UNIVERSIDADE DA BEIRA INTERIOR
Faculdade de Engenharia
Departamento de Informática

Web Services Approach for Ambient Assisted Living in Mobile Environments

Joel Lourenço Fernandes

Submitted to the University of Beira Interior in candidature for the Degree of Master of Science in Informatics Engineering

Supervised by Prof. Dr. Joel José Puga Coelho Rodrigues

Departamento de Informática
University of Beira Interior
Covilhã, Portugal
http://www.di.ubi.pt

This work has been partially supported by the *Instituto de Telecomunicações*, Next Generation Networks and Applications Group (NetGNA), Portugal, by National Funding from the FCT – *Fundação para a Ciência e Tecnologia* through the Pest-OE/EEI/LA0008/2011, and by the AAL4ALL (Ambient Assisted Living for All), project co-financed by the European Community Fund FEDER through COMPETE – *Programa Operacional Factores de Competitividade*.

UNIÃO EUROPEIA

Fundo Europeu
de Desenvolvimento Regional

Acknowledgements

First of all, I would like to thank Prof. Joel José Puga Coelho Rodrigues for giving me a chance to join the Next Generation Networks and Applications Research Group (NetGNA), for all the constant words of encouragement and for supervising my Master's Thesis. The last year was undoubtedly very important in my evolution has Informatics Engineer as a person as well.

I am most grateful to the University of Beira Interior, the *Instituto de Telecomunicações*, Next Generation Networks and Applications Group (NetGNA), Covilhã Delegation, Portugal, by National Funding from the FCT – *Fundação para a Ciência e a Tecnologia* through the PEst-OE/EEI/LA0008/2011 Project, by the AAL4ALL (Ambient Assisted Living for All), project co-financed by the European Community Fund FEDER through COMPETE – *Programa Operacional Factores de Competitividade*.

Many thanks to all members of NetGNA, particularly Ivo Lopes, Fábio Canelo, Edgar Horta and João Dias.

And of course, a enormous thanks to my parents, my sister, my family and my girlfriend, for their constant support and for all words of motivation.

Abstract

Web services appeared as a promising technology for Web environments independent of technologies, services, and applications. First, a performance comparison study between the two most used Web service architectures, SOAP and REST, is presented, considering messages exchange between clients and a server. Based on this study, the REST architecture was chosen to deploy the system because it gets better results compared to SOAP architecture. Currently, there are some issues related with this approach that should be studied. For instance, if massive quantities of data are sent to databases it can influence significantly the performance of the whole system. The Advanced Message Queuing Protocol (AMPQ) appears as a promising solution to address this problem. Then, in order to evaluate the performance of this approach, this work presents a performance evaluation and a comparison study of RESTful Web services and the AMQP Protocol considering exchanging messages between clients and a server. The study is based on the averaged exchanged messages for a certain period of time. It was observed and concluded that, for large quantities of messages exchange, the best results comes from the Advanced Message Queuing Protocol. Message Queuing Telemetry Transport (MQTT) was addressed in this work because it is a similar protocol to AMQP but it can be used by mobile devices with a processing capacity smallest unlike the AMQP that needs greater processing capacity. These studies are performed in the context of Ambient Assisted Living environments, since

the work was applied to this topic in order to experiment the effectiveness and evaluate the performance of these protocols in this scenario.

Keywords

Web Services, RESTful, Advanced Message Queuing Protocol, AMQP, Message Queuing Telemetry Transport, MQTT, RabbitMQ, Ambient Assisted Living, AAL.

Contents

Acknowledgements ..... **iii**

Abstract ..... **v**

Keywords ..... **vii**

Contents ..... **ix**

List of Figures ..... **xi**

List of Tables ..... **xiii**

Acronyms ..... **xv**

1. Introduction **1**

1.1. Focus and Research Scope **1**

1.2. Problem Definition **9**

1.3. Research Objectives **10**

1.4. Main Contributions **10**

1.5. Dissertation Structure **11**

2. Related Work **13**

2.1. Web Services **13**

2.1.1. Web Services Platform Elements 18

2.2. Advanced Message Queuing Protocol (AMQP) **23**

2.3. Message Queuing Telemetry Transport (MQTT)..... 27

2.4. Ambient Assisted Living (AAL)..... 30

3. Requirements Analysis 35

3.1. Essential Requirements..... 35

3.1.1. Entity-Relationship Model..... 36

3.1.2. Activity Diagrams 37

3.1.3. Sequence Diagrams 40

3.2. System Architecture..... 42

3.3. Used Technologies..... 43

4. Performance Evaluation 45

4.1. REST *versus* SOAP..... 45

4.1.1. Markup Languages..... 50

4.2. Advanced Message Queuing Protocol (AMQP) *versus* Message Queuing Telemetry Transport (MQTT)..... 53

4.3 Systems Demonstration and Validation..... 57

4.3.1 RESTful Web Service, RabbitMQ server and Back-end Service 58

4.4 Results Analysis 61

5. Conclusions and Future Work 67

5.1 Conclusions..... 67

5.2 Future Works 69

References 71

Appendice ..... A

List of Figures

Figure 1. Illustration of Two-tier architecture.....1
Figure 2. Illustration of a three-tier architecture.....2
Figure 3. Basic process of functioning of systems implemented for the AAL..... 7
Figure 4. Service Oriented Architecture (SOA) diagram..... 20
Figure 5. WCF endpoint – ABC (Address, Binding and Contract). 21
Figure 6. Illustration of RESTful Web Services architecture. 22
Figure 7. Illustration of the process of exchanging messages using RabbitMQ... 27
Figure 8. Example of MQTT network topology. 29
Figure 9. Topics, techniques and technologies applied in Ambient Assisted Living. 32
Figure 10. Entity-Relationship Diagram based in IDEF1X notation. 36
Figure 11. Request / send path to the Web services activity diagram.... 38
Figure 12. Request and sender path to the RabbitMQ activity diagram. . 39
Figure 13. Sequence Diagram – Communication between Client (Mobile Application) and Web service to insert or get data in / from database. ... 40
Figure 14. Sequence Diagram – Communication between Client (Mobile Application) and RabbitMQ Server. 41
Figure 15. Sequence Diagram – Communication between Back-end service and RabbitMQ server and database..... 41
Figure 16. Illustration of the global system architecture. 42
Figure 17. RabbitMQ management plugin interface..... 60

Figure 18. Performance comparison considering the number of message for tree different experimental scenarios showing the messages stored in the Database, RabbitMQ,and the average messages per second..... 63

List of Tables

Table 1. HTTP Methods and their corresponding CRUD Action 23
Table 2. Products that support AMQP 24
Table 3. Strengths and Weaknesses for both SOAP and REST..... 46
Table 4. Results of the Performed Experiment to Compare SOAP with REST Web Services. 48
Table 5. Methods implemented in the Web service reference to the HTTP method to use 59
Table 6. Results of the Performed Experiments. 62

Acronyms

AAL	:	Ambient Assisted Living
ACL	:	Access Control List
AMQP	:	Advanced Message Queuing Protocol
API	:	Application Programming Interface
CORBA	:	Common Request Broker Architecture
CPU	:	Central Processing Unit
DCOM	:	Distributed Component Object Model
DLL	:	Dynamic-Link Library
ECG	:	Electrocardiogram
HER	:	Electronic Health Record
HTML	:	Hypertext Markup Language
HTTP	:	Hypertext Transfer Protocol
ICT	:	Information and Communication Technology
ISO	:	International Organization Standardization
JSON	:	JavaScript Object Notation
LDAP	:	Lightweight Directory Access Protocol
MQTT	:	Message Queue Telemetry Transport
QoS	:	Quality of service
REST	:	Representational State Transfer
RFC	:	Request for Comments
RMI	:	Remote Method Invocation
ROA	:	Resource-Oriented Architecture
RPC	:	Remote Procedure Call
SASL	:	Simple Authentication and Security Layer
SMTP	:	Simple Mail Transfer Protocol

SOA	:	Service-Oriented Architecture
SOAP	:	Simple Object Access Protocol
TCP/IP	:	Transmission Control Protocol / Internet Protocol
TLS	:	Transport Layer Security
UDDI	:	Universal Description Discovery and Integration
UML	:	Unified Modeling Language
URC	:	Universal Remote Console
URI	:	Uniform Resource Identifier
W3C	:	World Wide Web Consortium
WADL	:	Web Application Description Language
WCF	:	Windows Communication Foundation
WSDL	:	Web Service Definition Language
XML	:	eXtensible Markup Language
YAML	:	YAML Ain't Markup Language

1. Introduction

1.1. Focus and Research Scope

The idea of distributed computing was born with the appearance of computer networks. Applications were divided first into two main components. One is the client, starting a distributed activity, and the other, the server, carrying out that activity. This reorganisation minimized bottlenecks by distributing the workload through several systems. It offered flexibility for application design previously nearby unknown on centralized hosts. But this two-tier architecture, Figure 1, had its limits [1].

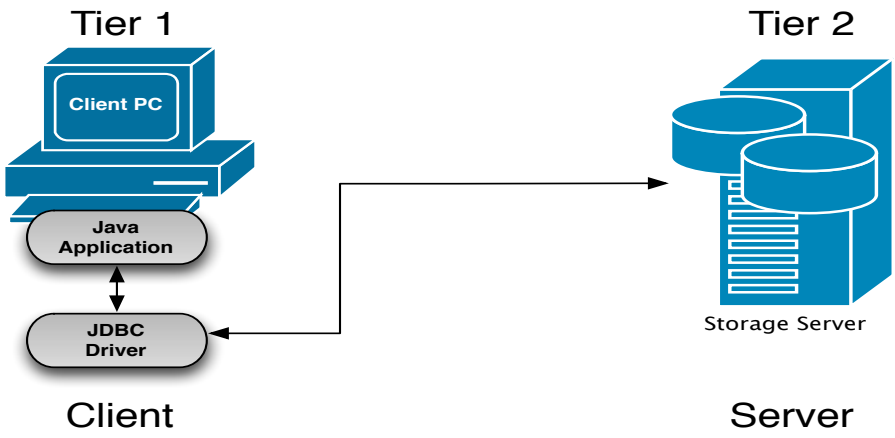


Figure 1. Illustration of Two-tier architecture.

For scalability and failover, issues a third-tier has been announced, dividing an application into a presentation part, an intermediate tier including the business logic, and a third-tier dealing with the data. This three-tier model of distribution, presented in Figure 2, has become the

most popular way of splitting applications. It makes application systems scalable.

The basis for the communication between the distributed parts of an application is the remote procedure call (RPC) [1].

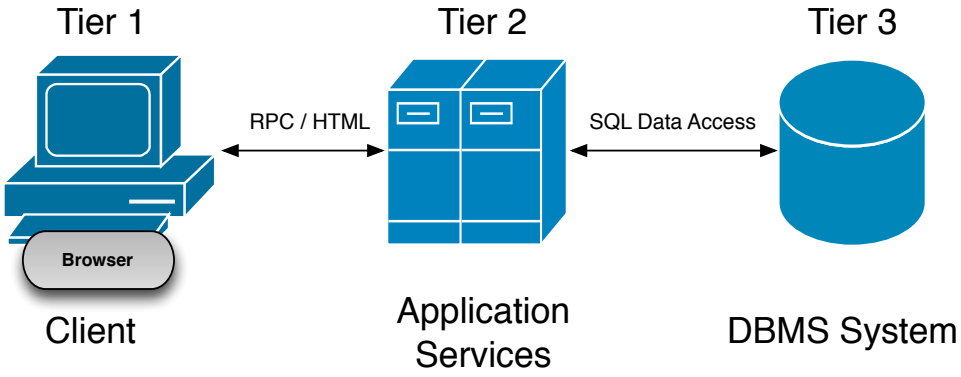


Figure 2. Illustration of a three-tier architecture.

The main use of the World Wide Web is interactive access to documents and applications. In nearly every case, access is by human users, generally working through Web browsers, audio players, or other systems of interactive front-end. The Web can grow greatly in power and scope if it is extended to support communication among applications, from one program to another.

The Web has been a phenomenal success at allows simple computer/human interactions at Internet scale. The original HTML and HTTP protocol stack used by current Web browsers has proven to be a cost-effective way to project user interfaces onto a wide range of devices. The main factor in the success of HTML and HTTP was their relative simplicity — both HTML and HTTP are primarily text-based and can be implemented using a multiplicity of operating systems and programming environments.

In the last few years, ad hoc styles have been used in business-to-business applications to enjoy the benefits of the basic Internet infrastructure [2]. The wondrous growth of the information space and the wondrous number of accessible information sources are factors which arise a growing interest for integrating information sources into Web services in

order to improve collaboration and knowledge sharing between enterprises. The appearance of Web services as a model for incorporating heterogeneous Web information has opened new possibilities of interaction and provided more potential for interoperability.

Currently, more and more Web services are present in applications, serving information or consuming a service. Within the context of distributed applications, the service has become a important, switching the old paradigm of using Dynamic-Link Library (DLL) [3].

The growing number of available Web services, the increase necessity to collaborate and to share knowledge directing to the best decisions are factors which generate a growing interest for meeting Web services into communities in order to facilitate and accelerate Web services discovery and selection. Web services communities provide a centralized access to several functionally-equivalent Web services through a unique endpoint which enables the query processing [4].

For end users, the large growth in available services can involve greater freedom or more chaos or both. For developers, Web services computing provide important theoretical and engineering challenges as they determine how to take advantage of emerging technologies to automate individual applications. The era of globalization as purchased nearly all services to the Internet environment with multiple providers offering the same service in different ways [5]. Several providers are provide the same functionality so it necessary to distinguish one self from the crowd with the aim to prosper against the cutthroat competition from the service provider's view. On the other hand it is absolutely necessary for the client to choose a service provider that not only meet the functional requirements of the client but also offers the best quality of service (QoS) to the customers.

Mobile Internet is driving the exponential growth of mobile devices in a faster way than previous computing technologies. Mobile Internet devices has surpassed ten billion units in 2010 [6]. With the increasing use

of mobile devices, mobile applications will create a larger percentage of Web service requests.

Mobile platforms have their own set of challenges as: bandwidth, memory and CPU availability, storage capacity, connectivity options and issues, security and user interaction and display. Since the capacity of storage and processing on mobile devices is reduced to best solution for mobile applications that generate large amounts of data or need to save too many messages, passes to use other services to send and store messages outside the device. Against this problem, increasingly, the option is to use Web services to send data from the mobile device to a server.

The Web services are the key point of integration for different applications pertaining to different platforms/languages/systems once they are based in a set of standards that make them independent of the underlying technologies used for providing them. There are currently two groups of thinking in the development of Web services: the traditional, standards-based approach (SOAP) and conceptually simpler and the trendier on the block (REST). Spite of the fact that the requirement of mobile computing has improved considerably in recent years [7], [8], using traditional Web services (i.e., SOAP-based services) models to mobile computing may result in unacceptable performance overheads.

Simple Object Access Protocol is extensively used as a messaging framework for Web services realization of Service Oriented Architecture (SOA). This standard is not suitable for mobile devices with resource constraints, provisioning Mobile Web Services, due to large payloads of SOAP message constructs. In addition to memory limitations, the data parsing of the intensive SOAP messages can lead to performance degradation of the mobile Web servers in terms of server utilization. In contrast, the Representational State Transfer (REST) messaging framework is lightweight enough to be appropriate for messaging communication in such devices. A single URL only identifies each service resource in this framework [9].

REST is an architectural style derived from the Web, and its constraints and architectural elements aim at gathering the fundamental design principles that enable the great scalability, growth, and success of the Web. A RESTful service is provided as a resource, which is meaningful concept and can be addressed in the Web. A RESTful service is supplied as a feature, which is the concept extremely useful and can be addressed on the Web. As the Internet grows continuously, new needs are identified, and to confront them, new approaches are emerging. One is the pub-sub (publish-subscribe), wherever the messaging middleware is an operating model with asynchronous and freely combined characteristics. As the message consumers and producers in control flow, space and time are totally dissociated, doing the two endpoints communicate using a distinct statement and registration messages for asynchronous communication, which can better serve dynamic distributed information systems integrations needs and the large-scale. The existing applications of subject based pub-sub were used effectively in the mobile computing, stock, financial and other situations [10].

SOAP-based services are heavy-weighted services, which are not applicable for mobile services in comparison to light weighted RESTful services. Migration of SOAP-based services to RESTful services makes the services more pervasive, faster, and suitable for thin clients. In addition, RESTful services are invoked using HTTP methods [3].

Whatever the technology used to deploy Web services, they provide several advantages [11], like language independence and distribution mechanisms; it also increases the interoperability between different soft-ware elements (for example, it is possible to add communication libraries without modifying existent code), and facilitates code distribution (it is not required the use of a concrete implementation or library) among geographically distributed work teams.

The healthcare industry offers many more situations in which Web services can be put to use effectively. A doctor carrying a handheld device can access your records, health history, and your preferred pharmacy using

a Web service. The doctor can also write you an electronic prescription and send it directly to your preferred pharmacy via another Web service.

The high penetration of mobile devices and networks implies that mobile technology cloud be used very effectively for health promotion in order to compensate the lack of resources problem, particularly in developing countries.

Over the past decade, new application areas in healthcare, such as Ambient Assisted Living (AAL) has improved the interest of community safety engineering. They are reliant upon information gathered from their environment through small, hidden, distributed, and communicating embedded devices for offer their services and are open to the integration of new and heterogeneous devices to optimize offered services. Under these circumstances they must be able to react appropriately to the variation observed in their environment, in the quality of information provided by the embedded devices and in the availability of resources.

The Information and Communication Technologies (ICT) can play a very important role to achieving these goals. ICT can contribute an to a better quality of life for older people contributing to a more healthy and independent life, opposing the difficulties of this age group. This therefore the starting point for the concept of AAL.

From the point of view of AAL, people with special needs are supported through an interaction with the intelligent environment that surrounds them, facilitated by devices capable of measuring several parameters, acting on the environment, with a capacity of computing and communications. So, the passage of the personal computer/device for a range of devices capable of sensing processing and distributed computing environment and by a new paradigm that yields a set of challenges related to the interaction of people with technology in their environment.

The concept AAL refers thus to the study and development of intelligent systems to support the requirements of quality of life, offering a secure conditions in daily environment preferred by user. In AAL are

proposing new approaches in design of products and services that interlink and enhance new technologies in a social environment.

Figure 3 shows the basic process of functioning of systems implemented for the AAL. The sensors / robot send data to the Home Node (gateway). Subsequently the gateway sends this data to a server. Consumers intending to have access to data make requests to the server and it returns the required information.

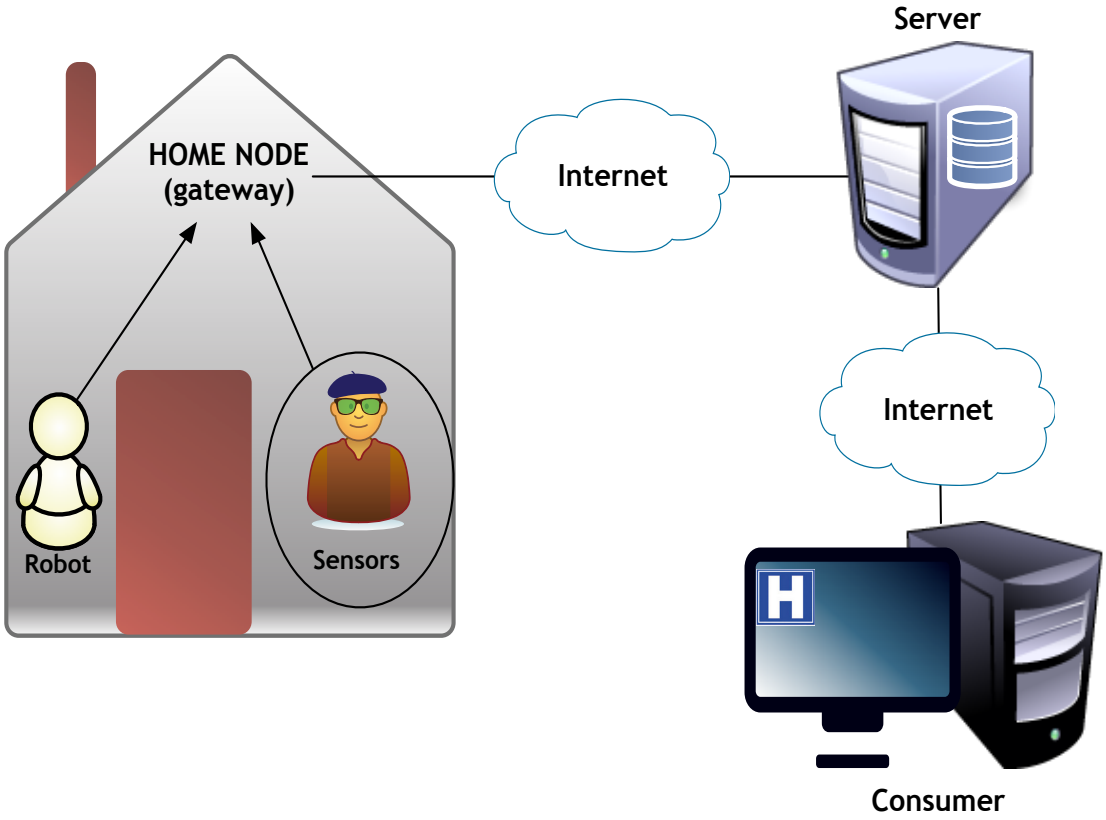


Figure 3. Basic process of functioning of systems implemented for the AAL.

Ambient Assisted Living (AAL) research the development of systems that monitor activities and vital signs of lonely elderly people in order to detect emergency situations or deviations from desirable medical patterns [12]. The discussion about Ambient Assisted Living started when political entities couldn't ignore the fact of the demographic change any longer [13].

The large potential of these AAL technologies related to the capacity to influence people to live in a positive way. In particular, the AAL has influences on the following:

- Support the maintenance of health and functional capacity of older people;
- Improved safety in the physical integrity of the end users, especially those who have illness and/or physical limitations;
- Promoting a healthier lifestyle for people at risk;
- Extension of time that user can live in there preferred environment with safety and proper socialization;
- Contribution to the extension of working lives of people, in that they provide a set of tools and in his own house, ensuring comfort conditions and adequate security and maintaining contact with the entities with whom they interact;
- Increased efficiency and productivity of the resources used in aging societies.

All of these features enable a greater capacity for participation of individuals in society regardless of age, sex, education level or economic origin. The enhancements in the health of users and development of numerous technologies such as, for example, telemedicine from home or use of wearable sensors also contribute to reduce the resources used and therefore the associated costs.

Moreover, the points identified as the weak current AAL solutions in general:

- Solutions are typically high cost;
- These solutions, in many cases, highly technological and complex use which can hinder their use by the target users;
- Solutions based on body monitoring devices, it’s hard to get the trio, comfort, volume and autonomy;

- The difficulty in achieving an objective study on the real needs of end users, leads to the development of solutions that often are not adapted or are not of practical use for these;

For making monitoring activity and vital signs are used devices, such as mobile devices, sensors, etc., with little processing power, low memory and limited battery. Given these problems it is necessary to implement solutions of low energy consumption, with data packets minimized and they can make the efficient distribution of information to one or many receptors. Advanced Message Queuing Protocol (AMQP) e Message Queuing Telemetry Transport (MQTT) are both open protocols for asynchronous message queuing which have been developed and matured over several years. AMQP has selected the OASIS industry standards group, whit the intention of moving to becoming an ISO/IEC standard [14]. MQTT has chosen to use the Eclipse foundation [15].

1.2. Problem Definition

Ambient Assisted Living focuses on the development and market of products and services that can be used for user monitoring and events detection. Thus, it is possible that users live in more secure conditions and in their preferred environment. Constant monitoring by sensors or other devices generates large amounts of data to be transmitted and stored.

A problem that arises in this area is the large amount of data that is generated and needs to be transmitted. In this way it is necessary to study and develop mechanisms and apply systems that allow the exchange of large amount of data to be made as quickly as possible and with no loss of data.

In the present study is carried out the comparison between the two models for transmitting large amounts of data with the aim of find the best solution to solve that problem.

1.3. Research Objectives

The main objective of this dissertation is the design, constructing, deployment, and performance evaluation of Web services when employed in system that generate large amounts of data. To carry out this study, Web services have been applied in mobility systems for the Ambient Assisted Living.

To reach this main objective the following intermediate objectives were define:

- Study of the state of the art, about Web services, Advanced Message Queuing Protocol, Message Queuing Telemetry Transport and Ambient Assisted Living (AAL);
- Detailed analysis of the RESTful Web services and RabbitMQ;
- Requirement analysis of Web services and message queuing protocols considering ambient assisted living (AAL) environments with mobility support;
- Proposal, deployment, and validation of the system based on requirements analysis;
- Performance evaluation of Web services and RabbitMQ considering an AAL system with mobility support.

1.4. Main Contributions

This section is devoted to the scientific contributions of this dissertation to the stat-of-the-art on Web services and Advanced Message Queuing Protocol (AMQP). The main contribution is the Performance Evaluation of RESTful Web Services and AMQP Protocol that presents a performance comparison study of RESTful Web services and the AMQP Protocol considering exchanging messages between client and server. The study is based on the averaged exchanged messages for a period of time.

This study was presented at The Fifth International Conference on Ubiquitous and Future Networks (ICUFN 2013), Da Nang, Vietnam, July 2-5, 2013.

1.5. Dissertation Structure

This dissertation is organized in five chapters. This Chapter, the first, presents the context of the dissertation, focusing on the topic under study, the problem definitions, the objectives, the main contributions and the dissertation structure.

Chapter 2 - Related Work – Presents the literature review on Web services, Advanced Message Queuing Protocol, Message Queuing Telemetry Protocol and Ambient Assisted Living. Referring some of the work done in each of the topics mentioned above.

Chapter 3 – Requirements Analysis – This chapter presents the design and implementation of the system architecture developed. Presents the system architecture, the conceptual design of the system proposed and the technologies used in the development of proposed application.

Chapter 4 – Performance Evaluation – This chapter presents the comparison between the implemented services and equivalents, and between communication protocols implemented. It also focuses on the demonstration of the system and evaluates the performance of two approaches developed that will be integrated into applications AAL.

Chapter 5 – Conclusions and Future Work – Concludes the dissertation and presents a few remarks for future work.

2. Related Work

In order to fully understand the context of the thesis objectives and have an awareness of the state-of-the-art, it is required to approach the related work, as a study of the literature review.

This chapter presents the literature review. In section 2.1 is presented the literature review in Web services with reference to multiple Web service technologies, making a comparison between the two most widely used. Section 2.2 present the literature review in Advanced Message Queuing Protocol focusing interest in the RabbitMQ, used in the development of this thesis. In section 2.3 is presented the literature review in Message Queuing Telemetry Transport protocol can also be used with RabbitMQ. Finally, section 2.4 presents the literature review in Ambient Assisted Living, area chosen to implement Web services and RabbitMQ to evaluate their performance.

2.1. Web Services

The predecessors, such as the Common Request Broker Architecture (CORBA), Remote Method Invocation (RMI) and Distributed Component Object Model (DCOM), Web Services is a set of standards and programming methods for the exchange of data between different software applications, Web services also is a standardized way for distributing services on the Internet [14].

In the last years, Web Services technologies have been used successfully to simplify interoperability, while providing scalability and flexibility in various applications, including distributed simulation software.

The W3C [16] defines a Web service as a method of communication between two electronic devices over a network, it is a solution used in systems integration and communication between different applications. With this technology it is possible that new applications are able to interact with those that already exist and that systems developed across different platforms are compatible. Primarily, the Web service makes the resources of the software application available on the network in a standardized way.

In current collaborative environments, Web Services seems to be a privileged means to interconnect applications across organizations, even if those organizations use different operating systems, different hardware, and different programming languages. Web Services are software systems projected to support interoperable machine-to-machine interaction over a network [17] by using URI (Uniform Resource Identifier) on the distributed environment of internet. They are modular applications with interface descriptions that can be published, located, and invoked across the Web [18].

Web Services they became gradually popular method of managing inter-system communications through networks, with the ubiquity of HTTP, XML, and JSON in modern computing. A Web services-based architecture results in complex system that demands coordination, communications, and distributions among loosely coupled system with dependent users in order to provide scalability, flexibility and extensibility. In its simplest form, Web services refer to software components that support interoperability between computer systems connected over communications network. The key elements of Web Services include, HTML, XML, JSON, SOAP, UDDI, WSDL.

The potential widespread use and benefits of Web Services are very compelling, because they facilitate:

- Automation of business processes distributed across multiple enterprises;
- Collaboration among multiple enterprises by coupling together the business processes running on their various computers.

The Web Service technology has become an industry standard for connecting remote and heterogeneous resources, mobile devices have become a vital part of people's everyday life. People use mobile devices anytime and anywhere, they may use their mobiles to check Email, access the Internet, or run other Web applications.

A Web service is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP [19]. Using a Web service could be as simple as logging into a site or as complex as facilitating a multi-organization business negotiation.

There is a multitude of Web services and applications programming interface (APIs) available on the Web offer a wider variety of various services and generally, has considerable semantic overlap between them, where several Web services deliver basically the selfsame functions. This superimposed feature allows redundancy in the ecosystem of Web services and offers developers the chance to transfer from one API to another while the API initially used is not available or inadequate for their requirements. To help the innovation of similar APIs, many novel methods are being developed to finding the Web services, including query-based methods relying on keywords and identifiers [20], [21], clustering [22] and more detailed structure matching [23], [24].

REST (Representational State Transfer), the architectural style underlying the Web [25], has gained acceptance as an alternative way to develop Web services [26]. This trend is witnessed by several deployed RESTful Web services in the industry, such as Amazon S3 and Facebook API, as well as active researches and studies on REST Web services [27], [28], [29].

RESTful Web Services are earning more and more approaches. They are used as application programming interfaces (APIs) for Web Services 2.9, and are considered a more flexible and lighter-weight alternative to the so-called Big Web Services [30].

RESTful Web Services technical topics become popular because the REST style includes a global identifier of all resources (eg, a uniform resource identifier) and the customer only need to know this handle and the action required. He must also know the right format of representation, which is typically an HTML, eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) meta-data. RESTful Web services (REST APIs) specify a set of resources, which includes three components: the URI of the Web service, the data type supported (JSON, XML, etc.), and the support operations through HTTP methods.

Much research has been done in the field of developing RESTful applications. Richardson and Ruby [26] provide best practice examples and hints on how to develop RESTful applications.

Kopecky et al. [31] present hRESTS as a solution for missing machine-readable Web APIs of RESTful services. They argue that a micro format is the easiest way to enrich existing human-readable HTML documentations. They introduce a model for RESTful services, but with a focus on documentation and discovery.

Alarcón and Wilde [32] introduce a metamodel for descriptions of RESTful services which is the basis for the Resource Linking Language. They focus on service documentation and composition.

Furthermore, Liu *et al.* [33] introduce an approach for reengineering legacy systems to RESTful Web Services. They outline the key issues in this area and propose a solution that covers identification of resource candidates, relation and operation analysis, URI and representation design, as well as service construction. In their opinion, the key problem is to find the right granularity for the resources.

In the domain of model driven development, Laitkorpi *et al.* [28] propose a process for designing RESTful services that focuses on a model based identification of the resources and on generating corresponding WADL descriptions.

Mayer and Lubke [34] also highlight the need to bring formal testing and validation into the area of Web applications, pointing out that many of

the existing tools for BPEL Testing concentrate only on black box testing, to the exclusion of any kind of white box testing.

In [35], they discussed a number of areas in which Web services create additional testing problems. They conclude that Web services need to confront additional problems of scale, such as work flow complexity, volume of data, number of nodes, complexity of operations and differences in usage patterns.

C. Riva et al. [36] investigate how to apply the REST principles to the design of mobile services. They identified several issues such as latency and data format that need particular attention when applying REST concepts to mobile environment. However, they only focused on consuming RESTful Web Services on mobile devices and did not address the provision of Web Services from a mobile host.

Web Services technology recognizes mobile computing as an area to which it should expand. Through integration, Web Services enable pervasive accessibility by allowing for user mobility as it overcomes the physical location constraints of conventional computing. However, mobile computing also requires a technology that connects mobile systems to a conventional distributed computing environment. Web services may be the perfect candidate for such connections, since a strong interoperable capability is the key requirement of the technology. The integrations of mobile computing with Web Services technology will give many advantages to both sides.

Mobile computing and Web Services are becoming popular in collaboration systems, with mobile computing adding heterogeneity to collaboration systems and Web Services providing interoperability. Mobile computing has largely improved in recent years, applying traditional Web Services (i.e. SOAP-based services) models to mobile computing may result in unacceptable performance overheads.

2.1.1. Web Services Platform Elements

Web services are a set of tools that can be used in a number of ways. The three most common styles of use are the Remote Procedure Calls (RPC), Service-Oriented Architecture (SOA) and Representational State Transfer (REST).

RPC – Remote Procedure Calls

RPC Web services present a distributed function (or method) call interface that is familiar to many developers. Typically, the basic unit of RPC Web Services is the WSDL operation.

- **WSDL**

The WSDL (Web Services Description Language) [37] Is a specification developed by W3C that describes the Web Service according to an XML format, working as a contract of service. The WSDL is extensible to allow description of services and their messages regardless of message formats and network protocols that are used. However, it is common to use the MIME (Multipurpose Internet Mail Extensions) and SOAP.

The WSDL describes the service available on the network through a semantic XML, this provides the necessary documentation to be called a distributed system and the procedure required for this communication is established. While SOAP specifies the communication between a client and a server, the WSDL describes the services offers.

A WSDL description contains all the details of a Web Service, including the service's URL, the communication mechanisms it understands, what operation it can perform and the structure of its messages.

- **SOAP**

Simple Object Access Protocol is a protocol for exchanging structured information in a decentralized and distributed platform. It is

based on Extensible Markup Language (XML) to format the messages, and usually based on other application layer protocols, most notably in Remote Procedure Call (RPC) and HTTP for trading and messaging, using GET/POST over HTTP, allowing the data to be exchanged independent where the user is in the network.

SOA – Service-Oriented Architecture

Web Services can also be used to implement architecture according to service-oriented architecture (SOA) concepts, where the basic unit of communication is a message, rather than an operation. This is often referred to as "message-oriented" services.

In the traditional client server world, we had the server offering some functionality that could be used or called by the client. Some kind of look up service acted as a broker between the client and the server.

Since Web Service represent just another paradigm for distributed applications, they consist of the same three components [1]:

- A service broker that acts as a look up service between a service provider and a service requestor. This is a logically centralized directory of services.
- A service provider that publishes its services to service broker. The service provider implements the service and makes it available on the Internet.
- A service requester that asks the service broker where to find a suitable service provider and that binds itself to the provider. This is any consumer of the Web Service. The requester utilizes an existing Web Service by opening a network connection and sending any request.

Each Web Service has an API that can be accessed over the network and executes the Web service at host location. Every service provides a role, such as service provider, a requester or a broker. In other words, Web

Services make possible the effective processing or machine readable information.

The Figure 4 illustrates the relationships between the Web Services components.

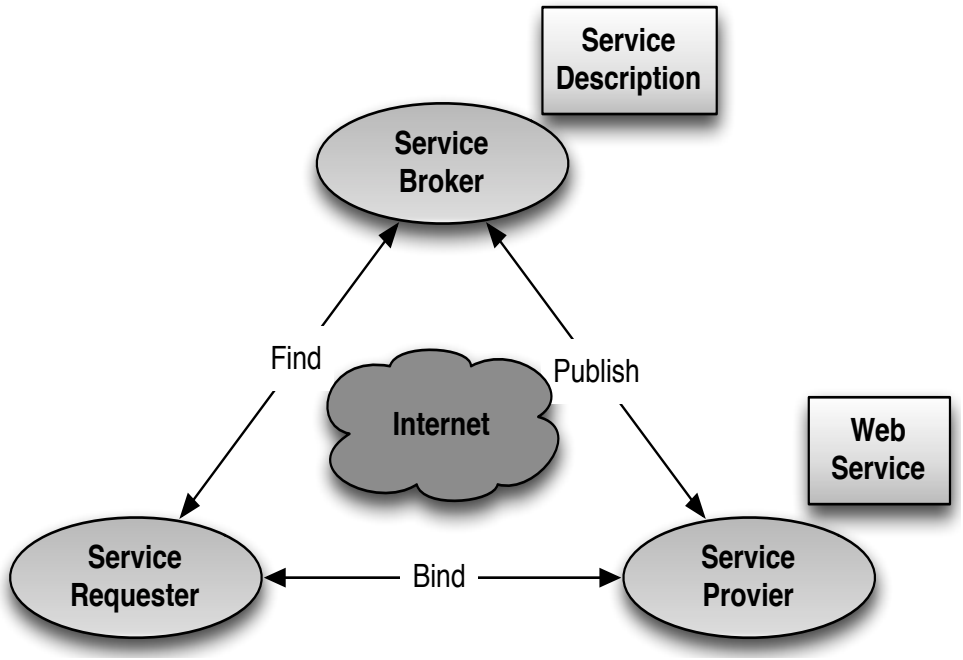


Figure 4. Service Oriented Architecture (SOA) diagram.

- **WCF**

Windows Communication Foundation (WFC) [38] is the Microsoft platform available since –NET Framework 3.0, used to handle the communication between systems. WCF is a union of a Web Service, and Remoting, all in a single platform, simple to use, robust and easy integration.

WCF is designed in accordance with service oriented architecture principles to support distributed computing where consumers consume services. A WCF client connects to a WCF service via an EndPoint. Each service exposes its contract via one or more endpoints. An endpoint has an address with a URL specifying where the endpoint can be accessed, and binding properties that specify how data will be transferred. An endpoint (Figure 5) contains three main features called ABC (Address, Binding and

Contract). An analogy for these features is that the Address is where the service is hosted, Binding is how the service can be accessed, and the Contract would be seen in service.

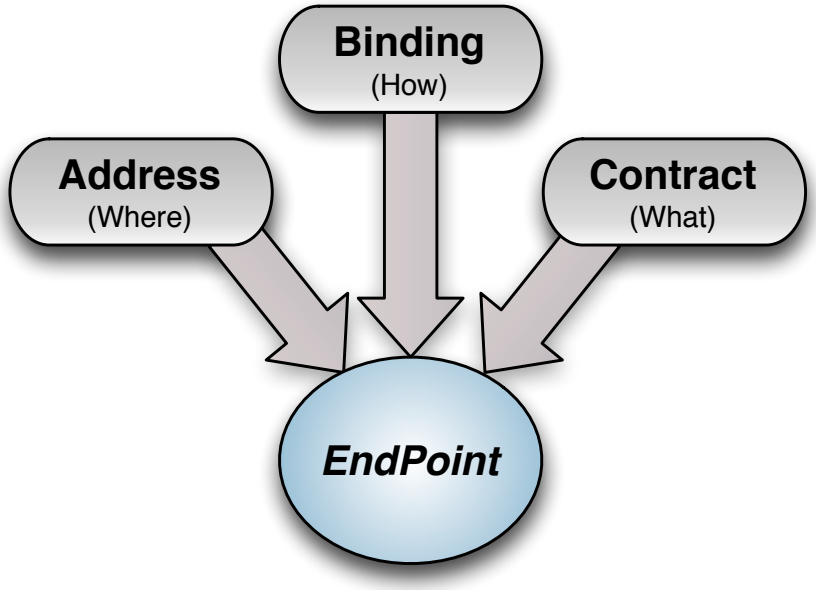


Figure 5. WCF endpoint – ABC (Address, Binding and Contract).

REST – Representational State Transfer

Roy Fielding introduced the term REST in his PhD dissertation [25], where he referred to "hypermedia as the engine of application state". This means that a resource is expected to contain hyperlinks. These hyperlinks are the method by which a transition can take place that changes the resource state or transfers to another resource. While hyperlinks are commonplace in (X)HTML applications meant to be used by humans, they have not typically been used in XML, which is meant to be consumed by machines. Like (X)HTML, REST Web Services make use of hyperlinks in XML.

To understand REST, it is necessary to understand the definition of resource, representation and state. A resource can be anything, may be a physical object or an abstract concept. Usually a resource is something that can be stored on a computer and represented as a stream of bits. A representation is any useful information about the state of a resource. A resource may have multiple different representations. In REST there are

two types of state. One is resource state, which is information about a resource, and the other is application state, which is information about the path the client has taken through the application. Resource state stays on the server and application state only lives on the client. Rest provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.[25]

In REST, the server is abstracted into a set of resources. A resource is a nominal concept, so the modeling based on REST is noun-centralized, which is a domain model [39]. Figure 6 shows the architecture of RESTful Web Services. Client communicates with server via a uniform interface and during the stateless communication; client and server swap features depictions. Therefore, the REST design restrictions supply a standardized method to develop an API wearing the HTTP protocol.

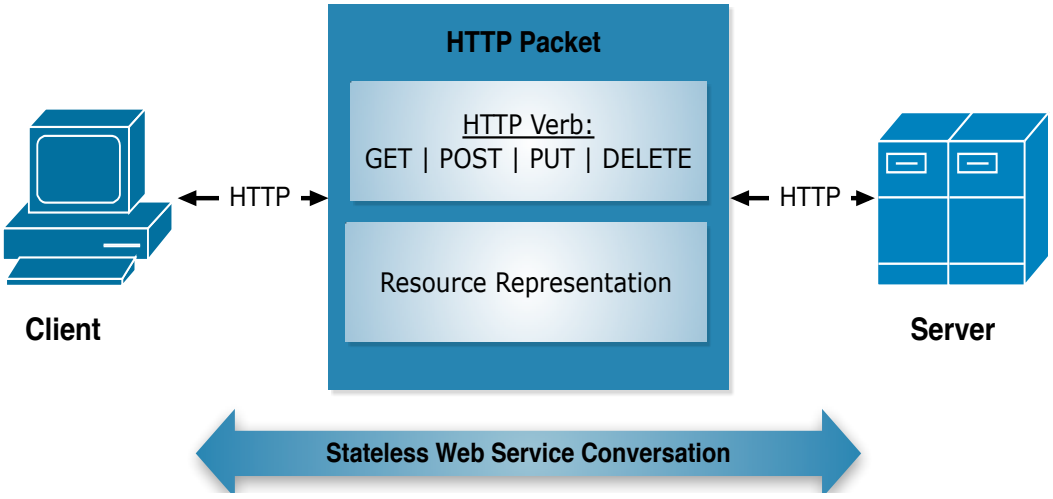


Figure 6. Illustration of RESTful Web Services architecture.

Old-style Web applications access methods using HTTP operations (GET and/or POST). Contrasting with this, applications RESTful access methods according to the following functions: create, read, update, and delete (CRUD) style using the full range of HTTP methods (GET, POST, PUT

and DELETE). Table 1 shows a mapping of HTTP methods to these CRUD actions.

Thus, the RESTful design constraints provide a standardized way to build an API using the HTTP protocol. This design includes correct use of standard HTTP methods and return codes.

Table 1. HTTP Methods and their corresponding CRUD Action

HTTP Method	CRUD Action
GET	Retrieve a resource
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

RESTful Web Services can be depicted over the Web Applications Description Language (WADL). A WADL file sets out the requirements that can be legally directed to a service involving the service uniform resource identifier (URI) and service data waiting and serves.

REST architecture is an architecture style for designing networked applications [40]. Grounded in distributed hypermedia systems and software engineering.

REST and Resource-Oriented Architecture (ROA) support a wide range of representation formats, including plain text, HTML, XML, and JavaScript Object Notation (JSON).

2.2. Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol (AMQP) is a open standard middleware message. Accordance with the AMQP standard, middleware products developing for various platforms and in various languages can submit messages from one to another. AMQP is borne by a number of

important companies, including JPMorgan Chase Bank, Cisco Systems, Credit Suisse, Deutsche Börse Systems and Red Hat [41].

Advanced Message Queue Protocol (AMQP) originated in the financial services industry in 2006 [41] [42] [43]. AMQP is an open standard for Message Oriented Middleware (MOM) communication. AMQP grew out of the need for MOM system integration both within and across corporate enterprise boundaries.

Although AMQP specification is not finalized yet, several products supporting different drafts of AMQP exist today (Table 2), as the Red Hat, VMware, Ltd and OW2 Consortium who use the 0-9-1 version of AMQP, the Apache Software Foundation and the Stormmq who use the 0-10 version. It is used to simplify critical tasks, for example, JPMorgan reported a AMQP environment support 2,000 users on five continents to process 300 million messages per day. Every products that are listed comes with client library for different programming language, such, C++, Ruby, Java, and Python [44].

Table 2. Products that support AMQP

Developer	Product	AMQP version
VMware, Ltd	RabbitMQ	0-9-1
Red Hat	Red Hat Enterprise MRG	n/a
OW2 Consortium	JORAM	0-9-1
Apache Software Foundaton	Apache Qpid™	0-10
Stormmq	stormmq	0-10

The primary reasons for choose the AMQP as proprietary alternatives are the following realize the savings commoditization brings, connect applications on different platforms, connect to business partners using a full featured open standard and position for innovations built upon the foundations of AMQP.

AMQP enables applications to send and receive messages. In this regard it works like instant messaging or eMail. AMQP differs enormously from other available solutions comes from the fact that it allows the

specification of what messages can be received and from, and how trade-offs are performed with respect to security, reliability, and performance. Systems built to integrate AMQP perform much better at functioning unattended or "lights-out" than other solutions.

Systems built to integrate with AMQP are much better at functioning unattended, or "lights-out", than other solutions.

AMQP is different from other middleware standards because it is:

- **Interoperable** – All AMQP clients interoperate with all AMQP servers. Diverse programming languages can communicate easily. Legacy message brokers can be retrofitted to remove proprietary protocols from your network. Messaging can be enabled as a cloud service.
- **Reliable** – Capable of eliminating the communication gaps and slowdowns between different platforms, critical systems and applications components both within enterprise and out to external systems and organizations.
- **Unified** – Provide a core set of messaging patterns via a single manageable protocol.
- **Complete** – AMQP provides a wire level transport for applications using that API. AMQP is broadly applicable and can be leveraged by any language, and identifies both store-and-forward and publish-and-subscribe semantics in on specification.
- **Open** – Vender and platform agnostic, and created by users and technology provides working in collaboration.
- **Safe** – A secure solution to the problem of transporting value-bearing messages across and between organizations, technology platforms and a virtual cloud computing environment.

I will show the performance of RabbitMQ that implements the Advanced Message Queuing Protocol standard. RabbitMQ is an open source message broker and queuing server that can be used to let disparate

applications share data via a common protocol, or to simply queue jobs for processing by distributed workers.

RabbitMQ server is written in Erlang and is built on the Open Telecom Platform framework for clustering and failover. The RabbitMQ project consists of:

- The RabbitMQ exchange server itself;
- Gateways for HTTP, STOMP, and MQTT protocols;
- AMQP client libraries for Java, .NET Framework, and Erlang;
- A plug-in platform for custom additions, with a pre-defined collection of supported plug-ins.

The main characteristics of RabbitMQ project include the following: the RabbitMQ interchange server; gateways for protocols (HTTP, STOMP, and MQTT); AMQP client libraries for .NET Framework, Java, and Erlang; and a plug-in platform for personalized add-ons, with a pre-stated collection of sustained plugins.

AMQP defines an efficient and flexible publish/subscribe interface that is independent of the data model. AMQP uses a central server, or broker, but the RabbitMQ implementation can be scaled linearly by distributing the broker over multiple physical nodes [45].

With the creation of a group of publishers and subscribers can access the nodes of messages, is possible to create an information network that can span from small to very large scale in a local area, or over the large geographical distance. The distribution of information sent from the publishers to the hub to be distributed to the necessary subscribers allows for applications to run while relying on data from other locations, wherever they may be. This allows RabbitMQ to be useful in designing architectures for small-localized systems to large, geographically dispersed interactive systems.

Figure 7 presents the basic process of exchanging messages using RabbitMQ.

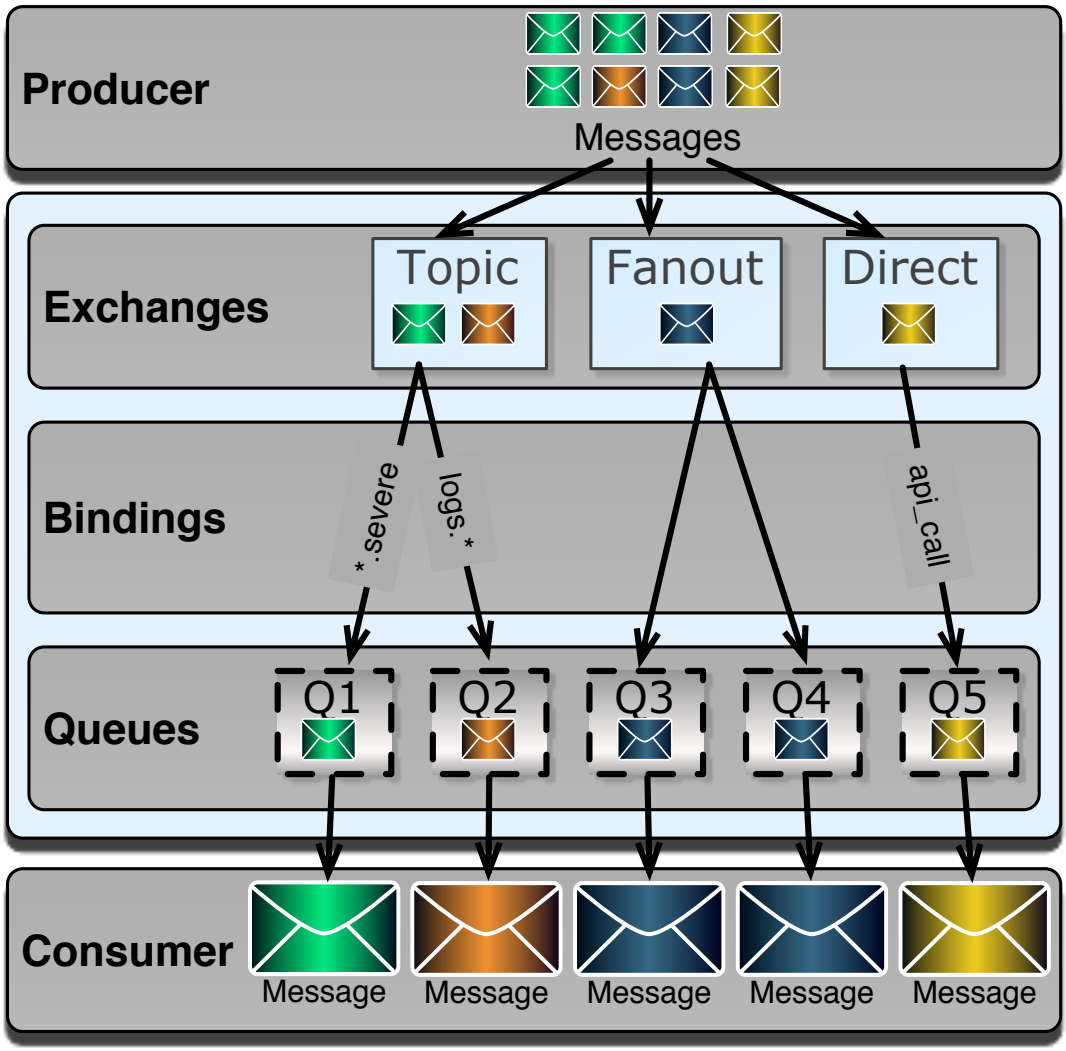


Figure 7. Illustration of the process of exchanging messages using RabbitMQ.

2.3. Message Queuing Telemetry Transport (MQTT)

Message Queue Telemetry Transport (MQTT) is a publish/subscribe messaging protocol created in 1999. The idea of MQTT is to is to be extremely simple and lightweight. It is designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The aim of MQTT is to minimize network bandwidth and device resource requirements while also attempting to ensure some level of reliability. MQTT is intended for "machine-to- machine" (M2M) or "Internet of Things" world of connected

devices and for mobile applications, where bandwidth and battery power are at a premium [15].

MQTT is one the protocols supported by the IBM WebSphere Message Broker products as a way of getting data in and put of the broker. The protocol was designed specifically for remote telemetry applications, with three specific design goals [46]:

1. It should offer a once-and-once-only assured delivery mode to enable a message to be reliably transferred ail the way from a remote sensor to a back-end applications;
2. The protocol should be as lightweight as possible across the "wire"; most remote telemetry is done over low bandwidth, high cost network, and so minimising the overhead of each message is highly desirable;
3. The protocol should be very easy to implement on embedded devices such as sensors and gateways.

The MQTT specification is openly published with a royalty-free license. There is no need to pay royalties to any authority in order implement and use the specification. Therefore, different companies have implemented the principles described in MQTT specification in various software products. Applications of MQTT differ from smart energy meters or smartphones to cars, trains or personal health devices and other remote sensors. They are connected to the central systems with the help of MQTT, which it is often used for these purposes. The central systems process data and emit control commands, which are sent back to the sensors over MQTT as well [47].

Since MQTT follows the topic-based publish/subscribe paradigm, the server decouples publishers and subscribers along the following three dimensions [48]:

- Space decoupling: sender of the message does not have to know the location of the receiver and even the number of receivers and their identifiers.

- Time decoupling: the participants do not have to interact at one and the same time. The publisher can publish events before the subscribers are connected and a subscriber may receive the message even after the publisher is disconnected.
- Synchronization decoupling: publishers are not blocked while producing events and subscribers can be asynchronously notified while performing other concurrent operations.

Figure 8 presents an example of possible MQTT network topology. There is a client, which acts as a publisher on the left hand side. When a message with a defined topic is published, it is delivered to the server first. The server forwards the message upon receiving to three subscribers on the right hand side. It is assumed in this case, that all three subscribers are subscribed to the topic of the particular message. Any client can act as a publisher and subscriber at the same time.

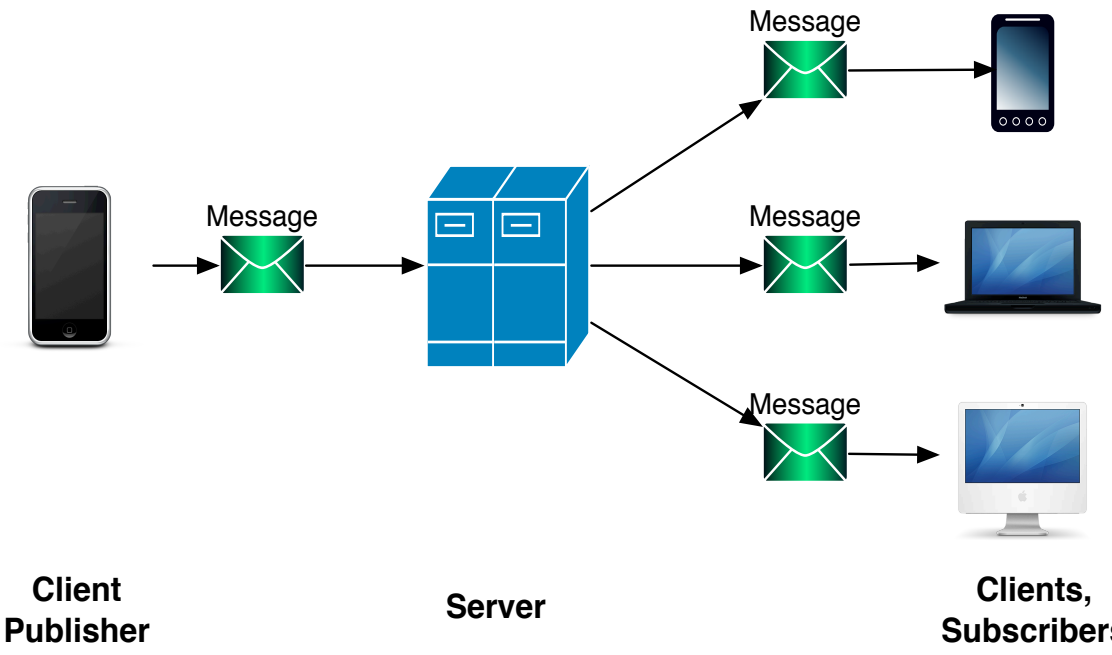


Figure 8. Example of MQTT network topology.

The MQTT protocol has an open, published specification, which is available for anyone to implement on a client device, and reference implementations are available from IBM in Java[49] and C[50].

2.4. Ambient Assisted Living (AAL)

The recent process of demographic change, exposed in an aging world population is reflected in the inversion of population pyramids, has brought new challenges to which families, governments and society in general are not prepared. Recent European projections [51] show a significant increase in the rate of aging.

This scenario offers new chances for innovation, strengthening research in scientific areas, the development of new approaches and business opportunities for technology companies, particularly those that already operate in the markets for products and services intended for elderly populations with special needs.

It is indispensable to help aging with health, autonomy and independence, allowing the aging to permanency active in the labour market by the greatest quantity of time. For companies raises the question of thinking elderly over the life, an approach more preventive and health-promoting and guaranteeing the autonomy and quality of life of populations [52]. A strong investment in new research activities, development of products and services which guarantee and improve the quality of living conditions at the level of health, safety, entertainment and communication is now a necessity.

The concept of Ambient Assisted Living (AAL) is a change of paradigm in the interaction person-computer. The transition of the personal computer/device for a variety of devices capable of sensing, processing and distributed computing environment and by a new paradigm that produces a set of challenges related to the interaction of people with technology in their environment.

The concept of AAL and often perceived as [53]:

- An extension of time that the user can live in their preferred environment through the improvement of autonomy, self-confidence and mobility;
- Support the maintenance of the health status and functional capacity of older persons;
- Promoting a healthy lifestyle for people at risk;
- Increased efficiency and productivity of the resources used in aging societies.

AAL initiative promotes the use of technologies for helping elderly people to maintain their autonomy, increasing their quality of life and facilitating their daily activities, but bearing in mind that it is crucial serving users in terms of usability. It is important to consider that an important number of elderly people present disorders of memory, orientation and cognition. Cases in which these disorders are severe need holistic attention by caregivers; however, slight cases can achieve a personal autonomy adapting technologies to the performance of their daily activities and needs.

In this case, to offer to elderly people new ways for getting natural and implicit services by using pervasive mechanisms we consider that the integration of new technologies in these environments conveniently is the key to improve the life quality of our independent or autonomous elders. To achieve a smart environment, the deployed services must be adapted and context-dependents.

AAL solutions intend to deploy technology approaches to assist people with some kind of disease and older people to living independently in their daily life [54]. ALL products and services have a huge potential not only to enhance the independence and quality of life of elderly population and patients, but also to greatly reduce the costs associated with health care services [53]. Numerous targets have been proposed by the major AAL stakeholders [55], [56].

Figure 9 shows the main topics related to Assisted Living Environment, and their utilities, as well as technologies and techniques that can be implemented to achieve the objectives of the Ambient Assisted Living.

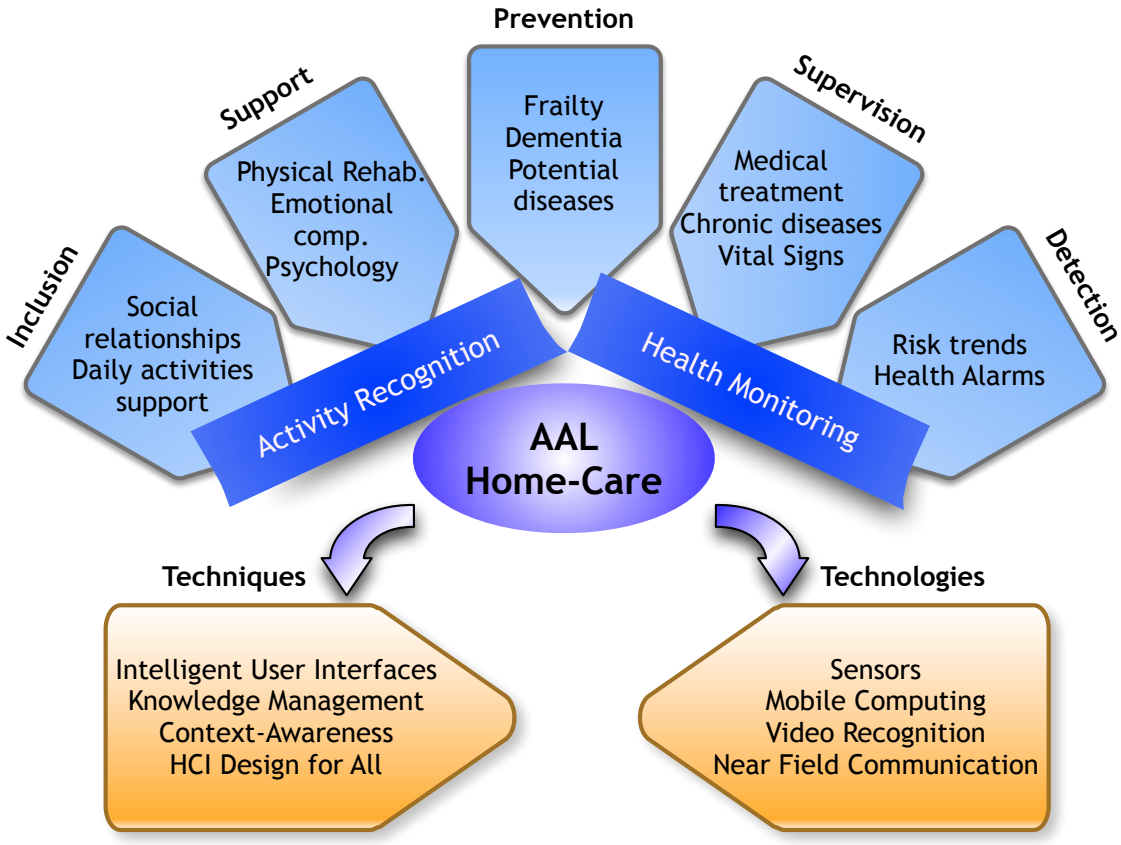


Figure 9. Topics, techniques and technologies applied in Ambient Assisted Living.

In terms of prevention, AAL systems can be considered for different situations, such as falls, physical immobility, monitoring of activities of daily living, occupying spaces at home, behaviour analysis, and other possibilities [57], [58], [59], [60] . All improvements on each of these scenarios are an important step towards the development of more effective and secure solutions enabling the further development of new mechanisms, products and even services [61]. However, to achieve these objectives, it is necessary to take into account developments of several distinct areas. On the one hand, mobile devices, including sensors, will be of great

importance to provide the ability to realize the environments they require. In terms of network devices, it is necessary that they are able to accommodate devices that support and dynamic way communication between them, regardless of their characteristics or features in an integrated manner. Another area of great importance is the ubiquitous computing, where it is expected that mobile devices may evolve towards greater autonomy and mobility, making it easier and more comfortable to carry. In terms of human-machine interaction, will be fundamental to the continued development of natural interfaces to humans, such as speech, gestures or even thoughts that allow minimal interaction, and possibly without the awareness of their interaction by the user. Finally, the area of Artificial Intelligence will contribute in various levels. Specifically, there is interest here in adaptation tools and learning environments that can provide them with the ability to learn the routines and preferences of the user in a manner not invasive in order to adapt their actions.

Based on recent developments in technology (especially in terms of mobile solutions) and given the increased rate of aging of the population and their needs, several solutions are emerging for AAL. Currently, it represents one of the most important topics of research and development, at both national and international level [62], [63], [64]. In this type of solutions the accessibility, usability, and learning process play an extremely important role [65].

Several relevant statistical information, are presented and discussed in [53]. It analyses the market size of the current AAL and eHealth in Europe. According to Silva et al. [66], it is possible to understand the impact of specific contributions of Information and Communication Technology (ICT) in the health sector, such as electronic health records (EHR) and ePrescribing systems. This report concludes the scale of returns on the socioeconomics and return on investment, concluding that EHRs and ePrescribing investments are beneficial to improving health. Several case studies are also presented and discussed.

Recently, Hristova et al. [67] presented a prototype system including a set of health services, such as monitoring heart rate, prescription medications, generating a schedule reminders, and emergency notifications. Flynn et al. [68] developed a wireless bio monitor, which integrates wearable sensors, blood volume pulse and electrocardiogram (ECG). On the other hand Stelis [69] has developed a system capable of providing location data of people associated with tools and alarm notifications.

In [70] the authors intends to develop a falls detector to be used in residences for seniors. The system uses an infrared sensor to monitor a low resolution for division. When an individual who falls into this division in the system automatically activates a process aid by sending a message to a caregiver. The Peach project [71] explores how health professionals can treat ulcers in the feet remotely through the use of images and video. With the use of ICT professional clinics can follow treatment progress despite the distance. In the framework of the project we developed a platform that allows the sharing of images and video.

The i2Home [72], based on industry standards and already existing development. Focuses on domestic use of devices (including consumer electronica) by people with cognitive difficulties and older people. I2Home simultaneously takes into account that the access strategies, developed and standardized, will apply to areas beyond the home. For that developed a standard, ISO 24752 (Universal Remote Console - URC), allows for easy integration of any user interface to any device or service.

The project Bedmond [73] aims to develop a system for continuous monitoring of the activities of day-to-day life of elderly thus allowing early detection of diseases neurodegenerative. The data collected will be used for an early diagnosis and for monitoring disease progression. This project has as one of its central blocks the ISO 24752 standard.

3. Requirements Analysis

This chapter presents the design and implementation of the system architecture developed. Section 3.1 presents the conceptual design of the system proposed and then presenting UML diagrams of the main actions and procedures. Section 3.2 presents the system architecture. Finally, section 3.3 presents the technologies used in the development of proposed application.

The analysis of requirements is an essential process for the development of a product, because is in the analysis of requirements that is define precisely the objectives to develop. When it comes to developing software, it is of extreme importance for all the analysis requirements will constrain both the software as its final operation.

3.1. Essential Requirements

In this section I will present the entity-relationship model and some UML diagrams, including activity and sequence diagrams.

The Unified Modeling Language (UML) is a graphical language for visualizing specifying, constructing, and documenting the artefacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

3.1.1. Entity-Relationship Model

The entity-relationship model (ER model) is a way to unify the network and relational database views. An ER model is an abstract way to describe a database proposed in Peter Chen's 1976 paper [74]. Diagram created to design these entities and relationships are called entity-relationship diagrams or ER diagrams. These models are used to describe information needs or the type of information that is to be stored in a database.

Figure 10 show the entity-relationship diagram based in IDEFIX [75] notation of the database developed and implemented in MySQL for testing the Web Services and the Advanced Message Queuing Protocol.

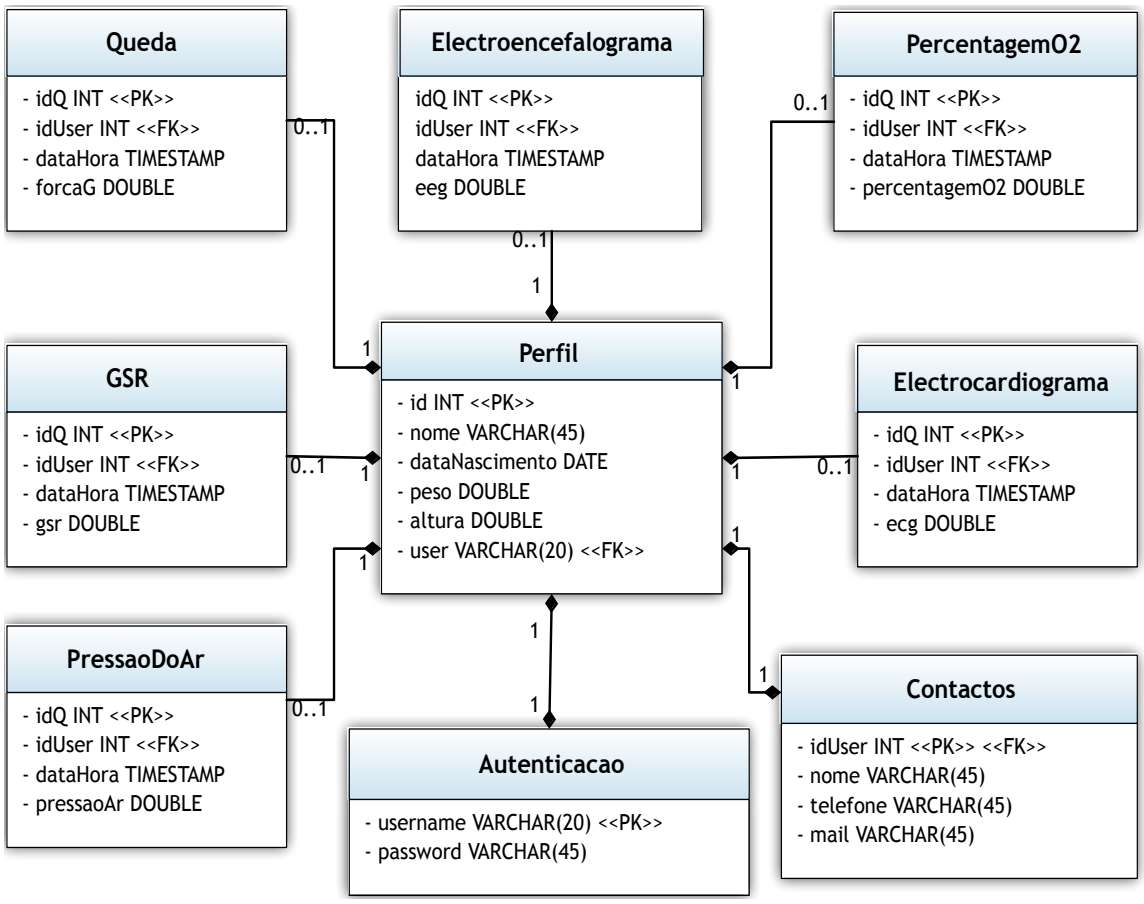


Figure 10. Entity-Relationship Diagram based in IDEF1X notation.

3.1.2. Activity Diagrams

The activity diagrams are used to describe the business operational components of a system step-by-step, and the overall flow of control.

Currently, Web services are gradually included in applications, and consume information, regardless of user location. Web Services triumph the goal with a technologically neutral way, which provides well-defined interfaces for dispersed resources, which are not dependent of the operating system, hardware platform and programming languages. Then dispersed resources or services that can be run on distinctive hardware platforms, on various operating systems, or even can be written in various programming languages communicating using Web services interfaces.

Web services are not the most suitable solution in case there is the exchange of huge amounts of messages daily because if the server has little processing power, the customer is waiting the response from the Web service to send the next message, thus existing the possibility of lost data during runtime of the application. Given this problem, it becomes necessary to use another service, in this case the RabbitMQ, which allows you to store messages in a cloud environment, and only subsequently or simultaneously through a back-end service go get the messages, analyze them and store the that interest.

In this project have been implemented the two cases, the Web service and RabbitMQ server. The activity diagrams presented in the following pictures show the operational components for each of the cases explaining each step-by-step operation of systems and how they are controlled.

Figure 11 describes a request workflow and sends processes through Web services. When a node needs to send or receive information, the requests go directly to the database. The requester or sender makes the solicitation in REST, the server will unfold the request to whether it is a request for data or to data store, and if it was a request for data, the

server sends the data directly from the database, otherwise if it was to store data, the server will save the data directly in the database.

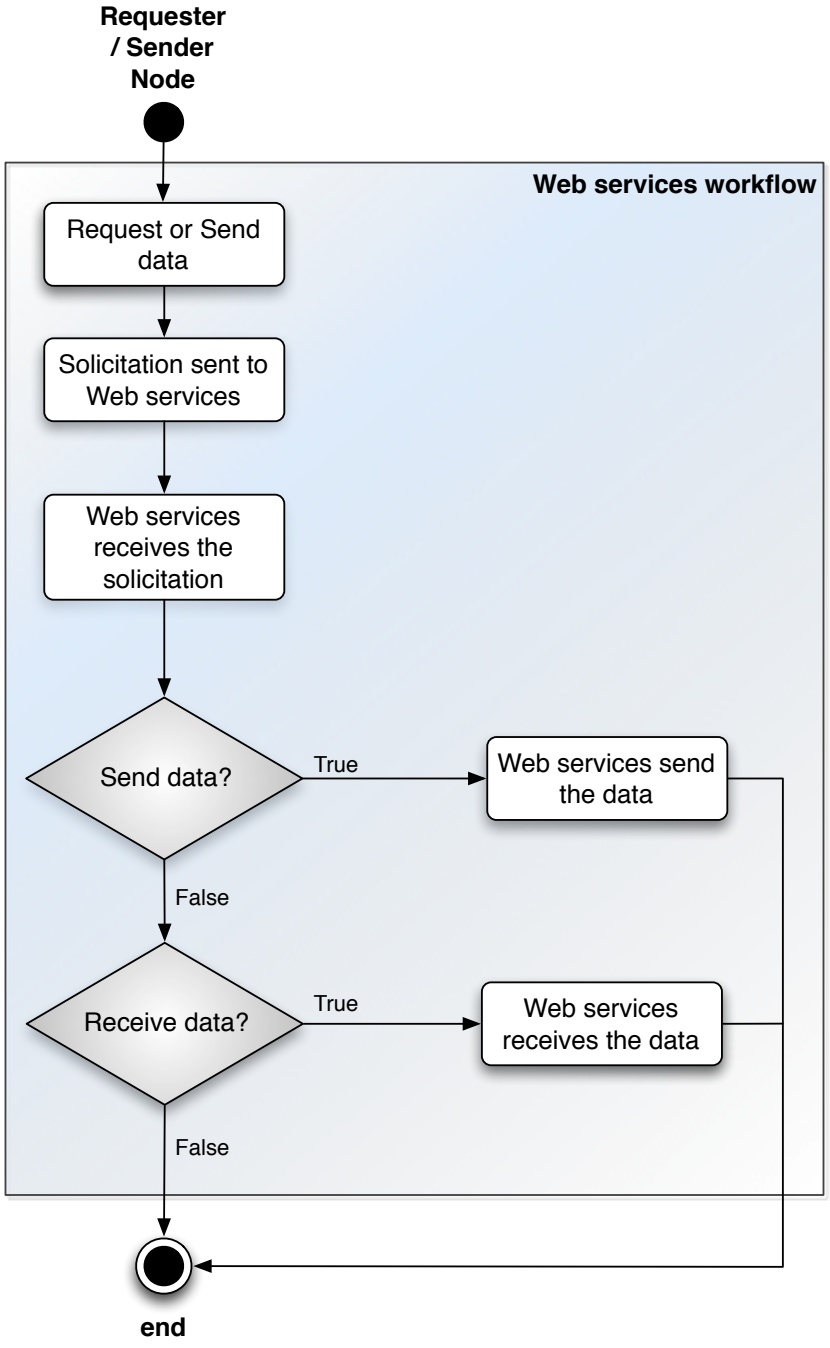


Figure 11. Request / send path to the Web services activity diagram.

Figure 12 shows the activity diagram of the RabbitMQ request and data sending. All the solicitations should go first through the RabbitMQ before reach the database either sending or receiving. When a node sends

information the RabbitMQ will put the message in a stack, then, when a request is made it will go directly to the stack. Messages will be stored in the RabbitMQ until they are consumed.

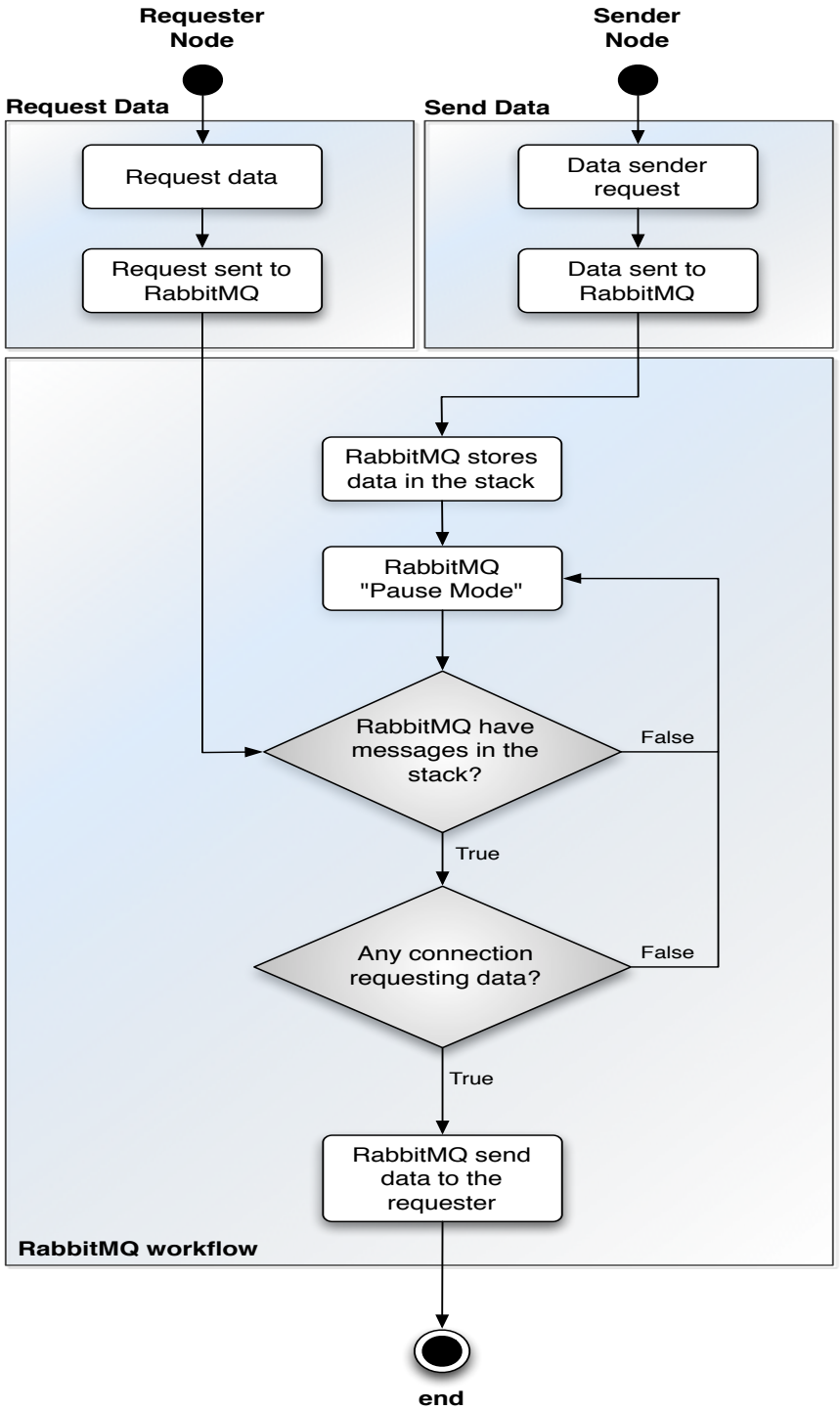


Figure 12. Request and sender path to the RabbitMQ activity diagram.

3.1.3. Sequence Diagrams

The sequence diagram shows how processes interact, every sequence diagram is important, but some have more importance those others.

Figure 13 shows the communication process between the client, the Web service, and the database. Initially, the client, a mobile application, makes a request to the Web service, i.e., calls one of the methods implemented, which can be a method of inserting or querying data. Thereafter, the Web service processes the received data and sends them to the database where they will be stored. Finally the database sends a response to the Web service, and this to the client, in order to the client knows if your request was successfully or unsuccessful performed.

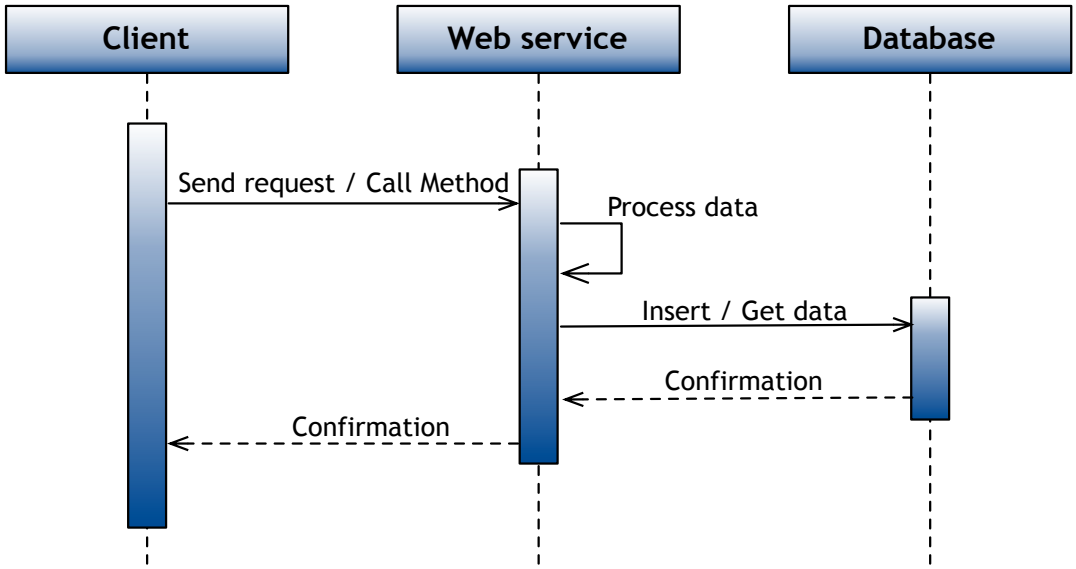


Figure 13. Sequence Diagram – Communication between Client (Mobile Application) and Web service to insert or get data in / from database.

Figure 14 presents the communication process between the client, a mobile application, and RabbitMQ server. In this process the client sends data into the RabbitMQ, which subsequently confirms to the client that the data was received or not. May be observed that using the RabbitMQ server, the client only has the possibility to send data being the query carried out merely using the Web service.

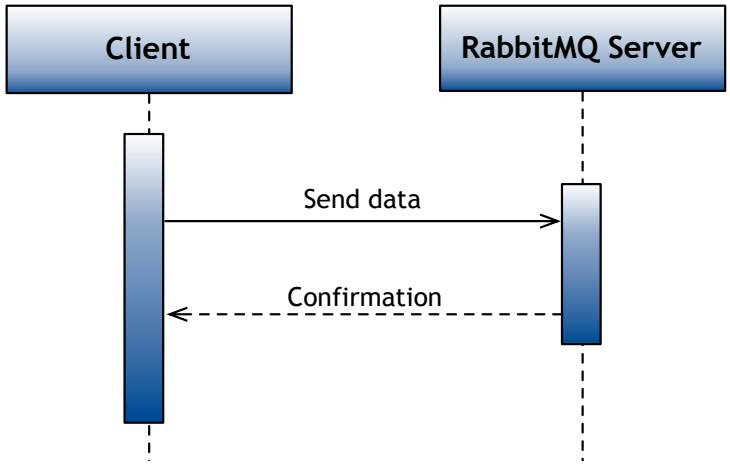


Figure 14. Sequence Diagram – Communication between Client (Mobile Application) and RabbitMQ Server.

In Figure 15 is presented the process of communication between the back-end service, the RabbitMQ server and the database. Initially, the backend service makes a request to RabbitMQ with the aim of receiving the data that are stored in the server queue. The RabbitMQ responds by sending to service an array with all the contents of the queues. Subsequently, the back-end service will insert the data in the database waiting for the answer about the success or failure of insertion.

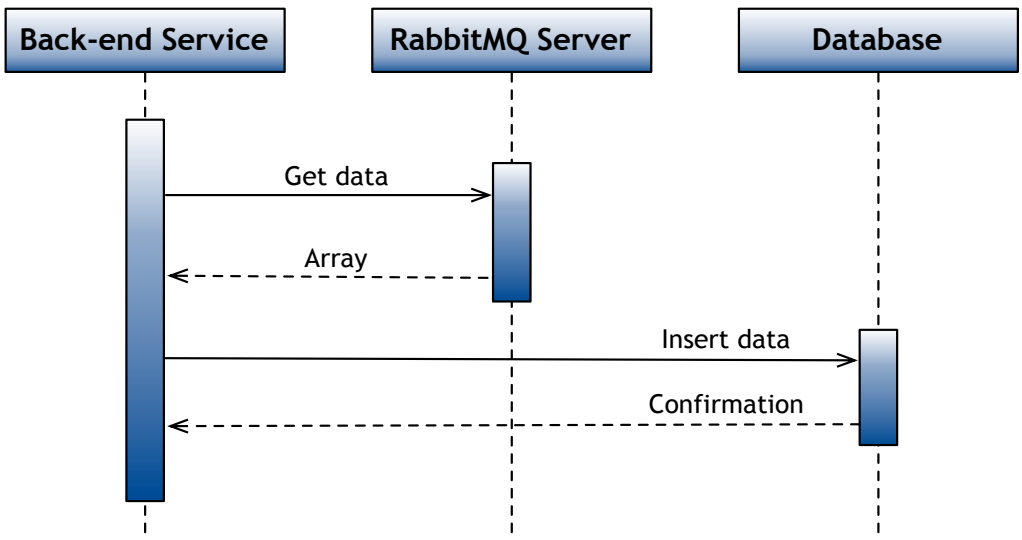


Figure 15. Sequence Diagram – Communication between Back-end service and RabbitMQ server and database.

3.2. System Architecture

Two different approaches have been developed to effect the exchange of messages between the client and the database. Initially created a Web service to which the user sends requests to send and receive data. It was subsequently installed a RabbitMQ server that uses the protocol Advanced Message Queuing Protocol, to where the client will send the messages. Was also implemented a back-end service that will consume messages on the server RabbitMQ.

Figure 16 shows a scenario that illustrates the global system architecture. The user can choose to send messages using a Web service or a service of message queuing. For one thing, if the user chooses to submit messages through the Web service, this will automatically store the messages in database. On the other hand, if the user choose to submit messages through message queuing service is need to use a back-end service to pick up the messages to the message queuing service and then send them to the database.

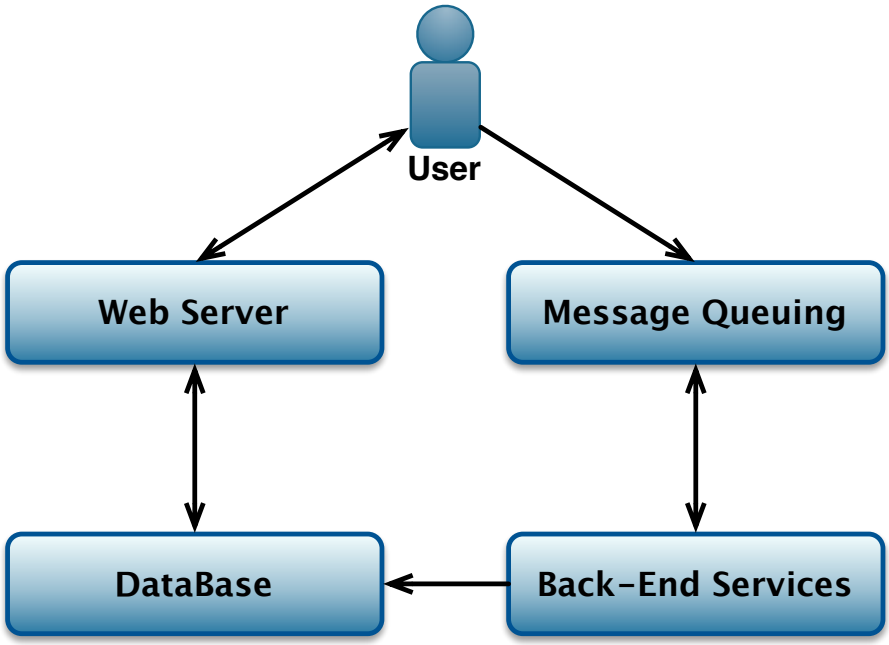


Figure 16. Illustration of the global system architecture.

In system architecture the Web Server runs a RESTful Web service architecture built on java language with which the user communicates to send and receive data, i.e., the user makes HTTP requests to the Web service in order to call the method necessary to insert or query data. With regard to Message Queuing protocol has been implemented AMQP (Advanced Message Queuing Protocol), which requires the installation of a RabbitMQ server. The back-end service was developed in Java and was used to send and get messages from RabbitMQ server and insert them into the database. Finally the database was developed in MySQL language and was used to hold all the system data. The user was a mobile application that communicates with the Web Service through HTTP requests and makes requests through the AMQP port to communicate with the service for message queuing.

3.3. Used Technologies

In the proposed system have been used diverse technologies. The NetBeans IDE has been used to develop and running services for sending and receiving messages. The Web service was developed in Java and runs on Glassfish server version 3.1, a Web server open-source application. To implement AMQP, a protocol application layer open standard for message-oriented middleware, was necessary to install a RabbitMQ server, a message broker software open source that stores messages in a cloud environment, and also develop a Java application to send and consume messages from RabbitMQ server. The database where messages are stored was created using a management system relational database, MySQL, also using MySQL

4. Performance Evaluation

This chapter presents the comparison between the implemented services and equivalents, and between communication protocols implemented. As well focuses on the demonstration of the system and evaluate the performance of two approaches developed that will be integrated into applications AAL.

Initially, in Section 4.1 presents a comparison between the two styles of Web services, SOAP and REST, referring the strengths and weaknesses of each. Yet in this section is presented the comparison between the two languages most used by Web services, and also presented the strengths and weaknesses of each of them. Section 4.2 shows the comparison between two different protocols of message queuing, MQTT and AMQP, referring some of the most important points, such as safety and the size of messages and others. In the Section 4.3, are presented the system demonstration and validation, which are presented in more detail the Web service, the RabbitMQ server and back-end service. Finally, Section 4.4 shows the performance evaluation of the system, where is made a comparison between the two systems implemented through several tests performed for the purpose.

4.1. REST *versus* SOAP

The philosophies of SOAP and RESTful Web Services are very different. Strictly, SOAP is a protocol for XML-based distributed computing, whereas REST adheres much more closely to a Web-based design. SOAP requires a greater implementation and understanding effort of the client

side to difference of REST based APIs, which focus these efforts on the server side. SOAP by itself is not that complex, it can get complex, however, when it is used with its numerous extensions.

Table 3 shows the main strengths and weaknesses for both SOAP and REST [76].

Table 3. Strengths and Weaknesses for both SOAP and REST.

SOAP	
Strengths	Weaknesses
• Language, platform, and transport agnostic • Design to handle distributed computing environments • Is the prevailing standard for Web services, and hence has better support from other standards	• Conceptually more difficult • More "heavy-weight" than REST • More verbose Harder to develop, requires tools
REST	
Strengths	Weaknesses
• Language and platform agnostic • Much simpler to develop than SOAP • Small learning curve, less reliance on tools • Concise, no need for additional messaging layer • Closer in design and philosophy to the Web	• Assumes a point-to-point communication model • Not usable for distributed computing environment • Lack of standards support security, policy, reliable messaging • Tied to the HTTP Transport model

Applying the traditional models of mobile communications may simply result in failure in terms of performance. First the encoding and decoding of verbose XML-based SOAP messages consumes important resources, endangering the performance. Then comes the issue of

communications, Wi-Fi, 3G/GSM are not as efficient and quick as wired, reverting to the processing speed and limited memory and battery problems.

Along with WSDL and XML Schema, SOAP has become the standard for exchanging XML-based messages. SOAP was also designed from the ground up to be extensible, so that other standards could be integrated into it, and there have been many, often collectively referred to as WS-*.

REST is an ancient philosophy more than a new technology. While SOAP looks to drive the next phase of Internet development with a series of new specifications, the REST philosophy advocates that the existing principles and protocols of the Web are enough to create robust Web services. This means that developers who understand HTTP, XML and JSON can start building Web services immediately, without any toolkits beyond that normally uses for the development of Internet applications.

The main point of the REST methodology for write Web services is using an interface that is already well known and used widely: the URI. For example, exposing a stock quote service, in which a user enters a stock ticker, to return to a real time price can be as simple as a script on a Web server accessible via the URI.

This interface method has significant advantages over SOAP-based services. Any developer may discover how to create and modify the URL to access the different Web resources. SOAP, on the other hand, requires specific knowledge of a new XML specification, and most developers will need a toolkit to make SOAP requests and analyse the results.

Other advantage is that the RESTful interface requests and responses can be short. SOAP requires an XML wrapper around each request and response. After typing and namespaces are declared a stock quote from four or five digits in a SOAP response may require more than 10 times as many bytes as would be the same answer in REST.

To compare the performance between REST and SOAP an experiment was performed. The test consists of sending messages through twenty-four users over a period of 30 minutes to the Web service REST and SOAP and

then store them in the database. The test was performed with the Web services installed in Mac OS X server running a virtual machine with Windows 7 Professional operating system and with a processor Inter (R) Xeon(R) CPU 2.67GHz and 1GB of RAM memory Mac. Results are present in Table 4.

Table 4. Results of the Performed Experiment to Compare SOAP with REST Web Services.

	Users	Number of messages stored in database	Average messages sent per second
SOAP	24	192060	106.7
REST	24	223740	124.3

The evaluation results demonstrate the performance advantages of using Web service REST in contrast with Web service SOAP. The Web services REST are faster than Web service SOAP to transmit messages. Like previously mentioned, this experiment confirms the fact that the XML SOAP require a wrapper to each request and response makes the communication slowest, because is necessary to use parsers to read the transmitted data.

SOAP proponents assert that strong typing is a necessary feature for distributed applications. In practice, however, both the request for the service and knowing the types of data ahead of time, thus transferring that information requests and answers is gratuitous.

Perhaps the most interesting aspect of REST vs. SOAP debate is the security angle. While the field of SOAP insists that sending remote procedure calls through the standard HTTP port is a good way to safeguard support of Web services across organizational boundaries, REST followers argue that the practice is a major design failure that compromise network security. REST calls also go over HTTP or HTTPS, but with REST the administrator (or firewall) may discern the intent of each message, examining the HTTP command used in the request, e.g., a GET request can always be regarded as safe because it cannot, by default, modify the data. It can only query data.

A typical SOAP request, on the other hand, will use POST to communicate with a particular service. And without seeing the SOAP envelope, a task that is both resource-consuming and not built into most firewalls, there is no way of knowing if the application simply wants to query data, or delete entire tables from the database.

Regarding the authentication and authorization, SOAP puts the burden on application developer's hands. The REST methodology instead ignores the fact that Web servers already have support for these tasks. By using industry-standard certificates and a common identity management, as a Lightweight Directory Access Protocol (LDAP) server, developers can make the network layer do all the heavy lifting. This is not only useful for developers, but it relieves the weight on administrators who can use something as simple as the files Access Control List (ACL) to manage its Web services just as you would any other URI.

Making calls to an REST/HTTP API is significantly easier than making calls to a SOAP API. The latter requires a client library, a stub and a learning curve. The first is native to all programming languages simple and involves the construction of an HTTP request with appropriate parameters appended to it. Even psychologically former seems much less effort.

Most programming languages make it extremely easy to expose a method using SOAP. The SOAP server library does the serialization and deserialization. To expose the methods of an object as an HTTP API can be relatively more difficult, since it may require serialization of output to XML. Doing the REST API involves additional work to map URI paths to specific handlers to import and the meaning of the HTTP request in the scheme of things. Off course there are many frameworks to make this task easier. However, starting today, it's even easier to expose a set of methods using SOAP, which is to expose them through regular HTTP.

Since HTTP based/RESTful APIs can be consumed through simple GET requests, intermediate proxy server/reverse proxy can cache the answer very easily. On the other hand, SOAP requests use POST and require a complex XML to be created, which caches the response difficult.

Rest is not perfect

REST is not the best solution for each Web service. Data that need to be safe should not be sent as parameters in URIs. And large amounts of data, such as purchase orders in detail, can quickly become cumbersome or even outside the boundaries within a URI. In these cases, the SOAP is actually a solid solution. But it is important to try REST first and resort to SOAP only when necessary. This helps keep the development of simple and affordable.

One comparison has been performance in [77], between SOAP framework and RESTful framework. The comparison demonstrate that RESTful Web services prove to be more suitable for mobile environment because it doesn't require large weight parses, witch supports caching and will save the limited network bandwidth, increase scalability and reliability, and doesn't consume a large amount of mobile resources.

4.1.1. Markup Languages

In this sub section I will present the two languages most used for Web services, the JavaScript Object Notation (JSON) and the eXtensible Markup Language (XML). Initially I will present shortly each of the languages and subsequently reference their strengths and weaknesses.

JavaScript Object Notation (JSON)

JSON is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data [78].

JSON is built on two structures, a collection of name/value pairs and an ordered list of values. A collection of name/value pairs, is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array. An ordered list of values is realized as an array, vector, list, or sequence.

The simplicity of JSON has resulted in its widespread use, especially as an alternative to XML. One of the claimed advantages of JSON over XML as a format for data exchange in this context is the fact that it is much easier to write a JSON parser. This was important for the acceptance of JSON within the community due to the presence of this feature of JavaScript on all Web browsers today.

eXtensible Markup Language (XML)

XML is a set of rules for encoding documents in machine-readable form. Originally designed to meet the challenges of large-scale electronic publishing, XLM is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. XML is a simple, very flexible text format derived from Standard Generalized Markup Language (SGML) [79]. The XML is defined in the XML 1.0 Specification [80] produced by the W3C, and several other related specifications.

The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support through Unicode for the languages of the world. Although the design of XML focuses on documents, it is generally used for the representation of arbitrary data structures, for example in Web services.

Strengths of JSON

JSON is simple and already well defined. Many people can generate invalid JSON (without quote identifiers, use single quotation marks, the use of a Byte Order Mark (BOM) at the beginning), it seems like a problem. This should get better as new libraries come out. Introducing JSON parsers (in the style of the HTML 5) seem to be unnecessary for a simple format that is usually generated by a computer. A strict JSON parser is not a lot of code.

JSON has a simple way to showing which of the permissible encodings it is in, based on the zero bytes at the beginning, as the first two characters must be in ASCII. Permitting a BOM and them allowing Unicode

whitespace might be more standard, but the whitespace has no function expect for use in text editors.

Despite attempts, for example, add an ECMAScript for XML (E4X) to a native XML format and a simple processing model of JavaScript, JSON, remains very easy in most languages to process, at it is built around structures that most languages have only natively while XML is not.

Another advantage of using JSON instead of XML is in the Internet quality of data structuring. JSON is closer to being a HashMap as in Java or an associative array in some other languages like PHP. Hence it does not require parsing libraries in an application as XML does. The time spent however, on parsing XML into the native language, is not so much significant considering the fast processing power of machines today.

The comparison of XML and JSON lies more on the network bandwidth aspect. JSON can take almost half of the bandwidth as XML for transferring the same data. It can achieve the same throughput as passing simple objects in the network in a distributed computing environment.

JSON has distinction between string, number and boolean e.g. {“result: “1”} versus {“result”: 1}. XML document itself doesn’t have the distinction. The distinction saves a bit of programmer’s work to convert to appropriate type manually. You can define data type of values in XML with XML Schema, but it’s complicated and not always available [81].

Weakness of JSON

JSON is not a widespread as XML. Its tooling and language support still fall a bit short of XML, is not a powerful as XML in namespaces. XML throws everything and won’t need to use every single one of its features, but when it’s necessary, does it come in handy.

There are some syntactically different representations: arbitrary white space, although this definition does not include the dull Unicode whitespace characters and backslash escape, which can be represented for the most part directly in Unicode encoding of the document. Whitespace clearly needs to be preserved for readability and use of line oriented

editors and tools. It is not clear how inconvenient it would be if the \u codes were normalizes to Unicode which is the sane default.

Strengths of XML

First, it has enough structure so it is possible to build rich data structures, and to add to that, it has some standard ways (like XHTML) with rich sets of attributes and elements that can be reused in multiple domains, and relations link patter. The other great thing it the set of tools for extraction and processing, which are generally well designed and fairly complete. There are stream and DOM parsers widely available.

Weakness of XML

XML's biggest disadvantage is that its parsers tend to be very large, although the large memory footprint of XML parsers may be reduced to a reasonable size by eliminating unneeded features, and the larger size of XML data records may be an issue for some applications. This can be dealt with by compressing the data before transmitting it or writing.

JSON almost perfect

Binary data is a problem in both XML and JSON. It is necessary a lot of other formats for anything that has binary data, they are just much more efficient, even after compression. Thus, the ideas of a universal format will not happen. The fact that the XML is a lot easier to read and JSON has a smaller footprint.

4.2. Advanced Message Queuing Protocol (AMQP) *versus* Message Queuing Telemetry Transport (MQTT)

AMQP and MQTT are both open protocols for asynchronous message queuing, which have been developed and matured over several years.

AMQP has selected the OASIS industry standards group [82], with the intention of moving to becoming an ISO/IEC standard. MQTT has chosen to use the Eclipse foundation [15].

Both provide basic messaging needs; beyond that, AMQP provides a very much richer set of messaging scenarios. AMQP is almost a complete superset, lacking only explicit protocol support for Last-Value-Queues and will messages.

Both protocols are being promoted for 'widespread' use in the internet:

- MQTT as a low-overhead, simple to implement way to send data, especially from embedded devices;
- AMQP as the asynchronous complement to HTTP.

As such, both are being promoted as being ideal for cloud computing and the "internet of things". Message queuing, with their asynchronous nature and minimal need for configuration when well done, is perfect for interoperating many different environments.

Beginnings

AMQP come from the financial community, and it is mainly customer-driven: its makers wanted an open way to communicate greatly increase over-the-counter trace, risk and clearing market data the transfer, and doing it without the necessity the pain of a bespoke protocol and its licensing headache. MQTT is vendor-driven; that comes from IBM and their partners as a response to the high cost of implementation MQSeries imposes on its customers using small devices. Both of these approaches have greatly influenced the design and features of the protocols.

Intended use of protocol

The two protocols "sit" above TCP / IP, and are designed to be used to enable programs to send and receive messages asynchronously, irrespective of their choice of hardware, operating system or programming

language. The protocols diverge; MQTT is designed to be helpful for many small, relatively dumb devices sending small messages on low-bandwidth networks. AMQP, on the other hand, is designed to provide the full vibrancy of messaging scenarios. MQTT's design goals are a subset of its intended uses.

In particular, MQTT really sees the networks between the involved parties as a controlled, near private infrastructure. AMQP, in turn, is designed supposing it is in use between parties under different controls and who use network and infrastructure resources outside of those parties control.

Framing Optimisation

Both of these protocols provide for heavily optimized "on-the-wire" framing of data. MQTT uses a more stream-orientated approach, making it easier for low-memory clients to write frames. AMQP uses a buffer-orientated approach, making possible high-performance servers. MQTT does not allow fragmentation of messages, making it hard to transmit large messages with constrained memory devices, however.

Messaging Scenarios

MQTT supports publish-subscribe messaging to topics. MQTT's messaging is actually ephemeral: it is optimised for the use case of active routing of simultaneously connected publishers and subscribers. Consequently, it is very difficult to use it for classic long-lived message queuing. AMQP supports this use case, and more, with five different kinds of message publisher-consumer "lifetime", from "as long as connected" to "nobody is using this queue".

AMQP permits almost any form of messaging including classic message queues, round-robin, store-and-forward and combinations thereof.

Transactions

This is short but poignant. MQTT not support transactions, it support basic acknowledgments. AMQP supports different acknowledgments uses cases and transactions across message queues, it allows separation of the different transactional semantics, should that be needed, and for acknowledgments to be out-of-order or even delayed, and batched us as a performance optimisation.

Connection Security

MQTT does not address connection security, although the community does provide advice. AMQP on the other hand, has specifically worked to integrated with Transport Layer Security (TLS) [83] and Simple Authentication and Security Layer (SASL)[84], the Internet Engineering Task Force set of RFCs that provide appropriate ways of securing the right to use a connection. AMQP core design allows separate negotiation of, and policies for, TLS and SASL mechanisms and upward replacement with alternative techniques as they develop.

User security

MQTT requires short user names and short passwords that do not provide enough entropy in the modern world. It has made these part of the protocol itself, so any change in policy, or security weakness, requires a new protocol version. AMQP uses SASL mechanisms, allowing organisations to choose the security that matters to them without protocol change.

Reliable Messaging

Essentially, most users of messaging either care a message is sent and definitively received once, or they do not. Both protocols provide for “fire-and-forget, do not try to hard” messaging. AQMP provides fine-grained control over this, should it be reliable, but order of delivery matters.

Both protocols claim to provide reliable messaging, essentially using a series of acknowledgments to give “exactly-once” receipt of a message.

However, under analysis, this is not always the case with MQTT. MQTT assumes general reliability of the parties involved. AMQP addresses these scenarios with link recovery, which allows fine-grained control, and will ensure eventual delivery under hostile conditions.

Implementation

It is certainly easier to implement MQTT, it is a much smaller protocol. Open protocols result in open source libraries. The vast majority of users will simply choose the open source client library for their operating system or language. However, a simple protocol does not necessarily mean less operation size.

MQTT and AMQP are both message queuing protocols, suitable for use in hardware and software and on all major operating systems and platforms. MQTT is suited to its use case to simple clients talking to a server, but any infrastructure using it is exposed to serious security weaknesses and an inability to make best use of resources or to support additional use cases. AMQP is suited to these uses cases and many others, supports far better use of resources, far more pragmatic security and message reliability and has a future place as an ISO standard.

4.3 Systems Demonstration and Validation

This section presents the systems developed and their validation. The RabbitMQ server, the Web server, the back-end service and the database were created on a Mac OS X server running a virtual machine with Windows 7 Professional operating system and with a processor Inter (R) Xeon(R) CPU 2.67GHz and 1GB of RAM memory. The user application was developed for mobile devices running the Android operating system. The application will run from the 2.0 version (Eclair) of Android and higher.

4.3.1 RESTful Web Service, RabbitMQ server and Back-end Service

In this section will be presented in more detail the different services implemented in this study, making reference to the methods of the Web service, the management and monitoring interfaces the RabbitMQ server and finally the functions of the back-end service.

RESTful Web Service

RESTful Web services are services built using the RESTful architectural style. In this study, the RESTful Web service has been developed in Java using the Java API for RESTful Web services (JAX-RS) and Jersey, the open source JAX-RS, supported by IDE NetBeans 7.2.1 and runs on the Web server Glassfish version 3.1. In addition, the IDE also supports testing, building client applications that access RESTful web services, and generating code for invoking web services.

On the Web service have been implemented a number of methods for inserting and querying data. The methods of the Web service using the HTTP PUT method are used to insert or update data while that use the HTTP GET method is used to query the data from the database.

In Table 5 are presented the different methods implemented on my Web service and the HTTP methods used by each of them. In the methods presented in Table 5, the parameters that are enclosed in braces ({}) concern the data that the user must provide when making a request to the Web service. Regarding the query methods (GET) in Table 5 are shown only methods that use "idUser" as search parameter, however, were also implemented other research methods to the same table but with other search parameters, such as the date and the respective fields in each table.

Table 5. Methods implemented in the Web service reference to the HTTP method to use

HTTP method	Web service method
PUT	/Insert_Profile/{name}/{ birthDate}/{weight}/{height}/{user} /Inserir_Contact/{idUser}/{name}/{phone}/{mail} /Insert_Fall/{idUser}/{forceG}/{date} /Insert_AirPressure/{idUser}/{pressure}/{date} /Insert_EEG/{idUser}/{eeg}/{date} /Insert_ecg/{idUser}/{ecg}/{date}
GET	/ProfileName/{name} /getcontacts/{idUser} /getfallsuser/{idUser} /getairpressureUser/{idUser} /geteeg/{idUser} /getecg/{idUser}

RabbitMQ server

To build the RabbitMQ server is required various tools. RabbitMQ requires a recent version of Python. In addition you also need the Erlang development and runtime tools, a recent version of GNU make a recent version of xsltproc.

For installing the RabbitMQ server on Windows is need install Erlang and then install RabbitMQ and run it as a service, using the basic configuration. If it is necessary configure names, ports, locations is also very easy, just configure the environment variables in Windows. In this system was used the basic settings.

For management and monitoring the RabbitMQ server more easily and with a visual interface it was necessary to also install an API based on HTTP, the rabbitmq-management plugin.

This plugin includes the following features:

- Declare, list and delete exchanges, queues, bindings, users, virtual hosts and permissions;
- Monitor queue length, messages rates globally and per channel, data rates per connection, etc.
- Send and receive messages;
- Export / import object definitions to JSON;
- Force the closing of connections, clean queues.

To use the Web UI is needed authenticating as a user of RabbitMQ and only then it is possible to manage exchanges, queues, bindings, etc. By default the server issues statistics every 5000ms. The average value of messages shown on the management plugin is calculated in this period of time. Figure 17 shows the visual interface RabbitMQ management plugin obtained during the execution of on of the experiments performed in this study. This Figure refers to experiment 1, since we have several connections but no consumer.

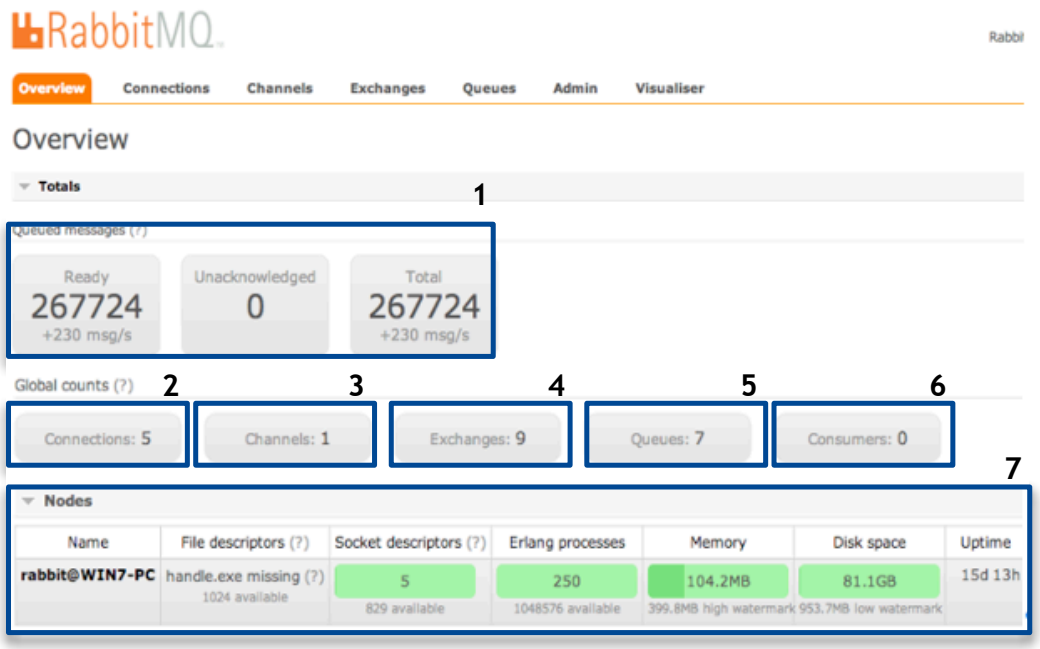


Figure 17. RabbitMQ management plugin interface

The image caption is a follows:

1. Number of messages stored in the existing queues on the RabbitMQ server, and average messages received by the server per second;
2. Number of clients connected to the server and sending messages;
3. Number of channels created to communicate between client and server.
4. Number of exchanges defined on the server.
5. Number of queues present on the server where it will be stored messages.
6. Number of consumers connected to the server
7. Name and information of node where the RabbitMQ server is installed.

Back-End Service

The back-end service has been developed in Java using Netbeans version 7.2.1. This service is used to consume the messages from RabbitMQ server, process them and store them in the database. This program when executed, tries to make the connection to RabbitMQ and then gets the messages are stored. Finally, the service processes the messages obtained so as to separate the various fields necessary to create and execute a query that will allow insert data in the database.

4.4 Results Analysis

Several experiments were performed and deployed for this study in order to compare both implemented services. With these experiments the authors tried to find out the main advantages regarding the use of a Web service, compared to using Message Broker open source software, or vice versa. In these experiments, twenty-four handsets were used continuously to send messages to servers.

The experiments were performed as follows:

- Experiment 1 – the user application sends messages for 30 minutes to RabbitMQ using AMQP protocol without no active consumer (back-end service);
- Experiment 2 – the user application sends messages for 30 minutes to Web service server using HTTP protocol, and then, the Web service communicates with the database using JDCB driver to store the messages;
- Experiment 3 - the user application sends messages for 30 minutes to RabbitMQ using AMQP protocol with a consumer (back-end service) to read the messages, process them and through JDBC driver connect to the database to store the messages.

In these experiments, the user had a single function to send data to the service without ever doing queries. In this way is able to obtain a more reliable comparison of the two services when many clients are constantly sending data. In experiment 3, the consumer is consuming messages and save then in the database at the same time that users are sending, the results are presented in Table 6 and in Figure 18.

Table 6. Results of the Performed Experiments.

	Users	Messages stored in RabbitMQ	Messages stored in Database	Average messages sent per second
Experim. 1	24	407793	n/a	226.6
Experim. 2	24	n/a	226530	125.9
Experim. 3	24	160747	219907	211.5

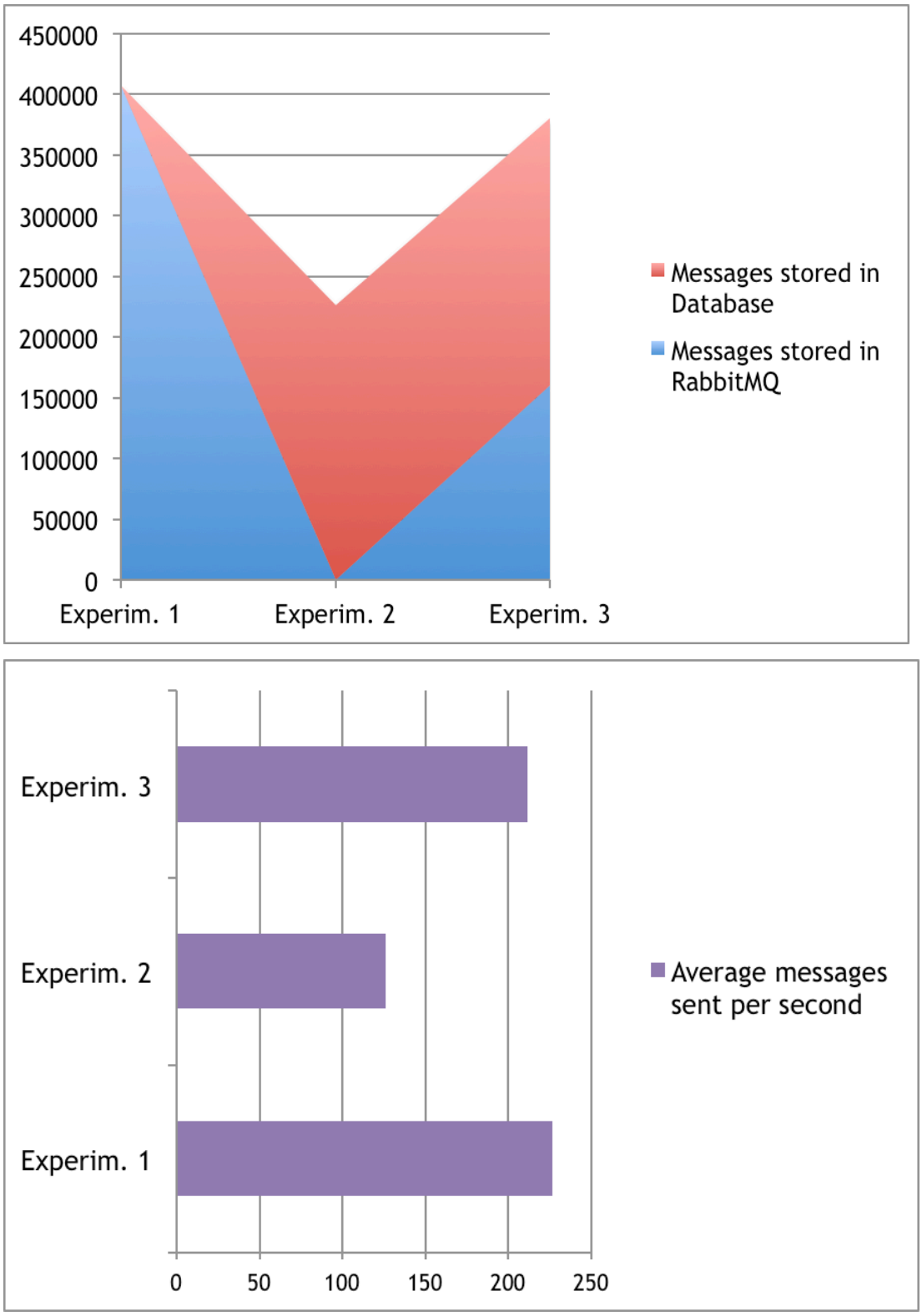


Figure 18. Performance comparison considering the number of message for tree different experimental scenarios showing the messages stored in the Database, RabbitMQ,and the average messages per second.

As above-mentioned, the objective of the experiment 1 passed through the objective to send messages to the RabbitMQ, during 30 minutes, using 24 users. As it can be observed in Table 6 and Figure 18, in the experiment 1, a total number of 407793 messages were sent and stored with an average of 226.6 messages per second. In the experiment 2, in the same interval of time as the experiment 1, but now with the objective to sent messages through Web services and storing them in the database, a total of 226530 messages were stored with an average of 125.9 messages per second. Finally. In the 3rd experiment, the same interval of time was used for the experiments 1 and 2. Its objective includes sending and storing the messages using the RabbitMQ, a total number of 380654 were sent and, after, the time was over a total of 160747 messages stayed stored in the RabbitMQ. A total number of 219907 messages where stored in the database, making a average of 211.5 messages per second. None of the messages in the experiment 3 where lost, some of them stay in the RabbitMQ because it takes extra time to the client in order to obtain the messages from the stack and put them in the database.

With these experiments it is possible to conclude that when the AMQP protocol is used to messages exchange it will send a larger number of messages per second as can be seen in the experiments 2 and 3. There is a big difference between RabbitMQ and the Web service since the Web Service is in charge of receiving the request introduced by the user and then, depending on the method called, save the message on respective table of the database. While RabbitMQ only sends messages to the server when the server asks for it and stores the messages in a queued from the client, the client connects with the RabbitMQ.

Is can be seen the number of messages in test 2 is significantly smaller compared to the tests 1 and 3. This is due to the fact that when using the Web service, the client waits that the services insert the data in the database and give him a response. Only after the client sends new data. The process of storing data in the database takes more time doing also with some client messages are lost while waiting for confirmation of

the Web service about the conclusion of the communication process, because the client does not exist any mechanism to temporarily store the messages. On the other hand when using the RabbitMQ server the client sends the data and waits for confirmation of RabbitMQ then sending new data. This process is almost twice faster than the first avoiding loss of messages.

5. Conclusions and Future Work

5.1 Conclusions

This chapter presents a synthesis of this dissertation along with the main achievements and important conclusions that result from this work and points some directions for future work. The main objective of this dissertation is the design, constructing, deployment, and performance evaluation of Web services when employed in system that generate large amounts of data. It was carried out with the development of the RESTful web service applied in a real system of Ambient Assisted Living. Thus, all the dissertation objectives were successfully accomplished and the all intermediate objectives were successfully achieved.

After introducing and presenting the topic of this dissertation, definition the problem, define its objectives and main contributions chapter two present the revision literature on Web services, with special interest in RESTful architecture. But also making references to other styles of Web services and use the existing approaches. It also presented the state-of-the-art on Message Queuing Protocol, the Advanced Message Queuing Protocol and the Message Queuing Telemetry transport, showing some of the products that use these protocols. And finally, some import points of Ambient Assisted Living as well as some developed and developing work in this area.

Chapter three presents the analysis of requirements before developing it. First define the essential requirements, that is, the database structure, the activity and sequence diagrams and the system architecture. Next, and finally discuss the technologies used for the implementations of system.

In chapter four are presented systems implemented in this study, and shows the comparison between the implemented technologies and their equivalents. In the first comparison is made between the two architectures of Web services, RESTful and SOAP, the forces and the weaknesses of each of them. In this study, after making the requirements analysis and the comparison between the two architectures concludes that RESTful was the best architecture to implement. Still about Web services are presented the strengths and weaknesses of the two main markup languages used by Web services, JSON and XML. Taking into account the needs of the system that should be prepared to exchange large amounts of messages, conclude that the way forward was to use JSON to be lighter than XML. Then, presents the comparison between two protocols message queuing, the MQTT and AMQP, making reference to some of the most important points to keep in mind when we want to implement these protocol. In this study chose to use the AMQP, since the newest mobile devices already has good capability of processing. Finally, is performed the demonstration of the system and presented the performance evaluation, accomplished through tests, the comparison between the use of RESTful Web services and RabbitMQ server to exchange messages between client and server. In this comparison, it was concluded that the use RabbitMQ obtain the best results, it is possible to exchange an increased amount of messages and lossless compared with the Web service.

5.2 Future Works

To conclude this work, it just remains to suggest future research directions based on current work:

- Implement a MQTT system to allow the use of sensors for sending data.

References

[1] H. Gunzer and S. Engineer, *Introduction to web services*. Scotts Valley, California: Borland Software Corporation, 2002, pp. 2–9.

[2] H. Wang, J. Z. Huang, Y. Qu, and J. Xie, "Web services: problems and future directions," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 309–320, Apr. 2004.

[3] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, "Migration of SOAP-based services to RESTful services," presented at the 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), Williamsburg, USA, 2011, pp. 105–114.

[4] H. Limam and J. Akaichi, "Managing and Querying Web Services Communities: A Survey," *International Journal of Database Management Systems (IJDMS),*, vol. 3, no. 1, Feb. 2011.

[5] V. Krithika, D. A. Kaur, and D. K. C. Sekaran, "Web Services Supply Chains: A Literature Review," *International Journal of Web Services Computing*, vol. 3, no. 1, Mar. 2012.

[6] *MobileBeyond, "Mobile Internet Research Report Shows Massive Growth" [Online]. Available: http://mobilebeyond.net/mobile-internet-research-report/#axzz1M5vzN8G7 [Accessed: February 2013].*

[7] S. Oh and G. C. Fox, "Optimizing Web Service Messaging Performance in Mobile Computing," *Future Generation Computer Systems*, vol. 23, no. 4, pp. 623–632, May 2007.

[8] S. Oh, H. Bulut, A. Uyar, W. Wu, and G. Fox, "Optimized communication using the SOAP infoset for mobile multimedia

collaboration applications," presented at the Proceedings of the 2005 international conference on Collaborative technologies and systems, Washington, DC, USA, 2005, pp. 32–39.

[9] F. Aijaz, M. A. Chaudhary, and B. Walke, "Performance Comparison of a SOAP and REST Mobile Web Server," presented at the Third International Conference on Open-Source Systems and Technologies (ICOSST 2009), Lahore, Pakistan, 2009.

[10] X. Xiong and J. Fu, "Active Status Certificate Publish and Subscribe Based on AMQP," presented at the 2011 International Conference on Computational and Information Sciences, Chengdu, China, 2011, pp. 725–728.

[11] C. Gagné and M. Parizeau, "Genericity in Evolutionary Computation Software Tools: Principles and Case-Study," *International Journal on Artificial Intelligence Tools*, vol. 15, no. 2, pp. 173–194, Apr. 2006.

[12] G. N. Rodrigues, V. Alves, R. Franklin, and L. Laranjeira, "Dependability Analysis in the Ambient Assisted Living Domain: An Exploratory Case Study," presented at the 2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS),, Bahia, Brazil, 2010, pp. 150–159.

[13] V. Fuchsberger, "Ambient assisted living," presented at the SAME '08 Proceedings of the 1st ACM international workshop on Semantic ambient media experiences, New York, USA, 2008, pp. 21–24.

[14] *OASIS, "OASIS Forms AMQP Technical Committee to Advance Business Messaging Interoperability Within Middleware, Mobile, and Cloud Environments" [Online]. Available: https://www.oasis-open.org/news/pr/amqp-tc [Accessed: March 2013].*

[15] *MQTT.org, "Eclipse Paho, Open Source, and other news" [Online]. Available: http://mqtt.org/2011/11/eclipse-paho-open-source-and-other-news [Accessed: March 2013].*

[16] *D. Booth, H. Haas, F. McCabe, E. NewComer, M. Champion, C. Ferris and D. Orchard, "Web Service Architecture" [Online]. Available: http://www.w3.org/TR/ws-arch/ [Accessed: February 2013].*

[17] H. Limam and J. Akaichi, “Web services synchronization health care application,” *International Journal of Web & Semantic Technology (IJWesT)*, vol. 2, no. 2, pp. 40–57, Apr. 2011.

[18] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts And Design*, 2nd ed. 2005.

[19] D. A. Chappell and T. Jewell, *Java web services*, 1st ed. O'Reilly, 2002, pp. 8–16.

[20] Y. Wang and E. W. Stroulia, “Flexible interface matching for Web-service discovery,” presented at the WISE 2003, Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003, pp. 147–156.

[21] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, “Similarity search for web services,” presented at the VLDB '04 Proceedings of the Thirtieth international conference on Very large data bases, Toronto, Canada, 2004, vol. 30, pp. 372–383.

[22] R. Nayak and B. Lee, “Web Service Discovery with additional Semantics and Clustering,” presented at the IEEE/WIC/ACM International Conference on Web Intelligence, Fremont, California, 2007, pp. 555–558.

[23] H. R. Motahari Nezhad, G. Y. Xu, and B. Benatallah, “Protocol-aware matching of web service interfaces for adapter development,” presented at the WWW '10 Proceedings of the 19th international conference on World wide web, Raleigh, USA, 2010, pp. 731–740.

[24] R. Mikhaiel and E. Stroulia, “Examining Usage Protocols for Service Discovery,” in *Lecture Notes in Computer Science*, vol. 4294, no. 46, A. Dan and W. Lamersdorf, Eds. Chicago, USA: Service-Oriented Computing – ICSOC 2006, 2006, pp. 496–502.

[25] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” University of California, Irvine, 2000.

[26] L. Richardson and S. Ruby, *RESTful web services*. Sebastropol, Ukraine:, 2007, pp. 23–106.

[27] C. Pautasso and E. Wilde, “RESTful web services: principles,

patterns, emerging technologies," presented at the WWW '10 Proceedings of the 19th international conference on World wide web, Raleigh, USA, 2010, pp. 1359–1360.

[28] M. Laitkorpi, P. Selonen, and T. Systa, "Towards a Model-Driven Process for Designing ReSTful Web Services," presented at the ICWS 2009. IEEE International Conference on Web Service., Los Angeles, USA, 2009, pp. 173–180.

[29] J. R. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. N. Taylor, "From representations to computations: the evolution of web architectures," presented at the ESEC-FSE '07 Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Dubrovnik, Croatia, 2007, pp. 255–264.

[30] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," *Communications Magazine, IEEE*, vol. 48, no. 12, pp. 92–100, 2010.

[31] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services," presented at the WI-IAT '08. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Sydney, Autralia, 2008, vol. 1, pp. 619–625.

[32] R. Alarcón and E. Wilde, "RESTler: crawling RESTful services," *WWW '10 Proceedings of the 19th international conference on World wide web*, pp. 1051–1052, 2010.

[33] Y. Liu, Q. Wang, M. Zhuang, and Y. Zhu, "Reengineering Legacy Systems with RESTful Web Service," *COMPSAC '08. 32 Annual IEEE International Computer Software and Applications*, pp. 785–790, 2008.

[34] P. Mayer and D. Lübke, "Towards a BPEL unit testing framework," *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pp. 33–42, Jul. 2006.

[35] S. Krishnamurthi and T. Bultan, "Discussion summary: Characteristics of web services and their impact on testing, analysis and verification," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 1, 2005.

[36] C. Riva and M. Laitkorpi, "Designing Web-Based Mobile Services with REST," in *Service-Oriented Computing-ICSOC 2007 …*, vol. 4907, no. 42, Viena, Austria: ICSOC 2007, International Workshops, 2009, pp. 439–450.

[37] *E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, "Web Service Description Language (WSDL)," [Online]. Available: http://www.w3.org/TR/wsdl [Accessed: February 2013].*

[38] *Windows Communication Foundation, "What Is Windows Communication Foundation," [Online]. Available: http://msdn.microsoft.com/en-us/library/ms731082.aspx [Accessed: February 2013].*

[39] X. Feng, J. Shen, and F. Ying, "REST: An alternative to RPC for Web services architecture," presented at the First International Conference on Future Information Networks, Beijing, China, 2009, pp. 7–10.

[40] W. Choi and J. Kim, "Development of the RESTful JPIC SDK for the Application Using Public Information," *Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops*, pp. 352–360, 2011.

[41] *OASIS, "AMQP is the Internet Protocol for Business Messaging" [Online]. Available: http://www.amqp.org/ [Accessed: February 2013].*

[42] J. O'Hara, *Toward a commodity enterprise middleware*, vol. 5, no. 4. ACM, 2007, pp. 48–55.

[43] S. Vinoski, "Advanced Message Queuing Protocol," *IEEE Computer Society*, vol. 10, no. 6, pp. 87–89, 2006.

[44] J. Kramer, "Advanced message queuing protocol (AMQP)," *Linux Journal*, vol. 2009, no. 187, p. 3, Nov. 2009.

[45] D. Gunter, E. Deelman, T. Samak, C. H. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, and K. Vahi, "Online workflow management and performance analysis with stampede," presented at the Proceedings of the 7th International Conference on Network and Services Management, Paris, France, 2011, pp. 152–161.

[46] J. Robinson, J. Frey, A. Stanford-Clark, A. Reynolds, and B. Bedi, "Sensor networks and grid middleware for laboratory monitoring," *Proceedings of the First International Conference on e-Science and Grid Computing*, pp. 569–575, Jun. 2005.

[47] M. Prihodko, "Energy Consumption in Location Sharing Protocols for Android Applications," Linköping University, The Institute of Technology, Linköping, Sweden, 2012.

[48] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[49] *IBM, "IA92: WBI Brokers - Java Implementation of WebSphere MQ Telemetry Transport" [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=swg24006006 [Accessed: April 2013].*

[50] *IBM, "IA93: WITHDRAWN: WBI Brokers - C Implementation of WebSphere MQ Telemetry Transport" [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=swg24006525 [Accessed: April 2013].*

[51] *"Eurostat (2008). Population projections 2008--2060 -- From 2015, deaths projected to outnumber births in the EU27 Almost three times as many people aged 80 or more in 2060," (STAT/08/119), 26AD.*

[52] H. Sun, V. D. Florio, N. Gui, and C. Blondia, "Promises and Challenges of Ambient Assisted Living Systems," presented at the 2009 Sixth International Conference on Information Technology: New Generations, Las Vegas, USA, 2009, pp. 1201–1207.

[53] *AAL - AMBIENT ASSISTED LIVING,_"Ambient Assisted Living (AAL)*

Joint Programme." [Online]. Available: http://www.aal-europe.eu [Accessed: February 2013].

[54] M. A. Estudillo-Valderrama and L. M. Roa, "Ambient Assisted Living: A methodological approach," *32nd Annual International Conference of the IEEE EMBS*, pp. 2155–2158, 2010.

[55] A. J. Quigley and B. Knapp, "Bridging Research in Aging and ICT Development," *Proceedings of the International Federation on Aging 10th Global Conference 2010*, 2010.

[56] D. Wright, "Structuring stakeholder e-inclusion needs," *Journal of Information, Communication and Ethics in Society*, vol. 8, no. 2, pp. 178–205, 2010.

[57] A. Jin, Bin Yin, G. Morren, H. Duric, and R. Aarts, "Performance evaluation of a tri-axial accelerometry-based respiration monitoring for ambient assisted living," presented at the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Minneapolis, USA, 2009, pp. 5677–5680.

[58] E. S. Sazonov, O. Makeyev, S. Schuckers, P. Lopez-Meyer, E. L. Melanson, and M. R. Neuman, "Automatic Detection of Swallowing Events by Acoustical Means for Applications of Monitoring of Ingestive Behavior," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 3, pp. 626–633, 2010.

[59] I. C. Lopes, B. Vaidya, and J. J. P. C. Rodrigues, "Towards an autonomous fall detection and alerting system on a mobile and pervasive environment," *Telecommun Syst*, pp. 1–12, Jun. 2011.

[60] S. S. Torkestani, S. Sahuguede, A. Julien-Vergonjanne, J. Cances, and J. C. Daviet, "Infrared Communication Technology Applied to Indoor Mobile Healthcare Monitoring System," *International Journal of E-Health and Medical Communications*, vol. 3, no. 3, pp. 1–11, 2012.

[61] V. Venkatesh, V. Vaithyanathan, M. P. Kumar, and P. Raj, "A secure Ambient Assisted Living (AAL) environment: An implementation view," presented at the 2012 International Conference on Computer

Communication and Informatics (ICCCI), Coimbatore, India, 2012, pp. 1–7.

[62] I. Martinez and J. Escayola, "Standard-based middleware platform for medical sensor networks and u-health," *Proceedings of 17th International Conference on Computer Communications and Networks*, pp. 1–6, 2008.

[63] F. Zhou, J. R. Jiao, S. Chen, and D. Zhang, "A Case-Driven Ambient Intelligence System for Elderly in-Home Assistance Applications," *IEEE Trans. Syst., Man, Cybern. C*, vol. 41, no. 2, pp. 179–189, 2011.

[64] N. Noury, P. Barralon, N. Vuillerme, and A. Fleury, *Fusion of Multiple Sensors Sources in a Smart Home to Detect Scenarios of Activities in Ambient Assisted Living*, vol. 3, no. 3. International Journal of E-Health and Medical Communications (IJEHMC), 2012, pp. 29–44.

[65] C. Stephanidis, *User interfaces for all: New perspectives into human-computer interaction*, 1st ed. User Interfaces for All-Concepts, Methods, and Tools, 2001, pp. 3–17.

[66] M. Silva, P. M. Teixeira, F. Abrantes, and F. Sousa, *Design and Evaluation of a Fall Detection Algorithm on Mobile Phone Platform*, vol. 70, no. 4. Ambient Media and Systems, 2011, pp. 28–35.

[67] A. Hristova, A. M. Bernardos, and J. R. Casar, "Context-aware services for ambient assisted living: A case-study," *First International Symposium on Applied Sciences on Biomedical and Communication Technologies*, pp. 1–5, 2008.

[68] B. O'Flynn, P. Angove, J. Barton, and A. Gonzalez, "Wireless biomonitor for ambient assisted living," *Oral presentation at …*, 2006.

[69] M. A. Stelios, A. D. Nick, M. T. Effie, K. M. Dimitris, and S. C. A. Thomopoulos, "An indoor localization platform for ambient assisted living using UWB," presented at the MoMM '08: Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia, Linz, Austria, 2008, pp. 178–182.

[70] P. A. Bromiley, P. Courtney, and N. A. Thacker, "Design of a visual system for detecting natural events by the use of an independent visual estimate: A human fall detector," *Empirical Evaluation Methods in Computer Vision*, 2002.

[71] J. Clemensen, S. B. Larsen, M. Kyng, and M. Kirkevold, "Participatory Design in Health Sciences: Using Cooperative Experimental Methods in Developing Health Services and Computer Technology," *Qualitative Health Research*, vol. 17, no. 1, pp. 122–130, Jan. 2007.

[72] G. Zimmermann, J. Alexandersson, C. Buiza, E. Urdaneta, U. Diaz, E. Carrasco, M. Klima, and A. Pfalzgraf, "Meeting the Needs of Diverse User Groups," in *Benefits and Costs of Pluggable User Interfaces in Designing for Older People and People with Cognitive Impairments*, no. 6, IGI Global, pp. 80–93.

[73] A. Hochgatterer, L. Roedl, and A. Martinez, "Requirements for a behaviour pattern based assistant for early detection and management of neurodegenerative diseases," *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pp. 346–353, 2011.

[74] P. P.-S. Chen, "The entity-relationship model---toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, Mar. 1976.

[75] *IDEF1X, "Data Modeling Method" [Online]. Available http://www.idef.com/IDEF1x.htm [Accessed: April 2013].*

[76] P. A. Castillo, J. L. Bernier, M. G. Arenas, J. J. Merelo, and P. Garcia-Sanchez, "SOAP vs REST: Comparing a master-slave GA implementation," *First International Workshop of Distributed Evolutionary computation in informal Environments,* May 25, 2011.

[77] F. AlShahwan and K. Moessner, "Providing SOAP Web services and RESTful Web services from mobile hosts," *2010 Fifth International Conference on Internet and Web Applications and Services*, pp. 174–179, 2010.

[78] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” 2006.

[79] *W3C, “Extensible Markup Language (XML)” [Online]. Available: http://www.w3.org/XML/ [Accessed: April 2013].*

[80] *W3C, “Extensible Markup Language 1.0 Recommendation” [Online]. Available: http://www.w3.org/TR/REC-xml/ [Accessed: May 2013].*

[81] *Douglas Crockford, “JSON: The Fat-Free Alternative to XML” [Online]. Available: http://www.json.org/fatfree.html [Accessed: May 2013].*

[82] *OASIS, “OASIS Advanced Message Queuing Protocol (AMQP) TC ” [Online]. Available: https://www.oasis-open.org/committees/ tc_home.php?wg_abbrev=amqp [Accessed: April 2013].*

[83] T. D. T. D. org, “The Transport Layer Security (TLS) Protocol Version 1.2,” 2008.

[84] J. G. M. J. N. com, “Simple Authentication and Security Layer (SASL),” 1997.

Appendice

In this appendix is included an article presented at *The Fifth International Conference Ubiquitous and Future Networks* (ICUFN 2013), Da Nang, Vietnam, July 02 - 05, 2013.

Performance Evaluation of RESTful Web Services and AMQP Protocol

Joel L. Fernandes[1], Ivo C. Lopes[1], Joel J. P. C. Rodrigues[1], and Sana Ullah[2]
[1] Instituto de Telecomunicações, University of Beira Interior, Covilhã, Portugal
[1] King Saud University, Riyadh, Saudi Arabia
joellfernandes@gmail.com; ivo.lopes@it.ubi.pt; joeljr@ieee.org; sullah@ksu.edu.sa

***Abstract*— Web services appeared as a promising technology for Web environments independent of technologies, services, and applications. Currently, there are some issues related with this approach that should be studied. For instance, if massive quantities of data are sent to databases it can influence significantly the performance of the whole system. The Advanced Message Queuing Protocol (AMPQ) appears as a promising solution to address this problem. Then, in order to evaluate the performance of this approach, this paper presents a performance comparison study of RESTful Web services and the AMQP Protocol considering exchanging messages between client and server. The study is based on the averaged exchanged messages for a period of time. It was observed and concluded that, for large quantities of messages exchange, the best results comes from the Advanced Message Queuing Protocol.**

Keywords*— Web Services; AMPQ; RESTful; Mobile Computing; Mobile Health Application

INTRODUCTION

The World Wide Web (or simply Web) has been a phenomenal success allowing simple computer/human interactions at the Internet scale. The original HyperText Transfer Protocol (HTTP) and HyperText Markup Language (HTML), current technologies used by Web browsers proven to be an effective way to design user interfaces in a wide variety of devices.

In recent years, Web services technologies were effectively used to simplify interoperability between different systems whilst providing litheness and scalability for several applications, inclusive the distributed simulation software. The World Wide Web Consortium (W3C) [1] defines a Web service like a method of communication between two electronic devices over a network. It is a key solution used in systems integration and interaction among diverse types of applications. With this technology it is possible that novel applications may interact with existent and implement a system on different platforms. Essentially, a Web service does software features available on a network in a standardized way.

The increasing number of available Web services, the growing need of collaboration, the need for knowledge sharing, and the necessity to take better decisions are the factors which generate a growing interest in the Web services. In 2000, Roy Fielding [2] introduced the term Representational State Transfer (REST) like an architectural style for dispersed hypermedia systems. In this context, an architecture is considered a set of characteristics and constraints on the elements of architecture that induce a set of desired properties. REST is an abstraction of a basic architecture of the HTTP Protocol and concentrates on concepts instead of on technical details and syntax. REST architectural features and restrictions aim to collect the fundamental design principles that allow high scalability, expansion, and success of the Web. A RESTful service is supplied as a feature, which is the concept extremely useful and can be addressed on the web.

As Internet grows continuously, new needs are identified, and to address them, new approaches are emerging. One of them is the publish-subscribe (Pub-Sub) where the messaging middleware is a working model with asynchronous and loosely coupled characteristics. As the message producers and consumers in time, space, and control flow are completely decoupled, making the two end-points communicating through a separate release and subscription messaging for asynchronous communication, which can better meet the large-scale and dynamic distributed information systems integration needs. The current subject-based publish-subscribe system has been successfully used in the financial, stock, mobile computing, and other situations [3].

In 2006, a new proposal originated in the financial services industry, called Advanced Message Queuing Protocol (AMQP), was proposed [4-6]. AMQP is an open standard for Message Oriented Middleware (MOM) communication. Then, in order to evaluate the performance of this approach, this paper presents a performance comparison study considering RESTful Web services and the AMQP Protocol considering exchanging messages between client and server.

The remainder of the paper is structured as follows. Section II elaborates on the related work with focus on available approaches, considering the two technologies under study (RESTful Web Services and AMQP Protocol). Section III presents the requirements analysis, namely, the application necessities, the system architecture, and the used technologies while Section IV describes the corresponding application development. The study to evaluate the performance of the

considered services is presented in Section V. Finally, Section VI concludes the paper and suggests topics for further works.

RELATED WORK

There are a myriad of Web services and Applications Programming Interface (APIs) available on the Web providing a wide range of different services. Usually, there is a substantial semantic overlapping among them where many Web services provide essentially the same functions. Such overlapping functionalities enables redundancy in the Web service ecosystem and give developers the opportunity to migrate from an API to another when the API originally used becomes unavailable or insufficient for their needs. To support the discovery of similar APIs, several new methods are being developed for a Web service discovery, including query-based methods relying on keywords and identifiers [7-8], clustering [9], and more detailed structure matching [10-11].

RESTful Web services are gaining more and more approaches. They are used as APIs in Web 2.9 services and are considered a more flexible and lighter-weight alternative to the so-called Big Web Services [12]. Much research has been performed in the field of developing RESTful applications. Richardson and Ruby [13] provide best practice examples and hints on how to develop RESTful applications.

Riva *et al*. [14] investigate how to apply the REST principles to the design of mobile services. They identified several issues such as latency and data format that need particular attention when applying REST concepts to mobile environments. However, they only focused on consuming RESTful Web services on mobile devices and did not address the provision of Web services from a mobile host.

In the domain of model driven development, Laitkorpi *et al*. [15] propose a process for designing RESTful services that focuses on a model based identification of the resources and on generating corresponding Web Application Description Language (WADL).

Although AMQP specification is not finalized yet, several products supporting different drafts of AMQP already exist, as Red Hat, VMware Ltd, the OW2 Consortium who use the 0-9-1 version of AMQP, the Apache Software Foundation, and the Sormmq who use the 0-10 version.

a. Representational State Transfer (REST)

Representational State Transfer (REST) architecture style behind the Web enlarged recognition as another way to develop Web services. RESTful Web Services are earning more and more approaches. They are used as application programming interfaces (APIs) for Web Services 2.9. RESTful Web Services technical topics become popular because the REST style includes a global identifier of all resources (e.g., a uniform resource identifier) and the customer only need to know this handle and the action required. He must also know the right format of representation, which is typically an HTML, eXtensible Markup Language (XML), or JavaScript Object Notation (JSON) meta-data. RESTful Web services (REST APIs) specify a set of resources, which includes three components: the URI of the Web service, the data type supported (JSON, XML, etc.), and the support operations through HTTP methods.

Previous Web applications access methods using HTTP operations (such as GET and/or POST). On the contrary with this, RESTful applications use methods according to the following functions: create, read, update, and delete (CRUD) style using the full range of HTTP methods (GET, POST, PUT and DELETE).

Figure 1 shows the architecture of RESTful Web Services. Client communicates with server through a uniform interface and during the stateless communication; client and server swap features depictions. Therefore, the REST design restrictions supply a standardized method to develop an API wearing the HTTP protocol.

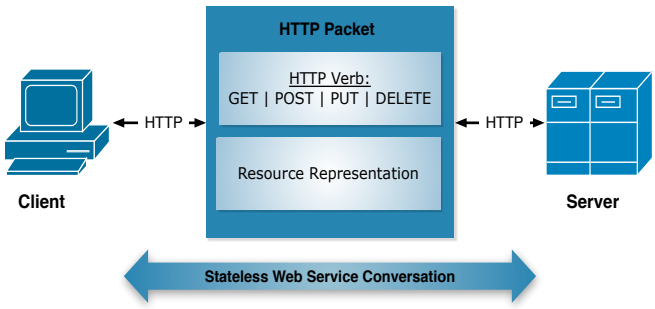


Figure 1. Illustration of a RESTful Web service architecture with client-server interaction

RESTful Web Services can be depicted over the Web Applications Description Language (WADL). A WADL include information about the requirements that can be addressed to a service involving the service uniform resource identifier (URI) and service data waiting and serves.

b. Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol (AMQP) is an open standard message middleware. According to the standard AMQP, middleware products written for different platforms and in different languages can send messages from one to another. AMQP is supported by a good number of key players, including Cisco Systems, Credit Suisse, Deutsche Borse Systems, Goldman Sachs, JPMorgan Chase Bank, Red Hat, and 29West.

AMQP enables applications to send and receive messages. In this regard it works like instant messaging or eMail. AMQP differs enormously from other available solutions because it allows the specification of what messages can be received and from, and how trade-offs are performed with respect to security, reliability, and performance. Systems built to integrate AMQP perform much better at functioning unattended or "lights-out" than other solutions. There are several reasons to choose the AMQP over the competition, including convenience, the possibility to connect applications on different platforms, the possibility to connect business partners using a full featured open standard, and a position for innovation built upon the foundations of AMQP.

Although AMQP specification is not finalized yet, several products supporting different drafts of AMQP today exist, as Red Hat, VMware, Ltd, and OW2 Consortium who use the 0-9-1 version of AMQP, the Apache Software Foundation, and the Sormmq who use the 0-10 version. It is used to simplify critical tasks, for example, JPMorgan reported a AMQP environment support 2,000 users on five continents to process 300 million messages per day. Every products that are listed comes with client library for different programming language, such, C + +, Ruby, Java, and Python.

For performance studies, this paper will consider the RabbitMQ that supports the standard AMQP Protocol. RabbitMQ is an open source message broker and queuing server that can be used to let disparate applications share data via a common protocol, or to simply queue jobs for processing by distributed workers.

RabbitMQ server is written in Erlang and is created on the Open Telecom Platform framework for failover and clustering. The main characteristics of RabbitMQ project include the following: *i*) the RabbitMQ exchange server itself; *ii*) gateways for HTTP, STOMP, and MQTT protocols; *iii*) AMQP client libraries for Java, .NET Framework, and Erlang; and *iv*) a plug-in platform for custom additions, with a pre-defined collection of supported plug-ins. Figure 2 presents the basic process of messages exchange using RabbitMQ.

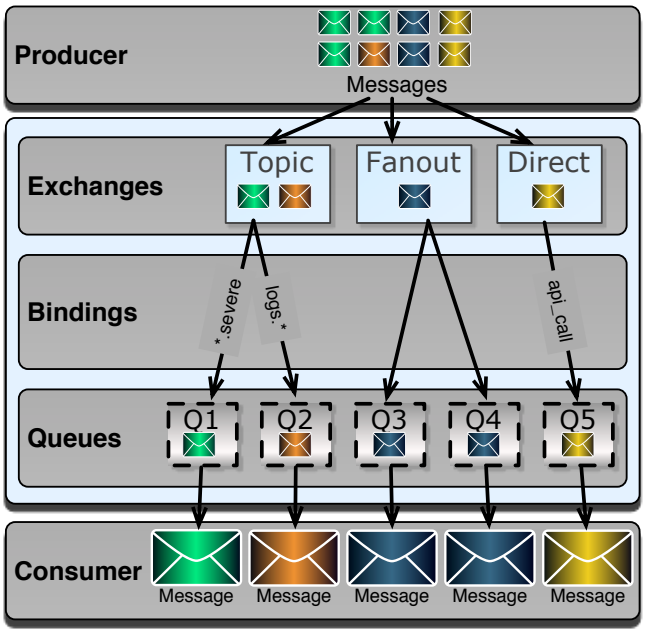


Figure 2. Illustration of the messages exchange process using RabbitMQ.

System Development

This section describes the application requirements and the system architecture, addressing also the used technologies.

c. *Requirements Analysis*

Nowadays, Web services are increasingly included in applications and consume information regardless of user location. Web Services triumph the goal with a technologically neutral way, which delivers interfaces clearly defined for dispersed features, which are not dependent of the operating system, hardware platform, and programming languages. Then, dispersed resources or services that can run in different hardware platforms, on various operating systems, or even can be written in various programming languages communicating through Web services interfaces.

Figure 3 describes a request workflow and sends processes through Web services. When a node needs to send or receive information, the requests go directly to the database. The requester or sender makes the solicitation in REST, the server will unfold the request to whether it is a request for data or to data store, and if it was a request for data, the server sends the data directly from the database, otherwise if it was to store data, the server will save the data directly in the database.

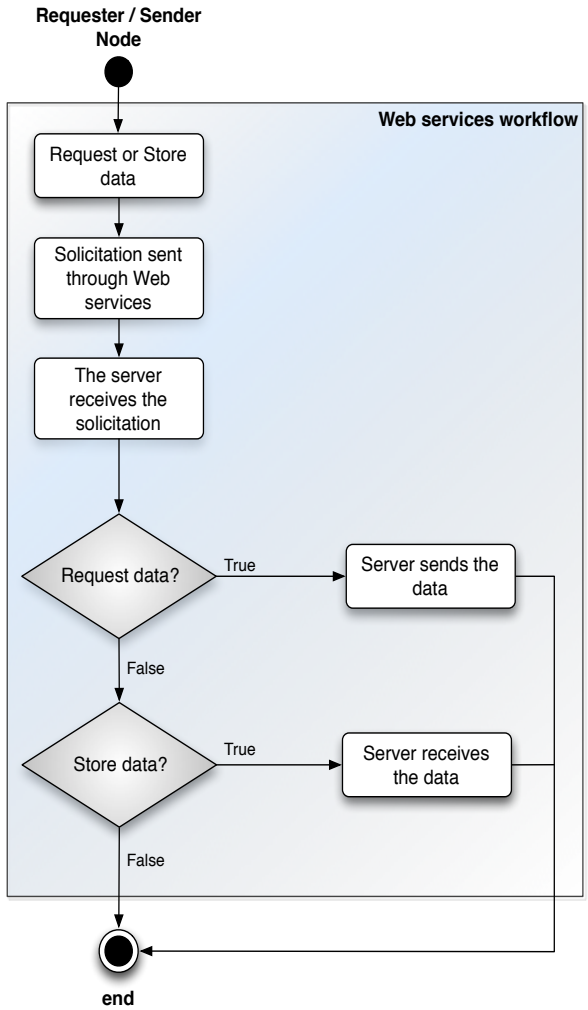


Figure 3. Request or Store workflow of the Web services activity diagram.

Figure 4 shows the activity diagram of the RabbitMQ request and data sending. All the solicitations should go first through the RabbitMQ before reach the database either sending or receiving. When a node sends information the RabbitMQ will put the message in a stack, then, when a request is made it will go directly to the stack. Messages will be stored in the RabbitMQ until they are consumed.

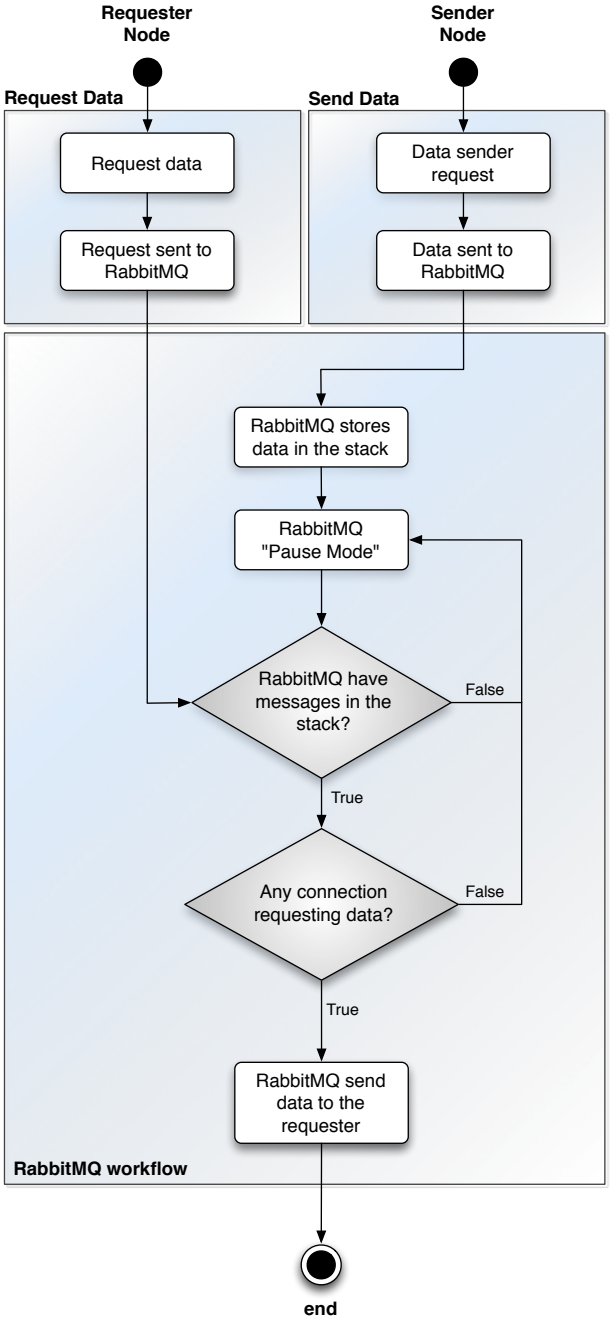


Figure 4. Request and sender path to the RabbitMQ activity diagram.

d. *System Architecture*

Figure 5 shows a scenario that illustrates the global system architecture. A user can choose messages to send using a Web service or a service of message queuing. If the user chooses messages submission through the Web service, it will automatically store the messages in the database. On the other hand, if a user chooses messages submission through message queuing service it will be necessary to use a back-end service to pick up the messages to the message queuing service and send them to the database.

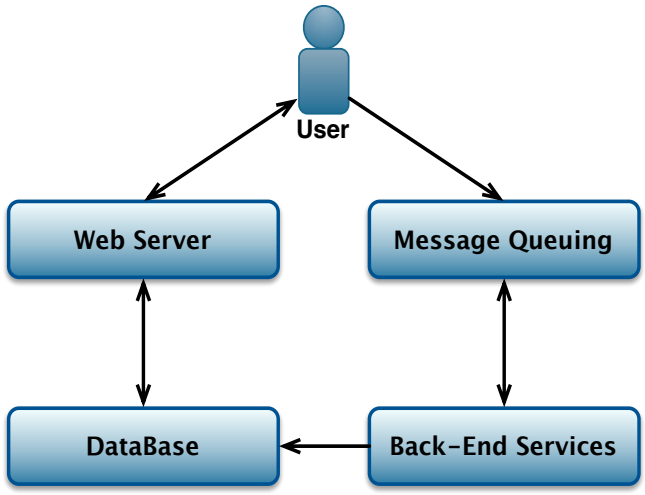


Figure 5. Illustration of the global system architecture with a database, a Web server, back-end services, and a message queuing.

In system architecture the Web server runs a RESTful Web service architecture built on java language, which an user communicates to send and receive data, i.e., the user makes HTTP requests to the Web service in order to call the method needed to insert or query data. With regard to Message Queuing protocol has been implemented AMQP (Advanced Message Queuing Protocol), which requires the installation of a RabbitMQ server. The back-end service was developed in Java and was used to send and get messages from RabbitMQ server and insert them into the database. Finally, the database was developed in MySQL language and was used to hold all the system data. The user was a mobile application that communicates with the Web Service through HTTP requests and makes requests through the AMQP port to communicate with the service for message queuing.

e. *Used Technologies*

For the proposed system several technologies were used. The Netbeans IDE was used to develop and execute the services for sending and receiving messages. The Web service was developed in Java and runs on Glassfish, an open-source application server. For the AMQP protocol, the RabbitMQ server and an open source message broker software were used, and also developed a Java application to publish and consume messages from the server. The database where messages were stored was created using a relational database management system, the MySQL.

WEB SERVICES MECHANISMS AND AMQP PROTOCOL FOR MOBILE APPLICATIONS

Mobile platforms present their own set of challenges that can be identified as follows: CPU availability, memory and bandwidth, storage capacity, connectivity options and issues, security and user interaction. As the capacity of storage and processing on mobile devices is reduced in order to find the best solution for mobile applications that generate large amounts of data or many messages should be saved, other services to send and store messages outside the device should be used.

Against this problem, increasingly, the option comes with the use of Web services to send data from the mobile device to a server. However, in this case, it is necessary that a Web service accepts a large number of requests per second, assuming the system will process all of them. Once it receives a large amount of data per second in order to process them and store in the database, if the system fail to manage the requests, messages may be lost while the system recovers because the mobile device is not ready to temporarily store the messages in case of service failure. Despite having been implemented in mobile application solutions to solve the problem, it should not be a good option given the small memory of mobile devices.

To avoid these situations a RabbitMQ server was proposed. Through it, the data is sent to server queues and the back-end service should get the messages. In this case even if the back-end service fails, the messages are stored on the RabbitMQ server and it will not be needed that the mobile device temporarily store messages and avoids that some of them may be lost during the time that the service is down.

The Web service and RabbitMQ server were developed separately in order to compare both of them and get more reliable results.

SYSTEM EVALUATION AND VALIDATION

The RabbitMQ server, the Web server, and the database were created on a Mac server over a virtual machine running Windows 7 Professional operating system with a processor Inter (R) Xeon(R) CPU 2.67GHz and 1GB of RAM memory.

The user application was designed for mobile devices running the Android operating system. The application will run from the 2.0 version (Eclair) of Android and higher.

Several experiments were performed and deployed for this study in order to compare both deployed services. With these experiments the authors tried to find out the main advantages regarding the use of a Web service, compared with the use of Message Broker open source software, or vice versa. In these experiments, twenty-four handsets were used continuously to send messages to servers. The experiments were performed as follows:

Experiment 1 – the user application sends messages for 30 minutes to RabbitMQ using AMQP protocol without no active consumer (back-end service);

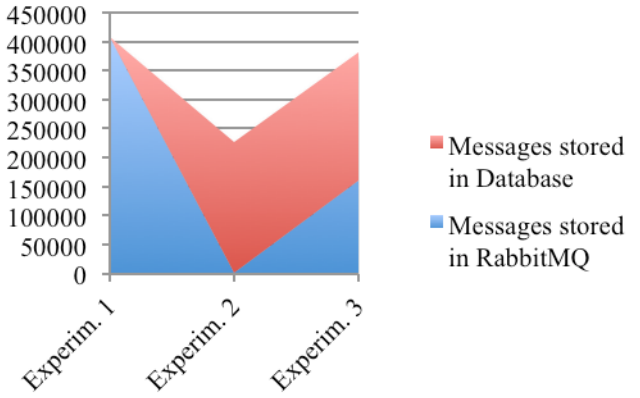
Experiment 2 – the user application sends messages for 30 minutes to the Web service server using HTTP protocol, and then, the Web service communicates with the database using JDCB driver to store the messages;

Experiment 3 - the user applications sends messages for 30 minutes to RabbitMQ using AMQP protocol with a consumer (back-end service) to read the messages, process them, and through JDBC driver connect to the database to store the messages.

In these experiments, the user had a single function to send data to the service without ever doing queries. In this way the authors were able to obtain a more reliable comparison of the two services when many clients are constantly sending data. The results are presented in Table I.

RESULTS OF THE PERFORMED EXPERIMENTS.

	Users	Messages stored in RabbitMQ	Messages stored in Database	Average messages sent per second
Experim. 1	24	407793	n/a	226.6
Experim. 2	24	n/a	226530	125.9
Experim. 3	24	160747	219907	211.5



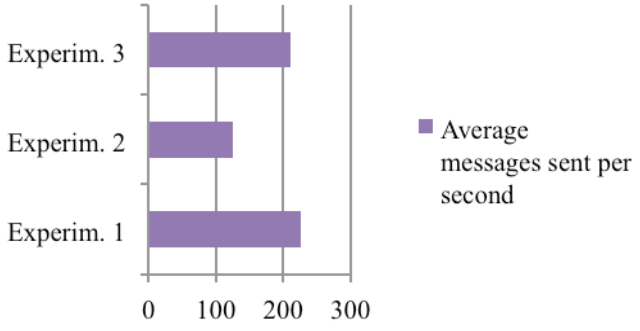


Figure 6. Performance comparison considering the number of message for 3 different experimental scenarios showing the messages stored in the Database, RabbitMQ, and the average messages sent per second.

As above-mentioned, the objective of the experiment 1 passed through the objective to send messages to the RabbitMQ, during 30 minutes, using 24 users. As it can be observed in Table I and Figure 6, in the experiment 1, a total number of 407793 messages were sent and stored with an average of 226.6 messages per second. In the experiment 2, in the same interval of time as the experiment 1, but now with the objective to sent messages through Web services and storing them in the database, a total of 226530 messages were stored with an average of 125.9 messages per second. Finally. In the 3rd experiment, the same interval of time was used for the experiments 1 and 2. Its objective includes sending and storing the messages using the RabbitMQ, a total number of 380654 were sent and, after, the time was over a total of 160747 messages stayed stored in the RabbitMQ. A total number of 219907 messages where stored in the database, presenting an average of 211.5 messages per second. None of the messages in the experiment 3 where lost, some of them stay in the RabbitMQ because it takes extra time to the client in order to obtain the messages from the stack and put them in the database.

With these experiments it is possible to conclude that when the AMQP protocol is used to exchange messages, it will send a larger number of messages per second, as can be seen in the experiments 2 and 3. There is a big difference between RabbitMQ and the Web service since the Web Service is in charge of receiving the request introduced by the user and then, depending on the method called, save the message on respective table of the database. While RabbitMQ only sends messages to the server when the server asks for it and stores the messages in a queued from the client, the client connects with the RabbitMQ.

CONCLUSIONS AND FUTURE WORK

This paper proposed a comparison study between a RESTful Web service and the AMQP protocol for exchanging messages between clients and servers. As it can be seen in Section V, it is possible to conclude that applications which will exchange an large amount of data, the best approach is to use the RabbitMQ server and use a Back-End Service to consume the messages, process them, and send them to the database. This approach will allow resources saving, prevent data loss, and a better organization of the messages. As for the other approach, when massive data is exchange, this approach has shown that it can send fewer messages per second, increasing the time for the exchange, and will consume more resources.

As future work it is intend to join the RESTful Web service with RabbitMQ server in order to obtain the most effective results. With this solution it is planned that there is less coupling between the two services and allows better communication between them when the number of applications that consume the services increases.

I. ACKNOWLEDGMENTS

This work has been partially supported by the *Instituto de Telecomunicações*, Next Generation Networks and Applications Group (NetGNA), Portugal, by National Funding from the FCT – *Fundação para a Ciência e a Tecnologia* through the PEst-OE/EEI/LA0008/2011 Project, by the AAL4ALL (Ambient Assisted Living for All), project co-financed by the European Community Fund FEDER through COMPETE – Programa Operacional Factores de Competitividade.

II. REFERENCES

[1] Web Service Description Language, “Web Service Definition Language (WSDL),” [Online]. Available: http://www.w3.org/TR/wsdl [Accessed: February 2013].

[2] Roy Thomas Fielding, "Architectural styles and the design of network-based software architectures," PhD diss., University of California, 2000.

[3] X. Xiong and J. Fu, "Active Status Certificate Publish and Subscribe Based on AMQP," 2011 International Conference on Computational and Information Sciences (ICCIS 2011), Chengdu, China, 21-23 October, 2011, pp. 725-728.

[4] OASIS, “AMQP is the Internet Protocol for Business Messaging,” [Online] Available: http://www.amqp.org/ [Accessed: February 2013].

[5] J. O'Hara, “Toward a commodity enterprise middleware,” in Queue, vol. 5, no. 4, pp. 48–55, May, 2007.

[6] S. I. C. I. Vinoski, “Advanced Message Queuing Protocol,” 2006 IEEE Internet Computing, Las Vegas, NE , USA, 26-29 June, 2006, vol. 10, no. 6.

[7] Y. Wang and E. Stroulia, "Flexible interface matching for Web service discovery," 4th Int. Conf. on Web Information Systems Engineering (IEEE WISE 2003), Rome, ITA, 10-12 Dec., 2003, pp. 147-156.

[8] X. Dong, A. Halevy, J. Madhavan, E. Nemes and J. Zhang, "Similarity search for web services," in Proceedings of the Thirtieth international conference on Very large data bases (VLDB 2004), Toronto, Canada, 29 August - 3 September, 2004, vol. 30, pp. 372-383.

[9] R. Nayak and Bryan Lee, "Web service discovery with additional semantics and clustering," 2007 IEEE/WIC/ACM International Conference on Web Intelligence (WI *2007)*, 2-5 November *2007*, Silicon Valley, CA, USA, pp. 555-558.

[10] H. R. M. Nezhad, G. Y. Xu and B. Benatallah, "Protocol-aware matching of web service interfaces for adapter development," 19th international conference on World Wide Web, Raleigh, North Carolina USA, April 26-30, *2010,* pp. 731-740.

[11] R. Mikhaiel and E. Stroulia, "Examining usage protocols for service discovery," 4th International Conference of Service-Oriented Computing (ICSOC 2006), Chicago, USA, Dec. 4-7, *2006,* pp. 496-502.

[12] C. Fu, F. Belqasmi and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," IEEE Communications Magazine, vol. 48, no. 12 , 2010, pp. 92-100.

[13] L. Richardson and S. Ruby, “RESTful web services,” O'Reilly Media, 2008.

[14] C. Riva and M. Laitkorpi, "Designing web-based mobile services with REST," 2007 Workshops in Service-Oriented Computing (ICSOC 2007), Vienna, Austria, September 17-20, pp. 439-450.

[15] M. Laitkorpi, P. Selonen and T. Systa. "Towards a model-driven process for designing restful web services," 2009 IEEE International Conference on Web Services (ICWS 2009), Los Angeles, CA, USA, July 6-10, 2009, pp. 173-180.