# Asynchronous Message Exchange System between Servers based on Java Message Service API

EZIN C. Eugène[#] LALEYE A. A. Fréjus*
Institut de Mathématiques et de Sciences Physiques
Unité de Recherche en Informatique et Sciences Appliquées
BP 613 Porto-Novo, Université d'Abomey-Calavi, République du Bénin
[#]eugene.ezin@imsp-uac.org *frejus.laleye@imsp-uac.org

**Abstract— Nowadays, local area networks appear as efficient means to connect servers for exchanging information through a transmission channel between various heterogeneous components. The observed interruption during communication is sometimes caused by connection failures between components. In this paper, we propose an asynchronous message exchange system for data transmission between two servers based on Java Message Service application programming interface. The proposed system allows the data to be exchanged between servers even when the disconnection is observed in an asynchronous mode. This is accomplished thanks to the buffer implementation that stores messages on a local server when the remote server is not available or reachable. Simulation results are presented to show the effectiveness of the implemented system.**

**Keywords**-Java Message Service; Queuing system; Data exchange.

## I. INTRODUCTION

In many networked applications, client/server architectures [1] that are distributed information networks accessible from anywhere at any-time are involved to information transmission. The rapid growth of client/server technology is due to powerful technical and economic forces. Several technical trends are converging around client/server technology. These include faster, smaller and standardized computer components, improved distributed communications and increased information dissemination. These advantages make client/server architectures very attractive for organizations facing pressures of increased global competition.

The client-server architecture consists of a service requester machine called client and service provider machine called server. The services are exchanged between these machines using simple message passing [2]. These request/response protocols can be implemented over either connectionless or connection-oriented protocols. The messages passing are into synchronous and asynchronous.

Synchronous request/response protocols are the simplest. In these protocols, requests and responses are exchanged in a lock-step sequence. Each request must receive a response synchronously before the next request is sent. In synchronous message passing, the machines exchange messages with a predefined order and time sequences. Synchronous request/response protocols are appropriate in many contexts. For instance, when the result of a request determines subsequent requests e.g. an application that requires an authentification exchange won't send sensitive information requests until the security certificate exchange completes successfully. In [3], Lee presented a study of modification devices on non-native discourse based on synchronous online exchanges.

Asynchronous request/response protocols stream requests from client to server without waiting for responses synchronously. Multiple client requests can be transmitted before any response arrives from the server. Asynchronous request/response protocols therefore often require a strategy for detecting lost or failed requests and resending them later. Asynchronous request/response protocols are appropriate in many contexts. For instance, a response is not required before deciding on subsequent requests as it is the case with Web browsers that use asynchronous strategies to fetch multiple embedded images from the same server. Since requests are independent, they can all be sent asynchronously without waiting for the intervening responses. Each response includes information the browser can use to match it to the corresponding request even when the order of responses is different from the one in which the requests were sent. When communication latency is high relative to the processing time required for the request, asynchronous request strategies help to make efficient use of the network, reducing the effects of high latency. The resulting performance improvement greatly out weights the extra complexity of associating responses with requests and correctly implementing retry strategies. In [3], Hsiao et al., proposed an asynchronous message exchange system on CORBA[a].

The distributed system overcomes from the complexity expansion of information systems and their transparency due to the heterogeneity. One or more distributed systems across multiple sites are involved in communication through a network. One of the main features of such a distributed architecture is the permanent

mode to exchange information between distributed applications in order to ensure data consistency across the network.

The permanent exchange of information in a network is often subject to problems such as the downtime and the unavailability of the communication systems.

Thus, intermittent connectivity failures prevent the exchange of messages between communicating systems.

We consider in this work a communication between two servers. We assume that the local server, server *A* must exchange information with a remote server *B*. In our work, we assume that the remote server may be unavailable. This can affect the process such that the communication becomes impossible. That will paralyze the server activity.

We then present in this paper, a message passing system-based on the *Java Message Service application programming interface*, that ensures the message is really transmitted and received by application at the remote server side. The two servers are loosely coupled to communicate in asynchronous mode in such a way that one server does not depend on the second one during the message exchange process. The developed prototype is implemented with Java Message Service API and is used with other tools such as the Enterprise Java Beans, Java Persistence application programming interface, the glassfish server, the relational database management system *MySQL*, the *JavaCv*, the database management system *MySQL*.

The paper is organized as follows: Section 2 presents the basic concepts about message exchange systems. Section 3 describes the Java Message Service application programming interface whereas Section 4 presents the full description of the proposed prototype and its architecture. The implementation procedure of the proposed system is presented in Section 5. Section 6 presents the simulation experiments we have done to show the effectiveness of the proposed system. Finally, we present the conclusion and further works in Section 7.

## II. BASIC CONCEPTS

To better understand the asynchronous message exchange system based on Java Message Service, it is worth clarifying some concepts, including exchange systems and queuing systems. Let us first define the asynchronous approach to message exchange system as mentioned in the introduction.

### *A. Message exchange system*

A message is a sequence of characters containing data allowing a sender to send information to a recipient. Often encoded as a text or an object type, a message consists of a header, the format and the body. A message header typically contains information about the message length, the sender, the shipment date, or the message object such as a request, a response, an unanswered sending, etc. The message format specifies the message type. The message body contains information to be transmitted from the sending application to the receiving application and vice versa [4]. The XML technology can also be involved in defining the data format as in [5, 6].

A message exchange system can be defined as a set of used techniques that enable objects to communicate by sending and receiving messages through a network. In any message exchange system, two forms of the exchange process appear that are inter-personal exchange and inter-application exchange.

The inter-personal exchange, also called messaging, is the message exchange between people.

The inter-application exchange is the communication between two or more applications.

### *B. Asynchronous approach of message exchange system*

Asynchronous messaging is a communication method wherein the system puts a message in a message queue and does not require an immediate response to continue processing. Unlike the synchronous exchange, the asynchronous exchange does not require the availability of communicating systems. It uses queues to store messages before forwarding them to the destination. By doing, it ensures the decoupling of the systems and warranties the integrity of the exchanged messages. Asynchronous approach of message exchange system has many advantages over synchronous approach, but is limited when routing messages.

### *C. Queuing message*

Queuing is the mechanism by which messages are held until an application is ready to process them. Queuing allows users to:

- communicate between programs (each might be run in different environments) without having to write the communication code;
- select the order in which a program processes messages;
- balance loads on a system by arranging for more than one program to serve a queue when the number of messages exceeds a threshold;
- increase the availability of applications by arranging for an alternative system to serve the queues if the primary system is unavailable.

Therefore, a queue manager is a system program that provides queuing services to applications. In other words, It provides an application programming interface so that programs can put messages on a queue, and get messages from a queue. A queue manager provides additional functions so that administrators can create new queues, alter the properties of existing queues, and control the operation of the queue manager.

Some benefits for application designers and developers of message queuing are given below:

- the possibility to design applications using small programs sharable between many applications;
- the possibility to quickly build new applications by reusing these building blocks;
- applications written to use message queuing techniques are not affected by changes in the way that queue managers work;
- there is no need to use any communication protocol. The queue manager deals with all aspects of communication;
- programs that receive messages need not be running at the time that messages are sent to them, messages are retained on queues.

In the asynchronous exchange mode, a queue is also called a buffer and its role is to store messages in order to allow applications to work in their own space without being dependent on each other. The maximum length of messages a queue can hold depends upon the queue size. One of the major functions expected from a queue is to secure messages by maintaining message integrity regardless the state of the destination application. The latter involves the shutdown of the destination application, the physical destruction of the medium containing the queue [7].

### *D. Different modes of message exchange systems*

When designing application to asynchronous message exchange system, one should take care about the message transmission mode. Indeed, there exists many modes including the simple transfer, the dialog mode, the ventilation, the chaining and the client/server mode.

- *The simple asynchronous transfer mode* (ATM): The ATM is a set of protocols used for computer networks which are mostly used for wide area networks (WAN). For short, the asynchronous transfer mode is a network technology designed for integrated services networks to carry multimedia data as well as conventional computer data traffic [1]. It splits the data, and encodes them into packets of axed size. Each packet can be routed differently, along virtual paths set up in the network. In this mode, the message exchange system is completely asynchronous since the transmitter does not wait for a message response from the recipient. Therefore a single queue is used between the two communicating systems. In [8], Bruce et al. examined the quality of service and asynchronous transfer mode in IP Internet works.
- *The dialog mode*: In a dialog mode, the receiver is constrained to send a response message to the sender. Two queues are used during the implementation especially one queue for the message to be transmitted and a second one for the receiver response.
- *The ventilation mode*: The ventilation mode introduces parallelism. An application sends several messages in parallel through different queues. Each queue is managed by a different application. This application can then send a response message to the sending application.
- *The chaining mode*: In the chaining mode, the treatments are done in batch. This is the mode that links together several distinct and successive treatments and store results in a queue. These results are used by another application which sends its results to another queue by creating a finite chain.
- *The client/server mode*: In the classical client/server mode, there is a transmitter called client and a receiver called server. The client sends a request to the server and the latter must answer. With the use of the queue, one can implement a server that responds to multiple requests from different clients. The clients send their messages as requests into the queue with the name of the queue to serve as a response to the request. The server listens to the queue, retrieves messages, performs the processing and returns the answer to the specified queue [4].

## III. JAVA MESSAGE SERVICE

Java Message Service [9,10] application programming interface is an asynchronous communication mode to exchange messages implemented in Java. It allows an application to invoke the services of a middleware oriented message. JMS is part of Java Enterprise Edition and thus is available to applications running on Java servers. The first version of JMS namely JMS 1.0.2b was released on June 25th, 2001. The second version JMS 1.1 was

released on March 18th, 2007, showing no significant difference. As any specification, JMS ensures that all applications have the same behavior regardless of the provider implementation. JMS offers two communication modes: queues and topics.

We used the JMS queue mode to implement the prototype of the proposed system. In this communication mode, the exchange takes place between two remote servers. The sending server sends messages in a queue while the remote server, once connected, consumes messages from the queue that are then automatically deleted from the queue. With Java Message Service, we provided the developed system with some specifications to ensure a weak coupling between the servers. So doing, we ensured a fully asynchronous exchange with a reliable service in delivering messages and a persistence in the stored message.

The JMS library is provided in the package textitjavax.jms and includes all necessary interfaces needed by the API for full functionalities. All Java Enterprise Edition application servers since version 1.4 should provide a JMS service. Some JMS libraries are open source namely Apache ActiveMQ, OpenJMS, JBoss Messaging and JBoss HornetQ, ObjectWeb JORAM OW2 Open Message Queue, etc. Commercial librairies are BEA Weblogic, Oracle AQ, SAP NetWeaver, SonicMQ, TIBCO, webMethods Broker Server, WebSphere MQ, Fiorano FioranoMQ, etc.

### *A. Encoding messages*

Java Message Service is used to encode the message body into five different types labeled textMessage, bytesMessage, mapMessage, streamMessage and objectMessage.

- The *textMessage* is the message body that contains a sequence of characters.
- The *bytesMessage* is the message body that contains a sequence of bytes according to the Java language.
- The *mapMessage* is the message body that contains a data type to connect a key (String encoded) to a value (coded String, Double or Long).
- The *streamMessage* concatenates several native types (String, Double or Long).
- The *objectMessage* allows users to send a Java object.

### *B. Message structure*

A message exchanged over a network consists of a header, body and ints property. The header contains the necessary fields for the identification and routing of the message. The two main fields are:

- the *JMSMessageID* that represents the unique identifier of a message.
- The *JMSDestination* is a mean to identify the queue, which is the destination of the message.

The properties are added to header with specific attributes.

## IV. DESCRIPTION OF THE PROPOSED PROTOTYPE

The proposed solution for the asynchronous exchange system presented in Fig. 1 involves two servers namely a local server and a remote server. The local server sends messages to the remote server even if the remote server is unreachable due to cable connection or other connection problems. It allows in this mode users to continue using the system when instant confirmation is not required. Once the remote server is connected, all messages in the queue are then transmitted on it.

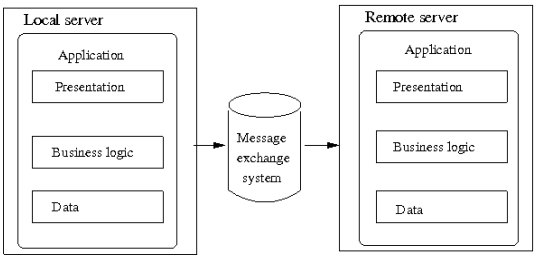


Figure 1. Proposed asynchronous message exchange system.

### *A. Functional description of the proposed asynchronous exchange system*

Once logged into the system, the user can choose between sending data as a text and sending the large data file. This distinction is introduced due to the data size since they do not use the same line during the exchanging process. Fig. 2 shows all the transactions of the message exchange system through the sequence diagram.

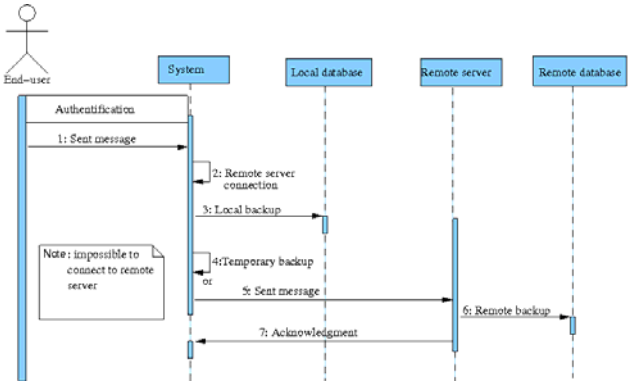


Figure 2. Functional description of an asynchronous message exchange system.

*1) Connection of the remote server onto the network*

When there is no connection problem between the two servers, the message exchange is done in real-time. But in asynchronous message exchange mode, the following steps present the procedure as in Fig. 2.

(1) The user sends a message from the local server to the remote server.

(2) The message is retrieved by the system. The latter attempts to connect to the remote server while the connection is on.

(3) The system makes a backup of the message in a local database.

(4) The system routes the message to the remote server.

(5) The message is persisted in the remote database.

(6) An acknowledgment note is returned to the system, which warranties that the message is actually sent.

*2) Disconnection of the remote server from the network*

Despite the disconnection of the remote server from the network, the local server continues to send messages.

(1) The user sends a message from the local server to the remote server.

(2) The message is retrieved by the system that attempts to connect to the remote server while the connection is off.

(3) The system is unable to connect to the remote server.

(4) The system stores the message onto its buffer.

*3) Reconnection of the remote server onto the network*

The remote server consumes messages automatically sent by the local server when it was out of the network.

(1) The system sends a connection request to the remote server.

(2) The system connects to the remote server.

(3) The system sends the stored messages during the disconnection to the remote server.

(4) The messages are persisted into the remote database.

(5) An acknowledgment note is returned to the system which ensures that the messages are actually sent.

*B. Plateform environment*

The prototype of the asynchronous message exchange system is developed in Java. This choice is justified by the various benefits of this technology and also motivated by the following reasons:

- Java Message Service application programming interface is a Java language specification.
- Java language is powerful and robust.
- Java portability to other platforms can be without any concern about system integrations.

*C. Proposed architecture*

The overall architecture of the asynchronous message exchange system that involves a local server and a remote server is presented in Fig. 3. All stages through which a message must be sent from the local server to the remote server are represented. The message is constructed in step 1 and is sent to the system. At step 2 the system checks the message structure and encoding. A local backup of the message is made at step 3. The asynchronous exchange is performed in steps 4, 5 and 6. The message is first placed in a queue before being delivered to the module that routes it to the remote server. At the step 6 the system sends connection request to the remote server. If the remote server is not connected into the network, the message is stored in the queue at step 4. Otherwise the message is delivered to the recovery module. The remote database then retrieves the message in step 8 and messages are received at step 9.

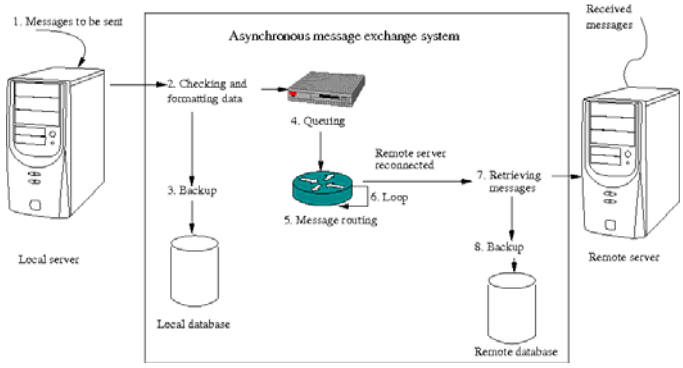


Figure 3. Global architecture of the proposed asynchronous message exchange system.

*D. Implementation of the prototype*

The implementation of the asynchronous message exchange system concern steps 4, 5, 6 and 7 as presented in 3. The different used databases were developed with the relational database management system MySQL. First, we present the procedure to create, send and receive a message from the JMS API.

## V. IMPLEMENTATION OF THE PROTOTYPE

The implementation of the asynchronous message exchange system concern steps 4, 5, 6 and 7 as presented in section III. The different used databases were developed with the relational database management system MySQL. First, we present the procedure to create, send and receive a message from the JMS API.

*A. JMS procedure to send/receive a message*

Fig. 4 presents the different steps for creating, sending and receiving a message from the JMS API. These steps are summarized in the following way:

- recovering a connection factory.
- opening a connection.
- opening a session.
- creating a message consumer or producer.
- sending or receiving a message through the queue.

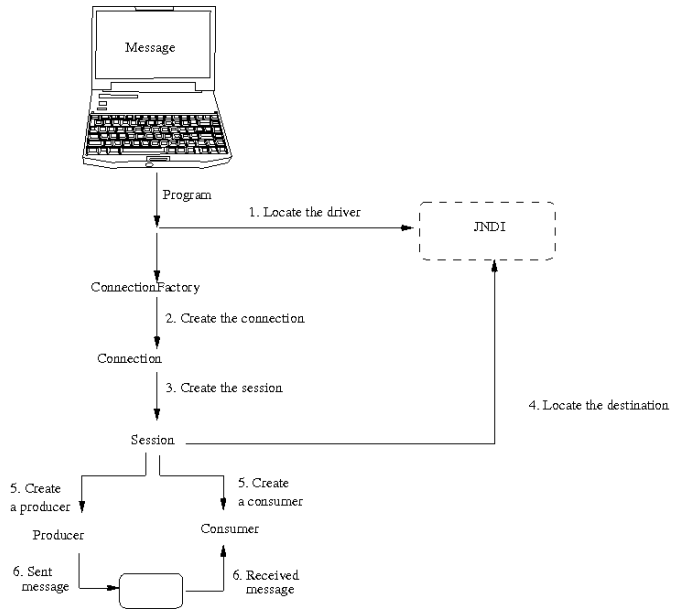


Figure 4. Process of sendind and receving messages

(1) locate the driver : At this level, the system searches for the JMS driver to send the message. We use a *ConnectionFactory* as driver. With this object of factory type, we return the object that can allow the connection opening.

(2) create the connection : The *QueueConnection* interface is implemented to create an object of connection type to connect to broker messages. The class that implemented the *QueueConnection* interface exchanges with the broker to start the connection that can carry messages.

(3) create the session : We implemented the *QueueSession* interface to allow transactions of sending and receiving messages. An object of session type is created for a given connection. With the object of session type, we create a message, validate or invalidate a transaction.

(4) locate the destination : Here, we must find the channel that will be used to transport the messages. It is obtained through Java Naming and Directory Interface.

(5) create the producer or the consumer : The producer creates an object that can send a message to a destination. The *MessageConsumer* interface allows us to create a consumer which works in asynchronous mode to receive messages. A listener is registered with the aim to serve as the events manager at the arrival of a new message.

(6) sending or receiving messages : At this stage, messages are transmitted or received through the transmission channel.

*B. Exchanged message format*

The user can send information as plain text or large objects such as files, images, etc.). The information is then encapsulated into an object of the type *javax.jms.Message.*

*C. Description of the implemented database*

On each server, a database named base message was implemented in console mode with the database management server MySQL. The implemented database has two tables. The table text contains the message information as plaintext and the table object contains objects of large size to transmit.

*D. Message routing*

This module in the proposed system is responsible for forwarding the messages received from the local server to the remote server and vice versa. It sends by loop action, the connection requests to the remote server. After sending these requests, an OK response is returned to the system if the remote server is connected to the network. Then the system uses the message routing service to transmit messages to the remote server. If the remote server is not connected to the network, the routing module ensures that messages are stored in the queue until another OK response from the remote server is effective.

*E. Presentation of involved tools*

Many tools are used to achieve the system namely the Java enterprise edition, the EJB, the JPA, the JMF, the SGBD MySQL, the Glassfish server, the JavaCv.

- The Enterprise JavaBeans or EJB for short, is the server-side component architecture for Java Platform that enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.
- The Java Persistence API or JPA is the standard application programming interface for the management of persistence and object/relational mapping. It provides an object/relational mapping facility for application developers using a Java domain model to manage a relational database. The JPA is part of the Java Enterprise Edition platform.
- The Java Media Framework or JMF is the java library that enables audio, video and other time-based media to be added to Java application and applets.
- The database management system MySQL is used to implement the relational database with the two tables.
- The Glassfish server is a Sun Java System Application Server, which implements the newest features in the Java Enterprise Edition platform.
- The JavaCv provides wrappers to libraries in computer vision.

## VI. SIMULATION

We simulated the implemented system by carrying out two experiments where both used the application we developed. The form we used to test the system is presented in Figure 5.

(1) The first experiment consisted in sending data from the local server to the remote server when the two servers were well-networked. This experiment was carried out to check the network functionality and the implemented system. We found that data were sent without any problem.

(2) The second experiment consisted in simulating the disconnection of the remote server from the network. The local server continued to send data to the remote server. The data were stored in the implemented buffer on the local server. When the connection was back, the data were transferred onto the remote server with an acknowledgment note.

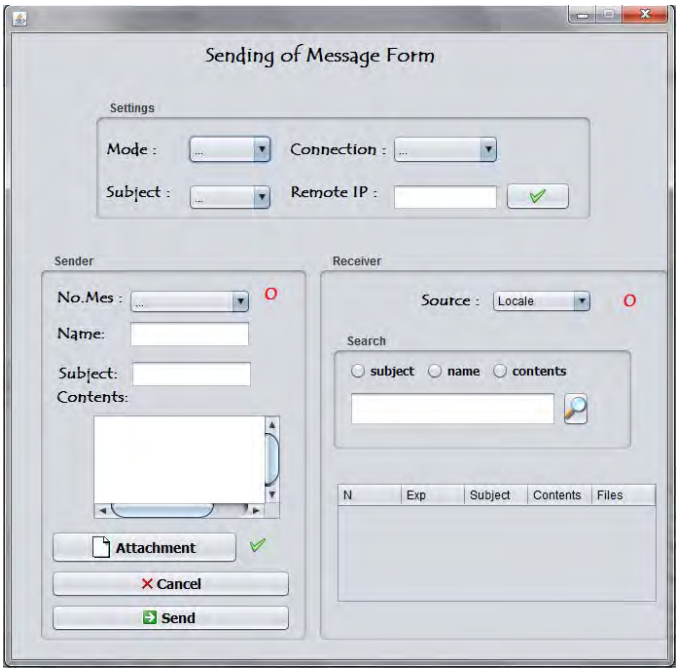


Figure 4. Graphical user interface form to send data.

## VII. CONCLUDING REMARKS

The asynchronous message exchange system between servers we proposed is able to transmit data from a local server onto a remote server even though the disconnection of the remote server from the network is

observed. This is done thanks to the use of Java Message Service Application programming interface and the buffer we implemented. It is worth mentioned that the cryptosystem we proposed in [11] can be used to secure message to be transmitted.

The next step is to associate a priority to each message in the queue. Such a priority could be useful for asynchronous message processing on the remote server in some cases.

Another important topic could be how to implement the asynchronous message exchange in grid computing environment using Java. One can based our analysis on the work done in [12] and in [13].

## ACKNOWLEDGMENT

The authors are grateful to Professors Eric Leclercq and Kokou Yetongnon for their fruitful comments and collaboration. We also appreciate the help of Prof Tchantcho Bertrand and Mr Doyigbé Etienne for proof-reading the article.

## REFERENCES

[1] C. Y. Subhash and K. S. Sanjay, " An Introduction to Client/Server Computing," New Age International Publishers, 2009.

[2] M. A. Seltzer, "Journalism versus Soft Updates: Asynchronous Meta-data Protection in File Systems," Proceedings of USENIX Annual Technical Conference, 1st ed, Johns and Bartlett, Boston, 2000.

[3] L. Lee, "Synchronous online exchanges: a study of modification devices on non-native discourse," System 30, 2002, pp.275-288

[4] L. Ye, "Embedded XML Data Exchange for Asynchronous Communication Between e-Commerce Systems," Management of e-Commerce and e-Gouvernment, 2009, pp. 164-167

[5] J. Yu, "XML Based Asynchronous Communication for Labor and Social Security," Advances in Intelligent and Soft Computing, 2012, pp. 311-318

[6] T.-Y. Hsiao, "An Asynchronous Message Exchange System on CORBA," Technology of Object-Oriented Languages and Systems, 2000, pp. 14-23.

[7] P. Roch, "Communication et Traitement en Mode Message avec MSQueries" at the http://www.patrick-roch.com/ingemeca/docs/Genie-Informatique/Arcchitectures des réseaux/Communication et traitement en mode message avec MSQueries.PDF visited on May 2012.

[8] B. A. Mah, "Quality of Service and Asynchronous Transfer Mode in IP Internet-works," PhD Thesis, University of California at Berkeley, 1996

[9] P. Roch, "Communication et Traitement en Mode Message avec MSQueries" at the http://middleware.smile.fr/Concepts-des-MOMs-et-JMS/Java-Messaging-System-ou-JMS visited on May 2012.

[10] ATM Forum, "ATM User-Network Interface Specification, version 3.1", PTR Prentice Hall, 1995.

[11] E.C. Ezin, "Implementation in Java of a Cryptosystem using a Dynamic Huffman Coding and Encryption Methods" International Journal of Computer Science and Information Security, *(IJCSIS),* volume 9, number 3, pp. 154-159, ISSN 1947-5500, 2010.

[12] Z. Li and M. Parashar, "Grid-based Assynchronous Replica Exchange," IEEE International Conference on Grid Computing, 2007, pp. 201-208

[13] M. A. Seltzer, "Journalism versus Soft Updates: Asynchronous Meta-data Protection in File Systems," Proceedings of USENIX Annual Technical Conference, 1st ed, Johns and Bartlett, Boston, 2000.

## AUTHORS PROFILE

Eugène C. Ezin received his Ph.D degree with highest level of distinction in 2001 after research works carried out on neural and fuzzy systems for speech applications at the International Institute for Advanced Scientific Studies in Italy. Since July 2012, he has been an associate professor in computer science. He supervised many master thesis in the same field.

He is a reviewer of Mexican International Conference on Artificial Intelligence and other journals. His research interests include neural network and fuzzy systems, high performance computing, signal processing, cryptography, modeling and simulation, information systems and network security.

Fréjus A.A. Laleye received his Bachelor degree in computer science in 2009. He is currently a student in a master program in computer science and applied sciences at the Institut de Mathématiques et de Sciences Physiques in Benin Republic. His research interests include information systems, signal and image process and network applications