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

I. 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.

II. 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.

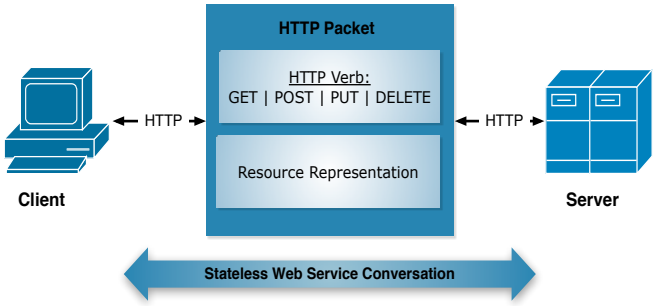

Figure 1. Illustration of a RESTful Web service architecture with client-server interaction

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.

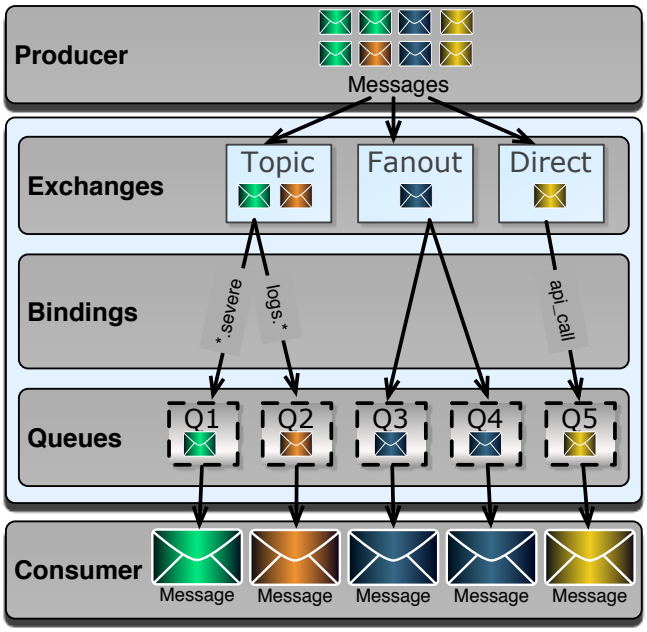

Figure 2. Illustration of the messages exchange process using RabbitMQ.

III. System Development

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

A. 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.

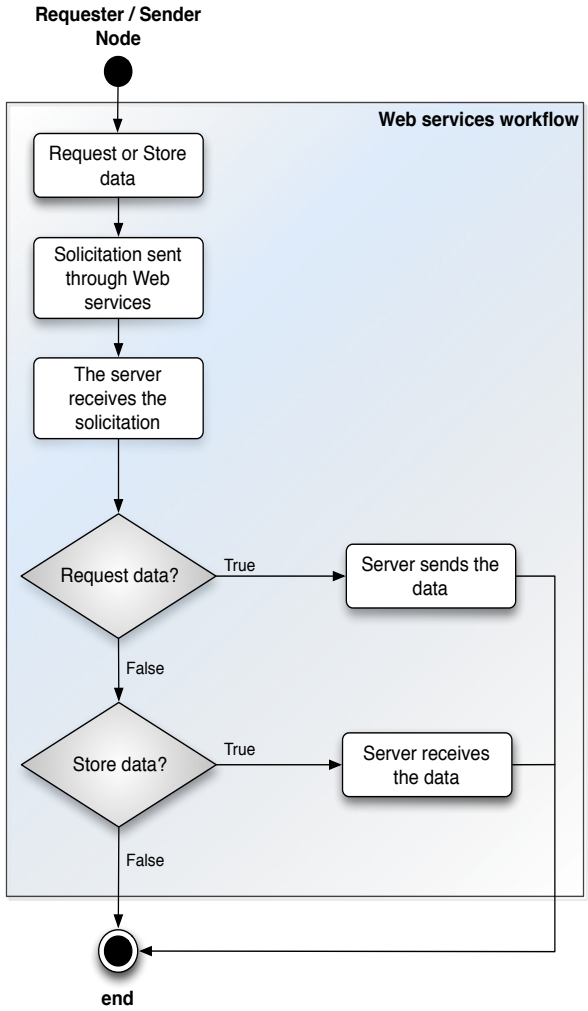

Figure 3. Request or Store workflow of the Web services activity diagram.

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.

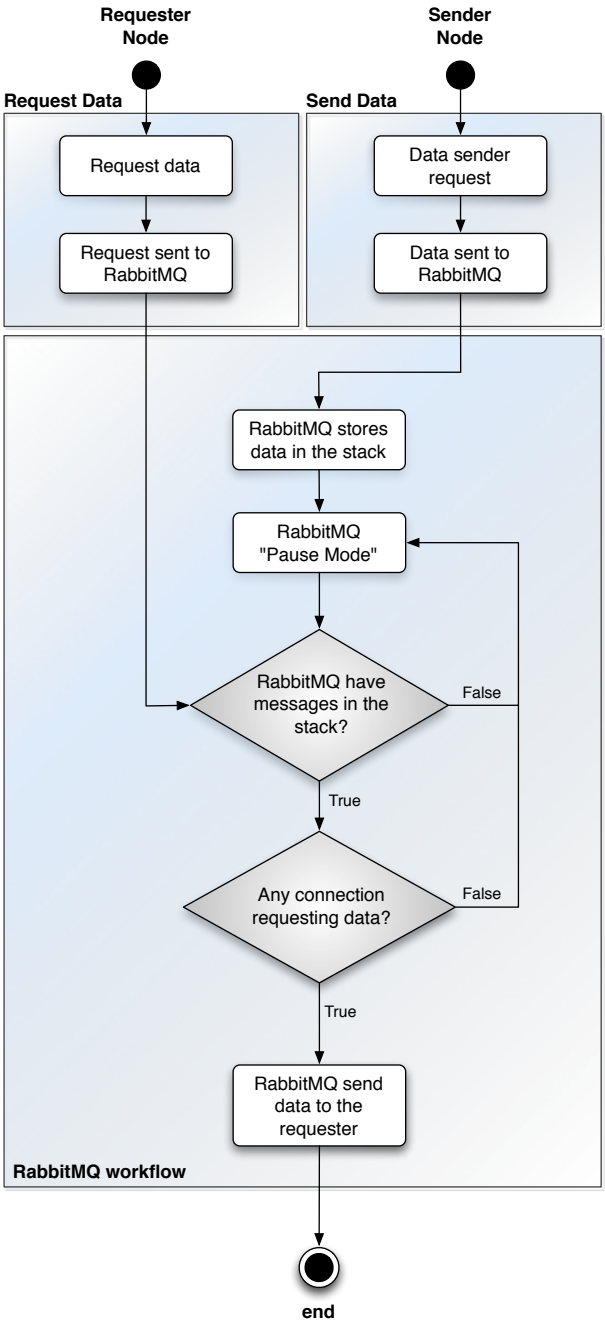

Figure 4. Request and sender path to the RabbitMQ activity diagram.

B. *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.

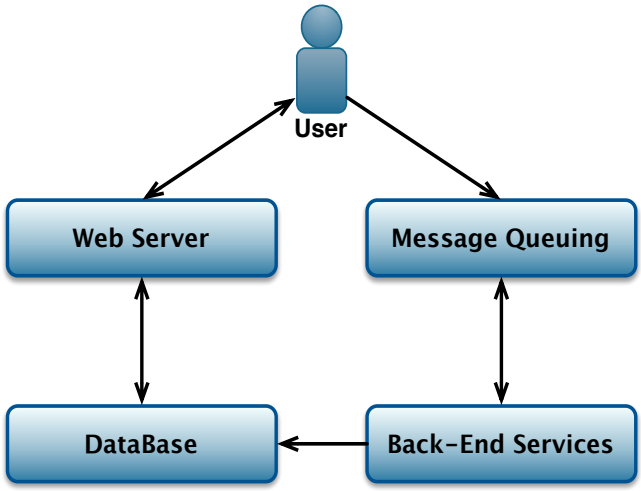

Figure 5. Illustration of the global system architecture with a database, a Web server, back-end services, and a message queuing.

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.

C. *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.

IV. 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.

V. 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.

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

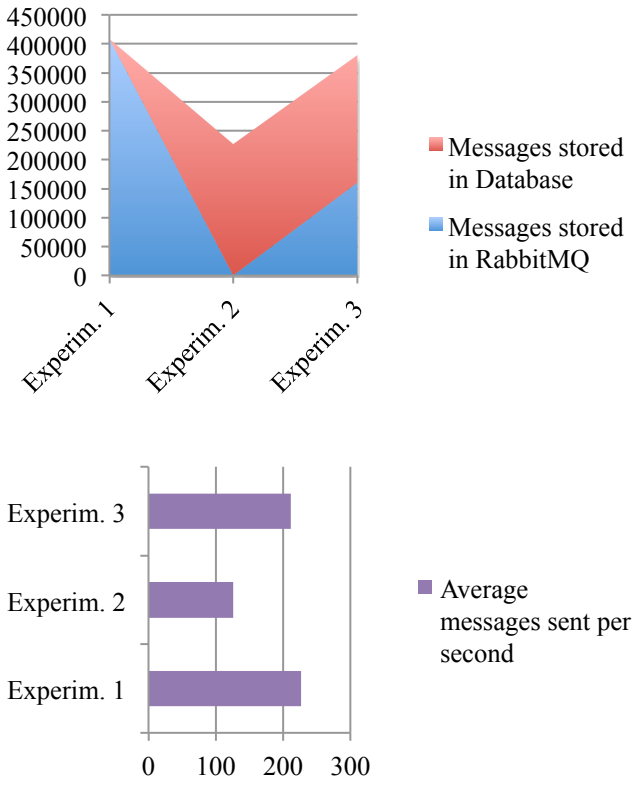

Figure 6. Performance comparison considering the number of message for 3 different experimental scenarios showing the messages stored in the Database, RabbitMQ, and the average messages sent per second.

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.

VI. 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.

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.

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.