Konzeption und Programmierung eines UPnP-AV/DLNA Plugins für den Video Disk Recorder

Denis Loh

16. November 2009

Dieses Dokument basiert auf dem Artikel „Entwicklung eines UPnP Plugins für den VDR“ von Denis Loh und Andreas Günther aus „Operating Systems 2 im Sommersemester 2009“. Es arbeitet die fehlenden Funktionen und Probleme auf, die dabei aufgetreten sind und erweitert das Plugin um weitere nützliche Funktionen. Dabei soll hauptsächlich Wert auf die Implementation des DLNA Standards Version 1.5 gelegt werden.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AVTS	AVTransport Service
CDS	Content Directory Service
CMS	Connection Manager Service
DHCP	Dynamic Host Control Protocol
DIDL	Digital Item Declaration Language
DLNA	Digital Living Network Alliance
DMC	Digital Media Controler
DMP	Digital Media Player
DMR	Digital Media Renderer
DMS	Digital Media Server
EPG	Electronic Program Guide
FF	Full-Featured
FTP	File Transfer Protocol
HND	Home Network Device
HTTP	Hypertext Transfer Protocol
HTTPMU	HTTP over Multicast UDP
IP	Internet Protocol
IPv4	IP Version 4
MAC	Media Access Control
M-DMC	Mobile Digital Media Controler
M-DMR	Mobile Digital Media Renderer
M-DMS	Mobile Digital Media Server
M-DMU	Mobile Digital Media Uploader
MHD	Mobile Handheld Device
MP3	MPEG1 Layer 3
NAS	Network Attached Storage
PID	Program Identifier

RCS	Rendering Control Service
RTP	Realtime Transport Protocol
SDK	Software Development Kit
SHA-1	Secure Hash Algorithm Typ 1
SOAP	Simple Object Access Protocol
SQL	Simple Query Language
SSDP	Simple Service Description Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
UPnP-AV	UPnP Audio/Video
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USN	Unique Service Name
UUID	Universally Unique Identifier
VDR	Video Disk Recorder
WLAN	Wireless Lokal Area Network
W3C	World Wide Web Consortium
XBMC	XBox Media Center
XML	Extended Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Einleitung

Der Video Disk Recorder (VDR) von Klaus Schmidinger ist eine äußerst vielseitige Anwendung für Linuxnutzer. Ursprünglich als Alternative zu handelsüblichen Satellitenreceivern gedacht, ist die Software zu einem umfangreichen Potpourri von Anwendungen herangewachsen, die sich mit einem Fernseher und einem Videorekorder kombinieren lassen. Der VDR ist dadurch zu einen typischen „Media Center“-Umgebung geworden und dient in vielerlei Hinsicht zu mehr, als nur TV-Mitschnitte anzusehen. Er wird als Zugang zu allen möglichen digitalen Medien genutzt, seien es Bilder oder Audiodateien, Fernsehaufzeichnungen oder DVD-Sammlungen. Aus diesem Grund eignet er sich ideal, um ihn mit den Methoden des Universal Plug and Play (UPnP) für weitere Geräte im Heimnetzwerk erreichbar zu machen. Mit UPnP kann man ohne umfangreiches Eingreifen des Endanwenders die Dienste des VDR weitervermitteln. Diese Arbeit basiert auf den Ergebnissen der Entwicklung für den Artikel „Entwicklung eines UPnP Plugins für den VDR“ aus „Operating Systems 2 im Sommersemester 2009“. Die gewonnenen Erkenntnisse werden hiermit fortgeführt und erweitert. Gerade in Hinblick auf die Unterstützung handelsüblicher Geräte soll Wert auf Standardkonformität gelegt werden. Daher werden im Vorfeld an die eigentliche Programmierung einige Ziele definiert. Darauf aufbauend ist zu bewerten, wie der derzeitige Stand des Plugins einzuschätzen und gegebenenfalls zu verbessern ist.

2 Theoretische Vorbetrachtungen

2.1 UPnP

Dieser Abschnitt beschreibt kurz die Funktionsweise von Universal Plug and Play. Dazu findet sich eine detailierte Erläuterung in [8]. UPnP hat das Ziel, die Verbindung und Konfiguration zwischen Endgeräten flexibel und einfach zu gestalten. Dabei sollen speziell netzwerkunkundige Heimanwender angesprochen werden, denen die Einrichtung eines Gerätenetzwerkes zu kompliziert oder zu aufwendig erscheint. UPnP bietet aufgrund dessen eine Architektur, die weitgehend autonom die Konfiguration und Kommunikation übernimmt. Sie basiert auf bestehenden Protokollen, die sich auch in größeren Netzwerken bewährt haben, wie zum Beispiel das Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP) und Hypertext Transfer Protocol (HTTP). Für den Nachrichtenaustausch zwischen den Geräten wird die Extended Markup Language (XML) und das Simple Object Access Protocol (SOAP) eingesetzt, wobei letzteres eine Schnittstelle für die Kommunikation der Geräte bereitstellt. Durch all diese standardisierten Protokolle ist UPnP unabhängig von der Plattform, auf der es angewendet wird. Somit stehen die Programmiersprache als auch das Betriebssystem dem Hersteller frei zur Auswahl. UPnP schränkt Art und Umfang der Anwendung in keinerlei Hinsicht ein, was es prinzipiell möglich macht, jedes Gerät mit einem Netzwerkanschluss UPnP-fähig zu gestalten. Von einem netzwerkgestütztem Toaster mit Bräunungsgradkontrolle [19] bis hin zur vollständigen Gebäudeautomation mit Videoüberwachung ist alles denkbar.

UPnP definiert zwei Arten von Geräten. Zum einen stellt ein *Device* Dienste im Netzwerk zur Verfügung. Es ist vergleichbar mit einem Server. Ein *Device* oder *Gerät* ist nicht zwangsläufig eine physikalische Aperatur, sondern mehr eine Einheit, welches die notwendigen Dienste bereitstellt, um die Eigenschaften eines Gerätes zu übernehmen. So kann beispielsweise ein UPnP-fähiger Internetgateway ein eigenständiges Gerät sein, es kann sich dabei aber auch um einen vollständigen Computer handeln, der lediglich die Fähigkeiten eines Internetgateways unterstützt. Man spricht dabei von eingebetten Geräten. Die angebotenen Dienste werden nun von der zweiten Geräteklasse, den *Control Points*, angewendet. Sie rufen die unterstützten Funktionen vom *Server* ab und führen die gewünschten Aktionen aus. Ein Anwendungsfall aus der Gebäudeautomation ist die Steuerung der Beleuchtung. Die Beleuchtung bietet die Dienste der Helligkeitssteuerung an, über die das Licht ein- beziehungsweise augeschaltet oder gedimmt werden kann. Der Kontrollpunkt greift auf die angebotenen Schnittstellen zurück und kann nun die Helligkeit entsprechend der unterstützten Parameter regeln. Dabei durchläuft der Aufruf der Funktion mehrere Phasen, die vom Suchen der Geräte bis zum Anzeigen einer Gerätewebseite reichen.

2.1.1 Adressierung

Die Phase der Adressierung ist die erste der sechs Phasen, die durch das UPnP Komitee vereinbart wird, und eine notwendige Vorbereitung für den Betrieb. Ohne gültige IP-Adresse kann das Gerät im Netzwerk nicht gefunden und angesprochen werden. Dabei wird mindestens eine Adresse aus dem IP Version 4 (IPv4)-Adressraum benötigt. Sie wird entweder fest vergeben, per Dynamic Host Control Protocol (DHCP) ermittelt oder aus dem Auto-IP-Bereich[1] 169.254.0.0/16 ausgewählt. Jedes UPnP-fähiges Gerät muss sowohl DHCP als auch Auto-IP unterstützen. Erhält das Gerät über einen DHCP eine gültige Adresse, kann es sofort mit der anschließenden Phase fortsetzen. Sollte allerdings kein DHCP-Server gefunden werden, sucht sich sich das Gerät eine Auto-IP heraus, prüft periodisch die Anwesenheit eines DHCP und schaltet automatisch um, wenn dieser gefunden wird.

2.1.2 Description

Die zweite Phase ist die Beschreibungsphase. Das heißt, dass die angebotenen Dienste mit den unterstützten Funktionen und Schnittstellen definiert werden. Hierzu wird eine XML-Datei[2] angelegt, worin die Gerätebeschreibung, die Gerätenummer, Herstellerinformationen und die Schnittstellendefinitionen für die einzelnen Dienste enthalten sind. Die Schnittstellendefinition enthält die Funktionsnamen, notwendige und optionale Parameter und deren Rückgaben[3].

2.1.3 Discovery

Im Anschluss an die Beschreibung beginnt das Discovery, dem Entdecken von anderen Geräten. Hierbei sind zwei unterschiedliche Wege möglich. Zum einen kann ein Control Point die Dienste gezielt abrufen, sofern das gewünschte Gerät beim Control Point bekannt ist, zum anderen kann das Gerät seine Fähigkeiten auch selbstständig im Netzwerk anbieten. Um dies realisieren zu können, müssen die Dienste unterscheidbar sein. Diesem Problem begegnet man mit dem Simple Service Description Protocol (SSDP), welches zwei Konzepte anbietet. Der Service Type bezeichnet die Art und Funktion des Gerätes und wird durch einen Uniform Resource Identifier (URI) eindeutig gekennzeichnet. Das UPnP Working Committee definiert diese Service Types für alle Standardgerätetypen. Ein weiteres Unterscheidungsmerkmal ist der Unique Service Name (USN), womit zwei Geräte mit gleichem Service Type differenzierbar sind. Es handelt sich ebenfalls um einen URI, welche in der Regel einen 128 Bit kodierten Universally Unique Identifier (UUID) enthält. Diese Zahl muss eindeutig einem Gerät zugeordnet werden. In früheren Versionen der UUID-Generierung wurde dies über die Media Access Control (MAC)-Adresse, einem Zeitstempel sowie einer Zufallszahl bewerkstelligt. Aktuell werden Secure Hash Algorithm Typ 1 (SHA-1)-Hashcodes zur

[1] auch Zeroconf genannt
[2] in der Regel description.xml
[3] Diese sind in Service Description Dokumenten enthalten, die eigenständige XML-Dateien sind und in der description.xml verlinkt werden

Generierung genutzt, welche eine höhere Zufallsdichte haben und keinen Rückschluss auf den Computer und den Zeitpunkt des Erstellens erlauben. Das heißt allerdings auch, dass es in einem Netzwerk keine Geräte geben darf, die eine identische UUID besitzen. Sie wird entsprechend persistent abgespeichert und muss auch Neustarts überdauern.

Obwohl SSDP ein HTTP-basiertes Protokoll ist und HTTP auf TCP aufbaut, benötigt es einen anderen Weg der Kommunikation mit anderen Endpunkten. TCP sieht eine gesicherte Punkt-zu-Punkt-Verbindung vor, welches eine breite Suche im Netzwerk unmöglich macht. Die Gegenstellen sind noch nicht bekannt. Daher wird ein weiterer Ansatz genutzt, der die Übertragung von HTTP over Multicast UDP (HTTPMU) nutzt. Das heißt, anstelle von TCP finden UDP-Multicasts Anwendung, welches eine ungerichtete Verbreitung der Nachrichten im Netzwerk zulässt. Dabei wird die IPv4-Adresse 239.255.255.250 und der Port 1900 genutzt. Die Anwendung von UPnP ist daher nur möglich, wenn Multicasts im Netzwerk übertragen werden können, weil ansonsten die Phase des Discovery blockiert wird und sich die Geräte untereinander nicht finden. Dieser Punkt ist bei der Einrichtung einer Firewall unter Umständen zu beachten. Der genaue Kommunikationsablauf sowie die Headeraufbauten der SSDP-Nachrichten werden an dieser Stelle weggelassen.

2.1.4 Control

Der eigentlich funktionale Teil der Geräte wird über die Control-Phase geregelt. Mittels SOAP-Nachrichten können die durch das Description beschriebenen Funktionen und Aktionen aufgerufen werden, um das Endgerät zu steuern. Das empfangende Gerät reagiert auf die Anfragen und antwortet mit den entsprechenden Ergebnissen oder Fehlern. SOAP basiert auf XML, weshalb es den Vorteil bietet auch komplexe Strukturen relativ einfach und übersichtlich darzustellen. Da es nur den groben Rahmen für den Nachrichtenaustausch vorgibt, müssen die Endpunkte die übertragenen Inhalte eigenständig interpretieren können. Alternativ zum Funktionsaufruf kann auch ein Zustandswechsel angewiesen werden. Beispielsweise kann ein Control Point ein binären Lampe, die die Zustände *an* und *aus* unterstützt, einschalten. Solche Wechsel können andere Geräte beobachten, da diese im Netzwerk mitgeteilt werden. Dies geschieht über das Eventing.

2.1.5 Eventing

Wie in 2.1.4 bereits erläutert, kann ein Kontrollpunkt auch eine Änderung des Zustands erwirken. Dabei kann sich ein anderes UPnP-Gerät für ein bestimmtes Ereignis registrieren, das heißt, es wird ein Abonnement vereinbart. Dieses Publisher/Subscriber oder auch Verleger/Abonnenten-Modell ist reaktiv. Wenn ein Ereignis ausgelöst wurde, teilt das entsprechende Gerät seine Änderung an alle angemeldeten Teilnehmer mit. Diese können dynamisch darauf reagieren, je nachdem ob das Ereignis für sie relevant ist oder nicht. Gleichzeitig können sie als Antwort wiederum ein Ereignis auslösen, welches für weitere Geräte interessant sein kann. Die Notwendigkeit dieses Systems zeigt sich am folgenden Beispiel sehr gut. Die im vorhergehenden Abschnitt

2.1.4 beschriebene Lampe wird nun ausgeschaltet. Im selben Raum ist eine Videokamera, welche über eine Infrarotfunktion verfügt. Würde das System nicht reaktiv sein, bliebe das Bild nach Abschalten der Lampe dunkel. So aber kann die Überwachungskamera automatisch auf die Nachtsichtfunktion umschalten, ohne dass ein Kontrollpunkt dies anweisen muss. Dieser Ansatz vereinfacht die Kommunikation und die Implementation der Kontrollpunkte erheblich.

2.1.6 Presentation

Jedes UPnP-Gerät besitzt einen eingebauten minimalen Webserver. Die Kommunikation läuft hauptsächlich über HTTP ab, daher ist das Anbieten von Nachrichten und Anzeigen von Gerätewebseiten nahezu der gleiche Ablauf. Diese Phase ist vielmehr eine zusätzliche Schicht, welche Eventing und Control vereint. Denn über die Gerätewebseite ist es teilweise möglich, ohne separate Kontrollpunkte das Gerät zu steuern. Inwiefern die Steuerung über das Webinterface erlaubt wird und welche Informationen angezeigt werden, obliegt dem Hersteller. Es kann daher auch sein, dass lediglich allgemeine Informationen angezeigt werden. Prinzipiell kann aber gesagt werden, dass eine zusätzliche Administrationsfähigkeit über dieses Interface die Bedienbarkeit und Benutzerfreundlichkeit erheblich steigern kann.

2.2 UPnP-AV

UPnP Audio/Video (UPnP-AV) ist eine Unterart von UPnP, welche speziell für Audio- und Videoübertragung konzipiert wurde. Dabei wird dem Benutzer der Aufwand abgenommen, sowohl Server- als auch Clienthard- und Software einrichten zu müssen. Hauptaugenmerk der Steuerung liegt lediglich beim Kontrollpunkt. Dieser koordiniert die Kommunikation zwischen den beiden Endpunkten. Das UPnP-AV Working Committee hat es sich zur Aufgabe gemacht, das Zusammenspiel von netzwerkgestützten Mediageräten zu vereinfachen. Angenommen man besitzt eine sehr umfangreiche Videobibliothek oder eine MPEG1 Layer 3 (MP3)-Sammlung auf einem Computer oder Network Attached Storage (NAS). Im Wohnzimmer befindet sich ein Flachbildfernseher mit einer Heimkinoanlage, in der Küche sowie im Bad ein Radio. Der Benutzer sah sich bisher gezwungen, die Videos auf dem Computer anzusehen anstatt im teuren Heimkino. Aus dem Radio ertönt auch nur die wenigen Sender, die über Rundfunk ausgestrahlt werden können. Mit UPnP kann man diese Geräte miteinander verbinden. Die Medien werden über einen Digital Media Server (DMS) im Netzwerk angeboten. Der Fernseher, welcher als Digital Media Player (DMP) agiert, kann diese in Echtzeit herunterladen und das Radio zieht sich über das Internet automatisch eine Liste der weltweit verfügbaren Radiostationen. Kontrolliert und gesteuert wird dies mit einem kleinen Handheld-Computer, der Kontrollpunkt oder auch Digital Media Controler (DMC) genannt wird. Wenn der Benutzer einen Raum verlässt, kann er mit dem DMC das Radio in der Küche anweisen, den Stream zu übernehmen, welcher gerade noch im Bad gespielt wurde. Für dieses Szenario sind mehrere Dienste zuständig, die im Folgenden kurz erläutert werden sollen.

2.2.1 Content Directory Service

Der Content Directory Service (CDS) ist die zentrale Zugangsstelle eines DMS. Dieser zeigt, welche Dateien im Moment verfügbar sind und wo diese gefunden werden können. In erster Linie ist es eine Datenbank, welche Informationen zu den Daten bereit hält. Darunter fallen zum Beispiel Titel, Beschreibung und die Art der Datei. Wenn es sich um ein Video handelt, stellt der Server weitere Informationen zur Verfügung, wie Schauspieler, Produzenten oder wann der Film gedreht wurde. Bei Radiostreams könnten die Senderbezeichnung und die übliche Sendefrequenz angezeigt werden. Die wichtigste Information ist allerdings die Resource, welche nicht nur den Ort der Datei enthält, sondern auch anzeigt, über welches Protokoll die Datei übertragen werden kann. Dies ist meistens HTTP. Weitere Protokolle können File Transfer Protocol (FTP) und Realtime Transport Protocol (RTP) sein. Die Navigation durch den Inhalt erfolgt in der Regel über die *Browse*-Funktion. Es ist vergleichbar mit der hierarchischen Struktur eines Dateisystems. Ordner werden durch *Container* und Dateien durch *Items* dargestellt. Der Kontrollpunkt fragt mittels SOAP-Nachricht den Inhalt eines bestimmten *Containers* ab, worauf der Server entsprechend die Inhalte als XML-Dokument zusammenfasst und zurücksendet. Als XML-Format wird die Digital Item Declaration Language (DIDL) verwendet, wobei UPnP-AV die Unterart DIDL-Lite nutzt, welches wenige Abwandlungen hat. Der Kontrollpunkt wertet die erhaltenen Daten aus und der Benutzer hat nun die Möglichkeit, sich die detailierten Informationen zu einem *Item* anzeigen zu lassen. Ein DMS muss nicht zwangsläufig auch die Dateien lokal verfügbar haben, die es im Netzwerk anbietet. Die Resourcen, die über *Browse* ermittelt werden können, liegen unter Umständen auf einen vollkommen anderen Host. Lediglich die Metadaten werden immer lokal verwaltet.

2.2.2 Connection Management Service

Der Connection Manager Service (CMS) ist nun das Bindeglied zwischen Server und Client, wobei der der Digital Media Renderer (DMR) als Client zu verstehen ist, obwohl er ähnlich wie der DMS Dienste bereitstellt. Strenggenommen ist der DMC der einzige richtige Client im Netzwerk, da er nur Dienste nutzt und selbst keine zur Verfügung stellt. Er stellt die Verbindung zwischen DMS und DMR her, in dem es die bereitgestellten Funktionen des CMS nutzt. Es ruft von jeweils beiden Seiten die unterstützten Protokollinformationen ab und kann somit feststellen, ob bestimmte Dateien auf einem DMR darstellbar sind oder nicht. Zusätzlich kann eine Verbindungssteuerung implementiert sein, welche parallele Verbindungen zwischen einem DMR und DMS erlauben.

2.2.3 Rendering Control Service

Der wichtigste Dienst, den ein DMR bereitstellen muss, ist der Rendering Control Service (RCS). Es bietet dem Kontrollpunkt die Möglichkeit, die üblichen Einstellungen eines Anzeigegerätes zu regulieren. Darunter gehören zum Beispiel Lautstärke-, Helligkeits-, Kontrast- und Farbregulierung. Es müssen allerdings nicht alle Funktio-

nen implementiert sein. Während ein Fernsehgerät meist alle Funktionen unterstützt, bietet ein UPnP-Radio keine Bildeinstellungsfunktionen.

2.2.4 AVTransport Service

Dieser Dienst ist optional, da die wichtigsten Funktion bereits durch die anderen Dienste abgedeckt werden. Dennoch kann der AVTransport Service (AVTS) eine interessante Verbesserung darstellen, welche die Endgeräte um die typischen Funktionalitäten eines Videorekorders erweitert. *Stop*, *Pause* oder *Vorspulen* versetzen den DMR in eine aktive Position. Bisher war der DMS der Aktor und bot die Dateien ohne Einfluss auf das Streaming an. Der DMR lag nun in der Verantwortung, Funktionen, wie Pausieren oder Spulen, eigenständig zu übernehmen. Mittels AVTS geht der DMR in die sogenannte *Pull*-Methode über. Gegenüber der *Push*-Methode weißt der DMR den DMS an, auf welche Art und Weise er die folgenden Daten zu übertragen hat. Es findet somit eine Übertragungsratenkontrolle statt. Der Vorteil liegt auf der Hand: ein Client benötigt keine umfangreiche Pufferung, falls die ankommenden Daten nicht in Echtzeit verarbeitet werden können. Gleichzeitig bietet sich hier auch ein großer Nachteil, denn der Dienst ist optional und funktioniert nur, wenn beide Seiten ihn unterstützen.

2.3 DLNA

Die Digital Living Network Alliance ist ein Zusammenschluss mehrerer Unternehmen der Unterhaltungsindustrie, welcher es sich zum Ziel gemacht haben, das Zusammenspiel zwischen Mediengeräten noch weiter zu vereinfachen und einen einheitlichen Standard zu verabschieden. Inzwischen sind über 240 Unternehmen eingetragene Mitglieder, welche bereits über 200 Millionen Digital Living Network Alliance (DLNA)-zertifizierte Geräte produziert und ausgeliefert haben. Die DLNA verabschiedet Richtlinien, welche für Gerätehersteller eine einheitliche Basis bieten und dementsprechend die Interoperabilität steigern.

2.3.1 Formate

Gerade im Hinblick auf die uneingeschränkte Formatvielfalt unter UPnP-AV ist eine klarere Regelung notwendig. Denn ein DMR kann unter Umständen nicht die Formate abspielen, die der DMS anbietet. Um diesen Problem zu begegnen, schränkt die DLNA die Formate auf eine stark begrenzte, aber dennoch weit verbreitete Anzahl ein. Jedes zertifizierte Gerät muss, je nach Geräteklasse, bestimmte Anforderungen zwingend erfüllen. Tabelle 2.1 zeigt einige wichtige Formate, welche in Version 1.5 des Standards vorgesehen sind.

Auch die Transportprotokolle werden auf ein Minimum reduziert. Während es bei UPnP-AV unerheblich ist, über welche Übertragungsart die Daten angefordert werden, erzwingt die DLNA mindestens HTTP in der Version 1.0 oder 1.1 oder RTP zu unterstützen, wobei letzteres optional ist. Auf der anderen Seite wird die Konnektivität der Geräte um Bluetooth erweitert, wodurch auch Handys und Pocket PCs

Formatgruppe	Beschreibung
JPEG	Bilder in drei Größenvarianten, Thumbnails und Icons
PNG	große Bilder, Thumbnails und Icons
AC-3	Dolby Digital Audiodateien
AMR	Sprachaufzeichnungen
ATRAC3+	Sony proprietäres Audioprotokoll
LPCM	Für Aufnahmen von Radiostationen und benutzerdefinierten Inhalten
MP3	Komprimierte Audiodateien
AAC	Komprimierte Audiodateien (MPEG4)
WMA	Windows Media Audio Dateien
MPEG1	Audio/Video-Dateien, z.B. VideoCDs
MPEG2	Audio/Video-Dateien, auch MPEG-TS und -PS, sowie DVDs
MPEG4 Part 2	DivX komprimierte Videos
MPEG4 Part 10	AVC (H.264) komprimierte Videos, HDTV
WMV9	Windows Media Video 9 Dateien
Media Collections	Zur Beschreibung von Medienzusammenfassungen
XHTML Dateien	XHTML-Dateien, welche mit einem DLNA-Drucker ausgedruckt werden können

Tabelle 2.1: unterstützte DLNA-Formatklassen

eingebunden werden können.

2.3.2 Geräteklassen

Die Geräteklassen, welche im Verlauf der Erläuterungen von UPnP-AV bereits verwendet wurden, werden durch den Standard genauer beschrieben. Dabei unterscheidet man grundsätzlich in Geräte zur Audio- und Videoverarbeitung und Drucker.

Digital Media Server
: Ein Digital Media Server ist ein Server, welcher die UPnP-AV-Funktionalität eines UPnP Media Server:1 unterstützt und die Dienste Content Directory Service und Connection Manager Service anbietet. Er übernimmt die Rolle des Anbietens und Verbreitens von Medieninhalten im Netzwerk.

Digital Media Player
: Ein Digital Media Player ist ein Zusammenschluss aus einem Digital Media Renderer und Digital Media Controler. Dieser stellt die Fähigkeiten eines UPnP Media Renderer:1 zur Verfügung, wobei der Kontrollpunkt direkt integriert ist und somit ein 2-Box-Pull System ermöglicht.

Digital Media Renderer
Ein Digital Media Renderer ist dementsprechend ein eigenständiger Renderer, welcher lediglich digitale Medien darstellen und nicht selbst abrufen kann. Er muss durch einen Digital Media Controler gesteuert werden.

Digital Media Controler
Ein Digital Media Controler kann auf die Medieninhalte eines DMS zugreifen und sucht passend zu den vom DMR unterstützten Protokollen die Medienresource heraus und stellt die Verbindung zwischen DMS und DMR her.

Digital Media Printer
Ein Digital Media Printer ist ein digitaler Drucker, welcher Bilder und HTML-Textseiten drucken kann.

Die letzten drei der eben genannten Geräte wurden mit der aktuellen Version 1.5 des DLNA-Standards definiert. Sie gehören zu der Kategorie Home Network Device (HND). Zustätzlich zu diesen wurden nun auch mobile Geräte eingeführt, welche grundsätzlich die gleichen Aufgaben übernehmen können. Ein solches Gerät wird auch als Mobile Handheld Device (MHD) bezeichnet.

Mobile Digital Media Server
Ein Mobile Digital Media Server ist ein mobiler DMS, welcher in den meisten Fällen über Wireless Lokal Area Network (WLAN) oder Bluetooth mit anderen Geräten verbunden ist.

Mobile Digital Media Renderer
Ein Mobile Digital Media Renderer ist ein mobiler DMR, welcher über die gleiche Funktionalität eines DMR verfügt, aber häufig nur eine geringere Auswahl an Medienformaten unterstützt. So werden in der Regel nur Medien abgespielt, die vom Gerät in Auflösung und Bitrate darstellbar sind.

Mobile Digital Media Controler
Ein Mobile Digital Media Controler ist ein mobiler DMC, welcher hauptsächlich Inhalte eines Mobile Digital Media Server (M-DMS) findet und die Verbindung mit einem DMR herstellt. Eine genauere Erläuterung zu den möglichen Verbindungsszenarien folgt im Anschluss.

Mobile Digital Media Uploader
Ein Mobile Digital Media Uploader ist ein mobiles Gerät, welches eigene lokale Inhalte auf einen DMS oder M-DMS hochladen kann. Ein Beispiel für solch ein Gerät ist eine digitale Fotokamera, welche die gemachten Bilder auf den Server lädt.

Mobile Digital Media Downloader
Ein Mobile Digital Media Downloader ist das Gegenstück zum Mobile Digital Media Uploader (M-DMU). Dieser lädt Inhalte von einem DMS oder M-DMS herunter und spielt sie lokal ab, nachdem der Download beendet wurde.

Die beiden Kategorien der HNDs und MHDs werden generell getrennt betrachtet, da sie sich in Hinblick auf die unterstützten Medienformate und auch deren Konnektivität schwer miteinander vereinen lassen. Dennoch darf man nicht explizit ausschließen, dass ein MHD nur mit anderen MHDs und ebenso ein HND nur Dienste für einen weiteren HND anbieten kann. Es ist durchaus möglich, dass beispielsweise ein Mobile Digital Media Renderer (M-DMR) ein Video von einem DMS abspielen kann, sofern er über die entsprechende Konnektivität verfügt. Gleichermaßen kann auch ein Mobile Digital Media Controler (M-DMC) einen DMR und DMS steuern. Durch diese Annahme lässt sich die Systemanwendung auch auf das volle Spektrum der unterstützten Geräte erweitern. Hierfür werden zusätzliche Geräteklassen eingeführt, welche eine Brücken- und Konvertiererfunktion übernehmen, sollten HNDs und MHDs nicht direkt miteinander kommunizieren können.

2.3.3 Modelle

Es existieren mehrere Szenarien, welche das Zusammenspiel zwischen Server und Renderer zeigen. In den meisten Anwendungen ist der Media Controller in den Renderer integriert, so dass nur zwei Geräte miteinander kommunizieren, wie Abbildung 2.1 zeigt.

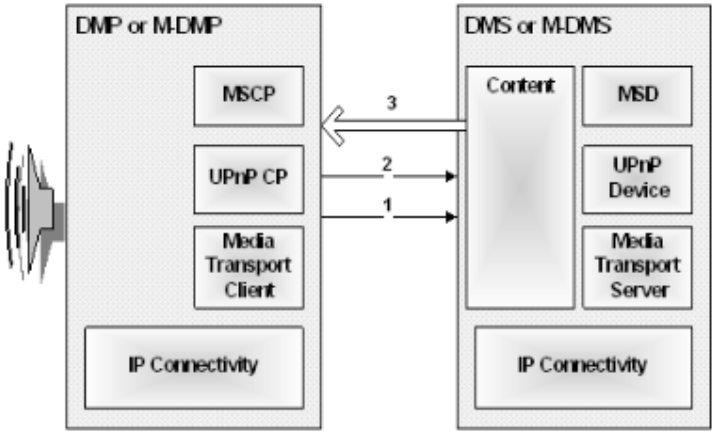

Abbildung 2.1: „2 box pull“-Methode [16]

Bei der Pull-Methode über zwei Geräte stellt der integrierte DMC eine *Browse*-Anfrage an DMS (1). Anschließend wird ein bestimmter Inhalt angefordert (2), welcher dann vom DMS geliefert wird (3).

Bei der Push-Methode in Abbildung 2.2 startet der Renderer eine Abspielsitzung und erhält vom Pushcontroller den Inhalt, welcher abgespielt werden soll (1). Der Renderer fordet anschließend die Daten an (2), welche der Controller dann überträgt (3).

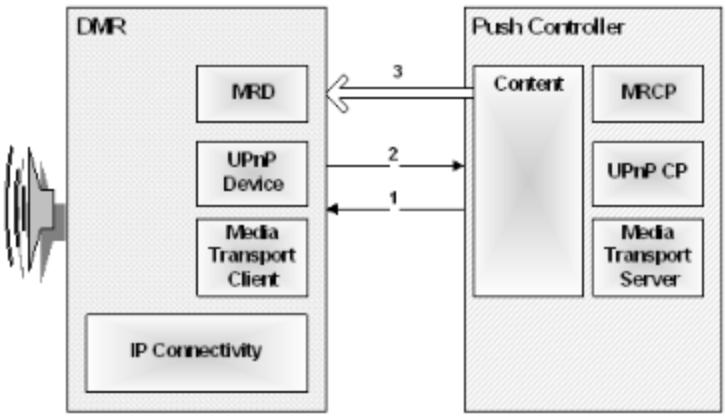

Abbildung 2.2: „2 box push“-Methode [16]

Im letzten Szenario 2.3 existieren alle drei Gerätetypen DMS, DMR und DMC. Ähnlich wie im ersten Fall fordert der DMC mittels *Browse* die Inhalte vom DMS an (1). Dieser antwortet entsprechend mit den Ergebnissen, woraufhin der DMC mit einer Anfrage beim DMR ermittelt, ob die gelieferten Inhalte abspielbar sind (2). Ist dies der Fall, kann der DMR die Inhalte beim DMS anfordern, welcher diese überträgt (4).

2.4 VDR

Der VDR ist eine Softwareentwicklung von Klaus Schmidinger, welche zur Zeit in der aktuellen stabilen Version 1.6.0 beziehungsweise der Entwicklerversion 1.7.9 verfügbar ist. Die Software ist unter Linux lauffähig und hat sehr geringe Hardwareanforderungen in der Grundausstattung. Daher kann auch ein äußerst alter Computer mit Pentium II Prozessor als Basis für den VDR dienen. Es wird lediglich eine Full-Featured (FF)-Karte benötigt, die digitale Videosignale sowohl empfangen als auch verarbeiten und ausgeben kann. Somit übernimmt diese TV-Karte die meiste Arbeit und der Prozessor wird geschont. Die Software bietet neben der üblichen Timerprogrammierung für Aufnahmen ein Electronic Program Guide (EPG), über den auch die Möglichkeit besteht, Aufnahmen zu tätigen. Außerdem unterstützt die Software zeitversetztes Fernsehen, eine Aufnahmeschnittfunktion und vieles mehr. Dabei ist die Software leicht zu erweitern, da sie eine gut beschriebene Pluginschnittstelle mitbringt. Diese Schnittstelle soll für die Entwicklung des UPnP/DLNA-Plugins genutzt werden. Eine Referenz zu den genutzten Funktionen des VDRs findet man in [8].

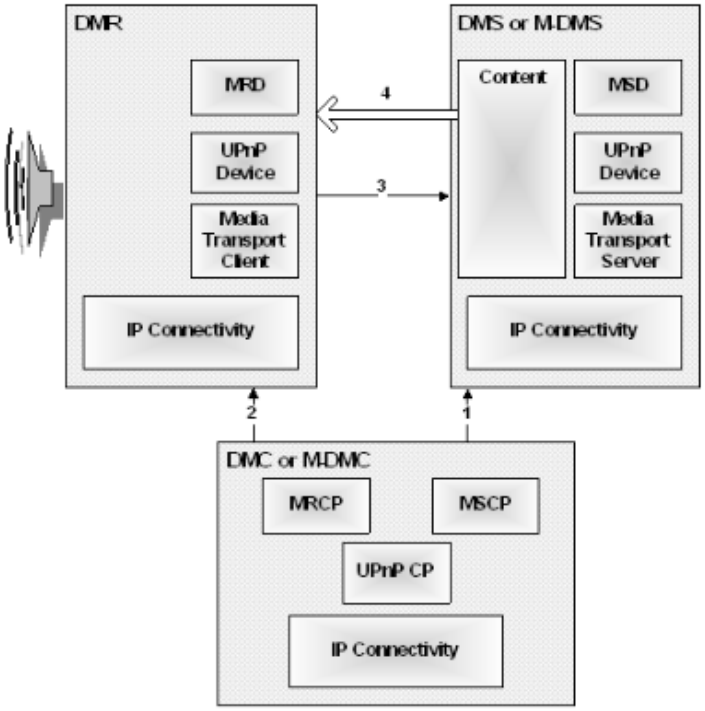

Abbildung 2.3: „2 box system“-Methode [16]

2.5 UPnP Plugin für den VDR

Die bisherige Entwicklung umfasste den grundlegenden Aufbau eines UPnP Media Servers. Dazu gehören die notwendigen Dienste und eine minimale Streamingfunktionalität. Der Server konnte in den meisten UPnP-Netzwerken gefunden werden. Jedoch verweigerten alle Clients die Suche nach Inhalten. Es ist somit nicht möglich, die Inhalte anzeigen und auswählen zu lassen. Dementsprechend fehlt es vollständig an Funktionalität, welche analysiert und verbessert werden soll. Dazu soll kurz der Aufbau des letzten Stands umrissen, deren grobe Funktionsweise erläutert und anschließend auf die aufgetretenen Probleme eingegangen werden. Im anschließenden Kapitel 3 wird der Quelltext unter die Lupe genommen. Die Gründe für das Fehlverhalten werden benannt und – wenn möglich – beseitigt.

2.5.1 Aufbau

Der Programmstruktur des Plugins ist in Abbildung 2.4 erkennbar. Dabei wurden die einzelnen Module nach Aufgaben zusammengefasst, so dass eine inhaltliche Trennung möglich ist.

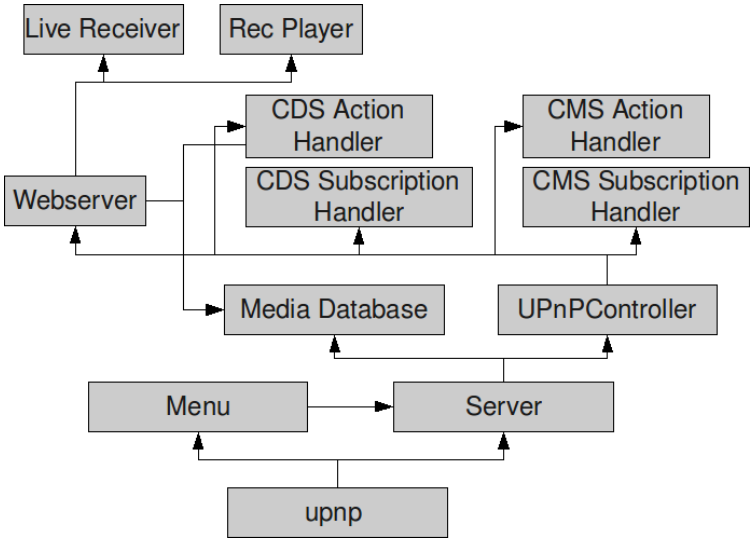

Abbildung 2.4: Struktur des Plugins

Die Klasse `upnp` bildet den Einstiegspunkt für das Plugin, welches vom VDR geladen und gestartet wird. Hierin werden die notwendigen Module vorbereitet und ebenfalls gestartet. In erster Linie ist dies der *Server*. Er verwaltet die UPnP-Funktionen und initialisiert das UPnP-Software Development Kit (SDK). Außerdem wird an dieser Stelle die Mediendatenbank `MediaDatabase` angelegt, welche automatisch alle vorhandenen TV-Kanäle und Aufnahmen in eine hierarchische Struktur lädt. Diese kann später per *Browse*-Anfrage abgerufen werden. Die Klasse `UPnPController` steuert die für einen Media Server notwendigen Dienste CMS und CDS, die jeweils durch einen *Action Handler* und einen *Subscription Handler* repräsentiert werden. Der Controller erhält alle UPnP-Anfragen aus dem Netzwerk und leitet sie an den entsprechenden Service weiter. Der *Action Handler* bearbeitet alle Nachrichten der *Control*-Phase, der *Subscription Handler* jene der Event-Phase. Relativ eigenständig agiert der interne Webserver, welcher ebenfalls über das SDK angeboten wird. Er stellt virtuelle Ordner bereit, die über einen entsprechenden Uniform Resource Locator (URL) abgerufen werden können. Hierüber läuft auch der gesamte Streamingverkehr. Das heißt, es wird je nachdem, ob es sich bei der Anfrage um einen TV-Kanal oder eine Aufnahme handelt, ein *Live Receiver* oder ein *Rec Player* gestartet.

2.5.2 Funktionsweise

Es handelt sich hierbei um eine grobe Beschreibung, wie das Plugin im bisherigen Zustand funktioniert beziehungsweise funktionieren soll. Wie im vorhergehenden Abschnitt 2.5.1 erläutert, bildet die Klasse `upnp` die Wurzel. Der VDR initialisiert das Plugin, welches die Datenbank anlegt. Das SDK und der Webserver werden einge-

richtet. Sobald eine gültige IP-Adresse über die UPnP-Funktionen ermittelt werden konnten, wird das Description eingeleitet. Die Daten werden automatisch im Netzwerk verbreitet und lassen sich über die Multicast-Adresse 239.255.255.250:1900 abrufen.

Gleichzeitig wird die Mediendatenbank initialisiert. Hier werden aus der Kanalliste des VDR alle Kanäle ausgelesen und anhand der gefundenen EPG-Daten die Metadaten ermittelt. Darunter fallen die Beschreibung, der Titel und die Langbeschreibung. Diese werden in die vorgegebene Ordnerstruktur nach 2.5 in *TV* einsortiert. Auf die gleiche Weise werden Aufnahmen geladen. Da diese eine ähnliche Struktur zeigen, geschieht das Laden analog. Sie sind anschließend in *Rec* zu finden.

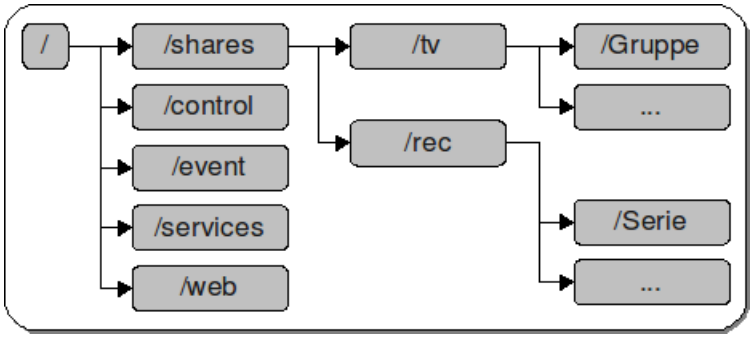

Abbildung 2.5: Virtuelle Verzeichnisse

Wird eine *Browse*-Anfrage durch einen Client gestellt, wird diese durch den *Controller* an die entsprechende Stelle geleitet. *Browse* ist eine Aktion des CDS und wird durch den *CDS Action Handler* bearbeitet. Dieser greift auf die *Media Database* zu und ruft alle Ergebnisse, die der Anfrage entsprechen, ab. Anschließend wird das Ergebnis in ein DIDL-Dokument umgewandelt und in einer SOAP-Antwortnachricht enkapsuliert.

Im nächsten Schritt kann der Client die Ressource vom Server abfragen. Dabei wird die Anfrage direkt an den Webserver gestellt, welcher anhand des Dateinamens Art und Ort der Anfrage ermitteln kann. Handelt es sich beispielsweise um einen TV-Kanal, so wird der letzte Teil der Adresse, welcher der Kanal-ID des VDR entspricht, ausgewertet und ein *Live Receiver*-Objekt erstellt. Über die üblichen Funktionen des Dateilesens kann der Webserver den Datenstrom übertragen. Da der *Live Receiver* kontinuierlich Daten liefert, die der Webserver nicht entsprechend bearbeiten kann, sind zwei Puffer notwendig. Der erste speichert die Daten, wie sie der VDR liefert. Der VDR übermittelt jedes Paket nur genau ein Mal, weshalb die Daten sofort gespeichert werden und die Funktion zurückkehren muss. Ein separater Thread kümmert sich um den zweiten Puffer, welcher durchgehend die vorhandenen Daten des ersten Puffers als Block in den zweiten Puffer verschiebt. Dieser ist nun unabhängig vom ersten Puffer, so dass keine Beeinflussung vom Webserver stattfinden kann. Die Daten werden entsprechend ausgelesen und vom Puffer gelöscht. Der Client kann die empfangenen Daten zusammenführen und darstellen.

2.5.3 Fehler und Probleme

Alle Fehler und Probleme, welche hier beschrieben werden, wurden bereits in der Vorarbeit [8] erwähnt und werden zur Vollständigkeit hier noch einmal zitiert:

> „Der Webserver des Intel-SDK arbeitet normalerweise threadorientiert. Dennoch ist es derzeit nicht möglich zwei separat ablaufende Anfragen zu stellen. Das heißt, dass bei einem aktiven Stream keine weitere Daten vom Webserver abgerufen werden können, da die `Read()`-Schleife den Webserver blockiert. Es wird kein weiterer Thread gestartet. Aktive Streams müssen vollständig geschlossen werden, um wieder Zugriff auf den Server zu erhalten. Dieser Fehler ist äußerst akut, da so weder die notwendigen Servicebeschreibungen übertragen werden können, noch irgendwelche parallelen Streams gestartet werden können.
>
> Ein weiteres Problem in diesem Zusammenhang ist, dass der Server bisher noch nicht auf Änderungen der Metadatenbank reagiert. Die Datenbank wird beim Start des Plugins vollständig erzeugt. Es gibt keine Möglichkeiten, eine Aktualisierung anzustoßen und diese Änderungen im Netzwerk preis zu geben. Ebenso fehlt die Such- und Filterfunktion, um die Ergebnisse, die der Server sendet, einschränken zu können.
>
> Im Moment ist kein Streaming von Aufnahmen möglich. Durch die Aufteilung der Aufnahmen auf mehrere Dateien, müssen die Dateizeigeroffsets neu berechnet werden, die noch fehlerhaft ist und beim Start einer Aufnahme im Speicherzugriffsfehler resultiert. Dieser Fehler sollte sich schnell durch Anpassung der Berechnung zu beheben lassen.
>
> [...] Der Quellcode [ist] teilweise unstrukturiert. Viele Speicheranforderungen werden nicht explizit wieder freigegeben. Es ist nicht ausgeschlossen, dass weitere Speicherzugriffsfehler auftreten, die noch nicht bemerkt wurden.
>
> Die geplante DLNA-Kompatibilität kann derzeit nicht sichergestellt werden, da uns mögliche Testgeräte fehlen, mit der die Funktionalität geprüft werden kann. Außerdem ist der Standard nicht frei verfügbar, so dass die Implementierung nur anhand der Open Source Bibliothek *libdlna* erfolgen kann. Weiterführende Informationen liegen uns nicht vor. Dementsprechend ist eine Fehlersuche sehr schwer.“

3 Planung und Analyse

3.1 Planung

Während der Planungsphase sollen notwendige Dokumente beschafft und die Phasen der Entwicklung umrissen werden. Umfang und Zeitaufwand der Programmierung ist nicht abschätzbar, daher ist eine Versionierungskontrolle vorzusehen, an denen der Fortschritt zu erkennen ist.

3.1.1 Beschaffung notwendiger Dokumente

UPnP

Das UPnP Working Committee hat frei verfügbare Dokumentationen und Vorschriften für die Implementierung veröffentlicht. Die Quellen für die Dokumente befinden sich im Quellenverzeichnis.

DLNA

Die Dokumente der DLNA sind leider nicht öffentlich. Sie werden nur an lizenzierte Partner herausgegeben. Während diese zu Beginn der Programmierung des vorherigen Version des Plugins noch für 500$ erstanden werden konnten, besteht diese Möglichkeit nun nicht mehr. Die Hochschule für Telekommunikation Leipzig hat mit dem Träger der Deutschen Telekom die Möglichkeit, als eingetragenes Mitglied die Dokumente kostenfrei zu beziehen. Über die Bibliothek können die Dokumente eingesehen werden.

API-Dokumentationen

Die Programmierschnittstellen der genutzten Bibliotheken können über die Webseiten der Hersteller bezogen werden. Dazu gehört auch die umfangreiche Application Programming Interface (API)-Dokumentation des Intel-SDK. Ebenfalls über die Webseite von Intel können Testwerkzeuge heruntergeladen werden[1], welche einfache UPnP-Aktionen ausführen können, darunter auch alle UPnP-AV-Funktionen. Zusätzlich kann die entwickelte Software mit dem beiliegenden *Device Validator* überprüft werden.

3.1.2 Planung Programmierung

Als Entwicklungsumgebung wird Ubuntu Linux 9.04 verwendet, worauf die aktuelle Entwicklerversion des VDR 1.7.9 und die notwendigen Bibliotheken[2] möglichst über

[1] Inzwischen leider nicht mehr direkt, da Intel den Support für UPnP eingestellt hat
[2] inklusive der Entwicklerpakete (`*-dev`-Pakete)

den Paketmanager installiert wurden. Die Programmierung erfolgt in Etappen und richtet sich nach dem Fortschritt bei der Beseitung der Fehler aus 2.5.3. Sobald eine lauffähige Version des Plugins mit den Mindestanforderungen *Browsen* und *Live-TV-Streaming* verfügbar ist, beginnt die Versionierung mit der Initialversion *0.0.1*. Für das Verwalten der Programmversionen soll eine Versionierungskontrolle eingesetzt werden, welche auch gleichzeitig für die Verbreitung des Plugins genutzt werden kann.

Hierfür wird eine entsprechende Projektwebseite eingerichtet, welche über die Adresse unter [10] erreichbar ist. Dort können über ein Ticketsystem Fehler und Funktionswünsche gemeldet und entsprechend bearbeitet werden. Die Tickets werden anhand der Priorität und Typ einer passenden Version zugeordnet. Der Fortschritt wird über die *Roadmap* ersichtlich.

Versionierung

Nach Muster *MAJOR.MINOR.MICRO*

MAJOR Hauptversionsnummer, starke Änderungen im Programm, es kann keine Kompatibilität zwischen zwei *MAJOR*-Versionen garantiert werden.

MINOR Nebenversionsnummer, kleinere Änderungen, welche die API beeinflussen könnten. Kompatibilität zwischen zwei *MINOR*-Versionen wird aber meist angenommen.

MICRO Minimalversionsnummer, kaum Änderungen, umfasst meist kleinere Bugfixes, Rechtschreibfehler oder Korrekturen in der Dokumentation.

Versionszweige

testing Tagesaktueller Zustand, äußerst instabil. Sicherung des letzten Programmierfortschritts. Keine eigene Versionierung.

unstable Aktueller Entwicklungs- und Testzweig. Kann auf Testumgebungen stabil laufen, muss aber genauer untersucht werden. Ungerade *MINOR*-Nummer.

stable Aktueller öffentlicher und stabiler Zweig. Wurde weitgehend getestet und läuft auf den meisten Umgebungen stabil. Gerade *MINOR*-Nummer.

3.1.3 Planung Testphase

Die Tests finden in Zusammenarbeit mit Mitgliedern des VDR-Portals statt. Hierfür konnten mehrere Nutzer angesprochen werden, welche UPnP-AV- beziehungsweise DLNA-fähige Geräte besitzen. Es stehen folgende Geräte zur Verfügung:

- Popcorn Hour A110
- Popcorn Hour C200
- Philips 42 PFL 9703D/10

- Sony Bravia 40W5500
- Sony Playstation 3
- Telegent TG100
- Samsung LE40B650
- Samsung i8910 HD

Die ersten beiden Geräte stehen in der Laborumgebung der Hochschule zur Verfügung. Zusätzlich können folgende Softwareclients genutzt werden:

- XBox Media Center (XBMC)
- Microsoft Windows Media Player 12 (Microsoft Windows 7)
- MythTV
- Nero Showtime

Unter Umständen können auch weitere Geräte zum Testen herangezogen werden. Da sich die Clients untereinander kaum unterscheiden dürften, ist es nicht notwendig, diese explizit zu nennen.

Die Verbreitung der Quelltexte für die Testphase findet über die Projektwebseite [10] statt, auf der sich die bereitwilligen Tester das Plugin herunterladen oder über das Git-System kopieren können. Die Installationsanleitung sowie die notwendigen Systemanforderungen liegen dem Quelltext bei.

Das Feedback der Nutzer wird über das VDR-Portal mitgeteilt oder per Ticket im Projektticketsystem. Die Anzahl der Tester ist uneingeschränkt.

3.2 Analyse

Durch eine Analyse des Programmes soll ermittelt werden, wie die in 2.5.3 aufgetretenen Fehlverhalten beseitigt werden können. Das heißt, an welchen Stellen im Quelltext sind grobe Fehler zu erkennen. Daraufhin soll aufgezeigt werden, inwieweit sich das Plugin erweitern lässt. Wenn neue Funktionen eingeführt werden sollen, muss die vorhandene Struktur weitgehend erhalten bleiben. Der Quellcode muss also skalierbar und wartbar sein.

3.2.1 Analyse des bisherigen Programmcodes

Im Folgenden werden Schwachstellen des Programmcodes gekennzeichnet, die für spätere Versionen problematisch sein können.

Server und UPnP-Controller

Die Klassen *cUPnPServer* und *cUPnPController* laufen separat, wobei der Server Funktionen des Controllers direkt aufruft, ohne dass irgendwelche Parameter oder Einstellungen beeinflusst werden. Lediglich die Einstellungen aus dem Setup werden vom Server übernommen.

Setup

Das Setup wird zwar in *cUPnPServer* abgelegt, allerdings nicht genutzt. Der Server kann eine gültige IP-Adresse aus einem angegebenen Interface generieren, welche aber nicht an das Intel-SDKweitergereicht wird. An dieser Stelle werden die Standardwerte verwendet. Das heißt, es wird immer nur das erste verfügbare Netzwerkinterface für den Server genutzt.

DLNA

Die Verwendung der *libdlna* erwies sich als nicht tauglich. Da bisher die Dokumente gefehlt haben, war es nicht möglich, ein Profil für einen Kanal exakt zuweisen zu können. Es musste grob abgeschätzt werden, welches am ehesten passen könnte. Diese große Ungenauigkeit könnte letzten Endes dazu führen, dass keine Kompatibilität gewährleistet werden kann. Des weiteren wurden viele Vorgaben, welche für eine DLNA-Tauglichkeit notwendig sind, nicht angewendet. Speziell die wichtigen Protokollinformationen, welche auch die DLNA-Profile enthalten, fehlen in den *Browse*-Ergebnissen.

Die *Device Description File* ist ebenfalls nicht nach Vorgaben der DLNA ausgelegt. Der Server wurde nicht entsprechend klassifiziert.

Browse

Im Moment werden die Ergebnisse einer *Browse*-Anfrage direkt in eine SOAP-Nachricht eingebettet. Dies ist jedoch falsch, da es sich um ein eigenständiges XML-Dokument handelt, welches nach den Regeln des World Wide Web Consortium (W3C) umgeformt werden muss, bevor es in den Nachrichtenkörper eingefügt wird. Dies wird vermutlich die Ursache für die fehlgeschlagenen Anfragen eines Clients sein.

Metadatenbank

Die Metadatenbank arbeitet mit verketten Listen, welche ineinander geschachtelt sein können, um die Ordnerstruktur abzubilden. Dies macht die Navigation beziehungsweise das Suchen nach bestimmten Einträgen äußerst kompliziert, da unter Umständen der vollständige Objektbaum abgesucht werden muss. Ferner ist das Löschen und Hinzufügen neuer Komponenten risikobehaftet, da nicht abgeschätzt werden kann, wie die Komponenten in Verbindung stehen.

Die *ObjectIDs*, welche jeweils den Einträgen zugeordnet werden, können sich über einen Neustart hinweg ändern. Sie werden auch verändert, sobald ein neues Objekt in die Datenbank eingefügt wird.

Zwingende Metaangaben, wie beispielsweise der Titel, werden nicht geprüft. Es kann vorkommen, dass ein leerer Titel übertragen wird, was ungültig ist.

3.2.2 Analyse gewünschter Funktionen

In einer Umfrage im Nutzerportal des VDR [9] soll ermittelt werden, welche Funktionen in den späteren Versionen des Plugins gewünscht werden. Dabei konnten folgende Funktionen ermittelt werden:

- DLNA-Support
- XBox-Support
- Windows Media Player 11-Support
- Radio
- Aufnahme anstoßen
- EPG
- Timeauswahl via EPG
- Timer löschen
- Spulen in Aufnahmen
- Springen in Aufnahmen (Zeit, Marken)
- Webinterface
- MP3-Support mit ID3-Tags
- Sortierung nach Genre/Titel/Album
- Webradio

3.2.3 Analyse der Skalierbarkeit

Die Konnektivität ist im Moment auf die erste Netzwerkschnittstelle des Computers beschränkt. Das heißt, sollte die erste Netzwerkkarte nicht mit dem lokalen Netzwerk verbunden sein, so ist das UPnP-Plugin nicht nutzbar.

Bei der Metadatenbank ist in diesem Entwicklungszustand die Skalierbarkeit nur sehr schwer zu erreichen, da gerade deren Struktur keine dynamischen Erweiterungen und Änderungen zulassen. Auch eine Suche ist unmöglich, denn es können hier immer nur zwei Objekte direkt miteinander verglichen werden und bei einer größer werdenden Datenbank die Vergleiche exponentiell steigen würden. Hier ist die Verwendung eines relationalen Datenbankmodells empfehlenswert.

Weil der Aufbau der Objekte sehr speziell auf TV-Kanäle und Aufnahmen zugeschnitten ist, können eigene Video- und Audiodateien nicht eingebunden werden. Dazu muss ebenfalls ein sehr speziell abgestimmtes Objekt erstellt werden, welches jeweils ein anderes Format wiedergeben kann.

4 Design des Programmes

4.1 Aufbau

Um eine verbesserte Struktur in den Quelltext einfließen und spätere Erweiterungen zuzulassen, wird der Quelltext reorganisiert. Durch eine verbesserte Sortierung und Gruppierung der Komponenten sollen Abhängigkeiten untereinander reduziert werden.

Die Klasse `cUPnPController` fließt im vollen Umfang in die bereits vorhandene Klasse `cUPnPServer` ein. Dies reduziert die Redundanz zwischen den Klassen. Außerdem werden nun alle serverspezifischen Anfragen direkt über die `cUPnPServer` geleitet. Lediglich der Webserver bleibt als eigenständige Komponente erhalten, da sie noch weitere Aufgaben übernimmt, welche nichts mit dem UPnP-Server zu tun haben. Dazu gehört auch das Verarbeiten und Anbieten der Webseiten und Dateien.

Die Klassen `ActionHandler` und `SubscriptionHandler` des CDS und CMS werden zu einem `cUPnPService` zusammengeführt. Gleichzeitig wird mit dieser Klasse eine einheitliche Schnittstelle für UPnP-Dienste vereinbart. Diese Herangehensweise soll die Erweiterbarkeit mit weiteren Diensten, wie dem AVTS, erleichtern.

Die gesamte Abwicklung des Inhalts wird über die Media Database geführt. *Browse*- und *Search*-Anfragen werden über den CDS an die Media Database weitergeleitet. Der CDS wertet dabei die SOAP-Nachrichten aus und erstellt entsprechend der Antwort der Datenbank wieder SOAP-Nachrichten.

Auch die Streaminganbindungen wurden überarbeitet und werden über die Interfaceklasse `cFileHandle` eingebunden. Hierüber soll ebenfalls eine erhöhte Skalierbarkeit erreicht werden. Neue Streamingtypen benötigen eine Resourcentypkennung, welche einen Stream eindeutig identifiziert und somit die passende Instanz erzeugt werden kann. Vorbereitete Kennungen sind

- TV-Kanäle
- Aufnahmen
- reguläre Dateien
- entfernte Resourcen via URL

Die Suchfunktion, welche durch UPnP bereitgestellt wird, ist sehr komplex. Die Grammatik wird in [14] im Abschnitt 2.5.5.1 definiert. Diese wird mittels *Boost::Spirit* Grammatikparser abgebildet. Dabei wird der Suchstring entsprechend in Teilstrings getrennt und später in eine Simple Query Language (SQL)-Anfrage umgewandelt. *Boost::Spirit* ist ein relativ umfangreiches Werkzeug, über das auch einfache Grammatiken erstellt werden können, so dass diese auch die Aufgabe des Filterns, Pfadparsens

und Abbildung der Sortierung übernehmen. Diese werden für die API nicht sichtbar über die Klassen `cSearch`, `cFilterCriteria`, `cPathparser` und `cSortCriteria` abgebildet. Die Rückgabetypen sind entsprechende Strukturen, die durch die aufrufende Klasse ausgewertet werden muss.

Alle weiteren überarbeiteten Klassen werden in den folgenden Kapiteln näher erläutert. Hauptsächlich werden dort die Datenbankanbindung und die Verwaltung von Objekten behandelt.

5 Datenbank

5.1 Wahl der Datenbank

In der Vorbereitung wurden mehrere Datenbanksysteme auf entsprechende Eignung für das Plugin geprüft. Hier sollte Wert auf folgende Punkte gelegt werden:

Relational, so dass Beziehungen zwischen Objekten einfach und dynamisch hergestellt werden können. Diese können durch Fremdschlüssel in den entsprechenden Tabellen erreicht werden. Relationale Datenbanken eigenen sich besonders, da komplexe Anweisungen, wie die einer *Search*-Anfrage, effizient bearbeitet werden können.

Leicht-gewichtig, um den Aufwand in Grenzen zu halten und auch eine spätere Portierung auf Systeme mit geringem Festplattenspeicher zu ermöglichen. Ein vollständiges Datenbanksystem, welches auch eine umfangreiche Businesslogik beinhaltet, wäre nicht gerechtfertigt für einen normalen Hausgebrauch.

Einfach, so dass keine komplexe API genutzt werden muss, um kleine Aufgaben zu bewältigen. Wenn eine einfache *SELECT*-Anfrage mehr als 10 Zeilen Quelltext benötigt, ist die Datenbank gänzlich ungeeignet. Der Grund hierfür liegt in der schweren Erweiterbarkeit des Quellcodes in späteren Versionen.

Gute Datenbankverwaltung, welche man über zusätzliche kleine Werkzeuge realisieren könnte. Diese sollen die Fehlersuche in der Datenbank erleichtern.

Transaktionssteuerung, obwohl dieser Punkt auch optional sein könnte, hat er eine große Bedeutung, denn wenn im Verlauf des Betriebes ein Objekt in die Datenbank eingefügt wird und aufgrund eines Fehlers in der Verarbeitung die Objektstruktur zerstört werden sollte, müssen diese fehlerhaften Einträge möglichst automatisch beseitigt werden. Notfalls ist eine manuelle Kontrolle über falsche Einträge notwendig. Dem Benutzer darf nicht zugemutet werden, dass er die Datenbank selbstständig bearbeiten muss, damit diese wieder funktionstüchtig ist. Ein Stichwort hierfür wäre eine gewisse Selbstheilung der Datenbank.

Fremdschlüsselbedingungen, oder Foreign Key Constraints, sollten von der Datenbank ebenfalls mitgebracht werden, so dass Abhängigkeiten zwischen den Tabellen automatisch aufgelöst werden können, ohne dass eine manuelle Überprüfung im Quelltext notwendig wird.

Wenn man alle Punkte zusammenfasst, wurden zunächst folgende Datenbanken als geeignet empfunden:

- MySQL 5
- SQLite 3

MySQL 5 ist eine sehr umfangreiche Datenbank, welche auf sehr vielen Systemen bereits eingesetzt wird, so dass eine zusätzliche Installation häufig nicht notwendig ist. Gleichzeitig bietet sie aber eine sehr einfache und gut strukturierte Datenanbindung über *C++*. *SQLite 3* hingegen ist, wie der Name schon impliziert, äußerst leichtgewichtig und bringt mit weniger als einem halben Megabyte an zusätzlichen Oberhead sehr geringe Anforderungen an den Speicherplatz mit. Die Datenbankanbindung über *C++* lässt sich mit weniger als fünf Zeilen inklusive Öffnen und Schließen der Datenbank beschreiben. Obwohl *SQLite 3* keine Fremdschlüsselbedinungen unterstützt, können diese über sogenannte Trigger simuliert werden, welche ebenfalls sehr einfach definiert werden können und eine dynamische Behandlung von Beziehungen ermöglichen.

Die Entscheidung fiel letztendlich auf *SQLite 3*, da es neben den gerade erwähnten Eigenschaften auch gleichzeitig eine gewisse Systemunabhängigkeit mit sich bringt. Alle Daten werden in einer einzelnen Datei abgespeichert, welche notfalls zu Fehlerfindungszwecken ausgetauscht werden kann. *MySQL* bietet diesen Vorteil nicht beziehungsweise nur bedingt, da hier notfalls auch ein Datenbank-Abbild der benötigten Daten erzeugt werden kann. Allerdings sollte auch Wert auf die Benutzerfreundlichkeit gelegt werden, weil nicht automatisch angenommen werden kann, dass ein normaler Benutzer einen solchen Dump erzeugen kann.

5.2 Datenbankstruktur

Die Datenbankstruktur wird in der folgenden Abbildung 5.1 gezeigt. Dabei werden Fremdschlüsselbedingungen über Trigger überwacht, welche automatisch auslösen, sobald ein bestimmtes Ereignis eintritt. Der Aufbau der Trigger wird im Abschnitt 5.3 erläutert.

5.3 Trigger

SQLite 3 unterstützt selbst keine *Foreign Key Constraints*, aber Trigger mit denen die gleichen Aufgaben bearbeitet werden können. Für eine vollständige Simulation eines *Foreign Key Constraints* sind drei Trigger notwendig, welche die Abfragen *INSERT*, *UPDATE* und *DELETE* überwachen.

5.3.1 allgemeine Beziehungen

Wie aus Abbildung 5.1 erkenntlich wird, bestehen Beziehungen zwischen den einzelnen Objektklassen. Der Ableitungsbaum wird mittels Ableitungstriggern kontrolliert.

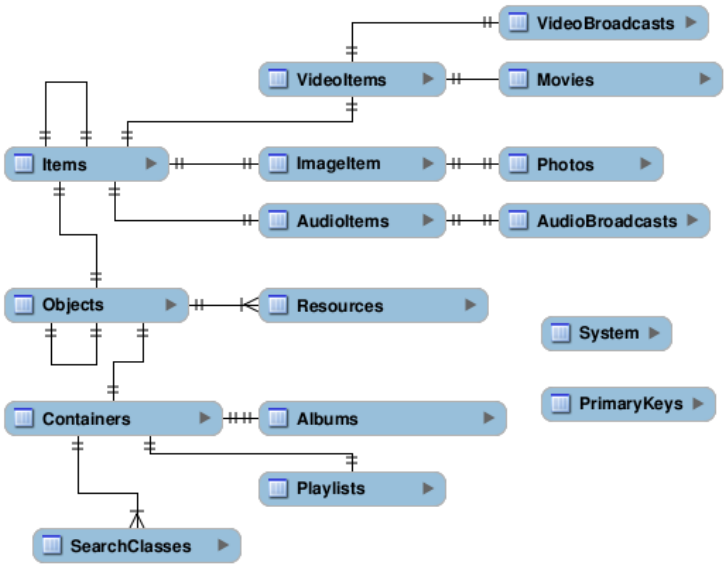

Abbildung 5.1: ER-Diagramm

INSERT-Trigger

Der *INSERT*-Trigger überprüft vor dem Anlegen eines Datensatzes, ob das Elternelement *Parent* existiert und ob die Klasse mit der Tabellenableitung übereinstimmt. Ein Beispiel für die Klasse *VideoItem*:

```
1  CREATE TRIGGER IF NOT EXISTS Items_I_VideoItems
2  BEFORE INSERT ON VideoItems
3  FOR EACH ROW BEGIN
4   SELECT CASE WHEN (
5    ((SELECT ObjectID FROM Items WHERE ObjectID=NEW.ObjectID) IS NULL) OR
6    ((SELECT ObjectID FROM Objects WHERE ObjectID=NEW.ObjectID
7      AND Class LIKE 'object.item.videoItem%') IS NULL)
8    )
9   THEN
10    RAISE(ROLLBACK, 'INSERT on table Items failed due constraint
11        violation on foreign key ObjectID')
12  END;
13 END;
```

Listing 5.1: INSERT Trigger für VideoItem auf Item

Die Überprüfung in Zeile 5 ist notwendig, da hiermit nachgesehen wird, ob das Objekt in der entsprechenden Elterntabelle existiert. Die `LIKE`-Klausel in Zeile 7 erlaubt alle Klassen und deren Ableitungen, die von einem `object.item.videoItem` abstammen, also zum Beispiel auch `object.item.videoItem.movie`. Somit können keine Elemente in den abgeleiteten Tabellen eingefügt werden, die nicht explizit von dieser Klasse abstammen.

UPDATE-Trigger

Der *UPDATE*-Trigger funktioniert analog zum *INSERT*-Trigger:

```
1  CREATE TRIGGER IF NOT EXISTS Items_U_VideoItems
2  BEFORE UPDATE ON VideoItems
3  FOR EACH ROW BEGIN
4   SELECT CASE WHEN (
5   ((SELECT ObjectID FROM Objects WHERE ObjectID=NEW.ObjectID
6     AND Class LIKE 'object.item.videoItem%') IS NULL)
7   )
8   THEN
9     RAISE(ROLLBACK, 'UPDATE on table Items failed due constraint
10        violation on foreign key ObjectID')
11  END;
12 END;
```

Listing 5.2: UPDATE Trigger für VideoItem auf Item

Hier muss allerdings nur geprüft werden, ob es in der Wurzeltabelle einen Eintrag gibt, welcher auf die neue Objekt-ID passt und bei dem die Klasse mit der entsprechenden Tabelle, die aktuallisiert werden soll, übereinstimmt.

DELETE-Trigger

Für den *DELETE*-Trigger gibt es zwei Herangehensweisen. Die erste ist ein kaskadierendes Löschen, wobei alle Elemente aus den Tabellen vollständig gelöscht werden, zwischen denen Beziehungen bestehen. Das heißt, sobald ein Objekt aus der Tabelle `VideoItems` gelöscht wird, werden auch die Einträge den Tabellen `VideoBroadcasts` beziehungsweise `Movies` entfernt. Dies geschieht aber nur auf die darunterliegende Ebene. Die Einträge in den Elterntabellen bleiben erhalten.

```
1  CREATE TRIGGER IF NOT EXISTS Items_D_VideoItems
2  BEFORE DELETE ON Items
3  FOR EACH ROW BEGIN
4   DELETE FROM VideoItems WHERE ObjectID=OLD.ObjectID;
5  END;
```

Listing 5.3: DELETE CASCADE Trigger für VideoItem auf Item

Der Löschtrigger wirkt anders als *INSERT* und *UPDATE* auf die Elterntabelle, so dass nicht die Tabelle `VideoItems` sondern `Items` geprüft wird. Wenn ein Löschversuch auf diese Tabelle ausgeführt werden soll, wird vorher in der referenzierenden Tabelle nach Einträgen gesucht, die vorher nacheinander gelöscht werden.

Auf der anderen Seite kann man das Löschen auch mittels Fehlermeldung abbrechen, sobald in der Kindstabelle noch Einträge gefunden wurden. Der Aufbau ist grundsätzlich gleich, wie der eines *INSERT*- oder *UPDATE*-Triggers und wird daher nicht noch einmal gesondert erläutert.

5.3.2 reflexive Beziehungen

Es existieren auch Beziehungen innerhalb einer Tabelle, also reflexive Beziehungen. Dazu gehören die Eltern-Kind-Objektbeziehung in der Tabelle `Objects` und die Referenzobjektbeziehung in der Tabelle `Items`. Diese unterscheiden sich kaum von den vorher genannten Triggern. Es wird lediglich die `ObjectID` mit der `ParentID` verglichen, um zu schauen, ob das angegebene Elternelement existiert. Als zusätzlicher

erlaubter Wert kann hier `-1` angegeben werden, um die Wurzel oder ein nicht vorhandenes Element zu symbolisieren. Referenzobjekte in der Tabelle `Items` dürfen niemals in beide Richtungen zeigen. Dies würde eine Schleife erzeugen, wodurch der Löschtrigger immer ausgelöst wird und das Element nicht aus der Tabelle entfernbar ist. Die Struktur wäre dann vollständig zerstört.

5.3.3 weitere Trigger

Zusätzlich zu der Gewährleistung der Integrität zwischen den Objekttabellen werden weitere Trigger eingesetzt, welche den Zusammenhang zwischen anderen Tabellen gewährleistet. Dazu zählen die Folgenden:

Primärschlüsseltrigger Diese Trigger verwalten die Primärschlüssel der Objekttabellen.

Trigger für Ressourcen Diese Trigger überwachen, ob die Objekte, denen die Ressource zugeordnet werden, soll existieren.

Trigger für Wurzelobjekte Hiermit wird bei jeder Einfüge- oder Aktuallisierungsoperation geprüft, ob die Eltern-ID einzigartig ist. Es dürfen keine zwei Elemente mit `ParentID=-1` in der Datenbank vorkommen.

6 UPnP Objekte

6.1 Struktur

Die UPnP-Objekte verwenden die Baumstrukturen in Abbildung 6.1 für Container-basierte und 6.2 für Item-basierte Objekte:

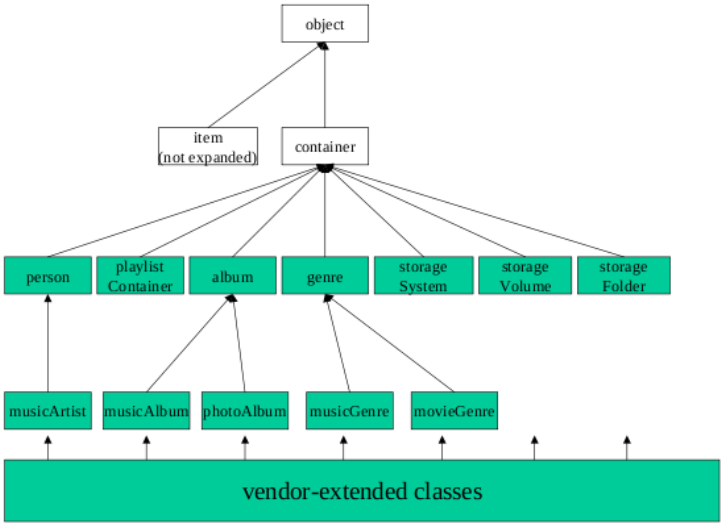

Abbildung 6.1: Container-basierte Objekte[14]

Den genauen Aufbau der Methoden und Eigenschaften können der Quelltextdokumentation und den dortigen UML-Diagrammen entnommen werden [20].

6.2 Objekt-relationales Mapping

Die Kommunikation mit der Datenbank findet über objekt-relationalem Mapping statt, wobei jedes Objekt mit einer bestimmten Tabelle zusammenhängt. Die Daten werden durch die Media-Database aus der Datenbank mittels Vermittlermuster geholt.

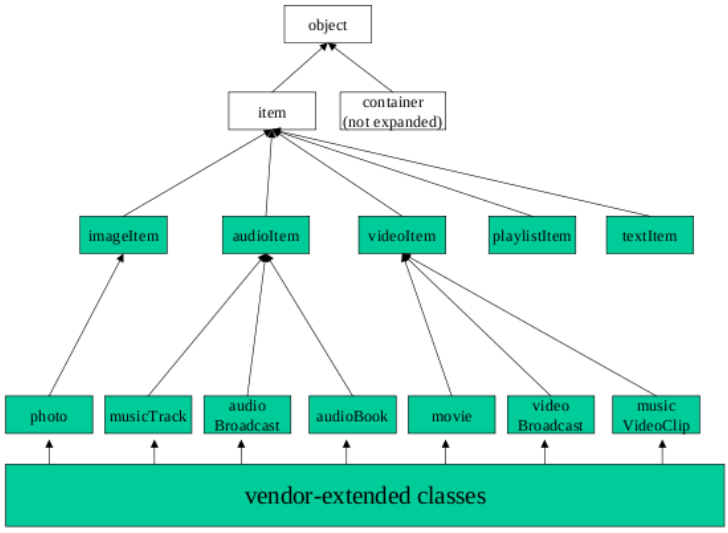

Abbildung 6.2: Item-basierte Objekte[14]

6.2.1 Mediator

Das Mediatormuster, oder auch Vermittlermuster, wird bei Anwendungen eingesetzt, wo der Client nicht zwangsläufig Kenntnis vom Aufbau der internen Logik haben muss. Hierfür werden austauschbare Mediatoren eingesetzt, welche die Funktionen beider Seiten kennen. Somit kann der Client auf den Mediator zugreifen, welcher die Anfrage entsprechend umwandelt und bei der Gegenseite die gewünschten Informationen abruft.

Im Projekt wird der Mediator für das Laden eines Objektes aus der Datenbank verwendet. Für jede Klasse existiert ein separater Vermittler, welcher die vollständigen Daten aus der Datenbank abrufen kann. Sie besitzen die gleiche Objektbaumstruktur, wie die Objekte selbst auch.

Der Mediator ist in der Lage Objekte anzulegen, zu ändern oder zu löschen. Dafür wird eine einheitliche Schnittstelle vereinbart, die `cMediatorInterface` heißt. Weitere Informationen dazu in der Quelltextdokumentation [20].

6.2.2 Mediatorfactory

Da es insgesammt über 25 mögliche Objektklassen gibt, wobei jede Ableitung einzeln gezählt wird, benötigt man eine weitere Struktur, welche in der Lage ist, zu erkennen, welcher Mediator für das gewünschte Objekt zuständig ist. Dazu wird eine Fabrik eingesetzt, welche passend zum Objekt eine Instanz des Mediators bereithält.

Die Fabrik verfügt über die gleichen Methoden und Funktionen wie ein Mediator. Über ihn wird der Mediator abgerufen, so dass keine direkte Kommunikation mit den

Mediatoren stattfindet. Die Fabrik nutzt dazu die von `cMediatorInterface` bereitgestellten Schnittstellen.

6.3 Media Database

Die Mediadatabase spielt dabei die Rolle des Objektcaches, welcher Zugriff auf alle Mediatoren über die Mediatorfabrik besitzt und somit auch die Verbindung zur SQLite-Datenbank bereitstellt. Sobald ein Objekt geladen wurde, wird es automatisch in den internen Objektcache geladen, so dass ein lokale Instanz verfügbar ist, die bei Bedarf ohne erneutes Laden verfügbar ist.

Im Moment wird der vollständige Objektbaum aus der Datenbank geladen, sobald ein Element gecached wird. Dies wird durch die Abhängigkeit zwischen den Objekten bedingt, welche ebenfalls abgerufen werden, sobald sie angefordert werden. Es ist also noch nicht möglich, nur einen Teilbaum im Speicher zu halten.

7 Streaming

7.1 Live-TV

Beim Streaming von Live-TV wird die gleiche Anbindung genutzt wie im alten UPnP-Plugin. Daher wird der Live-Receiver nicht erneut beschrieben. Geändert hat sich das Einlesen der Kanäle und des EPGs.

7.1.1 Kanalliste

Wie auch in der vorhergehenden Version werden alle Kanäle vollständig zu Beginn in die Datenbank eingetragen. Dabei werden allerdings nicht mehr schon zu Beginn die EPG-Daten ausgelesen, sondern lediglich alle zwingend erforderlichen Daten, um einen gültigen DLNA-fähigen TV-Kanal zu repräsentieren. Somit müssen Angaben, wie Kanalname, Kanalnummer und die Senderbezeichnung im Titel gesetzt werden.

Eine automatische Änderung der Kanalliste wird noch nicht überwacht. Sollten also während des Betriebs des VDR neue Sender hinzukommen, werden diese erst berücksichtigt, sobald der Server neugestartet wird. Ebenfalls werden sich ändernde Programm-IDs nicht berücksichtigt. Unter Umständen existiert ein Kanal unter einer der gespeicherten Program Identifier (PID) nicht mehr und kann durch einen UPnP-Client nicht mehr aufgerufen werden.

7.1.2 EPG-Daten

Neu ist, dass die EPG-Daten nun dynamisch aktuallisiert werden, sobald eine Sendung beginnt. Hierfür wird in regelmäßigen Abständen die Scheduletabelle auf Änderungen überprüft und anschließend jeder Kanal aktuallisiert, der von den neuen Daten betroffen ist.

7.2 Aufnahmen

Aufnahmen werden auf die gleiche Art und Weise eingebunden, wie im alten Plugin. Jedoch wird diese Methode noch nicht unterstützt, da das Auslesen der Metadaten noch unvollständig ist. Hier müssen noch Informationen zu den DLNA-Profilen zusammengestellt werden. Die Vorbereitungen dazu wurden bereits getroffen und werden im nächsten Abschnitt 7.3 erörtet.

7.3 Eigene Dateien

Das Plugin soll zukünftig auch eigene Dateien unterstützen, welche nicht durch den VDR angeboten werden. Damit soll der Anfrage nach Unterstützung von MP3-Dateien nachgegangen werden. Allerdings sollen so viele Formate wie möglich angeboten werden können, sofern ein entsprechendes DLNA-Profil ermittelt werden kann.

7.3.1 DLNA unterstützte Dateien

Da die DLNA die Unterstützung vieler Formate sehr stark eingegrenzt hat, so dass zertifizierte Geräte ein Minimum an Formaten abspielen müssen, soll zunächst großen Wert auf die Einbindung dieser Formate gelegt werden. Hierfür wird *ffmpeg* eingesetzt, um zusätzliche Informationen zum Streamtyp zu erhalten. Es ist in der Lage eine große Vielzahl von Medienformaten zu erkennen und zu analysieren. Entsprechend soll es die notwendigen Eigenschaften für ein bestimmtes Profil ermitteln. Im Projekt wird die Klasse `cAudioVideoDetector` zur Bewältigung dieser Aufgabe eingesetzt.

7.3.2 Nicht unterstützte Dateien

Formate, die kein passendes Profil aufweisen, können von den Mediaplayern unter Umständen nicht abgespielt werden. Sie müssen transkodiert werden, was sehr aufwändig ist und viel Rechenleistung erfordert. Diese Funktion wird durch das Plugin nicht gewährleistet und wird vermutlich erst in einer viel späteren Version eingeführt werden.

8 Übersicht des Quelltextes

8.1 Revidierte Programmstruktur

Die in Abschnitt 2.5.1 erläuterte Programmstruktur wurde mit den oben beschriebenen Änderungen überarbeitet. Die daraus resultierende neue Struktur zeigt die Abbildung 8.1. Die Zusammenfassung mehrerer Klassen in einzelne aufgabenorientierte Klassen führt dazu, dass die Beziehungen und Abhängigkeiten reduziert werden konnten.

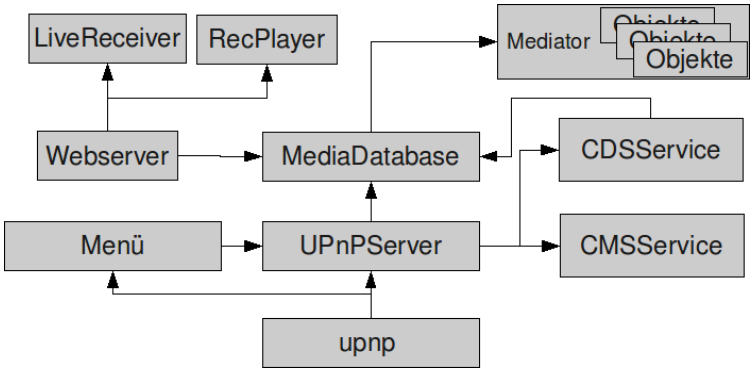

Abbildung 8.1: Struktur des neuen Plugins

8.2 Ablauf des Plugins

Das Plugin funktioniert nach wie vor nach dem selben Ablauf, wie die vorherige Version. In Abbildung 8.2 wird das Abrufen von einer Datei gezeigt. Das Diagramm stellt ein „3-box“-System dar, wobei Controller und Renderer grundsätzlich auch zusammengefasst werden können, um ein „2-box“-System zu erhalten. Da kein AVTransport-Service implementiert wurde, fehlen die optionalen Verbindungseinrichtungen zwischen Renderer und Server. Stattdessen erfolgt eine standardmäßige Datenübertragung mittels HTTP-GET. Auf der linken Seite wird angezeigt, welche Klasse für die aktuelle Aufgabe verantwortlich ist. Auf den Zeitpfeilen wird die Aufgabe anzeigt, die das entsprechende Gerät abarbeitet. Pfeile, zwischen den Geräten stellen kommunikativen Datenaustausch dar. Dies sind Aktionsanfragen und -antworten.

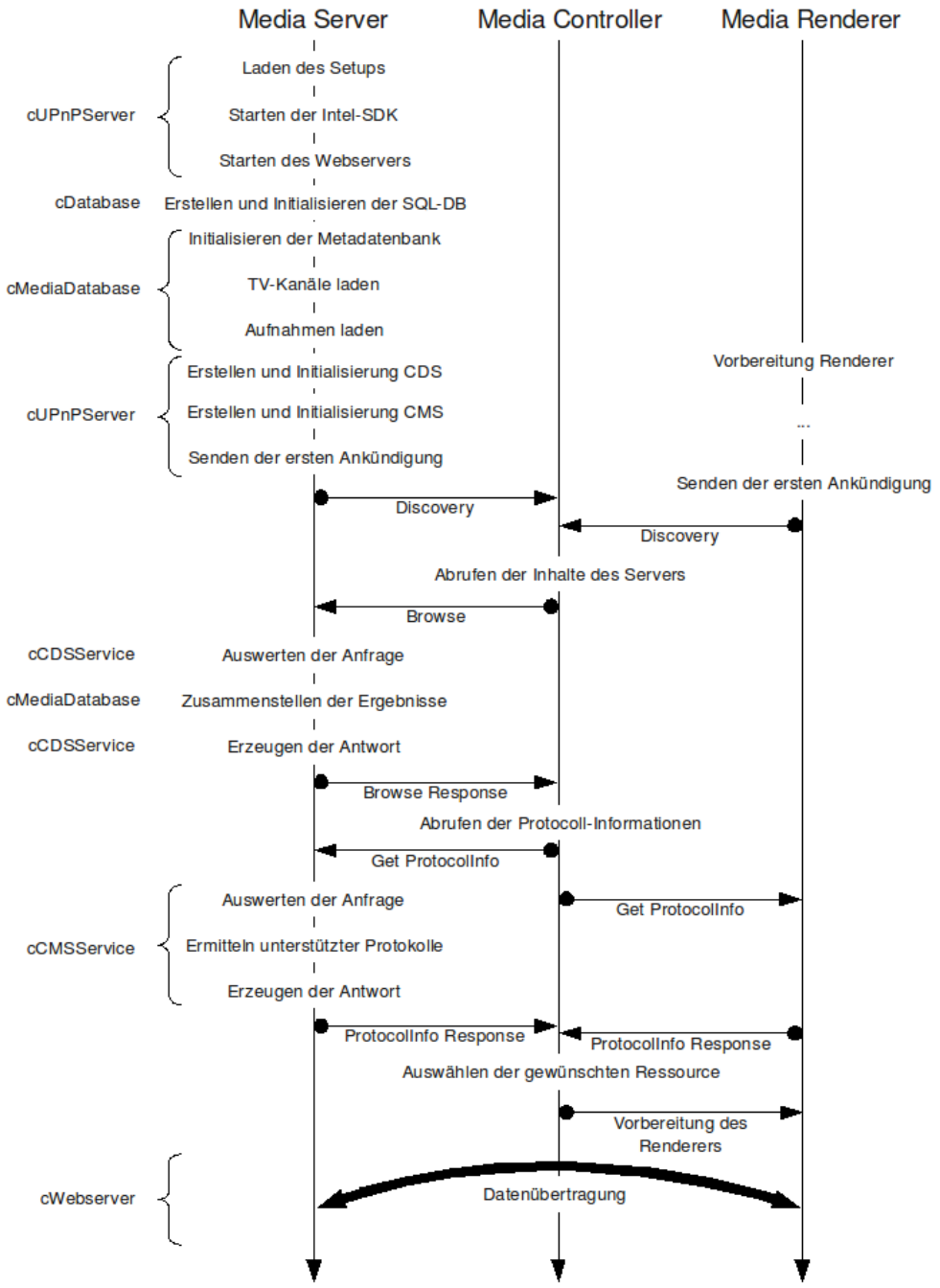

Abbildung 8.2: Ablaufplan des Plugins

9 Tests und Wartung

9.1 Testen der Umgebung

In der Hochschule ist eine Testumgebung vorhanden, welche über einen Router mit DHCP-Server, einem Fernsehgerät und einer Popcorn Hour A-100 beziehungsweise C-200. Beide Testgeräte sind UPnP- und DLNA-fähig. Der Server wird auf einem Dell XPS M1330 Laptop mit einer Hauppauge Nova-T DVB-T USB-TV-Karte betrieben. Das Setup sieht eine 54MBit/s Wireless Lokal Area Network (WLAN)-Verbindung zwischen Laptop und Router und eine direkte 100MBit/s schnelle Verbindung über CAT5-Kabel zwischen Router und dem Streaming Client vor. Da nur eine DVB-T-Karte vorhanden ist, kann immer nur ein einzelner Transponder eingestellt werden. Allerdings sind auf jedem dieser Transponder mehrere Kanäle mittels Multiplexing zusammengefasst. Es lassen sich demnach trotzdem mehrere unterschiedliche Kanäle abrufen, sofern sie die gleiche Frequenz besitzen. Als Beispiel können die Kanäle ZDF und ZDFInfokanal gleichzeitig angezeigt werden. Die Anwendung sieht zunächst eine Übertragung gleicher und unterschiedlicher Kanäle vor. Die Beobachtungen dabei werden in Tabelle 9.1 gezeigt.

Weitere Tests konnten in Zusammenarbeit mit Fabian Sattler gemacht werden. Er konnte die Unterstützung mit vielen weiteren Geräten prüfen, dazu zählen:

- XBox Media Center
- Microsoft Windows Media Player 12 (auf Windows 7)
- Linksys KISS-1600
- Pioneer BDPLX70A

Über das VDR-Portal konnten ebenfalls freiwillige Tester gefunden werden, welche mit diesen Geräten testen können:

- Sony Playstation 3
- Samsung UExxB8090
- Samsung LE40B650
- Samsung LE37B679
- Samsung LE46B650

Eigenschaft	Beschreibung
Bildfehler	Häufige Bildfehler (Artefakte) bei geringer Signalqualität, Ursache ist eine schlechte DVB-T Verbindung.
Fehlende Streamsynchronisation	Manchmal kann der VDR nicht mit dem Stream synchronisieren, so dass der Ausgabepuffer leer bleibt. Es kann dann kein Bild angezeigt werden.
Abbruch des Streams	In den ersten Versionen des Plugins bricht der Stream nach unbestimmter Zeit wegen leerem Puffers ab. Dies konnte jetzt durch eine entsprechende Überprüfung des Pufferinhalts geändert werden.
Abgestürzte Streamingverbindungen	Wenn ein Stream von der Gegenstelle nicht sauber beendet wird, bleibt dieser weiterhin aktiv.
Lange Pufferdauer	Auf manchen Clients dauerte das Puffern sehr lange, teilweise bis in den Minutenbereich. Vermutlich liegt das bei Live-TV an der fehlenden Größe der Dateien, so dass der Client nicht mehr feststellen kann, wann er den Stream starten kann.

Tabelle 9.1: Beobachtungen bei der Übertragung

Dabei wurden bei den meisten Geräten die Meldung „Nicht unterstütztes Format“ gemeldet, was bisher nicht gelöst werden konnte. Es kann vermutlich auf fehlende HTTP-Header zurückgeführt werden, welche zusätzlich zum Protokollinfo-Feld das DLNA-Profil enthält.

9.2 Wartung

Wie bereits in Abschnitt 3.1.2 beschrieben, können über das Ticketsystem auf der Projektwebseite [10] Fehler mitgeteilt werden, welche dann entsprechend bearbeitet werden können. Der Programmquelltext ist so gestaltet, dass zukünftige Änderungen und Erweiterungen recht schnell eingepflegt werden können. Sollte im Verlauf der Entwicklung der Umfang des Plugins so drastisch zunehmen, dass eine alleinige Programmierung nicht mehr möglich ist, können weitere Programmierer in das Projektteam eingespannt werden. Die Wartungsphase findet parallel zur normalen Entwicklungsphase statt, da jeder erkannte Fehler sofort in eine neue Version einfließt. Ein feste zyklische Veröffentlichung neuer Versionen ist nicht vorgesehen, so dass eine neue Version herausgebracht wird, sobald die Stabilität auf bestimmten Systemen nach ausgesuchten Kriterien bewiesen werden konnte.

10 Projektabschluss

10.1 Zusammenfassung

Die im Pflichtenheft vorgegebenen Ziele konnten weitgehend erfüllt werden. Das Kriterium der zu unterstützenden Clients wurde sogar mit einer sehr guten Quote getroffen. Native UPnP-Clients ohne Einschränkung der Formate durch die DLNA unterstützen den Server fast ausnahmslos. Die gravierendsten Fehler wurden beseitigt und der Quelltext restrukturiert. Die Anzahl der Abstürze konnten drastisch reduziert werden, so dass auf der lokalen Testumgebung nur noch in sehr seltenen Fällen das Plugin wegen Segmentierungsfehlern abbricht.

10.2 Ausblick

Das Plugin wird stetig weiter entwickelt, so dass zukünftig die meisten Medienformate der DLNA angeboten werden können. Ebenfalls soll das Spektrum der unterstützten Geräte konsequent erweitert werden. Hier sind gerade DLNA-zertifizierte Fernsehgeräte interessant, da diese kein zusätzliches Gerät benötigen, welches die Inhalte wiedergibt. Bisher fehlt dem Plugin noch die Unterstützung für Aufnahmen und eigene Dateien, was aber bereits vorbereitet wurde und nach Beseitigung bestehender Fehler eingeführt wird. Auch das Webinterface muss umgesetzt werden, so dass später auch Aufnahmen getätigt werden können. Hier könnten unter Umständen die Neuerungen des DLNA-Standards Version 2.0 einfließen, die diese Aufgabe nativ unterstützen. Die EPG-Anzeige ist ebenfalls davon betroffen. Im Moment können immer nur die aktuellesten Ereignisse angezeigt werden, zukünftige Sendungen fehlen vollständig. Dem Benutzer soll später größere Freiheiten in den Einstellungen gewährt werden. Hierfür wird das Setup um entsprechende Punkte, wie dem Aktivieren beziehungsweise Deaktivieren des Servers im laufenden Betrieb oder Freigeben bestimmter Dateien, erweitert.

10.3 Fazit

Das UPnP-Plugin kann sich zu einem der wichtigsten VDR-Plugins entwickeln, da es die Einführung des verteilten Fernsehens für Heimanwender stark vereinfacht. Bisher werden hauptsächlich auch als Clients VDR-Computer eingesetzt, welche entsprechend eingerichtet werden müssen. Dies könnte zukünftig entfallen, da bereits mit wenig Geld ein UPnP-fähiges Gerät angeschafft werden kann, welches diese Dinge bereits unterstützt.

Literaturverzeichnis

[1] VDR Wiki: http://www.vdr-wiki.de

[2] VDR Quellen (1.7.8, Entwicklerversion): ftp://ftp.cadsoft.de/vdr/Developer/vdr-1.7.8.tar.bz2

[3] FFMPEG Project Homepage: http://ffmpeg.org

[4] FFMPEG API Dokumentation: http://ffmpeg.org/documentation.html

[5] FFMPEG Tutorial: How to write a video player in less than 1000 lines: http://www.dranger.com/ffmpeg/

[6] Simple Directmedia Layer: http://www.libsdl.org/

[7] Digital Living Network Alliance: http://www.dlna.org

[8] Andreas Günther, Denis Loh: *Entwicklung eines UPnP Plugins für den VDR*, 2009

[9] VDR Portal: http://www.vdr-portal.de/board/thread.php?postid=829628#post829628, 16.07.2009

[10] VDR UPnP Plugin, Projektwebseite: http://upnp.vdr-developer.org, 05.11.2009

[11] UPnP Forum: *UPnP Device Architecture 1.0*, 2008

[12] UPnP A/V Working Committee: *UPnP AV Architecture:0.83*, 2002

[13] UPnP A/V Working Committee: *MediaServer:1 Device Template Version 1.01*, 2002

[14] UPnP A/V Working Committee: *ContentDirectory:1 Service Template Version 1.01*, 2002

[15] UPnP A/V Working Committee: *ConnectionManager:1 Service Template Version 1.01*, 2002

[16] Digital Living Network Alliance: *DLNA Networked Device Interoperability Guidelines Version 1.5*, Okt. 2006

[17] EtherGuide Systems: MPEG-TS Packets Documentation http://www.etherguidesystems.com/

[18] Unbekannter Autor: Portable SDK for UPnP Devices (libupnp 1.6.6), `http://pupnp.sourceforge.net/`,
Stand: 08.06.2009

[19] M. Jeronimo, J. Weast: *UPnP Design by Example*, 1. Auflage, 2003, Intel Press

[20] VDR UPnP Plugin, Source Code Documentation (Doxygen): `http://upnp.vdr-developer.org/docs/`, 12.11.2009