


Abbildung 28: Kopplung Record-Service

4.1.4.2 Ablaufbeschreibung

Der Record Service selbst startet und überwacht nur einen weiteren Prozess, den Recorder. Dies ist nötig, da aus einem Windows Service kein Zugriff auf Aufnahmegeräte über DirectSound möglich ist. In der OnStart-Methode wird einfach ein neuer Prozess instanziiert, ein Event registriert, welches ausgelöst wird, falls der Prozess beendet wird, und der Prozess anschließend gestartet.

Wird der Recorder Prozess nun beendet, gibt der Service den Speicher für diesen Prozess frei und startet den Recorder nach zehn Sekunden erneut. Wird der Service selbst beendet, wird der Recorder ebenfalls gestoppt und dessen Speicher freigegeben.

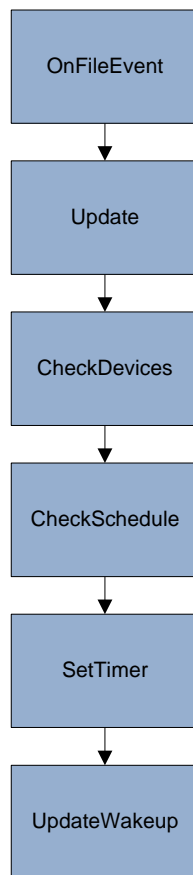


Abbildung 29: Programmablauf OnFileEvent

Der Recorder erstellt in seinem Konstruktor der Recordlib.cs zunächst einmal einige Datenstrukturen. „timer_list“ ist eine Liste von Timern aus der Klasse „Windows.Threading.Timer“ und verwaltet Timer, welche den Start einer Aufnahme auslösen. „timer_dic“ ist ein Dictionary aus der ID eines Aufnahmetimers und des Aufnahmetimers selbst. Hier werden für das System relevante Aufnahmetimer zwischengespeichert. Die Dictionary-Datenstruktur mit der ID als Schlüssel bietet einen schnellen Zugriff auf einen beliebigen Aufnahmetimer. Einige Pfade werden anschließend aus der Config.xml Datei gelesen, welche angeben, wo bestimmte Daten sich befinden oder gespeichert werden können. „schedulepath“ beschreibt hierbei den Pfad zur Datei Schedule.xml, „downloadpath“ den Pfad, wo vollendete Aufnahmen gespeichert werden sollen und „temppath“ den Pfad, wo Daten zwischengespeichert werden können.

Des weiteren wird noch ein Timer erzeugt, welcher alle fünf Minuten ausgelöst wird und eine aktuelle Timestamp in die Standby.xml schreibt, falls der Recorder gerade beschäftigt ist und ein Wechsel in den Standby-Modus verhindert werden soll.

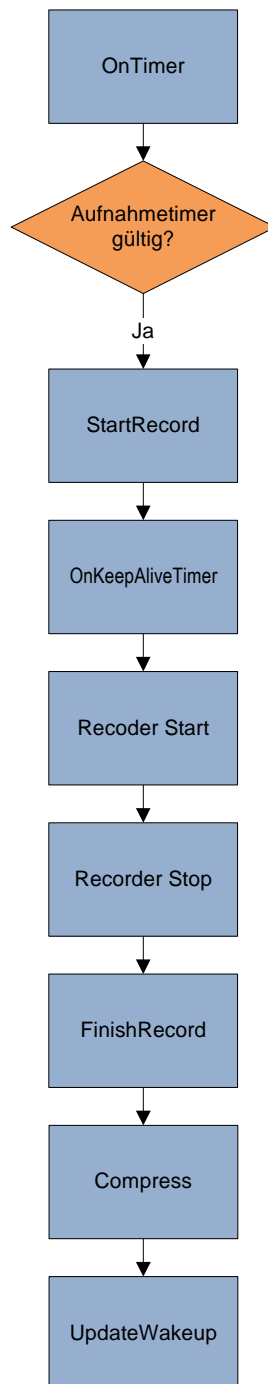


Abbildung 30: Programmablauf OnTimer

Nach erfolgreicher Instanziierung des RecordLib Objekts, startet der Recorder die Startup-Methode. Hier wird die Update-Methode aufgerufen und anschließend ein

FileSystemWatcher erzeugt, welcher die Schedule.xml auf Schreibzugriffe überwacht und dann entsprechend ein Event auslöst. In diesem Event wird wiederum die Update-Methode ausgeführt.

Die Update-Methode sorgt dafür, dass die aktuellen Aufnahmegeräte in die Status.xml eingetragen werden und die Schedule.xml auf Aufnahmetimer bezüglich lokal verfügbarer Aufnahmetimer überprüft wird. Diese gefundenen Aufnahmetimer werden im oben beschriebenen Dictionary „timer_dic“ abgelegt. Für jeden neuen Eintrag wird ein Timer gestartet und in der „timer_list“ abgelegt. Timer, die in der „timer_list“ vorhanden sind, aber deren Eintrag aus der Schedule.xml entfernt wurde, werden auch aus der „timer_list“ entfernt. Der zeitlich nächstgelegene Aufnahmetimer wird in der Standby.xml als „next“ eingetragen, diese Aufgabe führt die Methode UpdateWakeup aus.

Damit wird nun immer ein Timer gestartet, wenn für ein lokal vorhandenes Aufnahmegerät ein Aufnahmetimer vorliegt, und bei Änderung der Schedule.xml wird überprüft, ob sich für dieses Gerät relevante Aufnahmetimer geändert haben.

Läuft ein Timer ab und löst das OnTimer Event aus, so wird überprüft, ob es für diesen Timer im „timer_dic“-Dictionary ein Eintrag vorhanden ist. Sollte dies der Fall sein, so startet die Aufnahme mit der Methode StartRecord. Andernfalls wurde der Aufnahmetimer gelöscht und eine Aufnahme wird nicht gestartet.

StartRecord berechnet nun die Zeit von Startzeit zu Endzeit des Aufnahmetimers und erzeugt einen WavRecorder. Dieser wird in einem neuen Thread gestartet und nach Ablauf der Aufnahmezeit gestoppt. FinishRecord sorgt nun dafür, dass die Aufnahme komprimiert, die unkomprimierte Aufnahme bei Erfolg der Komprimierung gelöscht und in die Download.xml eingetragen wird. Schlägt die Komprimierung fehl, wird die unkomprimierte Version in die Download.xml eingetragen. Abschließend wird noch der Aufnahmetimer aus dem „timer_dic“ Dictionary entfernt.

Der WavRecorder basiert auf einem Beispiel von BLABLA (Quelle einfügen). Er sorgt dafür, dass ein Stream zum jeweiligen Aufnahmegerät geöffnet wird und schreibt die ankommenden Daten in eine Wav-Datei.

4.1.5 Battery-Service

Der Battery Service ist ein Windows Service, der die Aufgabe hat die Spannung einer externen Energieversorgung mittels eines USB-Voltmeters auszulesen und dem WCF-Service zur Verfügung zu stellen.

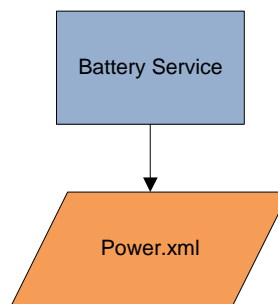


Abbildung 31: Kopplung Battery-Service

Der Service erstellt einen Timer, welcher alle fünf Minuten die Methode OnBatteryUpdate auslöst. Der aktuelle Spannungswert der externen Energieversorgung wird mittels der vom Hersteller zur Verfügung gestellten Funktion ausgelesen und in der Power.xml im Tag „<power>“ gespeichert.

4.1.6 ARM-Service

Der ARM Service ist eine Erweiterung des Systems von Marco Ziegert [13] und bindet Aufnahmegeräte auf ARM-Basis ein. Er greift ebenso wie der Record Service auf die Schedule.xml, Standby.xml, Config.xml, Compress.xml und Download.xml zu. Seine Aufgaben sind es relevante Aufnahmetimer an die entsprechenden ARM-Aufnahmegeräte weiterzuleiten, fertige Aufnahmen von den ARM-Aufnahmegegeräten zu übertragen, diese Aufnahmen mit WavPack zu komprimieren, anschließend in die Download.xml einzutragen und während Übertragungs-

vorgängen durch setzen eines aktuellen Zeitstempels in die Standby.xml den Rechner daran zu hindern in den Standby-Modus zu wechseln. Eine genauere Beschreibung ist in der Diplomarbeit von Marco Ziegert zu finden [13].

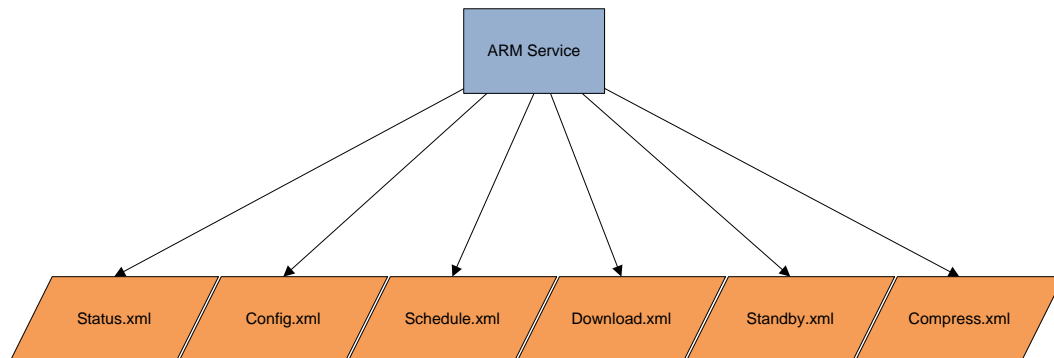


Abbildung 32: Kopplung ARM-Service

4.1.7 FTP-Service

Der FTP Service hat die Aufgabe fertige Aufnahmen zum Download per FTP bereitzustellen. Hierzu verwende ich die Open Source Software FileZilla Server (Quelle einfügen).

FileZilla Server ist ein Windows Service und startet automatisch im Hintergrund. Über die Konfigurations-Applikation kann der Server konfiguriert werden. Der in der Config.xml festgelegte Pfad für fertiggestellte Aufnahmen wird als Heimatverzeichnis des FTP-Servers angegeben. So sind die Aufnahmen an diesem Pfad über das FTP-Protokoll für den Benutzer erreichbar.

4.2 Benutzer-Komponente

Die Benutzer-Komponente stellt eine Management Console bereit, welche dem Benutzer eine einfache Möglichkeit auf das System zuzugreifen zur Verfügung stellt. Sie ist eine Windows Anwendung mit graphischer Benutzeroberfläche und bietet folgende Dienste an:

- Programmierung von Aufnahmetimern
- Löschen von Aufnahmetimern
- Löschen von Aufnahmen
- Download von Aufnahmen
- Setzen der Service Time
- Anzeige der geplanten Aufnahmen
- Anzeige von Zustandsdaten der Netbook-Komponenten
- Anzeige von verfügbaren Aufnahmegegeräten

Abbildung 33 zeigt den logischen Aufbau der Management Console. Sie kommuniziert mit der Netbook-Komponente über den angebotenen WCF-Service. Daten werden darüber hinaus auch in lokalen XML-Dateien gespeichert, da nicht zu jeder Zeit Kontakt zur Netbook-Komponente besteht. Die aktuellen Zeitpläne werden intern verwaltet (Schedule Verwaltung in Abbildung 33). Darauf aufbauend befindet sich die grafische Benutzerschnittstelle (User Interface in Abbildung 33).

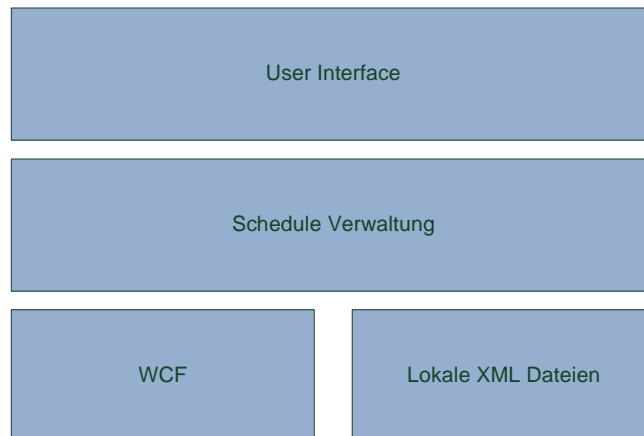


Abbildung 33: logischer Aufbau der Management Console

4.2.1 Architektur

Die Management Console ist eine Windows-Forms Anwendung (C#) und besteht aus drei Klassen. Zwei Windows Forms Klassen für die grafische Benutzerschnittstelle (MainUI und NewTimerUI) und eine Klasse für die Datenverarbeitung und Kommunikation mit entfernten Systemen (ClientBackend).

4.2.1.1 MainUI

Die Klasse MainUI stellt die grafische Benutzeroberfläche der Management Console bereit. Das Grundgerüst dieser Benutzeroberfläche ist ein TabControl mit drei Tabs: Connection, Timer und Service Time.

4.2.1.1.1 Connection-Tab

Im Tab Connection werden in einer Liste gefundene Rechner der Netbook-Komponente angezeigt. Bei einem Klick auf einen beliebigen Rechner werden im Feld Info einige Informationen zu diesem Rechner angezeigt: die Station ID, die Spannung der Stromversorgung und die lokale Systemzeit des entfernten Rechners. In einer weiteren Liste werden die verfügbaren Aufnahmegeräte aller gefundenen Rechner angezeigt. Als Name wird die ID der Aufnahmegerätes gewählt,

welche sich in der Status.xml auf dem entfernten Rechner befindet. Der Benutzer hat hier die Möglichkeit mit einem Rechtsklick ein Kontextmenü zu öffnen um diesen angezeigten Namen zu ändern. Diese Änderung hat allerdings keine Auswirkungen jenseits der Management Console. Sie wird nur lokal in einer XML-Datei abgelegt und ersetzt die Anzeige der ID des Aufnahmegeräts. Bei einem Neustart der Management Console werden diese Namen aus der XML-Datei gelesen. Darüber hinaus kann der Benutzer mit einem Klick auf „restore name“ im Kontextmenü den Originalnamen wiederherstellen.

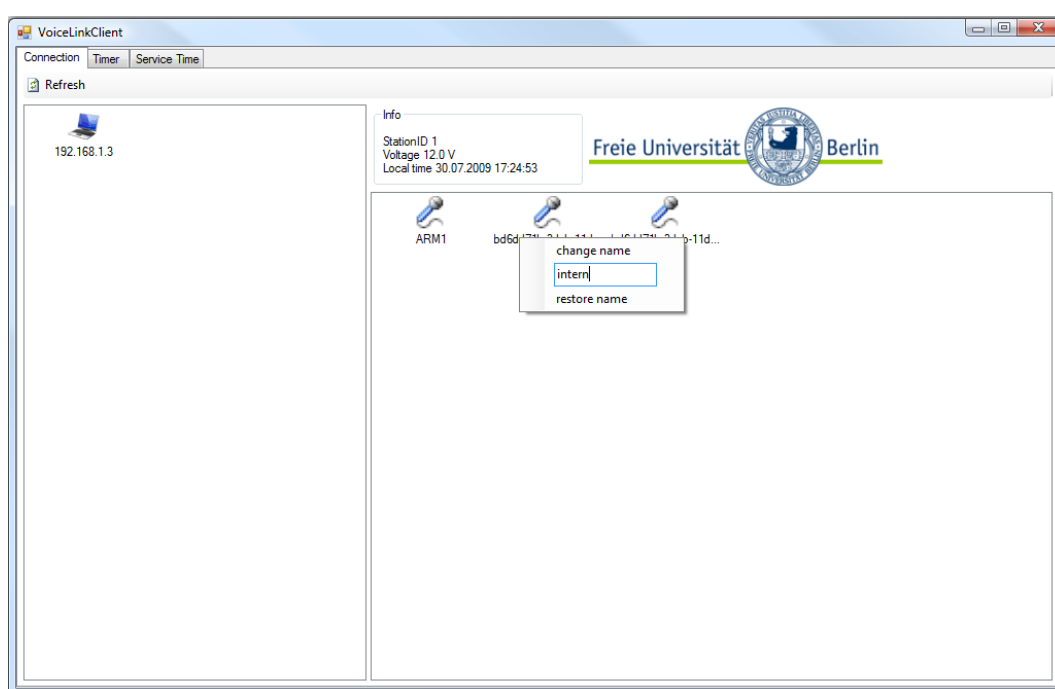


Abbildung 34: Änderung des Namens des Aufnahmegeräts

Der Button „Refresh“ bewirkt, dass die Management Console die Liste leert und neu nach verfügbaren Rechnern und Aufnahmegeräten sucht. Diese Suche wird asynchron von der ClientBackend-Klasse gestartet und Form1 registriert ein Event, welche ausgelöst wird, wenn ein Rechner mit zugehörigen Aufnahmegeräten gefunden wird. Im Eventhandler dieses Events werden dann die gefunden Geräte in die entsprechende Liste eingefügt.

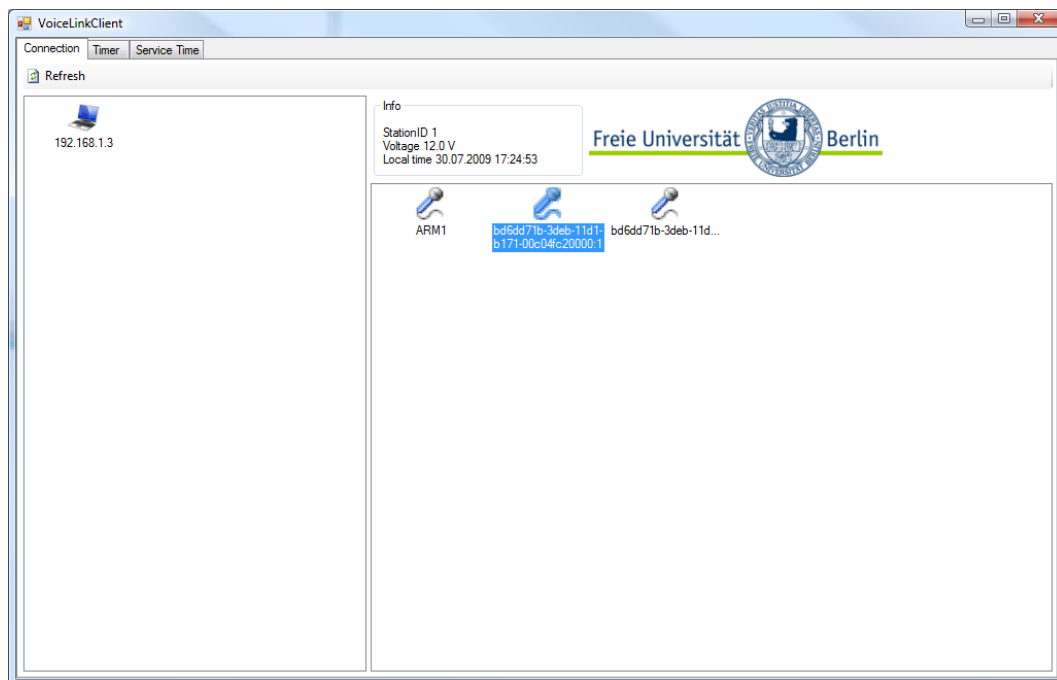


Abbildung 35: Connection Tab

4.2.1.1.2 Timer-Tab

Im Tab Timer werden die Aufnahmetimer verwaltet und fertige Aufnahmen heruntergeladen. Beim Start der Applikation wird die letzte bekannte Aufnahmetimerliste aus der lokalen Datei „Schedule.xml“ geladen und in der oberen Liste angezeigt. Alle Aufnahmetimer werden mit einem gelb schwarzen Achtungssymbol (siehe Abbildung 37) markiert um zu verdeutlichen, dass diese Liste möglicherweise veraltet ist, weil inzwischen von anderen Rechnern Aufnahmetimer erstellt und noch nicht auf diesen Rechner übertragen wurden. Mit einem Klick auf den Button Refresh verbindet sich die Management Console mit allen verfügbaren Rechner der Netbook-Komponente, fragt den neuesten Schedule ab und aktualisiert entsprechend die Aufnahmetimerliste. Die angezeigten Aufnahmetimer stimmen nun mit den aktuellen im System überein und die Markierung mit den gelb schwarzen Achtungssymbolen wird entfernt. Der Benut-

zer kann nun mit einem Klick auf den Button New einen neuen Aufnahmetimer erstellen. Es öffnet sich ein neues Fenster, welches von der Klasse NewTimerUI beschrieben wird. Eine weitere Möglichkeit neue Aufnahmetimer zu erstellen ist es, einen vorhanden Aufnahmetimer mit der rechten Maustaste anzuklicken. Es öffnet sich ein Kontextmenü mit der Option „use as template“. Mit einem Klick auf diese Option öffnet sich ebenfalls ein neues Fenster der Klasse NewTimerUI, in welchem allerdings die Daten des angeklickten Aufnahmetimers bereits eingetragen sind. So ist es für den Benutzer einfach ähnliche Aufnahmetimer wie bereits eingetragene zu erstellen.

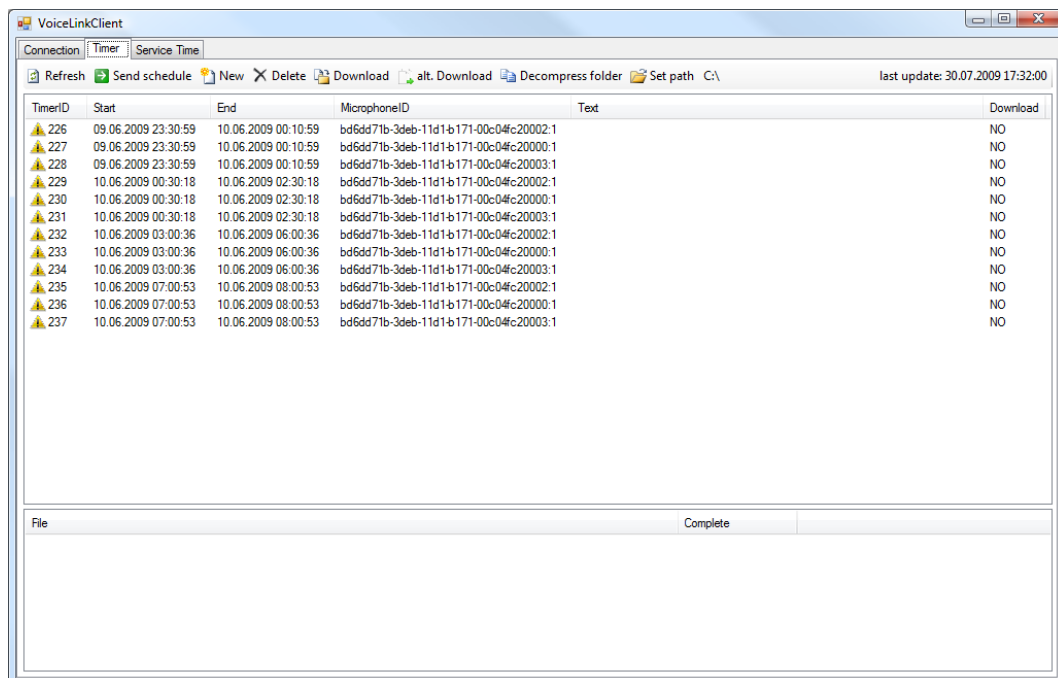


Abbildung 36: Timer-Tab, Schedule nicht synchronisiert

Neu erstellte Aufnahmetimer werden nicht automatisch an die Netbook-Komponente gesendet, sondern sind vorerst nur lokal auf diesem Rechner vorhanden. Dieser Zustand wird mit einem weiteren Symbol am jeweiligen Aufnahmetimer gekennzeichnet (siehe Abbildung 37). Erst mit einem Klick auf den Button Send Schedule wird der lokal angezeigte Schedule auf die Netbook-Komponente übertragen. Ist dies erfolgreich, erscheint ein Popup mit der Anzahl

der Rechner der Netbook-Komponente, die diesen neuen Schedule erhalten haben, und bei den neu erstellten Aufnahmetimern wird die oben beschriebene Markierung entfernt. Dieses Vorgehen hat folgende Vorteile: Zum einen wird der neue Schedule nur mit einem Klick auf den Button „Send schedule“ verschickt und nicht mit jeder Erstellung eines neuen Aufnahmetimers. Dadurch wird der Kommunikationsaufwand verringert. Zum anderen können Aufnahmetimer so ohne Verbindung zur Netbook-Komponente bereits erstellt und später, wenn der Benutzer in Reichweite der Netbook-Komponente ist, gesendet werden. Da das System im Feld eingesetzt werden soll, ist der Benutzer also nicht mehr gezwungen die Aufnahmetimer im Feld zu planen und zu programmieren, sondern kann dies bereits im Labor oder Büro tun.



Symbol	Verwendung
	Neuer Aufnahmetimer
	Nicht synchronisierter Aufnahmetimer

Abbildung 37: Verwendete Symbole [22]

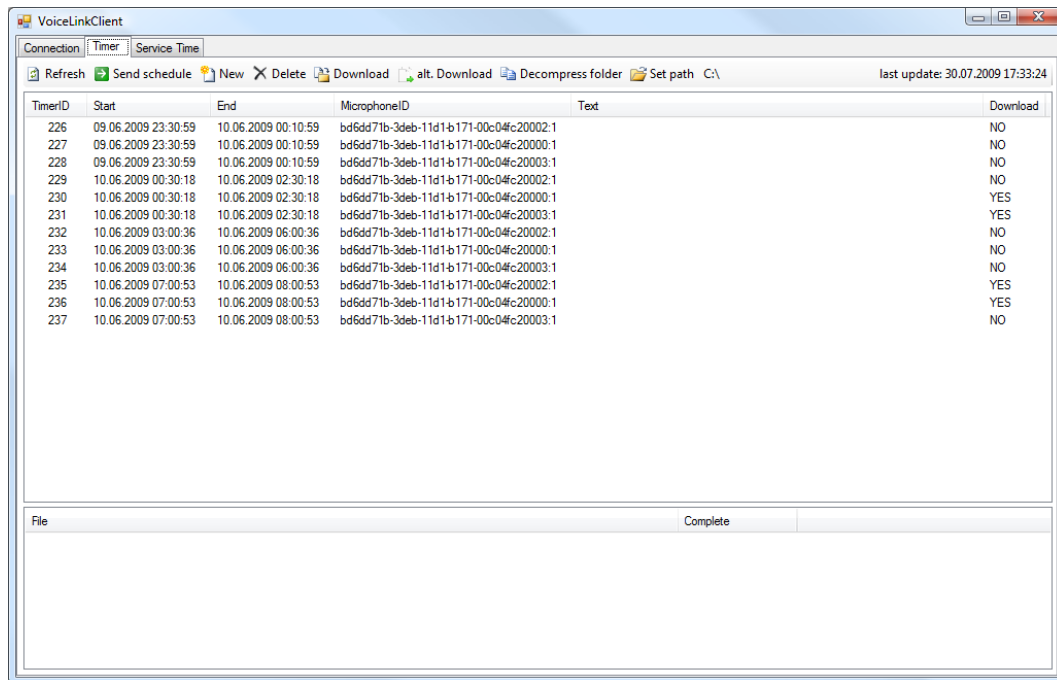


Abbildung 38: Timer-Tab, Schedule synchronisiert

Mit einem Klick auf den Button „Delete“ können ein oder mehrere zuvor in der oberen Liste markierte Aufnahmetimer gelöscht werden. Dieser Löschvorgang wird direkt auf die Netbook-Komponente übertragen, sofern eine Verbindung besteht.

Die Funktionen der Buttons „Refresh“ und „Delete“ werden von einem BackgroundWorker ausgeführt. Dadurch kann der Benutzer mit der Management Console weiterarbeiten, während diese Anfragen im Hintergrund bearbeitet werden.

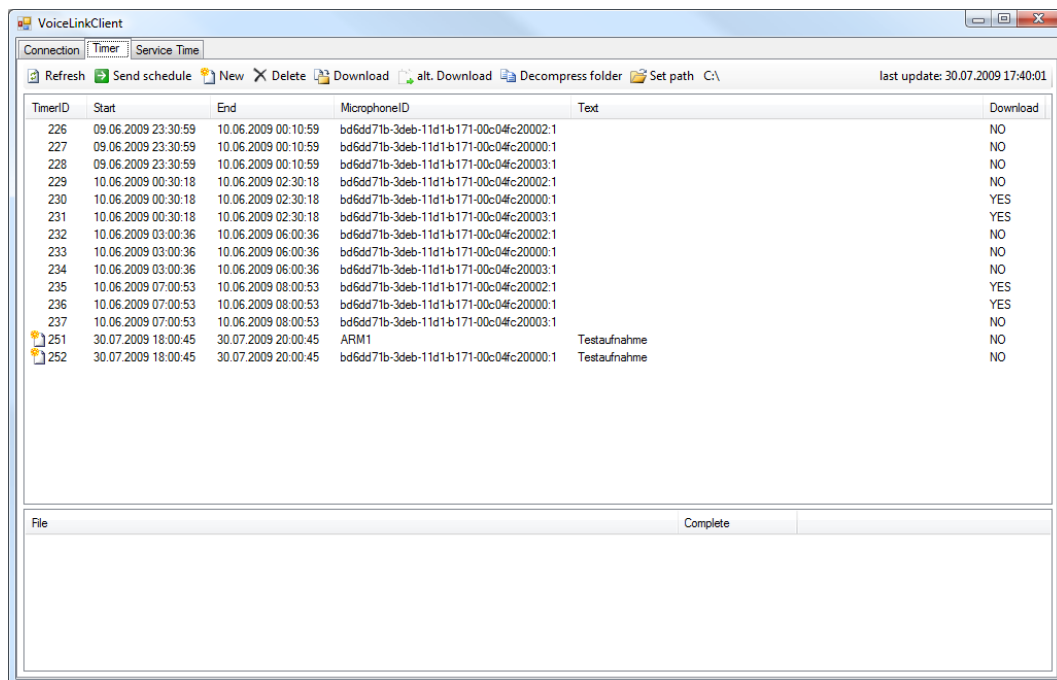


Abbildung 39: Timer-Tab, Schedule mit neuem Timer

Werden ein oder mehrere Aufnahmetimer markiert, für welche fertige Aufnahmen zum Download bereit stehen, und der Benutzer klickt auf den Button „Download“, so startet der Download der Aufnahmen der markierten Aufnahmetimer. In der unteren Liste werden die Dateien, welche nun übertragen werden, und der Status der Übertragung angezeigt. Die Aufnahmen werden in dem Ordner gespeichert, welcher im Label oben rechts angezeigt wird. Dieser kann durch Klick auf den Button „Set path“ geändert werden.

Der Button „alt. Download“ startet ebenso wie der Button „Download“ den Download der Aufnahmen der markierten Aufnahmetimer. Allerdings wird dieser Download vom Standardbrowser des Betriebssystems durchgeführt. Dies hat den Vorteil, dass der Download auch pausiert und später fortgesetzt werden kann, falls der Standardbrowser diese Funktion unterstützt.

Der Button „Decompress folder“ dekomprimiert alle WavPack Aufnahmen im vom Button „Set path“ gesetzten Pfad. Dafür wird ein Kommandozeilenprogramm von den Autoren von WavPack aufgerufen.

4.2.1.1.3 Service-Time-Tab

Im Service-Time-Tab kann der Benutzer die Service Time definieren. Voreingestellt bei Start der Management Console ist die letzte lokal bekannte Service Time. Ändert der Benutzer diese Werte mit Hilfe der beiden DateTimePicker und klickt auf den Button „Set service time“, so wird diese Service Time an alle verfügbaren Rechner der Netbook-Komponente übertragen. Diese Übertragung führt ein BackgroundWorker asynchron im Hintergrund aus, damit der Benutzer während der Übertragung die Management Console weiterhin bedienen kann.

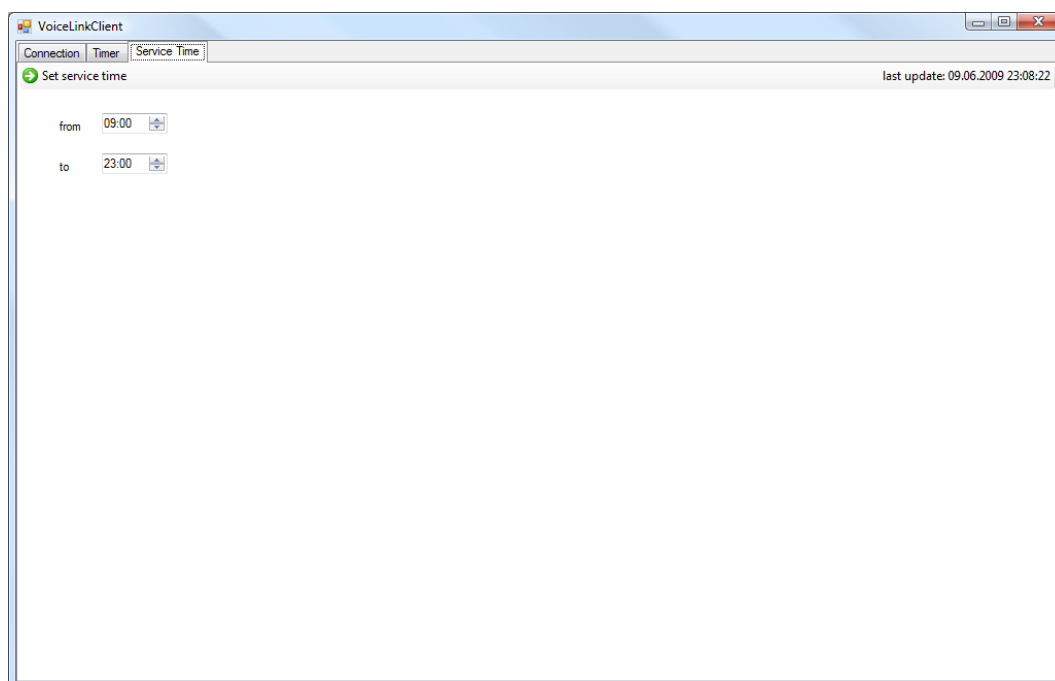


Abbildung 40: Service-Time-Tab

4.2.1.2 NewTimerUI

Die Klasse NewTimerUI stellt eine graphische Benutzeroberfläche zur Eingabe von neuen Aufnahmetimern bereit. Start- sowie Endzeit des Aufnahmetimers werden in den beiden DateTimePickern angegeben, in der Textbox kann ein zusätzlicher Freitext zu diesem Aufnahmetimer gespeichert werden und in der Liste kann der Benutzer ein oder mehrere Aufnahmegeräte auswählen. Auch hier besteht die Möglichkeit mit einem Rechtsklick ein Kontextmenü aufzurufen um den angezeigten Namen der Aufnahmegeräte lokal zu ändern. Die Liste zeigt allerdings nicht nur aktuell verfügbare Aufnahmegeräte an, sondern auch in der Vergangenheit verfügbare. Diese Informationen werden in der Datei „micro.xml“ zusammen mit den Namensänderungen der Aufnahmegeräte gespeichert. Dadurch ist es dem Benutzer möglich auch Aufnahmetimer für zur Zeit nicht verfügbare Aufnahmegeräte zu erstellen, was nötig ist, falls der Benutzer wie oben beschrieben im Labor oder Büro bereits Aufnahmetimer erstellen und diese im Feld an die Netbook-Komponente senden möchte.

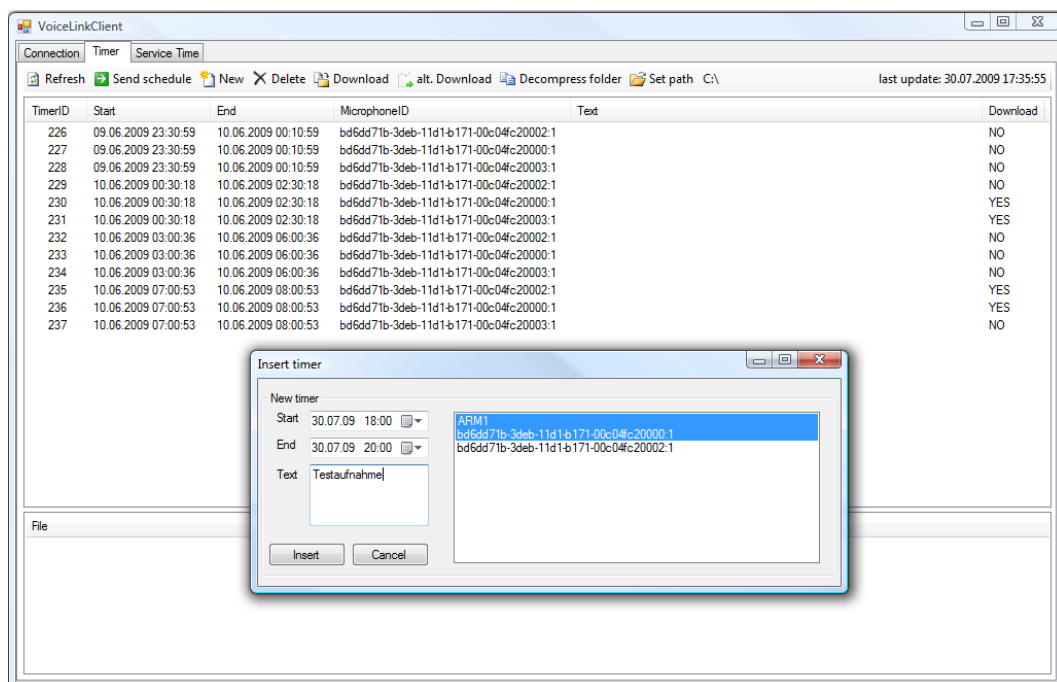


Abbildung 41: NewTimerUI

Mit einem Klick auf den Button „Insert“ wird der Aufnahmetimer erstellt und das Fenster geschlossen. Der Button „Cancel“ erstellt keinen Aufnahmetimer, sondern schließt nur das Fenster ohne Änderung des aktuellen Schedules.

4.2.1.3 ClientBackend

Die Klasse ClientBackend ist für die Kommunikation und Datenverarbeitung der Management Console verantwortlich. Sie stellt öffentliche Methoden bereit um alle Funktionen der Benutzeroberfläche auszuführen.

Eine Beschreibung der Methoden und deren Funktionsweise ist zu finden im Anhang, Kapitel 7.2.

5 Evaluation

In diesem Kapitel wird das System auf verschiedene Leistungsmerkmale getestet. Diese umfassen Laufzeit, Leistungsaufnahme, Komprimierungsgeschwindigkeit, Komprimierungsfaktor und Downloadperformance unter verschiedenen Bedingungen.

5.1 Leistungsaufnahme

5.1.1 Versuchsaufbau

Es wurde die Leistungsaufnahme der Netbook-Komponente in verschiedenen Zuständen gemessen. Dazu wurde das Netbook mit internen Akku an einem KFZ-Netzteil (Typ CLA-DC4.9521) und einer Autobatterie der Marke APS (12V 45Ah) betrieben, das Display war ausgeschaltet. Zwischen Netzteil und Autobatterie wurde ein Multimeter (Typ PeakTech 1050) als Amperemeter geschaltet.

5.1.2 Messwerte

Messwerte wurden, sofern nicht anders angegeben, mit zwei USB Soundkarten und aktiviertem WLAN Adapter gemessen.

Betriebsmodus	Messwert
Leerlauf, keine USB Soundkarte angeschlossen	0,96A
Leerlauf, eine USB Soundkarte angeschlossen	0,97A
Leerlauf, zwei USB Soundkarten angeschlossen	0,98A
Eine Aufnahme	1,01A
Zwei Aufnahmen	1,04A

Drei Aufnahmen	1,08A
Komprimierung von einer Aufnahme	1,10A
Komprimierung von zwei Aufnahmen	1,10A
Komprimierung von drei Aufnahmen	1,10A
Download von einer Aufnahme	1,05A
Download von zwei Aufnahmen	1,05A
Download von drei Aufnahmen	1,05A
Standby-Modus	0,23A
Ausgeschaltet	0,22A
Spannung bei Last	12,5V
Spannung bei Standby/Aus	12,6V

5.2 Download Performance

5.2.1 Versuchsaufbau

Es wurden von einem Notebook (Typ Apple Macbook Pro) eine Aufnahme von einem Netbook heruntergeladen. Dabei wurde die Geschwindigkeit gemessen, angezeigt vom Browser Mozilla Firefox.

5.2.2 Messwerte

Bei einer direkten WLAN Verbindung unter idealen Voraussetzungen (Entfernung 1m, keine anderen Netze auf diesem Kanal) wurde eine Geschwindigkeit von

2000kB/s erzielt. Bei einer Entfernung von ca. 50m betrug die Geschwindigkeit 350kB/s.

Bestand kein direkter Funkkontakt, sondern wurden die Daten über ein weiteres Netbook weitergeleitet, so betrug die Geschwindigkeit 170kB/s.

5.3 WavPack Geschwindigkeit

Es wurde die Zeit gemessen, die ein Netbook benötigt um eine, zwei oder drei sechsstündige Aufnahmen mit Wavpack zu komprimieren.

Anzahl Aufnahmen je 6 Stunden	Dauer der Komprimierung
1	0,19h
2	0,40h
3	0,63h

5.4 WavPack Komprimierungsfaktor

Es wurden fünf Testaufnahmen mit Nachtigallgesang mit Wavpack komprimiert. Die durchschnittliche Komprimierungsrate lag bei 55%.

5.5 Laufzeit

5.5.1 Versuchsaufbau

Die Laufzeit der Netbook-Komponente wurde mit einer handelsüblichen Autobatterie bestimmt. Dazu wurde das Netbook mit einem KFZ-Netzteil (Typ CLA-DC4.9521) an eine Batterie vom Typ APS 12V 45Ah angeschlossen. Da das System nicht dauerhaft eingeschaltet, sondern teilweise auch im Standby-Modus be-

trieben wird, wurde das Netbook in einem Testzyklus getestet. Ein Zyklus umfasst dabei 24 Stunden und zwei gleichzeitige Audioaufnahmen über die interne und eine externe Soundkarte, angeschlossen über den USB Extender Typ Digitus DA-70139-1. Die Audioaufnahmen laufen jeweils sechs Stunden, anschließend werden die Aufnahmen mit WavPack komprimiert und vom Benutzer heruntergeladen, wobei der Download ungefähr 15min für beide Aufnahmen dauert. Die übrige Zeit verbringt das System vollautomatisch im Standby-Modus. Dieser Versuch wurde einmal mit eingestecktem, internem Akku und einmal ohne diesen durchgeführt.

5.5.2 Theoretische Überlegungen

Basierend auf den Messwerten aus Kapitel XX ergibt sich folgender Energiebedarf für einen 24 Stunden Zyklus:

- 6 Stunden Aufnahme von zwei Aufnahmegeräten: $6h * 1,04A * 12,5V = 78,00Wh$
- Komprimierung der Aufnahmen: $0,4h * 1,10A * 12,5V = 5,5Wh$
- Download der Aufnahmen: $0,25h * 1,05A * 12,5V = 3,28Wh$
- Standby-Modus: $17,35h * 0,23A * 12,6V = 50,28Wh$

Es ergibt sich somit für einen 24 Stunden Zyklus ein theoretischer Energiebedarf von rund 137Wh. Damit müsste das System mit einer Batterie mit 45Ah bei 12V 3,9 Zyklen durchlaufen können.

5.5.3 Messwerte

Sowohl mit, als auch ohne internen Akku wurde der Versuch nach drei Zyklen abgebrochen, da die Spannung der externen Autobatterie auf 11,8V abgesunken war. Bei einem weiteren Absinken der Spannung bestünde laut Auskunft der Firma Varta die Gefahr eine Autobatterie irreparabel zu beschädigen.

Die Differenz von theoretischen 3,9 Zyklen und im Versuch ermittelten 3 Zyklen Laufzeit lässt sich dadurch erklären, dass die Autobatterie nicht die volle Kapazität zum Versuchszeitpunkt hatte, da sie schon mehrere Monate alt war und eventuell bereits tiefentladen wurde.

5.6 Zuverlässigkeit

Es wurden in etwa 400 Aufnahmen unterschiedlicher Länge erstellt. Dieser umfangreiche Test fand unter Laborbedingungen statt. Die Stromversorgung wurde sowohl mit der erwähnten Autobatterie, als auch mit mitgelieferten Netzteilen, die an das verfügbare Stromnetz angeschlossen wurden, realisiert.

100% dieser geplanten Aufnahmen wurden korrekt durchgeführt.

6 Zusammenfassung

In dieser Arbeit wurde ein System zur autonomen Aufzeichnung von Vogelgesang entworfen und umgesetzt. Das System besteht dabei aus einem Mesh-Netzwerk von mehreren Netbooks (Netbook-Komponente), welche über eine WLAN Schnittstelle gekoppelt sind, und einer Benutzer-Komponente zur Steuerung des Systems.

Die Netbook-Komponente führt vom Benutzer geplante Audioaufzeichnungen an angeschlossenen Aufnahmegeräten durch. Aufnahmegeräte können hierbei interne oder externe Soundkarten sein. Externe Soundkarten können über USB angeschlossen werden. Um die maximale Kabellänge von 5m bei USB Verbindungen zu erweitern, kann ein USB-Extender eingesetzt werden. Die Aufzeichnungen werden lokal auf den Netbooks gespeichert und durch das Wavpack-Verfahren komprimiert um anschließend vom Benutzer heruntergeladen werden zu können.

Die Netbooks verfügen über ein Powermanagement. Sofern möglich werden sie in den Standby-Modus versetzt um die Laufzeit des System zu steigern. Stehen Aufnahmen oder ein geplanter Zugriff vom Nutzer an, so reaktiviert sich das System automatisch.

Die Software der Netbook-Komponente besteht aus mehreren Microsoft .NET Diensten, die über eine XML-Schnittstelle kommunizieren. Die Verbindung zur Benutzerkomponente wird durch einen WCF-Service realisiert. Downloads werden über das FTP Protokoll durchgeführt.

Das System ist offen für Erweiterungen und stellt hierfür die XML-Schnittstelle bereit. Erweiterungs-Services können z.B. ebenfalls Aufnahmen auf direkt angeschlossenen Geräten oder entfernten Geräten durchführen bzw. auslösen.

Die Benutzer-Komponente besteht aus einem Benutzerprogramm zur einfachen Verwaltung des Systems. Es ist auf Basis von Microsoft .NET 3.5 implementiert und überall dort lauffähig, wo das entsprechende Framework installiert ist.

Der Benutzer kann sich verfügbare Netbooks der Netbook-Komponente und deren Aufnahmegeräte anzeigen lassen. Zusätzlich werden einige Informationen über die jeweiligen Komponenten angezeigt. Desweiteren kann der Benutzer Aufnahmen planen und diese an die Netbook-Komponente übertragen. Fertiggestellte Aufnahmen können heruntergeladen und dekomprimiert werden.

Ferner kann der Benutzer eine tägliche Zeitspanne festlegen, zu welcher die Netbook-Komponente verfügbar sein soll. So kann sichergestellt werden, dass Aufnahmen beispielsweise heruntergeladen werden können, auch wenn das System zu dieser Zeit keine Aufnahmen durchführt und sich daher eigentlich im Standby-Modus befinden würde.

Das System wurde umfangreich getestet. Mit einer Autobatterie betrug bei einem definierten 24-Zyklus die Laufzeit ca. 3 Zyklen. Ein Zyklus umfasst dabei zwei gleichzeitige sechsstündige Aufnahmen, deren Komprimierung und Download. Die übrigen Stunden verbrachte das System vollautomatisch im Standby-Modus.

Die Downloadgeschwindigkeit ist stark von der Qualität der Funkverbindung abhängig. Unter idealen Voraussetzungen sind 2 MB/s möglich. Werden die Daten nicht über eine Direktverbindung, sondern über das Mesh-Netzwerk über eine oder mehrere Zwischenstationen weitergeleitet, sinkt diese Geschwindigkeit beträchtlich. Bei einer Zwischenstation betrug diese ca. 170 kB/s.

6.1 Zukünftige Arbeiten

Das System könnte an einigen Stellen noch verbessert oder erweitert werden.

6.1.1 Verbesserungen

Das System weist unter den Testbedingungen eine Laufzeit von drei Tagen auf. Diese könnte durch den Einsatz optimierter Hardware erhöht werden. So könnten die Netbooks durch neuere Modelle mit geringerem Stromverbrauch getauscht werden. Werden hierfür kompatible Modelle ausgewählt, auf denen Windows XP lauffähig ist, so sind voraussichtlich keine Änderungen an der Software nötig. Sollte der Energiebedarf gering genug sein, wäre es auch möglich das System mit regenerativen Energien zu betreiben, z.B. durch Sonnenenergie.

Die Geschwindigkeit der Downloads bei Weiterleitung über Zwischenstationen (Multi-Hop) fällt bei einer Zwischenstation mit 170kB/s sehr gering aus. Dies könnte durch weitere WLAN-Adapter an den Netbooks verbessert werden. Die MCL Software unterstützt diesen Multi-Adapter-Modus. Dabei werden die einzelnen Adapter entweder für den Empfang oder zum Senden von Daten jeweils auf unterschiedlichen Kanälen genutzt. So kann eine Station gleichzeitig senden und empfangen, was mit einem Adapter nicht möglich ist. Die Geschwindigkeit soll sich nach [14] dabei stark erhöhen.

6.1.2 Erweiterungen

Mögliche Erweiterungen wären beispielsweise:

- Video-Aufzeichnungen, z.B. mit Infrarot-Kameras für Nachtaufnahmen
- Registrierung von Vögeln durch RFID
- Automatische Aufnahme, falls Gesang erkannt wird
- Temperatur- und Luftfeuchtigkeitmessungen

- Automatische Positionsbestimmung der Netbooks durch GPS und Anzeige in der Benutzerkomponente auf einer Karte
- Ansteuerung eines Voltmeters zur Spannungsmessung der externen Energieversorgung und Integration in den Battery-Service

7 Anhang

7.1 WCF Service API

7.1.1 Binding

Als Binding wird das „basicHttpBinding“ verwendet. Die Adresse, unter welchem der Service zu erreichen ist, lautet <http://IPADRESSE:8000/Service>, wobei IPADRESSE für die IP-Adresse des lokalen Rechners steht, über die man diesen erreichen kann.

7.1.2 Operations Contracts

7.1.2.1 PutSchedule

Die Methode PutSchedule überschreibt den aktuellen Schedule der Schedule.xml mit den übermittelten Daten. Als Argumente werden eine Liste vom Typ TimerType, ein Zeitstempel mit der aktuellen Zeit vom Typ DateTime und die bisher maximal vergebene TimerID vom Typ int erwartet. Als Rückgabewert liefert die Methode eine Zahl vom Typ int, wobei eine null zurückgegeben wird, falls keine Fehler auftraten, andernfalls eine eins.

7.1.2.2 GetSchedule

Die Methode GetSchedule erwartet keine Argumente und liefert als Rückgabewert eine Liste vom Typ TimerType mit allen Aufnahmetimern aus der lokalen Schedule.xml.

7.1.2.3 GetStatus

Die Methode GetStatus erwartet keine Argument und liefert als Rückgabewert eine Liste vom Typ StatusType mit allen Aufnahmegeräten aus der lokalen Status.xml.

7.1.2.4 GetTimestamp

Die Methode GetTimestamp erwartet keine Argumente und liefert als Rückgabewert den Zeitstempel vom Typ DateTime aus der Schedule.xml.

7.1.2.5 GetDownload

Die Methode GetDownload erwartet keine Argumente und liefert als Rückgabewert eine Liste vom Typ DownloadType. Die Elemente der Liste werden aus der Download.xml generiert.

7.1.2.6 DeleteDownload

Die Methode DeleteDownload erwartet als Argument eine TimerID. Alle Dateien mit dieser TimerID werden aus der Download.xml entfernt und gleichzeitig die zugehörigen Aufnahme-dateien gelöscht. Als Rückgabewert liefert die Methode eine Zahl vom Typ int, wobei eine null zurückgegeben wird, falls keine Fehler auftraten, andernfalls eine eins.

7.1.2.7 SetStandby

Die Methode SetStandby erwartet als Argumente zwei Zeitstempel vom Typ DateTime. Diese Zeitstempel werden in die Standby.xml eingetragen und definieren die Service Time. Als Rückgabewert liefert die Methode eine Zahl vom Typ int, wobei eine null zurückgegeben wird, falls keine Fehler auftraten, andernfalls eine eins.

7.1.2.8 GetMaxId

Die Methode GetMaxId erwartet keine Argumente und liefert als Rückgabewert eine Zahl vom Typ int, welche die bisher maximal vergebene TimerID darstellt. Diese ID wird aus der Schedule.xml aus dem zugehörigen Tag gelesen.

7.1.2.9 KeepAlive

Die Methode KeepAlive erwartet keine Argumente. Sie setzt in der Standby.xml einen aktuellen Zeitstempel im zugehörigen Tag, so dass der Rechner am Wechsel in den Standby-Modus vorerst gehindert wird. Als Rückgabewert liefert die Methode eine Zahl vom Typ int, wobei eine null zurückgegeben wird, falls keine Fehler auftraten, andernfalls eine eins.

7.1.2.10 GetInfoString

Die Methode GetInfoString erwartet keine Argumente und liefert als Rückgabewert einen String, welcher aus der ID des lokalen Rechners, der Spannung der Stromversorgung, welche aus der Power.xml gelesen wird, und der lokalen Systemzeit besteht.

7.1.3 Data Contracts

7.1.3.1 TimerType

TimerType ist der Datentyp, der einen Aufnahmetimer repräsentiert. Er besteht aus den zwei DateTime Variablen (start, end) die den Start- und Endpunkt des Aufnahmetimers definieren, einem String mit der GUID des Aufnahmegerätes, einem Integer mit einer eindeutigen ID, die diesen Aufnahmetimer identifiziert, und einem weiteren String (text), der den vom Benutzer eingegebenen Freitext zur Beschreibung des Aufnahmetimers speichert.

7.1.3.2 StatusType

StatusType ist der Datentyp, der ein Aufnahmegerät repräsentiert. Er besteht aus einem String (id), welcher die GUID für das Aufnahmegerät darstellt, einem Integer (type), der den Typ des Aufnahmegerätes beschreibt, wobei eine lokales Stan-

ardaufnahmegerät vom Typ 1 ist, und einem weiteren String (optional) für weitere Informationen von zukünftigen Erweiterungen.

7.1.3.3 DownloadType

DownloadType ist der Datentyp, der Informationen einer Aufnahme datei liefert. Er besteht aus zwei Integer (chunkID, id), welche zusammen die Datei eindeutig identifizieren, wobei „id“ den zugehörigen Aufnahmetimer und „chunkID“ das jeweilige Bruchstück (chunk) identifizieren, und einem String (url), welcher den Pfad zu dieser Datei liefert.

7.2 Management Console API

7.2.1 ClientBackend Kontruktor

ClientBackend ist der Konstruktor der Klasse. Hier werden einige Datenstrukturen und zwei Timer erstellt.

7.2.2 OnKeepAliveTimer

On KeepAliveTimer ist die Methode, die aufgerufen wird, falls der Timer keepalive ausgelöst wird. Über die WCF Schnittstelle wird bei allen verfügbaren Rechnern der Netbook-Komponente die Methode KeepAlive aufgerufen. Dadurch wechseln die Rechner der Netbook-Komponente nicht in den Standby-Modus solange dieser Timer aktiv ist.

7.2.3 saveSchedule

Die Methode saveSchedule serialisiert die Liste der Aufnahmetimer und schreibt sie in eine lokale XML-Datei (Schedule.xml).

7.2.4 saveMaxID

Die Methode saveMaxID serialisiert die maximale ID, die bisher für einen Aufnahmetimer vergeben wurde, und speichert diese in der lokalen Datei MaxId.xml.

7.2.5 loadSchedule

Die Methode loadSchedule deserialisiert den Inhalt der lokalen Datei Schedule.xml und schreibt die so erzeugte Liste von Aufnahmetimern in die dafür vorgesehene Datenstruktur (timerlist). Zusätzlich wird die Systemzeit des letzten Schreibzugriffs zurückgegeben.

7.2.6 loadMaxID

Die Methode loadMaxID deserialisiert den Inhalt der lokalen Datei MaxId.xml und schreibt die gelesene ID in die lokale Variable „maxID“. Ferner wird überprüft, ob es eine höhere ID in der aktuellen Aufnahmetimerliste existiert. Ist dies der Fall, so wird „maxID“ entsprechend erhöht.

7.2.7 saveServiceTime

Die Methode saveServiceTime serialisiert den Start- und Endpunkt der Service Time und speichert diese in der lokalen Datei ServiceTime.xml.

7.2.8 loadServiceTime

Die Methode loadServiceTime deserialisiert den Inhalt der lokalen Datei ServiceTime.xml und gibt die gelesene Start- und Endzeit und die Systemzeit des letzten Schreibzugriffs auf diese Datei zurück.

7.2.9 getSchedule

Die Methode getSchedule gibt die Datenstruktur “timerlist” zurück. „timerlist“ enthält die aktuellen Aufnahmetimer.

7.2.10 updateSchedule

Die Methode updateSchedule fragt über den WCF-Dienst alle verfügbaren Rechner der Netbook-Komponente nach dem Zeitstempel ihrer Schedule.xml mit Hilfe der WCF-Methode GetTimestamp. Von dem Rechner, der den aktuellsten Zeitstempel liefert, wird nun mittels der WCF-Methode GetSchedule der aktuelle Schedule angefordert. Dieser Schedule wird in der Datenstruktur „timerlist“ gespeichert und zurückgegeben.

7.2.11 GetMicroName

Die Methode GetMicroName sucht für einen übergebenen String im Dictionary „micro_dic“ den zugehörigen Namen des Aufnahmegerätes und gibt diesen zurück.

7.2.12 DeleteTimer

Die Methode DeleteTimer erzeugt eine neue Aufnahmetimerliste ohne die Aufnahmetimer, welche gelöscht werden sollen. Diese Aufnahmetimerliste wird dann mit einem aktuellen Zeitstempel per WCF-Methode PutSchedule an die Rechner der Netbook-Komponente weitergegeben.

7.2.13 SendTimer

Die Methode SendTimer trägt einen oder mehrere neue Aufnahmetimer in der zugehörigen Datenstruktur (timerlist) ein. Die maximal vergebene Aufnahmetimer-ID erhöht sich entsprechend und wird mit der Methode saveMaxID gespeichert.

7.2.14 sendSchedule

Die Methode sendSchedule sendet die lokale Aufnahmetimerliste (timerlist) mit einem aktuellen Zeitstempel mittels der WCF-Methode PutSchedule an alle verfügbaren Rechner der Netbook-Komponente.

7.2.15 GetStatus

Die Methode GetStatus liefert eine Liste von Strings mit den IDs aller Aufnahmegeräte, die über die WCF-Methode GetStatus von allen verfügbaren Rechnern der Netbook-Komponente gesendet werden. Ist für die jeweilige ID im Dictionary micro_dic ein Eintrag vorhanden, so wird der gespeicherte Name für dieses Aufnahmegerät zurückgegeben. Zusätzlich werden neu gefundene Aufnahmegeräte im Dictionary micro_dic eingetragen und durch die Methode WriteMicXML gespeichert.

7.2.16 Getmaxid

Die Methode `getmaxid` liefert die maximal vergebene Aufnahmetimer-ID. Dazu wird die WCF-Methode `GetMaxId` auf allen verfügbaren Rechnern der Netbook-Komponente aufgerufen. Der maximale Wert, den die WCF-Methode liefert, wird zurückgegeben.

7.2.17 Scan

Die Methode `Scan` erwartet zwei IP-Adressen als Argumente, die einen Adressbereich definieren. Für jede IP-Adresse in diesem Adressbereich startet `Scan` einen neuen Thread, welcher die Methode `checkIP` ausführt.

7.2.18 checkIP

Die Methode `checkIP` öffnet einen Kommunikationskanal über die WCF-Schnittstelle zu einer übergebenen IP-Adresse. Daraufhin werden die WCF-Methoden `GetTimestamp`, `GetInfoString` und `GetStatus` ausgeführt. Schlagen diese Aufrufe nicht fehl, so wird diese Verbindung in einer Liste gespeichert und das Event `OnServiceFound` ausgelöst, welchem die durch die WCF-Methoden gewonnenen Rückgabewerte als Argument übergeben werden.

7.2.19 OnServiceFound

`OnServiceFound` löst das nach außen sichtbare Event `ServiceFound` aus.

7.2.20 Download

Die Methode `Download` startet asynchron den Download der Aufnahmedateien, welche der übergebenen ID zugeordnet werden. Da bei einer Übertragung über FTP im passiven Modus die Dateigröße nicht bestimmt werden kann, wird ein Webrequest zu dieser Datei durchgeführt und deren Größe gespeichert. Ändert sich der Status der Übertragung, wird das Event `ftp_DownloadProgressChanged`

ausgelöst. Ist die Übertragung vollständig, wird das Event `ftp_DownloadFileCompleted` ausgelöst.

7.2.21 ftp_DownloadProgressChanged

`ftp_DownloadProgressChanged` delegiert das Event an `OnDownloadStatusUpdated`.

7.2.22 ftp_DownloadFileCompleted

`ftp_DownloadFileCompleted` delegiert das Event an `OnDownloadComplete`.

7.2.23 OnDownloadStatusUpdated

`OnDownloadStatusUpdated` löst das nach außen sichtbare Event `DownloadStatusUpdated` aus.

7.2.24 OnDownloadComplete

`OnDownloadComplete` löst das nach außen sichtbare Event `DownloadComplete` aus.

7.2.25 OnDownloadCheck

`OnDownloadCheck` ist die Methode, die aufgerufen wird, falls der Timer `downloadTimer` ausgelöst wird. Falls der letzte Aufruf des `ftp_DownloadProgressChanged` Events länger als 120 Sekunden in der Vergangenheit liegt und die Anzahl der aktiven Downloads größer null ist, so werden alle aktiven Downloads abgebrochen. Dies verhindert, dass Downloads, welche zwar aktiv sind, aber keine neuen Daten bekommen, endlos weitergeführt werden.

7.2.26 Download2

Die Methode Download2 öffnet die Aufnahme Dateien, welche der übergebenen ID zugeordnet werden, mit dem Standardprogramm des Rechners. Im Allgemeinen sollte dies ein Browser oder Downloadmanager sein, welche dann den Download durchführen.

7.2.27 getDownloadList

Die Methode getDownloadList gibt die Datenstruktur "downloadlist" zurück. „downloadlist“ enthält die Pfade zu allen verfügbaren Aufnahme Dateien auf den Rechner der Netbook-Komponente.

7.2.28 refreshDownloadList

Die Methode refreshDownloadList ruft auf allen verfügbaren Rechner der Netbook-Komponente die WCF-Methode GetDownload auf und speichert die so gelieferten Pfade zu verfügbaren Aufnahme Dateien in der Datenstruktur „downloadlist“. Zusätzlich wird „downloadloadlist“ zurückgegeben.

7.2.29 DeleteDownload

Die Methode DeleteDownload löscht Aufnahme Dateien von Rechnern der Netbook-Komponente mit Hilfe der WCF-Methode DeleteDownload.

7.2.30 SetStandby

Die Methode SetStandby setzt die Service Time auf Rechnern der Netbook-Komponente mit Hilfe der WCF-Methode SetStandby.

7.2.31 WriteMicXML

Die Methode WriteMicXML schreibt alle Einträge im Dictionary „micro_dic“ die die Datei „micro.xml“.

7.2.32 ReadMicXML

Die Methode ReadMicXML liest alle Einträge aus der Datei „micro.xml“ und speichert diese im Dictionary „micro_dic“.

7.2.33 RenameMic

Die Methode RenameMic ändert den Namen eines Aufnahmegerätes im Dictionary „micro_dic“ oder fügt einen neuen Eintrag für diese Aufnahmegerät hinzu, falls es für dieses keinen Eintrag gibt. Anschließend wird zum Speichern der Änderung die Methode WriteMicXML aufgerufen.

7.2.34 getMicros

Die Methode getMicros führt die Methode ReadMicXML aus und gibt für alle Aufnahmegeräte die eventuell geänderten Namen als String-Array zurück.

7.2.35 Decompress

Die Methode Decompress startet das Kommandozeilenprogramm „wvunpack.exe“. Durch die übergebenen Parameter werden alle WavPack Dateien am Pfad, der dieser Methode übergeben wurde, entpackt.

7.2.36 restoreMicName

Die Methode restoreMicName ändert einen eventuell vorhandenen, geänderten Namenseintrag für ein Aufnahmegerät im Dictionary „micro_dic“ in die ursprüngliche ID dieses Aufnahmegeräts.

8 Literaturverzeichnis

1. **Freie Universität Berlin.** Lernstrategien und adaptiver Wert des Gesang bei Nachtigallen. [Online] [Cited: 23 06 2009.] <http://www.biologie.fu-berlin.de/verhaltensbiologie/forschung/nachtigallenforschung/index.html>.
2. *Autonomous Monitoring of Vulnerable Habitats using a Wireless Sensor Network.* **Naumowicz, Tomasz, et al.** Workshop on Real-World Wireless Sensor Networks In conjunction with ACM EuroSys 2008, Pages 51-55, Glasgow, UK, 1. Apr, 2008 : s.n., 2008.
3. **Beier, Frank and Ziegert, Marco.** *MSB-430H: Implementierung eines Audiolinks.* Freie Universität Berlin : s.n., 2008.
4. *Automated wildlife monitoring using self-configuring sensor networks deployed in natural habitats.* **Trifa, Vlad M., et al.** 2007. International Symposium on Artificial Life and Robotics (AROB07). p. Article 1131.
5. **Chen, C. E., et al.** Design and testing of robust acoustic arrays for localization and beamforming. s.l. : In IEEE IPNS, 2006.
6. **Juang, Philo, et al.** Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *SIGARCH Comput. Archit. News.* 2002, Vol. 30, 5, pp. 96-107.
7. **Zhang, Pei.** ZebraNet Deployment. [Online] 2007. [Cited: 30 04 2009.] <http://www.peizhang.com/research/research/research.htm#zebranet>.
8. *Hardware design experiences in ZebraNet.* **Zhang, Pei and Sadler, Christopher M. and Lyon, Stephen A. and Martonosi, Margaret.** Baltimore, MD, USA : s.n., 2004. SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems. pp. 227-238. 1-58113-879-2.

9. **Texas Instruments.** MSP430F1612 datasheet (slas368e) and user's guide (slau049f). Dallas : s.n., 2006.
10. **Freie Universität Berlin.** ScatterWeb. [Online] [Cited: 20 04 2009.] <http://cst.mi.fu-berlin.de/projects/ScatterWeb/hardware/msb/index.html>.
11. **PC Engines GmbH.** PC Engines custom embedded PC hardware and firmware design. [Online] [Cited: 20 04 2009.] <http://www.pcengines.ch/alix2c2.htm>.
12. **Freie Universität Berlin.** DES-Testbed. [Online] [Cited: 10 05 2009.] <http://www.des-testbed.net>.
13. **Ziegert, Marco.** Autonomous Wildlife Monitoring: Design, implementation, and evaluation of an embedded platform for nightingale song recording. Berlin : Freie Universität Berlin, 2009.
14. **Microsoft Research.** Self Organizing Wireless Mesh Networks. [Online] [Cited: 10 4 2009.] <http://research.microsoft.com/en-us/projects/mesh/>.
15. **Kuhrmann, Marco and Beneken, Gerd.** *Windows Communication Foundation: Konzepte - Programmierung - Migration.* München : Elsevier GmbH, Spektrum Akademischer Verlag, 2007. 978-3-8274-1598-1.
16. **Microsoft.** MSDN Library. [Online] [Cited: 22 04 2009.] <http://msdn.microsoft.com/de-de/library/default.aspx>.
17. **Konerow, Jens.** *Managed DirectX und C#.* Frankfurt am Main : Entwickler.Press, 2007. 3-935082-17-4.
18. **Microsoft.** Microsoft Developer Network: DirectSound. [Online] [Cited: 4 9 2009.] [http://msdn.microsoft.com/en-us/library/bb219818\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb219818(VS.85).aspx).

19. **Draves, Richard, Padhye, Jitendra and Zill, Brian.** Comparison of routing metrics for static multi-hop wireless networks. *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. Portland, Oregon, USA : ACM, 2004, pp. 133-144.
20. **Johnson, David B., Maltz, David A. and Broch, Josh.** DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*. s.l. : Addison-Wesley, 2001.
21. **WavPack.** WavPack Audio Compression. [Online] [Cited: 15 4 2009.] <http://www.wavpack.com>.
22. **Microsoft.** *Visual Studio 2008 Image Library*.