FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Port of JMF to JME-CDC

João Manuel Martins Guerreiro e Silva

Integrated Master in Computer Engineering

Master in Informatics and Computing Engineering

Supervisor: Eurico Carrapatoso (Title)

3th March, 2009

Port of JMF to JME-CDC

João Manuel Martins Guerreiro e Silva

Integrated Master in Computer Engineering
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

31st July, 2008

Abstract

In an attempt to fill the need for the growing demand for entertainment systems, especially IPTV systems. The company Sun Microsystems has created a development platform for such systems. The Sun Java Media Client was born, a platform developed in the Java programming language, aimed at the IPTV market.

The Sun Java Media Client (SJMC) was developed using Java Micro Edition, Connected Device Configuration together with the Java Media Framework. This solution enables the control of media content and graphics using the Java programming language.

The project developed at the site of Sun Microsystems in Dublin, Ireland, had as main goal the development of JMF over JME-CDC. The goal of this project was the development of 3 features that would launch the SJMC to a commercial market level, competing with the most used IPTV solutions, as TiVO or AppleTV. The 3 features are implemented:

Create a module to add the support for videos with more than one audio track, enabling the detection of the audio tracks available and select the desired one, identical to the format used in DVDs

The second feature is the support for Video On Demand (VOD). The target for this phase of the project was to create support for Real Time Streaming Protocol. Implementing the logic of negotiation required to request and play the videos. This feature should also allow some advanced features such as pause or rewind.

Finally, the third feature is the implementation of features of rapid reproduction (Trick Play) of video files stored locally. This feature should be implemented both for videos in Mpeg2 and Mpeg4-H264.

This report describes the specification of the project as well as the details of implementation. It also has a section dedicated to testing and results as well as a brief summary work performed and conclusions.

Resumo

Na tentativa de satisfazer a crescente procura de sistemas de entretenimento, em especial sistemas de IPTV. A empresa Sun Microsystems viu-se na necessidade de criar uma plataforma de desenvolvimento para este tipo de sistemas. Nascendo assim o Sun Java Media Client, uma plataforma desenvolvida na linguagem de programação Java, dirigida primariamente para o mercado do IPTV.

O Sun Java Media Client (SJMC) foi desenvolvido recorrendo a Java Micro Edition, configuração Connected Device Configuration. E apresenta como novidade a integração de Java Media Framework numa plataforma de Java Mobile. Permitindo assim o controlo de conteúdos multimédia recorrendo a linguagem de programação Java.

O projecto desenvolvido nas instalações da Sun Microsystems em Dublin, Irlanda, tem como principal objectivo continuar o desenvolvimento do JMF sobre JME-CDC. O pretendido neste projecto era o desenvolvimento de 3 funcionalidades que iriam lançar o SJMC para um patamar de mercado mais comercial, concorrendo com as soluções mais usadas, como TIVO ou AppleTV. As 3 funcionalidades implementadas são:

Criação de um modulo de suporte para vídeos com mais do que uma faixa de áudio, permitindo detectar as faixas de áudio disponíveis e seleccionar a pretendida, idêntico ao modelo usado nos DVDs.

A segunda funcionalidade prende-se com a necessidade de Video On Demand (VOD). Desta forma o pretendido para esta fase do projecto era criar suporte para Real Time Streaming Protocol. Criando assim toda a lógica de negociação necessária para pedir e reproduzir os vídeos pretendidos. Esta funcionalidade também deveria permitir algumas funcionalidades avançadas como pause ou rewind.

Por ultimo, a terceira funcionalidade é a implementação das funcionalidades de reprodução rápida (Trick Play) para ficheiros guardados localmente. Esta funcionalidade deveria ser implementada tanto para vídeos codificados em Mpeg2 como Mpeg4-H264.

No presente relatório encontra-se descrita a a especificação do projecto assim como os detalhes de implementação. Encontra-se também uma secção dedicada a testes e resultados, assim como um breve resumo do trabalho realizado e das conclusões retiradas.

Acknowledgements

I would like to express my gratitude to all that contributed directly and indirectly during this project. I would like to thank my family, friends and colleagues whose expertise, understanding and patience adding value and providing guidance into my work. In particular I would like to thank my parents, Augusta and Álvaro and my brother and sister, Francisco and Emilia who gave me their personal support and guidance.
I would like to acknowledge Professor Eurico Carrapatoso of Faculdade de Engenharia da Universidade do Porto, for his coordination, suggestions and revisions.
A very special thank you to my work colleges Steven and Ronan who envisioned the project and provided me the appropriate orientation during the entire project. Their contribution strongly influenced this project. Always insisting in good programing practices, and for letting me learn for myself.
Finally i would like to thank all Sun Microsystems Ireland Inc., which provides me knowledge, infrastructure and financial support to complete this journey.

João Guerreiro

Contents

List of Figures

List of Tables

Abbreviations

API	Application User Interface
AGUI	Advanced Graphics and User Interface
AWT	Abstract Window Toolkit
BD-J	BlueRay Disk Java
CDC	Connected Device Configuration
CLDC	Connection Limited Device Configuration
CVM	C Virtual Machine
FP	Foundation Profile
H264	Mpeg4 Advanced Video Codec
IPTV	Internet Protocol Television
IDE	Integrated Development Environment
JME	Java Micro Edition
JMF	Java Media Framework
JSE	Java Standard Edition
MIDP	Mobile Information Device Profile
MPEG	Moving Picture Experts Group
Mpeg2	Mpeg2 Video Codec
Mpeg2-TS	Mpeg2 Transport Stream
PAT	Program Allocation Table
PBP	Personal Basis Profile
PMT	PRogram Mapping Table
PP	Personal Profile
RAM	Random Access Memory
RTSP	Real Time Streaming Protocol
SDK	Software Development Kit
SJMC	Sun Java Media Client
SMR	Sun Media Receiver
UDP	User Datagram Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
VLC	Video Lan Client
VOD	Video On Demand

Trick Play: Is the term used to define control operations over video content. ex: fast forward and rewind
Xlet: Is an applet implementation especially developed for TV applications, its part of the JavaTV 1.1 specification and the Xlet interface implements the methods: initXlet, startXlet, pauseXlet and destroyXlet

Chapter 1

Introduction

This chapter will show the context of the project, doing a brief description of the company where it was developed, explain the main purpose of this project, the motivations that took me to embrace this project and for last a small overview of the document structure.

1.1 Context

The work was developed during the Project course, in a business environment in the company Sun Microsystems Ireland Inc. situated in the city of Dublin.
This course is the final unit of studies of Mestrado Integrado em Engenharia Informatica e Computação, at Faculdade de Engenharia da Universidade do Porto.
The project had a total duration of 20 weeks, from 14 of October 2008 to 3 of March 2009 (including the elaboration of documentation in which this report is an essencial part).

Sun Microsystems is a multinational vendor of computers, computer components, computer software and information technology services with headquarters situated in Santa Clara, California.
I integrated the company in an internship program of one year, from July 2008 to July 2009, integrating the Engineering Services - Connected Device Configuration group. The main purpose of this group is to develop the Java Micro Edition - Connected Device Configuration. In this group my work was done together with the IPTV team.

1.2 Project Description

The main goal of this project was to develop a high level and rich content set of APIs as part of the current process of porting the JMF framework to the JME-CDC platform. This 2 modules together form the SJMC platform witch is a product developed in Sun Microsystems to enter the growing market of IPTV.

The project work was the development of 3 features that would take the SJMC (JMF + JME-CDC) to a level that could compete with the best IPTV solutions in the market. The goal was to provide a rich content set of APIs to application developers, so that they could develop simple but rich media applications.
This 3 features are:

- **Multiple Audio Tracks** — Create a set of APIs to be able to select an audio track in a file or stream with multiple audio tracks;
- **RTSP Support** — Implement full support for Video On Demand using the Real Time Streaming Protocol;
- **Trick Play** — Add the functionality and provide the APIs to control the rate of which a video is played, Fast Forward and Rewind functionalities.

1.3 Motivation

The motivation for this project comes from the growing market os IPTV and how this kind of product can increase the quality os life of people all around the world.
It's important to notice that this market is no longer available only for the high social classes, the reality is that almost every house in Europe and US has available an IPTV service and it's starting to have some penetration in less developed areas of the world like Asia, Latin America and even in Africa there are already some IPTV solutions available. It's just a question of time and the IPTV is going to replace the analog TV as the TV replaced the radio 50 years ago.

Another reason and maybe more important in personal matters is to be able to work with the bleeding edge of technology, work with technology that has a very rich visual effect, this work has a very pleasurable visual impact. Working with technology that at the end of the day has something to show, and can captivate the interest of the common human being is probably the greater motivation that I can find.

1.4 Document Structure

This document is divided into 6 chapters, the first - Introduction - was written to provide an overview of the project and do a small description of the company where it was developed.

The second chapter provides some background technological overview needed to understand the context of the development of this project. This chapter will explain what were the tools used to develop the project and how they contributed to the development process.

The third chapter is a description of the requirement specification needed for the development of the project. It includes the Use Cases and some additional information about the architecture of the software modules developed.

The fourth chapter explains the implementation details, class diagrams and the approach to solve some of the problems found during the development process.

The fifth chapter covers the tests done during and after the development was completed, the results obtained and some conclusions based on the results obtained.

The sixth and final chapter will cover the conclusions, a brief resume of the accomplished objectives and some improvements that could be done in the future.

In the end of this document there are some appendixes that will provide additional information to better understand the project context.

Chapter 2

Background Information

This chapter will explain some os the used technologies that had relevance to the development of this project. Trying to show some of the reasons which took me to use those applications and point the strengths and weaknesses.

This chapter is divided into 3 different parts, the first will explain what is the SJMC platform, architecture, components and an overview of the target market. The second part will do a brief explanation of what the Sun Media Receiver is, doing a small description of the hardware and software present on the device. The third and final part of this chapter will focus on other technologies used for the development of this project.

2.1 Sun Java Media Client

Sun Java Media Client is a Sun Microsystems product developed to fill the upcoming need for a Java based system for the digital TV market. The SJMC platform consists in the integration of JMF framework over JME-CDC platform, and the main goal is to be able to develop a small, lightweight and performant product capable of running in low cost devices.

To be able to achieve such a goal was decided to use the Java Micro Edition, on the CDC configuration, mainly because of the almost identical set of APIs available on the JSE 1.4 but with a significant lower memory, performance and size footprint.

In addition to the JME-CDC functionalities, the SJMC platform is able to open, play, and record media files, this need lead us to add the JMF platform to the package. The main problem was that the JMF framework was developed to the Java Standard Edition,

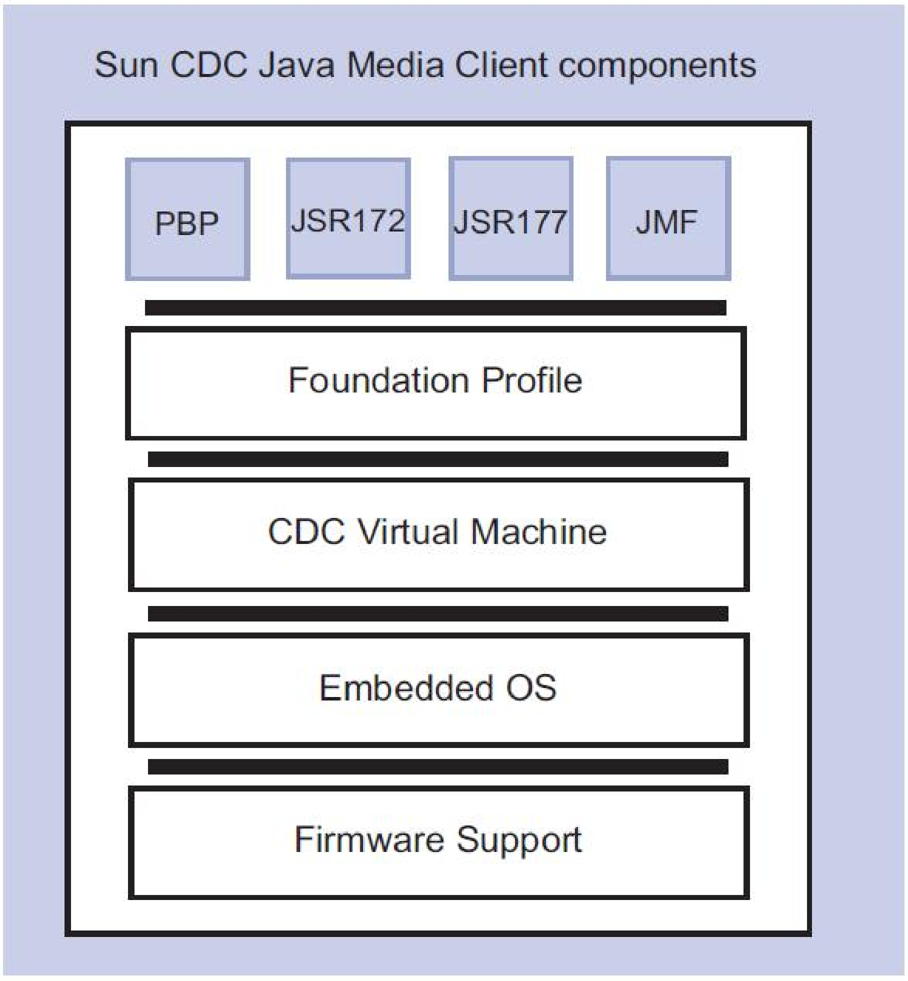


Figure 2.1: SJMC Platform Architecture

and it was intended to run os desktop PCs with a much higher performance and memory than the usual Set-Top box configuration.
The Sun Java Media client allows you to develop applications either as normal Java application or using the Xlet applet format. The Xlet is some how identical to the MIDlets used in the JME-CLDC configuration, but with a more oriented approach to the IPTV applications, in fact the Xlet is defined as a standard in the JavaTV 1.1 specification.

2.1.1 Java Micro Edition - Connected Device Configuration

The Java Micro Edition is a specification of a subset of Java platform aimed at providing a set of Java APIs for the development of software aimed at the limited resource devices. Target devices go from 1KB of memory devices as smart cards, to high end entertainment systems. Some of the most common usage of the JME are: cellphones, PDAs, Set-Top boxes and BlueRay players. [Mic09b, tfeW09c]

To be able to run on all kinds of devices the JME was developed in a modular architecture. The JME has a 2 different sets of configurations each one with multiple profiles available.

The two configurations available - CDC and CLDC - separate the mobile devices in two main groups, the devices with less hardware performance and memory (especially memory) usually run the CLDC configuration while the devices with better performance figures run the CDC configuration.

The CLDC configuration implements the minimum amount of Java class libraries to be able to run a Java Virtual Machine and the main target of CLDC are the small and more hardware restrained devices like cellphones, SmartCards or low end PDAs.
The CDC configuration is a subset of the Java Standard Edition and is aimed at the more capable devices like Set-Top boxes, high end cellphones (like the Nokia N95), high end PDAs and BlueRay players. This configuration is divided into 3 main profiles:

- **Foundation profile** — This profile implements the full JME Virtual Machine and includes the entire Java Platform, Standard Edition APIs.

- **Personal Basis Profile** — This profile extends the Foundation Profile to include a lightweight GUI support, in the form of AWT subset. This is the platform used in the BD-J (The java system present on all BlueRay players).

- **Personal Profile** — This profile extends the Personal Basis Profile including a more complete AWT subset. (Including AGUI: a lightweight version of the Swing GUI present on JSE).

Keeping in mind that most of the devices running the SJMC have over 64 MB of ram, the CDC configuration running the FP and PBP profiles was the platform used for its development. The almost identical APIs to between CDC and JSE 1.4.2 helped to reinforce the decision.

2.1.2 Java Media Framework

The Java Media Framework APIs enables audio, video and other time-based media to be added to Java applications and Xlets built on Java technology. This framework which

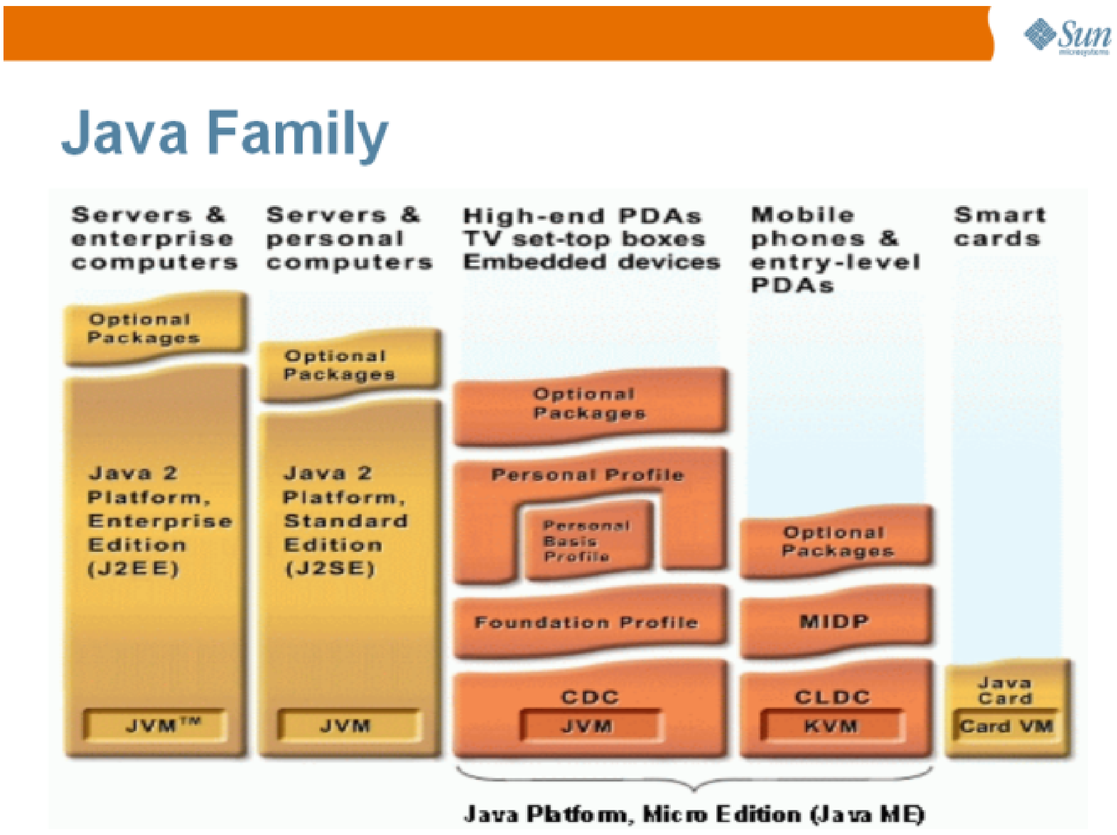

Sun microsystems
Java Family
Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
Optional Packages
Optional Packages
Optional Packages
Personal Profile
Personal Basis Profile
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Foundation Profile
MIDP
CDC
CLDC
Java Card
JVM™
JVM
JVM
KVM
Card VM
Java Platform, Micro Edition (Java ME)

Figure 2.2: Java Micro Edition Architecture [Blo09]

can capture, playback, stream, and transcode multiple media formats, was originally an extension of the Java 2 Platform, Standard Edition. [Mic09a, tfeW09b]

Porting it to the Java Micro Edition brought a number of performance related problems, so the way to go was to strip out some of the transcoding and decoding functionalities, letting only the features needed for the purpose of the SJMC.
Because the SJMC project was developed with the IPTV market in mind, for the moment the only video codecs supported are the ones used as standard in the IPTV definition: Mpeg2-Video used in most cases for Standard Definition videos, the Mpeg4-H264 regularly used to encode High Definition videos (up to 1920 X 1080) and Mpeg2-Transport Stream as the encapsulation protocol.
The decision to use the Mpeg2-TS as the encapsulation protocol, was taken because of its wide use in the network and satellite streaming, BlueRay disks, HD DVD and the support for error correction. This protocol was originally designed to be used as a transmission protocol for unreliable transmissions, like satellite transmissions or radio broadcast.

2.1.2.1 Current Implementation

The current implementation of the Java Media Framework in the SJMC already allows to do some basic player functionalities:

- **Play a file** — Support for playing a video file at a normal rate;
- **Play from a UDP Stream** — Support for playing a basic Mpeg2-TS stream using the UDP protocol;
- **Graphics Overlay** — Supports the overlaying of graphics over the video window. This allows to create applications that mix graphics with Video.

2.1.2.2 Architecture

In the Figure 2.3 can be seen a scheme of the architecture of the JMF platform.
The platform id divided into 3 main layers: The API layer, the Player with the multiple modules and player plugins and the Parser, responsible for extracting the video and audio tracks and do some control over the streams read.

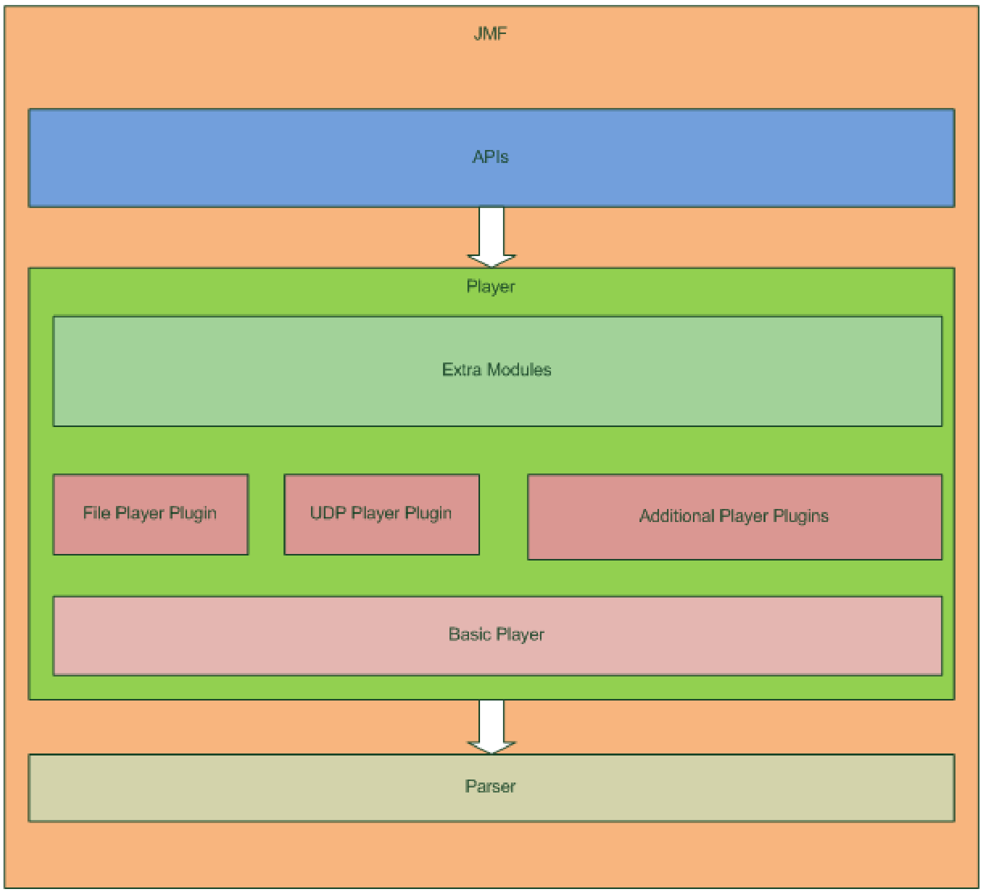


Figure 2.3: Java Media Framework Architecture

Table 2.1: SMR Hardware Specifications

	SMR SD	SMR HD
Processor	300 MHz MIPS	300 MHz MIPS
Memory	128 MB	256 MB
SD Output	Yes	Yes
HD Output	No	Yes
Hard Drive	40 Gb	80 Gb
Video Decoder	2 Sigma BlueRay Decoders	2 Sigma BlueRay Decoders

2.2 Sun Media Receiver

The Sun Media Receiver -Figure 2.4 is a Set-Top box developed by the SunLabs team working for Sun Microsystems, it is a non commercial device aimed at testing and demonstration of the SJMC features and capabilities. The main goal of the development of the SMR was to provide a low performance platform that would help to detect and improve any performance related issue present on SJMC.

The SMR comes in two slightly different versions, a Standard Definition only and a High Definition version. The hardware specifications of each of the versions can be found in the Table 2.1.

The standard Operating System in the SMR is a Linux distribution called BusyBox. Busybox is a well known linux distribution for embedded systems, that has as the main features a small memory footprint, good performance, small size and comes with most of the tools needed for the maintenance of embedded devices.

Figure 2.4: Sun Media Receiver

2.3 Other Technologies

This section will cover some software and technologies that were used for the development and testing during this work. It is divided into 2 different subsections, the first subsection will do a brief exposition of the software used for the development process including Integrated Development Environment of choice and the emulator used. The second subsection will do a small reference to the software and tools that helped on the test process and obtaining the results.

2.3.1 Development

In the following lines, a brief reference will be done covering the tools used to the development of the project, and the reasons that took me to use them.

2.3.1.1 SJMC Emulator

The SJMC Emulator is the emulator for the SJMC platform, runs on X86 desktop PCs over Linux (currently only Fedora and Ubunto Linux distributions are supported), It has all the Foundation Profile and Personal Basis Profile functionalities, and the majority of the features of the Java Media Framework. The main difference between the SJMC Emulator and the version running on the SMR is related with the video decoding process, the SMR has two built in hardware video decoders, while the emulator uses the FFmpeg as a software decoder.
The Figure 2.5 shows the modules used in the SJMC Emulator and provides a overview of how they are connected.

2.3.1.2 FFmpeg

FFmpeg is an open source tool used for decoding and transcoding of video and audio codecs, and can encode either media files or streams.
The tool can be used directly by command line, or through its own set of APIs, in this case it is being used as a software video renderer to integrate with the SJMC Emulator, as you can see in Figure 2.5. [tea09, tfeW09a]

2.3.1.3 Netbeans 6.5

The NetBeans IDE is an open-source integrated development environment written entirely in Java using the NetBeans Platform. NetBeans IDE supports development of all Java application types (Java SE, web, EJB and mobile applications) out of the box. Among other features are an Ant-based project system, version control and re-factoring.

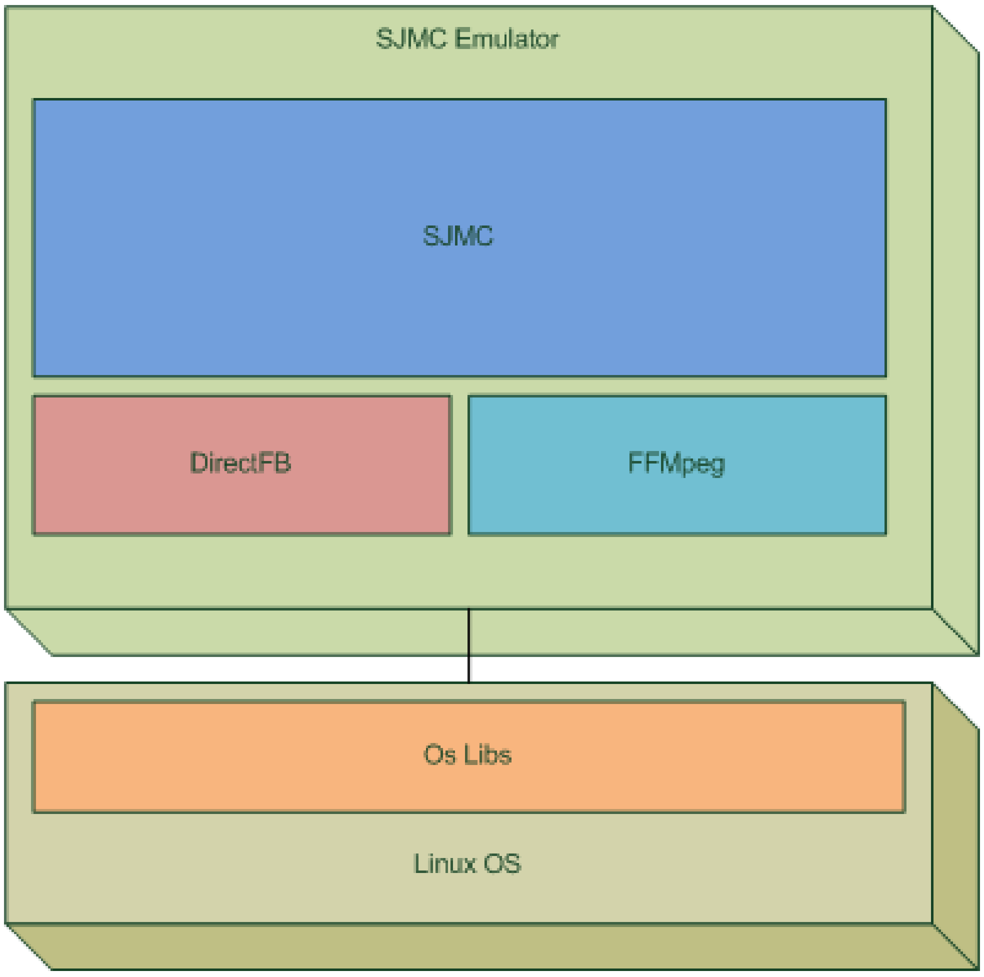


Figure 2.5: SJMC Emulator Architecture

[Mic09c, tfeW09e]

Modularity: All the functions of the IDE are provided by modules. Each module provides a well defined function, such as support for the Java language, editing, or support for the CVS versioning system, and SVN. NetBeans contains all the modules needed for Java development in a single download, allowing the user to start working immediately. Modules also allow NetBeans to be extended. New features, such as support for other programming languages, can be added by installing additional modules. For instance, Sun Studio, Sun Java Studio Enterprise, and Sun Java Studio Creator from Sun Microsystems are all based on the NetBeans IDE.

And for the first time in the history of the Netbeans, the version 6.5 has the SJMC emulator build in, so you can create a SJMC project in 5 clicks, and run it in less tens 2 minutes. The default SJMC application generated by the NetBeans 6.5 is a Xlet that draws a rectangle in the screen.

2.3.2 Test

This subsection will cover the tools used during the testing process along and after the development of the project.

2.3.2.1 Sun Streaming System - StreamStar

The Sun Streaming System (Stramstar) is a professional streaming system developed by Sun Microsystems, aimed at the IPTV market. It has a full implementation of the Real Time Streaming Protocol for Video On Demand systems, and consists of a multiple server configuration, providing high performance memory-based streaming and redundancy storage,
The system supports both Mpeg2 and Mpeg4-AVC(H264) video codecs, multiple audio tracks in a single stream, Standard and High definition videos, server-side Trick Play up to 6 different speeds and it can output up to 160 000 Standard Definition streams at the same time.[Mic09d]
There are 3 configurations possible for the Sun Streaming Systems:

- **StreamStar** — The full implementation of the system, uses one Sun Fire X4950 server and up to 32 Sun Fire X4100 blade servers, with up to 2TB of memory for

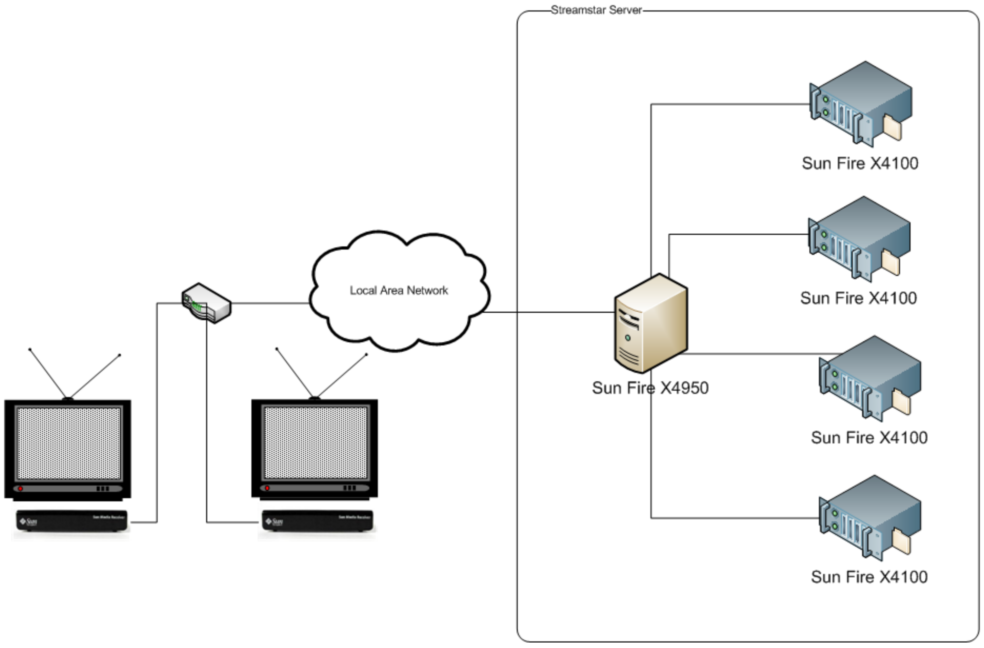


Figure 2.6: StreamStar Network Architecture

video caching and up to 320GB/s of network output. This kind of server has as it main target the cable TV companies wanting to provide to their customers a VOD system, in fact 40.000 of this systems would be enough to stream a different stream to each TV in the planet.

- **NanoStar** — Uses the same system (hardware and software) but in a more affordable package, providing the intermediate solution aimed at groups of apartments (up do 5.000 subscribers) who want a streaming system for Video On Demand.

- **PicoStar** — The smallest solution of the group, uses the same software as the other 2 but running on a single Sun Fire X4100 server with 2 AMD Operon X64 processors, 4GB of memory and up to 10GB/s of streaming power, its main target are single blocks of apartments (up to 100 subscribers). It is the option used for demonstrations of the technology and testing.

Since October 2008 a PicoStar system has been installed to be used as the RTSP server for testing purposes.
The Real Time Streaming Protocol works by using a HTTP-like text based protocol running over TCP/IP. All the negotiation of the streaming needs to be done using this protocol, and once the negotiation has finished with success, the server opens a UDP Port

directly to the destiny host IP (which could be a multicast address so that multiple client can share the same stream) and starts streaming the video content.

The great innovation introduced by the RTSP was the Trick Play functionality on the server side, so its possible to have a small and low performance Set-Top box that can do trick play without any additional effort, because the server will decode an re-encode the video into the speed the client wants, so if a request to set the rate of reproduction of a stream to go 4 times faster, what the server will do is send the content of the video 4 times faster.
The RTSP provides a Pause functionality that works the same way, a request for pause is sent to the server and the server will continue to send data with a Paused image. This approach puts all the processing power in the server side allowing the final client to have a high quality system with a low budget device.

2.3.2.2 VLC Media Player

The VLC Media Player is a multi-platform Open Source video and audio player capable of playing a number of different file formats including all the major audio and video codecs. It can capture network streams, it has support for a small set of the RTSP protocol (Can play RTSP videos but can't do Trick Play), and it can re-multiplex and re-encode files and streams in a number of different formats and codecs. [Pro09, tfeW09f]

In this project VLC Media Player was mainly used to multiplex Mpeg2-TS files with multiple sound tracks and for testing the streaming functionalities, using VLC as the streaming server.

2.3.3 TIVO and AppleTV

TIVO [Inc09b] and AppleTV[Inc09a] are 2 of the more commonly used IPTV solutions, they provide a hardware and software package for media viewing and recording, the main features of this platforms are Video On Demand (VOD), Private Video Recording (PVR) and the capability of mixing video and graphics. Compared to this kind of products, the SJMC provides the ability to develop applications in the Java programing language, using a public set of APIs and providing the same set of functionalities (apart from the PVR which is currently on the future work plan).
The AppleTV and TIVO are both using private APIs and they are not device independent, they can only run on a specific hardware provided by the vendor.

Chapter 3

Requirements Specification and Architecture

The general objectives for this project have been defined since the beginning, but there is still a need to describe in more detail the requirements and the specifications defined for the software that is under development.
This chapter describes the features implemented and how they will interact with the end users.

3.1 High Level Requirements

The first step in the requirement survey is meant to provide a global look over the entire system, so this first approach is to summarize the use cases and the requirements that apply to all the three features implemented.

3.1.1 Use Cases

The system has 2 actors: the Developer who wants to be able to code rich multimedia applications for IPTV using a powerful set of APIs and a small amount of effort, and the User who wants to take advantage over those applications, and to be able to have control over what he is watching and listening.

The Developer main use case is to be able to write multipurpose applications for TV using Java and having access to a advanced features just by calling one method. He wants to build applications that can at the same time play a video file or from a RTSP stream without any change to his application. He should be able to write an application that enables the same functionalities for videos stored on files or videos available over RTSP

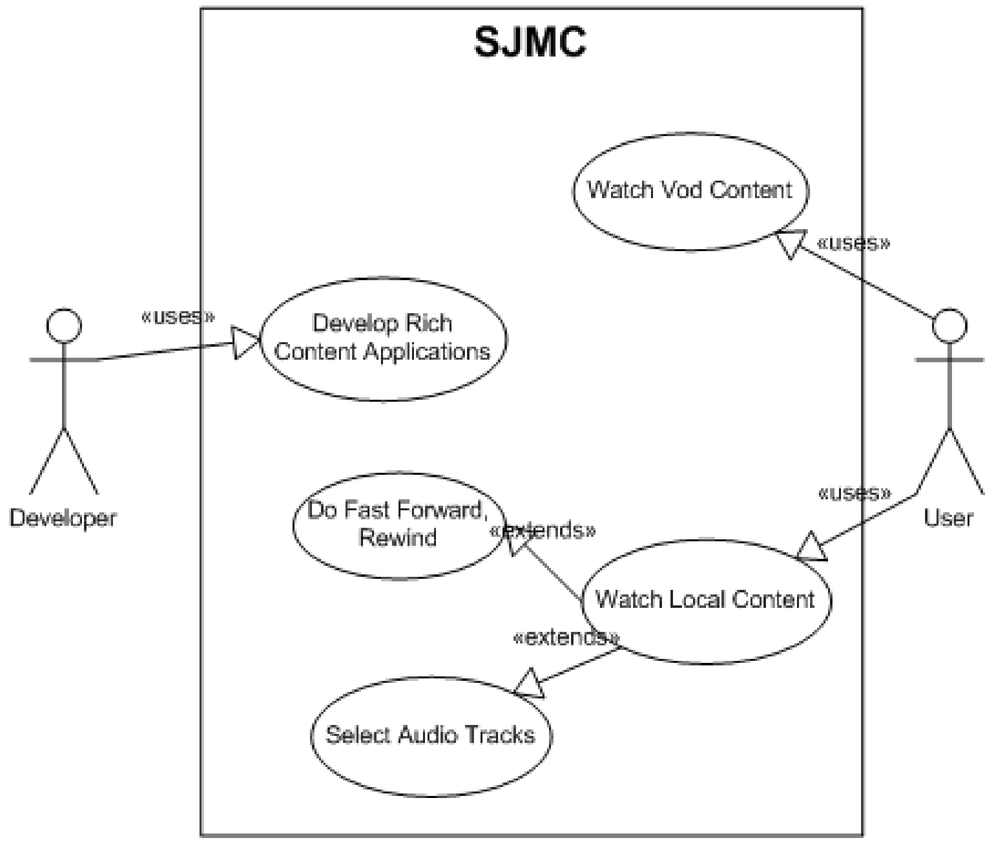


Figure 3.1: Use Cases of High Level Requirements

streams without having separate code for each case.

The User should be able to watch video content stored in files the same way he watches VOD content, this process should be transparent. Without having any knowledge regarding the source of the video. Have the same kind of functionalities for both VOD and local stored videos, and by functionalities we say Fast Forward, Rewind and select a different language or background music to the one that better suits his mood at the moment.

3.1.2 Requirements

In this section only the requirements that are common to the three features will be covered.

3.1.2.1 Platform Independent

One of the requirements for this project is the need to be able to run in a platform independent way, the implementation needs to be portable to new target devices without any extra effort, as soon as the device target is running a JME-CDC Virtual Machine, all the features implemented should run out of the box.

3.1.2.2 Reliability

The end system should be reliable, it should be able to offer the same performance after days of use. Should handle common errors, network errors and file errors. In a word the system needs to be reliable.

3.1.2.3 Performance

The system is intended to run on low cost and low performance systems, so it should have the performance we can find on a normal TV, changing channels in less then 300 ms, and providing short response times from the user requests.

3.2 Multiple Audio Tracks

The objective of this feature is to integrate the functionality of multiple audio detection and selection into the JMF framework, accessible through a small set of APIs. The final functionality should be similar to the one provided by DVD players where the user can select the audio track with the language he pretends to listen. In fact the DVD model, based on the Mpeg2-Program Stream is very similar to the system pretended.

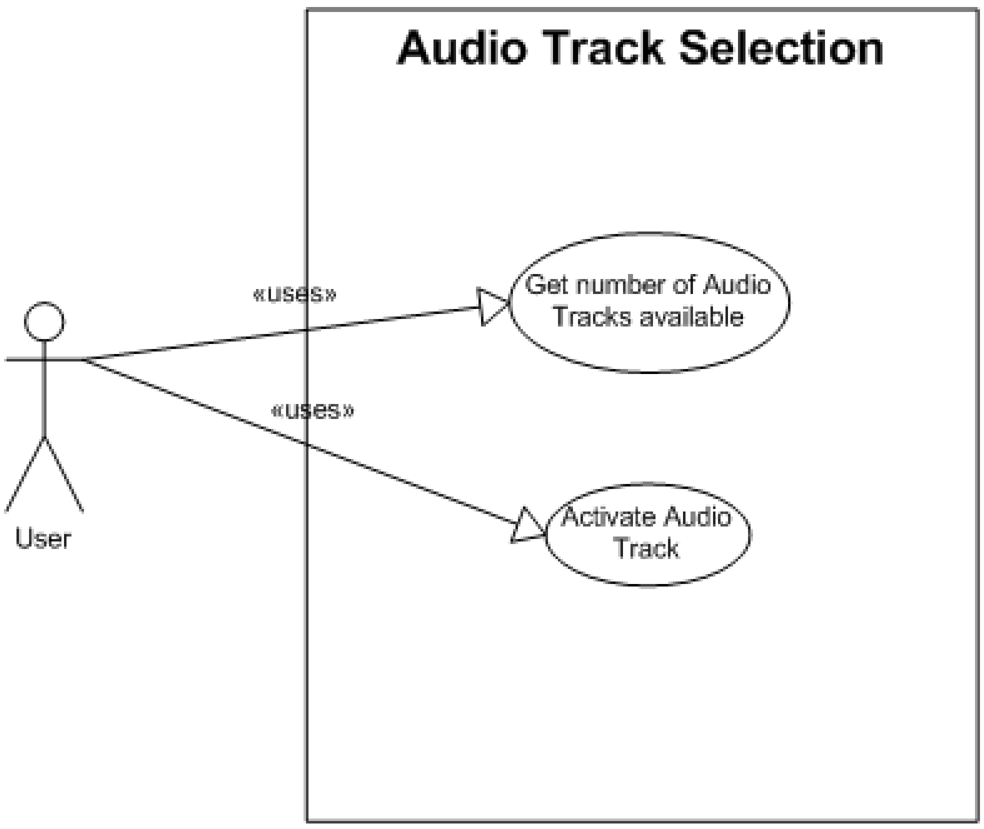


Figure 3.2: Use Cases for the Multiple Audio Tracks

In the next sections the use cases and requirements defined for this specific feature will be extensibly covered.

3.2.1 Use Cases

In the Figure 3.2 are showed the set of Use cases for this feature, a small description of each one can be found in the following lines:

- **UC1: Get Number of Audio Tracks** — Gives the information of the number of audio tracks available in the video.

- **UC2: Activate an Audio Track** — Allows the user to select and activate a specific Audio Track.

3.2.2 Requirements

In this section the requirements defined for the Multiple Audio Tracks selection and activation are defined. The way the APIs should behave and the kind of performance we should expect for this feature.

3.2.2.1 API

The APIs to get and activate an audio track should be accessible from the Player object and should have the following names and properties:

- **Player.getActiveAudioTrack()** — This call should return the PID value identifying the audio track that is playing at the moment.
- **Player.getAudioTrackList()** — This call should return an array with the PID list for all the different audio tracks that are available in the video.
- **Player.setAudioTrack(int)** — This method call should be change the audio track that is playing at the moment to the one identified by the PID sent as a parameter.

3.2.2.2 Performance

This feature should not affect at any time the reproduction of the video, it should change between audio tracks without any significant performance lost, during the process or after the audio track change is finished.

3.3 Real Time Streaming Protocol Support

As explained in previous sections of this document, the RTSP is a text based HTTP-like protocol especially developed for the streaming of media content, like videos and audio. But it can be used to stream any kind of data. In this specific implementation, the goal is to enable the JMF framework to negotiate video content with a RTSP server.

In the following sections can be found a a full description of the use cases and requirements for this specific feature.

3.3.1 Use Cases

In the Figure 3.3 are showed the set of Use cases for the RTSP feature, a small description of each one can be found in the following lines:

- **UC3: Play** — Allows the user to start a RTSP negotiation and play the video content of the stream.
- **UC4: Change Rate** — Allows the user to change the rate at which the video is being played.
- **UC5: Stop** — Stops the video stream.
- **UC6: Pause** — Stops the video but maintaining the current image on screen.
- **UC7: Rewind** — Plays the video in rates below zero.
- **UC8: Fast Forward** — Plays the video with rates over 1X.
- **UC9: Slow Motion** — Plays the video in slow motion, rates between 0 and 1.

3.3.2 Requirements

In the following lines the description of the requirements needed for this feature are presented.
This requirement include, performance requirements, protocol implementation, the format of the APIs to develop and the way to access the APIs.
The url format to request the RTSP assets is also a requirement.

3.3.2.1 URL Format

The URL format for RTSP assets should be in the following format: **rtsp://**server**:**port**/**asset

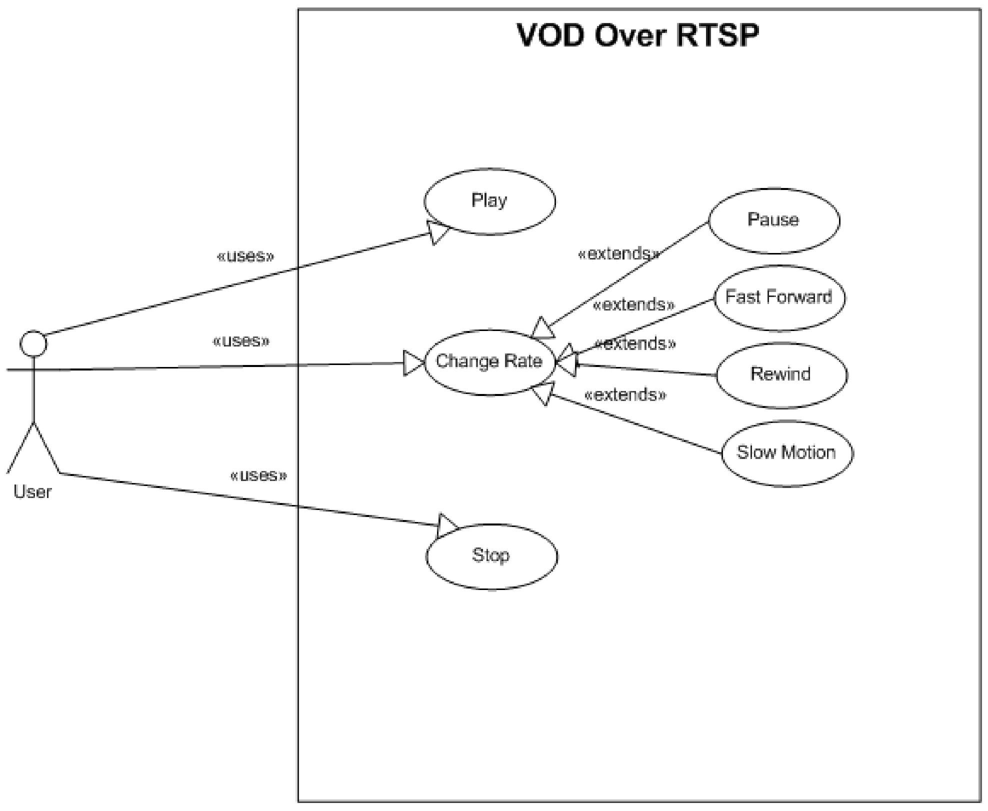


Figure 3.3: Use Cases for the RTSP

- **server** — The IP or Hostname for the server.
- **port** — The port number where the server accepts requests.
- **asset** — The name of the stream which the request is going to be made.

Example of a RTSP URL: `rtsp://plasma.ireland.sun.com:8553/blackbox`

3.3.2.2 Protocol

The RTSP protocol should be fully implemented, all the following commands should be implemented: [HS09]

- **DESCRIBE** — This command is used to request all the services available in the RTSP server.
- **SETUP** — This command is used to request a SessionID and to request the server to prepare the stream.
- **PLAY** — The PLAY command orders the server to start streaming the video. This command can be used to request rates diferrent that 1X. It's the command to use when fast forward or rewind are requested.
- **STOP** — This command requests the server to stop sending the stream.
- **PAUSE** — This command will request the RTSP server to pause the video in a determined moment. The server won't stop the stream, it will just send a stream with a fixed image.
- **TEARDOWN** — This command orders the server to drop all connections with the client, stop sending the stream and discard the client SessionID.

3.3.2.3 API

The implementation of the RTSP feature should use the same APIs used for the file player. The requirement is only for the more relevant APIs.

- **new Player(RtspUrl)** — This method should create a new RTSP player.
- **Player.start()** — This method should start the negotiation with the RTSP server and start playing the stream.
- **Player.stop()** — This method sends a stop request to the RTSP server and stops the player.

- **Player.setRate(int)** — This method is used for Trick Play functionalities as well as for pausing when the rate is set to 0.
- **Player.getRate()** — This method should return the numerical value of the rate in which the video is being played.
- **Player.getMediaTime()** — This method returns the time in seconds of the video.
- **Player.setMediaTime(int)** — This will send a setup command to the RTSP server to request that the video starts at a specific time.
- **Player.getDuration()** — This method will return the total duration of the stream.

3.3.2.4 Trick Play

The Trick Play over RTSP should be fully implemented, including fast forward, rewind, slow motion and pause.

3.3.2.5 Keep Alive

Most of the RTSP Servers need some kind of communication to be sure that the client is still alive, so a Keep Alive mechanism is mandatory.

3.3.2.6 Connection Drop

The system should be able to recover from connection drops.

3.3.2.7 Server Messages

The system should handle server messages like End Of Stream.

3.4 Trick Play

Trick Play is a term commonly used to describe de control of the rate at which the video is being displayed. In this feature the objective is to develop a set of APIs and functionalities to enable fast forward and rewind capabilities into the JMF Framework.

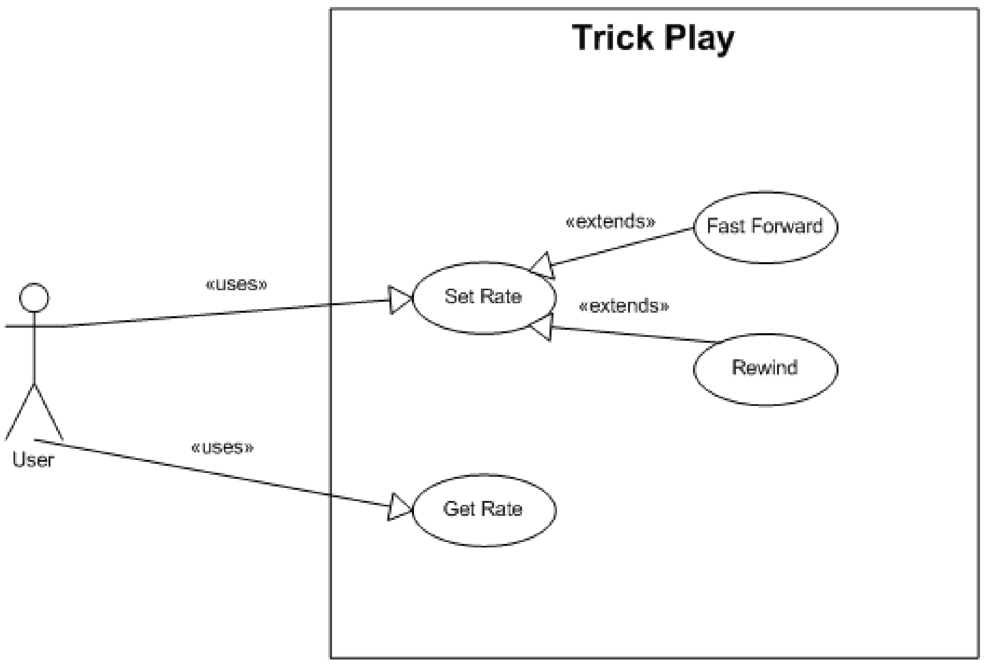


Figure 3.4: Trick Play Use Cases

In the following sections can be found the full description of the use cases and requirements defined for this specific feature.

3.4.1 Use Cases

In the Figure 3.2 are showed the Use Cases defined for this feature, a small description of each one can be found in the following lines:

- **UC10: Set Rate** — Changes the rate at which the video is being played.
- **UC11: Get Rate** — Returns the information of the rate at which the video is being played.
- **UC12: Rewind** — Sets the rate to values lower than 0.
- **UC13: Fast Forward** — Sets the rate to values above 1 up to 60X.

3.4.2 Requirements

This section will cover the requirements for the APIs to develop, the minimum and maximum rates required for the reproduction of video. Also covered in this section are the requirements for the codecs and the quality of reproduction when rates are different of 1X.

3.4.2.1 API

The trick play implementation should implement at least the following methods:

- **Player.setRate(int)** — This method is used for setting the play rate to a different value.
- **Player.getRate()** — This method should return the numerical value of the rate at which the video is being played.

3.4.2.2 Rates From 2X to 60X

The system should be able to do reproduce video from 2X to 60X.

3.4.2.3 No Sound

When the rate is different of 1X the video should be played without sound.

3.4.2.4 Codecs

Trick play should be implemented for the Mpeg2-Video and Mpeg4-H264 codecs.

3.5 Architecture

This section will describe the architecture of the final implementation. First an overview of the physical architecture, showing the software and hardware modules of the system and their relation. The Horizontal architecture is described on the second subsection, showing the modules of the system and the layers that represent the final product, showing the relation between modules and their hierarchy.

3.5.1 Physical Architecture

The Figure 2.6 can be used to show the hardware needed for the system to run, and the relations between the multiple modules.

The system can work on a standalone mode, if its playing files from a local Hard Drive or can have a client/server architecture if the system is playing VOD assets (RTSP).

If the system is playing VOD assets through RTSP, then it will use 2 hardware modules, the SMR running the SJMC platform or the SJMC Emulator in the client side and a RTSP server in the server side.

The SJMC platform (either running on the SMR or the Emulator) uses the Ethernet to connect to the RTSP server. A wireless connection could be used but in a large scale implementation could be compromised because of the high bandwidth required to stream high definition videos.

3.5.2 Horizontal Architecture

The Horizontal architecture shows the different layers of the system. Explaining the relation between each other.

As shown in Figure 3.5, the features developed are distributed by several layers in the JMF core.

The Trick Play feature acts as a module that can be applied to any video source (File, UDP stream, RTSP stream).
The RTSP Feature acts as a plugin player that extends the basic player, and implements the negotiation with the RTSP server, this plugin is independent from any other plugin already available and for the end developer is completely transparent. the system decides which plugin to use depending on the type os URL used to create the player.

The Trick Play feature is a module that can only run on videos stored in a local hard drive. This is a restriction caused by the need to seek the file in real time, something that the system can only do suing files stored in a local Hard Drive.

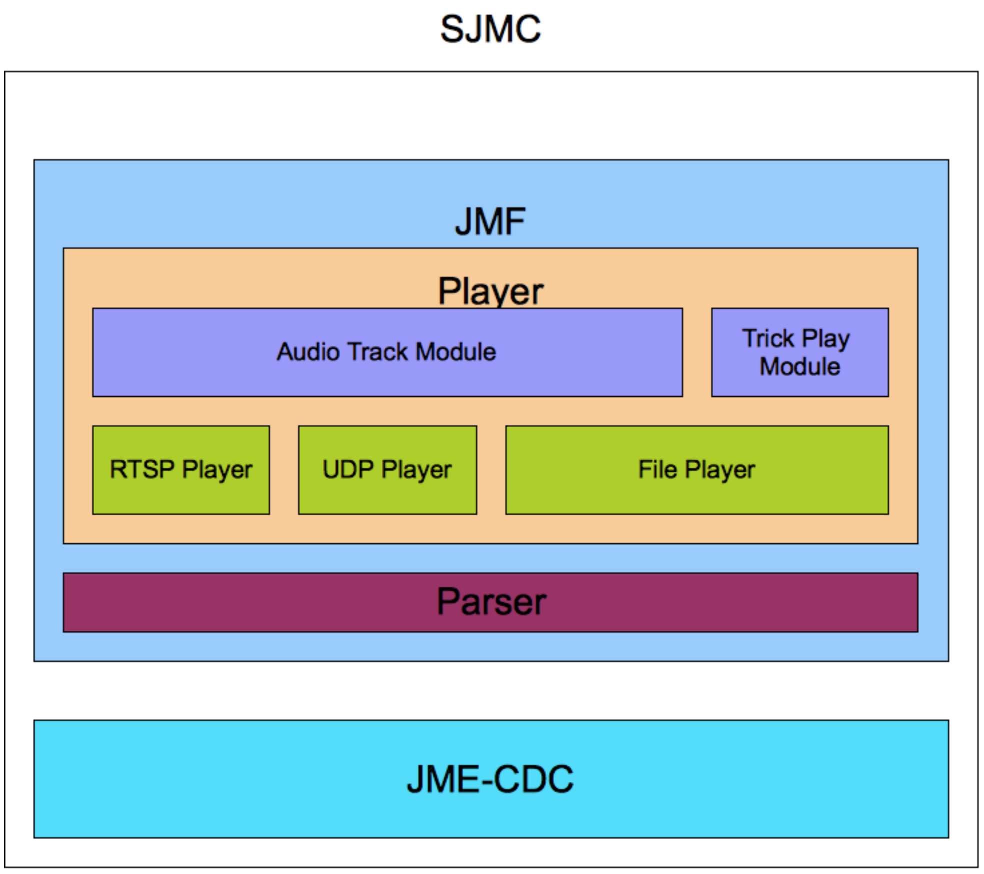


Figure 3.5: Horizontal Architecture

Chapter 4

Implementation

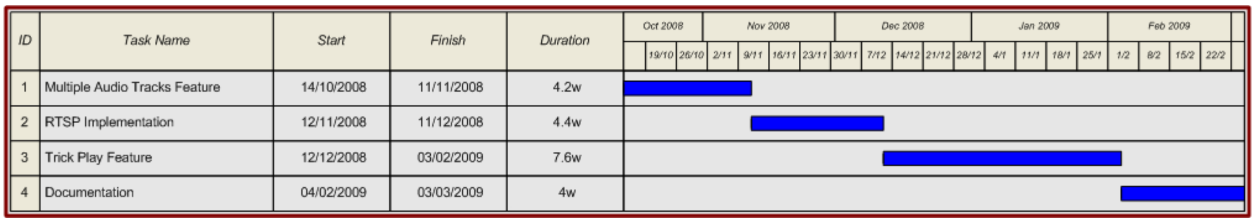
This chapter is dedicated to the description of all the work done during the development of the project.
It will explain how the development process went, explain the most important implementation details, the structure and sequence of the main modules. as the relations between the multiple classes.

4.1 Approach

This subsection will explain the project approach and the work plan defined.

Before the definition of the work plan displayed in the Figure 4.1, a carefull process of getting the requirements was done.

The work plan in Figure 4.1 was defined based on the average time taken by a developer who didn't had a full knowledge of the JME and JMF architectures, having this fact in consideration to the definition of the time estimates defined.
The implementations process was divided into 3 main features. 3 weeks for the development of the Multiple Track, 4 weeks for the RTSP implementation and 5 weeks for the Trick Play feature. Leaving 4 weeks for testing and documentation.
The work plan took in account the 2 weeks of christmas holidays from the end of December to the first week of January.
Although the estimates took in consideration extra time for any unexpected problem, the main plan was followed without any major deviations.



ID	Task Name	Start	Finish	Duration
1	Multiple Audio Tracks Feature	14/10/2008	11/11/2008	4.2w
2	RTSP Implementation	12/11/2008	11/12/2008	4.4w
3	Trick Play Feature	12/12/2008	03/02/2009	7.6w
4	Documentation	04/02/2009	03/03/2009	4w

Figure 4.1: Project Planning

4.2 Multiple Audio Tracks

As said before, in chapter 1 and 3, this feature was developed using the JME in the configuration CDC, using some of the APIs available on JMF.

The approach to to this module was to create a set classes to parse the Mpeg2-TS stream and detect the number of audio tracks available on the stream. This process is done in run-time and allows the stream to change the number or properties of the audio track. Once the process is started it will check for every Program Mapping Table in the Mpeg2-TS stream for the available audio tracks, and register them into a private data structure that is going to be used, in a later stage, to enable the user to select between the multiple audio tracks available.

To activate an audio track there were defined 3 different methods that can be selected through a group of java properties. This option was taken having in mind the multiple ways that a Mpeg2-TS decoder has to select the audio track, so implementing 3 different methods should cover almost the totality os the decoders available in the market.

All the details will be explained with in the next sections.

4.2.1 Multiple Tracks Detection

As said before the Multiple Audio Tracks Detection is based on a Mpeg2-TS parser, that gets all the information of the Mpeg2-TS stream and creates the structures that will enable the final user to select the audio track he intends to listen. [tfeW09d, Sta09]

As can be seen in Figure 4.2 the PMT packages in the Mpeg2-TS stream, work as an index to the programs that the stream contains, by analyzing its content, a structure with the information required can be generated.
This process of parsing that information is done on the Mpeg2PMT class.

4.2.2 Activate a Track

To activate an audio track 3 methods were defined, depending on the platform which is running a property can be set to define what is the method that is going to be used to activate the audio track.

From the developer point of view this is transparent, and this property should only be set once, as soon as you know what is the method that works better for the device that its going to be run. The method can be enabled in a Java property under the file jmf.properties.

The 3 methods used to activate an audio track are: Remove all the audio track entries that are not being used, replace the first entry with the entry that we pretend to play and change all the non used entries to a unknown Program.

This process runs at the same time that the player, generating PMT packets in runtime, for this reason the code had to be highly optimized.

In the following subsections will cover how each of the methods implemented is defined and their specific details of implementation.

4.2.2.1 Remove All Other Entries

This method was defined because some of the decoders available on the market only support one audio track per Mpeg2-TS stream, meaning that when this method is enabled the system will remove all the non used entries from the PMT, giving the decoder the perception that he only has one video and one audio program available.

On the Figure 4.2-a its shown how the process works and how the data is being modified.

The process for removing the non used entries has the following stages:

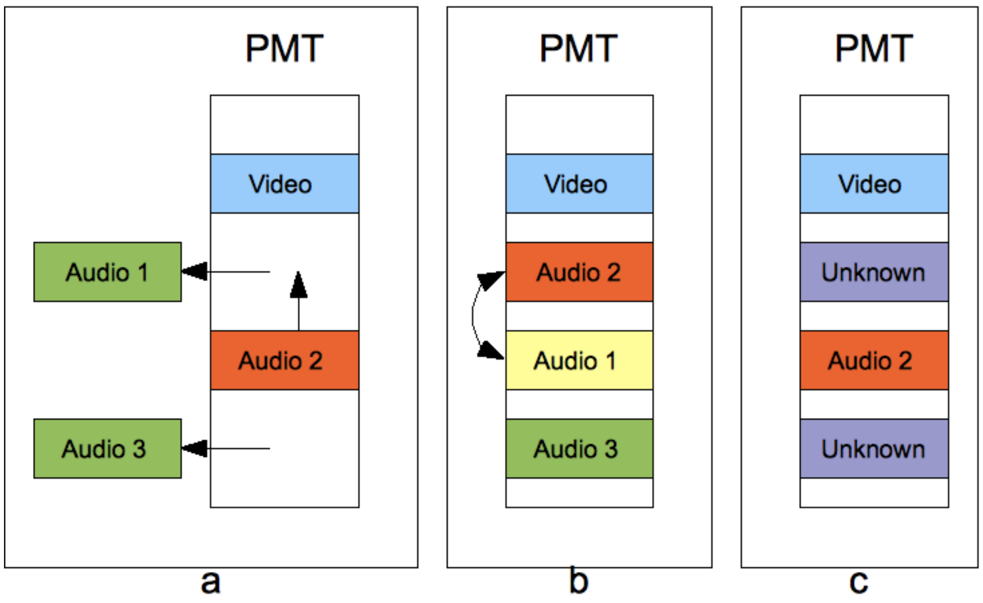


Figure 4.2: Audio Track Activation Methods

1. Check if the packet is a PMT.
2. If is a PMT check what is the current audio track enabled.
3. Generate a new PMT with only the current audio track entry and the current video entry.
4. Generate the new PMT packet and its CRC.
5. Put the packet into the original position and inject it to the decoder.

4.2.2.2 Replace First Entry

This method was implemented having in mind de usual case, where a decoder always plays the first audio program described in the PAT packet.

As described in the previous section this method also needs to generate a new PAT packet, so the behavior of the implementation is somehow similar to the previous one, having as the main difference the reorder of the program entries instead of removing all the ones that are not used.

Because the PAT program entries all have the same size, the only added method in the PAT generator class was the addition of a method to swap the first audio program entry by the one that in intended to play, then the new Packet is generated with the reordered entries and a new checksum field.

On the Figure 4.2-b it's shown the data flows through this process.

The process of putting the active entry as the first entry passes through the following stages:

1. Check if the packet is a PMT.

2. If it is a PMT check what is the current audio track enabled.

3. Swap the first entry with the active one.

4. Generate the new PMT packet and its CRC.

5. Put the packet into the original position and inject it to the decoder.

This process has a slightly better performance that the one described before because it doesn't change as many data, it only gets the active entry and puts it on the first audio entry available.

4.2.2.3 Change Entry Types

This method was especially developed having in mind the SJMC Emulator implementation, because of the non standard implementation of FFMpeg. FFMpeg first reads the data packets and then checks what is the type of media associated with that program. So changing the type of stream of the non used program entries will prevent FFMpeg to play the wrong audio track.

As described in the first 2 implementations, a new PMT packet will be generated carrying the entire list of programs but only with one audio and one video program entries. Having all the other entry types changed to an unknown program. This process enables to do an audio track changing even on the SJMC Emulator, despite the SJMC Emulator is not a commercial product it is widely used as a test platform for developers.
The data flow process is identical to the 2 previous implementations, changing only in the way the PMT packed is constructed, as you can see on the Figure 4.2-c.

The process of changing the type of the entries to an unknown type is slightly heavier that the other 2 because of the need to check and change all the program entries in the PAT table.
The process takes the following steps every time that the parser finds a PMT packet:

1. Check if the packet is a PMT.

2. If it is a PMT check what is the current audio track enabled.

3. Change all the non used entry type to Unknown type.

4. Generate the new PMT packet and its CRC.

5. Put the packet into the original position and inject it to the decoder.

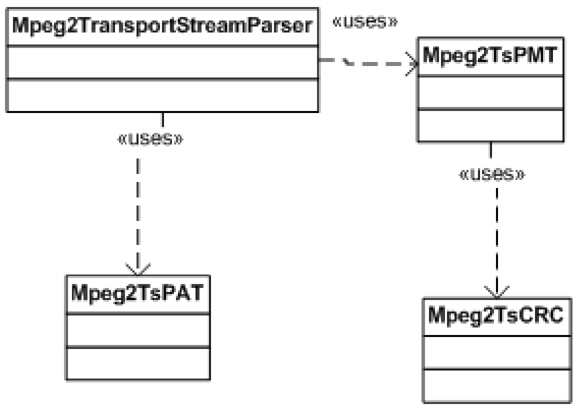


Figure 4.3: Audio Track Detection/Activation Class Diagram

4.2.3 Classes

This section was meant to describe the classes main functionalities and to give an overview of the class relations, thoe the class diagram on th Figure 4.3.

4.2.3.1 Class Mpeg2TransportStreamParser

The detection of the multiple audio tracks is done under this Java class file, and its main function is to act as a filter to all the Mpeg2-TS packets (See appendix 2 for more Mpeg2-TS details). The process starts by getting a Program Allocation Table packet and getting the PID to the Program Mapping Table packets. Once the Mpeg2TransportStreamParser class has a reference to the PMT PID it will catch all the PMTs and checking what are the audio tracks available in the stream.
The first time that he finds a PMT packet the class will build a structure with all the Audio Track Program details: active, type of codec and PID.
Once this structure is defined it will only be changed if the stream structure changes (more audio tracks or different properties like PID or Codec) or if the user selects to use another Audio Track, In this last case only the activate field will be changed.

4.2.3.2 Class Mpeg2PAT

This class is only used as a way to get the PID for the PMT packets.

4.2.3.3 Class Mpeg2PMT

This is the class responsible for getting all the information regarding the PMT packets, each PMT will have all the details to the programs available in the stream.

Each PMT will have references to all video, audio and data (ex: subtitles) programs available in the stream. Each of the entries will have the PID of the packets containing the program, the type of program and the codec used. It will act as a index to all media programs available on the stream.

In this class is also implemented a method to construct new PMT packets, this was an essential need because each control packet (Ex: PMT, PAT) in a Mpeg2-TS stream has a checksum field to guarantee that the data is not corrupted. So the need to change the PMTs structure took to the need to generate a new checksum, and rebuild an entire PMT.

4.2.3.4 Class Mpeg2TSCRC

This class was created as a way to generate the checksum needed to create the new PMTs. It extends the CRC32 class from java.util.zip package but implements a new CRC32 algorithm, it only extends the CRC32 class as way to allow the generation of both Mpeg2TSCRC and a normal CRC32.

4.3 Real Time Streaming Protocol

The Real Time Streaming Protocol is a protocol for use in streaming media systems, which allows a client to remotely control a streaming media server, issuing VCR-like commands such as ”play” and ”pause”.
The sending of the stream itself is not part of the RTSP protocol, its main purpose is only to negotiate the streams (For more information about RTSP please refer to appendix A).

The implementation process to this feature is based on 2 different modules, one module that negotiates with the RTSP server and another module that actually plays the stream. Having in mind this characteristic the system had to be developed over a multi thread architecture, one of the main reasons that supports this option is related with the fact that the media stream is sent over UDP protocol, which does not guarantee data delivery, for this reason most of the RTSP servers have a Keep-Alive system which makes sure the client is alive while the server is sending data. The server expects messages from the client over the negotiation channel in a predetermined time interval.

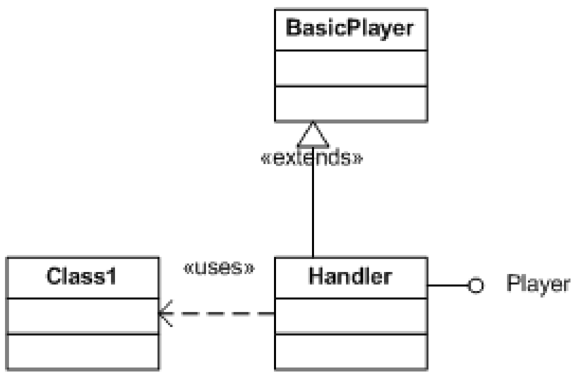


Figure 4.4: Class Diagram for RTSP

4.3.1 RTSP Implementation

As it was described in the Requirement specification, a full implementation of the RTSP protocol is mandatory.
To complete this requirement, a set of classes were developed in order to implement the full RTSP protocol.

4.3.1.1 Class Handler

The class Handler extends the class BasicPlayer which implements the interface Player, so this way all the logic of the RTSP negotiation is done in this class.

The main purpose of this class is to parse the RTSP url, connect to the server on the specified port, and do all the high level negotiation required. As it extends the BasicPlayer class which actually plays the stream, the Handler can override some methods to wait that the RTSP negotiation goes through.

So that when a developer writes an application and he wants to open an RTSP url, the Handler will do the following steps:

1. Parse the RTSP url to get the server, port and asset.

2. Sends a Describe message to the server and waits for a reply.

3. Once he gets the reply, parses the reply to understand what are the services available in the server.

4. Sends a Setup message to the server requesting for the setup of the asset present on the url, this request includes some details like preferred port, the type of stream supported (RTP or UDP) and waits for a reply.

5. If everything goes OK the Server will send a reply with the detailed information about the requested asset (bitrate, length and codec), the server will also provide the protocol in witch the stream will be sent, the port to which it will be sending to and a session ID. The session ID can be used to make additional requests to the server (stop, set rate, pause...).

6. The Handler will then create a UDP or RTP player listening to the port provided by the setup command and send a Play message to the server.

7. As soon as the server replies to the Play request, data comes and the player is started.

8. It launches the Keep Alive Thread which will be in charge of maintaining the connection with the server.

This is a basic negotiation between the RTSP implementation and the RTSP server. In the majority of the cases the negotiation doesn't change much.

If the client requests a rate change (Trick Play) by calling the setRate method, the system will launch a new negotiation with the server. Each handler has a session ID for stream that is playing, so the Handler will do the following steps:

1. The handler will send a Play command with the intended rate, and waits for a reply.

2. The server will send an OK message and the video stream starts to come at the rate requested.

4.3.1.2 Class RTSPUtil

The class RTSPUtil will work only as a wrapper to the protocol, it is responsible for the syntax of the implementation, and has methods to generate RTSP messages based on the

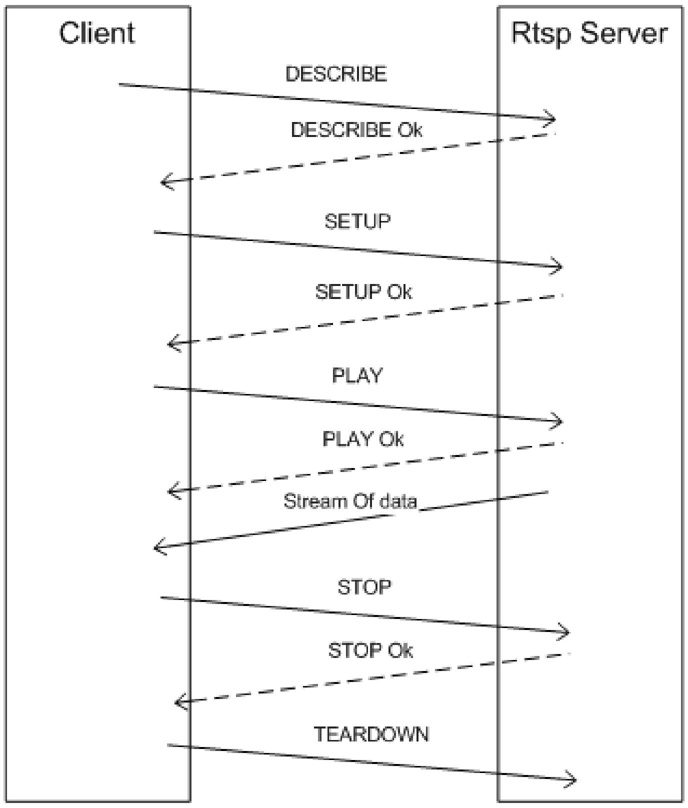


Figure 4.5: RTSP Session

parameters that are passed a arguments to the methods.
It also implements the Keep Alive thread and the parser for the RTSP server messages.

4.3.2 Error Handling

The implementation of the RTSP stack has some error handling features, it supports connection drops and wrong RTSP URLs.
This was done by adding some validation code along all the process of negotiation to check if the server is returning an error message. If the server replies with an error message the player will throw an Exception which can be caught by application that is creating the player.

4.3.3 Server Messages

The Handler class implements a thread just for receiving server messages, this feature is important because the server sends spontaneous messages to advert any change. A common application of this kind of messages is every time a stream is going to end, the server will send a message to the client notifying it of the End Of Stream.

4.4 Trick Play

As described in the requirement specification, the Trick Play functionality should be able to do Fast Forward and rewind a movie stored in a file in a rate up to 64X. This takes a lot of concerns, because the maximum amount of data in a normal Hard Drive is around 30Mb/s, this would restrict a normal Mpeg2-video file in Standard Definition (bitrate around 8Mb/s) to have a top speed of 4X. So the development of this feature had to take this need in consideration, and find a way to bypass that restriction.

The way to solve this problem is to create a 2 phase Trick Play mode. A system that can detect at runtime if the reading limit of the HD is being reached, and when it does, it changes to an IFrame mode, a mode where it will only show the KeyFrames of the video. This is achieved by calculating in runtime where it should be and jump directly to that position in the file.

The main problem with this approach was the kind of formats supported, its in the requirement specification that the system should be able to do Trick Play in both Mpeg2-Video and Mpeg4-H264, both of them are very different implementations, and have very

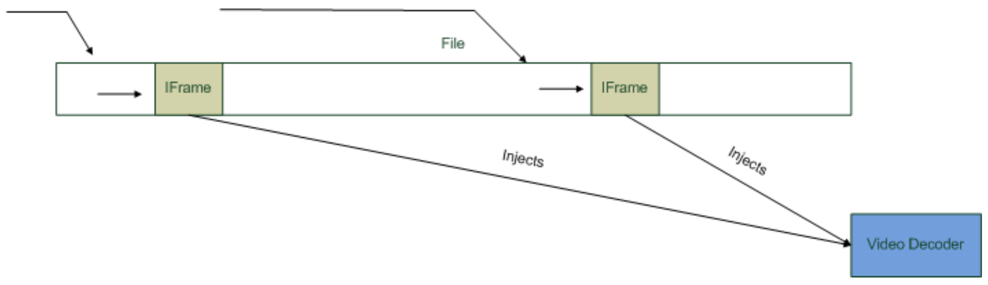


Figure 4.6: IFrame Mode

different ways to represent their data.
The solution was to create a set of classes to parse the information in both codecs and write the methods to find the IFrames.

4.4.1 Smooth Mode

Smooth Trick Play mode is when the device can reproduce the original video faster and with no breaks, it should look like the original video but faster.

To achieve this goal the system has to inform the decoder that the movie is going to be displayed at a different rate and inject the data into the decoder at the defined rate.

4.4.2 IFrame Mode

An InfraFrame or KeyFrame is a frame in which a complete image is stored in the data stream. In video compression only the changes that occur from one frame to the next are stored in the data stream in order to reduce the amount of data that must be stored. But IFrames are sent at a certain rate to improve the quality of the movie, if a video was encoded with only transition frames, one error at the start of the movie would affect most of even all the length of the movie, so for a good quality movie it should have at least one IFrame per second.

The Iframe mode starts when the system understands that the Hard Drive can't coupe with the demand of data, at that point starts jumping in the file to an estimate position that should correspond to the approximate time where it should be at that rate. This is in a brief explanation how the Iframe mode works. A graphical representation can be found in Figure 4.6.

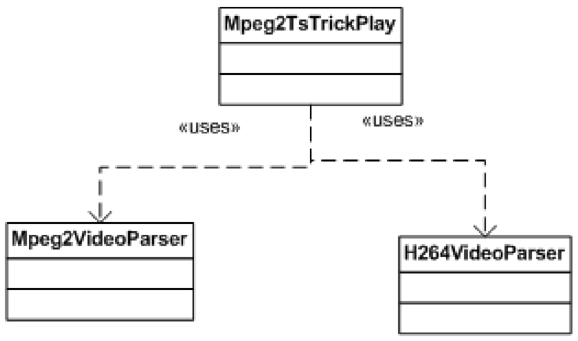


Figure 4.7: Class Diagram for the Trick Play Implementation

4.4.3 Classes

4.4.3.1 Mpeg2tsTrickPlay

This class has all the logic regarding the Trick Play features, its where all the logic and decisions are made.

4.4.3.2 MpegVideoParser

This class was developed as the Mpeg2-Video codec parser and has the necessary methods to look for the IFrames in the

4.4.3.3 H264VideoParser

The same way the class MpegVideoParser implements the methods to parse the Mpeg2-Video, the H264VideoParser imlements the equivalent methods to parse the Mpeg4-H264 videos. Because the Mpeg4-H264 codec has a much more complex specification, a separate class had to be created.

Chapter 5

Tests and Results

The features were developed using agile methodology. As such, most of the testing was done during the development phase. The 3 features implemented suffered unitary test for each of the sub features implemented and a final integration testing cycle was taken after all the features completed and integrated under the SJMC platform.

This chapter will cover the test cycle taken for each feature developed as for the final integration process. All the features were tested as independent as it was possible, and they were used in demonstrations and testing by some work colleges to detect any additional problem that could occur, especially the RTSP and the Trick Play feature were widely used by other developers and commercial employees of Sun Microsystems, reporting any problem that they have found.

The RTSP implementation had an especially heavy test cycle, mainly because of the multiple RTSP protocol implementations currently on the market.

5.1 Multiple Audio Tracks

This section will cover the test process done to ensure that the feature was working with the pretended performance and stability and the results obtained with the testing.
The testing for this features were done under the SJMC Emulator and the SMR, allowing me to check if the functionality was working with a software and a hardware decoder.

5.1.1 Tests

As refereed in the introduction of this section, this feature had a full test cycle on both the SJMC Emulator and the Sun Media Receiver, allowing to correct and fix any incompatibility issue that could appear.

The testing was divided into 2 main groups, the first test cycle was done over file streams, and the second using Video On Demand streams using the RTSP feature also implemented.
The first stress test was done by for a long period of time a user was changing between audio tracks in the same video to detect any possible memory leak.
The second set of tests was to create a small playlist of files, some of them with multiple audio tracks and the rest with only one audio track to check if was causing conflicts.
The third test was to let the player run a multiple audio track video in a loop and check if every time the video restarted it would restart with the previously selected audio track.
The forth test was to let the player run a playlist of files used to the second set of tests for several days, and check if the track changing had any significant performance lost over the time.

5.1.2 Results

The results in terms of performance were good, less than 0.5 seconds to change between audio tracks on the SMR and about 0.2 seconds in the SJMC Emulator, this difference was caused by the faster hardware available to the SJMC Emulator especially the processor speed and the memory available.

About the stress testing, some of the tests had some stability issues that were fixed as soon as they were found. Especially when running for a long period of time, the performance was dropping and in some cases causing out of memory problems. But in the end all the problems detected were fixed and the final product was stable and with a good performance.

5.2 Real Time Streaming Protocol

This section will cover all the testing done and results obtained during the development of this feature.

The RTSP testing has 2 separate phases, on a first approach we didn't had an RTSP server in the Sun Microsystems Dublin facilities so we had to use a server located in the Sun Microsystems office in Santa Clara, California, USA. This first phase was only to test the compliance of the RTSP protocol as we were unable to get a video stream from the US to Dublin mainly because of the connection speed, so we were using the RTSP server located in the US and VLC as the client providing the UDP stream, in this phase it was only possible to test the basic negotiation of the RTSP protocol: play a video stream, stop the video stream and ask for a teardown to the server.
After a couple of weeks on the development of the RTSP stack, there was installed a Picostar server (See section 2.2.X for further information) in the Dublin facilities, so that we could test the full capabilities of the RTSP stack.

Both of the test cycles were done over SMR and the SJMC emulator.

5.2.1 Tests

This cycle was intended to test the full list of functionalities (negotiate a video stream, Trick Play over RTSP, pausing the video and stop the stream) as for performance testing and error handling.

So for this purpose there were defined 3 groups of tests.

The first set of tests was to check the compliance of with the RTSP protocol, this was done by connecting and negotiating with several RTSP servers (mainly Streamstar servers) with slightly different protocol implementations.

The second test was stress testing were the player would be playing several streams in loop for several days, and testing some of the features (like Trick Play, and pausing) and do some basic benchmarks on performance to see if the performance was dropping along the time.

The third set of tests was to test the tolerance to connection errors, putting a test program playing in a loop and cutting the connection to the server to see if the player was able to connect once the connection was established.

5.2.2 Results

In terms of compliance, we had some problems with the multiple implementations, that were fixed during the test cycle, mainly protocol variations that were not supported in the first implementation.

Performance wise the requests had a lag of about 200 to 300 milliseconds in a closed network, this issue was due to the network lag and the time that the server takes to interpret and send the request. In the overall they were good performance figures (about the same time it takes to change a regular TV channel).

Regarding the connection drops, the player stopped as it was intended and was able to reconnect to the server as soon as the connection was established.

Regarding stress testing, the tests were very good, the demo application was running for several days with no problems, only a couple of connection drops that could be handled without additional intervention.

5.3 Trick Play

In this section will be covered the tests and results done over the development and after to check the accuracy of the speed and performance.
As said before, the Trick Play was only implemented to the SMR platform and not to the SJMC Emulator because of the way the FFMpeg works.

On this feature there were developed 2 different sets of tests, the first one for Mpeg2-Video codec and the second for Mpeg4-H264 codec. Both test cycles were testing the reliability and the overall performance of the feature, checking if the jumps were done in an accurate way (with 10% tolerance at 60X).

5.3.1 Tests

The tests were based on playing videos of both codecs in different rates and check if the time to run the full video at a N rate was taking N times less then the original file at normal rate.

This tests were done at speeds of 2X, 4X, 8X, 16X, 30X and 60X, the commercial version will only allow developers and users to go up to a 30X but for testing purposes the boundary was taken higher then the final product.

5.3.2 Results

Some bugs showed up as soon as the test cycles started, some Trick Play with sound in the background, and some lack of accuracy when running in higher rates. These issues were fixed as soon as possible, and in the end product there wer no errors to report.
The overall result was very good, the processor usage was never over the 30% and the rates were pretty accurate (having in mind the 10% tolerance).

5.4 Integration Tests

This section will cover all the integration tests, and the results obtained during the final integration of the 3 features. The Audio Track feature and the Trick play feature are running only on device so there was no conclusions to take having 2 devices running at the same time, but the RTSP feature was tested with 3 devices using the same RTSP server to check any problem that could occur with multiple connections.

5.4.1 Tests

The test cycle for the integration comprehended some sanity tests for each feature implemented just to check that all features were working in the final product.

The final integration testing was done by running some of the demos used to test each feature.
To test the Multiple Audio Tracks a playlist feature, we ran a demo that was looping 2 videos, one with multiple audio tracks and the other with only one audio track.
To test the Trick Play feature, there was developed a demo application to run a 90 minutes video in loop, every time that the video got to the end the video would restart with twice the speed of the previous one, ranging from 2X to 64X.
For the RTSP testing, we setup 2 SMRs (one SD and one HD) and the SJMC emulator to get the same video file and do some Trick Play, to test the performance and some eventual connection problem.

5.4.2 Results

There was nothing to report, all the tests were executed without any major problem, mainly because all the features have been tested at the same time, and with some level of integration.
The RTSP test had a small increase in the answer time for some requests, but that was mainly because of the overflow of the network, but no errors or connection drops were reported.

Chapter 6

Conclusions and Future Work

6.1 Objective Satisfaction

The main purpose o this project was to add some high level features to the JMF using the JME-CDC SDK, integrate them into the SJMC platform, and create a simple set of APIs to control those features.

In this context the following tasks have been done:

- Studied the SJMC platform and specification in order to trace de development plan.
- Studied the already implementation of the JMF features and APIs, evaluating the viability and of implementation into the SJMC platform, and the perspective of continuing its usage in the next phases of the implementation of the SJMC platform.
- Studied the JME-CDC SDK and is profiles in order to be able to get a high performance and reliable final product.
- Made some market research in some of the most used IPTV solutions, RTSP solutions and general multimedia devices.

All the the above tasks were part of a study to get to know the technologies that were going to be used, During the development phase all of the defined requirements were defined. This are:

- Creating a set of APIs to select the audio track that is being played.

- Creating a RTSP stack to enable Video On Demand, enabling server-side Trick play.
- Creating the Trick Play functionality for files encoded in Mpeg2-Video and Mpeg4-H264.
- Test the final product in a controlled environment.
- Have a full operational setup ready to be used in demonstrations.

As a result all the objectives for this project were fully accomplished. Having nothing to report.

6.2 Future Work

Most f the features i'd suggest to implement in the future are already planned in the SJMC project.

These are:

- Private Video Recording - be able to record and store in persistent storage streams that are received through a Video On Demand System.
- Subtitle support - Enable the SJMC platform to display subtitles, by rendering text on the screen or by an additional layer of subtitles in the media stream.
- Additional Codecs Support - Add the support for AVIs, FLV, MKV and WMV.
- Youtube support - Support for web casting and podcasting services.

References

[Blo09] Tao Sun Blog. Java family, February 2009. http://blogs.sun.com/tao/resource/CA-Demo/JavaFamily.png.

[HS09] R. Lanphier H. Schulzrinne, A. Rao. Real Time Streaming Protocol, February 2009. http://tools.ietf.org/html/rfc2326.

[Inc09a] Apple Inc. Apple tv, February 2009. http://www.apple.com/appletv.

[Inc09b] TIVO Inc. Tivo, February 2009. http://www.tivo.com.

[Mic09a] Sun Microsystems. Java Media Framework, February 2009. http://java.sun.com/javase/technologies/desktop/media/jmf/.

[Mic09b] Sun Microsystems. Java Micro Edition, February 2009. http://java.sun.com/javame/index.jsp.

[Mic09c] Sun Microsystems. NetBeans Project, February 2009. http://www.netbeans.org/kb/index.html.

[Mic09d] Sun Microsystems. Sun Streaming System, February 2009. http://www.sun.com/servers/networking/streamingsystem/features.xml.

[Pro09] VideoLAN Project. Vlc media player, February 2009. http://www.videolan.org/vlc/.

[Sta09] International Standard. Information technology — Generic coding of moving pictures and associated audio information: Systems, February 2009. http://neuron2.net/library/mpeg2/iso13818-1.pdf.

[tea09] FFmpeg team. Ffmpeg, February 2009. http://www.ffmpeg.org/.

[tfeW09a] the free encyclopedia Wikipedia. Ffmpeg, February 2009. http://en.wikipedia.org/wiki/FFmpeg.

[tfeW09b] the free encyclopedia Wikipedia. Java Media Framework, February 2009. http://en.wikipedia.org/wiki/Java_Media_Framework.

[tfeW09c] the free encyclopedia Wikipedia. Java micro edition, February 2009. http://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition.

[tfeW09d] the free encyclopedia Wikipedia. Mpeg transport stream, February 2009. http://en.wikipedia.org/wiki/MPEG_transport_stream.

[tfeW09e] the free encyclopedia Wikipedia. Netbeans, February 2009. http://en.wikipedia.org/wiki/NetBeans.

[tfeW09f] the free encyclopedia Wikipedia. Vlc media player, February 2009. http://en.wikipedia.org/wiki/VLC_media_player.

Appendix A

MPeg2-TS

The Mpeg2 - Transport Stream is a communication protocol for audio video and data designed to allow multiplexing of video and audio. The Transport Stream supports connection errors and it was developed to support transmissions over unreliable media, it is widely used for satelite transmissions, and Digital Video Broadcast. BlueRay Discs and HD-DVD use Mpeg2-TS files as well.

Transport Stream is a container format, it is used to encapsulate layers (programs) of video and audio and its implementation is independent from the audio and video codecs of the programs.
The basic unit of the Mpeg2-TS is the packet, a 188 bytes sequence that starts with the value 0x47. The packet contains a control section which has the information regarding the kind of packet, a Program ID and checksum for error control.

Some of the packets are used only as indexes, the 2 main kinds of index packets in a Mpeg2-TS stream are:

- **PAT: Program Allocation Table** — Works as an index with the information regarding other control packets. Has the program number (PID) for the PMT packets and other control packets. The PID of the PAT is always 0x00. As a good practice a stream should have 2 PATs/s to be able to provide all the information needed as soon as possible.
- **PMT: Program Mapping Table** — The Program Mapping Table contains information about the rprograms available in the stream, each entry in the PMT packet has a PID that points to the packets containing data for that program as well as some metadata information about the kind of content (video, audio or data) each program has. In the most common cases it even has the codec in which the program is encoded.

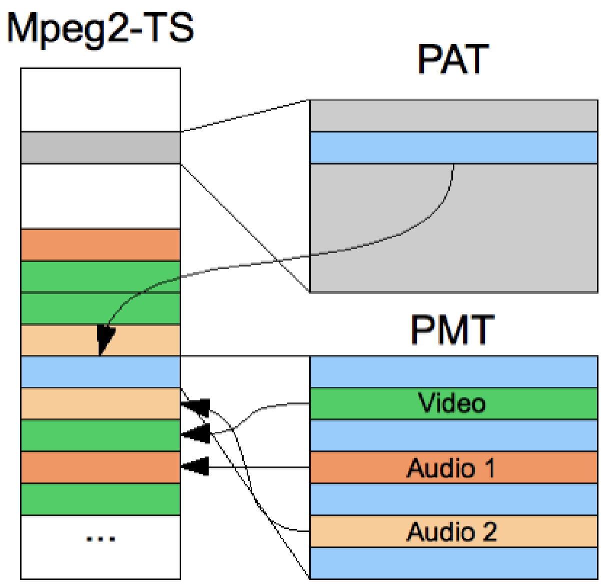


Figure A.1: Mpeg2-TS: Telation between PAT, PMT and the Programs

Appendix B

Development Environment

B.1 Subversion

To help structuring the work developed each feature was developed in a separate branch. With this approach, a copy of the trunk was created at the beguining of the development of each feature being merged when the feature was finished and tested. This way the development of each feature was isolated from the rest of the development.

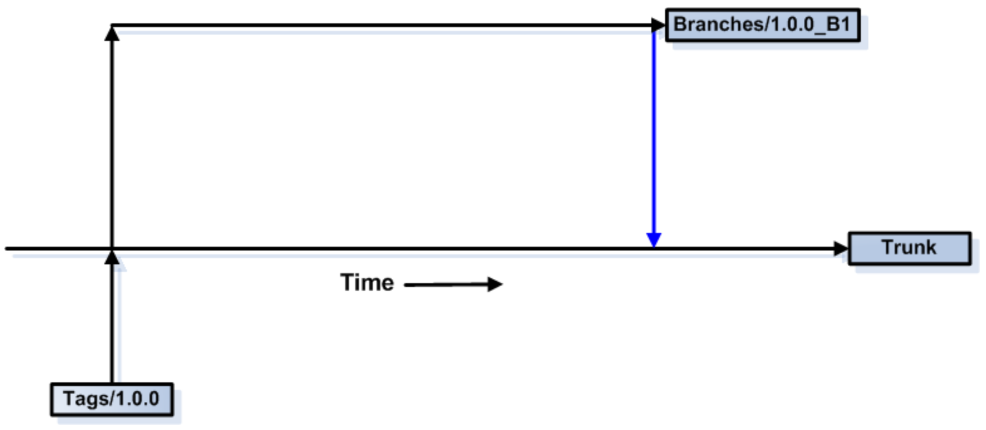


Figure B.1: Subversion: The process of branching and merging with the trunk