Endbericht

PG466

Beyond KaZaA: Peer-to-Peer Systeme für mobile Endgeräte mit Bluetooth und UMTS Technologie

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Teilnehmer

BETREUER: Prof. Dr.-Ing. Christoph Lindemann, Dipl. Inform. Christian Lambert, Dipl. Inform. Oliver Waldhorst

STUDENTEN: Nabil Ben Said, Chris Börgermann, Michael Diel, Carmen Göbel, Gregor Kasmann, Boris Kißner, Nicolas Krokowski, Paul Leikam, Sebastian Prost, Alain Tigyo, David Zok

2 Motivation und Zielanpassung

2.1 Motivation und erste Zielsetzung

Immer mehr mobile Geräte verfügen heutzutage über Bluetooth. Bluetooth operiert im lizenzfreien ISM[1]-Band, deshalb ist die Benutzung kostenfrei. Obwohl Bluetooth nur eine Reichweite von zehn[2] Metern besitzt, ist es in der Lage, bis zu acht Geräte in einem Netzwerk zu verbinden. Dadurch, dass einzelne Netzwerke zu größeren Scatternetzen zusammengeschlossen werden können, steigert sich die Reichweite und die Kapazität.

Peer-to-Peer (P2P) Filesharing Programme haben sich auf breiter Basis durchgesetzt, weil sie die epidemieartige Verbreitung von beliebten Dateien einfach erlauben. Programme wie KaZaA, eDonkey o. Ä. kennt jeder, der öfters Dateien über das Internet austauscht.

Wie wäre es also, wenn man diese beiden Technologien miteinander verschmelzen würde?
Genau das ist die Motivation dieser Projektgruppe. Ziel ist es, ein Programm zu entwickeln, das auf mobilen Endgeräten (sprich Handys) laufen kann und es erlaubt, Dateien auszutauschen. Dazu muss das Programm eine Reihe von Funktionen unterstützen, die sich aus dieser Zielbeschreibung zwangsläufig ergeben: angefangen von der Möglichkeit, Dateien auf dem festen Speicher des Endgerätes selektiv freizugeben, über eine intelligente Suchfunktion bis hin zu einer effizienten Ausnutzung der vorhanden Technologie im Hinblick auf Zeit und Datenraten.
Bluetooth-Netzwerke unterstützen höchstens acht aktive Endgeräte und sind deshalb *per se* eigentlich ungeeignet für eine Filesharing-Applikation. BT bietet jedoch die Möglichkeit, mehrere dieser Piconetze zu verbinden und damit ein sogenanntes Scatternetz zu erzeugen. Die Algorithmen, die dabei verwendet werden, genauso wie Algorithmen zur effizienten Verbreitung von Informationen (Dateien im Netzwerk) waren ebenso Ziele der PG. Im Falle der Scatternetz-Algorithmen sollte der ns2-Netzwerksimulator zum Einsatz kommen.

2.2 Revision der Ziele

Nach der Seminarphase wurden die ersten der folgenden Sitzungen dazu verwendet, die Möglichkeiten und angestrebten Features des finalen Produktes zu konkretisieren. Der generelle Konsens war dabei ein P2P-Filesharing-Programm zu entwickelt, welches auf normalen Mobiltelefonen ausgeführt werden sollte. Ferner war zunächst festgelegt, dieses in JAVA[3] zu programmieren. Das Programm sollte folgende Funktionen haben bzw. unterstützten:

- Austausch von Dateien beliebigen Typs und Länge durch eine graphische Oberfläche
- Vernetzen von mehr als acht Endgeräten in einem großen P2P-Netz
- Suche von Dateien innerhalb des Supernetzes (Scatternetzes) an Hand von Schlüsselworten

[1] **I**ndustrial **S**cientific **M**edical: 2,4000 - 2,4835 GHz mit 2 MHz lower bound und 3,5 MHz upper bound Sicherheitsgrenzen (Frequenzangabe spezifisch für Deutschland, andere Länder haben unter Umständen eine kleinere Bandbreite).
[2] Neuere BT-Versionen verfügen über eine größere Reichweite (bis zu 100 Metern).
[3] Es wurde für kurze Zeit überlegt, in C++ für *Symbian* zu programmieren (siehe 5.3, S.17 und 6, S.21), am Ende dann aber doch wieder zu JAVA zurückgekehrt.

- Die Möglichkeit, nach einer erfolglosen Suche auf einen persistenten (und kostenpflichtigen) Server im Internet zuzugreifen, um die gewünschten Dateien von dort zu beziehen

Im Verlauf der Projektgruppe und nach tiefgehender Evaluation der zur Verfügung stehenden APIs und weiterführenden Materialen, mussten die Ziele leicht angepasst werden (siehe 5.3, S.17 und 5.6, S.19).

Aufgrund der Kapselung der Bluetooth Verbindungsprozesse durch `acceptAndOpen()` war es nicht möglich, direkten Einfluss auf die Formierung von Pico- und Scatternetzen zu nehmen (siehe auch 6, S.21). Daraus folgte natürlich die (vorläufige) Unerfüllbarkeit eines der ursprünglichen Ziele, in dem eine Optimierung der Verbindungsprozesse gefordert war (effizienter Aufbau und Verwaltung von Scatternetzen, Scheduling mit ns2-Netzwerksimulator).

Die neue Zielsetzung forderte als Mindestziele für das erste Semester Folgendes:

- Ein lauffähiges Programm
- Die Erfüllung aller unter 2.2 aufgeführten Ziele
- Das Erstellen eines Zwischenberichtes

2.3 Weiterführende Ziele im zweiten Semester

Im zweiten Semester wurden Ziele aufgestellt, die teilweise den Abschluss bereits begonnener Arbeit beinhalteten, wie auch völlig neue, der PG-Beschreibung zuträgliche. Die endgültigen Ziele im zweiten Semesters waren wie folgt:

- Implementierung eines Scatternetz-Algorithmus auf logischer Ebene (wg. API)
- Fertigstellung des Backup-Servers
- Verbesserte Suche und das Auslesen von ID3-Tags bei bestimmten Dateien
- Entwicklung eines abgespeckten Programmes, zum Testen auf realen End-geräten
- Erstellen eines Endberichts

3 Installation und Benutzungsanleitung

3.1 Installation auf einem mobilen Endgerät

Um Blue Brewery Horse (BBH) auf einem mobilen Endgerät zu installieren, müssen lediglich entweder die „Programme.jad“ oder die „Programm.jar“ (vom Endgerät abhängig) auf das Gerät kopiert werden. Gestartet wird es dann durch einen einfachen Aufruf.

3.2 Installation auf dem Desktop-PC

Das Programm besteht aus zwei Dateien, „Programm.jad“ und „Programm.jar“. Folgende Schritte müssen ausgeführt werden, damit das Programm auf einem Rechner mit WTK 2.2 installiert und getestet werden kann:

1. Folgende Einstellungen im WTK vornehmen: CLDC 1.1, JSR 135 (120), JSR 75 und JSR 82 aktivieren
2. Dateien in ein beliebiges Verzeichnis kopieren
3. In das [WTK]/bin/-Verzeichnis wechseln
4. „runmidlet" aufrufen
5. In dem nun folgenden Dialog, die Datei „Programm.jad" wählen
6. Prozess kann zum Simulieren weiterer Geräte iteriert werden.

Das root-Verzeichnis der simulierten Geräte ist dann „WTK\AppDB\...".

3.3 Benutzungsanleitung

3.3.1 Programm starten

Bevor das Programm benutzt werden kann, wird abgefragt, ob ihm verschiedene Rechte zugestanden werden sollen (Siehe Abb. 1, Seite 4). Diese Abfragen müssen unbedingt mit „Ja“ beantwortet werden, da sonst das Programm nicht richtig funktioniert. Drücken Sie einfach auf den entsprechenden Knopf.

Jetzt sehen Sie den Splash-Bildschirm, von dem aus sie mit einem Druck auf „Done“ ins Hauptmenü kommen können.

3.3.2 Das Hauptmenü

Das Hauptmenü ist unterteilt in sieben Menüpunkte:

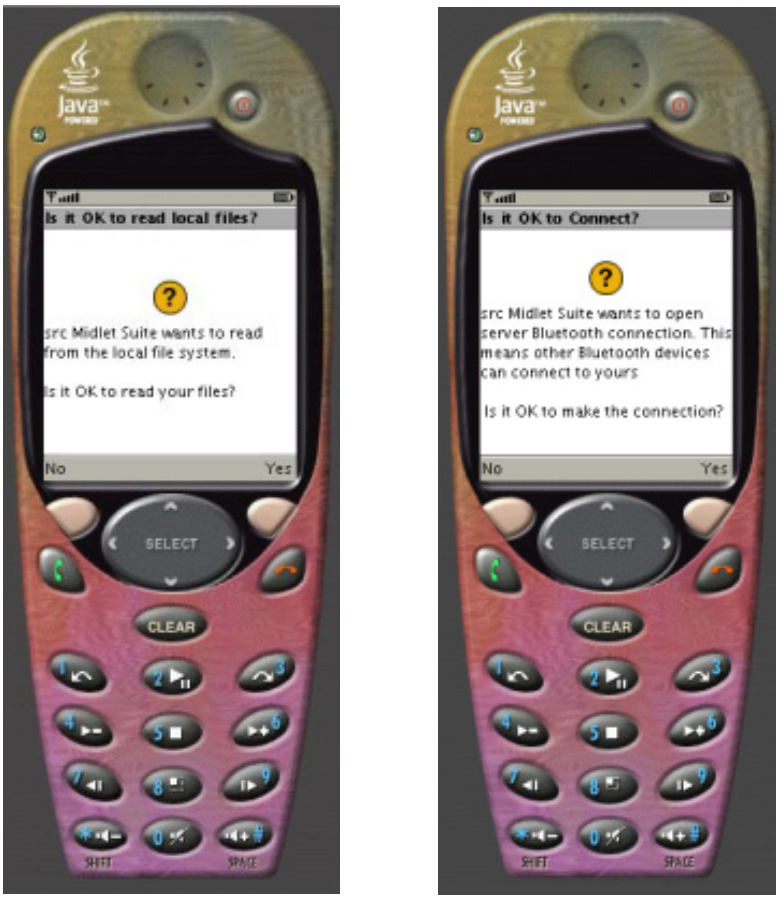

Abbildung 1: Schreibrechte müssen gewährt werden

FREIGABELISTE ANZEIGEN	Zeigt die Freigabeliste an, in der alle Dateien enthalten sind, die für den Upload freigegeben sind. Dateien, die durch das Programm heruntergeladen wurden, werden automatisch freigegeben und damit dieser Liste hinzugefügt.
DATEIEN FREIGEBEN	Erlaubt es, den Speicher des Endgerätes zu durchsuchen und Dateien zu markieren, die zum Upload freigegeben werden sollen.
SUCHEN	Dient dazu, das Netz nach Dateien zu durchsuchen. Für die Suche können Wörter und- oder oder-verknüpft werden. MP3-Dateien werden zusätzlich nach ihren ID3-Tags durchsucht.
DOWNLOAD	Hier werden alle momentan laufenden Downloads angezeigt. In dieses Untermenü muss man auch wechseln, wenn man einen Download starten möchte, der nicht direkt mit „Download direkt starten“ begonnen wurde. Downloads können hier angehalten, fortgesetzt und auch beendet werden.
UPLOAD	Zeigt an, welche Dateien im Augenblick an andere Teilnehmer verschickt werden. Hier kann man auch einzelne Uploads abbrechen.
GERAETE	Listet alle verbundenen Geräte in Reichweite auf und zeigt Informationen für den TSF-Algorithmus an.

3.3.3 Dateien freigeben

Navigieren Sie mit Hilfe des eingebauten Browsers durch Ihr Filesystem und geben Sie Dateien frei, indem Sie in „Menü“ „Freigeben“ wählen. Entsprechend heben Sie die Freigabe auch wieder auf. Zusätzlich können Sie sich eine Vorschau verschiedener Dateien anzeigen lassen[5].

3.3.4 Dateien suchen

Starten Sie eine neue Suche und geben Sie den Suchbegriff ein (standardmäßig verodert) und wählen Sie „Start“. Die Suche startet nun in einem Broadcast und liefert entsprechende Ergebnisse zurück.

3.3.5 Dateien downloaden

Wenn wenigstens ein Such-Ergebnis gefunden wurde (erkennbar an einer „(1)“ hinter der Suche), können Sie anfangen, eine Datei herunterzuladen. Selektieren Sie dafür die entsprechende Datei und wählen Sie entweder „Zu Downloadliste hinzufügen“, wenn Sie den Download ersteinmal nur vormerken wollen oder „Download sofort starten“, wenn Sie den Download sofort starten wollen.

Wurde der Download ersteinmal nur vorgemerkt, müssen Sie im Hauptmenü im Untermenü „Downloads“ die entsprechende Datei wählen und „Download beginnen“ auswählen.

3.3.6 Verschiedenes

Bei „Optionen“ haben Sie die Möglichkeit einzustellen, wie lange eine Suchanfrage dauern kann. Je höher die Zahl, desto länger wird auf Ergebnisse gewartet. Das ist insbesondere sinnvoll in größeren Netzen, wo eine Suchantwort viele verschiedene Geräte überspringen muss. In kleineren Netzen empfiehlt es sich dagegen, einen niedrigen Wert zu wählen.

[5] von Dateityp abhängig

4 Grundlagen

4.1 Grundlagen des P2P-Computing

Unter Peer-to-Peer versteht man die direkte, bilaterale Kommunikation zwischen zwei oder mehreren gleichberechtigten Partnern (Peers), die alle gleichwertige Fähigkeiten und Verantwortlichkeiten haben. Im Gegensatz zur klassischen Client/Server-Struktur, besteht bei P2P-Strukturen ein völlig dezentral gehaltenes Konzept wo es keine Rollenverteilung gibt. Die Peers teilen sich somit gegenseitig Ressourcen (Dateien, CPU-Zeiten, Speicherkapazität, etc.) und agieren dabei sowohl als Dienstbereitsteller als auch als Dienstnutzer.

Das Ziel des P2P-Computing ist es meist, in einem Netz auf mehreren Peers verteilte Ressourcen aufzufinden und gemeinsam zu nutzen. Dabei wird der Zugriff auf diese Ressourcen vom Benutzer des jeweiligen Peers autorisiert. Peer-to-Peer-Netze sind völlig dezentral. Es besteht direkte Verbindungen zwischen den Teilnehmern und auf jegliche Zentralinstanz wird verzichtet. Die Teilnehmer (Peers) können gleichzeitig als Server und Client fungieren. Peer-to-Peer-Netze nutzen freie Ressourcen. Jeder Peer weist vergeudete freie Ressourcen auf, die dadurch effizient ausgenutzt werden können.

P2P-Netze weisen verschiedene Architekturmodelle auf, unter anderem eine hybride Struktur, die die Vorteile der Client/Server und der dezentralen P2P-Struktur miteinander vereinen. Anfragen werden an einem zentralen Server gestellt, der andere Peers im Netzwerk dem suchenden Peer zu Verfügung stellt. Dieser baut anschliessend eine direkte Verbindung zu den jeweiligen Knoten auf. Hauptnachteil bei dieser Struktur ist der Single-Point-Of-Failure. P2P Systeme können auch dezentraler Natur sein, wobei auf zentrale Instanzen komplett verzichtet wird. Die Peers agieren gleichzeitig (Servents) und finden dynamisch andere verbundene Peers. Wegen der Unabhängigkeit der Knoten ist das Netzwerk sehr robust. Dem hierarchischen Modell entsprechen P2P in SuperPeer Netzen. SuperPeers verwalten den globalen Index, wobei jeder SuperPeer als Zentralserver für eine bestimmte Gruppe von Peers (Cluster) fungiert. Die Folge: Effiziente Suche innerhalb von Clustern, Dynamische Koordinierung, Kontrollierung des Netzes und Lastverteilung durch SuperPeers.

Zu den Anwendungsgebieten zählt das bekannteste Beispiel des Distributed Computing: SETI@home. Hierbei beteiligen sich mittlerweile mehr als 4,5 Millionen Peers, die die Daten eines Radio-Teleskops in Arecibo, Puerto Rico, nach Zeichen außerirdischen Lebens untersuchen. Dagegen sind die Anwendungen bei Collaborativ Computing event-basiert und reichen vom Instant Messaging über Online-Games bis hin zu Groupware (Groove). Die wesentliche Herausforderung an P2P-Systeme, die Collaborative Computing unterstützen, ist die Echtzeitübertragung von Daten, wo große Verzögerungen im Internet auftauchen, einige Funktionen beschränken und die Aktualisierungs-Dauer vom Peer bestimmen (Multiplayer-Spielen).

Die populärste Anwendung bei P2P-Systemen bleiben jedoch das FileSharing, in dem die bekannteste Programmen wie Napster, Gnutella oder KaZaA in den letzten Jahren stark an Bedeutung gewonnen haben. Napster basiert auf einer hybriden Peer-to-Peer-Struktur. Das Netz besteht aus einem oder mehreren zentralen Discovery Servern, die ein Indexverzeichnis der angebotenen Dateien beinhalten. Die Peers suchen Dateien durch den Server und erhalten dann eine Ergebnisliste inklusive IP-Adresse und Port der Knoten, die die gewünschte(n) Datei(en) gespeichert haben. Der Download erfolgt nach direkter Verbindung mittels HTTP-Protokoll.

Gnutella basiert auf einem reinen unstrukturierten P2P-Netzwerk. Die Datei-en liegen verstreut über das ganze Netz, wobei Dateien eines Peers nur diesem bekannt sind, und so eine gewisse Autonomie gewährleistet ist. Die Peers stehen alle auf der gleichen Ebene und erledigen dieselben Aufgaben: Ressourcen zu Verfügung stellen und in Anspruch nehmen, Suchanfragen weiterleiten (Router), aussenden und beantworten. Das Protokoll basiert auf fünf verschiedenen Verwaltungspakettypen (Ping, Pong, Query, QueryHit, Push). Nach erfolgreicher Suche wird die direkte Verbindung per HTTP-Protokoll zu einem geeigneten Rechner aufgebaut und die Datei herunter geladen. Gnutella hat schließlich wegen

seiner Einfachheit und Unabhängigkeit von zentralen Instanzen relativ viel Interesse gewonnen. Dabei weist dieses Protokoll jedoch ein Paar Lücken auf: Ressourcen, die sich hinter dem Horizont von sieben Hops (standardmäßig) befinden, können nicht erreicht werden. Das System skaliert nicht gut, da die Anzahl der gebroadcasteten Anfragen und mögliche Antworten exponentiell mit jedem Hop wächst, was zu einem starken Netzverkehr führt.

KaZaA ist im Gegensatz zu Gnutella kein Protokoll sondern ein Filesharingsoftware der FastTrack Protokoll-Familie. Die Struktur des KaZaA-Netz weist ein hierarchisches Konzept auf und bewegt sich im serverbasierenden und serverlosen Bereich, indem sie Aspekte des Gnutella Netzes mit einer flexiblen zentralen Struktur von SuperPeers verbindet. Nach erfolgreicher Suchanfrage werden die IP-Adressen der jeweiligen Fundstellen an den Peer geleitet. Die interne Kommunikation findet dann zwischen den Teilnehmern verschlüsselt statt, und der Download per HTTP-Protokoll. Das KaZaA-Netz ist sehr leistungsfähig und eignet sich gut für alle Dateitypen. Einziger Nachteil ist die Bereitstellung möglicher SuperPeers.

Die Suche und das Routing bei P2P-Netzen lässt sich in zwei Kategorien unterteilen: Man unterscheidet zwischen der Suche in unstrukturierten Systemen und der in strukturierten. In unstrukturierten Systemen kann die Suche mit oder ohne Routing-Indizes erfolgen. Bei der erste Methode wird das Fluten des Netzwerks durch einen TTL-Wert eingeschränkt, was sich als ineffizient erweisen kann, weil vorhandene Dateien womöglich nicht gefunden werden. Iterative Deeping umgeht das Problem indem bei erfolgloser Suche der TTL-Wert erhöht wird. Die Last bei der Suche kann auch reduziert werden durch Bewertung von Nachbarknoten oder Neustrukturierung der Narbarschaft. Knoten mit bester Bewertung oder gemeisamen Interessen können schnell besucht werden. Lokale Indizes können dazu dienen, Anfragen gezielt zu senden, wobei die Schwierigkeit bei dieser Methode ist, sicherzustellen, dass die lokalen Indizes beim Verlassen oder Hinzukommen von Peers aktualisiert werden. Unter der Suche mit Routing Indizes stehen verschiedene Methoden wie Compound Routing Indizes (Dokumenten werden Kategorien zugeordnet), Hop Count Routing Indizes (Gewichtung der Entfernung zu Ressourcen einer bestimmten Kategorie), Chord und CAN.

Chord realisiert eine konsistente, verteilte Hashfunktion, die die gleichmäßige Verteilung von Informationen über alle teilnehmenden Knoten ermöglicht. Jeder Knoten und jede Ressource im Chord-Netzwerk ist mit einem eindeutigen m-bit Identifier auf einem Chord-Ring einer Größe angeordnet, die durch eine konsistente Hashfunktion errechnet wird. Dabei muss m so gewählt sein, dass verhindert wird, dass zwei Knoten auf denselben Schlüssel gehasht werden (bzw. die Wahrscheinlichkeit sehr gering ist). Chord sieht zwei Arten von Routing vor:

- Einfaches Routing: Dabei wird durch einfaches Verfolgen der successor-Zeiger die Suchanfrage durchgereicht, bis der für den Schlüssel verantwortliche Knoten gefunden wird. Die Suchlaufzeit ist dann linear zur Knotenanzahl. Bei N Knoten beträgt sie O(N).
- Skalierbares Routing: Bei diesem Routingverfahren verfügt jeder Knoten über zusätzliche Routing-Informationen in Form einer Finger-Tabelle. Eine Finger-Tabelle hat eine maximale Größe von m Einträgen.

Die Basis von CAN ist ein d-dimensionaler kartesischer Koordinatenraum, auf den Schlüssel-Wert-Paare (key, value) durch eine verteilte Hashfunktion abgebildet werden. Die Hashfunktion ordnet einem Key k einen Punkt P im Koordinatensystem zu. Ein (key, value)-Paar wird von demjenigen Knoten gespeichert, in dessen Zone P liegt. Jeder Knoten im Netzwerk speichert die Zonen-Koordinaten und IP-Adressen seiner Nachbarn. Bei der Suche wird der passende Punkt P zu einem Suchbegriff durch die Hashfunktion bestimmt. CAN bietet eine Reihe von Verbesserungen, die z.B. die Dauer der Suchanfrage, die Länge des Suchpfades (auf Anwendungs- oder IP-Ebene) oder die Fehlertoleranz reduzieren.

P2P-Architekturen haben durch ihre grundlegende Struktur mit offenen Problemen zu kämpfen. Die

Kriterien und Anforderungen, die an P2P Netzen gestellt werden, sind: Skalierbarkeit, Sicherheit, Administrierbarkeit/Selbstorganisation, Fehlertoleranz/Robustheit, Anonymität, Performanz und Interoperabilität. Ein Beispiel als Plattform für P2P-Anwendungen ist die Opensource-Lösung JXTA der Firma Sun, mit dem Ziel, Programmierern eine einfache, flexible Möglichkeit zu schaffen, verteilte Anwendungen zu implementieren ohne zuerst ein eigenes P2P Framework zu entwickeln (siehe Sun Website).

Mobile-Computing ist neben Peer-to-Peer-Computing in den letzten fünf Jahren sehr populär geworden, vor allem wegen der schnellen Weiterentwicklung mobiler Endgeräte, die den Markt immer mehr erobert haben. Die Kombination von mobilen Endgeräten und mobilen Ad-Hoc Netzwerken (MANETs), ermöglicht die Entwicklung mobiler Peer-to-Peer-Systeme, mit denen mobile Peers mittels drahtloser Kommunikation (z.B. Bluetooth, UMTS) Verbindungen miteinander aufbauen können. Einige technische Herausforderungen, mit denen Mobile-P2P-Computing zu kämpfen hat, sind vor allem die dezentralen Natur mobiler P2P Systeme, das erhöhtes Risiko bei der Mobilität, die Tatsache, dass die drahtlose Verbindung stark variabel in Leistung und Robustheit ist, die dynamische Topologie mobiler P2P Systeme, die Abhängigkeit mobiler Geräte von einer endlichen Energiequelle, sowie die Tatsache, dass mobile Geräte wenige Ressourcen wie z. B. Speicherkapazität, Prozessorleistung zur Verfügung stellen. Ein gutes Beispiel ist 7DS[6], ein P2P-Filesharing-System, das neben einer eigenen Architektur eine Reihe von Protokollen und eine Implementierung beinhaltet.

Abschliessend muss man klar erwähnen, dass P2P-Systeme nur mühsam zu kontrollieren sind, da sie ohne zentrale Instanzen auskommen und deshalb mangelnde QoS[7]-Garantien (unterschiedliche Anzahl von Ergebnisse und Bearbeitungszeiten)
aufweisen, wodurch Arbeitsprozesse schwer kalkulierbar sind. Auf Grund dieser Tatsache ist es für Unternehmen kaum möglich, gewinnbringende Architekturen für den Markt zu erschließen. Es ist mehr davon auszugehen, dass sich eher eine Mischung von P2P- und Client/Server-Modellen in Unternehmensbereich durchsetzen wird. Erst langsam im Laufe der Zeit wird man künftig erkennen können, welche Potenziale hinter der P2P-Technologie stecken.

4.2 Java 2 Micro Edition

Die J2ME (JAVA 2 Platform Micro Edition), ist unsere, von Sun erstellte, Ausgangs-Plattform gewesen, die speziell für Mobile Applikationen entwickelt wurde. Die Grundlage von J2ME bilden dabei die Konfigurationen und die Profile, wobei die Konfigurationen aus CDC (Connected Device Configuration) und CLDC (Connected Limited Device Configuration) bestehen.

Wir haben in unserem Projekt das CLDC benötigt, da es im Gegensatz zum CDC für mobile Endgeräte gedacht ist, die stark eingeschränkt sind im Bezug auf Speicher und Prozessorleistung. Mit Profilen sind dabei die APIs[8] gemeint, die es zu einer Konfiguration gibt. Einige der APIs der J2ME waren aus der J2SE (JAVA 2 Standart Edition) oder J2EE (JAVA 2 Enterprise Edition) bekannt, teilweise waren sie stark gekürzt, um den Speicherbedarf so gering wie möglich zu halten, während andere, komplett neue APIs hinzugekommen sind.

Die J2ME verfügt zusätzlich über viele verschiedene optionale Pakete und kann somit verschieden möglich erweitert werden. Wir mussten abwägen, welches der optionalen Pakete für unser Projekt von Bedeutung war. Als wir mit unserem Projekt begonnen hatten, standen insgesamt sieben dieser optionalen Pakete zur Verfügung. Einige waren noch im Status *in progress*, so dass wir auf diese verzichten mussten. Oft stellen sie ein grobes Gerüst dar, welches vom Entwickler, bis auf ein paar kleine Ausnahmen, beliebig fertig gestellt werden kann. Sie sind in so genannten JSRs (Java Specification Requests) festgelegt und spezifiziert.

Hier eine Übersicht der einzelnen Pakete :

[6] Seven (7) Degrees of Separation
[7] Quality of Service
[8] Application Programming Interface - eine Programmierschnittstelle

- JSR 75 (PDA -> Personal Information Management (PIM))
- JSR 82 (Bluetooth, siehe S. 12)
- JSR 120 (Wireless Messaging API)
- JSR 172 (Web Services Specification)
- JSR 179 (Location)
- JSR 180 (SIP API -> Session Initiation Protocol)
- JSR 234 (Advanced Multimedia Supplements)

Wir haben für unser Projekt die JSR 75 und die JSR 82 benötigt.

Die JSR 75 ermöglichte uns den Zugriff auf das Handy File-System, während die Bluetooth JSR 82 das nötige Protokoll lieferte. Laut Java-Doc war die JSR 75 vorgesehen, um einen Zugang zu Personal Information Management (PIM) Daten auf einem J2ME Gerät zu ermöglichen.

Mit PIM-Daten sind Informationen im Adressbuch, Kalender oder in to-do-list-Applikationen gemeint. Diese API stellt Zugänge zu den jeweiligen PIM-Daten zur Verfügung. Oft tauchte auch der Begriff Generic Connection Framework (low-level Connections) auf. Das GCF wurde mehrfach in den einzelnen optionalen Paketen erwähnt und ließerahnen, dass es sich um ein zentrales Framework handelt. GCF ist ein Satz Interfaces, die im `javax.microedition.io`-Paket definiert sind, mit denen eine Vielzahl an Verbindungen erstellt werden kann. Die Aufgabe des GCF ist es, die input/output- und networking-Funktionalitäten in J2ME-Profilen zu übernehmen. Es ist somit das Fundament aller Applikationen, die beliebige Verbindungen und Netzwerke implementieren. Es wurde ursprünglich definiert, um auf der J2ME-Plattform aufzubauen, da die vertrauten J2SE `java.net` und `java.io` APIs zu großwaren, um in den begrenzten Speicher der mobilen Geräte zu passen. Beispielsweise besteht das `java.io` Paket aus ca. 60 Klassen plus Interfaces und ca. 15 Exception-Klassen. Zusammen, mit den ca. 20 `java.net` Klassen und 10 Exception Klassen, würden sich insgesamt 200 kb ergeben, ohne eine einzige Zeile implementiert zu haben.

Das war entschieden zuviel, um auf einem JAVA-fähigem Handy zu laufen.

4.3 Bluetooth

4.3.1 Baseband

Das Baseband befindet sich im Transport-layer des Bluetooth Protocol Stacks und setzt direkt auf die Funkschnittstelle auf.

Der eigentliche BT-Funk findet auf dem 2,4GHz ISM-Band (ISM steht für Industrial, Scientifical and Medical), dem kosten- und lizenzfreien Band statt. Der Frequenzbereich geht von 2.400 MHz bis 2.483,5 MHz. Aus Interferenzgründen startet das Bluetooth-Band bei 2.402 MHz und geht bis 2480 MHz. Das gesamte Band ist dabei in 79 Kanäle zu je einem MHz unterteilt. Durch die Frequenzmodulation entsteht eine Datenrate von 1 Mbps.

Baseband selbst ist für die Verteilung der Datenpakete zwischen den einzelnen BT-Endgeräten über die Funkschnittstelle zuständig.

BT ist eine Master/Slave orientierte Verbindung. Sobald ein ad-hoc-Netzwerk aufgebaut wird, sind automatisch ein Gerät der Master und alle anderen die Slaves. Ein Bluetooth ad-hoc-Netzwerk besteht im Prinzip aus dem oben genannten Master und bis zu sieben weiteren aktiven Geräten - die aktiven Slaves. Es gibt Möglichkeiten, um weitere Endgeräte in dieses sogenannte Piconet als passive Slaves einzubinden. Je nach Adressierungsart kann man damit dem Piconet bis zu theoretisch 255 weitere

Geräte hinzufügen.

Das Baseband benutzt vier verschiedene logische Kanäle: Den Basic-Piconet und Adapted-Piconet Kanal, den Inquiry-Scan und den Page-Scan Kanal. Ein solcher logischer Kanal ist im physikalischen dadurch charakterisiert, dass in einer pseudo-zufälligen Reihenfolge durch die verfügbaren Kanäle gesprungen wird. So nutzt der Basic-Piconet Kanal alle 79 Frequenzen, während der Adapted-Piconet normalerweise nicht durch das volle Spektrum geht. Die verschiedenen Kanäle können weiterhin durch ihren eindeutigen Access-code unterschieden werden.

Der eigentliche Sende-/Empfangsvorgang ist strikt geregelt. Ein Slave kann immer nur an den Master und nicht an die anderen Slaves in seinem Piconet senden. Der Master selbst bestimmt das Sendeverhalten der Slaves dadurch, dass ein Slave nur dann senden darf, wenn es in dem Sendeslot vorher durch den Master angesprochen wurde. Ein Slot entspricht dabei ein Hop.

4.3.2 L2CAP

Mit dem Logical Link Control and Adaption Protocol (L2CAP) wird eine Möglichkeit geboten, bequem eigene Protokolle zu definieren. Es vereinfacht mit seiner Architektur das Implementieren neuer Protokolle, so dass man nicht jedes Mal direkt auf der Hardware programmieren muss, sondern schon eine Software-Schnittstelle verwenden kann. L2CAP ist diese unterste Software-Schnittstelle. Grundsätzlich unterscheidet man zwischen zwei Verbindungsarten, die mit Bluetooth möglich sind, nämlich SCO (synchronous connection-oriented) und ACL (asynchronous connection-less). SCO-Verbindungen werden überwiegend bei
Sprachübertragung eingesetzt, während ACL-Verbindungen bei Datenübertragungen zum Einsatz kommen. Mit L2CAP sind nur ACL-Verbindungen möglich, weshalb man bei Sprachübertragungen direkt auf das Baseband-Protokoll zurückgreifen muss.

4.3.3 L2CAP Kommunikation

Das L2CAP Protokoll setzt auf Paket-basierten Übertragung und gehört zur Sicherungsschicht des OSI-Modells. Das heißt, es ist für das Aufteilen der Daten in Pakete verantwortlich. Auch das evtl. Anfordern fehlerhaft übertragener Pakete gehört zu seinem Aufgabenbereich. Insbesondere gehören folgende Begriffe hinzu:

- Die Maximum Transmission Unit (MTU) definiert die maximale Paketgröße, die übertragen werden kann. Das Paket kann kleiner sein, darf aber unter keinen Umständen größer als der MTU-Wert sein.
- Der Flush Timeout bestimmt wie lange der Empfänger brauchen darf, bis er das Paket bestätigt. Kommt in dieser Zeit keine Bestätigung beim Sender an, wird das Paket nochmals gesendet.
- Die Quality of Service (QoS) legt Prioritäten für den Datenverkehr fest. So kann z.B. Sprachübertragung stärker bevorzugt werden als Datenübertragung.

Die Werte Flush Timeout und Quality of Service werden bereits vom Bluetooth-Stack gesetzt. Lediglich die MTU kann man mit der JSR 82-API einstellen. Wird die Einstellung nicht vorgenommen, so wird ein DEFAULT_MTU von 672 Bytes benutzt. Der Aufbau einer L2CAP-Verbindung läuft im Wesentlichen so ab, wie mit SPP[9].

4.3.4 Applikationen mit Bluetooth

Eine Applikation, basierend auf Bluetooth Spezifikation, besteht im Wesentlichen aus 6 logischen Schritten:

1. Stack Initialisation

[9] Serial Port Protocol (serielle Schnittstelle)

2. Device Management
3. Device Discovery
4. Service Registration
5. Service Discovery
6. Kommunikation

Unter Stack Initialisation versteht man eine Reihe von Zuweisungen, wobei die benötigten Geräteeigenschaften wie z.B. Name, Adresse, Portnummer, Status (sichtbar oder versteckt) u.s.w. eingestellt werden. Diese sind gerätespezifisch und daher nicht in der Bluetooth API eindeutig festgelegt. Die Einstellungen werden vom Gerät vorgenommen und schließlich an die Methode `getProperty()` der Klasse `javax.bluetooth.LocalDevice` angebunden.

In der Phase DeviceManagement beschäftigt man sich mit den Klassen LocalDevice und RemoteDevice. Für die Konfiguration des eigenen Gerätes wird zunächst eine Instanz der Klasse LocalDevice gebildet. Dies geschieht jedoch durch den Aufruf der statischen Methode `getLocalDevice()`, weil der klasseneigener Constructor geschützt ist.

Alle Bluetooth Geräte haben ähnlich wie Netzwerkkarten eindeutige Netz-werk-Adressen. Diese werden in Form eines 12-stelligen Strings gebildet. Mit der Methode `getBluetoothAddress()` ist es nun möglich diese Adresse auszulesen. Um einem Gerät die Möglichkeit zu gewährleisten von anderen Geräten gefunden zu werden sorgt die Methode `setDiscoverable(int mode)`. Dabei ist mode eine Zahl die die Sichtbarkeitszustände, entsprechend Bluetooth Spezifikation, darstellt. Um diesen Zustand abzufragen ist die Methode `getDiscoverable()` zuständig.

Ähnlich wie mit eigenem Gerät, verhält es sich auch mit dem entfernten Gerät (RemoteDevice). Die Klasse RemoteDevice ermöglicht einen Zugriff zu einem entfernten Gerät in der Umgebung.

Um eine Verbindung zwischen zwei Bluetooth Geräten herzustellen ist es notwendig, dass mindestens einer von beiden den anderen kennt. Um eine Erkennung zu ermöglichen, besitzt die Java Bluetooth API (JSR 82) die so genannte Device Discovery Phase. Nach einer erfolgreichen Initialisierung eines eigenen Bluetooth Gerätes ist es nun möglich, den DiscoveryAgent zu starten. Ab diesem Zeitpunkt ist nun der Agent bereit, die entfernten Bluetooth-Geräte zu erkennen. Damit der aktuelle Thread nicht blockiert wird, ist der Prozess der Erkennung in Form von Events gemacht. Die Schnittstelle DiscoveryListener enthält alle Ereignisse, die ausgelöst werden können, während eine Device-Erkennung ausgeführt wird. Um die Erkennung nun zu starten, ruft man die Methode `startInquiry(int accessCode, DiscoveryListener listener)`. Sobald ein neues Device gefunden wurde, wird die Methode `public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)` als Ereignisbehandlung aufgerufen. Mit der Methode `retrieveDevices(int option)` bekommt man einen Array von Instanzen aller von StartInquiry gefundenen Geräte.

Als Server legt man eigene Information in der Service Discovery Database ab. Für jede Instanz der Klasse LocalDevice werden die angebotenen Dienste in Form von ServiceRecord-Einträgen eingestellt. Java Bluetooth API bietet dafür die Schnittstelle namens ServiceRecord. Darin werden die Attribute in Form einer UUID[10] Nummer gespeichert und durchnummeriert. Der ganze Prozess umfasst folgende 6 Schritte:

1. Initialisierung des eigenes Gerätes:
2. Bilden von Attributen und UUIDs
3. Zusammensetzen von URL

[10] Universal Unique IDentifier

4. Öffnen eines StreamConnectionNotifiers
5. Erstellung eines ServiceRecords (mit Hilfe der statischen Methode `getRecord()`)
6. Erstellung eines neuen Datenelements und Setzen von Attributen

Nachdem ein Eintrag in die SDDB erfolgt ist, wird der Thread angehalten, solange kein Client eine Verbindung herstellt. Nachdem das geschehen ist muss der aktuelle StreamConnectionNotifier geschlossen werden.

Hat man als Client ein Gerät erkannt ist es möglich mit Hilfe folgender Methode der Klasse discoveryAgent nach unterstützten Diensten zu erfragen. Sinnvoll ist hier mehrere UUIDs anzugeben. Als Ergebnis wird ein Integer geliefert, der eine Transaktions-ID für spätere Behandlungen darstellt.

4.4 JSR 82

Ziel der JSR 82-Spezifikation (JAVA Specification Request 82) ist es, eine JAVA API zu schaffen, mit welcher eine Kommunikation via Bluetooth möglich ist. Der vorwiegende Einsatz wird auf Handys, sowie PDAs sein. Die API ist deshalb so definiert, dass die o.g. Vorraussetzungen erfüllt sind. JSR 82 ist so konstruiert, dass ständig neue Profile hinzugefügt werden.

Eines der Hauptpakete (`javax`) beinhaltet zwei weitere unabhängige Unterpakete (`javax.obex` und `javax.bluetooth`). Diese Aufteilung ist sinnvoll, weil OBEX-Protokoll auf Bluetooth angepasst wird. OBEX-Protokoll kann somit auch auf anderen physischen Schichten, wie IR[11] angewandt werden. Eine Applikation, aufbauend auf der JAVA Bluetooth API, kann eine von beiden oder auch beide Paketen importieren.

Mit der Methode `Connector.open()` wird bei der clientseitigen Anwendung eine Verbindungsanforderung geöffnet. Die Schnittstelle `javax.bluetooth.L2CAP`
`Connection` bietet eine Reihe von Methoden, die für eine Kommunikation benötigt werden. Darunter sind die Methoden zum Senden und Erhalten eines Byte-Arrays. Bei der Arraylänge ist auf die MTU-Größe zu achten.

4.5 Tree Scatternet Formation Algorithmus

4.5.1 Piconetze & Scatternetze

Bei der Vernetzung von Bluetooth-fähigen Geräten unterscheidet man zwischen Piconetzen und den aus mehreren Piconetzen bestehenden Scatternetzen.

Piconetze Ein Piconetz besteht aus bis zu 8 aktiven Knoten und kann max. 255 weitere geparkte, inaktive Knoten besitzen. Man unterscheidet hierbei zwischen Master- und Slave-Knoten.

In jedem Piconetz gibt es genau einen Master-Knoten, der den gesamten Transfer zwischen den Slaves steuert und sich normalerweise im Zentrum des jeweiligen Piconetzes befindet. Der Master bestimmt außerdem die Sequenz der Frequenzen, über die nacheinander gesendet wird. Direkte Transfers zwischen zwei Slaves sind nicht erlaubt und müssen über den Master laufen.

Für das Finden von potenziellen Kommunikationspartnern wird das Inquiry-Verfahren verwendet, in dem die Geräte abwechselnd durch die verschiedenen Frequenzen springen und nach anderen Geräten suchen bzw. auf einkommende Nachrichten warten. Im Paging-Verfahren kann dann erst der tatsächliche Verbindungsaufbau stattfinden, weil dort erst bekannt ist, welche Geräte sich in der Umgebung befinden und wie ihre Addressen (und damit die Sprungvektoren) lauten.

[11] Infrarot

Festzustellen ist hierbei, dass ein Slave nur Daten mit dem Master austauschen kann, nicht aber mit anderen Slaves. Um seinerseits Daten zu den übrigen Einheiten des Piconetzes transferieren zu können, muss ein Slave die Daten zuerst an seinen Master schicken, der diese dann weiterleitet. Dies ist jedoch nur in dem Slot möglich, der einer vorhergenden Verbindungsaufnahme durch den Master selbst folgt.

Scatternetze Mehrere Piconetze können in derselben Umgebung nebeneinander existieren, ohne dass der Durchsatz in den Teilnetzen wesentlich beeinträchtigt wird. Einen solchen Verbund mehrerer Piconetze nennt man Scatternetz. Die Piconetze sind hierbei über Brückenknoten, die auch Gateways genannt werden, verbunden. Gateways sind Knoten mit mehr als einer Rolle zwischen Piconetzen. Diese Gateways können sowohl Master- als auch Slave-Knoten in den verschiedenen Netzen sein, aber nur in maximal einem Netz als Master fungieren.

Bluetooth Link Formation Der Formationsprozess eines Bluetooth Piconetzes besteht aus zwei Phasen: Inquiry-Phase und Page-Phase:

Inquiry-Phase In dieser Phase sollen sich alle Geräte, die sich in Reichweite befinden, kennenlernen. Dazu wechselt ein Gerät ständig zwischen dem Inquiry Modus und Inquiry Scan-Modus. Im Inquiry-Modus versucht das Gerät andere Geräte zu finden, indem es ID-Pakete versendet und auf eine Antwort wartet. Ein Gerät im Inquiry Scan-Modus wartet auf Anfragen der Geräte, die sich im Inquiry Modus befinden. Wenn es eine Anfrage erhält, sendet es seine Geräte-ID (BT_ADDR) und seine Clock zurück. Das Gerät im Inquiry Modus wird Master und das im Inquiry Scan-Modus Slave in dieser Verbindung. Wenn nach einem Timeout ein Gerät mindestens ein anderes Gerät gefunden hat, geht es in den Page-Modus über.

Page-Phase In diese Phase werden die Informationen, die in der Inquiry-Phase gesammelt wurden, genutzt, um eine Kommunikation zwischen zwei Ge-räten aufzubauen.

Da der Master, der sich jetzt im Page-Modus befindet, die Geräte-ID und die Clock des Slaves kennt, nutzt er diese Informationen um mit dem Slave in Verbindung zu treten. Der Slave geht dann in den Page Scan-Modus über. Aufgrund des nicht synchronisierten Clocks von Master und Slave kann es zu Verzögerungen im Verbindungsaufbau kommen.

4.5.2 TSF: Tree Scatternet Formation

Beschreibung Der TSF Algorithmus von Tan[12] wurde entwickelt, um der Dynamik, die in Scatternetzen durch das Ankommen und Verlassen von Geräten entstehen kann, gerecht zu werden, damit stets ein zusammenhängendes Scatternetz gewährleistet ist.

Um dies zu erreichen, wird eine Baumstruktur geschaffen. Zu Beginn gibt es nur einzelne Knoten, so genannte freie Knoten, die jeweils einen Teilbaum des noch nicht verbundenen Scatternetzes bilden.

Das Verbinden zu einem Baum geschieht entsprechend den vom TSF zugewiesenen Rollen:

[12] Godfrey Tan. Self-organizing Bluetooth Scatternets. Massachusetts Institute of Technnology, Januar 2002.

Freier Knoten	Zunächst versucht jeder freie Knoten, d.h. der noch mit keinem anderen Knoten in Verbindung steht, andere Knoten zu finden, um einen Baum zu bilden. Haben sich zwei Knoten gefunden, wird der Master zur Wurzel des Baumes, der Slave zu seinem Kind.
Baumknoten	Knoten, die bereits einen Elter haben, können sich nur mit freien Knoten verbinden. Dieser klinkt sich damit entweder in ein bereits bestehendes Piconetz ein, oder der Elter erzeugt ein neues. Der ehemalige Freie Knoten bildet damit also eine Brücke zwischen dem alten und neuen Piconetz.
Wurzeln	Wurzeln können sich nur mit Wurzeln aus anderen Bäumen verbinden. Bei einer solchen Verbindung wird eine Wurzel Master des neuen Baumes und die andere Slave der Wurzel des anderen Baumes. Um dies auszuführen, werden Koordinatorknoten in den einzelnen Bäumen bestimmt.
Koordinatorknoten	Koordinatorknoten sind Wurzeln oder Baumknoten in einem Baum. Sie versuchen Koordinatorknoten in anderen Bäumen zu finden. In jedem Baum gibt es jedoch nur einen Koordinatorknoten.

Wenn eine Wurzel nur ein Kind hat, ist sie selbst Koordinatorknoten. Ansonsten wählt sie zufällig eines ihrer Kinder und bittet es Koordinatorknoten zu werden. Wenn der ausgewählte Kinderknoten kein Koordinator sein will, wählt das Kind zufällig einen seiner Kinderknoten. Dieser Prozess wird solange ausgeführt, bis ein Koordinator gefunden ist oder ein Blattknoten erreicht ist. Ein ausgewählter Blattknoten muss Koordinator werden. Blattknoten haben keine Kommunikationsengpässe, und deshalb haben sie mehr gesparte Energie zum Entdecken der Nachbargeräte. Wenn ein Knoten Koordinator wird, sendet er ein Wissenspaket an seine Wurzel und beginnt nach Koordinatorknoten anderer Teilbäume zu suchen.

Wenn zwei Koordinatoren sich gefunden haben, informieren sie beide ihre Wurzelknoten. Die Koordinatoren brechen dann den Link zwischen sich ab und nehmen ihre vorherigen Rollen wieder auf. Währenddessen verbinden sich die Wurzelknoten, wie oben beschrieben. Die Wurzel wählt dann zufällig einen anderen Koordinator. Da auch ein Koordinatorknoten das Netzwerk jederzeit verlassen könnte, wird der Koordinatorknoten immer nur auf eine begrenzte Zeit bestimmt.

State Machines Auf jedem Knoten im Netzwerk läuft eine State Machine, welche stets zwischen zwei von drei möglichen Stati wechselt. Die Stati der State Machine können sein: Inquire, Comm, und Comm/Scan Status

- Inquire-Status: Ein Knoten in diesem Status führt den Inquiry Modus aus.
- Comm-Status:Ein Knoten in diesem Status ist entweder untätig oder überträgt Daten zu den mit ihm verbundenen Knoten.
- Comm/Scan-Status: Ein Knoten im Comm Status beginnt im Scan-Status, um Daten zu übertragen und wechselt periodisch in den Scan-Status, um Inquiry Scan-Operationen auszuführen.

Die Zuweisungen der State Machine erfolgt folgendermaßen:

Freier Knoten und Koordinatorknoten wechseln zwischen Inquire und Comm/
Scan, um Verbindungen mit anderen möglichen Knoten aufzubauen. Wurzeln bleiben immer im Comm Status, und Baumknoten wechseln zwischen Comm und Comm/Scan.

Schleifenfreiheit Innerhalb der Baumstruktur des TSF können keine Schleifen auftreten, da nur folgende Verbindungen erlaubt sind:

1. Das Verbinden von zwei freien Knoten: Ein Knoten wird zum Master, der andere zum Slave des neuen Baums.
2. Das Verbinden von einem freien Knoten mit einem Baumknoten: Hier wird der freie Knoten zum Slave und der Baumknoten zum Master.
3. Das Verbinden von zwei Wurzeln: Eine Wurzel wird zum Master, die andere zum Slave.

Das Verbinden von zwei Baumknoten ist nicht erlaubt, da es zur Schleifenbildung führen könnte.

Heilung TSF garantiert, dass die Teilbäume innerhalb kurzer Zeit wieder miteinander verbunden werden. Es gibt zwei Arten, wie Verbindungen verloren gehen können:

1. wenn ein Master die Verbindung zu einem Slave verliert oder
2. wenn ein Slave die Verbindung zu seinem Master verliert.

Wenn ein Master merkt, dass er kein Kind mehr hat, braucht er nichts zu tun. Er muss lediglich überprüfen, ob er jetzt ein freier Knoten ist. Wenn ja, muss er in diese Rolle wechseln. Wenn nicht, bleibt er weiter Master der anderen Kinder.

Wenn ein Slave merkt, dass er keinen Master mehr hat, muss er seine Rolle verändern. War er ein Blattknoten, wird er freier Knoten. War er Baumknoten, wird er zur Wurzel des neuen Teilbaumes.

Um die in der Bluetoothspezifikation enthaltene Beschränkung auf maximal sieben Slaves pro Master zu gewährleisten, muss eine Wurzel, die die maximale Anzahl von Slaves erreicht hat, ein Kind zum Master und Wurzel eines neuen Teilbaumes machen.

4.6 Bluetooth Tree Routing Protokoll

Verschiedene Nachrichten müssen nicht an alle Knoten im Netzwerk geschickt werden. Es wäre ineffizient diese per Broadcast zu verschicken.

Da der antwortende Knoten die Zieladresse des Pakets kennt, kann ein direkter Weg über das Netzwerk gefunden werden. Da es sich aber nicht immer um direkt verbundene Knoten handelt, muss über mehrere Knoten geroutet werden.

Dieses Routing kann effizient mit dem Bluetooth Tree Routing (BTR) erfüllt werden. Die Routingtabelle werden direkt während der Netzwerkerzeugung erstellt. Es handelt sich um ein proaktives Routingprotokoll und neuzt die Baumstruktur des TSF. Die Routingdaten werden nicht erst erstellt, wenn sie benötigt werden, sondern bereits während des Aufbaus der Baumstruktur, im Gegensatz zu ein reaktiven. Dieses Verfahren funktioniert aber nur bei Baumstrukturen.

4.6.1 Funktionsweise

Routing-Tabelle Eine Routing-Tabelle wird aus sieben Hash-Tabellen erstellt. Dies entspricht den sieben Slaves, mit denen sich ein Master verbinden kann. Als Schlüssel wird die Bluetoothadresse des Geräts genutzt. Dazu wird die Entfernung zum Gerät gespeichert. Aus dieser Kombination kann der kürzeste Weg den gleichen Inhalten gefunden werden. Die Routing-Tabellen können verpackt über das Netz an andere Knoten geschickt werden.

Tabellenaufbau Der Tabellenaufbau erfolgt während des Scatternetzaufbaus. Es gibt zwei Möglichkeiten des Baumwachstums. Dann müssen Routingupdates versendet werden.

1. Ein freier Knoten will sich an ein bestehendes Netz anschließen. Der freie Knoten schickt seine Bluetoothadresse dem Baumknoten, an den er sich angeschlossen hat. Dieser erstellt für den angeschlossenen Knoten eine neue Hash-Tabelle und trät die Bluetoothadresse mit einer Distanz von eins ein. Der Baumknoten schickt anschließend ein Routingupdate an seinen Elter, das die aktualisierte des Baumknotens enthält. Der Elter aktualisiert seine Routingtabelle mit den erhaltenen Daten des Kindes und sendet wiederum eine Routingupdate-Nachricht an seinen Elter. Dabei müssen die einzutragenden Distanzen stets um einen erhöht werden. Dieser Vorgang setzt sich so lange fort, bis die Wurzel erreicht ist.
2. Es kommt zu einer Baumverschmelzung zweier Wurzelknoten. Dabei sendet die Wurzel, die sich als Kind anschließt ein Routingupdate an die andere Wurzel. Diese legt eine neue Hashtabelle für diese Wurzel als ihr Kind an und trägt die übertragenen Daten in diese ein. Die Entfernungen müssen dabei für alle eingetragenen Knoten um eins erhöht werden.
3. Routingupdates müssen auch versendet werden, wenn ein Knoten das Netzwerk verlässt.
4. Wenn ein Elter sein Kind verliert, muss er seine Routingtabelle aktualisieren. Die aktualisierte Routingtabelle muss anschließend an seinen Elter weitergeleitet werden. Dieser Vorgang wird ebenfalls solange weitergeleitet bis die Wurzel erreicht ist.

Aus diesen ständig aktuellen Routingtabellen können nun stets alle Kinder eines Elter bis in die Blätter des Baumes ausgelesen werden. Ein Knoten hat somit immer Wissen über den gesamten Teilbaum, der an ihn angeschlossen ist.

Routing Um eine Nachricht an einen weiter entfernten Knoten zu senden, durchsucht der Knoten zuerst seine Routingtabelle, ob der gesuchte Knoten ein entferntes Kind ist. Findet er den Knoten in seinem Teilbaum, sendet er die Nachricht an den Knoten, der die Wurzel für den entsprechenden Teilbaum bildet. Dieser leitet die Nachricht an den nächsten Knoten des Teilbaums, bis der Zielknoten erreicht ist.

Wird der gesuchte Knoten nicht in der Routingtabelle gefunden, befindet er sich nicht im Teilbaum. Also sendet er den Knoten an seinen Elter. Dieser sucht den Zielknoten in seiner Routingtabelle. Wird er dort gefunden, wird die Nachricht in den entsprechenden Teilbaum geleitet, wird er nicht gefunden, wird die Nachricht wieder an den Elter weiter geleitet. Spätestens wenn die Wurzel erreicht ist, kann die Nachricht in den richtigen Teilbaum geleitet werden, da die Wurzel alle Knoten im Netzwerk kennt. Kann auch die Wurzel den Zielknoten nicht finden, wird das Paket verworfen.

5 Ablauf der Projektgruppe

5.1 Seminarphase

In der Seminarphase bekam jeder PG-Teilnehmer ein Thema (oder ein Teil davon im Falle einer Gruppenarbeit) aus dem für die Projektgruppe relevantem Bereich, der aufgearbeitet und präsentiert werden sollte. Im Einzelnen waren das:

- Der Netzwerk-Simulator ns-2 *(Prost)*
- Software-Entwicklung mit der JAVA 2 Micro Edition *(Ben Said)*
- J2ME Pakete und Wireless Toolkit *(Krokowski)*
- Bluetooth mit J2ME *(Leikam)*
- Entwicklungswerkzeuge unter Linux (CVS, Autotools, Debugger, ...) *(Zok)*
- Grundlagen des Peer-to-Peer Computing (Napster, Gnutella, Distributed Hash Tables (CAN, Chord), KaZaA) *(Tigyo, Kißner)*
- Mobiles Peer-to-Peer Computing mit Passive Distributed Indexing *(Kasmann)*
- Bluetooth Grundlagen und Spezifikation *(Börgermann, Diel)*
- Bluetooth Scatternetz-Konstruktion und Link Scheduling *(Göbel)*

5.2 Programmiertest

Im Anschluss an die Seminarphase fand ein 24-Stunden-Programmmiertest statt, in dem die PG-Teilnehmer ihre Programmierkompetenzen unter Beweis stellen sollten. Der Test bestand aus der Aufgabe, mathematische Funktionen auf schwach besetzten Vektoren und Matrizen zu realisieren.

Die PG-Teilnehmer wurden dabei in zwei Gruppen aufgeteilt. Eine Gruppe sollte den Test in JAVA schreiben, die andere in C++. Die Gruppeneinteilung war wie folgt:

- JAVA: Ben Said, Kißner, Krokowski, Leikam, Tigyo, Zok
- C++: Börgermann, Diel, Göbel, Kasmann, Prost

5.3 Design Prozess gemäßUML/RUP

5.3.1 Brainstorming

In einem anfänglichen Brainstorming wurden verschiedene Funktionalitäten des Programms erdacht und wieder verworfen. Ferner gab es Anfangs Schwierigkeiten bei der Entscheidung, welche Programmierumgebung benutzt werden und für welches System das Programm entwickelt werden sollte (siehe „Probleme während des 1. Semesters“, S.21).

5.3.2 Rational Unified Process (RUP) mit Unified Modelling Language (UML)

Das komplette Projekt wurde gemäßdem Unified Process Model von UML entwickelt. Folgende Dokumente finden sich deshalb im Anhang:

- Anwendungsfalldiagramm
- Aktivitätsdiagramme
- Problembereichsmodell

- Sequenzdiagramme
- Klassendiagramm

Dem Erstellen der Diagramme und der daraus resultierenden klareren Spezifikation des Programmes und seiner Eigenschaften folgte die Implementierung (siehe 5.6, S.19), der Klassentest, sowie der Produkttest. Im Anhang finden sich deshalb ebenso die dort erzeugten Artefakte.

5.4 Konventionen

Wir haben uns auf folgende Konventionen geeinigt:

- Klassennamen haben große Anfangsbuchstaben, normal
- Attribute haben kleine Anfangsbuchstaben, normal, Unterstrich am Ende
- Variablen haben kleine Anfangsbuchstaben, normal
- Konstanten werden großgeschrieben
- Reihenfolge in einer Klasse:
  1. Attribute, Variablen, Konstanten
  2. Konstruktor
  3. Methoden
  4. set/get-Methoden
- Alles auf Deutsch
- Referenzen auf die Teilnehmer durch Nachnamen

5.5 Programmdesign

Der Name unseres Programmes, auf den wir uns einigten, ist „Blue Brewery Horse“, basierend auf einer PG-internen Anekdote. Blue Brewery Horse wird im Folgenden mit „BBH“ abgekürzt.

Um die vielfältigen Aufgaben unseres Programms besser zu strukturieren, entschieden wir uns schon zu Beginn der Entwurfsphase für eine klassenbasierte Aufteilung - ein sogenanntes Managersystem. Wir benutzten dabei die Unterteilung des Anwendungsfalldiagramms. Es entstanden SuchManager, EigenDateienManager, DownloadManager und UploadManager. Im folgendem werden nur die Zusammenhänge zwischen den wichtigsten Klassen beschrieben. Eine ausführliche Klassenbeschreibung aller Klassen ist im nächsten Abschnitt zu finden (S. 33).

Die Kommunikation zwischen den einzelnen Managern der jeweiligen Geräte findet über verschiedene Nachrichtentypen statt. BBHNachricht ist die Oberklasse unserer BBHNachrichtentypen, die jeweils von bestimmten Managern bearbeitet und versendet werden. Die Nachrichtentypen sind: SuchAnfrage, SuchAnwort, DownloadAnfrage, DownloadAntwort, DownloadAnforderung, DownloadAbbruch, Upload, UploadStart und UploadAntwort.

Der `SuchManager` erstellt `SuchAnfragen` und bearbeitet alle einkommenden `SuchAntworten`. Zur Verwaltung dieser Nachrichten benutzt er die Klasse `SuchMatchListe`, in der die `MatchListenEinträge` gehalten werden. Zu jedem `MatchListenEintrag` werden die Downloadquellen mit Verweisen auf die entsprechenden Peers gespeichert.

Der `EigeneDateienManager` beantwortet eingehende `SuchAnfragen`, speichert eingehende Downloads und verwaltet die Dateifreigabe in der `FreigabeListe`. In ihr werden die Dateien gehalten.

Der DownloadManager verwaltet die Downloads unserer Anwendung. Er erstellt DownloadAnfragen und DownloadAnforderungen. Dazu benutzt er die Klasse DownloadListe, die die DownloadEinträge verwaltet.

Der UploadManager stellt die Gegenseite des DownloadManagers dar. Er bearbeitet eingehende DownloadAnfragen und DownloadAnforderungen. Außerdem erstellt und verwaltet er die Uploads. Um auf die Uploads zuzugreifen, verwendet der Manager die UploadListe, die die einzelnen UploadEintraege enthält.

Die Manager erben vom Interface Nachrichtenmanager. Sie müssen die Methode empfangeNachricht() implementieren, damit über diesen Methodenaufruf die Nachrichten an den dafür vorgesehenen Manager gesendet werden kann.

SkyNet ist die Verbindung zwischen Netzwerkschicht und Applikationsschicht und leitet eingehende Nachrichten über die Methode empfangeNachricht() an die Manager weiter, die sich zu Beginn bei SkyNet für ihre Nachrichten registiert haben.

Programm erbt von Midlet und ist der Rumpf unserer Anwendung. Diese Klasse initialisiert alle Manager, den Timer, die GUIKontrolle und SkyNet. Im Programm halten wir eine Instanz von BBHTimer, der allen Managern zur Verfügung steht und alle Timingaufgaben unseres Programms übernimmt. BBHTimer hat einen Vektor von BBHTimerEintrag, den er bei jedem Aufruf von der Methode starteTimer() um einen neuen Eintrag erweitert. Er besitzt weiterhin einen BBHTimerTask, welcher sekündlich überprüft, ob ein Zeitpunkt von einem der BBHTimerEinträgen überschritten worden ist. Falls dies der Fall ist, wird der jeweilige Manager über den EreignisHandler benachrichtigt.

GUIKontrolle dient als visuelle Schnittstelle für den Benutzer.

Auf der Netzwerkschicht wird durch die Klasse Netzwerk ein Thread der Klasse Empfaenger für jede eingehende Verbindung, gelinkt auf das jeweilige entfernte Endgerät erstellt. Die Endgeräte werden in der GeraeteListe verwaltet. Zum Senden aller Nachrichten dient ein weiterer, einzelner Thread der Klasse Sender, der je nach Bedarf aktiviert wird. Außerdem stehen die Klassen Listener und ServiceSearch zur Verfügung, die die Bluetooth-spezifische Aufgaben für das Finden anderer Geräte erfüllen (ServiceSearch, Inquiry etc.).

Im 2. Semester wurde der VerwaltungsManger in die Netzwerkebene eingefügt. Dieser implementiert den TSF- und BTR-Algorithmus. Die Kommunikation zum Verbindungsaufbau des Scatternetzes wird über Verwaltungspakete, welche von Nachricht erben, geregelt.

5.6 Implementierung

Mit der Implementierung wurde nach Zeitplan gemäßUMP begonnen. Auf das Schreiben der einfachen Funktionen folgte ein Verlinken der Klassen, bis schließlich das gesamte Program an der SkyNet-Schnittstelle zusammengesetzt wurde. Im Zuge der Implementierung fanden rigorose Tests und parallele Fehlerbehebung statt.

Folgende Gruppenaufteilung haben wir am Anfang der Implementierungsphase vorgenommen:

- Netzwerkschicht: *Börgermann, Göbel, Diel*
- SuchManager (SuchMatchListe, MatchListenEintrag, DownloadQuelle, Peer), Downloadmanager (DownloadListe, DownloadEintrag, Datei), SuchIndexListe, IndexEintrag: *Tigyo, Kißner*

- UploadManager (UploadListe, UploadEintrag), Interfaces, BBHTimer, EigeneDateienManager (FreigabeListe): *Zok, Prost*
- Programm, Pakete: *Krokowski, Leikam*
- GUI: *Ben Said, Kasmann*

Im zweiten Semester sah die Aufteilung dann folgendermaßen aus:

- Fertigstellen des Backup-Servers: *Göbel*, *Diel*
- Konstruktion eines BBHLight Programms zum Testen auf richtigen Handys: *Börgermann*, *Prost*
- Verfeinerte Suche, Credit-Point-System und ID3-Tags auslesen: *Kißner, Leikam*
- Scatternetzkonstruktion: *Kasmann*, *Ben Said*, *Krokowski*, *Zok*, *Tigyo* (später dann *alle*)
- Endbericht: *Diel* (+ Inhalt von *jedem*)

5.7 Test und Fehlerbehebung

Das Programm wurde auf zwei Arten getestet[13]: Erstens als unabhängiger Klassentest und zweitens als kompletter Produkttest.

Bei dem Klassentest wurde jede Klasse mit einem Treiber und einem Dummy versehen, die die Schnittstellen abdeckten. Dann wurden der Klasse bestimmte Inputs gegeben, die in verschiedenen, für die Klasse sinnvolle Äquivalenzklassen eingeteilt waren. Dadurch wurde ein umfangreicher, eigenständiger Test sichergestellt

Für den Produkttest wurde das Program vollständig zusammengeführt und gestartet. Dort wurde dann das Zusammenspiel der Interfaces, sowie diverser, vorher nicht aufgetretener Fehler beobachtet. Durch den Produkttest wurde sichergestellt, dass das Program als solches auch fehlerfrei läuft.

5.8 Dokumentation

Die Dokumentation wurde während des UMP angefertig und zu jedem Treffen präsentiert. Änderungen, die sich im Dialog mit der Projektgruppe ergaben, wurden analysiert und entsprechend aufgenommen. Das Klassendiagramm wuchs dabei kontinuierlich an, da die Notwendigkeit des Hinzunehmens einer ganzen Reihe von Hilfsklassen noch während der Implementierung deutlich wurde. Bis zur Implementierung war das gesamte Klassendiagramm auch noch in Programm- und Netzwerkebene getrennt, bis es dann am Ende in ein großes überführt wurde.

Die JavaDoc befindet sich online auf http://mobicom.cs.uni-dortmund.de/pg466/intern/Doku/index.html (passwortgeschützt).

[13] Siehe auch Anhang A, S. 46

6 Ergebnisse, 1. Semester

6.1 Generelles

Während des Brainstormings arbeitete man sich parallel auch in die notwendigen APIs und den für die Verbindung als Modell genutzten BlueChat[14] ein. Dabei wurde festgestellt, dass eine direkte Manipulation der Bluetooth-Algorithmen zur Pico- und Scatternetzkonstruktion nicht möglich ist. Die bereits angesprochene `acceptAndOpen()`-Methode liefert eine L2CAP-Connection zurück und damit eine direkte logische Verbindung - einen Socket. Diese Methode ist allerdings vollständig abgekapselt und verbietet jegliche Einmischung in die Baseband-Ebene. Im Übrigen gibt es auch keine API, die Funktionen für die Manipulierung derselben zur Verfügung steht. Es wurde sich darauf geeinigt, das Programm ersteinmal unter Benutzung von `acceptAndOpen()` zu realisieren, da die darüberliegenden Schichten, die zum Verbindungsaufbau nötig sind, ohnehin kompliziert genug sind.

Daraus folgte natürlich, dass ein nutzbringender Einsatz des ns2-Netzwerk-Simulators vorerst nicht möglich war[15].

Während des Sichtens der APIs wurde weiterhin festgestellt, dass JAVA eigentlich kein direktes Speichern zulässt - für eine P2P-Anwendung recht hinderlich. Normalerweise laufen JAVA-Anwendungen im *Sandbox*-System, d.h., dass ein Speichern zwar möglich, aber nur innerhalb des Programmes zugelassen ist. Da wir aber die Dateien auf dem Endgerät direkt versenden und empfangen wollten, ohne dafür extra Kopien anlegen zu müssen, bedeutete das ein Problem.

Es gab zwei Alternativen: Die erste war, in C++ und für *Symbian*-Endgeräte zu programmieren. Das *Symbian*-Betriebssystem erlaubt einen direkten Speicherzugriff und wäre damit eine (workaround-) Lösung gewesen. Die zweite Alternative wäre, das Programm zu signieren lassen, da dies ebenfalls einen direkte Zugriff erlaubt.

Glücklicherweise erschien wenig später mit der JSR 75 eine Spezifikation, die das direkte Schreiben auf dem Filesystem des Endgerätes nach Benutzereinwilligung auf normalen Endgeräten ohne weitere Probleme zulässt. Wir entschieden uns deshalb für Letzteres.

Anfängliche Schiwerigkeiten gab es auch wegen dem Wireless ToolKit (WTK), der erst nur in der Version 2.1 verfügbar war. Diese Version unterstützt kein Bluetooth. Glücklicherweise kam auch hier kurze Zeit später der WTK 2.2 heraus, sodass wir damit auch keine Probleme mehr hatten.

6.2 Anwendungsebene

Beim Umwandeln von Variablen, bzw. Datenstrukturen in ein Bytearray zum Versenden über das BT-Baseband und zurück traten einige Probleme auf:

1. Die eindeutige Länge fehlte.
2. Zugriff auf einzelne Bytes eines Integer- oder Long-Wertes war nicht möglich.
3. Bei komplexeren Datentypen gibt es keine Möglichkeit diese als ein Objekt zum ByteArray zu konvertieren.
4. Bei einer einfachen Umwandlung eines Arrays oder eines Vektors in Form eines ByteArrays ist es schwierig die einzelne Werte aus diesem ByteArray wieder zu extrahieren.

Die Probleme wurden wie folgt gelöst:

[14] BlueChat ist ein Freeware-Programm, das einen Text-chat via Bluetooth ermöglicht. Der Autor ist Ben Hui, weitere Informationen gibt es unter http://www.benhui.net/bluetooth.

[15] Der Netzwerksimulator sollte ursprünglich dafür eingesetzt werden, einen Scatternetzalgorithmus zu entwerfen. Da wir im zweiten Semester einen bereits vorhandenen verwendeten, wurde der Simulator dadurch überflüssig.

1. Eine universelle statische Methode, die es ermöglicht, primitive, sowie komplexere Datentypen mit Hilfe von festgelegten Steuerzeichen aus einem ByteArray korrekt auszulesen.
2. Die Länge kann variieren. Das Einlesen einer Dateneinheit (Integer-wert, String...) wird mit erstem Auftreten eines „Start-Steuerzeichen" begonnen und endet sobald ein "Stopp-Steuerzeichen" ausgelesen wird.
3. Alle primitiven Datentypen werden zunächst in eine Zeichenkette umgewandelt. Anschliessend werden die ASCII Werte dieser Zeichenkette in ein Bytearray geschrieben.
4. Ein komplexer Datentyp wird als eine Abfolge von primitiven Datentypen interpretiert. Einzelne Elemente des komplexen Datentyps werden in dem ByteArray mit einem "Aufzählung-Steuerzeichen" getrennt.
5. Ähnlich wie bei komplexen Datentypen werden auch die Elemente eines Arrays oder eines Vektors durch spezielle "Trenn-Steuerzeichen" getrennt.

Ursprunglich war geplant, dass sich der `DownloadManager` für die Pakete `Upload` und `Upload_Start` registrieren sollte. Diese Pakete werden nur noch vom `EigeneDateienManager` empfangen. Wenn der `EigeneDateienManager` ein `Upload_Start`-Paket bekommt oder der Download abgeschlossen ist, werden beim `DownloadManager` die entsprechenden Methoden aufgerufen.

Erst war angedacht, dass Dateien sequentiell übertragen werden, und dass das Startbyte aufgrund der übertragenen Daten laufend aktualisiert wird. Da nicht sichergestellt werden kann, dass die Upload-Pakete in der richtigen Reihenfolge ankommen, sodass ein Array angelegt wird, das verwaltet, welche Dateiensegmente bisher übertragen worden sind (`rohDatenTeile[]`). Bei der jetzigen Modellierung enthält die `DownloadAnforderung` ein Startbyte von Typ `long`. Da wir während der Implementierungsphase die Schnittstellen nicht wesentlich ändern wollten, berechnen wir das Startbyte aus `rohDatenTeile[]`.

Beim `EigeneDateienManager` gab es Probleme beim Empfangen von Upload-Paketen. Ursprünglich war geplant, dass die Rohdaten einer Datei komplett gelesen werden und in einem Upload-Paket verschickt werden. Da aber der Speicher bei mobilen Endgeräten stark begrenzt ist und der Speicher des Emulators bei größeren Dateien nicht mehr ausgereicht hat, musste eine Alternative gefunden werden.

Die Dateien werden jetzt in Teilstücken versendet und müssen dann vom `EigeneDateienManager` wieder zusammengefügt werden. Ursprünglich war geplant, eine temporäre Datei im Dateisystem zu erstellen, die dieselbe Größe wie die zu übertagende Datei hat. Dies sollte gewährleisten, das man ankommende Upload-Pakete in beliebiger Reihenfolge in das Dateisystem schreiben kann.

Das Vergrößern der Datei auf die Zielgröße war in Tests aber viel zu ineffizient (das Erstellen einer 3MB großen Datei hat ca. 15 Minuten gedauert) sodass, die Upload-Pakete jetzt immer in der richtigen Reihenfolge geschrieben werden müssen.

Ein weiteres Problem ist, dass einige Aufrufe der FileConnection-API unter Windows nicht funktionieren. So funktioniert der Aufruf von z.B. `rename()` nicht. Dies hat zur Folge, dass unter Windows Systemen z.B. die Dateien nicht umbenannt werden können. Unter dem WTK 2.2 für Linux funktioniert der Aufruf von `rename()` hingegen ohne Probleme.

In der Entwurfsphase planten wir, die Datei, die übertragen werden sollte, in eine unserer "Upload-Nachrichten" zu packen. Bei der Implementierung wurde deutlich, dass dies bei großen Dateien den Arbeitsspeicher des Geräts überlasten kann. Aus diesem Grunde teilten wir den Versand einer Datei in mehrere UploadNachrichten auf.

6.3 Netzwerkebene

Ganz am Anfang war angedacht, dass SkyNet alle von unten empfangene Pakete nach oben an eine einzige Klassen weiterleiten sollte. Sehr früh stellte sich das als unpraktikabel und hinderlich raus, da sonst die Klasse Programm die Rolle eines Verteilers hätte annehmen müssen (→Bottleneck). So entschieden wir uns dafür, alle Manager ein Interface implementieren zu lassen und dieses dafür zu benutzen, die richtigen Pakete an die richtigen Klassen routen zu lassen[16].

Während der Datenübertragung kam es zu Problemen, die zu nicht mehr lesbaren ZIP-Archiven und korrumpierten MP3-Dateien führten. Die Ursache war ein Parsefehler von einem Object in ein Byte-Array in der Sender-Queue, bei dem Fragezeichen falsch geparset wurden. Als Lösung wurde eine neue Klasse implementiert, die den Adressaten und das Paket hält. Dadurch wird verhindert, dass beide in das Object-Array erst eingefügt werden.

Ein weiteres Problem war, dass zu viele Threads auf dem Endgerät später laufen würden. Wenn für jedes andere Gerät zwei Threads (Sender, Empfänger) laufen, sind das schon bei 8 Teilnehmern im Netz 16 Threads. Als Antwort darauf wurden alle Sender-Threads zu einem einzigen Thread gebündelt, der zusätzlich zu der Nachricht auch noch den Empfänger bekommt. Die Empfänger-Threads dagegen blockieren sich automatisch, wenn keine Nachricht auf ihrem Link anliegt. Dadurch wird ein effizientes Link-Scheduling gesichert.

Außerdem gab es noch das Problem, dass sich das empfangende Gerät nicht an die Maximum Recieve MTU[17] gehalten hat. Dadurch gab es große Löcher in den Daten, wenn der Sender weniger geschickt hat, als der Empfänger erwartete. Die Lösung war, die MTU nicht manuell zu setzen, sondern sich das Gerät selbst darum kümmern zu lassen.

Während der Planungsphase hatte sich die Projektgruppe dazu entschieden, für den Datenverkehr eine paketbasierte Verbindungsart zu benutzen, die bei Bluetooth L2CAP[18] heisst. Keine streambasierte Verbindung zu nehmen hatte den Hintergrund, dass L2CAP laut Spezifikation Flusskontrolle und Retransmission (etwa bei fehlerhaft übertragenen Paketen) zur Verfügung stellt. Bei der Implementierung traten dann allerdings zwei Probleme auf: Zum einen war es uns aufgrund des sehr begrenzten Arbeitsspeichers unmöglich, vollständige Dateien zu verschicken, da diese über L2CAP als ByteArray verschickt werden müssen (es gibt keine andere Methode). Wegen des beschränkten Arbeitsspeichers im mobilen Endgerätes, können nur Arrays bis zu einer bestimmten Größe (etwa 50KB) erstellt werden.

Das zweite Problem war, dass J2ME nicht die Fragmentierung großer Dateien übernimmt, um der MTU gerecht zu werden. Letzteres Problem war relativ einfach zu lösen. Mit Hilfe einer Methode lässt sich die MTU einer Verbindung abfragen. Es wurde somit ein kleines Array mit Größe der MTU erzeugt und dieses mit den Daten des ByteArray, welches von höheren Schichten an die Netzwerkschicht weitergereicht worden war, gefüllt und dann verschickt. Diesen Vorgang wiederholten wir solange bis das komplette ByteArray mittels kleiner, MTU-großer Pakete verschickt worden war. Für den Rest eines großen Paketes, also für die letzten Bytes, die in der Regel kein ganzes Array mehr füllten, wurde ein eigenes Array mit entsprechender Größe initialisiert und verschickt.

Auf der Empfängerseite mussten nun die Einzelteile wieder zusammengefügt werden und in einem großen ByteArray an die höheren Schichten weitergeleitet werden. Dazu muss allerdings die Größe bekannt sein um ein entsprechendes Array zu initialisieren und vor allem, um zu wissen, wieviele Bytes von der Leitung gelesen werden sollen bzw. wie oft auf ein MTU-großes Paket gewartet wird und wie großdann das "Restpaket" ist. Aus diesem Grund entschlossen wir uns vor jedem Paket ein 5-Byte-

[16] Siehe Abschnitt über Programmdesign, S. 18

[17] Maximum Transmission Unit - stellt die maximale Paketgröße dar, die auf einmal über eine L2CAP Verbindung geschickt werden kann

[18] Logical Link Control and Adaptation Protocol

großes Paket zu verschicken[19], in dem die Größe der folgenden Übertragung kodiert ist. Auf dieses wartet die Empfängerseite blockierend und fährt nach dem Erhalt dieses Paketes fort.

Das zweite zu lösende Problem war das Aufsplitten der Datei. Da große Pakete eine gewisse Zeit zur Übertragung brauchen, wir aber nicht das komplette Programm in einen Wartezustand legen wollten wenn ein großes Paket verschickt wird, führten wir eine Wartequeue für Pakete ein. Da wir aber wussten, dass zu große ByteArrays den Arbeitsspeicher des mobilen Endgerätes überfordern würden, mussten wir sicherstellen, dass immer nur ein Uploadpaket (alle anderen Pakete sind relativ klein) in der Sendequeue steht. Da aus Architekturgründen die Netzwerkschicht aber die Pakete nicht einsehen darf, um herauszufinden ob es sich um ein Uploadpaket handelt, musste sich eine obere Schicht merken, an welcher Stelle das Uploadpaket in der Sendequeue befindet. Als Lösung wollten wir die Größe der Sendequeue in einer Variablen abspeichern, nachdem das Paket eingefügt worden war, und diese Position jedesmal im 1 dekrementieren, wenn das Netzwerk eine Meldung gab, ein Paket verschickt zu haben. Wenn die Position dann von 1 auf 0 gesetzt wurde, würde der Uploadmanager benachrichtigt werden, dass er das nächste Uploadpaket in die Sendequeue einfügen darf. Dabei trat ein weiteres Problem auf.

Nachdem das zweite Uploadpaket verschickt worden war, bekam der Upload-Manager keine Nachricht mehr zugeschickt. Merkwürdigerweise trat dieser Fehler bei einem Mitglied der Projektgruppe nicht auf, während er bei allen anderen reproduzierbar war. Nach einer langen Reihe von Tests fand sich das Problem bei der Geschwindigkeit des Sender-Threads: Wie oben erwähnt speicherten wir die Position nachdem das Paket in der Sendequeue eingetragen worden war. Allerdings war der Thread so schnell, dass das Paket schon verschickt war, bevor wir auf die Länge der Queue zugriffen. Das führte dazu, dass wenn kein weiteres Paket in der Sendequeue stand die Position 0 abgespeichert wurde. Der Downloadmanager wurde aber nur benachrichtigt, wenn die Variable den Wert 1 hatte, also das Paket an erster Stelle gestanden hatte. Der Fehler wurde beseitigt indem die Queueposition vor dem Einfügen berechnet wurde (Länge der Queue plus eins).

[19] L2CAP ist ein paketorientiertes Verfahren, kein streamorientiertes. Das bedeutet, dass man nicht einfach alle Daten komplett in einen großen Strom bündeln und übertragen kann, sondern sie in entsprechende Stücke partitionieren muss.

7 Ergebnisse, 2. Semester

7.1 Implementierung des TSF Algorithmus

Da die J2ME-Spezifikation uns keine Möglichkeit bot, einen Scatternetzalgorithmus auf physikalischer Ebene zu implementieren, haben wir eine Adaption auf logischer Ebene vorgenommen.

Die Spezifikation bietet keine Einflussmöglichkeiten auf die Inquiry-Phase der Geräte. Es werden alle erreichbaren Geräte in eine Geräteliste eingetragen und anschließend in der ServiceDis-covery-Phase überprüft, ob diese unser Programm unterstüzten. Da es sich bei den Methoden um vom Listenerinterface DiscoveryListener vorgeschriebene Methoden handelt, können wir erst nach der Inquiryphase mit der Auswahl der möglichen Verbindungen ansetzten.

Wir haben die Geräte, die in der Inquiryphase gefunden wurden, zunächst daraufhin geprüft, ob bereits Verbindungen bestehen. Wenn dies der Fall ist, wird das Gerät sofort verworfen. Alle Geräte, zu denen noch keine Verbindung besteht, werden zur Geräteliste andereGeraete im Netzwerk hinzugefügt. Diese Liste wird an die Klasse Verwaltungsmanager, die die gesamte Verwaltung der Verbindungen übernimmt, übergeben.

Dem Verwaltungsmanager stehen verschiedene Typen von Verwaltungspaketen zur Verfügung, über die die Kommunikation im Scatternetz bzgl. des Verbindungsaufbaus geregelt wird. Nach der Inquiryphase wird anhand der aktuellen Rolle des Gerätes entschieden, wir weiter verfahren wird.

Ein freier Knoten muss nach der Inquiryphase erst die Geräte in der dem Verwaltungsmanager übergebenen Geräteliste durchgehen und jedem eine Anfrage als Verwaltungspaket zum Verbindungsaufbau senden. Erst, wenn ein Knoten eine Antwort auf diese Anfrage erhalten hat, wird das Gerät, welches positiv geantwortet hat, in die endgültige Geräteliste ElterundKinder im Netzwerk eingefügt und als Elter gekennzeichnet. Zu allen anderen Geräten, die in der andereGeraete-Liste standen, wird die physikalische Verbindung geschlossen und die Referenzen auf die Geräte gelöscht.

Nach diesem Prinzip wird auch nach Ende einer Inquiry eines Koordinatorknotens auf der Suche nach anderen Koordinatoren und Wurzelknoten auf der Suche nach der physikalischen Verbindung zu der zu verschmelzenden Wurzel verfahren.

Wenn BBH-Nachrichten über das Scatternetz geroutet werden müssen, darf das Gerät nur an seine direkten Kinder und seinen Elter senden. Diese Geräte befinden sich in der Geräteliste ElterundKinder. Soll eine Nachricht zu weiter im Baum entfernten Geräten geroutet werden, wird der Knoten, über den geroutet werden muss, mit Hilfe des Routingprotokolls bestimmt.

7.2 Adaption des BTR Protokolls

Das BTR Protokoll wurde in der Klasse Verwaltungsmanager implementiert. Dieser hält eine Instanz der Klasse Routingtabelle, welche die aktuellen Bluetoothadressen der topologisch darunterliegenden Kinder und Kindeskinder verwaltet. Die Routingtabelle beinhaltet eine Liste, in der sich immer sieben Einträge befinden. Der siebente Eintrag beinhaltet ein Stringarray, in dem die Bluetoothadressen der direkten Kinder und Kindeskinder eines direkten Kindes gespeichert sind. Zusätzlich zu den Bluetoothadressen werden die Anzahl der Hops gespeichert, d.h. wie weit die Knoten entfernt sind.

Wenn sich ein neues Gerät an einen Baumknoten anschließt, wird die BT-Adresse dieses Geräts als direktes Kind in die Routingtabelle des Baumknotens eingetragen. Es wird eine RoutingUpdate-Nachricht an den Elter geschickt, der die aktuelle Routingtabelle enthält. Der Elter ändert an der entsprechenden Stelle seine Routingtablelle und leitet das RoutingUpdateConnect an seinen Elter weiter. Dieser Vorgang wird fortgeführt, bis die Wurzel des Baumes erreicht ist.

Wenn ein Gerät bemerkt, dass die Verbindung zu einem direkten Kind abgebrochen ist, entfernt er dessen BT-Adresse aus seiner Routingtabelle und benachrichtigt seinen Elter durch eine RoutingUpdateDisconnect-Nachricht, welche wie die RoutingUpdateConnect-Nachrichten bis zur Wurzel hochgereicht werden muss.

Wenn eine Nachricht empfangen wird und die Zieladresse nicht die eigene BT-Adresse ist, wird in der Routingtabelle nach der Zieladresse gesucht. Die Routingtabelle liefert die BT-Adresse des direkten Kindes, in dessen Teilbaum sich das gesuchte Gerät befindet. An dieses Gerät wird die Nachricht gesendet.

7.3 Adaption an den Scatternetzalgorithmus

Um die korrekte Funktionsweise des Scatternetzalgorithmus garantieren zu können, mussten wir eine neue Nachrichtenform modellieren und implementieren. Diese haben wir „Verwaltungspaket“ genannt.

Wir unterscheiden zwischen Verwaltungsnachrichten, die ausschließlich dem Scatternetzalgorithmus zur Verfügung stehen und sich in der Netzwerksicht befinden, und unsere vorherigen Nachrichtentypen BBH-Nachrichten, die unserer Applikationsschicht angehören.

Sowohl BBH-Nachrichten, als auch Verwaltungsnachricht erben von der Klasse „Nachricht“. Die bisher für die eindeutige Konvertierung genutzte Methode `byteArrayToObjects()` wurde in die Klasse Nachricht verschoben. Der Aufbau von BBH-Nachrichten und Verwaltungsnachricht sind identisch. Allerdings sollte es eine klare Trennung zwischen Netzwerknachrichten(VerwaltungsPaket) und Nachrichten auf der Applikationsebene(BBHNachricht) geben. Die Standard-Attribute (physikalische Adressen des Initiators, des Senders und eigene Adresse) und einige Konstanten wurden in die Klasse Nachricht verschoben.

Im Unterschied zu BBH-Nachricht, wird im Verwaltungspaket kein Typ der Nachricht aus der Sicht der Vererbung beachtet. Aus Kompatibilitätsgründen wird dieser bei der Umwandlung in den Datenstrom dennoch gesetzt.

Um eine Verwaltungsnachricht von einer BBH-Nachricht zu unterscheiden, werden alle anstehenden Nachrichten in der Netzwerkschicht unmittelbar vor dem Absenden mit einem voranstehenden Bit ergänzt. Dies ermöglicht eine frühzeitige Erkennung einer Verwaltungsnachricht auf der Seite des Empfängers. Aus diesem Grund wurde auch die Methode `toStream()` ebenfalls zu der Klasse Nachricht verschoben und als abstrakt gekennzeichnet.

Über das Attribut nachrichtTyp_ der Verwaltungsnachricht kann übermittelt werden, für welchen Zweck die Nachricht eingesetzt wird. Alle möglichen Aufgaben (Zwecke) wurden als Konstanten eindeutig festgelegt und in der Klasse VerwaltungsNachricht abgelegt. Zusätzlich wird noch im Attribut rolle_ die eigene Bluetooth-Rolle, ob das versendende Gerät ein Master oder Slave ist, kodiert.

7.4 Backup-Server

Die Implementierung des Backup-Servers verlief größtenteils ohne Probleme. Wir haben die Java-Klassen des J2ME-Programmes übernommen und lediglich den Sender und den Empfänger für die Server-Version angepasst. Das Resultat war ein zweites Programm, das als Standalone auf dem Server aufgesetzt wird und auf Verbindungen wartet. Die Kommunikation läuft über standard-TCP/IP.

7.5 Erweiterte Suche

Im ersten Semester war die Suche auf die OR-Suche beschränkt, und es konnten keine Substrings gefunden werden (z.B. bei der Suche nach "text", wurde "meintext" nicht gefunden).

Die erste Erweiterung ist die AND-Suche. Um die bestehende Struktur so wenig wie möglich zu verändern, haben wir die Unterscheidung zwischen AND- und OR-Suche über das erste Keyword, das in der Suchanfrage verschickt wird, vorgenommen. Auf dem Bildschirm „Neue Suche" kann man jetzt die Art der Suche festlegen. Wenn dort AND angewählt ist, trägt die GUI automatisch in dem Keyword-Array, welches über das SuchAnfrage-Paket verschickt wird, das "keyword" AND an der ersten Stelle ein. Bei der OR-Suche steht an der ersten Stelle entsprechend das Keyword OR.

Auf der Gegenseite wird beim Empfang einer SuchAnfrage im EigenenDateienManager die Methode `sucheKeywords()` aufgerufen, welche eine SuchAntwort zurückliefert. Hier wird immer zwischen der AND- und OR-Suche unterschieden.

Der wesentliche Unterschied zum ersten Semester liegt darin, dass wir anstatt der Methode `gibDateiname()` die neu implementierte Methode `gibAlleIndexEintraegeZuKeyword()` benutzen.

Die Methode `gibAlleIndexEintraegeZuKeyword()` wurde hauptsächlich deswegen hinzugefügt, damit wir die Substring-Suche realisieren können. Bei einer Suche ohne Substring-Unterstützung kann es pro Keyword immer nur einen Treffer (MatchListenEintrag) in der SuchIndexListe geben. Nun ist es aber möglich, dass ein Keyword (z.B. text) innerhalb mehrerer MatchlistenEintraege (z.B. meintext, neuertext, unseretexte) vorkommt. Dazu liefert die Methode "gibAlleIndexEintraegeZuKeyword" einen Vector mit allen gefundenen MatchListenEintraegen zurück. Für die Substring-Suche haben wir die Methode `indexOf(String)`, die in J2ME enthalten ist, benutzt.

Während der Debugging-Phase traten einige Probleme auf:

- SuchAntwort/hinzufuegenSuchAntwortEintrag: mit jedem hinzufuegen eines SuchAntwortEintrags bei einer SuchAnwort wurde die Anzahl der SuchAntwortEintraege verdoppelt.
- EigeneDateienManager/SucheKeywords: in einigen Fällen hat die Methode `sucheKeywords()` eine NullPointerException verursacht, die dazu führte, dass ein Knoten zum freien Knoten wurde.

7.5.1 ID3-Tags

Zum besseren Auffinden von Audiodateien im "mp3"-Format wurde beschlossen, die ID3-Tags zusätzlich aus den mp3-Dateien zu extrahieren. Zum jetzigen Zeitpunkt werden dennoch nur die ID3-Tags gemäßVersion 1.0 unterstützt. Der Unterschied der beiden Versionen 1.0 und 2.x liegt in der festen Position von Tags innerhalb der Datei. Da bei der Version 2.x die Feldgröße der Tags und dementsprechend die Position dynamisch ist, hätte ein Algorithmus zum Auslesen die Rechenleistung eines mobilen Geräts überlastet.

Als erstes wurde die Klasse ID3 realisiert, die den Titel, den Interpreten und Albumnamen zu einem Dateinamen ausliest. Um die Implementierung schlank zu halten, werden die ID3-Tags als String mit dem vorhandenem Dateinamen konkateniert um anschließend die vorhandenen Methoden zum Keyword-Generieren benutzen zu können.

Schwierigkeiten der Implementierung traten beim Positionieren des Zeigers innerhalb des Inputstreams einer Datei auf. Leider funktionierte die J2ME-eigene Methode zum selben Zweck nicht ordnungsgemäß. Hilfreich war dort die noch im ersten Semester implementierte Methode `positioniereStream()` aus der Klasse EigeneDateienManager.

7.5.2 Upload-Prioritäten und Credit-System.

Eine Möglichkeit das Download/Upload-System fair zu organisieren, bietet das bereits erprobte Credit-System, welches erfolgreich bei p2p-Netzwerken wie eMule eingesetzt wird, gute Ansätze. Der Gedanke dabei ist folgender: die Warteschlange zum Upload beim Uploader wird nach Punkten (Credits) sortiert, welche für bereits hochgeladene Datenmengen, Wartezeit vom Uploader und der jeweiligen Da-

teipriorität der gewunschten Datei vergeben werden. Die Punkte werden für jeden Peer, bei welchem heruntergeladen wurde, beim Empfänger gespeichert. Z.B. ein Peer A lädt eine Datei "datei1" beim Peer B, eine Datei "datei2" beim Peer C und eine Datei "datei3" beim Peer D. Somit bekommen die Peers B, C und D beim Peer A die Punkte. Dennoch handelt es sich um ein lokales System, d.h. nach dem ein Gerät das Programm verlassen hat, gehen die Credits verloren.

Die Formel zur Berechnung lautet:

$$\text{CREDITS} = \text{WARTEZEIT} \times \text{MODIFIER} \times \text{DATEIPRIORITÄT}$$

- WARTEZEIT ist die Zeit seit dem Eintragen des Peers in die Warteschlange.
- MODIFIER = 2 × UPLOADUMFANG / DOWNLOADUMFANG
- DATEIPRIORITÄT wird direkt aus der GUI vom Benutzer vergeben.

Demnach wurde die Klasse Peer um weitere zwei Attribute anzahlAngekommene-Pakete_ und anzahl-VersendetePakete_ erweitert. Diese werden immer dann hochgezählt, sobald ein neues Datenpaket ankommt und vom EigeneDateienManager verarbeitet wurde (anzahlAngekommenePakete), oder falls ein Paket versandt wird (anzahlVersendetePakete). Die Klasse UploadEintrag wurde um Attribut eintragszeit_ erweitert, um die entsprechende Wartezeit ermitteln zu können (Wartezeit = Aktuelle Zeit - Eintragszeit). Die Creditsberechnung erfolgt mit der neuen Methode der Klasse UploadEintrag.

Nach einem abgeschlossenem Upload werden die Credits für alle DownloadEintraege neu berechnet und die UploadListe wird neu sortiert (dafür wurde der Quicksort-Algorithmus implementiert).

7.6 BBH-Light™

Diverse Testergebnisse führten während unserer Arbeit zu der Annahme, dass das WirelessToolkit in einigen Bereichen noch nicht ganz ausgereift war und sich reale Geräte entsprechend anders verhalten würden. Da uns bei der Suche nach geeigneten Testgeräten (mit JSR 75 Unterstützung) die Projektleitung nicht helfen konnte, entstand die Idee, eine BBH-Light-Version zu entwerfen, die auch auf uns verfügbaren Endgeräten laufen würde.

Zu diesem Zweck wurden alle JSR 75 Anteile aus dem BBH-Projekt entfernt und das übrige Programm entsprechend angepasst. Am Ende stand eine abgespeckte Version des BBH-Projektes zur Verfügung, mit der zwar kein Datenaustausch stattfinden konnte, aber immerhin das reale Bluetoothverhalten getestet werden konnte. Als Testgeräte standen drei Nokia 6230 und ein Siemens S65 zur Verfügung.

Hier die zusammengefassten Ergebnisse unserer Tests:

1. Die realen Endgeräten hatten kein Problem damit, das Bluetooth-Netz erneut aufzubauen, nachdem das Master-Gerät ausgeschaltet wurde. Dies ist ein entscheidender Unterschied zum WTK Emulator 2.2, der nach der Schließung des Master-Gerätes den Dienst versagte.
2. Die Nokia Geräte haben nur die Möglichkeit, eine eins-zu-eins-Verbindung herzustellen. Diese Beobachtung wurde bei der darauffolgenden Recherche im Nokia Developer Forum bestätigt. Nokias Serie 60 Geräte können hingegen Point-to-Multipoint-Verbindungen verwalten. Ebenso das S65.
3. Das Auffinden von Geräten und Eintragen in die Geräteliste funktionierte wie erwartet und konform zum Emulator.

8 Leistungsmessungen

Die Messungen wurden auf einem Windows-Rechner P4 durchgeführt. Die Werte gelten für die simulierten Geräte und können mit denen auf realen Geräte abweichen.

8.1 Anwendungsfälle

8.1.1 Freigabe von Dateien

Eine Datei wird freigegeben, damit andere Peers sie laden können. Während diesem Vorgang wird die Datei auch gehasht und entsprechende Suchdaten extrahiert.

- Kleine Dateien (< 5 MB) < 50 sek.
- Mittlere Dateien (5 MB - 10 MB) 50 sek. - 1min 57 sek.
- Große Dateien (> 10 MB) > 1min. 57 sek.

8.1.2 Suche von Dateien

Dateien werden im Netzwerk nach Keywords gesucht. Jeder Peer muss dabei eine komplette Liste seiner freigegebenen Dateien durchgehen und nach den entsprechenden Wörtern suchen.

- Kleine Dateien(< 5 MB) < 2 sek.
- Mittlere Dateien (5 MB - 10 MB) < 2 sek.
- Große Dateien (> 10 MB) < 2 sek.

8.1.3 Download von Dateien

Ein Datei wird von einem Client zu einem anderen Übertragen. Dies kann direkt oder über mehrere Hops geschehen.

- Kleine Dateien (< 5 MB), einzeln < 1min 35 sek.
- Mittlere Dateien (5 MB - 10 MB), einzeln 1min 35sek. - 3min 09 sek.
- Große Dateien (> 10 MB), einzeln > 3min 09 sek.
- Kleine Dateien (< 5 MB), mehrere < 3min 40 sek.
- Mittlere Dateien (5 MB - 10 MB), mehrere 3min 40sek. - 5min 38 sek.
- Große Dateien (> 10 MB), mehrere > 5min 38 sek.

8.1.4 Resume von Dateien

Ein Dateidownload wurde pausiert (manuell oder durch Verbindgunsabbruch). Zu einem Resume wird eine erneute Suchanfrage in das Netz geschickt und der beste Peer mit der vorrätigen Datei gewählt und für einen Download angesprochen.

- Download wiederaufnehmen kleiner Dateien (< 5 MB) < 8 sek.
- Download wiederaufnehmen mittlerer Dateien (5 MB - 10 MB) < 8 sek.
- Download wiederaufnehmen großer Dateien (> 10 MB) < 8 sek.

8.1.5 Tree Scatternet Formation

Gestartete Endgeräte verbinden sich zu einem großen Baum mittels TSF. Es wird getestet, wie lange es dauert, bis ein neuer Client in das Netzwerk eingebunden wird, sich zwei separate Bäume zu einem großen verschmelzen und ein beschädigtes Netz sich selbständig wieder heilen kann.

- Anschluss Freierknoten < 7 sek.
- Baumverschmelzung < 15 sek.
- Scatternetz-Heilung < 15 sek.

9 Diagrammbeschreibungen, 1. Semester

9.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm weist drei Akteure auf: Benutzer, System und den Fallback-Server. Der Benutzer kann seine eigenen Dateien managen, wie etwa eine Datei freigeben und sie wieder sperren, oder eine Auflistung seiner freigegebenen Dateien sehen. Zusätzlich kann er nach Dateien suchen und sie downloaden. Die Downloads können gestartet/fortgeführt, abgebrochen und angehalten werden. Alle diese Anwendungsfälle fallen unter dem Oberanwendungsfall "Dateien austauschen". Zusätzlich fallen unter diesen Anwendungsfall das Managen von einkommenden Verbin-dungen, sprich Uploads einsehen und upload abbrechen.

Der Akteur System muss auf Such- und Downloadanfragen reagieren. Darüberhinaus muss er sich um den Aufbau und die Aufrechterhaltung der Verbindung kümmern.

9.2 Aktivitätsdiagramme

9.2.1 Suchanfrage erstellen

Bei der Suchanfrage werden Keywords eingegeben. Zuerst werden sie auf ihre Korrekheit überprüft. Bei falschen Eingaben wird erneut aufgefordert die Suchwörter einzugeben. Anderfalls wird die Verbindung geprüft und aufgebaut, wenn dies noch nicht getan wurde. Anhand der Keywords, der GeräteID und der SuchID wird die Suchanfrage generiert und anschließend geschickt. Der nächste Schritt startet den Timer und wartet auf Antworten. Solange der Timeout noch nicht erreicht wird, bleibt man in diesem Zustand. Falls jedoch der Timeout erreicht wird und keine Antwort erhalten wurde, wird der Fallback-Server eingeschaltet. Beim Erhalt einer Antwort wird sie auf die richtige ID überprüft. Wenn eine falsche SuchID vorliegt, wird in den wartenden Zustand zurüchgekehrt. Bei richtiger SuchID, werden die Antworten in die Ergebnisliste eingetragen und angezeigt. Weiterhin wird auf weitere Antworten gewartet falls der Timeout noch nicht abgelaufen ist, ansonsten ist man am Ende der Suchanfrage gelangen.

9.2.2 Download starten

Es wird hierbei davon ausgegangen, dass nach einer erfolgreichen Suche ein Download zu Download-Liste hinzugefügt und nicht mit SofortStarten gestartet wurde. Eine erneute Broadcast-Anfrage wird nochmals versendet und auf einen Treffer gewartet. Hier gibt es die Möglichkeit, zum Fallback-Server auszuweichen. Falls genug Speicherplatz zur Verfügung steht, wird eine Sendeaufforderung an Dateianbieter abgeschickt und auf eine Antwort mit QueuePosition 1 gewartet. Schließlich werden die Nutzdaten übertragen, auf Platte gespeichert, und die Datei aus Downloadliste entfernt und zu Freigabeliste hinzugefügt.

9.2.3 Auf Suchanfrage Reagieren

Dieses Diagramm wurde im EigeneDateienManager implementiert. Es beschreibt die Reaktion des Programmes wenn ein SuchAnfrage-Paket im System eintrifft. Das Programm wartet in einer Schleife, bis eine Suchanfrage eintrifft. Die angekommene Suchanfrage wird dann verarbeitet, indem in der SuchIndexListe überprüft wird, ob diese Datei vorhanden bzw. freigegeben wurde. Danach wird die Queue Position ermittelt und eine passende SuchAntwort generiert und verschickt.

9.2.4 Auf DownloadAnfragen reagieren

Dieses Diagramm beschreibt die Dateianbieterperspektive und die diversen Aktivitäten beim Empfang von bestimmten Nachrichten. Bei DownloadAnfragen wird mit dem Versand einer DownloadAntwort reagiert. Bei DownloadAnforderungen (Sendeaufforderungen) wird die Anforderung in die UploadListe (Queue) aufgenommen und gegebenenfalls wird die Datei versendet. Bei Uploadabbrüchen durch eine

der beiden Seiten sowie bei erfolgreicher Beendigung des Uploads wird der UploadEintrag aus der UploadListe entfernt.

9.2.5 Verbindung aufbauen

Dies ist das Diagramm, das den initialen Aufbau aller Verbindungen beschreibt. Zuerst wird eine Referenz auf das eigene Gerät gebildet, damit damit gearbeitet werden kann. Dann werden die eigenen Services registriert und ein Service Record kreiert. Anschließend folgt die Inquiry und Service-Search-Phase.

Wir gehen hier eigentlich von zwei grundlegenden Zuständen aus, in denen sich das Gesamtsystem befinden kann:

Wenn das momentane Gerät das erste ist, das das Programm gestartet hat, findet es keine weiteren Geräte während der Inquiry-Phase. Alle temporären Listen bleiben leer und es öffnet mit `acceptAndOpen()` die Serververbindung.

Wenn es bereits ein bestehendes Netzwerk gibt, wird durch die Inquiry und anschließende Service-Search-Phase eine Verbindung zu den übrigen Geräten aufgebaut (und zwar nur zu denen, die auch das richtige Programm laufen haben). Nach diesem uplink als „slave" geht das Gerät in seinen eigenen Wartemodus und stellt sich ebenfalls als „server" zur Verfügung, indem es in `acceptAndOpen()` geht und auf einkommende Verbindungen wartet.

9.2.6 Verbindung aufrechterhalten

Das Programm checkt, ob ein Server ausgefallen ist und wenn ja, ernennt einen neuen Master. Durch das Kapselungsproblem kann das nur auf der logischen Ebene überprüft werden. Diese besteht allerdings bei uns aus point-to-point socket Verbindungen, bei denen es keinen Master im eigentlichen Sinne gibt.

9.3 Problembereichsmodell

(siehe Klassendiagramm)

9.4 Klassendiagramm

9.4.1 Klassendiagrammbeschreibung

Um die vielfältigen Aufgaben unseres Programms besser zu strukturieren, entschieden wir uns schon zu Beginn der Entwurfsphase für eine klassenbasierte Aufteilung - ein sogenanntes Managersystem. Wir benutzten dabei die Unterteilung des Anwendungsfalldiagramms. Es entstanden SuchManager, EigenDateienManager, DownloadManager und UploadManager. Im folgendem werden nur die Zusammenhänge zwischen den wichtigsten Klassen beschrieben. Eine ausführliche Klassenbeschreibung aller Klassen ist im nächsten Abschnitt zu finden.

Die Kommunikation zwischen den einzelnen Managern der jeweiligen Geräte findet über verschiedene Nachrichtentypen statt. `BBHNachricht` ist die Oberklasse unserer Nachrichtentypen, die jeweils von bestimmten Managern bearbeitet und versendet werden. Die Nachrichtentypen sind: `DownloadAnfrage`, `DownloadAntwort`, `DownloadAnforderung`, `DownloadAbbruch`, `SuchAnfrage`, `SuchAnwort`, `Upload` und `UploadStart`.

Der SuchManager erstellt SuchAnfragen und bearbeitet alle einkommenden SuchAntworten. Zur Verwaltung dieser Nachrichten benutzt er die Klasse SuchMatchListe, in der die MatchListenEinträge gehalten werden. Zu jedem MatchListenEintrag werden die Downloadquellen mit Verweisen auf die entsprechenden Peers gespeichert.

Der EigeneDateienManager beantwortet eingehende SuchAnfragen, speichert eingehende Downloads und verwaltet die Dateifreigabe in der FreigabeListe. In ihr werden die Dateien gehalten.

Der DownloadManager verwaltet die Downloads unserer Anwendung. Er erstellt DownloadAnfragen und DownloadAnforderungen. Dazu benutzt er die Klasse DownloadListe, die die DownloadEinträge verwaltet.

Der UploadManager stellt die Gegenseite des DownloadManagers dar. Er bearbeitet eingehende DownloadAnfragen und DownloadAnforderungen. Außerdem erstellt und verwaltet er die Uploads. Um auf die Uploads zuzugreifen, verwendet der Manager die UploadListe, die die einzelnen UploadEintraege enthält.

Die Manager implementieren das Interface Nachrichtenmanager. Sie müssen die Methode `empfangeNachricht()` implementieren, damit über diesen Methodenaufruf die Nachrichten an den dafür vorgesehenen Manager gesendet werden kann.

SkyNet ist die Verbindung zwischen Netzwerkschicht und Applikationsschicht und leitet eingehende Nachrichten über die Methode `empfangeNachricht()` an die Manager weiter, die sich zu Beginn bei SkyNet für ihre Nachrichten registiert haben.

Programm erbt von Midlet und ist der Rumpf unserer Anwendung. Diese Klasse initialisiert alle Manager, den Timer, die GUIKontrolle und Skynet. Im Programm halten wir eine Instanz von BBHTimer, der allen Managern zur Verfügung steht und alle Timingaufgaben unseres Programms übernimmt. BBHTimer hat einen Vektor von BBHTimerEintrag, den er bei jedem Aufruf von der Methode starteTimer(), um einen neuen Eintrag erweitert. Er besitzt weiterhin einen BBHTimerTask, welcher sekündlich überprüft, ob ein Zeitpunkt von einem der BBHTimerEinträgen überschritten worden ist. Falls dies der Fall ist, wird der jeweilige Manager über den EreignisHandler benachrichtigt.

GUIKontrolle dient als visuelle Schnittstelle für den Benutzer.

Auf der Netzwerkschicht wird durch die Klasse Netzwerk ein Thread der Klasse Empfaenger für jede eingehende Verbindung, gelinkt auf das jeweilige entfernte Endgerät erstellt. Die Endgeräte werden in der GeraeteListe verwaltet. Zum Senden aller Nachrichten dient ein weiterer, einzelner Thread der Klasse Sender, der je nach Bedarf aktiviert wird. Außerdem stehen die Klassen Listener und ServiceSearch zur Verfügung, die die Bluetooth-spezifische Aufgaben für das Finden anderer Geräte erfüllen (ServiceSearch, Inquiry etc.).

9.4.2 Klassenbeschreibung

Die Klassen sind semantisch geordnet in einem Top-Down-DFS.

Programm Die Klasse Programm erbt von MIDlet und ist damit die "Startklasse". Beim Programmstart werden Instanzen von allen Mangern (DownloadManger, EigeneDateienManager, UploadManager und SuchManager) erstellt, sowie von SkyNet und GuiKontrolle. Außerdem hält sie einen Vector aller Peers, von denen ein Download möglich ist.

EigeneDateienManager Die Klasse EigenDateienManager implementiert das Interface Nachrichten-Manager, um BBHNachrichten empfangen zu können. Dafür registriert sich die Klasse bei SkyNet. Der EigenDateienManager ist zuständig für die Verwaltung der freigegebenen Dateien und des Suchindexes. Diese Klasse enthält weiterhin eine Methode mit der der Suchindex durchsucht werden kann, um auf eine Suchanfrage zu reagieren.

FreigabeListe Die FreigabeListe hält einen Vector der Dateien, die freigebenen wurden und verwaltet diese.

Datei Jede Datei, die von dem Benutzer freigegeben wurde, wird mit dieser Klasse beschrieben. Die Attribute, die in Datei zur Verfügung gestellt werden, beinhalten alle wichtigen Informationen die wir für eine Datei benötigen. Es besteht eine Komposition zu IndexEintrag. Sobald eine Datei erstellt wird, wird anhand des Keywords ein IndexEintrag erstellt.

MessageDigest5 Berechnet Hashwerte mit Hilfe von Algorithmus "MessageDigest5". Hat eine Methode, die eine URL erwartet und einen fertigen Hashwert als String zurückliefert.

IndexEintrag Der Eintrag verwaltet für ein Keyword alle Dateien, die dieses Keyword enthalten.

UploadManager Diese Klasse implementiert NachrichtManager und EreignisHandler. Sie ist bei SkyNet registriert, um BBHNachrichten zu empfangen. Diese Klasse verwaltet eine UploadListe mit UploadEinträgen, um auf DownloadAnfragen antworten zu können und bei DownloadAnforderungen den Upload zu starten. Außerdem reagiert sie auf die BBHNachrichten DownloadAbbruch und UploadAntwort.

UploadListe UploadListe erbt von Vector. Jeder Upload wird als UploadEintrag in der Liste repräsentiert.

UploadEintrag Die Klasse UploadEintrag hält als Attribute den Dateinamen, die Warteposition, den Peer, der die Datei anfragt und das Startbyte.

TmpEintrag Diese Klasse wird benötigt, um temporäre Dateien zu verwalten. Temporäre Dateien werden erzeugt, wenn ein Uiplod/Upload_Start-Paket hereinkommt. Die Rohdaten werden in diesen temp. Dateien gespeichert, damit sie nicht im Speicher gehalten werden müssen.

Quicksort Die Klasse ermöglicht das Sortieren der UploadListe. Als Kriterium für die Position werden die Credits benutzt.

SuchManager Der Suchmanager ist für die Suche verantwortlich. Er verschickt SuchAnfrage-Pakete und empfängt SuchAntwort-Pakete. Er verwaltet eine Liste von Suchmatchlisten für den Fall, dass mehrere Suchanfragen gesendet werden. Jeder Suchmatchliste wird eine SuchID zugeordnet.

SuchMatchListe Die SuchMatchListe speichert nach einer Suchanfrage die Ergebnisse. Diese werden in Form von MatchListenEinträgen verwaltet.

MatchListenEintrag Zu jeder gefundenen Datei, die durch einen Hashwert identifiziert wurde, gibt es genau einen MatchListenEintrag. Jeder MatchListenEintrag verwaltet eine Liste von DownloadQuellen, von denen man die Datei runterladen kann.

DownloadQuelle Die DownloadQuelle beinhaltet eine Referenz auf einen Peer und die entsprechende Warteposition.

SuchIndexListe Hierbei handelt es sich um eine Datenstruktur, die Keywords verwaltet. Wenn eine neue Datei in die FreigabeListe eingefügt und die Keywords für sie erzeugt wurden, wird die Dateireferenz mit den dazugehörigen Keywords in die SuchIndexListe eingefügt. Wir haben also ein Keyword und alle dazugehörigen Dateienreferenzen.

ID3 In der Klasse ID3 werden aus einer MP3-Datei die ID3-Tags Titel, Artist und Album extrahiert und zur Keywordliste der Datei hinzugefügt.

Peer Der Peer ist die Klasse für das eindeutige Verwalten von mobilen Geräten, die Dateien im Netzwerk freigegeben haben und diese auch tauschen wollen.

DownloadManager Die Klasse DownloadManager implementiert das Interface NachrichtenManager, um BBHNachrichten empfangen zu können. Der DownloadManager ist für alle Belange des Downloadens zuständig. Er verwaltet eine Downloadliste, deren Einträge (DownloadEintrag) die Dateireferenzen enthalten, die heruntergeladen werden sollen. Er empfängt die BBHNachrichten DownloadAntwort, DownloadAbbruch und UploadStart und reagiert auf diese.

DownloadListe Die DownloadListe verwaltet einen Vector von DownloadEinträgen.

DownloadEintrag Ein DownloadEintrag repräsentiert eine Datei, die von einem Peer heruntergeladen werden kann. Dazu enthält die Klasse die Attribute Dateiname, Hashwert der Datei, die Dateigröße, den Dateityp, den Zeitstempel, ein boolesches Attribut, ob die Datei modifiziert wurde, die url und das Startbyte.

EreignisHandler Dies ist ein Interface, das alle Klassen, die einen Timer benutzen, implementieren. Diese implementieren die Methode `timerEvent(int nachricht)`, über die der BBHTimerTask die Klassen nach Ablauf des vorher in der Klasse BBHTimer registrierten Timers benachrichtigen kann.

BBHTimer Die Klasse BBHTimer bietet allen Klassen des Programms, die einen Timer benötigen (SuchManager, DownloadManager und GUIKontolle), die Möglichkeit sich über Timeouts informieren zu lassen. Der Manager trägt einen Timeout mit Hilfe der Methode `starteTimer(Ereignis-Handler anforderer, int sekunden, int nachricht)` ein. Dabei übergibt er sich selbst als EreignisHandler, die Zeit, wie lange der Timer laufen soll und die Nummer der Nachricht, um nach dem Ablauf des Timers diesen wieder einer Nachricht zuordnen zu können.

Der BBHTimer hält außerdem noch einen Thread der Klasse BBHTimerTask für die Abarbeitung der Timer.

BBHTimerEintrag Für jeden in BBHTimer eingehenden Timeraufruf wird ein BBHTimerEintrag erstellt.

BBHTimerTask In diesem Thread wird zyklis überprüft, ob ein Benachrichtigungszeitpunkt überschritten wurde. Falls ja, wird der entsprechende Anforderer mittels Methodenaufruf der Methode `timerEvent(int nachricht)` des Interfaces EreignisHandler benachrichtigt, in dem die entsprechende Nachrichtennummer zur Identifizierung übergeben wird.

Nachricht Nachricht ist die Oberklasse der Klasse BBHNachricht der Klasse VerwaltungsPaket. Sie liefert eine abstrakte Methode `toStream()`, die in den Unterklassen implementiert wurden, um aus den Objekten ein Bytearray zu machen. Außerdem liefert sie die Attribute zum Setzen der eigenen Bluetoothadresse, der Bluetoothadresse des Suchenden, wenn es sich um eine weitergeroutete Nachricht handelt und der Zielbluetoothadresse.

BBHNachricht Die Klasse BBHNachricht erbt von der Klasse Nachricht und ist selbst die Oberklasse für alle weiteren BBHNachrichten, die der Kommunikation innerhalb des BBH Systems dienen.

Die Methode `toBBH(stream:Byte[])` ist statisch und ermöglicht BBHNachrichten des entsprechenden Typs aus einem Bytearray zusammengestellt zu werden. Um eine BBHNachricht zusammenzustellen wird der Konstruktor einer BBHNachricht aufgerufen, der für den gewünschten Typ zuständig ist.

SuchAnfrage Die Klasse SuchAnfrage erbt von der Klasse BBHNachricht. Sie implementiert die Methode `toStream()`. Zusätzlich enthält diese Klasse Attribute für eine SuchID, Keywords und einen Timeout. Eine Suchanfrage wird per Broadcast an alle erreichbaren Peers gesendet.

SuchAntwort Die Klasse SuchAntwort erbt von der Klasse BBHNachricht. Die Klasse SuchAntwort besitzt als zusätzliches Attribut einen Vector von Objekten der Klasse SuchAntwortEintag, die SuchID und die Warteposition. Als Resultat für eine SuchAnfrage wird eine SuchAntwort von jedem Peer erwartet.

SuchAntwortEintrag Die Klasse SuchAntwortEintrag stellt eine Antwort auf eine Suchanfrage dar. Sie enthält als zusätzlicheAttribute den Dateinamen, die Dateigröße und den Hashwert der Datei zur eindeutigen Identifizierung.

DownloadAnfrage Die Klasse DownloadAnfrage erbt von der Klasse BBHNachricht. Als zustätzliches Attibut enthält die DownloadAnfrage den Hashwert der gesuchten Datei. Sie wird vom Peer geschickt, der etwas downloaden möchte, nachdem bereits eine DownloadAntwort eingegangen ist. Wenn ein Download aus der DownloadListe gestartet wird, wird diese Nachricht per Broadcast an alle erreichbaren Peers geschickt. Als Antwort wird eine DownloadAntwort erwartet.

DownloadAnforderung Die Klasse DownloadAnforderung erbt von der Klasse BBHNachricht. Sie enthält zusätzlich die Attribute Hashwert und das Startbyte. Nachdem eine DownloadAnfrage geschickt wurde, werden im DownloadManager einige Peers ausgewählt, von denen die Datei herunterladen werden kann (Das Auswählen geschieht nach den Kriterien „Warteposition“ und „Hops“). An diese Peers wird dann eine DownloadAnforderung geschickt. Es wird eine DownloadAntwort erwartet.

DownloadAntwort Die Klasse DownloadAntwort erbt von der Klasse BBHNachricht. Die zusätzliche Attribute der Klasse sind die Warteposition, der Hashwert, die Anzahl der Hops, wie weit der Peer im Netzwerk entfernt liegt und das boolsche Attribut istDownloadAnfrage, das zeigt, ob die Anfrage beim anbietenden Peer in der Queue aufgenommen worden ist. Die DownloadAntwort wird entweder als Antwort auf auf eine DownloadAnfrage oder eine DownloadAnforderung geschickt.

DownloadAbbruch Die Klasse DownloadAbbruch erbt von der Klasse BBHNachricht und enthält zusätzlich die Attribute Hashwert und Richtung, die angibt, ob sie von einem Dateianbieter oder Dateisuchenden geschickt wurde. Ein DownloadAbbruch wird geschickt, wenn ein Peer einen anderen Peer aus seiner UploadListe bzw. Downloadliste entfernt hat. Wenn ein suchender Peer einen DownloadAbbruch empfängt, löscht er die zugehörige DownloadQuelle des Peers. Wenn ein anbietender Peer einen DownloadAbbruch empfängt, löscht er die Anfrage aus seiner Queue.

UploadStart Die Klasse UploadStart erbt von der Klasse BBHNachricht und enthält zudem die Attribute AnzahlPakete und Hashwert. UploadStart ist eine Klasse, die für die Ankündigung von Upload benötigt wird. Wenn ein Empfänger diese Nachricht bekommt, kann er sich bei den anderen Queues mit Hilfe eines DownloadAbbruch abmelden.

Upload Die Klasse Upload erbt von der Klasse BBHNachricht. Die Uploadnachrichten beinhalten zusätzlich die Startposition innerhalb der angeforderten Datei, ein Bytearray mit Rohdaten, den Hashwert und die aktuelle Länge der enthaltenden Rohdaten. Falls ein Upload zu einer Gegenstelle erfolgen soll, wird die Orginaldatei auf mehrere Uploadnachrichten aufgeteilt und an SkyNet übergeben.

UploadAntwort Die Klasse UploadAntwort erbt von der Klasse BBHNachricht. Sie beinhaltet das Attribut Hashwert zur eindeutigen Identifizierung und wird jeweils auf eine BBHNachricht Upload gesendet. Erst wenn eine UploadAntwort wieder beim sendenden Peer ankommt, wird der nächste Upload verschickt.

VerwaltungsPaket Die Klasse VerwaltungsPaket erbt von der Klasse Nachricht. Über das Attribut `nachrichtTyp_` der Verwaltungsnachricht kann übermittelt werden, für welchen Zweck die Nachricht eingesetzt wird. Alle möglichen Aufgaben (Zwecke) wurden als Konstanten eindeutig festgelegt und in der Klasse VerwaltungsNachricht abgelegt. Darunter sind:

AnhaengeAnfrage Damit wird erfragt, ob ein Gerät von einem anderen Gerät als Kind anhängen kann. Fall nach einer gewählten Zeit der Timer ausläuft, wird ein neues Gerät aus der Liste der gefundenen zufällig ausgesucht und diesem eine AnhaengeAnfrage geschickt. Wenn diese Nachricht auf der Empfängerseite ankommt, muss eine AnhaengeAntwort zurückgeschickt, der Status eventuell auf Bridge und unter Umständen die Rolle geändert werden.

AnhaengeAntwort Die AnhaengeAntwort wird von dem Empfänger der AnhaengeAnfrage zurückgeschickt. Falls die Rolle des Empfängers ein Wurzelknoten ist, oder der Knoten die Kapazität von 7 Kinderknoten erreicht hat, ist eine Verbindung bzw. das Anhängen nicht mehr möglich. Falls dieser Fall eintritt, wird in dem Attribut daten_ eine "0" übermittelt, sonst eine "1".

RoutingUpdateConnect Wenn der Baum erweitert wurde, wird an den Elter die neue Knoteninformation hochgeleitet (bis es bei der Wurzel ankommt). Jeder Empfänger dieser Nachricht aktualisiert seine Routingtabelle.

RoutingUpdateDisconnect Wenn der Baum verkleinert wurde, wird diese Information an den Elter hochgereicht (bis es bei der Wurzel angekommen ist). Alle Empfänger dieser Nachricht aktualisieren ihre Routingtabellen.

KoordinatorErnennen Die Wurzel ernennt zyklisch und zufällig einen Knoten aus seinem Teilbaum als Koordinatorknoten. Das Attribut Daten_ enthält die BT_Adresse des Wurzelknotens. Der Empfänger ändert seine Rolle zum Koordinatorknoten und speichert die BT_Adresse der Wurzel.

KoordinatorAbwaehlen Die Wurzel wählt ihren Koordinator zyklisch ab. Dieser ändert damit seine Rolle zum Baumknoten. Diese Nachricht wird auch bei der Baumverschmelzung benutzt.

GibWurzelBTAdresseAnfrage Diese Nachricht wird zyklisch von einem Koordinatorknoten gesendet, um andere Koordinatorknoten zu finden. Sie wird nach einer Inquiry an alle gefundenen Geräte versendet, die nicht direktes Kind oder Elter des Koordinators sind. Wenn diese Nachricht empfangen wird, muss eine GibWurzelBTAdresseAntwort-Nachricht zurückgeschickt werden.

GibWurzelBTAdresseAntwort Diese Nachricht ist eine Antwort auf eine Koordinatorknotenanfrage. Wenn der angefragte Knoten kein Koordinatorknoten ist, dann steht im Attribut Daten_ eine Null, ansonsten wird dort die BT_Adresse des Wurzelknotens gesendet. Falls die Bluetoothadresse einer Wurzel als Antwort ankommt, muss eine NeuerBaumGefunden-Nachricht zu der eigenen Wurzel geschickt werden.

NeuerBaumGefunden Diese Nachricht wird vom Koordinatorknoten zu seiner Wurzel gesendet. Damit weißdie Wurzel, dass es noch einen Baum in der Umgebung gibt. Im Attribut Daten_ ist die BT_Adresse der fremden Wurzel enthalten. Daraufhin startet die Wurzel eine WurzelInquiry und verschickt dann eine Nachricht BaumVerschmelzung zu dem fremden Wurzelknoten.

BaumVerschmelzung Diese Nachricht wird von einer Wurzel zu der fremden Wurzel geschickt, um diese als Teilbaum anzuhängen. Diese wählt bei Erhalt dieser Nachricht ihren Koordinatorknoten ab, ändert die Rolle zu einem Baumknoten um, trägt den Sender der Nachricht als Elter bei sich ein, wird zur Bridge und schickt eine BaumVerschmelzungAntwort-Nachricht zurück.

BaumVerschmelzungAntwort Diese Nachricht enthält die komplette RoutingTabelle des ehemaligen Wurzelknotens. Die eigene Routingtabelle muss aktualisiert (bzw. erweitert) werden.

VerwaltungsManager Die Klasse VerwaltungsManager implementiert den TSF- und BTR-Algorithmus. Dazu reagiert der Verwaltungsmanager auf alle eingehenden VerwaltungsPakete, wie im Abschnitt VerwaltungsPaket beschrieben. Er hält eine Instanz der RoutingTabelle. Zusätzlich verwaltet er Referenzen auf seinen Elterknoten und seine direkten Kinder.

RoutingTabelle Die Klasse RoutingTabelle ist eine Datenstuktur die alle Bluetooth-Adres-sen der Kinder und Kindeskinder eines Teilbaumes verwaltet. Unter anderem ermöglicht die Routingtabelle eine effiziente Suche nach Bluetooth-Adressen eines direkten Kindes, um Nachrichten in einem Teilbaum in den richtigen Pfad routen zu können.

SkyNet SkyNet ist die direkte Schnittstelle zwischen Netzwerk- und Applikationsschicht. SkyNet startet das Netzwerk und beendet es bei Programmende. Die Klasse stellt Methoden für den Austausch von Nachrichten zur Verfügung. Außerdem können sich die Klassen bei SkyNet für den Nachrichtentyp registrieren, den sie empfangen möchten. SkyNet routet diese dann bei Empfang weiter.

Netzwerk Diese Klasse ist für die Verwaltung der Geräte zuständig. Die Klasse implementiert das Interface Runnable und lauscht in der `run()`-Methode ständig auf eingehende Verbindungen. Auch der Sender zum Senden aller Nachrichtentypen befindet sich in der Klasse Netzwerk. Außerdem wird in ihr eine Geräteliste namens andereGeraete gehalten, in der sich die Geräte befinden, zu denen eine vorläufige Verbindung nach der Inquiry-Phase besteht, und eine Geräteliste ElterundKinder, in der die Geräte stehen, zu denen eine permante Verbindung besteht. Die Klasse bietet Methoden an, um eine Verbindung zum Fallbackserver aufzubauen und Nachrichten mit diesem auszutauschen, Methoden zum Senden und Empfangen von BBHNachrichten und Verwaltungsnachrichten und zum Verwalten der enthaltenen Gerätelisten.

Sender Implementiert das Interface Runnable. Wenn ein Paket versandt werden soll, kommt es hier als Bytearray an, zusammen mit der Zieladresse. Der Sender wird vom Netzwerk geweckt. Die Nachricht wird im Sender in MTUs aufgeteilt und per L2Cap-Verbindung an den übergebenen Adressat versendet.

Empfaenger Implementiert das Interface Runnable. Es gibt für jedes Gerät einen Empfänger, der auf eingehende Nachrichten lauscht. Wenn eine Nachricht empfangen wird, wird diese an das Netzwerk weitergeleitet. Wenn die L2CAP-Verbindung zum sendenden Gerät abbricht, wird die Verbindung geschlossen, der Thread gestoppt und eine Nachricht über den Verbindungsabbruch an das Netzwerk geschickt.

Geraet Repräsentiert ein entferntes Endegerät und hält als Attribut den Empfänger für dieses Gerät und die Attribute für die Verbindung.

GeraeteListe Die Klasse erbt von Vector und dient dazu, Geräte zu verwalten.

Listener Die Klasse Listener implementiert das Interface DiscoveryListener. Die implementieren Methoden werden in der Inquiry-Phase vom DisvoceryAgent, der vor dem Start der Inquiry-Phase im Netzwerk erzeugt wurde, angesprochen, wenn ein Gerät entdeckt wurde, wenn der Service entdeckt wurde, wenn die Suche nach den Services abgeschlossen wurde und wenn die Inquiry beendet ist. Wenn die Inquiry-Phase beendet ist, werden die gefundenen Geräte an das Netzwerk übergeben.

Konvertierung Eine Klasse, die verschiedene Kovertierungsmethoden zur Verfügung stellt. Das wird zum Beispiel benutzt, um Nachrichten in ByteArrays zu verwandeln.

GUIKontrolle Die GUIKontrolle implementiert das Interface EreignisHandler, um den Timer nutzen zu können. Dazu wird `timerEvent(nachricht:int)` implementiert. Die Klasse GUI Kontrolle erbt von dem Interface CommandListener und implementiert damit implizit die Methode `commandAction( c:Command, d:Displayable)`. Zusätzlich muss es alle NachrichtManager explizit kennen, um Informationen von diesen zu erhalten und zu senden.

FileBrowser Diese Klasse wurde seperat implimentiert. Sie erbt von Liste, stellt einen FileBrowser dar und wird von GUIKontrolle benutzt.

9.5 Sequenzdiagramme

9.5.1 Suchanfrage erstellen

Von der GUI aus startet der Benutzer eine Suchanfrage indem er die Keywords eingibt. Vom SuchManager wird Suchanfrageobjekt und eine leere SuchMatchListe erzeugt. Anschließend wird der Timer gestartet und die SuchAnfrage an SkyNet geschickt. Durch Aufruf der Methode `empfangeNachricht()` werden durch den SuchManager SuchantwortPaketen anhand der SuchID empfangen und in die entsprechenden SuchMatchListe in Form von MatchListenEinträge (MatchListenEintragobjekten werden dabei erzeugt) hinzugefügt bis den Timer erreicht wird. Beim unbekannten SuchID werden die Antwortpaketen verworfen. Mit `gibSuchMatchListen()` werden von der GUI die Listen aktualisiert.

9.5.2 Download starten

Von der GUI aus wählt ein Benutzer einen DownloadEintrag in der DownloadListe, den er runterladen möchte und bestätigt mit "start" den Download. Bei einer positiven Überprüfung des Speicherplatzes wird vom DownloadManager eine DownloadAnfrage (broadcast) durch SkyNet geschickt, falls der Eintrag keine DownloadQuellen enthält. Falls doch wird eine DownloadAnforderung an den nach Warteposition bestplatzierten Quellen durch SkyNet geschickt. Der Timer wird gestartet. Im ersten Fall wird für jede eingegangene DownloadAntwort jeweils ein neues DownloadQuelle-Objekt erzeugt und in die DownloadQuellenListe des DownloadEintrages hinzugefügt.

Beim zweiten Fall bekommt man eine DownloadAntworten, in der steht, wo man in der Queue eingetragen wurde. Alle DownloadAntworten werden allerdings verworfen in dem Fall wo wir Antwortpakete mit Warteposition -1 und -2 erhalten oder der Timer abgelaufen ist.

Beim Erhalt eines UploadStart-Pakets wird von der Gegenstelle signalisiert, dass wir in der Queue an erster Stelle gelangen und bereit sind, die Daten zu empfangen. Dann werden Download-Abbruch-Nachrichten generiert und an die restlichen Partner gesendet. Weitere UploadStart-Nachrichten für diese Datei werden verworfen.

9.5.3 Auf Suchanfrage Reagieren

Das Reagieren auf Suchanfragen wurde in der Klasse EigeneDateienManager realisiert. Es be-schreibt die Reaktion des Programmes, wenn ein SuchAnfrage-Paket im System eintrifft. Wenn eine Suchanfrage im System eintrifft, wird die Methode `empfangeNachricht()` im EigeneDateienManager aufgrufen.

In dieser Methode wird überprüft, um welchen Nachrichtentyp es sich handelt. Wenn die ankommende Nachricht eine SuchAnfrage ist, werden die Keywords aus der Nachricht mit den Keywords in der SuchIndexListe verglichen. Dabei wird eine Referenz jeder Datei auf die die Keywords zutreffen in eine Menge eingefügt. Hier wird auch sichergestellt, das jede Referenz nur einmal eingefügt wird, auch falls mehrere Keywords auf die Datei zutreffen. Aus diesen Dateireferenzen werden jetzt die SuchAnwortEinträge generiert und in ein SuchAntwort-Paket eingefügt. Dieses SuchAntwort-Paket wird dann an den Suchenden verschickt, indem das Paket an SkyNet mittels der Methode `senden()` weitergereicht wird.

9.5.4 StartApp

Zunächst wird eine Instanz der Klasse Programm erzeugt. Diese instanziiert alle Managern (`EigeneDateienManager, SuchManager, DownloadManager, UploadManager), GuiKontrolle, Timer, SkyNet` und `Peers`. Dabei werden alle nötigen Listen, wie `Freigabeliste, SuchIndexliste, DownloadListe` u.s.w. angelegt. Anschliessend wird die Freigabeliste überprüft, eventuell aus Datei geladen und Netzwerk (`starteNetzwerk()` bei SkyNet) gestartet.

9.5.5 Auf DownloadAnfragen reagieren

Dieses Diagramm beschreibt das Verhalten des UploadManagers nach dem Empfang einer DownloadAnfrage mittels `empfangeNachricht()`, welche von SkyNet aufgerufen wird. Zuerst wird `existiertDatei()` des EigeneDateienManager aufgerufen um zu überprüfen, ob die angefragte Datei auf diesem Gerät freigegeben wurde. Falls dies zutrifft, wird eine DownloadAntwort generiert und diese via SkyNet an den Anfrager gesendet.

9.5.6 Auf DownloadAnforderung reagieren

Dieses Diagramm beschreibt das Verhalten des UploadManagers nach dem Empfang einer DownloadAnforderung mittels `empfangeNachricht()`, welche von SkyNet aufgerufen wird. Bei einer DownloadAnforderung wird ein UploadEintrag erstellt und dieser in die UploadListe eingetragen, wenn nicht schon zu viele Uploads eingetragen sind und die Datei freigegeben wurde. Falls dies nicht der Fall ist, wird eine DownloadAntwort mit einem Errorcode versendet. Wenn alles ok ist und es nur diesen einen UploadEintrag gibt, wird eine UploadStart Nachricht versendet, die anzeigt, dass der Upload stattfinden wird und danach wird die Datei in Uploadnachrichten geschickt.lassendiagramm

10 Diagrammbeschreibungen, 2. Semester

10.1 Aktivitätsdiagramme

10.1.1 Baumknoten

Dieses Diagramm beschreibt die Aufgaben eines Geraetes, welches die Rolle "Baumknoten" hat.

Wenn der Knoten zum Koordinator ernannt wird (ein VerwaltungsPaket mit Typ `KoordinatorErnennen` trifft ein), dann wird ein Rollenwechsel zum Koordinator durchgefuehrt.

Wenn der Knoten eine Verbindungsanfrage bekommt (ein VerwaltungsPaket mit Typ `AnhaengeAnfrage` trifft ein), dann wird die Rolle des Anfragers ueberprueft. Falls die Rolle des Anfragers "FreierKnoten" ist, dann wird die Liste der eigenen Kinderknoten erweitert und der Status wird auf Bridge gesetzt, falls er zuvor noch nicht Bridge war. Danach wird der Anfrager als Kind in die RoutingTabelle (mit Distanz 1) eingetragen. Schließlich wird die neue HashTabelle an den Elter weitergeleitet (mittels eines "RoutingUpdateConnect" VerwaltungsPakets).

10.1.2 Fehlerbehandlung

Dieses Diagramm beschreibt die Behandlung von Verbindungsabbruechen. Eine auf diese Art abgebrochene Verbindung ist immer eine Verbindung zu einem direkt verbundenen Geraet (direktes Kind oder Elter).

Zuerst wird ueberprueft, ob die Verbindung zum Elter abgebrochen ist. Falls ja, wird in jedem Fall ein Rollenwechsel vollzogen und die RoutingTabelle wird aktualisiert. Wenn die Verbindung zum Elter abgebrochen ist und keine weiteren Kinder existieren, wird das Geraet zu einem Freien Knoten, ansonsten zu einem Wurzelknoten (mit Status = Master). Falls nein, muss ebenfalls die RoutingTabelle aktualisiert werden und zusaetzlich muss der Elter (bzw alle Knoten auf dem Pfad zur Wurzel) ueber den Abbruch informiert werden. Dies geschieht mittels des Verwaltungspakets RoutingUpdateDisconnect, welches die RoutingTabelle des Geraets an den Elter weitergibt. Sind keine Kinder mehr vorhanden wird der Status auf Slave gesetzt.

10.1.3 Freierknoten

Dieses Diagramm beschreibt die Aufgaben eines Geraetes, welches die Rolle `Freierknoten` hat. Nach der Initialisierung des Geraetes wird ein Timer randomisiert gestartet. Wenn dieser Timer sein TimeOut liefert, dann wird eine Inquiry gestartet. Diese Inquiry liefert eine Geraeteliste mit temporaer verbundenen Geraeten zurueck. Solange das Geraet noch in keinen Baum integriert wurde, werden VerwaltungsPakete des Typs `AnhaengeAnfrage` an andere Geraete versendet. Diese Geraete antworten mit einem VerwaltungsPaketes des Typs `AnhaengeAntwort`. Bei einem erfolgreichen Anhaengen wird die Rolle auf Baumknoten und der Elternzeiger auf das antwortende Geraet gesetzt. Die temporaeren Verbindungen werden nach diesem Ablauf unterbrochen. Sollten zwei Geraete mit Rolle Freierknoten Verwaltungspakete des Typs `AnhaengeAnfragen` austauschen, so wird der Antwortende zu einem Wurzelknoten und der Anfragende zu einem Baumknoten. Der Wurzelknoten muss das andere Geraet in diesem Fall in seine Routingtabelle aufnehmen.

10.1.4 Koordinatorknoten

Der Koordinatorknoten wird in periodischen Zeitabstände von seiner Wurzel (Verwaltungspaket vom Typ `Koordinatorabwaehlen`) abgewählt, womit seine Rolle dann zu einem Baumknoten wechselt.

In der gleichen Phase kann der Koordinatorknoten eine Wurzelanfrage eines anderen Koordinatorknotens erhalten (Verwaltungspaket vom Typ `GibWurzelBTAdresseAnfrage`), auf die er dann reagiert indem er die Adresse seiner Wurzel zurückschickt (Verwaltungspaket vom Typ `GibWurzelBTAdresseAntwort`).

Wenn ein Koordinatorkonten eine Inquiry startet, findet er eine gewisse Anzahl an Geräten, die in einer Geräteliste abgespeichert werden. Falls die Geräteliste leer ist, wird ein Timeout gestartet und anschließend eine neue Inquiry durchgeführt. Die Geräteliste wird dann sequentiell durchlaufen, um zu jedem Gerät (außer zum Elter und den direkten Kindern) eine temporäre Verbindung aufzubauen, in der die Rolle des jeweiligen Gerätes abgefragt wird, bis der Koordinatorknoten einen anderen Koordinatorknoten findet. Falls ein Gerät nicht mehr antwortet oder die Rolle des Gerätes gleich einem WurzelKnoten, BaumKnoten oder FreierKnoten ist wird das nächste Gerät aus der Geräteliste ausgewählt. Der gefundene Koordinatorknoten K bekommt ein Verwaltungspaket vom Typ `GibWurzelBTAdresseAnfrage` und wird nach der BT-Adresse seiner Wurzel gefragt. Die Antwort des Koordinatorknotens K mit der BT-Adresse der Wurzel (Verwaltungspaket vom Typ `GibWurzelBTAdresseAntwort`), wird an die eigene Wurzel weitergeleitet.

Wichtig ist zu erwähnen, dass der KoordinatorKnoten auch die gleiche Funktionalität eines Baumknotens besitzt, das heißt, ein FreierKnoten der ein Inquiry startet, kann sich auch an einen KoordinatorKnoten hängen, wenn er diesen in seiner Geräteliste findet.

10.1.5 Routingaenderung

Bei einer Routingaenderung wird die eigene Routingtabelle aktualisiert und an den Elter weitergeleitet, solange bis sie an der Wurzel angekommen ist.

10.1.6 Wurzelknoten

Ein Wurzelknoten hat verschiedene Aufgaben: zum einen muss er seinen Koordinatorknoten in bestimmten Zyklen abwählen und zum anderen einen neuen Koordinatorknoten ernennen.

Wenn der Wurzelknoten eine BT-Adresse eines anderen Wurzelknotens bekommt, startet dieser ein Inquiry und sucht nach der erhaltenen BT-Adresse um eine Baumverschmelzung einzuleiten. Wenn das Gerät mit der zugehörigen BT-Adresse gefunden wird, muss der Wurzelknoten ein Verwaltungspaket vom Typ `Baumverschmelzung` an dieses senden. Die Rolle des gefundenen Geräts wird dann von Wurzelknoten auf Baumknoten gewechselt und anschließend an den Wurzelknoten, der die Inquiry gestartet hat angehängt. Falls die Wurzel bereits sechs direkte Kinder hat, findet die Baumverschmelzung nicht statt und beide Bäume bleiben nebeneinander bestehen. Die Routingtabelle des Wurzelknotens muss anschließend aktualisiert werden. Der Wurzelknoten, der die Inquiry gestartet hat, muss dann die Liste seiner Kinder aktualisieren. Der Koordinatorknoten wird beibehalten, da dieser zyklisch bestimmt wird.

Der Wurzelknoten der an den neuen Baum angehängt wurde, muss die Wurzel als Elter eintragen und seinen eigenen Koordinatorknoten abwählen.

10.2 Sequenzdiagramme

10.2.1 Bäume durch Wurzel vereinigen

Passiv Die Rolle des Geräts ist Wurzelknoten. Wenn übers Netzwerk ein Verwaltungspaket vom Typ `BaumverschmelzungsAntwort` eintrifft, wird dieses an den Verwaltungsmanager weitergeleitet (`empfangenachricht(VerwaltungsPaket)`). Dort wird das Paket an die Methode `reagiereAufBaumverschmelzung()` bearbeitet, indem die Rolle der Wurzel auf Baumknoten gesetzt wird, an den Koordinatorknoten ein Verwaltungspaket Koordinatorabwählen geschickt wird und als neuer Elter des Geräts der Wurzelknoten aus dem Verwaltungspaket gesetzt wird.

Anschließend muss noch die neue Wurzel über den dem Gerät angehängten Teilbaum mit Hilfe eines Verwaltungspakets RoutingUpdate informiert werden. Die Verwaltungspakete werden zum Senden an das Netzwerk in der Methode `sendeVerwaltungsPaket(VerwaltungsPaket())` geschickt.

Aktiv Die Rolle des Geräts ist Wurzelknoten. Wenn übers Netzwerk ein Verwaltungspaket vom Typ `BaumverschmelzungsAnfrage` eintrifft, wird dieses an den Verwaltungsmanager weitergeleitet (`empfangenachricht(VerwaltungsPaket)`). Es muss eine Inquiry zum Finden der anderen Wurzel gestartet werden (`wurzelInquiryStart(String)`). Wird die andere Wurzel gefunden, wird dies über die Methode `wuzelInquiryFertig()` vom Netzwerk an den Verwaltungsmanager gemeldet. Anschließend wird ein Verwaltungspaket BaumverschmelzungsAntwort an die andere Wurzel über die Methode `sendeVerwaltungsPaket(VerwaltungsPaket)` im Netzwerk. Danach erhält der Verwaltungsmanager ein Verwaltungspaket RoutingUpdate, in dem die Informationen des Teilbaumes der anderen Wurzel mitgeteilt werden. Diese werden in die Routingtabelle der Wurzel aufgenommen.

10.2.2 Bäume vereinigen

Koordinatorknoten starten in gewissen Zeitabständen Inquirys, um andere Koordinatorknoten zu finden und somit die Teilbäume zu vereinigen (`koordinatorInquiryStart()`). Wird ein anderer Koordinator gefunden, wird dieser über die Methode `vereinigungBaeume(Geraet)` an den Verwaltungsmanager geschickt. Mit einem Verwaltungspaket wird die Wurzel des Koordinators erfragt. Wenn diese Nachricht eintrifft, wird ein Verwaltungspaket an die Wurzel geschickt, um die Wurzel über die Baumverschmelzung zu informieren.

10.2.3 Freier Knoten integrieren

Wenn ein freier Knoten eine Inquiry zum finden eines Baumes gestartet hat, wird nach der Inquiry in der Klasse Listener die Methode `inquiryCompleted(int discType)` ausgelöst. Diese startet im Netzwerk die Methode `inquiryAbgeschlossen()`. In dieser Methode wird die mit der in denInquiry gefundenen Geräten gefülltenGeraeteListe an den Verwaltungsmanager übergeben. Dieser sucht sich randomisiert ein Gerät aus der Liste und sendet diesem ein Verwaltungspaket über die Methode `sendeVerwaltungspaket(VerwaltungsPaket Paket)` im Netzwerk. Ist nach einem Timeout keine Antwort gekommen, wird ein anderes Gerät aus der Gerätliste ausgewählt und diesem eine Verwaltungsnachricht geschickt. Ist eine positive Antwort eingegangen, wird diese an den Verwaltungsmanager weitergeleitet. Dort wird der Elter gesetzt und die Rolle auf Baumknoten geändert.

10.2.4 Piconetz

In diesem Diagramm wird der Programmablauf von BBH dargestellt, wenn sich einzelne Geräte zu einem Piconetz zusammenschliessen. Nachdem die Bluetooth Inquiry-Phase abgeschlossen ist, wird in der Klasse Netzwerk die Methode `gibMaster()` aufgerufen. Diese Methode liefert ein Gerät zurück, das Master ist und noch einen weiteren Slave aufnehmen kann. Falls kein solches Gerät existiert, wird `NULL` zurückgeliefert. Die Methode geht folgendermaßen vor: von allen Geräten, die während der Inquiry-Phase gefunden worden sind, wird ein Gerät zufällig ausgewählt. An dieses Gerät wird dann eine Anfrage gesendet, ob es Master ist und noch einen freien Platz für ein Slave-Gerät hat.

Falls das angefragte Gerät Master ist und noch mindestens einen freien Platz hat, kann die Methode beendet werden, und es kann ein Verweis auf dieses Gerät zurückgegeben werden. Falls dies noch nicht der Fall ist, werden die anderen während der Inquiry-Phase gefundenen Geräte gefragt. Dies wird solange wiederholt bis ein freier Platz gefunden wurde oder alle Geräte gefragt wurden.

Falls `gibMaster()` `NULL` zurückgeliefert hat, wird der eigene Status auf Master gesetzt. Falls `gibMaster()` ein Gerät zurückgeliefert hat, kann die Methode `setzeGeraete()` aus der Klasse AndereGeraete aufgerufen werden, um das eben gefundene Gerät dort einzutragen. Diese Klasse verwaltet alle Geräte mit denen die Filesharing-Schicht von BBH kommunizieren kann.

10.2.5 Routing von BBH-Paketen

Dieses Diagramm stellt den Ablauf des Routings von BBH Nachrichten dar. Wenn die Applikationsschicht eine Nachricht verschicken will, ruft sie die Methode `sende()` aus der Klasse SkyNet auf und

übergibt die zu sendende Nachricht als Parameter. SkyNet reicht die Nachricht zu der Klasse Netzwerk weiter, die das eigentliche Routing in die Wege leitet. Dazu wird die Methode `gibEchtenEmpfaenger()` aus der Klasse VerwaltungsManager aufgerufen. Als Parameter dieses Methodenaufrufes wird die zu versendende BBHNachricht übergeben. Diese Methode sucht nach der Zieladresse der BBHNachricht in der eigenen Routingtabelle. Falls die Zieladresse gefunden wurde, wird die Adresse des nächsten Hops der Nachricht zurückgegeben. Falls die Zieladresse nicht in der Routingtabelle zu finden ist und wir kein Wurzelknoten sind, wird die Adresse des eigenen Elters zurückgeliefert.

Für den Fall, dass wir selber Wurzelknoten sind, und die Zieladresse nicht in der Routingtabelle finden können, verwerfen wir das Paket und liefern `NULL` zurück.

Nachdem wir von der Methode `gibEchtenEmpfänger()` eine Bluetooth-Adresse geliefert bekommen haben, können wir die Nachricht jetzt versenden. Dazu wird die Methode `gibGeraet()` aus der Klasse andereGeraete aufgerufen, um einen Verweis auf das Gerät mit der eben zurückgelieferten Bluetooth Adresse zu bekommen. Danach kann die Nachricht in einen Bytestream konvertiert werden, indem die Methode `toBBH()` aufgerufen wird. Dieser Stream kann jetzt dem Sender-Thread zum Verschicken übergeben werden.

Der Ablauf des Routings bei einer einkommenden Nachricht ist ähnlich: Zuerst wird die Zieladresse der einkommenden Nachricht überprüft, ob sie mit der eigenen Adresse übereinstimmt. Falls das der Fall ist, wird die Nachricht von SkyNet an die entsprechenden BBHManager weitergeleitet. Falls die Nachricht nicht an uns adressiert wurde, wird wieder `sendeNachricht()` aus Netzwerk aufgerufen.

Wenn eine Nachricht per Broadcast verschickt werden soll, wird die Nachricht an alle Geräte in der Routingtabelle und den Elter verschickt. Dabei wird allerdings das Gerät ausgenommen, das uns die Nachricht geschickt hat.

10.2.6 Verbindungstrennung abfangen

Das Diagramm beschreibt den zeitliche Ablauf des Programms bei einem Verbindungsabbruch. Ein Verbindungsabbruch erfolgt dann, wenn Geräte, die unmittelbar mit anderen Geräten in Verbindung stehen, plötzlich ausfallen. Dieser Ausfall kann durch den Benutzer erfolgen (z.B. nach Beendigung des Programms) oder auch unabhängig sein (z.B. Energieausfall).

Die Trennung der Verbindung bleibt allerdings dem Benutzer verborgen. Sie wird auf der Netzwerkebene gehandhabt und durch entsprechende Verwaltungsmaßnahmen abgefangen. Falls jedoch die Verbindung während eines Downloads abbricht, wird dieser gestoppt und durch Resume von anderen potentiellen Quellen weiter heruntergeladen.

Nachdem ein Gerät vom Netzwerk getrennt ist, wird das Netzwerk vom Listener durch Aufruf der Methode `verbindungAbgebrochen(Geraet)` benachrichtigt. Dabei wird als Parameter das Gerät, das das Netzwerk verlassen hat, übergeben. Daraufhin wird die Methode `empfangeReport(String)` im Verwaltungsmanager aufgerufen. Diese Methode bekommt vom Netzwerk die BTAdresse des Gerätes und ruft dann die Methode `reagiereAufVerbindungsAbbruch(String)` im Verwaltungsmanager auf. Diese Methode leitet dann alle erforderlichen Maßnahmen ein, um die Verbindungstrennung abzufangen. Alle Knoten, die unmittelbar mit dem ausgefallenen Knoten eine Verbindung zum Zeitpunkt des Abbruchs haben, reagieren dann auf diese Methode.

In der Methode wird zuerst geprüft, ob der aktuelle Knoten ein Wurzelknoten ist, dann ob er noch Kinder hat. Falls er keine Kinder mehr hat, wird er auf FreierKnoten gesetzt. Bei noch vorhandenen Kinder überprüft er, ob der verschwundene Knoten sein KoordinatorKnoten ist. Falls ja, setzt er den alte KoordinatorKnoten auf null und ernennt einen neuen. Ansonsten muss er nichts tun. In dem Fall, in dem der aktuelle Knoten kein Wurzelknoten ist, uberprüft er, ob der Ausfall vom Elter kommt. Wenn ja, werden die TimeOuts, falls der aktuelle Knoten ein KoordinatorKnoten ist, gestoppt. Wenn er zuvor

keine Kinder hatte, wird er zum FreienKnoten ernannt, ansonsten wird er als WurzelKnoten. Dabei werden die Routingtabellen aktualisiert (das entsprechende Gerät wird gelöscht). Falls der Ausfall jedoch vom Kind stammt, wird das Kind aus der Routingstabelle entfernt, und ein RoutingUpdatePaket wird an den Elter bis zum Wurzel hochgereicht.

11 Anhang A - Tests und Fehlerbehebung

11.1 1. Semester

Folgendes Testprotokoll wurde gemäßden Anwedungsfällen erstellt:

11.1.1 Dateien freigeben

Datum:	27.01.05
Verantwortliche Person:	Zok, Prost
Was wird erwartet:	Man kann in seinem Gerätefilesystem browsen. Es wird eine Hierarchie von Ordnern und Dateien angezeigt. Wenn man eine Datei oder Ordner makiert hat, kann man ihn auch freigeben. Dieser wird dann in der Freigabeliste angezeigt.
Was ist passiert:	Funktioniert; Freigabe von einer 6MB Datei dauert 100sec.; ca. 10 Tage vorher wurde das hashen schon auf einem richtigem Handy gestestet, was dort wesentlich schneller ging.

11.1.2 Dateien sperren

Datum:	27.01.05
Verantwortliche Person:	Zok, Prost
Was wird erwartet:	Die Datei, die zuvor freigegeben wurde, wird gesperrt. Somit wird sie aus der der Freigabeliste entfernt.
Was ist passiert:	Funktioniert.

11.1.3 Freigegebene Dateien auflisten

Datum:	27.01.05
Verantwortliche Person:	Zok, Prost
Was wird erwartet:	Dateien, die man freigegeben hat, werden angezeigt.
Was ist passiert:	Funktioniert.

11.1.4 Eine Datei austauschen

Datum:	27.01.05
Verantwortliche Person:	Börgermann, Diel, Göbel, Prost, Zok
Was wird erwartet:	Eine Datei, die man sucht und downloaden will, soll von einem Gerät zum nächsten übertragen werden.
Was ist passiert:	Funktioniert bei kleinen Dateien.
Falls Fehler, welche?:	Bei großen Dateien kommen `OutOfMemoryExceptions`.
Maßnamen:	Die maximale Paketgröße wurde von 480kb auf 50kb verringert. Der Arbeitsspeicher des Geräts wurde zuvor überlastet.
Fehler behoben am:	02.02.05
Fehler behoben durch:	Die Paketgröße wurde verringert (auf 50kb)

11.1.5 Mehrere Dateien austauschen

Datum:	27.01.05
Verantwortliche Person:	Börgermann, Diel, Göbel, Prost, Zok
Was wird erwartet:	Mehrere Dateien sollen von gleichem Gerät runtergeladen werden. Dann sollen mehrere Dateien von unterschiedlichen Geräten runtergeladen werden.
Was ist passiert:	Ab zweiter Datei wurde nicht übertragen. Stattdessen kommt eine DownloadAntwort an.
Falls Fehler, welche?:	Dateien nicht übertragen
Maßnamen:	Queueverhalten des UploadManagers prüfen.
Fehler behoben am:	31.01.05
Fehler behoben durch:	Keine Fehler beim Überprüfen des UploadManagers gefunden. Danach überpüft, ob SkyNet nach dem Versand einer Upload-Nachricht dem UploadManager Bescheid gibt. SkyNet tat dies nicht, sondern benachrichtigte den EigeneDateienManager. Parameter verändert - nun funktioniert es!

11.1.6 Dateien suchen

Datum:	27.01.05
Verantwortliche Person:	Tigyo, Kißner, Zok, Prost
Was wird erwartet:	Wir bekommen SuchAntworten, von den Peers, die Dateien mit den eingegeben Keywords freigegeben haben. Diese werden in die richtige SuchMatchListe eingetragen und bei „MeineSuchen" angezeigt.
Was ist passiert:	Funktioniert, aber bei nicht vorhandenen Keywords gab es eine `NullPointerException`. Dies wurde direkt behoben.
Falls Fehler, welche?:	Bei nicht vorhandenen Keywords gab es eine `NullPointerException`.
Maßnamen:	direkt behoben
Fehler behoben am:	27.01.05
Fehler behoben durch:	-

11.1.7 Dateien als Download auswählen

Datum:	27.01.05
Verantwortliche Person:	Tigyo, Kißner
Was wird erwartet:	Es wird ein MatchListenEintrag, den man downloaden aber nicht direkt starten möchte, dem DownloadManager übergeben. Dort wird der MatchListenEintrag in die DownloadListe hinzugefügt.
Was ist passiert:	Funktioniert.

11.1.8 Download starten

Datum:	27.01.05
Verantwortliche Person:	igyo, Kißner, Prost, Zok
Was wird erwartet:	Man wählt aus der DownloadListe oder der SuchMatchListe eine Datei aus, die man gerne runterladen möchte. Die Datei wird in der Downloadliste angezeigt und direkt heruntergeladen. Wenn sie fertig geladen ist, wird sie aus der Downloadliste ausgetragen und in die FreigabeListe eingetragen.
Was ist passiert:	Funktioniert.

11.1.9 Download resumen

Datum:	02.02.05
Verantwortliche Person:	Tigyo, Kißner, Prost, Zok
Was wird erwartet:	Wenn eine Datei pausiert wurde, kann sie ab dem gestoppten Zustand wieder gedownloaded werden. Die Datei sollte dabei nicht beschädigt werden.
Was ist passiert:	Der Download wird wieder aufgenommen.
Falls Fehler, welche?:	Die TempDatei wird komplett neu geschrieben und nicht erst ab dem aktuellem Startbyte.
Maßnamen:	ktualisiereStartByte muss vom EigenenDateienManager aufgerufen werden sobald die Rohdaten eines UploadPacketes in die Temp-Datei geschrieben worden sind.
Fehler behoben am:	08.02.05
Fehler behoben durch:	aktualisiereStartByte wird jetzt im EigeneDateienManager aufgerufen

11.1.10 Download abbrechen

Datum:	27.01.05
Verantwortliche Person:	Kißner, Tigyo, Prost, Zok
Was wird erwartet:	Der Download wird mittels einer DownloadAbbruch Nachricht abgebrochen und der DownloadEintrag aus der DownloadListe gelöscht und die evtl. bereits geschriebene Temp-Datei gelöscht.
Was ist passiert:	Beim Klicken auf den MenuEintrag "abbrechen" von der DownloadListe wird abgebrochen und der DownloadEintrag gelöscht. Funktioniert ohne die loescheTemp()-Methode
Falls Fehler, welche?:	Es wird eine `NullPointerException` ausgelöst.
Maßnamen:	Lokalisierung des Fehlers.
Fehler behoben am:	01.02.05
Fehler behoben durch:	Der DownloadManager hatte den EigeneDateienManager nicht richtig initialisiert. Wird jetzt ihm durch das Programm übergeben. Jetzt funktioniert alles.

11.1.11 Download anhalten/pause

Datum:	02.02.05
Verantwortliche Person:	Tigyo, Kißner, Kasmann
Was wird erwartet:	Der Download einer Datei soll angehalten werden. Dabei wird der Startbytezeiger an das Ende der schon heruntergeladenen Daten gesetzt und der Download angehalten. Diese temporäre Datei darf nicht gelöscht werden, solange sie nicht vollständig empfangen wurde.
Was ist passiert:	Funktioniert, Temp-Datei wird nicht weiter geschrieben, der „Uploader" wird benachrichtigt und sendet nicht weiter.
Falls Fehler, welche?:	Das StartByte wird nicht auf die aktuelle Position gesetzt.
Maßnamen:	aktualisiereStartByte muss vom EigenenDateienManager aufgerufen werden sobald die Rohdaten eines UploadPacketes in die Temp-Datei geschrieben worden sind.
Fehler behoben am:	08.02.05
Fehler behoben durch:	Zok

11.1.12 Upload einsehen

Datum:	31.01.05
Verantwortliche Person:	Zok, Prost
Was wird erwartet:	Es wird im Upload-Fenster angezeigt, welche Dateien vom eigenen Gerät upgeloaded werden.
Was ist passiert:	Funktioniert.

11.1.13 Upload abbrechen

Datum:	31.01.05
Verantwortliche Person:	Zok, Prost
Was wird erwartet:	Hier wird durch den Benutzer explizit ein Upload abgebrochen. Danach wird die Datei nicht mehr im Upload-Fenster zu sehen sein. Aber immer noch im Freigegebene Dateien-Fenster.
Was ist passiert:	Funktioniert.

11.1.14 Fallback aktivieren

Datum:	31.01.05
Verantwortliche Person:	Börgermann, Diel, Göbel
Was wird erwartet:	Man kann auf einem Fallback-Server zugreifen. Wenn man ihn aktiviert, kann man dort wie sonst nach Dateien suchen.
Was ist passiert:	Man kommt nur auf ein GUI Fenster, welches den Fallbackserver symbolisiert.

11.1.15 Persistenz prüfen

Datum:	31.01.05
Verantwortliche Person:	Krokowski
Was wird erwartet:	Alle Einstellungen und alle Listen werden kurz vor Programmende bzw., falls das Midlet pausiert wird, persistent auf dem mobilen Gerät gespeichert werden.
Was ist passiert:	Funktioniert.

11.1.16 Spezialfall: Es dürfen keine „leeren" Suchantworten geschickt werden

Datum:	31.01.05
Verantwortliche Person:	Prost
Was wird erwartet:	Falls das Suchfeld leer ist, darf keine Nachricht an Skynet geschickt werden.
Was ist passiert:	Funktioniert.

11.1.17 Spezialfall: Mehrere Peers haben gesuchte Datei. Kommunikation und Aktualisierung zwischen DLManager und ULManagern. (Downloadabbruch selbstständig senden)

Datum:	02.02.05
Verantwortliche Person:	Alle
Was wird erwartet:	siehe oben
Was ist passiert:	Die Datei wird nur bei einem Peer gefunden.
Falls Fehler, welche?:	GUI macht Quark
Maßnamen:	GUI geändert
Fehler behoben am:	10.02.05
Fehler behoben durch:	Alle

11.1.18 Spezialfall: DownloadAntwort oder SuchAntwort trifft nach Ablaufen des Timers ein

Datum:	31.01.05
Verantwortliche Person:	Kißner, Tigyo
Was wird erwartet:	Falls DownloadAntwort oder SuchAntwort nach Ablaufen des Timers eintrifft, werden diese verworfen.
Was ist passiert:	Funktioniert.

11.1.19 Spezialfall: Das Überschütten des Netzwerkes mit vielen, gleichzeitigen Suchanfragen (Stabilitätstest)

Datum:	31.01.05
Verantwortliche Person:	Alle
Was wird erwartet:	Die mobilen Geräte sollten dadurch nicht überlastet werden. Zur Not werden Pakete verworfen.
Was ist passiert:	Unbekannt
Falls Fehler, welche?:	Unbekannt
Maßnamen:	Kann nur in realer Umgebung vernünftig getestet werden.

11.1.20 Spezialfall: Downloadwunsch einer Datei, für die nicht mehr genügend Speicherplatz zur Verfügung steht

Datum:	31.01.05
Verantwortliche Person:	Prost
Was wird erwartet:	Es mußeine Fehlermeldung auf dem Bildschirmangezeigt werden, die den Benutzer hinweißt Platz zu schaffen.
Was ist passiert:	Kann nicht getestet werden, da wir kein reales Gerät haben
Falls Fehler, welche?:	Unbekannt
Maßnamen:	Sicherheitsabfrage eingebaut, Datei wird nicht mehr geschrieben, wenn kein Speicherplatz vorhanden ist.
Fehler behoben am:	08.02.05
Fehler behoben durch:	Zok, Tigyo, Kißner

11.1.21 Spezialfall: Start eines Downloads für den keine Quellen zur Verfügung stehen

Datum:	31.01.05
Verantwortliche Person:	Kasmann
Was wird erwartet:	Falls ein Download, für den es keine Quellen zur Verfügung gibt (oder wo alle Quellen „Nein" sagen) gestartet wurde, wird er sofort abgebrochen.
Was ist passiert:	Alles hängt sich auf.
Falls Fehler, welche?:	Unbekannt
Maßnamen:	Abfrage eingebaut, die einen nicht vorhandenen Addressaten abfängt.
Fehler behoben am:	08.02.05
Fehler behoben durch:	Diel

11.2 2. Semester

Hinweise zum Testprotokoll:

- Bei der Durchführung der Testfälle wurde das Gerät (erstes angeschaltetes Gerät) mit der Bezeichnung "+5550000", was für den Simulator der physikalisch Master darstellt, nicht ausgeschaltet, da sonst das JSR-82-Netzwerk nicht mehr verfügbar war.
- Die Testfälle wurden mit dem WTK auf einem Windows-Rechner durchgeführt.

11.2.1 Verbindung zweier Freierknoten.

Datum:	06.07.05
Verantwortliche Person:	Topologen (siehe oben)
Was wird erwartet:	Beide Knoten schließen sich an. Einer wird Wurzelknoten und ernennt der andere zum Koordinatorknoten. Die Routingtabellen werden aktualisiert.
Was ist passiert:	Funktioniert erwartungsgemäßproblemlos und schnell.

11.2.2 Anschluss einer Freierknoten am vorhandenen Baum

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Der Freierknoten wird dann zum Baumknoten. Alle Knoten im Baum aktualisieren ihre Routingtabellen.
Was ist passiert:	Funktioniert.

11.2.3 Verschmelzung zweier unabhängigen Bäume.

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Verschmelzung der Bäume, indem die fremde Wurzel sein Koordinatorknoten abwählt und sich an der neuen Wurzel als Kind anhängt und zum Baumknoten wird. Alle Routingtabellen werden aktualisiert.
Was ist passiert:	Die Bäume verschmelzen sich problemlos nach ca. 7 Sek.

11.2.4 Wiederanschluss der selben Knoten (BK, KK, WK) am vorhandenen Baum nach Verlassen (Programm) des Netzes

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Die Knoten schließen wieder an und werden dann zum Baumknoten. Der Wurzelknoten wird gewählt und ernennt ein neuer K-Knoten. Alle Knoten im Baum aktualisieren ihre Routingtabellen.
Was ist passiert:	Funktioniert erwartungsgemäß.

11.2.5 Nach Verbindungsabbruch (Wurzelknoten) freigewordene Knoten schließen sich wieder an

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Die Freierknoten schließen sich an und je nach Konstellation bildet sich ein neuer Baum. Alle Routingtabellen werden aktualisiert.
Was ist passiert:	4 Geräten gestartet. Der Wurzelknoten ausgeschaltet (Programm), nach 3 Sek. hat sich ein Baum formiert.

11.2.6 Nach Verbindungsabbruch (Koordinatorknoten) freigewordene Knoten schließen sich wieder an

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	ie Freierknoten schließen sich an und je nach Konstellation bildet sich ein neuer Baum. Der Wurzelknoten ernennt einen neuen K-Knoten. Alle Routingtabellen werden aktualisiert.
Was ist passiert:	4 Geräten gestartet. Der Koordinatorknoten mehrmals ausgeschaltet (Programm), nach 5 Sek. hat sich ein Baum formiert.

11.2.7 Verschmelzung nach Verbindungsabbruch zweier unabhängig gewordene Bäume

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Die beide Bäume verschmelzen sich (sieh. Anwendungsfall 3).
Was ist passiert:	Eine Kette von 5 Geräten gestartet. Das mittlere Gerät ausgeschaltet(Programm), nach ca. 20 Sek. findet die Verschmelzung statt. Die Routingtabellen werden aktualisiert.

11.2.8 Suche/Download über mehrere Hops

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	ie Dateien werden gefunden und problemlos herunter geladen. Die Routing von Verwaltungsnachrichten muss anhand von Lämpchen sichtbar sein.
Was ist passiert:	Das suchende Gerät befand sich 4 Hops von der Quelle entfernt. Die Suche ergab 1 Treffer und der Download (5,5MB) lief problemlos.

11.2.9 Resume über mehrere Hops

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Der Download muss stoppen und die Lämpchen der beteiligten Geräte nicht mehr leuchten. Dann muss der Download fortgesetzt werden und die Lämpchen leuchten.
Was ist passiert:	Das Gerät befand sich 4 Hops von der Quelle entfernt. Der Resume der Datei (5,5MB) lief problemlos.

11.2.10 Gleichzeitige Downloads über Brigde-Geräte

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Die Dateien werden problemlos herunter geladen und die Lämpchen leuchten entsprechend.
Was ist passiert:	Die Dateien (4MB / 3MB) werden vollständig übertragen, bei jedem Gerät läuft nur ein Thread (Sender oder Empfänger), nicht beides.

11.2.11 Gleichzeitige Downloads (=2) über Brigde-Geräte (Große Dateien). Geräte fungieren als Sender und Empfänger

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Die Dateien werden problemlos herunter geladen und die Lämpchen leuchten entsprechend.
Was ist passiert:	Die Dateien (11MB / 15MB) werden vollständig übertragen, bei jedem Gerät laufen beide Threads (Sender und Empfänger). Die beteiligten Geräte leuchten.

11.2.12 Gleichzeitige Downloads(>2) über die Wurzel (Große Dateien). Geräte können als Sender und Empfänger fungieren

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Über die Wurzel laufen mehr als 2 Downloads. Die Geräte können gleichzeitig senden und empfangen. Die Dateien werden problemlos herunter geladen.
Was ist passiert:	Die Dateien (6MB / 11MB / 15MB) werden vollständig übertragen. Die Geräte leuchten aber der Download dauert einige Minuten.

11.2.13 Download von einer entfernten Quelle, die das Scatternetz plötzlich verlässt.

Datum:	06.07.05
Verantwortliche Person:	Topologen
Was wird erwartet:	Der Download wird abgebrochen und durch Resume von einer anderen Quelle ergänzt. Die entsprechenden Geräte leuchten.
Was ist passiert:	Der Download der Datei (4 MB) wurde abgebrochen und durch Resume fertig herunter geladen. Funktioniert.

11.2.14 AND-Suche nach 1 Keyword / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.15 AND-Suche nach 3 Keywords / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.16 AND-Suche nach 1 Keyword (Substring Anfang vom ersten Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.17 AND-Suche nach 1 Keyword (Substring mitten im Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.18 AND-Suche nach 3 Keywords (Substring Anfang vom ersten Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.19 AND-Suche nach 3 Keywords (Substring mitten im Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.20 OR-Suche nach 1 Keyword / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.21 AND-Suche nach 3 Keywords / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.22 OR-Suche nach 1 Keyword (Substring Anfang vom ersten Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.23 OR-Suche nach 1 Keyword (Substring mitten im Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.24 OR-Suche nach 3 Keywords (Substring Anfang vom ersten Wort) / Groß/ Kleinschreibung / Gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.25 OR-Suche nach 3 Keywords (Substring mitten im Wort), Groß-/Kleinschreibung, gemischt

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Datei wird gefunden.
Was ist passiert:	Datei wurde gefunden.

11.2.26 Gutschrift von Credits beim Uplaod

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Der Peer, der hochlädt, bekommt beim Empfänger Credits gutgeschrieben.
Was ist passiert:	Credits werden gutgeschrieben (Durch Konsolenausgabe getestet).

11.2.27 Sortierung der Warteliste nach Credits

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Die Warteliste wird per Quicksort nach Credits sortiert.
Was ist passiert:	Geräte mit den meisten Credits stehen am Anfang der Liste (Durch Konsolenausgabe getestet).

11.2.28 Uploadpriorität vergeben

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	Dateien mit der höheren Priorität werden zuerst hochgeladen.
Was ist passiert:	Dateien mit niedrigerer Priorität zuerst in der Warteschlange, werden aber nach Dateien mit hoher Priorität hochgeladen.

11.2.29 Auslesen von Keywords aus ID3-Tags

Datum:	14.07.05
Verantwortliche Person:	Paul, Boris, David
Was wird erwartet:	ID3-Tags werden als Keywords hinzugefügt.
Was ist passiert:	Alle Tags wurden korrekt extrahiert.

12 Anhang B1 - Diagramme, 1. Semester

12.1 Anwendungsfalldiagramm

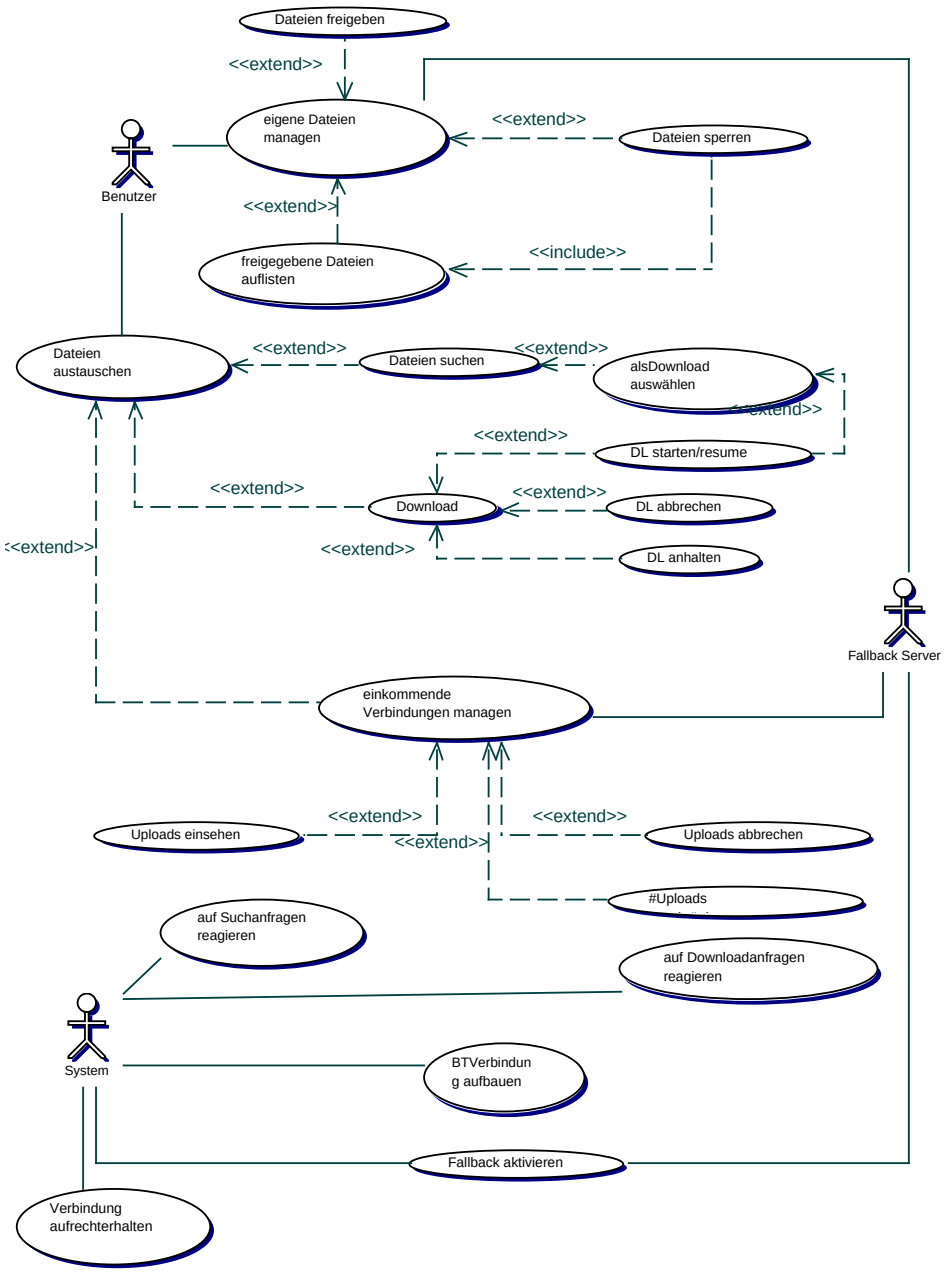

Abbildung 2: Anwedungsfalldiagramm

12.2 Aktivitätsdiagramme

12.2.1 Suchanfrage Erstellen

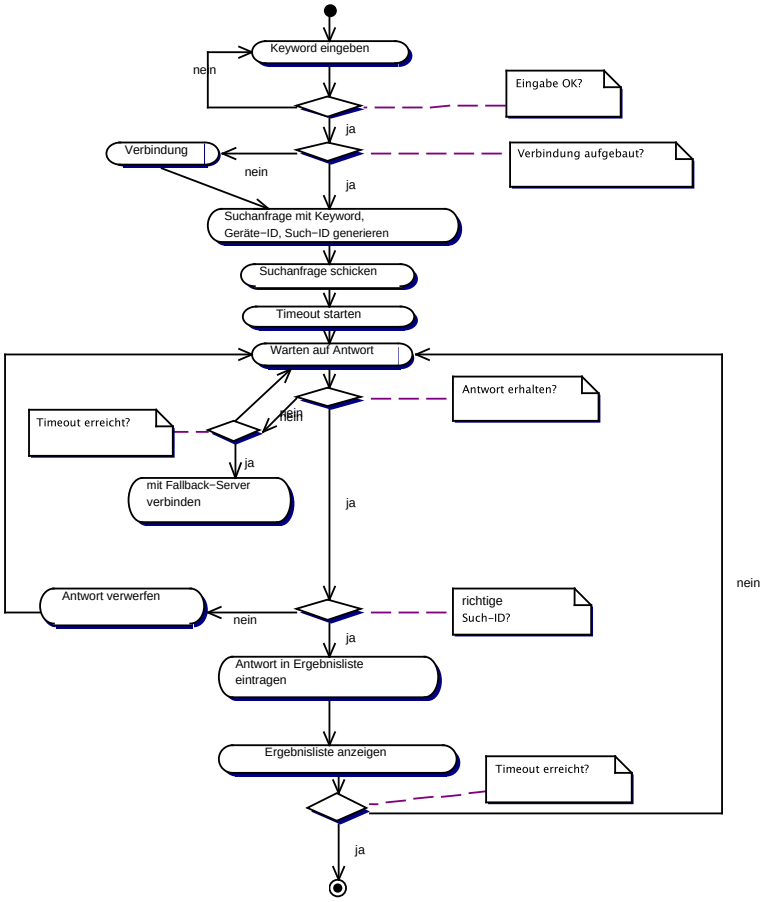

Abbildung 3: Suchanfrage erstellen

12.2.2 Download Starten

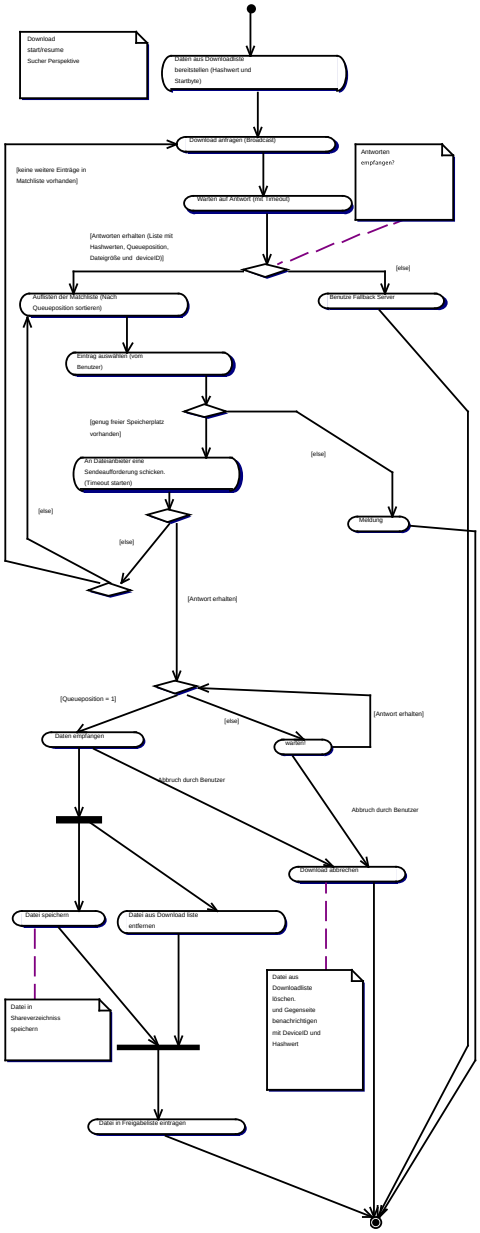

Abbildung 4: Download starten

12.2.3 Auf Suchanfrage Reagieren

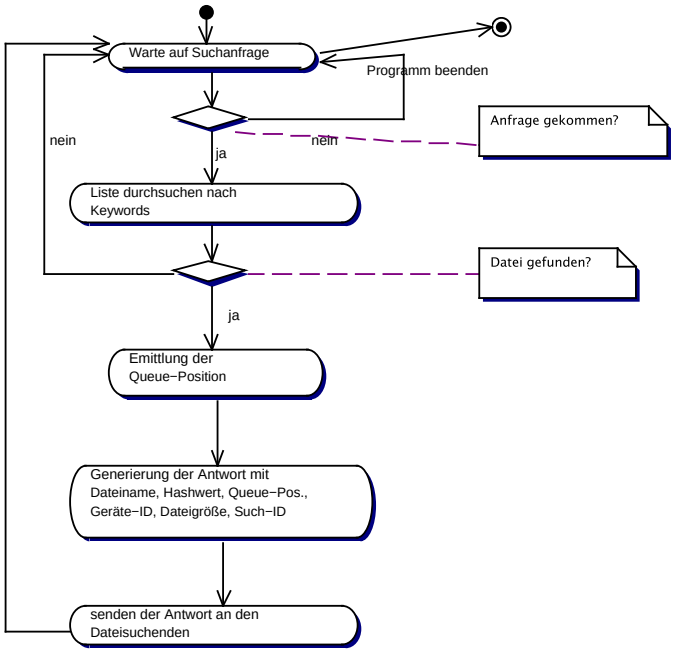

Abbildung 5: Auf Suchanfrage reagieren

12.2.4 Auf Downloadanfrage Reagieren

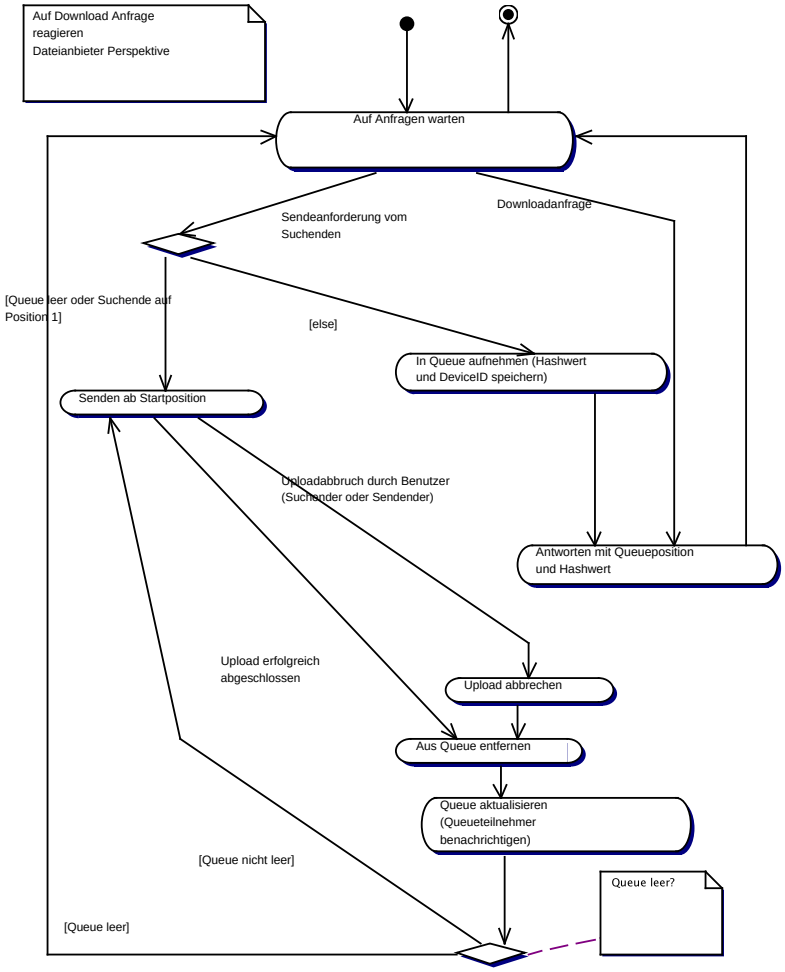

Abbildung 6: Auf Downloadanfrage reagieren

12.2.5 Verbindung Aufbauen

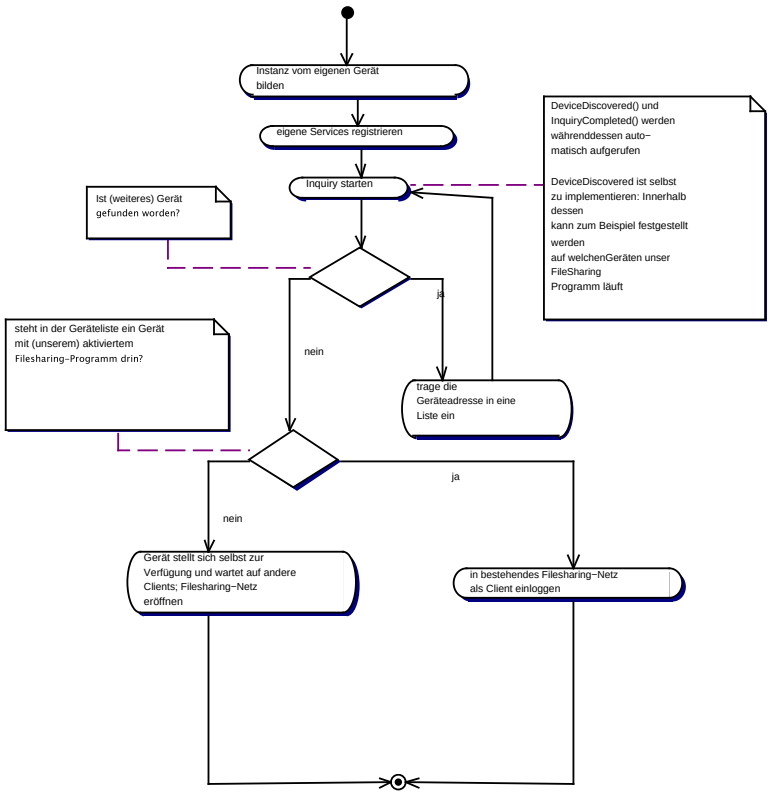

Abbildung 7: Verbindung aufbauen

12.2.6 Verbindung Aufrechterhalten

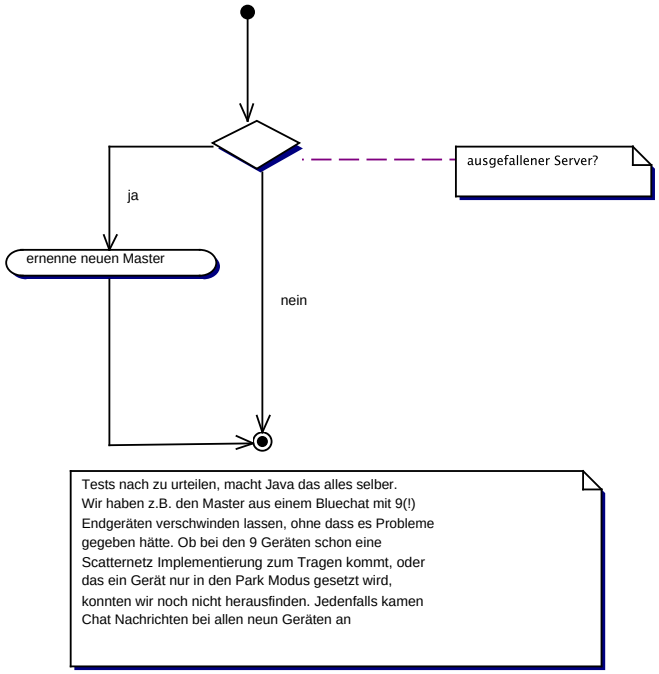

Abbildung 8: Verbindung aufrechterhalten

12.3 Sequenzdiagramme

12.3.1 Suchanfrage Erstellen

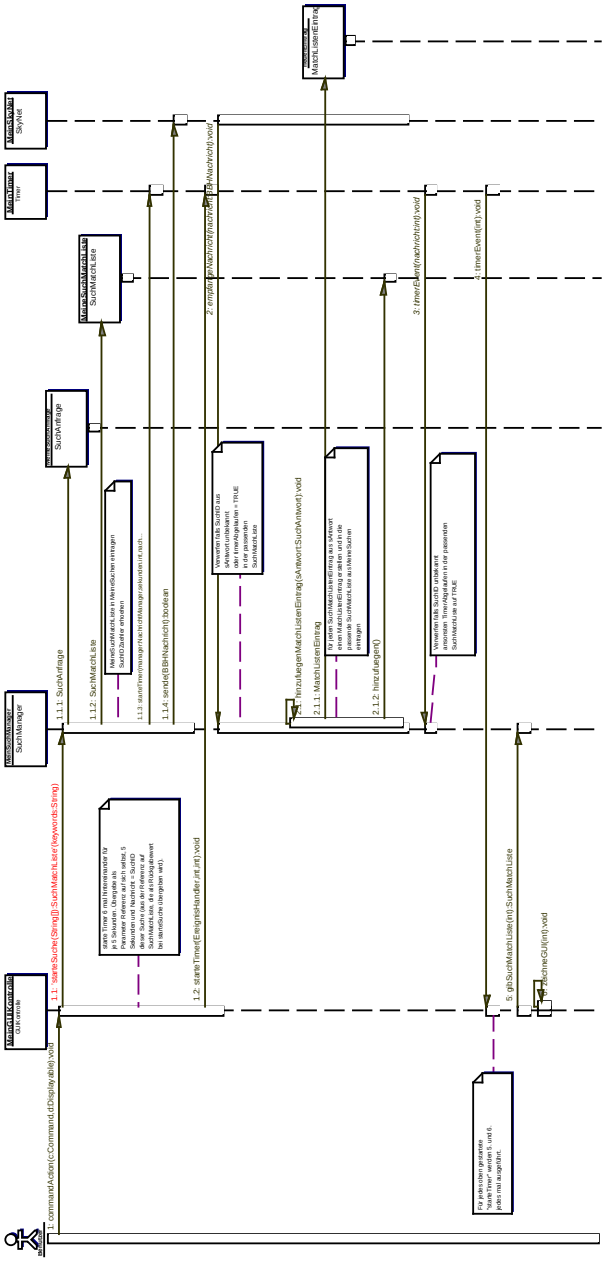

Abbildung 9: Suchanfrage erstellen

12.3.2 Download Starten

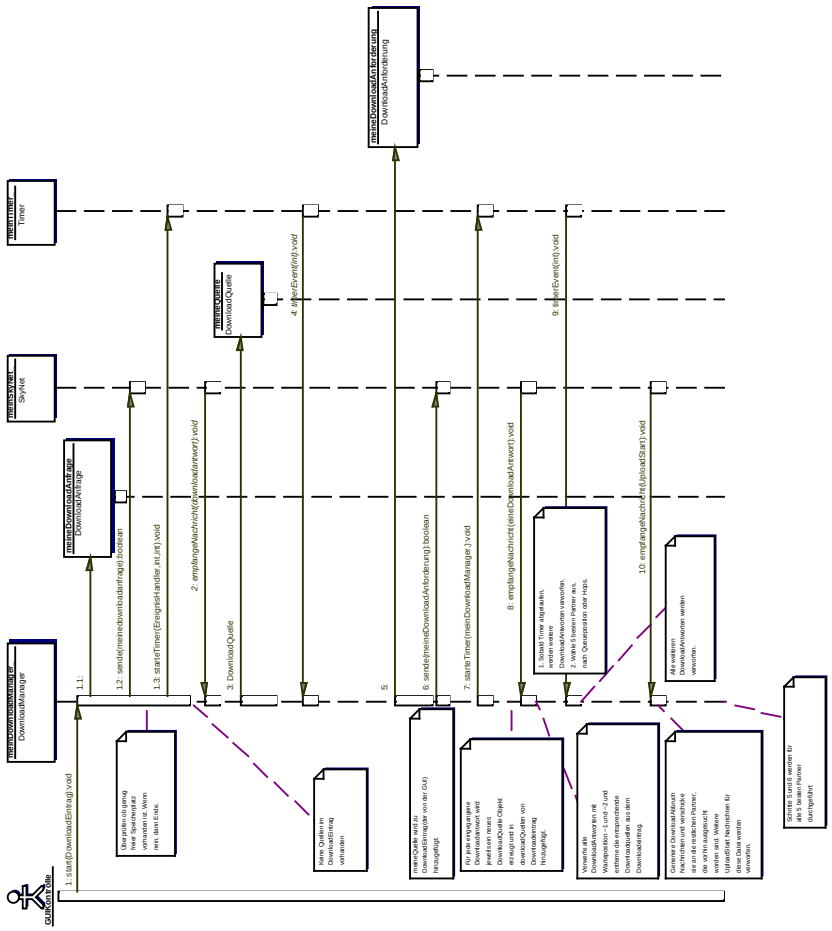

Abbildung 10: Download starten

12.3.3 Auf Suchanfrage Reagieren

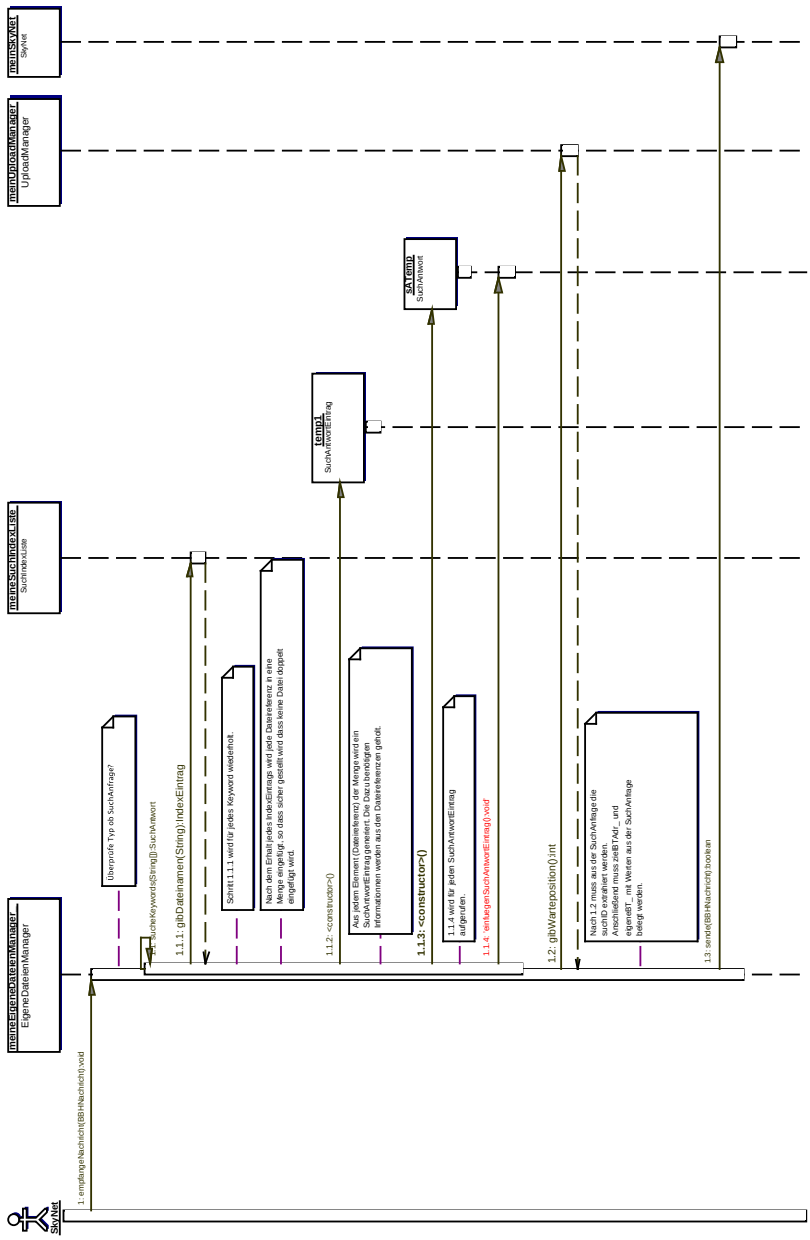

Abbildung 11: Auf Suchanfrage reagieren

12.3.4 StarteApp

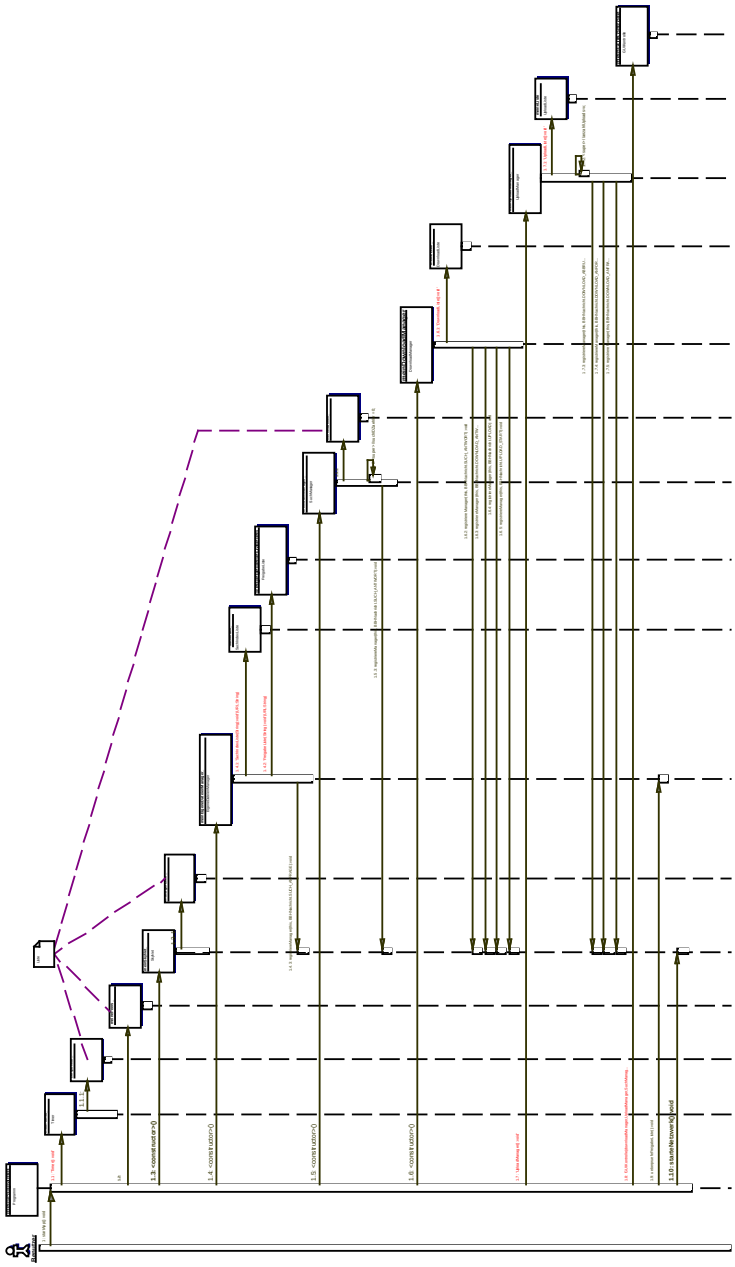

Abbildung 12: Applikation starten

12.3.5 Auf Downloadanfragen Reagieren

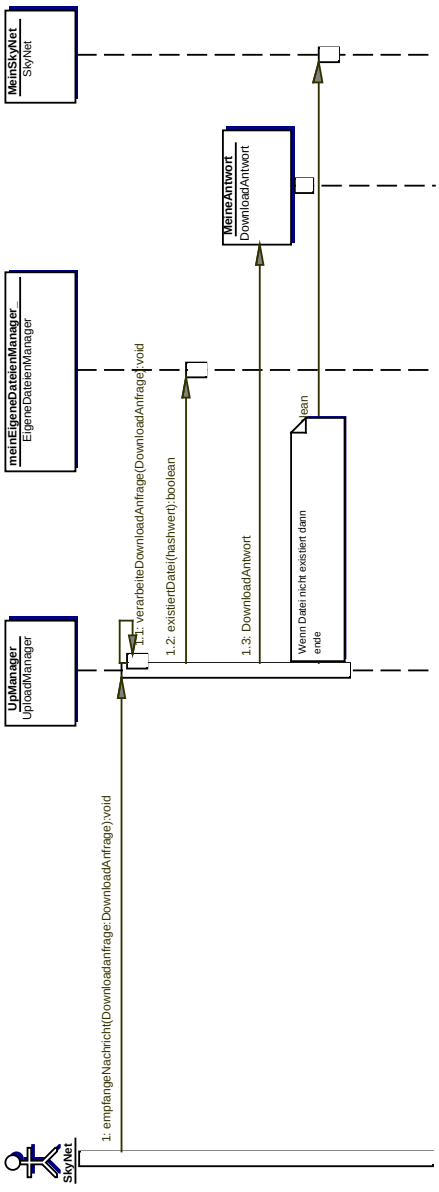

Abbildung 13: Auf Downloadanfrage reagieren

12.3.6 Auf Downloadanforderungen reagieren

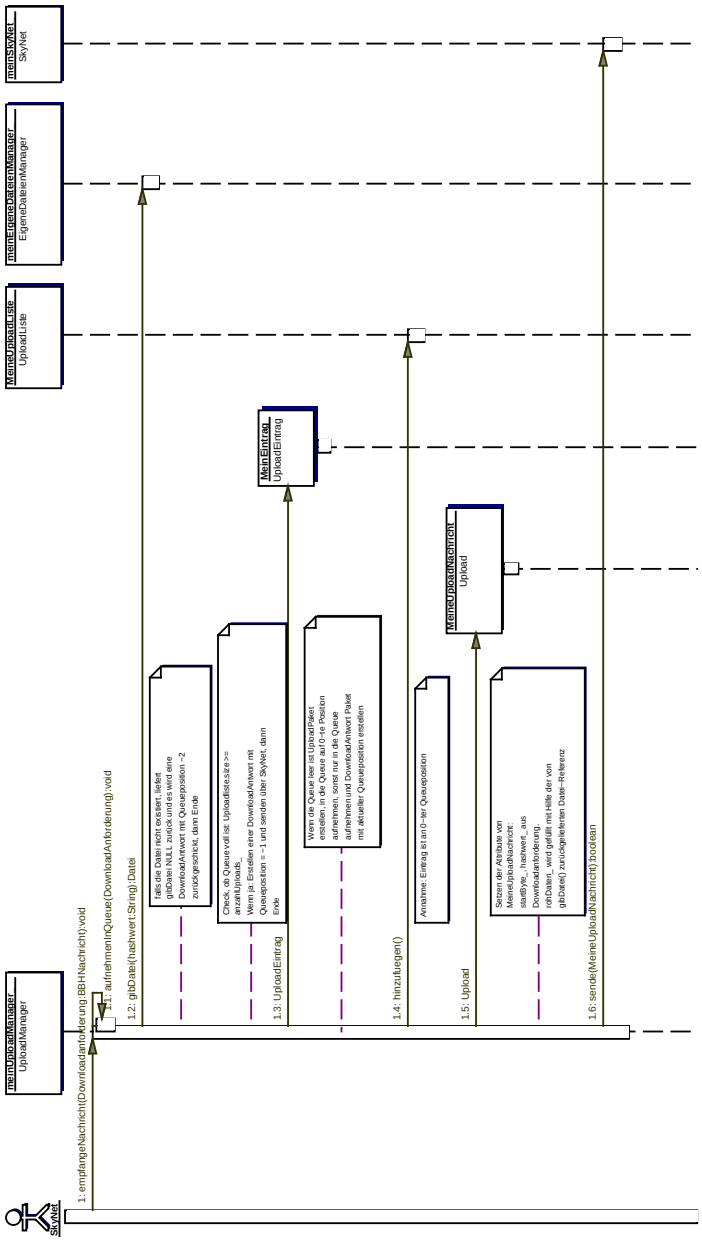

Abbildung 14: Auf Downloadanforderung reagieren

13 Anhang B2 - Diagramme, 2. Semester

13.1 Aktivitätsdiagramme

13.1.1 Baumknoten

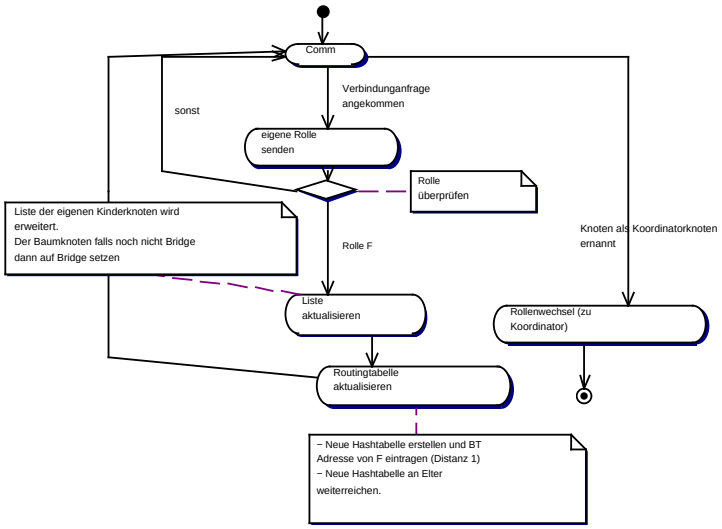

Abbildung 15: Baumknoten

13.1.2 Fehlerbehandlung

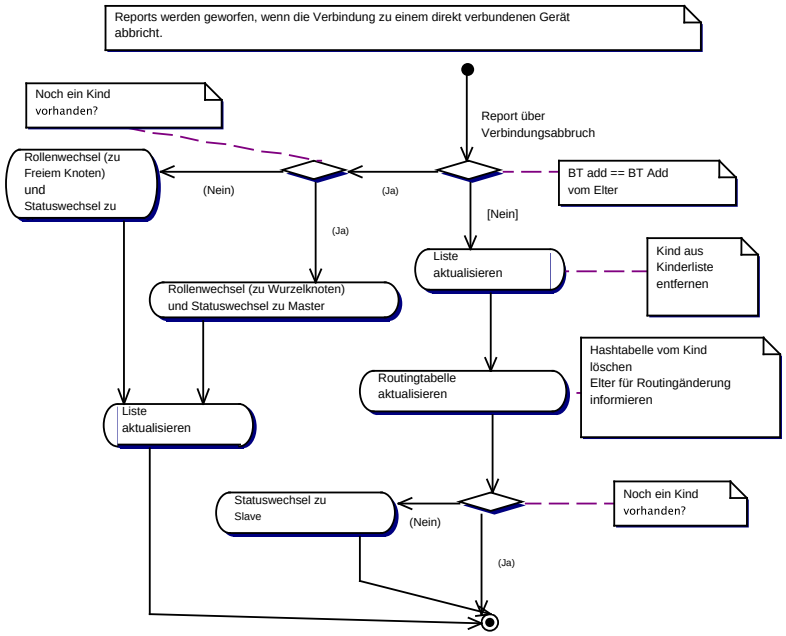

Abbildung 16: Fehlerbehandlung

13.1.3 Freierknoten

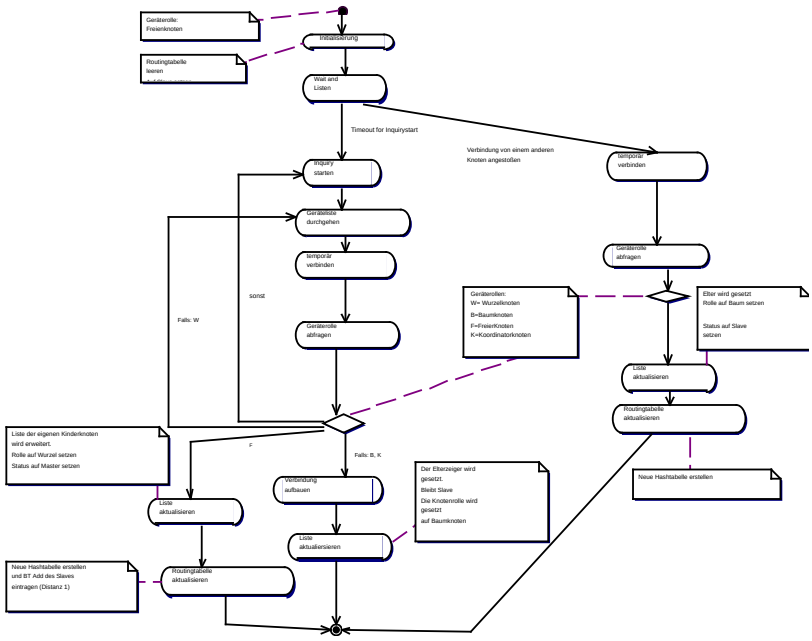

Abbildung 17: Freierknoten

13.1.4 Koordinatorknoten

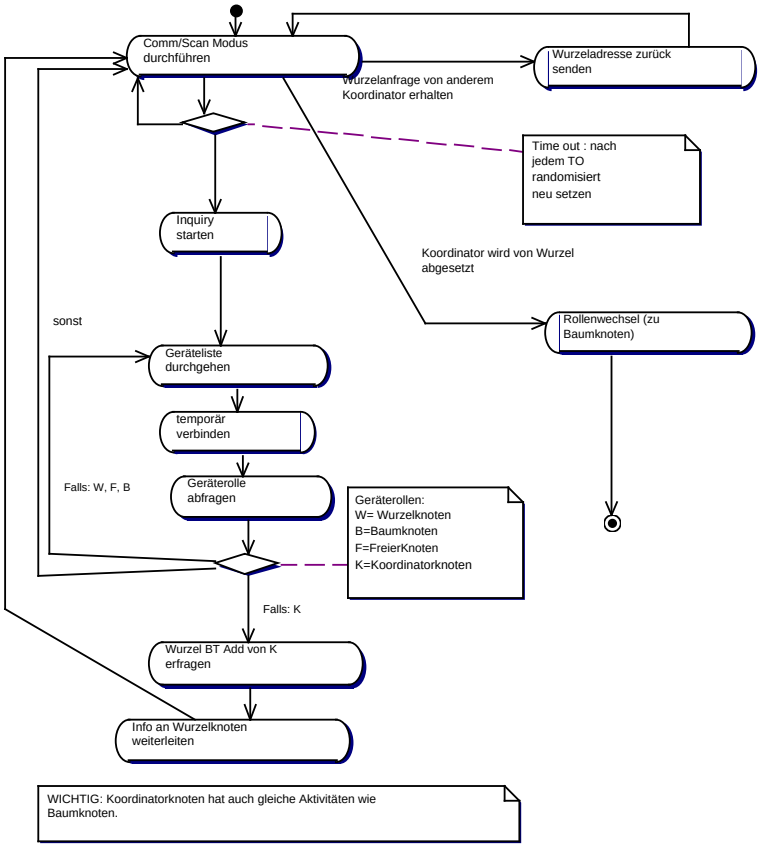

Abbildung 18: Koordinatorknoten

13.1.5 Master-Slave-Entscheidung

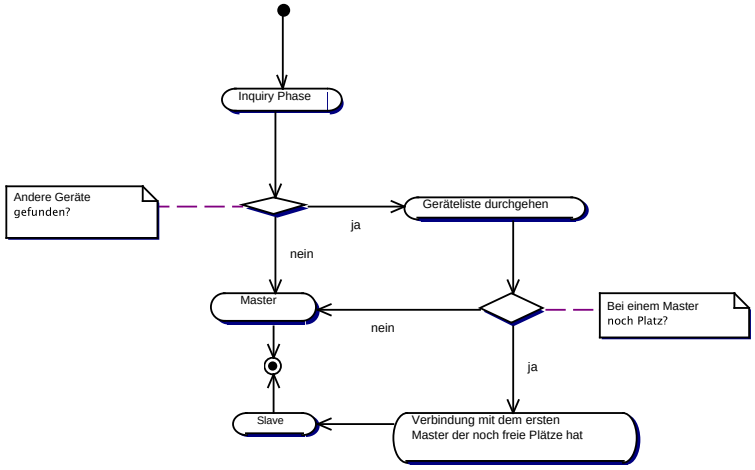

Abbildung 19: Master-Slave-Entscheidung

13.1.6 Routingaenderung

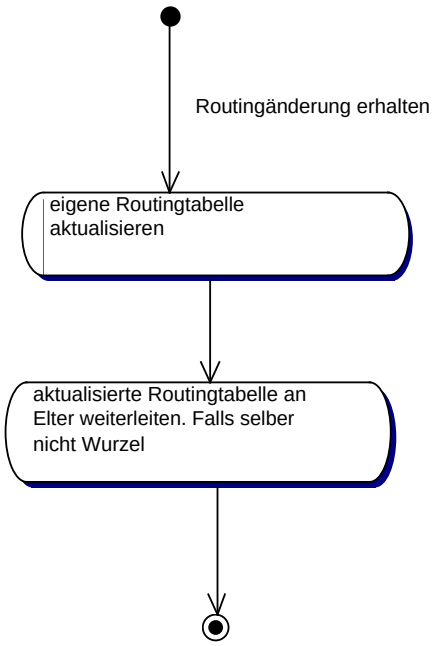

Abbildung 20: Routingänderung

13.1.7 Wurzelknoten

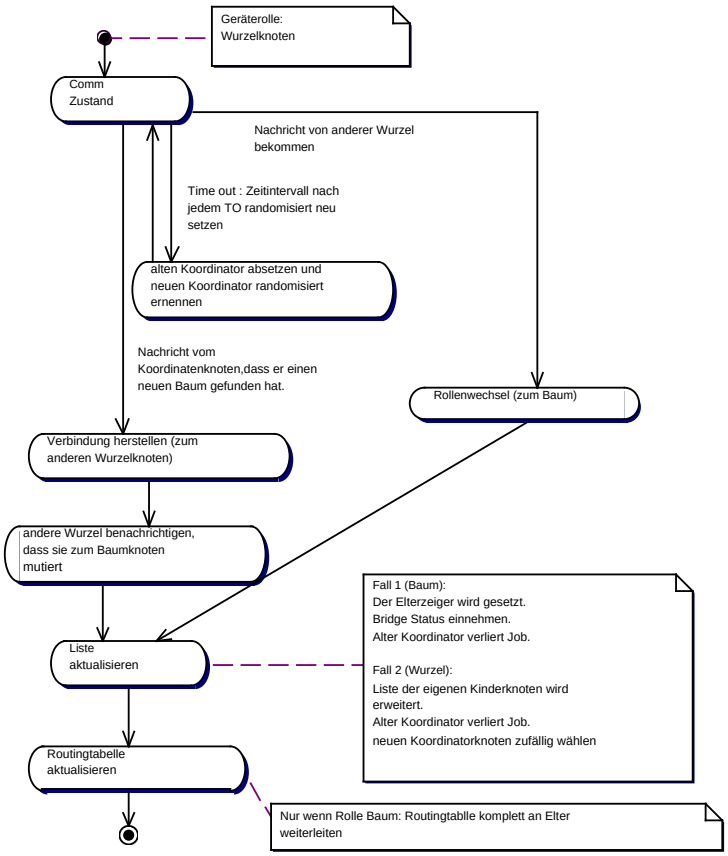

Abbildung 21: Wurzelknoten

13.2 Sequenzdiagramme

13.2.1 2 Bäume durch Wurzel vereinigen (aktiv)

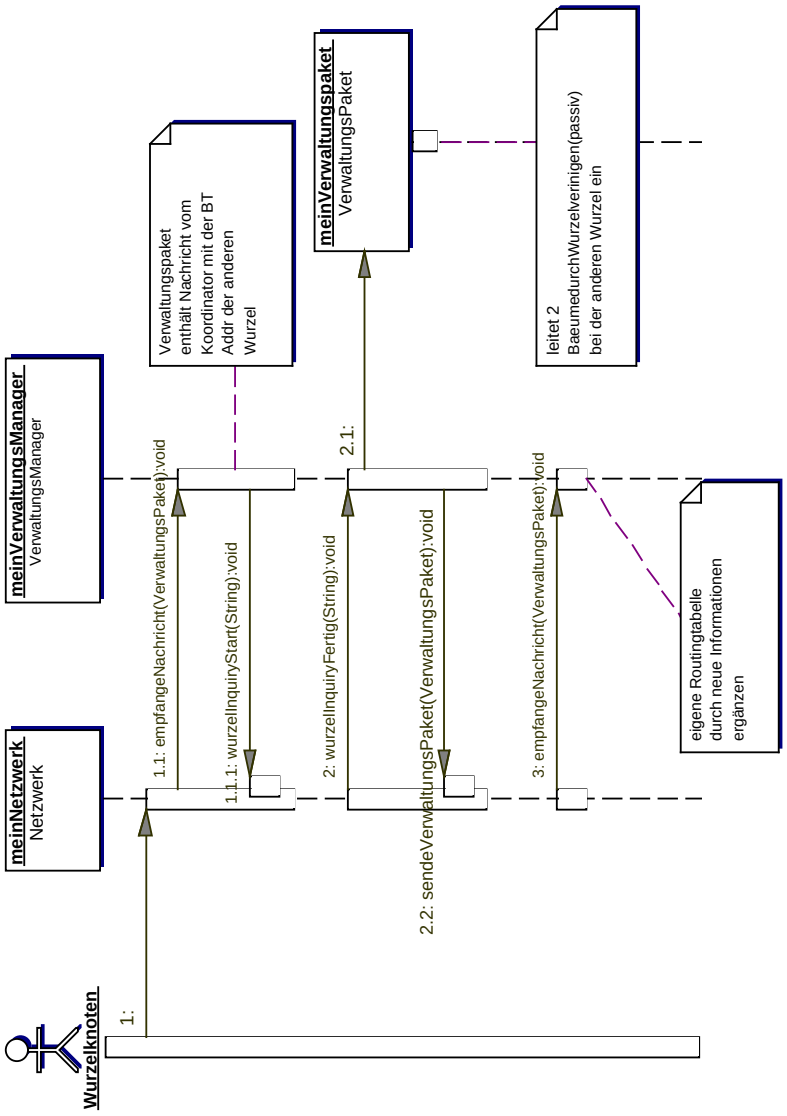

Abbildung 22: Zwei Bäume durch Wurzel vereinigen, aktiv

13.2.2 2 Bäume durch Wurzel vereinigen (passiv)

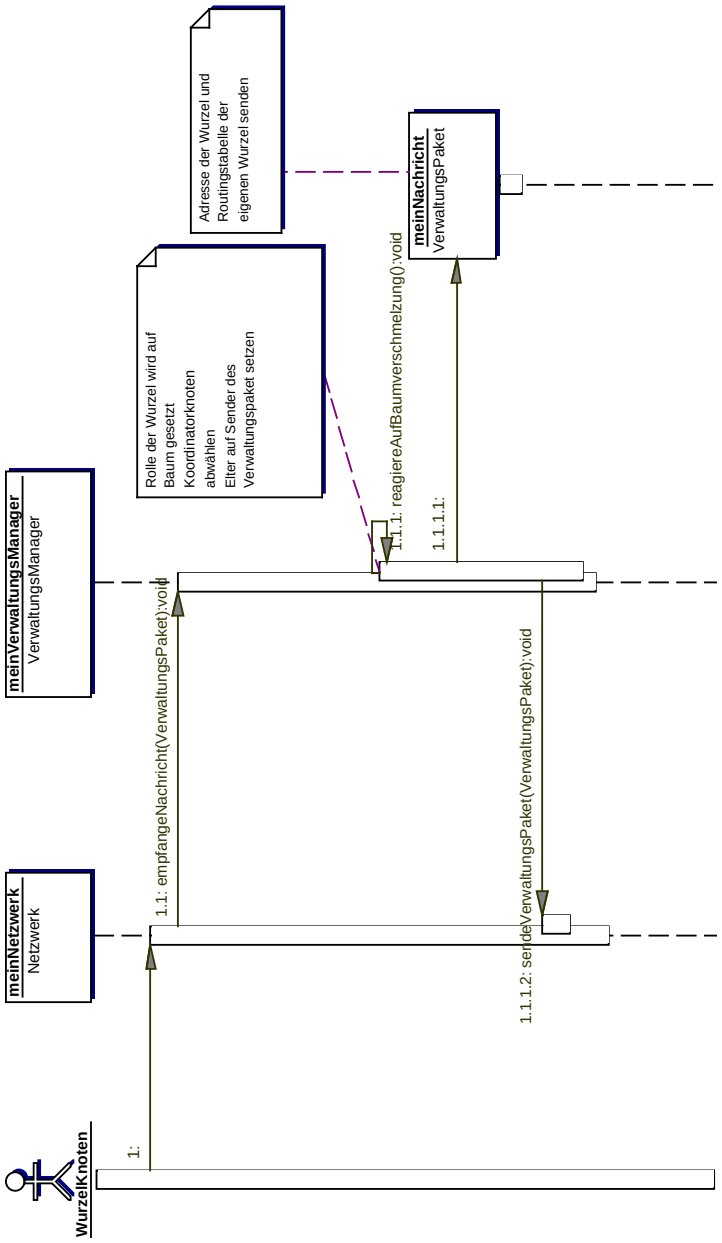

Abbildung 23: Zwei Bäume durch Wurzel vereinigen, passiv

13.2.3 2 Bäume vereinigen (Koordinator)

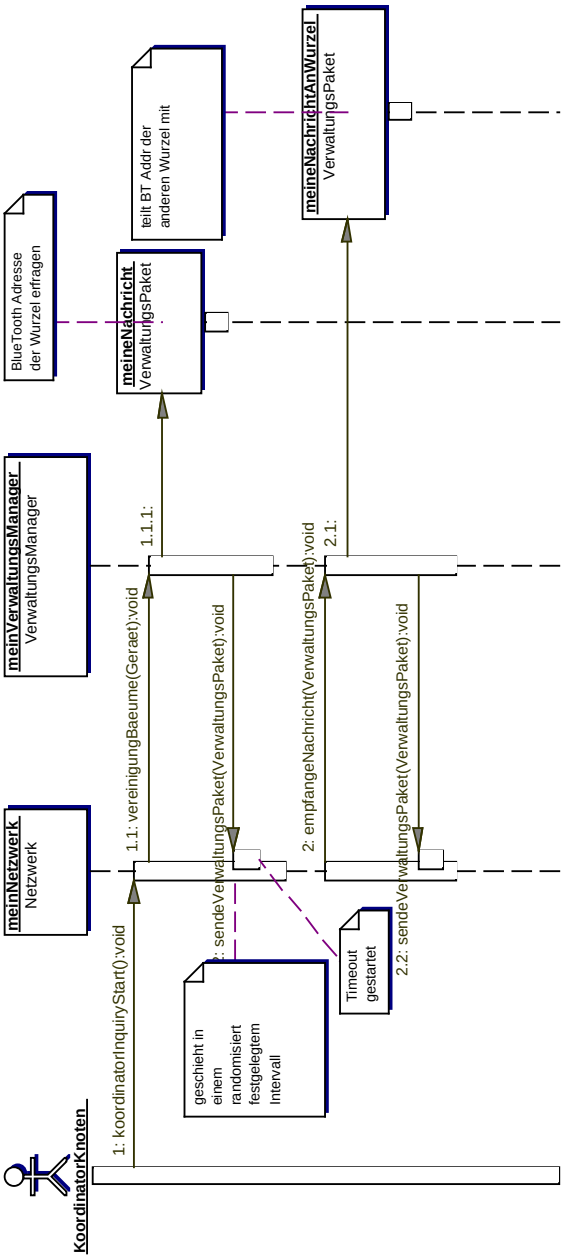

Abbildung 24: Zwei Bäume vereinigen, Koordinator

13.2.4 Freien Knoten integrieren

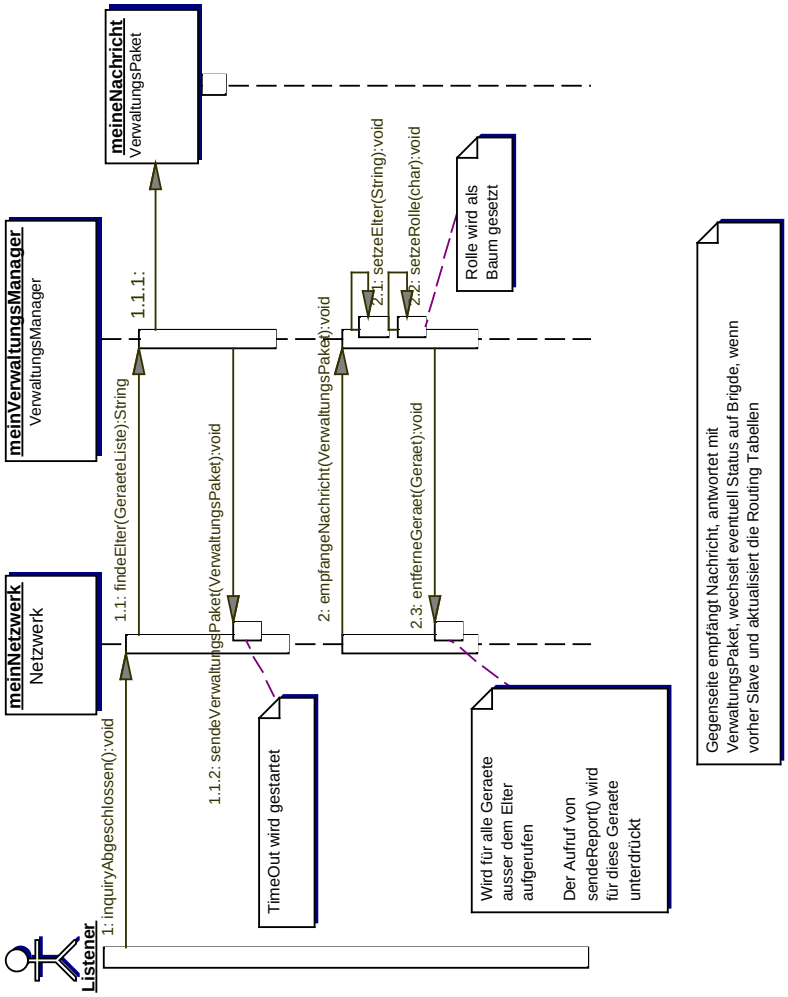

Abbildung 25: Freien Knoten integrieren

13.2.5 Piconetz

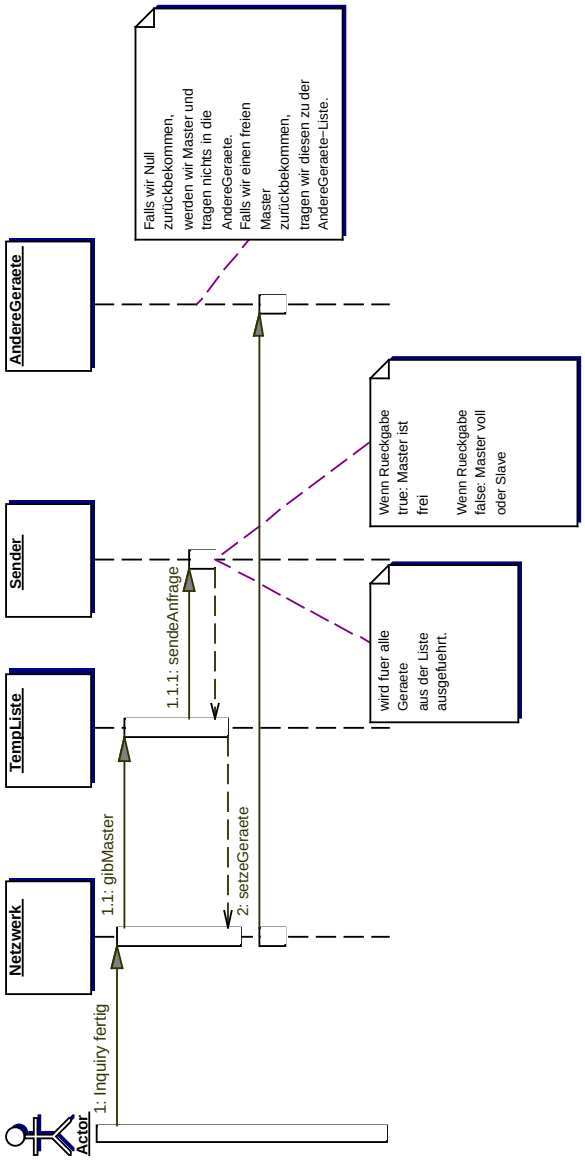

Abbildung 26: Piconetz aufbauen

13.2.6 Routing von BBH-Paketen

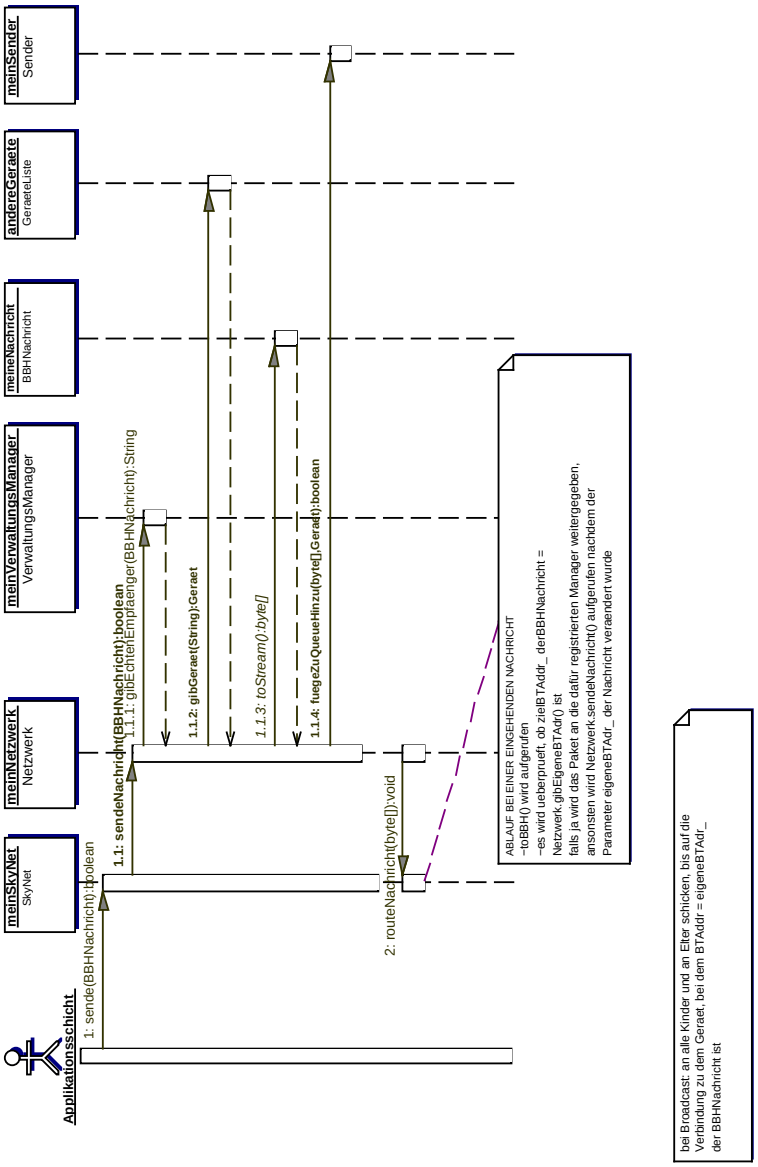

Abbildung 27: Routing von BBH-Paketen

13.2.7 Verbindungstrennung abfangen

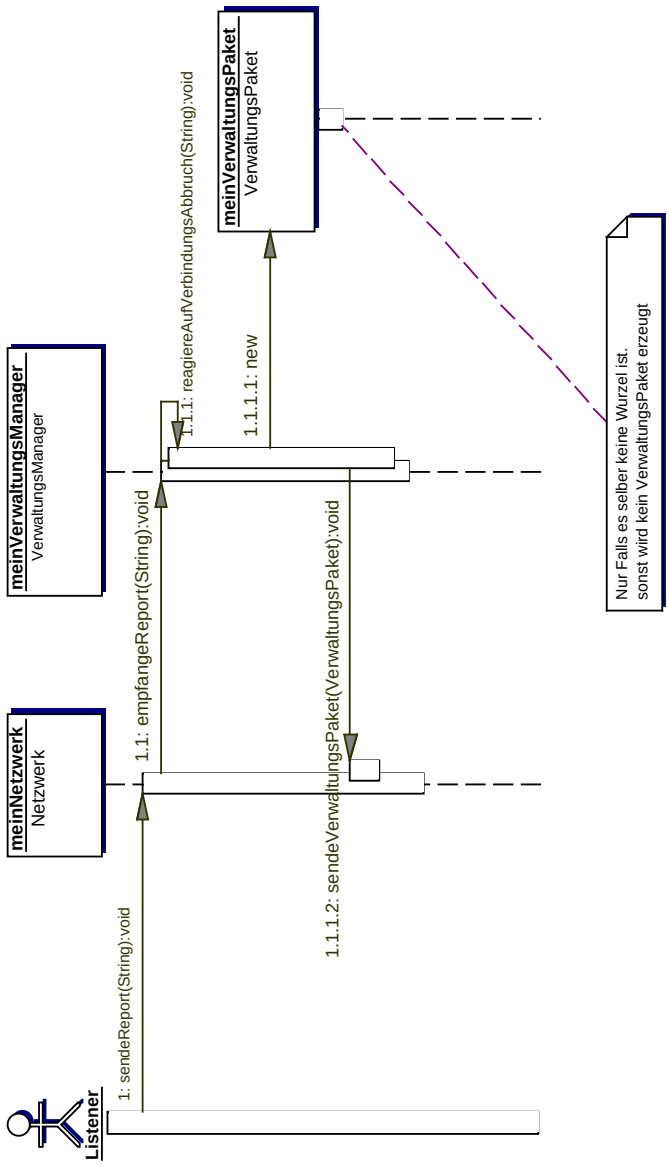

Abbildung 28: Verbindungstrennung abfangen

14 Anhang B3 - Klassendiagramm

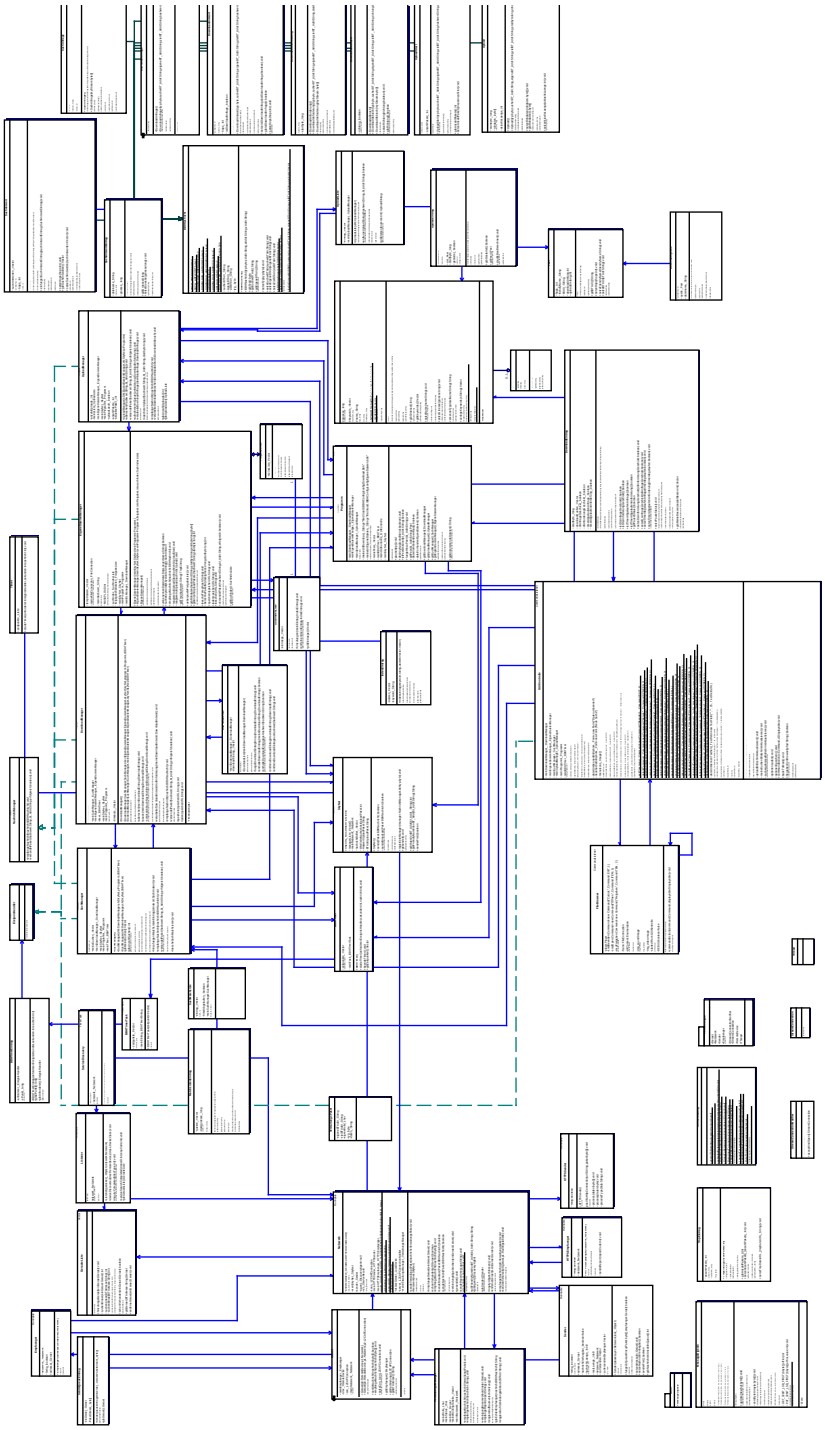

Abbildung 29: Klassendiagramm

15 Anhang C - Literaturverzeichnis

Literatur

[1] Rüdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, http://csdl.computer.org/comp/proceedings/p2p/2001/1503/00/15030101.pdf

[2] Peer-to-Peer DatenManagement; Prof. Dr. Kai-Uwe Sattler, Computer Science and Automation Department Technical University Ilmenau, http://www3.tu-ilmenau.de/site/dbis/fileadmin/template/FakIA/StruktFakultaet_IA/ipim/dbis/vdm/vdm-1.pdf

[3] Peer-to-Peer Computing, Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins 2 ,Zhichen Xu; HP Laboratories Palo Alto, http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf

[4] P2P Systems, Keith W. Ross Polytechnic University: http://cis.poly.edu/~ross, Dan Rubenstein Columbia University: http://www.cs.columbia.edu/~danr, http://cis.poly.edu/~ross/tutorials/P2PtutorialInfocom.pdf

[5] Peer-to-Peer Computing, Björn Jung, Technische Universität Kaiserslautern, Datenbanken und Informationssysteme, http://wwwdvs.informatik.uni-kl.de/courses/seminar/SS2004/ausarbeitung2.pdf

[6] 1. Einführung (S.Gebhardt), http://dbs.uni-leipzig.de/en/seminararbeiten/semWS0304/ar-beit1/ausarbeitung1.pdf

[7] 2. File Sharing Systeme (Napster, Gnutella, KazaA .) (X.Baldauf), http://dbs.uni-leipzig.de/en/seminararbeiten/semWS0304/arbeit2/Skript.pdf

[8] Routing Indices for Peer-to-Peer Systems; Arturo Crespo, Hector Garcia-Molina Computer Science Department Stanford University Stanford, CA 94305-2140, USA, http://www-db.stanford.edu/~crespo/publications/crespoa_ri.pdf

[9] CHORD: A Scalable Peer-to-peer Lookup Service for Internet Applications; Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. In the Proceedings of ACM SIGCOMM 2001, San Deigo, CA, August 2001, http://www.pdos.lcs.mit.edu/chord/

[10] CAN: A scalable Content-Addressable Network SIGCOMM 2001, Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, Dept. of Electrical Eng. & Comp. Sci. UC Berkeley Berkeley, CA, USA, AT&T Center for Internet Research at ICSI Berkeley, CA, USA

[11] Open Problems in Data-Sharing Peer-to-Peer Systems; Neil Daswani, Hector Garcia-Molina, and Beverly Yang Stanford University, Stanford CA 94305, USA, http://www-db.stanford.edu

[12] JXTA: An Open, Innovative Collaboration, Sun Microsystems, Inc., http://www.jxta.org

[13] Napster Webseite, http://www.napster.com

[14] Emule Webseite, http://www.emule-project.net

[15] Paralleler, Kooperativer Download mit eDonkey, http://www.edonkey2000.com/documenta-tion/mftp.html

[16] Kademlia: A Peer-to-peer Information System Based on the XOR Metric, Petar Maymounkov and David Mazieres, New York University, http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf

[17] Gnutella Homepage http://rfc-gnutella.sourceforge.net, The Annotated Gnutella Protocol Specification v0.4, http://rfc-gnutella.sourceforge.net/developer/stable/#t1

[18] KaZaA Homepage, http://www.kazaa.com

[19] P2P Papers (Übersicht über diverse Quellen), http://cis.poly.edu/~ross/p2pTheory/P2Prea-ding.htm

[20] 7DS Webseite, Maria Papadopouli, Henning Schulzrinne; Department of Computer Science Columbia University New York, NY 10027, http://www.cs.unc.edu/~maria/7ds/

[21] Design and Implementation of a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users

[22] Seven Degrees of Separation in Mobile Ad Hoc Networks

[23] Fundamental Challenges in Mobile Computing, M. Satyanarayanan, School of Computer Science Carnegie Mellon University, http://www-2.cs.cmu.edu/afs/cs/project/coda/Web/docdir/podc95.pdf

[24] iMobile ME - A Lightweight Mobile Service Platform for P2P Mobile Computing, Yih-Farn Chen, Huale Huang, Bin Wei, AT&T Labs - Research, USA; Ming-Feng Chen, National Chiao-Tung University; Taiwan Herman Rao Far Eastone Telecommunications Co., Ltd., Taiwan

[25] Proem: A Middleware Platform for Mobile Peer-to-Peer Computing, Gerd Kortuem, Department of Computer Science, University of Oregon, USA, ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), Special Feature on Middleware for Mobile Computing, Vol. 6, No 4, October 2002, www.cs.uoregon.edu/research/wearables/papers.html

[26] XMIDDLE: A Data-Sharing Middleware for Mobile Computing, Cecilia Mascolo, Licia Capra, Stefanos Zachariadis and Wolfgang Emmerich, Dept. of Computer Science, University College London, UK

[27] FolkMusic: A Mobile Peer-to-Peer Entertainment System, Mikael Wiberg, Department of Informatics, Umea, Schweden

[28] A Platform and Applications for Mobile Peer-to-Peer Communications, Takeshi Kato, Norihiro Ishikawa, Hiromitsu Sumino, Johan Hjelm, Ye Yu, Shingo Murakami, Ericsson Research (Stockholm Schweden, Kanagawa Japan), NTT DoCoMo Inc.(Kanagawa Japan)

[29] SETI@home Homepage, http://setiathome.ssl.berkeley.edu/index.html

[30] Bluetooth Spezifikation v1.2, Bluetooth SIG, http://www.bluetooth.org/spec

[31] JAVA API for Bluetooth Wireless Technology (JSR-82) 1.0a, Motorola, Wireless Software, Applications and Services

[32] C. Lindemann and O. P. Waldhorst, Passive Distributed Indexing (PDI) v1.0, Internet-Draft 2004

[33] C. Lindemann and O. P. Waldhorst, Customizing a Distributed Lookup Service for Specific Mobile Applications, Department of Computer Science University of Dortmund 2004

[34] C. Lindemann and O. Waldhorst, Consistency Mechanisms for a Distributed Lookup Service supporting Mobile Applications, MobiDE 2003

[35] C. Lindemann and O. Waldhorst, A Distributed Search Service for Peer-to- Peer File Sharing in Mobile Applications

[36] C. Lambert,C. Lindemann and O. Waldhorst, PG 466 Beyond KaZaA -Peer-to-Peer Systeme für mobile Endgeräte mit Bluetooth und UMTS Technologie

[37] https://www.cvshome.org/docs/

[38] http://www.kde.org/apps/cervisia/

[39] http://www.wincvs.org/

[40] http://sources.redhat.com/autobook/

[41] http://autotoolset.sourceforge.net/tutorial.html

[42] http://sources.redhat.com/gdb/documentation/

[43] http://www.gnu.org/manual/ddd/

[44] http://www.bluemarsh.com/java/jswat/

[45] NS by Example, http://nile.wpi.edu/NS/

[46] The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/v

[47] Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator „ns", http://www.isi.edu/nsnam/ns/tutorial/index.html

[48] UCBT - Bluetooth extension for NS2 at Univ. of Cincinnati http://www.ececs.uc.edu/~cdmc/ucbt/

[49] Blueware: Support for Self-organizing Scatternets http://nms.lcs.mit.edu/projects/blueware/

[50] Godfrey Tan, Self-organizing Bueltooth Scatternets, Massachusetts Institute of Technology, Januar 2002

[51] http://today.java.net/pub/a/today/2004/07/27/bluetooth.html

[52] Bluetooth for Java, Bruce Hopkins and Ranjith Antony, Apress l' 2003

[53] Java APIs for Bluetooth Wireless Technology (JSR-82) 2002, 6501 William Cannon Drive West MD: OE112 Austin, TX 78735-8598

[54] http://medien.informatik.uni-ulm.de/lehre/current/PS%20Mobile%20JAVA/Bluetooth.pdf

[55] http://www-ds.e-technik.uni-dortmund.de/embedded/de/textonly/content/diplom/java_-bluetooth/java_bluetooth.html

[56] http://www.fm-i.de/de/downloads/pm_270504_activepilot_Technik.doc

[57] http://java.sun.com/xml/jaxp/index.jsp

[58] http://java.sun.com/j2se/1.4.2/docs/api/index.html

[59] http://developers.sun.com/techtopics/mobility

[60] http://java.sun.com/j2me/index.jsp

[61] Die offizielle Java Website java.sun.com von Sun Microsystems

[62] J2ME Developer's Guide von Michael Kroll und Stefan Haustein, Markt + Technik Verlag, 2003

[63] Java 2 Micro Edition: Klaus-Dieter Schmatz 2004, dpunk