Bachelorarbeit

Thilo Stallherm

Ein Streaming Media Player für Symbian Smartphones in NAT-beschränkten Netzen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Thilo Stallherm

Ein Streaming Media Player für Symbian Smartphones in NAT-beschränkten Netzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof. Dr. rer. nat. Hans H. Heitmann

Abgegeben am 28. September 2007

Thilo Stallherm

Thema der Bachelorarbeit
Ein Streaming Media Player für Symbian Smartphones in NAT-beschränkten Netzen
Stichworte
Smartphone NAT Streaming RTP Progressive-Download
Kurzzusammenfassung
Trotz starker Fortschritte im Bereich des mobilen Streaming, treten immer wieder Inkompatibilitäten zwischen der Smartphone Software und den Netzbetreibern auf. Alternative Streaming Methoden scheinen ein Ausweg aus dieser Situation zu sein. Im Folgenden wird ein Streaming Media Player entwickelt, der mögliche Einschränkungen seitens der Netzbetreiber umgeht.

Thilo Stallherm

Title of the paper
Streaming Media Player for Symbian Smartphones in NAT restricted Networks
Keywords
Smartphone NAT Streaming RTP Progressive-Download
Abstract
Although the development of mobile Streaming has been fast in recent years, there are still inconvenient incompatibilities between the Smaprtphone Software and the Network Operator. Alternative streaming methods could become a solution to this problem. This Thesis deals with the Media Player application that was developed in order to employ such a streaming method.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren ist die Entwicklung im Bereich der mobilen Netzwerke und Geräte stetig fortgeschritten. Heutzutage haben Smartphones genug Rechenkapazität, Speicher und Multimedia Features, die man bis vor gar nicht so langer Zeit nur auf Laptops und PDAs sehen konnte. Zusammen mit schnellen packetorientierten Internettarifen wird das mobile Internet Realität.

Die ersten mobilen Internet Angebote beinhalteten Text basiertes Web Browsing, E-Mail und andere einfache Service Angebote, die keine hohen Verbindungsgeschwindigkeiten oder geringe Latenzzeiten benötigten. Die hohe Geschwindigkeit der mobilen Weiterentwicklung hat es möglich gemacht, fortschrittlichere Online-Angebote auf das mobile Internet zu adoptieren. Einer der interessantesten aber auch anspruchsvollsten Dienste ist das bei das Streamen von Multimedia Inhalten. Streamen bedeutet für den Anwender, dass er sofort mit der Betrachtung beginnen kann und keine langen Wartezeiten verbringen muss. Moderne Kompressionsalgorithmen und die neuste Generation an Smartphones machen dies heutzutage möglich. Dennoch gibt es immer noch starke Unterschiede bei den Netzbetreibern in der Geschwindigkeit in der diese neuen Technologien unterstützt werden.

Die Virtualreel Software der Firma portrix.net speichert und archiviert Werbespots. Der Benutzer kann dabei über Stichwörter Spots finden und auswählen. Beim Einspielen von Werbespots erstellt die Software automatisch Kopien in verschiedenen Videoformaten. Für eine Darstellung auf Mobiltelefonen werden auch Kopien im 3gp Dateiformat vorgehalten. Zur Darstellung der Videos lassen sich anschließend so genannte Online-Reels erstellen. Ein Online-Reel ist eine Präsentation verschiedener vorher festgelegter Videos. Über eine Webseite wird der Inhalt des Reels übersichtlich dargestellt und kann online betrachtet werden. Bei ersten Versuchen diese Clips über verschiedene Deutsche Netzbetreiber zu streamen, sind starke Unterschiede aufgefallen. Bei dem Einen funktionierte der Abspielvorgang problemlos, bei dem Anderen dagegen nicht. Um dieser Unzulänglichkeit zu begegnen, wurde entschieden, alternative Möglichkeiten des Streamens zu erkunden und bei Erfolg mit einer eigenen Media Player Applikation umzusetzen.

Kapitel 2

Mobile Media

2.1 Smartphones

Der Begriff „Smartphone“ kommt aus einer Zeit, zu der die meisten Mobiltelefone ausschließlich telefonieren konnten. Neben regulären Mobiltelefonen gab es noch einen anderen Technologiezweig der immer mobiler wurde: der Computer. Zuerst mit Laptops und später mit PDAs wurde der Bedarf an einer Kombination beider Technologien immer deutlicher. Mit dem Erscheinen der ersten Smartphones konnte man nun nicht nur telefonieren, sondern man hatte auch Zugriff auf ein grafikfähiges Display und konnte Fähigkeiten nutzen wie:

- Internetzugriff: E-Mails, WWW
- Verwaltung von Terminen und Kontakten
- Erweiterbarkeit durch Installation von neuen Anwendungen.

2.1.1 Symbian OS

Symbian OS ist ein spezielles Betriebssystem für mobile Geräte. Produziert von Symbian Ltd. wurde es von Anfang an speziell für dieses Einsatzgebiet entwickelt. Andere Smartphone Betriebssysteme (wie z.B. Windows Mobile) gingen einen anderen Weg und entwickelten sich aus Betriebssystemen die für größere Systeme geschrieben wurden.

Wie jedes andere Mobiltelefon ist ein Smartphone ein „embedded System“. In diesem Zusammenhang gelten für ein Betriebssystem besondere Anforderungen.

Neben geringer Größe und geringem Gewicht ist vor allem eine hohe Batterielaufzeit ein wichtiges Kriterium. Im Gegensatz zu Desktop Computern müssen deswegen deutliche Einschränkungen bei Speichergröße und Rechenpower gemacht werden. Das Betriebssystem untersützt diese Voraussetzungen indem es selbst sehr sparsam mit den Hardware Ressourcen, insbesondere Speicher, umgeht. Ein Descriptor-Framework unterstützt den

Programmierer speichersparend zu implementieren. Außerdem gestaltet sich die Programmierung für Symbian Smartphones Ereignis gesteuert. Durch diese Technik kann die CPU abgeschaltet werden, sobald kein neues Ereignis ansteht.

Ein Mobiltelefon läuft im Regelfall viele Monate am Stück. Ein regelmäßiges Neu starten des Betriebssystems ist nicht akzeptabel. Ein wichtiger Punkt dabei ist das Verhindern von Memory- und Ressourcen-Leaks. Das Symbian OS bietet neben dem ANSI C++ Exception-Handling ein alternatives leicht gewichtiges Trap/Leave Error-Handling Konstrukt. Durch einen integrierten Cleanup Stack werden benutzte Ressourcen im Fehlerfall automatisch frei gegeben.

Durch diese zusätzlichen Programmiertechniken ist der Symbian C++ Slang gewöhnungsbedürftig und erfordert zusätzliche Einarbeitungszeit.

Symbian Ltd. stellt selbst keine Geräte her. Stattdessen wird das Betriebssystem an Gerätehersteller lizenziert. Da sich das Gerätedesign der verschiedenen Hersteller stark unterscheidet liefert Symbian nur eine sehr rudimentäre User Interface Unterstützung und es bleibt dem Hersteller überlassen eine UI Plattform zu entwickeln oder zu lizenzieren.

2.1.2 S60

Die Series60 (S60) Plattform ist eines der bekanntesten Plattformen für Symbian Smartphones. S60 erweitert Symbian nicht nur durch ein UI-Framework, sondern bietet auch erweiterte Application Services wie z.B.:

- erweiterte Multimedia Unterstützung: Real-Player API
- Web-Browser API
- Instant Messaging Framework
- Communications services, u.a.: HTTP-API

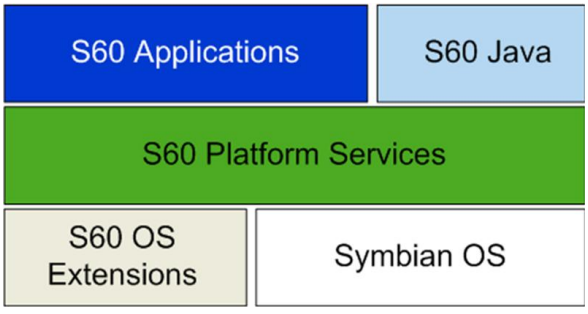

Abbildung 2.1: S60 High level Architektur

Neben C++ werden noch verschiedene andere Programmiersprachen durch S60 unterstützt. Zur Verfügung stehen zusätzliche Software Development Kits für Java ME, Open C und Python. Um performante Applikationen zu entwickeln und alle Features, vor allem im Bereich Multimedia, nutzen zu können, wird aber eine Entwicklung in C++ empfohlen. Abbildung 2.1 zeigt diese Zusammenhänge.

Entwickelt für eine ein händige, Tasten basierte Bedienung stellt das Framework bestimmte Anforderungen an das Layout des Gerätes und bietet ein einheitliches Look And Feel der Applikationen.

2.1.3 Das Entwicklungsgerät

Abbildung 2.2: Nokia 6120 Classic

Als Entwicklungsgerät stand ein Nokia 6120 Classic zur Verfügung. Das Smartphone läuft mit der Plattform Nokia S60 3rd Edition, Feature Pack 1 die wiederum auf Symbian OS v9.2 basiert.

Wichtige Eckdaten:

- Display: 240 x 320 bei 24 bit
- freier ausführbarer RAM Speicher: 20 MB
- Netzwerk Unterstützung u.a.: HSDPA, UMTS, EDGE, GPRS

2.2 Mobile Streaming

Streaming ist eine Methode der kontinuierlichen Übertragung von Daten mit Echtzeitcharakter. Im Gegensatz zum Download kann ein Benutzer sofort mit dem Betrachten der Repräsentation beginnen während die Datenübertragung weiter läuft. Der Nachteil dabei ist, dass

die Repräsentation stark von der maximal verfügbaren Datenrate abhängt. Obwohl theoretisch jede Art von Daten übertragen werden kann, beschäftigt sich diese Arbeit ausschließlich mit der Übertragung von Audio und Video Inhalten.

Der Begriff Mobile Streaming wird benutzt, wenn der Inhalt an ein Gerät in einem mobilen Netzwerk gesendet wird. Die Datenpakete des Streams werden von einem Medienserver generiert. Der Server erhält dabei die benötigten Audio und Video Daten aus einer vorher bereitgelegten Datei oder direkt live z.B. aus einer angeschlossenen Kamera. Streamen von einer Datei wird als On-Demand Streaming bezeichnet. Hier kann der Benutzer gewöhnlich den gewünschten Stream aus einer Liste selektieren. Im Gegensatz zum reinen Live Streaming ermöglicht On-Demand Streaming darüber hinaus ähnliche Funktionen wie ein Videorekorder. Z.B.: Pause, Vorlauf oder Rücklauf.

Übertragunskapazität ist gerade im mobilen Bereich stark begrenzt. Deswegen werden die Mediendaten komprimiert übertragen. Die Bitrate eines Medienstreams legt dabei die Anzahl an Nutzdaten in einer Zeiteinheit fest. Die benutzte Bitrate hat einen starken Einfluss auf die erreichbare Qualität der Repräsentation. Die Bitrate beeinflusst auch die mindestens benötigte Bandbreite der Datenübertragungsstrecke. Bedingt durch Protokoll Overhead sollte die verfügbare Bandbreite größer sein als die theoretisch benötigte Mindestbandbreite.

Die generierten Datenpakete werden von dem Streamingserver kontinuierlich an den Empfänger gesendet. Kommt es dabei zu Übertragungsfehlern oder Engpässen der Datenrate entstehen bei der Betrachtung Unterbrechungen und Aussetzer. Um solche Fehler zu vermeiden, verzögert der Media Player üblicherweise die Wiedergabe um einen Datenpuffer (siehe Abbildung 2.3) zu füllen. Dieser Prozess wird als Buffering bezeichnet. Buffering kann auch während der Wiedergabe auftreten, wenn sich Übertragungsfehler häufen. Umso größer der Datenpuffer ist, umso besser können solche Übertragungsfehler ausgeglichen werden. Zum Nachteil entwickelt sich allerdings dabei die ebenfalls erhöhte Verzögerung.

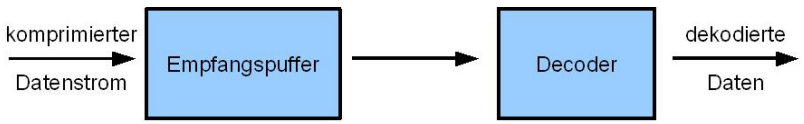

Abbildung 2.3: Position des Puffers im Stream

2.2.1 GSM Übersicht

Global System for Mobile Communications (GSM) ist ein digitaler Standard für mobile Telefone. GSM wurde entwickelt, um die analogen, miteinander inkompatiblen Systeme zu ersetzen. Der Standard ist heutzutage der am meisten verbreitete Mobilfunk Standard und wird regelmäßig, vor allem im Bereich der Datenübertragung, erweitert.

Ursprünglich unterstützte GSM nur eine leitungsvermittelnde Datenübertragung (Circuit

Switched Data). Dabei wird jedem Benutzer ein eigener Kanal exklusiv zur Verfügung gestellt. Während dieses Vorgehen für Telefongespräche angemessen ist, stellt die Exklusivität einen unnötigen Ressourcenverbrauch bei der Datenübertragung dar.

Um nicht jedes mal einen exklusiven Kanal auf- und wieder abbauen zu müssen, wurde mit GPRS (General Packet Radio Service) eine paketvermittelnde Datenübertragung (Packet Switched Data) als Zusatz zum GSM Standard eingeführt. Die erreichbare Übertragung hängt von der Anzahl an zugeteilten Zeitschlitzen (Timeslots) und dem gewählten Kodierungsverfahren ab. Das Kodierungsverfahren wird je nach Qualität des Kanals zur Basisstation gewählt. Zeitschlitze bekommt man anhand der vorherrschenden Netzauslastung zugewiesen. Bei der mittleren Kodierung CS-2 erreicht man pro Zeitschlitz 10 kbit/s. Üblicherweise bekommt man 3-4 Zeitschlitze zugeteilt. Somit lässt sich mit GRPRS eine Übertragungsrate von 30-40 kbit/s erreichen.

Mit EDGE (Enhanced Data Rates for GSM Evolution) steht eine Erweiterung der GPRS Technik zur Verfügung. Diese Erweiterung beschränkt sich auf die Luft-Schnittstelle der Mobilstationen und bietet somit eine deutliche Verbesserung des Datendurchsatzes bei geringen Kosten für den Netzbetreiber. Bei 3-4 Zeitschlitzen und dem mittleren Kodierungsverfahren MCS-6 erreicht man Übertragungsraten von 75-100 kbit/s.

UMTS (Universal Mobile Telecommunications System) ist ein Standard der so genannten dritten Generation (3G). Verabschiedet von der Internationale Fernmeldeunion (ITU) soll der Standard hauptsächlich Breitbandangeboten dienen. Um UMTS anbieten zu können muss der Mobilfunkbetreiber neue Basisstationen einrichten. Aus diesem Grund ist gerade in ländlichen Gebieten die UMTS Netzabdeckung unvollständig. In städtischen Gebieten kann UMTS eine Datenrate von 384 kbit/s erreichen. Wobei der Netzanbieter diesen Wert beliebig wählen kann, um unterschiedliche Tarifmodelle anbieten zu können.

2.2.2 Streaming Protokolle

Netzwerkprotkolle sind entworfen und standardisiert worden , um die Kommunikation zwischen Netzwerkteilnehmern festzulegen. Man kann die Protokolle nach ihrer Funktionalität klassifizieren.

Im Zusammenhang mit Streaming Media sind zwei Kategorien besonders interessant:

- Transport Protokolle: User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Real-Time Transport Protocol (RTP), Real-Time Control Protocol (RTCP)
- Session Control: Real-Time Streaming Protocol (RTSP), Hypertext Transfer Protocol (HTTP)

UDP und TCP werden auch als Lower-Layer Transport Protokolle bezeichnet und RTP und RTCP als Upper-Layer Transport Protokolle die über UDP/TCP implementiert werden.Schulzrinne u. a. (2003)

Die UDP und TCP Protokolle ermöglichen rudimentäre Transport Funktionen wie Bündelung, Fehlerüberwachung, Stauvermeidung und Flusskontrolle. Diese Funktionen können folgendermaßen zusammengefasst werden.Tanenbaum (2003) Zuerst einmal können UDP und TCP verschiedene Datenströme verschiedener Applikationen der gleichen Quelle zusammen bündeln. Darüber hinaus wird von TCP und den meisten UDP Umsetzungen eine Checksumme zum Zwecke der Überwachung und Vermeidung von Bit Fehlern implementiert. Tritt so ein Fehler auf, wird das Paket nicht an die nächste Schicht (z.B. RTP) weitergegeben, sondern verworfen. Anders als bei UDP werden bei TCP verlorene Pakete neu versendet. TCP bietet also im Gegensatz zu UDP eine verlässliche Verbindung. Des Weiteren wendet TCP Stauvermeidung an um eine Überlastung der Verbindung zu vermeiden. Dieses ist ein weiteres Merkmal das TCP von UDP unterscheidet. Als Letztes wird von TCP eine Flusskontrolle angewendet, um zu verhindern, dass der Empfangspuffer des Empfängers überläuft. UDP hat keine Möglichkeiten der Flusskontrolle.

Das erneute Versenden von Paketen bei TCP kann Verzögerungen verursachen die für Streaming Anwendungen mit strengen Echtzeitbedingungen nicht akzeptabel sind. Idealerweise wird deswegen UDP als Transport Protokoll für Mediendaten benutzt. Da UDP die Paketzustellung nicht garantieren kann, muss ein Protokoll der nächsten Schicht (z.B. RTP) verlorene Pakete erkennen können.

2.2.2.1 Progressive Streaming über HTTP

Die einfachste Art der Medienlieferung ist ein Download von einem Webserver. Ein Webserver wie z.B. Apache benutzt dabei das HTTP Protokoll über TCP. Eine vorher archivierte Datei wird von dem Empfänger angefordert und so schnell geladen, wie es das System und das Netzwerk erlauben. Ist die Datei gespeichert, kann sie abgespielt werden. Abbildung 2.4 zeiget einen typischen Ablauf.

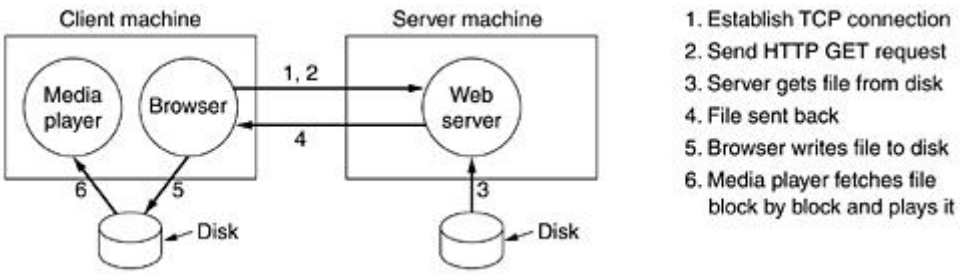

Abbildung 2.4: Ablauf eines HTTP Downloads

Anstatt nun die Datei erst zu speichern, kann man die bisher empfangenen Daten auch direkt abspielen. Diese Technik wird als Progressive Streaming bezeichnet. Progressive Streaming kann man als Zwischenschritt zwischen Download und Echtzeit Streaming betrachten. Der Streamingserver ist in diesem Fall nichts weiter als ein Fileserver. Voraussetzung für eine

vorzeitige Wiedergabe ist ein Fileformat, dass zwei besondere Eigenschaften erfüllt. Erstens müssen die Ton und Audiodaten gebündelt vorliegen. Das heißt, dass Teile der Ton- und Videodaten jeweils hintereinander liegen. Würden diese Daten getrennt gespeichert, müsste man die Datei erst bis zu der Stelle laden an der der zweite Stream beginnt. Zweitens wird verlangt, dass benötigte Metainformationen im Header gespeichert werden und nicht erst am Ende der Datei. Die Metadaten werden von dem Media Player benutzt, um sich auf die Eigenschaften der Präsentation einzustellen.

Progressive Streaming über HTTP ist heutzutage eine beliebte Methode um On-Demand Medieninhalte ohne viel Aufwand an Empfänger im Festnetz zu verteilen. Der Aufwand ist gering, da ein Webserver meist eh schon vorhanden ist. Videoportale wie Youtube.com oder Clipfish.de verwenden diese Technik. Im mobilen Bereich gibt es aber zusätzliche Einschränkungen die sich negativ auf eine Unterbrechungsfreie Wiedergabe auswirken können.

Anders als beim Echtzeit Streaming hat die Bitrate des Streams keinen direkten Einfluss auf die Übertragungsrate. Wenn der Puffer des Empfängers entsprechend groß ist, kann auch mit einer kleinen Übertragungsrate ein Progressive Streaming betrieben werden. Die Verzögerung zwischen Übertragungsstart und Wiedergabestart vergrößert sich dabei entsprechend dem Unterschied zwischen Übertragungsrate und Bitrate. Smartphones aber haben üblicherweise nur einen sehr begrenzten Speicher. Dadurch kann die Größe des Empfangspuffers wesentlich kleiner sein als die Gesamtgröße der Mediendatei. Um nun eine Präsentation unterbrechungsfrei abspielen zu können, muss die Übertragungsgeschwindigkeit die Bitrate mindestens übertreffen. Wie beim Download werden beim Progressive Streaming die Daten möglichst schnell über TCP versandt. Ist die Übertragung fehlerfrei, wird ein Überlaufen des Empfangspuffers durch die TCP Flusskontrolle begrenzt.

Besonders negativ wirkt sich sich die Neuversendung verlorener Pakete beim TCP Protokoll aus. Diese Prozedur bedeutet immer eine Unterbrechung des Datenstroms. Ist die Dauer solcher Fehler geringer als die Wiedergabedauer der im Empfangspuffer vorgehaltenen Daten, kann eine erhöhte verfügbare Bandbreite Wiedergabeunterbrechungen auffangen, indem der Empfangspuffer nach dem Fehler wieder aufgefüllt wird. Bei Festnetzverbindungen sind im Normalfall solche Unterbrechungen nur von kurzer Dauer. Im Mobilfunkbereich dagegen treten solche Fehler vor allem bei Wechseln zwischen Basisstationen auf. Kann der Empfangspuffer diese Unterbrechungen nicht mehr ausgleichen, kann man nur noch die Bitrate des Medienstreams verringern, um die effektive Dauer der im Puffer vorgehaltenen Daten zu erhöhen.

Ein weiterer Nachteil sind Einschränkungen in der Verfügbarkeit von typischen Videorekorder Funktionen. Während ein Pausieren der Wiedergabe dank der TCP Flusskontrolle bei gefülltem Empfangspuffer kein Problem darstellt, ist ein Vor- und Zurückspulen nicht ohne weiteres möglich. Im Desktopbereich kann in der Regel die ganze Mediendatei zwischengespeichert werden. In dem Fall ist ein Zurückspulen jederzeit möglich. Außerdem kann mindestens bis zu dem Punkt, der schon übertragen wurde, vorgespult werden, falls die Übertragungsgeschwindigkeit die Wiedergabegeschwindigkeit deutlich übertrifft. Im mobilen

Bereich sind die Rahmenbedingungen etwas anders. Abgesehen von stärker begrenztem Speicher, der ein vollständiges Zwischenspeichern der Datei möglicherweise verhindert, gibt es auch Unterschiede bei den Kosten. Heutzutage sind Datentarife im mobilen Bereich zum Großteil volumenbasiert. Möglicherweise möchte der Benutzer gar nicht die ganze Präsentation betrachten. Ein vorzeitiges Puffern der ganzen Datei könnte so zu unnötigen Kosten führen. Sollte der Arbeitsspeicher des Smartphones nicht ausreichen, könnte man zu mindestens ein Zurückspulen ermöglichen, indem man die schon betrachteten Teile auf einer üblicherweise vorhandenen Memory Karte zwischenspeichert.

2.2.2.2 Echtzeit Streaming über RTP

RTP ist ein Internet Transport Protokoll, das speziell für den Transport von Echtzeitdaten konzipiert wurde. Echtzeit bedeutet in diesem Zusammenhang, dass die benutzte Bandbreite der Bitrate des Streams angepasst wird. Um einen robusten Rahmen für den Transport solcher Daten zu schaffen, wurde vor allem das Prinzip des Application-Level Framing verfolgt.Perkins (2003)

Die Idee hinter Application-Level Framing besagt, dass eigentlich nur die Applikation ausreichendes Wissen über die zu versendenden Daten hat und deswegen auch die Paketeinteilung bestimmen sollte. Es hängt von der Art der Daten ab wie ein Übertragungsfehler sicher behandelt werden kann. Man kann sich das bei Videodaten folgendermaßen vereinfacht vorstellen: Videostreams sind in Frames eingeteilt. Beinhaltet nun ein Datenpaket den hinteren Teil eines Videoframes und den ersten Teil des folgenden Videoframes, sind bei einem Paketverlust direkt beide Frames unvollständig. Besser wäre es deswegen, die Paketgrößen an die Framegrenzen anzupassen.

Ein weiterer Designgedanke hinter RTP beschäftigt sich mit der Verantwortlichkeit für sicheren Transport. Unter TCP/IP übernimmt der TCP Stack die Verantwortung, indem verlorene Pakete erkannt und neu versendet werden. Diese Art der Fehlerbehandlung ist aber nicht für jede Anwendung optimal. Gerade im Bereich des Multimedia Streaming sind andere Möglichkeiten besser geeignet. Dieses Prinzipien widerspricht dem Design von TCP. Auch wenn der RTP Standard explizit kein Transportprotokoll vorschreibt, ist UDP eine weitaus bessere Wahl als unterliegendes Transportprotokoll. RTP verwendet verschiedene Techniken, um den Applikationen Möglichkeiten zu geben, die Unzuverlässigkeit des Netzwerkes auszugleichen, auf die später eingegangen werden.

Dem unterliegenden Transportprotokoll werden durch den RTP Header vor allem folgende Informationen hinzugefügt:

- Sequence Numbering: Beim Transport über ein unzuverlässiges Protokoll wie UDP wird nicht garantiert, dass jedes versendete Paket den Empfänger erreicht. Pakete können verloren gehen oder die Reihenfolge kann sich ändern. Damit der Empfänger

dies erkennen kann, werden RTP Paketen eine fortlaufende Sequenznummer hinzugefügt.

- Payload Type ID: Videopräsentationen bestehen normalerweise aus Video- und Tondaten. Über RTP wird jeder Nutzdatentyp in einem eigenen Stream versendet. Die ID gibt dem Empfänger die Möglichkeit den Typ richtig einzuordnen.

- Time-Stamping: Der Time-Stamp Header ist ein 32-bit Feld, dass den genauen Zeitpunkt des ersten Octets der Nutzdaten angibt. Vom Server aus gesehen ist dies der Zeitpunkt zu dem das Octet gesamplet wurde. Für den Client stellt der Wert den Zeitpunkt dar, an dem die Daten präsentiert werden sollen.

Das Real-Time Control Protocol gehört fest zum RTP Standard. Es dient hauptsächlich der Qualitätskontrolle. Dabei senden alle Teilnehmer einer RTP Sitzung regelmäßige Berichte über die Verbindungsqualität. Besonders interessant sind dabei die Berichte des Empfängers. Sie enthalten Statistiken über die Anzahl an verloren gegangener Pakete, Schwankungen des Paketempfangs (Jitter) und vergangener Zeit seit des letzten Senderberichtes. Der Sender kann diese Informationen Nutzen, um Probleme in der Netzwerkverbindung zu bestimmen.

Damit RTP vielseitig eingesetzt werden kann, ist die Spezifikation mit Absicht unvollständig und benötigt zusätzlich für eine funktionierende Implementation ein Profil und eine Nutzdaten Spezifikation. Für Medienstreaming ist dabei das „RTP Profile for Audio and Video Conferences with Minimal Control“ relevant. Unter anderem werden dort die Benutzung des RTCP Protokolls genauer festgelegt und die Nutzdaten ID im RTP Header mit einer entsprechenden Nutzdaten Spezifikation verknüpft. Nutzdaten Spezifikationen beschreiben detailliert die Versendung eines bestimmten Datentyps mit RTP Paketen. Es wird bestimmt welche Informationen in den RTP Headern benötigt werden und wie die Paketgrenzen bestimmt werden.

Durch den Echtzeitcharakter und die RTP Designprinzipien werden zwei wesentliche Unterschiede zum Progressive Streaming deutlich: Übertragungsgeschwindigkeit und Transportsicherheit.

Progressive Streaming über HTTP versendet die Daten mit der maximal möglichen Geschwindigkeit bis entweder der Empfangspuffer überläuft oder die Präsentation bzw. die Verbindung beendet wird. Einfluss auf die maximal verfügbare Geschwindigkeit hat dabei die TCP Stauvermeidung, die eine Überlastung der Netzwerkverbindung verhindert. Echtzeitstreaming über UDP dagegen versendet die Daten mit der Geschwindigkeit die durch die Bitrate des Medienstreams bestimmt wird. Eine automatische Stauvermeidung durch das Transportprotokoll ist nicht vorgesehen. Mit Hilfe der Rückmeldung des Empfängers über RTCP kann die Sendeapplikation aber einen Stau erkennen und die Bitrate des Medienstreams und somit die Transportgeschwindigkeit verringern. Streamingserver halten dazu

üblicherweise mehrere Bitratenversionen einer Präsentation bereit, zwischen denen umgeschaltet werden kann. Ist eine Verminderung der Bitrate trotz Staus nicht möglich, kann der Server die Verbindung nur noch unterbrechen.

Die Transportsicherheit beim TCP Protokoll verhindert, dass Pakete verloren gehen oder in falscher Reihenfolge bei der Empfangsanwendung eintreffen. Diese Sicherheit ist unter UDP nicht gegeben. Mit Hilfe der RTP Sequenznummerierung kann der Empfänger diese Probleme erkennen. Zur Behandlung einer falschen Reihenfolge steht dem Empfänger üblicherweise ein Empfangspuffer zur Verfügung der die Wiedergabe verzögert. Sind die Verschiebungen in der Paketreihenfolge geringer als die Puffergröße, kann die Anordnung vor der Ausgabe noch korrigiert werden. Tritt das Paket zu spät ein, gilt es als verloren. Für Übertragungen von z.B. Textmedien sind verlorene Daten nicht akzeptierbar. Bei Videoanwendungen dagegen können Lücken im Stream bis zu einem bestimmten Grad mit mehr oder weniger starken Artefakten in der Präsentation ausgeglichen werden. Es gibt vielfältige Techniken zur Behandlung von Packetloss die auch Teilweise sehr stark von dem benutzten Datentyp abhängen. Abgesehen vom wiederholten Versenden von verlorenen Paketen, kann man grundsätzlich drei Techniken unterscheiden:

- Vorwärtsfehlerkorrektur(FEC): FEC fügt den Mediendaten redundante Informationen hinzu, die von dem Dekoder benutzt werden können um fehlende Daten auszugleichen.Wang u. a. (1997) Dieses Verfahren kann sich aber auch Negativ auswirken, da Packetloss ja meist durch einen Datenstau entsteht und sich die zusätzlichen FEC-Daten deswegen eher Kontraproduktiv auswirken.

- Error-Resilient Encoding: Hier ist das Ziel die Erhöhung der Robustheit gegenüber Packetloss. Dem Empfänger soll es leichter gemacht werden Fehler zu verdecken. Interleaving von Audiostreams ist ein einfaches Beispiel dieser Technik. Ein Audiopaket besteht aus mehreren Samples. Geht ein Paket verloren, sind mehrere aufeinander folgende Samples verloren. Da aber einzelne fehlende Samples einfacher vom Audiodekoder zu verstecken sind, mischt man beim Interleaving Samples mehrerer aufeinander folgender Pakete abwechselnd.

- Fehlerverdeckung: Diese Technik wird ausschließlich vom Empfänger angewendet. Fehlt ein Paket, muss der Empfänger versuchen diese Lücke zu schließen. Dabei kann der Empfänger versuchen zwischen dem vorigen und dem nachfolgenden Paket zu interpolieren.

Zusammenfassend kann gesagt werden, dass sich die Präsentationsqualität bei Progressive Streaming nicht ändert. Können Transportschwierigkeiten durch den Empfangspuffer nicht mehr ausgeglichen werden, kommt es zu Unterbrechungen. Beim Echtzeitstreaming dagegen passt sich die Präsentationsqualität langsam den Übertragungsvorraussetzungen an. Unterbrechungen werden möglichst vermieden. Dieses dynamische Anpassen erhöht

aber auch die Komplexität der Wiedergabeanwendung und erhöht somit die benötigte CPU Leistung und den Stromverbrauch.

Um ein Echtzeitstreamingsystem zu vervollständigen werden noch Methoden benötigt, um eine Streaming Sitzung aufzubauen und zu kontrollieren. Das RTSP Protokoll erfüllt diese Aufgabe.Schulzrinne u. a. (1998) Der Zusammenhang der Echtzeitstreaming Protokolle wird in Abbildung 2.5 dargestellt.

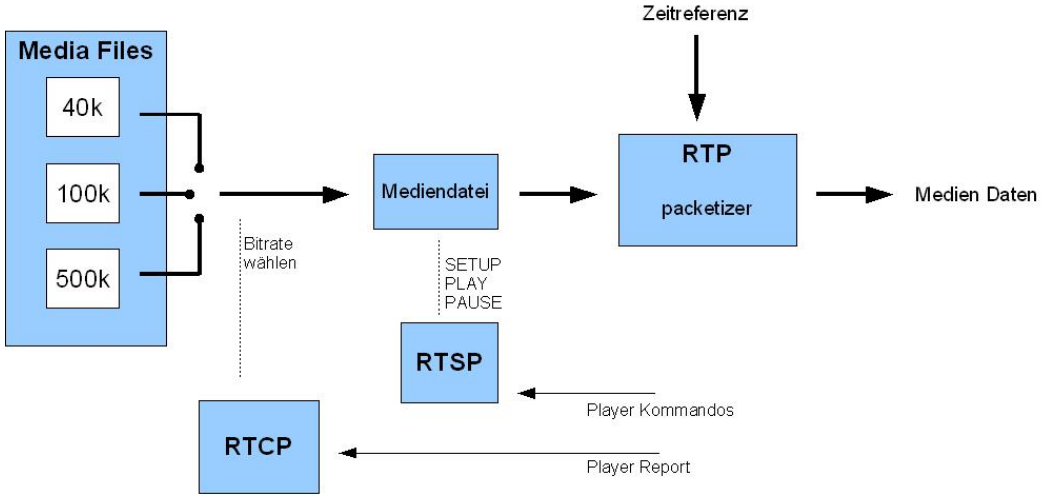

Abbildung 2.5: Zusammenhang der Echtzeitstreaming Protokolle

RTSP ist ein textbasiertes Protokoll mit einem HTTP ähnlichen Aufbau. Im Unterschied zu HTTP ist es aber Zustandsbehaftet. RTSP kann sowohl über UDP als auch über TCP transportiert werden. Aufgrund der Transportsicherheit und dem zustandsbehafteten Charakter wird aber üblicherweise TCP benutzt. Ähnlich wie bei HTTP werden bei RTSP ebenfalls Request- und Response-Nachrichten verwendet. Die RTSP Sitzung kümmert sich dabei um den Aufbau der RTP-Sitzung, indem zwischen dem Server und dem Client alle Verbindungsparameter festgelegt werden. Durch zusätzliche Kontrollmechanismen können Viderekorder Funktionen wie Play, Pause und Vor- und Zurückspulen realisiert werden.

Folgende RTSP Requests werden während einer RTSP Sitzung benötigt:

- DESCRIBE: Mit diesem Request fordert der Client vom Server eine Beschreibung einer Präsentation an. Der Request beinhaltet eine RTSP URL (rtsp://) um die Präsentation zu identifizieren. Die Antwort des Servers enthält üblicherweise eine Beschreibung im SDP FormatHandley u. a. (2006) und listet die in der Präsentation enthaltenen Streams. Typischerweise gibt es einen Video und einen Audiostream.

- SETUP: Dieser Request wird vom Client für jeden benötigten Stream gesendet und enthält neben der Präsentations URL und der Stream ID diverse Transporteinstellungen. Diese Einstellungen legen das Transportprotokoll fest und einen Port auf dem der

Client den RTP Stream erwartet. Ein zweiter Port wird übertragen, auf dem der Client die RTCP Daten für diesen Stream erwartet. In der Antwort des Servers werden diese Einstellungen im Idealfall bestätigt und die fehlenden Parameter, also die Ports auf Seiten des Servers, übermittelt. Außerdem wird eine Session ID festgelegt, die diesen Stream bei nachfolgenden Requests identifiziert.

- PLAY: Ein Play Request startet die RTP Übertragung eines Streams. Es kann zusätzlich ein Abspielbereich angegeben werden, wodurch Vor- und Zurückspulen ermöglicht wird.
- PAUSE: Pausiert einen RTP Stream. Es kann zusätzlich ein Bereich angegeben werden ab dem Pausiert werden soll.
- TEARDOWN: Beendet den RTP Stream und gibt alle Ressourcen frei die dieser Session angehören.

2.2.3 Network Address Translation

Damit ein Media Player im Internet mit einem Server kommunizieren kann benötigt er eine IP Adresse. Eine IP Adresse ist eine eindeutige 32-bit Nummer. Somit gibt es theoretisch 4,294,967,296 verschiedene IP Adressen. Durch die Art und Weise wie IP Adressen klassifiziert werden, ist die tatsächlich verfügbare Anzahl noch weitaus geringer. Mit der steigenden Anzahl an Geräten mit Internetfunktionen ist diese verfügbare Anzahl heutzutage nicht mehr ausreichend. Als Langzeitlösung wurde deswegen IPv6 mit einem 128-bit Adressschema entwickelt. Eine Einführung von IPv6 wird sich aber noch um Jahre verzögern, da Modifikation an der gesamten Infrastruktur notwendig sind. Als Alternative hat sich heutzutage Network Address Translation durchgesetzt.

Ein NAT System erlaubt es einem Gerät, z.B. einem Router, als Agent zwischen Internet und dem lokalen privaten Netzwerk zu agieren. Das bedeutet, dass nur eine eindeutige IP benötigt wird, um das lokale Netzwerk zu repräsentieren. Dabei ersetzt der Router die Quelladressen der ausgehenden Pakete mit der öffentlichen Adresse des Routers. In der Gegenrichtung wird dagegen die Zieladresse eingehender Pakete auf die lokale Adresse des Teilnehmers umgewandelt. Gibt es in dem lokalen Netz nur einen aktiven Teilnehmer, reicht dieses Verfahren aus, damit auch eingehende Pakete dem lokalen Teilnehmer eindeutig zugeordnet werden können. Soll das NAT-System aber mehrere aktive lokale IP Adressen repräsentieren wird zusätzlich eine Portumwandlung betrieben. Neben der IP-Adresse wird dabei der Quellport ausgehender Pakete dynamisch auf einen freien externen Quellport geändert. Zusätzlich merkt sich der NAT Router welcher lokale Teilnehmer auf welchem Port aktiv ist, damit später eingehender Verkehr eindeutig zugeordnet werden kann. Wichtig zu beachten hierbei ist, dass nur eine ausgehende Verbindung diese Verknüpfung im Speicher

des NAT Routers anlegen kann. Es gibt verschiedene Implementationen wie diese Verknüpfung im Detail angewendet wird. Man kann einen NAT Router deswegen in vier Kategorien klassifizierenRosenberg u. a. (2003):

- Full Cone: Hier werden alle Pakete einer internen IP Adress/Port Kombination dem gleichen externen Port zugeordnet. Jedes eingehende Paket auf dem externen Port wird dem lokalen Teilnehmer zugeordnet.
- Restricted Cone: Auch hier werden alle Pakete einer internen IP Adress/Port Kombination dem gleichen externen Port zugeordnet. Aber anders als bei der Full Cone Implementierung werden eingehende Pakete eines öffentlichen Internetteilnehmers mit der Adresse X nur zugelassen, wenn der interne Host vorher schon mal ein Paket an Adresse X gesendet hatte.
- Port Restricted Cone: Zusätzlich zu den Bedingungen der Restricted Cone Variante werden hier nur eingehende Pakete mit Quelladresse X und Quellport P zugelassen, wenn der interne Host vorher schon mal ein Paket an Adresse X und Port P gesendet hatte.
- Symmetric: Dieses Verfahren ist sehr restriktiv. Jedes ausgehende Paket mit einer bestimmten internen Adress/Port Kombination an ein bestimmtes Ziel mit einer bestimmten Adress/Port Kombination wird einer bestimmten externen Adress/Port Kombination des Routers zugewiesen. Sendet der gleiche interne Host ein Paket mit dem gleichen Quellport an eine andere Zieladresse, wird eine neue Verknüpfung angelegt. Somit kann nur der Empfänger der Ziel eines Paketes war über die gleichen Parameter antworten.

Ein NAT kann die Verfügbarkeit von Echtzeitstreaming mit RTSP und RTP verhindern. Dabei ist das Problem, dass bei dem Aufbau der Verbindung per RTSP die Port Daten der Teilnehmer auf Applikationsebene verhandelt werden. Befindet sich nun der Empfänger hinter einem NAT-Router, stimmt der vereinbarte Port nicht mit dem vom NAT-System zugewiesenen externen Port überein. Der RTP Stream des Servers wird somit später von dem Router abgewiesen.

Es gibt verschiedene Techniken, um diesem Problem zu begegnen:

- Simple Traversal of UDP Through Network Address Translation devices (STUN)
- Application Layer Gateway (ALG)
- Tunnel Techniques

Das STUN Protokoll schlägt einen speziellen STUN Server vor, der im öffentlich zugänglichen Adressraum operiert. Ein STUN fähiger Media Player sendet zuerst eine spezielle Nachricht an diesen Server mit dem gleichen Quellport der auch später für den Medienstream benutzt werden soll. Der STUN Server inspiziert anschließend die externe IP und den externen Port des empfangenen Paketes und meldet diese Daten zurück an den Client. Dieser wird dann die externen Portdaten bei der späteren RTSP Verhandlung mit dem Streamingserver verwenden. Abgesehen davon, dass man einen speziellen Server bereitstellen muss, hat dieses Vorgehen zwei Probleme. Zuerst muss der Medienplayer das STUN Protokoll unterstützen. Der zusätzliche Aufwand macht dieses Verfahren daher unpopulär. Zweitens funktioniert dieses Verfahren nicht mit symmetrischen NATs. Da der Server eine andere Host IP benutzt, bekommt er auch eine andere Verknüpfung des NAT Routers zu sehen. Eine Erweiterung des STUN Verfahrens nennt sich TURN (Traversal Using Relay NAT). Neben den STUN Funktionen fungiert der TURN Server dabei zusätzlich als Proxy Server für den Stream. Anstatt des Medienplayers nimmt der TURN Server Kontakt mit dem Streamingserver auf und leitet den Stream anschließend weiter an den Client. Somit kommt der Stream von der gleichen Adresse die auch bei der NAT Erkennung benutzt wird, und ein symmetrisches NAT stellt kein Problem mehr da. Durch die erhöhte Bandbreite die der TURN Server bereitstellen muss, ist auch diese Methode unpopulär.

Unter ALG versteht man eine Erweiterung des NAT Routers. ALG Erweiterung befähigen den Router Pakete bestimmter Protokolle zu verstehen und den Anforderungen entsprechend zu modifizieren. Eine ALG Erweiterung für RTSP würde also die SETUP Requests des Clients dahingehend verändern, dass die vereinbarten Ports den externen Ports entsprechen. Die Einrichtung von einem ALG fähigen Router ist umfangreicher und darf nicht unterschätzt werden.

Es existieren verschiedene Tunneltechniken für RTP. Zwei Varianten sind dabei häufig anzutreffen:

- RTP over RTSPSchulzrinne u. a. (1998): Die Spezifikation von RTSP erlaubt es, RTP und RTCP Pakete der zugehörigen Präsentation in den RTSP Kanal einzubetten. Diese Technik erfordert erhöhten Aufwand seitens des Clients und des Servers und verursacht zusätzlichen Overhead. Die RTP und RTCP Pakete behalten dabei ihren Header. Dieses Vorgehen ist vor allem interessant, wenn ein Teil der Netzwerkverbindung nur TCP Verbindungen erlaubt, man aber später die Streams wieder auf UDP weiterleiten möchte.

- RTSP/RTP in HTTP: Diese Technik kann bei sehr restriktiven Firewalls benutzt werden die nur HTTP Verkehr zulassen. Dabei öffnet der Client zwei HTTP Verbindungen zu dem Streamingserver. Eine Verbindung benutzt GET Requests für eingehende Daten. Die andere Verbindung benutzt POST Requests für ausgehende Datenpakete. Der Server bildet aus diesen beiden Verbindungen eine virtuelle Vollduplex Verbindung über die RTP over RTSP angewandt werden kann.

Kapitel 3

Entwurf

3.1 Anforderungen

Wie schon beschrieben, ist dieses Projekt entstanden durch Schwierigkeiten beim abspielen online hinterlegter Medieninhalte über Symbian Smartphones. Der bei aktuellen Nokia Smartphones mitgelieferte RealPlayer kann Medieninhalte ausschließlich über RTP/UDP streamen.

Da alle Mobilfunkanbieter eine Form von Network Address Translation benutzen, ist man darauf angewiesen, dass der Anbieter besondere Vorkehrungen getroffen hat. Tests über verschiedene deutsche Netzbetreiber lieferten dabei aber sehr inkonsistente Ergebnisse. Die Hälfte der getesteten Netze konnte dabei keine Streamingsitzung aufbauen.

Obwohl das Symbian Betriebssystem die Entwicklung zusätzlicher Anwendungen fördert, sind zum Zeitpunkt dieser Arbeit keine alternativen Streaming Media Player für die jüngste Version verfügbar. Der Hauptgrund für diesen Umstand ist der „Binary Break“ zwischen S60v2 und S60v3. Eine grundlegende Umgestaltung des S60 Frameworks hat diese drastische Maßnahme verursacht. Zusätzlich zur grundlegenden Inkompatibilität der Anwendungen zwischen S60v2 und S60v3, wurden eine große Anzahl der Framework APIs überarbeitet. Diese Umstände erfordern einen hohen Aufwand, möchte man eine Anwendung auf die aktuelle Plattform migrieren.

Im Rahmen dieser Bachelorarbeit soll deswegen ein Konzept für einen eigenen Media Player entwickelt werden, der sich leicht mit der Virtualreel Software integrieren lässt. In den folgenden Abschnitten werde ich dafür auf die besonderen Voraussetzungen eingehen, die bei der Erstellung zu beachten sind.

3.1.1 Netzbetreiber

Vier verschiedene deutsche Netzbetreiber wurden im Vorfeld dieser Arbeit mit dem zum Nokia S60v3 Lieferumfang gehörenden RealPlayer auf Streamingkompatibilität getestet. Die

Netze von T-Mobile und o2 Germany machten dabei keine Probleme. Trotz vorhandener Network Address Translation wurden die Streamingsitzungen beim Aufbau über RTSP korrekt durch ein Application Layer Gateway wiedergegeben. Über E-Plus und Vistream dagegen war ein Abspielen nicht möglich.

E-Plus bietet zwar einen vollwertigen Internetzugangspunkt an, dennoch gelingt es dem RealPlayer nicht den Stream abzuspielen. Im Gegensatz zu der Desktop Version des RealPlayers beherrscht die S60 Variante neben reinem Echtzeitstreaming über RTP/UDP weder Tunneltechniken noch Progressive Streaming. Man kann diese Aussage überprüfen, indem man einen eigenen Streamingserver einrichtet und die Verbindungsversuche des Smartphones mit einem geeigneten Analysetool wie Wireshark am Server überwacht. Verbindet man anschließend das Telefon über USB mit einem Laptop, kann man die NAT Eigenschaften des Mobilzugang genauer untersuchen. Unter Linux bietet sich dazu das Kommandozeilentool netcat an. Für Windows gibt es mit dem UDP Test Tool ebenfalls eine geeignete freie Analysesoftware. Man wird feststellen das die NAT-Installation bei E-Plus symmetrische Eigenschaften besitzt.

Unter Vistream sieht die Situation noch schlechter aus. Vistream ist recht neu auf dem deutschen Markt und kann als virtueller Netzwerkbetreiber oder MVNE verstehen werden. Ein MVNE baut nur Teile einer vollwertigen Netzbetreiber Infrastruktur selbst auf. Bei Vistream bedeutet das konkret, dass die Basisstationen von E-Plus genutzt werden. Kern Infrastrukturelemente wie Abrechnugsserver und Internetzugangspunkte werden dagegen von Vistream gestellt. Auch wenn ein vollwertiger Internetzugangspunkt schon lange angekündigt ist, steht momentan nur ein so genannter WAP-Zugangspunkt zur Verfügung. Dieser lässt nur Verbindungen über einen Proxy Server zu. Und der Proxy Server erlaubt ausschließlich HTTP Verbindungen, wodurch jeder Versuch mit Echtzeitstreaming über RTP natürlich zum Scheitern verurteilt ist. Progressive Streaming dagegen ist ohne weiteres über das WAP Gateway möglich.

3.1.2 Voraussetzungen für eine Integration mit Virtualreel

Die Virtualreel Software stellt eine webbasierte Schnittstelle zu den erstellten Online-Reels zur Verfügung. Bei der Erstellung des Reels können dabei das gewünschte Fileformat und die verwendeten Komprimierungsformate gewählt werden. Zusätzlich kann die Webdarstellung durch ein Passwort gesichert werden. Abbildung 3.1 zeigt eine typische Darstellung eines Online-Reels. Ein Benutzer kann sich mit einem Webbrowser diese Übersicht anzeigen lassen und die Videos nacheinander betrachten, indem er die Videos auswählt.

In einer Situation in der kein Desktoprechner oder Laptop zur Verfügung steht, würde sich ein Smartphone als eine willkommene Alternative zur Betrachtung des Reels anbieten. Moderne Smartphones bieten mittlerweile ausreichend große und hochauflösende farbliche Displays. So kann man auch unterwegs auf die Schnelle das Video Portfolio anzeigen lassen. Für eine nahtlose Integration sollten die gleichen Schritte mit dem Smartphone möglich

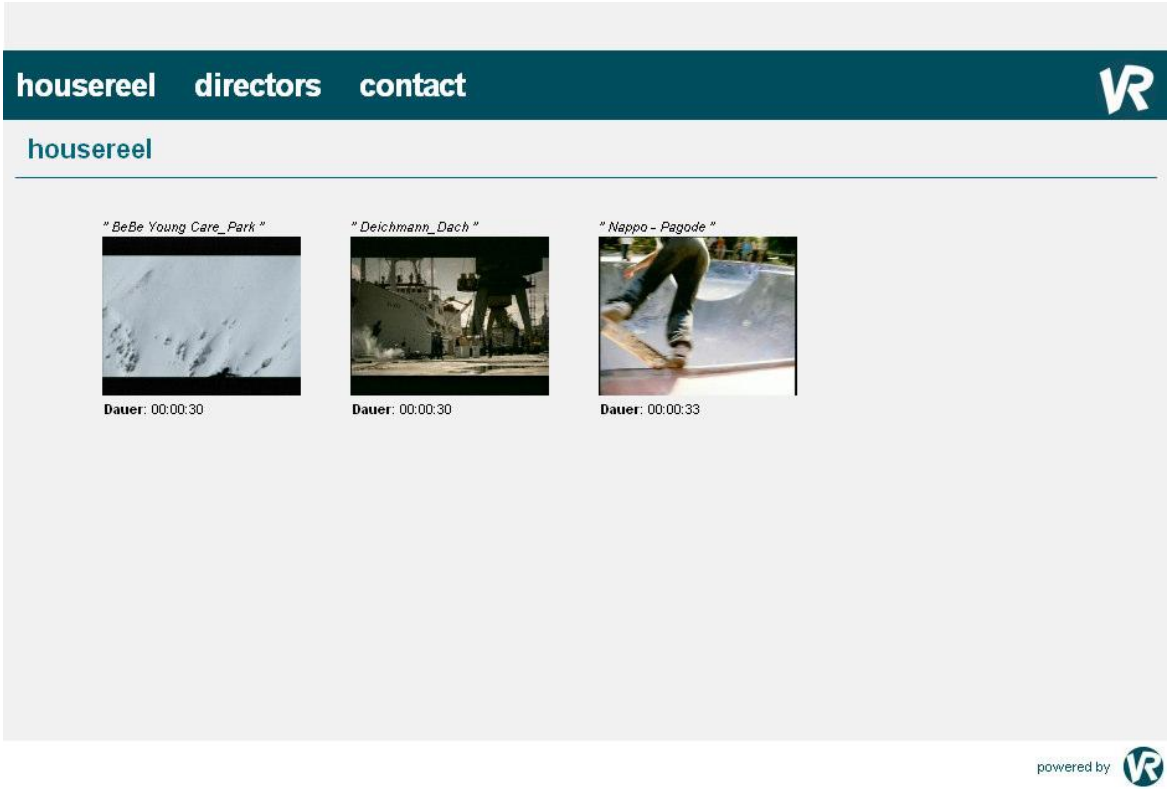

Abbildung 3.1: typisches Online-Reel

sein, die auch zur Betrachtung vom Desktop aus durchgeführt werden. Dazu wird eine Integration des Media Players mit einem Webbrowser verlangt, der die Online Reels darstellen kann. Die Integration bedeutet, dass bei der Auswahl eines Videos der Media Player automatisch gestartet wird und mit der Wiedergabe beginnt. Zum Lieferumfang von Nokia S60v3 Smartphones gehört ein Webbrowser der diese Eigenschaften erfüllt. Man kann außerdem eigene Dateitypen definieren, die anschließend eine bestimmt Applikation starten.

Multimediakompression ist ein sich ständig weiterentwickelnder Bereich. Verbesserte Kompressionsformate erhöhen die Wiedergabequalität bei geringerer Bitrate. Die Virtualreel Software trägt diesem Umstand Rechnung, indem sich leicht neue Kompressionsformate integrieren lassen. Ein Media Player für so ein System sollte sich ebenfalls leicht um neue Formate erweitern lassen. Dazu müssen entsprechende Schnittstellen bei der Implementierung vorgesehen werden.

Die Virtualreel Software beinhaltet einen Webserver aber keinen Streamingserver. Eine Lösung die keine zusätzlichen Komponenten bei dem System erfordert, würde die Integration deutlich vereinfachen.

Ein Online-Reel stellt die künstlerischen Fähigkeiten eines Artisten dar. Üblicherweise werden sie zu Promotionszwecken vorgeführt. Auch auf einem mobilen Gerät sollte die Dar-

stellungsqualität beim Abspielen möglichst hoch sein und der Qualität des Originals entsprechen. Artefakte oder Qualitätsänderungen während der Wiedergabe sollten vermieden werden.

3.1.3 Besondere Voraussetzungen in mobiler Umgebung

Bei der Entwicklung einer mobilen Anwendung sind besondere Rahmenbedingungen zu beachten. Besonderen Wert muss dabei auf die Einschränkungen eingegangen werden die durch die limitierte Hardware entstehen. Mobile Hardware soll klein und handlich sein. Das hat insbesondere Auswirkungen auf die Größe des Bildschirms. Aktuelle Nokia Smartphones bieten im Regelfall eine Auflösung von 240*320. Bei der Darstellung eines Videos sollte möglichst der gesamte Bildschirm ausgenutzt werden können und besondere UI Elemente sollten sich leicht verstecken lassen. Unter S60 bieten sich dazu Menü-Elemente an die durch betätigen der Softkeys geöffnet werden und sich dabei über die Applikation legen. Hat man eine Auswahl getätigt, schließt sich der Menü-Rahmen wieder und gibt den Bildschirm frei. Zusätzlich sollten Funktionen die den Abspielvorgang begleiten, unsichtbar über die Tastatur ausführbar gestaltet werden. Das Ändern der Lautstärke z.B. könnte man durch Betätigung der Richtungstasten modifizieren.

Mobiltelefone sind meist zur einhändigen Bedienung konzipiert. Dabei liegt der Daumen im oberen Bereich der Tastatur und erreicht besonders gut die beiden Softkeys und die Navigation. Abbildung 3.2 verdeutlicht die Anordnung. Durch diese Ausrichtung haben sich bestimmte Grundlagen bei der Bedienung von S60 Applikationen durchgesetzt. Um dem Benutzer einen schnellen Einstieg zu erleichtern, sollten diese Prinzipien ebenfalls angewendet werden. Einerseits sollten dazu alle Funktionen des Medien Players ausschließlich über die Softkeys und die Navigation erreichbar sein. Andererseits sollte dem Konzept gefolgt werden, dass der linke Softkey eine Funktion aktiviert und der rechte Softkey eine Funktion abbricht bzw. beendet. Befindet sich der Media Player z.B. im Abspielmodus sollte der rechte Softkey die Wiedergabe jederzeit stoppen und beenden und der linke Softkey zwischen Pause- und Playmodus wechseln.

Zukünftige Smartphones werden sich von jetzigen Smartphones in vielerlei Hinsicht unterscheiden. Änderungen beim Bedienungskonzept sind nicht zuletzt seit der Popularität von Touchscreen Handys zu erwarten. Kurzfristig wird die kommende Version von Nokias S60 Framework einen dritten Softkey einführen. Eine wichtige Anforderung an eine Applikation ist also eine strikte Trennung zwischen der Engine und der UI.

Der RAM Speicher, der ausführbaren Applikationen zur Verfügung steht, ist begrenzt. Oft haben Smartphonebenutzer zusätzliche Applikationen installiert die im Hintergrund laufen. Man kann deswegen nicht davon ausgehen, genügend Speicher für einen großen Empfangspuffer zur Verfügung zu haben. Deswegen sollte sich der Empfangspuffer auch nicht während der Übertragung dynamisch vergrößern, sondern auf einen festen dem Abspielformat angepassten Wert begrenzen. Bei der Programmierung der Anwendung ist außerdem

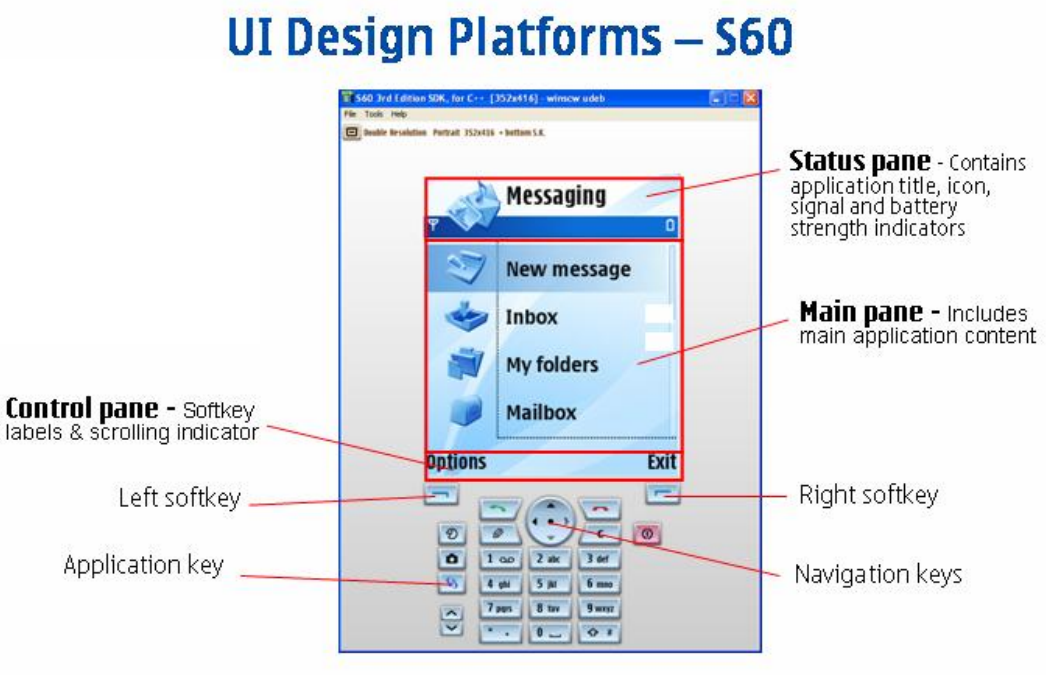

Abbildung 3.2: Anordnug der Bedienschnittstelle

drauf zu achten unnötige Zwischenpuffer und überflüssige Kopiervorgänge zu vermeiden. Wenn möglich sind Zeiger bzw. Referenzen einer Deep-Copy vorzuziehen. Das gleiche gilt für Betriebssystemressourcen die unter Symbian OS meist über ein Client/Server Framework zur Verfügung gestellt werden. Nicht benötigte Ressourcen sollten dabei sobald wie möglich freigegeben werden. Beispielsweise möchte ein Benutzer nach beendeter Wiedergabe eine andere Anwendung öffnen aber behält den Media Player im Hintergrund geöffnet. Es wäre in der Situation überflüssig für die Playeranwendung die Ressourcen des Netzwerkkerns unnötig zu belegen.

Neben einer speichersparenden Programmierung sollte auch die CPU geschont werden. Man kann nicht davon ausgehen, dass jedes erhältliche und zukünftige S60 Smartphone eine Mindestleistung garantiert. Zusätzliche im Hintergrund betriebene Anwendungen, die je nach Benutzerverhalten unterschiedlicher Art sind, machen eine Einschätzung der verfügbaren Leistung unmöglich. Hinzu kommt der Stromverbrauch der gerade bei Smartphones einen kritischen Punkt darstellt. Aktuelle Smartphone CPUs bieten dabei Stromsparfunktionen die wirksam sind, sobald die CPU Belastung gering ist. Unter Symbian bietet sich dazu die Verwendung des Active-Object-Framework an, anstatt eine vollwertige multitask Anwendung zu erstellen. Dabei wird dem Smartphone sowohl die Möglichkeit gegeben die CPU abzuschalten sobald keine neuen Aufgaben anstehen, als auch werden teure Kontextwechsel vermieden. In dem gleichen Zusammenhang steht die Frage nach den gewählten

Kodierungsformaten. Einerseits ermöglicht ein komplexer Codec eine niedrigere Bitrate und somit eine geringere benötigte Bandbreite, andererseits werden auch höhere Ansprüche an die CPU Leitstung gestellt.

Ein Smartphone ist ein Multifunktionsgerät. Beim Abspielen eines Medienstreams bleibt die Telefoniebereitschaft des Gerätes immer bestehen. Deswegen muss man bei der Erstellung des Media Players auch beachten, dass jederzeit ein Telefongespräch ankommen kann. Um das Telefongespräch nicht zu stören sollte der Media Player mindestens pausiert werden. Dies sollte zudem automatisch geschehen, da der Benutzer sich bei einem eingehenden Anruf auf das anstehende Telefongespräch konzentriert und sich in der Eile nicht mit der Bedienung einer Applikation auseinander setzen möchte.

Der wahrscheinlich problematischste Bereich in einer mobilen Umgebung ist die sichere Übertragung des Streams. In vielen Teilen Deutschlands ist UMTS noch nicht verfügbar wodurch man auf (E)GPRS angewiesen ist. Hinzu kommt die undurchsichtige Situation der Tarifmodelle der unterschiedlichen Betreiber. Dabei gibt es unter Umständen Tarife die die Benutzung des UMTS Netzes nicht erlauben. Erwartungsgemäß werden sich die möglichen maximalen Transferraten der verschiedenen Benutzer der Software sehr stark unterscheiden. Hinzu kommt die Tatsache, dass sich die Netzqualität der Betreiber sehr stark unterscheiden kann und die versprochene maximale Geschwindigkeit in der Praxis nicht erreicht wird. Grundsätzlich ist allen Übertragungsstandards gemein, dass die effektive Datenrate von der Anzahl der Mitbenutzer einer Zelle abhängt. Wechselt man eine Zelle kann es passieren, dass sich die verfügbare Datenrate deutlich ändert. Diese Unterschiede treten sowohl örtlich als auch zeitlich auf. Die Anzahl an Mibenutzern ist sicherlich höher zu typischen Pendelzeiten als tief in der Nacht.

Das Wechseln einer Zelle hat noch weitere Auswirkungen. Ist man mit dem Mobiltelefon unterwegs treten solche Zellenwechsel je nach Reisegeschwindigkeit mehr oder weniger regelmäßig auf. Bei jedem Wechsel wird der Datenstream unweigerlich kurzzeitig unterbrochen. Das kann dazu führen, dass Pakete erst mit langer Verzögerung wieder den Empfänger erreichen.

Bei Echtzeit Streaming über UDP bedeutet das, dass bei jedem Wechsel einige Pakete verloren gehen werden. Der Streamingserver würde dann unter Umständen die Bitrate und somit die Videoqualität verringern. Stabilisiert sich die Verbindung nach dem Wechsel wieder für einen bestimmten Zeitraum, kann die Qualität wieder erhöht werden. Ein ständiges Wechseln der Wiedergabequalität fällt natürlich negativ auf und sollte wenn möglich verhindert werden.

Bei Progressive Streaming ist das TCP Protokoll der entscheidende Faktor. Durch die Transportsicherheit gehen zwar keine Pakete verloren, aber die Zeit bis der Player bei einem Zellenwechsel wieder Daten bekommt, ist größer. Das liegt vor allem an der TCP Stauvermeidung die zwar sehr schnell auf Staus reagiert und die Geschwindigkeit herabsetzt, anschließend aber nur sehr langsam wieder hoch regelt. Das ist bei regelmäßig auftretenden Kanaleinbrüchen, wie es in mobilen Netzen typisch ist, weniger gut. Diese Regelge-

schwindigkeit ist sehr stark abhängig von der Round-Trip-Time. (E)GPRS Verbindungen sind deswegen besonders anfällig für dieses Symptom. Für den Benutzer bedeutet das unter Umständen, dass die Wiedergabe unterwegs bei jedem Zellenwechsel unterbrochen wird, damit der Media Player den Stream neu zwischenspeichern kann.

Bei der Konzeption kann man diesen Effekten nur durch eine Vergrößerung des Empfangspuffers oder durch eine Verminderung der Bitrate entgegen wirken.

3.1.4 Zusammenfassung

Im folgenden möchte ich die bisher genannten Anforderungen zusammenfassen. Sie lassen sich dabei in zwei grundlegende Kategorien unterteilen:

Anforderungen an den Streamingmodus:

- Funktionalität in möglichst vielen Betreibernetzen
- einfache Integration mit bestehender Virtualreel Hardware
- möglichst geringe Qualitätsvarianz
- stabile Übertragung

Anforderungen an die Implementation:

- einfache Integration mit Webbrowser
- einfach um neue Formate erweiterbar
- Trennung Engine und UI
- einhändige Bedienung über Softkeys und Menüs
- Ressourcensparend
- automatisch Pause bei externen Events wie Telefonanruf

3.2 Architektur

3.2.1 Festlegung des Streamingmodus

Die Festlegung des Streaming-Modus ist einer der ersten Parameter die festgelegt werden müssen. Diese Wahl hat einen starken Einfluss auf die weitere Konzeption des Media Players. Der wichtigste Unterschied besteht dabei in der Form der empfangenen Daten. Während bei Echtzeitstreaming die Ton- und Videodaten getrennt in eigenen Datenströmen

übertragen werden, empfängt man Diese im Progressive Modus zusammen und der Media Player muss sich selbst um die saubere Trennung kümmern bevor die Daten weiterverarbeitet werden können. Bei Echtzeitstreaming hat der Streamingserver somit schon einen wichtigen Teil der Arbeit erledigt. Auf der anderen Seite ist die Implementation der drei am Echtzeitstreaming beteiligten Protokolle (RTP, RTCP, RTSP) um einiges aufwendiger als ein simpler Downloadstream über HTTP.

Im folgenden möchte ich die beiden Streamingarten auf die bisher genannten Anforderungen vergleichen:

Funktionalität in möglichst vielen Betreibernetzen

E-Plus benutzt zum Zeitpunkt der Anfertigung dieser Arbeit eine Network Address Translation mit symmetrischen Charakter. Möchte man Echtzeitstreaming betreiben, steht die Möglichkeit eines externen STUN Server somit nicht zur Verfügung. Tunneltechniken streamen die RTP Pakete über TCP wodurch sich der Transportvorteil gegenüber Progressive Streaming relativiert. Ein zusätzlicher Overhead und eine deutlich erhöhter Implementationsaufwand machen diese Variante eigentlich nur Interessant, wenn man die ursprüngliche Struktur der RTP Pakete später weiterverwenden will. Z.B. wenn man den Stream an einen anderen Teilnehmer im Originalzustand weiterleiten möchte. Die letzte Möglichkeit wäre eine TURN ähnliche Implementierung. Um einen dritten Server zu sparen, könnte man die TURN-Funktionalität direkt auf den Streamingserver einsetzen. Entweder als externe Applikation oder als Erweiterung eines Open-Source Streaming Servers wie z.B. Apple Darwin Streaming Server. Diese Möglichkeit würde aber den Umfang dieser Arbeit weit übersteigen. Progressive Streaming dagegen hat keine dieser Einschränkungen und würde die Funktionalität im Netz von E-Plus ermöglichen. Bei Vistream sieht die Situation noch einfacher aus. Da der WAP-Zugangspunkt ausschließlich Daten über ein HTTP-Proxy zulässt, bleiben Progressive Streaming oder ein RTP over HTTP Tunnel die einzigen Möglichkeiten.

Einfache Integration mit bestehender Virtualreel Hardware

Für Progressive Streaming müssen an der vorhandenen Hardware keine zusätzlichen Änderungen vorgenommen werden, da ein Webserver bereits zur Verfügung steht. Ein zusätzlicher Streaming Server dagegen, der für Echtzeitstreaming nötig wäre, würde die Server Installation nicht nur zusätzlich komplizieren, sondern würde unter Umständen zusätzliche Ressourcen benötigen und somit den Hardwarepreis einer Installation erhöhen.

Möglichst geringe Qualitätsvarianz

Progressive Streaming liefert das Video nur in einer Qualität aus. Kommt es zu Übertragungsproblemen die die Ausgleichsmöglichkeiten des Empfangspuffer übersteigen, entstehen bei der Wiedergabe Pausen. Bei Echtzeitstreaming hält man die Präsentation üblicherweise in mehreren Qualitätsstufen bereit. Dadurch kann sich die Wiedergabequalität während des Abspielens ständig ändern. Um dem zu begegnen, könnte man natürlich nur eine Qualitätsstufe anbieten um diesen Wechsel zu deaktivieren. Da aber bei einer Übertragung über das UDP Protokoll Pakete verloren gehen können und diese auch nicht wiederholt gesendet werden, würden bei solchen Übertragungsfehlern keine Pausen in der Wiederga-

be entstehen, sondern im schlechtesten Fall Lücken. Die Präsentation wäre in dem Fall nur unvollständig. Die Sichtbarkeit dieser negativen Effekte ist aber stark abhängig von dem verwendeten Codec. Moderne Codecs bieten spezielle Mechanismen wie FEC oder Error-Resilient Encoding (s. Kapitel x) um solche Artefakte gut zu verbergen. H.264 (MPEG-4 Part 10) ist ein solcher Videocodec der besondere Möglichkeiten für ein Streamen in fehleranfälligen Netzen bietet.

Stabile Übertragung

Progressive Streaming kann keine stabile Übertragung garantieren. Die Nachteile von TCP werden gerade in mobilen Netzen deutlich. Erwartungsgemäß wird ein besonders großer Empfangspuffer benötigt und die verwendete Bitrate sollte die durchschnittlich mögliche Übertragungsgeschwindigkeit deutlich unterschreiten. Ein Streamen unterwegs sollte ebenfalls vermieden werden. Echtzeitstreaming mit unterschiedlichen Qualitätsstufen oder einem Fehlerresistenten Codec kann dagegen viele Übertragungsprobleme mit mehr oder weniger starken Artefakten bei der Präsentation ausgleichen.

Es gibt noch weitere Aspekte die bisher nicht genannt wurden. Im Folgenden möchte ich diese kurz auf einen Einfluss auf dieses Projekt hin untersuchen:

Vor- und Zurückspulen

Videorekorder Fähigkeiten wie Vor- und Zurückspulen sind bei Progressive Streaming im Gegensatz zu Echtzeitstreaming nur unzureichend verfügbar. Da einzelne Werbeclips in dem Virtualreel System aber üblicherweise wenige Minuten nicht überschreiten, ist der Nachteil als eher gering zu bewerten.

Buffering Dauer

Da Progressive Streaming für eine stabile Übertragung höhere Ansprüche an einen Empfangspuffer stellt, ist die Zeit bis zum Start des Streams auch grundsätzlich höher als bei Echtzeitstreaming. Man kann diesen Effekt minimieren indem man die Wiedergabe schon bei unvollständig gefüllten Buffer beginnt und von einer hohen Anfangstransferrate ausgeht.

Copyright

Inhalte die auf einem Webserver hinterlegt sind, können auch grundsätzlich runter geladen werden. Das Kopieren der Inhalte wird so vereinfacht. Es gibt zwar im Grunde auch die Möglichkeit einen Echtzeitstream mit zuschneiden und später wieder in eine Mediendatei zu verwandeln, die Tatsache, das dann aber auch alle Übertragungsfehler mit geschnitten werden, macht diese Möglichkeit eher unpopulär. Die Online-Reels der Virtualreel Software können wahlweise durch ein Passwort geschützt werden. Daher spielt dieser Punkt keine große Rolle.

Live Streaming

Ein einfacher Webserver bietet grundsätzlich keine Möglichkeit ein Live Medium z.B. von einer Kamera über Progressive Streaming zu versenden. Es gibt aber Streamingserver Lösungen wie Shoutcast, die ein Live-Streaming auch über HTTP anbieten. Diese Funktionalität ist aber für das Virtualreel Projekt nicht relevant.

Zusammenfassung

Der wichtigste Punkt ist ohne Zweifel die Möglichkeit über möglichst viele Betreiber streamen zu können. Zusammen mit der Einfachheit der Integration mit der bestehenden Hardware ist Progressive Streaming die beste Wahl für eine Implementation. Ein weiterer angenehmer Nebeneffekt ist, dass der Virtualreel Server ebenfalls ohne besondere Probleme hinter einem NAT-Router betrieben werden kann. Gegebenfalls muss Port 80 fest auf den Virtualreel Server eingestellt werden, was bei einer Installation des Systems aber ohnehin eine Voraussetzung wäre.

3.2.2 Besonderheiten des Frameworks

Die S60 Plattform von Nokia befindet sich mittlerweile in der dritten Generation (S60v3). Dabei sieht das S60 Konzept zusätzliche Revisionen innerhalb einer Generation vor. Diese Revisionen werden Feature Packs genannt. Dabei wird die Plattform um neue Fähigkeiten erweitert. Diese werden aber nicht unbedingt von älteren Geräten der dritten Generation unterstützt. Das Testgerät ist kompatibel zum ersten Feature Pack (S60v3FP1). Um aber einen Media Player zu entwickeln der auch zu älteren Geräten der dritten Generation kompatibel ist, wurde entschieden die ursprüngliche SDK Umgebung zu verwenden.

Das S60 SDK liefert eine große Menge an APIs. Im Folgenden möchte ich auf die Besonderheiten der Entwicklungsumgebung eingehen die bei der Konzeption eine Rolle gespielt haben.

Nokia Smartphones bieten im Lieferumfang Unterstützung für verschiedene Codecs. Einige der Audio Codecs bieten eine zusätzliche Hardware unterstützte Beschleunigung. Eine speziell vorhandene DSP Einheit entlastet dabei den Hauptprozessor. Der Zugriff auf diese Codecs wird durch verschiedene öffentliche APIs realisiert. Dabei gibt es aber bedeutende Unterschiede zwischen dem Zugriff auf Audio Codecs und Video Codecs. Die VideoPlayerUtility API benutzt bedauerlicherweise die RealPlayer Engine. Das hat für den Entwickler einige Nachteile. Diese API unterstützt nämlich keine Möglichkeit einen Buffer an die Engine zu übergeben. Es gibt im Grunde nur drei Zugriffsarten. Man kann eine Datei des Filesystems öffnen, eine RTSP URL oder eine HTTP URL. Übergibt man eine RTSP Adresse öffnet die Engine den Stream im Echtzeitstreaming Modus mit allen Nachteilen des mitgelieferten RealPlayers. Öffnet man dagegen eine HTTP URL, wird die Datei erst komplett runter geladen und kann anschließend abgespielt werden. Progressive Streaming wird nicht unterstützt. Bei den Audio Codecs sieht die Situation anders aus. Mit der AudioOutputStream API hat man zumindestens direkten Zugriff auf die Hardware beschleunigten Audio Codecs und kann einen eigenen Buffer übergeben.

Die S60 Umgebung bietet verschiedene Möglichkeiten auf das Internet zuzugreifen. Neben direktem Zugriff auf die Socket API wird auch eine HTTP API angeboten die die Erstellung der benötigten Request Pakete und die Auswertung der empfangenen Response Pakete erleichtert.

Für unseren Anwendungsfall bedeuten diese Voraussetzungen folgendes Vorgehen der auch durch Abbildung 3.3 verdeutlicht wird:

1. Zugriff auf das Netzwerk geschieht mit Hilfe der HTTPClient API.
2. Die empfangenen File Daten werden mit einem eigenen Demultiplexer in Ton und Audio Daten aufgeteilt.
3. Tondaten werden über die AudioOutputStream API gerendert.
4. Videodaten müssen mit einem eigenen Video Codec über direkten Bildschirmzugriff dargestellt werden.
5. Synchronisation der beiden Streams wird durch die Geschwindigkeit der Abarbeitung durch die AudioStream API vorgegeben.

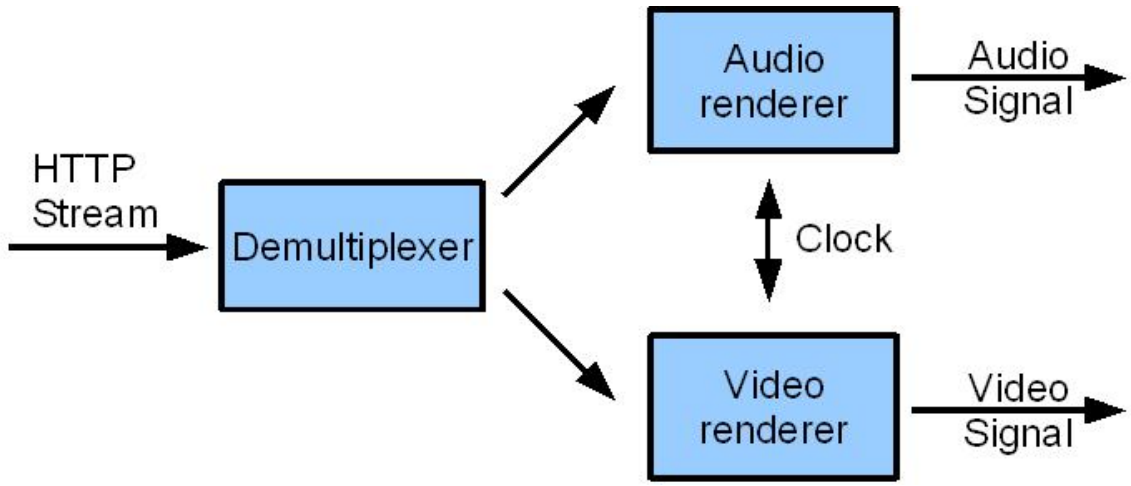

Abbildung 3.3: Veranschaulichung des Streamverlaufs

Da die Entwicklung eines eigenen Videodekodierers den Umfang dieser Bachelorarbeit übersteigt, wird zunächst auf eine Videodarstellung verzichtet. Es wird aber eine Schnittstelle vorbereitet die eine entsprechende Erweiterung erleichtert.

Ein besonderer Aspekt bei der Programmierung unter Symbian OS ist die Verfügbarkeit des Active Object Framework. Nutzt man diese Technik, kann man häufig auf eine Multi Threaded Programmierung verzichten. Dabei verbringt eine Applikation die meiste Zeit darauf wartend, dass ein Event auftritt. Events können dabei Benutzereingaben darstellen oder besondere Ereignisse von Systemressourcen. Jeder Anwendung wird zu dem Zweck ein Scheduler zugewiesen, der mit den Events, sortiert nach Priorität, ein bestimmtes Interface

der Anwendung aufruft[1]. Events können sich dabei nicht gegenseitig unterbrechen. Besondere Vorkehrungen gegen Deadlocks müssen deshalb nicht getroffen werden. Durch das oben geschilderte Vorgehen wird es in unserer Media Player Applikation, abgesehen von Benutzereingaben, zwei Eintrittspunkte geben:

1. Die AudioOutputStream API ruft eine Callback Methode auf, um einen neuen Abspielpuffer anzufordern.

2. Die HTTP Client API ruft eine Callback Methode auf, sobald neue Daten empfangen wurden.

3.2.3 User Interface

Eine Anforderung an den Media Player ist eine Trennung der Engine von der UI um eine Anpassung an zukünftige Smartphone Designs zu erleichtern. Das Model View Controller Pattern (MVC)Krasner und Pope (1988) ist für diesen Zweck besonders geeignet. Das Designpattern besteht aus drei logischen Komponenten die im folgenden kurz für den Einsatz in dem Media Player erläutert werden.

Das Model repräsentiert die Media Player Engine. Hier werden alle relevanten Ressourcen und Daten verwaltet die zur Verarbeitung des Streams benötigt werden. Dazu gehören die Streamingbuffer und sämtliche zugehörigen Parameter. Die Engine modelliert die Funktionalität der Applikation und besitzt verschiedene Zustände. Bei einer Zustandsänderung wird der aktive View benachrichtigt. Die Funktionalität der Engine besteht hauptsächlich aus dem Erstellen und Verwalten der Netzwerkverbindung und der Steuerung des Demultiplexers und der Medienausgabe.

Der View repräsentiert eine besondere Darstellung des User Interfaces. Er ist für die Wiedergabe des Zustandes der Engine auf dem Smartphone Bildschirm verantwortlich. Die zu diesem Zweck benötigten Ressourcen werden von ihm verwaltet. Dabei wird auch der Bereich, der für eine spätere Videoausgabe verwendet werden kann, vorbereitet. Außerdem empfängt er Benutzereingaben und gibt diese an die Engine weiter. Es kann mehrere Views geben die je nach Zustand der Engine wechseln können.

Der Controller, der im weiteren Verlauf der Arbeit AppUI genannt wird, stellt die Verbindung zwischen dem Zustand der Engine und dem aktivierten View dar. Dabei hat ein typischer Media Player zwei grundlegende Zustände. Zu Programmbeginn oder nach dem Beenden einer Wiedergabe befindet er sich in dem Grundzustand. Hier sind keine Ressourcen belegt die unmittelbar an dem Netzwerkverkehr oder dem Abspielen beteiligt sind. Wird eine Streamingdatei zur Repräsentation ausgewählt, ist der Wiedergabemodus aktiv. Diese substanzielle Trennung wird durch zwei unterschiedliche Views modelliert und die AppUI ist dafür verantwortlich, den entsprechenden View zum richtigen Zeitpunkt zu aktivieren. Durch

[1] so ein Interface wird als Callback Interface bezeichnet

diese Technik ist es sehr einfach möglich, die grundlegenden Unterschiede dieser beiden Zustände zu implementieren. Zusätzlich wird eine spätere Erweiterung der Darstellung erleichtert. Abbildung 3.4 stellt die genannten Zusammenhänge dar.

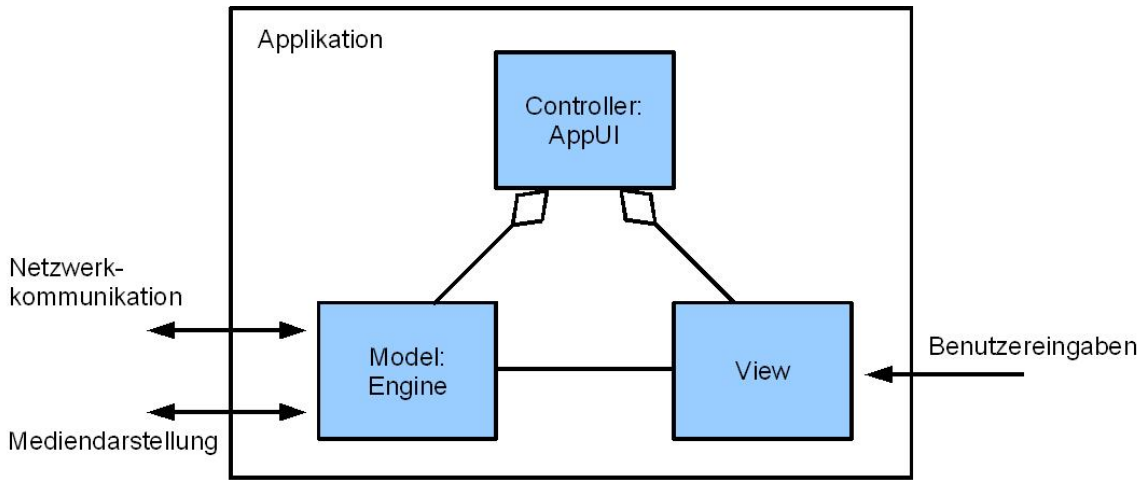

Abbildung 3.4: Konzeption des MVC Patterns

3.2.4 Engine

Um die Engine in verschieden logische Module aufzutrennen, wird im Folgenden der Startvorgang der Wiedergabe genauer umschrieben:

Nachdem die Engine den Auftrag zum Abspielen einer Streamingdatei erhalten hat, ist der erste Schritt die Aufnahme der Netzverbindung. Dazu wird ein Request Header erstellt und direkt an den Server gesendet. Mit dem Response des Servers treffen anschließend, in TCP Paketen verpackt, die ersten Teile der Datei ein. (vgl. 2.2.2.1 auf Seite 12). Sobald der Empfangspuffer gefüllt ist, werden keine neuen TCP Pakete ausgelesen und über die TCP Flusskontrolle stoppt der Server die Versendung bis wieder genügend Platz vorhanden ist.

Wie schon angesprochen, benötigt man ein geeignetes Dateiformat für die Wiedergabe über Progressive Streaming. Erstens müssen die Ton- und Videodaten gebündelt sein, zweitens müssen die Meta-Informationen vor den Mediendaten übertragen werden. Das liegt daran, dass ein Kodierer mit verschiedenen Einstellungen betrieben werden kann. Bei Tonformaten sind das beispielsweise die Samplingrate[2] und der Kanalmodus[3]. Ein Dekodierer benötigt diese Parameter für eine akkurate Wiedergabe. Somit muss die Engine diese Informationen auslesen, bevor der Stream überhaupt wiedergegeben werden kann. Zusätzlich zu den Dekodiereinstellungen werden genauere Informationen über die Bündelung benötigt,

[2]Samplingrate bezeichnet die Anzahl an Samples pro Sekunde
[3]beispielsweise Mono oder Stereo

damit Ton und Video getrennt dekodiert werden kann. Das Auslesen der Meta-Informationen wird ausgeführt, sobald der Empfangspuffer zum ersten Mal gefüllt ist.

Der nächste Schritt besteht aus dem Konfigurieren und Aktivieren der benötigten Dekodierer. Zusätzlich wird für jede Dekodiereinheit ein separater Abspielpuffer angelegt. Die Abspielpuffer werden gefüllt und für eine Wiedergabe freigegeben. Außerdem wird der entsprechende Bereich im Empfangspuffer freigegeben, damit neue TCP Pakete empfangen werden können. Sind die Abspielpuffer abgearbeitet, werden sie erneut mit Daten aus dem Empfangspuffer gefüllt. Sind für eine erneute Füllung nicht genügend Daten im Empfangspuffer vorhanden, wird die Wiedergabe Pausiert, bis der Empfangspuffer wieder genügend Daten enthält.

Die Engine lässt sich in dem oben beschriebenen Ablauf in vier verschiedene Module aufteilen. Im Folgenden werden diese zusammen mit einer ersten Abstraktion der Schnittstellen erläutert:

Das **Netzwerk-Modul** ist für den Start des Abspielvorgangs verantwortlich Es erstellt und verwaltet die HTTP Verbindung. Kommuniziert wird mit dem Streaming-Modul und den User-Interface Views. Das Modul stellt folgende Funktionalität zur Verfügung:

Play, Pause, Stop und Volume werden von dem User-Interface aufgerufen, wenn der Benutzer einen entsprechenden Befehl über die Tastatur eingibt, und je nach Situation an das Streaming-Modul weiter gesendet.

TCP-Paket empfangen wird von dem S60 Framework aufgerufen, sobald ein neues TCP Paket bereit liegt. Das Paket wird direkt an das Streaming-Modul weitergegeben. Die HTTP-API des S60-Frameworks ruft diese Methode nicht wieder auf, solange das Paket nicht freigegeben wurde.

Paket freigeben wird von dem Streaming-Modul aufgerufen, sobald das letzte Paket in den Empfangspuffer übernommen wurde. Der Aufruf dieser Methode wird an das S60 Framework weitergeleitet.

Das **Streaming-Modul** verwaltet die internen Puffer. Dazu gehören der Empfangspuffer und die verschiedenen Abspielpuffer. Es kommuniziert mit dem Netzwerk-Modul um neue Pakete zu behandeln. Das Streaming-Modul kümmert sich außerdem um die Einrichtung der Dekodier-Module und des Meta-Moduls. Das Interface kann folgendermaßen beschrieben werden:

Play, Pause, Stop und Volume werden von dem Netzwerk-Modul aufgerufen (s.o.).

neues Paket wird von dem Netzwerk-Modul aufgerufen, sobald ein neues Paket bereitliegt. Falls genügend Platz im Empfangspuffer vorhanden ist, wird das Paket kopiert und die erfolgreiche Übernahme dem Netzwerk-Modul gemeldet. Falls der Abspielvorgang pausiert ist, z.B. aufgrund eines Leerlaufens des Empfangspuffers, wird er reaktiviert.

Abspielpuffer kopiert wird von den Dekodier-Modulen aufgerufen sobald der jeweilige Abspielpuffer erfolgreich übernommen wurde. Ein neuer Abspielpuffer kann anschließend vorbereitet werden und direkt den Dekodiermodulen angeboten werden. Über das Meta-Modul wird eine Entbündelung der Daten im Empfangspuffer ermöglicht. Hat der Empfangspuffer nicht genügend Daten vorliegen, wird dieser Vorgang bis zum Eintreffen des nächsten Paketes verzögert.

Dateiende wird von dem Netzwerk-Modul aufgerufen, wenn die Übertragung abgebrochen wurde oder alle Pakete der Datei im Empfangspuffer übernommen wurden. Die restlichen Daten werden noch ausgespielt. Danach werden alle Ressourcen freigegeben.

Das **Dekodier-Modul** kümmert sich um die Präsentation der Daten, die von dem Streaming-Modul übergeben worden sind. Jeder Dekodierer hat einen zusätzlichen internen Puffer, der die Daten hält, die zum aktuellen Zeitpunkt wiedergegeben werden. Die öffentliche Schnittstelle benötigt folgende Funktionalität:

Initialisierung wird von dem Streaming-Modul aufgerufen, nachdem die Meta-Daten ausgewertet worden sind. Übergibt die benötigten Grundeinstellungen an den Dekoder.

Einstellungen setzen wird von dem Streaming-Modul während des Abspielvorgangs aufgerufen um z.B. die Lautstärke oder andere Parameter zu ändern.

neuer Buffer wird von dem Streaming-Modul aufgerufen wenn ein neuer Abspielbuffer bereitsteht. Sobald die Daten aus dem Abspielbuffer in den internen Buffer kopiert worden sind, wird das dem Streaming-Modul gemeldet.

Das **Meta-Modul** ist für das Auslesen der Meta-Informationen des Dateiheaders verantwortlich. Die extrahierten Informationen werden gespeichert und dem Streaming-Modul zur Verfügung gestellt. Folgende Schnittstellen sollten implementiert werden:

Meta-Daten einlesen wird von dem Streaming-Modul aufgerufen nachdem der Empfangsbuffer zum ersten mal gefüllt wurde. Es wird ein zusätzlicher Pointer übergeben der nach Abschluss der Einleseoperation auf das Ende des Headers zeigt. So kann das Streaming-Modul diesen Bereich freigeben.

Eigenschaften auslesen wird von dem Streaming-Modul aufgerufen um die Eigenschaften der Streaming-Datei auszulesen. Dazu gehört die genaue Anzahl der Medienstreams und die dem jeweiligen Medienstream zugehörigen Parameter.

nächstes Sample auslesen wird von dem Streaming-Modul für jeden Stream aufgerufen der abgespielt werden soll. Hier wird die absolute Position der nächsten Sample-Einheit in der Mediendatei berechnet.

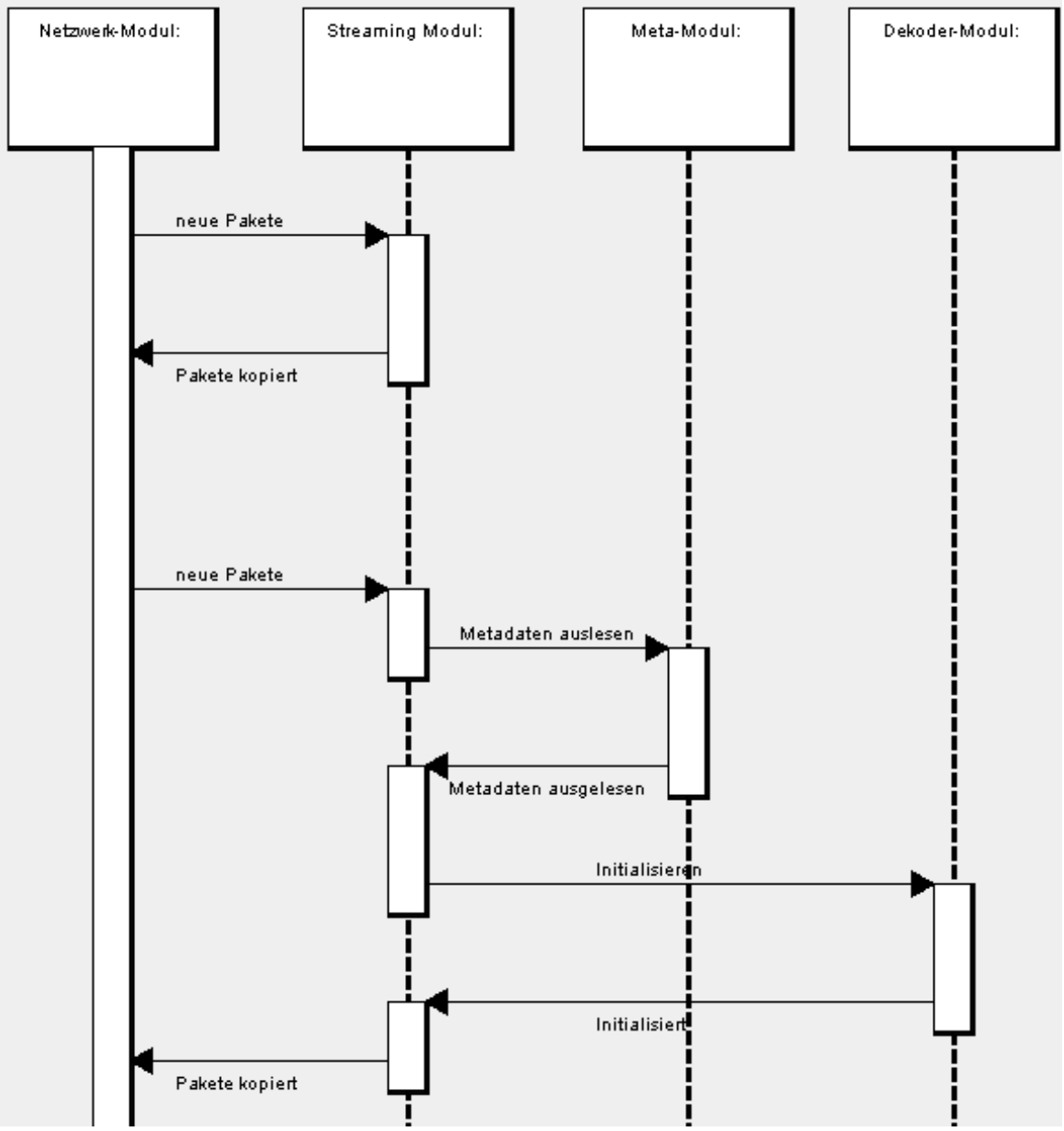

Abbildung 3.5: Sequenzdiagramm des Startvorgangs

Bild 3.5 zeigt die Kommunikation der Module während des Startvorganges. Um das Diagramm einfach zu halten, wurde auf eine Darstellung der Interaktion mit dem Framework verzichtet.

Ein Punkt auf den bisher noch nicht eingegangen wurde, ist die Synchronisation zwischen Audio und Video Stream. Da Störungen in der Audiowiedergabe, die durch leichte Synchronisationsanpassungen entstehen können, wesentlich auffälliger wirken, bietet es sich an, die Videodarstellung der Tonausgabe anzupassen. Dabei können wir die Callback Methode der AudioOutputStream API, die nach einem neuen Abspielpuffer verlangt, als Synchronisationspunkt benutzen. Diese Methodik verlangt nach einer Unterscheidung zwischen einem Dekoder-Modul für Ton und einem für Video. Die Abstrakte Methode „neuer Buffer“ eines Video Dekoder-Modul muss es demnach zulassen können, dass mit dem Abspielen eines neuen Abspielpuffer begonnen werden kann, bevor die letzten Bits des vorherigen noch nicht abgeschlossen sind.

3.3 Codec und Fileformat

Da im Rahmen dieser Bachelorarbeit auf die Implementierung eines Videdekoders verzichtet wird, eine spätere Erweiterung aber leicht möglich bleiben soll, werden sowohl ein reines Ton Dateiformat als auch ein vollwertiges Ton und Video Dateiformat unterstützt. Dabei fällt die Wahl des reinen Tonformats auf das populäre mp3 DateiformatMPEG (1993). Die Implementation gestaltet sich dabei sehr einfach, da das S60 Smartphone einen entsprechenden Hardware beschleunigten Dekoder bereits mitliefert. Da die Datei ausschließlich einen Tonstream enthält, entfällt auch eine Implementation eines speziellen Meta-Moduls. Dieses Vorgehen ermöglicht ein frühes Testen der Funktionalität des Netzwerk-Moduls und der Puffer Verwaltung des Streaming-Moduls während der Entwicklung der Software.

Bei dem gemischten Fileformat fällt die Wahl auf 3gp. Das 3gp Fileformat wurde von dem 3rd Generation Partnership Project(3GPP), einer Kooperation verschiedener Telekommunikations Standardisierungs Organisationen, definiert. Das Ziel der 3GPP Gruppe ist es global anerkannte Standards zu schaffen, die von einer Vielzahl von mobilen Systemen unterstützt werden sollen, um so die Kompatibilität zu erhöhen. Damit sich ein Smartphone als 3GP fähig bezeichnen darf, müssen unter anderem bestimmte Dateiformate und Codecs benutzt werden können.

Das 3gp Fileformat basiert dabei auf dem ISO Base Media FileformatMPEG (2005). Die Datei besteht aus mehreren Abschnitten. Diese Abschnitte werden je nach Spezifikation Box oder Atom genannt. Ein Atom kann dabei ein Container für andere Atoms[4] sein. Es gibt drei wesentliche Atoms auf dem höchsten Level:

[4] bezeichnet als Container Atom

ftyp Dieses Atom liefert eine Liste an Spezifikationen, die von dem Media Player unterstützt werden sollten, um die Datei erfolgreich wiedergeben zu können.

moov Dieses Container-Atom enthält die ganzen Meta-Daten. Dabei sind die Streams aufgeteilt in mehreren trak-Atoms.

mdat Das mdat Atom enthält die eigentlichen gebündelten Media Daten. Es nimmt fast die ganze Datei ein.

Abbildung 3.6 veranschaulicht eine typische 3gp-Datei.

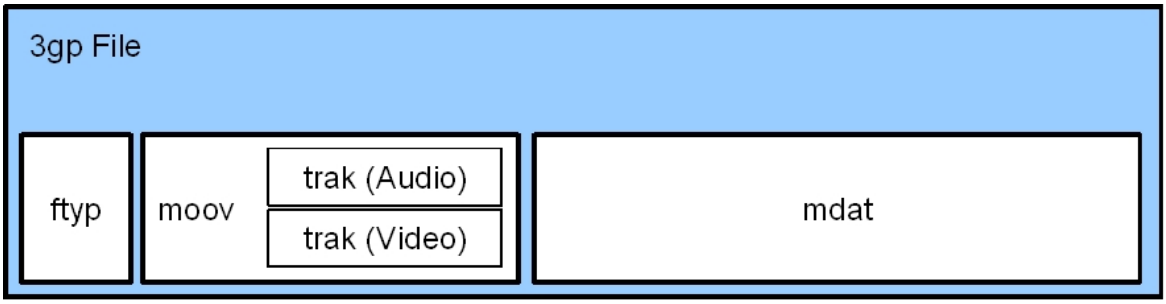

Abbildung 3.6: typische 3gp Dateistruktur

Zusätzlich zum Aufbau der Datei wurden auch die unterstützten Codecs spezifiziert. Tabelle 3.1 zeigt die unterstützten Codecs durch 3GPP PSS Rel-4. Für eine erste Implementation wurde der AMR-NB Codec gewählt, der auf allen S60 Smartphones in einer Hardware beschleunigten Version vorliegt.

Typ	Codec	Unterstützung	max. Bitrate
Speech	AMR-NB	Benötigt	12,2 kbps
Speech	AMR-WB	Benötigt	23,85 kbps
Audio	MPEG-4 AAC-LC	Empfohlen	N/A
Audio	MPEG-4 AAC-LTP	Optional	N/A
Video	H.263 profile 0 level 10	Benötigt	64 kbps
Video	H.263 profile 3 level 10	Empfohlen	64 kbps
Video	MPEG-4 Simple Visual Profile Level 0	Empfohlen	64kbps

Tabelle 3.1: 3GPP PSS Audio und Video Codecs

Kapitel 4

Implementation

In diesem Kapitel wird auf einige wichtige Aspekte der Implementation hingewiesen. Durch den hohen Code-Umfang der Anwendung kann in diesem Umfang nur ein kleiner Ausschnitt vorgestellt werden.

4.1 Meta-Modul

Zur Implementation des Meta-Moduls hat der Atomizer des Helix-Players als Referenz gedient. Obwohl das Helix-Projekt ebenfalls C++ als Entwicklungssprache benutzt, ist der Migrationsaufwand auf das Symbian OS aufgrund des hohen funktionellen Umfangs des Helix-Systems ungewöhnlich hoch gewesen.

Der Atomizer besteht besteht aus drei Hauptklassen:

CQTAtom Diese Klasse ist eine Abstrakte Atom Klasse. Für jeden Atom Typ gibt es eine konkrete Implementation. Der Inhalt des Atoms wird beim Auslesen der Datei in dem Objekt gespeichert. Ausserdem gibt es besondere Zugriffsfunktionen.

CQTTrack Diese Klasse kapselt die Eigenschaften eines trak-Atoms (vgl. 3.3 auf Seite 37). Das Streaming-Modul kommuniziert mit dieser Klasse um Zugriff auf Positionsinformationen des nächsten Samples eines Streams zu erfahren.

CAtomizer Der Atomizer bildet die Atom Struktur des Headers der 3gp-Datei nach, indem konkrete CQTAtom Objekte erstellt und miteinander verknüpft werden. Man kann auf diese Art und Weise bequem durch die Atome iterieren. Dieses Konzept ist leicht erweiterbar indem neue konkrete Atom-Objekte implementiert werden können.

Listing **??** zeigt die wesentlichen Schritte bei der Berechnung des nächsten Samples.

Zeile 1 Es werden zwei Pointer übergeben. aOffset beziffert nach Beendigung der Methode die absolute Position des nächsten Samples dieses Streams. aSize übernimmt die Größe. Der Rückgabewert ist eine Enumeration, um mögliche Fehler oder das Ende der Datei anzuzeigen.

Zeile 5 Das Chunk to Offset Atom zeigt an, wo ein Chunk beginnt. Die Methode EstablishByChunk setzt das Hilfsobjekt deises Atoms auf ein bestimmtes Chunk, dass durch die nächste Zeile genauer festgelegt wird.

Zeile 6 Der Sample to Chunk Table merkt sich in welchem Chunk das aktuelle Sample sich befindet.

Zeile 7 Der Samplesize Table hält die Größe aller Samples. Das Hilfsobjekt wird durch die nächste Zeile auf einen bestimmten Punkt festgelegt.

Zeile 8-9 Das Time to Sample Atom zeigt an welche Zeit mit welchem Sample dieses Tracks verbunden ist. Zusammen mit dem Sample To Chunk Atom wird der SampleSize Table auf das nächste Ample gesetzt.

Zeilen 11-12 Die Position des Samples ergibt sich aus dem Chunk Offset der mit der Position des Samples in diesem Chunk addiert wird.

Zeilen 13 Die Größe kann einfach aus dem Samplesize Table ausgelesen werden.

Listing 4.1: Beispielcode

```
1  HX_RESULT CQTTrack::GetNewPacket(ULONG32 *aOffset, ULONG32 *aSize){
2          HX_RESULT retVal = HXR_STREAM_DONE;
3          if (!m_bTrackDone){
4                  m_PendingState = QTT_SampleRead;
5                  if (m_ChunkToOffset.EstablishByChunk(
6                          m_SampleToChunk.GetChunkNum()) &&
7                      m_SampleSize.EstablishBySample(
8                          m_TimeToSample.GetSampleNumber(),
9                          m_SampleToChunk.GetChunkSampleNum())
10                         ...){
11         ULONG32 ulFileOffset = m_ChunkToOffset.GetChunkOffset() +
12                         m_SampleSize.GetChunkSampleOffset();
13                 *aOffset = ulFileOffset;
14                 m_ulReadSize = m_SampleSize.GetSampleSize();
15                 *aSize = m_ulReadSize;
16                 ....
```

```
17                                  }
18                          }
19  }
```

Kapitel 5

Stabilitätstests

Um die Funktionalität der Applikation in dem mobilen Umfeld zu testen, wurde eine Testreihe mit vier deutschen Mobilfunknetzen betrieben. Zuerst wird das Test-Setup beschrieben, um anschließend die Ergebnisse der beiden Test-Cases zu präsentieren. Zum Schluss gibt es noch ein abschließendes Urteil.

5.1 Test-Setup

Es wurden zwei typische Testszenarios entwickelt, um das typische Anwenderverhalten zu simulieren. Diese Szenarien wurden mit allen 3 Netzbetreibern durchgeführt. (vgl. 3.1.1 auf Seite 21) Zum Testen wurde eine einminütige MP3 Datei in drei verschiedenen Bitraten vorbereitet. Die Test-Dateien wurden auf dem Webserver der HAW-Hamburg hinterlegt. Der Empfangspuffer wurde auf 128 kByte eingestellt.

1. 56 kbit/s
2. 112 kbit/s
3. 320 kbit/s

5.2 Case 1

Der erste Test-Case simuliert die Benutzung zu Hause. Dabei bewegt sich der Anwender nicht. Die Ergebnisse in Tabelle 5.1 wurden in unmittelbarer Umgebung der HAW-Hamburg gegen 17:00 Uhr durchgeführt.

Operator	56 kbps	112 kbps	320 kbps
E-Plus	0 Unterbrechungen	0 Unterbrechungen	2 Unterbrechungen
T-Mobile	0 Unterbrechungen	0 Unterbrechungen	0 Unterbrechungen
Vistream	0 Unterbrechungen	0 Unterbrechungen	0 Unterbrechungen

Tabelle 5.1: Ergebnisse Test-Case 1

5.3 Case 2

Der zweite Test-Case simuliert die Benutzung unterwegs. Die Ergebnisse in Tabelle 5.2 wurden während einer Busfahrt in der Linie 3 von der Haltestelle Holstenstrasse aus gegen 9:00 Uhr durchgeführt.

Operator	56 kbps	112 kbps	320 kbps
E-Plus	0 Unterbrechungen	2 Unterbrechungen	>10 Unterbrechungen
T-Mobile	0 Unterbrechungen	0 Unterbrechungen	3 Unterbrechungen
Vistream	0 Unterbrechungen	0 Unterbrechungen	>10 Unterbrechungen

Tabelle 5.2: Ergebnisse Test-Case 2

5.4 Ergebnisse

Die beiden Testcases haben

Erwartungsgemäß ergeben sich starke Unterschiede zwischen den beiden Testcases. Die guten Ergebnisse von T-Mobile kann man aber auch mit der schon eingeführten HSDPA Technik in Hamburg erklären. Da HSDPA doch eine deutlich höhere maximale Downloadrate bei geringerer Round-Trip-Time bietet.

Kapitel 6

Zusammenfassung und Erweiterungsmöglichkeiten

Die Entwicklung hat gezeigt, dass es sich Lohnt die unterschiedlichen Alternativen abzuwägen. T-Mobile zeigt bereits, dass sich mit HSDPA das mobile Streamen in bisher nicht geahnte Qualitätsstufen entwickeln wird und mit der Adoption moderner Komprimierungsalgorithmen werden Anwendungen wie Online TV und mobile Konferenzräume bald zum Alltag gehören. HSDPA verspricht durch geringere Latenzzeiten und einer höheren Maximalgeschwindigkeit gerade im Progressive Streaming eine hohe Leistungsföhigkeit.

Auch wenn sich die Netzwerktechnik weiterentwickeln wird, kann ein Nachteil nicht ohne weiteres beseitigt werden. Die schlechte Unterstützung von Videorekorder Funktionen macht das Progressive Streamen für große Clips eigentlich unbrauchbar. Eine Möglichkeit wäre vielleicht die Ausnutzung der Resume-Fähigkeit des HTTP Protokolls. Man kann sich eine Erweiterung des Progressive Streaming Modus vorstellen, bei der sich unter Einbeziehung der Dateigröße eine rudimentäre Spulfunktionalität implementieren ließe. Weitere Erweiterungsmöglichkeiten bieten sich vorallem im Bereich der Unterstützung zusätzlicher Codecs und Dateiformate. Außerdem könnte man sich einen Dateibrowser vorstellen, der das lokale Dateisystem des Smartphones durchsuchen kann.

Die Entwicklung des Media Players kann man als erfolgreich betrachten. Vor allem positiv aufgefallen ist die Möglichkeit, auch über WAP-Gateways streamen zu können.

Literaturverzeichnis

[Handley u. a. 2006] HANDLEY, M. ; JACOBSON, V. ; PERKINS, C.: *SDP: Session Description Protocol*. RFC 4566 (Proposed Standard). Juli 2006. – URL http://www.ietf.org/rfc/rfc4566.txt

[Krasner und Pope 1988] KRASNER, Glenn E. ; POPE, Stephen T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. In: *J. Object Oriented Program.* 1 (1988), Nr. 3, S. 26–49. – ISSN 0896-8438

[MPEG 1993] MPEG: *ISO/IEC 11172-3, Coding of Moving Pictures And Associated Audio For Digital Storage Media at up to About 1.5 Mbit/s Part 3 Audio*. ISO/IEC (Veranst.), 1993

[MPEG 2005] MPEG: *ISO/IEC 14496-12, Information technology Coding of audio-visual objects Part 12: ISO base media file format*. ISO/IEC (Veranst.), 2005

[Perkins 2003] PERKINS, Colin: *RTP: Audio and Video for the Internet*. Addison Wesley, 2003

[Rosenberg u. a. 2003] ROSENBERG, J. ; WEINBERGER, J. ; HUITEMA, C. ; MAHY, R.: *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. RFC 3489 (Proposed Standard). März 2003. – URL http://www.ietf.org/rfc/rfc3489.txt

[Schulzrinne u. a. 2003] SCHULZRINNE, H. ; CASNER, S. ; FREDERICK, R. ; JACOBSON, V.: *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Standard). Juli 2003. – URL http://www.ietf.org/rfc/rfc3550.txt

[Schulzrinne u. a. 1998] SCHULZRINNE, H. ; RAO, A. ; LANPHIER, R.: *Real Time Streaming Protocol (RTSP)*. RFC 2326 (Proposed Standard). April 1998. – URL http://www.ietf.org/rfc/rfc2326.txt

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computer Networks*. 4th edition. Prentice Hall, 2003

[Wang u. a. 1997] WANG, Y. ; ORCHARD, M. T. ; REIBMAN, A. R.: Multiple description image coding for noisy channels by pairing transform coefficients. In: *Proc. IEEE Workshop on Multimedia Signal Processing*, 1997, S. 419–424

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. September 2007 Thilo Stallherm