FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Web Services' Integration into a Peer-to-Peer BitTorrent Client

Francisco A. Barbosa

Thesis submitted for the degree of

Master in Electrical and Computers Engineering

Major in Telecommunications

Supervisor: Maria Teresa Andrade (Ph.D.)

Supervisor: Asdrúbal Costa (Ing.)

March, 2009

Resumo

Actualmente, quando se fala em computação distribuída e disseminação rápida de dados, a primeira tecnologia em que se pensa é em sistemas *peer-to-peer*. Este método alternativo de comunicação, por oposição à tradicional arquitectura cliente-servidor, permite que, numa rede, todos os nós comuniquem simultaneamente entre si, aumentando a rapidez e a eficiência das transmissões de dados.

Tomando em consideração este facto, não é pois de estranhar que esta seja a tecnologia adoptada no âmbito do projecto europeu MOSAICA, um projecto que pretende fornecer uma plataforma para que conteúdos multimédia de diversas culturas, etnias e religiões possam ser disseminados por todo o mundo, numa tentativa de promover a igualdade e tolerância entre povos e combater as diferenças culturais através do conhecimento das mesmas.

Esta dissertação pretende não só analisar as tecnologias inerentes à rede MOSAICA, como também contribuir com ferramentas que tornem este projecto mais próximo do seu objectivo: fazer com que os conteúdos que circulam na rede MOSAICA cheguem a qualquer lado e que possam ser acedidos a partir de qualquer lado, com a máxima simplicidade possível. Em particular, o objectivo desta tese é o de especificar e desenvolver uma aplicação *Web* e respectivos módulos de suporte, tornando possível a interacção com um cliente *BitTorrent*, permitindo a qualquer utilizador com ligação Internet e um *browser Web* usufruir das mesmas vantagens que um utilizador de redes *peer-to-peer*, podendo aceder aos conteúdos distribuídos nessa rede, com hipótese de transferir os mesmos para o seu computador, sem necessidade de estar associado à rede *peer-to-peer* e, consequentemente, sem necessidade de instalar qualquer tipo de software *peer-to-peer*.

Simultaneamente, é proposto um mecanismo que, funcionando em conjunto com o cliente *BitTorrent* escolhido - Vuze -, e de uma forma totalmente transparente para o utilizador, impeça que um conteúdo se torne raro ou mal distribuído na rede MOSAICA, através de consultas periódicas, e para cada conteúdo, do número de *seeders* no *swarm*.

Abstract

Nowadays, when somebody talks about distributed computation and widely distributed data, the first thing that comes to people's mind is peer-to-peer technology. Peer-to-peer communication comes as an alternative for the traditional server-client architecture, allowing all nodes in a network to communicate simultaneously with each other, increasing the efficiency and quickness of data's exchange.

It's no surprise that, with all this advantages, peer-to-peer architecture became the one to be used in the european project MOSAICA, a project that aims to provide a platform for multimedia contents of different cultures, ethnics and religious beliefs to be widely spread by the globe, attempting to promote equality and tolerance among people, fighting cultural differences by sharing knowledge.

This dissertation intends not only to analyze the different technologies involved in MOSAICA network but also to provide tools that take MOSAICA closer to its objective: help contents reach anywhere and be available from anywhere, with the utmost simplicity. In particular, the objective of this thesis is to specify and develop a web application and its support modules to interact with a BitTorrent client, allowing users with an Internet connection and a Web browser to enjoy the same advantages available to peer-to-peer users, like accessing contents available on that network, with possibility of transferring them to their computer without the need to be connected to the peer-to-peer network and therefore no need to install any peer-to-peer software.

Simultaneously, it is proposed a mechanism that, working together with the chosen BitTorrent client - Vuze -, and in a completely transparent way to the user, prevents contents to become rare or poorly distributed in the MOSAICA network by periodically controlling the number of seeders in the swarm.

Acknowledges

I would like to acknowledge my supervisors, Dr. Maria Teresa Andrade and Ing. Asdrúbal Costa, for the enormous assistance and support they both gave me, which was fundamental to the accomplishment of this thesis. Thank you both for providing me some documents to examine and ideas to implement in the developed software, as well as suggestions on the best way to perform some tasks and remarkable suggestions to the early versions of this thesis.

I would also like to express my special gratitude to my family - my parents Francisco and Odete, my sister Helga, Hélder and Tomás -, for their huge encouragement and the constant optimism about my future, and to Vera, for all her love and support. It is to them that I dedicate this thesis.

Finally, I would like to thank my friends for all the great moments we spent together, which also contributed to the achievement of my goals.

The Author

"Life is pretty simple: You do some stuff. Most fails. Some works.
You do more of what works.If it works big, others quickly copy it.
Then you do something else. The trick is the doing something else."

Leonardo da Vinci

Contents

1 Introduction **1**
1.1 The MOSAICA Project . . . 1
1.2 Goals . . . 3
1.3 Dissertation's Structure . . . 4

2 Used Technologies **5**
2.1 Peer-to-Peer Architecture . . . 5
2.1.1 P2P Generations . . . 7
2.1.2 P2P Network's Topologies . . . 8
2.1.3 MOSAICA P2P Network . . . 12
2.2 BitTorrent . . . 12
2.2.1 The Protocol . . . 13
2.2.2 BitTorrent algorithms . . . 16
2.2.3 Distributed Hash Tables (DHTs) . . . 17
2.3 Service-Oriented Architecture (SOA) . . . 18
2.3.1 eXtensible Markup Language (XML) . . . 20
2.3.2 Web Services . . . 21
2.3.3 SOAP . . . 25
2.3.4 Java Remote Method Invocation . . . 28
2.4 Summary . . . 29

3 State of the Art **31**
3.1 Peer-to-Peer . . . 31
3.1.1 JXTA Platform: Framework to P2P networks . . . 31
3.1.2 Helpers: A new concept of peer . . . 34
3.1.3 P4P: Proactive Network Provider Participation for P2P . . . 35
3.2 BitTorrent . . . 36
3.2.1 Top-BT: An Infrastructure Free BitTorrent Client . . . 36
3.2.2 Vuze . . . 36
3.3 Web Services . . . 37
3.3.1 Apache Axis2 . . . 37
3.3.2 Representational State Transfer (REST) . . . 38
3.3.3 WSPDS: Web Services Peer-to-Peer Discovery Service . . . 39
3.3.4 WS^{EXP}: A tool for experimenting with Web Services . . . 39
3.4 SOAP . . . 40
3.4.1 SOAP Optimization via parameterized Client-Side Caching . . . 40
3.4.2 Wireless SOAP: Optimizations for Mobile Wireless Web Services . . . 41
3.5 Summary . . . 42

4 The Project **43**
4.1 Introduction to the developed components 44
4.2 Development Environment 48
4.3 Web Services 48
4.3.1 *Get Content* Web Service 50
4.3.2 Apache Configuration Checker 53
4.3.3 *List Azureus' Activities* Web Service 56
4.3.4 Web Services PHP clients 58
4.4 Vuze: The BitTorrent Client 58
4.4.1 Vuze Remote Invocation Methods 60
4.4.2 RSS Import Plugin 60
4.4.3 Shared Folder's Maximum Size Controller Applet 65
4.4.4 Seed Limiter Plugin 68
4.5 Deployment 72

5 Analysis of the developed software **75**
5.1 Performance of Web Services 76
5.1.1 Get Content Web Service Tests 76
5.1.2 List Azureus' Activities Web Service Tests 77
5.2 Functional Tests 84
5.3 Distribution of contents 90
5.4 Conclusions of performed tests 92

6 Conclusions and Future Work **93**
6.1 Objectives' Achievement 93
6.2 Future Work 94

References **96**

List of Figures

2.1 Network architectures 6
2.2 Partially Centralized P2P architecture 9
2.3 Hybrid Decentralized P2P architecture 10
2.4 Probability of success under various TTLs 10
2.5 Average response time of search mechanisms used in structured and unstructured networks 11
2.6 P2P decision tree 12
2.7 Number of active peers over time 15
2.8 DHT API 18
2.9 Basic SOA 19
2.10 SOA entities and operations 19
2.11 Web Services Conceptual Architecture 22
2.12 Performance of different distributed computing technologies 22
2.13 SOAP message's exchange 26
2.14 SOAP routing capability 26

3.1 JXTA Software Architecture 33
3.2 Helpers' influence in multiple configurations 35
3.3 Internet traffic along the years 36
3.4 Axis2 Simple Object Access Protocol (SOAP) messages handling 37
3.5 Comparison of SOAP with client-side caching with JavaRMI and traditional SOAP 41

4.1 Initial P2P-Content Management System (CMS) Deployment Diagram for MOSAICA Peer Deploy Development package 44
4.2 Initial P2P-CMS Use Cases Diagram 46
4.3 Integration with the MOSAICA Peer Deploy Development package 47
4.4 Integration with the MOSAICA Final User package 47
4.5 P2P-CMS Use Cases Diagram 49
4.6 Class Diagram for *Get Content* Web Service 51
4.7 Sequence Diagram for *Get Content* Web Service 52
4.8 Unified Modeling Language (UML) Collaboration Diagram for *Get Content* Web Service 53
4.9 Class Diagram for Apache Configuration Checker 54
4.10 Collaboration Diagram for Apache Configuration Checker 54
4.11 Sequence Diagram for Apache Configuration Checker 55
4.12 Class Diagram for *List Azureus Activities* Web Service 56
4.13 Sequence Diagram for *List Azureus' Activities* Web Service 57
4.14 Collaboration Diagram for *List Azureus Activities* Web Service 57

4.15 Vuze UML Use Cases 59
4.16 Class Diagram for Azureus Remote Methods 60
4.17 Class Diagram for Vuze's plugin *RSS Import* 63
4.18 Sequence Diagram for Vuze's plugin *RSS Import* 64
4.19 Collaboration Diagram for Vuze's plugin *RSS Import* 64
4.20 Class Diagram for Shared Folder's Maximum Size Controller Applet 66
4.21 Sequence Diagram for Shared Folder's Maximum Size Controller Applet 67
4.22 Collaboration Diagram for Shared Folder's Maximum Size Controller Applet . . 68
4.23 Class Diagram for Vuze's plugin *Seed Limiter* 70
4.24 Sequence Diagram for Vuze's plugin *Seed Limiter* 71
4.25 Collaboration Diagram for Vuze's plugin *Seed Limiter* 71
4.26 Three-layer model for MOSAICA platform 72
4.27 MOSAICA's developed components 73
4.28 Final P2P-CMS Deployment Diagram for MOSAICA Peer Deploy Development package 74

5.1 Comparison of minimum response times in List Azureus' Activities Web Services 78
5.2 Comparison of maximum response times in List Azureus' Activities Web Services 79
5.3 Comparison of average response times in List Azureus' Activities Web Services . 80
5.4 Comparison of minimum response times in Get Content Web Services 81
5.5 Comparison of maximum response times in Get Content Web Services 82
5.6 Comparison of average response times in Get Content Web Services 83

List of Tables

2.1 Advantages and disadvantages of Peer-to-Peer (P2P) networks according to its centralization level . . . 11
2.2 Advantages and disadvantages of P2P networks according to its structural architecture . . . 12

4.1 Get Content use case description . . . 50
4.2 Download Content through HTTP use case description . . . 50
4.3 List Azureus' Activities use case description . . . 56
4.4 Check Occupied Disk Space use case description . . . 61
4.5 Define Disk Space use case description . . . 62
4.6 Download Contents from Feeds use case description . . . 62
4.7 Define Seeds' Number Limit use case description . . . 69
4.8 Define Recheck Time for SeedLimiter Plugin use case description . . . 69
4.9 Check Contents' Seeds use case description . . . 69
4.10 Remove Content by SeedLimiter's order use case description . . . 69

5.1 Get Content Web Service Local Host Test . . . 76
5.2 Get Content Web Service Remote Host Test . . . 76
5.3 List Azureus' Activities Web Service Local Host Test . . . 77
5.4 List Azureus' Activities Web Service Remote Host Test . . . 77
5.5 Functional Test 1 . . . 84
5.6 Functional Test 2 . . . 84
5.7 Functional Test 3 . . . 84
5.8 Functional Test 4 . . . 85
5.9 Functional Test 5 . . . 85
5.10 Functional Test 6 . . . 85
5.11 Functional Test 7 . . . 85
5.12 Functional Test 8 . . . 86
5.13 Functional Test 9 . . . 86
5.14 Functional Test 10 . . . 86
5.15 Functional Test 11 . . . 86
5.16 Functional Test 12 . . . 86
5.17 Functional Test 13 . . . 87
5.18 Functional Test 14 . . . 87
5.19 Functional Test 15 . . . 87
5.20 Functional Test 16 . . . 87
5.21 Functional Test 17 . . . 88
5.22 Functional Test 18 . . . 88

5.23 Functional Test 19 88
5.24 Functional Test 20 89
5.25 Functional Test 21 89
5.26 Distribution of Contents Test 1 90
5.27 Distribution of Contents Test 2 91
5.28 Distribution of Contents Test 3 91

Acronyms

API Application Programming Interface

CAN Content Addressable Network

CGI Common Gateway Interface

CMS Content Management System

CORBA Common Object Request Broker Architecture

CPU Central Processing Unit

CSS Cascading Style Sheets

DAML-S DARPA agent markup language for services

DCOM Distributed Component Object Model

DHT Distributed Hash Table

DNS Domain Name System

DTD Document Type Definition

ERP Endpoint Routing Protocol

ETA Estimated Time of Arrival

FTP File Transfer Protocol

GIS Geographic Information System

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

IP Internet Protocol

ISP Internet Service Provider

JDK Java Development Kit

JVM Java Virtual Machine

MEP Message Exchange Pattern

NAT Network Address Translation

OWL Web Ontology Language

P2P Peer-to-Peer

PBP Pipe Binding Protocol

PDP Peer Discovery Protocol

PIP Peer Information Protocol

PHP Hypertext Preprocessor

PRP Peer Resolver Protocol

REST Representational State Transfer

RLF Rarest Local First

RMI Remote Method Invocation

RPC Remote Procedure Call

RSS Really Simple Syndication

RVP Rendezvous Protocol

SHA Secure Hash Algorithm

SMTP Simple Mail Transfer Protocol

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

TIS Temporal Information System

TTL Time To Live

UDDI Universal Description, Discovery and Integration

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML eXtensible Markup Language

W3C World Wide Web Consortium

WSDL Web Service Definition Language

WWW World Wide Web

Chapter 1

Introduction

The work conducted during this thesis focussed on the MOSAICA project, adding new functionalities to it, having one major goal in mind: everybody can use it, independently of place, knowledge, operating system and, in a deeply context, independently of culture, race or religious beliefs.

1.1 The MOSAICA Project

MOSAICA - Semantically Enhanced Multifaceted Collaborative Access to Cultural Heritage is a research and development project, co-funded by the European Commission. It started in June, 2006 and its duration is two and a half years. This project is being carried out by a consortium of eleven organizations from eight different countries, gathering expertises from Information Technologies and Culture fields. INESC Porto is responsible for the design and development of the MOSAICA distributed content management system.

This project aims to promote cultural, religious and racial pluralism by distributing cultural heritage contents, spreading knowledge and habits from every culture to any other cultures. This project's beliefs are that, by knowing different cultures/religions, tolerance and comprehension between people can be increased, abolishing "walls" that still exists, in the form of racism and xenophobia.

MOSAICA intends to be more than a merely Web portal for access and presentation of cultural heritage from different cultures. To accomplish this goal, MOSAICA planned to use some of the most advanced technological resources, each of which transforms the act of surfing in this Web portal into a full-featured multimedia experience.

Online semantic annotator is one of the utilities available, allowing users to link contents with free text annotations, that can be simple comments, instructions to other users regarding contents' use or sophisticated associations of cultural objects with relevant ontological concepts. This additional and semantically related information, apart the content itself, is known as *metadata* and aims to automatically enrich cultural contents [1].

This utility is empowered with the use of an **online ontology editor**, that allows the edition of *metadata*, and provides two different interfaces: one for regular users and domain experts and other for ontology engineers [2]. While the second interface has an integrated full-featured Web Ontology Language (OWL) editor, the first one is much simpler, allowing edition of contents' fields such as *index type*, *property* and/or *value*.

These tools rely on semantic annotation, a concept brought to life thanks to Semantic Web initiative [3] and to the development of ontology interoperability standards, to which the wide adoption of OWL has largely contributed. Semantic Web was born with the growing need of annotating audio-visual contents in an increasing multimedia-oriented Web, mainly with the massive adoption of MPEG and MPEG-based formats.

Also related with Semantic Web technology and ontologies, MOSAICA integrates semantics with Geographic Information System (GIS)/Temporal Information System (TIS), providing data from multiple ontologies, dynamically and online. This innovation, although already discussed on research papers, has never been attempted before.

One other innovation has to do with **Distributed Content Management**. Due to its nature, MOSAICA may deal with a high load of contents and users, and this fact implies the use of a distributed architecture, because traditional server-client architecture had already demonstrated some weaknesses with high number of nodes. To accomplish this goal, MOSAICA has adopted the P2P paradigm to implement its content management system, thus benefiting from the high availability and transfer rates typical in P2P filesharing systems.

MOSAICA also needs an efficient search mechanism, one that can be both reliable and fast as well as flexible to allow semantic based searches. Whereas DHTs being used in the new generation of structured P2P networks enables very fast search times by providing a distributed structure of indices used for searches, it does not allow to perform searches based in proximity, and therefore semantic-based searches, but only exact-match searches. This lead to the use DHTs at the BitTorrent layer, for finding peers, and semantic searches are performed through exact-keyword lookup in distributed tables, in a separate and unstructured layer - the JXTA layer.

The adoption per se of the P2P paradigm for the deployment of the MOSAICA content management system (P2P-CMS) does not fulfill all the MOSAICA requirements. Accordingly, the

developed system, although relying on the use of P2P technologies and protocols, presents a two-layer architecture with a distributed overlay devoted to search functionality and accessed via Web Services. These layers consist of the BitTorrent layer, where Vuze BitTorrent clients run, and the JXTA layer, where peers can connect to the MOSAICA network and perform searches, using the JXTA framework, and publish Web Services, using Axis2.

Finally, a **Virtual Expedition Maker** allow users to design a Virtual Expedition through cultures, save them, add instructions and/or activities, enriching this multimedia interactive experience.

1.2 Goals

This dissertation's objective is to specify and develop a web application that interacts with Vuze BitTorrent client, so that users with an Internet connection and a web browser can search and download contents, namely cultural heritage oriented ones, from a P2P network, without the need of installing software nor act as peers of the network. This application interacts with MOSAICA P2P network, which requires a study of some of the technologies used in it, to a better integration. In the MOSAICA project, as said before, the network is based on a peer-to-peer infrastructure, using the BitTorrent suite of protocols. Accordingly, a thorough study was done of the BitTorrent protocol, commonly used in structured P2P networks, in order to have a clean understanding of the operations executed in the network, namely how connections are established, which methods are available in the protocol and the type of information exchanged between peers beyond contents' data.

Establishing and using a BitTorrent network, within the MOSAICA system, requires the deployment of BitTorrent clients in the MOSAICA peers. The MOSAICA project has chosen Vuze (formerly known as Azureus), a well-known Java BitTorrent client implementation, currently in its fourth version. As an open-source project and with strong plugin capabilities, Vuze turns to be a premier choice.

In the context of the work proposed for this thesis, it was foreseen to deliver a modified version of the Vuze BitTorrent client whereby the usability would be improved as well as fairness of use. Although Vuze already provides optimizations for sharing contents, in MOSAICA it was intended to go a step further, implementing the means to fight against poor distribution of contents and uneven usage, as well as to simplify as much as possible the access to media resources distributed across the MOSAICA P2P system. In P2P networks nothing forbids one BitTorrent user of stopping sharing contents once he completes his downloads, thus downloading more data than he had uploaded - users who do this are commonly known by *leechers*. The adopted solution was to develop a set of plugins to the Vuze client which, once installed and running, start to download

and seed poorly distributed contents, transparently to the user. To download these contents, Vuze can retrieve *torrent* files from a Really Simple Syndication (RSS) server and while uploading - or seeding, once Vuze has the content fully downloaded - it ensures that a content is always available with a minimum number of seeders - peers which have a complete copy of the content - minimizing stalled downloads. To the user only is left the control of the disk space used by Vuze, that can allow Vuze to use all the disk space if necessary, or use a portion of it.

However, these actions were not enough according MOSAICA's desire of getting cultural heritage to everywhere. Whoever already used or heard about BitTorrent tends to associate it with piracy or illegal contents, and this commonsense idea lead network administrators to implement policies that aim to eradicate all P2P traffic. So, how can someone access to cultural contents when is behind firewalls, NATs or restrictive network policies?

It is known that, in a network - even the public ones -, at the very least, HTTP traffic is allowed and correctly forwarded. This protocol allows users to "*surf the web*", using a browser to access *World Wide Web (WWW)* pages. These connections use port 80 by default, although it is common to use also port 8080. So, the answer to enable users to access cultural heritage using P2P networks is to use Web Services, a solution that enables a wide range of different services to be accessed by a browser. Together with this, an additional ease is also present: with Web Services, there's no need to install any additional software in the client machine, allowing users to enjoy MOSAICA in any common computer, with any operating system (all currently used operating systems have browsers and HTML/PHP interpreters), even when users have limited privileges.

Together with Web Services, it was also required as beneficial to have the means to control the amount of disk space Vuze may use in each computer. This mechanism, that is controlled in Vuze interface, is also available under an *applet* form, that can be accessed by a browser, turning Vuze able to be controlled from anywhere, without the need to install any additional software or change local permissions.

1.3 Dissertation's Structure

This document is organized according with the following structure. The current chapter is an introduction to this dissertation. In chapter 2, all used technologies are explained and analyzed. Chapter 3 completes this information by providing a description of the State-of-the-Art of the most relevant technologies. Chapter 4 describes the work done, explains the choices made and methods used. In chapter 5 an analysis of the project is provided, which allows extrapolation of conclusions, presented in chapter 6, along with references to future work that can still be made.

Chapter 2

Used Technologies

During the development of this dissertation, different technologies were used in order to accomplish all the proposed goals. This chapter provides a detailed description of all used technologies, so that all of them can be fully understood and its choice can be comprehended by all. As P2P is the base architecture of all the system, this will be the starting point of approached technologies.

2.1 Peer-to-Peer Architecture

When Internet suffered its *boom*, the traditional client-server architecture became insufficient to answer to all the needs that were emerging. Hardware became more sophisticated and faster, and each time more used to do complex tasks. If we think of, for instance, signal analysis (as in SETI@Home - Search for Extra Terrestrial Intelligence - project) or even the study of human genome (as in the Genome@Home project), which require complex and long duration calculus, it is easy to understand that there is not in the world a single machine fast enough to perform these, in a reasonable time. So the idea of using multiple computers working as one to do such tasks was born, giving birth to the concept of distributed computing, a way of sharing data, storage or CPU cycles [4] (like both examples mentioned above), being also found in communications, like instant messaging applications or Skype [5].

One of the facets of distributed computing is the Peer-to-Peer paradigm. With this type of network architecture, scalability and flexibility are largely increased. Instead of using a single server to multiple clients, all nodes in a network become simultaneously clients and servers, retrieving and offering data to other nodes - the peers. The concept of these architectures is shown in figure 2.1. The fact that peers have no fixed Internet Protocol (IP) address does not turn peers dependent of Domain Name Systems (DNSs), using instead another mechanisms to resolve them.

Based on this premise, Shirky [6] states that P2P applications must deal with variable duration connections and temporary network addresses as a norm, turning peers quite autonomous.

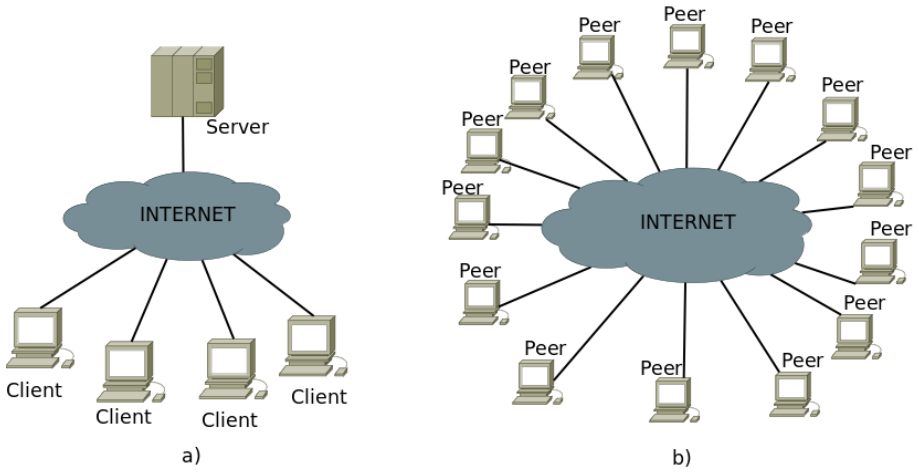


Figure 2.1: Network architectures
a)Client-server architecture; b) Peer-to-Peer architecture

Without the need of a centralized server - like, for instance, to coordinate connections - peers are able to establish directly connections between them. The first advantage of this is that it requires less bandwidth to transfer data, because data does not travel between the source to the server, and them from there to the destination, but it goes directly from the source peer to the destination peer. Together with these, P2P paradigm also allows:

- If one peer wants some data that multiple peers have, instead of establishing only one connection to the server, it can establish multiple connections, each to a different peer, increasing the speed of the data's transfer;
- If two or more peers want some data that is available in some other peers, they can connect each to a different peer, without having to share the upload capacity of only one data's holder, like they would in a client-server architecture, distributing network's load by the peers.

These two cases show how P2P can increase the speed of data's transfer. Of course that, in the second case, if only one peer have the desired data, the behaviour of the system will be similar to a client-server architecture network. However, when some peers complete the download, they will stop downloading and start uploading the same data they had previously downloaded, increasing network's resources and transfer's speed, which will be increased every time one more peer has concluded the download. This means that each peer that has downloaded a content will assume the role of source of that content, thus increasing the content's availability in the network and, consequentially, the transfer speed for new downloads. Popular contents, those that most people are

looking for, will thus be highly distributed and available, assuming a well dimensioned and "well behavioured" P2P network. This shows another characteristic of P2P networks: its efficiency.

Associated with the last advantage mentioned is also robustness: if there are a group of peers holding a content, and some of them disconnect, the others that remain connected assure the availability of that content to other peers who are downloading it or are going to begin downloading it. So, if some peer is downloading from some other peer, and the second goes away, the first only has to connect to other peer who has what he wants, and continue to download data. Supposing this scenario in a large network, with many peers, the number of nodes that join or leave the network become irrelevant because, when compared to the total number of peers in the network, its performance is not affected, turning P2P networks quite scalable.

Androutsellis-Theotokis et al. [4] propose the following definition of what are P2P systems:

> "Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize network topologies with the purpose of sharing resources such as content, Central Processing Unit (CPU) cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority".

2.1.1 P2P Generations

First generation of P2P networks was focused in decentralized networks, with quite basic search mechanisms, and total anonymous navigation - peers didn't know each others, creating a pure end-to-end connection. Also, user's connection was totally free, without any kind of access restriction or control.

Second generation was more sharing-oriented. It introduced the concept of *swarm* - a group of peers connected to one another -, enabling that a user with a certain content could upload it, when requested, to multiple users, simultaneously. Another innovation was that, to download a certain content, instead of downloading the whole file(s), each file was downloaded in small parts, that could be downloaded from different peers, and once all parts were fully downloaded, were then reassembled into the original form. In second generation, users' anonymity no longer existed, and every user could know which peer was connected to him or which peer he was connected to.

Third generation of P2P is still in an early stage, but it is becoming oriented to business and research, getting far away of the stigmata of piracy and being more and more used for legal purposes [7].

2.1.2 P2P Network's Topologies

P2P networks can be categorized according to its centralization and its structure, which implies different mechanisms in what concerns to searches performed and data's storage methods, along with network's maintenance. From the structural point of view, P2P networks can be *structured*, *unstructured* or a mixture of both topologies, being classified as *loosely structured*. Each of these networks' architectures can then adopt a different level of centralization, according with each node's role.

Being historically the first ones to be implemented, unstructured P2P networks have contents spread randomly by its peers, which implies that peers do not know where each content is located and consequently, when performing searches across the network, search mechanisms tend to be less scalable than those found in structure networks [4]. The first search mechanisms were based in queries propagated across the entire network, which later become more sophisticated and efficient, replacing the flood of queries through the network by the use of random paths - when receiving a query, peers replicate it to a random neighbour - or using history from past search results. Due to these facts, unstructured networks may be preferred in networks where the population consists of highly transient nodes and search mechanisms are keyword-based [8].

Opposing to unstructured P2P networks, structured networks are a consequence of an attempt to improve scalability issues found in unstructured systems. In these networks, the overlay topology is strictly controlled, because all the contents are stored at precise locations, and searches mechanisms are based in routes, using for that distributed routing tables, where contents and its location are mapped. Although these kind of networks have bigger concerns in what comes to maintenance - like inserting, updating or removing contents and its location, mainly with highly transient nodes -, exact-match based queries have great performances and searches have, most of the times, an high level of success. This implies, however, that users know exactly what to search, which is not always the case.

An attempt to solve issues of both topologies resulted in networks where contents' location is not completely specified, but searches are improved with routing hints, which prevents networks from being flooded with queries, and search times become smaller.

Each of these topologies can also be classified according with different centralization schemes, according to its peers' role. In *Purely Decentralized architectures* all peers perform the same role - they act simultaneously as clients and servers, and are commonly designated as "servents", mixing the words SERVers and cliENTS -, without the need for central coordination of their activities. The *Partially Centralized architectures* are a variant of purely decentralized architecture, in which some peers - the *supernodes* - are hierarchically superior, acting as local central indexes for files shared by its neighbours, as diagrammatically shown in figure 2.2. The existence of supernodes in

the network does not compromise network's liability, being these dynamically assigned and, when one fails, another one is elected, as long as it has enough bandwidth or processing power.

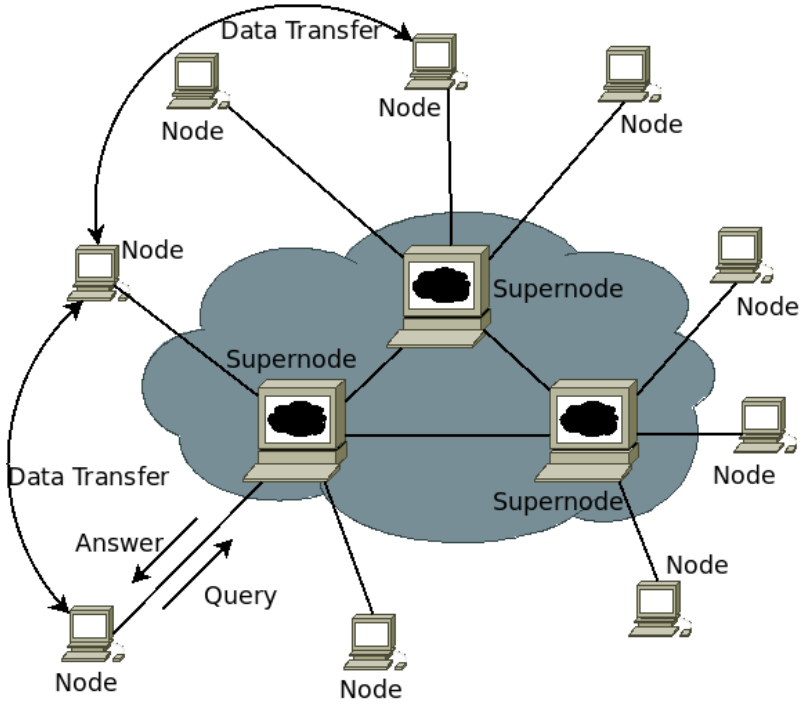


Figure 2.2: Partially Centralized P2P architecture

Opposing to purely decentralized architecture is *hybrid centralized architecture* (figure 2.3), in which there exists a central server that acts as a local concentrator for peers and maintains a directory with metadata of the contents, describing contents shared by peers, which improves searches by identifying the peers with searched contents. The server also coordinates peers' connections and gathers information about peers, like IP address, available bandwidth and files shared. The fact that is the server the entity responsible for connecting peers implies that, if the server goes offline, beyond the searches become unavailable, peers cannot connect to each other.

Centralization level of P2P networks implies different search methods. In purely decentralized networks - Gnutella is an example of a popular unstructured and purely decentralized P2P network -, all searches are nondeterministic, since peers have no way to guess where files may be located. Gnutella uses a flooding mechanism that propagate queries through the network. However, to prevent the entire network of being flooded with query messages, messages' headers contains a Time To Live (TTL) field, limiting the number of hops the query is propagated to. Together with this, messages are assigned with a unique identifier and hosts have dynamic routing tables, with messages identifiers and nodes' addresses, which prevents duplicated messages, improves

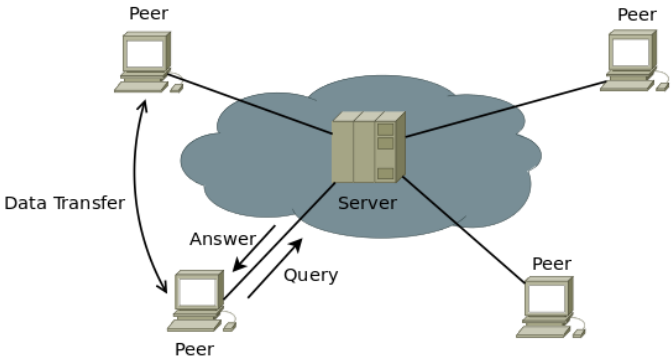


Figure 2.3: Hybrid Decentralized P2P architecture

searches' efficiency and preserves bandwidth. Figure 2.4, extracted from [8], shows the probability of success in four different network topologies according to the number of TTLs.

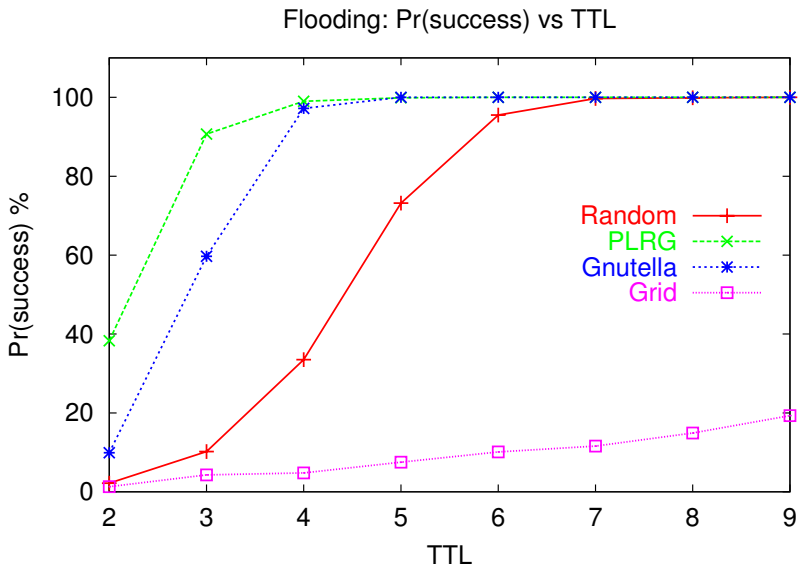


Figure 2.4: Probability of success under various TTLs

In partially centralized networks, like Kazaa, supernodes act as proxies, indexing the files shared by its neighbour peers. This way, when one peer generates a query, the search process is made at the supernode's level, which avoids propagation of queries to all peers, consequentially saving bandwidth, and increases searches' efficiency by consulting first the local supernode's index and only then, if no match is found, the query is propagated to other supernodes.

Searches performed in structured P2P networks benefit of different mechanisms for routing messages and locating data, being more efficient, although more complex, than those of unstructured networks. Freenet, Chord, Kademlia, Content Addressable Network (CAN), Pastry and Tapestry

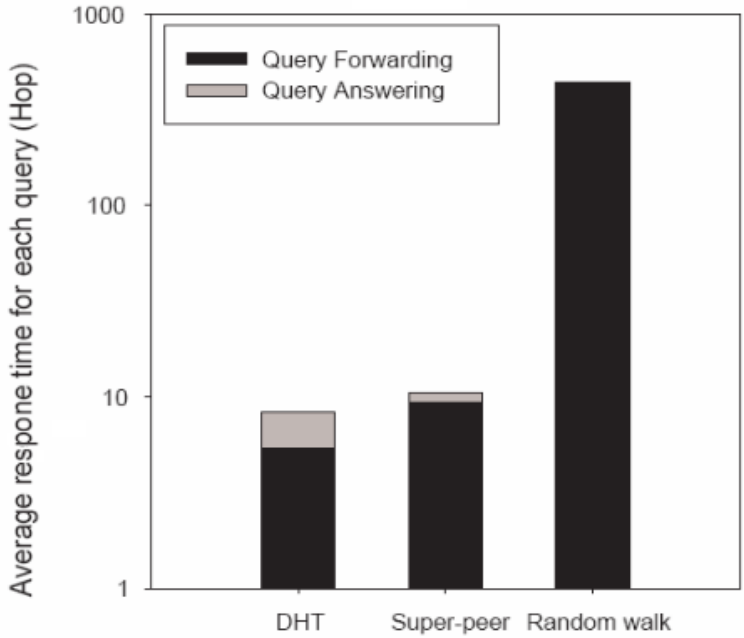


Figure 2.5: Average response time of search mechanisms used in structured and unstructured networks

are the most used in structured P2P architectures, and all of them are based in DHTs protocols, recording indexes (hashes) of files together with a location identifier. New approaches to DHT's search mechanisms, like addition of metadata to contents, and storing it together with contents' keys, have been developed, in order to improve the location of data when using incomplete information. Searches in structured networks, besides using more complex search algorithms, tend to obtain quicker results than those performed in unstructured networks, as it is comproved in figure 2.5, extracted from [9].

Advantages and disadvantages are summarized in tables 2.1 and 2.2, to provide a global view of each architecture.

Table 2.1: Advantages and disadvantages of P2P networks according to its centralization level

	Purely Decentralized Architecture	Partially Centralized Architecture	Hybrid Decentralized Architecture
Advantages	Availability of contents.	Better discovery times; Nodes are lightly loaded.	Simple implementation; Searches are quick and efficient.
Disadvantages	Nodes are subject to heavy loads.	If supernodes fail, searches and connection to peers becomes unavailable.	Vulnerable to malicious attacks and technical failures; Unscalable.

Table 2.2: Advantages and disadvantages of P2P networks according to its structural architecture

	Unstructured Architecture	Structured Architecture
Advantages	Simple implementation.	High scalability with exact-match queries.
Disadvantages	Search mechanisms are brute-force or flooding based.	Demanding maintenance and complex search mechanisms.

2.1.3 MOSAICA P2P Network

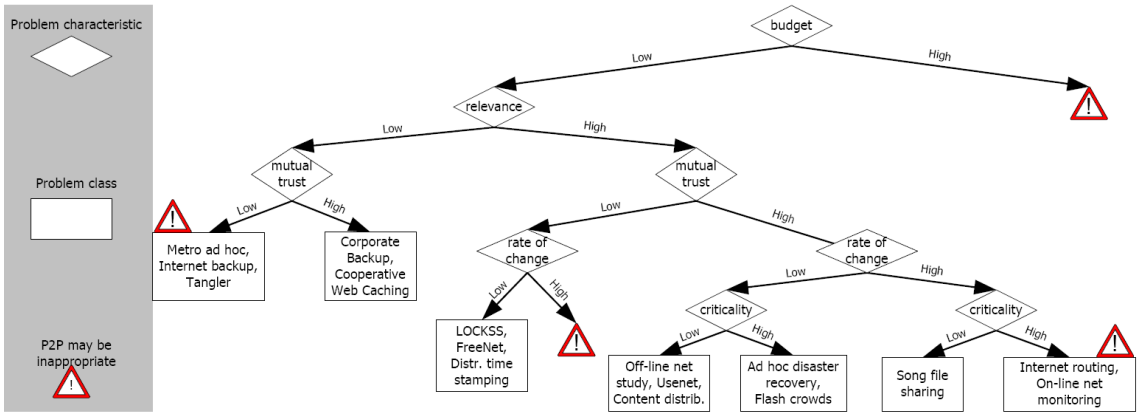


Figure 2.6: P2P decision tree

Using the decision tree of figure 2.6, extracted from [10], and taking into consideration the objectives and requirements of MOSAICA, it becomes clear that P2P is the best suitable choice to use in MOSAICA. Given that MOSAICA aims at building a system to be widely used across different sectors of society, providing easy, low-cost, reliable and efficient access to multimedia resources, the decision tree shows a system that is desired to be cheap - *low budget* -, with contents that are potentially important to many people - *high relevance* - and requires that peers may download from each other without fear of getting malicious data - *high mutual trust*. Also, in MOSAICA it is expected that users may randomly leave and enter at any time, but it is possible to implement measures to control available resources, so the *criticality* of the system becomes low, associated with *low rate of change*.

2.2 BitTorrent

It was decided that MOSAICA would have a structured P2P architecture, and would use DHT to perform searches in the network. To transfer contents using the P2P paradigm, MOSAICA decided to use BitTorrent, a protocol created by Bram Cohen in 2001, specifically designed for files' transfer, delivering better performance in terms of downloading times. It is P2P-based, since

it allows users to connect directly, receiving and sending data from/to each others [11]. However, BitTorrent needs a central server - the tracker - to coordinate peers and informing them of other peers' location, an architecture much similar to Hybrid Decentralized Architecture, as seen in chapter 2.1.2.

The major innovation brought in by BitTorrent is closely related with the number of users. While in the past, using the server-client architecture, the more users were downloading the same data, the less download bandwidth was available (a problem commonly known as server-bottleneck), with BitTorrent and P2P architecture networks, the more users are downloading the same data, the higher download bandwidth is available, because every user, while downloading, is also uploading data pieces of what already is downloaded. This is possible because BitTorrent specifies that files are splitted into identical sized pieces, apart of the last one, that can be smaller. These pieces - or blocks - have typical sizes between 65KB and 1 MB [12]. This process is done by the initial *seeder* - the first user that has the whole data - when he's creating the torrent file, and at the same time, hashes of each block are calculated, to grant block's integrity.

Avoiding server bottleneck problems and allowing high transfer's speed, BitTorrent is a major choice when compared with other transfer's method, like Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP). And splitting data into small blocks allows quick dissemination and replication of data between users. These blocks' download is done according to BitTorrent's algorithms, freeing clients of sequential download, assembling the full data once the download is finished. These advantages lead BitTorrent to become more robust and effective in resource's utilization than others cooperative techniques [13]. So, it was massively adopted by Internet users and three years after its first release was estimated that eighteen to thirty-five percent of all the Internet traffic was due to BitTorrent [12].

2.2.1 The Protocol

BitTorrent protocol specification [14] defines that, in order to share contents, some elements are required:

A torrent file This file contains shared content's metadata and information about the tracker;

A web server The server is where the torrent file is published, and from where it can be downloaded;

A BitTorrent tracker The tracker is the central server that acts as a central coordinator for the peers;

An initial uploader This peer, usually the torrent maker, is the first to upload data to others, in order to get data distributed over the network;

The end user web browser The browser is needed so that a user accesses the web server that holds the torrent file and downloads it to his computer;

The end user downloader The downloader is a BitTorrent client that uses the torrent file previously retrieved to contact the tracker first and then the peers with some content's parts, getting the desired data.

When the initial seeder - the peer that has the full data before others - decides to share it over a P2P network using BitTorrent Protocol, he will start by making the torrent file. The tool used to do it divides the file (or files) into small blocks (*pieces*) - typically with 256KB -, calculating each block's hash with SHA-1 algorithm, saving it, along with each block's size in the torrent file [12]. In the end, the torrent file is *bencoded*, having the fields [14]:

Announce The tracker's Uniform Resource Identifier (URI);

Name A name's suggestion to the file or folder when saving it;

Piece Length The number of bytes in each block into which data is splitted. Usually, it's a power of two;

Pieces Map with each block's hash;

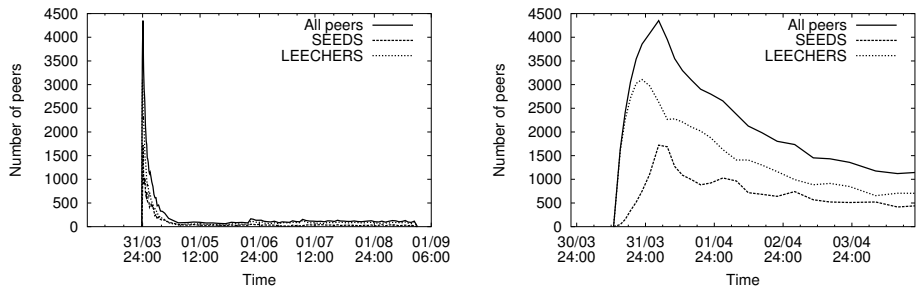
Length or files Information about content's size, in bytes. Keys *length* and *files* cannot exist simultaneously: the first indicates that torrent contains a single file, and the second indicates that torrent contains multiple files or folders.

After the torrent file is created, the initial seeder has to make it available to others. The most common method is to publish it in a web server, so others can download it. When a user wants the data that someone is already sharing with BitTorrent protocol, this user has to download the torrent file from the web server and open it with his BitTorrent client. When this file is opened, the client knows immediately the tracker's URI, the full content's size and how many blocks the content is splitted into.

Before starting the download, the BitTorrent client contacts the tracker, informing it that he will start downloading a specified content, and the tracker answers with a list of peers that have that same content, or pieces of it. Periodically, the tracker is contacted by the client to inform him of how much is downloaded and how much it had uploaded, but for statistical purposes only [13]. The tracker can also be contacted when a node has less than twenty peers, to renew the list of available peers.

As stated before, for P2P networks and from what can be also seen in the analysis made by Izal et al. [15] to a torrent's content, network users have a highly transient behaviour. From figure 2.7 by [15], we can see that, in a torrent's lifetime, the maximum number of active peers happens in

the first days. For the example shown, which corresponds to a torrent of a Linux distribution of 1.77 GB, 51,000 clients were connected only in the first five days, in a total of 180,000 clients during the five months covered by mentioned analysis, which represents more than 28,3% of users in only 3% of the time of analysis. In figure 2.7 b), that zooms the first five days, we can see that near 4,500 peers were simultaneously connected, which started to exponentially decrease after twenty four hours.



(a) Complete trace (b) Zoom on the first five days

Figure 2.7: Number of active peers over time

Once the download is finished, the client informs the tracker of his state's change, from *downloading* to *seeding*, and at the same time informing that, from now on, it has one full copy of the content. User then becomes a seeder, and now it only uploads pieces to others, on demand. It is a common situation that, when users finish the download of a content, their share ratio is inferior to 1, mostly because of asymmetrical Internet connections - bandwidth for download is greater than for upload -, which results in getting more data than what they had offered to others, in a given amount of time. This situation can be also result from the action of users, which can impose an upload limit rate quite low, while using all the download bandwidth for all that they can get from others.

This fact introduces fairness issues in BitTorrent protocol. By default, BitTorrent clients automatically start seeding once a download is finished. However, nothing forbids users to stop uploading that content, in order to save bandwidth or monthly traffic quota. This fact can be observed with greater detail in figure 2.7 b), where *leechers* - peers that get more data than what they offer to others - are in greater number than seeders during all the time of the analysis. In an attempt of controlling this behaviour, BitTorrent protocol has some algorithms that try to get this protocol a little more fair to all users, trying to, at the same time maintain this protocol's advantages.

2.2.2 BitTorrent algorithms

BitTorrent algorithms try to reduce fairness issues to a minimum by offering best download rates to peers who give others good upload rates. This, however, is a problem when a peer starts a download, and has nothing to offer to others. This situation requires a specific algorithm for download the first pieces, known as *Random First Piece* mode, and only performed once, at the download's start. This algorithm's concern is to get a block as fastest as possible, independently of peer's bandwidth or block's rarity. So, a random block is selected to be downloaded and, once the download of this block is concluded, this download strategy is replaced by other.

One other feature of the BitTorrent protocol that tries to reduce fairness issue is the impossibility of users to make sequential downloads: when downloading a content of several files, each of which is splitted into several pieces, users cannot ask the BitTorrent client to download sequentially all the pieces of a file. Instead, after leaving the random first piece mode, BitTorrent protocol starts by selecting which pieces are going to be downloaded first. This is quite important, because a lazy management of pieces' download may result in a peer getting all available pieces, but not having other pieces, not so available, that others would want. So, a technique used by BitTorrent is *Rarest Local First (RLF)* [16]. This consists in getting the rarest pieces available in the swarm first. This way, one peer can later download better distributed pieces, offering to others the rarest pieces. This way, not only rare pieces become better distributed, but also all blocks become evenly available.

Despite the RLF algorithm, when data's download is near its end, there's a tendency for last pieces' download to be quite slow if missing pieces are rare among peers. In this step, a different algorithm is applied - the *Endgame Mode* [13]. Although not having a formal definition of when to enter in Endgame Mode, common BitTorrent clients use one of two possibilities:

1. When all blocks have been requested;
2. When the number of downloading blocks is greater than the number of blocks remaining and less than or equal to twenty.

When in Endgame Mode, the BitTorrent client requests all peers for all the remaining blocks. Once a block comes from one peer, the client sends a *CANCEL* message to all the other peers who were requested.

While downloading, to benefit from the best transfer rates, each peer is responsible for maximize its own download speed. To accomplish this, BitTorrent uses a Tit-For-Tat strategy [12, 13]. This policy may be seen as a "if you scratch my back, I'll scratch yours" scheme, and is inspired by the Prisoners' Dilemma, formulated in 1950 by Albert Tucker. Tit-for-tat is used through choking algorithms, in which peers prefer to upload to others who offer higher download rates. When a user is uploading data at his connection's maximum limit (or if he's uploading at the maximum rate he set the BitTorrent client to), and one other peer sends a request to download, the first one

denies the connection - this is choking, a temporary refusal to upload. Choking is also applied to peers that are just downloading and not uploading, encouraging this way a fair data trade.

As choke is temporary, every ten to twenty seconds, a peer performs unchoke in some connections, typically four, analysing the download rate of data from unchoked peers during twenty seconds, deciding after that if connections must remain unchoken or not. Together with this, in every thirty seconds, a peer decides to unchoke one other peer independently of the current download rate from it - this technique is known as *optimistic unchoke* [13] and it serves two purposes: it may allow the peer to discover other with higher download rates and allow peers to retrieve the first block, when in *Random First Piece* mode [16].

2.2.3 Distributed Hash Tables (DHTs)

Although much common in BitTorrent networks, trackers are not the only way peers have of finding each others. Current BitTorrent clients have implementations of DHTs as an alternative to trackers, improving networks' decentralization - there's no need of trackers and any central coordination -, scalability [17] - DHTs can handle great number of nodes -, fault tolerance - reliability is guaranteed, even with highly transient peers' behaviour - and fairness - all nodes have the same role, being able of entering or leaving the network at any time.

DHTs provide a lookup service, maintained and distributed by nodes at arbitrary locations that communicate with each others - the peers in the network. DHTs store *name/value* key pairs, which allow every peer to efficiently access the *value* from the *name*. To do so, DHTs have two basic operations [18]:

- *store(key, value)*
- *val = retrieve(key)*

In P2P systems, data elements are hashed - typically, with a SHA-1 based algorithm - to an unique numeric key, and so are peers, to an unique ID, in the same key space [19]. Each peer becomes then responsible for a certain number of keys, which means that peers should hold that key and also the data elements represented by it or, in some cases, pointers to data's location. In what concerns to search mechanisms, DHTs support two basic operations, available in its Application Programming Interface (API), as shown in figure 2.8, by [5]:

- *lookup(hash)*, used for finding the node responsible for the *hash* key, and
- *put(hash)* to store a data item (or pointer to it) with *hash* key.

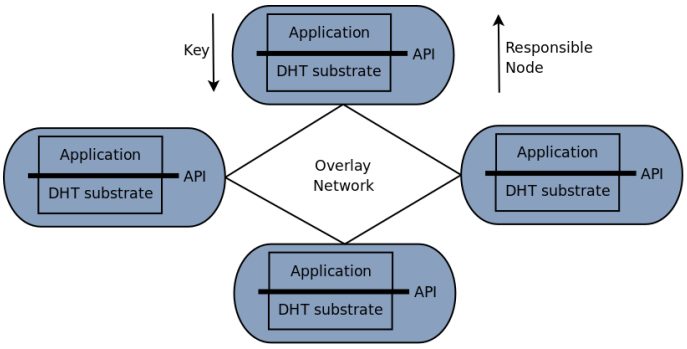


Figure 2.8: DHT API

When applied to file sharing systems, DHTs assures a quick file finding, and also that, if it exists, it will always be found, with good resource's management [5]. Beyond this, it is also quite robust against node failures, except for bootstrap nodes. However, DHTs are not the best solution in what concerns to security, because it's hard to check data integrity with it, and its performance when under attack - node impersonation - exponentially decreases with the number of peers [18].

MOSAICA's lowest layer - the BitTorrent layer -, as a structured network of peers, benefits from a DHT implementation, used by the peers to quickly find and connect to other peers in the network. The use of DHTs is imposed to the BitTorrent client through the torrent files, which contains nodes' IDs, resulting from the hash of node's IP and port. Together with it, the torrent file also contains metadata of the content, registered in an upper layer - the JXTA layer -, which allows users to search for contents using semantics. To do so, contents have to be annotated when are being submitted and the semantic metadata has to be inserted in a distributed database.

This way, the act of searching and downloading a content shared across the MOSAICA network consists of two different mechanisms [20]: first, the user performs a semantic search, through propagated queries, in the upper unstructured layer. This search is based on exact-match lookups, but using semantic expressions instead of the exact file name, which is, in most cases, unknown by the user. Then, after downloading the torrent file provided by the performed search, the BitTorrent client uses DHTs during the download process to retrieve a list of peers that are sharing the content, together with its location, helping the establishment of connections.

2.3 Service-Oriented Architecture (SOA)

Enterprises found, in distributed computing solutions, a way of enlarging their business markets and a response to increasing software complexity [21]. Having the need of lowering costs, reducing cycle times and improving integration across enterprises, SOA is nowadays one of the most important architecture style for enterprises [22], with principles based in the ones from distributed

computing and introducing a new concept of nodes in a network: services providers and services consumers (figure 2.9), where the former may also be a service consumer itself.

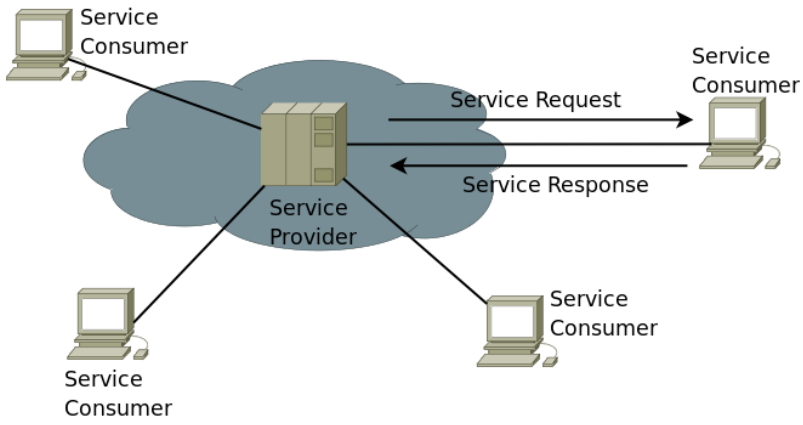


Figure 2.9: Basic SOA

SOA may be seen as a collection of services, being a service a well-defined and self-contained function, capable of one or multiple operations, independent of context or other services' state [21]. IBM states that SOA comprises three entities and three operations [23], as shown in figure 2.10:

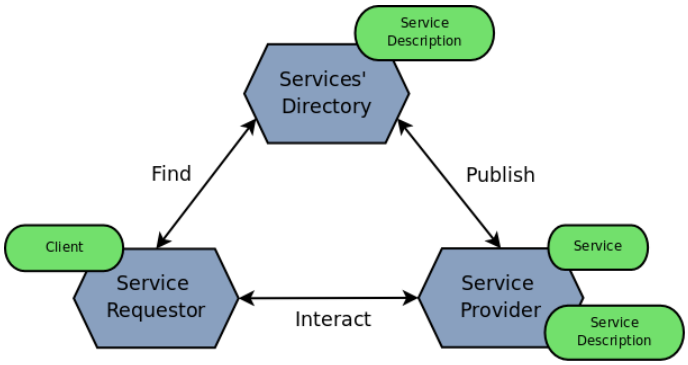


Figure 2.10: SOA entities and operations

A service provider is the node that provides the service or set of services, along with its interface, *publishing* services to the service broker.

A service requester is the node that *finds* services, using the service broker, and invokes other services, *binding* services from the service provider.

A service broker is the node responsible for finding services provided, working like a repository of services.

Development of SOA tried to reach independency of several factors widely presented a few years ago: platforms, vendors, operating systems, locations, programming languages or even functional areas [24]. In fact, Michael Stal [25] defends that, when developing software in the SOA way, five strong factors must be taken into account:

Distribution Services and clients must be able to communicate across networks, independently of its running environment.

Heterogeneity Service developers do not know what kind of clients will use the service and clients' developers hardly will have services' implementation details.

Dynamics Decisions on what to do with services must be chosen at runtime, and must be decided by clients' developers and not services' developers.

Transparency Communication's infrastructure details can't be a concern to service providers or its clients, which can be seen as a consequence of the *Distribution* factor.

Process-orientation Services must be as simple and discrete as possible, so others can develop software clients that uses multiple services with easiness.

A definition of SOA is proposed by Thomas Erl, in [26].

> "SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise."

In fact, with SOA, multiple machines can be providing the same service, which results in splitting network and processor's load over them, making services' providers more reliable and computation times greatly decreased, achieving this way a new way of parallel program model [27].

2.3.1 eXtensible Markup Language (XML)

As previously mentioned, SOA's success is related with its cross-platform capabilities. This was achieved through the use of a base communication's language that is, itself, platform-independent. This language - XML - aims to be legible by machines, independently of operating systems, architectures, programming languages or communication protocols, and easily understood by humans, since it is plain text based. Together with this, XML can describe most types of data, including strong typed method parameters.

Its internal structure is defined by a root element and its child elements - tags -, whose names can be defined according to user's needs or wishes. This introduces great flexibility, but also requires that XML clients (parsers) are in accordance with tags used at XML file's creation process. Given the freedom to choose tags' names, when XML is used in standard technologies (or architectures, like SOA), naming conflicts may occur. To prevent this from happening, standard namespaces can be included in the root element, which requires that the whole document must obey strict rules defined in those namespaces to be considered valid. To help developers to write valid XML documents there are XML generators, pieces of software that translates user's code to XML documents that respect a standard and that will be understood by all clients that use that same namespace. There are also XML parsers that analyse XML documents against specified schemes, which can be used to infer about XML's validity or extract data from provided documents.

Due to its characteristics, XML can be adapted to almost any technology, being usually at the "edge" of several components or applications [28], and is quickly becoming a standard method for data's exchange between applications [29].

2.3.2 Web Services

Knowing that, nowadays, everyone can be connected to the WWW, enterprises encountered in SOA a way of reaching almost everywhere, using this fact to deploy their services through online publication. However, it is very common to encounter restrictive policies that try to regulate Internet's accesses, which caused developers to implement services' communications over HTTP, bypassing those policies, and allowing users to access them through an Internet browser, which is actually present in almost every operating systems available, without the need of any third-party application nor the need to install it. Web Services are the result of these developments and, although most of them use HTTP, they can also use other known protocols, like FTP or Simple Mail Transfer Protocol (SMTP), and apart from binary data attachments, messages exchanged are in XML format [30].

Instead of selling software that can perform certain operations, enterprises deploy services through the Internet. In order to use these services, clients are no longer limited to one computer - for instance, the one from their work place - and can access from almost everywhere, in a very quick way, not needing to install any specific software. Clients are also freed from the need of computational power, because, with Web Services, all the processing is made in the provider's side. The client only needs to know the services' location - the URI - and the inputs to be entered, which are sent to the provider, processed and then the result is displayed in the client's computer.

So, a service can be composed by one or multiple operations, each of which is a function, that must be implemented in that service in order to be used. The indication of which inputs and outputs are supported by a service is defined in Web Service Definition Language (WSDL) file

(in section 2.3.2.2), being also defined error cases, much similar to Java's *exceptions*. This file is XML-based, which implies that it also is platform and programming language independent. In fact, independence of all these factors is the main difference between Web Services and all of its predecessors in distributed computing (like CORBA, Remote Procedure Call (RPC), Microsoft DCOM or Java RMI) [31, 32]. Together with WSDL, Web Services use also SOAP, a standard protocol that uses XML (more details are provided in section 2.3.3). The Web Services overall architecture's concept, as defined by IBM [33], is illustrated in figure 2.11.

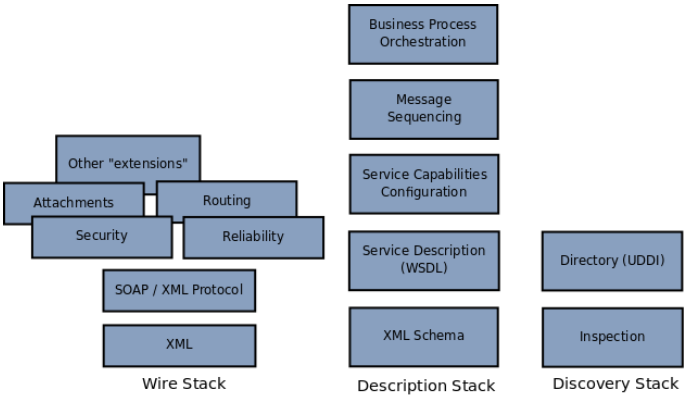


Figure 2.11: Web Services Conceptual Architecture

In what concerns the performance, Juric et al. [31] showed that Web Services are the best alternative to other distributed computing solutions, only slightly overtaken by Remote Method Invocation (RMI). Figure 2.12 [31] illustrates, for instantiation method, simple data types and string specific case. Although being the second best method in terms of performance, Web Services are quite near of RMI methods, also having huge advantage in scenarios with firewalls or Network Address Translations (NATs).

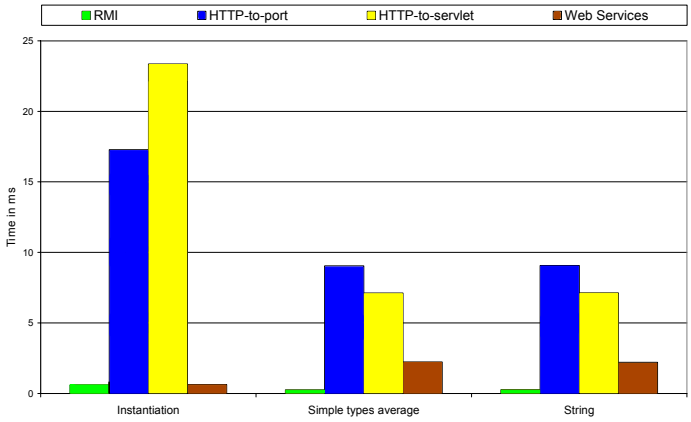


Figure 2.12: Performance of different distributed computing technologies

Nowadays there are no more doubts about Web Services' usefulness. However, Web Services can only be consumed once clients know its location, or URI. In fact, many authors state that Web Services are composed by inputs, outputs and the location. In the actual Internet there are already plenty of Web Services and, for a user who uses lots of them, remembering all the URIs is not a reasonable solution. To avoid this, Web Services may be used together with Universal Description, Discovery and Integration (UDDI), which acts as a directory for published services, as represented in figure 2.10, in page 19.

UDDI stores services' description, and XML is the format in which data is stored. Once stored, data is divided into three categories [34]:

White pages, which contains general informations of the service provider;

Yellow pages, which contains business or services' classification, based on standards taxonomies;

Green pages, which contains technical informations about services, providing useful information to services-based applications.

The easiness with which Web Services can communicate all over the Internet using, most of them, messages encapsulated on HTTP packages, lead companies to deploy a big number of services, available from the exterior, exposing resources over the Internet. This exposure made security analysts and not only, to be against such implementations, arguing that Web Services, along with bypassing firewalls and easiness of communication over different networks, carries several security issues to companies who expose them, being this the major fact by which enterprises still didn't massively adopt this as a standard to their services [35].

In order to fix these issues, in 2004, Oasis [36] released the first version of WS-Security, a protocol that provides security measures to messages exchanged between Web Services, using *security tokens*, like X.509 certificates, username tokens and Kerberos tickets [37, 38]. With these, Web Services saw, without a doubt, its security improved, but its performance lowered substantially and services' administration became very complex [39]. This way, Web Services are considered to be moving from an initial *Describe, publish, interact* [40] state to a new and more robust one, in which business interactions are supported. However, when opting to use Web Services, security and efficiency must be considered in order to choose which one is more important.

2.3.2.1 Web Services's Generations

Web Services' first generation was very similar to regular Internet connections: services were not integrated with each others, and they were not designed to be easily integrated with third-party applications. This was mainly due to the fact that standards were not being used [41].

With the second generation of Web Services, standards like XML or HTTP were adopted, allowing Web Services to be used by several and different entities, achieving great level of interoperability. This generation was also the base for Representational State Transfer (REST), a new concept of architectural style for Web Services, which is exposed with greater detail level in section 3.3.2.

2.3.2.2 Web Service Definition Language (WSDL)

WSDL is an XML-based language used to describe Web Services, providing the indication to service's consumers of the service name, the operations available within the service, the type of parameters the service accepts as input, the type of output parameters that the service generates, and also error handlers, much similar to *exception* handlers used in Java [42]. Metaphorically speaking, if XML is a language, WSDL is its correspondent grammar for the Web Services' dialect, defining how words should be placed so listeners can understand what is said. As it can be seen in figure 2.11, in page 22, WSDL appears in the Web Services description stack associated to the "Description" block.

To describe a service, a WSDL document needs to contain five elements [27]:

types, that defines the structural details of messages. In Web Services, it's a common pratice to use XML Schema, usually schema elements from the `http://www.w3.org/2001/XMLSchema` namespace;

message, that is an abstract message definition for input or output messages for each available operation;

portType (renamed to **interface** in WSDL v2.0) defines a group of operations (zero or more) available for a service. Each operation contains a combination of input and output elements, being possible that the latter can be a *fault* element, used to handle errors. The sequence in which input and output elements appear, for each operation defined in the portType, defines the Message Exchange Pattern (MEP) [43]:

One way The operation only has an input element;

Notification The operation only has an output element;

Request-Response The operation has an input element followed by an output element;

Solicit-Response The operation has an output element followed by an input element.

binding associates *portTypes* with a given protocol (nowadays, the most used one is SOAP - details on SOAP are provided in section 2.3.3). *Binding* also specifies which *portType* it is describing through the *type* element;

service defines the collection of *ports* associated with a particular *binding*.

The first three elements are abstract definitions of the Web Service's interface, whereas the last two have concrete description of how abstract definitions are mapped to messages exchanged between both endpoints, bounding abstracts definitions into concrete network protocols [44]. So, a WSDL document associated to a Web Service gathers messages into operations and these into interfaces, giving to the developer all the informations needed to build a client to that Web Service [28, 43]. The WSDL document should be published at the time the provider registers the service at UDDI registries.

When creating a Web Service, a developer can use one of two methods to do so: use the *top-down* approach or the *bottom-up* approach [32]. These methods differ from each other in terms of what is made first. In the top-down approach, developers start by creating manually the WSDL file, in which the service's interface is defined along with other specifications. Once this file is created, the service is created obeying to the specifications in that document. Opposing to this, the bottom-up approach consists in creating first the business logic, which is then translated to a WSDL document, normally through the use of WSDL generators.

2.3.3 SOAP

Looking once more, to figure 2.11, in page 22, it can be seen, in the Wire stack, one major protocol, quite important when building Web Services - SOAP. SOAP, like WSDL, is an XML based protocol which contributes to the Web Services platform programming language independency, as well as to the easiness of accessing services or resources over the Internet. Specificities of the programming language become irrelevant and can even be unknown to the client. SOAP is considered to be the primary messages' transport mechanism for Web Services architecture [29]. In the Web Services context, SOAP specifies concrete definitions for the abstract definition of messages presented in WSDL documents. SOAP's main concerns are the encapsulation and encoding of XML data, and the definition of rules for sending and receiving that same data. Although it may be used with protocols like SMTP or FTP, most of the times SOAP is used on top of HTTP, mainly because SOAP has explicit definitions for HTTP binding, allowing HTTP tunneling, a technique of hiding SOAP inside HTTP messages bypassing firewall's policies [45]. This is an important aspect, specially when thinking that most of services' providers are enterprises whose networks are heavily controlled by firewalls, due to obvious reasons.

When used on top of HTTP, and because HTTP is stateless, (for every established connection between two computers, a request and a reply are needed) for every SOAP message (the request) there's another SOAP message (the reply), which can be the answer for the request previously made, a simple acknowledge or a fault message, in case of something went wrong somewhere in the process [46].

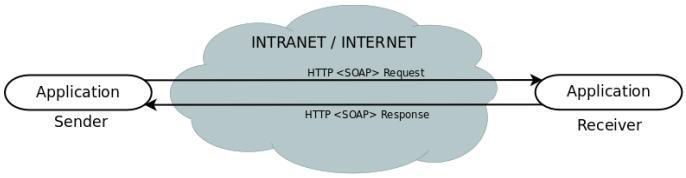


Figure 2.13: SOAP message's exchange

As represented in figure 2.13, a SOAP message has a starting point - the sender - and the final destination - the receiver. However, SOAP was widely adopted because it can do more than this. In fact, SOAP messages, like all messages travelling in Internet or even in an intranet, may pass through several nodes before they reach their final destination. These nodes are sometimes SOAP intermediaries, which are simultaneously receivers and senders, and process the messages header's blocks, like represented in figure 2.14, and different protocols may be used in each path to transport SOAP messages.

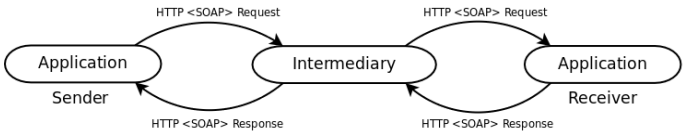


Figure 2.14: SOAP routing capability

Being transmitted over HTTP, SOAP messages are sent and received by the web server. However, its building and handling is not the web server's job, but the SOAP processor's responsibility. This processor is running behind the web server and has a routing engine - the SOAP proxy[1] -, which is responsible for routing incoming SOAP messages to an internal and appropriated application, and when a request is issued, to send it using the web server [29].

Although SOAP can be, at the first sight, understood as a protocol oriented for a server-client architecture, and not so P2P-oriented as would be expected, SOAP can be integrated into P2P environments using frameworks like JXTA [47] (more details on the JXTA framework are provided in section 3.1.1), which creates bidirectional pipes between endpoints, allowing both endpoints to act as peers. This integration with P2P is quite explicit in SOAP's definition given by Box, Ehnebuske, et al. [48]:

> "SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism

[1]The server where is located its name and the port on which the SOAP request is transmitted.

for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules".

Concerning its internal structure, every SOAP message is composed by three elements:

Data encoding rules, which provides a standard definition of SOAP data's serialization, identifying also the message as a SOAP message;

RPC call conventions, which is optional, that defines an uniform way for RPCs, without mapping it to any specific programming language;

The Envelope, the root node of SOAP's message. It consists of two parts:

- **The Header,** an optional field, that describes the data in the body section and may contain XML elements, like security credentials, routing instructions or transactions' ID;
- **The Body,** where data - the methods or the results - are located.

Web Services firewall bypassing ability is inherited from SOAP, and so are the security issues that this skill raised. The reality is that SOAP messages can contain powerful RPCs, but the only thing firewalls can see is harmless HTTP traffic. Together with this, there's also a privacy question related to the use of HTTP: normally, HTTP traffic is non-encrypted traffic, which means that it can be seen by other entities. These issues have been much discussed in the past. Some of them are now fixed, while the solution to others is currently in progress.

Privacy was the easiest problem to fix. Because SOAP was already used on top of HTTP, binding it to Hypertext Transfer Protocol Secure (HTTPS) [47] was quick, effective and easy, because all the privacy and security questions were left to HTTPS, which has already proved to be a secure protocol. The use of this protocol do not harm SOAP's travelling skills, because HTTPS is, like HTTP, normally correctly forwarded, passing through firewalls.

Regarding the security risks that may come from SOAP messages' contents, there is not, yet, an ideal solution. One solution would pass by firewalls inspecting all HTTP packets and find out which are carrying SOAP messages, and then inspect those in greater detail, by analyzing the message's header, from where can be known the intent of the message, delegating to the firewall the decision of allow or deny messages based in its intent, much similar to what already happen with e-mail messages. Although it may be the best way to secure Web Services, it will cause SOAP to loose two of its major advantages - simplicity and lightweighting.

Another solution, which is being already adopted by some companies, is the development of services capable of being correctly executed in underprivileged accounts [29], restricting the privileges of users, so when a service is attacked, intruders have few or no rights to perform malicious actions.

2.3.4 Java Remote Method Invocation

When developing distributed applications, it is frequently necessary to remotely invoke an object's method implemented in a different host. In distributed environments, remote methods invocation can be done with RPC, SOAP-RPC or with Java RMI. The choice of which to choose is mainly related to the programming language [49].

Java RMI is basically an improved RPC mechanism [50], consequence of Java's object-oriented approach. Being part of the Java API, Java RMI allows programmers to invoke remote objects' methods the same way as local objects, freeing them of concerns with communication's details or network protocols. In fact, [50] states that Java RMI's philosophy is "Write once, run anywhere", due not only to Java's easiness of use in most of the actual system's architectures, but also to the fact that a client application can access to a remote object across networks as easy as it can access to a local object.

Although this technology is not language independent, its performance is outstanding when compared with other similar technologies and it benefits of several Java's abilities [50]:

- it can pass full and complex objects as arguments and return values;
- it can move behaviors between client and server;
- it can use design patterns;
- it uses Java's security mechanisms, like security managers;
- it is multi-threaded, enabling parallel computing;
- it uses distributed garbage collection.

Remote method's invocation fits in a client-server communication architecture. Both endpoints communicate across networks using stubs (in the client's side), skeletons and a RMI registry (in the server's side). In the server side, the objects that are desired to be remotely accessed are implemented in the application's code and exposed to clients using interfaces. These interfaces are then binded through the RMI registry, that stores the services and ports in which objects can be contacted. A client application can then establish a connection to the server's registry using remote objects available in the server.

As RMI provides communication between applications that may be in different networks, firewalls may be a problem, and connections may be slowed down or even blocked. To avoid this, the establishment of connections may follow one of the three possible scenarios [50]:

1. The client communicates directly to the server's port using sockets.

2. When not succeed, the client application sends an HTTP POST request using the URL with the server's address, together with the port, usually the port 1099. If this step is successfully accomplished, the server's skeleton answers back to the client's stub.

3. If the previous step also fails, the client tries to contact the server using port 80, and uses a Common Gateway Interface (CGI) script to forward RMI requests to server. If this also fails, the remote method invocation also fails.

2.4 Summary

It is clear, by now, that the use of P2P networks as opposition to the traditional client-server architectures greatly improves not only data's transfer speed, but also the availability of the contents shared in a network as well as the network's liability. The topology that should be adopted for each P2P network is closely related to the search mechanisms that developers would want to implement, having both topologies advantages and disadvantages, which should be considered at the time of the decision of implementing a P2P network.

One of the best P2P protocols is, actually, the BitTorrent protocol. This protocol, besides using algorithms that try to make an evenly distribution of contents, is implemented in clients that are able to use, in structured P2P networks, DHTs as a way to find peers in the network, an alternative to trackers, although the DHT mechanism can be used together with the tracker system.

With MOSAICA platform, Web Services are bridged with a BitTorrent client, enabling users to access to certain functionalities of the BitTorrent client through a web browser. With Web Services, the act of publishing services in the Internet becomes an easy process, freeing users of the need to have computational power and allowing them to access and use published services everywhere.

The use of Web Services in MOSAICA platform result from an attempt to become as universal as possible, and having that in mind, Web Services were built using SOAP and WSDL technologies, which are both XML based. This way, the result - the services - can be accessed and used in any actual platform, being independent of the operating system, since all current operating systems have browsers capable of interpreting HyperText Markup Language (HTML) language and Hypertext Preprocessor (PHP) scripts.

Chapter 3

State of the Art

Following the description of the technological solutions used in this dissertation in the previous chapter, this chapter presents the State of the Art in related areas, so readers may know what are the best solutions that these technologies allow developers to achieve and also justify some of the used technologies adopted in the development of the presented work.

3.1 Peer-to-Peer

3.1.1 JXTA Platform: Framework to P2P networks

Created by Sun Microsystems in 2001, JXTA is a complete framework that gathers a set of generalized and open protocols to create P2P networks. It is based on Java technology, which allows it to be platform-independent. According to [51], JXTA is intended to be used in scenarios where:

- centralization is not required or possible;
- resilience is needed;
- massive scalability is an important factor;
- relationships are transient or ad hoc;
- resources are highly distributed.

JXTA protocols intend to standardize the way peers discover and communicate with each other, organize themselves into peer groups and advertise and discover network resources [52]. The following concepts establishes the main JXTA parts in a P2P network [52, 53]:

Peer Any network device able to perform computational operations. A JXTA peer can be:

- **Minimal-Edge Peer** if it has implementations of only the required JXTA core services, relying on other peers to act as their proxy for other services;
- **Full-Edge Peer** if it has implementations of all the JXTA core and standard services, being able to participate in all of the JXTA protocols;
- **Super-Peer** if it has implemented and provides resources to support the deployment and operation of a JXTA network. When a peer is categorized as a Super-Peer, it can act as a *Relay*, *Rendezvous* or *Proxy*, and it is capable of supporting one or more of these functions.

Peergroup A group of peers that share the same set of operations and resources;

Pipes Virtual channels for unidirectional and asynchronous communication between two endpoints. Pipes can be categorized as [54]:

- **Point-to-point Pipes** connect two endpoints together. In JXTA, it is possible to multiple peers bind into a single input pipe;
- **Propagate Pipes** is a connection between an output pipe and multiple input pipes;
- **Secure Unicast Pipes** Type of point-to-point pipe in a secure and reliable communication channel.

Messages Containers for transmitted data in a pipe;

Endpoints Network interfaces, such as source or destination, to where messages are sent;

Services Functionalities that a peer can remotely invoke to produce some results. JXTA services can be divided into:

- **Peer Services** Services associated exclusively to a peer. When a peer that holds a service leaves the network, that service gets unavailable;
- **Group Services** Functionalities offered by a group of peers to any member of the same group.

Advertisements XML announces of a given entity in the network. Advertisements are used in JXTA networks to announce peers, peer groups, pipes or services. In JXTA, advertisements can be divided into:

- **Peer Advertisement** describes peers' resources;
- **Peer Group Advertisement** describes peer-group's specific resources;
- **Pipe Advertisement** describes a pipe communication channel;
- **Module Class Advertisement** describes a module class;
- **ModuleSpec Advertisement** defines a module specification;

ModuleImpl Advertisement defines an implementation of a module specification;

Rendezvous Advertisement describes peers that act as a rendezvous peer;

Peer Info Advertisement describes the peer info resource.

Modules Functionality that can be loaded and instantiated in a peer in runtime;

Rendezvous Peers Special peers that act as concentrators to other peers, services or resource advertisements;

Security In each endpoint, messages are encrypted when sent, allowing integrity, authentication and confidentiality of them.

In what concerns its internal structure, JXTA can be divided into three layers - the core layer, the services layer and the application layer -, as shown in figure 3.1, extracted from [52]. The *Core* encapsulates the essential primitives to P2P networking and mechanisms for discovery and transport, which includes NAT and firewall traversal. The *Services* layer include searching and indexing mechanisms, protocol translation and authentication mechanisms, which may be used or not in a distributed environment. Finally, the *Application* layer includes the implementation of the integrated applications.

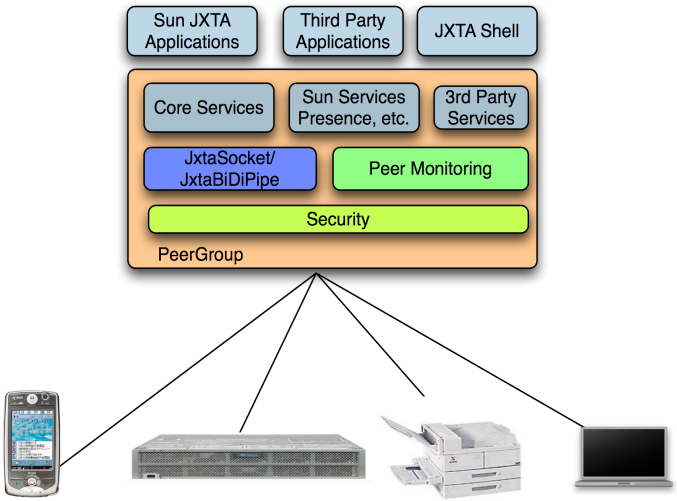


Figure 3.1: JXTA Software Architecture

As JXTA can be defined as a set of protocols to standardize P2P networks, it uses six different protocols for peers to discover and communicate with each other. These protocols define generic syntax for queries and responses and do not specify behaviours of algorithms used by implementations [55]:

Peer Discovery Protocol (PDP) is used by peers to announce their own resources and to discover resources from other peers;

Peer Information Protocol (PIP) is used by peers to get status information about other peers;

Peer Resolver Protocol (PRP) is used by peers to query one or more peers and receive generic responses;

Pipe Binding Protocol (PBP) is used to establish pipes between two or more peers;

Endpoint Routing Protocol (ERP) is used by peers to find routes to other peers;

Rendezvous Protocol (RVP) is used by rendezvous peers to resolve resources, propagate messages and advertise local resources.

From the above protocols, ERP and PRP are lower level protocols, also known as *core specification protocols*, being the other four - PDP, PIP, PBP and RVP known as *standard services protocols*. All of them use XML messages, due to the fact that JXTA wants to be as universal as possible and JXTA developers consider XML as "fast becoming the default standard for data exchange" [56].

3.1.2 Helpers: A new concept of peer

As discussed previously in chapter 2.1, the effectiveness of P2P systems, and particularly BitTorrent, is loosely coupled with the altruism of peers. Knowing this, Wang et al. [57] propose a special type of peers - the *helpers* - that will contribute to the improvement of availability of resources and increase of speeds during transfers, in a way that is totally compatible with the existing networks, protocols and clients.

Oriented to the BitTorrent protocol, helpers are meant to use its spare upload capacity and save the swarm's download bandwidth by not downloading entire files or significant parts of it, but only small parts, specially those that are rarest among all other peers. Once these small parts are downloaded, helpers search for peers who may be interested in them, turning itselves into seeder-like entities.

After joining the swarm, helpers act like normal peers, with two restrictions:

1. A helper only downloads a fixed small number of pieces of a file;
2. Helpers are not allowed to download pieces from each others.

Figure 3.2 extracted from [57] shows clearly that the use of helpers improve download speeds, independently of peers bandwidth.

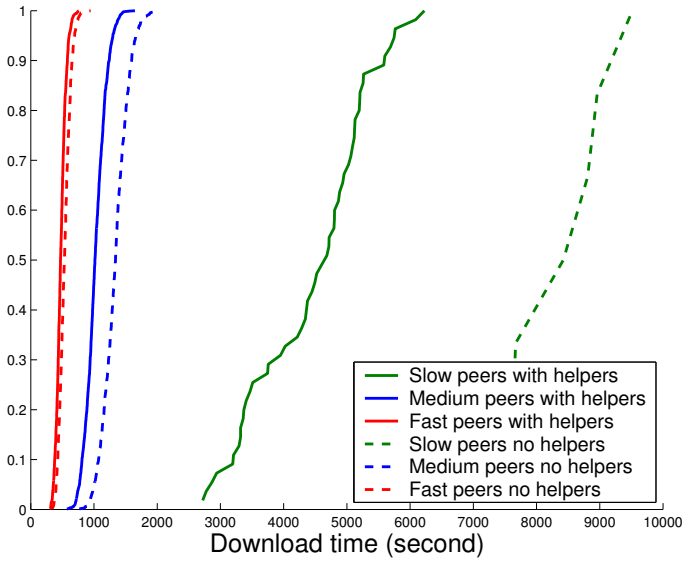


Figure 3.2: Helpers' influence in multiple configurations

3.1.3 P4P: Proactive Network Provider Participation for P2P

It's a known fact that, nowadays, P2P traffic is a significant part of all the Internet traffic. As it can be seen in figure 3.3, extracted from [58], P2P traffic is in an increasing expansion, and its costs for Internet Service Providers (ISPs) are becoming larger. In an attempt to optimize P2P traffic and reduce costs to ISPs, Pando Networks developed, together with some leading ISPs, P2P software distributors and technology researchers, a new approach to P2P networks: the Proactive Network Provider Participation for P2P, or P4P. Its objectives are:

- to provide ISPs with the ability to optimize the use of network resources while enhancing service levels for P2P traffic;
- to provide P2P software distributors with the ability to accelerate content delivery while enhancing efficient usage of ISPs bandwidth;
- to provide researchers who are developing P4P applications with the support to advance and the ability to publish their work and encourage the joint work between ISPs and P2P software distributors.

P4P does not replace P2P networks, but works together with them. In fact, P4P optimize P2P traffic within each ISP, reducing the volume of data being exchanged in each ISP and speeding up transfers up to 45%, according to [59]. This is mainly achieved by geographically organizing peers, choosing nearer (inside the same ISP) and faster peers first. Such advantages lead that several major companies, like Verizon, NBC and Telefonica, already embraced P4P technology.

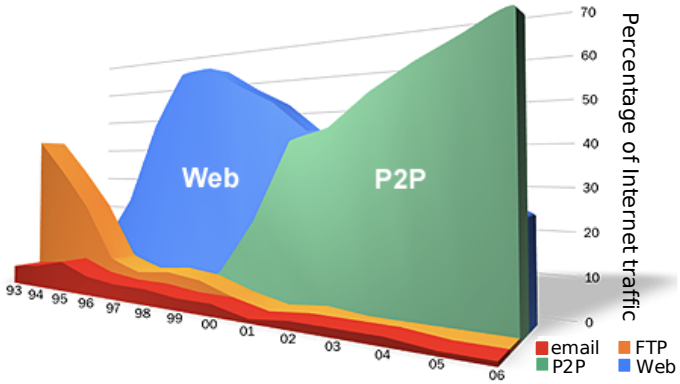


Figure 3.3: Internet traffic along the years

3.2 BitTorrent

3.2.1 Top-BT: An Infrastructure Free BitTorrent Client

An attempt to improve BitTorrent's performance is Top-BT, a BitTornado-based client, written in Python, property of Ohio State University [60]. Its advantages centers in downloading files fast and generate less Internet traffic than other popular BitTorrent clients. Top-BT achieves greater download speeds by using low rates of upload and the minimization of Internet traffic is accomplished through the measurement of routing paths to other peers and use measured data to choose the closest peers to perform unchoke.

As a client application, Top-BT can perform all the above tasks without need of any modification in the existing networks' infrastructure, and can be used in a swarm with other BitTorrent clients. However, since this is a project still in development, there is not much documentation or discussion on it.

3.2.2 Vuze

Currently in its fourth version, Vuze is quite popular between BitTorrent users and remains known by the name of its previous versions - Azureus. Vuze is an open-source, free and rich-featured BitTorrent client written in Java, which allows it to be executed in any platform that runs Java Virtual Machine (JVM). All these advantages lead Vuze to be, in the first week of January 2009, at the second place in the top download list of SourceForge [61], with more than three hundred millions of downloads.

Among the available features, Vuze distinguish itself from other BitTorrent clients from the ability of working with DHTs as easy as it works with trackers (Vuze uses a modified Kademlia implementation), allowing torrent files' creation and supporting plugins. This last aspect can

greatly increase Vuze's features, mainly due to the fact that Vuze's plugins development gather a large community of programmers around the world, concentrated in forums, providing an huge support and knowledge base to those who wish to develop something new in Vuze. This was considered as a major advantage, as it would allow Vuze to be customized to MOSAICA's specific requirements.

3.3 Web Services

3.3.1 Apache Axis2

When creating Web Services using SOAP, one of the most used tools, nowadays, is Apache Axis2, a Java-based SOAP implementation running on Apache Tomcat server. Axis2 combines SOAP's management with the hosting of services. As a consequent of Apache SOAP's improvement and redesign, Axis2 became a great contribution to the third generation of Web Services. Facing the new requirements of Web Services, like faster and more robust SOAP engines, Axis2 evolved to a more stable platform [62], with a new XML processing model, a messaging-based extensible core, an improved deployment model, pluggable data binding support and synchronous and asynchronous Web Services invocation, making it one of the most desired tools for deploying Web Services [63].

Easing the task of developing Web Services, Axis2 handles all the SOAP messages exchanged between the server and the clients (figure 3.4, by [64]), being responsible for sending, receiving and processing them, with or without attachments. It also generates Web Services from Java classes and classes from WSDL documents, being able of creating and using REST-based Web Services and optionally using standards like WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination or WS-Atomic Transaction [64].

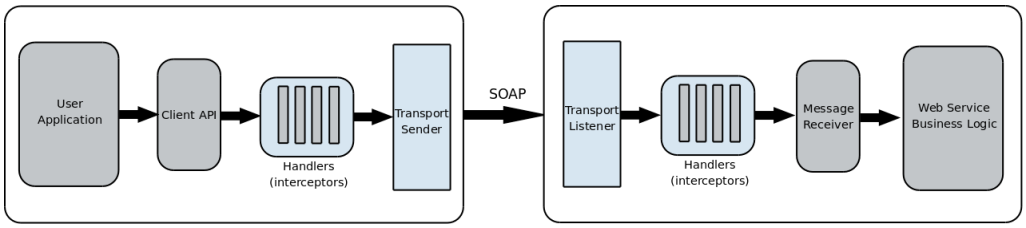


Figure 3.4: Axis2 SOAP messages handling

The chosen version of Axis2 in MOSAICA - version 1.3 - has improvements in what concerns to speed - the use of SAX parsing engine increases the speed of execution of SOAP messages -, flexibility - it allows the addition of extensions to Axis2 architecture -, transport framework - data's transport is completely transparent, allowing SOAP messages to be transported using HTTP, SMTP, FTP among others - and full WSDL v1.1 support [65].

3.3.2 Representational State Transfer (REST)

Introduced by Roy Fielding [66] in year 2000, REST intends to be an architecture style of networked systems, or another way of developing Web Services. Not being a standard - although it uses standards like HTTP and XML -, REST can be seen as a set of principles, based on the premise that every item of interest in the Web is a *resource*, available through *representations*, which place client applications in a certain *state* [67]. Roy Fielding explains REST the following way:

> "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

When building Web Services according to the REST way, there are some principles that should be respected, in order to obey the philosophy of this architecture [67]:

- All the conceptual entities should be identified and exposed as services;
- Every resource should be available through an Uniform Resource Locator (URL), which should be built with nouns instead of verbs, making URLs self-described;
- Resources should be categorized: if clients should only receive a representation of the resource, this must be accessible through HTTP GET. If clients can modify the resource, then it must be accessible through HTTP POST, PUT or DELETE. These four methods are the basis for all operations that can be performed in the REST way [68];
- When invoking a resource, the return should be a representation of it and no modifications should be made;
- Every resource must have hyperlinks to other resources, so data can be gradually revealed;
- The response data must use standard schemes, like Document Type Definition (DTD) or World Wide Web Consortium (W3C) Schema;
- All services should be described, either with WSDL documents or as simple HTML documents.

3.3.3 WSPDS: Web Services Peer-to-Peer Discovery Service

Altough useful for finding Web Services published in the Internet, UDDI narrows Web Services' scalability and fault tolerance. As Web Services are a face of SOA, with UDDI, the overall architecture of the whole system becomes centralized, because clients need to access to one server first in order to be able to use Web Services afterwards, in a decentralized way.

To use Web Services, and discovering them, using all of its advantages, Banaei-Kashani, et al. [69] propose a new way of discovering Web Services, in a fully decentralized and interoperable manner, offering also support to semantic services' search, a feature not available in UDDI registries.

WSPDS uses unstructured P2P networks, and is implemented as a cooperative service. This means that, instead of a single machine answering to peers requests, WSPDS has a Gnutella protocol-based network of collaborative servents, each of which with two separate engines. One is used for communication and collaboration, providing an interface to users, receiving queries from users and from neighbours, replying to them, acting also as a peer in the P2P network. The other engine is used for local queries: when the servent receives a query from the communication engine, it will inspect the local services for a match before propagating the query to its neighbours, which is done if the local engine has no matches and if the query has a TTL greater than zero.

However, as clients may be searching the same service from different perspectives, a semantic search method may greatly improve results compared to a search based in keywords. So, together with WSPDS, a semantic network is also proposed, called Sem-WSPDS. WSPDS uses semantic-annotated WSDL to describe Web Services interfaces, instead of DARPA agent markup language for services (DAML-S). Although both semantic-annotated WSDL and DAML-S add ontologies to Web Services' interfaces (the inputs and outputs of the operations), the first one was chosen mainly because WSDL has already been accepted as an industry standard.

3.3.4 WS^{EXP}: A tool for experimenting with Web Services

WS^{EXP} is a tool developed by J. Nandigam and V. Gudivada [70], that aims to teach Web Services concepts and respective building blocks. It is written in C# and .NET, and provides a Graphical User Interface (GUI) divided into *UDDI Search*, *WSDL Analyzer*, *Service Invocation* and *Web Services Resources*.

By dividing the GUI into those parts, WS^{EXP} allows users to understand and see every step and technology involved in a Web Service. Although these parts may not be visited sequentially, their order reflects how Web Services can be used or developed.

In the first one, *UDDI Search*, a user can enter one or more keywords and search services within IBM UDDI Business Registry, Microsoft UDDI Business Registry, SAP UDDI Business Registry and XMethods UDDI Business Registry. Search results provide information about the service's name, its URI and the location of the WSDL document. In the *WSDL Analyzer* tab, the WSDL document can be loaded using the location previously provided, being displayed in XML notation. This helps users to identify which operations are available for that service, and also the type of inputs and outputs expected by the service, allowing users to use the service by introducing data and seeing what is the answer, together with the SOAP messages, in XML format, exchanged between the service provider and the service consumer.

Finally, in the *Web Services Resources* tab, users have pointers that they may consult in order to get a deeper knowledge in Web Services, SOAP, WSDL, UDDI or SOA, through tutorials, articles or free online books.

3.4 SOAP

Besides the issues presented in chapter 2.3.3, SOAP is already a standard in the Web Services matter, and little glitches that still exists are being subject of new implementations, being all of them SOAP-based.

3.4.1 SOAP Optimization via parameterized Client-Side Caching

Devaram and Andresen demonstrate in [71] a set of techniques to optimize SOAP's performance, in the client's side, maintaining compliance with SOAP standards and avoiding changes in services already deployed in the server's side, oriented mainly to high-performance applications.

The main disadvantage of SOAP pointed by the two authors is its poor performance, inherited from XML, which is basically plain text. This has two consequences in Web Services' performance:

As XML is, basically, plain text, packets become larger, and compressing and decompressing plain text, although efficient in what comes to the final size of the message, takes CPU time that do not compensate this operation;

Every time a message is sent, binary data has to be converted into plain text, which consumes additional CPU time and up to ten times more memory than if binary data was used [72].

Knowing that clients have a finite number of different requests for a service, for every different request made for the first time to the server, the generated SOAP payload is cached into a file and indexed by a key, that contains information about the type of request. Then, every time the same request is made, it first checks the cache, and if that operation is cached, instead of generating

the payload, it reads the cache and sends the request to the server, increasing the execution speed of the whole process. This caching mechanism is also capable of generate similar requests with some different parameters, that may result from the need a user sometimes have of submitting several times the same request, but with different values for the same parameters, avoiding the serialization of the whole SOAP envelope. The use of time validity in the cached items prevent the cache of growing so much that input/output operations' duration become long enough to degrade the global performance. Also, when a fault element is generated by the server, cached items are flushed to prevent errors from happening again.

The use of such techniques has a significant impact on the system's total execution time, as can be seen in figure 3.5 (extracted from [71]), corresponding to performances 800% better than those obtained with traditional SOAP's methods.

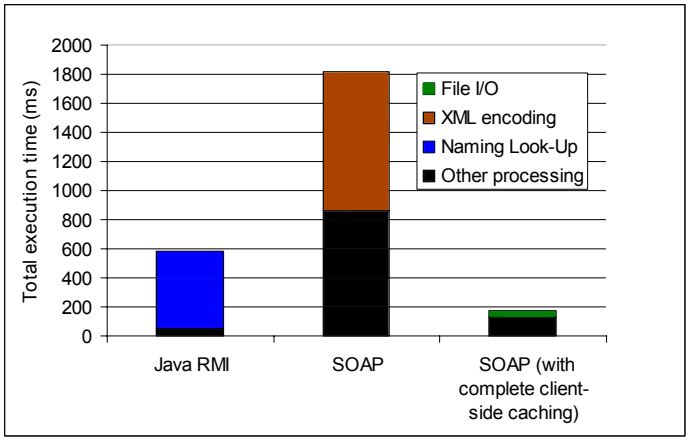


Figure 3.5: Comparison of SOAP with client-side caching with JavaRMI and traditional SOAP

3.4.2 Wireless SOAP: Optimizations for Mobile Wireless Web Services

Together with the issues approached by Devaram and Andresen in section 3.4.1, Naresh Apte, et al. [73] present another set of optimizations for SOAP, but with an additional concern: the evolution of technology brought Web Services into mobile devices, which generally have limited bandwidth compared to devices wired to Internet or intranet and also have less processing power. This concern gained more importance after studies showed that Web Services uses approximately three to ten times more bandwidth than JavaRMI. Wireless SOAP implementation is divided into two main parts: one that concentrates in bandwidth saving and the other in encoding or compressing SOAP messages.

Wireless SOAP assumes that both client and provider are aware of a service's WSDL document, and that this knowledge allows devices to create coding tables from which SOAP messages can

be derived. This is done using a protocol called *WSDL Aware Encoding*, that synchronises mobile devices and gateways.

To reduce messages' size, Wireless SOAP uses *Name Space Equivalency*, that encode SOAP element tags. Although this is somehow against XML namespace's nature, a tag can be repeated several times in a single message, and coding it can shorten the message's size from three to twelve times when compared to those of traditional SOAP and it still preserves the document's structure.

As a consequence of transmitting shorter messages and encoding them using little processing power results in additional savings in energy, which is also limited in mobile devices, as well as reduction in message loss.

3.5 Summary

This chapter presented the state-of-the-art for the technologies used within MOSAICA. As a P2P platform, MOSAICA's network was created using the JXTA framework, which is the basis for the entire system. To this network are going to be connected BitTorrent clients, specifically Vuze, which was chosen among others due to all the advantages mentioned, although it can be considered by some as an "heavy" implementation, due to its Java nature.

To expose Web Services to users, the chosen tool was Apache Axis2, which can easily deploy Web Services, using the standard technologies, like SOAP and WSDL, which obey to the conditions cited in section 2.4.

Chapter 4

The Project

The current chapter describes in detail the developed software that is the subject of this thesis and also the procedures followed during the performed work. The developed components must enable users to download shared contents from a P2P network to their computer using HTTP, without requiring users to install any other software. Also, another objective is to improve the availability of the contents shared in MOSAICA network, by controlling if a content is properly or poorly distributed and, if the second case occurs, contribute by seeding the content until it becomes properly distributed. The solutions encountered to accomplish both objectives are a result of the research done for information on the used technologies and to the state-of-the-art, already mentioned in chapters 2 and 3.

In order to make references clear, from now on every developed component will be mentioned by its name. Concerning the developed Web Services, the Web Service that enables users to download contents shared in MOSAICA P2P network using HTTP is denominated by *GetContent* Web Service (detailed in section 4.3.1), while the Web Service that allows users to consult the list of contents being managed by Vuze is named *List Azureus' Activities* Web Service (detailed in section 4.3.3). The former Web Service needs an extra tool in order to execute its task correctly, designated as ApacheConfigChecker (detailed in section 4.3.2). Concerning Vuze's plugins, this thesis focus in two of them, the *RSS Import* and the *SeedLimiter* (detailed in sections 4.4.2 and 4.4.4), being the former able to be configured, when using one of the deployed MOSAICA packages, through an applet designated as *Shared Folder's Maximum Size Controller Applet* (detailed in section 4.4.3).

4.1 Introduction to the developed components

The starting point for the work done for this thesis is the *MOSAICA Peer Deploy Development* package. This package, as represented in figure 4.1, includes:

Java Development Kit (JDK), necessary for Java programs, so they can be executed in JVM;

MOSAICA Azureus, the BitTorrent client used within MOSAICA peers;

JXTA, the framework used by peers to connect, use and search contents within MOSAICA P2P network;

Axis2 for deploying developed Web Services;

Apache Ant for installation and running purposes;

Wamp, consisting of Apache Web Server, MySQL database and PHP interpreter, for exposing services in the web and providing MOSAICA with a database where users, contents and metadata can be registered.

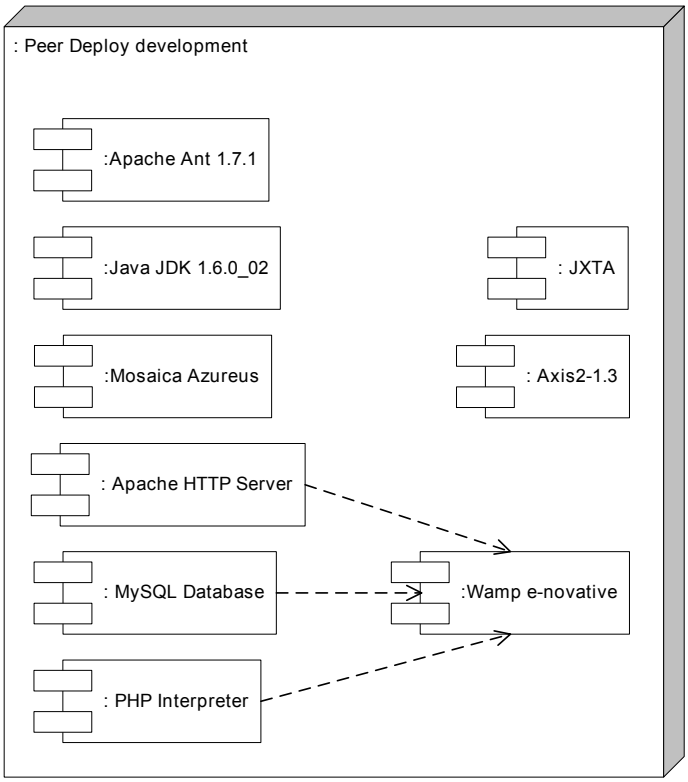


Figure 4.1: Initial P2P-CMS Deployment Diagram for MOSAICA Peer Deploy Development package

MOSAICA has also another package, the *MOSAICA Final User* package, similar to the first one, but without Web Services. This package is meant for users who wish to improve contents' availability. In order to accomplish this, this package installs Vuze and, once installed and running, it periodically reads feeds from a RSS server and downloads contents, obeying to two conditions:

1. The content is considered to be poorly distributed;
2. The shared folder's size, where contents are downloaded to, must not overcome the size defined by the user.

The *MOSAICA Peer Deploy Development* package allows users to use the MOSAICA platform, according with the use cases of figure 4.2. As it can be seen in this use case diagram, all operations become available to users after a successful login, and some of those require an identifier of the content - the *Select contentID* use case - to be performed. One of the goals of this thesis is to empower common users, i.e., users who are not directly working with the MOSAICA tools, to also gain access to MOSAICA's media resources using a normal Web browser. Additionally and although not initially identified as a requirement, it was also decided to include a functionality to allow users to check in real time the list of contents being downloaded and uploaded by Vuze. This extra functionality was developed and implemented in the form of Web Services, enabling users to check, for instance, if the contents they required for download are completely downloaded or not before invoking the *GetContent* Web Service to download contents using HTTP.

The other goal of this thesis is the improvement of Vuze concerning usability and fairness of use. To accomplish this, and taking profit of Vuze's plugin capability, it was decided to use two plugins for this BitTorrent client:

1. SeedLimiter - This plugin intends to periodically check the number of seeds that each content in the Vuze has;
2. RSS Import - Developed by Markus Baeker [74], but modified so it can accomplish better the desired objective of getting contents evenly distributed.

Both plugins are intended to be used by both MOSAICA's packages, but the latter can be configured through an applet when using the *MOSAICA Final User* package. The interaction of both MOSAICA's packages with developed components is represented in the block diagrams of figures 4.3 and 4.4.

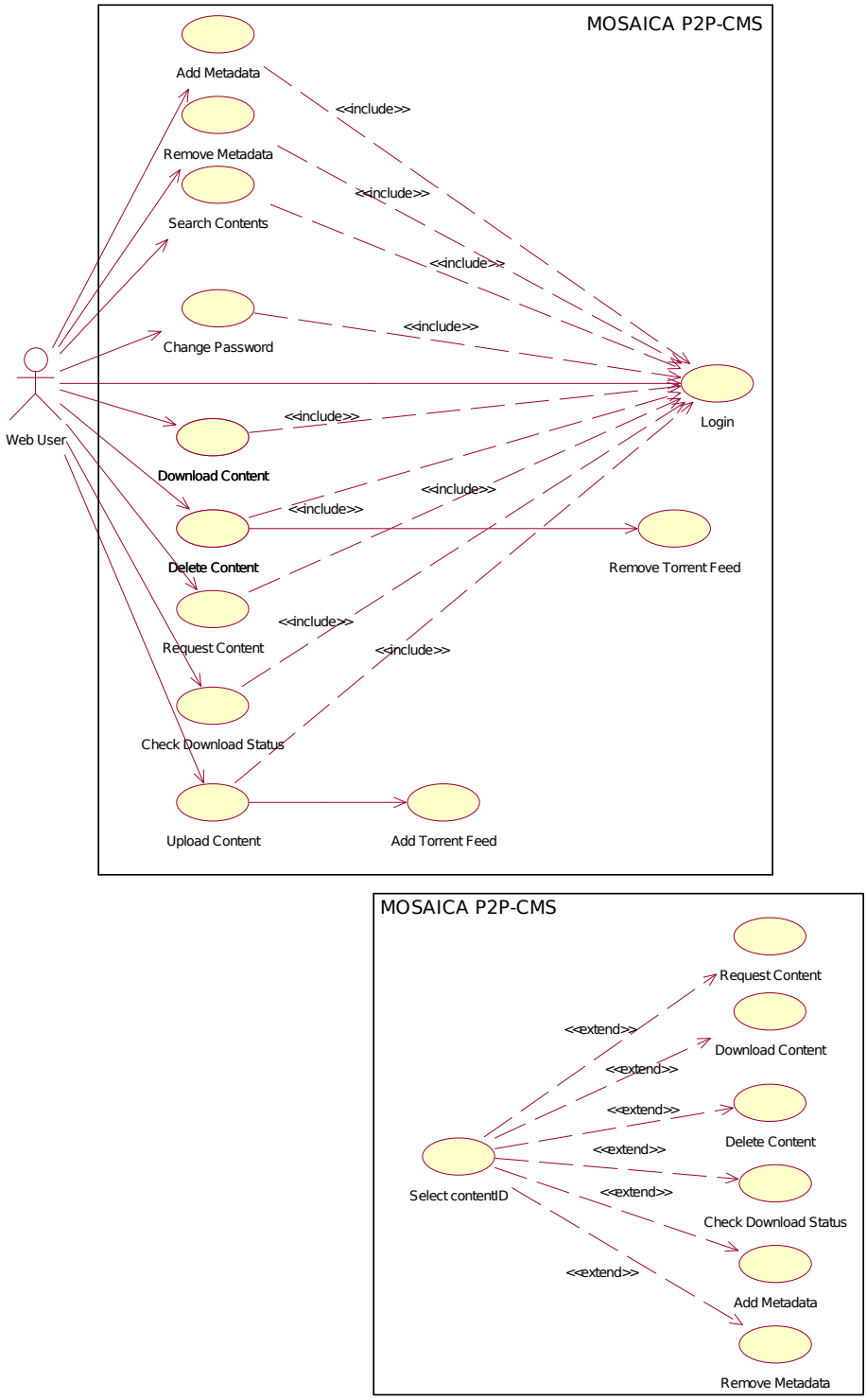


Figure 4.2: Initial P2P-CMS Use Cases Diagram

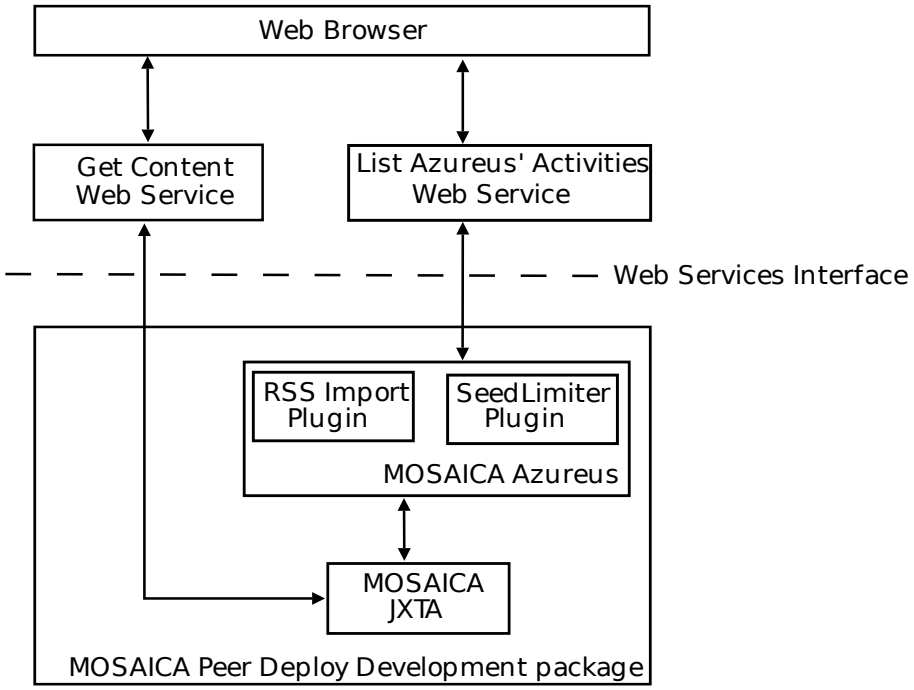


Figure 4.3: Integration with the MOSAICA Peer Deploy Development package

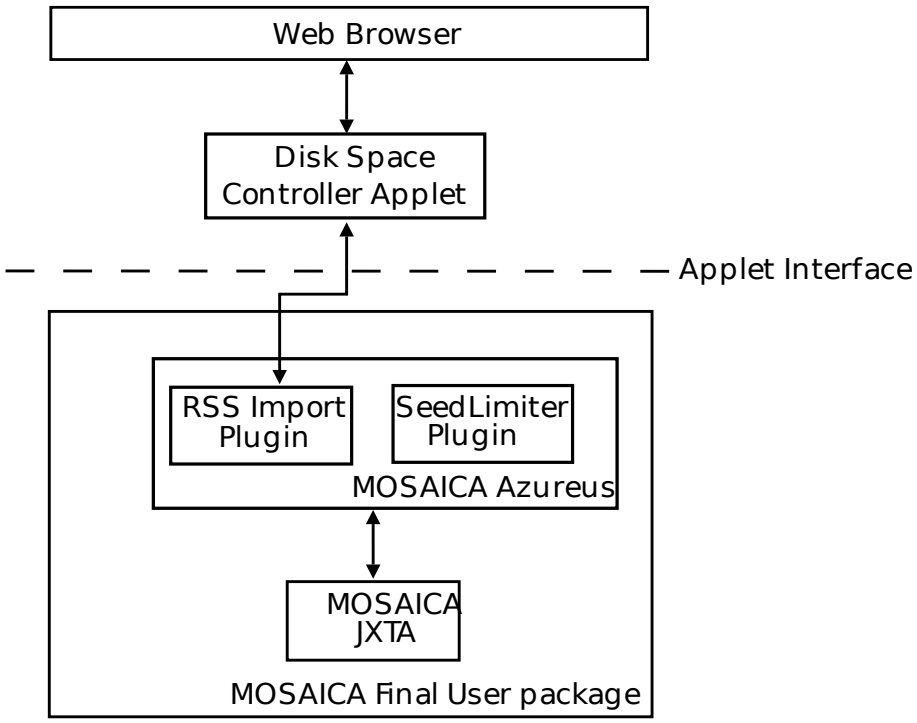


Figure 4.4: Integration with the MOSAICA Final User package

4.2 Development Environment

The development of the applications described in this chapter resulted from a set of processes, as described below:

- Study of the BitTorrent protocol and research about used technologies - P2P, SOA, Axis2, JXTA, JavaRMI, SOAP, WSDL, Web Services technologies and MOSAICA deployed packages -, together with familiarization with the Vuze's source code;
- Assessment of requirements for the components to be developed;
- Formal definition of the components (Web Services and Vuze's plugins) to be developed using UML diagrams;
- Development of components, using Eclipse Ganymede Integrated Development Environment (IDE), together with Eclipse's *Axis2 Service Archive* plugin;
- Specification of testing scenarios and contents to be used during the tests;
- Realization of defined tests, according to the scenarios previously defined.

In the following sections, each component's implementation is detailed, accompanied by the explanation of its purpose and its UML diagrams.

4.3 Web Services

During the work performed, two Web Services were developed, one for making contents shared over P2P networks available through HTTP and the other one for listing in real time contents being managed by Vuze. These Web Services introduce two additional use cases to the use case diagram previously presented in figure 4.2: the *Get Content* and the *List Azureus' Activities* use cases, highlighted in the new use case diagram in figure 4.5.

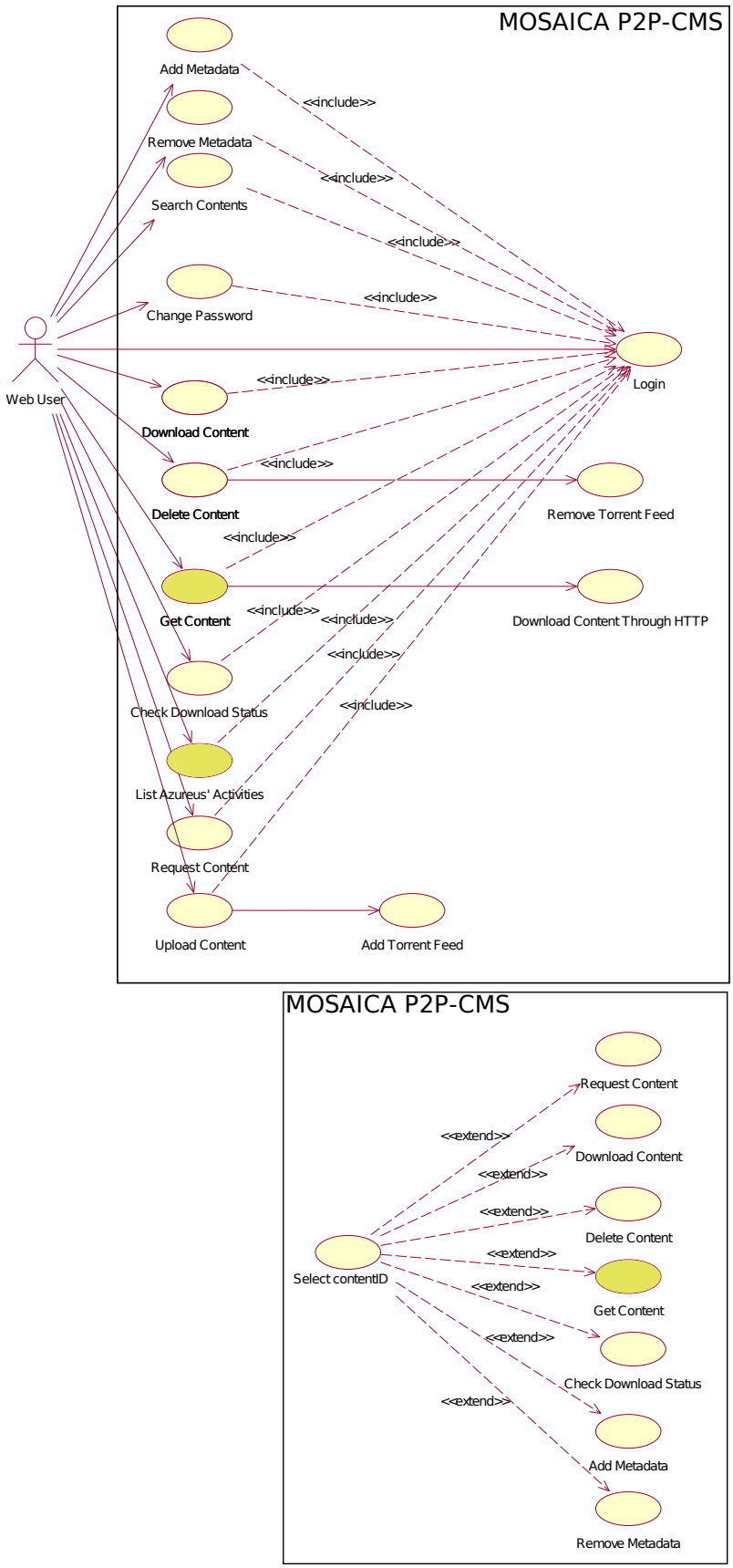


Figure 4.5: P2P-CMS Use Cases Diagram

4.3.1 *Get Content* Web Service

The main idea of this Web Service is, once the computer where Web Services are running - the peer - downloads some content, using Vuze, it becomes available to authenticated users through HTTP. The process of downloading a content, corresponding to the *Download Content* use case of figure 4.5, may be initiated using two possible ways [20]:

1. After performing a search in the MOSAICA network, and if the search query matches some available content, a *contentID* is returned to the user, with which he can order the download of that specific content;

2. Vuze's RSSImport plugin retrieved a feed with the torrent file for a specific content that is considered poorly distributed in the network by the SeedLimiter plugin, and starts downloading it automatically after registering the content in the JXTA layer of the peer in which the Web Service is running, assigning it with a *contentID*.

This Web Service takes advantage of Apache web server, which is required to publish Web Services, being also needed so contents could be transfered over HTTP. This Web Service corresponds to the use case *Get Content* in the use cases diagram (figure 4.5) and it is described in table 4.1.

Table 4.1: Get Content use case description

Use Case	Get Content
Actor	Web user
Main Scenario	Include "Login User" use case. Providing the content's *contentID* - that can be obtained performing a search in the MOSAICA network, the user can get an URL, displayed as an hyperlink, allowing him to download the content to his computer through HTTP. This triggers the *Download Content through HTTP* use case.
Alternative Scenario 1 (*contentID* not valid)	If the *contentID* is not valid, a message is returned, informing the user that there is no content matching the provided contentID.
Alternative Scenario 2 (Content consists in several files)	If the content has more than one file, a message is returned informing the user that content is not a single file.

Table 4.2: Download Content through HTTP use case description

Use Case	Download Content through HTTP
Actor	Web user
Main Scenario	When displaying the link, browsers allow users to save contents to a local folder. The transfer is made on top of HTTP.

To meet the desired objective, this Web Service requires three conditions:

1. The user has to be authenticated,

2. The content's download has to be completed, and

3. The running Apache web server has to know the location of the content, so it can be accessible through HTTP.

In addition to these conditions, this Web Service also performs additional verifications, checking if the *contentID* parameter matches any content in the peer and if the contentID corresponds to a single file. If all the conditions are verified, then the Web Service builds the URL to the content, using the protocol - HTTP -, the IP address of the peer, the port in which Web Services are published and the path to the content, along with its name, following the syntax "<protocol>://<IP>:<PORT>/<path>/<content name>", according with RFC3986 [75].

When building the URL, the Web Service asks the peer for its IP address, appending to it a predefined port - in this case, the port 8000. The path where contents are located is always the same - contents are always downloaded to a folder named "mosaicashared", in MOSAICA Azureus folder -, and the content name, together with its extension is provided by MOSAICA Azureus, when inquired about a *contentID*. However, when performing this operation, a problem arises due to the path where the content is stored. In order to get contents available through HTTP, Apache needs to know in which folder contents are located and, more important, that folder has to be accessible from the exterior. To solve this question, an extra tool was needed, so this Web Service can work properly: the Apache Web Server Configuration Checker (in section 4.3.2).

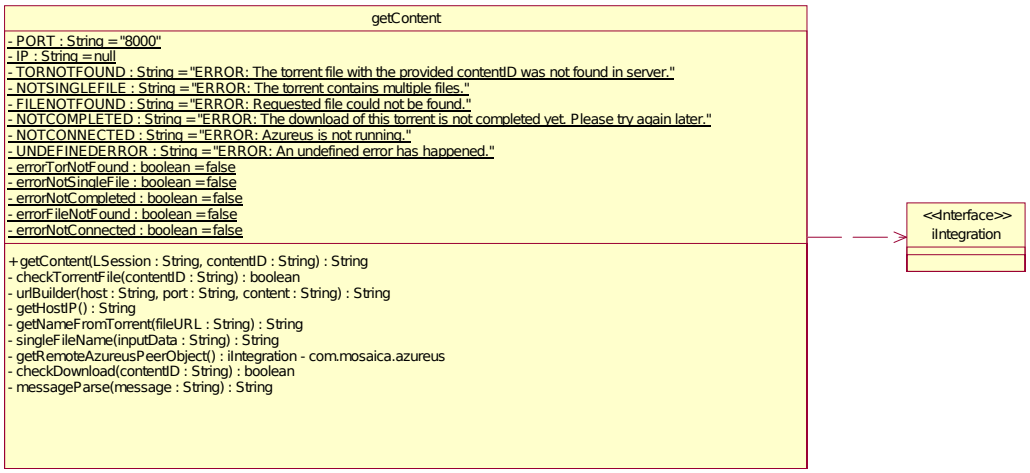


Figure 4.6: Class Diagram for *Get Content* Web Service

The details and the functioning of this Web Service is illustrated in the following UML diagrams: this Web Service's details are shown in the class diagram (figure 4.6) and its functioning is illustrated through both sequence and collaboration diagrams (figures 4.7 and 4.8).

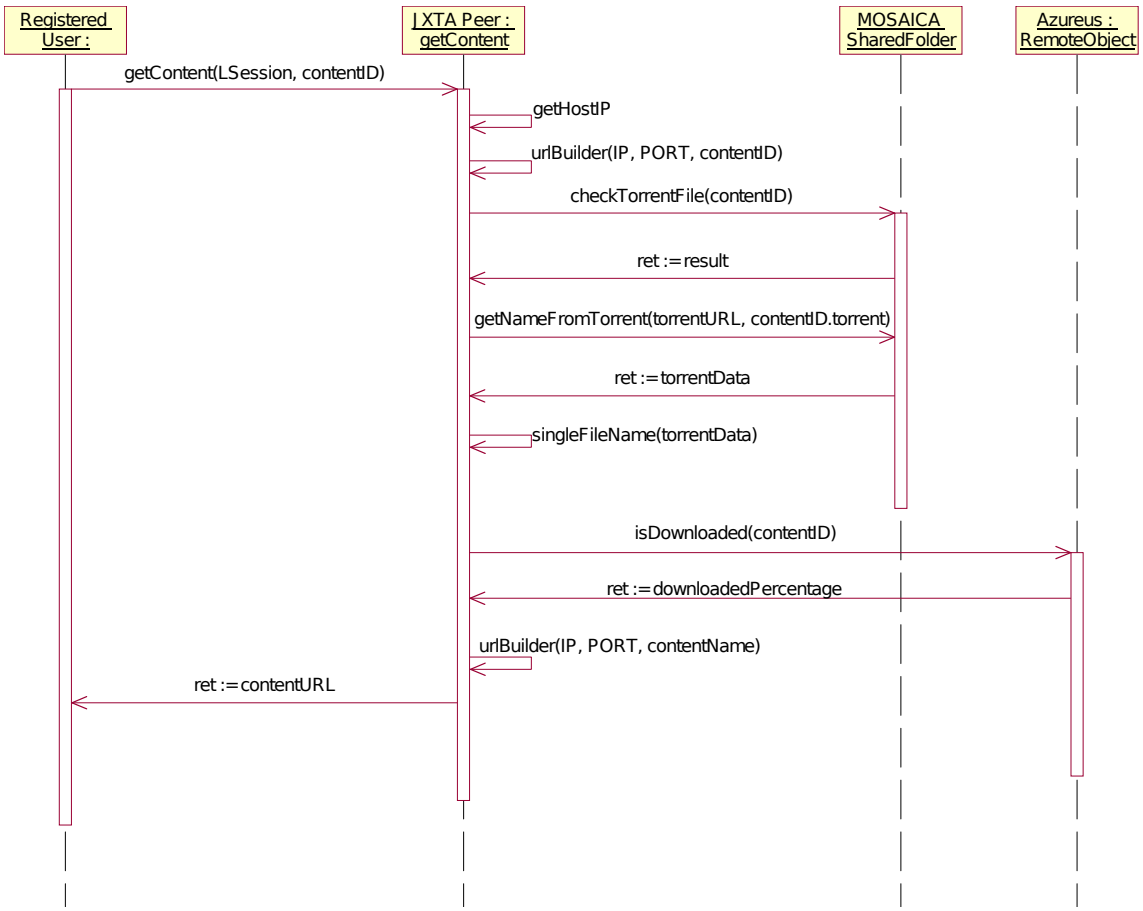


Figure 4.7: Sequence Diagram for *Get Content* Web Service

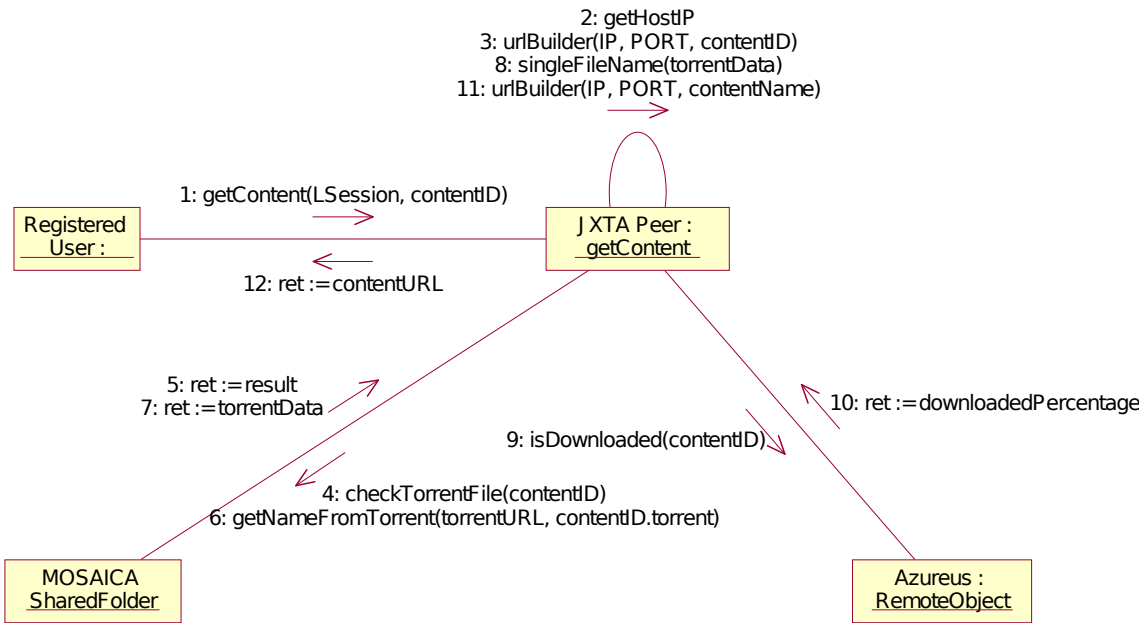


Figure 4.8: UML Collaboration Diagram for *Get Content* Web Service

4.3.2 Apache Configuration Checker

The need for this component came from the combination of two aspects:

1. the fact that Apache Web server only makes accessible to the exterior files located within a specific folder, usually named “www” - the DocumentRoot;

2. the fact that in MOSAICA files downloaded by Vuze are always stored in a folder named “mosaicashared”.

This raised a problem when invokig the *Get Content* Web Service given that, even though the URL to the content could be correctly generated by this service, contents would not be accessible to users. To solve this problem, the solution passed by the use of an *Alias* in the Apache configuration, which allows the use of virtual folders outside the DocumentRoot. This *Alias* is configured in the “httpd.conf” file, making reference to the name of the virtual folder - which was chosen as “mosaicashared”.

Together with the need of using an *alias* it was decided to allow some flexibility concerning the location of files. The entire *MOSAICA Peer Deploy Development* package, where the “mosaicashared” folder is located, is inside one folder and, although not common, users may want to move the folder to any other location like, for example, the Program Files folder. When moving the entire folder, the user is moving also the ”mosaicashared“ folder and this change has to be updated in the *alias*. To make these situations completely transparent to the user, a change was performed

in the launcher script so, every time the peer starts, the Apache Web Server is stopped, if running, and the Apache Web Server Configuration Checker is invoked. It will examine the configuration file and the shared folder's location, updating the *alias* if necessary, and starting Apache once operations are completed.

The construction's details of this tool are presented in its class diagram, in figure 4.9, and its operations are described in the sequence diagram, in figure 4.11 and in the collaboration diagram, in figure 4.10.

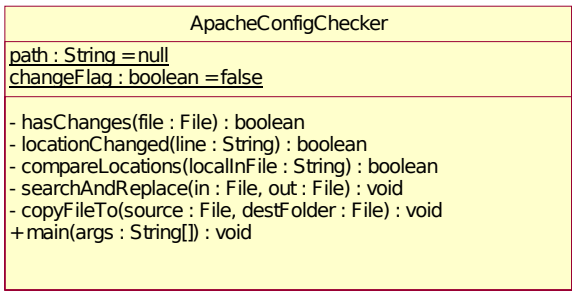


Figure 4.9: Class Diagram for Apache Configuration Checker

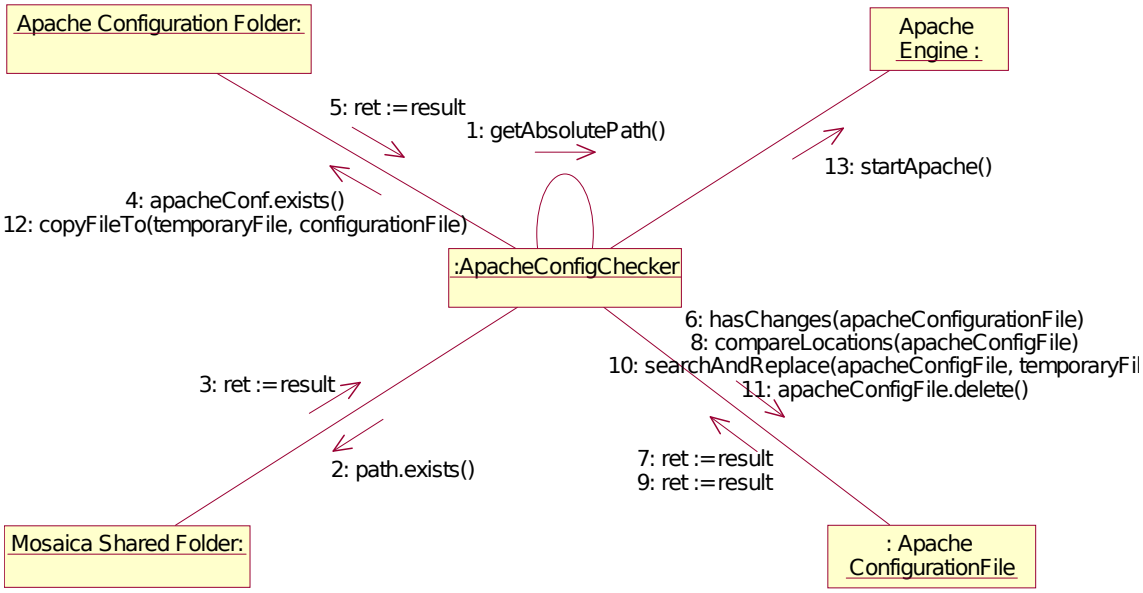


Figure 4.10: Collaboration Diagram for Apache Configuration Checker

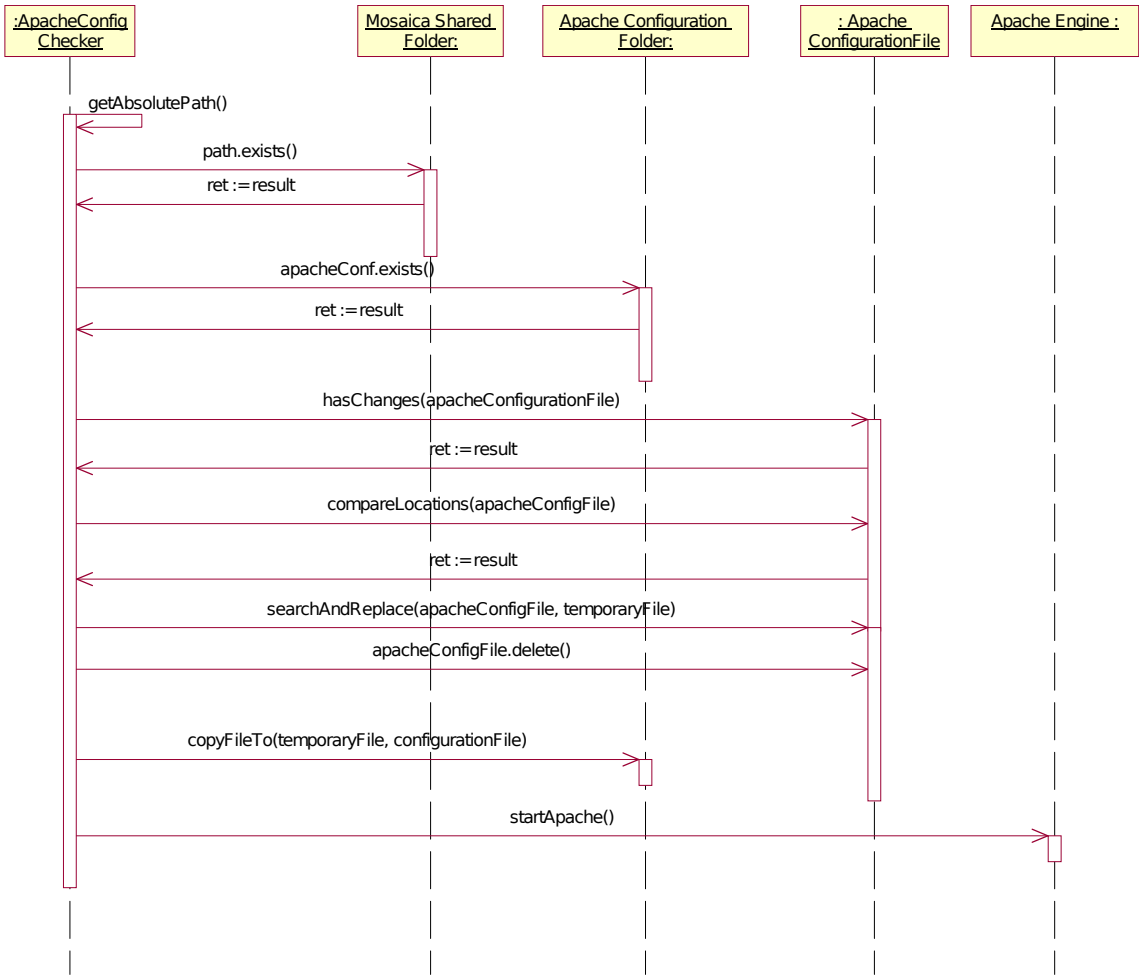


Figure 4.11: Sequence Diagram for Apache Configuration Checker

4.3.3 *List Azureus' Activities* Web Service

List Azureus' Activities is a Web Service that allows users through a browser to consult the contents that are currently being managed by Vuze. It uses JavaRMI technology to contact the BitTorrent client. This operation is illustrated in MOSAICA peer's use case diagram, in figure 4.5, corresponding to *List Azureus' Activities* use case, which is described in table 4.3.

Table 4.3: List Azureus' Activities use case description

Use Case	List Azureus' Activities
Actor	Web User
Main Scenario	When requested, the web service retrieves a list of all the contents being downloaded or uploaded by Vuze. Together with the name of each content and its status, the answer also includes information about the amount of downloaded data, as a percentage, the number of seeds, the actual download or upload speed and the Estimated Time of Arrival (ETA) for downloads and the share ratio for uploads.
Alternative Scenario (Vuze cannot be contacted)	If the Web Service does not get any answer of Vuze, a message is returned to the user, notifying him of the occured error.

The internal structure of this Web Service is represented through its class diagram, in figure 4.12. There, the package responsible for its connection to Vuze's JavaRMI server is represented, being this part of software subject of a more detailed analysis in section 4.4.1. The operations done by this Web Service are illustrated in figures 4.13 and 4.14, corresponding to its sequence and collaboration diagrams.

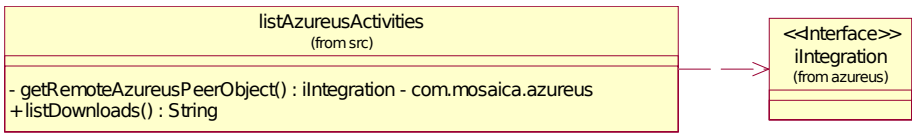


Figure 4.12: Class Diagram for *List Azureus Activities* Web Service

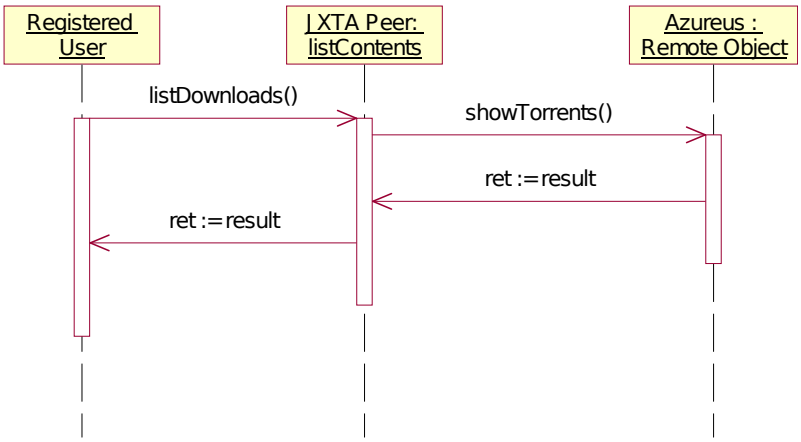


Figure 4.13: Sequence Diagram for *List Azureus' Activities* Web Service

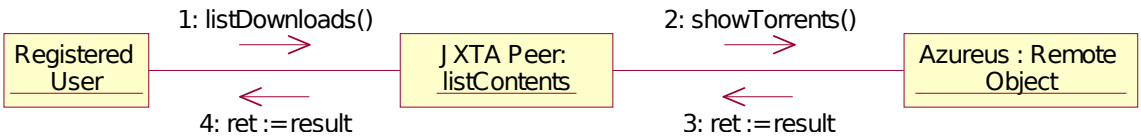


Figure 4.14: Collaboration Diagram for *List Azureus Activities* Web Service

4.3.4 Web Services PHP clients

Web Services are available to users through Web pages, built in PHP and HTML language, and using DTD HTML 4.01 Transitional, for compliance with the highest number of browsers as possible. PHP has specific APIs to SOAP, which make development of SOAP's clients quite straightforward. PHP was also used for authentication, using its *session handlers* together with *sessionID* returned by the *Login* method, so users can navigate through pages without having to re-authenticate. Cascading Style Sheets (CSS) were added to all Web pages, increasing pages' usability and making the use of MOSAICA web platform more intuitive and easy.

In addition, to offer an interface between users and the developed Web Services, PHP scripts are also responsible for displaying results. In the *GetContent* Web Service, the PHP script displays the URL returned by the server and it also builds an hyperlink, allowing users to save the content just by clicking on it. In the *List Azureus' Activities* Web Service, the script has an integrated XML parser that deserializes the messages sent by the server, presenting them to the user as a list, making the readability of the information much more easy.

4.4 Vuze: The BitTorrent Client

Although Vuze has been greatly improved since its first version, becoming one of the world's favorite BitTorrent client, integrating it in MOSAICA platform required some changes: the major ones were made at plugin's level, but interaction with Vuze was also needed, requiring the development of a JavaRMI server, working together with Vuze.

Improvements and developments performed in Vuze's plugins were made according to highlighted use cases from the use case diagram in figure 4.15. For these, details of each use case are described in tables 4.4 to 4.10, in sections 4.4.2 and 4.4.4.

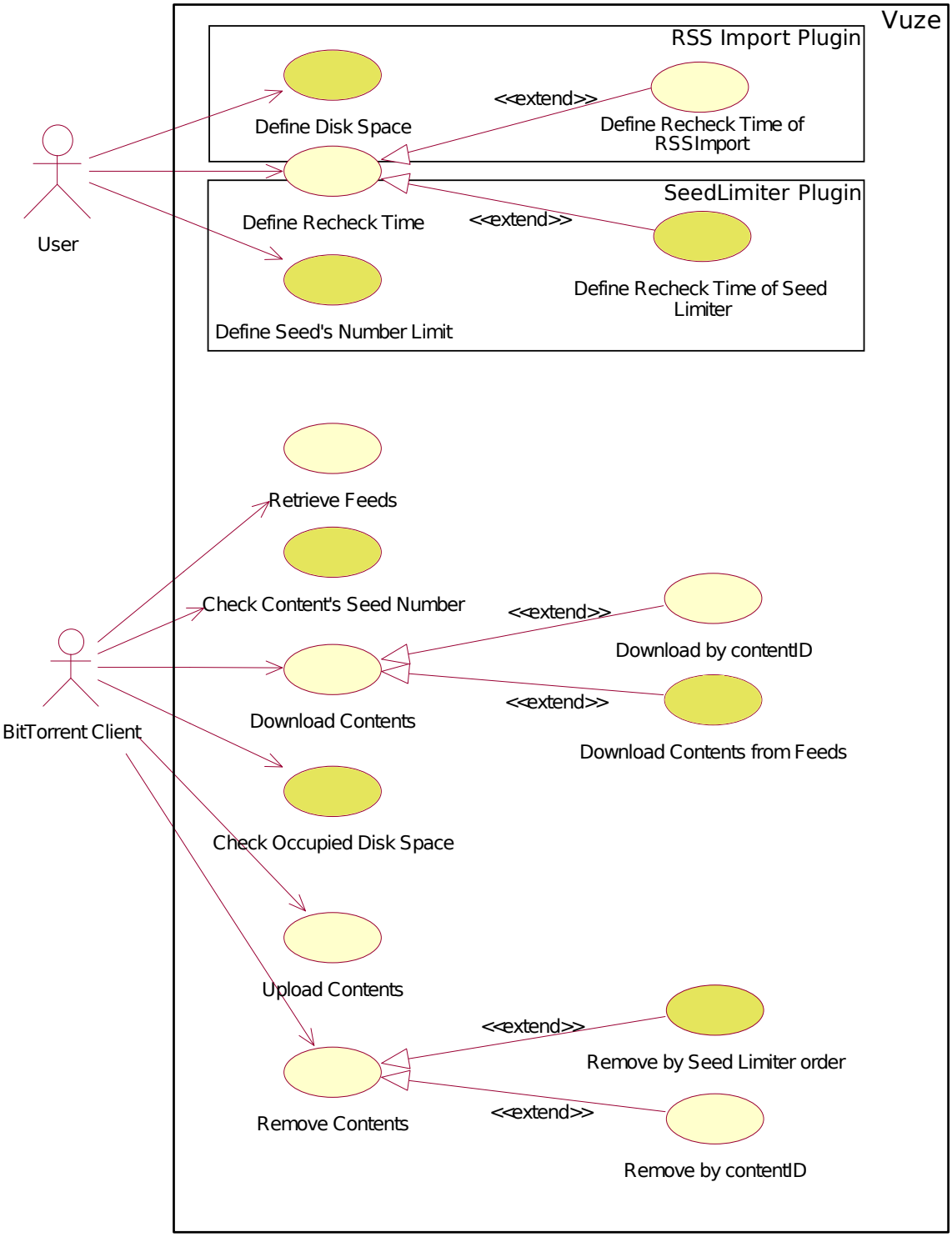


Figure 4.15: Vuze UML Use Cases

4.4.1 Vuze Remote Invocation Methods

As described in the previous subsection, the work developed to this thesis included the development of swervices directly interacting with the BitTorrent client. The implementation of this interactivity required the development of a JavaRMI server. This technology was chosen, among others, mainly due to two factors:

1. Both Web Services and Vuze are Java-based, so maintaining the same programming language for communications between them keeps system's simplicity;

2. JavaRMI allows the invocation of remote objects as easy as if they were local, which greatly simplifies communications and implementations.

The class diagram of RMI server is presented in figure 4.16, in which classes *getMaxSize()* and *setMaxSize()* are the ones invoked by the applet (section 4.4.3) and *showTorrent()* class is invoked by the *List Azureus' Activities* Web Service (section 4.3.3).

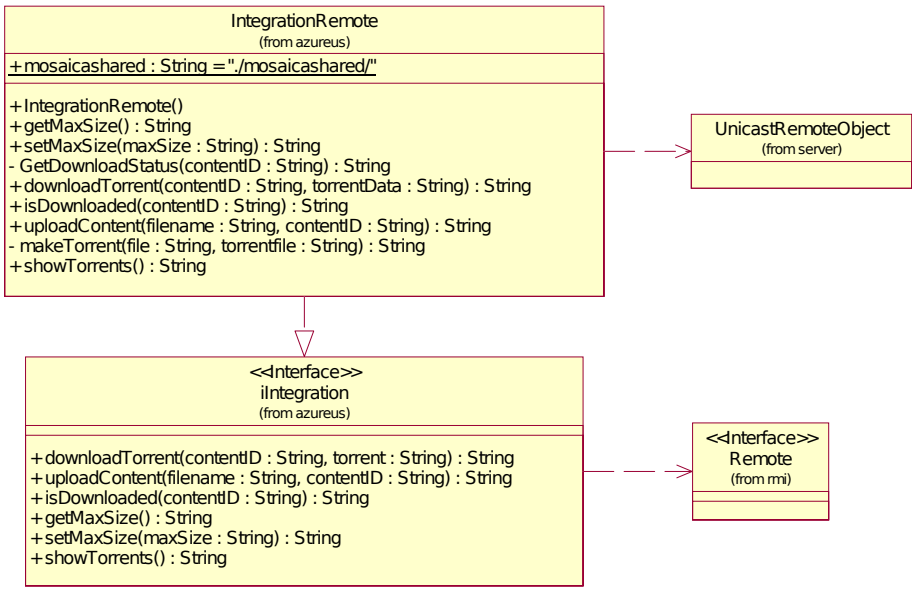


Figure 4.16: Class Diagram for Azureus Remote Methods

4.4.2 RSS Import Plugin

RSS Import is a plugin for Vuze, developed by Markus Baeker, currently in version 2.2.2 [74]. This plugin provides Vuze with the option to periodically check one or more RSS servers and download published torrent files. This process can be done according to rules expressed by filters created by the user. This plugin provides the answer to the objective of having the BitTorrent client downloading new contents without the interaction of users. It periodically consults the MOSAICA

RSS server, which returns, in each request, a random torrent file. Changes performed in this plugin allows it to check if the content's size - the *Check Occupied Disk Space* use case, described in table 4.4 -, together with other contents that could have been previously downloaded, exceeded or not the size defined by the user for the folder where contents are downloaded to. If there is enough space to download that content, then the *Download Content from Feeds* use case - in table 4.6 - is triggered.

This plugin, in its original version, accepts, as inputs, the URL of RSS servers, the filters to download torrent files and the auto-check time interval. The modifications made to it added one more field, in which a user can define the maximum size that MOSAICA shared folder may reach - *Define Disk Space* use case (figure 4.15) described in table 4.5. This modified plugin's behaviour is shown in figures 4.18 and 4.19, the sequence and collaboration diagrams - both diagrams focus only in the developed part and actions that are directly related with them, as this plugin was not created during this thesis. However, the class diagram of the plugin is presented in figure 4.17, gathering the developed and the pre-existing Java classes.

Table 4.4: Check Occupied Disk Space use case description

Use Case	Check Occupied Disk Space
Actor	BitTorrent client
Main Scenario	When Vuze gets a torrent file from the RSS server, it will extract the content's size and checks the space already occupied in the shared folder, calculating if the sum of both values are less than or equal to the size defined by the user.
Alternative Scenario (Not enough space allowed for content)	Vuze will discard that torrent file, waits until the next check and gets another torrent.

Table 4.5: Define Disk Space use case description

Use Case	Define Disk Space
Actor	User
Main Scenario	By default, Vuze starts with unlimited space for its shared folder (defined by *limit=0*), restricted to space available in the hard drive or partition in use. In RSS Import configuration panel, the user can define the maximum size of MOSAICA Azureus' shared folder. When saving configurations, the plugin checks if the desired size is available in the disk.
Alternative Scenario 1 (Not enough space on disk)	Vuze informs the user that the space he entered is not available, and will ask him to provide a new size.
Alternative Scenario 2 (Size definition not allowed)	If user enters a combination of characters different than an integer, Vuze does not accept it.

Table 4.6: Download Contents from Feeds use case description

Use Case	Download Contents from Feeds
Actor	BitTorrent client
Main Scenario	After checking if there is still enough space to download the content in the torrent file provided by the retrieved feed, Vuze starts the download process.

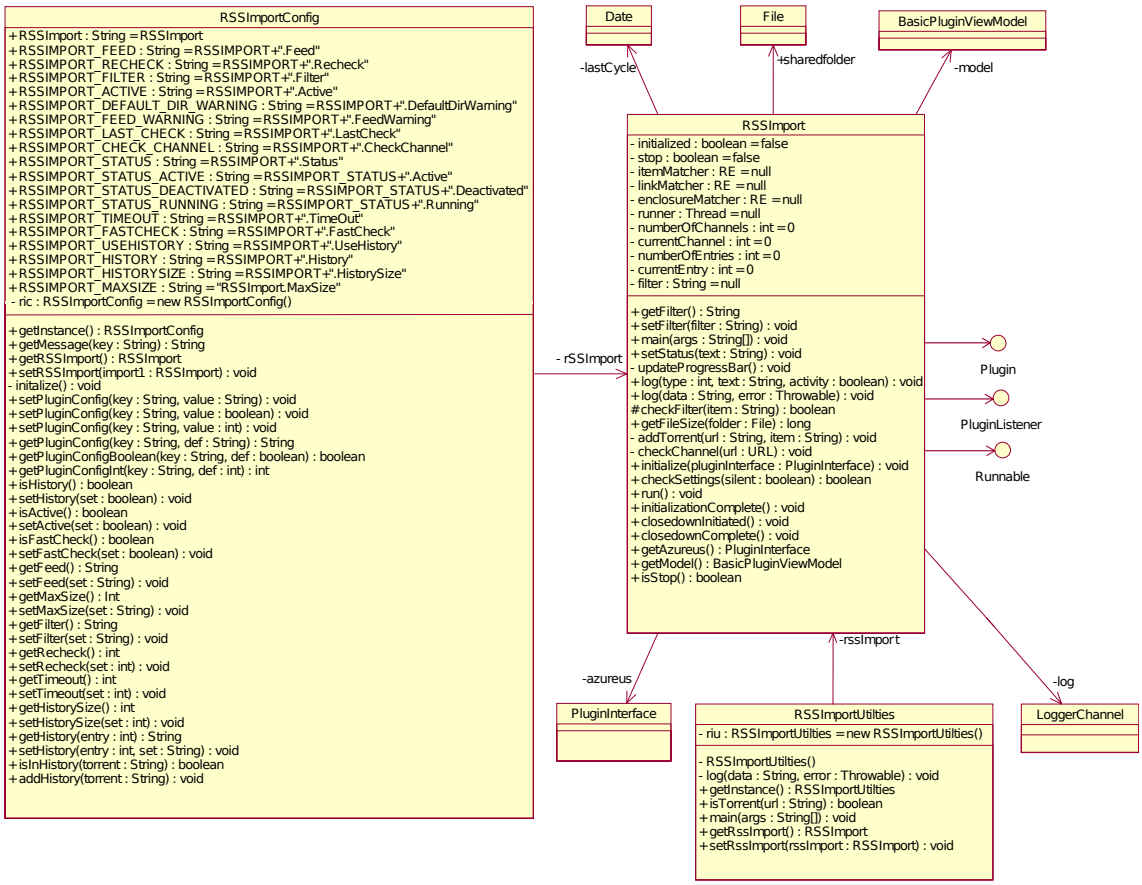


Figure 4.17: Class Diagram for Vuze's plugin *RSS Import*

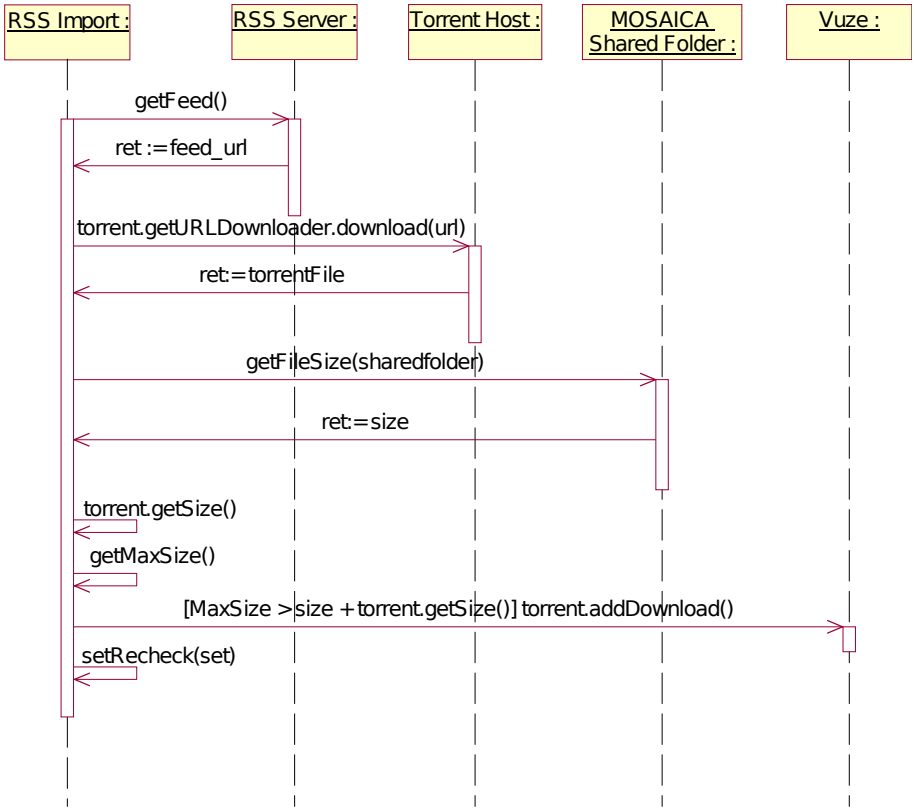


Figure 4.18: Sequence Diagram for Vuze's plugin *RSS Import*

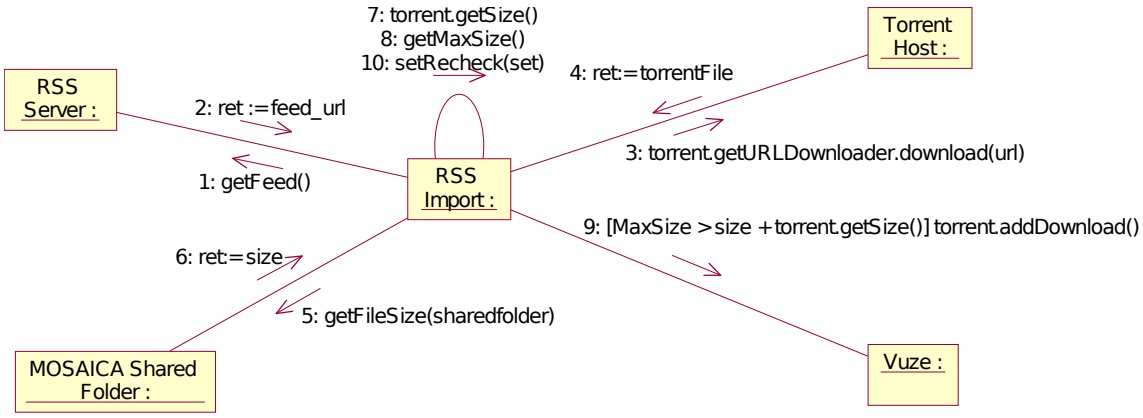


Figure 4.19: Collaboration Diagram for Vuze's plugin *RSS Import*

4.4.3 Shared Folder's Maximum Size Controller Applet

One requirement in the MOSAICA project was the possibility to control the maximum size of the shared folder used by Vuze, besides the existing configuration field in RSS Import configuration panel. The solution came under the form of a Java applet, which allows users to access it through a browser, just like they do when accessing to a Web Service. The only difference is that the applet is opened in a browser, but executed in the local machine, where Vuze is running. The use of an applet to control the size's definition, being published in only one place, allows users to access it just by typing the URL, thus quickly changing the size's definition. It may nevertheless, be replicated to other Web pages or servers.

This applet is intended to be used only by users that have the *MOSAICA Final User* package. This restriction is imposed by the installation of the package, which sets an environment variable, used by the applet to check if the Vuze application is installed or not. The applet, before allowing users to change the maximum size of the shared folder, checks first if the application is installed and then if it is running, because it makes no sense, besides being impossible, to allow changes when both condition are not satisfied. If, on the other hand, Vuze is installed and running, then the applet starts by getting the maximum size definition set in Vuze, which may then be changed. The operations performed by this applet are directly related with the RSS Import plugin - read the maximum size definition and setting it up - and are accomplished through the use of JavaRMI technology, establishing a connection between the applet and the Vuze's JavaRMI server.

Applets have, however, severe restrictions when running from browsers. As a security mechanism, when applets are executed inside browsers, they run in a "sandbox", which prevents the applet of accessing local data. As this applet has to, necessarily, access local data - it has to search for the operating system's environment variables -, an additional measure was needed. In fact, applets can access local data, but to do that, they need to be signed. So, the developed applet was self-signed with a Java generated digital certificate that, although not issued by a trust entity, allows users to run it successfully after accepting a security warning presented in the browser. This security warning informs users that the certificate for that applet is not recognized by any trusted source, but users can still accept it, executing the application at their own risk. This was a reasonable solution, mainly because of the high prices taken by certificate issuers companies.

When running this applet, the user is presented with information about Vuze's status - whether it is installed or running - through checkboxes, automatically checked accordingly, and with information messages, that can be emphasized or not, according with the message's importance. When submitting a new maximum size's definition, the applet interprets values as being in megabytes and then internally converts them to bytes, which is the magnitude that RSS Import plugin deals with. Whether submitting values is completed with success or not, a message is always presented to the user, informing him of the result of the operation. These methods are described in this

applet's class diagram (figure 4.20), and their interaction with other objects are shown in the sequence and collaboration diagrams in figures 4.21 and 4.22.

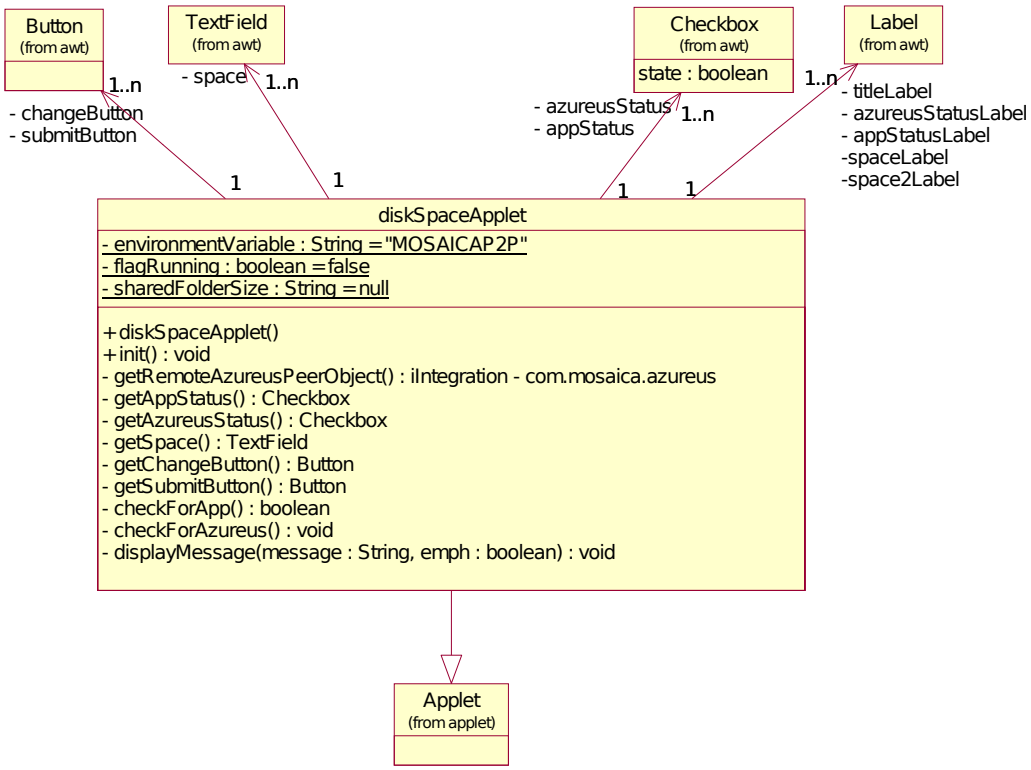


Figure 4.20: Class Diagram for Shared Folder's Maximum Size Controller Applet

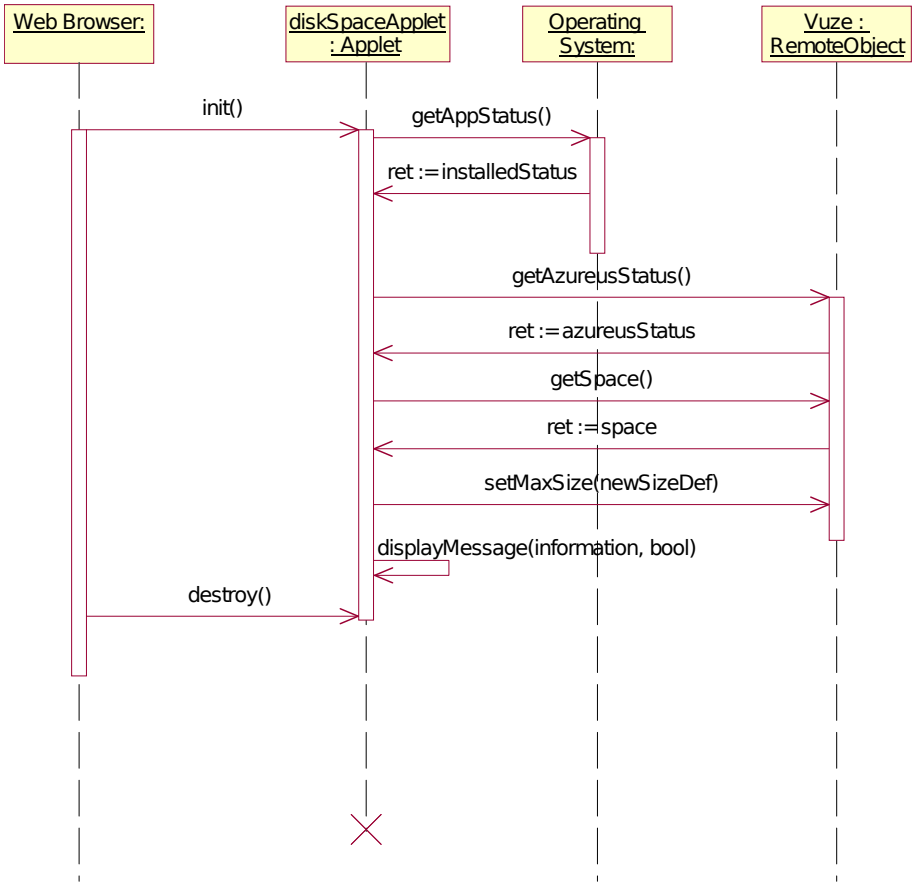


Figure 4.21: Sequence Diagram for Shared Folder's Maximum Size Controller Applet

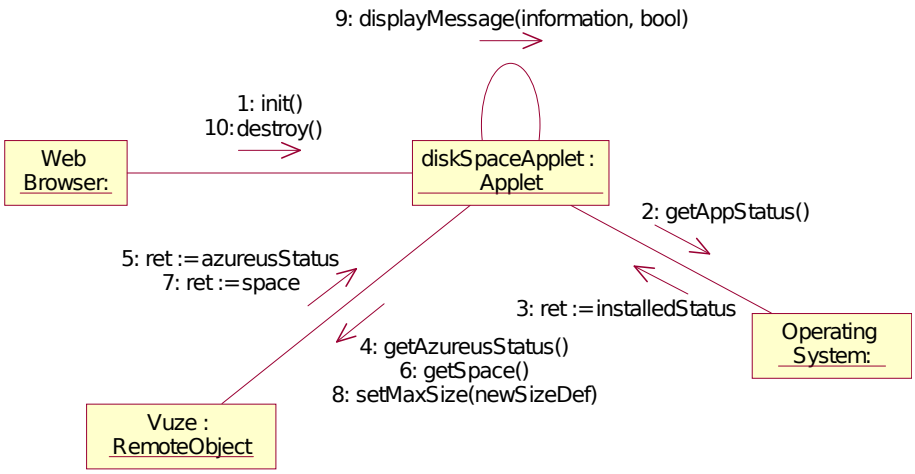


Figure 4.22: Collaboration Diagram for Shared Folder's Maximum Size Controller Applet

4.4.4 Seed Limiter Plugin

One of the goals of the MOSAICA platform is to get contents properly distributed, improving contents' availability for all MOSAICA users. However, in P2P networks there is not an ideal way of deciding if a content is well distributed or not, mainly due to highly transient crowds of users. And, as it can be seen in figure 2.7, in page 15, most P2P users normally upload data while downloading contents, leaving the swarm once the download finishes, limiting resources for users who later join the swarm. The two P2P packages developed in MOSAICA try to eliminate this situation, enabling peers to monitor if a content is becoming rare in the network, by checking the total number of seeders, for each content, and deciding if it is too low or not. But, as the download of contents is made in an automatic manner, with Vuze downloading all the contents that RSS Import finds in the RSS server, the objective is to distinguish which contents are worth being seeded. Automatically downloading and consequently seeding contents that are already distributed by a large number of peers is obviously not the objective. So, the SeedLimiter has been developed to ensure that the BitTorrent client, when operating without users' interaction, only downloads and seeds contents with few seeders. Moreover, that it seeds them while the number of known seeders does not overcome a defined number, checking each content's seeds number periodically, by asking the tracker for it, or consulting the swarm, when using DHTs.

The operations made by this plugin are illustrated in use case diagram of figure 4.15, and are detailed in tables 4.7 to 4.10. The class diagram of this plugin is presented in figure 4.23, and its sequence and collaboration diagrams are in figures 4.24 and 4.25.

Table 4.7: Define Seeds' Number Limit use case description

Use Case	Define Seeds' Number Limit
Actor	User
Main Scenario	By default, Vuze starts with limit of seeders set to 0, which disables limitations of seeds. In Seed Limiter configurations panel, the user can set the maximum number of a content's seeds, from which a content is considered to be properly distributed.
Alternative Scenario (Definition not allowed)	If user enters a combination of characters different than an integer, Vuze does not accept it.

Table 4.8: Define Recheck Time for SeedLimiter Plugin use case description

Use Case	Define Recheck Time for SeedLimiter Plugin
Actor	User
Main Scenario	By default, SeedLimiter recheck time is set to ten minutes. This value can be changed in Seed Limiter configurations panel.
Alternative Scenario (Definition not allowed)	If user enters a combination of characters different than an integer, Vuze does not accept it.

Table 4.9: Check Contents' Seeds use case description

Use Case	Check Contents' Seeds
Actor	BitTorrent client
Main Scenario	Vuze's SeedLimiter plugin checks the swarm for the number of seeds of each content that it is currently downloading or uploading. When some content has more seeds than the number defined by the user, the *Remove Content by SeedLimiter's order* use case is triggered.
Alternative Scenario (Cannot retrieve the number of seeds)	Vuze waits until the next check and then tries again to get the number of seeds in the swarm.

Table 4.10: Remove Content by SeedLimiter's order use case description

Use Case	Remove Content by SeedLimiter's order
Actor	BitTorrent client
Main Scenario	Content's download or upload is stopped and content and torrent file are removed from the disk.
Alternative Scenario (Content or torrent file cannot be removed)	An exception is triggered and a message is presented in the shell.

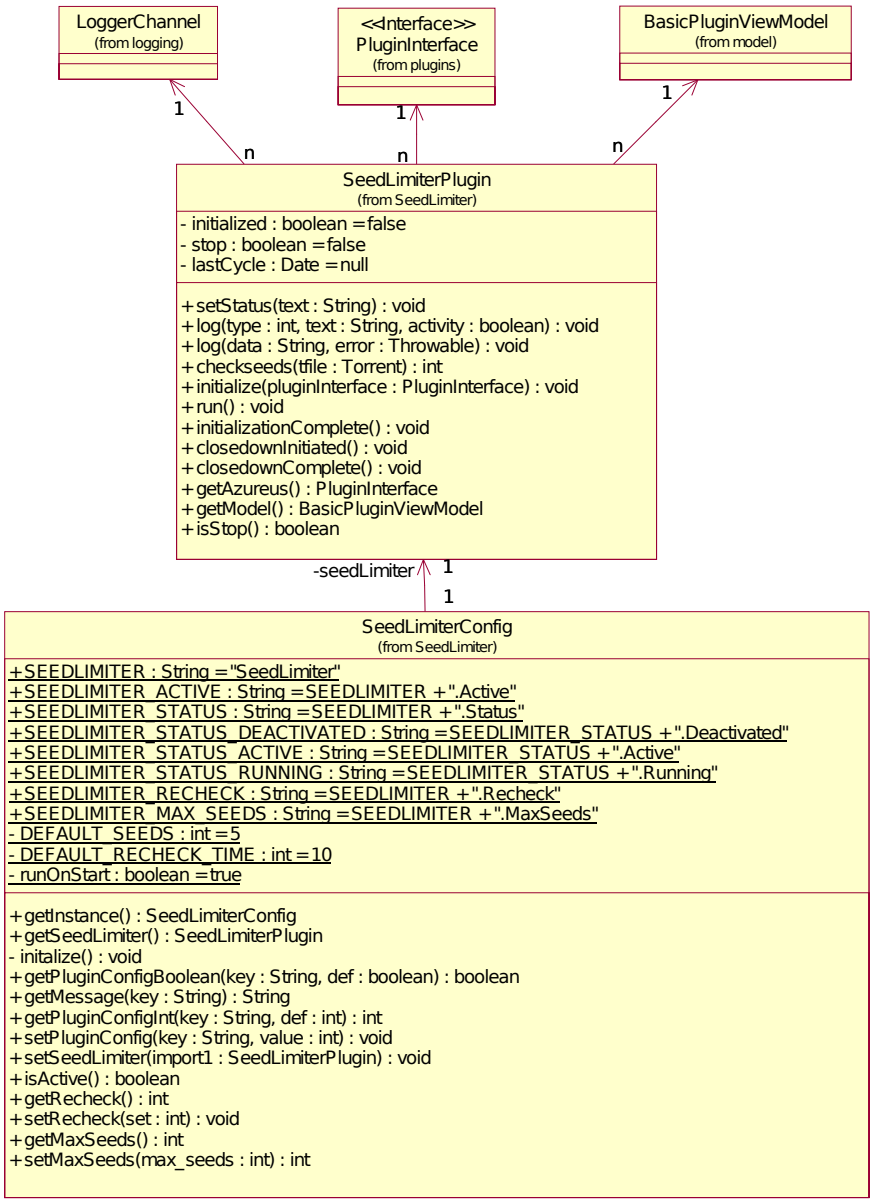


Figure 4.23: Class Diagram for Vuze's plugin *Seed Limiter*

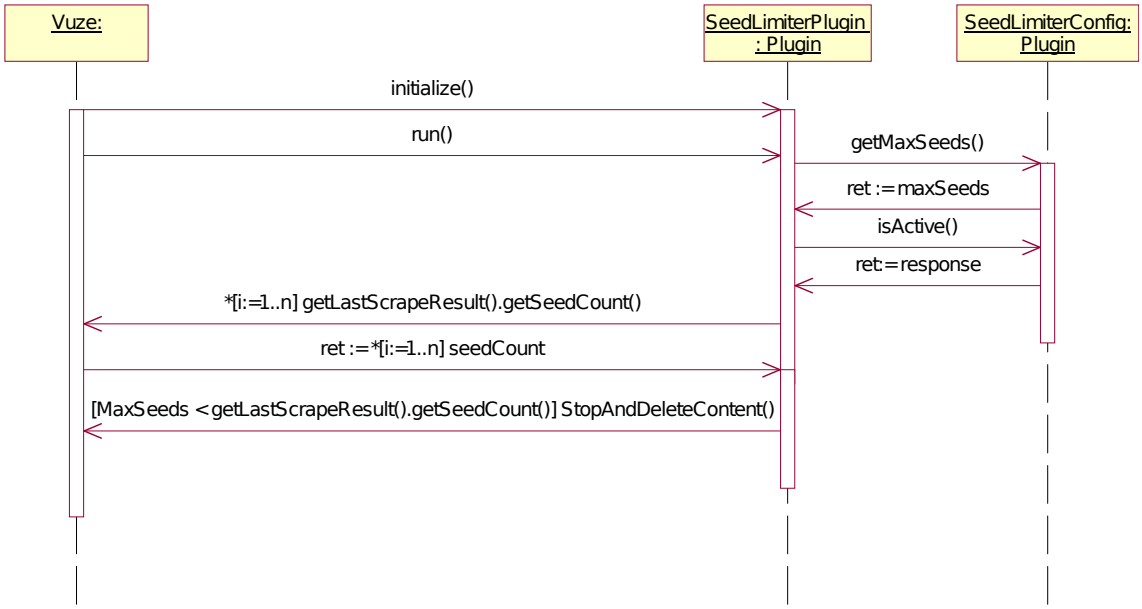


Figure 4.24: Sequence Diagram for Vuze's plugin *Seed Limiter*

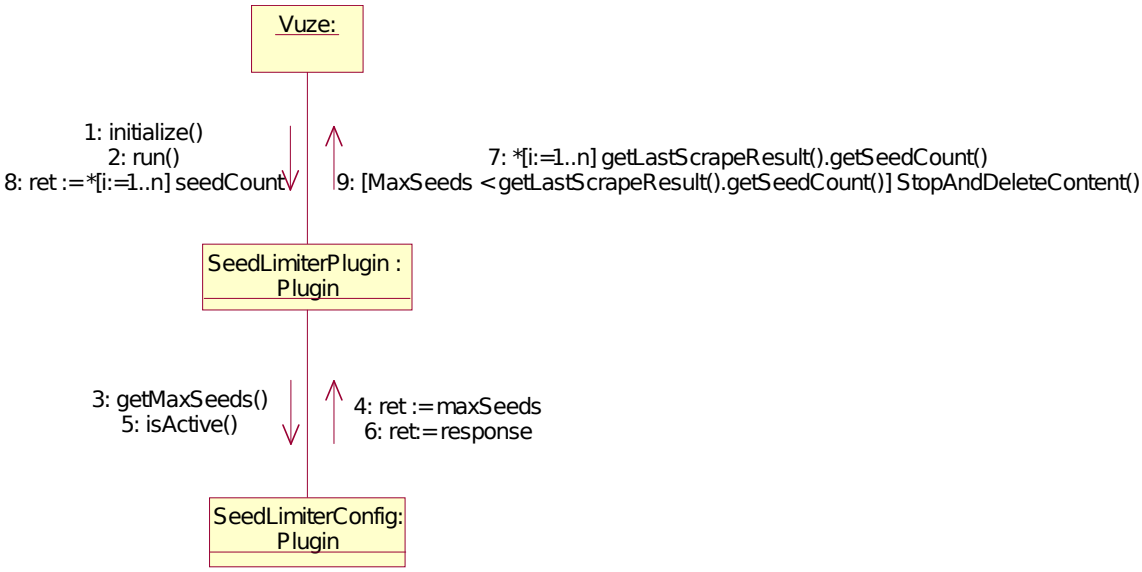


Figure 4.25: Collaboration Diagram for Vuze's plugin *Seed Limiter*

4.5 Deployment

Considering MOSAICA's two-layer model presented in [76], the work done for this thesis, mainly in the Web Services part, added a third layer to it. Figure 4.26 shows the overview of the three layers, with the JXTA and BitTorrent layers being part of the already deployed package, and the upper layer is related with the work done. The figure 4.27 represents the developed components, separated according to its place in the package.

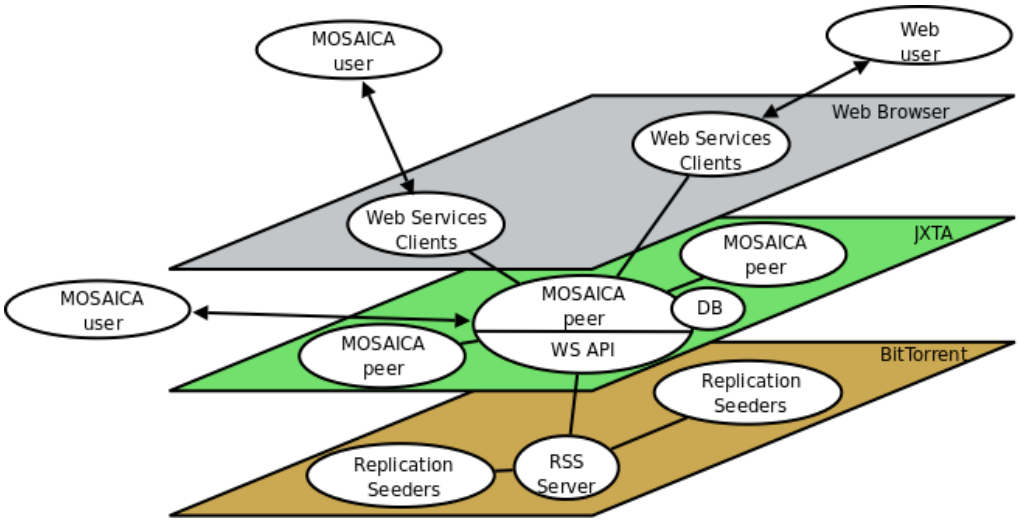


Figure 4.26: Three-layer model for MOSAICA platform

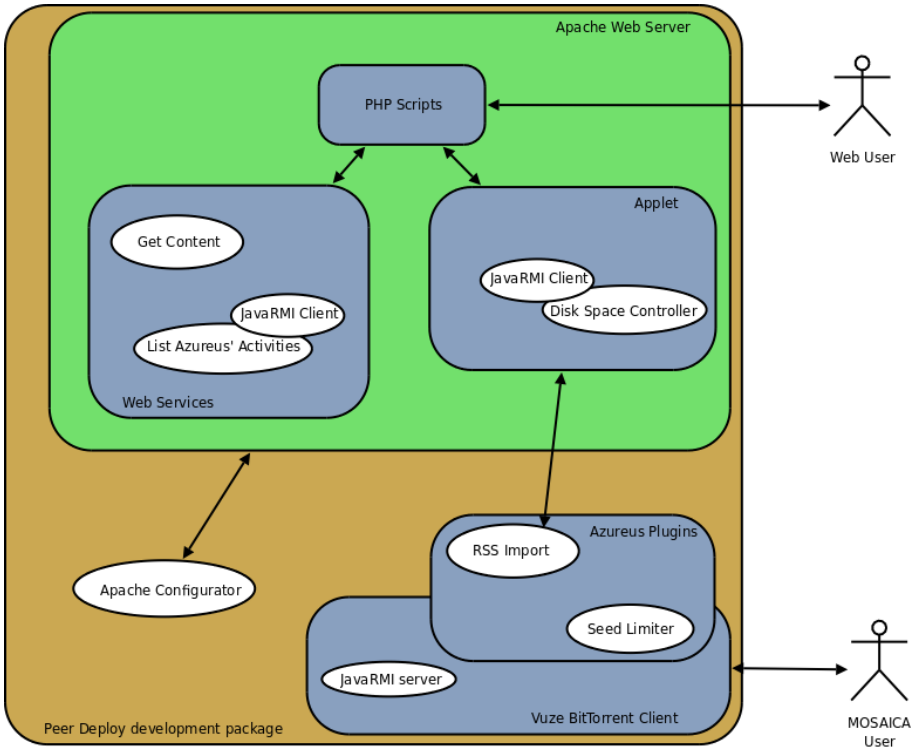


Figure 4.27: MOSAICA's developed components

As a result of the development and changes made, mainly in the *MOSAICA Peer Deploy Development* package, the deployment diagram previously presented in figure 4.1 was subject of a change, now replaced by the one from figure 4.28.

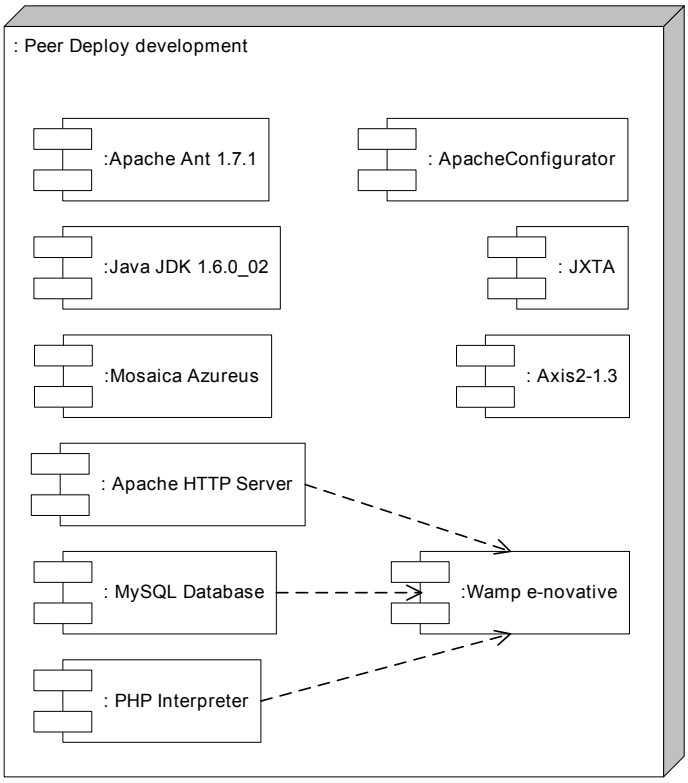


Figure 4.28: Final P2P-CMS Deployment Diagram for MOSAICA Peer Deploy Development package

Chapter 5

Analysis of the developed software

When introducing the MOSAICA platform, one question that may be raised is the reason why people should use it. In fact, people may think that is quicker to search contents in a search engine, like Google, and directly download contents, or torrent files, using, for the latter, any BitTorrent client. This is, however, easily explained: MOSAICA is oriented to a specific range of contents - cultural contents belonging to different religions and cultures, like Jewish cultural heritage contents. These contents may not be easily found in generalist search engines, and people who may search for such contents are the target of MOSAICA. Using MOSAICA also eases the search and the download processes: to use MOSAICA, users have to access to a web page, which only requires a Web browser, avoiding the need to install any other piece of software. Then, to search for contents, users may enjoy of semantic-based search mechanisms, allowing users to get contents even without knowing the complete name of it. All this tasks are performed according to the SOA paradigm, requiring no processing power to the user, since all the tasks are performed in the server's side, which is responsible for performing searches, download contents and present information to users through Web Services. Although the use of MOSAICA platform implies a slightly bigger overhead when using Web Services, it becomes irrelevant when facing all the advantages it has associated.

To demonstrate the advantages of using MOSAICA and Web Services, this chapter presents the tests made to MOSAICA and to the developed components. These tests were grouped into three different categories:

1. Test the Web Services' efficiency, in section 5.1;

2. Test functionalities of the developed components, in section 5.2;

3. Test Vuze's efficiency, in section 5.3.

5.1 Performance of Web Services

The tests done on the developed Web Services allowed to measure the response time of each request made to the peer who is executing them. To do this, it was used the JAMon API [77], integrated within *Get Content* and *List Azureus' Activities* Web Services' code. This tool measures the total amount of time elapsed since a request is issued until the answer is received back, using a configurable number of requests in each cycle. This way, it records, for each group of requests, the average time, the total time, the minimum time, the maximum time and the elapsed time for the last cycle of requests.

For these tests, JAMon was configurated to generate a hundred requests per cycle, repeating this cycle one hundred times, which results in a total of ten thousand requests made to the peer, a number that was considered to be enough to minimize deviations and to prove Web Services' liability, given the large number of requests processed in a small period of time. Also, these tests were performed twice: in the first one, the tests were executed from the peer in which the Web Services were running - the local host - and in the second one, representing a more realistic scenario, the requests were made from a remote machine to the peer. Although the second scenario corresponds to the most frequent situation when using Web Services, the first one serves as a comparison, providing an idea of the delay introduced by the network. The logs provided by JAMon were then parsed by a developed Java application, which extracted the three most relevant parameters: the minimum time, the maximum time and the average time.

To make comparisons easier, the result of the performed tests are grouped by Web Services. The tables 5.1 to 5.4 contain the average of each cycle and the average of each request, resulting from the amount of time obtained in the first row, divided by the number of requests of each cycle. The behaviour of all the extracted parameters along the duration of tests can be seen in the graphics of figures 5.1 to 5.6.

5.1.1 Get Content Web Service Tests

Table 5.1: Get Content Web Service Local Host Test

	Minimum Time	Maximum Time	Average Time
Average of each cycle	1,65 (ms)	172,91 (ms)	20,72 (ms)
Average of each request	16,50 (μs)	1,73 (ms)	207,17 (μs)

Table 5.2: Get Content Web Service Remote Host Test

	Minimum Time	Maximum Time	Average Time
Average of each cycle	21,61 (ms)	2497,13 (ms)	211,66 (ms)
Average of each request	216,10 (μs)	24,97 (ms)	2,12 (ms)

5.1.2 List Azureus' Activities Web Service Tests

Table 5.3: List Azureus' Activities Web Service Local Host Test

	Minimum Time	Maximum Time	Average Time
Average of each cycle	0,28 (ms)	32,04 (ms)	4,85 (ms)
Average of each request	2,80 (μs)	320,40 (μs)	48,52 (μs)

Table 5.4: List Azureus' Activities Web Service Remote Host Test

	Minimum Time	Maximum Time	Average Time
Average of each cycle	6,10 (ms)	89,10 (ms)	10,16 (ms)
Average of each request	61,00 (μs)	891,00 (μs)	101,58 (μs)

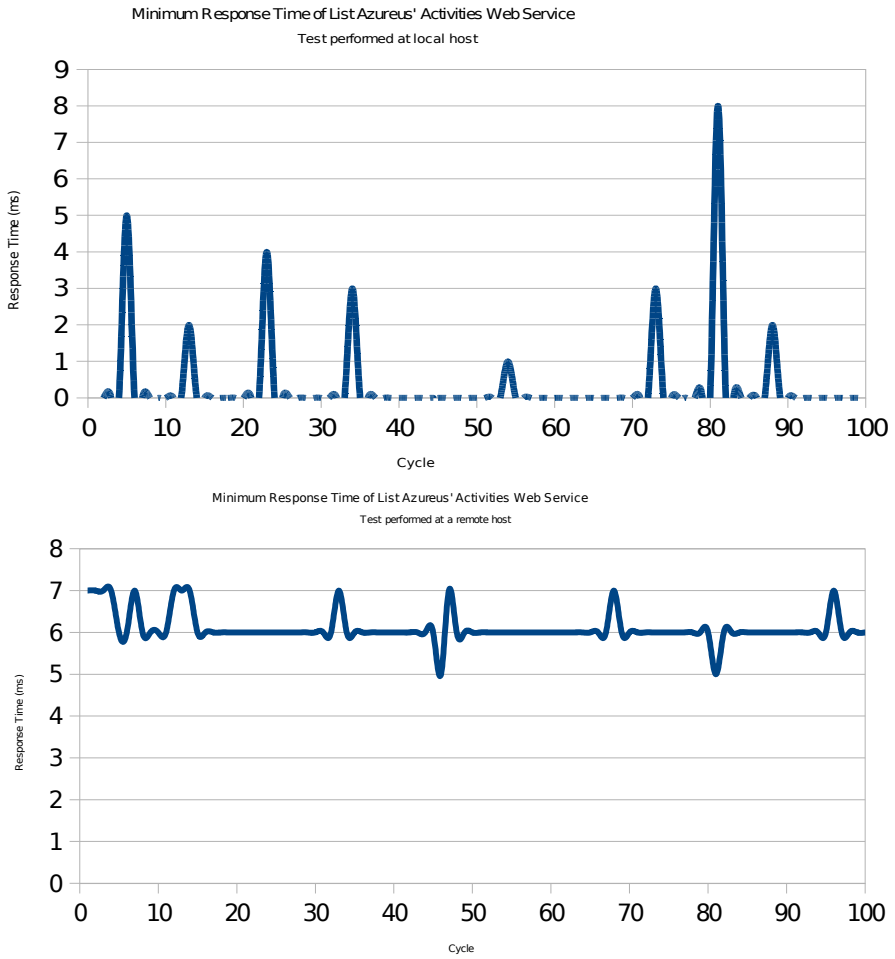


Figure 5.1: Comparison of minimum response times in List Azureus' Activities Web Services

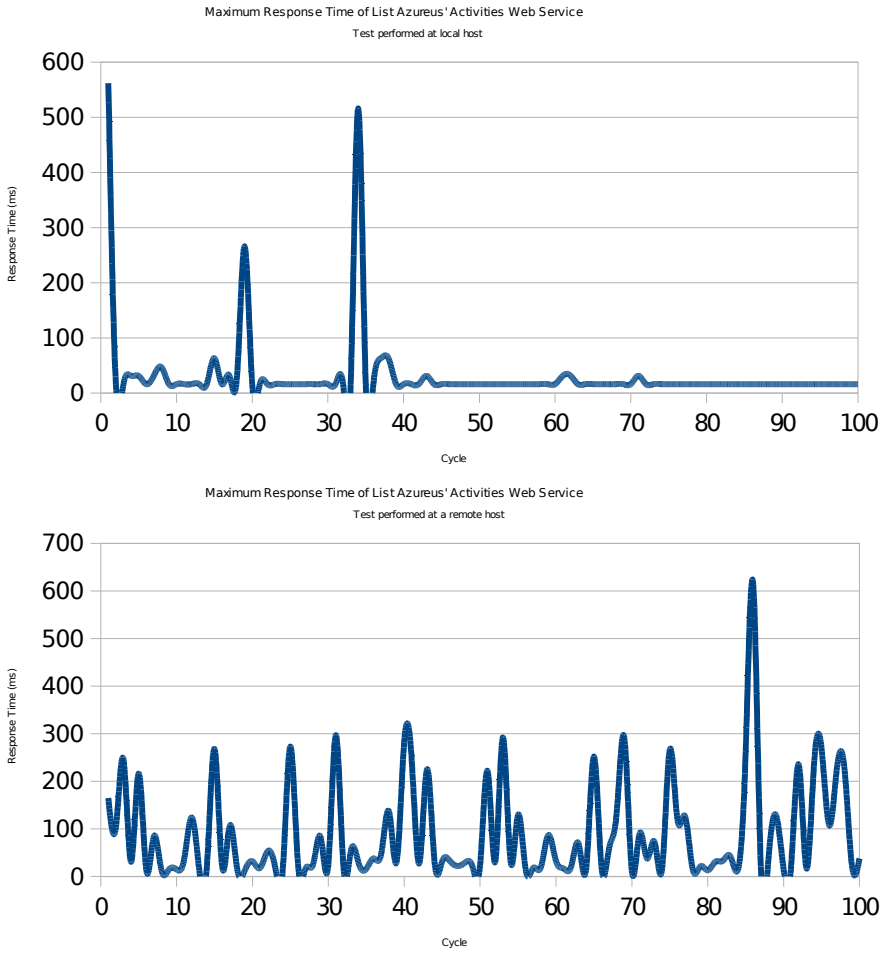


Figure 5.2: Comparison of maximum response times in List Azureus' Activities Web Services

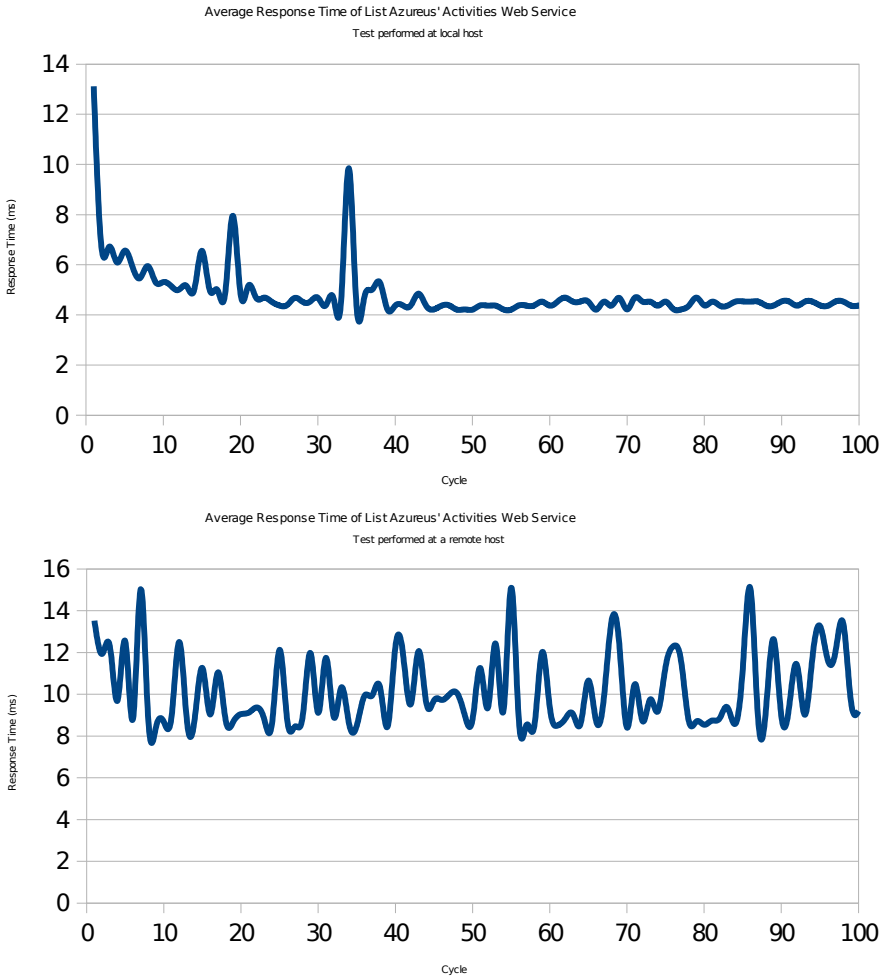


Figure 5.3: Comparison of average response times in List Azureus' Activities Web Services

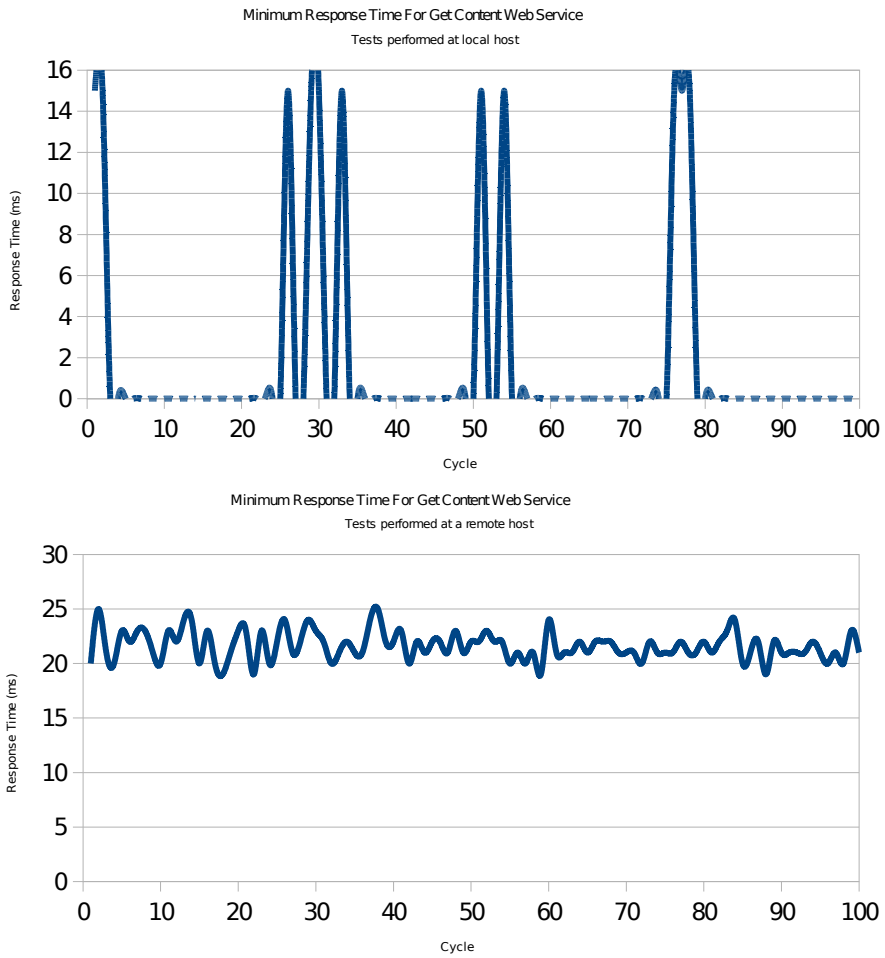


Figure 5.4: Comparison of minimum response times in Get Content Web Services

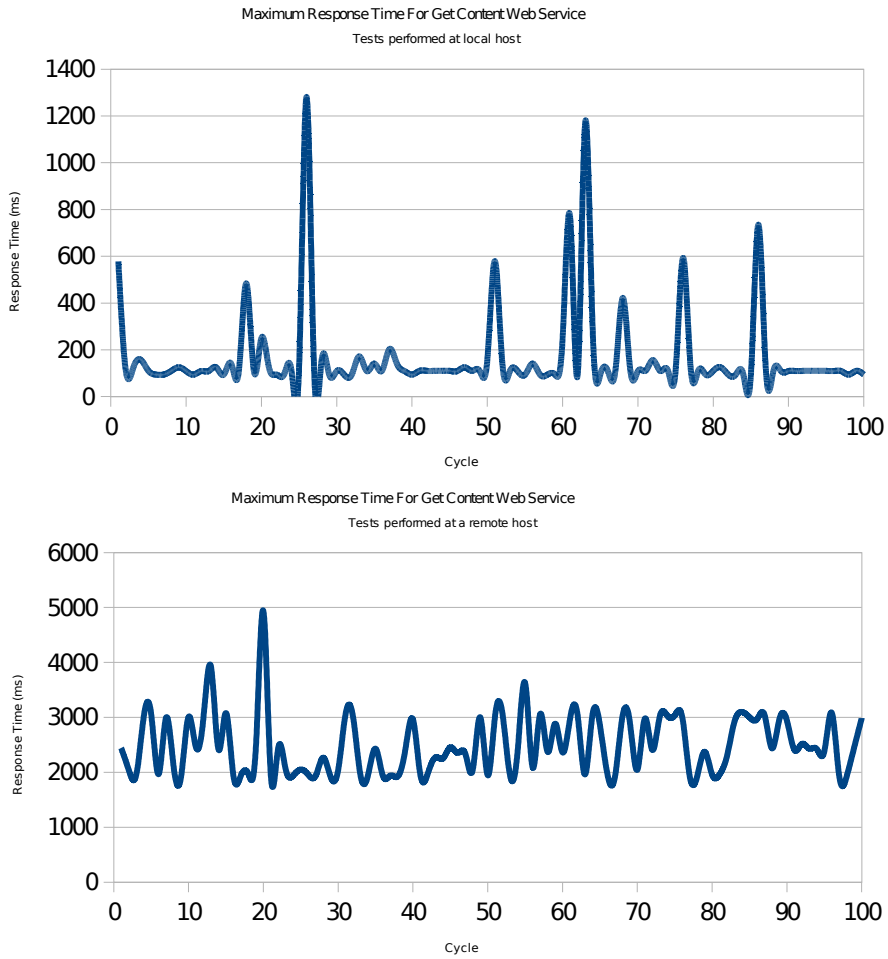


Figure 5.5: Comparison of maximum response times in Get Content Web Services

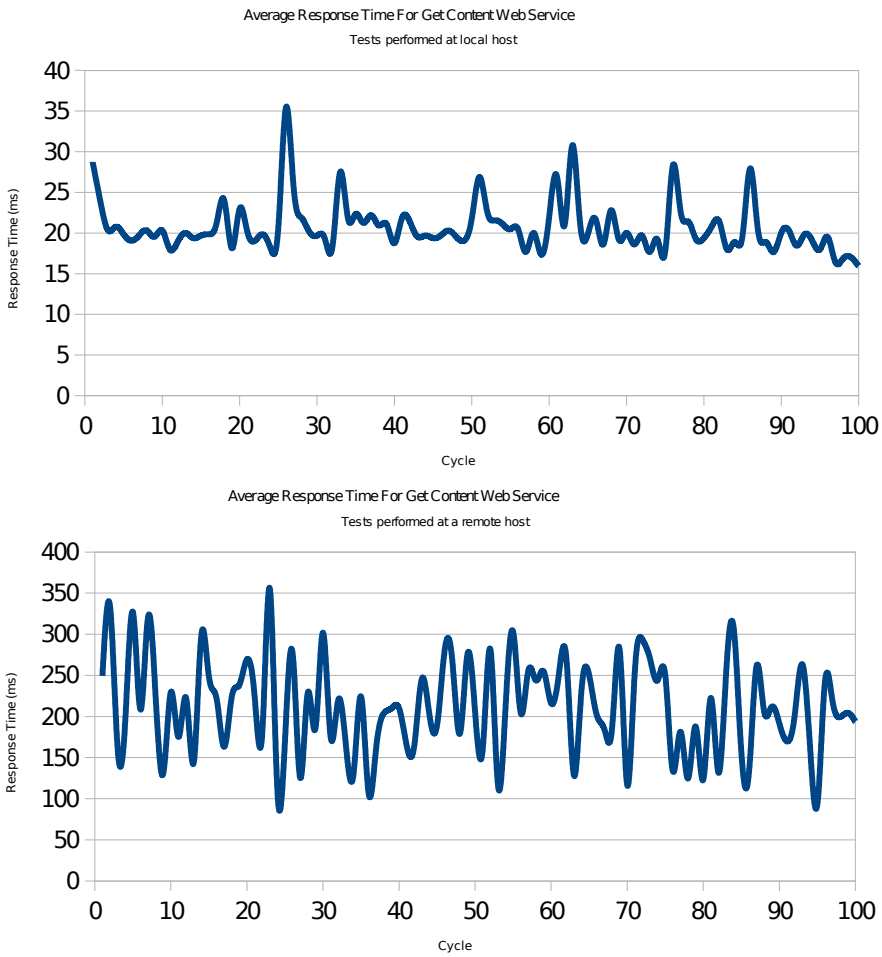


Figure 5.6: Comparison of average response times in Get Content Web Services

5.2 Functional Tests

The following functional tests try to embrace several situations that may occur in a normal use of the MOSAICA platform, verifying if software's behaviour matches the specifications described in chapter 4. During the tests, the contents registered in MOSAICA P2P network consisted of three ISO files, all three of them with different Linux distributions: DamnSmallLinux 0.5, a file of 35.2 MB, Ubuntu 8.10 Desktop Edition, with 698 MB and FEUPLive DVD 2.0, with 1.84 GB. Despite the fact that these files cannot be considered as cultural contents, and consequentially are not the type of contents shared in MOSAICA, they allow the execution of tests with different file sizes and the execution of tests with large files, as the ones that can be shared across MOSAICA platform, like movies, in the context of Jewish cultural heritage.

Table 5.5: Functional Test 1

Targeted Application	Disk Space Controller Applet
Scenario Definition	The applet is executed when Vuze is not installed.
Expected Result	An alert message should be displayed and shared size field should not be changeable.
Result	The applet is presented with all options disabled and the message "MOSAICA application is not installed" is displayed.

Table 5.6: Functional Test 2

Targeted Application	Disk Space Controller Applet
Scenario Definition	The applet is executed when Vuze is installed but not running.
Expected Result	An allert message should be displayed and size should not be changeable.
Result	Applet detects that Azureus is installed, but all options are disabled and the message "Azureus is not running." is displayed.

Table 5.7: Functional Test 3

Targeted Application	Disk Space Controller Applet
Scenario Definition	Shared size field is left empty when Vuze is installed and running.
Expected Result	Applet must ask user for filling the shared size field.
Result	Applet displays the message "You must enter some value.", and waits for another input.

Table 5.8: Functional Test 4

Targeted Application	Disk Space Controller Applet
Scenario Definition	Shared size field is filled with'0' when Vuze is installed and running.
Expected Result	Applet must accept this value, and update Vuze RSS Import plugin's Maximum Value setting with the same value.
Result	It is displayed the message "Azureus shared folder's size is now unlimited.", and RSS Import Maximum MOSAICA shared folder size field has the value '0'.

Table 5.9: Functional Test 5

Targeted Application	Disk Space Controller Applet
Scenario Definition	Shared size field is filled with '100' (a valid value) when Vuze is installed and running.
Expected Result	Applet must accept this value, and update Vuze RSS Import plugin's Maximum Value setting with the same value.
Result	It is displayed the message "Azureus shared folder is now limited to 100MB.", and RSS Import Maximum MOSAICA shared folder size field has the value '104857600'.

Table 5.10: Functional Test 6

Targeted Application	Disk Space Controller Applet
Scenario Definition	Shared size field is filled with '-100' (a negative number) when Vuze is installed and running.
Expected Result	Applet must not accept this value and must ask the user for a valid number.
Result	Applet displays the message "You must enter an integer value, equal or bigger than 0.", and waits for another input.

Table 5.11: Functional Test 7

Targeted Application	Disk Space Controller Applet
Scenario Definition	Shared size field is filled with 'abcd' (string of characters) when Vuze is installed and running.
Expected Result	Applet must not accept this value and must ask the user for a valid number.
Result	Applet displays the message "You must enter an integer value, equal or bigger than 0.", and waits for another input.

Table 5.12: Functional Test 8

Targeted Application	Disk Space Controller Applet
Scenario Definition	After applet detects that Vuze is installed and running, connection is broke and shared size field is filled with a valid value.
Expected Result	Applet must warn the user that value could not be submitted.
Result	Applet displays the message "There was an error submitting new value." and RSS Import Maximum MOSAICA shared folder size field remains with its value unchanged.

Table 5.13: Functional Test 9

Targeted Application	Vuze RSS Import Plugin
Scenario Definition	Maximum Mosaica Shared Folder size field is filled with '0', in RSS Import options page.
Expected Result	Plugin must accept this value, disabling space's limit for download.
Result	RSS Import plugin accepts value's submission.

Table 5.14: Functional Test 10

Targeted Application	Vuze RSS Import Plugin
Scenario Definition	Maximum Mosaica Shared Folder field is filled with '-100' (negative number), in RSS Import options page.
Expected Result	Plugin must not accept this value.
Result	The field does not accept the input of the minus signal, so entering '-100' is not possible.

Table 5.15: Functional Test 11

Targeted Application	Vuze RSS Import Plugin
Scenario Definition	Maximum Mosaica Shared Folder field is filled with 'abcd' (string of characters), in RSS Import options page.
Expected Result	Plugin must not accept this value.
Result	The field does not accept characters as inputs, so entering 'abcd' is not possible.

Table 5.16: Functional Test 12

Targeted Application	Vuze RSS Import Plugin
Scenario Definition	Maximum Mosaica Shared Folder field is left empty, in RSS Import options page.
Expected Result	Plugin must not accept this value.
Result	To left the field empty is necessary to erase the previous value. When any action is performed, the empty field recovers the last value, being impossible to submit requests without any value.

Table 5.17: Functional Test 13

Targeted Application	Vuze SeedLimiter Plugin
Scenario Definition	Recheck time and Maximum number of seeds fields are filled with '0', in SeedLimiter options page.
Expected Result	Plugin must accept both values, updating each defined value in the next seeds' check.
Result	SeedLimiter plugin accepts both values.

Table 5.18: Functional Test 14

Targeted Application	Vuze SeedLimiter Plugin
Scenario Definition	Recheck time and Maximum number of seeds fields are filled with '-10' (negative number), in SeedLimiter options page.
Expected Result	Plugin must not accept any of these values.
Result	The field does not accept the input of the minus signal, so entering '-10' is not possible.

Table 5.19: Functional Test 15

Targeted Application	Vuze SeedLimiter Plugin
Scenario Definition	Recheck time and Maximum number of seeds fields are filled with 'abcd' (string of characters), in SeedLimiter options page.
Expected Result	Plugin must not accept any of the values.
Result	The field does not accept characters as inputs, so entering 'abcd' is not possible.

Table 5.20: Functional Test 16

Targeted Application	Vuze SeedLimiter Plugin
Scenario Definition	Recheck time and Maximum number of seeds fields are left empty, in SeedLimiter options page.
Expected Result	Plugin must not accept any of these values.
Result	To left fields empty is necessary to erase the previous values. When any action is performed, the empty field recovers the last value, being impossible to submit requests without any value.

Table 5.21: Functional Test 17

Targeted Application	Get Content Web Service
Scenario Definition	Request is submitted with an empty contentID.
Expected Result	Web service must not do anything.
Result	Browser presents an empty answer.

Table 5.22: Functional Test 18

Targeted Application	Get Content Web Service
Scenario Definition	Request is submitted with a valid contentID.
Expected Result	Web service must return the URL for matching content's HTTP download.
Result	After providing the contentID "FXG7ilVTHTpi9aK+aba+", the browser displays "http://194.117.26.55:8000/mosaicashared/ubuntu-8.10.iso" as an hyperlink to the matching content. The download made through HTTP took 2h15m29s to complete, with an average download rate of 78.02 KB/s.

Table 5.23: Functional Test 19

Targeted Application	Get Content Web Service
Scenario Definition	Request is submitted with a non-existing contentID.
Expected Result	Web service must return a message informing that provided contentID was not found.
Result	Browser presents the message "ERROR: The torrent file with the provided contentID was not found in server."

Table 5.24: Functional Test 20

Targeted Application	List Azureus' Activities Web Service
Scenario Definition	A first request is submitted when Vuze has nothing in download/upload list. A second request is submitted when Vuze is downloading or uploading a content.
Expected Result	First, the web service must return an empty message and, in the latter request, the web service must return information about the contents currently at Vuze's download or upload list.
Result	When Azureus has no items in its download/upload list, the returning message is empty. After registering a content in JXTA peer, Vuze puts the content in the upload list, and browser displays the information: NAME: ubuntu-8.10.iso STATE: Seeding CONCLUDED: 100,0% SEEDNUMBER: 1 UPSPEED: 0 bits/s SHARERATIO: 0.0

Table 5.25: Functional Test 21

Targeted Application	ApacheConfigChecker Utility
Scenario Definition	Peer Deploy Development package is initiated from a location, closed and, after moving the package to another location in the disk, is re-initiated.
Expected Result	When executed the first time, the alias to the current location of MOSAICA's shared folder must have been defined in Apache's configuration file. After the second execution, the alias must have been updated with the new location of MOSAICA's shared folder.
Result	When Peer Deploy Development folder is located in disk's root (C:\) and the package is executed from that location (C:\Peer Deploy Development), Apache's configuration file (httpd.conf) has the alias defined as *Alias /mosaicashared "C:\Peer Deploy Development\Mosaica azureus\mosaicashared"*. After moving the folder to, for instance, programs' folder (C:\Program Files), the alias defined in httpd.conf is updated to *Alias /mosaicashared "C:\Program Files\Peer Deploy Development\Mosaica azureus\mosaicashared"*.

5.3 Distribution of contents

Finally, to assess the correct functioning of the developed plugins working together with Vuze, three other tests are defined. These tests were made in a swarm with four peers, using the contents already mentioned for the functional tests.

Table 5.26: Distribution of Contents Test 1

Targeted Application	Vuze RSS Import plugin
Scenario Definition	Three different contents, with different sizes, are registered at one peer. The rest of the peers will check this peer's RSS server in every five minutes. The amount of time that takes to all the contents to be downloaded by all peers is measured.
Expected Result	Contents should be available in all peers. The total amount of time will depend on the size of each content and the probability of always retrieving different feeds, which depends on the total number of registered feeds.
Result	It took 39m39.719s to the first peer to have all the registered contents, 44m26.672s to the second peer and 47m20.015s to the last peer download all the contents, in a total of 2618.22 MB transfered to each peer, corresponding to an average download rate of, approximately, 944 kB/s.

Table 5.27: Distribution of Contents Test 2

Targeted Application	Vuze RSS Import plugin
Scenario Definition	Three different contents, with different sizes, are registered at one peer. Two of these peers have a shared folder's size limitation of 50 MB and the other have no space limitation. All peers will check this peer's RSS server in every five minutes. The total amount of time until all contents become available in all peers is measured.
Expected Result	Both peers with space limitation will only download one content (the one that occupies 35.2 MB), and the other peer must download all available contents. Contents should be available at all peers, obeying to each peer space's limitation.
Result	The two first peers to get all allowed contents downloaded had accomplished this after 4m43.156s and 19m25.906s, and the third peer had downloaded all the contents after 24m59.641s. The first two only had downloaded the content with 35.2 MB, while the third peer downloaded all the contents.

Table 5.28: Distribution of Contents Test 3

Targeted Application	Vuze RSS Import and SeedLimiter plugins
Scenario Definition	Three different contents, with different sizes, are registered in one peer. All peers use SeedLimiter to limit seeds' number to "2", not downloading contents available in more than two peers. RSS server is contacted every five minutes, the same amount of time used between each check on the contents' seeds number. The total amount of time it takes for contents to be available in all peers is measured.
Expected Result	Contents should be available at all peers, obeying to each peer seeds' limitation.
Result	All content started by having one seed (the peer into which they were registered). The smallest content (35.2MB) became available in two of the peers after 9m25.516s, the middle one (698MB), after 13m01.078s and the biggest one (1.84GB) after 23m02.735s. Meanwhile, these contents were being downloaded by more peers, but were deleted when Seedlimiter detected that there were already the defined number of peers in the swarm. The final scenario consisted of one of the peers with no contents being seeded, while one peer had the biggest content, and the other peer had the other two contents. The primary peer (the one that have all contents) remained with all the contents.

5.4 Conclusions of performed tests

The performed tests demonstrate that the use of Web Services do not imply additional significant latencies when compared with generalist search engines. In fact, when performing a search in Google, for a list of results with five hundred millions matches, Google took approximately four hundred miliseconds. This difference is, however, irrelevant to the user because of the human perception, and both response time may seem to the user as an "almost-instantaneous" response.

Also, all functional tests were successfully accomplished, delivering results that proved that the developments made were in accordance to the specifications made, thus fulfilling the proposed functional goals.

In what concerns to the achieved transfer rates, in tests described in tables 5.26 to 5.28, it can be seen that the use of the P2P paradigm leads to high download rates, that can be even higher in swarms with more peers. However, remarking to the test described in table 5.22, the download using HTTP was a little slow, with download rates never overcoming a few hundreds of KB/s, which is however faster and better than the download rate obtained in networks that block all P2P traffic. These rates are, however, related to the hardware of the web server, being possible to achieve, with this protocol, greater download rates with faster machines.

Chapter 6

Conclusions and Future Work

The work conducted during this thesis had the objective to provide contents shared in MOSAICA P2P network to all users using a minimum of equipment: a computer, an Internet connection and a browser. Independently of where users may access the Web, with the developed software users can enjoy of P2P advantages as easy as checking their e-mail. In addition, fairness of use issues of the BitTorrent protocol were also addressed and implemented through a modified version of the Vuze client, the MOSAICA Azureus client, with appropriate plugins. These modifications improved in a seamless manner, without the direct interaction of the user, the availability of contents, thus helping achieving its objectives of spreading cultural heritage contents.

This chapter discusses the achievement of the proposed objectives of this thesis and points suggestions on the future work that can be developed in MOSAICA platform.

6.1 Objectives' Achievement

P2P and BitTorrent are, nowadays, widely adopted technologies to the most disparate ends, due to all the advantages they have, as shown in chapters 2 and 3. Benefiting of so many advantages, not only domestic Internet users but also corporate entities started to use them, as a mean to be up-to-date in the world markets, providing technologically advanced solutions and reaching the maximum number of clients as possible. The latter motive is the main concern of MOSAICA: the project intends to promote cultural, religious and racial pluralism through the dissemination of multi-cultural contents, which can only be achieved by providing these contents to the maximum number of users as possible, instructing people of different culture's pratices and habits. This task can only be accomplished using tools that are themselves capable of reaching people through standard technologies, independently of platforms. This, in part, explains the fact why MOSAICA and the developed software mentioned in chapter 4 use Java as a base for the entire system.

Keeping these objectives in mind, the development of the *GetContent* and the *List Azureus' Activities* Web Services gives users the ability to enjoy the advantages of P2P networks, downloading shared contents using a browser instead of specific P2P software, and monitoring, in real time, the downloads and uploads managed by the BitTorrent client. These Web Services, together with the Disk Space Controller Applet, being presented to users using HTML and PHP assure the compliance with most operating systems and browsers, overcoming difficulties that other technologies face when being deployed for large-scale use and some networks' limitations concerning to the use of the BitTorrent protocol.

Controlling the distribution of contents shared in MOSAICA is directly related with the number of available copies in the swarm, i.e., the number of peers that have concluded the download of a certain content and remain connected, uploading parts of it to others - the *seeders*. As mentioned in section 2.2.2, the more seeds a content have, the more effective is the BitTorrent protocol transferring data, a consequence of its algorithms for contents' distribution. Knowing this, the chosen parameter that decides if a content is properly distributed or not is precisely the number of seeds a content has. Also, as mentioned previously, peers in a network have a random behaviour, and it is quite possible for a content to have, at a certain moment, an high number of peers and moments later that number has largely decreased. So, the task of checking the number of seeds has to be performed periodically. Also, if these plugins equip a large number of peers (ideally, the plugins are integrated into all MOSAICA users' BitTorrent clients, but plugins can be disabled), even when a massive number of seeds disconnect, stop seeding contents, the other peers will detect that the number of peers is becoming low, depending on the number defined by each user in the plugin configuration panel, and start to act as seeders for the contents which are becoming rare in the network. The operation of helping a content that is running out of seeds may however imply a certain delay, because of the aleatory retrieval of feeds, that can return a torrent file for a content that does not have the need to be distributed by one more peer, as well as for a poorly distributed content.

6.2 Future Work

Although MOSAICA platform is actually a full functional platform, it has a great potential to grow and to be extended, as well as to integrate the latest level of development of the technologies involved in its creation. The following suggestions aim precisely at this, and are based in the research made during this thesis, addressing subjects discussed in chapters 2 and 3.

One of the improvements that can be made to MOSAICA Web Services is the use of bindings for HTTPS (mentioned in section 2.3.3), as a way to improve security when using MOSAICA, since actually, when requests are sent to the peer, data is unencrypted, and can be seen using a

packet sniffer, like WireShark. Although the *contentIDs* and search queries may not be sensitive information, the user name and password can be.

Another idea for the Web Services is related with the display of results from searches. When performing a search, when a content is found, all the user can see is the correspondent *contentID*. This could be subject of another Web Service, that could match the *contentID* with the content's title that is registered in the metadata distributed database and even presenting part of this metadata to the user. This could be helpful to allow the user to decide which of the returned results are the most useful or desired.

Another additional mechanism that could be implemented in the developed GetContent Web Service. Although MOSAICA is oriented to the distribution of single files, the distribution of folders containing multiple files could be possible, making the transfer using HTTP impossible, since the Web Service checks if the content consists only in one file. So, the solution may pass by making the Web Service to check whether the content consists in multiple files and, in that case, it would create an archive using, for instance, the zip format, which could then be normally downloaded, using HTTP.

Finally, and because MOSAICA developed packages are Windows-based, modifications could be made, mainly at the installation process, so this platform could be installed and executed in other platforms, such as Linux-based systems, MacOS or even Solaris, among others prepared to run Java virtual machines.

References

[1] Mosaica innovation memorandum, October 2006. Available in http://www.mosaica-project.eu/fileadmin/MEDIA/PDF_Documents/MOSAICA_Innovation_Memorandum.pdf, last accessed in November 10, 2008.

[2] Definition of mosaica conceptualisation platform, October 2006. Available in http://www.mosaica-project.eu/fileadmin/MEDIA/PDF_Documents/MOSAICA_Conceptualisation_Platform.pdf, last accessed in November 10, 2008.

[3] W3C. W3C Semantic Web Activity. http://www.w3.org/2001/sw/.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of Peer-to-Peer content distribution technologies. *ACM Computing Surveys, Vol. 36, No. 4*, pages 335–371, December 2004.

[5] Keith W. Ross and Dan Rubenstein. Tutorial on P2P systems. *Infocom 04*, 2004.

[6] Clay Shirky. What is P2P... and what isn't?, November 2000. Available in http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html, last accessed in November 17, 2008.

[7] Gregory Block. Third-generation Peer-to-Peer, January 2004. Available in http://www.ctoforaday.com/articles/000023.html, last accessed in November 13, 2008.

[8] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured Peer-to-Peer networks. *International Conference on Supercomputing 2002*, pages 84–95, June 2002.

[9] Yong Yang, Rocky Dunlap, Michael Rexroad, and Brian F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. *IEEE INFOCOM 2006*, August 2006.

[10] Mema Roussopoulos, Mary Baker, David S. H. Rosenthal, TJ Giuli, Petros Maniatis, and Jeff Mogul. 2 P2P or not 2 P2P? 2004.

[11] Brian Dessent. Brian's BitTorrent FAQ and guide, May 2003. Available in http://www.dessent.net/btfaq/, last accessed in September 16, 2008.

[12] Ricardo Salmon, Jimmy Tran, and Abdolreza Abhari. Simulating a file sharing system based on BitTorrent. *SpringSim '08: Proceedings of the 2008 Spring simulation multiconference*, April 2008.

[13] Bram Cohen. Incentives build robustness in BitTorrent. Technical report, May 2003.

[14] Bram Cohen. The BitTorrent Protocol Specification, January 2008. Available in http://www.bittorrent.org/beps/bep_0003.html, last accessed in September 16, 2008.

[15] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. *Proceedings of Passive and Active Measurements (PAM)*, pages 1–11, April 2004.

[16] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and improving BitTorrent performance. Technical report, Carleton University and University of Western Ontario, February 2005.

[17] Daishi Kato and Toshiyuki Kamiya. Evaluating DHT implementations in complex environments by network emulator. *IPTPS '07: Proceedings of the 6th International Workshop on Peer-to-Peer Systems*, February 2007.

[18] Eric Rescorla. Introduction to Distributed Hash Tables, March 2006.

[19] Xiuqi Li and Jie Wu. Searching techniques in Peer-to-Peer networks. *Handbook of Theoretical and Algorithmic Aspects of Ad-Hoc, Sensor, and Peer-to-Peer Networks*, 2006.

[20] Maria Teresa Andrade, Asdrúbal Costa, Francisco Barbosa, and Sílvio Macedo. D3.2 mosaica distributed content management system - services and protocols. November 2008.

[21] Joseph Bih. Service Oriented Architecture (SOA), a new paradigm to implement dynamic e-business solutions, 2006.

[22] Mamdouh Ibrahim, Brenda Michelson, Kerrie Holley, Dave Thomas, Nicolai M. Josuttis, and John deVadoss. The future of SOA: What worked, what didn't, and where is it going from here? Technical report, IBM, Beddara Research Labs, Microsoft, Elemental Links and SOA Consortium, October 2007.

[23] IBM Services Architecture Team. Web Services architecture overview, September 2000. Available in http://www.ibm.com/developerworks/webservices/library/w-ovr/, last accessed in November 24, 2008.

[24] Derek T. Sanders, J.A. Hamilton, and Richard A. MacDonald. Supporting a Service-Oriented Architecture. *Proceedings of the 2008 Spring simulation multiconference*, pages 325–334, 2008.

[25] Michael Stal. Using architectural patterns and blueprints for Service-Oriented Architecture. *IEEE Software, Volume 23*, pages 54–61, March 2006.

[26] Thomas Erl. SOA and XML, 2004. Presentation available in http://www.gca.org/xmlusa/2004/slides/erl/XML%20and%20Service-Oriented%20Architecture%20(SOA).ppt, last accessed in November 27, 2008.

[27] Peter M. Kelly and Andrew L. Wendelborn Paul D. Coddington. A simplified approach to Web Service Development. *Conferences in Research and Pratice in Information Technology, Vol. 54*, pages 79–88, August 2006.

[28] DevelopMentor Web Services Team. Understanding Web Services. November 2002.

[29] Conan C. Albrecht. How clean is the future of SOAP? *Communications of the ACM, Vol. 47*, pages 66–68, February 2004.

[30] Hao He. What is Service-Oriented Architecture. September 2003. Available in http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html, last accessed in November 27, 2008.

[31] Bostjan Kezmah Matjaz B. Juric, Marjan Hericko, Ivan Rozman, and Ivan Vezocnik. Java RMI, RMI tunneling and Web Services comparison and performance analysis. *ACM SIGPLAN Notices, Vol. 39*, pages 58–65, May 2004.

[32] Yi-Hsuan Lu, Yoojin Hong, Jinesh Varia, and Dongwon Lee. Pollock: Automatic generation of virtual Web Services from web sites. *Symposium on Applied Computing 2005*, pages 1650–1655, March 2005.

[33] James Snell. The Web Services insider, part 2: A summary of the W3C Web Services Workshop, April 2001. Available in http://www.ibm.com/developerworks/webservices/library/ws-ref2/, last accessed in December 3, 2008.

[34] Kamalsinh F. Chavda. Anatomy of a Web Service. *Journal of Computing Sciences in Colleges, Vol. 19*, pages 124–134, January 2004.

[35] Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An XML-based approach to combine firewalls and Web Services Security specifications. *ACM Workshop on XML Security*, pages 69–78, October 2003.

[36] Oasis – Advanced Open Standards for the information society. http://www.oasis-open.org/home/index.php.

[37] Web Services Security: SOAP message security 1.0. Technical report, Oasis, March 2004.

[38] Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Andrew D. Gordon. Secure sessions for Web Services. *ACM Transactions on Information and System Security, Vol. 10*, May 2007.

[39] Meiko Jensen, Torben Dziuk, and Nils Gruschka. Event-based application of WS-SecurityPolicy on SOAP messages. *Proceedings of the 2007 ACM workshop on Secure web services*, pages 1–8, November 2007.

[40] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, and Sanjiva Weerawarana. The next step in Web Services. *Communications of the ACM, Vol. 46*, pages 29–34, October 2003.

[41] Paul Prescod. Second generation web services, February 2002. Available in http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html?page=1, last accessed in September 15, 2008.

[42] W3C World Wide Web Consortium. W3C — Web Services Description Language (WSDL) 1.1, March 2001. http://www.w3.org/TR/wsdl.

[43] Aaron Skonnard. Understanding wsdl. *Web Services Technical Articles*, October 2003. Available in http://msdn.microsoft.com/en-us/library/ms996486.aspx, last accessed in December 5, 2008.

[44] Martin Tsenov. Example of communication between distributed network systems using Web Services. *Proceedings of the 2007 international conference on Computer systems and technologies*, 2007.

[45] Richard Monson-Haefel. *J2EE Web Services*. Addison-Wesley, 2003.

[46] Reuven M. Lerner. At the forge: Introducing SOAP. *Linux Journal*, March 2001.

[47] Robert A. van Engelen and Kyle A. Gallivan. The gSOAP toolkit for Web Services and Peer-to-Peer Computing Networks. *2nd IEEE International Symposium on Cluster Computing and the Grid*, pages 1–9, May 2002.

[48] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1. May 2000. Available in http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, last accessed in December 11, 2008.

[49] Brett McLaughlin. Soapbox: Industrial strength or suds? - a closer look at SOAP, RPC, and RMI. May 2001.

[50] Java Remote Method Invocation - Distributed Computing for Java, August 2008. Available in http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp, last accessed in December 15, 2008.

[51] Beyond file sharing., May 2003. Available in http://www.sun.com/2003-0506/feature/, last accessed in September 10, 2008.

[52] Jxta java standard edition v2.5: Programmers guide. September 2007.

[53] Jorge Cardoso. Web services over peer-to-peer infrastructure, July 2001. Available in http://dme.uma.pt/jcardoso/Projects/WS-jxta/index.html, last accessed in September 16, 2008.

[54] JeanMarc Seigneur, Gregory Biegel, and Christian Damsgaard Jensen. P2p with JXTAJava pipes. *PPPJ '03: Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pages 207–212, June 2003.

[55] Gabriel Antoniu, Loïc Cudennec, Mathieu Jan, and Mike Duigou. Performance scalability of the JXTA P2P framework. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, March 2007.

[56] Navaneeth Krishnan. The JXTA solution to P2P, October 2001. Available in http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html?page=1, last accessed in December 29, 2008.

[57] Jiajun Wang, Chuohao Yeo, Vinod Prabhakaran, and Kannan Ramchandran. On the role of helpers in Peer-to-Peer file download system: design, analysis and simulation. *International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2007.

[58] PANDO Networks. The p4p working group. http://www.pandonetworks.com/p4p.

[59] Haiyong Xie, Arvind Krishnamurthy, Avi Silberschatz, and Y. Richard Yang. P4P: Explicit Communications for Cooperative Control Between P2P and Network Providers. 2007.

[60] Top-BT: An infrastructure free BitTorrent client with fast download time and less internet traffic, August 2008. Available in http://www.cse.ohio-state.edu/~sren/topbt/, last accessed in December 23, 2008.

[61] SourceForge. Sourceforge all time top downloads. http://sourceforge.net/top/topalltime.php?type=downloads.

[62] S. W. Eran Chinthaka. Axis2: The next generation of apache web services. July 2006. Available in http://today.java.net/pub/a/today/2006/09/07/axis2-next-generation-web-services.html, last accessed in December 29, 2008.

[63] Wei Ma, Vladimir Tosic, Babak Esfandiari, and Bernard Pagurek. Extending Apache Axis for monitoring of Web Services offerings. *BSN '05: Proceedings of the IEEE EEE05 international workshop on Business services networks*, 2005.

[64] Apache Software Foundation. Apache Axis2/Java — Next Generation Web Services. http://ws.apache.org/axis2/.

[65] Berndt Hamboeck. Apache Axis - The Third Generation SOAP Implementation, July 2002. Available in http://www.sybase.com/content/1020318/axis_wp.pdf, last accessed in December 30, 2008.

[66] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000. Available in http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, last accessed in January 2, 2009.

[67] Roger L. Costello. Building Web Services the REST way. Available in http://www.xfront.com/REST-Web-Services.html, last accessed in January 2, 2009.

[68] Bill Venners. Why PUT and DELETE?, December 2006. Available in http://www.artima.com/lejava/articles/why_put_and_delete.html, last accessed in January 5, 2009.

[69] Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi. WSPDS: Web Services Peer-to-Peer Discovery Service. *Proceedings of the International Conference on Internet Computing*, pages 733–743, 2004.

[70] Jagadeesh Nandigam and Venkat N. Gudivada. WS^{EXP}: a tool for experimenting with web services. *Journal of Computing Sciences in Colleges, Vol. 22*, pages 36–45, October 2006.

[71] Kiran Devaram and Daniel Andresen. SOAP Optimization via parameterized client-side caching. 2004.

[72] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, and Dennis Gannon. Requirements for and evaluation of RMI protocols for scientific computing. *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.

[73] Naresh Apte, Keith Deutsch, and Ravi Jain. Wireless SOAP: Optimizations for mobile wireless web services. *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, 2005.

[74] Markus Baeker. RSS Import 2.2.2. http://azureus.sourceforge.net/plugin_details.php?plugin=RSSImport.

[75] T. Berners-Lee, R. Fielding, and L. Masinter. RFC3986: Uniform Resource Identifier (URI): Generic syntax. http://www.ietf.org/rfc/rfc3986.txt.

[76] Asdrúbal Costa, Sérgio Marques, Maria Teresa Andrade, and Sílvio Macedo. D3.1 MOSAICA Distributed Content Management System - Reference Architecture. *Project IST-034984*, January 2008.

[77] Steve Souza. JAMon - Java Application Monitor. http://jamonapi.sourceforge.net/.