Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Studienarbeit

Nutzung eines Bluetooth-Headsets zur Sprachsteuerung eines PCs

vorgelegt von
Pascal Pein
am 4. Juni 2004

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Kai von Luck

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In den letzten Jahrzehnten hat die Computertechnik immer mehr Einfluss auf das tägliche Leben. Beispielsweise war das Internet vor 10 Jahren noch eine Technologie, die nur von wenigen Menschen genutzt wurde. Heutzutage besitzen sehr viele Privathaushalte einen PC mit Internetzugang. Möglich wurde diese Entwicklung nur durch eine Vereinfachung der PC-Bedienung. Grafisch orientierte Betriebssysteme wie OS/2 und Windows 95 lösten die kryptische Kommandozeile ab. Damit wurde es auch Laien ermöglicht, ohne Vorkenntnisse Computer und Internet zu nutzen.

Langsam kommen die Grenzen der Bedienung mit Tastatur und Maus in Sicht. Der Trend zu immer komplexeren Systemen erhöht den Bedarf an einer intuitiven Benutzerführung. Immer leistungsfähigere Prozessoren ermöglichen die Erschließung neuer Schnittstellen, die die Bedienung der Hardware erleichtern können.

Das Projekt SmartKom (DFKI GmbH 2004) beschäftigt sich mit Möglichkeiten der multimodalen Eingabe. Es wird untersucht, mit welchen Mitteln die Interaktion zwischen Mensch und Maschine verbessert werden kann.

Ein Pionier dieser Denkweise ist Mark Weiser, der 1988 den Begriff des *Ubiquitous Computing* (Weiser 1996) prägte. Seine Vision ist die Abkehr von der bewussten Interaktion mit Technologie. Die Geräte mit denen der Mensch umgeht, treten dabei in den Hintergrund. Lediglich die angebotenen Dienste werden direkt wahrgenommen.

Die derzeit am häufigsten verwendeten Eingabegeräte am PC sind Tastatur und Maus. Diese Art der Interaktion erzwingt einen direkten Kontakt zu den Eingabegeräten, wodurch zwangsläufig die Mobilität des Benutzers stark eingeschränkt ist. Zusätzlich werden dabei beide Hände benötigt, die somit nicht mehr für andere Aufgaben zur Verfügung stehen.

Ein Teilaspekt der mulitmodalen Eingabe ist die Verwendung von Sprache. Handhelds / PDAs ermöglichen zwar seit einiger Zeit eine hohe Mobilität, benötigen aber immer noch beide Hände für Eingaben. Spracherkennungssoftware ist in der Regel auf einem leistungsfähigen PC und ein Mikrofon angewiesen. Hierbei wird der Benutzer von Tastatur und Maus befreit, aber ist trotzdem an die Maschine gebunden.

Die Interaktion mit einem technischen System über Sprache wirft nicht erst Probleme bei der eigentlichen Spracherkennung auf. Je besser sich die Sprache des Benutzers von Hintergrundgeräuschen abhebt, desto einfacher ist die Auswertung des Signals. Daher ist es nicht ohne weiteres möglich, das Mikrofon direkt in das zu steuernde Gerät einzubauen. Denn ein großer Abstand zwischen Sprecher und Mikrofon verursacht eine verminderte Qualität der Aufnahme durch andere Geräuschquellen und Echos. Natürlich ist es möglich, diese Effekte durch verschiedene Tricks zu minimieren (Gelbart 2002).

Ein wesentlich einfacherer Ansatz, um dieses Problem zu lösen, ist die Nutzung eines kompakten, kabellosen Mikrofons, das nahe am Mund des Benutzers getragen wird. Ein positiver

Nebeneffekt dieses Vorgehens ist die Möglichkeit, mehrere im Raum befindliche Personen klar voneinander zu unterschieden. Die Idee des *Ubiquitous Computing* ist hierbei nicht vollständig umgesetzt, da der Benutzer ein Headset anlegen muss. Trotzdem ist es ein Schritt in die gewünschte Richtung. Die gewünschte Abkehr von den technikzentrierten Eingabegeräten ist bereits erkennbar.

Ich möchte mich an dieser Stelle bei der Firma Stollmann GmbH bedanken. Die Anfertigung dieser Studienarbeit wurde mit ihrer Hilfe wesentlich erleichtert. Ein großer Teil dieser Arbeit baut auf ihrer Software und Dokumentationen auf (3.4.4).

Ein besonderer Dank geht an meinen Betreuer in der Firma, Karsten Aalders. Bei auftretenden Problemen war er jederzeit ansprechbar. Auch sonst war der regelmäßige Gedankenaustausch fruchtbar.

Außerdem möchte ich mich bei allen Mitarbeitern der HAW-Hamburg bedanken, die mich bei der Ausarbeitung der Studienarbeit unterstützt haben. Insbesondere danke ich meinem Betreuer Prof. Dr. Kai von Luck, der die hilfreiche Kooperation mit Stollmann erst ermöglicht hat.

Die Studienarbeit beschreibt ein mögliches Design für die Verwendung eines Bluetooth-Headsets an einem PC. Kapitel 2 enthält die eigentliche Aufgabe und die einzuhaltenden Rahmenbedingungen. Im Kapitel 3 wird ein Überblick über die Technologien gegeben, die zur Lösung der Aufgabe genutzt werden. Auf Basis dieser Technologien werden im Kapitel 4 das resultierende Gesamtdesign, sowie die einzelnen Komponenten im Detail vorgestellt. Abschließend enthält Kapitel 5 die Ergebnisse der Arbeit und Vorschläge für zukünftige Entwicklungen.

2 Aufgabe

2.1 Technische Möglichkeiten

Ein handelsübliches Bluetooth-Headset kann als kostengünstiges Eingabegerät genutzt werden. Es besitzt Mikrofon und Lautsprecher, die am Ohr befestigt werden. Es ist in der Lage, einen bidirektionalen Audiostream über eine Distanz von ca. 10 Metern drahtlos zu übertragen. Die Ansteuerung des Headsets wurde von der Bluetooth-SIG festgelegt und sollte somit vom Hardwarehersteller unabhängig sein.

Als Gegenstelle ist ein Audiogateway vorgesehen, das ebenfalls das Headset-Profil unterstützt. Dies ist entweder ein Handy oder ein beliebiges Gerät mit Bluetooth und der entsprechenden Software. Denkbar ist beispielsweise ein PC mit USB-Dongle und entsprechender Software.

2.2 bisherige Lösungen

Das normale Headset-Profil sieht keine direkte Kommunikation mit dem PC vor. Die Software, die bei einem handelsüblichen USB-Dongle enthalten ist, kann in der Regel nicht als Audio-Gateway dienen.

Eine Alternative sind Komplettlösungen mit kabellosem Headset, die ähnlich wie eine kabellose Maus/Tastatur arbeiten. Dafür ist eine konventionelle Soundkarte nötig, an die ein stationärer Empfänger angeschlossen wird, mit dem das Headset dann über Funk kommunizieren kann.

Der „Bluetooth Wireless Hub“ von Logitech bietet die Möglichkeit, ein Bluetooth-Headset mit dem PC zu verbinden (Logitech Inc. 2003) . Nach Angabe von Logitech arbeitet dieser Hub mit vielen verschiedenen Bluetooth-Geräten, auch von anderen Firmen, zusammen.

2.3 Ziel / Rahmenbedingungen

Ziel ist es, ein normales Bluetooth-Headset als Eingabegerät für PCs zu nutzen. Das Headset erlaubt standardmäßig die Kommunikation mit einem Audiogateway, meistens einem Handy. Bisher nicht vorgesehen ist die Möglichkeit, das Headset über ein USB-Dongle als normales Mikrofon am PC zu betreiben. An dieser Stelle wird ein Audiotreiber benötigt, der die Daten vom Headset als Audio-In an eine Spracherkennungssoftware weiterleitet. Aus designtechnischen Gründen ist bei dieser Gelegenheit die Implementierung eines Duplex-Treibers sinnvoll.

Der benötigte Bluetooth-Stack wird von der Firma Stollmann zur Verfügung gestellt. Der BlueCode+ erfüllt alle in Kapitel 3.4.4 genannten Anforderungen.

Die Echtzeit-Performance des Systems ist vorerst nebensächlich. Die Anwendung auf einem dedizierten PC wird angestrebt.

3 Technologieübersicht

Im Folgenden werden die Hardware- und Softwarekomponenten beschrieben, die für eine erste Realisierung der Aufgabe vorgesehen sind.

3.1 Hardware

Als Grundlage für die Bearbeitung der Aufgabe wird ein PC mit dem Betriebssystem Windows 2000/XP gewählt. Welches Bluetooth-Headset für die Audioaufnahme/-wiedergabe verwendet wird, ist an sich unerheblich, da das Headset-Profil eine exakt definierte Schnittstelle besitzt. Für die Tests wird das Headset AHS-10 der Firma Anycom verwendet. Um dem PC die nötige Bluetooth-Funktionalität zu geben, wird ein handelsübliches USB-Dongle angeschlossen. Es stehen unterschiedliche Geräte der Firmen Sitecom und Acer zur Verfügung.

3.2 Entwicklungsumgebung

Die verwendete Plattform ist das Betriebssystem Windows 2000. Es wird zudem angestrebt, die Lösung auch unter Windows XP laufen zu lassen. Für die Entwicklung des Codes wird das Tool „Visual Studio 6“ eingesetzt. Um Treiber entwickeln zu können, ist zudem das „Driver Development Kit 2003“ notwendig, das zudem einige Tools mitliefert.

Hinzu kommen ergänzende Tools, die die Treiberprogrammierung erleichtern. Das Programm „Debugview“ ermöglicht das Lesen von Debugmeldungen, die im Kernel auftreten. Für das Debugging werden zudem die „Symboldateien“ von Microsoft installiert. Eine große Hilfe beim Testen ist ein virtueller PC wie „VMWare“, der innerhalb von Sekunden auf einen konsistenten Zustand zurückgesetzt werden kann, wenn das System vom fehlerhaften Treiber beschädigt wurde.

3.3 Spracherkennung

Die Software, die für die Spracherkennung verwendet werden kann, soll möglichst unabhängig vom Rest der Aufgabe sein. Einige Programme sind beispielsweise bei Scansoft (ScanSoft, Inc. 2004) zu finden. Die wohl bekanntesten sind „Dragon NaturallySpeaking“ und „IBM ViaVoice“, die auch als vollwertiges Diktiersystem eingesetzt werden können. Für die Sprachsteuereung reichen aber in der Regel bereits einfachere Programme aus, die in der Lage sind, ein begrenztes Vokabular zuverlässig zu erkennen.

3.4 Bluetooth

Dieser Abschnitt beschreibt Eigenschaften der Bluetooth-Technologie, die im Zuge der Studienarbeit wichtig sind. Als Grundlage dient ein Buch, das eine Übersicht über alle wesentlichen Aspekte von Bluetooth gibt (Muller 2001). Technische Details sind aus den Whitepapern der Bluetooth SIG entnommen.

Die Bluetooth[1]-Technologie geht zurück auf eine Machbarkeitsstudie der Firma Ericsson im Jahr 1994. Ziel war es, eine Funkschnittstelle zwischen Mobiltelefon und Zubehör zu schaffen, die gleichzeitig preiswert, energiesparend und universell einsetzbar ist. Bisherige Erweiterungen benötigten entweder einen speziellen Slot im Handy oder ein Kabel für die Datenübertragung.

Diese Studie führte 4 Jahre später zur Gründung der *Bluetooth Special Interest Group* (SIG). Gründungsmitglieder waren Firmen wie Ericsson, IBM, Intel, Nokia und Toshiba. In den folgenden Jahren ist die Mitgliedsanzahl auf mehrere Tausend angewachsen. Aufgabe der SIG ist die Entwicklung eines offenen Standards zur Kommunikation.

3.4.1 Übertragungsmodi

Der Bluetooth-Standard legt 2 grundlegende Arten von Verbindungen fest, die von den Profilen genutzt werden können. Bis auf Audioverbindungen nutzen die meisten Komponenten des Bluetooth-Stacks (siehe 3.4.2) ACL.

Asynchronous Connectionless: Die Kommunikation zwischen 2 Geräten läuft in den meisten Fällen über eine ACL-Verbindung. Die Daten werden hierbei in unregelmäßigen Abständen paketweise verschickt. Eine Fehlerkorrektur durch erneutes Senden ist bei Bedarf möglich. Hauptaufgabe dieser Verbindung ist die sichere Übertragung von Daten. Es ist möglich, den Datenkanal sowohl symmetrisch als auch asymmetrisch zu nutzen. Im ersten Fall beträgt die maximale Übertragungsrate 433.9 kb/s in beide Richtungen. Bei asymmetrischem Betrieb kann in einer Richtung eine Rate von 723,2 kb/s erreicht werden. Für die Gegenrichtung verbleiben dann 57,6 kb/s.

Synchronous ConnectionOrientated: Dass die Wurzeln von Bluetooth im Handy-Bereich liegen, wird am zweiten Modus erkennbar. SCO-Verbindungen dienen der Übertragung von Sprache. Bei Bedarf ist es möglich bis zu 3 Duplex-Verbindungen gleichzeitig aufzubauen. Die Übertragung der einzelnen Datenpakete erfolgt dabei isochron und gleichzeitig in beide Richtungen. Eine Fehlerkorrektur falsch übermittelter Pakete ist wegen der Echtzeitanforderungen nicht möglich. Jeder Kanal bietet eine Übertragungsrate von 64 kb/s, was in etwa dem ISDN Standard entspricht.

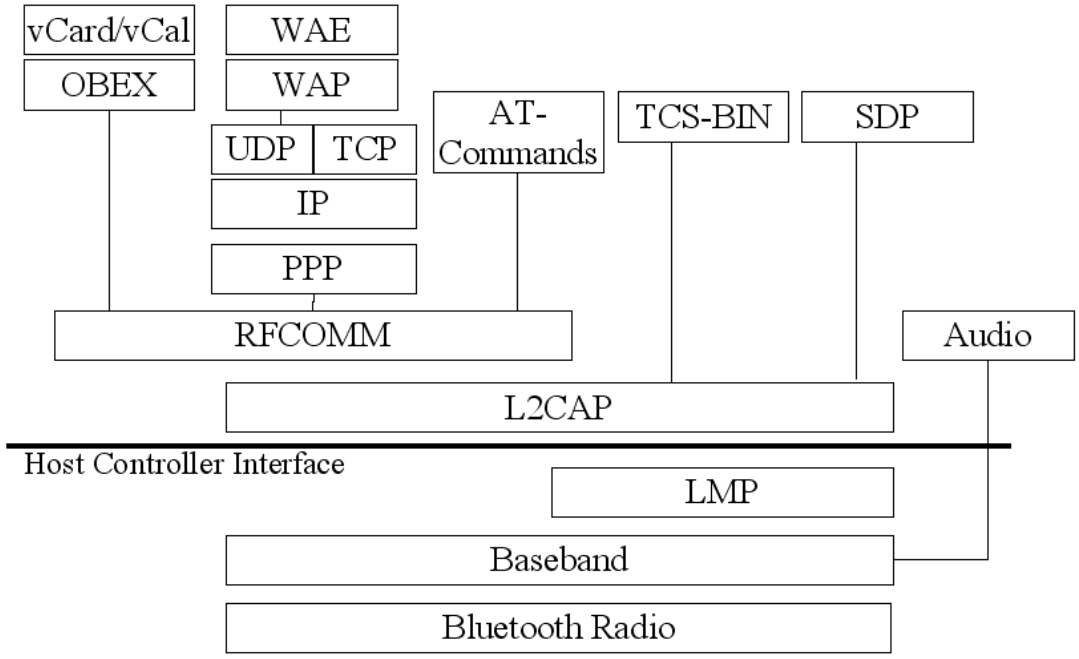

Abbildung 3.1: Bluetooth Stack (Mettala 1999, Seite 5)

3.4.2 Bluetooth-Stack

Der Bluetooth Stack enthält eine Sammlung verschiedener Protokolle, die für unterschiedliche Anwendungen genutzt werden können. Detaillierte Informationen sind im WhitePaper der SIG nachzulesen (Mettala 1999). Im Folgenden wird nur ein kurzer Überblick über die wichtigsten Eigenschaften des Stacks gegeben.

Die Abbildung 3.1 zeigt die Zusammenhänge zwischen den einzelnen Basisprotokollen. Dabei wird erkennbar, dass die meisten Protokolle aufeinander aufbauen. Typischerweise verwenden einzelne Programme nur bestimmte vertikale Schnitte des vorhandenen Stacks. Dabei sind die tieferen Schichten Bluetooth-spezifisch. In höheren Schichten werden meist bereits existierende Protokolle wie OBEX und IP verwendet. Dadurch wird die Anpassung älterer Programme deutlich vereinfacht.

Das Host Controller Interface (HCI) stellt ein Interface zur Verfügung, mit dem man Zugriff auf den Baseband Controller, den Link Manager und einige Hardwareeigenschaften hat. Es kann wie in Abbildung 3.1 unterhalb von L2CAP liegen oder in einigen Fällen auch darüber. Die exakte Position ist nicht festgelegt.

Die Protokolle können in 4 Untergruppen eingeteilt werden. Wichtigstes Kriterium ist hierbei die Herkunft des jeweiligen Protokolls, insbesondere der Einfluss der SIG.

[1] Der Name Bluetooth bezieht sich auf den dänischen Wikingerkönig Harald Blauzahn (940-985). Er führte dort das Christentum ein und vereinte die lokalen Stämme. Die Marketing-Analogie: Geräte unterschiedlicher Hersteller kommunizieren auf gemeinsamer Basis.

Die *Bluetooth Core Protocols* wurden komplett von der SIG entwickelt und sind eng mit der Bluetooth Technologie verknüpft. Sie werden von den meisten Bluetooth Geräten genutzt. Die anderen Protokollklassen sind dagegen optional.

Das *Cable Replacement Protocol* und die *Telephony Control Protocols* sind ebenfalls Bluetooth-spezifisch, aber sie basieren auf bereits vorhandenen Standards. Zusammen mit den *Adopted Protocols* sind sie anwendungsorientierte Protokolle.

Bluetooth Core Protocol

Baseband: Das Baseband arbeitet nahe an der Hardware und ermöglicht physikalische Verbindungen. Die zwei möglichen Arten sind ACL und SCO (siehe 3.4.1). Eine Besonderheit ist die Audio-Verbinding, die nicht über L2CAP gesteuert wird.

LMP: Das *Link Manager Protocol* ist verantwortlich für Verbindungsaufbau und Kontrolle. Diese Schicht ist außerdem für Verschlüsselung, Authentifikation und die Energieverwaltung zuständig.

L2CAP: Das *Logical Link Control and Adaptation Protocol* sorgt für die Vermittlung zwischen dem Baseband und den upper layer Protokollen. Es arbeitet parallel zum LMP und ist nur für ACL Verbindungen spezifiziert.

SDP: Das *Service Discovery Protocol* ist ein entscheidender Teil des Bluetooth Frameworks. Sämtliche Nutzungsmodelle basieren auf der Möglichkeit, andere Geräte detailliert nach ihren Diensten fragen zu können.

Cable Replacement Protocol

RFCOMM: RFCOMM emuliert die Kommunikation über ein serielles RS-232 Kabel. Die meisten höher liegende Dienste nutzen diesen Transportmechanismus.

Telephony Control Protocol

TCS BIN: Das *Telephony Control protocol - Binary* ist bit-orientiert. Es wird genutzt, um Sprachverbindungen zwischen Bluetooth Geräten zu steuern.

AT-Commands: Die Bluetooth SIG hat AT-Kommandos definiert, die für Mobiltelefone und Modems genutzt werden können.

Adopted Protocol

PPP: Das *Point-To-Point Protocol* läuft über RFCOMM und ermöglicht Punkt-zu-Punkt Verbindungen.

TCP/UDP/IP: Die weltweit meist genutzte Protokollgruppe wird von einer großen Anzahl verschiedener Geräte genutzt. Die Unterstützung dieser Protokolle ermöglicht einen relativ unkomplizierten Zugriff auf das Internet.

OBEX: Das *Object Exchange* Protokoll wurde von der Infrared Data Association (IrDA) entwickelt und ermöglicht den spontanen Austausch verschiedener Daten (vCard, vCalendar, ...)

WAP: Das Ziel des *Wireless Application Protocol* ist es, Internetinhalte auf tragbaren Geräten nutzen zu können.

3.4.3 Profiles

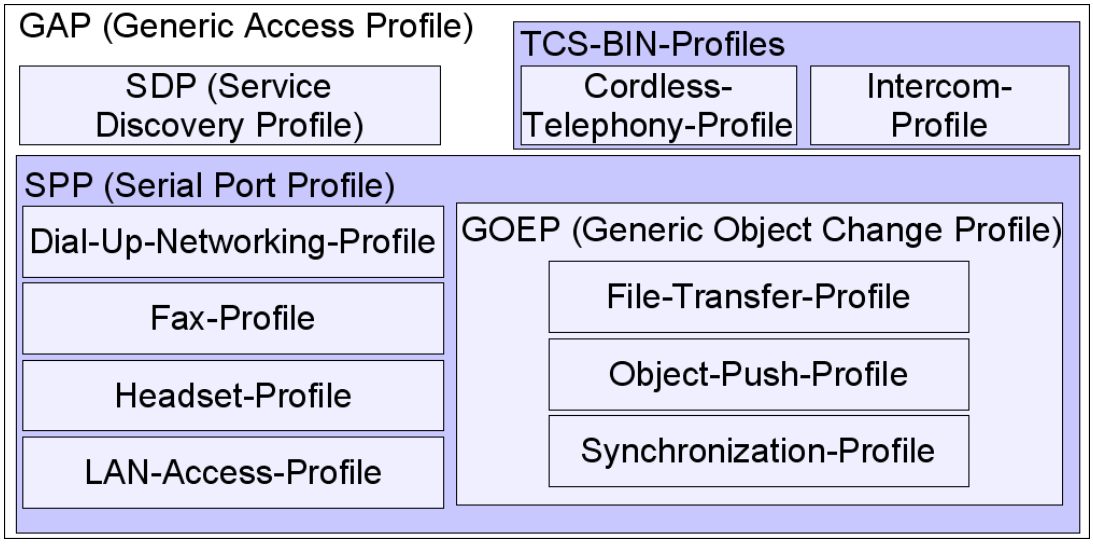

Abbildung 3.2: Bluetooth Profile

Zur Gewährleistung einer genormten Kommunikation zwischen Geräten verschiedener Hersteller wurden von der Bluetooth SIG Nutzungsmodelle entworfen, die jeweils eine bestimmte Aufgabe erfüllen. Diese so genannten *Profile* enthalten Vorschriften, an die sich die Hersteller halten müssen. Es ist genau festgelegt, welche Teile des Bluetooth Protokollstacks genutzt werden und welche Protokolle für die Kommunikation sorgen. Beliebige Geräte mit dem selben Profil sollen somit problemlos miteinander kommunizieren können. Die existierenden Profile sind in Abbildung 3.2 aufgeführt (Ericsson 2002). 2003 hat die SIG 13 neue Profile festgelegt. Aufgrund des Bluetooth-Designs war es möglich, die alten Spezifikationen beizubehalten (Özkilic und Zivadinovic 2003).

Für die Studienarbeit wird das *Headset Profile* benötigt. Dieser Abschnitt geht auf die Eigenheiten dieses Profils ein. Darunter liegen das *generic access profile* (GAP) und das *serial port profile*(SPP). Die vollständige Spezifikation aller Bluetooth 1.1 Profile sind in einem Whitepaper der Bluetooth SIG (Bluetooth SIG 2001) zu finden.

3.4.3.1 Generic Access Profile

Dieses grundlegende Profil wird von jedem Bluetooth Gerät unterstützt. Es ist zuständig für den Verbindungsaufbau, der über die tieferen Schichten des Stacks (Baseband/LC und LMP) gesteuert wird. Die Suche nach anderen Geräten, sowie die Verwaltung verschiedener Sicherheitsstufen wird zusätzlich mithilfe höherer Schichten (L2CAP, RFCOMM und OBEX) realisiert. Das Profil stellt sicher, dass prinzipiell zwei beliebige Bluetooth Geräte eine Verbindung aufbauen können. Sei es nur, um herauszufinden, dass es keine gemeinsam nutzbaren Dienste gibt.

In gekürzter Fassung wird durch das Profil folgendes definiert:

- Das Profil gibt die Erfordernisse von Namen, Werten und Coding-Schemata an, die sich auf Ebene der Benutzerschnittstelle befinden. Auf diese Weise werden inkompatible Implementation der einzelnen Hersteller verhindert.
- Generische Abläufe von Operationen, die nicht Dienst- oder Profil-spezifisch sind, werden beschrieben. Diese können von abgeleiteten Profilen genutzt und erweitert werden.
- Für die Suche nach anderen Geräten, die *discoverable* sind, werden verschiedene Funktionen bereitgestellt. Mit deren Hilfe kann die Identität, der Name und grundlegende Fähigkeiten der anderen Geräte ermittelt werden. Diese Funktionen benötigen keine aufgebaute Verbindung.
- Das Profil definiert den Ablauf des *Bonding*[2] zwischen 2 Bluetooth Geräten.
- Das Profil beschreibt die Standardfunktionen, die genutzt werden können, um eine Verbindung zu anderen Bluetooth Geräten aufzubauen.

3.4.3.2 Serial Port Profile

Dieses Profil ist für die emulierte serielle Verbindung zuständig. Es benutzt RFCOMM, um Daten zwischen 2 Geräten auszutauschen.

Alte Applikationen, die über die serielle Schnittstelle kommunizieren, können das SPP nutzen. Dazu muss ein zusätzliches Programm eine RFCOMM-Verbindung zum Zielgerät herstellen und der Applikation eine virtuelle Schnittstelle bieten. Diese hat die selben Eigenschaften, wie eine echte serielle Schnittstelle. So ist es möglich, das bisher notwendige Kabel durch eine virtuelle Abstraktion auszutauschen, ohne dass bestehender Code geändert werden muss.

3.4.3.3 Headset Profile

Das Headset Profil beschreibt die Erfordernisse für die Nutzung eines Bluetooth Headsets. Es baut auf GAP und SPP auf. Damit andere Geräte das Profil korrekt identifizieren können, wurden Service Records definiert, die das Auffinden des gesuchten Dienstes ermöglichen. Es gibt zwei unterschiedliche Rollen in diesem Profil:

[2] Das Bonding generiert für beide beteiligten Geräte einen Link-Key, mit dem sie sich dann gegenseitig identifizieren können. Dieser Key wird auch verwendet, um den Datenaustausch zu verschlüsseln

Audio Gateway: Dieses Gerät ist dafür zuständig, die Audiodaten zwischen der Bluetooth-Welt und mindestens einer anderen Technologie zu vermitteln. Das Audio Gateway ist eine der Hauptkomponenten, die im Zuge der Studienarbeit benötigt werden. Das konkrete Design wird im Abschnitt 4.3 genauer behandelt.

Headset: Das Headset liefert dem Audio Gateway den Bluetooth-basierten Input und Output.

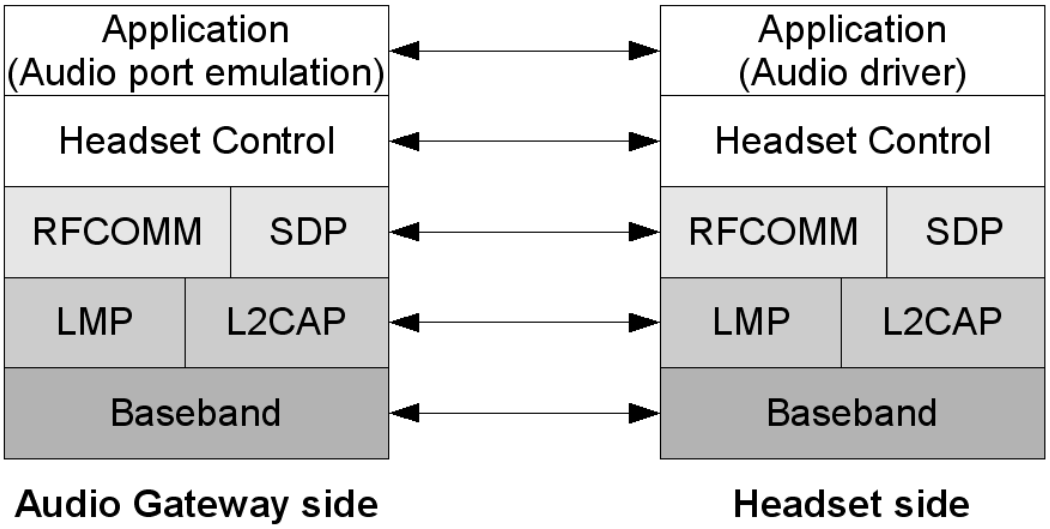

Abbildung 3.3: Protokoll Modell

Die Abbildung 3.3 stellt die Protokolle dar, die von diesem Profil verwendet werden.

Headset Control ist für die eigentliche Headset Steuerung zuständig, die auf AT-Kommandos basiert. Zusätzlich ist es notwendig, dass dieses Programm Zugriff auf einige Befehle im unteren Bereich des Stacks hat. Wichtigstes Beispiel hierfür ist der Aufbau einer SCO-Verbindung. Die *audio port emulation* im Audio Gateway ist der Programmteil, der im Handy oder PC für die Verarbeitung der Audiodaten zuständig ist. Alle grau hinterlegten Protokolle werden bereits im SPP (siehe 3.4.3.2) beschrieben.

Das Headset-Profil sieht vor, dass genau ein Audio Gateway mit genau einem Headset kommuniziert. Es existiert dabei nur eine einzige SCO-Verbindung. Die Daten werden dabei mittels CVSD[3] komprimiert gesendet (64kbit/s, mono). Der Benutzer kann über das Headset nur sehr beschränkt Eingaben tätigen, beispielsweise eine Taste drücken oder die Lautstärke ändern.

Eine sichere Verbindung mit Authentifikation und Verschlüsselung ist nicht zwingend erforderlich. Wenn dies erwünscht ist, kann eine sichere Verbindung über das GAP (3.4.3.1) aufgebaut werden. Der erforderliche PIN-Code wird dann in der Regel über das Audio Gateway eingegeben, da das Headset nur über ein minimales Benutzerinterface verfügt.

[3] Continuous Variable Slope Delta

Ein Verbindungsaufbau findet bei eingehenden oder ausgehenden Anrufen statt. Das Headset muss in der Lage sein, beide Richtungen zu bedienen. Das Audiogateway muss lediglich in der Lage sein, eingehende Verbindungen zu vermitteln. Der Ablauf bei diesen Aktionen wird hier kurz dargestellt.

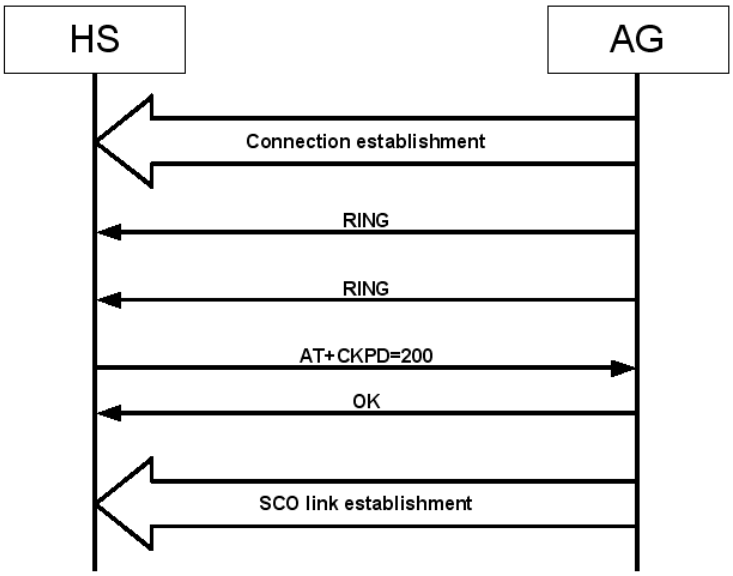

Abbildung 3.4: eingehende Audio-Verbindung

Der erste Fall ist eine Verbindung, die vom Audio Gateway initiiert wird. Sobald ein entsprechendes Ereignis auftritt, baut das Gateway eine ACL-Verbindung zum Headset auf. Sobald die Verbindung steht, kann der Result Code *RING* beliebig oft versendet werden, der beim Headset eine Meldung (meistens akustisch) für den Benutzer auslöst. Die SCO Verbindung kann zu jedem beliebigen Zeitpunkt nach dem Aufbau der ACL-Verbindung gestartet werden. Sobald der Benutzer den eingehenden Anruf annimmt, sendet das Headset den Code *AT+CKPD* an das Gateway. Dieses bestätigt mit *OK* und startet dann die SCO-Verbindung, wenn es nicht bereits vorher geschehen ist.

Die zweite Möglichkeit ist, dass das Headset eine Verbindung startet. Dazu baut es die ACL-Verbindung auf und sendet den Code *AT+CKPD* an das Gateway. Wird eine SCO-Verbindung benötigt, ist in jedem Fall das Audio Gateway dafür zuständig.

Sobald das Headset die Verbindung wieder abbauen soll, sendet es erneut *AT+CKPD* an das Gateway. Generell ist es egal, welche Seite den Abbruch wünscht. Zuständig für die Schliessung von SCO und ACL Verbindung ist immer das Audio Gateway.

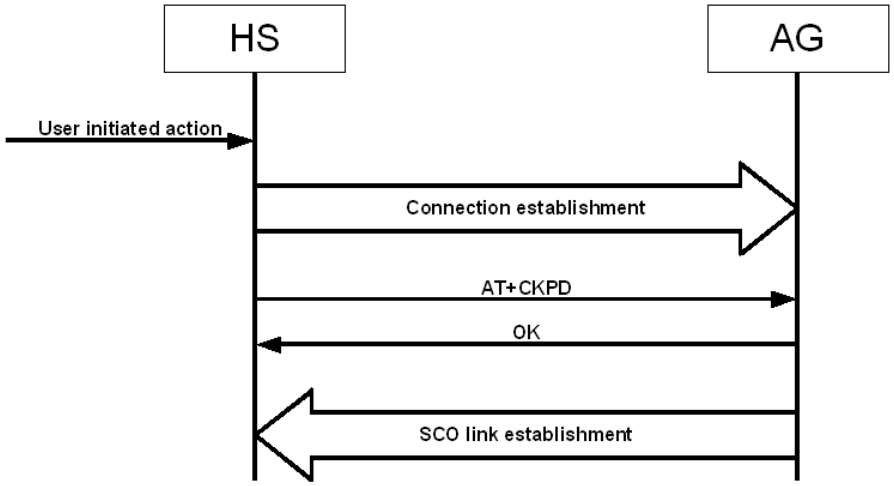

Abbildung 3.5: ausgehende Audio-Verbindung

Weiterhin ist es möglich, über die AT-Kommandos *+VGM* und *+VGS* die Lautstärke von Mikrofon und Lautsprecher in 16 Stufen zu regeln.

3.4.4 Stollmann BlueCode+

Für die Realisierung der Aufgabe stehen generell mehrere Bluetooth-Stacks zur Verfügung. Die Wahl fiel auf die Lösung der Firma Stollmann, da folgende Kriterien erfüllt werden:

- Der Stack arbeitet unter Windows 2000/XP.
- Der Stack unterstützt das Generic Access Profile (GAP). Bei der Suche nach einem geeigneten Headset wird das Service Discovery Protocol (SDP) genutzt. Ausserdem kann das Headset eine sichere Verbindung herstellen, indem der Bonding-Mechanismus genutzt wird. Headset und Audiogateway tauschen hierbei einen *Linkkey* aus, der für eine verschlüsselte Kommunikation genutzt wird.
- Das Headset wird hauptsächlich über eine RFCOMM-Verbindung gesteuert. Das Serial Port Profile wird ebenfalls unterstützt.
- Für die Übertragung der Audiodaten kann eine SCO-Verbindung hergestellt werden. Ausserdem ist es möglich, die Daten auf diesem Kanal zu manipulieren.
- Eine Zusammenarbeit mit Stollmann ermöglicht den Zugriff auf Beispiele und Sourcecode. Weiterhin wurde ein Mitarbeiter der Firma zur fachlichen Betreuung zur Verfügung gestellt.

Der Vollständigkeit halber seien einige alternative Stacks erwähnt:

BlueZ: BlueZ ist ein Stack, der als Open Source Projekt begann und später offiziell in den Linux-Kernel integriert wurde. (Holtmann und Krasnyansky 2004)

AXIS Open-Bluetooth-Stack: AXIS stellt ebenfalls einen offenen Bluetooth Stack zur Verfügung. Auch dieser ist unter Linux einsetzbar. (Axis Communications 2004)

Microsoft Platform SDK: Microsoft bietet für Entwickler ein SDK zur Verfügung, in dem unter anderem auch ein Bluetooth Stack enthalten ist. (Microsoft Corporation 2004)

WIDCOMM: Das kommerzielle „BTW - Bluetooth for Windows“ von Widcomm enthält diverse Programme, Treiber und Stacks, die unter Windows genutzt werden können. (Widcomm Inc. 2004)

Im Folgenden werden die Besonderheiten der BlueFace-API vorgestellt. Dieser Abschnitt orientiert sich an der Dokumentation von Stollmann (Stollmann E+V GmbH 2003).

3.4.4.1 Konzept

BlueFace+ ist eine Bluetooth API, die eine vereinheitlichte Schnittstelle zu Hardwarelösungen verschiedener Hersteller anbietet. Auf BlueFace+ aufsetzende Applikationen sind somit unabhängig von der verwendeten Hardware. Der Verbindungsaufbau läuft somit immer nach dem gleichen Muster ab. Weiterhin werden die verschiedenen Bluetooth-Protokolle und -Profile (3.4.3) unterstützt.

Da BlueFace+ auf verschiedenen Plattformen eingesetzt werden kann, sind die damit entwickelten Applikationen einfach zu portieren. Spätere Erweiterungen der API werden für bestehende Anwendungen transparent gehalten, um die Kompatibilität zu gewährleisten.

Die Nutzung der Bluetooth-Hardware kann vollständig über BlueFace+ abgewickelt werden. Bei Bedarf können sich mehrere Applikationen einen einzelnen Bluetooth Adapter teilen. In Zukunft wird es zudem möglich sein, mehrere Adapter parallel zu betreiben.

3.4.4.2 Message-System

Ein Kernkonzept der API ist das Message-System. Informationen zwischen BlueFace und Applikationen werden über einen asynchronen Mechanismus ausgetauscht. Dies ermöglicht eine Unabhängigkeit vom darunter liegenden Betriebssystem.

Für die Kommunikation zwischen BlueFace und Applikation existiert folgendes Protokoll:

Nachrichten werden generell durch eine passende Antwort bestätigt. Eine Nachricht, die von der Anwendung ausgeht, wird *Request* genannt und die dazu gehörige Antwort von BlueFace+ ist eine *Indication*. In der Gegenrichtung kann BlueFace+ eine *Indication* senden, die von der Anwendung mit einer *Response* beantwortet wird. Alle Nachrichten haben daher eine entsprechende Endung (REQ, CONF, IND, RESP). Eine Antwort ohne vorhergehende Anfrage ist nicht erlaubt und wird von BlueFace+ ignoriert.

Nachrichten können mehrere Parameter besitzen. Diese können elementare Datentypen oder Handles/Referenzen sein.

3.4.4.3 Nachrichtenaustausch

Die Kommunikation mit Applikationen läuft über Funktionsaufrufe und Callbacks. Jede Applikation registriert eine Callback-Funktion BlueFace+. Der Zugriff auf BlueFace+ geschieht über die Funktion *BLUEFACE_PUT_MESSAGE*.

Es gibt 3 grundlegende BlueFace+ Operationen, die eine Anwendung nutzen muss.

blueFaceRegister: Bevor eine Anwendung BlueFace+ nutzen kann, muss sie sich registrieren. Dabei werden die Callback-Funktion und der Kontext angegeben. Zusätzlich kann der zu verwendende Controller gewählt werden.

blueFacePutMessage: Diese Funktion wird von der Applikation verwendet, um BlueFace+ eine Nachricht zu senden. Als zusätzlicher Parameter wird das Applikations-Handle übergeben, damit der Kontext identifizierbar ist.

blueFaceRelease: Zur Beendigung der Kommunikation ruft die Applikation blueFaceRelease auf. Damit gibt BlueFace+ alle Ressourcen frei, die mit dieser Anwendung zusammenhängen. Vor dem Release sollte die Applikation sämtliche bestehenden Bluetooth-Verbindungen beenden, da das Verhalten on BlueFace+ ansonsten nicht definiert ist.

Zur Identifikation von logischen Verbindungen erstellt BlueFace+ eindeutige Link Handles. Diese werden bei allen Nachrichten verwendet, die die zugehörige Verbindung betreffen. Parallel dazu kann eine Applikation einen eindeutigen Link Context erstellen, der von BlueFace+ genutzt werden soll.

3.5 Windows Treibermodell (WDM)

Für die Realisierung wird ein einfacher Audiotreiber benötigt. Dieser stellt anderen Anwendungen einen standardisierten Zugriff auf die Audiostreams vom Headset zur Verfügung. Microsoft stellt für die Entwicklung von Windows-Treibern ein *Driver Development Kit* zur Verfügung. Es enthält eine große Anzahl an Beispielen für die unterschiedlichsten Treiberarten. Diese Beispiele sind relativ klein gehalten und verdeutlichen meist einen bestimmten Aspekt der Treiberprogrammierung. Sie können als Template für neue Treiber herangezogen und angepasst werden.

WDM konforme Treiber sollen unter jedem Betriebssystem von Microsoft (ab Windows 2000) lauffähig sein. Die korrekte Funktion unter Windows 98/Me ist unter Umständen möglich aber nicht garantiert.

Im Folgenden wird ein Überblick über die Treiberarchitektur von Windows NT/2000 gegeben. Der Inhalt dieses Abschnittes orientiert sich eng an einem Buch (Baker und Lozano 2000), das sich detailliert mit dieser Thematik auseinander setzt.

3.5.1 Überblick über das Windows NT-Design

Gerätetreiber arbeiten sehr eng mit dem darunter liegenden System zusammen. Daher werden hier Architektur und Design von Windows 2000 kurz dargestellt. Art Baker und Jerry Lozano beschreiben die grundlegenden Ziele bei der Entwicklung von NT:

Kompatibilität: Das Betriebssystem soll möglichst viel der existierende Hard- und Software unterstützen.

Robustheit und Verlässlichkeit: Das Betriebssystem soll vor versehentlichem oder beabsichtigtem Missbrauch geschützt sein. Benutzerprogramme dürfen die Stabilität des Systems nicht gefährden.

Portabilität: Das Betriebssystem soll auf möglichst vielen der existierenden und zukünftig entwickelten Plattformen lauffähig sein.

Erweiterbarkeit: Da sich die Marktanforderungen mit der Zeit ändern, muss das System auf einfache Weise anpassbar sein. Neue Funktionen und Hardware müssen dabei mit minimalem Aufwand am bereits existierenden Code unterstützt werden.

Leistung: Das Betriebssystem soll auf jeder gegebenen Hardwareplattform eine angemessene Leistung erbringen.

Dabei ist zu bedenken, dass es in der Realität nicht möglich ist, alle diese Ziele gleichermaßen zu erreichen. Zum Teil muss an einer Stelle ein Kompromiss eingegangen werden, um an anderer Stelle weiter zukommen.

Die beiden ersten Ziele *Kompatibilität* und *Robustheit / Verlässlichkeit* werden mit einer Client/Server-Architektur angegangen. Eine Benutzeranwendung arbeitet als Client eines oder mehrerer OS Services.

Dieser Ansatz stellt gewöhnlichen Anwendungen den *user mode* zur Verfügung. In diesem werden nur Funktionen erlaubt, die die Funktion des gesamten Systems nicht gefährden. Es wird beispielsweise verhindert, dass Applikationen unkontrolliert auf fremde Speicherbereiche zugreifen können. Ein direkter Zugriff auf die Hardware ist ebenfalls nicht erlaubt.

Der Code des Betriebssystems läuft im *kernel mode*. Dieser Code hat uneingeschränkten Zugriff auf sämtliche CPU-Befehle und I/O-Operationen. Ausserdem besteht Zugriff auf den gesamten Speicher.

Die Gerätetreiber laufen im kernel mode. Deshalb kann ein schlecht programmierter Treiber die Stabilität des gesamten Systems gefährden.

Die *Portabilität* wird durch eine Schichtarchitektur erreicht. Das *Hardware Abstraction Layer (HAL)* trennt die Hardware von System- und Treibercode. Damit ist es nicht nötig, einen Treiber für jede neue Plattform neu zu schreiben. Damit dies funktioniert, arbeitet der Code mit Funktionen aus dem HAL, um Hardwareadressen und Busse anzusteuern.

Das Ziel der *Erweiterbarkeit* wird ebenfalls mithilfe des Schichtenmodells erreicht. Die beiden Schichten *Kernel* und *Executive Components* sind voneinander getrennt. Dabei übernimmt der Kernel hauptsächlich die Aufgabe, das Scheduling der einzelnen Tasks vorzunehmen. Die Verwaltung von Speicher, Prozessen, Sicherheit und I/O Management werden von den Executive Components übernommen, die modular aufgebaut sind.

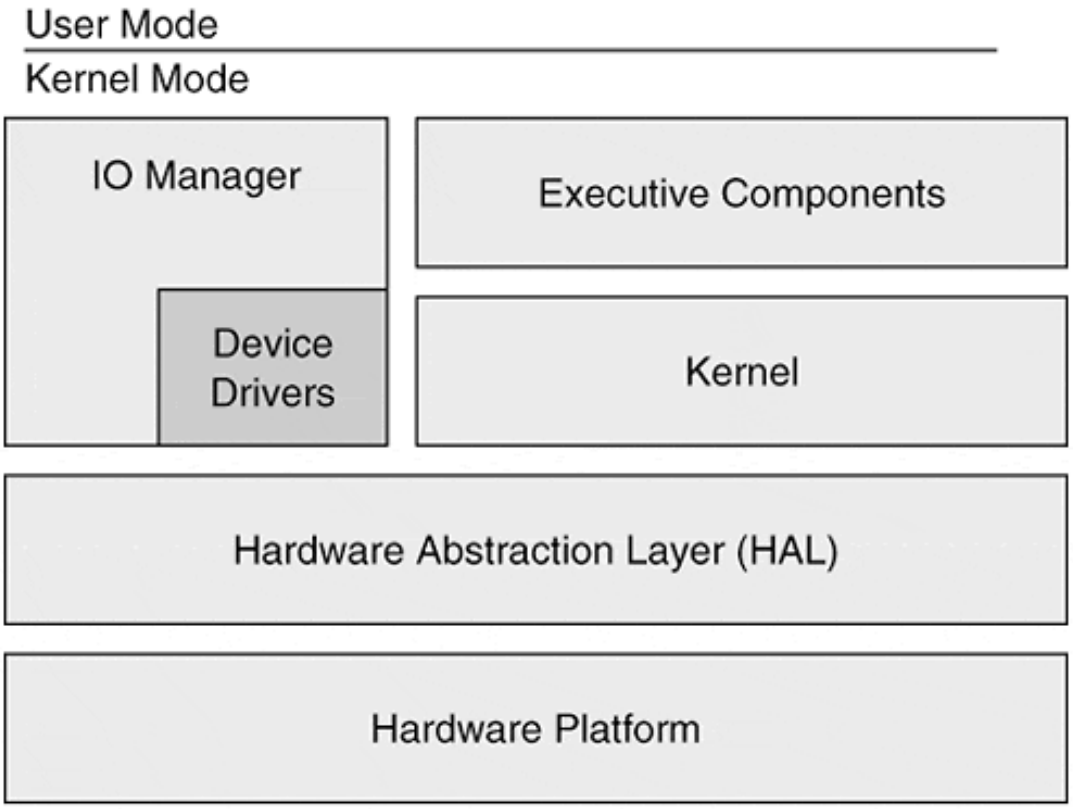

Abbildung 3.6: Die Schichtarchitektur von Windows 2000 (Baker und Lozano 2000)

Das letzte Hauptziel ist eine angemessene *Leistung* des Systems. Die Schichtarchitektur ist generell langsamer als ein monolithischer Ansatz. Deshalb wird die Kommunikation zwischen den Schichten möglichst effizient gestaltet. Viele Aufrufe im HAL sind als kurze Makros realisiert. Weiterhin wurden möglichst viele Tasks wie möglich parallelisiert. Als Folge der Geschwindigkeitsoptimierungen dürfen Gerätetreiber ihre Ausführung nicht blockieren. Wenn ein Treiber nicht in der Lage ist, eine Anfrage sofort zu verarbeiten, wird diese in eine Warteschlange eingefügt.

Der Audiotreiber wird folglich im kernel-mode parallel zum Systemkernel laufen. Er wird direkt auf dem HAL arbeiten können, was aber bei einem virtuellen Gerätetreiber nicht notwendig ist.

3.5.2 Treiberklassen

Im Gegensatz zu den früheren monolithischen Gerätetreibern bestehen Windows 2000-Treiber aus mehreren Schichten. Dadurch besteht die Möglichkeit, Code aus bestehenden Schichten wiederzuverwenden. Wichtig hierbei ist, den Unterschied zwischen den verschiedenen Treiber-

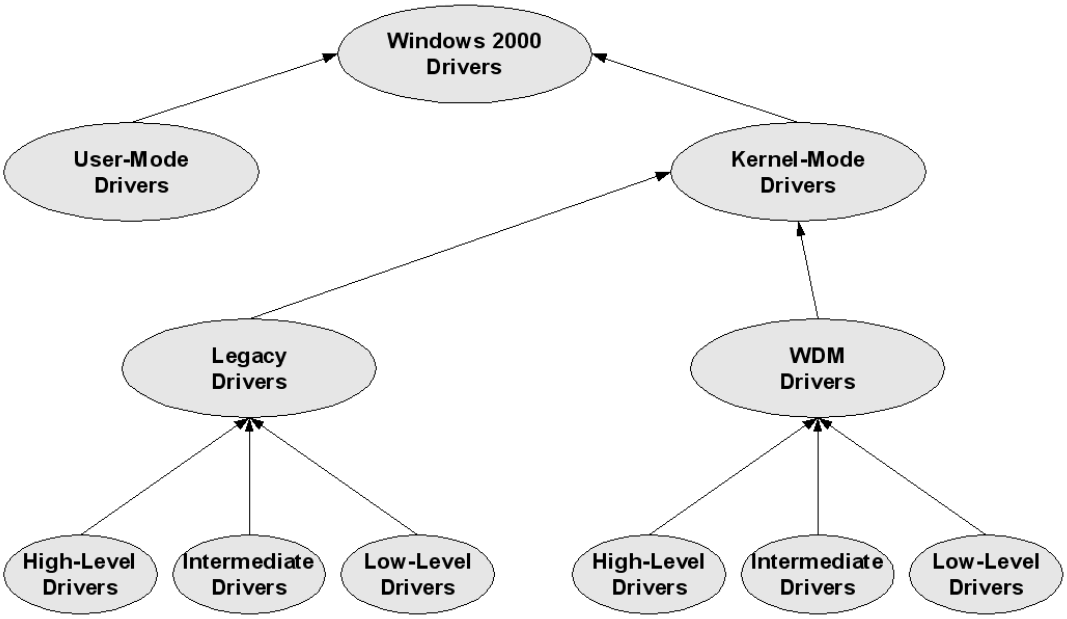

Abbildung 3.7: Treiberklassen (Baker und Lozano 2000)

klassen zu kennen.

Windows 2000 kennt grundsätzlich 2 verschiedene Arten von Treibern, *user-mode* und *kernel-mode*. Erstere arbeiten im user mode, haben also keinen direkten Zugriff auf Hardware. Sie können beispielsweise für virtuelle Hardware verwendet werden. Dadurch sind sie auf die Unterstützung durch kernel-mode Treiber angewiesen. Diese arbeiten auf Systemebene und können direkt mit der Hardware kommunizieren. In diesem Modus arbeiten alle „echten“ Treiber, die sich wiederum in 2 Kategorien unterteilen lassen.

legacy: Legacy-Treiber sind meist ältere Treiber. Sie verrichten den Großteil der nötigen Arbeit selber und sind in der Regel auf ein bestimmtes Betriebssystem angewiesen.

Windows Driver Model: WDM-Treiber nutzen bestimmte Schnittstellen, die bei unterschiedlichen Versionen des Betriebssystems normiert sind. Dadurch ist es möglich, den Code ohne nennenswerte Änderungen auf die verschiedenen Systeme zu portieren. Meistens reicht es aus, den Code für das entsprechende Zielsystem neu zu kompilieren.

Die unterste Ebene unterscheidet zwischen *high-level*, *intermediate*, und *low-level* Treibern.

High-level Treiber nutzen die anderen beiden Typen, um ihre Aufgaben zu erfüllen. Sie stellen eine virtuelle Abstraktion dar, die genutzt werden kann, um Zugriffe auf vorhandene Treiber zu vereinfachen oder das Interface anzupassen.

Intermediate Treiber arbeiten als mittlere Schicht. Beispiele hierfür sind *class driver*, *mini driver* und *filter driver*. Eine Anfrage von einem höher liegenden Treiber wird in eine oder mehrere Anfragen an darunter liegende Treiber weitergeleitet und gegebenenfalls verändert. Die ande-

ren beteiligten Treiber merken dabei nicht, in welcher Weise die ursprüngliche Anfrage verändert wurde. Die class driver werden eingesetzt, um Gemeinsamkeiten zwischen ähnlichen Treibern in einer generischen Klasse zusammenzufassen. Herstellerspezifische Eigenschaften können dann als mini driver erstellt werden, der mit der generischen Komponente interagiert. Filter driver fangen Daten ab und sind in der Lage, sie vor dem Weiterleiten zu modifizieren. Das WDM bietet zudem die Möglichkeit, *Functional Drivers* zu erstellen. Sie sind entweder ein class oder mini driver und stellen eine Schnittstelle zwischen einer abstrakten I/O-Anfrage und dem hardwarenahen Code dar.

Low-Level Treiber arbeiten sehr nahe an der Hardware. Sie kommunizieren beispielsweise direkt oder über das HAL mit einem Bussystem. Im WDM existiert der *Physical Driver*, der mit einem oder mehreren Functional Driver interagiert.

Für die Lösung der Aufgabe wird ein WDM-Treiber aus einem DDK-Beispiel erstellt. Da dieser die Audio-Hardware emuliert, muss dies ein low-level-Treiber sein.

3.5.3 I/O-Verarbeitung

Der Audiotreiber wird keine Hardware direkt ansprechen. Daher muss er seine Daten auf einem anderen Weg beziehen. Der Standardmechanismus für die I/O-Verarbeitung bei Treibern sind IRP-Messages. Mithilfe dieser Messages wird es möglich, über ein externes Programm Daten aus einer beliebigen Quelle zu liefern.

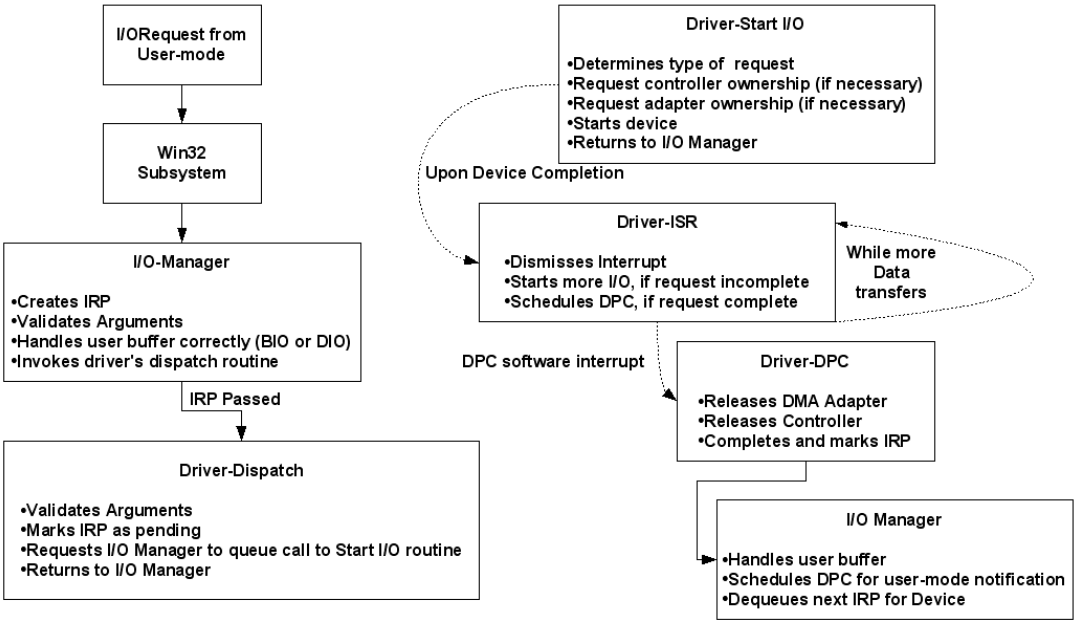

Abbildung 3.8: I/O-Verarbeitung (Baker und Lozano 2000)

I/O Anfragen laufen nach einem bestimmten Schema ab. Im Folgenden wird der Weg einer Anfrage an eine Hardwarekomponente beschrieben. Bei der Programmierung eines virtuellen

Treibers müssen die hardwarenahen Operationen in diesem Ablauf ersetzt werden. Dies betrifft insbesondere die DMA-Zugriffe. An dieser Stelle geschieht lediglich der Zugriff auf den Arbeitsspeicher.

Eine I/O Anfrage durchläuft mehrere Schritte:

Vorverarbeitung durch den I/O Manager: Die Anfrage wird unabhängig vom angesprochenen Gerät vorbereitet und geprüft.

1. Die Anfrage wird in einen system service call umgesetzt. Das System wechselt in den kernel-mode und startet den I/O Manager.
2. Der I/O Manager erzeugt ein *I/O Request Packet*(IRP). Dies ist eine Struktur, die den Typ der Anfrage und weitere benötigte Informationen enthält.
3. Der I/O Manager überprüft die Parameter auf Gültigkeit. Dies beinhaltet einen Check des Filehandles, der Zugriffsrechte, der Bufferadressen und ob das angesprochene Device die gewünschte Operation unterstützt.
4. Ist *buffered I/O* aktiviert, stellt der I/O Manager einen festen Speicherbereich bereit, in dem die Daten für das user mode Programm manipuliert werden können. Bei *direct I/O* wird der gesamte Buffer des Programms für die Dauer des Vorgangs im Arbeitsspeicher gehalten.
5. Der I/O Manager ruft die entsprechende driver dispatch Routine auf.

Vorverarbeitung durch den Gerätetreiber: Jeder Treiber besitzt eine Tabelle, in der alle unterstützen Funktionen aufgelistet sind. Der I/O Manager nutzt diese, um die passende Routine aufzurufen.

1. Die Routine führt weitere Gültigkeitsprüfungen der Parameter durch, die vom zugehörigen Gerät abhängig sind.
2. Ist keine weitere Aktion des Treibers notwendig, wird die IRP Anfrage als *complete* markiert und an den I/O Manager zurückgegeben.
3. Im Normalfall ist eine weitere Behandlung notwendig und das IRP Paket wird als *pending* markiert. Der I/O Manager ist jetzt verpflichtet, die IRP Nachricht in eine Warteschlange zu stellen und bei der nächsten Gelegenheit an die Start I/O Routine des Treibers zu leiten.

Verarbeitung durch das Gerät: Die eigentliche Verarbeitung der Befehle wird durch die *Start I/O* und die *Interrupt Service Routine* des Treibers vorgenommen.

Start I/O Soll das Gerät gestartet werden, prüft der I/O Manager zuerst, ob das Gerät bereits beschäftigt ist. Dazu wird die Warteschlange auf andere IRPs überprüft. Ist das Gerät frei, wird die Start I/O Routine sofort aufgerufen.

1. Die Art der IRP Anfrage wird ermittelt und nötige Vorbereitungen für diese Operation durchgeführt.

2. Falls nötig, wird der Hardware Controller für exklusiven Zugriff konfiguriert
3. Bei DMA Operationen wird die benötigte DMA Hardware für exklusiven Zugriff konfiguriert
4. Mithilfe einer *SynchCriticalSection* Routine wird das Gerät in einem konsistenten Zustand gehalten.
5. Die Kontrolle wird wieder an den I/O Manager gegeben.

ISR Sobald ein entsprechender Interrupt auftritt, wird die ISR des Treibers aufgerufen.

1. Prüfung, ob der Interrupt erwartet wurde
2. Wird gerade ein I/O Vorgang durchgeführt, wird der nächste Interrupt abgewartet
3. Bei einer laufenden DMA Operation wird ein DPC erzeugt, um die DMA Hardware auf die nächste Übertragung vorzubereiten.
4. Bei Auftreten eines Fehlers oder bei Erfolg wird über ein DPC die Nachverarbeitung begonnen.

Nachverarbeitung durch den Gerätetreiber: Nach der eigentlichen Verarbeitung führt der Treiber die nötigen Aufräumarbeiten durch.

1. Müssen weitere Daten über DMA gesendet werden, wird die Hardware erneut vorbereitet und auf einen weiteren Interrupt gewartet. Das IRP Paket bleibt *pending*.
2. Bei einem Fehler oder Timeout kann der Vorgang entweder erneut durchgeführt oder abgebrochen werden.
3. Alle DMA und Controller Resourcen werden wieder freigegeben.
4. Die Ergebnisse des Transfers und der aktuelle Status werden im IRP gespeichert.
5. Das IRP wird als *complete* gekennzeichnet und dem I/O Manager übergeben. Dieser kann dann einen weiteren IRP aus der Warteschlange verarbeiten.

Nachverarbeitung durch den I/O Manager: Sobald die Anfrage als complete markiert ist, führt der I/O Manager die restliche Säuberung durch.

1. Bei einem *buffered I/O* Schreibvorgang wird der angelegte Pool Buffer wieder freigegeben.
2. Bei einem *direct I/O* Vorgang werden die gesperrten Speicherseiten wieder freigegeben.
3. Es wird ein kernel-mode *asynchronous procedure call* (APC) für den ursprünglich aufrufenden Thread vorbereitet. Dieser APC führt den I/O Manager Code im Kontext dieses Threads aus.
4. Der APC kopiert Status und Transferinformationen in den User Bereich zurück.
5. Bei einem *buffered I/O* Lesevorgang wird der Pool Buffer in den Userbereich kopiert und danach freigegeben.

6. Wenn die ursprüngliche Anfrage eine „Win32 overlapped operation“ war, wird der Status des dazugehörigen Events / File Objekts auf signaled gesetzt.
7. Wenn die ursprüngliche Anfrage eine „completion routine“ enthielt (z.B. ReadFileEx), wird ein user-mode APC generiert, der diese ausführt.

3.5.4 Aufbau eines Kernel-Mode Treibers

Treiber unterscheiden sich stark von gewöhnlichen Applikationen. Sie bestehen hauptsächlich aus einer Sammlung von Routinen, die vom Betriebssystem (in der Regel der I/O Manager) aufgerufen werden. So lange kein Aufruf vom System stattfindet, verrichtet ein Treiber keine Arbeit.

Abhängig von der Art des Treibers, kann der I/O Manager in folgenden Situationen eine Treiberroutine starten:

- der Treiber wird geladen (DriverEntry, Reinitialize)
- der Treiber wird entfernt oder das System fährt herunter (Unload, Shutdown)
- ein Gerät wird angeschlossen oder entfernt
- ein user-mode Programm führt einen I/O Systemaufruf durch (CreateFile, CloseHandle, ReadFile, WriteFile, DeviceIoControl)
- eine benötigte Hardwareressource wird frei (ControllerControl, AdapterControl, SynchCritSection)
- in verschiedenen Situationen, wenn das Gerät arbeitet (Timer, ...)

3.5.5 Wichtige Datenstrukturen

Windows 2000 nutzt Techniken der Objekt-Orientierung. Es definiert Datenstrukturen und Sätze von dazugehörigen Operationen, um die Inhalte zu manipulieren. Von Modulen, die diese Strukturen nutzen, wird erwartet, dass die entsprechenden Funktionen verwendet werden.

Der zu erstellende Audiotreiber besteht aus einem Driver Object und mindestens einem Device Object. Von außen kann mithilfe von IRP-Anfragen auf den Treiber zugegriffen werden.

3.5.5.1 I/O Request Packets (IRPs)

Die meisten I/O Operationen in Windows 2000 werden über Pakete abgewickelt.Die Transaktionen werden mit Datenstrukturen, genannt *I/O Request Packets*, gesteuert. Ein einfaches Modell ohne mehrere Treiber-Schichten des IRP Handlings sieht folgendermaßen aus:

1. Für jede I/O Anfrage aus dem user mode generiert der I/O Manager Speicher für einen IRP. Die Zuordnung zu einer bestimmten Treiberfunktion geschieht über ein Filehandle und die Funktion, die im user mode gestartet wurde.

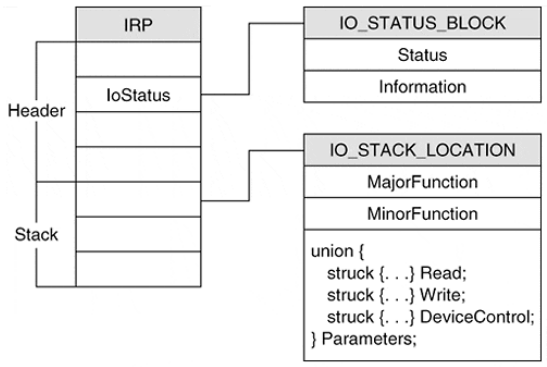

Abbildung 3.9: I/O Request Packets (Baker und Lozano 2000)

2. Die aufgerufene Dispatch Routine des Treibers überprüft die Übergabeparameter auf ihre Gültigkeit und sendet den IRP weiter an die Start I/O Routine.

3. Die Start I/O Routine nutzt die Informationen aus dem IRP, um eine Operation auf dem dazugehörigen Gerät durchzuführen.

4. Nach Beendigung der Operation wird der finale Statuscode in den IRP geschrieben und dieser an den I/O Manager zurückgegeben.

5. Der I/O Manager nutzt die Informationen aus dem IRP, um die Anfrage fertig zustellen. Das Ergebnis wird an die user mode Applikation übergeben.

IRPs sind Strukturen mit variabler Größe. Der erste Teil ist der Header, in dem generelle Status-Informationen enthalten sind. Der Rest besteht aus einer oder mehreren *Stack Locations*, abhängig von der Anzahl der Schichten, die ein Treiber besitzt. Diese enthalten den Funktionscode und die Parameter eines I/O Requests. Das Feld *MajorFunction* enthält die ID der angeforderten Funktion. Mit ihrer Hilfe kann die folgende Parameter-Struktur (Read, Write, DeviceIoControl, ...) korrekt interpretiert werden. Der genaue Ablauf eines IRP-Requests ist in Abschnitt 3.5.3 nachzulesen.

3.5.5.2 Driver Objects

Die einzige Treiberroutine, die direkt zugänglich ist, ist *DriverEntry*. Andere Treiberfunktionen kann der I/O Manager über das *Driver Object* erreichen, das mit einem Gerät verbunden ist. Dieses Objekt ist im Grunde eine Sammlung von Pointern auf verschiedene Funktionen. Die Existenz des Driver Objects läuft folgendermaßen:

1. Der I/O Manager erstellt ein Treiberobjekt, sobald ein Treiber geladen wird. Tritt bei der Initialisierung ein Fehler auf, wird das Objekt wieder gelöscht.

2. Bei der Initialisierung schreibt die Driver Entry Routine alle Pointer auf andere Treiberfunktionen in das Objekt.

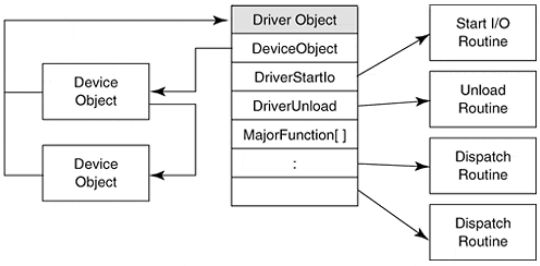

Abbildung 3.10: Driver Object (Baker und Lozano 2000)

3. Sobald ein IRP zu einem bestimmten Device geschickt wird„ benutzt der I/O Manager das Driver Objekt, um die Start I/O Routine des Treibers zu finden.

4. Wird der Treiber entladen, sucht der I/O Manager eine Unload Routine über das Objekt. Sobald die Unload Routine beendet wurde, wird das Objekt gelöscht.

Jeder Treiber besitzt ein eindeutig zugeordnetes Treiberobjekt. Das Objekt enthält einen Pointer auf eine Linked List der Devices, die zu dem Treiber gehören. Eine Unload Routine kann diese Liste verwenden, um bei Bedarf Devices zu finden, die gelöscht werden müssen.

3.5.5.3 Device Objects

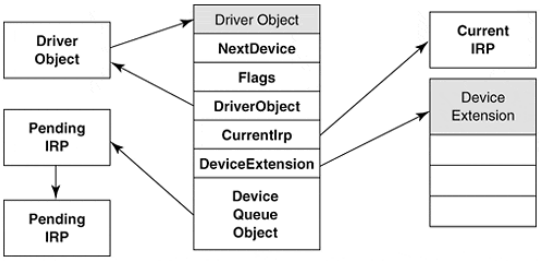

Abbildung 3.11: Device Object (Baker und Lozano 2000)

I/O Manager und Treiber müssen jederzeit informiert sein, in welchem Zustand die zugeordneten Devices sind. *Device Objects* enthalten Informationen über deren Charakteristiken und den aktuellen Zustand. Jedes virtuelle, logische und physikalische Gerät besitzt eines dieser Objekte. Der Lebenszyklus wird hier kurz dargestellt:

1. Die *Driver Entry* Routine erstellt für jedes Gerät ein Device Objekt. Bei WDM Treibern geschieht dies in der *AddDevice* Routine.

2. Der I/O Manager nutzt einen Pointer im Device Objekt, um das übergeordnete Driver Objekt zu finden. Dort stehen die Routinen, die für I/O verwendet werden. Weiterhin verwaltet er die Warteschlange des Devices, in der die IRPs enthalten sind, die noch abgearbeitet werden müssen.

3. Verschiedene Treiberroutinen nutzen das Device Objekt, um die dazugehörige *Device Extension* zu ermitteln. Diese enthält gerätespezifische Informationen.

4. Die Unload Routine des Treibers löscht das Device Objekt, wenn der Treiber entladen wird. Dabei wird auch die korrespondierende Device Extension gelöscht. WDM Treiber übernehmen diese Aufgabe beim Aufruf der Funktion RemoveDevice.

Die Device Objekte werden nicht nur vom low-level Gerätetreiber verwendet. Auch darüber liegende Schichten können darauf zugreifen.

Ein Device Objekt enthält viele Daten, die nur für den I/O Manager bestimmt sind. Der Treiber selbst soll nur auf einen kleine Teil der Daten zugreifen.

Die Device Extension ist ein Speicherbereich, im dem der Programmierer individuelle Daten ablegen kann, die er für das zugehörige Device benötigt. Größe und Inhalt sind frei definierbar. Typischerweise enthält die Extension folgende Variablen:

- Einen Pointer auf das Device Objekt
- Den Zustand des Devices oder Kontextinformationen
- Einen Pointer zu dem Interrupt Objekt und ein „Interrupt-Expected“ Flag
- Einen Pointer auf ein Controller Objekt
- Einen Pointer zu einem Adapter Objekt und „Mapping Register“

Da die Extension frei definierbar ist, muss die Struktur in einem Headerfile definiert werden.

3.5.6 Kernelstreaming

Für die Erstellung von Audio und Videotreibern enthält das WDM eine spezielle *Kernel Streaming* Architektur (Microsoft Corporation 2003). Das KS baut auf dem vorhandenen WDM auf und soll typische Probleme beim Echtzeit-Streaming von Daten mindern.

Das KS modelliert Treiber als Filter, durch die Streams geleitet werden können. Diese Filter besitzen 2 verschiedene Arten von *Pins*. Es gibt Input-Pins, die eine Stream aufnehmen und Output-Pins, die einen Stream ausgeben können. Mehrere dieser Filter können über die Pins miteinander verbunden werden. Der entstehende *Filter-Graph* transformiert eingehende Streams in mehreren Schritten zu einem Ausgabestream. Modifikationen am diesem Stream können relativ einfach vorgenommen werden, indem weitere Filter an den Graphen angefügt werden.

Beispielsweise ist es möglich, den Stream auf relativ einfache Weise zu komprimieren. Dazu wird an den letzten Filter ein weiterer angehängt, der diese Aufgabe übernimmt. Der bisherige Graph kann unverändert übernommen werden.

Audiotreiber sind eine Abstraktion dieser Filter. Sie besitzen Input-Pins, die einen bestehenden Stream annehmen und an die eigentliche Hardware weiterleiten. Ein Output-Pin leitet einen Stream an das System weiter, der von der Hardware (beispielsweise über ein Mikrofon) erzeugt wurde.

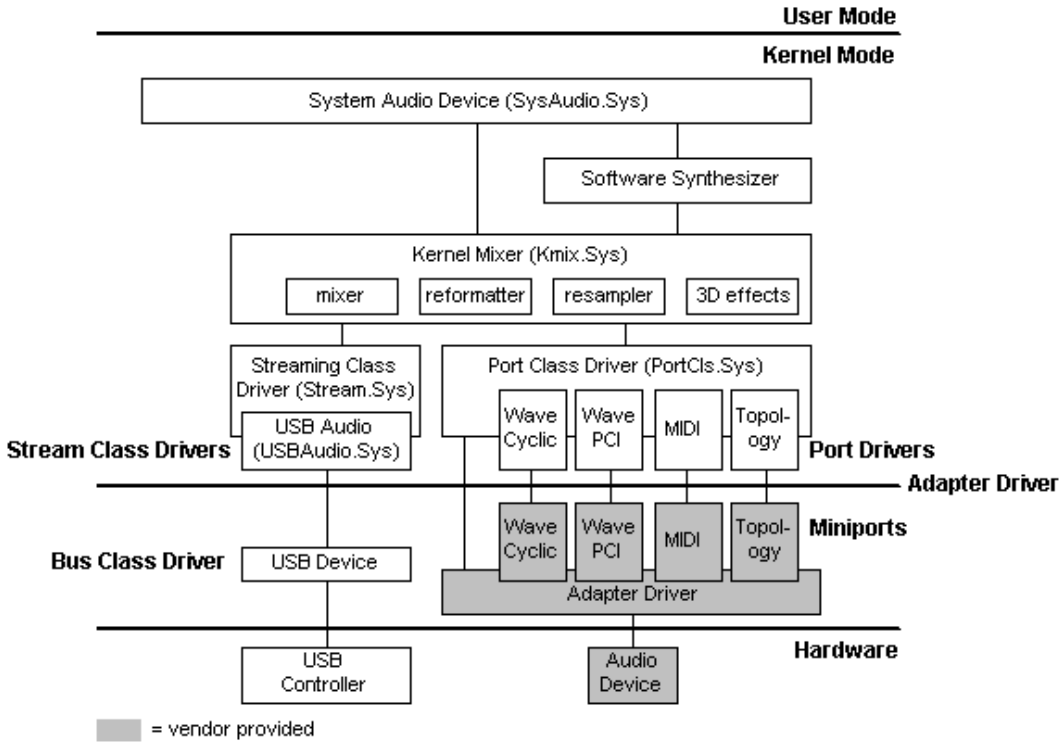

Abbildung 3.12: Kernel Streaming - Aufbau (Microsoft Corporation 2003)

3.5.7 Audio Port Class/Miniport Driver Model

Soundkarten, die einen Bus wie PCI oder ISA benutzen, können das *Port class/miniport driver model* nutzen. Windows besitzt einen generischen Port-Treiber, der direkt mit dem Kernel kommuniziert. Ein herstellerspezifischer Miniport, der eine bestimmte Hardware unterstützt, kann die meisten benötigten Funktionen über den Port-Treiber aufrufen. Ein direkter Kontakt des eigenen Treibers zum Kernel ist somit in der Regel nicht notwendig.

Das Treibermodell besteht aus 4 Komponenten:

Port Driver: Ein Port Driver stellt einen WDM Streaming Filter dar, der von einem Miniport Driver genutzt wird. Er ist in einem Port Class Driver enthalten. Der Port Driver ist von der Audiohardware unabhängig und bildet die Schnittstelle zwischen Miniport und Kernel.

Port Class Driver: Der Port Class Driver koordiniert die Funktionen mehrerer Port Driver und bildet ein vollständiges *Device*, das mit der tatsächlichen Audiohardware korrespondiert. Weiterhin werden verschiedene Funktionen für die Adapter- und Miniport Driver bereitgestellt.

Miniport Driver: Ein Miniport Driver implementiert ein funktionsspezifisches Interface für eine einzelne Funktion (Wave, MIDI, ...) eines bestimmten Adapters, wie zum Beispiel einer Soundkarte. Mehrere Miniport Driver werden in einem Adapter Driver zusammengefasst.

Es gibt mehrere Arten von Miniport Driver Interfaces:

- WaveCyclic
- Wave PCI
- MIDI
- DirectMusic
- Topology

Adapter Driver: Ein Adapter Driver dient als Behälter für alle Miniports, die zu einem Adapter gehören. Er wird vom Betriebssystem als einzelner Treiber behandelt und ist in einer .SYS Datei untergebracht. Zusätzlich zu den Miniports besitzt er Code, der für die Initialisierung der Hardware notwendig ist.

4 Entwurf

Dieses Kapitel beschäftigt sich mit dem geplanten Lösungsweg. Es werden kurz die verschiedenen Ausgangsprogramme und ihre bisherige Funktion vorgestellt. Das Design der daraus entworfenen Teilkomponenten und deren Zusammenspiel wird detailliert beschrieben.

Der Entwurf wurde bewusst einfach gehalten, da der Zeitrahmen einer Studienarbeit relativ eng gesetzt ist. Dieses Modell verarbeitet die Streams nicht konsequent in Echtzeit. Stattdessen wird ein Teil im user-mode behandelt, um die Implementierung zu vereinfachen. Das Hauptziel hierbei ist die Überpüfung, ob das Konzept generell funktioniert.

Ein deutlich effizienteres Design wird in Kapitel 5.3 vorgestellt.

4.1 Gesamtdesign

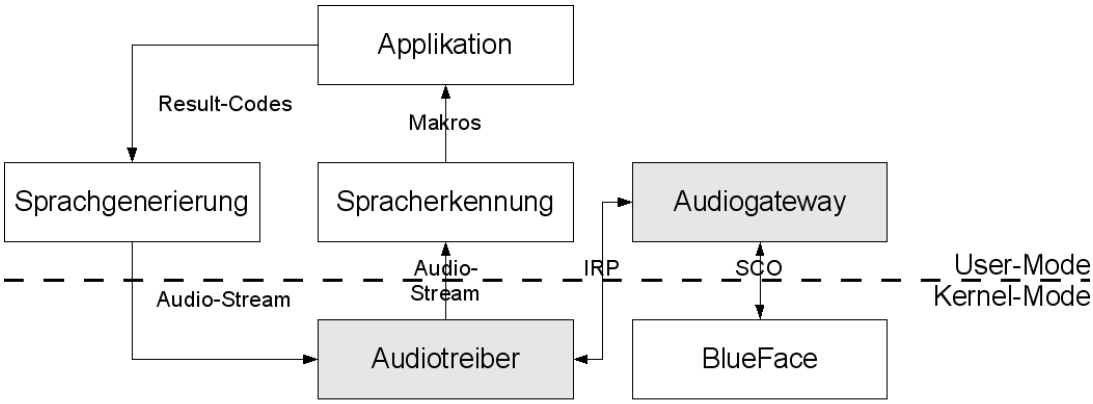

Abbildung 4.1: Gesamtdesign

Die Aufgabe kann durch die Zusammenarbeit mehrerer Teilkomponenten gelöst werden. Hierfür werden mehrere Programme miteinander verkettet 4.1. Die grau hinterlegten Komponenten (Audiogateway und virtueller Audiotreiber) sind Teil der Studienarbeit. Die anderen Komponenten sind bereits vorhanden. BlueFace dient der Kommunikation mit dem Headset. Die Verarbeitung der Sprache geschieht über eine externe Spracherkennung/-Synthese.

Der Bluetooth-Stack wird über das das im user-mode laufende Audiogateway gesteuert. Es ist dafür zuständig, die Verbindung zum Headset aufzubauen und die Verbindungsdaten zu verwalten. Ein Anwender aktiviert hiermit sein Headset und kann ab diesem Zeitpunkt in sein Mikro sprechen.

4.1.1 Weg der Audiodaten

Die Datenpakete aus dem Bluetooth-Stack werden vom Audiogateway empfangen. Sie werden dann als Stream an den Audiotreiber im kernel-mode weitergeleitet. Dieser sorgt dafür, dass der Stream als Mic-In zum Kernel-Mixer gelangt. Eine Spracherkennung, die den Mic-In Kanal des Treibers als Eingang nutzt, kann nun wie bei einem normalen Mikrofon die Audiodaten verarbeiten. Mithilfe eines geeigneten Vokabulars oder durch Makros ist es nun möglich, einen bestimmten Sprachbefehl in eine Zeichenkette oder eine Tastenkombination umzusetzen. In dieser Form des vorher gesprochenen Befehls ist es ein Leichtes, eine Anwendung entsprechend reagieren zu lassen.

Sobald die Anwendung die gewünschte Aktion ausgeführt hat oder dabei ein Fehler aufgetreten ist, kann ein Feedback der Applikation an den Benutzer erfolgen. Dazu wird ein Sprachgenerator benötigt, der eine Meldung in einen Audiostream erzeugt. Entweder es wird ein Programm verwendet, das beliebige synthetische Sätze erzeugt oder man greift auf einen Satz vorgefertigter Audiofiles zurück. Die so gewonnenen Audiodaten werden auf einem Lautsprecher ausgegeben. Es macht Sinn, diese Rückkopplung auf dem selben Weg zurückzuschicken, den der gesprochene Befehl bereits hinter sich hat. Der Audiotreiber sendet die Daten über das Audio Gateway an das Headset zurück, wo sie vom Anwender gehört werden können.

4.1.2 Echtzeitanforderungen

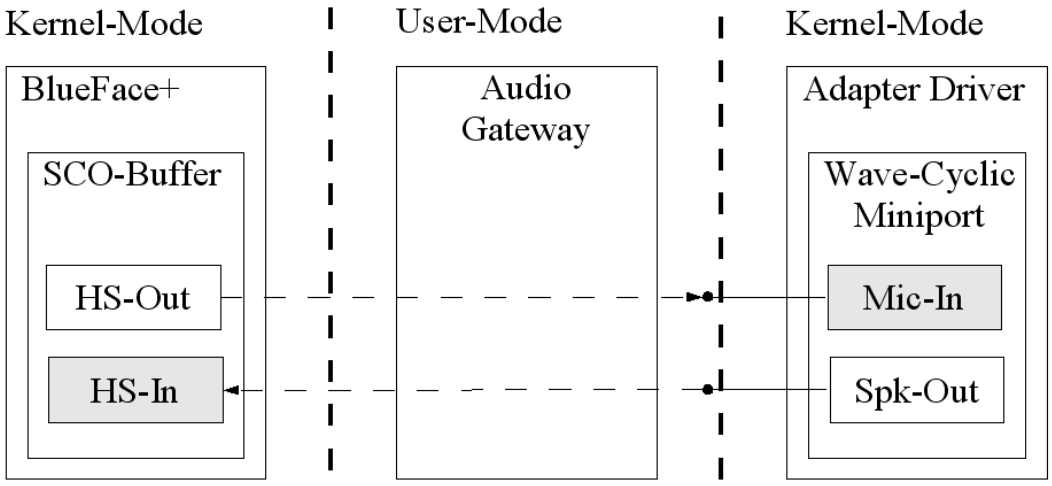

Abbildung 4.2: Echtzeitanforderungen

In beiden Richtungen kann es ein Problem mit dem Jitter[1] geben. Entweder die Pakete kommen nicht regelmäßig an oder können nicht rechtzeitig gesendet werden. In beiden Fällen ist eine erhebliche Minderung der Tonqualität zu erwarten, die sich durch Knackser oder längere

[1] Die zeitlichen Übertragungsabstände der einzelnen Datenpakete weichen in der Realität immer geringfügig vom theoretischen Wert ab.

Aussetzer bemerkbar macht. Probleme hiermit können auf Bluetooth-Ebene und beim Audiotreiber auftreten.

Die SCO-Verbindung über Bluetooth verwirft fehlerhafte oder zu spät eintreffende SCO-Pakete. Daher ist es wichtig, gerade beim Senden der einzelnen Pakete an das Headset einen kontinuierlichen Fluss der Audiodaten zu garantieren. Der Audiotreiber benötigt daher einen Buffer mit angemessener Größe. Das Audiogateway füllt mit diesen Daten den SCO-Sendebuffer des Bluetooth-Stacks auf. Dieser muss immer mindestens ein komplettes SCO-Paket enthalten. Läuft der Buffer leer, muss das Paket mit „Ersatzdaten“ aufgefüllt werden, welche die Qualität mindern.

In der Gegenrichtung ist das Headset für das Senden verantwortlich. Um Unregelmäßigkeiten beim Empfang auszugleichen, ist im Audiotreiber ein weiterer Buffer notwendig. Dieser nimmt die Daten mehrerer Paketlängen auf und fügt sie zu einem durchgängigen Stream zusammen, der an die Spracherkennung weitergeleitet wird.

Bei einem starken Jitter und einem zu kleinen Buffer kann es vorkommen, dass einzelne Pakete nicht rechtzeitig ankommen und somit verworfen werden müssen. Wird der Buffer hingegen zu groß eingestellt, wächst die Latenzzeit merklich. Das Audiogateway, das im user-mode operiert, hat an dieser Stelle einen erheblichen Einfluss auf die Verarbeitungszeiten.

Eine hohe Latenzzeit ist für eine Anwendung zur Spracherkennung weniger kritisch als die Qualität der Audiodaten. Der wichtigste Faktor ist eine hohe Qualität der Daten, die vom Mikrofon zur Sprachsoftware gesendet werden. Die eigentliche Spracherkennung beansprucht in der Regel ebenfalls eine gewisse Zeit, so dass die Übertragungsverzögerung eine eher untergeordnete Rolle spielt. Die Hoffnung bei diesem Design ist, dass ein großer Buffer die Echtzeitproblematik mit dem Audiogateway im user-mode ausreichend mindert. Das Gateway muss „nur“ dafür Sorge tragen, dass Audiotreiber und BlueFace zu jedem Zeitpunkt ausreichend Daten im Buffer besitzen. Dies betrifft insbesondere die beiden grau hinterlegten Buffer in der Abbildung 4.2. Sobald einer dieser Buffer leer ist, kann die Echtzeitverarbeitung nicht mehr gewährleistet werden.

Der Flaschenhals ist das Audiogateway im user-mode. Ein dediziertes Windows-System mit entsprechenden Buffergröße sollte in der Lage sein, die Streams mit ausreichender Geschwindigkeit zu verarbeiten. Bei höheren Anforderungen an die Übertragung ist es unumgänglich, die Streamverarbeitung vollständig in den kernel-mode zu verlagern.

Die theoretische Audioqualität von unkomprimierten 64 kbit/s ist primär auf die menschliche Sprache ausgelegt. Weiterhin dürfte die Komprimierung mit CVSD eine zusätzliche Steigerung bieten. Da das Headset nicht als HiFi-Anlage, sondern nur der Spracherkennung dienen wird, sollte die Qualität ausreichen. Außerdem wird das Vokabular voraussichtlich relativ klein und möglichst auch kontextsensitiv sein. Exakte Erkenntnisse sind nur mit Tests möglich, sobald das System komplett fertig gestellt ist.

4.1.3 Kommunikation zwischen Audiogateway und Treiber

Das Audiogateway übernimmt die Steuerung des Treibers. Es muss ihm mitteilen, wann Audiodaten übertragen werden und die Datenströme lenken. So lange zwischen Headset und Gateway eine SCO-Verbindung besteht, wird der Treiber mit den entsprechenden Daten versorgt, die an den Systemmixer weitergeleitet werden. Das Gateway vermittelt dann die Stre-

ams zwischen Bluetooth-Stack und Audiotreiber. Daraus ergibt sich die benötigte Schnittstelle zwischen Audiogateway und Treiber:

- Start der Übertragung
- Datentransfer vom Headset zum Treiber
- Datentransfer vom Treiber zum Headset
- Ende der Übertragung

Die Standardmethode, einen Treiber mit einer Schnittstelle auszustatten, arbeitet über *I/O Request Packets* (3.5.5.1). Applikationen erhalten darüber die Möglichkeit, auf kernel-mode-Treiber zuzugreifen. Die vier nötigen Aktionen können wie gewöhnliche Zugriffe auf Dateien abgewickelt werden.

Zuerst holt sich das Audiogateway mittels *CreateFile* ein Handle auf den Treiber. Damit wird dem Treiber signalisiert, dass eine Datenübertragung erfolgt. Das Handle kann nun genutzt werden, um *read*- und *write*-Operationen durchzuführen. Dabei wird das write verwendet, um die empfangenen SCO-Daten in den Mic-In-Buffer des Treibers zu schreiben. Das read holt im Gegenzug die Daten aus dem Speaker-Out-Buffer und sendet sie an das Headset. Wird das Headset deaktiviert, beendet der Befehl *CloseFile* die Verbindung zum Treiber.

4.1.4 Kommunikation zwischen Treiber und Spracherkennung

Die Übermittlung der Audiostreams an die Spracherkennungssoftware ist unproblematisch. Der Treiber stellt für Windows-Applikationen eine gewöhnliche Soundkarte dar. Deshalb ist es ohne weiteres möglich, den Mic-In-Kanal des Treibers als Quelle anzugeben. Die Spracherkennung selbst ist in keiner Weise von Treiber oder Audiogateway abhängig.

4.2 Spracherkennung

Wie bereits erwähnt, ist die Auswahl einer Spracherkennungssoftware nebensächlich. Jedes Programm, das ein Mikrofon als Audioquelle verwenden kann, ist geeignet. Daher werden hier nur einige der wichtigen Kriterien, die die Software erfüllen sollte, genannt.

Für die praktische Anwendung macht es Sinn, das Vokabular frei gestalten zu können. Auf diese Weise kann die Erkennungsrate deutlich gesteigert werden, da nur relevante Wörter überprüft werden. Weiterhin ist die Unterstützung von Makros sehr nützlich. Makros setzen gesprochene Befehl in eine festgelegte Tastenkombination um. Diese kann von einer beliebigen interaktiven Anwendung interpretiert werden.

4.3 Audiogateway

Das Audiogateway übernimmt mehrere Aufgaben. Es ist dafür zuständig, die Verbindung zum Headset zu steuern und die Audiodaten an den Treiber für die virtuelle Soundkarte weiterzuleiten.

Um einen Verbindungsaufbau zum Headset überhaupt zu ermöglichen, muss das Audiogateway den Mechanismus des Bluetooth-Pairing unterstützen. Das Headset verlangt bei einer Anfrage einen PIN-Code, der sicherstellen soll, dass kein unbefugter Zugriff stattfindet. Mithilfe dieses Codes wird ein eindeutiger Linkkey ausgehandelt, der ab diesem Moment für die Kommunikation zwischen beiden Seiten verwendet wird.

Das Headset-Profil sieht eine emulierte serielle Schnittstelle vor, die über RFCOMM realisiert wird. Diese Verbindung wird genutzt, um Steuersignale und AT-Kommandos zwischen Gateway und Headset zu übertragen. (HS-Profile 2.4) Beispiele hierfür sind der Aufbau einer Sprachverbindung, sowie die Regelung der Lautstärke am Headset.

4.3.1 gegebene Beispiele / Ausgangscode (BT-Chat)

Das Beispielprogramm BT-Chat ist eine simple Anwendung, die 2 Bluetooth-Geräte miteinander verbinden kann. Voraussetzung ist, dass beide Seiten das Serial Port Profil unterstützen. Der Verbindungsaufbau wird in mehreren Schritten durchgeführt:

1. Suche nach allen verfügbaren Bluetooth-Geräten in der näheren Umgebung
2. (optional) Ermittlung der einzelnen Gerätenamen
3. Auswahl eines der Geräte aus einer Liste
4. Abfrage der angebotenen Dienste des Partners über SDP
5. Auswahl eines geeigneten Dienstes (Serial Port Profile) aus einer weiteren Liste
6. Aufbau der seriellen Verbindung
7. => Übertragung von Datenpaketen (hier Strings) in beiden Richtungen möglich

4.3.2 Design

Das Audiogateway nutzt das Programm BT-Chat als Grundlage. Die meisten benötigten Funktionen sind bereits in diesem Beispiel vorhanden. Die Suche nach geeigneten Zielgeräten und der anschließende Verbindungsaufbau sind nahezu unverändert einsetzbar. Einige Programmteile müssen an die neuen Anforderungen angepasst oder erweitert werden.

Die Arbeitsweise des Programms wird anhand des Zustandsdiagramms 4.3 verdeutlicht. Die für das Audiogateway neu hinzugefügten Zustände sind grau hinterlegt.

4.3.2.1 Registrierung

Bei Programmstart ist das Audiogateway im Startzustand *Not Ready*. Es sendet eine Anfrage an BlueFace, um den Registrierungsvorgang zu starten und wechselt in den Zustand *Registering*. Sobald eine positive Antwort empfangen wird, geht das Programm davon aus, dass die Bluetooth-Hardware ordnungsgemäß eingerichtet ist.

Zur Identifizierung der angebotenen Dienste übermittelt das Audiogateway ein Service Record. Dieses wird von der Bluetooth-Profilspezifikation vorgegeben (Bluetooth SIG 2001). Es

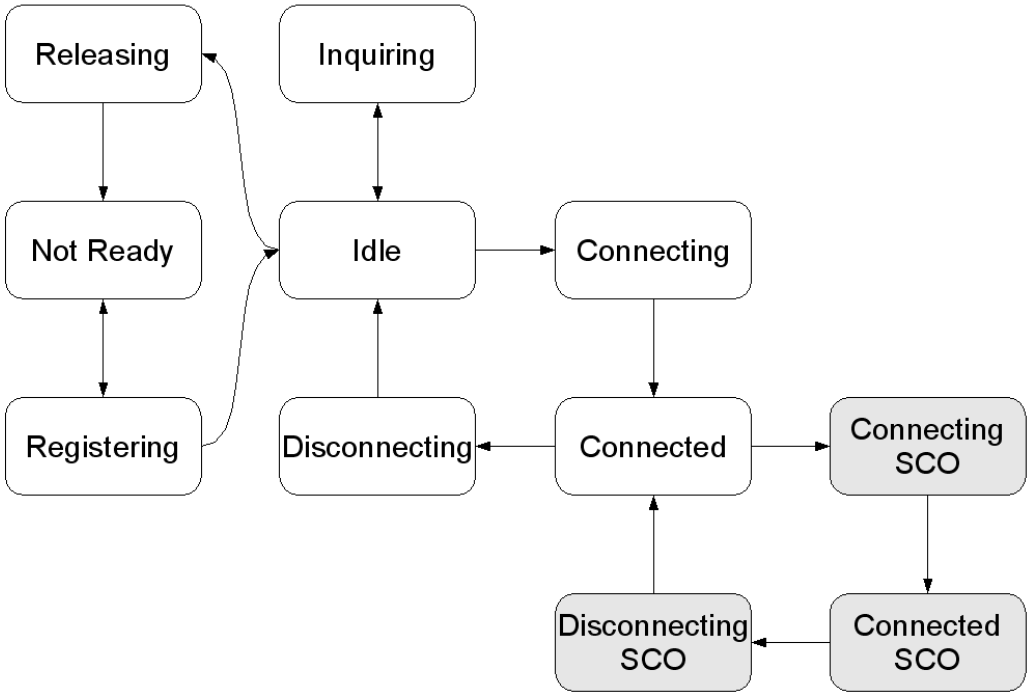

Abbildung 4.3: BTHS-Zustandsdiagramm

enthält Informationen, welche Serviceklassen, Protokolle und Profile vom Gateway unterstützt werden. Die ursprüngliche Einstellung weist die Applikation als RFCOMM-Anwendung aus. Daher wird das neue Service Record auf die Funktion als Audiogateway umgestellt.

Nach erfolgreicher Registrierung befindet sich das Programm im *Idle*-Modus.

4.3.2.2 Suche nach anderen Bluetooth-Geräten

Sobald der Benutzer *Search Devices* auswählt, beginnt die Applikation mit der Suche nach geeigneten Bluetooth-Partnern. Hierbei werden alle Geräte im *discoverable-mode* aufgelistet, die in Reichweite des Senders liegen. Als Identifikationsmerkmal dient die Bluetooth-Adresse der einzelnen Geräte. Soll ein neues Headset eingerichtet werden, muss sich dieses ebenfalls im *discoverable-mode* befinden. Andernfalls kann es nicht gefunden werden.

Wenn eine Liste von Geräten vorliegt, kann in einem 2. Durchgang der jeweilige Name abgefragt werden. *Get Names* fragt bei jedem der bereits gefundenen Geräte nach dem Namen und zeigt ihn an.

Während beider Aktionen befindet sich das Audiogateway im Zustand *Inquiring*.

4.3.2.3 Profilbestimmung

Nachdem ein oder mehrere Geräte gefunden wurden, kann jedes einzelne nach den jeweils angebotenen Profilen abgefragt werden (*Search Profiles*). Dazu ist es notwendig, eine einfa-

che Verbindung mit dem gewünschten Partner aufzubauen. Diese wird dann verwendet, um über das Service Discovery Protocol Informationen einzuholen. Als Suchparameter wird nun nicht mehr *UUID_SERIALPORT* verwendet, sondern *UUID_HEADSET*. Diese Einschränkung bewirkt, dass nur noch kompatible Geräte akzeptiert werden. Nach Austausch der Daten wird die Verbindung wieder abgebaut.

Beim Verbindungsaufbau kann es nun geschehen, dass das angesprochene Gerät eine Authentifikation verlangt. Im Fall eine Headsets ist dies in der Regel ein 4-stelliger numerische PIN-Code. Sobald das Gateway den gewünschten Code übermittelt hat, generieren beide Seiten daraus einen Linkkey. Mithilfe dieses Keys können spätere Verbindungen direkt aufgebaut werden, ohne den Umweg über Suchfunktionen und PIN machen zu müssen.

Die Zustandsabfolge ist während dieser Prozedur: *Connecting*, *Connected*, *Disconnecting* und wieder *Idle*.

4.3.2.4 Steuerkanal zum Headset

Wurde ein Gerät gefunden, welches das Headset-Profil unterstützt, kann dieses vom Audiogateway verwendet werden. Der erste Schritt ist der Aufbau einer RFCOMM-Verbindung zum Headset. Der Schalter *Connect* baut hierzu eine ACL-Verbindung auf, die fortan als Steuerkanal dient.

Headset und Audiogateway sind nun betriebsbereit. Das Gateway ist *Connected* und es können bereits AT-Kommandos ausgetauscht werden. Diese müssen vom Audiogateway korrekt verarbeitet werden. Das ursprüngliche BTChat-Beispiel reagierte bisher nicht auf den Inhalt der gesendeten Strings.

Das Kommando *RING* benachrichtigt das Headset beispielsweise über einen eingehenden Anruf, der angenommen werden kann. Bei Bedarf kann das Headset den Code *AT+CKPD=200* senden, um einen Verbindungswunsch zu äußern.

4.3.2.5 Nutzung des Sprachkanals

Für den Nutzung des Sprachkanals muss eine SCO-Verbindung aufgebaut werden. Die Profil-Spezifikation verpflichtet das Audiogateway dazu, diese Aufgabe zu übernehmen. Verbindungsanforderungen vom Headset können, aber müssen nicht zwingend berücksichtigt werden. Das Ausgangsprogramm BTChat ist nicht in der Lage, SCO-Verbindungen aufzubauen.

Das Audiogateway baut mithilfe von BlueFace eine SCO-Verbindung zum Headset her. Ab diesem Zeitpunkt überträgt das Headset sofort Datenpakete in regelmäßigen Abständen. Gleichzeitig erwartet es vom Gateway ebenfalls Daten, die über den Lautsprecher abgespielt werden sollen. Das spätere Beenden der Verbindung geht immer vom Audiogateway aus.

Das Gateway befindet sich dabei nacheinander in den Zuständen *ConnectingSCO*, *ConnectedSCO*, *DisconnectingSCO*. Start- und Endpunkt ist der Zustand *Connected*.

4.3.2.6 Beenden des Audiogateways

Wenn das Audiogateway beendet wird, muss es sich ordnungsgemäß beim Bluetooth-Stack abmelden. Während dieses Vorgangs befindet es sich im Zustand *Releasing*, bevor es mit

Not-Ready beendet wird.

4.4 Audiotreiber

Der Audiotreiber bildet die Verbindung zwischen Audiogateway und Windows-System. Er leitet die Streams vom Headset an andere Windows-Applikationen und zurück. Als universelle Schnittstelle zur Windows-Welt dient hier das Audio-Interface für normale User-Mode Applikationen, wie zum Beispiel DirectSound.

Der Treiber kann wie eine reguläre Soundkarte verwendet werden. Der Unterschied besteht darin, dass das Audiogateway die Hardware hinter dem Treiber ersetzt.

4.4.1 gegebene Beispiele / Ausgangscode (SB16 / MSVAD)

Microsoft hat dem DDK mehrere Treiberbeispiele im Quellcode beigefügt. Einige von ihnen können als Ausgangspunkt für den benötigten virtuellen Audiotreiber dienen.

4.4.1.1 SB16

Eine Variante ist der Treiber für die SoundBlaster 16 Karte. Die Dokumentation zu dem Treiber deutet an, dass es hierbei weniger um eine korrekte Unterstützung der Hardware, sondern viel mehr um eine Veranschaulichung des Eventhandlings geht. (/audio/sb16/readme.htm) Der Code verlangt einige Hardwareressourcen, um korrekt ausgeführt zu werden.

4.4.1.2 MSVAD - Virtual Audio Device

Es gibt mehrere Beispiele mit identischer Basis, die alle ein virtuelles Audiodevice implementieren. Technologien wie DRM oder AC3 werden angeboten, aber für diese Aufgabe nicht benötigt. Daher ist der „MSVAD simple“ Beispieltreiber am ehesten brauchbar. Er funktioniert ohne Hardware und die Soundausgabe wird in eine WAV-Datei geleitet. Ein Inputstream von einem simulierten Mikrofon ist nicht realisiert. Trotdem werden bereits zwei Streamobjekte mit eigenem Speicherplatz initialisiert, die jeweils eine Richtung des Datenflusses steuern.

Der Treiber nutzt die WDM-Schnittstelle über den WaveCyclic-Miniport, der auch von normalen Soundkarten-Treibern implementiert wird. Die DMA-Zugriffe laufen über einen Ringbuffer im Arbeitsspeicher. In regelmäßigen Abständen werden Interrupts durch einen Timer ausgelöst.

4.4.2 Design

Wie in Abschnitt 4.1 erwähnt, ist es notwendig, einen virtuellen Audiotreiber zu erstellen. Seine Aufgabe besteht darin, dem Kernel-Mixer des Windows-Systems die notwendigen Streaming-Pins bereitzustellen. Zum einen ist dies ein Mic-In, das die Daten des Headsetmikros verarbeitet. Zum anderen bietet der Treiber ein Speaker-Out, der die Systemsounds für den Headsetlautsprecher aufbereitet.

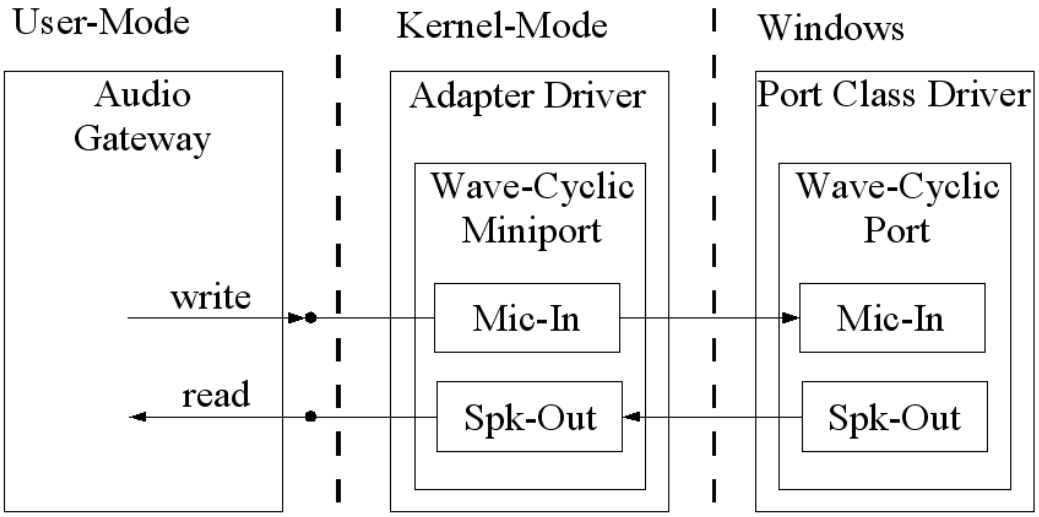

Abbildung 4.4: Audiotreiber

Als Ausgangspunkt für die Programmierung des Audiotreibers ist das MSVAD-Beispiel gut geeignet. Es besteht keine Abhängigkeit von irgendeiner Hardware. Ausserdem enthält es bereits vorkonfigurierte Streaming-Pins, die einer normalen Windows-Applikation Schnittstellen für die Mic-In und Spk-Out Funktionalität bieten. Damit kann der Treiber als reguläres Audiodevice genutzt werden. Eine Abhängigkeit der Anwendung von der verwendeten Hardware wird hiermit vermieden.

Die Streamdaten, die über die Schnittstelle übermittelt werden, werden normalerweise mittels DMA zwischen Soundkarte und Treiber ausgetauscht. Da hier keine tatsächliche Soundkarte angesteuert wird, müssen die Daten über einen anderen Mechanismus bereitgestellt werden. MSVAD besitzt zwei Funktionen, die periodisch auf den „DMA-Buffer“ zugreifen. *CopyFrom* kopiert ein Datenpaket aus dem Ringbuffer in einen zugewiesenen Speicherbereich des Treibers und *CopyTo* in die andere Richtung. Im Beispiel ist nur die *CopyTo*-Funktion mit Code gefüllt. Dieser sorgt dafür, dass Audiodaten aus dem Windows-Mixer in einen weiteren speziellen Ausgangsbuffer des Treibers kopiert werden. Sobald der aktuelle Stream beendet wird, werden die angekommenen Daten in eine Wavedatei gespeichert.

Diese Funktionen des Beispiels ermöglichen es, eigene Audiodaten an den Treiber zu übergeben. Es bietet sich an, hier die Schnittstelle für das Streaming zwischen Treiber und Audiogateway einzurichten. Der Treiber muss 2 Grundoperationen an seiner neuen Schnittstelle anbieten: „read“ und „write“. Damit wird ein externes Programm in die Lage versetzt, den Treiber mit Audiodaten aus einer beliebigen Quelle zu versorgen. Abbildung 4.4 zeigt den Weg, den die beiden Streams durch den Treiber nehmen. Die Verbindung zum Kernel bleibt unverändert. Die wichtigste Änderung ist die Erstellung eines zweiten Interfaces, über das die virtuelle Hardware Daten austauschen kann.

Für eine universelle Anwendung des Treibers ist es wichtig, welches Format diese Daten

haben (Mono/Stereo, Samplingrate, ...). Bei der Verwendung als Treiber für SCO-Daten reicht aber eine Fixierung auf 64kbit/s, mono aus. Da der Treiber im kernel-mode arbeitet und das Audiogateway im user-mode, ist das Gateway nicht in der Lage, die Daten in konstanten Abständen zu verarbeiten. Zur Vermeidung größerer Qualitätsprobleme besitzt der Treiber in beide Richtungen einen Buffer, der Schwankungen der Verarbeitungszeiten abfängt. Die Buffergröße muss in Abhängigkeit von der Übertragungsqualität bestimmt werden.

Später besteht an dieser Stelle viel Raum für Optimierungen. Ein erster wichtiger Schritt ist es, den Audiotreiber über kernel-mode Software zu versorgen.

Wie in Abschnitt 4.1.3 beschrieben, soll die Kommunikation zwischen Treiber und Audiogateway über *IRPs* realisiert werden.

Der Treiber erhält eine eindeutige GUID, die von anderen Programmen genutzt werden kann, um über einen *Create*-Aufruf ein Handle zu erhalten. Ab diesem Zeitpunkt ist möglich, den Treiber ähnlich wie eine normale Datei zu behandeln. In diesem Fall kann mit *Write* ein Audiostream von einem „virtuellen Mikrofon“ an den Treiber übergeben werden. Das *Read* liest dagegen den Stream, der vom Systemmixer generiert wurde und an den Lautsprecher weitergeleitet werden soll. Zum Beenden der Kommunikation wird *Close* aufgerufen, um dem Treiber mitzuteilen, dass seine „Hardware“ die Streams gestoppt hat.

5 Ergebnisse

In diesem Kapitel erfolgt eine Bewertung des Entwurfes. Im Laufe der praktischen Umsetzung sind Probleme aufgetreten, die anschließend diskutiert werden. Weiterhin sollen einige Denkansätze für zukünftige Lösungen vorgestellt werden.

5.1 Realisierung und Evaluation

In diesem Abschnitt wird dargestellt, welche Besonderheiten zu beachten sind, wenn der in Kapitel 4 beschriebene Entwurf realisiert wird. Hierbei sind der Audiotreiber und das Audiogateway die entscheidenden Komponenten. Die angekoppelte Spracherkennung spielt an dieser Stelle keine Rolle, da eine fertige Lösung verwendet werden soll.

Wie in Abschnitt 3.2 angegeben, wurde hauptsächlich mit dem System Windows 2000 gearbeitet. Die Aufwärtskompatibilität zu Windows XP ist theoretisch durch die Verwendung des Windows Driver Models gegeben. Der Code wurde regelmäßig für Windows 2000 compiliert und getestet.

5.1.1 WDM-Treiber

Das Debugging eines Treibers ist erwartungsgemäß schwierig. Hardwarenahe Treiber laufen in der Regel im kernel-mode, der ohne spezielle Tools nicht zu debuggen ist. Spätestens an API-Aufrufen verliert man die Kontrolle über die Geschehnisse. Die Arbeit im kernel-mode kann zudem sehr schnell zu einem Bluescreen führen, der das gesamte System lahmlegen kann.

Außerdem kommt beim Streaming die Echtzeitproblematik hinzu. Künstliche Wartezeiten im Treiber enden sehr schnell in einem Timeout oder einem inkonsistenten System. Gerade Interrupts sind in dieser Hinsicht sehr empfindlich.

Die im DDK von Microsoft enthaltenen Treiber sind eine zweischneidige Angelegenheit. Einerseits bieten sie brauchbare Templates für die Treiberprogrammierung unter Windows, aber andererseits sind sie leider nicht ausgereift. Die Dokumentation ist sehr knapp gehalten und lückenhaft. Als Ergänzung ist ein Buch über Windows-Treiber (Baker und Lozano 2000) sehr zu empfehlen. Andernfalls verliert man sich schnell in den unübersichtlichen Weiten des MSDN.

SB16-Treiber Das Beispiel für SoundBlaster 16 Karten funktionierte nicht korrekt mit der entsprechenden Hardware. Außerdem muss ein sehr hoher Aufwand getrieben werden, um die Hardwareabhängigkeit zu beseitigen.

Virtual Audio Device Das Virtual Audio Device bietet eine deutlich bessere Ausgangsbasis. Da es ein fast vollständiger virtueller Treiber ist, sind nur wenige Umbauarbeiten nötig.

Bevor das MSVAD-Beispiel korrekt für Windows 2000 kompiliert werden kann, müssen unter Umständen einige Maßnahmen unternommen werden. Dies ist nur notwendig, wenn der Compiler ebenfalls unter Windows 2000 gestartet wird. (Clearwater Software 2004)

Bei Tests mit dem Treiber trat ein Phänomen auf, das die weitere Arbeit erschwert hat. Die Ausgabe der Streams erfolgt im Normalfall in WAV-Dateien auf die Festplatte. Allerdings funktionierte dies nur in sehr wenigen Fällen korrekt. Nach einer systematischen Analyse stellte sich heraus, dass die Write-Routine des Treibers fehlerhaft ist. Nach der Initialisierung des Treibers wird nur der erste Stream korrekt auf die Festplatte geschrieben. Alle folgenden Streams werden ignoriert. Einige Programme, wie zum Beispiel der Microsoft Audiorecorder, öffnen beim Programmstart einen Output-Stream und schließen ihn sofort wieder. Dadurch wird der Treiber bereits „verstellt“ und kann daher keine Audiodaten mehr abspielen. Glücklicherweise ist dieser Bug für den endgültigen Treiber unerheblich, da der Output-Stream ohnehin nicht in einer Datei gespeichert, sondern über Bluetooth verschickt werden soll.

IRP-Interface Die Kommunikation zwischen Audiogateway und Treiber soll über IRPs erfolgen. Dieses bietet die unter anderem die Kommandos Create, Read, Write und Close. Die Verwendung dieser Kommandos ist prinzipiell möglich. Das Lesen und Schreiben im Speicher des Treibers funktioniert vom user-mode aus, wenn der Treiber nur diese Aufgabe wahrnimmt.

Allerdings besitzt ein Driver-Object nur genau ein Interface und damit einen Create-Aufruf. Dabei ist es egal, ob der Treiber nur ein Device oder mehrere enthält. Das Kernproblem ist, dass das virtuelle Audiodevice bereits diese Schnittstelle bereits für das Kernel-Streaming nutzt. Versucht eine Anwendung das Create auszuführen, wird die vorhandene Funktion des WDM gestartet. Da diese nur für Soundkarten gedacht ist, ist der Aufruf ohne Erfolg und gibt ein ungültiges Handle zurück.

Es ist möglich, eine selbst geschriebene Create-Prozedur zu schreiben und damit den Aufruf des Create abzufangen. Das Problem besteht bei diesem Vorgehen aber darin, dass die aufrufende Applikation nicht mitteilen kann, welches Device angesprochen werden soll. Damit ist eine unterschiedliche Behandlung der einzelnen Devices nicht mit einfachen Mitteln realisierbar.

5.1.2 Bluetooth

Der Bluetooth-Stack der Firma Stollmann hat wenig Probleme bereitet. Eine genauere Analyse ist aber an zwei Stellen erforderlich.

USB-Dongle mit „SCO over HCI“ Für das Audiogateway wird ein USB-Dongle benötigt, das den Modus „SCO over HCI“ unterstützt. Dieser wird benötigt, um eine SCO-Verbindung zum Headset aufbauen zu können. Das verwendete Sitecom-Dongle war in der Standardkonfiguration nicht dazu in der Lage. Ein kurzfristig eingespielter Hack konnte dieses Problem auf einigen Systemen beseitigen. Allerdings war es notwendig, den Bluetooth-Treiber nach jedem Programmstart zu reinitialisieren.

SCO Buffer Das Audiogateway hat für Tests die Aufgabe erhalten, SCO-Pakete als Echo an das Headset zurückzusenden. Sobald eine Verbindung steht, beginnt das Headset mit dem regelmäßigen Senden von Paketen. Nach dem Eintreffen eines Paketes wird dieses in den lokalen Speicher kopiert und der Empfang sofort dem BlueFace bestätigt. Damit soll der belegte Buffer im Stack so schnell wie möglich wieder freigegeben werden. Die Daten werden dann über den SCO-Kanal wieder an das Headset zurückgesendet.

Bei dieser Vorgehensweise scheint der Buffer im Bluetooth-Stack nach kürzester Zeit voll zulaufen, was einen Verbindungsabbruch nach sich zieht. Eine Erweiterung des Buffers von 4 auf 20 Pakete, sowie die Deaktivierung des Loggings brachte keine Lösung. Laut Stollmann ist es nicht auszuschließen, dass der Testrechner (Laptop mit AMD 1700+ CPU) zu langsam für die Abwicklung in Echtzeit ist. Insbesondere die Realisierung im user-mode scheint hierbei für Zeitprobleme verantwortlich zu sein.

5.2 aktueller Entwicklungsstand

In diesem Abschnitt wird ein kurzer Überblick über die bisherigen Fortschritte in der Realisierung des Projektes gegeben. Der praktische Teil der Arbeit befindet sich noch im experimentellen Stadium. Die einzelnen Komponenten sind vorhanden, aber die im vorhergehenden Abschnitt beschriebenen Schwierigkeiten wurden noch nicht behoben.

5.2.1 Audiotreiber

Der Audiotreiber besteht noch nahezu vollständig aus dem MSVAD-Beispiel von Microsoft. Einige Änderungen am Code waren nötig, um den Treiber für Windows 2000 kompilierbar zu gestalten. In 4.4.2 ist beschrieben, an welcher Stelle das Interface für das Audiogateway programmiert werden muss. Der bisherige Ansatz mit IRPs konnte nicht wie geplant implementiert werden (siehe Abschnitt 5.1.1), so dass hierfür kein ordnungsgemäß funktionierender Code vorliegt. Es sollte geprüft werden, ob es Alternativen zum IRP-Ansatz gibt oder wie eine Realisierung mit IRPs doch noch möglich ist. Der Abschnitt 5.4 beschreibt eine weitere Design-Möglichkeit, in der das Interface nicht mehr vom Audiogateway abhängig ist.

5.2.2 Audiogateway

Das Audiogateway kann über einen USB-Dongle Bluetooth-Geräte im *discoverable-mode* entdecken. Nach Auswahl eines Gerätes wird überprüft, ob das Headset-Profil unterstützt wird. Hierbei verlangt das Headset eine Authentifikation, bei der standardmäßig die Pin „0000“ übermittelt wird. Es wird ein Linkkey erzeugt, der als Variable im Audiogateway festgehalten wird. Danach kann die RFCOMM-Verbindung hergestellt werden, über die ein Austausch von AT-Kommandos möglich ist (z.B. wird durch das Senden von *RING* ein Klingelton im Headset abgespielt).

Bis zu dieser Stelle funktioniert das Gateway wie in Abschnitt 4.3.2 vorgesehen. Der Aufbau einer SCO-Verbindung gelingt, wenn ein passendes Dongle vorhanden ist. Nach der Übertragung einiger Datenpakete bricht die Verbindung allerdings zusammen. Die Hauptursache

scheint hier in der mangelnden Echtzeitfähigkeit des user-mode zu liegen. Der SCO-Buffer unter BlueFace läuft somit voll und erzeugt eine Fehlermeldung.

5.3 Ausblick

Dieser Abschnitt beschreibt kurz, welche sinnvollen Änderungen am Entwurf denkbar sind. Ein vollständiges Redesign des Projektes wird im Abschnitt 5.4 vorgestellt.

5.3.1 Optimierung der Geschwindigkeit

Zur Verkürzung der Latenzzeiten bietet BlueFace+ eine Möglichkeit, die im Zuge der Studienarbeit nicht weiter behandelt wird. Es kann über das *Channel Management* (Stollmann E+V GmbH 2003) gesteuert werden, welche Applikation die Daten einer Verbindung erhalten soll.

Die Steuerung des Headsets wird hierbei ebenfalls über das Audiogateway abgewickelt. Allerdings besteht der entscheidende Unterschied darin, die SCO-Daten direkt vom Stack an den Treiber zu leiten. So wird der Umweg der zeitkritischen Streams über eine usermode-Applikation vermieden.

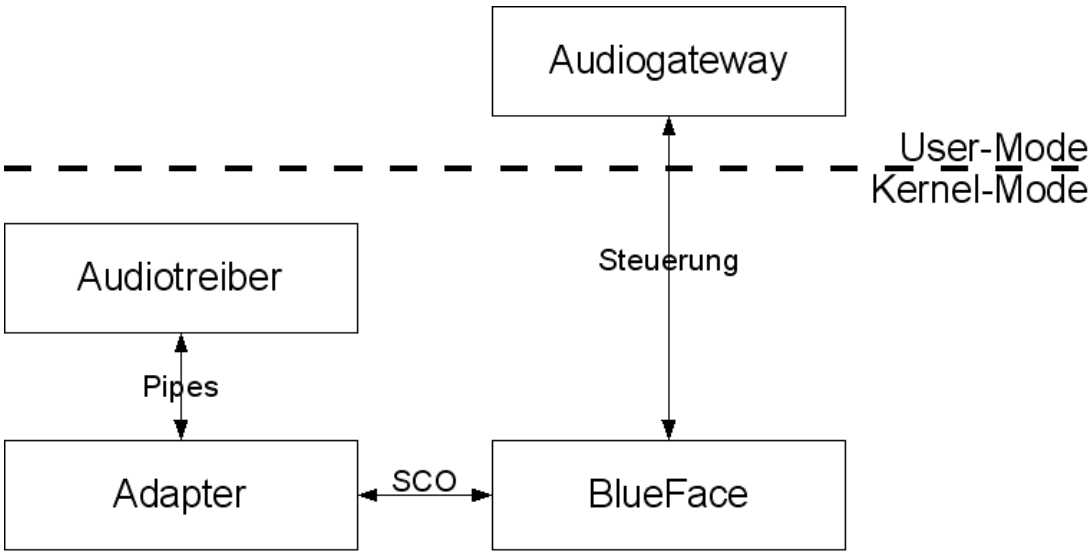

Abbildung 5.1: Redesign mit Adapter

Weiterhin ist die Realisierung mithilfe zweier Pipes denkbar. Dabei kommuniziert ein weiterer kernel-mode Treiber über Pipes mit dem Audiotreiber. Auf der anderen Seite besitzt der zusätzliche Treiber eine Schnittstelle zu BlueFace.

Auf diese Weise kann der Audiotreiber komplett von Audiogateway und BlueFace abgekoppelt werden. Eine Nutzung des Treibers für andere Zwecke wird so vereinfacht. Wie in Kapitel 4 und Abschnitt 5.1.1 bereits erwähnt, steigt dabei der Aufwand bei der Programmierung, da hier im kernel-mode gearbeitet wird.

5.3.2 ergonomisches Audiogateway

Das Audiogateway ist in der einfachen Version lediglich in der Lage, ein einziges Headset zu bedienen. Bei jedem Wechsel muss die Prozedur des Pairings erneut durchgeführt werden. Dabei geht der alte Linkkey verloren. In Zukunft ist es denkbar, mehrere Headsets parallel zu konfigurieren. Auf diese Weise kann jeder Benutzer sein individuelles Headset verwenden.

Weiterhin kann das Interface des Headsets genutzt werden, um aktiv den SCO-Kanal zum PC aufzubauen. Das Headset kann im Normalfall über eine Taste eine Verbindung zum zugehörigen Handy initiieren. Das hierbei gesendete Kommando kann vom wartenden Audiogateway ausgewertet werden, um entsprechend zu reagieren.

5.4 Redesign

Im Verlauf der Studienarbeit wurde deutlich, wo kritische Stellen bei der Realisierung des Projektes liegen. Eine zukünftige Realisierung muss die gesamten Echtzeitkomponenten konsequent im kernel-mode enthalten. Andernfalls besteht die Gefahr, dass die Komponenten im user-mode eine zu hohe Systemlast erzeugen.

Der ursprüngliche Entwurf in Kapitel 4 ist bewusst einfach gehalten. Es wurde wenig Wert auf die Verarbeitungsgeschwindigkeit gelegt. Der Hauptaspekt der Studienarbeit war der Beweis der generellen Machbarkeit. Die Systemlast, die der Umweg über den User-Mode bewirkt, ist allerdings zu hoch, um einen funktionierenden Prototypen zu erstellen.

Ein mögliches Gesamtdesign des Projektes kann wie in Abbildung 5.2 aussehen.

Der entscheidende Unterschied ist die geänderte Verarbeitung der SCO-Daten, wie es in Abschnitt 5.3.1 beschrieben wird. Auf diese Weise werden sämtliche zeitkritischen Operationen im kernel-mode durchgeführt.

5.4.1 Audiotreiber

Der Audiotreiber an sich ist nur geringfügig zu ändern. Er bleibt eine passive Komponente, die von außen getriggert werden kann. Die Schnittstelle zum Systemkernel kann unverändert bleiben. Die Hauptarbeit liegt am Interface für die virtuelle Hardware. Diese benötigt weiterhin Zugriff auf die beiden Streamingbuffer.

Der bisherige Ansatz sieht vor, dass das Audiogateway im user-mode die Rolle der Hardware übernimmt. Die Kommunikation sollte über den Standardmechanismus der IRPs ablaufen. Die damit verbundenen Probleme werden in Abschnitt 5.1.1 beschrieben.

Das neue Interface sollte im Idealfall eine bidirektionale Pipe zur Verfügung stellen, die für den Zugriff auf die Streamingbuffer genutzt werden. Hierbei ist es nicht mehr notwendig, das Interface für den user-mode sichtbar zu machen. Der zugehörige Treiber (in diesem Fall der Adapter), der Audiodaten liefert, kann sie aus einer beliebigen Quelle beziehen.

5.4.2 Audiogateway

Das Audiogateway ist auch im neuen Entwurf die Kontrollinstanz für das Headset. Verbindungsaufbau und Authentifizierung sind die Hauptaufgaben. Der RFCOMM-Kanal für die AT-

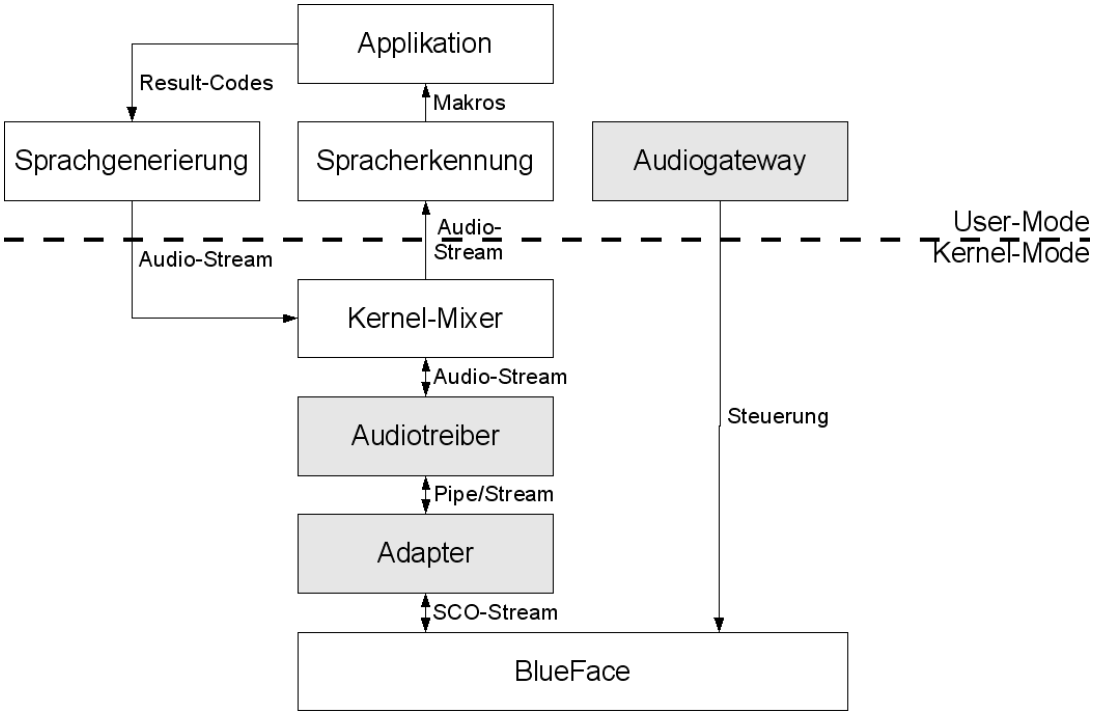

Abbildung 5.2: Redesign

Kommandos bleibt wie bisher bestehen.

Die neue Anforderung an das Gateway ist die Nutzung des *Channel Management* von BlueFace+. Der Aufbau der SCO-Verbindung wird vom Audiogateway übernommen. Dabei wird BlueFace mitgeteilt, welche Anwendung die Nutzdaten erhalten soll. In diesem Fall ist dies der neue Adapter im kernel-mode. Um dies zu ermöglichen, benötigt das Gateway das entsprechende BlueFace-Handle des Adapters.

5.4.3 Adapter

Der Adapter bildet die Brücke zwischen Audiotreiber und BlueFace. Dieser Treiber wird im kernel-mode betrieben und ersetzt die alte SCO-Vermittlung des Audiogateways. Auf diese Weise bleibt die Streambehandlung vollständig im schnellen kernel-mode. Der angestrebte Echtzeitbetrieb wird auf diese Weise deutlich besser unterstützt.

Bei der Initialisierung des Adapters muss eine Verbindung zum Interface des Audiotreibers hergestellt werden. Vorzugsweise ist an dieser Stelle eine Pipe für die Lese-/Schreiboperationen auf den Buffer einzusetzen. Alternativ kann ein Message-System, das die Reihenfolge der Nachrichten einhält, eingesetzt werden. Zusätzlich muss eine Registrierung bei BlueFace stattfinden, um ein gültiges BlueFace-Handle zu erhalten. Mit Hilfe dieses Handles kann BlueFace alle empfangenen SCO-Pakete direkt an den Adapter weiterreichen und umgekehrt können neue SCO-Pakete in Gegenrichtung verschickt werden.

Empfängt BlueFace ein SCO-Paket vom Headset, sendet es dem Adapter eine Message mit dem Kommando *BT_DATA_IND*. Eine Callback-Routine im Adapter wertet nun die Nachricht aus und liest die Daten aus dem SCO-Buffer. Diese Daten werden sofort über die Pipe an den Audiotreiber weitergeleitet.

Für den Datentransfer vom System zum Headset läuft ein ähnlicher Mechanismus ab. Sobald der Audiotreiber neue Daten im Spk-Out-Buffer besitzt, wird der Adapter benachrichtigt. Dieser liest die Daten aus der Pipe und sendet sie sofort mit dem Kommando *BT_DATA_REQ* an BlueFace.

5.5 Schlussfolgerungen

Die Idee der Studienarbeit ist die Nutzung eines Bluetooth-Headset als Eingabegerät. Im Verlauf der Arbeit wurde deutlich, dass die Grundvoraussetzung zur Realisierung ein effizienter Audiotreiber ist, der die Echtzeitverarbeitung beherrscht.

Es wurde deutlich, dass das Projekt mit vertretbarem Aufwand realisierbar ist. Interessant ist, dass zu Anfang der Arbeit noch keine Lösung auf dem Markt vorhanden war. Erst der Hub von Logitech bietet erstmals die Nutzung eines Bluetooth-Headsets als reguläre Audiohardware am PC.

Die Studienarbeit zeigt, dass an sich ein minimaler Hardware-Einsatz (Headset und USB-Dongle) ausreicht, um Headset und PC zusammen zu nutzen. Es wird sicherlich verschiedene Anwendungsgebiete für dieses Projekt geben, sobald die notwendigen Komponenten entwickelt sind. Das modulare Design ermöglicht hierbei sogar eine Entkopplung der einzelnen Programme. Beispielsweise eröffnet die Möglichkeit, die Audiodaten einer beliebiger Software zur Verfügung zu stellen, ein weites Feld an zusätzlichen Einsatzgebieten. Das Headset kann unter anderem als Funkmikrofon bei Präsentationen eingesetzt werden oder für IP-Telefonie. Auf der anderen Seite kann die Bluetooth-Technologie vollständig ersetzt werden. Der Audiotreiber sorgt in diesem Fall dafür, dass Streams aus einer anderen Quelle in das Windows-System gelangen können.

Literaturverzeichnis

Axis Communications 2004 : *AXIS OpenBT Stack*. 2004. – URL http://developer.axis.com/software/bluetooth/. – Zugriffsdatum: 2004-04-25

Baker und Lozano 2000 BAKER, Art ; LOZANO, Jerry: *The Windows 2000 Device Driver Book, a Guide for Programmers, Second Edition*. Prentice Hall PTR, 2000. – ISBN 0-13-020431-5

Bluetooth SIG 2001 : *Specification of the Bluetooth System - Profiles*. 2001. – URL http://www.it.lut.fi/~doc/bluetooth/Bluetooth_11_Profiles_Book.pdf. – Zugriffsdatum: 2004-04-27

Clearwater Software 2004 : *Compiling MSVAD under Windows 2000*. 2004. – URL http://www.clearwater.com.au/msvad/. – Zugriffsdatum: 2004-05-03

DFKI GmbH 2004 : *SmartKom - Dialogische Mensch-Technik-Interaktion durch koordinierte Analyse und Generierung multipler Modalitäten*. 2004. – URL http://www.smartkom.org/. – Zugriffsdatum: 2004-04-28

Ericsson 2002 : *Bluetooth Beginner's Guide*. 2002. – URL http://www.ericsson.com/bluetooth/beginners_files/beginners_guide.pdf. – Zugriffsdatum: 2004-04-26

Gelbart 2002 GELBART, David: Reducing the Effect of Room Acoustics on Human-Computer Interaction. (2002), Mai. – URL http://www.smartkom.org/reports/Report-NR-20.pdf. – Zugriffsdatum: 2004-04-25

Holtmann und Krasnyansky 2004 HOLTMANN, Marcel ; KRASNYANSKY, Max: *Official Linux Bluetooth protocol stack*. 2004. – URL http://www.bluez.org/. – Zugriffsdatum: 2004-04-25

Logitech Inc. 2003 : *Benutzen des Bluetooth-Headsets mit dem kabellosen Logitech-Hub*. 2003. – URL http://www.logitech.de/pub/pdf/bluetooth/deu/bluetooth_headset_how_to.pdf. – Zugriffsdatum: 2004-04-25

Mettala 1999 METTALA, Riku: *Bluetooth Protocol Architecture - Version 1.0*. 1999. – URL https://www.bluetooth.org/foundry/sitecontent/document/Protocol_Architecture. – Zugriffsdatum: 2004-04-26

Microsoft Corporation 2003 : *WDM Audio Drivers*. 2003. – URL http://www.microsoft.com/whdc/archive/wdmaudio.mspx. – Zugriffsdatum: 2003-02-28

Microsoft Corporation 2004 : *Microsoft Platform SDK*. 2004. – URL `http://msdn.microsoft.com/platformsdk/`. – Zugriffsdatum: 2004-04-25

Muller 2001 MULLER, Nathan J.: *Bluetooth*. 1. Auflage. MITP-Verlag, 2001. – ISBN 3-8266-0738-4

Özkilic und Zivadinovic 2003 ÖZKILIC, Murat ; ZIVADINOVIC, Dusan: *Vertikale Schnitte - 13 neue Bluetooth-Profile*. 2003. – URL `http://www.heise.de/mobil/artikel/2003/04/18/bluetooth-profile/`. – Zugriffsdatum: 2004-04-27

ScanSoft, Inc. 2004 : *ScanSoft*. 2004. – URL `http://www.scansoft.de/`. – Zugriffsdatum: 2004-05-03

Stollmann E+V GmbH 2003 : *BlueFace+ Version 1.14*. dec 2003

Weiser 1996 WEISER, Mark: *Ubiquitous Computing*. 1996. – URL `http://www.ubiq.com/hypertext/weiser/UbiHome.html`. – Zugriffsdatum: 2004-04-24

Widcomm Inc. 2004 : *Widcomm BTW - Bluetooth for Windows*. 2004. – URL `http://www.widcomm.com/Products/bluetooth_comm_software_btw.asp`. – Zugriffsdatum: 2004-04-25