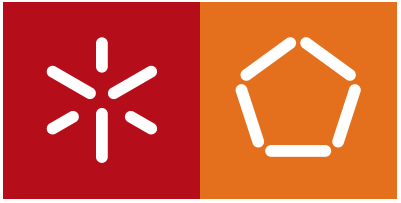
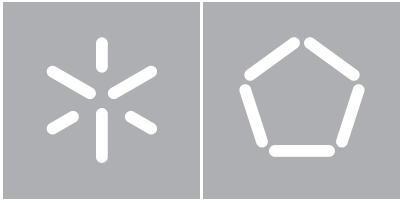


Universidade do Minho
Escola de Engenharia

Vasco Miguel Gonçalves Coelho

An SNMP-Based Audio Distribution Service Architecture

Fevereiro de 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Vasco Miguel Gonçalves Coelho

An SNMP-Based Audio Distribution Service Architecture

Dissertação de Mestrado
Mestrado em Engenharia de Redes e Serviços Telemáticos

Trabalho realizado sob orientação de

Professor Bruno Alexandre Dias

Fevereiro de 2018

Anexo 1

Declaração relativa ao depósito de dissertações e teses no RepositóriUM

Nome: Vasco Miguel Gonçalves Coelho

Correio eletrónico: vasco.vmgc@gmail.com Tel./Telemóvel: +351 964231338

Número do Cartão de Cidadão/Bilhete de Identidade: 13420962 1 ZY5

Título(s) da dissertação de mestrado ☑ / tese de doutoramento ☐:

An SNMP-Based Audio Distribution Service Architecture

Orientador(es): Bruno Alexandre Dias

Data de conclusão: 02/2018

Mestrado e área de especialização/Doutoramento e especialidade:

Mestrado em Engenharia de Redes e Serviços Telemáticos

Declaro que concedo à Universidade do Minho o direito não-exclusivo e irrevogável de arquivar, reproduzir, comunicar e/ou distribuir através do seu repositório institucional, nas condições abaixo indicadas, a versão final da minha dissertação/tese em suporte digital, aprovada após a realização das provas de defesa pública e, quando for caso disso, após confirmação pelo(s) orientador(es) e homologação pelo presidente do júri[1] da introdução das alterações solicitadas.

Declaro que autorizo a Universidade do Minho a arquivar mais de uma cópia da dissertação ou tese e a, sem alterar o seu conteúdo, convertê-la para qualquer formato de ficheiro, meio ou suporte, para efeitos de preservação e acesso

Declaro que a dissertação ou tese agora entregue é um trabalho original e que, contendo material do qual não detenho direitos de autor, obtive autorização prévia do detentor dos referidos direitos para conceder à Universidade do Minho os termos requeridos por esta licença.

Declaro também que a entrega do documento não infringe, tanto quanto me é possível saber, os direitos de qualquer outra pessoa ou entidade.

Se o documento entregue é baseado em trabalho financiado ou apoiado por organismo/financiador que não a Universidade do Minho, declaro que cumpri quaisquer obrigações exigidas pelo respetivo contrato ou acordo.

Retenho todos os direitos de autor relativos à dissertação ou tese e o direito de a usar em trabalhos futuros, como artigos ou livros.

Concordo que a minha tese ou dissertação seja colocada no repositório da Universidade do Minho com o seguinte estatuto (assinale apenas um dos quatro estatutos):

1. ☐ Disponibilização imediata do conjunto do trabalho para acesso mundial
2. ☑ Disponibilização do conjunto do trabalho para acesso exclusivo na Universidade do Minho durante o período de ☑ 1 ano, ☐ 2 anos ou ☐ 3 anos, sendo que após o tempo assinalado autorizo o acesso mundial.
3. ☐ Disponibilização do conjunto do trabalho para acesso exclusivo na Universidade do Minho.
4. ☐ Outro (se assinalar este estatuto, a declaração deve ser obrigatoriamente acompanhada de uma exposição que explicite a necessidade de um estatuto de excecionalidade)

Braga/Guimarães, 20 / 02 / 2018

Assinatura: Vasco Coelho

[1] *Vide* artigo 124.º Anexo ao Despacho RT-41/2014.

Anexo 2

Formulário para depósito legal[2] de dissertações e teses no RepositóriUM

Elementos de identificação:

Nome do autor: Vasco Miguel Gonçalves Coelho

Título(s) da dissertação de mestrado ☑ / tese de doutoramento ☐:

An SNMP-Based Audio Distribution Service Architecture

Orientador(es): Bruno Alexandre Dias

Data da concessão do grau:

Mestrado e Área de Especialização/Doutoramento e Especialidade:

Mestrado em Engenharia de Redes e Serviços Telemáticos

Escola/Instituto: Escola de Engenharia

Departamento/Centro de afiliação[3]: Departamento de Informática

Área disciplinar (área FOS)[4]: Engenharia Electrotécnica, Electrónica e Informática

Identificador único e permanente do trabalho (TID) atribuído pelo RENATES:

Classificação:

Classificação final:

No caso de dissertação de mestrado:

N.º ECTS da dissertação:

Classificação em valores (0-20):

Classificação ECTS com base no percentil (A a F):

Financiamento público:

Sem financiamento público: ☐

Pelo estabelecimento que confere o grau (caso seja público): ☐

Pela Fundação para a Ciência e Tecnologia (FCT): ☐

Identificador da Bolsa FCT:

Outro financiamento público: ☐

Obs:

[2] *Vide* Decreto-Lei n.º 115/2013 e regulamentação presente na Portaria n.º 285/2015.

[3] Indicar se aplicável o Departamento/Centro para efeitos de associação à respetiva comunidade no RepositóriUM.

[4] Preencher de acordo com lista abaixo apresentada.

To my parents and brother
António, Zulmira and Pedro

Acknowledgments

In the first place, I thank my advisor, Professor Bruno Alexandre Dias that agreed to guide me through this work with such interesting subject matter, for the support and advice given during the duration of the thesis.

Next, I want to express my gratitude to all my teachers of the MSc course in Engineering of Computers Networks and Telematic Services, for sharing their knowledge, as well as, their support during my time in the University of Minho. I thank my colleagues for the work done together, the talks, advice and good times shared.

I thank my closest friends for showing their great support, motivation, concern, interest in my work and their big patience of dealing with me.

Finally, I thank all the members of my family for the shown support, especially, my parents who are always a source of great support and advice, and always encourage me to study.

Abstract

The constant growth of integration and popularity of "Internet of Things" devices is affecting home automation systems, where new technologies were introduced, in the recent years for this particular sector. These automation systems integrate devices that can be anywhere in the house, connected to a home network, either through a wire or wireless connection. A home automation system can be used to control air conditioning, lighting, pool control systems, home-entertainment systems and much more.

Within the field of home-entertainment systems, the best known technologies are the Digital Living Network Alliance and the Digital Audio Access Protocol, which provide interoperability to allow sharing of digital media content between devices across a home network. However, these technologies have the disadvantage of being proprietary, maintaining restrict documentation access, complex architectures and concepts and not optimal to specific purposes, like audio distribution.

The main goal of this project was to prove that is possible to use standardized protocols, such as the Simple Network Manager Protocol and open source tools in order to develop a music distribution service that allows the implementation of similar features than the ones already existing proprietary technologies. As such, the implementation prototype system allows a user to manage and play audio from a music collection that is stored in a single home audio server. The system architecture enables audio streaming between the server and the various devices in the same local network. Further more, the music collection, can integrate virtual audio files that are available from external music sources, like iTunes, etc.

Resumo

O constante crescimento de integração e popularidade da "Internet das coisas" tem atualmente afetado sistemas de domótica, onde cada vez mais tecnologias têm vindo a ser desenvolvidas nos últimos anos para este sector em particular. Estes sistemas de domótica integram dispositivos que podem estar em qualquer parte de uma casa, ligados à rede seja através de um cabo ou por wireless. Um sistema de domótica pode ser usado para controlar: ar condicionado, iluminação, sistemas de controlo de piscinas, sistemas de entretenimento, entre outros.

Na área de sistemas de entretenimento, as tecnologias mais conhecidas são Digital Living Network Alliance e Digital Audio Access Protocol, que fornecem interoperabilidade de modo a permitir a partilha de conteúdos digitais multimédia entre dispositivos que se encontram na mesma rede local. Contudo, possuem a desvantagem de serem tecnologias proprietárias, com documentação e manuais restritos, arquiteturas e conceitos complexos, e não otimizados para fins específicos, tal distribuição de áudio.

O principal objetivo deste projeto foi provar que é possível usar protocolos normalizados, como o Simple Network Manager Protocol e ferramentas open source de forma a desenvolver um serviço de distribuição de música que permite a implementação de funcionalidades semelhantes às tecnologias proprietárias já existentes. Assim, o protótipo implementado permite a um utilizador gerir e reproduzir áudio de uma coleção de música que se esteja armazenada num servidor de áudio domestico. A arquitetura permite streaming de áudio entre o servidor e os diferentes dispositivos que se encontram na mesma rede local. Consequentemente, a coleção de música pode integrar ficheiros de áudio visuais que estejam acessíveis através de fontes externas de

música, como por exemplo: iTunes, etc.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

3GP Third Generation Partnership Project file format

AAC Advanced Audio Coding

AMR Adaptive Multi-Rate

API Application Programming Interface

ASF Advanced Systems Format

ASN.1 Abstract Syntax Notation 1

ATRAC3 Adaptive Transform Acoustic Coding 3

DAAP Digital Audio Access Protocol

DHCP Dynamic Host Configuration Protocol

DLNA Digital Living Network Alliance

DNS Domain Name System

DTCP Digital Transmission Content Protection

ER Entity-Relationship

FLAC Free Lossless Audio Codec

GNU GPL GNU General Public License

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

HTTPs HTTP Secure

IANA Internet Assigned Numbers Authority

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

JPEG Joint Photographic Experts Group

LPCM Linear Pulse-Code Modulation

MIB Management Information Base

MP3 MPEG Audio Layer III

MP4 MPEG-4

MPEG Moving Picture Experts Group

NAS Network Attached Storage

NMS Network Management Station

OID Object Identifier

OSI Open Systems Interconnection

PNG Portable Network Graphics

RTCP Real-time Control Protocol

RTP Real-time Transport Protocol

RTSP Real Time Streaming Protocol

SMI Structure of Management Information

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UI User Interface

UPnP Universal Plug n' Play

UPnP AV UPnP Audio Video

URL Uniform Resource Locator

WMA Windows Media Audio

XBMC XBox Media Center

XML Extensible Markup Language

Chapter 1

Introduction

The rapid growth of increasingly powerful microprocessors and microcontrollers, combined with the acceptance and integration of computers into our homes and lifestyle, are triggering the popularity of home automation systems. The "Internet of Things" (IoT) brought us the concept where every electronic equipment in a home is connected to a communication network.

Home automation systems refer to the use of computer and information technology to enable the control of domestic activities and systems, which include: air conditioning, lighting, home entertainment systems, houseplant and watering, pet feeding, pool management and house security systems. Such automation systems use a home network to integrate devices, providing control and monitoring through personal computers, smartphones or tablets, either locally or remotely from the Internet. Home automation systems should enhance security and energy efficiency.

Some of the first home automation systems were developed for a home entertainment application. Their main ability was to allow distribution of audio content throughout one or more buildings. Some would also have control voice operations, allowing to change music, TV channels, or redirecting a phone call to the home speakers.

Recent buildings and houses are currently being built with already integrated home automation systems. In other cases, where home automation systems are installed afterwards, it tends to make hardware integration much

more difficult. The majority of present automation solutions rely on close integration between the hardware and software, which the software subsystem is tight to a specific hardware solution. Further more, the software subsystems for home entertainment sector or more specifically for audio distribution systems uses proprietary technologies.

The best known technologies for digital media sharing between home network devices, as well as for remote control management of media servers, are the Digital Living Network Alliance (DLNA) [3], developed by Sony, and Digital Audio Access Protocol (DAAP) [4], by Apple. These technologies are responsible to provide interoperability for sharing digital media between multimedia devices across a network, as well as auto addressing and device discovery.

As the DLNA technology results from a partnership of many electronic manufacturers, it is based on an universal and open standard, and aims to be more device independent used by large number of vendors. On the other hand, DAAP is based only on proprietary technologies which makes media sharing only possible within Apple's *ecosystem* and a few media vendors that support apple products and technologies.

Both DLNA and DAAP have proprietary licenses, maintaining a very restrict and closed environment. It is required to pay licenses or royalties or consortium memberships in order to incorporate their technology into commercial or undergoing application development. Also, access to technical documentation is very restricted. Both systems rely on complex architectures and concepts as they must cover access management and sharing of all types of media: audio, video and image.

For music entertainment systems these solutions tend to be overly complex. So there are many vendors that use alternative solutions that present simpler and more efficient architecture and technology. Although, these are best for audio only entertainment systems, they are also based on closed and proprietary technology, like Sonos [5], Bluesound [6] and Bose [7].

With this in mind, it seemed evident the advantage of developing a specific architecture with a restricted context just for audio, entirely based on open standardized protocols and mechanisms.

So, the cornerstone objective of the project was to prove that is possible to develop an automated music distribution system to be used across several platforms, containing the main features of existing proprietary technologies. In the context of this project the Simple Network Management Protocol (SNMP) [8, 9] is used to control and manage collections of audio files in a music server, that can be distributed across a home network. Further more, the architecture allows the access to audio files from external music accessibility through the Internet.

In order to prove the concept, the prototype was built with open source tools, implementing basic functions to control and play audio files, by streaming them from a music server to a device within the local network. Figure 1.1 shows the solution concept built. In order to use SNMP, it was necessary to build a Management Information Base (MIB), so the music controller client could access and control the audio files from the music server. While the open source tools were used to allow audio streaming and its control from the music server to the playing devices.

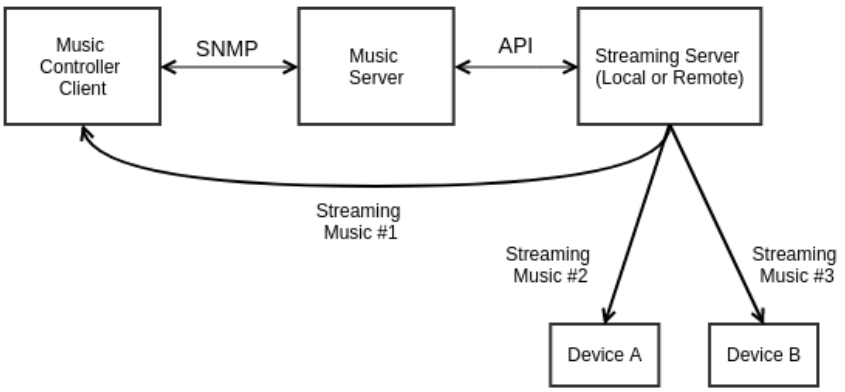


Figure 1.1: Music distribution service solution schematic.

In the remainder of this dissertation, Chapter 2 provides a review of related work, as well as, a study about the SNMP protocol, streaming protocols and audio codecs. Chapter 3 provides an overview of the prototype's architecture, as well as, detailed aspects regarding its implementation. Chapter 5

concludes the document.

Chapter 2

Related Technologies

This chapter introduces some work related the scope of the project. In particular, it describes the DLNA and the DAAP technologies, presenting a comparative study between them. Finally, it introduces the SNMP, some streaming technologies and open source audio codecs.

2.1 Digital Living Network Alliance

The DLNA [3], founded by Sony Corporation, provides interoperability guidelines that allow a wide selection of multimedia devices to easily connect with each other using the local network. DLNA devices uses the Internet Protocol (IP) in order to find and recognize each other and share their media content. DLNA is based on the Universal Plug n' Play (UPnP) technologies (which will be approached in Section 2.1.2). This serves for discovering, managing and controlling multimedia devices. It defines how the media content is identified, managed and distributed.

2.1.1.1 DLNA Protocol Stack

Table 2.1 (adapted from [3]) shows the DLNA Protocol Stack layers, its functionalities and related technologies or protocols.

The **Connectivity** and **IP Networking** layers resemble the same as in the Open Systems Interconnection (OSI) model. They establish how devices

Layer	Protocols	Functionalities
Link Protection	DTCP	How commercial content is protected on the network
Media Format	JPEG, PNG/LPCM, MP3/MPEG2	The media formats that can be identified
Media Transport	HTTP/RTP	How media content is transferred
Media Management	UPnP Architecture	How media is identified, managed and distributed
Discovery & Control		How devices self configure, discover and control
IP Networking	IP	How devices communicate
Connectivity	Ethernet/Wi-Fi/Bluetooth	How devices connect to the network

Table 2.1: DLNA Layers.

are connected in a home network (Ethernet/Wi-Fi) and how such devices communicate to each other (IPv4 and IPv6). In theory, any home device can be connected to any other connected device in the IP world, allowing applications running over different media to communicate transparently. Currently, DLNA supports connectivity over Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11 a/b/g/n), Bluetooth, HPNA[1] and MoCA[2].

The next two layers (**Discovery and Control** and **Media Management**) follow the UPnP architecture, as described below (Section 2.1.2.1). These layers enable devices to automatically configure themselves about network properties, device discovering and lastly, for device control. After these configurations have been established, the devices can identify, manage and distribute their media content.

The **Media Transport** layer defines how media content is transferred over the network. Currently, it uses Hypertext Transfer Protocol (HTTP) as mandatory and Real-time Transport Protocol (RTP) as optional [10].

The **Media Format** layer designates a set of required and optional media formats for every device category (see Subsection 2.1.1.2) and for each of

[1]Home Phoneline Networking Alliance
[2]Multimedia over Coax Alliance

the three classes of media: image, music and video[3]. DLNA Media Format expects to achieve a baseline of interoperability while encouraging continued innovation in the media codec technology. Hopefully improvements in this area will result in a better network bandwidth usage and media quality.

The last layer is the **Link Protection** layer. It uses Digital Transmission Content Protection (DTCP), which defines a cryptographic protocol for protecting media entertainment content from unauthorized copying, allowing only legitimate content to be delivered from a source device to one that has been approved with a copy protection [11]. This protection system enables secure sharing of copyright content between devices in a home network, by ensuring that the content is protecting it from piracy and/or illegitimate redistribution.

2.1.1.2 DLNA Device Classes

DLNA separates the devices within three Certified Device Classes: Home Network, Mobile Handheld and Home Infrastructure Devices [12]. Table 2.2 shows these categories for Home Network and Mobile Handheld Devices of DLNA Architecture Device, including some examples. Each class contains multiple subclasses, which will be briefly explained:

- **Home Network Devices** are subdivided into five categories. Digital Media Server (DMS) that can store content and make it available over the network; Digital Media Player (DMP), which finds the content stored in the DMS and can also provide playback and rendering capabilities; Digital Media Renderers (DMR) plays the content received from a Digital Media Controller (DMC), which can also be found in a DMS or DMP; Digital Media Printers (DMPr) provide print services where the DMP and DMC can invoke a print action.

- **Mobile Handheld Devices** are quite similar to Home Network Devices, despite being linked through a wireless connection to the network. Mobile Handheld Digital Media Server (M-DMS) and Mobile Handheld

[3]This document considers that video also contains embedded audio

Home Network Devices	
Digital Media Server (DMS)	PCs and NAS devices
Digital Media Player (DMP)	TVs, home theater, stereos and game consoles
Digital Media Renderer (DMR)	Audio/video receivers (TVS, remote speakers)
Digital Media Controller (DMC)	PC
Digital Media Printer (DMPr)	Network printers
Mobile Handheld Devices	
Mobile Digital Media Server (M-DMS)	Mobile phones and portable music players
Mobile Digital Media Player (M-DMP)	Smartphones and tablets
Mobile Digital Media Controller (M-DMC)	Smartphones and tablets
Mobile Digital Media Uploader (M-DMU)	Digital cameras, mobile phones and tablets
Mobile Digital Media Downloader (M-DMD)	Portable music players
Home Infrastructure Devices	
Mobile Network Connectivity Function (M-NCF)	Routers and access points
Media Interoperability Unit (MIU)	Interoperability devices

Table 2.2: DLNA Device Classes.

Digital Media Player (M-DMP) are almost the same as DMS and DMP, but for both wired and wireless devices. The Mobile Handheld Digital Media Controller (M-DMC) can find content in both DMS and H-DMS, but as a DMC, it is only able to send it to DMRs. The Mobile Handheld Digital Media Uploader (M-DMU) and Mobile Handheld Digital Media Downloader (M-DMD) are responsible to, respectively, send or retrieve content into/from DMSs and M-DMSs.

- **Home Infrastructure Devices** are subdivided into two categories: the Mobile Network Connectivity Function (M-NCF) that can establish a bridge between Mobile Handheld Devices and the home network; and the Media Interoperability Unit (MIU) which is required to provide content transformation between required media formats for Home Network and Mobile Handheld Devices, as some devices are not able to play directly from the DMS or M-DMS.

2.1.1.3 DLNA Audio Codecs Supported

A brief description of audio codecs and containers supported by DLNA is presented in Table 2.3.

Codec/Format	Container	File Extension
LPCM	-	.wav
MPEG-1 Audio layer3	-	.mp3
WMA	ASF	.asf, .wma
AAC	MP4/3GP	.m4a, .m4b, .m4p, .m4v, .m4r, .3gp, .mp4, .aac
AMR	3GP	.amr, .3ga
ATRAC	-	.aa3, .oma, .at3

Table 2.3: Audio formats and codecs supported by DLNA.

A container holds the various components of a media file. For example, a video file can have multiple audio and video streams, subtitles and metadata along with the synchronization information needed to play the various streams together. For audio files, a container is much simpler than a video as it does not need to carry the video stream and other components, like the subtitles or synchronization information.

The term codec implies coder/decoder. A codec compress or decompress data, in order to make it possible to store and transmit files with a smaller file size. There are a lot of codecs available, each one with their own advantages and disadvantages. The biggest difference is between lossy and lossless formats. Lossless codecs are employed in cases where there is a need to keep all of the original information. But, when it is reasonable to lose some data, in exchange for greater compression, and therefore obtain smaller audio files, it is best to use a lossy format [13].

2.1.2 Universal Plug n Play

This subsection presents an overview of the UPnP standard. It details its Device Architecture, the Audio Video Architecture and discusses some security issues.

2.1.2.1 UPnP Device Architecture

The UPnP Device Architecture was the first International Standard promoted by the UPnP Forum [1]. It is part of Microsoft's IP-based home networking and device control protocol [14]. The concept of UPnP is that, upon the first utilization, no special device driver support is required. Instead, common network protocols are used.

There are only two types of devices in the UPnP Device Architecture: the controlled devices (or just "devices") and the control points. The controlled devices work as servers by responding to requests from control points. Devices and control points can work on a great variety of machines, including personal computers, smartphones, tablets and embedded systems.

The UPnP Protocol uses User Datagram Packet (UDP) (port 1900) due to its lower overhead. Configuration procedures like: addressing, discovering, descriptions, controls, events and presentation should be almost automatic with none or very few configurations entered by the user. Such configurations, that allow communication between devices, are described below.

0. **Addressing** is the step 0 of UPnP. Every device that does not implement a Dynamic Host Configuration Protocol (DHCP) server, must have a DHCP client enabled so it can search for a DHCP server when connecting to the network. If there isn't any DHCP server in the network or if the device does not get a response after a DHCP Request message, it has to be capable of assigning an address to itself - this process is also called AutoIP.
1. **Discovering** allows a device to advertise its services to the control points on the network. In contrast to **Addressing**, when a control point connects to the network, the UPnP discovery protocol also known as Simple Service Discovery Protocol, allows it to search for devices of interest within the network. The key exchange is a message that carries specific information about the devices and its services like type, identifier and a URL that contains more information.
2. **Description** provides the functionality of retrieving more detailed information about a device. As at this stage, a control point has very

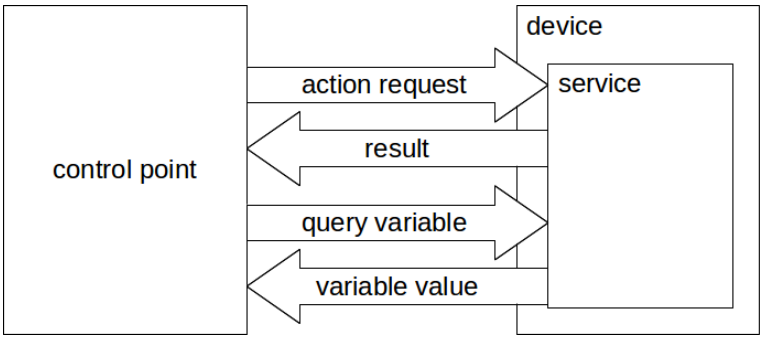


Figure 2.1: Actions and responses during the Control step (adapted from UPnP Device Architecture [1]).

little information about a device and thus, in order to interact with the control point, must learn more about it. A full device description and its capabilities are provided from the Uniform Resource Locator (URL) previously sent in the UPnP discovery message, as previously described. A device description also includes a list of variables corresponding to the service state at running time.

3. **Control**. When a control point already knows a device's service description and capabilities, it is able to invoke actions and receive back the responses with the correspondent data. Figure 2.1, shows that the invoked action is received by the device's service, which will reply back to the control point indicating if that action was successful or failed.

4. **Event Notification**. As previously introduced, a device contains a list of variables. The control point can subscribe to receive a notification when a certain variable changes. When there is more than one control point in the same network, this eventing service makes it possible to keep the interested control points informed about certain devices rather than all of them.

5. **Presentation**. This is when the control point, with a device's description previously retrieved, is ready to present it's list of variables. The control point retrieves a page from the device's URL, loading it into a browser, allowing the user to control the device or view its status, as it can be seen in Figure 2.2.

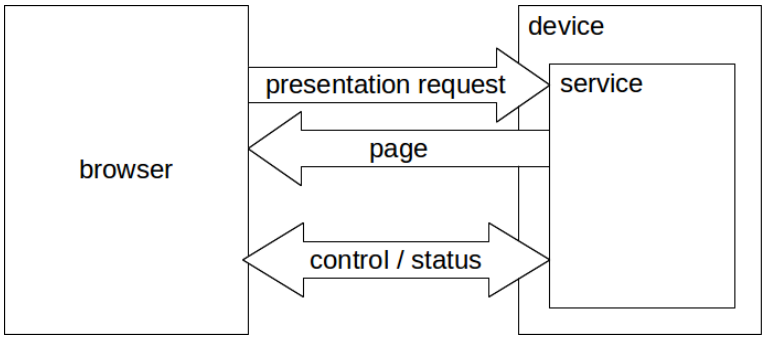


Figure 2.2: Presentation request and its response [1].

2.1.2.2 UPnP Audio Video Architecture

UPnP Audio Video (UPnP AV) Architecture is an audio and video extension of the UPnP which defines the general interaction between UPnP Control Points and UPnP AV Devices [2]. It supports a large variety of devices such as TVs, CD/DVD players, stereo systems, MP3 players, videocameras and PCs. The architecture allows those devices to support different types of formats for entertainment content, as well as different types of transfer protocols. The main goal of the UPnP AV Architecture is to enable audio and video content to flow directly among end-devices without any interaction of the Control Points and therefore saving resources on devices that can have limited memory and processing power.

Figure 2.3 depicts the three UPnP AV Components and its roles. The MediaServer holds content (local or remotely) that the user will browse and render in a MediaRenderer. Through the User Interface (UI) of a Control Point, the user can locate and select the desired content stored in a MediaServer and then choose in which MediaRenderer it will render/play that entertainment content. The type of content which a MediaRenderer can receive depends on the transfer protocols and data formats that both MediaServer and MediaRenderer support [2].

The MediaServer and the MediaRenderer do not control each other via UPnP actions. Instead, the Control Point uses UPnP actions to initialize the communication and configuration of the media end-devices. However, once a Control Point sets up the devices and triggers the flow of media content, it

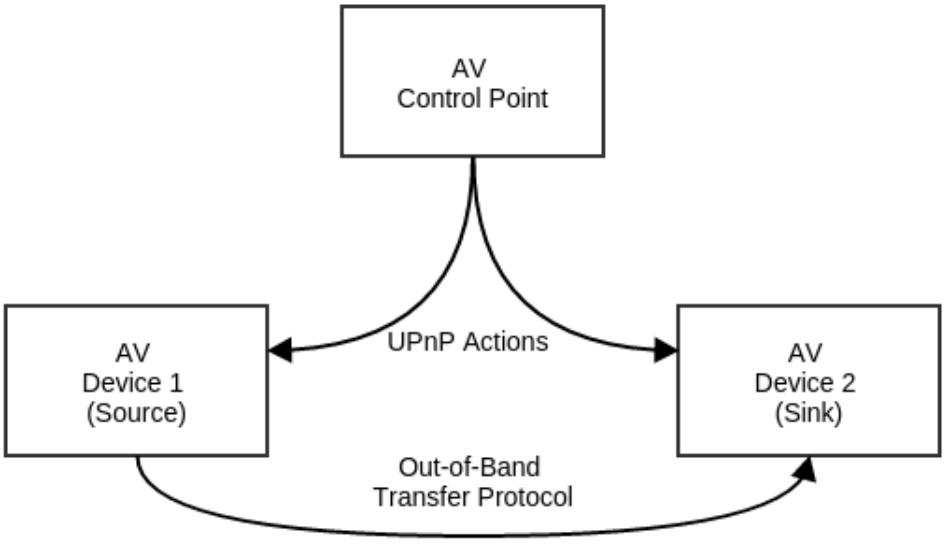


Figure 2.3: UPnP AV Device Interaction Model.

steps out of the communication. The content is transferred through an "out-of-band" protocol, as the data transmission does not need to flow through another device or machine besides the source (MediaServer) and destination (MediaRenderer).

A more detailed explanation about the UPnP AV Components and their main functions is given below.

MediaServer - The MediaServer holds the media content. It allows Control Points to browse and search for content items that are available for the user, like DVD Players, satellite/cable box receivers, TVs, stereo systems, PCs, etc. A MediaServer implements three services, ContentDirectory Service, ConnectionManagement Service and AVTransport Service.

The main function of the ContentDirectory Service is to allow the Control Points to browse for content stored in the MediaServers. This also retrieves detailed information about each media item in the server. The information is provided as meta-data, which includes fields such as: title, artist, date created, size, genre, etc. Additionally, the returned meta-data can inform

about the transfer protocols and data formats that are supported by the MediaServer for accessing a specific media item. The Control Point will use this information to determine if a given MediaRenderer is capable of receiving/rendering that content in its available formats.

The ConnectionManager Service has the role to prepare MediaServers for the connections and to manage them when associated to a particular device. The Control Points invoke an action in order to give the MediaServer an opportunity to prepare itself for an upcoming content transfer. Depending on the transfer protocol and the media data format chosen, the invocation may contain an InstantID (Instance Identifier) of an AVTransport Service allowing the Control Points to be able to control the flow of the content for actions like: start, stop, pause, resume, seek and volume change. This identifier is used to distinguish multiple instances of the AVTransport Service, allowing just one MediaServer to handle multiple MediaRenderers at the same time. When a Control Point wants to terminate a connection, it invokes a *ConnectionComplete* action in order to release it from the MediaServer and MediaRenderers.

The AVTransport Service is an optional service that is used by the Control Points to control the flow of media content. Depending on the supported transfer protocols and media data formats supported by the MediaServer, this service may contain control actions like: stop, pause, resume, seek, etc. The MediaServer may also support the access to multiple MediaRenderers at the same time.

Implementation of UPnP media servers are available for many operating system and hardware platforms and they can be software-based or hardware-based. Devices like PCs, tablets or smartphones can run software-based media servers, while a NAS device is hardware-based.

Table 2.4 shows a list officially supported and freely available software-based media servers. Foobar2000 [15], Jamcast [16] and Windows Media Connect are the only ones developed just for Windows platforms, while the rest of them are cross-platform. From those cross-platforms software media servers, XBox Media Center (XBMC) [17] currently stands out as one of the best known cross-platform Media Centers. It can play music, video and

Name	Platform	Audio	Images	Video	Transconding	Web Interface
TVMOBiLi	Cross-Platform	Yes	Yes	Yes	Yes	Yes
Foobar2000	Windows	Yes	No	No	Yes	No
Jamcast	Windows	Yes	Yes	No	Yes	No
PS3 Media Server	Cross-Platform	Yes	Yes	Yes	Yes	Yes
Universal Media	Cross-Platform	Yes	Yes	Yes	Yes	Yes
XBMC Media Center	Cross-Platform	Yes	Yes	Yes	No	Yes
Windows Media Connect	Windows	Yes	Yes	Yes	Yes	No

Table 2.4: Media Servers using UPnP.

images and allows the installation of many Add-ons that can provide contents like weather forecast, movie subtitles and live streaming. Universal Media Player [18], TVMOBiLi [19] and PS3[4] Media Server [20] are similar to XBMC, although they support transcoding and XBMC does not, which is needed in cases where the target device does not support a specific media format or has low storage capacity which requires a smaller file size in order to be played or rendered.

Foobar2000 is the only one that only supports audio files and the implementation of UPnP Media Server and DLNA are supported in recent versions as an official and third-party components, respectively.

MediaRenderer - A MediaRenderer is where the media content is going to be rendered/displayed or played. It allows the Control Points to control how the content can be rendered by changing parameters related to brightness, contrast, volume, mute, etc. A MediaRenderer may also allow the user to control the flow of the content (stop, pause, resume). MediaRenderers implement a RenderingControl Service, a ConnectionManager Service and an optional AVTransport Service.

The RenderingControl Service allows the Control Points to control how the MediaRenderers render certain contents by providing a set of actions, which may include some rendering characteristics like: brightness, contrast,

[4]Playstation 3

volume, etc.

The ConnectionManager Service, in the context of a MediaRenderer, has the primary function of getting the information about the transfer protocols and data formats that are supported by the MediaRenderer. With this information, the Control Points can determine if a given MediaRenderer is capable of rendering a specific media content. A MediaRenderer may also implement an optional action, which assigns a ConnectionID (Connection Identifier), so that a Third-Party Control Point retrieves information about the connections that the MediaRenderer is currently using. Depending on the specified transfer protocol and the data format being used, this service may also provide an AVTransport Instance ID and a Rendering Control InstanceID, from which the Control Points are able to perform various actions to control the flow of the media content and to control the rendering characteristics, as described above.

The AVTransport Service has similar functions as the AVTransport Service of a MediaServer. It is used by the Control Points to control the content's flow with actions like stop, pause, seek, etc.

Some devices can handle multiple items at the same time (e.g. an audio mixer like a Karaoke device). Thus, in order to support those kinds of devices, the RenderingControl and AVTransport Services must contain multiple independent instances of these services, where each instance is bound to an incoming connection. This allows the Control Point to control every incoming content connection separately.

Control Point - The Control Point coordinates the operations of the MediaServer and the MediaRenderer, typically controlled by an user through the Control Point's UI. A Control Point should implement a series of functionalities when interacting with the Media devices, as it is explained below:

1. Media Devices Discovery - It uses the UPnP's Discovering process to discover MediaServers and MediaRenderers in the network;
2. Locate Desired Content - It uses the browse/search actions from MediaServer's ContentDirectory to locate a desired media content. The

information returned includes the transfer protocols and data formats that must be supported for transfer and play media content.

3. Retrieve Renderer's Protocols and Formats - By using the ConnectionManager Service of the MediaRenderer, a Control Point retrieves information about the transfer protocols and data formats that a MediaRenderer supports;
4. Compare and Match Protocols and Formats - After the retrieval of the transfer protocols and data formats supported by the MediaServer and the MediaRenderer, the Control Point compares and selects a matching transfer protocol and data format supported by both devices;
5. Server and Renderer Configuration - The device's ConnectionManager service informs both the MediaServer and the MediaRenderer about an incoming or outgoing connection by using the specified protocol and data format that was previously selected by the Control Point. Depending on the transfer protocol, the MediaServer or the MediaRenderer will return an AVTransport InstanceID which will allow the control of the content's flow (see Section 2.1.2.2). Additionally, the MediaRenderer may also return a Rendering Control InstanceID that is used by the Control Point to adjust Rendering characteristics (see Section 2.1.2.2);
6. Content Selection - By using the AVTransport service, an action is invoked in order to identify the media content that needs to be transferred;
7. Start and Control of the Content Transfer - Invoking one of the transport control actions (play, pause, resume, stop, etc.) of the AVTransport service that is desired by the user;
8. Adjustment of Rendering Characteristics - Through the Rendering Control service, the user can adjust various rendering characteristics such as: brightness, contrast, volume, etc.;
9. Select Next Content - The AVTransport service provides actions that enable the identification of the next media content to be transferred from the Server to the Renderer. It is possible to repeat the previous

content as desired;

10. Cleanup Server and Renderer - When the session is terminated, the MediaServer and the MediaRenderer will invoke a *Connection Complete* action from the ConnectionManager service in order to close the connection.

Figure 2.4 shows the generic interaction sequence between a Control Point, a MediaServer and a MediaRenderer [2].

2.1.2.3 Security Problems With UPnP

The lack of concern by the UPnP Forum [21] regarding the implementation of security led the UPnP to end up with a set of risks that are not addressed in the original standard, which diminished the interest to support the architecture. The DLNA identified several scenarios that contributed to the urgency of developing a deployable framework for security in UPnP, including the Device Protection Service and the Device Security Service.

The Device Protection Service provides a set of mechanisms designed to support authentication and access control for UPnP Devices [22], while the Device Security Service supplies the services for strong authentication, authorization, replay prevention and privacy of UPnP SOAP (Simple Object Access Protocol) actions [23].

The main security concerns for UPnP are user authentication, content privacy and integrity protection. By default, the UPnP does not implement any kind of authentication, therefore, the UPnP Device Architecture needs to be complemented with an additional Device Protection Service and the Device Security Service.

Moreover, deployment over a LAN network stacks do not validate data. As UPnP was created for LAN environment, there are no functions to check if the IP is on the LAN, therefore allowing UPnP actions coming from the WAN, which contradicts the UPnP specification [24].

Device Protection and Device Security services implement privacy and integrity protection which are on top of Transport Layer Security (TLS) over HTTP Secure (HTTPs).

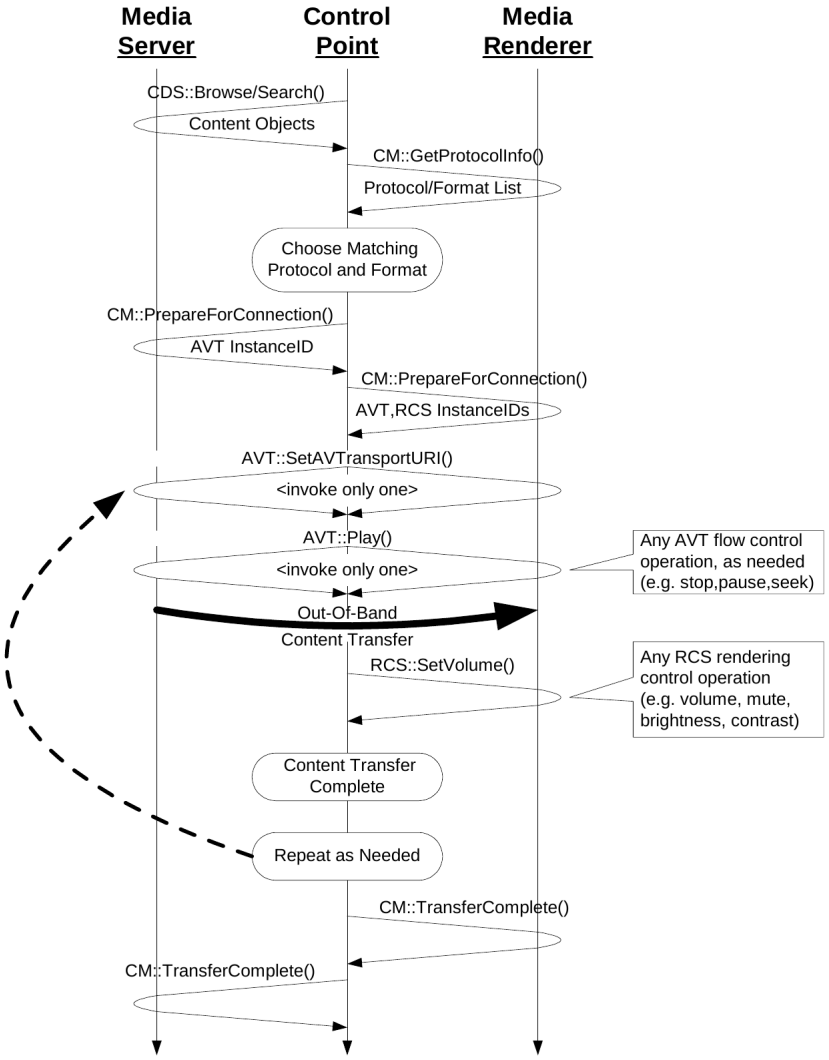


Figure 2.4: Generic Interaction diagram between Media Devices and Control Point [2].

Nevertheless, implementing Device Protection and Device Security Services does not necessarily means that a device will be secured against an attack. If a device exposes some resources to legacy Control Points, then an attacker can explore vulnerabilities present in those legacy services.

There is also the possibility of an user's Control Point being compromised with a running malware. Such malware could launch an attack against a protected UPnP Device by injecting malicious messages into the TLS channel previously established. As the protected UPnP device assumes that a Control Point is still legitimate, it cannot distinguish malicious messages from those originating from the Control Point.

The possibility of UPnP devices becoming compromised was one of the primary motivations for establishing security mechanisms for UPnP. If a given device is likely to be compromised, users are advised not to grant high privileges (e.g. administrative) to Control Points. Besides that, in order to reduce the risk of cross-site scripting attacks injecting messages, the UPnP Forum advises that devices and Control Points should use random port numbers [22, 23].

2.2 Digital Audio Access Protocol

The DAAP [4] was introduced by Apple as part of its iTunes software, which enables sharing of music libraries over a local network. This protocol and the Digital Photo Access Protocol (DPAP) rely on the Digital Media Access Protocol (DMAP) that is used by Apple in the iTunes, iPhoto and Remote software applications for sharing media over the home network.

The DAAP works as a specialized HTTP server, as it provides the capability not only to stream audio from one computer to another, but also, to retrieve a list of the host's playlists. This list is requested by the client in form of URLs and the server replies with data encapsulated in a flattened form of Extensible Markup Language (XML) file.

Along with the DAAP, there is also the Digital Audio Control Protocol (DACP). These protocols are used to control DAAP servers and to exchange information between them and the clients.

Like the DLNA makes use of UPnP, iTunes/DAAP uses the ZeroConf service, also known as Bonjour, to announce and discover DAAP media shares on a local network, using the Transmission Control Protocol (TCP) port 3689 by default, as it is explained in the following Section.

2.2.1 Bonjour

Bonjour [25] is from Apple's implementable open source standard ZeroConf Working Group, which is part of the Internet Engineering Task Force (IETF). It allows service providers, hardware manufacturers and programmers to easily support IP networks on theirs user applications. The Bonjour's Zero-configuration enables users not to worry about IP addressing or hostnaming. They are simply asked to choose what kind of network services they want to enable from a given list. This functionality is useful as applications can automatically detect services or other applications that they may interact with, making it possible to connect, communicate and exchange data without any user intervention.

Similarly to the UPnP Device Architecture, the Bonjour requirements and proposed solutions for zero-configuration networking over IP, cover addressing, naming and service discovery, which are explained below:

- **Addressing** self-assigns link-local addresses and it uses a range of reassigned addresses for the home network. The IPv6 specification already includes a self-assigned link-local address as part of the protocol, thus making it simpler and more reliable than IPv4, where it needs to pick up a random link-local address and test if that address is already in use or not.
- For **Naming**, Bonjour uses names instead of addresses, implementing Multicast Domain Name System (mDNS) to translate names-to-address (by sending DNS-format queries to a multicast address; no DNS server is required), as each device provides its own capabilities. In contrast to DNS host names, the mDNS only has significance on the home network.

- **Service Discovery** is used to discover DAAP services and enables applications to find all available instances of a particular type of service and maintains a list of named services and port numbers. Applications can resolve the service hostname to a list of IP addresses, as described earlier in **Naming**. This list of named services allows applications to keep a persistent list of available services. Bonjour is service-oriented, which means that queries are made according to the type of service needed and not dependent on the hosts who which provide them. Applications store service instance names, not addresses, allowing IP addresses, port numbers or even host names, to change and retaining the capability to connect to services. As a consequence, the user's experience should be more graceful and trouble-free.

2.2.2 Software using DAAP

DAAP has two versions implemented. The first one is supported by iTunes prior to version 7.0 and the other one afterwards this version. Apple did not make DAAP publicly available, instead they released it for third-party software licenses like SoundBridge[5]. In iTunes 4.2 Apple implemented an authentication method so that only iTunes clients could connect to iTunes servers. Later, in iTunes 4.5, Apple changed to another authentication method that used a custom hashing algorithm. However, both authentications were reversed-engineered, which allowed to DAAP non-Apple client software to connect to iTunes servers [26, 27]. From iTunes 7.0 Apple changed the method so that a certificate exchange is performed in order to calculate the hash sent in a *Client-DAAP-Validation* package header.

DAAP has been implemented in audio applications like Amarok [28] or Rhythmbox [29] and the already referenced XBMC Media Center [30]. Some players like Banshee, Songbird and Exaile require the installation of a plug-in in order to connect using DAAP [31].

Table 2.5 shows a brief comparison of audio players that implement

[5]Hardware device designed to play audio streaming across a local network

Software	Role	Supports iTunes	Platform
Amarok	Client/Server	No	Unix-like
Banshee	Client/Server[8]	No	Cross-platform
DAAP Client	Client	No	Android
Exaile	Client[8]	No	Unix-like
Firefly	Client/Server	No	Cross-platform
iTunes	Client/Server	Yes	Mac OS X/Windows
LimeWire	Server	No	Cross-platform
Rhymthbox	Client/Server[8]	No	Cross-platform
Songbird	Client[8]	No	Cross-platform
SoundBridge/ Roku	Client	Yes	Dedicated hardware device
Tangerine	Server	No	Cross-platform
XBMC Media Center	Client	No	Cross-platform

Table 2.5: Audio Players using DAAP.

DAAP. It is possible to verify that just SoundBridge/Roku[6], a part of Apple's iTunes, is capable of supporting earlier versions of iTunes with DAAP. That is due to the fact that Apple has only licensed the DAAP to Roku [32, 33]. However, since January 2012, Soundbridge is no longer available from Roku, as they stopped manufacturing SoundBridge hardware and only continue supporting Pinnacle-branded hardware, which was not licensed by Apple [34]. iTunes stands out by being the only software that was made for both Mac OS X and Windows platforms, since the majority of the other software parties were released under the GNU General Public License[7] (GNU GPL), thus there is little or no interest of those parties to make this kind of software restrictive to certain platforms.

An increasing number of mobile devices, such as the ones with Android and iOS[9], are including DAAP clients.

[6]Roku Inc. - private company that manufactures home digital media players
[7]Most widely used free software license
[8]With a plug-in
[9]iPhone OS - Apple's mobile operating system

2.3 DLNA and DAAP Comparison

This section provides a brief comparison study between the two technologies previously introduced, DLNA and DAAP.

DLNA is similar to UPnP AV but with added restrictions, as it only shares certain image, video and audio files, which contrasts with UPnP servers, as they can share any kind of media file just like any HTTP server can do. For a file or media item to be totally DLNA compliant, it needs to fulfill the requirements of DLNA profiles. This means that it does not just need to comply with the codec but also with its container and media properties like bitrate and resolution. For example, although a MPEG Audio Layer III (MP3) audio format is supported by DLNA, it can not be played or streamed if it does not match a specific bitrate.

As presented on Table 2.6, both technologies support MP3 and AAC audio codecs, which are present in many digital audio files, allowing most of them to be able to play and stream through DLNA or DAAP. However, both lack support to open source or lossless codecs like Ogg Vorbis or FLAC.

	DLNA	**DAAP**
Ports	UDP 1900	TCP 3689
Audio Codecs	LCM (.wav), MP3, WMA and AAC	AAC, ALAC[10], AIFF[11], MP3 and WAV
Discovery Service and Self-configuration	UPnP Device and UPnP AV Architectures	ZeroConf/Bonjour
Security	Device Protection and Device Security Services	Authentication, copy-right protections
Number of users	Unlimited	Limited
Compatibility	Aims to be universal	Only between Apple's devices

Table 2.6: Comparing DLNA and DAAP.

[10] Apple Lossless Audio Codec
[11] Audio Interchange File Format

The UPnP Forum released Device Protection and Device Security Services specifications, which cover authentication, privacy and integrity protection to UPnP devices, but there are still some security issues that should be taken care of. On the other hand, Apple implements authentication methods since 2003 on iTunes version 4.2. Although security problems of DAAP/Bonjour are not frequent, there were some reports concerning the installation of spyware[12] with faked iTunes updates on some iTunes versions [35] and the increased Bonjour's discovery traffic in a subnet with Apple clients. Another issue is that, by default, Bonjour has the discovery service (sharing service) enabled, therefore announcing their existence. Thus, a simple user can browse a list of nearby computers and its shared files [36].

Another downside for Apple's DAAP is that recent versions of iTunes limit the number of clients to 5 unique IP addresses [37], while DLNA can have an unlimited number of clients and connections.

Regarding compatibility, Apple focus on its own ecosystem, meaning that, sharing is only possible among iOS devices and OS X devices, making an exception with iTunes version for Windows platforms and Soundbridge/Roku hardware. By contrast, DLNA compatibility aims to be universal, as it results from a consortium of electronic manufacturers and software makers like Erickson, Intel, Microsoft, etc.

As to the service discovery, Bonjour and UPnP use similar techniques, relying on solutions for self-configuration, in particular addressing and service discovery. UPnP aims to be universal and supporting more than just media files. It defines an architecture where there are devices that play/render the media content, while some devices can store it and other devices can control the previous two. While on the other hand, Apple's Bonjour comes built-in with OS X and iOS operating systems and also within iTunes and Safari software (for Windows platforms), wherein any device can either be the server or the renderer. This fact makes DAAP/Bonjour more simpler to deploy than UPnP.

With these solutions for zero-configuration networking, users no longer

[12]Malicious software that monitors and collects data on a computer

need to worry about IP addressing or host name assignment in order to access services on the network. This kind of services also enables applications to automatically detect services they need or may interact with, allowing automatic connection establishment, communication and data exchange, without any user intervention.

The purpose of DLNA and DAAP is to simplify user experience, although the protocols themselves, may contain some complexability.

2.4 Simple Network Management Protocol

The SNMP [9] lies in the Application layer of the OSI model for managing devices on IP networks, making it easy to exchange management information among equipments on the network. The purpose of SNMP is to manage and monitor network equipments performance and usage, detect faults and configure. All of enterprise network brand devices (e.g. Cisco, HP and Juniper) come already with SNMP capabilities implemented. As for home network brand devices, the majority are also SNMP-capable but usually, only for monitoring functionalities.

The protocol was built to be deployed on the largest possible number of network devices, so it needs to follow some principles, like consume minimal resources (e.g. CPU, RAM), and have minimal transport requirements in order to consume minimum bandwidth while transfer information between management entities. Furthermore, it should be easier to implement, configure and be resilient, so it can still work when most of other network applications fail.

This section introduces the SNMP in more detail, its operations and a brief description of its released versions.

2.4.1 SNMP Concepts

Due to its operation commands, the SNMP may not follow the regular concept of client-server model, as the managed elements on the network can behave both as client or server. As such, the terms **Agent** and **Manager** are respectively used to designate the managed device and the device that manages.

Figure 2.5 shows the three SNMP key components: the managed devices, the agent (software that runs on the managed device) and the Network Management Station (NMS - software that runs on the manager) [38].

The manager access information that is controlled by the agent using a database module by a MIB definition.

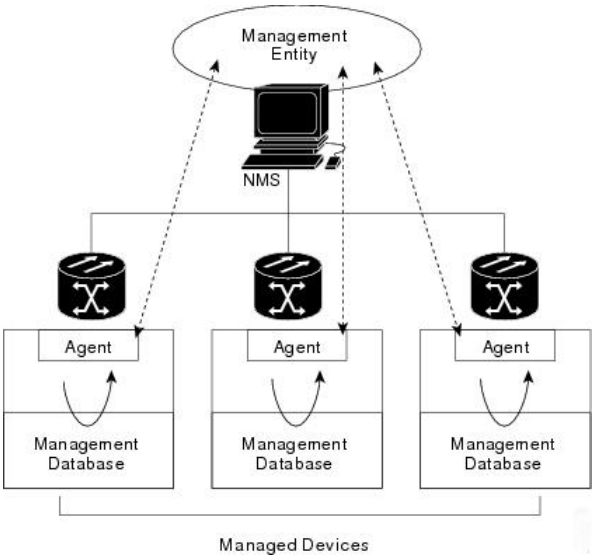


Figure 2.5: The SNMP key components and their relationships.

2.4.2 Management Information Base

SNMP agents provide access to management data information on the devices as variables (or management objects). Figure 2.6 shows that such variables are organized in a tree-structured hierarchical database. MIBs objects are defined using Structure of Management Information (SMI), which is a subset of the Abstract Syntax Notation (ASN.1), standard used to describe data structures to be transferred between the Application Layer and the Presentation Layer of the OSI model in telecommunications and computer networking [39, 40].

Each object entry in a MIB is addressed through an *object identifier* (OID). The managed objects can either be scalar (single object instance) or sequential tabular where multiple related object instances are grouped in sequence tables. SMI specifies the allowed data types and divides them into two major categories:

- Simple Data Types:
  - Integer;
  - Octet string;

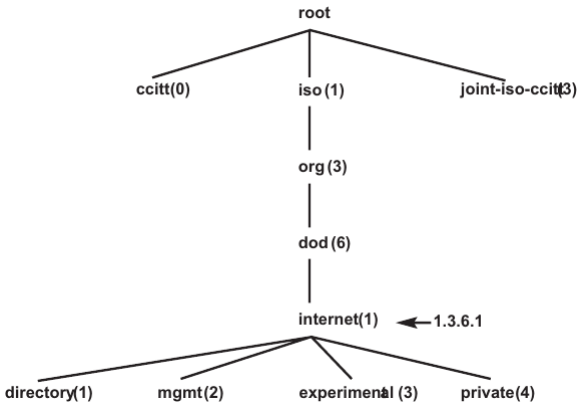


Figure 2.6: Branch of a MIB Object Identifier tree.

  - Object ID.

- Application-wide Data Types:

  - Network Address;
  - Counter, non-negative *integer* that increase until they reach the maximum value and then restart again from zero;
  - Gauge, like *counter* but can also decrease;
  - Time ticks, represent the time since an event has occurred;
  - Opaque, arbitrary encoding that is used to pass arbitrary information strings that do not conform to the strict data typing used by SMI.

A MIB should not be mistaken as a normal database. Unlike most types of databases, a MIB does not offer the capability to interpret queries, as it just provides a vision of the paradigm, which is called instrumentation.

Each new MIB has to be defined in under private (OID: 1.3.6.1.4.1) or experimental (OID: 1.3.6.1.3) subtrees, which are assigned and controlled by the Internet Assigned Numbers Authority (IANA).

2.4.3 SNMP Operations

SNMP has a simple way of exchanging information and defines three major operations: GET, SET and TRAP/NOTIFICATION. The operation request for the GET and the SET originates from the application that controls or manages the network and aims for the UDP port (161 by default) of an agent, while the TRAP/NOTIFICATION operation is originated in the managed device/network equipment that transmits the information to the controller (using UDP port 162 by default).

Manager's requests retrieve or modify variable/object stored in an agent's MIB:

- GET - This request is sent from the manager towards the agent in order to retrieve one or more variable values;
- GET-NEXT - Similar to the GET requests but retrieves the value of the variable with the next OID in the MIB tree;
- GET-BULK - Used to retrieve larger amounts of data from MIB tables;
- SET - This operation modifies/updates multiple variables values in a MIB.

An agent can send:

- TRAP/NOTIFICATION - a message to warn the manager about the occurrence of a predefined event;
- RESPONSE - used to carry the values requested by the manager.

From manager to manager:

- INFORM - similar to a TRAP but requires a confirmation from the manager upon receiving the message;

2.4.4 SNMP Versions

The first version of the protocol was released in 1988 and was defined in RFC 1157 [9]. This version was criticized for its poor security functionalities,

as the authentication of clients was performed by a "community string" that acted like a password and was transmitted in clear text.

SNMPv2 was defined in RFC 1441 [41]. It includes improvements in the areas of performance, security, confidentiality, and new features like: manager-to-manager communications, packet types, and transport mappings. The GET-BULK request was introduced as an alternative to interactive GET-NEXT requests for retrieving large amounts of data from MIB tables. However, the new party-based security system in this second version was not widely accepted as it was considered too complex, and so, two variants of SNMPv2 were released: SNMPv2c and SNMPv2u.

Community-based SNMPv2 or SNMPv2c, is SNMPv2 without any new security mechanism. Instead, it uses the same community-base system from SNMPv1. This variant of SNMPv2 is defined in RFC 1901 [42].

User-based SNMPv2 or SNMPv2u was defined in RFC 1909 and RFC 1910, with better security than SNMPv1 but not as complex as the original SNMPv2 [43].

The third version of SNMP, defined in various RFCs (from 3410 to 3415), did not get any new relevant improvements but defined all security mechanisms created for SNMPv2 as mandatory, which include: strong authentication (ensuring that the message comes from a valid source), integrity in order to guarantee that a packet hasn't been intercepted and forged while in transit, and confidentiality by encryption of packets for privacy. This version also added new textual conventions, concepts and terminology [44].

The use of the SNMP provides an integrated management architecture regardless the device manufacturer. A good network management system is important to whom has the responsibility to plan, deploy and manage the network and its devices.

Therefore, SNMP seemed a suitable tool to manage home automation systems, like intrusion detection, air conditioning, lighting/windows controllers, gas/smoke detection and media systems.

2.4.5 Home Automation Systems using SNMP

SNMP is already being used in a few home automation systems. A solution provided with Moxa devices [45] connected to different sensors (doorbell, mailbox, air conditioning, lighting control, fire sprinkler, etc.), using SNMP in order to monitor and control them. A Moxa server is then connected to a control panel device, which the user is able to configure and monitor the various sensors status connected through the home via SNMP and receive alarm messages with SNMP Traps.

Another solution relies on SNMP for consumption management of electric power [46]. The consumption readings of electricity is provided from a meter/converter with an ethernet port. These readings are then available in a MIB from which the client application can access in order to present them to the user.

The ease of integration, configuration and use, makes the utilization of SNMP an advantage in home automation systems.

2.5 Streaming

Streaming applies when there is a need to transport multimedia content (audio, video or images) between a source (server) and one or more destinations (clients). In general, does not require disk space resources as the content is played while it is being received. Through the use of streaming it is possible to enjoy listening an audio file without having to wait for the file to completely downloading. It requires, in a certain way, a good network bandwidth with low delay and small jitter, so the stream can be played without dropouts or media breaks (otherwise a delay will be experienced and large buffers must be used).

Streaming can be achieved using some standard network transport protocols:

- The User Datagram Protocol (UDP) uses simple connectionless transmission with a minimum of protocol mechanisms. This means that there is no guarantee that the packets are successfully delivered and,

therefore, if some packets are lost some dropouts may occur to the stream [47];

- The RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Applications usually run RTP on top of UDP to make use of its multiplexing and checksum services. RTP, also supports data transfer to multiple destinations by using multicast distribution [48];
- As for the Real Time Streaming Protocol (RTSP), it is designed to control streaming media servers, by establishing and controlling media sessions between end points, acting like a "network remote control" for multimedia servers. As the transmission of streaming data itself is not a task of the RTSP protocol, most of RTSP servers use the RTP with the Real-time Control Protocol (RTCP[13]) for media stream delivery [49].

2.6 Open Source Audio Codecs

In the scope of this project there was the need to approach audio codecs, especially, open source audio codecs. As stated in Section 2.3, the current software for media sharing over a network (DAAP and DLNA) do not support any open source audio codecs and DAAP only supports one lossless format (ALAC) .

From current open source audio codecs, some are lossy and others are lossless. The Ogg Vorbis and the FLAC codecs are two of the most known and used open source formats.

The next section briefly describes these two codecs, which are supported in this project.

[13]RTCP provides out-of-band statistics and control information for an RTP session

2.6.1 Ogg Vorbis

Ogg Vorbis [50] is an open source method of encoding, compressing and streaming digital audio, identical but arguably technologically superior to MP3 or AAC. The success key of MP3 was its ability to reduce the size of uncompressed CD[14]-Quality audio file up to a tenth of its original size and barely without perception of audio quality loss (at least at the highest bit rates). MP3 brought ease of sharing music files through the web, due to the fact that it was now possible to download or share music files with good sound quality and small file sizes, when the network and Internet connections were relatively poor, in comparison to the speeds and bandwidth of nowadays connections [51].

MP3 mainly uses constant bitrates while on the other hand, Vorbis uses variable bitrates by default, allowing the quality to stay constant. As such, an Ogg Vorbis encoded audio file should sound better as it has more detailed highs, solid lows, acoustic ambiance and it is generally clearer than MP3 converted at the same bit rate. This means that a very acceptable sound quality could be streamed over the Internet even at low bitrates/bandwidth (e.g. dial-up) using Vorbis.

However, if an audio file is transcoded from MP3 to Ogg Vorbis, the outcome will be a file with lower sound quality than the original MP3. So it is recommended to encode Ogg Vorbis directly from the original or uncompressed source that has never been compressed before (e.g. CD or an analog recording encoded in wave format).

The Ogg Vorbis codec is nowadays widely used. Many video games have in-game audio in Ogg Vorbis format like: Minecraft, Unreal Tournament, Grand Theft Auto: San Andreas, Guitar Hero, Eve Online or World of Warcraft. The codec is also supported by many music player softwares, websites and audio streaming services, like Spotify [52].

Ogg Vorbis's higher fidelity, being completely free, open and unpatented, makes it a good audio format replacement for patented and restricted formats.

[14]Compact Disc (PCM 16/24, 44.1KHz sampling frequency)

2.6.2 Free Lossless Audio Codec (FLAC)

FLAC [53] is the fastest, non-proprietary and most widely supported lossless audio codec. It has an open source implementation and a well-documented format and Application Programming Interface(API).

Lossless means that audio is compressed and no loss in sound quality is injected by the compression mechanisms. Kind of similar to how ZIP[15] works, but with the peculiarity that FLAC was designed specifically for audio. It can compress a PCM audio file by 40-50% of its original size. It allows to directly playback compressed audio files just like any other audio codec (e.g. MP3, AAC, Ogg Vorbis) would do.

FLAC's libraries use a compression level parameter that can vary between 0 (fastest) and 8 (smallest). If the parameter has the lowest value, the compression will be fast but the outcome compressed file will also have a higher size. Comparatively, if the parameter is set to the higher value, the compression will take longer, but it will result in a smaller file. The compression process involves a trade-off between speed and size. However, the decoding process is usually quick and not dependent on the compression level.

This format is convenient as an archive format for media owners that wish to preserve their audio collections without loss of a digital quality. A FLAC copy ensures that an exact duplicate of the original digital audio track can be recovered, which does not happen with a lossy archive format, as a restoration from it, is impossible. FLAC by being lossless, also gives the possibility to transcode to another format, like MP3 or Ogg. FLAC support compared to lossy formats, is somewhat limited, specially in portable audio players. On the other hand, it is currently better supported than the competing lossless formats. FLAC also has a further advantage, which is the ability to be streamed and quickly decoded, independently of the compression level used, which enhances user experience.

[15] Archive file format that supports lossless data compression

Chapter 3

Audio Distribution Using Open Source Protocols and Codecs

In this chapter is presented the project motivation, the adopted solution and a description of architecture and functionalities of the system modules.

3.1 Motivation

As seen in the previous chapter, Apple's DAAP and Sony's DLNA use different communication protocols and mechanisms, either for auto addressing, device discovery and media content access. DLNA mainly relies in the UPnP standard and aims to be universally compatible. On the other hand, Apple's software uses proprietary technologies and is to be used on the Apple ecosystem, allowing media sharing only from an Apple's device (iOS or iMac[1]). DLNA technology results from a partnership of many electronic manufacturers in contrary to DAAP, which is only compatible with the newer versions of Apple TV and with some audio manufacturers.

DLNA and DAAP cover a large set of protocols that support many technologies like video, audio and access to image folders. Thus, their implementation requires the usage of resources which are often misfits to more restrict and specific purposes, like distribution of audio streams over generic

[1] Apple's desktop computer

IP networks or home automation applications contexts.

In order to have a more universal and compatible inter-operability between devices, some of these technologies have complex architectures integrating a lot of concepts, which may be another obstacle, mainly if good documentation support is not provided.

UPnP is a complex architecture with many device types. This can hamper the development, implementation and deployment of solutions for the end user. DLNA, for instance, has some documentation available but it is required a payment in order to access DLNA entire documentation guidelines. So, although this approach integrates some universal technologies it still works much like a proprietary solution.

As for the DAAP, it is very easy for the end user to setup the devices and use Apple's software/devices. However, like other proprietary technologies, it is required to pay licenses and/or royalties in order to incorporate the DAAP into specific hardware or into an application.

With this in mind, it seemed obvious the advantages of developing a specific architecture for a more restricted context and entirely based on open source technologies, standardized mechanisms, protocols and audio codecs. Thus, the aim of the project was to define such an architecture and develop a modular audio distribution system with similar capabilities that could be implemented on top of common IP networks. More specifically, the proposed solution is based on the internet standard network management protocol in order to attain, not only communication between their entities but also an abstract interface syntax to manage audio file collections. It will also allow the use of open source mechanisms and technologies for audio streaming from the server to the playing devices.

3.2 Architecture

While this architecture's solution presents four main devices, DLNA defines many device types which makes it more complex to implement and deploy. On the other hand, DAAP only defines two types of devices but does

not provide any solutions for support of external audio file databases and audio server independency.

Solutions like DLNA and DAAP define architectures where the software and hardware are extremely bonded and very hard to separate one from another, thus requiring the payment of licenses or copyrights in order to use the software in another hardware. In some other cases, the hardware already has embedded software that is not possible to modify or even substitute with other third party applications.

Defining a totally open solution implies that its technologies should be free to implement (with complete access to all needed documentation) and its architecture should be free to deploy. Furthermore, solutions could integrate modules created by different developers, freely or commercially available, with no compatibility issues.

Despite that, UPnP and Bonjour are able to be used by any developer on any other solution, they are only a part of the components needed and, thus, requiring additional modules and protocols which will increase the complexity of the system. Proprietary solutions tend to be updated less frequently and seldom providing relevant new features.

As shown in Figure 3.1, the proposed model has the following main components:

- Controller Application - this module has the functionality of managing information on the Music Server, deploying the client application to the final user and controlling music playback on the Playing Devices;
- Music Server - this component holds the metadata information (implemented on a MIB database) that will be accessed by the Controller Applications and initiates the indirect audio streaming server; The Music Server is able to manipulate communication with multiple Audio File Databases through a Gateway Module;
- Audio File Database - the database systems that hold the music file collections;
- Playing Devices - the devices that actually play the music for the user.

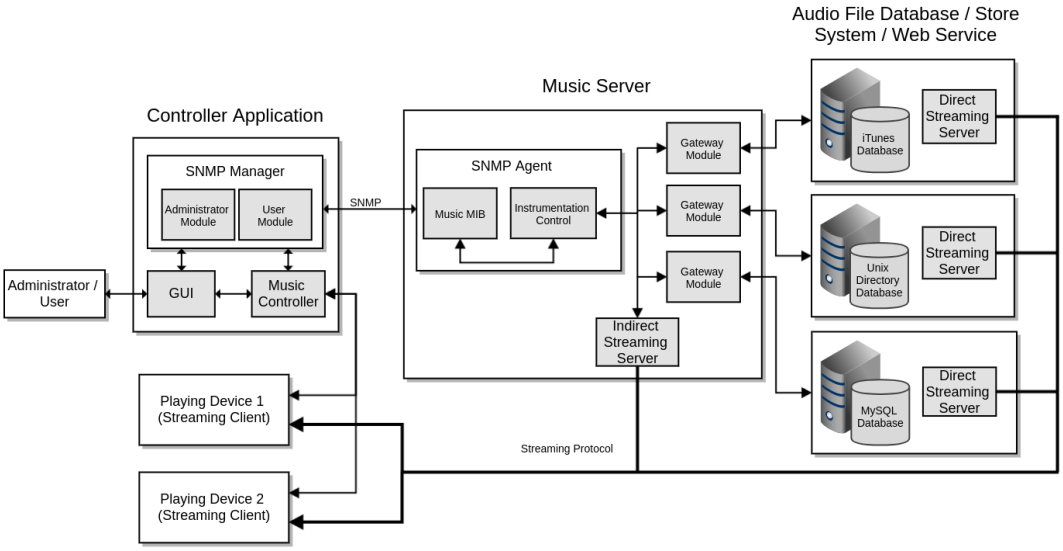


Figure 3.1: Architecture diagram.

Control Applications access music metadata information via SNMP and use it to implement all functionalities available to the users. Within the architecture, several modules implementing a Controller Application can coexist. Also, a Controller Application can control multiple Music Servers.

3.2.1 Music Server

The Music Server includes an SNMP Agent implementing a Music MIB database which objects represent metadata information from music collections stored on Audio File Databases, as well as other server system's information.

The SNMP Agent has a high-level Instrumentation Control Module which implements the methods to transform the abstract semantics of the Music MIB objects into generic mid-level operations that access and control music collections. The best approach to this transformation (or mapping) is when these operations are not dependent on a particular Audio File Database technology. This allows the implementation of the SNMP Agent Module to be independent of any Audio File Database technology. On the other hand,

for each particular Audio File Database technology to be supported by the Music Server, a Gateway Module must be implemented, which finally maps the generic mid-level operations into specific technologies supported by the external music database servers (the systems that really store/manage the audio files of the music collection).

On first initialization, the Music Server, through the Instrumentation Control and the Gateway Module, will load all music files metadata information from the Audio File Databases into the relevant MIB objects. This information is then available to be accessed and controlled by the Controller Application through SNMP. With time, the data on the MIB will be updated with new information gathered from the Audio File Database systems.

One of the most important operations that the Music Server supports is the launching and control of streaming audio files from streaming servers (either internal or external). Every time a user requests to play an audio file from the music collection in one of his Playing Devices, the Gateway Module commands the Audio File Database system containing the respective file to stream it to the destination Playing Device. If the Audio File Database system does not support direct streaming, then the Gateway Module will request the raw file from the Audio File Database system and will use an internal audio streaming server (implemented inside the Music Server). In this case, to guarantee protection of copyrights, the raw file must not be saved to any internal non-volatile file system in the Music Server.

3.2.1.1 Music MIB

The Music MIB specification was defined accordingly to the SMIv2 standard and appended under the experimental subtree (OID: 1.3.6.1.3). The MIB structure, including all tabular (sequences) and scalars objects, is presented in Appendix A.

The Music MIB objects either represents specific metadata information about all music tracks in the collections (artist, album, genre, etc.) or more general information about system resources and application users (users accounts, Playing Devices, usage statistics, etc.).

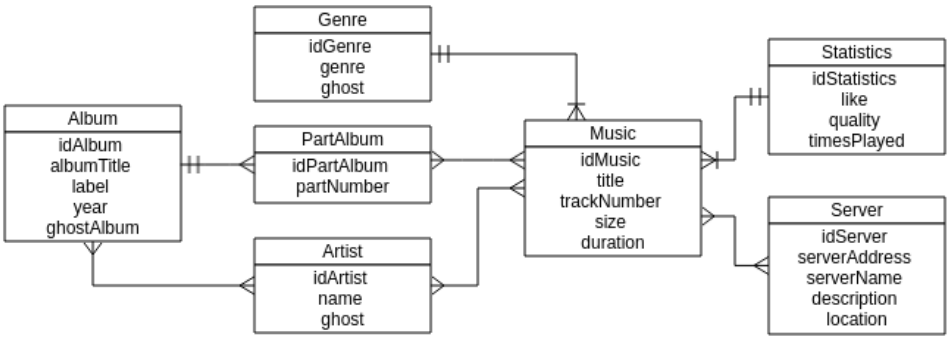


Figure 3.2: A Part of the Music MIB in Entity-Relationship Model.

As an example, the Album table with an index *idAlbum* and the columnar objects *albumTitle*, *label*, *year* and *ghostAlbum* is depicted in Figure 3.2 as part of an Entity-Relationship (ER) model. This means that every *idAlbum* uniquely identifies a set of metadata values for one album with a particular *albumTitle*. With this abstraction, an album title does not need to be unique, allowing different music albums on a music collection to have the same title.

For mapping the ER model into a SMIv2 specification, the Album table (OID 1.3.6.1.3.2.1.2) contains the following columnar objects:

albumTable.entry.1 – idAlbum (integer, table key)

albumTable.entry.2 – albumTitle (string)

albumTable.entry.3 – label (string)

albumTable.entry.4 – year (integer)

albumTable.entry.5 – ghostAlbum (integer)

As in a conventional database, MIB tables may have relationships between them. As the index of a table behaves as a primary key in an ER model, it is necessary to define a relationship by cross-reference the indexes (primary keys). There are two different kinds of relationships as they define how MIB tables are related: Many-to-Many relationship (N:N) and One-To-Many relationship (1:N).

Many-to-Many Relationship (N:N)

The Artist table (OID 1.3.6.1.3.2.2.2) has a relationship of many-to-many with the Album table, meaning that an album can be owned by more than just one artist and an artist can own/release more than one album. It contains the following columnar objects:

artistTable.entry.1 – idArtist (integer, table key)

artistTable.entry.2 – artistName (string)

artistTable.entry.3 – ghostArtist (integer)

In order for the Album and Artist tables to be related it was created an auxiliary table that holds the primary keys (now called foreign keys) of both tables. This auxiliary table, the AlbumArtist table (OID 1.3.6.1.3.2.23.2), contains the following columnar objects:

albumArtistTable.entry.1 – idAlbumArtist (integer, table key)

albumArtistTable.entry.2 – albumIDalbumArtist (string, foreign key)

albumArtistTable.entry.3 – artistIDalbumArtist (string, foreign key)

One-To-Many Relationship (1:N)

An album can have more than one part or CD, but a CD is associated with just one album so this kind of relationship is one-to-many. In this situation it is the primary key of the table with more unique values that will be inserted on the other table as a foreign key. Therefore, the PartAlbum table (OID 1.3.6.1.3.2.14.2) contains the following columnar objects:

partAlbumTable.entry.1– idPartAlbum (integer, table key)

partAlbumTable.entry.2 – partNumber (integer)

partAlbumTable.entry.3– albumID (integer, foreign key)

Tables like Music, Artist, Genre, etc. will contain object values from the metadata information embedded into the Audio File Databases and fetched through the Gateway Modules. On the other hand, tables like Server, Statistics, User, etc. contain object values implemented directly in the instrumentation module running on the agent's system.

There is no standard methodology to create a MIB's definition, but it is

important to design and make a plan beforehand, as the objects that will make part of it should maintain the same level of abstraction and style of organization. If there is a table or object that would be needed to be added later on (due to architectural changes or optimization purposes), this will be time consuming as it will require redefinition of the MIB specification and modification of the SNMP Agent instrumentation source code.

3.2.2 Controller Application

As the Controller Application is the component that allows the user or the administrator to interact with the system, it should support many functionalities in order to allow a proper utilization or management. The implementation of those functionalities relies on the manipulation of the Music MIB on the Music Server.

The Controller Application is composed by a Music Controller Module, a Graphical User Interface (GUI) and an SNMP Manager which includes the Administration Module and an User Module.

The user module should permit to:

- List or browse music tracks according to search criteria (e.g. a simple search or an advanced search with a few fields for input);
- List or browse the user's playlists and liked/starred music tracks;
- List or browse playable devices on the network and select music tracks to play on them, independently;
- To manage (add or remove music tracks, etc.) of user's playlists;
- Play, stop and pause a music track;
- Show statistics of a track for a given user;
- Show user's preferences page (user options, e.g. volume, password);
- Etc.

The administrator module should permit to:

- Add, remove music tracks;

- Manage music collections (name and rename, define music genres, etc.);
- Define operation parameters for gateways, servers and devices;
- Manage user accounts;
- Clear/reset statistical data;
- Edit music tracks metadata information (artist name, genre, etc.);
- Etc.

The GUI is responsible to present the information retrieved, as well as allowing the user to control music playback through the Music Controller Module. Audio streaming should be initiated by the Gateway Module on the Music Server.

3.2.3 Audio File Database

An Audio File Database can be an external source like the iTunes service or a specific a MySQL database or just a simpler hierarchical structured directory system (for example, a structure directory for every artist, containing sub-directories with their respective albums and, in turn, each sub-directory of each album containing the respective music tracks). This component needs to provide any sort of access technology to a Gateway Module inside the Music Server.

The Gateway Module must take into consideration that the communication with the Instrumentation Control Module must be independent of the Audio File Database technologies used. The Gateway Modules should ensure compatibily between the Instrumentation Control Module and the external music sources.

3.2.4 Playing Devices

The Playing Devices components are controllable by the Controller Applications and are responsible for playing the audio stream. These components

can be integrated on the same system as the Controller Applications or in an external device on the same local network.

3.3 Functionalities

Both DLNA and DAAP support an addressing and discovery service by means of UPnP and Bonjour. In contrast, the proposed model takes for granted that the devices are already connected to the local IP network, preconfigured and ready to start communicating with each other, therefore reducing complexity, as there are already existing standard TCP/IP protocols and services (like DHCP and DNS that can take care of these issues).

As for discovery, Controller Applications, Music Server and Playing Devices should be able to find themselves through an "Hello" message which is sent frequently. The "Hello" message carries the IP address, application port and type of the component, announcing it to the other components in the same local network. This "Hello" message does not need to be generated if each of the server components (Music Server, Audio File Databases and Playing Devices) would register as services into one or more DNS servers.

As for playback actions such as play, stop, pause, changing volume, etc., the proposed solution integrates these operations into the semantics of the Music MIB objects allowing total control through SNMP messages. On the other hand, DLNA and DAAP use HTTP in order to carry a structured message (SOAP or XML), which can hold unnecessary information for such simple actions.

Other more advanced features, such as searching audio files with criteria or filters are also not available by default on DLNA and DAAP, therefore requiring the applications that run on top of these technologies to support and implement it. In some cases, it is needed to install an add-on in order to have that kind of feature available, while in our solution its support this is already built in the Music MIB definition.

Regarding audio codecs, while DLNA and DAAP make use of the most used audio codecs such as MP3, AAC and WAV, they lack support for other important open sources audio codecs (like FLAC and OGG), which are known

to be technologically more efficient for audio streaming on local IP networks in comparison to the existing ones that are usually used in streaming. The proposed solution makes no restrictions on audio codecs, thus allowing support for all audio codecs.

3.4 Security

As for security, an SNMP-based solution can implement authentication and encryption (SNMPv3), while the other two referred mainstream solutions have their own mechanisms and procedures to implement some security guarantees like authentication and copyright protection.

However, in the context of a home network environment, it does not make much sense to have some of these security constrains and thus making the system even more complex. For example, this proposed design has no direct support for copyright protection systems. Nevertheless, the architecture does not promote or allow audio files duplication and it is granted that the user has legally acquired the right to access the audio files on the Audio File Databases.

Chapter 4

Prototype Implementation

Based on the architecture, which was presented on the previous chapter, a prototype system, depicted in Figure 4.1, was planned, developed and tested. In order to implement a complete prototype system that could be considered a valid and relevant proof of concept, all of the following components were included: a Controller Application, a Music Server, one Playing Device and one Audio File Database.

The core communication technology adopted in the prototype was the SNMPv2c and a complete Music MIB was defined in SMIv2, which was then implemented in the SNMP agent module on the Music Server. More specifically, for the development of the SNMP modules (agent and manager) it was used the SNMP4J Java based API [54]. This widely used open source programming library contains implementations of SNMP primitives that are required for the interaction between agents and managers on Java platforms. For the creation of Music MIB two additional tools were used: the MibDesigner [55], MIB graphical editor that helps on the creation of MIBs and provides syntax and semantic verification, and the AgenPro [56], which generates a Java or C++ based source code from a given MIB definition file. This source code can then be used as the basis for programming of the agent's MIB instrumentation.

As an Audio File Database technology, a structured UNIX directory file system was used and its correspondent Gateway Module for the Music Server

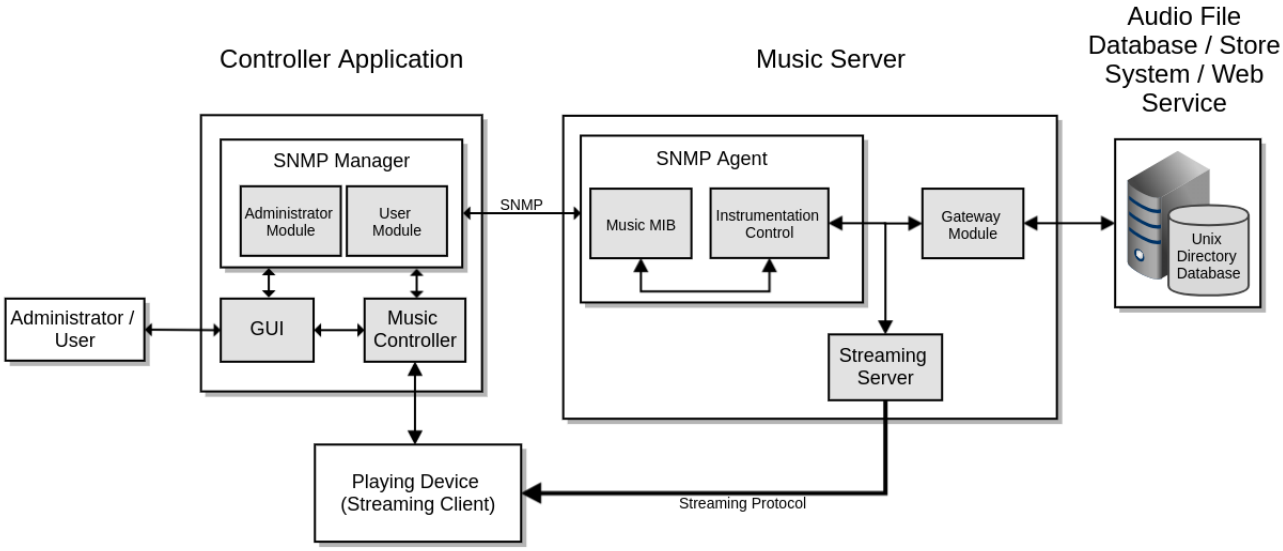


Figure 4.1: Implemented Architecture Components.

component were created. An Indirect Streaming Server was deployed by means of the integration of a streaming server on the Music Server component and a streaming client on the Playing Device component. These modules were based on the VLCJ Open Source API [57] that provides Java libraries to build a media server or a player, allowing either to play media files locally or to stream them over a local IP network. On the Java platform, the VLCJ offers bindings with the VLC Open Source media player from VideoLan, providing a well-documented API and good support for developers.

The remainder of this chapter explains in more detail the development of the prototype solution.

4.1 SNMP Agent

After the Music MIB definition, the first major step towards the development of an SNMP agent is the setting up of the MIB database instrumentation. Its management objects were syntactically and semantically verified in the MIB Designer tool. Semantic verification is only possible for verification of data types as semantic meaning can only be verified by testing the

prototype system's behavior.

The specification output file generated by the MIB Designer was then used as the AgenPro input to generate the Java based source code with data objects for implementation of the MIB instrumentation module for the SNMP Agent module. This source code, together with the SNMP4J SDK libraries for communication primitives, was used to code the complete SNMP Agent implementation included in the Music Server.

The Music Server also implements a complementary Gateway Module that is able to manage audio meta-information (title, artist, album, track number, genre) from a UNIX structured directory file system containing audio files from music collections. This information is mapped to the SNMP Agent Music MIB instrumentation. The Agent cross-references this information on Music MIB tables, in order to associate the music files with their respective albums, artists and genre. It also implements other more generic MIB tables, referring to varied system information, other Music Servers, Gateway Modules, users and administrators.

More specifically, the created *MusicMib* Java class provides access to the object values of the MIB tables, including row manipulation. This means that, when adding a new row of information into a MIB table, the table-model provides knowledge about in which row it will be inserted to. For example, the method *getRowCount* returns the number of rows a MIB table currently has.

In the development of the MIB instrumentation it was taken into consideration that some MIB tables cannot have duplicated value attributes (or object's instances values), like genre, artist, format and album of an artist. A music genre (e.g. Rock/Jazz/Classical) can only have one entry in the Genre table. For Artist (e.g. Queen/Guns N' Roses) and Format (e.g. OGG/FLAC) it was used the same principle, as they can only have one entry in the respective MIB tables. The Album table is different, due to the fact that it can have duplicated values, as long as they belong to different artists. This means that an artist can not have two albums with the same title, but two artists can have an album with the same title.

In order to prevent duplicated values the information is previously veri-

fied if it already exists in the table or not (Algorithm 1). If, in fact, it already exists, the method returns the respective number ID of the information that was about to be inserted (artist, format, genre). If it does not exist, the information is normally inserted into a row and this last one into the respective table and the method returns the row number where it was add to.

Algorithm 1 shows how a music genre is verified and then added to the MIB. This is a generic algorithm as it used for most of the rest of MIB tables. For tables like ArtistMusic, it is not need to return any value as it cross-references the music files with the respective artist.

```
input  : genreName
output: rowNumber
rowNumber ← GenreTable.getRowCount+1;
for i ← 1 to rowNumber do
    if GenreTable.getRow(OID(i)).
     getGenreName().equals(genreName) then
        return i;
    end
end
newRow ← (rowNumber , genreName);
GenreTable.addRow(new GenreRow(OID(rowNumber),
 newRow));
return rowNumber;
```

Algorithm 1: Addition of a genre to the MIB.

To verify the information that was added into the MIB instrumentation objects, the following NET-SNMP command should be used:

```
snmpwalk -v 2c -c public 127.0.0.1:2000 1.3.6.1.3.18
```

As detailed previously, the OID 1.3.6.1.3 is the OID of the root of MusicMIB and the number 18 represents the root of the Server table.

4.2 Streaming Server

The Music Server is also responsible for starting an audio streaming once a playing request is submitted by the manager on the Controller Applica-

tion. The Agent monitors the MIB instrumentation for when the object *musicIDuser* (in the User MIB table) changes to the correct enumerated value. It then requests the streaming server for playing the music track identified by the respective *musicID*. More specifically, the object *musicIDuser* value is changed when a valid SNMP SET command is received by the Agent, sent by the Manager on the Controller Application when a user triggers a play, pause or application port where the user's Playing Device is listening. The destination unicast streaming port numbers and network addresses of the Playing Devices are defined in the Device MIB table. By allowing different stream ports to be configured, the system allows the server to support multiple stream connections on the same or on different devices/destinations at the same time. When the music ends the value of musicIDuser is reset and the stream for that destination stops.

4.3 SNMP Manager

The Manager implements methods that retrieve or set new values from or to the MIB tables of an SNMP Agent. The manager works like a *bridge* as it sends SNMP request primitives (or just SNMP commands) depending on the actions originated from the user interface. A user or an administrator action will trigger an SNMP command. For example: if an user change his password, this will trigger a SET command, while selecting an artist it will trigger a GET command in order to retrieve the music tracks associated to that artist.

Note: A GET command, in the context of this document, was implemented as one of the three SNMP GET request primitives: *Get.Request*, *Get-Next.Request* or *Get-Bulk.Request*. Also, a SET command was implemented as an SNMP *Set.Request* primitive.

The GET and SET commands are executed with the table OIDs and column IDs, which are provided by the Java class *MusicMib*, as it is shown in Algorithm 2. When requesting a GET or a SET, it is necessary to know the OID of the object's instance to be obtained or modified. As it is shown

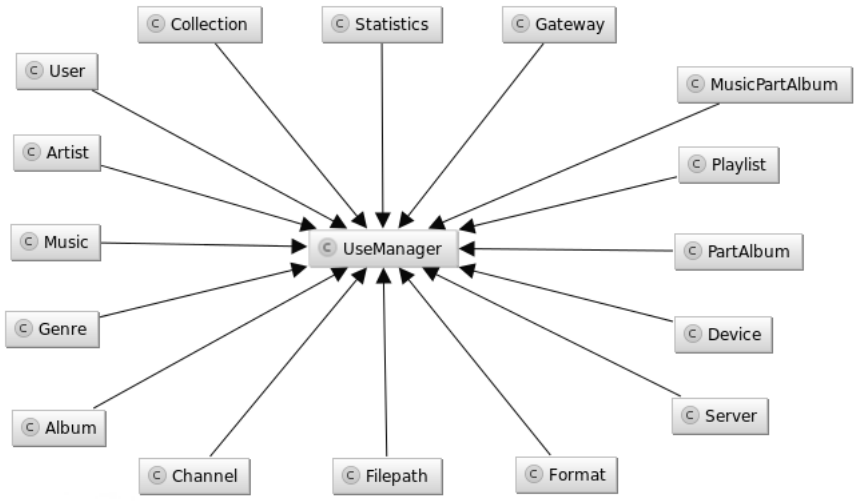


Figure 4.2: Class relationships between the *Usemanager* and the table classes.

in Figure 4.2, a Java class was created on the Manager for each MIB Table that the Agent has. Once the values are retrieved from the MIB Tables of an Agent, they need to be organized and structured in the Manager's software. For example, as there is an User MIB table, the Manager also has a User Java class with the same objects/attributes that exist in the User table of the Agent's MIB instrumentation. Those attributes have their own *getters* and *setters*, so they can be easily accessed and modified.

The manager has a Java class, the *Usemanager*, which makes use of the Java table classes (Figure 4.2). The *Usemanager* contains methods that allow to send commands to an Agent, without the need of storing any data. Like the *isSNMPServer* method, which checks if a given IP address is running an SNMP Music Server, or the *getLoginCredentials* method returns an object of the type *User* if the user or administrator inserted his credentials correctly in the GUI Login window.

To execute an SNMP GET command, the *Usemanager* makes use of the Manager's *getAsString* and *getAsInteger* methods (Algorithms 2 and 3), which are dependent on the type of data that is about to be requested, a String or an Integer type, for example. OIDs are passed as parameters to these methods, in order to indicate which information from the Agent's MIB

is to be retrieved. If successful, the Agent replies with a *Get.Response* with the values that were requested. The *Usemanager* contains object lists of the type of each table class. The lists are filled when they are needed, preventing a big retrieval of data upon startup of the Manager. When retrieving a MIB table, the *Usemanager* uses both *getAsString* and *getAsInteger* methods. After the Agent replied, the values retrieved are then added to the respective object class and then to a list. Algorithm 2 details the steps of how a MIB table is obtained and added to a Java list object, which in this case is for the Album table.

```
output: albumList
for i ← 0 do
    id ← getAsString(OID(
     oidAlbumEntry + "." + colIdAlbum + "." + i));
    if id.equals("noSuchInstance") then
        break;
    end
    title ← getAsString(OID(
     oidAlbumEntry + "." + colAlbumTitle + "." + i));
    label ← getAsString(OID(
     oidAlbumEntry + "." + colLabel + "." + i));
    year ← getAsInteger(OID(
     oidAlbumEntry + "." + colYear + "." + i));
    albumList.add(new Album(id,title,label,year));
end
return albumList;
```

Algorithm 2: Use of the GET command in order to retrieve a MIB table.

Relationships between tables must also be implemented in the Manager's module. These associations are needed as it allows cross-reference the instances of the objects. For example, they are necessary to correctly associate music tracks to their respective artists and respective music albums. Again, using the Album/Artist example, to make the association between Album and Artist the Album class must contain a list of Artist and Artist class must contain a list of Album. Both of these classes have a method that searches for a given ID in a given list and returns the object that matches with that ID. So, when associating two tables, it is required to obtain the

object of the respective ID and then adding this returned object to the other object's list. Algorithm 3 details the steps to make these kind of associations.

```
input : albumList,artistList
for i ← 0 do
    id ← getAsString(OID(
     oidAlbumArtistEntry + "." + colIdAlbumArtist + "." + i));
    if id.equals("noSuchInstance") then
        break;
    end
    //Get Album object from the AlbumList;
    albumID ← getAsInteger(OID( oidAlbumArtistEntry +
     "." + colAlbumIDalbumArtist + "." + i));
    album ← Album.getAlbum(albumList,albumID);
    //Get Artist object from the ArtistList;
    artistID ← getAsInteger(OID( oidAlbumArtistEntry +
     "." + colArtistIDalbumArtist + "." + i));
    artist ← Album.getAlbum(artistList,artistID);
    //Add to each other's list if not already in them;
    if !album.getArtistsList().contains(artist) then
        album.getArtistsList().add(artist)
    end
    if !artist.getArtistsList().contains(album) then
        artist.getArtistsList().add(album)
    end
end
```

Algorithm 3: Associating an album with an artist.

The SET command is used when a new value is to be set to the object's instance of the OID (both OID and new value are passed as parameters). An exception is thrown if there was no response or if there was error during the execution of the command on the Agent. Algorithm 4 shows a simple example of how to send a SET command in order to change a value in the MIB.

Although SET command allows to set/configure a value in the MIB table, this can only be done to an object that has been previously added to the MIB, that is, to an instance that already exists in the MIB instrumentation. So,

input : $musicID$,$userID$
//After check if $musicID$ and $userID$ are within the parameters;
set((oidUserEntry+"."+colMusicIDuser+"."+$userID$),$musicID$);
//If successfull, then the music track of musicID should start playing;

Algorithm 4: A SNMP SET command.

in order to add or remove a user, a playlist, a device, a server, a collection or statistics, it is used a SET command with the OID of the MIB object *AddRemoveRow*.

The values that are sent within the SET command are a composition of parameters required to add or remove a row in a specific MIB table. The values need to indicate if it is to add or remove the row, followed by the name of the MIB table (where to make changes) and then followed by the new values that will be in the new row. For example, to add a new row in the Playlist table, this needs the values "add:playlist:RockPlaylist", to be sent. When the Agent receives the SET command, it checks if the value of the object OID *AddRemoveRow* has changed and adds or removes a row based on the received parameters. After the row has been added or removed the value of *AddRemoveRow* is reset.

These methods are the core methods used on the Manager software, either to obtain or to modify instance's values and also to add or remove rows in MIB tables. The methods can be triggered by the user interface actions through the *Usemanager*.

4.4 Graphical User Interface

The GUI was developed with the JavaFX API which has available a set of media packages that allows to design and creation of graphical client applications.

The user interface module consists of five windows: Login, User Preferences, Administrator Panel, About and the Player. All of these windows

contain graphical objects that once the user interacts with them, triggers specific actions.

Here is a brief description of the main functionalities for these windows:

- The Login window is the first window that appears upon the start of the controller client application. The user enters his username and password to login into the system. There is also a button to register a new user and a field to show errors if any occur;

- The main window allows browsing the list of tracks of the music collections, dependently on the selected object (a user's playlist, device, server or a search text). There are also some media buttons like, play, pause, stop, previous and next, a slider for the audio volume, a panel with the information about the music that is currently playing, buttons to add or remove a music track to a playlist. From his window, the user has access to the other windows, except for the Admin Panel, where only administrators are allowed;

- The Admin Panel is only accessible to administrators and allows them to manage users, servers, musics, collections and statistics;
- The User preferences window enables a user to change his password, manage his devices and playlist and reset his statistics;
- The About window is just to show general information of the Controller Application.

User Types

There are three types of users: guest, user and administrator. The administrator is the top level of the user types and manages the accounts of guests and users with the authority to transform a guest into a user or a user into an administrator. He can delete guests and users accounts. An administrator is also able to manage music collections directly from the Controller Application interface.

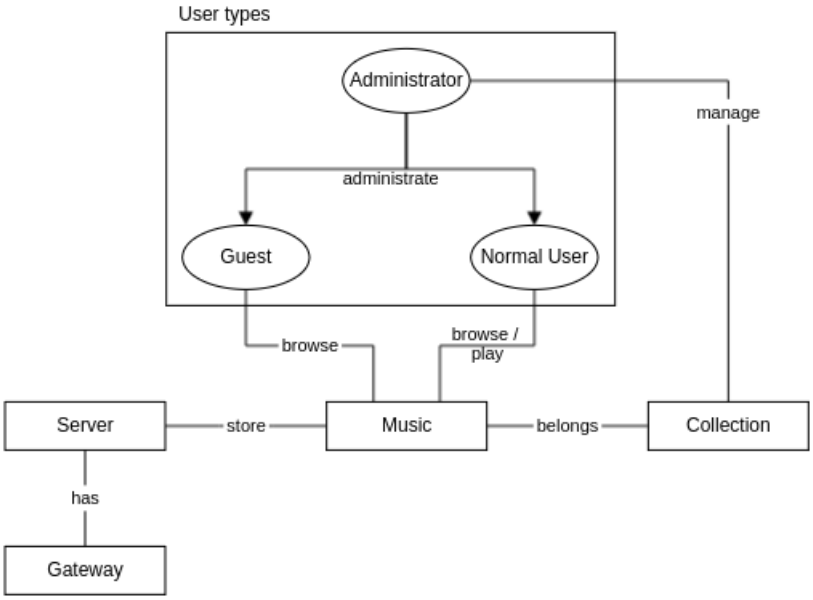


Figure 4.3: User types actions and how entities are related.

Upon registration, a guest can only browse for music collections but is not able to listen to audio file. On the other hand, a normal user can browse and play any music present in the collections.

Figure 4.3 illustrates what actions each type of user can perform, as well as how the entities are related among them.

4.5 Audio Streaming

The prototype audio Playing Device receives the audio stream provided by a streaming server implemented inside the Music Server. The audio file source is from the Audio File Database system. A streaming client process is always in a standby state on the player. The prototype Controller Application also supports redirection of the audio stream to third party software or hardware Playing Devices through streaming.

Figure 4.4, shows the communication exchanged between the Music Server and the Controller, and how a streaming is started and terminated. Once a

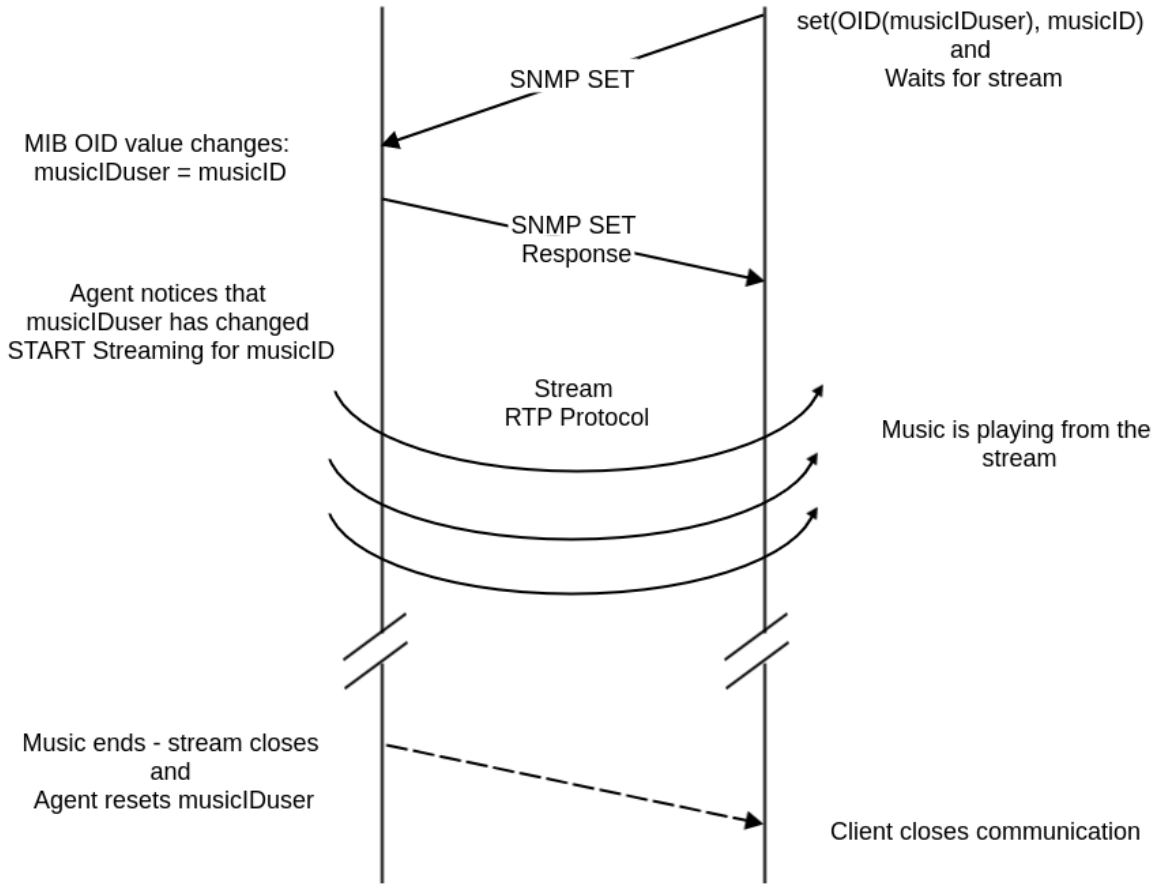


Figure 4.4: Flowchart between the Controller Application and the Music Server.

user selects a music track to play, the Manager Module sends a SET command to the Agent Module. Then, a stream is opened with the Playing Device's IP address and port waiting for the connection to be established. On the other side, the Agent requests a streaming service to the local streaming server for the selected music.

When the music ends, the Agent resets the value that the SET command changed earlier in order to indicate which track to play. To allow multiple stream connections at the same time, the streaming's client port is randomly chosen from a valid range of values.

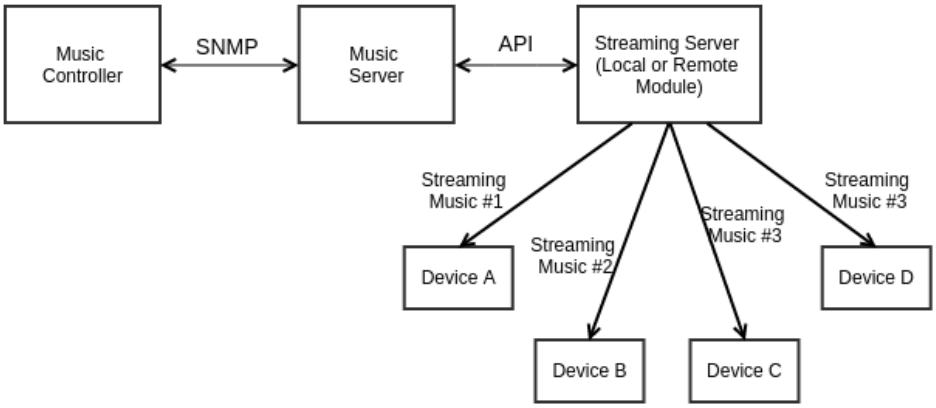


Figure 4.5: Streaming to remote devices.

4.5.1 Remote Devices

The Controller Application allows a user to remotely control the playing of a music track in a device other than where the GUI is being executed.

An example is when there are devices around the house connected to the home network. These devices can be any kind host stations (e.g. desktop, notebook, mini-PC, smartphone, multimedia systems, etc.) that have local audio playing functionalities available or are connected to a classic external digital sound system (DAC, amplifier, speakers), as long as they include a streaming client application.

Figure 4.5 shows the connections between Controller Application, server and the playing remote devices. It is possible to notice that SNMP messages only flow between the Controller Application (SNMP manager) and the Music Server (SNMP agent), while the streaming happens between the Playing Devices and the streaming server. As noted earlier, is possible to have several streams from the same music track occurring at the same time.

4.6 Search Feature

The search feature is implemented on the Music Server side, as it holds the needed tables in order to perform searches. If not, the data required would

have to be transferred to the controller, which would be highly inefficient for big volumes of data. Furthermore, having most of the processing on the server side has the advantage of lighten the controller.

In order to implement this feature, it was necessary to extend the initial Music MIB (see Figure A.1). It was added four string scalar objects, one for every text field/filter supported by the search feature, three tables (artist, album and music) and two enumerated scalars objects.

When the user uses at least one of the search text boxes available in the user interface, it triggers the SNMP Manager to send a SET command to the SNMP Agent in the Music Server. This command carries the typed in data from the text boxes and the user's search threshold. The searchStatus object is also set accordingly:

- 1 - The agent has the result search tables emptied and is ready to perform a new search;
- 2 - Executing the search;
- 3 - Search has ended and the results are ready for retrieval.

Strings Comparison

In order to perform a search, the agent instrumentation executes an algorithm that compares the search terms with the artist names, album titles and music titles in the Audio File Database.

In the first implementation effort it was used the *compareToIgnoreCase* method of the Java class *String*, which compares two strings lexicographically. The result is a negative or positive integer if the string object lexicographically precedes or follows the argument string. The distance is zero if the strings are equal. However, as it is shown in Table 4.1, this comparison method did not return proper search results.

As such, it was decided to investigate other solutions. The Hamming distance [58] is widely deployed, but it only considers string substitutions to calculate the metric and so it requires that the string must have the same length [59] which would not apply here.

The Levenshtein distance algorithm is one of the most widely used comparison string metrics and it has been extensively used in spell checkers and dictionary correction systems [60, 61]. The Levenshtein distance algorithm, also known as edit distance, measures how different two strings are. The result distance is the minimum number of editions required to change one string into another. The original Levenshtein distance takes in consideration that an edition can be an insertion, deletion or substitution. The Levenshtein distance is zero if the strings are equal (no editions are required) and greater than zero if different. The Algorithm 5 shows how the Levenshtein distance calculates the similarity between strings. Tests were made and Table 4.1 resumes the results. Its possible to verify a clear improvement on the desired search results in comparison to the initial method.

On the other hand, the *contains* method of the Java String class yields a result of zero if the first string (searched text string) is contained in any of the other. This method (which ignores case) is used after the Levenshtein distance has been calculated, as it improves even further the quality of search results when the user inserts only a segment of the word to search for (for example, when the user inserts "red hot" in order to search for the tracks of the "Red Hot Chili Peppers" band).

Algorithm 6 shows the full process of similarity calculation between a given string and string on a target list of strings.

Table 4.1 presents results for various examples of an artist search on a list with ten different artists in it. It shows the terms for searching and the top five results with the respective score. The difference when using the *contains* method is noticeable for the "queen" search term, as the result "Queens of the Stone Age" comes ranked second whereas otherwise it would not even appear in the top five results.

Generic Search

The generic search uses the search keyword to search in the artist, album and music tables. Each table search returns a hash map with the ID of

```
input  : string1,string2
output: result
string1 ← string1.toLowerCase();
string2 ← string2.toLowerCase();
if string1 == string2 then
    return 0;
    //The strings are equal
end
int costs[] ← new int[string2.length()+1];
costs[0] ← 0;
for i ← 0 to string1.length() do
    int lastValue ← 0;
    for j ← 1 to string2.length() do
        int newValue ← costs[j − 1];
        if string1.charAt(i-1) != string2.charAt(j-1) then
            newValue ← MIN(newValue, lastValue, costs[j]) +1;
        end
        costs[j − 1] ← lastValue;
        lastValue ← newValue;
    end
    if i > 0 then
        costs[string2.length()] ← lastValue;
    end
end
return costs[string2.length()];
```

Algorithm 5: Levenshtein Distance Algorithm.

```
input  : string,list
output: hashmapResult
for each item in list do
    comparison ← LevenshteinDistance(string,item);
    if containsIgnoreCase(string,item) then
        comparison ← 0;
    end
    hashmapResults.put(item,comparison);
end
hashmapResults ← sortByValues(hashmapResults);
return hashmapResults;
```

Algorithm 6: Full process of similarity calculation.

	Resulting List		
Word for Searching	*compareToIgnoreCase*	Levenshtein	Levenshtein with *contains*
queen	**Queen - 0**	**Queen - 0**	**Queen - 0**
	Phill Collins - 1	Daft Punk - 8	**Queens of The Stone Age - 0**
	Red Hot Chili Peppers - 1	Joe Cocker - 8	Daft Punk - 8
	Tara Perdida - 3	Blink 182 - 9	Joe Cocker - 8
	Xutos e Pontapés - 7	Tara Perdida - 11	Blink 182 - 9
quen	Phill Collins - 1	**Queen - 1**	**Queen - 1**
	Red Hot Chili Peppers - 1	Daft Punk - 8	Daft Punk - 8
	Tara Perdida - 3	Blink 182 - 8	Blink 182 - 8
	Xutos e Pontapés - 7	Joe Cocker - 9	Joe Cocker - 9
	Joe Cocker - 7	Tara Perdida - 11	Tara Perdida - 11
pil colins	**Phill Collins - 1**	**Phill Collins - 3**	**Phill Collins - 3**
	Queen - 1	Joe Cocker - 7	Joe Cocker - 7
	Queens of The Stone Age - 1	Daft Punk - 9	Daft Punk - 9
	Red Hot Chili Peppers - 2	Queen - 9	Queen - 9
	Tara Perdida - 4	Blink 182 - 9	Blink 182 - 9

Table 4.1: String comparison examples.

artist/album/music as the hash map key and the comparison score as the value. Afterwards, the three hash maps are sorted by their values and their IDs added to the respective MIB result search tables.

Specific Search

The specific search performs each search in each respective table. It searches the artist keyword in the artist table, the album keyword in the album table and the music keyword in the music table. However, if the user types in more than one search text box, entries that match each search text box will be presented. For example, if the artist and album text boxes are fill in, the results listed are matches from the list of artist and also from the list of albums.

The match is done through a nested loop join algorithm [62]. A nested loop join is the simplest form of join algorithm: for each entry in the first table, it goes through all the entries in the second table. For example, the process for a join search of artist with album verifies the resulting album

table against the resulting artist table and a match happens when an album ID that was previously associated to an artist ID is also in the resulting album table.

The average calculated from the hash maps values (comparison metric scores) of artist and album entries updates the entry value in the resulting album table. Algorithm 7 demonstrates how the nested loop join is performed with artist and album resulting lists.

```
input  : artistList,albumList,albumArtistTable
results ← albumList;
for Each entry in artistList do
    for i ← to albumArtistTable.size() do
        if albumArtistTable(i).getArtistID == entry.getKey then
            albumID ← albumArtistTable(i).getAlbumID();
            if results.containsKey(albumID) then
                results.replace(albumID,(results.get(albumID)+
                  entry.getValue())/2);
            end
        end
    end
end
albumList.putAll(results);
return hashmapResults;
```

Algorithm 7: Match between artist and album.

For search requests with matching between three tables, two nested loop join algorithms are executed. The first nested loop join is performed with the first two tables (artist and album). Then, the second join is executed with the result of the first join and the last table (music). Just like for the generic search, the resulting hash maps list is sorted by values and IDs are added to the corresponded MIB result search tables.

This algorithm is not optimized on the prototype Music Server implementation, as it is not relevant for the scope of the project. For example, the sorted nested join and the hash join algorithms should yield a better performance, in comparison to the implemented nested join [62].

After the search is completed, the Agent instrumentation modifies the value of searchStatus accordingly and sends an SNMP TRAP.Request primitive to the Manager in the Controller, informing that the results are ready to be retrieved. The Manager obtains the results through GET commands, and the Controller displays them into the user interface.

Multi-Context/Multi-Client Search

The simple specific search previously described had the limitation of being available for one client at a time. If a user wanted to do a search, while there was one already in progress, it would had to wait until the first had finished the process.

Therefore, in order to make a multi-client search algorithm, the MIB had to be extended again. It was added four more scalars objects for each of the search fields. These fields also exist in a new table called SearchTerms. It was also added, the SearchUsers table and the column SearchID to each one of the three result tables (Artist, Album and Music). Also, to allow the concurrent execution of the algorithm, the implementation of multi-threads for the search processes was introduced.

Figure 4.6 shows a flowchart of the algorithm. When a user requests a search, the strings from the search text boxes are sent on via SET commands to the Agent in the Music Server. A value for the userIDSearch scalar is set and the Agent checks if the exact combination of these terms already exists in the SearchTerms table.

Figure 4.7 shows how the SearchTerms, the SearchUser and search resulting tables are related. If the combination is already in the SearchTerms table, the server takes the value of IDSearchTerms correspondent of that combination and looks for it in the termsID column of the SearchUsers table. It verifies if the search has already been completed updating the corresponding userID table entry with the userIDSearch value. If the combination are not yet in the SearchTerms table, it is added and the corresponding values of the IDSearchTerms and userIDSearch objects are added to the SearchUsers table.

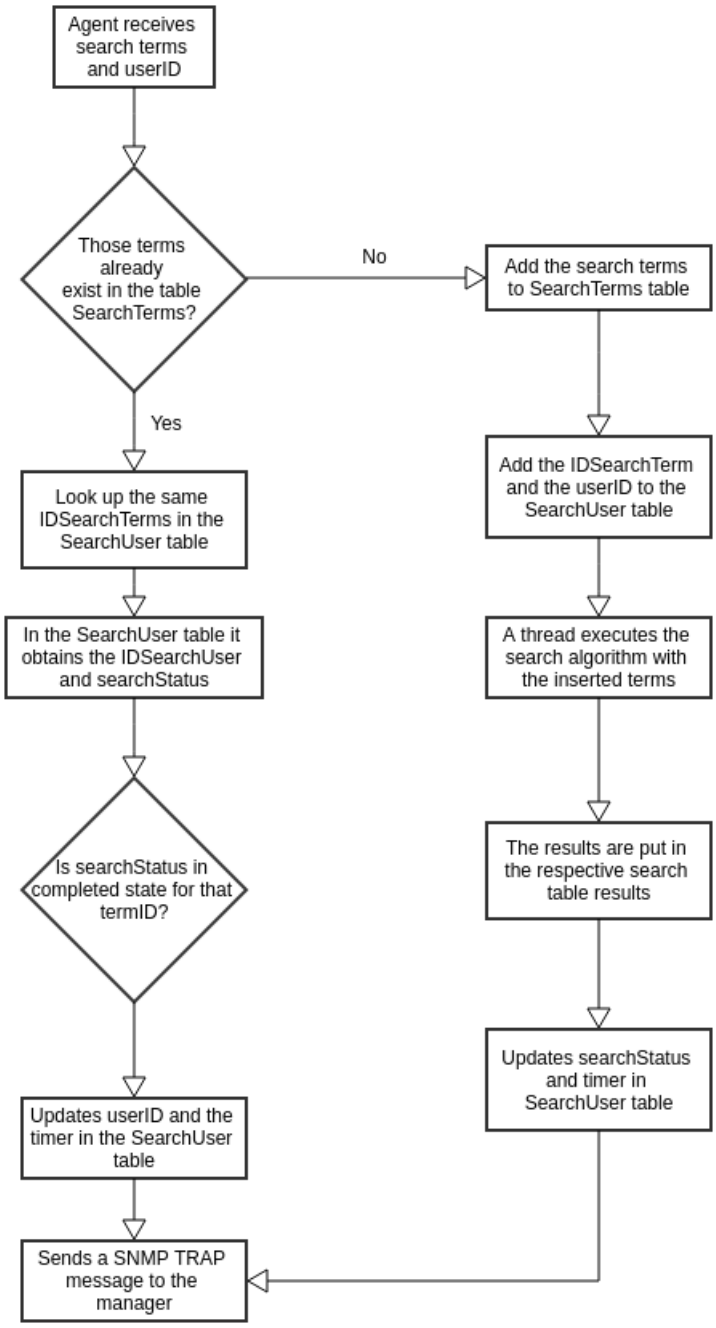


Figure 4.6: Flowchart of the Multi-Client Search.

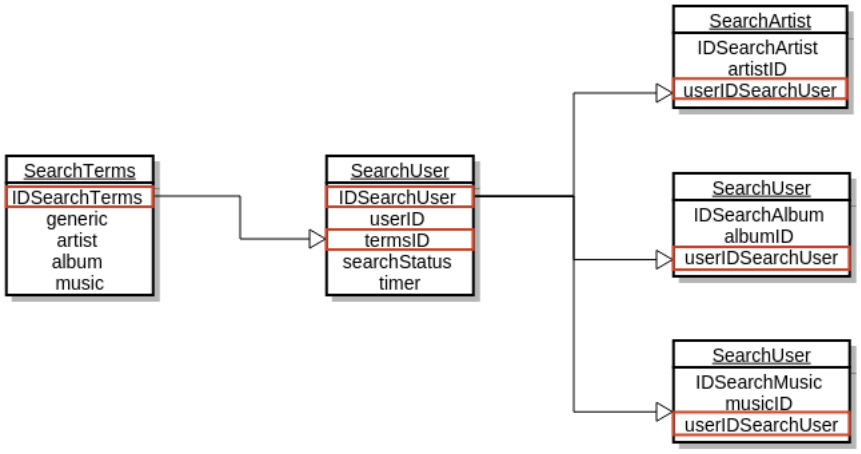


Figure 4.7: How the MIB tables are related in Multi-Client Search.

One thread is created for execution of the search algorithm for each combination on SearchUsers table. After completion, the search results are inserted into the respective table results and the searchStatus fields of the SearchUsers table are updated. Lastly, the Agent sends a Trap.Request primitive to the Manager, informing that the search results are ready to be retrieved. Upon receiving the notification, the client retrieves the search results that correspond to its user's ID.

Caching Search Results

The SearchUser has a timer field, which marks the time of when a search has been completed or re-accessed. This field is used so the entries can be cleared after some predefined time is passed without the entries being used.

So, the search functionality implemented on the Music Server permits caching of search results for a temporary period of time. These temporary tables improve the performance of the search feature.

This search feature is supported directly on the SNMP Agent instrumentation by means of the MIB table that store temporary searching results. This cache table allow much shorter access times on common requests.

4.7 Testing The Prototype System

In order to setup the entire system, it is firstly required to deploy the Music Server, including the SNMP Agent Module with the MIB instrumentation and the Gateway Module for interaction with the Audio File Database. In this prototype, the music database is based on a structured directory file system and the Gateway Module is embedded on the Music Server software. Directory paths for music collections are configuration parameters of the Music Server so the Agent's instrumentation, through the Gateway Module, can import/update the music meta-data information from all files of the music collections into the correspondent MIB tables. Other Music Server configuration parameters must also be setup the first time the system is installed: network and applications addresses, identification, description and location, etc.

It follows an excert of the log file of the setup process when the Music Server starts up:

```
Adding tracks
(...)
Adding Music: 13 Title: Contact
/home/plut/Music/Daft Punk/Random Access Memories/13 Daft Punk - Contact.mp3
Adding Music: 14 Title: Horizon
/home/plut/Music/Daft Punk/Random Access Memories/14 Daft Punk - Horizon.mp3
Adding Music: 23 Title: Smells Like Teen Spirit
/home/plut/Music/Nirvana/Nevermind/01 Smells Like Teen Spirit.mp3
Adding Artist: 2 Nirvana
Adding Album: 2 Nevermind
Adding Music: 24 Title: In Bloom
/home/plut/Music/Nirvana/Nevermind/02 In Bloom.mp3
(...)
```

Information like artist, album or genre, are only added to the MIB tables when these have not yet been added. For example, when adding the music track "Smells Like Teen Spirit" from the artist "Nirvana", the module verifies that the artist is not yet on the Artist MIB table and so it adds the respective meta-data.

Testing the SNMP Agent

It is possible to execute a simple test to verify if the Agent is working correctly and accepting requests. In order to do so, some SNMP GET and SET commands can be performed using NET-SNMP commands.

```
snmpwalk -v 2c -c public 127.0.0.1:2000 1.3.6.1.3.2.2.2.1.2
SNMPv2-SMI::experimental.2.2.2.1.2.1 = STRING: "Daft Punk"
SNMPv2-SMI::experimental.2.2.2.1.2.2 = STRING: "Nirvana"
```

The OID 1.3.6.1.3.2.22.2.1.16.1 represents the musicID object of the first entry/row on the User table, this OID is modified when a music is requested to be played or stopped (detailed in Appendix A).

```
snmpset -v 2c -c public 127.0.0.1:2000 1.3.6.1.3.2.22.2.1.16.1 i 13
SNMPv2-SMI::experimental.2.22.2.1.16.1 = INTEGER: 13
```

4.7.1 The Controller Application

Once the Music Server is up and running as a service, the Controller Application can be executed to access the music collection.

Login Window

The login window (Figure 4.8) is where a user logs into the system and is composed by two text boxes and two buttons, so that the user can type in his username and password.

When the "Log in" button is clicked, the application verifies if the entered credentials exist and are correct in the MIB User table. A text message is displayed informing the user if an error occurred, otherwise the login window is closed and the main window displayed.

Main Window

The main window holds the interface objects that are used by a user. Each one of the GUI objects has an action: to control the music track, to change

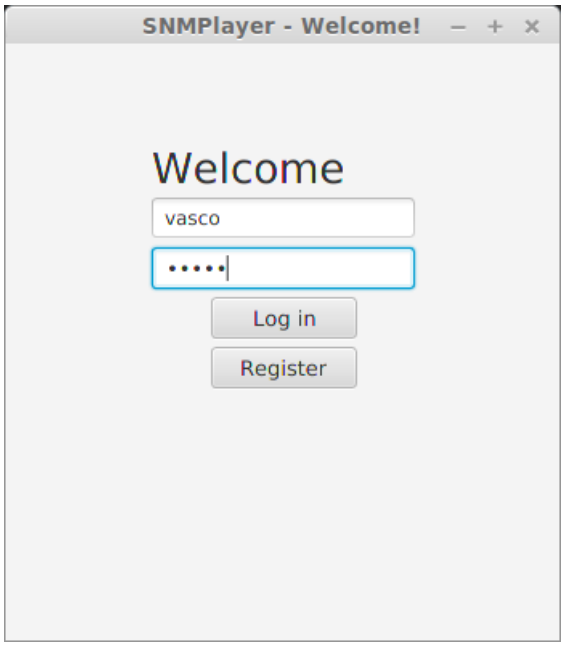


Figure 4.8: Login window GUI.

the audio volume, to search for a specific music track, or even add devices and create playlists. The interface is composed by the following objects:

- Buttons for controlling the streaming (e.g. play, stop) music track;
- Text fields for searching;
- Sliders for volume and music progress;
- Lists in order to display the music track, servers and playlists;
- Menus for username options and device selection access;
- Labels for displaying the current music track information and playing time;
- Time.

Figure 4.9 shows what the main window interface looks like. After the GUI objects have been loaded, the application retrieves the needed information from the MIB instrumentation, that is, the SNMP Manager retrieves

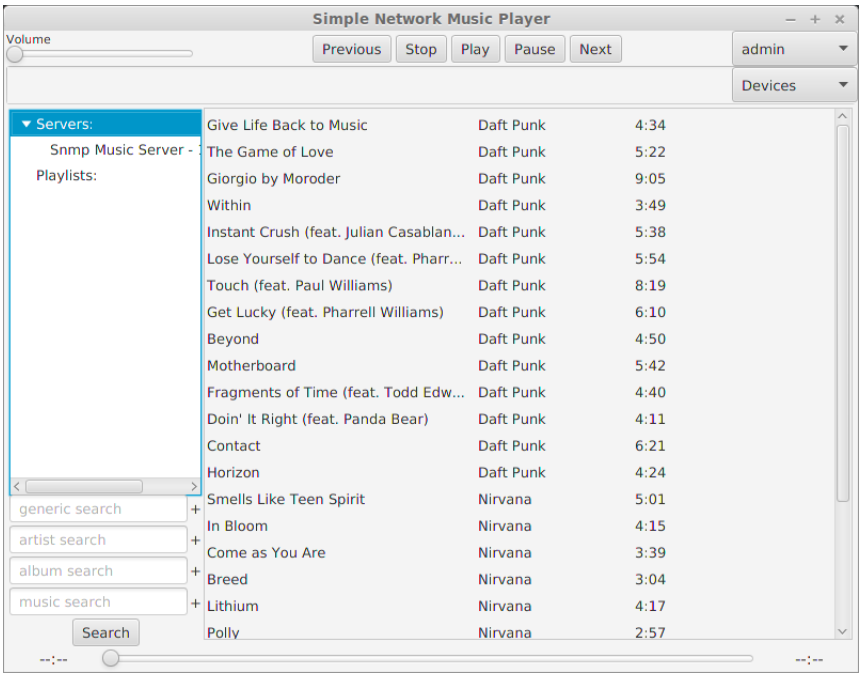


Figure 4.9: Main window interface with retrieved entries.

entries from the following MIB tables: playlist, server, device and music. It is possible to observe that a server has been added to the server list on the left side of the GUI, the user menu has changed to the current logged in the username and some music tracks were listed.

Playing a Music Track

To play a music from the list, the user simply selects one and clicks on the Play button or just double-clicks on top of the music title. This action sends a SET command to the Music Server for the respective user's musicID OID (1.3.6.1.3.2.22.2.1.16.1) with the value of the music's ID. Depending on which value goes in the SET message, the server initiates or stops the streaming server. For example, when selecting the music track "Touch" from the "Daft Punk" artist (see Figure 4.9) the Manager sends a SET command with the music ID value 6 to the respective OID of musicID column in the User table:

```
agent: a value has changed on the User table
        Row/UserID: 1
        MusicID: 6
        Filepath: /server/Music/Daft Punk/Random Access Memories
                        /07 Daft Punk - Touch (feat. Paul Williams).mp3
        Streaming server INITIATED on port 5555
```

Search Feature Implementation

Figure 4.9 shows that the search box is composed by four text boxes: generic, artist, album and music. The generic search looks for the search keyword in the artist, album and music tables, while the specific search looks for each specific keyword (if typed) in the respective table.

On one hand, if the user just uses the generic search field, he will be presented with artist names, album titles and music titles that are similar to the keyword typed. On the other hand, when performing a specific search, depending on which search fields the user types in (if artist, album or music search field), he will be presented with the search result of the respective table(s) of the searched keyword.

Figure 4.9 shows a plus sign next to each text box. This allows a user to add more searching criteria. For example, searching for more than on artist in the same search.

Table 4.2 shows examples of generic and specific searches along with their results. In the first search, it is performed a generic search as the rest of the search fields are empty. Here, the search word ("queen") is searched on all three fields (artist, album, and music) without doing any match among them. Therefore, the results on the three presented lists are not related with each other. This explains how albums and music tracks from different artists are presented in the artists list. However it is possible to verify that two music tracks from the respective album, also appeared in the album list. This is due to the fact that there are music tracks with the same title as the album in which they have been released.

The second, third and fourth searches have two of the search fields typed in so they execute a combined search as detailed before. Due to that reason,

		Results		
#	Search	Artist	Album	Music
1	Generic:"queen" Artist: Album: Music:	Queen Queens of The Stone Age Queensryche	Queen by Perfume Genius Queen - Raheen Vaughn Queens of the Cloud by Tove Lo	Queen by Perfume Genius Queen of California by John Mayer Queens of the Cloud by Tove Lo
2	Generic: Artist:"queen" Album:"magic" Music:	Queen Queens of The Stone Age Queensryche	Kind of Magic by Queen Magic by Coldplay Magic by Bruce Springsteen	-
3	Generic: Artist:"queen" Album: Music:"love"	Queen Queens of The Stone Age Queensryche	-	Somebody To Love by Queen Crazy Little Thing Called Love by Queen Love Kills by Queen
4	Generic: Artist: Album:"magic" Music:"friends"	-	A Kind of Magic by Queen Magic by Coldplay Magic by Bruce Springsteen	Friends Will Be Friends by Queen Friends by Ed Sheeran Internet Friends by Knife Party
5	Generic: Artist:"queen" Album:"magic" Music:"friends"	Queen Queens of The Stone Age Queensryche	A Kind of Magic by Queen Magic by Coldplay Magic by Bruce Springsteen	Friends Will Be Friends by Queen Friends by Ed Sheeran Internet Friends by Knife Party

Table 4.2: Examples of generic and specific search functionality.

their results are related. For example, in the second search, the album tittle "A Kind of Magic" comes first because it was a match with the artist "Queen". The last example presents the results that matched the music "Friends Will Be Friends" that was released in the album "A Kind of Magic" by the artist "Queen".

4.8 Comparison With Other Solutions

As stated earlier, the search feature of the DLNA and DAAP solutions has some limitation as there is no support for implementations of this functionality on the Music Server side of the architectures nor there are any guidelines on how to implement it on the client application side. UPnP only has the MediaServer's Content Directory service, which is responsible for just retrieving the meta-data about each media item and make it available for Control Points to browse them, which can be problematic on large music

collections. All implementation issues must be resolved on the client controller side with proprietary mechanisms, which can have a great impact on the overall performance of the system as the servers do not have the possibility of temporarily store search results. Some of these client applications require third-party software, like XBox Media Center (XBMC) [17] in order to implement it.

On the other hand, the prototype system of the proposed model has direct support for this functionality on the Music Server side by means of the Music MIB definition itself. Also, as the Music Server does most of the processing, the clients can be much lighter. With this paradigm, client mobile applications can be easily developed, which could be a fundamental key advantage of this approach.

In the new client application prototype is possible for an administrator to authorize, or not, a selected group of users to play/listen certain music collections (or parts of them). For example, XBMC or iTunes applications do not support local user management account.

The developed prototype also has the ability to stream music to remote Playing Devices/machines other than where the GUI is being executed, although it is required some streaming technology to be supported in these devices in order to receive the audio stream from the Music Server. Through the GUI, a user can manage a group of devices and easily choose to play the music track independently to each device.

Furthermore, the prototype is very modular and is prepared to provide interaction with any hardware/software platforms, as long as they implement the same music MIB and support at least version 2c of the SNMPs.

Chapter 5

Conclusions

The functionalities and conceptual requirements implemented on the prototype system of this project proves that audio distribution systems for application on local area networks, like home multimedia systems, can be built using only standard management protocols (in particular SNMP), multimedia technologies and other open sources software. The SNMP architecture has been shown to allow an integrated management of network services independently of the equipment manufacturer or software makers and therefore, enables the monitoring, controlling and managing of a vast range of systems, including home electronic appliances. In the scope of this project, this protocol was used for home automation in the field of home-entertainment.

All elements of the prototype system were built taking with into consideration the conceptual requirements defined for the proposed architecture, relying solely on management protocols, standardized mechanisms and open source software. The prototype offers the possibility to control and manage music files in Music Servers using Controller Application. These controllers allow music files to be played on a local or remote device, through standard audio streaming protocols.

SNMP is used between the Music Servers and Controller Applications. Music Servers implement an SNMP Agent Module and a MUSIC-MIB installation. This MIB was defined specifically on the scope of this project. The prototype was also built using fundamental open source APIs in order

to enable the use of SNMP and streaming protocols. However, not all of the functional requisites initially established were implemented, leaving just a few for future development.

This work proves that the SNMP protocol can be easily deployed in an electronic house for domestic systems and in particularly for audio distribution, where every media device is connected to a home network. This project, shows advantages when compared to those already existent media device sharing technologies, which are based on more complex models and are content restrictive or device incompatible, like DLNA/UPnP systems which are not easy for the final user to understand and/or set up, and the DAAP/Bonjour is only compatible within Apple's system devices.

The purpose of this project was to develop an alternative system to these already established architectures that could include the same or even more functional features and using SNMP and open source technologies. Both DLNA and DAAP are proprietary systems, using technologies that are very restrictive with private and closed software environments.

Results

The developed prototype system includes all components and modules of the proposed architecture. All major functionalities of such type of audio distribution systems (managing user accounts and music collections, playing music tracks locally or remotely, searching tracks on music collections, etc.) were implemented and successfully tested.

The Controller Application is the software component that a general user will interact with and allows browsing, searching and playing music tracks like a normal music player would. The user will not notice that the music collection may be stored in a different location from where the interface is being executed and he can specify where he wants to listen the audio stream: in the local device or on a/multiple device(s).

The system implements a search feature where is possible to perform a generic search or a combined search using various fields such as artist, album and music title. The search are temporary stored allowing results to be

retrieved faster if the exact same search is performed again.

The prototype allows an administrator to manage user accounts. The administrator can also authorize or not, specific users to playlist certain parts of the music collections.

There is also a Gateway Module which allows importing of music metadata from an external Audio File Database on a structured directory file system into the Music Server.

From the usability/functionality tests executed on the prototype system could be concluded that all implemented features worked as expected.

Future Work

The project still has some features that can be further developed.

The Music Server still needs some development on the Gateway Modules so they could integrate support to other types of Audio File Database systems like the ones based on web-services. On the client application side, some additional functional features could also be added.

One of the next goals would be to develop a new streaming server to incorporate on mobile remote devices or audio renderers. Some networked audio speakers could include this new streaming software.

Another advance would be to develop and test a Controller Application for a mobile device platform based on an Android operating system. By being connected through the wireless home network, a user with a smartphone could control audio playing on any room/part of the house where the playing devices could be installed.

Also, the Music Server component could support streaming of individual audio channels, which would be very useful for surround sound applications.

The new solution could be technologically competitive when compared with other already existing solutions.

Bibliography

[1] Allegro Software Development Corporation, Conexant Systems Inc, Intel Corporation, Microsoft Corporation, Motorola, Nokia Corporation, Philips Electronics, Pioneer and Sony Electronics, "UPnP Device Architecture 1.0," no. October, 2008.

[2] J. Ritchie, T. Kuehnel, Intel Corporation and Microsoft Corporation, "UPnP AV Architecture:1," 2002.

[3] "DLNA - Technical Overview." http://www.dlna.org/dlna-for-industry/technical-overview.

[4] "DAAP Documentation." https://code.google.com/p/ytrack/wiki/DAAPDocumentation.

[5] "Sonos." https://www.sonos.com/en-gb/home.

[6] "Bluesound." http://www.bluesound.com/en-eu/.

[7] "Bose Multi-Room Speakers." https://www.bose.com/en_us/products/speakers/multi_room_speakers.html.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol," August 1988. https://tools.ietf.org/html/rfc1067.

[9] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "RFC 1157: Simple Network Management Protocol (SNMP)," May 1990.

[10] C. Rus, K. Kontola, I. Curcio, and I. Defee, "Mobile TV Content to Home WLAN," *Consumer Electronics, IEEE Transactions on*, vol. 54, no. 3, pp. 1038–1041, 2008.

[11] Intel Corporation, Panasonic Corporation, Sony Corporation, Toshiba Corporation and Hitachi Ltd, "Digital Transmission Content Protection Specification Volume 1 (Informational Version)," vol. 1, 2011. http://www.dtcp.com/documents/dtcp/info-20130605-dtcp-v1-rev-1-7-ed2.pdf.

[12] "DLNA Certified® Device Classes - DLNA." http://web.archive.org/web/20101222205822,http://www.dlna.org/digital_living/devices/.

[13] "The DLNA standard: a real mess?." http://www.digitalversus.com/tv-television/dlna-standard-real-mess-a971.html.

[14] "ISO/IEC standard on UPnP Device Architecture Makes Networking Simple and Easy." http://www.iso.org/iso/home/news_index/news_archive/news.htm?refid=Ref1185, 2008.

[15] "foobar2000." http://www.foobar2000.org/.

[16] "Jamcast." https://getjamcast.com/.

[17] "XBMC (Kodi)." https://kodi.tv/.

[18] "Universal Media Server." http://www.universalmediaserver.com/.

[19] "TVMOBiLi." http://www.tvmobili.com/.

[20] "PS3 Media Server." https://github.com/ps3mediaserver/ps3mediaserver.

[21] "UPnP Forum." http://upnp.org/.

[22] V. Lortz and M. Saaranen, "DeviceProtection:1 Service," pp. 1–67, 2011.

[23] Carl Ellison and Intel Corporation, "DeviceSecurity:1," pp. 1–66, 2003. http://www.upnp.org/specs/sec/UPnP-sec-DeviceSecurity-v1-Service.pdf.

[24] D. Garcia, "UPnP Mapping," *Defcon 2011*, 2011.

[25] Apple Inc., "Bonjour Overview," 2006. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html.

[26] "iTunes 4.5 Authentication Cracked - Slashdot." http://beta.slashdot.org/story/45544.

[27] "crazney.net - iTunes stuff." http://craz.net/programs/itunes/.

[28] "Amarok Music Sharing DAAP." https://community.kde.org/Amarok/Archives/Amarok_1.4/User_Guide/Music_Sharing.

[29] "Rhythmbox GNOME - DAAP Share Source." https://help.gnome.org/users/rhythmbox/stable/daap.html.en.

[30] "DAAP in XBMC." http://kodi.wiki/view/ITunes_(DAAP).

[31] "Banshee DAAP Plug-in." http://linuxappfinder.com/package/banshee-daap.

[32] "Roku Forums - Attention iTunes Users!." http://forums.roku.com/viewtopic.php?p=55884, September 2006.

[33] "DAAP Licensing | nanocr.eu." http://nanocr.eu/2007/02/07/daap-licensing/, 2007.

[34] "Roku Forums - Pinnnacle and DAAP." http://forums.roku.com/viewtopic.php?t=17116, June 2008.

[35] M. Rosenbach, "Troublesome Trojans: Firm Sought to Install Spyware Via Faked iTunes Updates - SPIEGEL ONLINE." http://spon.de/adv5r, 2011.

[36] J. Cox, "Apple's Bonjour/AirPlay poses network challenges for IT | CITEworld." http://www.citeworld.com/article/2115354/mobile-byod/apples-bonjourairplay-poses-network-challenges-it.html, July 2012. CITEworld.

[37] "Digital Audio Access Protocol - Protocol documentation v0.2." http://www.gyfgafguf.dk/raspbian/daapdocs.txt.

[38] Cisco, "Cisco - Example of the Primary SNMP Components." http://www.cisco.com/c/en/us/td/docs/optical/15000r5_0/15310/troubleshooting/guide/310r50ts/310snmp.html#wp29007.

[39] J. Ostell, "Using ASN.1 (Abstract Syntax Notation 1): A Data Description Language." http://www.nal.usda.gov/pgdic/Probe/v2n2/using.html, 1992.

[40] K. McCloghrie, K. McCloghrie, J. Schoenwaelder, and D. Perkins, "RFC 2578: Structure of Management Information Version 2 (SMIv2),"

[41] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "RFC 1441: Introduction to Version 2 of The Internet-standard Network Management Framework," April 1993.

[42] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "RFC 1901: Introduction to Community-based SNM," January 1996.

[43] G. Waters, "RFC 1910: User-based Security Model for SNMPv2," 1996.

[44] J. Case, R. Mundy, D. Partain, and B. Stewart, "RFC3410: Introduction and Applicability Statements for Internet Standard Management Framework," 2002. https://tools.ietf.org/html/rfc3410.

[45] Moxa Inc., "Using SNMP for a Ethernet-Based Home Automation System." https://www.moxa.com/application/Using_SNMP_for_a_Ethernet_based_Home_Automation_System.htm.

[46] Santos F. S.,Cagnon J. A. and Silva E. C. G., "Sistema Integrado para o Gerenciamento Energético de Edifícios Utilizando o Protocolo de Rede de Computadores SNMP para a Integração da Produção Limpa e Sustentabilidade," 2013.

[47] J. Postel, "RFC 768 User Datagram Protocol," *ISI*, 1980.

[48] Jacobson, V. and Frederick, R. and Casner, S. and Schulzrinne, H., "RFC 3550 - RTP: A Transport Protocol for Real-Time Applications,"

[49] Schulzrinne, H. and Rao, A and Lanphier, R., "RFC 2326 - Real Time Streaming Protocol (RTSP)," 1998.

[50] "Xiph.org: Ogg." `https://xiph.org/ogg/`.

[51] "Ogg Vorbis - Better Than Mp3?." `http://h2g2.com/edited_entry/A6556511`, December 2005.

[52] "What bitrate does Spotify use for streaming?- Spotify." `https://support.spotify.com/pt/learn-more/faq/#!/article/What-bitrate-does-Spotify-use-for-streaming`.

[53] "FLAC - Free Lossless Audio Codec." `https://xiph.org/flac/index.html`.

[54] Fock, F and Katz, J, "SNMP4J - Free Open Source SNMP API for Java." `http://www.snmp4j.org/index.html`.

[55] "MIB Designer." `http://www.mibdesigner.com/`.

[56] "AgenPro." `http://www.agentpp.com/agen/agen.html`.

[57] Caprica Software, "VLCJ - API for Java Platforms." `http://www.capricasoftware.co.uk/projects/vlcj/index.html`.

[58] Hamming, Richard W, "Error Detecting and Error Correcting Codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[59] Wagner, Robert A. and Fischer, Michael J., "The String-to-String Correction Problem," *J. ACM*, vol. 21, pp. 168–173, Jan. 1974.

[60] Vladimir I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," 1965. English translation in Soviet Physics Doklady, 10(8):707-710, 1966.

[61] Vreda Pieterse and Paul E. Black, "Levenshtein Distance,"

[62] Taniar, David and Leung, Clement HC and Rahayu, Wenny and Goel, Sushant, *High Performance Parallel Database Processing and Grid Databases*, vol. 67. John Wiley & Sons, 2008.

Appendix A

MUSIC-MIB

In this Appendix, it is shown the MUSIC-MIB tree, the MUSIC-MIB in Entity-Relationship Model and the description of its objects.

Album (Table) 1.3.6.1.3.2.1.2

albumTitle 1.3.6.1.3.2.1.2.1.2 - OctetString - The album title of a music track.

label 1.3.6.1.3.2.1.2.1.3 - OctetString - Label name of the album.

year 1.3.6.1.3.2.1.2.1.4 - Year in which the album was released.

ghostAlbum 1.3.6.1.3.2.1.2.1.5 - Integer32 - If there is no information available regarding the album title, this object will have the value of "1".

Artist (Table) 1.3.6.1.3.2.2.2

artistName 1.3.6.1.3.2.2.2.1.2 - OctetString - Name of the artist.

ghostArtist 1.3.6.1.3.2.2.2.1.3 - Integer32 - If there is no information available regarding the artist name, this object will have the value of "1".

Channel (Table) 1.3.6.1.3.2.3.2

channelDescription 1.3.6.1.3.2.3.2.1.2 - OctetString - Textual description of a channel.

Collection (Table) 1.3.6.1.3.2.4.2

collectionName 1.3.6.1.3.2.4.2.1.2 - OctetString - Collection name.

collectionDescription 1.3.6.1.3.2.4.2.1.3 - OctetString - Textual description of a collection.

numberOfMusics 1.3.6.1.3.2.4.2.1.4 - Integer32 - Number of music tracks that exist in a collection.

Device (Table) 1.3.6.1.3.2.5.2

deviceAddress 1.3.6.1.3.2.5.2.1.2 - OctetString - The IP address of a device in the network.

deviceType 1.3.6.1.3.2.5.2.1.3 - OctetString - The type of the device, if it is a dedicated server, a desktop/notebook or a smartphone.

deviceLocation 1.3.6.1.3.2.5.2.1.4 - OctetString - Physical location of the device (e.g. bedroom, living room).

deviceDescription 1.3.6.1.3.2.5.2.1.5 - OctetString - Textual description of the device.

Format (Table) 1.3.6.1.3.2.8.2

format 1.3.6.1.3.2.8.2.1.2 - OctetString - The format of a music track (e.g. Ogg, FLAC).

Gateway (Table) 1.3.6.1.3.2.10.2

gatewayAddress 1.3.6.1.3.2.10.2.1.2 - OctetString - The gateway's IP address.

gatewayPort 1.3.6.1.3.2.10.2.1.3 - Integer32 - The gateway's application port.

gatewayType 1.3.6.1.3.2.10.2.1.4 - OctetString - Type of a gateway (e.g. SQL, iTunes).

Genre (Table) 1.3.6.1.3.2.11.2

genreName 1.3.6.1.3.2.11.2.1.2 - OctetString - The genre name of a music track (e.g. Rock, Pop, Classic, Jazz).

ghostGenre 1.3.6.1.3.2.11.2.1.3 - Integer32 - If there is no information available regarding the genre, this object will have the value of "1".

Music (Table) 1.3.6.1.3.2.13.2

title 1.3.6.1.3.2.13.2.1.2 - OctetString - The title of a music track.

trackNumber 1.3.6.1.3.2.13.2.1.3 - Integer32 - The track number of the music track.

size 1.3.6.1.3.2.13.2.1.4 - OctetString - The size in megabytes (MB) of the music file.

duration 1.3.6.1.3.2.13.2.1.5 - OctetString - The duration of the music track in the format of "Minutes:Seconds".

PartAlbum (Table) 1.3.6.1.3.2.14.2

partNumber 1.3.6.1.3.2.14.2.1.2 - Integer32 - This object represents the CD number in an album.

Playlist (Table) 1.3.6.1.3.2.15.2

playlistName 1.3.6.1.3.2.15.2.1.2 - OctetString - Name of a playlist defined by the user.

Server (Table) 1.3.6.1.3.2.17.2

serverAddress 1.3.6.1.3.2.17.2.1.2 - OctetString - The IP address of a Audio SNMP server.

serverName 1.3.6.1.3.2.17.2.1.3 - OctetString - The server's name.

serverDescription 1.3.6.1.3.2.17.2.1.4 - OctetString - Textual description of the server.

serverLocation 1.3.6.1.3.2.17.2.1.5 - OctetString - Server's location.

Statistics (Table) 1.3.6.1.3.2.20.2

like 1.3.6.1.3.2.20.2.1.2 - Integer32 - This object represents if the user liked or disliked a music track. It defines the value "0" for dislike, "1"(default value) for neither dislike or like and "2" if the user likes a music track.

quality 1.3.6.1.3.2.20.2.1.3 - Integer32 - Ranging values from "0" (no quality) to "4" (high quality). This value is also defined by the user and represents the user's classification of music audio track quality.

timesPlayed 1.3.6.1.3.2.20.2.1.4 - Integer32 - Counters how many times did the user played that music track.

SubGenre (Table) 1.3.6.1.3.2.21.2

subgenreName 1.3.6.1.3.2.21.2.1.2 - OctetString - The sub-genre of a music track (e.g. Heavy-Rock, Alternative-Rock, New-Jazz).

ghostSubgenre 1.3.6.1.3.2.21.2.1.3 - The selected objects were chosen for the MUSIC-MIB, in order to satisfy the systems functionality requirements.Integer32 - If there is no information available regarding the sub-genre, this object will have the value of "1".

User (Table) 1.3.6.1.3.2.22.2

username 1.3.6.1.3.2.22.2.1.2 - OctetString - The user name, which identifies a user in the system.

password 1.3.6.1.3.2.22.2.1.3 - OctetString - The user's secret password.

active 1.3.6.1.3.2.22.2.1.4 - Integer32 - Range from "0" (false) to "1"(true), depending if the user is currently active or not.

state 1.3.6.1.3.2.22.2.1.5 - Integer32 - State of the user, if he is listening or paused to a music track, or if he is not listening anything. The values range from "0" to "2", depending on what the user is doing at the current moment.

accessType 1.3.6.1.3.2.22.2.1.6 - Integer32 - Type of access which a user is allowed to.

allowStatistics 1.3.6.1.3.2.22.2.1.7 - Integer32 - When this object is true (value "1"), it allows an associated user to use/view statistics.

playTime 1.3.6.1.3.2.22.2.1.8 - OctetString - The exact time of a music track that a user is listening to at the moment. This is used when there is a need to "transfer" the music audio from one device to another.

playedOnPlaylist 1.3.6.1.3.2.22.2.1.9 - OctetString - Playlist name of the music track that is being listening to. When a user demands to change device while playing a music, it will continue to play from that playlist too.

lastLoggin 1.3.6.1.3.2.22.2.1.10 - OctetString -The date of user's last login in the system.

normalization 1.3.6.1.3.2.22.2.1.11 - Integer32 - Normalization is disabled if value is "0" and enabled when value is "1". Audio normalization brings the average or peak audio amplitude to a target audio level.

volume 1.3.6.1.3.2.22.2.1.12 - Integer32 - The level of audio volume, ranging values from 0 (silent) to 100 (loudest).

searchRelaxation 1.3.6.1.3.2.22.2.1.13 - Integer32 - Level of "relaxation" when a user search for a music title, an artist, an album or a playlist.

nMaxReplies 1.3.6.1.3.2.22.2.1.14 - Number of maximum replies that the client can receive from a server.

previousPlayed 1.3.6.1.3.2.22.2.1.15 - OctetString - Path of the last music track that the user has listened.

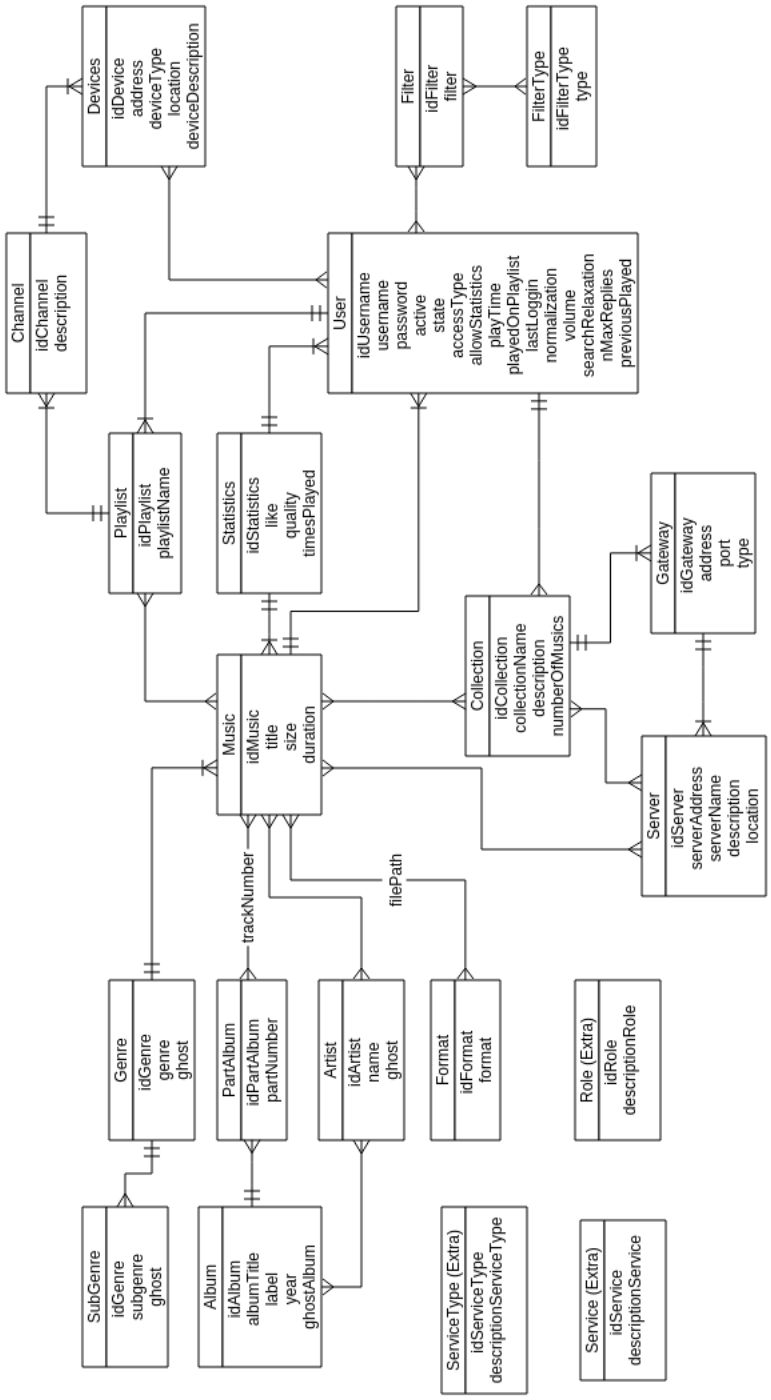


Figure A.1: MUSIC-MIB in Entity-Relationship Model.

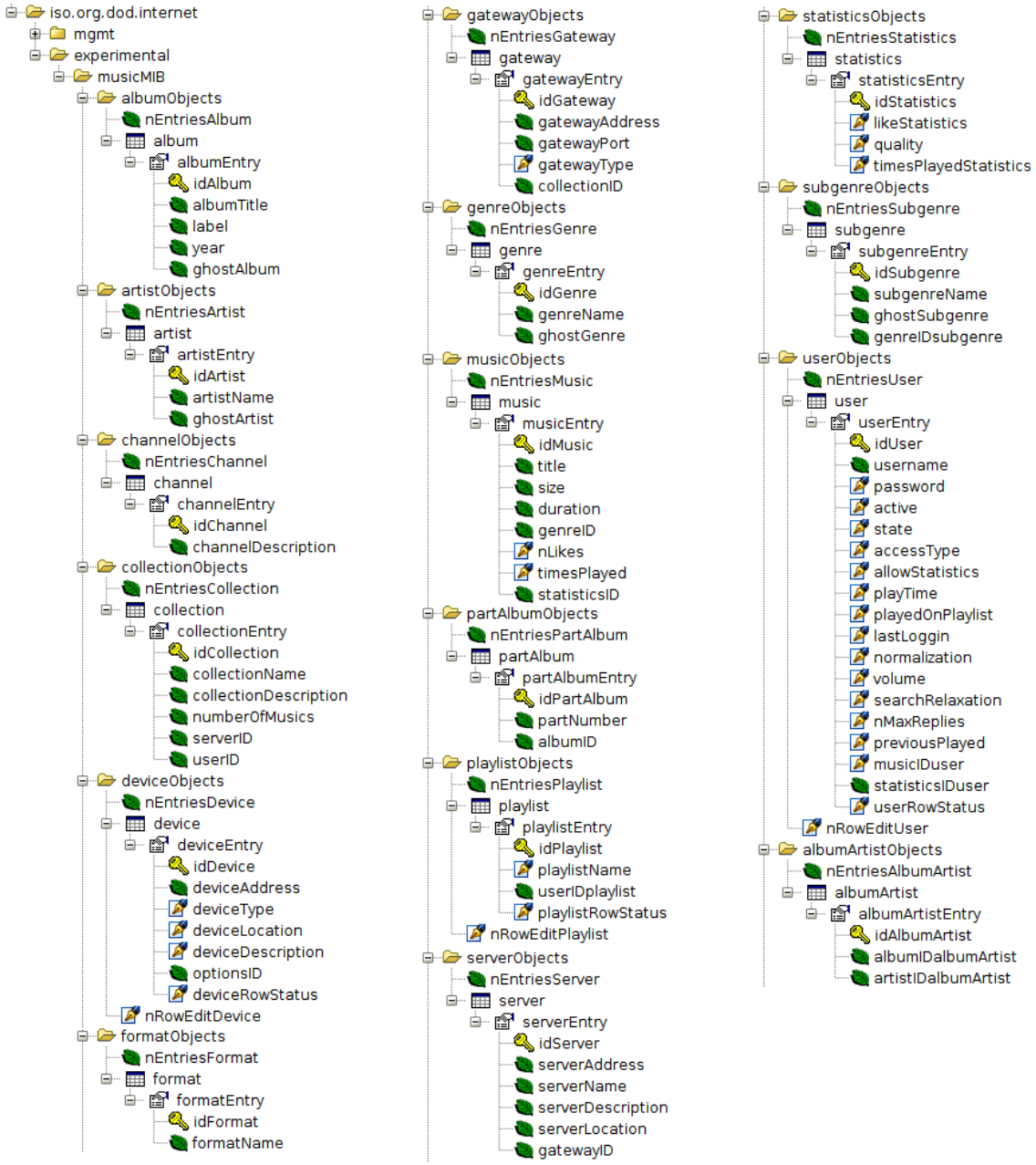


Figure A.2: MUSIC-MIB Tree (Part 1).

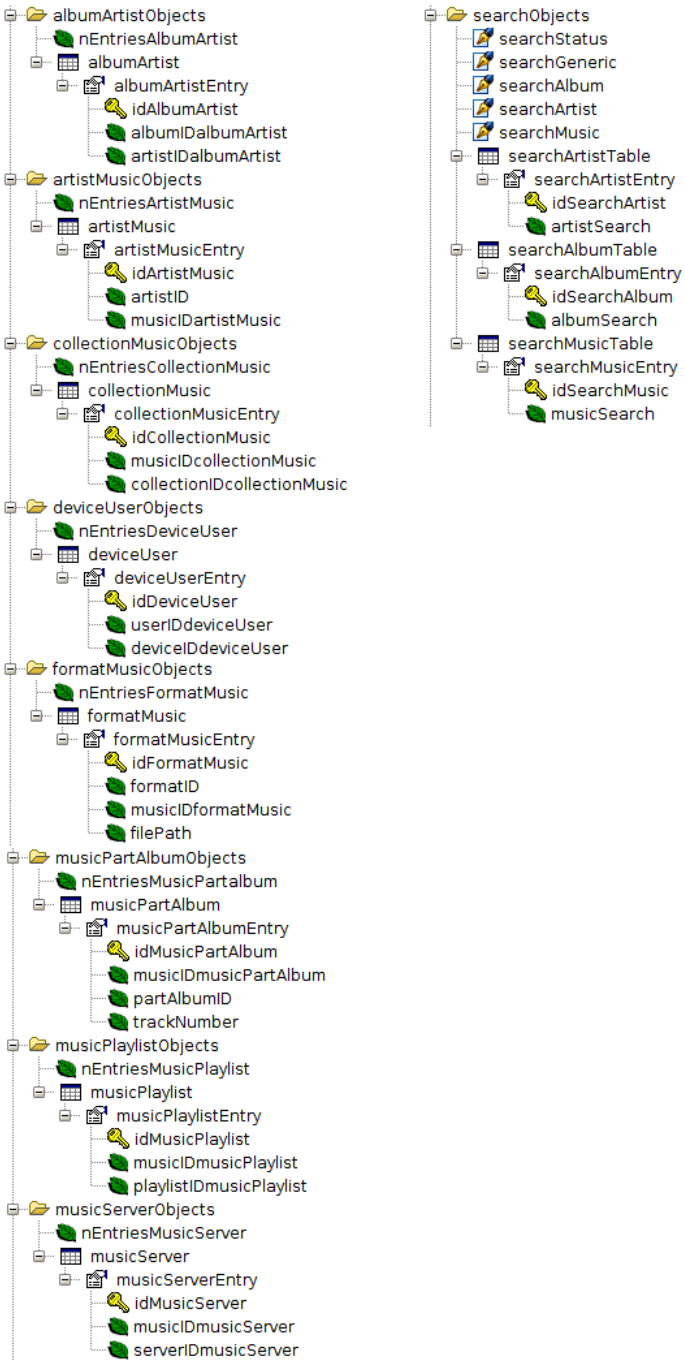


Figure A.3: MUSIC-MIB Tree (Part 2).

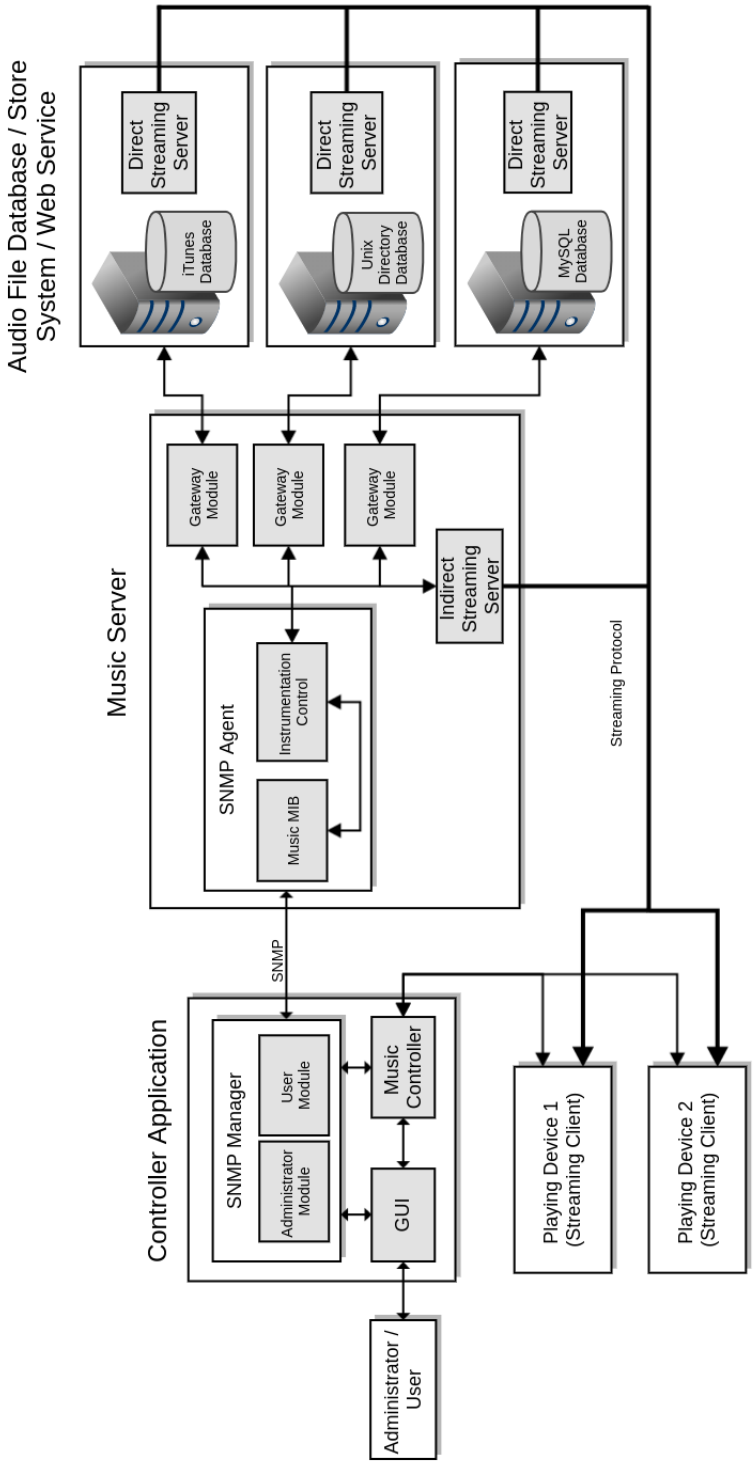


Figure A.4: Architecture diagram.