Interaktive Fernsehszenarien unter Verwendung von Smart TV API und externen Devices

Bernadette Zlabinger

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2013

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 30. Juni 2013

Bernadette Zlabinger

Inhaltsverzeichnis

Kurzfassung

Das Interesse an Zusatzinformationen zu einer laufenden TV-Sendung wird immer größer. Es gibt zwar zahlreiche Quellen für Informationen, wie Wikipedia, Google Maps und IMDb, allerdings sind diese nicht an die TV-Sendung gebunden.

Es wird oft viel Zeit in die Suche der Zusatzinformationen investiert. Häufig werden die gewünschten Informationen nicht gefunden, oder der Zuseher ist zu sehr abgelenkt, um die laufende Sendung weiter zu verfolgen.

Die vorliegende Arbeit stellt daher ein System vor, mit dem es für einen Anwender möglich ist, parallel zu einer Sendung synchronisiert Zusatzinformationen auf einem Tablet zu empfangen. Beispielsweise werden Informationen zu einem Schauspieler, der gerade am Bildschirm zu sehen ist, am Tablet angezeigt. Außerdem wird einer Community die Möglichkeit geboten, diese Zusatzinformationen selbst zu erstellen und für jeden zugänglich zu machen.

Der Erhalt der Zusatzinformationen zum richtigen Zeitpunkt bietet den Vorteil, dass ein direkter Bezug zum Sendungsinhalt besteht und die Relevanz der Metainformationen besser nachvollzogen werden kann. Um eine uneingeschränkte Darstellung des Fernsehbildes zu gewährleisten, werden die Informationen auf einem eigenen Gerät dargeboten. Die miteinbezogene Community erweitert und wartet vorhandenen Zusatzinformationen.

Abstract

The demand for additional information about currently viewed TV programmes is increasing constantly. There are numerous sources of information, considering Wikipedia, Google Maps and IMDb, but these are not related to the TV programmes.

A lot of time is invested in search for additional information. Often the desired information cannot be found or the viewer is distracted too much to pay attention to the currently viewed TV programmes.

This thesis therefore introduces a system that allows a user to receive synchronised additional information parallel to a TV programme via tablet. By way of example a user gets information about an actor who is currently on the TV screen on his tablet. Moreover, a community has the possibility to create and share this additional information.

Receiving additional information in time offers the advantage of direct reference to the TV programme. The user is able to perceive the relevance of the information provided. The information is presented on an external device so that the user's view on the TV screen is not restricted. The included community allows a simple extension and actualisation of existing additional information.

Kapitel 1

Einleitung

Der Wunsch nach Zusatzinformationen zum laufenden Fernsehprogramm steigt, und die technischen Möglichkeiten dafür sind bereits gegeben. Es gibt zahlreiche Quellen für Informationen, wie Wikipedia, Google Maps und IMDb, die in unserem Informationszeitalter einen wichtigen Platz eingenommen haben. Allerdings wird oft viel Zeit in die Suche der gewünschten Zusatzinformationen investiert. Somit stellt sich die Frage, wie dem Zuseher[1] parallel zur laufenden TV-Sendung synchronisierte Zusatzinformationen zur Verfügung gestellt werden können.

Das „Multitasking“, also das gleichzeitige Verrichten von mehreren Tätigkeiten, ist für viele Menschen zum Normalzustand geworden. Das trifft auch beim Fernsehen zu. Auf der Couch zu sitzen und einfach nur fernzusehen ist zu passiv geworden. Daher geht der Trend zum *Second Screen* – dem zweiten Bildschirm. Der Begriff sagt aus, dass neben dem Fernsehen mit einem zweiten Gerät zusätzliche Aktivitäten durchgeführt werden. Es werden Mails abgerufen, Soziale Plattformen besucht und Zusatzinformationen zum aktuellen Programm gesucht.

Der letzte Punkt stellt die Motivation für die vorliegende Arbeit dar. Um das Fernsehen aktiver zu gestalten, soll der Anwender parallel zur Sendung automatisch und zeitgerecht Zusatzinformationen erhalten. Der Zuseher soll sich weder fragen müssen, wie der Schauspieler im Bild heißt, noch wer der Interpret des Soundtracks ist, der im Film zu hören ist. Diese Fragen sollen auf einem externen Gerät sofort beantwortet werden, sobald sie im Film relevant sind.

Das Ziel dieser Arbeit ist es, ein System zu entwickeln, das es einem Anwender ermöglicht, parallel zu einer laufenden Sendung am Smart TV Zusatzinformationen zu empfangen. Diese Informationen sollen zum aktuellen Geschehen passend *Timestamp* basierend erhalten werden. Um auf Einbußen beim Fernsehbild zu verzichten, soll das System die Informationen auf einem

[1] Zugunsten der einfacheren Lesbarkeit wird sowohl für die männliche als auch die weibliche Form, die männliche Form verwendet.

externen Gerät anzeigen. Einen wichtigen Punkt stellt eine Community dar, die die Möglichkeit erhalten soll, die nötigen Metainformationen selbst zu erstellen. Durch eine Kapselung der Daten soll das System leicht erweiterbar sein. Damit es weitläufig einsetzbar ist, muss die Applikation plattformunabhängig gestaltet und auch für Laien zugänglich gemacht werden.

Die Arbeit beginnt nach diesem einleitenden Kapitel mit den technologischen Grundlagen des Systems in Kapitel 2. Neben einer Beschreibung zweier Protokolle, die für den Verbindungsaufbau relevant sind, wird der Aufbau einer Samsung Smart TV Applikation erläutert und die zugehörigen APIs erklärt.

Kapitel 3 handelt von existierenden Systemen aus diesem Bereich. Zuerst werden bestehende *Second Screen* Lösungen vorgestellt, die ebenfalls Metainformationen auf einem externen Gerät anzeigen. Nach weiteren Beispielen zu Metainformations-Systemen werden diese anhand bestimmter Kriterien miteinander verglichen.

Auf diese allgemeinen Ausführungen folgt schließlich der eigene Ansatz zur Entwicklung des Systems. Die nächsten zwei Kapitel stellen den Hauptteil der vorliegenden Arbeit dar. Kapitel 4 widmet sich der theoretischen Herangehensweise zur Umsetzung des Systems. Neben den Systemanforderungen finden sich hier Erklärungen und schematische Darstellungen zur Architektur der technischen Umsetzung.

In Kapitel 5 wird im Anschluss die Implementierung im Detail erläutert. Begonnen wird mit den verwendeten Technologien und der Beschreibung der Schnittstellen zur Samsung Smart TV Applikationsentwicklung. Daraufhin wird der Aufbau des gesamten Systems, unterstützt mit Code-Beispielen und abschließenden Ergebnissen, genau behandelt.

Kapitel 6 beschäftigt sich mit der Evaluierung des entwickelten Ansatzes. Hier wird das Ergebnis den zuvor definierten Anforderungen gegenübergestellt und ein Vergleich mit den in Kapitel 3 erörterten Systemen gezogen. Zusätzlich werden mögliche Erweiterungen und Verbesserungsvorschläge diskutiert.

Ein abschließendes Resümee der gesamten Arbeit erfolgt in Kapitel 7. Außerdem wird ein Ausblick im Bereich von *Second Screen* Anwendungen und deren möglichen zukünftigen Entwicklungen gegeben.

Kapitel 2

Technologische Grundlagen

In diesem Kapitel werden die grundlegenden Techniken zum Entwickeln eines interaktiven Fernsehszenarios erläutert. Zuerst werden die zwei Protokolle UPnP und DLNA, die für einen Verbindungsaufbau zwischen unterschiedlichen Geräten, wie z. B. zwischen Tablet und TV-Gerät, zuständig sind, erklärt. Weiters wird darauf eingegangen, was eine Samsung Smart TV Applikation ausmacht, und wie sie funktioniert. Am Schluss dieses Kapitels werden die wichtigsten APIs – also Applikationsschnittstellen – des Samsung Smart TVs behandelt.

2.1 UPnP – Universal Plug and Play

UPnP steht für *Universal Plug and Play* und wird vom 1999 gegründeten *UPnP Forum*[1] vorangetrieben. Es wurde entwickelt, um eine nahtlose Verbindung zwischen verschiedenen elektronischen Geräten unterschiedlicher Hersteller zu ermöglichen. Dazu zählen unter anderem Router, mobile Geräte, Fernseher, Drucker oder Audio-Geräte, die über ein IP-basiertes Netzwerk angesteuert werden [34]. Die UPnP Architektur bietet allgegenwärtige Peer-to-Peer Netzwerk Verbindungen von PCs, intelligenten Anwendungen und Wireless-Geräten. Als Vorteile der UPnP Technologie werden unter anderem folgende genannt [33]:

- **Medien- und Geräteunabhängigkeit:** Sie läuft auf allen Netzwerktechnologien wie Wi-Fi, Koaxialkabel, Telefonkabel, Stromkabel oder Ethernet.
- **Plattformunabhängigkeit:** Hersteller können verschiedene Betriebssysteme und Programmiersprachen verwenden, um UPnP Produkte herzustellen.
- **Internet basierende Technologien:** UPnP baut z. B. auf IP, TCP, UDP, HTTP und XML auf.

[1] http://www.upnp.org/

Durch die *Device Architecture Specification* wird Steuergeräten und -applikationen ermöglicht, alle UPnP zertifizierten Geräte in einem lokalen Netzwerk zu finden [34].

Beispiele für das Anwendungsgebiet von UPnP gibt es einerseits im Bereich der digitalen Unterhaltung, wie z. B. Fotos von der Kamera über das TV-Gerät anzuzeigen oder Musik vom Handy über die Stereoanlage abzuspielen. Andererseits wird UPnP auch bei diversen *Smart Devices* verwendet, um das Tablet beispielsweise als Fernbedieung zu nutzen [33].

Der Ablauf der UPnP Technologie, wie in [35, S. 3–5] spezifiziert, ist in Abb. 2.1 grafisch dargestellt und kann wie folgt skizziert werden: Die Grundlage für ein UPnP Netzwerk ist die IP-Adressen Vergabe. In einer IPv4 Umgebung muss jedes *Device*[2] und jeder Kontrollpunkt[3] einen DHCP (*Dynamic Host Configuration Protocol*)[4] Client haben und einen DHCP-Server suchen, wenn das Device oder der Kontrollpunkt das erste Mal im Netzwerk aufscheint. Steht ein DHCP-Server zur Verfügung, muss die zugewiesene IP-Adresse verwendet werden. Gibt es keinen verfügbaren DHCP-Server, wird dem Device oder Kontrollpunkt automatisch eine IP-Adresse zugeteilt.

Sobald die IP-Adresse vergeben wurde, ist der erste Schritt in der UPnP Vernetzung die **Lokalisierung**. Wird ein Device dem Netzwerk hinzugefügt, erlaubt das UPnP *discovery protocol* dem Device seine Dienste den verschiedenen Kontrollpunkten im Netzwerk zu melden. In gleicher Weise wird einem hinzugefügten Kontrollpunkt erlaubt, nach passenden UPnP-Devices zu suchen. Der grundlegende Austausch in beiden Fällen ist eine *discovery message*, also eine Nachricht, die die wichtigsten Angaben der jeweiligen Devices und ihre Dienste beinhaltet, wie z. B. der Gerätename, Gerätetyp und ein Link zu einer genaueren Beschreibung des Devices.

Im nächsten Schritt geht es um die **Beschreibung**. Nachdem ein Kontrollpunkt ein Device gefunden hat, verfügt er noch nicht über genug Informationen über dieses. Daher wird eine genauere Beschreibung der durch die *discovery message* zu Verfügung gestellten URL entnommen. Die im XML-Format vorliegende Beschreibung umfasst neben herstellerspezifischen Informationen auch eine Liste aller eingebetteten Devices oder Dienste, genauso wie verschiedene URLs für Steuerung, Ereignisverwaltung und die Präsentation. Für jeden Dienst wird auch eine Liste aller Kommandos oder Aktionen, auf die der Dienst antwortet, bereitgestellt. Genauso wie Parameter oder Argumente, die von den einzelnen Aktionen erwartet werden.

Der dritte Schritt des UPnP Netzwerks umfasst die **Steuerung**. Hat ein Kontrollpunkt die detaillierte Beschreibung eines Devices erhalten, kann er

[2] Der Begriff „Device“ bedeutet „Gerät“ und wird in weiterer Folge synonym verwendet.

[3] Ein Kontrollpunkt kann z. B. ein Computer sein und muss sich im selben Netzwerk wie die zu steuernden Geräte befinden. Er erhält Geräte- und Service-Beschreibungen, sendet Anweisungen an die Services und bekommt Events von den Services.

[4] Das *Dynamic Host Configuration Protocol* ist für die Zuweisung der Netzwerkkonfiguration durch einen Server an Clients verantwortlich.

den Diensten des Devices Aktionen übermitteln. Um dies zu bewerkstelligen, sendet der Kontrollpunkt eine passende *control message*, eine Steuerungsnachricht, an die dafür vorgesehene Steuerungs-URL. Steuerungsnachrichten sind ebenfalls im XML-Format geschrieben und verwenden dabei das *Simple Object Access Protocol* (SOAP)[5].

Der folgende Schritt behandelt die **Ereignisverwaltung**. Um zu verhindern, dass ein Kontrollpunkt ständig den Status eines Dienstes abfragen muss, werden bei jeder Veränderung einer Statusvariable oder eines Zustands Ereignismeldungen mit der jeweiligen Aktualisierung versendet. Ein Kontrollpunkt kann diesen Dienst, der die Aktualisierungen versendet, abbonieren, um die sogenannten *event messages* zu erhalten und automatisch über den Gerätestatus informiert zu werden.

Die **Präsentation** ist der letzte Schritt des UPnP Netzwerks. Besitzt ein Device eine Präsentations-URL, kann mit einem Webbrowser auf das Device zugegriffen bzw. dieses gesteuert werden. Weiters wird das Einsehen des Status ermöglicht. Diese Präsentationsseite kann je nach Spezifikation und Hersteller andere Funktionen übernehmen.

2.2 DLNA – Digital Living Network Alliance

DLNA steht für *Digital Living Network Alliance* und wurde 2003 gegründet. Es handelt sich dabei um eine internationale Vereinigung von Herstellern, die Computer, Unterhaltungselektronik und Mobiltelefone produzieren. Diese Vereinigung hat als Ziel, dass informationstechnische Geräte von unterschiedlichen Herstellern aus dem Bereich Heim- und Eigengebrauch untereinander kompatibel sind und die Fähigkeit zur Zusammenarbeit aufweisen [38]. Das bedeutet, dass verschiedene Geräte die Möglichkeit besitzen sollen, innerhalb eines Netzwerkes miteinander zu kommunizieren. So können digitale Fotos, Musik und Videos zwischen verschiedenen Geräten wie Fernseher, Computer, Tablets und Smartphones über Räume hinweg untereinander ausgetauscht werden.

Als Beispiel wird in einem *Whitepaper* der DLNA folgendes genannt [38, S. 2–3]. Es soll ein Video gezeigt werden, das die Tochter mit ihrem Smartphone aufgenommen hat. Der herkömmliche Weg wird als umständlich beschrieben, indem das Video vom Smartphone zuerst per Mail an den Computer gesendet wird. Danach wird die Datei auf einen USB-Stick gespielt, um dann den Stick am großen Flat-Screen-Fernseher anzuschließen. Dann erst wird das Video am TV angezeigt (siehe Abb. 2.2). Im Gegensatz dazu wird die Einfachheit von DLNA erläutert: die direkte Verbindung zwischen Smartphone und Fernseher. Sind beide Geräte DLNA zertifiziert, funktioniert die Übertragung des Videos vom Smartphone an das TV-Gerät direkt

[5] Das *Simple Object Access Protocol* ist ein Netzwerkprotokoll, mit dem Daten zwischen Systemen ausgetauscht werden können.

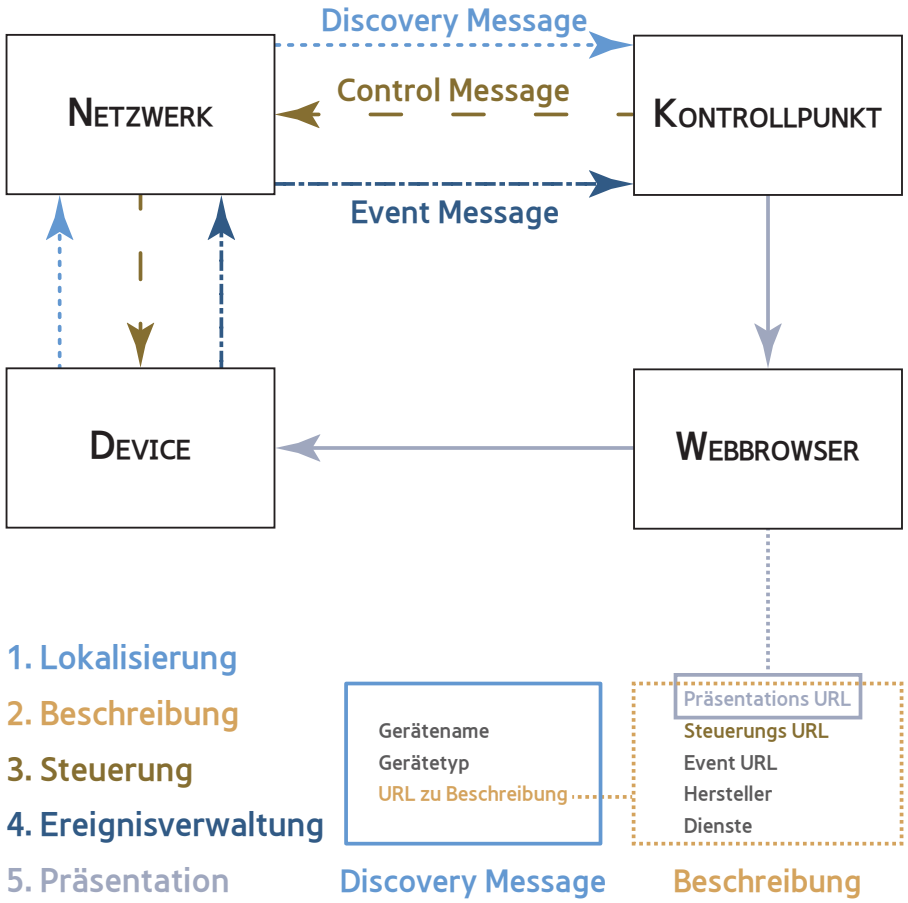

Abbildung 2.1: Der Ablauf der UPnP Technologie schematisch dargestellt.

über das gemeinsame W-Lan (siehe Abb. 2.3).

Um diese Interoperabilität zu gewährleisten, werden von der DLNA genaue Richtlinien definiert, die auf öffentlichen Standards basieren [38, S. 3–4]. Diese Richtlinien müssen von den verschiedenen Herstellern genau eingehalten werden, damit das jeweilige Produkt von der DLNA zertifiziert wird und mit dem *DLNA-Logo* versehen werden darf (siehe Abb. 2.4).

2.2.1 Unterschied zwischen DLNA und UPnP

Laut [20] besteht der größte Unterschied zwischen UPnP und DLNA im Anwendungsbereich, den sie abdecken. UPnP besteht aus Protokollen, die verschiedenen Devices erlauben sich zu finden und die eigenen Services zu benutzen. DLNA im Gegensatz ist ein *Device Standard*, der einen größeren Anwendungsbereich abdeckt, und Vernetzung, Formate, Systemverwaltungen und Inhaltsicherungssysteme beinhaltet. Allerdings verwendet DLNA

Abbildung 2.2: Der traditionelle Weg ein Video von einem Smartphone auf einem Fernseher darzustellen, erfordert einige Zwischenschritte und einiges an Zeit [38].

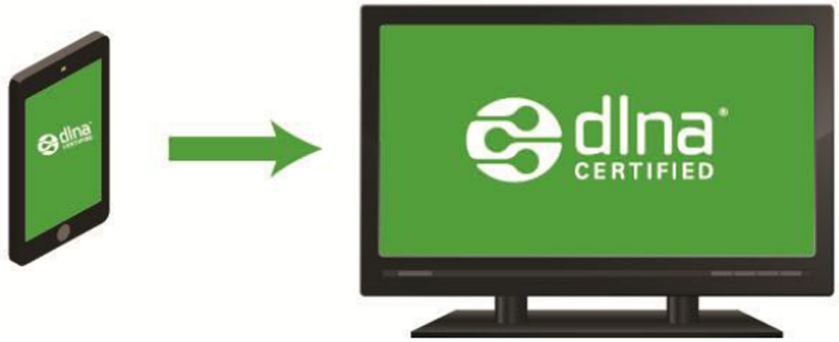

Abbildung 2.3: Mit Hilfe von DLNA soll es ganz einfach und nahtlos funktionieren, verschiedene Inhalte direkt von einem Smartphone an einen Fernseher zu senden und zu teilen [38].

Abbildung 2.4: Produkte, die mit diesem Logo ausgezeichnet sind, wurden von DLNA zertifiziert.

UPnP für die Medienverwaltung, für das Finden von anderen Devices und deren Steuerung [38, S. 5]. DLNA ist also eine Obermenge von UPnP, und lässt darauf schließen, dass alle Devices die DLNA unterstützen, genauso UPnP unterstützen [20].

2.3 Samsung Smart TV Applikationen

Im offiziellen *Guide* von Samsung [27] werden die grundlegenden Elemente rund um Samsung Smart TV und Applikationen beschrieben. Applikationen sind kleine, webbasierte Anwendungen, die auf digitalen, mit dem Internet verbundenen Fernsehgeräten laufen.

Mit dem von Samsung angebotenen Service ist es möglich, durch ver-

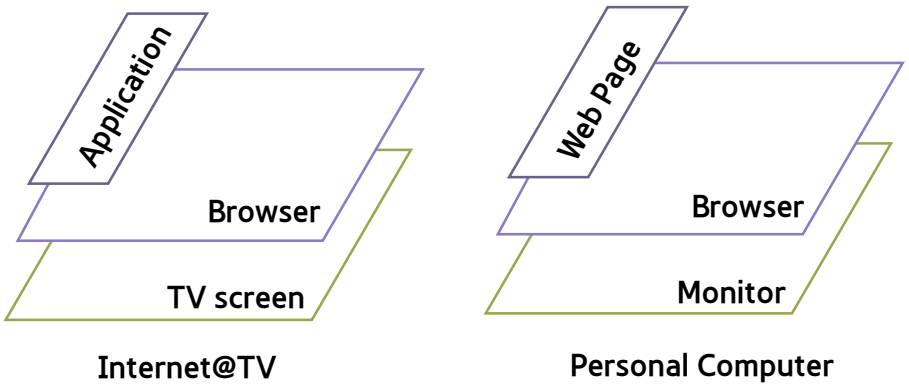

Abbildung 2.5: Der Vergleich einer Applikation am Fernseher mit einer Webseite am PC (nach einer Grafik aus [27, S. 9]).

schiedene Webfunktionen die vorhandenen Möglichkeiten des Fernsehers zu erweitern. Der Benutzer des TVs kann durch Verwendung von Internet Diensten, wie Nachrichten, Wettervorhersagen oder Aktien, nützliche Informationen und Inhalte empfangen. Neben den angebotenen Applikationen, die beispielsweise von der Samsung Seite[6] bezogen werden können, besteht die Möglichkeit, Applikationen selbst zu entwickeln.

Eine Applikation ist eine Webseite, bestehend aus HTML, CSS und JavaScript, die in einen Webbrowser – auf Samsung Smart TVs ist das *Maple*[7] – implementiert ist und am Fernsehbildschirm angezeigt wird. Unterschiede zwischen einer Applikation und einer Webseite, die am Computer betrachtet wird, sind nur durch Bildschirmauflösung, Hardware Spezifikationen und die verschiedenen *Interfaces* gegeben. Statt Maus und Tastatur am PC, wird die Fernbedienung des Fernsehers zur Steuerung der Applikation verwendet (vgl. Abb. 2.5).

Neben dem HTML Dokument für den Aufbau der Seite, der CSS-Datei für das Aussehen und das JavaScript Dokument für die Funktion der Applikation, muss eine Konfigurationsdatei angelegt werden. Diese ist notwendig, um Informationen wie Betriebsumgebung und Versionsnummer der Applikation anzugeben.

2.4 Samsung Smart TV APIs

Zum Entwickeln von Samsung Smart TV Applikationen werden einige Applikations-Schnittstellen zur Verfügung gestellt. Im Folgenden werden die wichtigsten ausgewählt und beschrieben. Dazu gehören die *Device API*, die *AllS-*

[6] http://de.samsung.com/de/microsites/smarttvapps/

[7] Maple steht für ***Ma****rkup Engine* ***Pl****atform for* ***E****mbedded Systems* und ist der Webbrowser von Samsung Smart TVs.

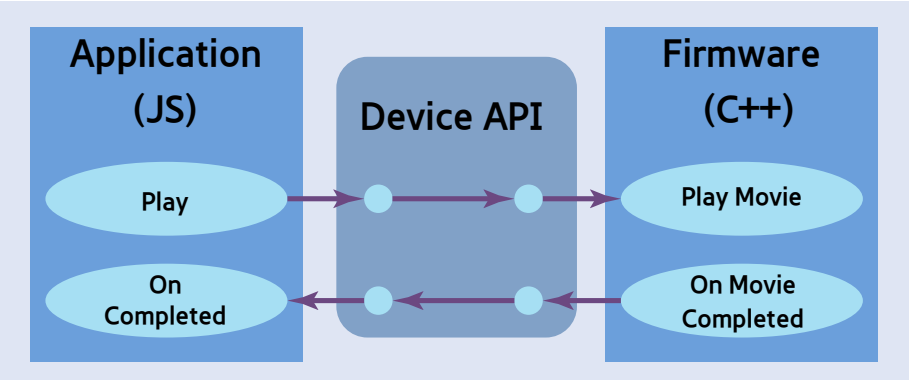

Abbildung 2.6: Darstellung der Funktionsweise der Device API (nach einer Grafik aus [19]).

hare API und das *Messaging System*.

2.4.1 Device API

Die Device API[8] kommt zum Einsatz, wenn fernsehspezifische Funktionen, wie z. B. das Senderwechseln, durch gewöhnliche JavaScript Funktionen nicht unterstützt werden [19]. Um sie zu verwenden, kann die API als JavaScript in Form von Methoden oder Callback-Funktionen aufgerufen werden. Das bedeutet, dass sie die Charakteristik einer bidirektionalen Kommunikation aufweist. Die Device API ist immer an einen Browser gekoppelt. Die Architektur der Device API wird nachfolgend erklärt [19]. Grundsätzlich übernimmt sie die Kommunikation zwischen der *Firmware*[9] des Geräts und JavaScript (vgl. „*language binding*“). Dazu siehe auch Abb. 2.6. Werden die Befehle von JavaScript nicht direkt unterstützt, übersetzt sie die API und leitet sie an die Firmware weiter. Die Antwort wird dann erneut umgewandelt und an die JavaScript Applikation zurückgesendet.

2.4.2 AllShare API

Diese API definiert ein Applikationsinterface, um geräteübergreifende Services zu ermöglichen. So sollen Funktionen wie Inhalte zwischen mehreren Geräten austauschen, verschiedene Geräte im selben Netzwerk finden und der Zugriff auf andere Geräte, gestattet werden. Die AllShare API besteht aus sechs Modulen [14]:

[8] API steht für ***A**pplication **P**rogramming **I**nterface* (zu deutsch *Schnittstelle zur Anwendungsprogrammierung*), und ist ein Programmteil, der als Anbindung zu einem System für andere Programme zur Verfügung gestellt wird.

[9] Eine Firmware ist eine Softwawre, die in elektronische Geräte eingebettet ist.

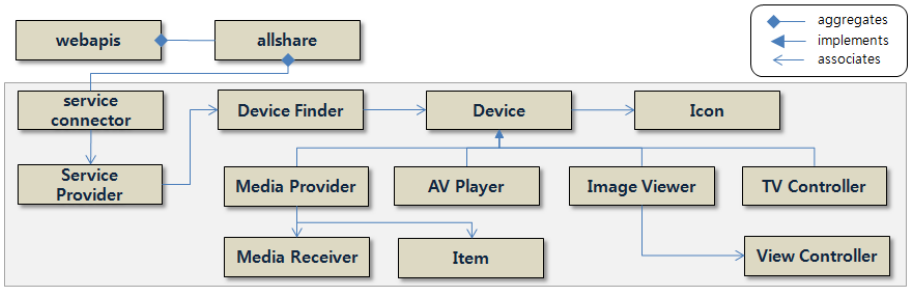

Abbildung 2.7: Zusammenspiel der APIs aus [26].

`device`: Repräsentiert jedes DLNA Device im lokalen oder entfernten Netzwerk.

`devicefinder`: Stellt Funktionen zur Verfügung, um Devices, die verschiedene Services wie Medienwiedergabe oder eine Gerätesteuerung bieten, zu finden.

`provider`: Ist ein spezifischer Gerätetyp, der seine Medieninhalte mit anderen Devices teilt.

`item`: Repräsentiert jeden Medientyp, der von einem *Provider* zur Verfügung gestellt wird. Das kann z. B. ein Dateiordner, ein Bild, Musik oder ein Video sein.

`mediasharing`: Wird verwendet, um die *devicefinder* Schnittstellen mit *serviceapis* Modulen zu verbinden.

`serviceapis`: Bietet generelle Funktionen wie Fehlerbehandlung speziell für AllShare APIs.

Soll ein Video von einem Smartphone auf einem Smart TV wiedergegeben werden, müssen folgende Schritte durchlaufen werden: Zuerst werden alle DLNA Devices in einem Netzwerk gesucht. Das Video, das vom Smartphone auf dem Fernseher wiedergegeben werden soll, muss zuerst freigegeben werden, um es zu teilen. Danach kann das Smart TV Gerät darauf zugreifen und es wiedergeben, vorausgesetzt er ist ein DLNA Device und befindet sich auch im selben Netzwerk.

Wie die verschiedenen APIs zusammenspielen, wird in Abb. 2.7 gezeigt.

2.4.3 Messaging System

Es ist möglich, zwischen einem Samsung Smart TV und einem verbundenen Device wie ein Tablet, Smartphone oder Laptop bidirektional Nachrichten zu verschicken. Bidirektional heißt, dass sowohl die einzelnen Devices an den Fernseher, wie auch der Fernseher an die einzelnen Devices Nachrichten senden kann. Überdies können auch Nachrichten an eine Gruppe von Devices

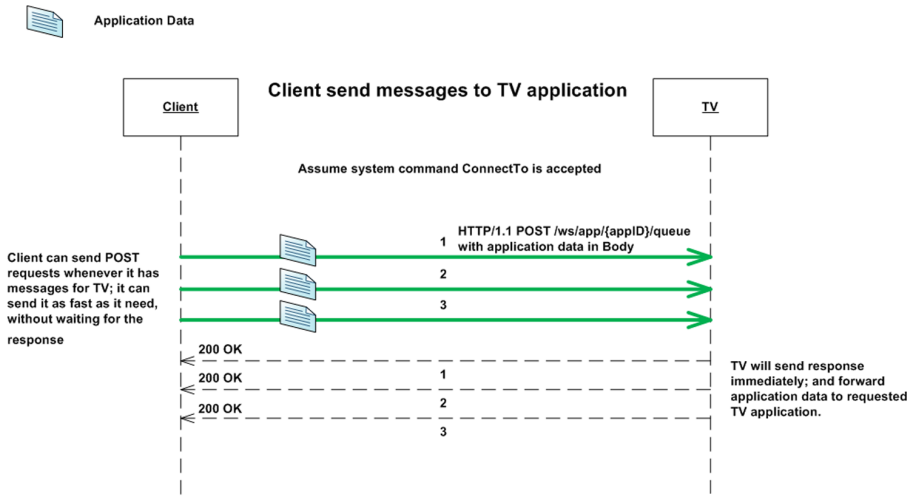

Abbildung 2.8: Schematische Darstellung der Nachrichtenübertragen von einem Device an ein Samsung Smart TV Gerät mittels einem *POST Request* (Grafik aus [28]).

geschickt werden. Eine Nachricht kann im XML oder JSON Format vorliegen [28].

Um die Nachricht von einem Device an das Smart TV Gerät zu übermitteln, wird ein *POST Request* verwendet (siehe Abb. 2.8). Wenn das TV-Gerät eine Nachricht an ein Device schickt, wird allerdings ein *GET Request* verwendet (siehe Abb. 2.9).

Da das Samsung Smart TV *Framework* ein *REST*[10] basierendes Interface zur Verfügung stellt, ist es für die Client-Applikation möglich, sich zu einer TV-Applikation zu verbinden, Nachrichten zu schicken und zu empfangen, einer Gruppe beizutreten und sie zu verlassen beziehungsweise Applikationsinformationen zu erhalten.

[10]REST steht für *Representational State Transfer* und bezeichnet einen Software Architekturstil, wo die Idee verfolgt wird, dass eine URL genau einem Seiteninhalt als Ergebnis entspricht.

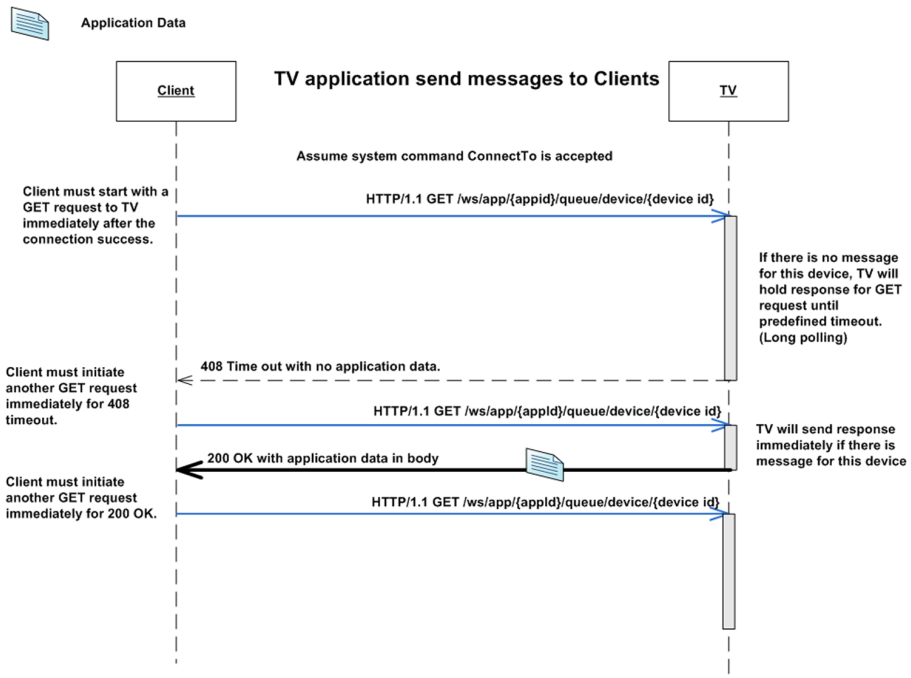

Abbildung 2.9: Schematische Darstellung der Nachrichtenübertragen von einem Samsung Smart TV an ein verbundenes Device mittels eines *GET Requests* (Grafik aus [28]).

Kapitel 3

State of the Art

Dieses Kapitel handelt von verschiedenen Systemen, die es erlauben, Zusatzinformationen zu dem im Fernsehen Gesehenen zu erhalten. Begonnen wird mit *Second Screen* Lösungen, die Metainformationen auf einem externen Gerät anzeigen können. Danach werden Applikationen vorgestellt, die die Metainformationen direkt auf dem Fernsehgerät darstellen. Weiters werden Standards zum Speichern von Metadaten erläutert. Bevor diese Systeme abschließend anhand bestimmter Kriterien verglichen werden, handelt der vorherige Abschnitt von interaktivem Fernsehen und interaktiven Projekten.

3.1 TV Second Screen

Ein *Second Screen*, ein zweiter Bildschirm, meint einen zusätzlichen Bildschirm, der parallel zu einer laufenden Fernsehsendung verwendet wird. Häufig stellt dieser Bildschirm ein Gerät mit Internetverbindung dar, wie etwa Smartphones oder Tablets. Der *Second Screen* wird oft genutzt, um zusätzliche Informationen zur aktuellen Sendung abzurufen oder gerade Gesehenes auf Sozialen Plattformen zu kommentieren.

3.1.1 Nutzung von Second Screens

Laut einer Studie von *Nielsen*[1], veröffentlicht im zweiten Quartal 2012, verwenden in Amerika 39 Prozent der Bürger mindestens einmal am Tag ihr Smartphone während sie fernsehen. 62 Prozent sagen, sie nutzen den *Second Screen* mehrmals die Woche und 84 Prozent mindestens einmal im Monat [42].

In Deutschland hat die Unternehmensberatung *Anywab*[2] aus Darmstadt im Mai und Juni 2012 sowie auch im Februar 2013 ebenfalls Studien zu diesem Thema durchgeführt. 2012 haben bereits 49 Prozent der Internetnut-

[1] http://www.nielsen.com/us/en.html
[2] http://anywab.com/

zer in Deutschland zwischen 14 und 49 Jahren zumindest gelegentlich den *Second Screen* verwendet. 66 Prozent davon haben den *Second Screen* zur Suche von Zusatzinformationen zur Sendung verwendet, und 57 Prozent für Soziale Netzwerke [16]. Anfang 2013 waren es 62 Prozent der Tablet PC Nutzer, 54 Prozent Smartphone Benutzer und 45 Prozent aller Internetnutzer die den *Second Screen* zumindest gelegentlich verwendeten [15].

Im Gegensatz zu diesen Nutzerbefragungen gibt es einen Versuch, der auf Grund von Seitenaufrufen auf Wikipedia zu Stande kam. Ziel dieses Versuchs war es zu erkennen, ob es einen Zusammenhang zwischen der laufenden Fernsehsendung und *Second Screen* Aktivitäten gibt, oder ob z. B. nur E-Mails abgerufen werden. Als Ergebnis ließ sich ein direkter Programmbezug bei den meisten Top-50-Artikeln herstellen. Die Besucher der Wikipedia suchten Informationen zu den Gästen der laufenden Sendung „Wetten, dass. . . ?“, zu laufenden Serien, Spielfilmen und deren Schauspielern [25].

3.1.2 Spezielle Angebote

Im Fachmagazin *c't*[3] wurden 2012 einige Applikationen vorgestellt, die das Fernsehen mit Tablets und Smartphones verknüpfen.

Als Beispiel wird eine Web-App vom Fernsehsender ZDF beschrieben, die am 6. Oktober 2012 parallel zu „Wetten, dass. . . ?“ angeboten wurde, um über Erfolgsaussichten der Kandidaten zu spekulieren oder Making-of-Szenen zu bestaunen [9] (siehe Abb. 3.1).

Auch der andere öffentlich-rechtliche TV-Sender Deutschlands, die ARD, entwickelte eine *Second Screen* Web-Applikation zur Krimisendung „Tatort“, die es Zuschauern ermöglichte, nach der für sich abgeschlossenen Sendung, online selbst den Täter zu ermitteln [18].

Vom TV-Sender RTL gibt es eine eigenen App namens „RTL INSIDE“, die parallel zu den laufenden RTL-Sendungen Informationen liefert. Gleichzeitig kann sich der Benutzer auch in den Sozialen Netzwerken Twitter und Facebook einloggen, und relevante Kommentare seiner Freunde lesen bzw. selbst verfassen [43].

Auch der österreichische öffentlich-rechtliche Fernsehsender ORF bot zur Ski-WM 2013 in Schladming eine *Second Screen* App mit Zusatzinformationen an (siehe Abb. 3.2). Als Zusatzinformationen am Tablet standen unter anderem andere Kameraeinstellungen, Streckenprofile oder Informationen zum Skifahrer zum Angebot [40]. Die Daten dafür kamen aus der Datenbank, aus der auch die (Co-)Moderatoren die Informationen beziehen [39].

[3] http://www.heise.de/ct/

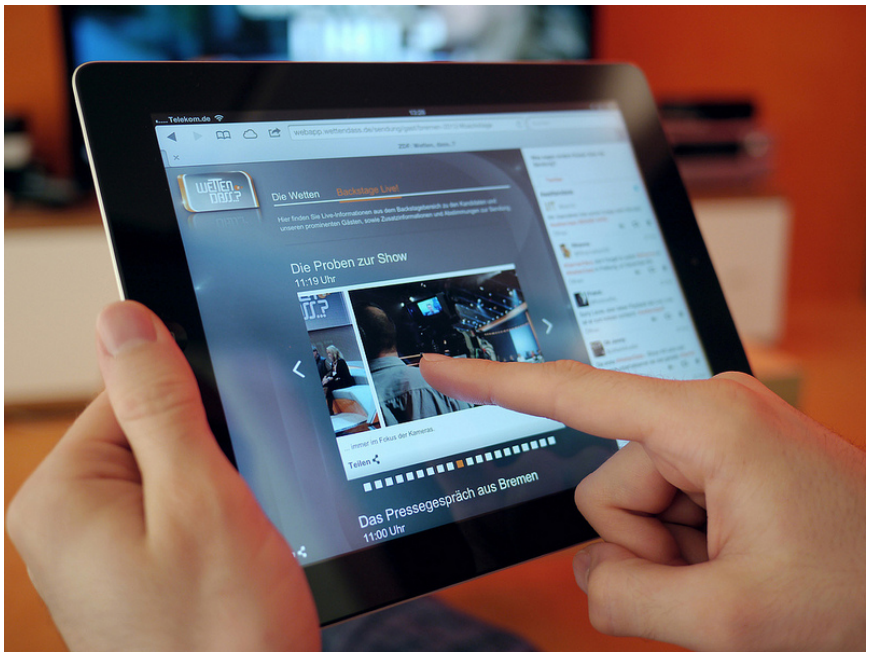

Abbildung 3.1: Die *Second Screen* Applikation des ZDF zur Sendung „Wetten, dass...?“ mit beispielsweise zusätzlichen Probe-Szenen zur Show (Bild aus [10]).

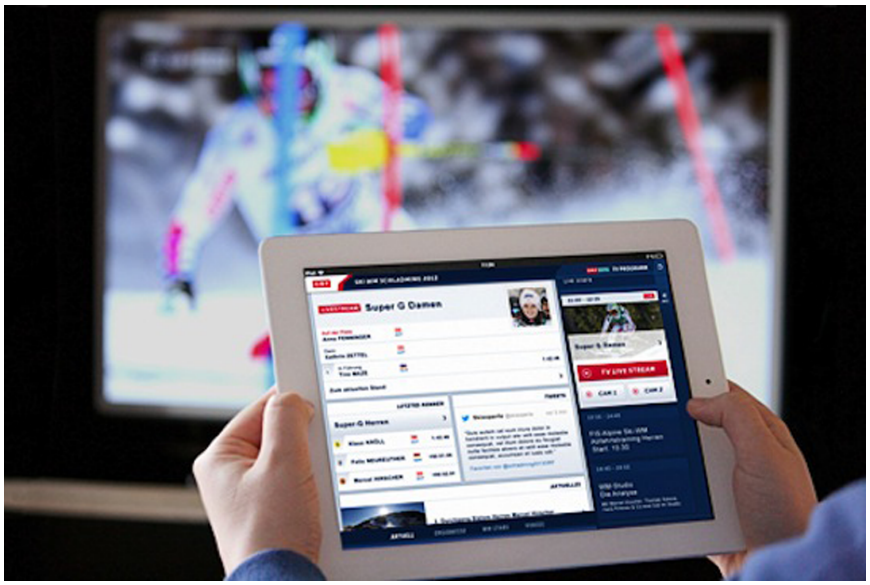

Abbildung 3.2: Die *Second Screen* Applikation des ORF zur Ski-WM 2013 in Schladming, mit Zusatzinformationen zu beispielsweise den Skiläufern (Bild aus [11]).

3.2 Blu-Ray Second Screen Applikationen

Zu einigen Blu-Ray Filmen gibt es bereits begleitende Applikationen zum Download, mit denen zustäzliche Informationen parallel zum Film empfangen werden können.

3.2.1 Disney Second Screen

Bei *Disney Second Screen*[4] handelt es sich um eine Applikation von Disney für iPad oder Computer, die es dem Anwender erlaubt, mit einem dazu passenden Blu-Ray Film zu interagieren. Nachdem der Film über den Blu-ray-Player gestartet wurde, kann die *Second Screen* App synchronisiert werden. Dies funktioniert entweder über den *Audio-Sync*, oder über einen *Manual Sync*. Sobald das iPad mit dem Blu-Ray Film verbunden ist, kann der Benutzer interaktive Galerien entdecken, Spiele spielen, oder Hintergrundinformationen zu den aktuell gezeigten Szenen erfahren. Dabei kann er sich frei bewegen, und immer wieder zu der synchronisierten Zeitleiste zurückgehen. Blu-Ray Filme von Disney, zu denen es bereits *Second Screen* Applikationen gibt, sind:

- TRON: Legacy,
- Pirates of the Caribbean: On Stranger Tides,
- The Lion King,
- Lady and the Tramp,
- John Carter und
- Cinderella.

Der Download der Applikationen ist kostenlos, allerdings können sie nur verwendet werden, wenn der passende Code, der sich in der Blu-Ray Verpackung befindet, eingegeben wurde [41].

3.2.2 Weitere Blu-ray Second Screen Apps

Neben den *Disney Second Screens* gibt es auch *Second Screen* Applikationen zu anderen Filmen. So werden Apps zu *Spider-Man*, *The Avengers*, *The Dark Knight Rises* oder zu *The Hunger Games* angeboten [24]. Bei diesen Applikationen spielen oft Szenen hinter den Kulissen, Making-ofs, gelöschte Szenen und Storyboards eine wesentliche Rolle.

3.3 Google Info Cards

Seit März 2013 gibt es von Google für US-Bürger eine Erweiterung zur Applikation *Google Play Movies*[5]. Durch die hinzugefügten *Info Cards* kann

[4] http://disneysecondscreen.go.com
[5] https://play.google.com/store/apps/details?id=com.google.android.videos

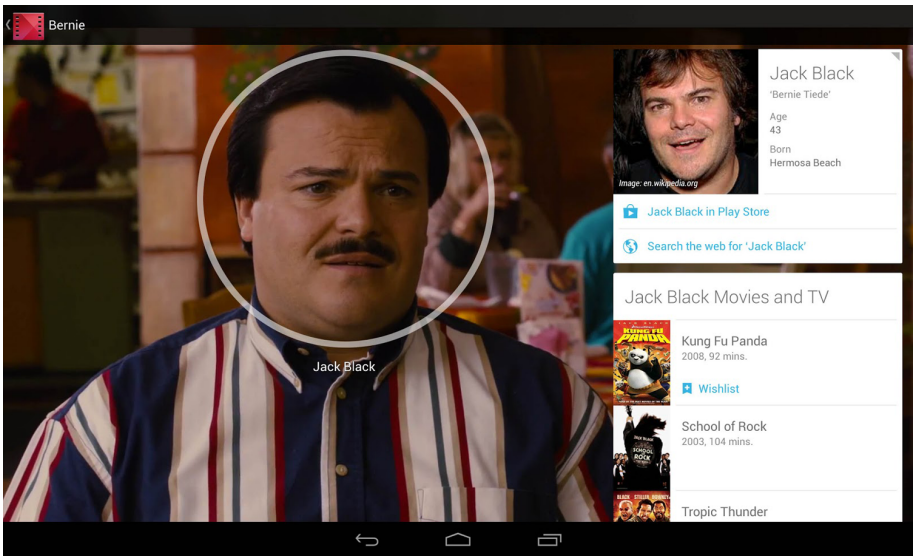

Abbildung 3.3: Wenn der Film, der über *Google Play Movies* auf einem Android Tablet ab Version 4.0 angesehen wird, pausiert wird, erscheinen *Info Cards* mit näheren Informationen zu dem Schauspieler (Bild aus [12]).

der Anwender den Namen des laufenden Musiktitels erfahren oder Informationen über die Schauspieler und deren Filmen bekommen. Dazu wird ein Android Tablet ab der Version 4.0 benötigt, auf dem ein Film über *Google Play Movies* angesehen wird. Wenn der Film pausiert wird, erscheinen die *Info Cards* am Bildschirm, mit Informationen zu dem Schauspieler, der gerade im Film zu sehen ist (siehe Abb. 3.3). Der Benutzer kann mehr über den Schauspieler selbst, seinen Charakter im Film, oder Informationen zum Film und dessen Soundtrack nachlesen. Wird der Film fortgesetzt, werden die *Info Cards* wieder ausgeblendet [32].

3.4 XBMC Media Center

XBMC Media Center (ursprünglich *XBox Media Center*) wurde 2006 gegründet. Es handelt sich dabei um einen Open-Source-Software Mediaplayer und ein Unterhaltungszentrum für digitale Medien. Der Player unterstützt das Abspielen von Videos, Bildern und Musik von Festplatten, DVDs und Servern genauso wie die Wiedergabe von Audio- und Videodaten aus dem Internet [36]. Darüber hinaus kann das *XBMC Media Center* zusätzliche Inhalte aus dem Internet laden, um den Benutzer mit weiteren Informationen zu versorgen. So werden unter anderem Poster, Listen von Darstellern, verschiedene Trailer oder extra Untertitel abgerufen, die von unterschiedlichen

Quellen, wie z. B. *IMDb*[6] bezogen werden.

3.5 MHP – Multimedia Home Platform

Multimedia Home Platform ist ein auf Java basierender, offener Standard aus dem *DVB*[7] *Projekt* und spezifiziert die Darstellung und Übertragung von interaktiven Inhalten im Digitalen Fernsehen. Es sollen neben komplexeren Programmübersichten und Spielen auch interaktive Dienste wie Abstimmungen und Homeshopping-Angebote möglich sein. MHP wurde entwickelt, um als nicht-proprietäres System für alle zugänglich zu sein und die Entwicklung von interaktivem Fernsehen für den horizontalen Markt zu ermöglichen. Durch die verwendete Java-Technologie kann überdies plattformübergreifend entwickelt werden. MHP kann vereinfacht dargestellt werden, als eine Reihe von Anweisungen, die dem Betriebssystem eines digitalen Fernseh-Empfänger mitteilen, wie er mit einer empfangenen interaktiven TV-Applikation umgehen soll. Außerdem wird definiert, in welcher Form die Applikationen geliefert werden [17].

2008 wurde darüber berichtet, dass MHP in Deutschland als sehr zukunftsträchtig angesehen wurde. Durch das offene System und die interaktiven Zusatzanwendungen für einen offenen Markt, „setzte fast die gesamte Fernsehbranche auf den Standard MHP“ [21]. Pro Sieben und Sat 1 wie auch die öffentlich-rechtlichen Kanäle ARD und ZDF boten zusätzlich MHP-Dienste in ihren Bouquets an, wodurch auch mehrere MHP-fähige DVB-T Boxen zum Verkauf standen. Aufgrund zu geringer Nachfrage an MHP-Empfängern konnte sich MHP allerdings nicht durchsetzen, und ist 2008 bereits fast vollständig vom Markt verschwunden. Da keine MHP-fähigen Geräte mehr verkauft wurden, stellten ARD, ZDF und Co ihre Zusatzdienste wieder ein.

2007 startete auch Österreich mit diversen MHP-Diensten (vgl. *ORF OK MultiText* und *ATV OK Multitext*). Allerdings wurde der MultiText von ATV mangels Zuschauerakzeptanz bereits im Jahr 2008 eingestellt, und der ORF Multitext im Juni 2011. Im Gegensatz zu Deutschland und Österreich wird MHP in Italien besser angenommen [37].

3.6 MPEG-7

MPEG-7 ist ein ISO Standard (ISO/IEC 15938) der 2001 von der Moving Picture Experts Group, kurz MPEG[8], entwickelt wurde. In [6, S. 49–50] wird

[6] *Internet Movie Database*, ist eine Datenbank mit Informationen zu Filmen, Serien, und Personen die daran mitgewirkt haben. http://www.imdb.com

[7] DVB steht für *Digital Video Broadcasting* (zu deutsch *Digitaler Videorundfunk*) und bezeichnet den offenen, technischen Standard zur globalen Übertragung von digitalem Fernsehen und von interaktiven Diensten.

[8] http://mpeg.chiariglione.org/

er folgendermaßen erklärt:

> „Der im Jahre 2001 verabschiedete Standard MPEG-7 (ISO/IEC 15938: Multimedia Content Description Interface) ermöglicht eine standardisierte Beschreibung von Metadaten für multimediale Inhalte. Damit wird die Indexierung, Kategorisierung, Verwaltung und erweiterte Suche audiovisueller Inhalte möglich. MPEG-7 bietet ein umfangreiches Toolset für die Beschreibung von Metadaten unterschiedlicher Art mit Hilfe so genannter Description Schemas an. MPEG-7 nutzt hierfür XML-Schema und ist die Grundlage für viele neuartige Merkmale und Dienste in multimedialen Anwendungen."

Ein Beispiel für die Verwendung von MPEG-7 ist das *VideoAnnEx Annotation Tool* von *IBM*[9] [23]. Das *VideoAnnEx Annotation Tool* dient dazu, Videos mit Schlagwörtern zu versehen. Jede Video-Sequenz kann neben Schlüsselwörtern mit *Static Scene*, *Key Object* und *Event* Deskriptoren beschrieben werden. Die verwendeten Beschreibungen werden mit jedem Videoteil assoziiert und automatisch als MPEG-7 Darstellung in einem XML Dokument gespeichert. Jedes Video muss in kleinere Segmente, genannt *Video shots*, zerteilt werden, um die Metainformationen für jede Szene bestimmen zu können. Dafür werden z. B. Schnitte oder Übergänge verwendet. Dies kann entweder automatisch durch das Tool passieren, oder es wird das erforderliche *Shot-File* angegeben. Die grafische Oberfläche des *VideoAnnEx Annotation Tools* ist in vier Bereiche aufgeteilt – *Video Playback*, *Shot Annotation*, *Views Panel* und *Region Annotation* (siehe Abb. 3.4). Interessant ist der *Shot Annotation* Bereich dieses Tools. Hier besteht die Möglichkeit, mit Hilfe von verschiedenen Checkboxen die jeweilige Szene zu beschreiben. Es gibt drei verschiedenen Typen der Bestimmung. Unter *Events* kann die generelle Aktion, die in dieser Szene stattfindet, wie ein Personengespräch oder eine Explosion, gewählt werden. Als nächstes wird die *Static Scene*, also der statische Hintergrund näher definiert (vgl. Wolke, Berg oder Straße). Zum Schluss werden die *Key Objects*, die signifikanten Objekte im Vordergrund der Szene, bestimmt. Das können verschiedene Tiere, Menschen oder bestimmte Bauten sein. Der vergrößerte Ausschnitt der *Shot Annotation* aus Abb. 3.4 befindet sich in Abb. 3.5 [22].

3.7 XRM – eXtensible Rich Media

Im Zuge der Masterarbeit „Kombination von Metainformationen und Videodaten" [5] von Anna Gruber wurde ein eigener Standard namens *eXtensible*

[9] IBM steht für ***I**nternational **B**usiness **M**achines Corporation* und ist ein US-amerikanisches IT- und Beratungsunternehmen http://www.ibm.com.

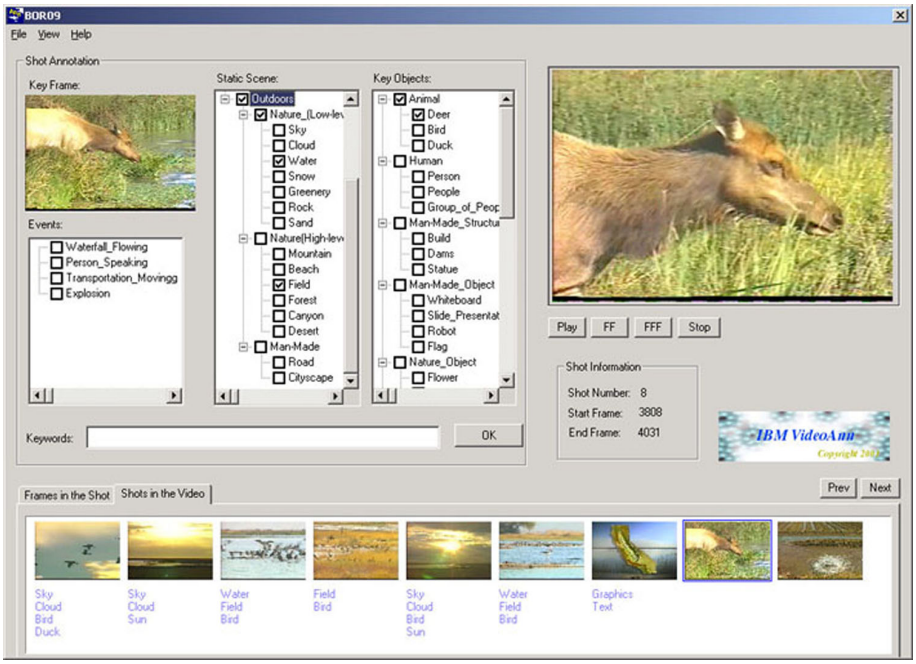

Abbildung 3.4: Das *VideoAnnEx Annotation Tool* ist in vier Bereiche unterteilt – links oben befindet sich die *Shot Annotation*, rechts oben das *Video Playback*, unten sieht man das *Views Panel* und der vierte Bereich *Region Annotation* ist in dieser Grafik nicht sichtbar [23].

Rich Media entwickelt. Es handelt sich dabei um einen benutzerorientierten und plattformunabhängigen Standard zur Speicherung von Metadaten zu Videodaten. Die Autorin beschreibt XRM in [5, S. 41] wie folgt:

> „XRM ist ein Endbenutzer Standard zur Speicherung von Metainformationen zu Filmen/Videos. Genauer ist XRM ein Schema zur Speicherung, Verwaltung und Strukturierung von beschreibenden Metainformationen zu Videodaten. Es wird seit 2008 entwickelt, basiert auf XML und ist zum gegebenen Zeitpunkt in der Version 0.14.2 verfügbar. XRM ist jedoch so aufgebaut, dass es in späterer Folge auch dazu verwendet werden kann, Metainformationen zu Bild- und Tonmedien zu speichern."

Es ist dabei wichtig, dass zusätzlich zu den Metainformationen auch *Timestamps* angegeben werden können, wann diese Metainformationen von Bedeutung sind. Außerdem soll sichergestellt werden, dass auch Informationen unterschiedlichster Art gespeichert werden können. Als Beispiele werden Werbung, eine alternative Szene oder eine genaue Beschreibung einer im Film auftauchenden Blume genannt. Überdies ist es möglich, den „Informationen Quellennachweise, weiterführende Links und Urheberrechte zuzuweisen. In-

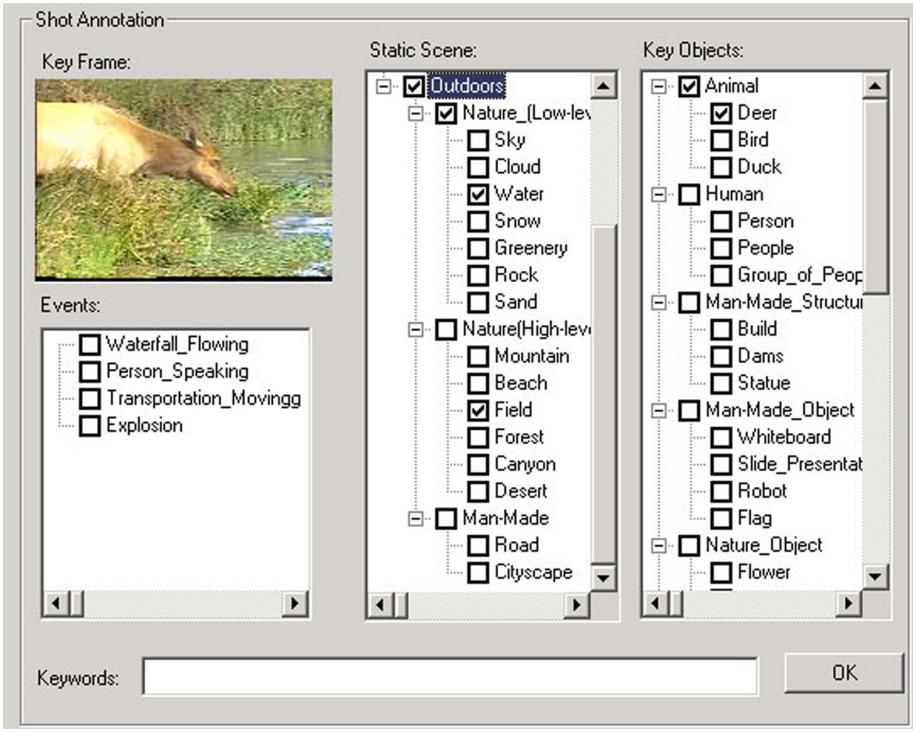

Abbildung 3.5: Der *Shot Annotation* Bereich, zum näheren Bestimmen der einzelnen Szenen, besteht aus *Events*, *Static Scene* und *Key Objects*. [22].

formationen können also zurückverfolgt werden und in einem wissenschaftlichen Kontext verwendet werden“ [5, S. 43].

Als wichtig wird herausgehoben, dass XRM objektorientiert programmiert wird. Das ist bedeutend, weil sich somit alle Informationen in objektähnlichen Strukturen kapseln lassen. So sollen auch Redundanzen vermieden werden.

Es wurde ein eigener Videoplayer *CoolPlayer* entwickelt, der mit XRM-Dokumenten umgehen kann. So zeigt er gleichzeitig Videos und Zusatzinformationen an (siehe Abb. 3.6). Der Videoplayer selbst wurde dabei in verschiedene Bereiche aufgeteilt. Mittig ist das aktuelle Video zu sehen. Links davon werden *statische Informationen*, also allgemeine Informationen, die das ganze Video betreffen, angezeigt. Rechts davon befinden sich *Strukturinformationen* wie die einzelnen Szenen und Kapitel. Unter dem Video befindet sich der *dynamische Bereich* für *Timestamp* basierende Informationen, die nur eine begrenzte Gültigkeitsdauer besitzen [5, Kap. 5.2, S. 89–91].

Für das Video bleibt wenig Platz, da die unterschiedlichen Metainformationen viel Platz einnehmen. Soll ständig auf alle Zusatzinformationen zugegriffen werden können, muss eine geringere Anzeigegröße des Videos in Kauf genommen werden.

Abbildung 3.6: Gliederung des Videoplayers in drei Bereiche für Zusatzinformationen. Links befindet sich der statische Bereich, rechts die Strukturinformationen und unten der dynamische Bereich [5, Abb. 5.4].

3.8 Interaktives Fernsehen

Jörg Broszeit ist der Auffassung, dass Fernsehen und Internet bisher zwei getrennte Welten waren. Doch im Rahmen der Medienkonvergenz würden die Grenzen der verschiedenen Einzelmedien immer stärker verschwimmen. Denn dank der Digitalisierung der Programme und Geräte, ist inzwischen „Fernsehen über das Internet und Internet auf dem Fernseher“ möglich [1, S. 13].

Weiters schreibt er, dass „die Rezeption von Fernsehen als eine sehr passive Tätigkeit angesehen“ wird. Mit dem interaktiven Fernsehen allerdings (auch iTV genannt), ist eine stärkere Beeinflussung des TV-Programmes gemeint. Als Beispiele nennt er, neben einer linearen Steuerung oder durch eine Zuseherabstimmung den Ablauf einer Sendung zu verändern, auch „Zusatzinformationen und -dienste zum laufenden Programm“ [1, S. 16].

Nicholas Negroponte, ein amerikanischer Informatiker, Begründer und Direktor des MIT Media Lab, beschreibt das interaktive Fernsehen folgen-

dermaßen: ein „Medium, das wie ein Buch oder eine Zeitung einen zufälligen Zugriff erlaubt. Man kann es durchblättern und verändern und ist nicht länger von bestimmten Tagen, Zeiten oder einer festgelegten Übertragungsdauer abhängig“ [8, S. 67].

3.8.1 Interaktive Projekte

Es gibt inzwischen einige bestehende Projekte, um eine umfassende Interaktion mit einem Smart TV Gerät zu gewährleisten. Beispielsweise beschreibt das Dokument „Controlling the Smart Home from TV“ [3] ein Szenario, in dem es darum geht, eine interaktive Applikation zu einem Film zu starten, die die Haushaltsgeräte je nach Spannung im Film reguliert. So wird in einer stressigen Situation das Licht abgedunkelt und der Staubsauger gestartet um die Spannung noch zusätzlich zu erhöhen.

Ein anderer Fall den Fernseher mit einem technischen Gerät kommunizieren zu lassen ist während einer Kochsendung direkt ein Rezept an den Drucker zu senden und drucken zu lassen [2].

Ein nützlicher Anwendungsfall ist auch die Vernetzung der Eingangskamera mit dem Smart TV Gerät. Angenommen der Ehemann sieht gerade einen Film, während seine Frau nach Hause kommt und an der Tür klingelt. Am Monitor erscheint ein kleines Fenster mit dem Bild der Videoübertragung der Kamera, und der Mann kann seiner Frau mittels der Fernbedienung die Türe öffnen, ohne den Film unterbrechen zu müssen [3].

Die Arbeit „Smart Audio/Video Playback Control Based on Presence Detection and User Localization in Home Environment“ von vier serbischen Autoren [7] bietet auch einen interessanten Ansatz bezüglich Interaktion. Hierbei geht es darum, mithilfe dreier Sensoren (3D-Kamera, Mikrofone und Passiver Infrarot Sensor) die aktuelle Position eines Zusehers zu bestimmen. Je nachdem wie nahe oder wie entfernt der Benutzer dem Smart TV ist, wird die Musik leiser oder lauter reguliert, oder ab einer bestimmten Entfernung der Film pausiert. Eine Beispielabbildung aus Sicht der Kamera bietet Abb. 3.7.

3.8.2 Verwendung von externen Geräten

Wie die Verwendung von beispielsweise Mobiltelefonen das Fernsehen an sich verändern kann, soll folgendes Szenario aus [4] verdeutlichen: Angenommen jedes Familienmitglied besitzt ein eigenes Handy und trägt es immer bei sich. Das Handy wird ab einer gewissen Distanz automatisch via Bluetooth mit dem Smart TV verbunden und ein neues Profil wird angelegt. In diesem Profil wird gespeichert, welche Sendung angesehen wird, solange das Mobiltelefon mit dem Fernseher verbunden ist. So kann nach einer Weile ein Muster erkannt werden, welche Sendungen der jeweilige Zuseher gerne sieht, und in weiterer Folge können Empfehlungen zu anderen Programmen

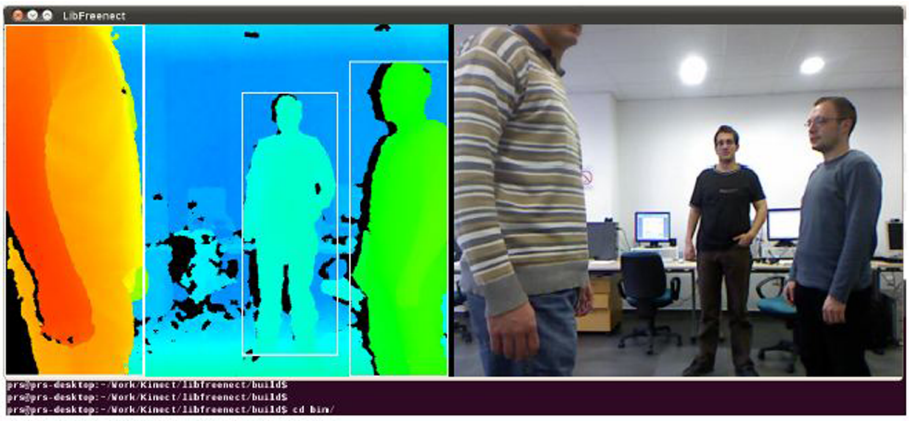

Abbildung 3.7: Personenerkennung und Verfolgung mithilfe einer Microsoft Kinect Kamera [7].

gegeben werden, die dem Benutzer gefallen könnten. Auch wenn mehrere Familienmitglieder zusammen fernsehen, kann aufgrund der einzelnen Vorlieben vielleicht ein Film vorgeschlagen werden, der jedem zusagt.

Ein weiterer Ansatz ist eine Information zum Fernsehbild auf ein externes Gerät auszulagern. Um bei einem Pferderennen keinen Platz am Bildschirm zu verbrauchen, können zugehörige Wettinformationen am Handy angezeigt werden. Ebenso können bei den Olympischen Spielen Informationen über parallel ablaufende Bewerbe auf das Mobiltelefon übertragen werden, oder Videos von einer Wiederholung [4].

3.9 Vergleich der vorgestellten Systeme

Im Nachfolgenden werden die zuvor genannten Systeme anhand einiger Kriterien verglichen. Aus Platzgründen wurden die Ergebnisse auf Tabelle 3.1 und Tabelle 3.2 aufgeteilt. Als Kriterien wurden die folgenden gewählt:

Zusatzinformationen sagt aus, ob das System dem Benützer Zusatzinformationen liefert.

Timestamp basierend definiert, ob die Zusatzinformationen die angeboten werden allgemein gelten, oder nur zu einem bestimmten Zeitpunkt zur Verfügung gestellt werden.

Parallel zur Sendung sagt aus, ob die zusätzlichen Informationen gleichzeitig zur Sendung oder zu einem Film zur Verfügung stehen, oder nur stattdessen bzw. danach.

Selbst erweiterbar bezeichnet, ob die Metainformationen selbst angepasst und erweitert werden können, oder fix von einem Anbieter kommen.

Info nicht im Bild ist das Kriterium das besagt, ob die Informationen ex-

Tabelle 3.1: Überblick der Zusatzinformationsdienste *TV Second Screen*, *Blu-ray Second Screen* und *Google Info Cards*.

Funktion	*TV SS*	*Blu-ray SS*	*Google IC*
Zusatzinformationen	✓	✓	✓
Timestamp basierend	✓	✓	✓
Parallel zur Sendung	✓	✓	✓
Selbst erweiterbar	✗	✗	✗
Info nicht im Bild	✓	✓	✗
App	✓	✓	✓
Web	✓	✗	✗
Geräteunabhängig	✓	✗	✗
Proprietär	✓	✓	✓

tern auf einem eigenen Gerät zur Verfügung gestellt werden, oder ob sie im selben Bild wie die Sendung Platz einnehmen.

App bezeichnet, ob eine eigene Applikation oder ein Programm installiert werden muss, bevor die Zusatzinformationen empfangen werden können.

Web sagt aus, ob das System frei über eine Internet-Applikation zugänglich ist, ohne dasselbe installieren zu müssen.

Geräteunabhängig beschreibt, ob das System für viele Geräte zur Verfügung steht, oder auf bestimmte beschränkt ist.

Proprietär bedeutet, dass das System im Gegensatz zu einer freien Software nicht verändert werden kann oder darf.

Tabelle 3.2: Überblick der Zusatzinformationsdienste *XBMC Media Center*, *Multimedia Home Platform*, *MPEG-7* und *eXtensible Rich Media*.

Funktion	*XBMC*	*MHP*	*MPEG-7*	*XRM*
Zusatzinformationen	✓	✓	✓	✓
Timestamp basierend	✗	✓	✓	✓
Parallel zur Sendung	✗	✓	✓	✓
Selbst erweiterbar	✓	✗	✓	✓
Info nicht im Bild	✗	✗	✗	✗
App	✓	✗	✓	✓
Web	✗	✗	✗	✗
Geräteunabhängig	✓	✗	✓	✓
Proprietär	✗	✗	✗	✗

Kapitel 4

Entwurf

Dieses Kapitel behandelt die theoretische Herangehensweise für die Umsetzung des Systems. Zuerst werden die gewünschten Anforderungen und eine beispielhafte Anwendungsmöglichkeit definiert, um die Rahmenbedingungen abzustecken. Als Nächstes folgt eine Beschreibung der Architektur für das Zusammenspiel der einzelnen Systeme. Einen wesentlichen Punkt stellt das technische Design dar, das die essenziellen Aktionen erläutert. Zum Schluss wird die am externen Gerät sichtbare visuelle Oberfläche beschrieben und skizziert.

4.1 Anforderungen

Nach der Erläuterung der bereits vorhandenen Technologien werden die eigenen Anforderungen erörtert. Die Anforderungen werden in funktionale und nichtfunktionale Anforderungen unterteilt. Die Punkte 1–7 sind funktionale Anforderungen, wohingegen die Punkte 8 und 9 nichtfunktionale Anforderungen darstellen.

1. **Zusatzinformationen:** In erster Linie soll es möglich sein, zu laufenden Sendungen im Fernsehen zusätzliche Informationen zu bekommen. In einer Umfrage von Jörg Broszeit zu den Themen Medien, Fernsehen und Internet sollten auch verschiedene interaktive Möglichkeiten beim Fernsehen bewertet werden [1]. Neben „Werbung überspringen“ und „Video-on-Demand“ spielte auch der *Electronic Program Guide* (EPG[1]) eine wesentliche Rolle bei den beliebtesten interaktiven TV-Diensten. Bei den zusätzlichen Nutzertests empfanden die Probanden das Abfragen der Zusatzinformationen zum Programm mithilfe eines EPG ebenfalls wichtig. Eine weitere Erkenntnis der Umfrage ist, dass ne-

[1] Der *Electronic Program Guide* (zu deutsch *elektronischer Programmführer*) ist im Wesentlichen ein Ersatz für die gedruckte Programmzeitschrift, und als Zusatzangebot von den verschiedenen Fersehsendern zu verstehen. Es werden Informationen zum aktuellen und zum zukünftigen Fernsehprogramm geliefert.

ben „den Informationen zum laufenden Programm [...] durchaus tiefer greifende Zusatzinformationen erwünscht“ sind [1, S. 53]. Als Beispiele dafür werden „eine direkte Verlinkung zu z. B. Lexikoneinträgen aus dem Themenbereich der aktuellen Sendung oder nähere Informationen zu den Schauspielern und Mitwirkenden“ genannt [1, S. 53].

2. **Timestamp basierend:** Um diese Funktion attraktiver zu gestalten, sollen die Zusatzinformationen zu einem bestimmten Zeitpunkt, also zum aktuellen Geschehen passend, erhalten werden. Somit besteht ein direkter Bezug und die Relevanz der Metainformation zum Sendungsinhalt kann besser nachvollzogen werden.
3. **Extra Bildschirm:** Einen weiteren Aspekt stellt die Einbuße des Fernsehbildes dar. Laut der Studie von Jörg Broszeit wurde es in vielen Fällen „als besser empfunden, wenn die eingeblendeten Informationen nur einen Teil des Fernsehbildes überdecken, und ein Weiterverfolgen des Programms möglich ist“ [1, S. 53]. Eine Möglichkeit der uneingeschränkten, vollständigen Darstellung des Fernsehbildes trotz Metainformationen ist deren Auslagerung auf ein externes Gerät. Dadurch ergibt sich ein zusätzlicher Vorteil der besseren Lesbarkeit durch die unmittelbare Nähe der Anzeige beim Benutzer. Wenn eine Zusatzinformation in einer zu kleinen Schriftgröße am Fernseher dargestellt wird, muss der Zuseher aufstehen, und sich dem Fernseher nähern, um alles lesen zu können. Wird der Text allerdings auf einem Tablet, das man in der Hand hat, angezeigt, kann der Benutzer zoomen oder das Tablet näher an die Augen halten. Mit diesen Möglichkeiten können dem Anwender in kleinerer Schriftgröße mehr Text-Informationen übermittelt werden.
4. **Kategorien:** Durch Kategorisierung soll der Benutzer die Chance haben die Metainformationen auf den ersten Blick zuordnen zu können. Außerdem kann sich der Anwender dadurch auf eine bestimmte Kategorie, wie beispielsweise alle Ortsangaben, fokussieren. Die verschiedenen Genres sollen einerseits grob gehalten werden, aber dennoch eindeutig zuordenbar sein. Sinnvoll sind Angaben zu Orten, Musik, Schauspielern, Produktplatzierungen und allgemeinen Informationen. Der Benutzer soll die Möglichkeiten haben jede angebotene Zusatzinformation zu empfangen, sich auf eine spezielle Kategorie zu konzentrieren, oder alle bereits gesendeten Informationen in einem Verlauf nachlesen zu können.
5. **Plattformunabhängig:** Um die Applikation für den Anwender weitläufig einsetzbar zu machen, soll deren Benützung plattformunabhängig funktionieren. Es soll keine Rolle spielen, von wem das mobile Endgerät hergestellt wurde oder welche Größe das Display hat.
6. **Offen für Community:** Damit die erstellten Metainformationen leicht erweiterbar und aktualisierbar sind, ist eine Wartung der Informatio-

nen durch eine Community sinnvoll. So können veraltete Links z. B. erneuert, oder etwaige Fehler von anderen ausgebessert werden. Dazu müssen die Zusatzinformationen gekapselt abgespeichert sein, um eigenständig bzw. lose zu bleiben.

7. **Backend:** Um das im vorherigen Punkt vorgestellte Prinzip technisch umzusetzen, soll ein gemeinsames Backend für jeden zugänglich gemacht werden. Somit kann die Community effektiver Zusatzinformationen hinzufügen. Dieses Backend soll unkompliziert gestaltet sein, um die Funktionalität einer breiten Masse verständlich zu machen. Durch eine selbsterklärende Eingabemaske sollen Titel, Kategorie, *Timestamp* und Information einfach eingegeben werden können.
8. **Unkomplizierte Konfiguration:** Eine zusätzliche Anforderung an das System ist, dass der Aufbau unkompliziert zu konfigurieren ist. Es soll für einen Laien einfach sein, die interaktive Applikation zu bedienen.
9. **Intuitives Interface:** Überdies wird ein ansprechendes und intuitives Interface benötigt. Der Anwender soll instinktiv die Funktionalitäten der Client-Applikation ausfindig machen und verstehen. Die Benutzeroberfläche soll ohne Worte auskommen und durch aussagekräftige Symbole die verschiedenen Eigenschaften zu erkennen geben.

4.2 Szenario

Die Arbeit geht von folgender Annahme aus: Der Benutzer besitzt ein Smart TV Gerät und ein Tablet. Während er einen Film sieht und im Bild ist beispielsweise der Central Park zu sehen, soll ihm auf seinem Tablet die Lage auf Google Maps oder der zugehörige Artikel auf Wikipedia zur Verfügung gestellt werden (siehe Abb. 4.1). Diese zwei Informationen sind dann in diesem Fall in der Kategorie „Ortsangabe“ zu finden. In gleicher Weise können auch automatisch Informationen zu z. B. Soundtracks oder Schauspieler angeboten werden (siehe Abb. 4.2), die dann in den Kategorien „Musik“ und „Personen“ auftauchen.

Das Tablet dient einerseits als Empfänger für diese gezielten Zusatzinformationen, andererseits kann der Benutzer auch in Eigenregie selbst nach weiteren Informationen suchen und weiterführende Links verfolgen.

4.3 Architektur

Die Architektur des Systems und das Zusammenspiel aller benötigten Komponenten werden in den folgenden Zeilen erläutert und in Abb. 4.3 abgebildet.

Die Medien, die später mit Metadaten angereichert werden, sind auf einem externen Medien-Server gespeichert. Um z. B. einen Film, der auf diesem

Abbildung 4.1: Im Fernseher ist der Central Park zu sehen, und am Tablet wird der dazu passende Artikel auf Wikipedia angezeigt.

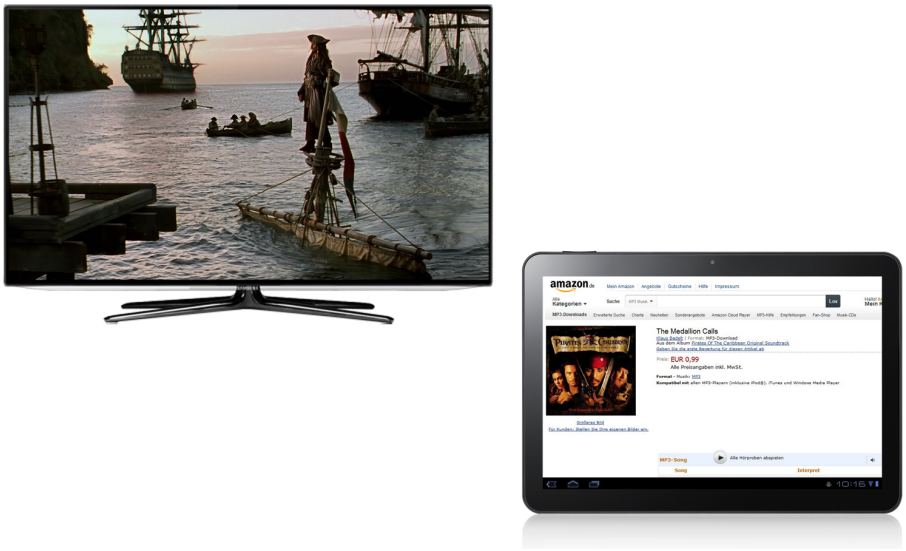

Abbildung 4.2: Hier wird am Tablet der im Film „Fluch der Karibik“ gespielte Soundtrack „The Medallion Calls“ auf der Amazon-Webseite zum Kauf angeboten.

Server abgelegt wurde, auf einem Smart TV wiederzugeben, wird mittels der Applikation die Datei aufgerufen und per *Streaming* übertragen.

Auf einem weiteren Server, dem Metadaten-Server, sind mehrere XML-Dateien mit Zusatzinformationen gespeichert. Jede der Dateien enthält eine

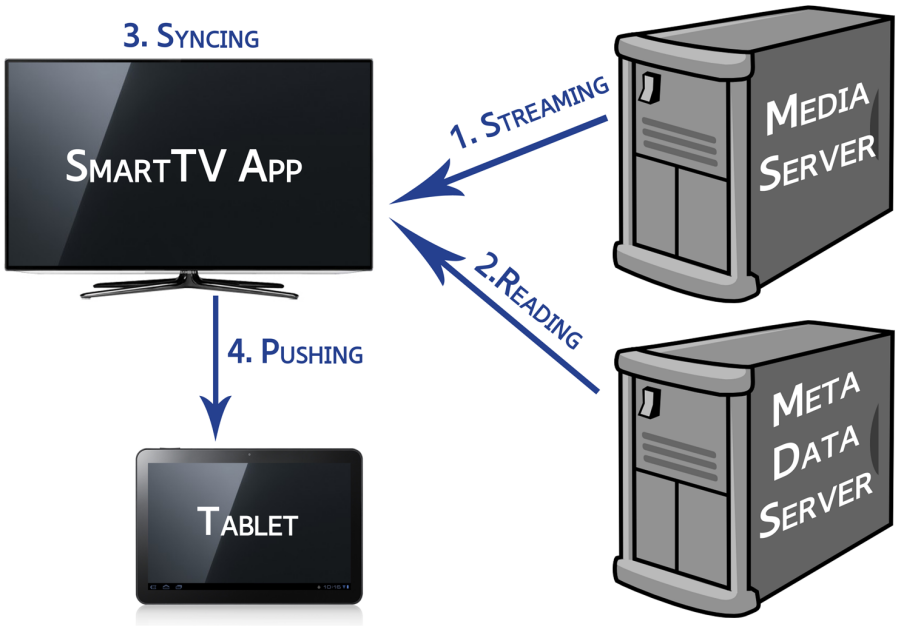

Abbildung 4.3: Die Architektur des Systems setzt sich aus *Streaming*, *Reading*, *Syncing* und *Pushing* zusammen.

eindeutige Identifikationsnummer, damit sie einem bestimmten Film zugeordnet werden kann. Weiters besteht jedes XML-File aus mehreren Events mit zugehörigen *Timestamps*. Ein *Timestamp* gibt an, zu welcher Sekunde die Metainformation im Film relevant ist. Diese Informationen können ebenfalls über die Applikation ausgelesen werden.

Wenn ein Film und die zugehörige Datei mit den Zusatzinformationen durch die Applikation geladen wurden, müssen sie zusammengeführt und synchronisiert werden.

Sobald der Film gestartet wird, wird die momentane Laufzeit des Films in kurzen Abständen mit dem nächsten *Timestamp*, der im XML-File angegeben wurde, verglichen. Das geschieht, um zu überprüfen, ob eine relevante Zusatzinformation zum aktuell Gesehenen vorliegt. Ist dies der Fall, sendet die Applikation ein Signal an das verbundene externe Gerät und zeigt die passende Information am Gerät an (z. B. in Form eines Hyperlinks).

Das externe Gerät, beispielsweise ein Tablet, kann sich nur zur Applikation verbinden, wenn sich das Smart TV Gerät und das Tablet im selben Netzwerk befinden und die IP-Adresse des Fernsehers bekannt ist.

4.3.1 XML-Datei für Metainformationen

In diesem Abschnitt wird einerseits der korrekte Aufbau einer XML-Datei für die Speicherung von Metainformationen erläutert, andererseits wird eine mögliche Generierung dieses Files vorgestellt.

Format

Damit die Zusatzinformationen richtig ausgelesen und später verwendet werden können, muss deren Speicherung nach einem bestimmten XML-Schema erfolgen. Wie dieses Schema aussehen kann, wird in Programm 4.1 gezeigt. Das zugehörige XML-Dokument mit einer eindeutigen Identifikationsnummer `MovieID`, einem `Title`, dem zugehörigen `Timestamp` in Millisekunden, einem weiterführenden `Link` und einer `Category` sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8" ?>
<nodes>
  <node>
    <MovieID>123456</MovieID>
    <Title>Der Titel der Metainformation</Title>
    <Timestamp>3000</Timestamp>
    <Link>http://www.beispiellink.at</Link>
    <Category>music</Category>
  </node>
</nodes>
```

Dabei ist zu beachten, dass bei `Category` nur bestimmte Stichwörter zugelassen werden, da sie auf fünf verschiedene Kategorien beschränkt sind:

location wird für alle geografischen Angaben verwendet.

music wird bei Musik und Soundtracks angegeben.

actor wird geschrieben, wenn man Schauspieler bzw. Personen im Allgemeinen beschreiben möchte.

product wird dann verwendet, wenn zu eingeblendeten Produktplatzierungen nähere Informationen geliefert werden.

other wird für alle allgemeinen und sonstigen Informationen ausgewählt.

Die angegebene `MovieID` ist jene, die für die eindeutige Zuweisung zu einem Film benötigt wird. Das in Abschnitt 4.4.3 beschriebene Verfahren zum Auslesen der filmrelevanten XML-Dateien verwendet genau dieses XML-Tag, um die passenden Dateien zu finden.

Generierung

Die Metadaten in den XML-Files müssen, wie Abschnitt 4.3.1 bereits erwähnt, einem bestimmten Schema entsprechen, damit sie richtig ausgelesen werden können. Wenn der Gedanke einer großen Community verfolgt wird, die die Zusatzinformationen gemeinsam wartet, soll die Eingabe dieser Informationen einfach und einheitlich erfolgen. Durch ein zur Verfügung gestell-

Programm 4.1: Das XML-Schema zum Abspeichern von Metainformationen zu einem Film.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xs:schema attributeFormDefault="unqualified" elementFormDefault="
      qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="nodes">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element maxOccurs="unbounded" name="node">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="MovieID" type="xs:unsignedInt" />
10              <xs:element name="Title" type="xs:string" />
11              <xs:element name="Timestamp" type="xs:unsignedShort" />
12              <xs:element name="Link" type="xs:string" />
13              <xs:element name="Category" type="xs:string" />
14            </xs:sequence>
15          </xs:complexType>
16        </xs:element>
17      </xs:sequence>
18    </xs:complexType>
19  </xs:element>
20 </xs:schema>
```

tes *Content-Management-System* (CMS)[2] als Backend, das die XML-Dateien automatisch generiert, ist dies realisierbar.

Wird eine neue Metainformation zum System hinzugefügt, muss das Backend in einem Browser geöffnet werden. Dort wird auf ein Formular weitergeleitet, das die essenziellen Formularfelder enthält. Die Felder sind:

MovieID um die Metainformation einem speziellen Film eindeutig zuordenbar zu machen.

Title als kurze Überschrift der Metainformation.

Timestamp ist der Zeitpunkt, zu dem die Information relevant ist.

Link ist die URL zu der detaillierten Zusatzinformation.

Category ist als Auswahlliste vorhanden, wobei aus den Kategorien *location*, *music*, *actor*, *product* und *other* eine gewählt werden kann.

Wurde alles ausgefüllt, können die Eingaben gespeichert werden. Jede Metainformation wird anhand der eindeutigen Identifikationsnummer, der *MovieID*, sortiert und zusammengefasst. Für jede Gruppe an *MovieIDs* gibt es einen Link, der einen Export der Metadaten erlaubt. Wird dieser ausgewählt, wird automatisch eine XML-Datei mit den vorhandenen Zusatzinformationen im richtigen Format erstellt und am Metadaten-Server gespeichert.

[2]Ein *Content-Management-System* (deutsch *Inhaltsverwaltungssystem*) ist ein Programm zur gemeinschaftlichen Inhaltserstellung und -verwaltung.

Somit ist sichergestellt, dass die eingegebenen Informationen genau in der Form vorliegen, wie sie von der Smart TV Applikation benötigt werden. Überdies kann der Benutzer intuitiv damit arbeiten, ohne sich mit dem Schema des XML-Files beschäftigen zu müssen.

4.3.2 Applikation am Smart TV

Um eine Vorstellung davon zu haben, wie die Applikation, die am Smart TV ausgeführt wird, funktioniert, handelt dieser Abschnitt von ihren Aufgabenbereichen und der Ablaufreihenfolge.

Aufgabenbereiche

Die Applikation, die am Fernseher ausgeführt wird, besteht aus folgenden Elementen, die bestimmte Aufgaben übernehmen müssen:

- Videoplayer,
- Display,
- Applikationsserver,
- Datenverwaltung,
- Eventverwaltung,
- Hauptapplikation und
- Device Verwaltung.

Vorweg wird ein **Videoplayer** benötigt, der die Befehle zur Videosteuerung ausführt, und die gewünschten Videodateien wiedergibt. Die gewöhnlichen Funktionen eines Videoplayers wie die Auflistung der vorhandenen Filme und die Steuerfunktionen wie Abspielen, Pausieren, etc. sollen gegeben sein.

Um die Funktionalitäten und das Interface des Players anzuzeigen, wird ein **Display** gebraucht, das diese Forderungen übernimmt. Dazu gehört das Anzeigen der Videoliste, der Laufzeit und der Lautstärke.

Der **Applikationsserver** ist für das Anfordern der Video- und Zusatzinformations-XML-Dateien zuständig. Nachdem diese erhalten wurden, werden sie dort ausgelesen und zum Verarbeiten weitergegeben.

Weiterverarbeitet werden die Inhalte der videospezifischen XML-Dateien in der **Datenverwaltung**, indem sie organisiert und strukturiert abgespeichert werden.

Mit ähnlichen Aufgaben wie die der Datenverwaltung übernimmt die **Eventverwaltung** die relevanten Dateien für die Zusatzinformationen. Zusätzlich passiert hier der Abgleich der Laufzeit des Videomaterials mit den vorhandenen *Timestamps*.

Die **Hauptapplikation** hat die Aufgabe zuerst alle einzelnen Bereiche zu initialisieren. Danach ist sie dafür zuständig alle Eingaben durch die Fernsteuerung anzunehmen und die zugehörigen Befehle an die jeweiligen Bereiche weiterzuleiten.

Außerdem gibt es eine **Device Verwaltung**, unter deren Verantwortung der Verbindungsaufbau zu einem externen Gerät steht. Das verbundene Gerät wird hier verwaltet, es wird auf alle Statusänderungen reagiert, und die vorhandenen Zusatzinformationen weitergeleitet.

Ablauf

Nachdem die relevanten Bereiche erläutert wurden, wird im Folgenden der grobe Ablauf der Applikation skizziert:

1. Start der Applikation.
2. Die Verbindung zu einem externen Gerät wird aufgebaut.
3. Die vorhandenen Videos werden ausgelesen.
4. Der gewünschte Film wird ausgewählt.
5. Der Film wird wiedergegeben.
6. Die den Videos zugehörigen Metadaten werden ausgelesen.
7. Die Metadaten werden lokal gespeichert.
8. Die verfügbaren *Timestamps* werden mit der Laufzeit abgeglichen.
9. Bei Übereinstimmung von *Timestamp* und Laufzeit wird die Metainformation an das externe Gerät gesendet.

Auf Details, wie die einzelnen Schritte durchgeführt werden, wird in Abschnitt 4.4 eingegangen.

4.3.3 Applikation am externen Gerät

Bei der Applikation, die am externen Gerät ausgeführt wird (in weiterer Folge auch Client genannt), gibt es auch verschiedene Aufgabenbereiche. Auf der Seite des Clients muss ebenso dafür gesorgt werden, dass eine Verbindung zustande kommt. Sobald die Verbindung steht, wird der Benutzer auf eine neue Seite weitergeleitet und das externe Device ist bereit Metainformationen zu empfangen. Wenn eine Information bereitsteht, wird ihre Kategorie abgefragt und der verfügbare Link im Datenstromkanal, in der richtigen Kategorie und im Verlauf angezeigt.

4.4 Technisches Design

Dieser Abschnitt handelt vom Zusammenspiel aller benötigten Geräte, Systeme und Dateien. Es wird erklärt, wie das Auslesen, Verarbeiten und Weiterleiten der Informationen ausgeführt wird. Jeder Schritt wird einzeln und mit Grafiken unterstützt näher erläutert. Bevor jeder Teil separat behandelt wird, wird der gesamte Systemablauf mithilfe eines Sequenzdiagramms in Abb. 4.4 dargestellt.

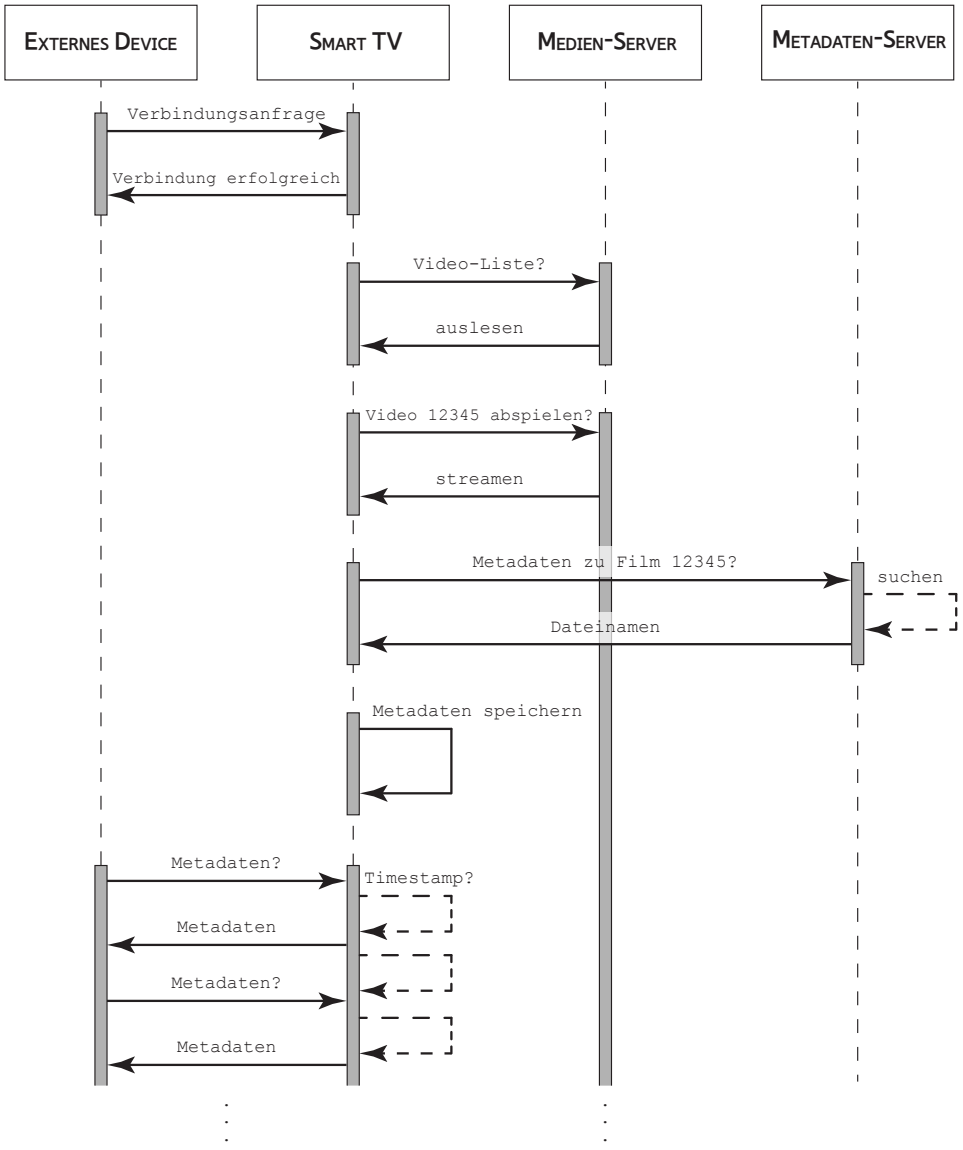

Abbildung 4.4: In diesem Sequenzdiagramm wird der komplette Ablauf des Systems gezeigt. Zuerst wird eine Verbindung zwischen einem externen Device und einem Smart TV hergestellt. Danach werden alle verfügbaren Videodateien angefordert. Wenn ein Video, z. B. 12345, zum Abspielen ausgewählt wurde, wird dieses vom Medien-Server gestreamt. Gleichzeitig werden die zugehörigen Metadaten vom Metadaten-Server angefordert, zurückgeliefert und am Smart TV gespeichert. Während der Film läuft, wird vom externen Device ständig nach Metadaten gefragt, und vom Smart TV geliefert, sobald vorhanden.

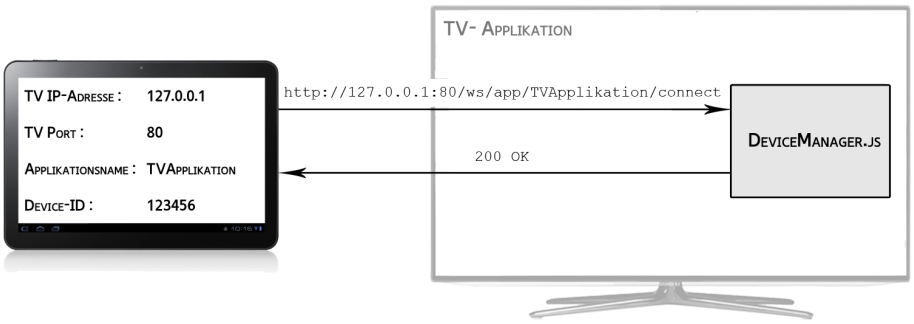

Abbildung 4.5: Um eine Verbindung zwischen Client und Server herzustellen, sendet der Client einen *AJAX-Call* an die Smart TV Applikation. Wenn sich der Client erfolgreich verbinden konnte, sendet der Server `200 OK` als *HTTP Response* zurück.

4.4.1 Verbindung zu externem Gerät

Um eine Verbindung zu einem externen Device herzustellen, müssen die Applikation am externen Device (Client), und die Applikation am Smart TV (Server) zusammenarbeiten.

Zu aller erst muss die Applikation am Fernseher gestartet sein. Dann erhält der Client ein Interface angezeigt. Dieses Interface stellt eine Eingabemaske dar, die vom Benutzer die IP-Adresse und den Port des Smart TVs, mit dem sich der Client verbinden soll, und den Namen der am Fernseher laufenden Applikation erwartet. Die Identifikationsnummer des externen Geräts wird zufällig generiert.

Mit diesen Informationen kann der Client eine Verbindungsanfrage an den Server senden. Die Anfrage wird mithilfe eines *AJAX-Calls* und einer *POST*-Operation durchgeführt, die sich aus den Angaben folgendermaßen zusammensetzt:

```
http://<TV IP Adresse>:<TV Port Nummer>/ws/app/<App Name>/connect
```

Waren die Angaben korrekt, kommt die Anfrage beim Server an und die gerätespezifischen Informationen werden ausgelesen. Wenn alles passt, schickt die Smart TV Applikation eine Antwort zurück, dass die Verbindung erfolgreich war. Dieser Ablauf wird serverseitig vom Objekt `DeviceManager` übernommen. Die Illustration dieses Vorgangs befindet sich in Abb. 4.5.

Sobald die Verbindung hergestellt wurde, wird der Benutzer auf eine andere Seite weitergeleitet. Das Device ist nun bereit weitere Nachrichten von der Smart TV Applikation zu erhalten.

4.4.2 Öffnen der externen Videodatei

Um eine Videodatei von einem externen Server öffnen zu können, muss ein spezielles `videoList.xml` Dokument angelegt werden. Dort werden die URLs

Programm 4.2: Das XML-Schema zum Abspeichern von Film-Titel, die URL zum Speicherort des Films und eine kurze Beschreibung.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="items">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="item">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string" />
              <xs:element name="url" type="xs:string" />
              <xs:element name="description" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

zu den gewünschten Filmen angegeben. Außerdem besteht die Möglichkeit, einen Titel und eine kurze Beschreibung zu speichern. Das definierte XML-Schema, um das `videoList.xml` File richtig aufzubauen, befindet sich in Programm 4.2. Ein Beispiel, wie eine Videoliste aussehen kann ist folgendes:

```
<?xml version="1.0" encoding="UTF-8" ?>
<items>
  <item>
    <title>Titel von Film A</title>
    <url>http://www.medienserver_1.at/Film_A_123456.avi/</url>
    <description>Das ist die kurze Beschreibung zu Film A (von einem
    externen Server bezogen).</description>
  </item>
  <item>
    <title>Titel von Film B</title>
    <url>http://www.medienserver_2.at/Film_B_654321.avi/</url>
    <description>Das ist die kurze Beschreibung zu Film B (von einem
    externen Server bezogen).</description>
  </item>
</items>
```

Es ist dabei darauf zu achten, dass eine eindeutige Identifikationsnummer (ID) des jeweiligen Films am Ende des Dateinamens angefügt wird. Um die passenden XML-Dateien, die für den Film verfügbar sind, zu finden, spaltet die Applikation den letzten Teil des Namens ab und filtert die ID heraus.

Das `ApplicationServer` Objekt fordert die Videoliste von der `videolist.xml` Datei an. Dann werden alle XML-Tags mit den Inhalten der Video-

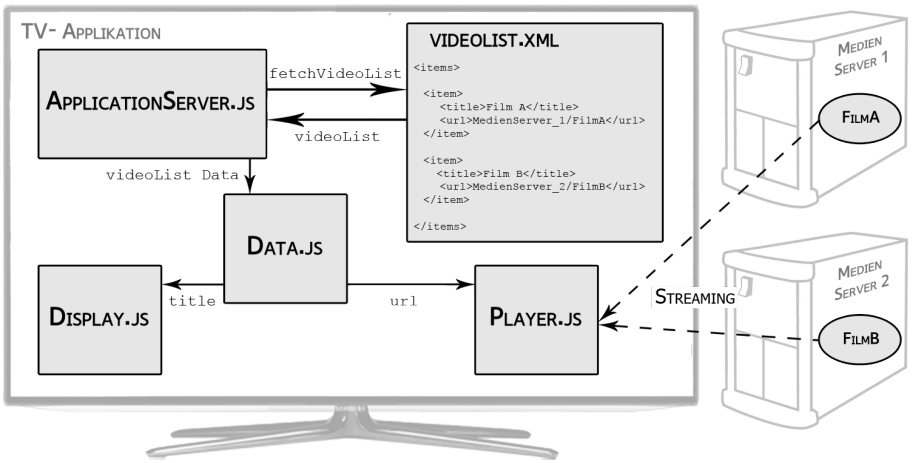

Abbildung 4.6: Vorgang beim Auslesen der Videoliste. Die `videolist.xml` Datei wird vom `ApplicationServer` Objekt angefordert und in das `Data` Objekt gespeichert. Von dort kann das `Display` Objekt z. B. den Titel beziehen, und der `Player` den Speicherort des Videos als URL. Wenn ein Video zum Abspielen gewählt wurde, streamt der Player das Video direkt vom jeweiligen Medien-Server.

liste durchgegangen und an das Datenverwaltungsobjekt `Data` zum Speichern weitergegeben. Um die vorhandenen Videos anzuzeigen, werden der jeweilige Titel und die kurze Beschreibung vom `Display` Objekt ausgelesen und im Videoplayer dargestellt. Alle angezeigten Videos können vom Benutzer ausgewählt werden. Wird ein Video zur Wiedergabe ausgesucht, fordert das `Player` Objekt die zugehörige URL vom `Data` Objekt an, um den Speicherort des Videos zu erfahren. Ist die URL bekannt, wird das Video direkt von einem Medien-Server gestreamt und im Videoplayer angezeigt. Dieser Prozess wird in Abb. 4.6 dargestellt.

Gleichzeitig wird die Identifikationsnummer des Videos ausgelesen und die zugehörigen Metainformationen abgerufen. Dieser Vorgang wird im folgenden Abschnitt näher erläutert.

4.4.3 Auslesen der externen XML-Dateien

In Abb. 4.7 wird illustriert, welche Dateien welche Aufgaben übernehmen, damit das Auslesen eines XML-Files von einem externen Metadaten-Server funktioniert.

Jeder Film, der zum Abspielen am Fernseher geladen wird, besitzt, wie zuvor erwähnt, eine eindeutige Identifikationsnummer. Diese ID wird von dem Objekt `ApplicationServer`, das in der TV-Applikation verwendet wird, ausgelesen. Mittels eines *Ajax-Requests* wird die ID an das `do_query.php` File übermittelt, das auf einem Metadaten-Server liegt. Dieser Metadaten-

Server muss derselbe sein, wie jener auf dem auch die Metadaten für den Film abgespeichert sind.

Wenn die Identifikationsnummer dort angelangt ist, übernehmen verschiedene Funktionen die Aufgabe alle verfügbaren Metadaten-Files, die zu der gesuchten Video-ID passen, zu finden. Jeder Dateiname eines relevanten Metadaten-Files wird in einem Array gespeichert. Ist das Array mit allen bedeutsamen Dateinamen befüllt, wird es JSON[3] encodiert und als Resultat (*Response*) an die Applikation zurückgesendet.

Damit das `ApplicationServer` Objekt mit dem Ergebnis weiterarbeiten kann, wird das empfangene Array wieder decodiert. Nun können die Dateinamen der relevanten XML-Files ausgelesen werden. Mithilfe der konkreten Namen können die Metadaten Files direkt vom Metadaten-Server geöffnet werden. Die Inhalte der XML-Dateien werden analysiert und auf dem Applikations-Server am Fernseher abgespeichert.

4.4.4 Verwaltung der Metainformationen

Für die Verwaltung der am Fernseher abgespeicherten Zusatzinformationen ist das Objekt `Event` zuständig. Hier werden die Informationen mithilfe verschiedener `getter`- und `setter`-Funktionen für den Titel, die Zeit, den Link und die Kategorie organisiert. Außerdem findet in diesem Objekt die Kontrolle statt, ob das laufende Video eine Zeit erreicht hat, zu der es eine Metainformation gibt. Es wird die aktuelle Laufzeit mit den vorhandenen *Timestamps* abgeglichen. Liegt eine Übereinstimmung vor, wird die Applikation darüber benachrichtigt, die Zusatzinformation wird JSON encodiert, und an ein verbundenes externes Gerät gesendet (siehe Abb. 4.8).

4.4.5 Empfangen der Metainformationen

Die Applikation am externen Gerät befindet sich in einem sogenannten *long polling* Zustand. Das heißt, der Client sendet einen *HTTP Request* an den Server und die aufgebaute Verbindung bleibt solange bestehen, bis der Server Daten zur Verfügung hat, die er dem Client zurückschicken kann, oder ein *Timeout* abgelaufen ist. In beiden Fällen wird wieder ein neuer *HTTP Request* geschickt. Es handelt sich also um eine „Endlosschleife“ an *HTTP Requests* (siehe Abb. 4.9 und Abb. 2.9). Diese Methode wird verwendet, um bei Echtzeitanwendungen ein ständiges Neuladen der Seite zu vermeiden, und Antworten des Servers, die in unregelmäßigen Abständen kommen, gut zu nutzen.

Sobald eine Zusatzinformation an das Device gesendet wurde, also eine *Response* des Servers erfolgt, wird sie von der Client-Applikation angenommen und abgespeichert. Da die Metainformation JSON encodiert vorliegt,

[3] JSON steht für ***J**ava**S**cript **O**bject **N**otation* und ist ein kompaktes Datenformat für den Datenaustausch zwischen Anwendungen.

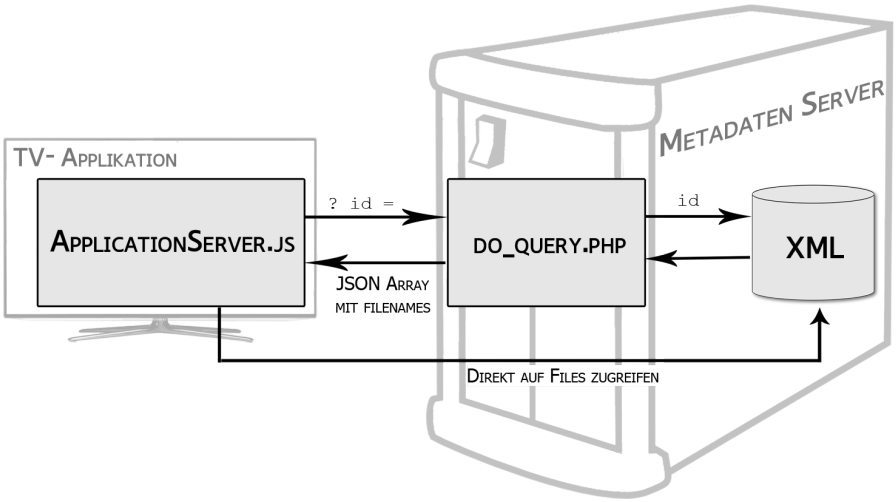

Abbildung 4.7: Zusammenspiel des Objekts `ApplicationServer`, das in der TV-Applikation läuft, und dem auf dem Metadaten-Server gespeicherten `do_query.php` File beim Auslesen der filmrelevanten XML-Files.

wird sie zuerst wieder decodiert. Danach können die einzelnen Informationen ausgelesen werden.

Zuerst kommt die Information in den Datenstromkanal, wo *alle* verfügbaren Metadaten angezeigt werden. Danach wird überprüft, welcher Kategorie das Event angehört, damit die Zuordnung in den Kategorien-Kanal richtig erfolgt. Außerdem wird die Information im Verlauf abgelegt, wo alle Metadaten in der richtigen chronologischen Reihenfolge abrufbar sind.

4.5 Benutzerschnittstelle

Unter der Benutzerschnittstelle sind die Darstellung und die Interaktionsmöglichkeiten der Applikation am externen Device zu verstehen. Da die Benutzerschnittstelle übersichtlich und intuitiv gestaltet sein soll, müssen aussagekräftige Symbole verwendet werden, um ohne Worte auszukommen.

Abgebildet wird zuerst die Eingabemaske zum Konfigurieren der Verbindung zum Server. Es werden Felder zur Angabe der IP-Adresse des Fernsehers, zur Angabe des Ports und des Applikationsnamens benötigt. Vorausgefüllt ist bereits eine zufällig generierte Identifikationsnummer für das externe Gerät.

Nach einem erfolgreichen Verbindungsaufbau wird zur eigentlichen Zusatzinformations-Seite weitergeleitet. Wie auch in Abb. 4.10 dargestellt, gibt es dort drei unterschiedliche Bereiche – einen Kopfbereich für Titelinformationen, einen großen Inhaltsbereich für die Zusatzinformationen und einen Navigationsbereich, um die verschiedenen Kategorien auswählen zu können.

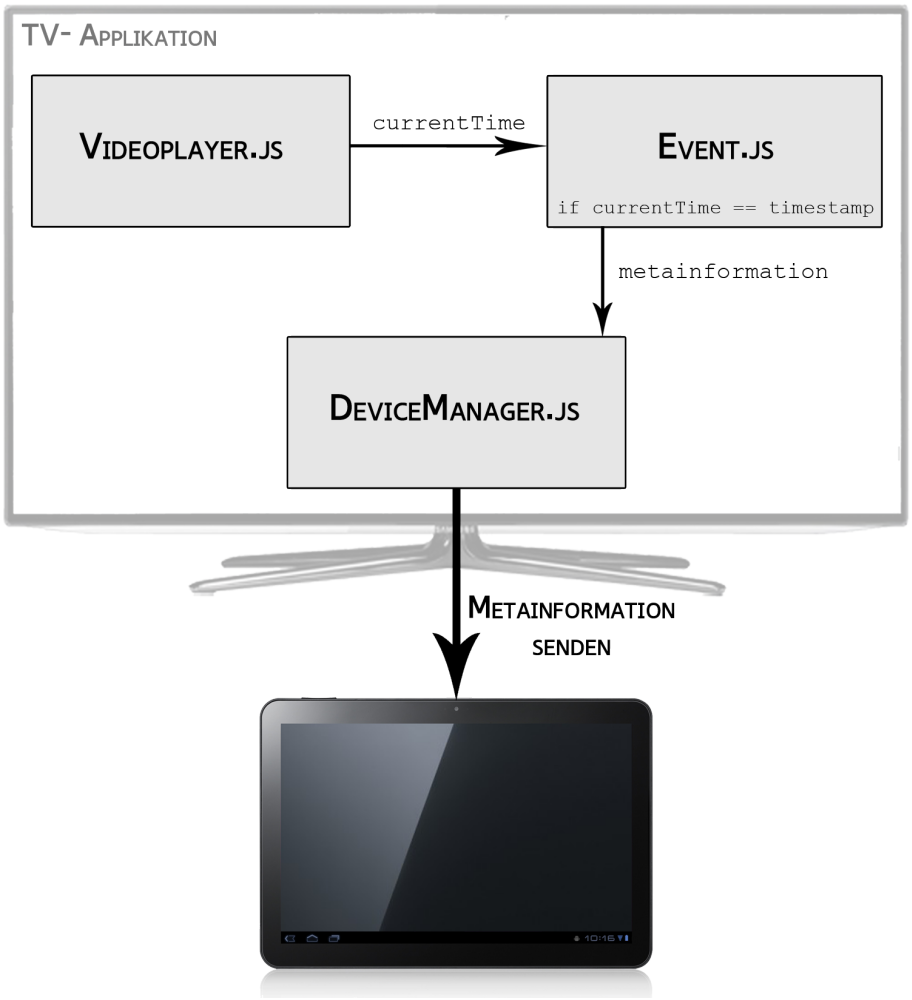

Abbildung 4.8: Der `Videoplayer` sendet die aktuelle Laufzeit des Videos an das `Event` Objekt, welches die aktuelle Zeit mit dem `Timestamp` vergleicht. Wenn diese übereinstimmen, wird das `DeviceManager` Objekt informiert, das die entsprechende Metainformation an das verbundene Device sendet.

Wenn die Anzeigeform *Stream* ausgesucht wird, soll der komplette Datenstrom dargestellt werden. Das heißt, die Zusatzinformationen werden unabhängig ihrer Kategorie der Reihe nach angezeigt. Es wird jede verfügbare Metainformation zum passenden Zeitpunkt angezeigt.

Weitere Anzeigeformen stellen die Kategorien dar. Wird beispielsweise die Kategorie *Musik* ausgesucht, werden alle Soundtracks aus den gesamten Metadaten gefiltert und angezeigt. Neben der Information zum aktuellen Lied können auch Informationen zu bereits vergangenen Liedern abgerufen werden. Die Anzeige wächst mit jeder Information aus der Musik-Kategorie.

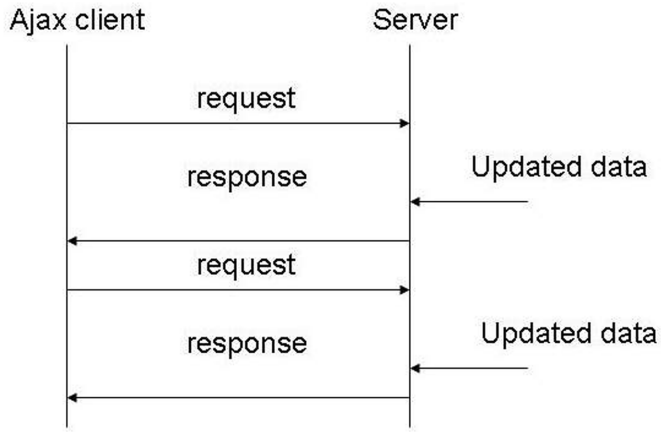

Abbildung 4.9: Beim *long polling* sendet der Client einen *HTTP Request* an den Server, und die Verbindung bleibt solange bestehen, bis der Server eine *Response* mit den gewünschten Daten zurückschickt. Danach wird ein neuer *HTTP Request* gesendet (Grafik aus [13]).

Wird zur Kategorie *Product Placement* gewechselt, scheinen die Zusatzinformationen zu allen bisherigen Produktplatzierungen auf.

Neben den Kategorien für *Ortsangaben*, *Personen* und *Sonstiges* gibt es noch einen Reiter für den *Verlauf*. Beim Verlauf geht es darum alle bisher eingelangten Zusatzinformationen in der richtigen Reihenfolge noch einmal aufrufen zu können. Hier spielen die Kategorien abermals keine Rolle. Der Verlauf ist das Äquivalent zum *Stream*, mit „Merkfunktion“.

Abbildung 4.10: Die Einteilung für die Benutzerschnittstelle der Applikation am externen Device. Oben befindet sich der Kopfbereich, für diverse Titelinformationen, darunter der Inhaltsbereich, für die eigentlichen Zusatzinformationen, und seitlich der Navigationsbereich, um zwischen den verschiedenen Kategorien zu unterscheiden.

Kapitel 5

Implementierung

In diesem Kapitel wird die Umsetzung des Projekts behandelt. Begonnen wird mit den verwendeten Technologien. Danach werden die Schnittstellen zur Samsung Smart TV Applikationsentwicklung beschrieben. Darauf folgt der Aufbau des Systems mit abschließenden Ergebnissen. Zuerst wird die Applikation, die am Fernseher ausgeführt wird, in alle bestehende Objekte zerlegt und genau erläutert. Weiters werden die Dateien der Applikation am externen Device näher beschrieben. Zuletzt wird auch auf das Backend zur Metainformations-Erstellung eingegangen. Die verschiedenen Elemente werden jeweils mit Code-Beispielen unterstützt.

5.1 Verwendete Technologien

Bei der Umsetzung der Arbeit werden verschiedene Technologien verwendet. Im nächsten Abschnitt werden das Samsung Smart TV Gerät, das Samsung TV SDK und der zugehörige Samsung TV Emulator kurz vorgestellt und erläutert, warum diese ausgewählt werden.

5.1.1 Samsung Smart TV

Bei der Wahl eines geeigneten Smart TVs, fiel die Entscheidung auf das *Samsung Smart TV*[1] Gerät. Der Grund für diese Entscheidung liegt darin, dass Samsung Geräte unter den Smart TVs und den mobilen Endgeräten weit verbreitet sind. Außerdem bietet Samsung die notwendigen Schnittstellen, um eine Smart TV Applikation zu entwickeln, die mit einem externen Device kommunizieren kann (vgl. *Convergence App API*[2]). Überdies ist das zur Verfügung gestellte *Software Development Kit* (SDK)[3] ausgereift und steht zur freien Nutzung bereit. Das zugehörige *Samsung Smart TV Apps De-*

[1] http://www.samsung.com/at/microsite/smarttv/
[2] http://www.samsungdforum.com/Guide/ref00003/index.html
[3] http://www.samsungdforum.com/Devtools/Sdkdownload

veloper Forum[4] wird von einer bestehenden Community aktiv betreut und ständig erweitert.

Samsung TV SDK

Zu Beginn der Entwicklung der Applikation, Ende des Jahres 2012, war das Samsung TV SDK mit der Versionsnummer 3.5.2 das Aktuellste. Inzwischen gibt es bereits Version 4.1 (Stand Mai 2013). Das SDK besteht neben einem Editor, einer Projekthierachie und einer Konsole auch aus den *Project Settings*, in denen applikationsspezifische Änderungen vorgenommen werden können. Außerdem inkludiert das SDK einen *TV Emulator*, auf dem Applikationen in der Entwicklung getestet werden können, und einen *Debugger*.

Samsung TV Emulator

Der zum SDK zugehörige Emulator lag Ende 2012 ebenfalls in der Version 3.5.2 vor. In dieser Version werden die Samsung TV Modelle aus dem Jahr 2012 emuliert. Ab der Version 4.0 hingegen werden die TV-Modelle aus dem Jahr 2013 berücksichtigt.

Der Emulator wird genutzt, um die entwickelte Applikation direkt am Entwicklungs-Rechner testen zu können, ohne die Applikation auf den Fernseher exportieren zu müssen. Wie in Abb. 5.1 gezeigt, ahmt der Emulator ein originales 2012-Modell eines Samsung Smart TVs nach. So kann die Funktionalität der Applikation schnell überprüft, und etwaige Fehler ausgebessert werden. Mit der ebenfalls verfügbaren Fernsteuerung werden die Benutzereingaben via Fernbedienung nachgestellt.

Da zur Zeit der Entwicklung kein Samsung Smart TV Modell aus dem Jahr 2012 zur Verfügung stand, und die Vorgänger-Modelle die benötigte `Device API` noch nicht unterstützten, konnte die Applikation ausschließlich am Emulator getestet werden.

JetBrains PhpStorm

Da der Editor von Samsung Smart TV SDK keine Code-Vervollständigung und das Springen zu Variablen-Deklarierungen bietet, wurde unterstützend die IDE *PhpStorm* von *JetBrains IntelliJ*[5] zum Entwickeln herangezogen. Das SDK wurde lediglich zum Anlegen und zum Starten der Applikation verwendet.

[4] http://www.samsungdforum.com/
[5] http://www.jetbrains.com/phpstorm/

Abbildung 5.1: Der Emulator eines Samsung Smart TVs aus der Reihe der 2012-Modelle.

5.2 Schnittstellen

Als Schnittstellen werden die verschiedenen APIs bzw. Objekte bezeichnet, die von Samsung zur Smart TV Applikationsentwicklung zur Verfügung gestellt werden. Die folgenden Elemente müssen im `index.html` File eingebunden werden:

`TVKeyValue Object` ist dafür verantwortlich den TV *Key Code* zu definieren. Das heißt, jeder Knopf auf der Fernbedienung entspricht einer Variable, die einen vom Benutzer gedrückten Knopf referenziert. Wird beispielsweise die Pause-Taste gedrückt, ist die equivalente Variable `KEY_PAUSE` [29].

`Widget Object` wird benötigt, um eine Applikation effizient zu nutzen. Das `Widget Object` inkludiert z. B. eine Methode, die den Applikations Manager über den Status der Applikation informiert [30].

`deviceapis` ist ein Modul der `Comon Web Device API` und wird z. B. verwendet, um generische *Callbacks* für Erfolgs- oder Fehler-Situationen aufzurufen [31].

5.2.1 Plug-ins

Darüber hinaus gibt es noch spezielle Plug-ins zur Samsung TV Entwicklung, die eingebunden werden können. Die folgenden drei müssen in die Applikation integriert sein, damit der Videoplayer funktioniert:

```
1 <object id="pluginPlayer"
2   classid="clsid:SAMSUNG-INFOLINK-PLAYER">
```

```
 3 </object>
 4
 5 <object id="pluginTVMW"
 6   classid="clsid:SAMSUNG-INFOLINK-TVMW">
 7 </object>
 8
 9 <object id="pluginAudio"
10   classid="clsid:SAMSUNG-INFOLINK-AUDIO">
11 </object>
```

Wie sie verwendet werden, wird in Abschnitt 5.3.1 erläutert.

5.3 Aufbau

Der Aufbau des gesamten Systems, alle benötigten Dateien und ihre Aufgaben, werden in den nächsten Unterpunkten beschrieben. Die Basis der TV-Applikation und der Client-Applikation bietet dabei eine Beispielanwendung[6] aus dem *Samsung Smart TV Apps Developer Forum.*

5.3.1 TV-Applikation

Die TV-Applikation ist jene, die am Samsung Smart TV installiert wird und die Hauptaufgaben der Funktionalität übernimmt. Abbildung 5.2 zeigt ein UML-Diagramm der wichtigsten Dateien des Systems, die im Anschluss kurz erläutert und mit Code Beispielen verdeutlicht werden.

MainApplication.js

Im File `MainApplication.js` wird das Objekt `MainApplikation` definiert, und besitzt folgende Eigenschaften:

```
1 var MainApplication = {
2   mode: 0,
3   WINDOW: 0,
4   FULLSCREEN: 1,
5
6   selectedVideo: 0,
7   currentEvent: null
8 };
```

Die Eigenschaften `mode`, `WINDOW`, `FULLSCREEN` regulieren die Anzeige des Videos. Hat `mode` den Wert 0 von `WINDOW`, wird das Video klein angezeigt, wohingegen beim Wert 1 von `FULLSCREEN` der gesamte Bildschirm ausgefüllt wird.

Die Eigenschaft `selectedVideo` gibt an, welches Video ausgewählt ist, und `currentEvent` zählt, wieviele Metainformationen bereits gesendet wurden.

[6] Beispielanwendung *Creating a Convergence Application* http://www.samsungdforum.com/Guide/tut00024/index.html

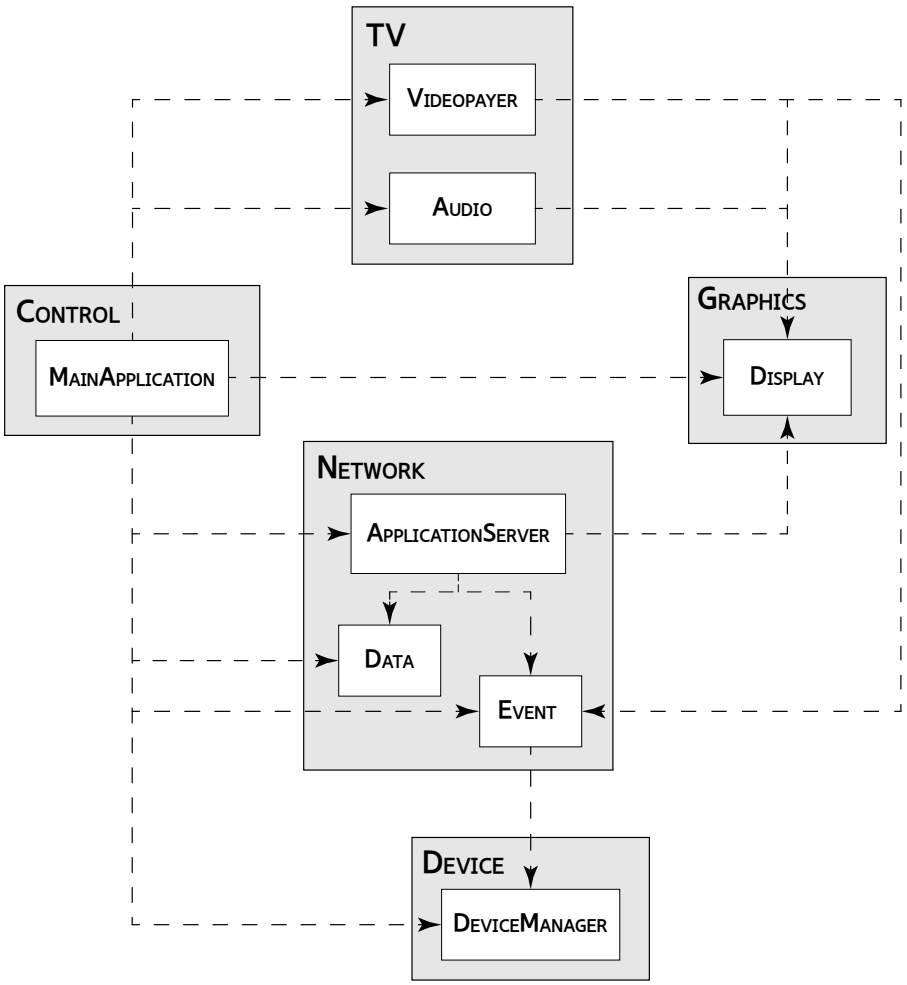

Abbildung 5.2: Die zentralen Dateien der Smart TV Applikation und ihre Beziehungen.

Dieses `MainApplication` Objekt stellt das Herzstück der ganzen Applikation dar. Es wird beim Start der Applikation als erstes aufgerufen, und initialisiert in Folge alle weiteren Objekte, die später in der Applikation benötigt werden. Erst wenn das erfolgreich war, kann die Applikation laufen.

Neben dem Starten aller notwendigen Prozesse, wie die Videoliste anfordern und anzeigen lassen, kümmert sich die `MainApplication` vorallem um die Benutzereingaben. Bei einem `keyDown`-Event, das gesendet wird, wenn bei der Fernsteuerung ein Knopf gedrückt wird, entscheidet die `MainApplication` was diese Aktion zur Folge hat. Um den `keyCode` richtig interpretiern zu können, muss die `TVKeyValue API` eingebunden sein. Wo die API eingebunden wird, wurde bereits in Abschnitt 5.2 erläutert. Verwendet

wird sie folgendermaßen:

```
var tvKey = new Common.API.TVKeyValue();
```

Im nächsten Code-Stück wird vom `keyDown`-Event der `keyCode` ausgelesen, welche Taste auf der Fernbedienung gedrückt wurde, und das Pendant ausfindig gemacht. Wird zum Beispiel die *Play*-Taste gedrückt, deligiert die `MainApplication` dem `Videoplayer` das Video zu starten.

```
1  MainApplication.keyDown = function () {
2      var keyCode = event.keyCode;
3
4      switch(keyCode) {
5          case tvKey.KEY_PLAY:
6              alert("PLAY");
7              this.handlePlayKey();
8              break;
9
10         case tvKey.KEY_STOP:
11             alert("STOP");
12             Videoplayer.stopVideo();
13             break;
14
15             .
16             .
17             .
18     }
19 }
```

Im `MainApplication` Objekt wird auch die `Widget API` verwendet:

```
var widgetAPI = new Common.API.Widget();
```

Diese wird benötigt, um dem Applikations Manager mit dem Funktions-Aufruf `widgetAPI.sendReadyEvent` mitzuteilen, dass die Applikation bereit ist, um angezeigt zu werden.

DeviceManager.js

In diesem File wird das `DeviceManager` Objekt definiert. Es dient zur Verwaltung der externen Devices.

In diesem Objekt spielt die `Device API` eine wesentliche Rolle.

```
var custom = window.deviceapis.customdevice || {};
```

Damit der `DeviceManager` richtig arbeiten kann, werden drei `Event Handler` angemeldet:

`registerManagerCallback` wird verwendet, um eine *Callback*-Funktion anzumelden, die bei einer Verbindungs-Status-Änderung eines Devices vom `Custom Device Manager` aufgerufen wird.

`getCustomDevices` wird verwendet, um eine *Callback*-Funktion anzumelden, die aufgerufen wird, wenn ein neues Device gefunden wurde.

`registerDeviceCallback` wird verwendet, um eine *Callback*-Funktion anzumelden, die aufgerufen wird, wenn beispielsweise eine Nachricht von einem verbundenen Device empfangen wurde.

Im Folgenden befinden sich die zugehörigen Funktionsaufrufe:

```
custom.registerManagerCallback(DeviceManager.onDeviceStatusChange);
custom.getCustomDevices(DeviceManager.onCustomObtained);
deviceInstance[i].registerDeviceCallback(DeviceManager.onDeviceEvent);
```

Nachdem mittels der Funktion `registerManagerCallback` ein Statuswechsel eines Devices erkannt wurde, wird die zugehörige *Callback*-Funktion `DeviceManager.onDeviceStatusChange` aufgerufen, die überprüft, ob sich das jeweilige Device verbunden oder getrennt hat. Danach ermittelt die Funktion `getCustomDevice` alle verbundenen Devices und aktualisiert in der *Callback*-Funktion `DeviceManager.onCustomObtained` die Liste aller vorhandenen Devices vom Typ `DEV_SMART_DEVICE`. Auf jedes verbundene Device wird im Zuge dessen das `registerDeviceCallback` mit der *Callback*-Funktion `DeviceManager.onDeviceEvent` angewandt.

Einen weiteren wesentlichen Teil dieses `DeviceManager` Objektes nimmt das Senden einer Metainformation an ein verbundenes, externes Device ein. Wenn das `Event` Objekt, wie in Abschnitt 5.3.1 „Event.js“ beschrieben, eine passende Metainformation zur Laufzeit findet, wird die `DeviceManager.sendEventToDevice` Funktion aufgerufen und die Zusatzinformation als Parameter mitübergeben.

```
1  DeviceManager.sendEventToDevice = function(event){
2      var jsonMessage = {link: event[0], name:event[1], category:event
       [2]};
3      var eventMessage ={"message": JSON.stringify(jsonMessage) };
4
5      for(var i=0; i<deviceInstance.length; i++){
6
7          if(deviceInstance[i]!=null && deviceInstance[i].getType() ==
       custom.DEV_SMART_DEVICE){
8
9              deviceInstance[i].sendMessage(eventMessage.message);
10         }
11     }
12 }
```

In dieser Funktion wird die Metainformation, die als Array mitübergeben wurde, zuerst in die einzelnen Bestandteile `link`, `name` und `category` zerlegt, und als JSON-Objekt in der Variable `jsonMessage` neu abgespeichert. Danach wird dieses erzeugte Objekt in einer weiteren Variable `eventMessage` als JSON-String nochmal gespeichert. Sobald die Informationen richtig codiert abgespeichert wurden, wird das Array aller gespeicherten, verbundenen Devices durchlaufen und nochmal auf ihren Typ überprüft. Wenn die Anforderungen erfüllt sind, wird die codierte Zusatzinformation mit dem Befehl `sendMessage` an jedes der im Array befindlichen Devices gesendet.

`sendMessage` wird genauso wie die Funktionen `registerManagerCallback`, `getCustomDevices` und `registerDeviceCallback` von der `Samsung Device API` zur Verfügung gestellt.

ApplicationServer.js

In dieser Datei liegt die zentrale Schnittstelle zwischen der Applikation und den externen Servern. Das `ApplicationServer` Objekt besteht aus den folgenden Eigenschaften:

```
1 var ApplicationServer = {
2     dataReceivedCallback : null,
3
4     XHRObj : null,
5     url : "XML/videoList.xml",
6
7     EventXHRObj : null,
8     eventUrl : "http://Metadatenserver.at/xml/",
9     count : 0,
10
11    allEventNames : new Array(),
12    allEventTimes : new Array(),
13    allEventLinks : new Array(),
14    allEventCategories : new Array()
15 };
```

`dataReceivedCallback` wird als *Callback*-Funktion aufgerufen, wenn alle Daten angekommen und gespeichert wurden.

`XHRObj` wird beim Abrufen der Videoliste verwendet, um einen *Ajax-Call* durchzuführen.

`url` ist der Pfad zum Speicherort der Videoliste.

`EventXHRObj` wird beim *Ajax-Call* zum Abrufen der Metadaten verwendet.

`eventUrl` gibt die URL zu einem entfernten Server an, der die Metadaten-XML-Dateien gespeichert hat.

`count` wird als Hilfsvariable eingesetzt.

Die weitern vier Eigenschaften, `allEventNames`, `allEventTimes`, `allEventLinks` und `allEventCategories` dienen jeweils zum Zwischenspeichern der ausgelesenen Metainformationen.

Um dem Benutzer die vorhandenen Videodateien anzuzeigen, müssen diese vom `ApplicationServer` ausgelesen werden. Dies wird in der Funktion `ApplicationServer.fetchVideoList` erledigt. In dieser Funktion wird ein *Ajax-Call* an die zuvor definierte URL `this.url` gesendet, wo die Pfade zu den Videodateien gespeichert sind. Wenn die Anfrage erfolgreich war, wird die Antwort in die Eigenschaft `this.XHRObj` gespeichert, und als nächstes die Funktion `ApplicationServer.createVideoList` aufgerufen.

Damit die Videoliste erstellt werden kann, muss die XML-Datei, die als Antwort auf den *Ajax-Call* erhalten wurde, richtig geparst werden.

```
1 ApplicationServer.createVideoList = function() {
2
3   var xmlElement = this.XHRObj.responseXML.documentElement;
4   . . .
5   var items = xmlElement.getElementsByTagName("item");
6
7   for (index = 0; index < items.length; index++) {
8       titleElement = items[index].getElementsByTagName("title")[0];
9       . . .
10
11      if (titleElement && descriptionElement && linkElement) {
12          videoNames[index] = titleElement.firstChild.data;
13          . . .
14      }
15  }
16
17  Data.setVideoNames(videoNames);
18  . . .
19  if (this.dataReceivedCallback) {
20        this.dataReceivedCallback();
21    }
22 }
```

Anhand dieses schematischen Stücks Code soll das Parsen des XML-Files kurz erklärt werden. Es ist dabei zu beachten, dass dieses XML-Dokument dem Schema in Programm 4.2 entspricht. Um auf die XML-Datei zugreifen zu können, wird die *Response* in die Variable `xmlElement` gespeichert (siehe Zeile 3). Da sich jedes Video innerhalb eines `<item>` Tags befindet, werden die Inhalte jedes Tag-Elements namens „item“ in Zeile 5 in die Variable `items` gespeichert. Da der folgende Teil (Zeile 7–15) für die Elemente `titleElement`, `descriptionElement` und `linkElement` vergleichbar abläuft, wurden im Code nur die relevanten Teile für `titleElement` gezeigt. Für jedes `item`-Element wird auf das Tag mit dem Namen „title“ zugegriffen, und in Folge der Inhalt, also der Titel des Videos, in das Array `videoNames` gespeichert. Ist dieser Vorgang für alle `item`-Tags abgeschlossen, wird in Zeile 17 das gesamte `videoNames` Array an das `Data` Objekt zum Speichern übergeben. Wurden alle Daten erhalten und gespeichert, wird die `dataReceivedCallback`-Funktion aufgerufen.

Der Aufruf dieser Funktion hat zur Folge, dass die ausgelesenen Namen der Videodateien an das `Display` Objekt zur Anzeige übergeben werden. Der Benutzer kann nun durch die angezeigte Liste an verfügbaren Videos navigieren. Wird ein Videoname selektiert, folgt der Aufruf, die zum Video zugehörigen Metadaten zu laden.

```
ApplicationServer.getOutput(this.selectedVideo);
```

Diese Funktion erhält über den Parameter `this.selectedVideo` den internen Index des Videos. Über diesen Index wird vom `Data` Objekt, wie in Abschnitt 5.3.1 „Data.js“ erklärt, die `MovieID` angefordert. Nach Erhalten der `MovieID`, kann ein *Ajax-Call* gesendet werden, um mit Hilfe der `MovieID`

die richtigen Metadaten ausfindig zu machen.

```
ajax.open("GET", "http://Metadatenserver.at/do_query.php?movieID="+this.
    videoMovieID, true);
```

Mit diesem Aufruf, wird der in Abschnitt 4.4.3 beschriebene Vorgang zum Auslesen der relevanten XML-Dateien gestartet. Wenn die Anfrage erfolgreich war, wird ein Array mit den entsprechenden Dateinamen der XML-Files zurückgesendet und folgenderweise weiterverarbeitet:

```
1 if(ajax.readyState == 4){
2   if(ajax.responseText != "null"){
3     var JsonText =  ajax.responseText;
4     var JsonObject = JSON.parse(JsonText);
5     var fileArray = JsonObject.result;
6
7     ApplicationServer.readXMLFile(fileArray);
8   }
9   else{
10    alert("[ApplicationServer] : no XML File");
11  }
12 }
```

Durch diese Zeilen kann das encodierte JSON-Array mit den beinhaltenden Filenamen der XML-Dateien mit Metadaten, wieder decodiert und ausgelesen werden. In der Funktion `ApplicationServer.readXMLFile` wird jeder empfangene Dateiname geöffnet und an die Funktion `ApplicationServer.fetchEventList` weitergegeben:

```
1 ApplicationServer.readXMLFile = function(fileArray){
2     for(i = 0; i < fileArray.length; i++){
3         ApplicationServer.fetchEventList(fileArray[i]);
4     }
5 }
```

In der Funktion `ApplicationServer.fetchEventList` wird noch ein *Ajax-Call* gestartet, der die betroffenen XML-Dateien anfordert (ähnlich zu der bereits beschriebenen `ApplicationServer.fetchVideoList` Funktion). Dieses Ergebnis wird an die `ApplicationServer.createEventList` Funktion weitergereicht, die die gleichen Schritte wie in der bereits veranschaulichten Funktion `ApplicationServer.createVideoList` durchführt – das Parsen der XML-Datei und das Speichern der einzelnen Informationen in die Arrays `allEventNames`, `allEventTimes`, `allEventLinks`, und `allEventCategories`.

Im Unterschied zu den Videoinformationen, werden die befüllten Arrays nicht an das `Data`, sondern an das `Event` Objekt zur Verwaltung weitergegeben. Nach dem erfolgreichen Speichern der Metainformationen in das `Event` Objekt, wird wieder die `dataReceivedCallback` *Callback*-Funktion aufgerufen.

Somit wurden alle Verbindungen zu den externen Servern für die Video- und Metadaten-Dateien hergestellt, die Inhalte ausgelesen und mithilfe des

Data und des Event Objekts lokal gespeichert.

Data.js

Das Data Objekt, ist für die Verwaltung der Videodateien zuständig. Als Eigenschaften besitzt das Data Objekt drei Arrays für die Speicherung von Video-Name, Video-URL und Video-Beschreibung.

```
1 var Data = {
2     videoNames : [],
3     videoURLs : [],
4     videoDescriptions : []
5 };
```

Den Hauptteil dieses Objekts machen die getter- und setter-Funktionen für diese Arrays aus. Die setter-Funktionen, die zum Setzen der Arrays verwendet werden, werden vom ApplicationServer Objekt aufgerufen, sobald die Informationen aus den XML-Files ausgelesen wurden. Die getter-Funktionen hingegen sind für das Auslesen der Arrays da, und werden von MainApplication und ApplicationServer benötigt.

Die Funktion getMovieID übersteigt die Funktionalität einer gewöhnlichen getter-Funktion. Sie hat die Aufgabe, die MovieID aus der bekannten VideoURL heraus zu lesen. Wie in Abschnitt 4.4.2 erwähnt, wird der Dateiname des Videos mit einer abschließenden Identifikationsnummer ergänzt. So ist der Dateiname eines Beispielfilms z. B. Beispielfilm_12345.avi. Um die Identifikationsnummer „12345“, also die MovidID zu filtern, wird folgender Code ausgeführt:

```
1 Data.getMovieID = function(index){
2   var fileUrl = Data.getVideoURL(index);
3
4   if(fileUrl){
5       var lastIndexUnderscore = fileUrl.lastIndexOf('_');
6       var lastIndexDot = fileUrl.lastIndexOf('.');
7       var movieID = fileUrl.substring(lastIndexUnderscore+1,
      lastIndexDot);
8       return movieID;
9   }
10  else return null;
11 }
```

In diesen Zeilen wird zuerst die Stelle des letzten vorkommenden Unterstrichs abgefragt, und danach die Stelle des letzten Punktes. Die Identifikationsnummer ergibt sich aus den Zeichen, die sich zwischen diesen zwei Stellen befinden. Daher ist darauf zu achten, dass die Konvention „[Dateiname]_[Identifikationsnummer].[Dateiendung]“ eingehalten wird.

Event.js

Das `Event` Objekt ist ähnlich aufgebaut, wie das zuvor beschriebene `Data` Objekt. Es dient zur Verwaltung der Metadaten, deren einzelne Felder in verschiedenen Arrays gespeichert werden.

```
1 var Event = {
2     eventNames : [],
3     eventTimes : [],
4     eventLinks : [],
5     eventCategories: [],
6     nextEvent : 0,
7     event: []
8 };
```

Auch dieses Objekt besteht hauptsächlich aus `getter`- und `setter`-Funktionen, um die Arrays zu befüllen und auszulesen. Neben den Aufrufen der `MainApplication` und `ApplicationServer` Objekte, werden die Funktionen des `Event` Objekts auch vom `Videoplayer` Objekt verwendet, weil dadurch die Metainformationen mit dem laufenden Video synchronisiert werden können. So wird z. B. beim Stoppen des Videos der Zähler für das nächste Event wieder auf 0 gesetzt. Die wichtigste Funktion stellt die `Event.checkTime` Funktion dar. Aufgerufen wird sie vom `Videoplayer` Objekt im Sekundentakt, gleichzeitig mit dem Setzen der aktuellen Zeit. Der Code für diese Funktion setzt sich wie folgt zusammen:

```
1 Event.checkTime = function(time){
2     if(time == this.eventTimes[this.nextEvent]){
3         this.event[0]=Event.getEventLink(this.nextEvent);
4         this.event[1]=Event.getEventName(this.nextEvent);
5         this.event[2]=Event.getEventCategory(this.nextEvent);
6 
7         DeviceManager.sendEventToDevice(this.event);
8 
9         this.nextEvent++;
10    }
11 }
```

Der Parameter `time` wird vom `Videoplayer` Objekt übergeben und enthält die aktuelle Spielzeit des laufenden Videos in Millisekunden. Die `if`-Abfrage überprüft, ob für die angegebene Millisekunde eine passende Metainformation vorliegt, indem die aktuelle Zeit mit dem *Timestamp* des nächsten Events verglichen wird. Ist dies der Fall, wird ein `event`-Array an den ersten drei Stellen mit dem Informations-Link, dem Informations-Namen und mit der Kategorie der Metainformation befüllt. An die in Abschnitt 5.3.1 „DeviceManager.js"beschriebene Funktion `DeviceManager.sendEventToDevice` wird das `event`-Array als Parameter übergeben, und der Zähler für das nächste Event um eins erhöht.

Videoplayer.js

Das `Videoplayer` Objekt übernimmt die Umsetzung der Steuerbefehle. Es ist dafür verantworlich, das Video zu starten, stoppen, pausieren, vor- und zurückzuspulen. Ein wesentlicher Teil des `Videoplayer` Objekts ist das *Player-Plug-in* von Samsung. Dieses Plug-in erlaubt der Applikation den Zugriff auf die Video-Steuerungs Funktionen eines Fernsehers. Nachdem es in das `index.html` File eingebunden wurde (siehe Abschnitt 5.2.1), kann es wie folgt verwendet werden:

```
this.plugin = document.getElementById("pluginPlayer");
```

Wenn das Video gestartet oder pausiert werden soll, werden diese Befehle über das Plug-in gemanagt.

```
this.plugin.Play(this.url);
this.plugin.Pause();
```

Das Player-Plug-in stellt *Events* zur Verfügung, die eigenen Funktionen innerhalb des Objekts zugewiesen werden müssen.

```
1 this.plugin.OnCurrentPlayTime = 'Videoplayer.setCurTime';
2 this.plugin.OnStreamInfoReady = 'Videoplayer.setTotalTime';
3 this.plugin.OnBufferingStart = 'Videoplayer.onBufferingStart';
4 this.plugin.OnBufferingProgress = 'Videoplayer.onBufferingProgress';
5 this.plugin.OnBufferingComplete = 'Videoplayer.onBufferingComplete';
```

Immer wenn eines dieser Events auftritt, wird die zugehörige Funktion automatisch aufgerufen. Das `OnCurrentPlayTime` Event z. B. gibt die aktuelle Wiedergabezeit des laufenden Videos an, und übergibt diese an die zugewiesene Funktion `Videoplayer.setCurTime` über einen Parameter.

```
1 Videoplayer.setCurTime = function(time) {
2     Display.setTime(time);
3     Event.checkTime(time);
4 }
```

Innerhalb dieser Funktion wird, neben dem Setzen der Spielzeit am Display, auch die `Event.checkTime` Funktion aufgerufen, zur stetigen Überprüfung, ob zur Laufzeit eine Metainformation vorliegt.

Audio.js

Das `Audio` Objekt kümmert sich um die Steuerung der Tonspur eines Videos.

```
1 var Audio = {
2     mute: 0,
3     NMUTE: 0,
4     YMUTE: 1,
5
6     plugin: null
7 };
```

Die Eigenschaften `mute`, `NMUTE`, `YMUTE` werden für die Regelung der Stummschaltung benötigt, wobei sich der Wert bei `mute` dementsprechend auf 0 (`NMUTE`, nicht stummgeschalten) oder 1 (`YMUTE`, stummgeschalten) ändern kann. Die vierte Eigenschaft `plugin` wird, wie beim `Videoplayer` Objekt, dazu verwendet das unterstützende Plug-in von Samsung einzubinden. Zum Regeln der Lautstärke werden die Plugin-in Funktionen `SetVolumeWithKey`, `GetVolume` und `SetUserMute` verwendet.

Display.js

Das Objekt `Display` repräsentiert die grafische Oberfläche der Applikation. Hier werden alle Informationen an ihrem richtigen Platz zum Darstellen vorbereitet.

Wurden alle Namen der Videoliste ausgelesen, wird

```
Display.setVideoList(Data.getVideoNames());
```

aufgerufen, damit diese für den Benutzer sichtbar am Fernseher angezeigt werden. Das `Display` Objekt kümmert sich auch um das Anzeigen der zugehörigen Beschreibungen der Videos, das Anzeigen der Lautstärke und der Laufzeit.

Ergebnis

Das visuelle Ergebnis der Smart TV Applikation kann der Abb. 5.3 entnommen werden. Im Weitern werden die mit Zahlen gekennzeichneten Bereiche kurz beschrieben:

1. **Videoliste:** In diesem Bereich werden die aus dem XML-File `videolist.xml` ausgelesenen Video-Namen dargestellt.
2. **Beschreibung:** In disem Feld wird eine kurze Beschreibung zum ausgewählten Video angezeigt.
3. **Video:** Rechts oben wird das Video selbst abgespielt. Wurde der Vollbildmodus ausgewählt, nimmt das Video den ganzen Fernsehbildschirm ein, und die anderen Elemente werden ausgeblendet (Siehe Abb. 5.4).
4. **Informationsbereich:** Hier befinden sich alle Informationen, die die Spiellänge des Videos, die Lautstärke und den aktuellen Status des Videos betreffen.
5. **Verbundenes Device:** An dieser Stelle wird das neu verbundene Device mit zugehöriger ID angezeigt. Es dient als Feedback, wenn die Verbindung hergestellt wurde.
6. **Metainformations Link:** Dieser Bereich dient eigentlich nur Testzwecken, um zu überprüfen, ob die Metainformationen gesendet werden, ohne ein Gerät verbinden zu müssen.

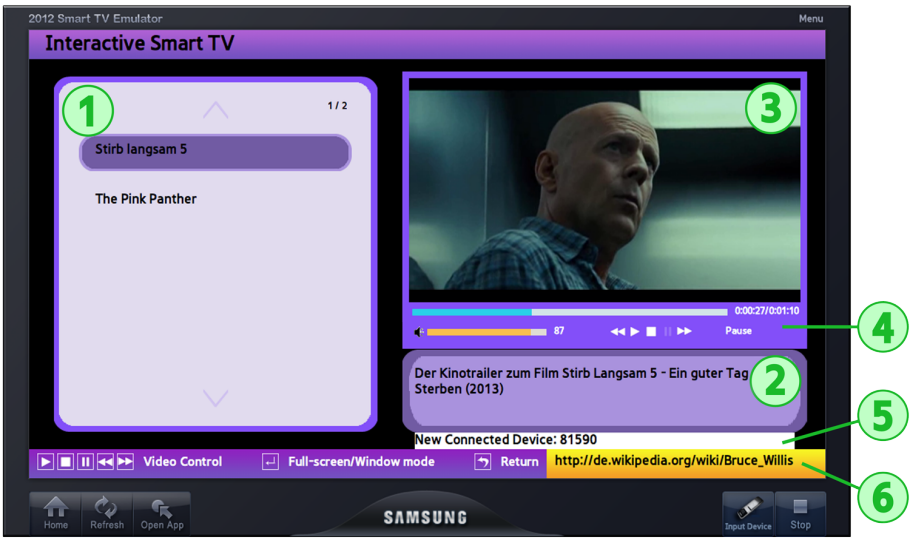

Abbildung 5.3: Der Videoplayer der Smart TV Applikation, der in folgende Bereiche aufgeteilt ist: 1. Videoliste, 2. Beschreibung, 3. Video, 4. Informationsbereich, 5. Verbundenes Device und 6. Metainformations Link.

Abbildung 5.4: Die Smart TV Applikation, wenn der Videoplayer auf *Fullscreen* angezeigt wird.

5.3.2 Client-Applikation

Die Client-Applikation bezeichnet die Benutzerschnittstelle, die der Benutzer über sein externes Gerät aufrufen kann. Innerhalb dieser Applikation

muss die Aufbereitung der Metadaten erfolgen, um sie dem Anwender zu veranschaulichen. Dem Nutzer werden über diese Applikation die Zusatzinformationen mit weiterführenden Links zugänglich gemacht.

Die Client-Applikation ist eine Webseite, die am selben Server wie die Smart TV Applikation gehostet wird. Die URL zu dieser Webseite besteht aus der IP-Adresse des Fernsehers und dem Ordner für den Speicherort der Client-Applikations Dateien.

Neben einer CSS-Datei, die das Aussehen der Webseite bestimmt, besteht die Applikation aus dem `index.html` File und der `script.js` JavaScript Datei, auf die im Folgenden kurz eingegangen wird.

index.html

Die `index.html` Datei führt beim ersten Aufruf auf die Konfigurationsseite der Applikation, die in Abb. 5.5 zu sehen ist. Sie kann nur erreicht werden, wenn die Smart TV Applikation bereits gestartet wurde.

Der Benutzer wird aufgefordert die `Service base URL` anzugeben, die sich aus der IP-Adresse des Fernsehgerätes und der Port-Nummer zusammensetzt. Bei der Angabe der Port-Nummer ist darauf zu achten, dass diese beim Ausführen der Smart TV Applikation am Emulator immer „8008“ sein muss, wohingegen beim Ausführen am Fernseher die Nummer „80“ erwartet wird. Überdies muss die `App ID`, also der Name der am Smart TV laufenden Applikation, angegeben werden. Die Identifikationsnummer des Devices wird zufällig generiert, kann aber mit einem eigenen Namen belegt werden.

Wurden alle Eingabefelder ausgefüllt, kann der Benutzer auf den `Connect` Button klicken, um die Verbindung aufzubauen. Wie der Verbindungsaufbau hergestellt wird, wird in Abschnitt 5.3.2 „script.js“ genau erklärt. Steht die Verbindung bereit, wird dem Benutzer eine neue Oberfläche angezeigt.

Die `index.html` Datei stellt nur das Grundgerüst für die Applikation dar und besteht aus vielen, leeren `div`-Containern, die mit Klassen und IDs angereichert wurden. Die Inhalte werden erst durch das `script.js` File erzeugt und eingesetzt.

script.js

Im `script.js` Dokument befindet sich der gesamte Code der Client-Applikation. Wurde der Verbindungsaufbau gestartet, wird im `script.js` File die Funktion `connect` aufgerufen. In dieser Funktion wird die URL für die Verbindungsanfrage an den Fernseher zusammengestellt, und mittels eines *Ajax-Calls* gesendet.

```
1 function connect() {
2     var URL = $("#baseURL").val() + "/ws/app/" + $("#widgetID").val() +
      "/connect";
3
4     $.ajax({
```

Abbildung 5.5: Die Eingabemaske der Client-Applikation, zur Konfiguration der Verbindung zur Smart TV Applikation.

```
5          type:"POST",
6          url:URL,
7          headers:{
8              SLDeviceID:$("#deviceID").val(),
9              VendorID:"VendorID",
10             DeviceName:$("#deviceID").val(),
11             GroupID:"GroupID",
12             ProductID:"SMARTDev"
13         },
14         async:true,
15         cache:false,
16         timeout:3000,
17
18         success:onConnect,
19         error:notConnect,
20         statusCode:{
21             200:function () {addLogMsg("200 TV Application Running");},
22             404:function () {alert("404 TV Application Not Running.");},
23             405:function () {alert("405 Method Not Allowed");},
24             409:function () {alert("409 Conflict On Device ID.");},
25             500:function () {alert("500 Server Internal Error 500");},
26             503:function () {alert("503 Server May Reach Maximum
       Connections");}
27         }
28     });
29 }
```

Die URL setzt sich aus folgendem Schema zusammen:

```
http://<TV IP Adresse>:<TV Port Nummer>/ws/app/<App Name>/connect
```

Die dafür notwendigen Informationen werden den Konfigurationsfeldern mit den IDs `baseURL` (IP-Adresse und Port-Nummer) und `widgetID` (Name der Applikation) der `index.html` Datei entnommen. Danach folgt der *Ajax-Call*, der alle `headers` über eine *POST*-Operation an die erstellte URL sendet. Der *Request* erfolgt asynchron, wird nicht in den Cache gespeichert, und besitzt ein *Timeout* von 3000 Millisekunden. Die *Response* der Smart TV Applikation enthält einen Status-Code, der über den Erfolg der Anfrage Auskunft

gibt. Wurde die Verbindung hergestellt, lautet dieser 200 und die Funktion `onConnect` wird aufgerufen. Sollte die Verbindung nicht erfolgreich gewesen sein, wird die Funktion `notConnect` aufgerufen und die dementsprechende Fehlermeldung des Status-Codes ausgegeben.

Sobald die Verbindung zwischen der Applikation am externen Device und der Applikation am Smart TV besteht, wird die Funktion `longpolling` aufgerufen. Diese hat die Aufgabe immer wieder einen *GET-Request* an den Fernseher zu senden, um anzufragen, ob bereits Metainformationen zur Verfügung stehen. Wie bereits in Abschnitt 4.4.5 beschrieben, bleibt die Anfrage des Clients solange erhalten, bis das Smart TV Gerät eine Antwort in Form einer Zusatzinformation sendet oder das vorgegebene *Timeout* abgelaufen ist. In beiden Fällen wird sofort ein neuer *GET-Request* gesendet und wieder gewartet.

```
1 function longpolling() {
2     var URL = $("#baseURL").val() + "/ws/app/" + $("#widgetID").val() +
      "/queue/device/" + $("#deviceID").val();
3     $.ajax({
4         type:"GET",
5         url:URL,
6         headers:{ SLDeviceID:$("#deviceID").val() },
7         async:true,
8         cache:false,
9         timeout:50000,
10        statusCode:{
11            200:function(){
12                addLogMsg("200 OK: Longpolling success");
13            },
14            404:function () {
15                addLogMsg("error 404, stop long polling");
16            },
17            408:function () {
18                addLogMsg("longpolling timeout 408, start again...");
19                longpolling();
20            },
21            500:function () {
22                addLogMsg("longpolling error 500, start again in 5s");
23                setTimeout('longpolling()', 5000);
24            },
25            503:function () {
26                addLogMsg("longpolling error 503, start again in 5s");
27                setTimeout('longpolling()', 5000);
28            }
29        },
30        success: function (data){
31            longpollingSuccess(data);
32        }
33    });
34 }
```

Die URL für den *Request* setzt sich wie folgt zusammen:

```
http://<TV IP>:<TV Port>/ws/app/<App Name>/queue/device/<Device Name>
```

An diese URL wird ein *Ajax-Call* gesendet, woraufhin wieder ein Status-Code von der Smart TV Applikation zurückgesendet wird. Hat die Smart TV Applikation Metainformationen zur Verfügung, wird die Metainformation als Parameter mitgeschickt. In Folge wird die Funktion `longpollingSuccess` aufgerufen und der Parameter übergeben:

```
1 function longpollingSuccess(data) {
2     var eventData = JSON.parse(data);
3
4     if (data.search("widgetInfo") != -1) {
5         var recv_msg = eval('(' + data + ')');
6         if (recv_msg.widgetInfo.Status == "TERMINATED") {
7             alert("TV application terminated, quit");
8             self.close();
9         }
10        else {
11            addEventMsg(eventData);
12            longpolling();
13        }
14    }
15    else {
16        addEventMsg(eventData);
17        longpolling();
18    }
19 }
```

In der Funktion `longpollingSuccess` wird der Parameter `data` geparst, und der Funktion `addEventMsg` übermittelt. Überdies wird überprüft, ob als Statusinformation mitgesendet wurde, dass die TV-Applikation inzwischen geschlossen ist. Ist dies der Fall, wird die Client-Applikation ebenfalls gestoppt, ansonsten wird ein neuer `longpolling` Aufruf gestartet.

Die `addEventMsg` Funktion ist dafür verantworlich, dass dem Benutzer die ausgelesenen Metainformationen am externen Gerät richtig angezeigt werden. Durch diese Funktion wird das erwähnte Grundgerüst in der `index.html` Datei mit den Informationen befüllt.

```
1 function addEventMsg(eventData) {
2     var link = eventData.link;
3     var name = eventData.name;
4     var category = eventData.category;
5
6     //Stream Section
7     $("#iframe_stream").attr("src", link);
8     $("#description_stream").html(name);
9
10    //Category Section
11    $("#accordion_" + category).prepend('<h3>' + name + '</h3><div><p><
      iframe src="' + link + '"></iframe></p></div>')
12        .accordion('destroy').accordion({collapsible:true, heightStyle:"
      fill"});
13
14    //History Section
```

```
15      $("#accordion_history").prepend('<h3>' + name + '</h3><div><p><
        iframe src="' + link + '"></iframe></p></div>')
16          .accordion('destroy').accordion({collapsible:true, heightStyle:"
        fill", active:false});
17  }
```

Der Code ist in drei Bereiche – *Stream Section*, *Category Section*, *History Section* – geteilt, da die ausgelesenen Zusatzinformationen drei mal aufbereitet werden müssen.

Stream Section: In diesem Bereich wird die aktuellste Zusatzinformation angezeigt. Wurde dieser Kanal gewählt, erhält der Benutzer jede Information, die parallel zum Film angeboten wird, als Datenstrom – ohne Beachtung der Kategorie, der sie zugeteilt ist.

Category Section: Hier wird die Kategorie der Metainformation ausgelesen und dynamisch die entsprechende ID gewählt.

```
$("#accordion_" + category)
```

Das heißt, die Information wird dem vorgesehenen Reiter der Kategorie hinzugefügt.

History Section: Dieser Bereich dient dazu, alle erhaltenen Metainformationen in der richtigen, chronologischen Reihenfolge noch einmal abrufen zu können. Wie beim *Stream* werden die Informationen aller Kategorien angezeigt, und der Verlauf wird ständig um eine neue Zusatzinformation erweitert.

Alle Informations-Links werden von *iframes*[7] eingebunden. Für die Darstellung der Informationen im Kategorien- und Verlaufs-Bereich, wird der Name des Events als Titelbeschreibung für das *Accordion*-Element[8] verwendet. Die folgende Zeile Code beschreibt das Zerstören des aktuellen *Accordion*-Elements und das sofortige neu Erstellen des selbigen. Dieser Schritt ist notwendig, um das *Accordion* um die neu erhaltene Information zu erweitern.

```
.accordion('destroy').accordion({collapsible:true, heightStyle:"fill"});
```

Da der Bereich für den *Stream*-Kanal keine Informationen speichert, sondern nur die aktuellste Metainformation anzeigt, wird dort kein *Accordion*-Element benötigt. Die grafische Umsetzung wird im nächsten Abschnitt behandelt.

Ergebnis

Dieser Abschnitt handelt von der Umsetzung der grafischen Oberfläche der Client-Applikation. Es galt darauf zu achten, ein ansprechendes und intuiti-

[7] iframe steht für ***I**nline**frame*** und bezeichnet ein HTML-Element, das zur Einbindung und Darstellung von anderen Webseiten auf einer eigenen Seite dient.

[8] Das *Accordion*-Element wird von jQuery UI zur Verfügung gestellt. Es zeigt aufklappbare Inhaltsbereiche, um Informationen in einer begrenzt verfügbaren Fläche zu präsentieren.

Abbildung 5.6: Die Symbole der Navigation stehen für *Datenstrom*, *Ortsangaben*, *Musik*, *Personen*, *Produktplatzierungen*, *Allgemeines* und *Verlauf* (v.l.n.r).

ves Interface anzubieten, damit der Benutzer instinktiv die Funktionalitäten ausfindig machen kann. Damit die Benutzeroberfläche ohne Worte auskommt und die Funktionalitäten dennoch erkannt werden, müssen aussagekräftige Symbole verwendet werden. In Abb. 5.6 werden die in der Client-Applikation verwendeten Symbole gezeigt. Sie stellen die Navigation der Applikation dar, und ermöglichen dem Anwender zwischen den verschiedenen Kanälen zu wählen.

In Abb. 5.7 ist die ganze Applikationsoberfläche zu sehen. In dieser Grafik ist der *Stream*-Kanal ausgewählt, und es wird die Metainformation zu der Szene in Abb. 5.4 gezeigt. Der ausgewählte Kanal wird grafisch durch drei Merkmale angezeigt:

1. Das ausgewählte Symbol ist größer, als die anderen Symbole.
2. Der zugehörige Reiter ragt in den Inhaltsbereich hinein.
3. Die Linie, die den Kopfbereich vom Inhaltsbereich trennt, hat die dem Icon entsprechende Farbe.

Im Falle des *Stream*-Kanals, wird der Titel der Zusatzinformation im Kopfbereich angezeigt.

Bei den anderen Kategorien befinden sich im Inhaltsbereich noch zusätzliche *Accordion*-Elemente, wie in Abb. 5.8 ersichtlich. Auf diese wird deshalb zurückgegriffen, da im Gegensatz zum Datenstrom-Kanal auch auf die vorhergehenden Informationen zugegriffen werden soll. Hier wird der Name der Zusatzinformation als Überschrift des *Accordion*-Elements verwendet, damit die Informationen unterschieden werden können. Die Grafik zeigt den Reiter für den Verlauf, in dem alle empfangenen Metainformationen chronologisch gespeichert werden. Die aktuellste Information befindet sich dabei immer oben. Die Abbildung zeigt einerseits die Applikation mit geschlossenen *Accordion*-Elementen, wie auch ein ausgeklapptes *Accordion*-Element.

Als Veranschaulichung der Kategorie-Kanäle dient Abb. 5.9. In dieser Grafik wird der ausgewählte Personen-Kanal gezeigt. Dieser besteht aus den Informationen zu den Schauspielern, die in einem Film zu sehen sind. Ist der Anwender nur an den Informationen zu diversen Schauspielern interessiert, kann er mit Hilfe dieses Kanals die Metainformationen dahingehend filtern.

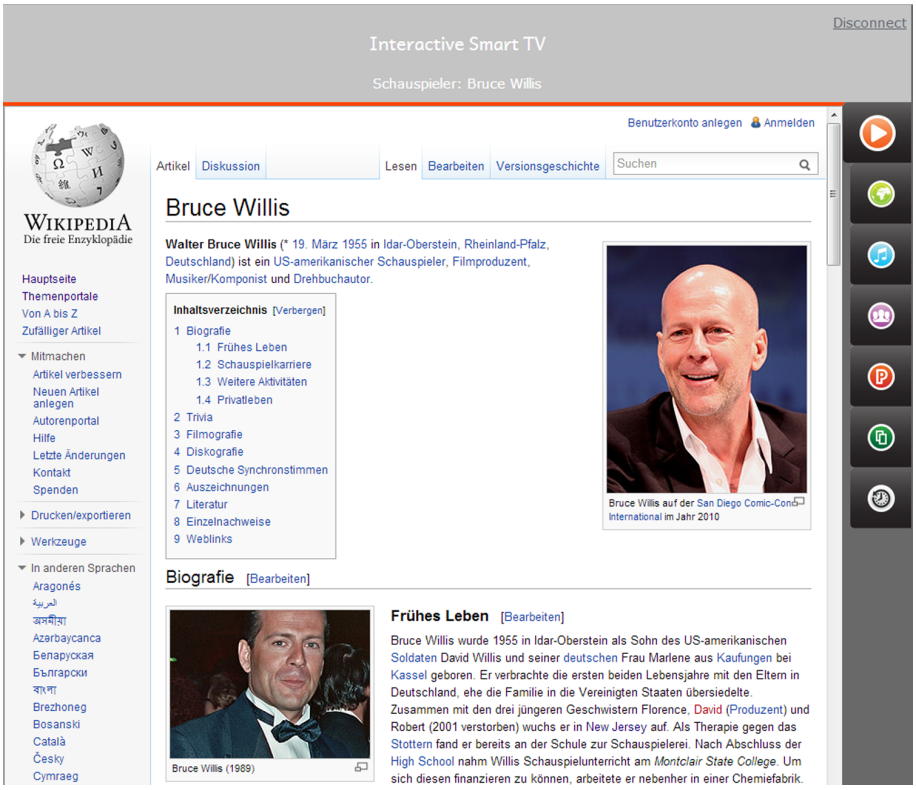

Abbildung 5.7: Die Oberfläche der Client-Applikation. Hier wurde der *Stream*-Kanal gewählt, was bedeutet, dass immer die aktuellste Metainformation angezeigt wird.

5.4 Backend zur Metainformations-Eingabe

In diesem Abschnitt wird eine Möglichkeit vorgestellt, die Metainformationen zum Film strukturiert abzuspeichern. Die Metainformationen entsprechen dem in Programm 4.1 dargestellten XML-Schema. In Abschnitt 4.3.1 wurde bereits angesprochen, dass dies mit einem gemeinsamen *Content-Management-System* als Backend möglich gemacht werden soll. Als geeignetes CMS bot sich *Drupal*[9] an, da die Bedienung einfach ist, die Funktionalität umfangreich, und für dieses System das nötige Vorwissen bereits vorhanden war.

5.4.1 Drupal

Das *Content-Management-System* Drupal wurde in der Version 7.21 verwendet. Es dient zum Erstellen, zum geordneten Anzeigen und zum Exportieren von Metainformationen.

[9] https://drupal.org

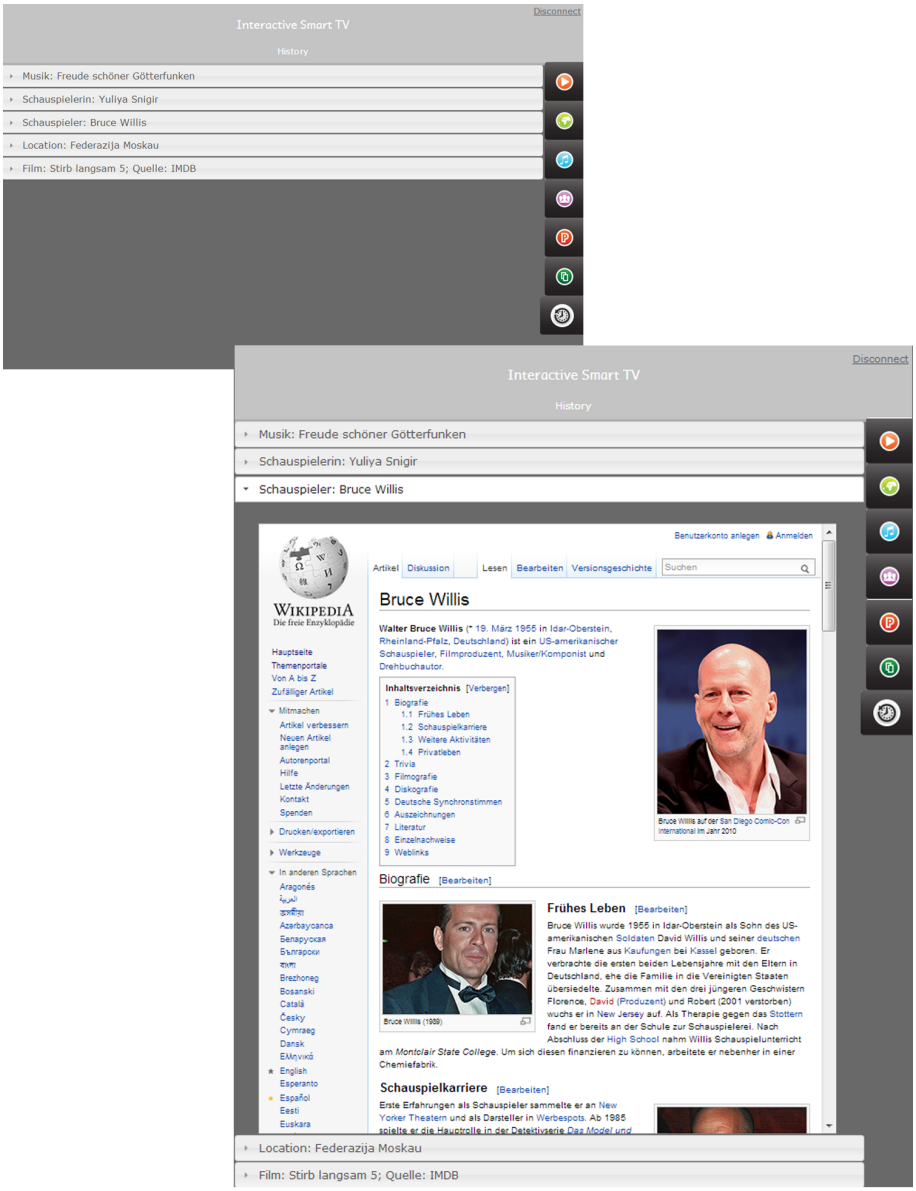

Abbildung 5.8: Der ausgewählte Verlauf-Reiter, mit eingeklappten *Accordion*-Elementen und einer ausgeklappten Metainformation.

Zur Eingabe der Daten wurde das Inhaltselement „Metainformation“ erstellt, bestehend aus fünf Feldern:

`MovieID`: Hier muss eine eindeutige Identifikationsnummer für einen Film gewählt werden. Diese muss mit dem Dateinamen des Video-Files übereinstimmen, damit sie einander zugeordnet werden können. Als mögliche ID kann z. B. die Nummer des Films von der Filmdatenbank IMDb

Abbildung 5.9: Der ausgewählte Personen-Kanal, wo ausschließlich nach Metainformationen zu vorkommenden Schauspielern im Film gefiltert wird. Diese *Accoridon*-Elemente können ausgeklappt werden, um die Information anzuzeigen.

übernommen werden, da diese sehr verbreitet ist, und viele Filme bereits abgespeichert sind. Das Feld ist vom Datentyp *Integer*, was bedeutet, dass nur Ziffern eingegeben werden dürfen.

`Title:` Der Titel der Metainformation ist der Name, der in der Client-Applikation als Überschrift in den *Accordion*-Elementen angezeigt wird.

`Timestamp:` Dieses Feld erwartet als Eingabe den Zeitpunkt, wo die Metainformation im Film relevant ist, in Millisekunden.

`Link:` In dieses Feld muss die URL zu detaillierteren Informationen eingetragen werden. Die Webseite zu dieser URL wird in der Client-Applikation als *iframe* eingebunden und angezeigt.

`Category:` Um die passende Kategorie zu wählen, wird bei diesem Feld ein Drop-down-Menü angezeigt. Hier kann von den Kategorien *location*, *music*, *actor*, *product* und *other* eine ausgewählt werden, die die Metainformation am besten beschreibt.

Aus diesen Feldern wurde mit dem *View*-Modul von Drupal, eine Ansicht erstellt, die alle Metainformationen tabellarisch anzeigt. Als Sortierkriterium wurde das *Timestamp*-Feld gewählt, damit die Zusatzinformationen in der richtigen Reihenfolge abgespeichert werden. Außerdem werden für die Ansicht alle Informationen mit der selben ID zusammengefasst. Diese Gruppierung wurde mit dem *Contextual Filter* von Drupal bewerkstelligt.

Die Standard-*View* wurde abgewandelt. Es wurde ermöglicht, dass jede

Gruppe an Metainformationen, sprich alle, einem Film zugehörigen Informationen, als XML-File exportiert werden können. Dazu wurde ein Hyperlink zu einem *Views-Data-Export*-Modul erzeugt, der die jeweilige `MovieID` beinhaltet.

```
<?php print $caption . $title?>
 -<a href="metainformation/<?php echo $movieID ?>/exportXML">
    Export XML for <?php echo $movieID ?>
  </a>
<?php ; ?>
```

Wenn dieser vom Anwender geklickt wird, öffnet sich ein Speicherdialog, der ihm das jeweilige XML-File zum Download anbietet. Der Name dieses Files setzt sich aus dem Wort „metainformation“ und der richtigen Identifikationsnummer des Films zusammen: `metainformation_12345.xml`. Zuvor wurden die gewünschten Dateien vom *Views-Data-Export*-Modul, dem XML-Schema aus Programm 4.1 entsprechend, abgespeichert. Somit erhält der Benutzer die gewünschten Metainformationen, im richtigen Format, ohne das XML-Schema beachten zu müssen, und die Informationen sind automatisch nach *Timestamp* aufsteigend sortiert.

Ergebnis

Nachdem die Webseite des Backends für die Metainformations-Eingabe geöffnet wurde, kann der Benutzer ohne Einloggen entweder selbst Zusatzinformationen zu einem Film abspeichern oder vorhandene XML-Dateien exportieren. Möchte der Anwender neue Informationen anlegen, kann er dies über das Formular, das in Abb. 5.10 abgebildet ist. Alle fünf Felder sind verpflichtend auszufüllen und selbsterklärend gestaltet. Wurde die neue Metainformation gespeichert, wird ihm der neu erstellte Eintrag angezeigt. Der Benutzer kann auch die aktualisierte Tabelle aller Zusatzinformationen ansehen, die in Abb. 5.11 gezeigt wird. In dieser Abbildung ist zu erkennen, dass die Einträge mit gleichen `MovieIDs` in Gruppen zusammengefasst wurden. Die Informationen selbst sind aufsteigend nach dem *Timestamp* sortiert. Neben jeder Überschrift einer Gruppe, steht auch ein Link zum Exportieren der Daten zur Verfügung.

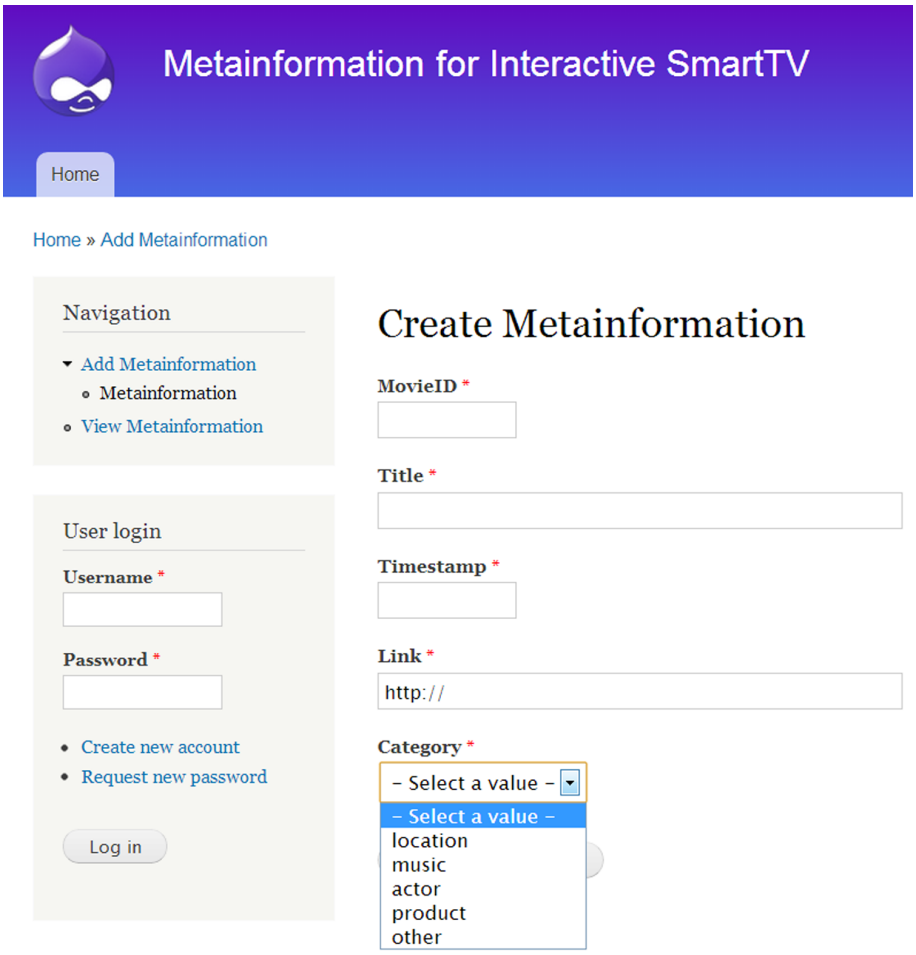

Abbildung 5.10: Das Formular zur Eingabe einer Metainformation im dafür vorgesehenen Backend. Die Felder *MovieID*, *Title*, *Timestamp*, *Link* und *Category* sind verpflichtend auszufüllen.

Metainformation

MovieID: 1323594 - Export XML for 1323594

MovieID	Title	Timestamp	Link	Category
1323594	Titel: Despicable me	1000	http://www.imdb.com/title/tt1323594/	other

MovieID: 1606378 - Export XML for 1606378

MovieID	Title	Timestamp	Link	Category
1606378	Film: Stirb langsam 5; Quelle: IMDB	2000	http://www.imdb.de/title/tt1606378/	other
1606378	Location: Federazija Moskau	19000	http://goo.gl/maps/3hgq8	location
1606378	Schauspieler: Bruce Willis	26000	http://de.wikipedia.org/wiki/Bruce_Willis	actor
1606378	Schauspielerin: Yuliya Snigir	34000	http://en.wikipedia.org/wiki/Yuliya_Snigir	actor
1606378	Musik: Freude schöner Götterfunken	40000	http://www.golyr.de/ludwig-van-beethoven/songtext-freude-schoener-goetterfunken-89358.html	music

Abbildung 5.11: Die Tabelle aller Metainformationen im Backend. Die Informationen sind in Gruppen zusammengefasst und aufsteigend nach dem *Timestamp* sortiert. Über den Hyperlink neben dem Gruppennamen, können die Informationen der Gruppe als XML-File exportiert und gespeichert werden.

Kapitel 6

Evaluierung

Dieses Kapitel dient der Evaluierung der vorliegenden Arbeit. Zu Beginn werden die in Abschnitt 4.1 gestellten Anforderungen berücksichtigt. Darauf folgt die Gegenüberstellung des *Interactive Smart TV* Systems den in Kapitel 3 vorgestellten Systemen. Abschließend werden zusätzliche Erweiterungsmöglichkeiten thematisiert.

6.1 Evaluierung anhand der Anforderungen

Im Abschnitt 4.1 wurden als Entwurf einige Anforderungen an das Projekt gestellt. Diese gilt es nun zu überprüfen und abzugleichen, welche davon erfüllt wurden. Die ersten sieben Punkte stellen funktionale Anforderungen dar, wohingegen die Punkte 8 und 9 nichtfunktionale Anforderungen sind. Um die letzten zwei Punkte auszuwerten, wurde keine Benutzerstudie durchgeführt. Allerdings wurden sie als einsichtlich und nicht komplex eingestuft.

1. **Zusatzinformationen:** Diese Anforderung behandelt die Lieferung von Zusatzinformationen zu einer laufenden Sendung. Dieser Punkt wurde erfüllt, da passende Hyperlinks zu einem Film ausgelesen und dem Zuseher angezeigt werden können. Allerdings besteht das Problem, dass manche Webseiten verhindern in fremde *iframes* eingebunden werden zu können.
2. **Timestamp basierend:** Die Forderung, dass die Zusatzinformationen zu einem bestimmten Zeitpunkt erhalten werden, ist ebenfalls erfüllt. Die in der Zusatzinformationsdatei enthaltenen *Timestamps* geben an, zu welcher Zeit eine Information von Bedeutung ist, und diese wird genau dann dem Anwender zur Verfügung gestellt.
3. **Extra Bildschirm:** Um das Einbüßen des Fernsehbildes zu umgehen, werden die Informationen auf ein separates Device, einen *Second Screen* ausgelagert und dort angezeigt. Das Fernsehbild bleibt daher zur Gänze sichtbar.

4. **Kategorien:** Der nächste Punkt behandelt die Kategorisierung von Informationen. Im System wurde eine Unterscheidung von fünf Kategorien implementiert. Diese sind *Ortsangaben*, *Musik*, *Personen*, *Produktplatzierungen* und *Allgemeines*. Der Benutzer hat dadurch die Möglichkeit, sich auf eine bestimmte Kategorie zu konzentrieren und er erhält einen besseren Überblick, wenn er nach einer bestimmten Information sucht. Außerdem wurde ein Kanal zur Verfügung gestellt, der alle Informationen, unabhängig deren Kategorie, anzeigt. Zusätzlich dazu gibt es auch noch einen Kanal, in dem alle Zusatzinformationen in einen Verlauf gespeichert werden.
5. **Plattformunabhängig:** Bezüglich Plattformunabhängigkeit muss zwischen der TV-Applikation und der Client-Applikation unterschieden werden. Die TV-Applikation wurde für ein Samsung Smart TV Gerät aus der Modellreihe aus 2012 konzipiert. Geräte aus den Jahren davor unterstützen die Applikation nicht. Da die Applikation selbst nur auf dem Emulator getestet werden konnte, ist das Verhalten auf den verschiedenen Fernsehgeräten unbekannt. Was die Client-Applikation betrifft, ist sie auf jedem internetfähigen Endgerät ausführbar, weil die Applikation als Webseite realisiert wurde. Da der Fokus der Arbeit nicht auf der Darstellung der Client-Applikation lag, wurde das Design vorwiegend dem *Samsung Galaxy Tab 2 10.1* angepasst.
6. **Offen für Community:** Damit eine Informationsverwaltung durch eine Community funktioniert, wurden die Metainformationen gekapselt abgespeichert. Die Dateien befinden sich auf einem externen Server, auf den die TV-Applikation zugreifen kann. Entsprechen die Dateien dem richtigen XML-Schema, kann die Community die eigenen Files mit Metainformationen zur Verfügung stellen. Die TV-Applikation ist zwar darauf ausgelegt, dass sie mehrere Dateien mit Metainformationen öffnen und verarbeiten kann, allerdings wurde ein anderer Lösungsweg gewählt. Der gewählte Lösungsweg benötigt nur ein File mit Informationen. Wie der folgenden Punkt zeigt, wurde ein Backend erstellt, das die Eingabe von Metainformationen zulässt.
7. **Backend:** Um die Community zu unterstützen und das Einhalten des XML-Schemas zu gewährleisten, wurde ein Backend erstellt. Die Community hat dadurch eine zentrale Anlaufstelle, wo ein vorgegebenes Formular ausgefüllt werden muss. Der Benutzer füllt die Felder für die Identifikationsnummer des Films, den Metainformationstitel, den *Timestamp* und den weiterführenden Hyperlink aus. Abschließend wählt er eine passende Kategorie für die Information. Mit dem Absenden des Formulars werden die Informationen in die richtige Form gebracht, dem zugehörigen Film zugeordnet und dem *Timestamp* entsprechend eingeordnet. Im Anschluss können alle einem bestimmten Film zugeordneten Informationen gemeinsam exportiert werden. Zum aktuellen Zeitpunkt

passiert dies, indem die resultierende Datei als Download angeboten wird. Effektiver ist es, diese Datei sofort auf den Metadaten-Server zu speichern. Dadurch wird die bestehende Metainformations-Datei am Server upgedatet und die Informationen liegen in der richtigen Reihenfolge vor. Um die Benutzerfreundlichkeit zu steigern, kann die Eingabe des *Timestamps* überarbeitet werden. Die derzeitige Eingabe erwartet den Wert in Millisekunden. Besser ist die Angabe der relevanten Zeit in Stunden, Minuten und Sekunden.

8. **Unkomplizierte Konfiguration:** Um dem Benutzer eine einfache Konfiguration des Verbindungsaufbaus von einem externen Gerät zu einem Smart TV zu ermöglichen, wurden folgende Dinge berücksichtigt: Die vom Anwender geforderten Eingaben beschränken sich auf das Wesentlichste. Wie Abb. 5.5 entnommen werden kann, gibt es drei Textfelder, von denen zwei bereits vorausgefüllt sind. Der Benutzer muss lediglich die IP-Adresse des Fernsehers und den richtigen Port eingeben. Die vorausgefüllten Felder für die „Device ID“ und die „App ID“ *können* geändert werden, *müssen* aber nicht.
9. **Intuitives Interface:** Diese Anforderung betrifft ein ansprechendes und intuitives Interface der Client-Applikation. Die Navigation des Systems besteht aus verschiedenen Symbolen, die die Kategorien und die Kanäle für Datenstrom und Verlauf symbolisieren. Wie in Abb. 5.6 ersichtlich ist, wird der Datenstromkanal durch das von Videoplayern bekannte *Play* Symbol dargestellt. Es wurde gewählt, um auszudrücken, dass in diesem Kanal alle Informationen von selbst angezeigt werden. Orstangaben werden von einer Weltkugel repräsentiert, Musik durch ein Notensymbol, Personen durch eine schematische Darstellung dreier Personen, Produktplatzierungen durch das offizielle Icon dafür, Allgemeines durch zwei Rechtecke (die oft für „Sonstiges“ verwendet werden) und der Verlauf durch eine Uhr, die von einem Pfeil entgegen dem Uhrzeigersinn umgeben ist. Obwohl diese Symbole für sich selbst stehen, wurde die Lernkurve des Anwenders berücksichtigt und zusätzlich die Bedeutung des Symbols angezeigt. Der Name der Kategorie wird im Kopfbereich angezeigt, wie in Abb. 5.9 zu sehen ist (vgl. „Actors“).

   Überdies wurde eine schlichte, dunkle Gestaltung gewählt, mit Farbakzenten durch bunte Symbole. Das Userinterface war dabei kein zentraler Punkt der Arbeit, sondern stellte nur einen Teilaspekt dar.

Tabelle 6.1 veranschaulicht den Vergleich des Systems mit den zuvor gestellten Anforderungen noch einmal.

6.2 Evaluierung anhand von „State of the Art“

In Kapitel 3 wurden verschiedene Systeme thematisiert, die es ebenfalls erlauben Zusatzinformationen zu dem im Fernsehen Gesehenen zu erhalten.

Tabelle 6.1: Vergleich der Applikation *Interactive Smart TV* mit den gestellten Anforderungen aus Abschnitt 4.1.

Anforderung	*Interactive Smart TV*
1. Zusatzinformationen	✓
2. Timestamp basierend	✓
3. Extra Bildschirm	✓
4. Kategorien	✓
5. Plattformunabhängig	✓
6. Offen für Community	✓
7. Backend	✓
8. Unkomplizierte Konfiguration	✓
9. Intuitives Interface	✓

In diesem Abschnitt werden diese Systeme mit dem *Interactive Smart TV* System verglichen.

Alle Systeme, bis auf *XBMC Media Center*, bieten die Zusatzinformationen auch *Timestamp* basierend an. Bei *Google Info Cards* ist es allerdings so, dass das laufende Video zuerst pausiert werden muss, bevor die Zusatzinformationen ersichtlich sind.

Bei dem Kriterium, ob die Informationen parallel zur Sendung angeboten werden, ist folgendes zu unterscheiden: Werden die Informationen *ausschließlich* parallel zur Sendung angeboten, oder sind sie auch eigenständig verfügbar? Die Applikationen zu *Blu-ray Second Screen* stellen die Informationen großteils auch ohne laufende Sendung zur Verfügung. Bei den Applikationen zu *TV Second Screen* variiert diese Funktion je nach Anbieter. *Interactive Smart TV* stellt die Zusatzinformationen erst ab dem Zeitpunkt, in dem diese im Film relevant sind, bereit. Allerdings kann der Benutzer nach dem Erhalt der Informationen über die Kategorien und den Verlauf darauf zugreifen, solange die Client-Applikation nicht geschlossen wird. *XMBC Media Center* bietet die Zusatzinformationen nur außerhalb des Films an.

Die Erweiterung um eigene Zusatzinformationen funktioniert bei den wenigsten Systemen. Neben *Interactive Smart TV* bieten nur *XBMC*, *MPEG-7* und *XRM* die Möglichkeit zur Erweiterung der Daten an. Im Gegensatz zu *Interactive Smart TV* können die Daten allerdings nur für den Eigengebrauch verändert werden. Die Idee einer Community, die die Zusatzinformationen gemeinsam erstellt und für jeden zugänglich macht, verfolgt ausschließlich das System *Interactive Smart TV*. Außerdem wird zu *Interactive Smart TV* auch ein Backend angeboten, das die Vereinheitlichung der Datenstruktur übernimmt. Bei den übrigen Systemen *TV Second Screen*, *Blu-ray*

Tabelle 6.2: Überblick des eigenen Systems.

Funktion	*Interactive Smart TV*
Zusatzinformationen	✓
Timestamp basierend	✓
Parallel zur Sendung	✓
Selbst erweiterbar	✓
Info nicht im Bild	✓
App	✓
Web	✓
Geräteunabhängig	✓
Proprietär	✗

Second Screen, *Google Info Cards* und *Multimedia Home Platform* ist der Benutzer jeweils vom Anbieter der Informationen abhängig. Der Anwender kann die Zusatzinformationen lediglich konsumieren.

Ein wichtiges Kriterium ist, ob die Informationen auf einem eigenen Gerät angezeigt werden. Erfüllt wird es nur von den *TV Second Screen* und *Blu-ray Second Screen* Applikationen sowie dem *Interactive Smart TV* System. Diese Funktion ist als klarer Vorteil zu sehen, da das Fernsehbild dadurch nicht eingeschränkt wird.

Ob die Zusatzinformationsdienste als Applikation oder als Webdienst angeboten werden, ist eine Geschmacksfrage. Die Vorteile eines Webdienstes sind einerseits, dass sie nicht installiert werden müssen und andererseits geräteunabhängiger sind. *Interactive Smart TV* wurde als Webdienst entwickelt, um weitläufiger einsetzbar zu sein. Ein Vorteil einer Applikation ist beispielsweise die Möglichkeit, die Verbindung zu einem Fernseher automatisch herzustellen (vgl. *Blu-ray Second Screen* Applikationen).

Ein Pluspunkt des *Interactive Smart TV* Systems ist die Möglichkeit der Kategorisierung der Informationen. Der Benutzer kann sich dadurch auf eine für ihn interessante Kategorie fokussieren und sich besser orientieren. Dieser Aspekt wurde auch beim *eXtensible Rich Media* System berücksichtigt.

In Tabelle 6.2 ist die Auswertung des Systems *Interactive Smart TV* anhand der selben Kriterien wie in Abschnitt 3.9 ersichtlich.

Zusammenfassend ist zu sagen, dass das entwickelte *Interactive Smart TV* System vorallem durch den Gedanken einer Community im Hintergrund hervorsticht. Auch das verfügbare Backend zur Informationseingabe und die Möglichkeit der Kategorisierung sind neue Ansätze.

6.3 Erweiterungsmöglichkeiten

Da das *Interactive Smart TV* System ein Prototyp ist, gibt es noch einige Erweiterungsmöglichkeiten. Im Folgenden werden angedachte Verbesserungen erläutert:

Integrierte Fernsteuerung: Eine integrierte Fernsteuerung am externen Device wird als sinnvoll erachtet. Möchte sich der Benutzer beispielsweise ungestört auf eine eingetroffene Zusatzinformation konzentrieren, ist eine Pause-Funktion praktikabel. So verpasst er nichts von der weiterlaufenden Sendung und es muss dafür kein weiteres Gerät als Fernbedienung verwendet werden. Zusätzliche Befehle, wie Szenenauswahl oder Lautstärkeregelung sind ebenfalls erweiterbar.

Informationen aus dem Internet: Um die Community zu unterstützen, sollen Informationen automatisch aus dem Internet bezogen werden können. Allgemeine Angaben, die nicht auf eine bestimmte Szene bezogen sind sondern den ganzen Film betreffen, können aus diversen Online-Film-Datenbanken entnommen werden (z. B. IMDb). Beispiele dafür sind Regisseur, Länge des Films, kurzer Plot, verbrauchtes Budget, etc. So ist ein gewisser Grundstock an Informationen, der auch von *Electronic Program Guides* geboten wird, gegeben.

Responsive Design: Die Client-Applikation wurde nicht in erster Linie *responsive* gestaltet. Das bedeutet, dass sich das Design nicht allen Displaygrößen perfekt anpassen kann. Vorallem die Darstellung auf kleinen Devices, wie etwa Smartphones, wird nicht ideal angezeigt. Die Gestaltung der Applikation kann deshalb dahingehend verbessert werden.

Verbindungspartner finden: Damit bei dem Verbindungsaufbau auf die Eingabe der IP-Adresse des Fernsehers verzichtet werden kann, kann diese automatisch ausgelesen werden. Sobald sich ein Client-Device und ein Smart TV Gerät mit der laufenden Applikation im selben Netzwerk befinden, kann die Verbindung von selbst hergestellt werden.

Informationen selber schreiben: Es gilt zu überdenken, ob einfache Hyperlinks zu anderen Webseiten langfristig sinnvoll erscheinen. Das System kann dahingehend verändert werden, dass eigene Texte als Zusatzinformationen am Empfangsgerät angezeigt werden. Bei Hyperlinks besteht die Gefahr, dass die zugehörige Seite nicht mehr zur Verfügung steht. Außerdem enthalten viele Webseiten Werbung, die den Benutzer stören kann. Beim eigenen Verfassen von Zusatzinformationen kann auf Werbung und irrelevante Bestandteile einer Webseite verzichtet werden.

Kapitel 7

Schlussbemerkungen

Im Rahmen der vorliegenden Arbeit wurde ein System zur Lieferung von Zusatzinformationen parallel zu einer laufenden Sendung entwickelt und vorgestellt. Der Anwender der Applikation erhält gleichzeitig zu einem laufenden Film verschiedene Zusatzinformationen auf einem externen Gerät angezeigt. Dabei ist zu erwähnen, dass die Informationen genau in dem Moment erscheinen, zu dem sie im Film relevant sind. Ein Beispiel für Zusatzinformationen ist ein Artikel zu einem Schauspieler, der gerade am Fernseher zu sehen ist. Ein weiteres Beispiel wäre der Name des eben zu hörenden Soundtracks und mögliche Kaufoptionen. Diese Informationen sind in verschiedene Kategorien unterteilt, um einen besseren Überblick zu gewähren. Außerdem hat der Benutzer so die Möglichkeit den Fokus auf einen bestimmten Aspekt zu legen. Eine offene Community erweitert die Metainformationen, die für jeden zugänglich sind. Das System *Interactive Smart TV* besteht dabei aus drei essenziellen Komponenten:

1. Samsung Smart TV Applikation,
2. Client-Applikation und
3. Backend zur Metainformationseingabe.

Die TV-Applikation übernimmt die Hauptaufgaben des Systems. Dazu gehört das Auslesen der zur Verfügung stehenden Filme und Metainformationen. Diese Daten werden synchronisiert, damit die Zusatzinformationen zum richtigen Zeitpunkt an ein verbundenes Empfangsgerät gesendet werden können.

Die Client-Applikation wird auf einem externen Device ausgeführt und ist dafür zuständig, die von der TV-Applikation gesendeten Informationen zu empfangen. Im Anschluss werden sie aufbereitet und dem Anwender ansprechend dargeboten. Durch ein Kategorie-System können die Informationen strukturiert verwaltet werden.

Das zugehörige Backend erleichtert die Eingabe der Metainformationen für eine Community. Durch das Ausfüllen eines Formulars werden die Informationen im richtigen XML-Schema gespeichert. Überdies gruppiert das

Backend zusammengehörende Informationen und zeigt diese sortiert an.

Ein wesentlicher Punkt ist das Darstellen der Zusatzinformationen auf einem eigenen Gerät, um das Fernsehbild nicht einzuschränken. Die Informationen sind näher beim Anwender und er kann zusätzlich selbst nach weiteren Informationen suchen. Durch das offene System *Interactive Smart TV* und die Einbindung einer Community entstehen neue Möglichkeiten. Informationen werden von Anwendern für Anwender erzeugt und geteilt. Das Angebot an Metainformationen kann so „von selbst“ wachsen, ohne dass ein Filmproduzent dahinter steht. Das System findet großen Zuspruch und wird als praktisch und zukunftsorientiert angesehen.

Allerdings steckt *Interactive Smart TV* noch in den Kinderschuhen. Es handelt sich dabei nur um einen Prototypen. Das System muss durch eine Benutzerstudie ausführlich getestet und dahingehend angepasst werden. Es gibt einige Erweiterungsmöglichkeiten, wie das Integrieren einer Fernsteuerung oder das automatische Beziehen von Informationen von Online-Datenbanken.

Das System selbst hat aber durchaus Potenzial. Der Ansatz des *Second Screens* erfährt zum Zeitpunkt einen großen Aufschwung. Einige Filmproduzenten bietet zu bestimmten Blu-ray Filmen bereits eigene Applikationen, um Zusatzinformationen zu gewähren. Auch einzelne Fersehsender bieten immer mehr Details zum ausgestrahlten Fernsehprogramm. Ein neues Anwendungsgebiet für *Second Screen* Applikationen bietet auch der Spielemarkt. Der Ansatz einer Community zur Erstellung von Metainformationen wird zukünftig sicher auch eine bedeutende Rolle beim Fernsehkonsum einnehmen. Somit kann das Fernsehen in vielerlei Hinsicht noch interaktiver gestaltet werden.

Anhang A

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 Masterarbeit

Pfad: /

Zlabinger_Bernadette_2013.pdf Masterarbeit (Gesamtdokument)

A.2 Online-Quellen

Pfad: /Literatur

/DLNA/*.pdf	Kopien der Literatur und Online-Quellen zum Thema DLNA
/MHP_GEM/*.pdf	Kopien der Literatur und Online-Quellen zum Thema MHP und GEM
/MPEG-7/*.pdf	Kopien der Literatur und Online-Quellen zum Thema MPEG-7
/SamsungSmartTV/*.pdf	Kopien der Literatur und Online-Quellen zum Thema Samsung Smart TV
/SecondScreen/*.pdf	Kopien der Literatur und Online-Quellen zum Thema Second Screen
/SmartTVAPIs/*.pdf	Kopien der Literatur und Online-Quellen zum Thema Smart TV APIs
/UPnP/*.pdf	Kopien der Literatur und Online-Quellen zum Thema UPnP
/XBMC/*.pdf	Kopien der Literatur und Online-Quellen zum Thema XBMC Media Center

A.3 Abbildungen

Pfad: /Abbildungen

*.pdf	Vektorgrafiken
*.png	Screenshots und Grafiken
*.jpg	Screenshots und Bilder

A.4 Projektdateien

Pfad: /Projekt

Installation.pdf	Installationsanleitung für das gesamte System
InteractiveSmartTVApp.zip	TV-Applikation für Samsung Smart TV
Client_InteractiveSmartTV.zip	Client-Applikation für externes Gerät
drupal.zip	Backend zur Eingabe der Metainformationen

Quellenverzeichnis

Literatur

[1] Jörg Broszeit. *IPTV und Interaktives Fernsehen – Grundlagen, Marktübersicht, Nutzerakzeptanz*. Saarbrücken: VDM Publishing, 2007.

[2] M.R. Cabrer u. a. „Controlling the smart home from TV“. In: *Proc. International Conference on Consumer Electronics*. (Las Vegas, Nevada). Institute of Electrical und Electronics Engineers (IEEE), Jan. 2006, S. 255–256.

[3] M.R. Cabrer u. a. „Controlling the smart home from TV“. In: *IEEE Transactions on Consumer Electronics* 52.2 (Mai 2006), S. 421–429.

[4] A.G. Foina und J. Ramirez-Fernandez. „How a cell phone can change dramatically the way we watch tv“. In: *IEEE EUROCON 2009 International Conference Devoted to the 150th Anniversary of Alexander Popov*. (St Petersburg, Russia). Institute of Electrical und Electronics Engineers (IEEE), Mai 2009, S. 1265–1271.

[5] Anna Gruber. „Kombination von Metainformationen und Videodaten“. Magisterarb. Hagenberg, Austria: Digitale Medien; FH Oberösterreich – Fakultät für Informatik, Kommunikation und Medien, 2009.

[6] Uwe Kühhirt und Marco Rittermann. *Interaktive audiovisuelle Medien*. München: Carl Hanser Verlag, 2007.

[7] B. Mrazovac u. a. „Smart Audio/Video Playback Control Based on Presence Detection and User Localization in Home Environment“. In: *2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC 2011)*. (Bratislava, Slovakia). Institute of Electrical und Electronics Engineers (IEEE), Sep. 2011, S. 44–53.

[8] Nicholas Negroponte. *Total digital – die Welt zwischen 0 und 1 oder die Zukunft der Kommunikation*. München, Deutschland: Wilhelm Goldmann Verlag, Jan. 1997.

[9] Frank Puscher. „Mit dem Zweiten sieht man besser“. In: *c't Magazin für Comupter Technik* 26 (Dezember 2012), S. 74–76.

Online-Quellen

[10] URL: https://secure.flickr.com/photos/35889705@N04/8245238768/ (besucht am 08.06.2013).

[11] URL: http://werbeplanung.at/news/agenturen/2013/02/cellular-bringt-ski-wm-auf-mobile-geraete (besucht am 08.06.2013).

[12] URL: http://2.bp.blogspot.com/-odfAvlz9tig/UVM0P85U_aI/AAAAAAAAAA8/P_rBf2txYrY/s1600/Jack+Black.jpeg (besucht am 08.06.2013).

[13] URL: https://www.java.net//blog/jfarcand/archive/Presentation2.jpg (besucht am 08.06.2013).

[14] *AllShare API Reference*. URL: http://www.samsungdforum.com/Guide/ref00005/index.html (besucht am 09.04.2013).

[15] Anywab GmbH. *Second Screen One – Die Macht des zweiten Bildschirms*. Feb. 2013. URL: http://anywab.com/wp-content/uploads/2013/02/PM-_Second_One_final1.pdf (besucht am 08.06.2013).

[16] Anywab GmbH. *Second Screen Zero – Die Macht des zweiten Bildschirms*. Juli 2012. URL: http://anywab.com/wp-content/uploads/2012/06/Final_PM1-Studie-Second-Screen-Zero.pdf (besucht am 08.06.2013).

[17] DVB Project Office. *Multimedia Home Platform – Open Middleware for Interactive TV*. Mai 2011. URL: http://www.dvb.org/technology/fact_sheets/DVB-MHP_Factsheet.pdf (besucht am 22.04.2013).

[18] DasErste.de. *Schade, Tatort+ ist zu Ende!* 2012. URL: http://www.daserste.de/unterhaltung/krimi/tatort/specials/tatort-plus-100.html (besucht am 08.06.2013).

[19] *DeviceAPI Guide for Samsung Smart TV*. URL: http://freethetv2011.s3.amazonaws.com/DeviceAPI_Guide%5BV2.20%5D%5B1%5D.pdf (besucht am 09.04.2013).

[20] DifferenceBetween.net. *Difference Between UPnP and DLNA in Digital Home*. URL: http://www.differencebetween.net/technology/difference-between-upnp-and-dlna-in-digital-home/ (besucht am 13.04.2013).

[21] Digital fernsehen. *Middleware – die gescheiterten Versuche in Deutschland*. Dez. 2008. URL: http://www.digitalfernsehen.de/news_697264.html (besucht am 22.04.2013).

[22] IBM Research. *VideoAnnEx Annotation Tool – User Manual*. URL: http://www.research.ibm.com/VideoAnnEx/usermanual.html (besucht am 25.04.2013).

[23] IBM Research. *VideoAnnEx Annotation Tool*. URL: http://www.research.ibm.com/VideoAnnEx/ (besucht am 25.04.2013).

[24] Richard Lawler. *Spider-Man Blu-ray comes with a Second Screen app for iPad and Sony's Android tablets*. Okt. 2012. URL: http://www.engadget.com/2012/10/03/spider-man-blu-ray-second-screen-app/ (besucht am 09.06.2013).

[25] Martin Rycak. *Wikipedia-Zugriffszahlen bestätigen Second-Screen-Trend*. Okt. 2012. URL: http://www.martinrycak.de/wikipedia-zugriffszahlen-bestatigen-second-screen-trend/ (besucht am 08.06.2013).

[26] *Samsung Web API: AllShare API*. URL: http://img-developer.samsung.com/onlinedocs/WebAPI_beta/index_allshare.html (besucht am 11.04.2013).

[27] Samsung. *Application Development Guide for Samsung Smart TV*. URL: http://ebookbrowse.com/2011-sdk2-0-app-development-guide-for-samsung-smart-tv-v1-10-pdf-d396017176 (besucht am 13.01.2013).

[28] Samsung. *Messaging System*. URL: http://www.samsungdforum.com/Guide/art00028/index.html (besucht am 11.04.2013).

[29] SamsungForum.com. *TVKeyValue Object*. URL: http://www.samsungdforum.com/upload_files/files/guide/data/html/html_2/api_reference/javascript_apis/common_modules/common_module_tvkeyvalue_object.html (besucht am 28.05.2013).

[30] SamsungForum.com. *Widget Object*. URL: http://www.samsungdforum.com/upload_files/files/guide/data/html/html_2/api_reference/javascript_apis/common_modules/common_module_widget_object.html (besucht am 28.05.2013).

[31] SamsungForum.com. *deviceapis*. URL: http://www.samsungdforum.com/upload_files/files/guide/data/html/html_3/api_reference/javascript_apis/web_device_api/web_common_device_api/web_device_common_api_deviceapis_module.html (besucht am 28.05.2013).

[32] Ben Serridge. *Get inside your favorite movies with Google Play*. März 2013. URL: http://officialandroid.blogspot.co.at/2013/03/get-inside-your-favorite-movies-with.html (besucht am 10.06.2013).

[33] UPnP Forum. *About UPnP Forum*. URL: http://upnp.org/about/what-is-upnp/ (besucht am 08.04.2013).

[34] UPnP Forum. *UPnP Certified Technology – Your Simple Solution for Home, Office and Small Business interoperability*. Sep. 2010. URL: http://upnp.org/resources/whitepapers/UPnPWhitePaper_2010.pdf.

[35] UPnP Forum. *UPnP Device Architecture 1.1*. Okt. 2008. URL: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf (besucht am 03.05.2013).

[36] XBMC.org. *About XBMC*. URL: http://xbmc.org/about/ (besucht am 01.05.2013).

[37] derStandard.at. *Aus für ORF OK: RTR kritisiert Zeitpunkt und Informationspolitik*. URL: http://derstandard.at/1304553038058/Digitale-Teletext-Aus-fuer-ORF-OK-RTR-kritisiert-Zeitpunkt-und-Informationspolitik (besucht am 13.06.2013).

[38] dlna. *DLNA for HD Video Streaming in Home Networking Environments*. URL: http://www.dlna.org/docs/white-papers/dlna-for-hd-video-streaming-in-home-networking-environments.pdf (besucht am 12.04.2013).

[39] futurezone.at. *Second Screen: ORF-Zusatzinfos auf Tablets*. Nov. 2012. URL: http://futurezone.at/digitallife/12650-second-screen-orf-zusatzinfos-auf-tablets.php (besucht am 08.06.2013).

[40] news.orf.at. *Adrenalinschub für Handy und Tablet*. Feb. 2013. URL: http://news.orf.at/stories/2164288/2164296/ (besucht am 08.06.2013).

[41] news.orf.at. *Disney Second Screen*. 2011. URL: http://disneysecondscreen.go.com/bambi/help.php (besucht am 09.06.2013).

[42] nielsen. *The Cross-Platform Report*. Nov. 2012. URL: http://www.nielsen.com/us/en/reports/2012/state-of-the-media--cross-platform-report-q2-2012.html (besucht am 08.06.2013).

[43] rtl.de. *Mit der RTL INSIDE App wirst du zum Insider!* Sep. 2012. URL: http://www.rtl.de/cms/mein-rtl/rtl-mobil/rtl-inside-app.html (besucht am 08.06.2013).