Construcción de un Sistema de Comunicación Distribuido Basado en la Reutilización de un Framework de Objetos

Enrique Luna, Apolinar Velarde, Juan Nungaray, Iván Castillo y Roberto Cruz

E. Luna, A. Velarde, J. Nungaray, I. Castillo y R. Cruz
Instituto Tecnológico El Llano Aguascalientes, Carr. Aguascalientes-S.L.P. Km. 18, El Llano, Ags. elunaram@hotmail.com

M. Ramos., V.Aguilera., (eds.) .Ciencias de la Ingeniería y Tecnología, Handbook -©ECORFAN- Valle de Santiago, Guanajuato, 2014.

Abstract

It is presented a distributed communication system for sending instant messages, based on a framework called Object Client-Server Framework, which provides a set of abstract classes for developing distributed applications based on the client-server architecture. Particularly, the framework allows the development of interfaces for displaying on a client side the messages coming from other clients, thanks to which, it was possible to generate different versions of the communication system, with more and better functionalities for the end user in each new version. In order to manage this, it was necessary to carry on changes on the design of the system modules, experience that has been very useful in teaching and learning basic topics related to Software Engineering and Distributed Systems.

3 Introducción

El desarrollo de frameworks es una excelente forma de promover la reutilización de software: las aplicaciones que realizan tareas diferentes, pero relacionadas en su diseño, suelen tener patrones similares de interacción entre sus componentes (Lethbridge et al., 2004; Guelfi et al. , 2004). Esto es verdad aun si las aplicaciones son concebidas para dominios diferentes.

Así, lo que distingue a un framework de otros tipos de software es que éste es intrínsecamente incompleto, lo que significa que un desarrollador debe completar un framework para generar una aplicación que satisfaga necesidades específicas. Particularmente, en el contexto del paradigma orientado a objetos, un framework está compuesto de una biblioteca de clases, por lo que los servicios ofrecidos por el framework quedan definidos por el conjunto de todos los métodos públicos de las clases públicas, varias de las cuales típicamente son abstractas. De esta manera, para usar un framework en el desarrollo de una aplicación (distribuida), se deben crear clases concretas derivadas de las clases abstractas.

Para el desarrollo de nuestro sistema de comunicación, además de un framework específico (descrito en los fundamentos teóricos), se utilizó una arquitectura específica como base de la parte distribuida, a decir, la arquitectura cliente-servidor, bajo la cual diferentes operaciones son ejecutadas por programas separados, en equipos de hardware separados, que interactúan para operar el sistema como un todo.

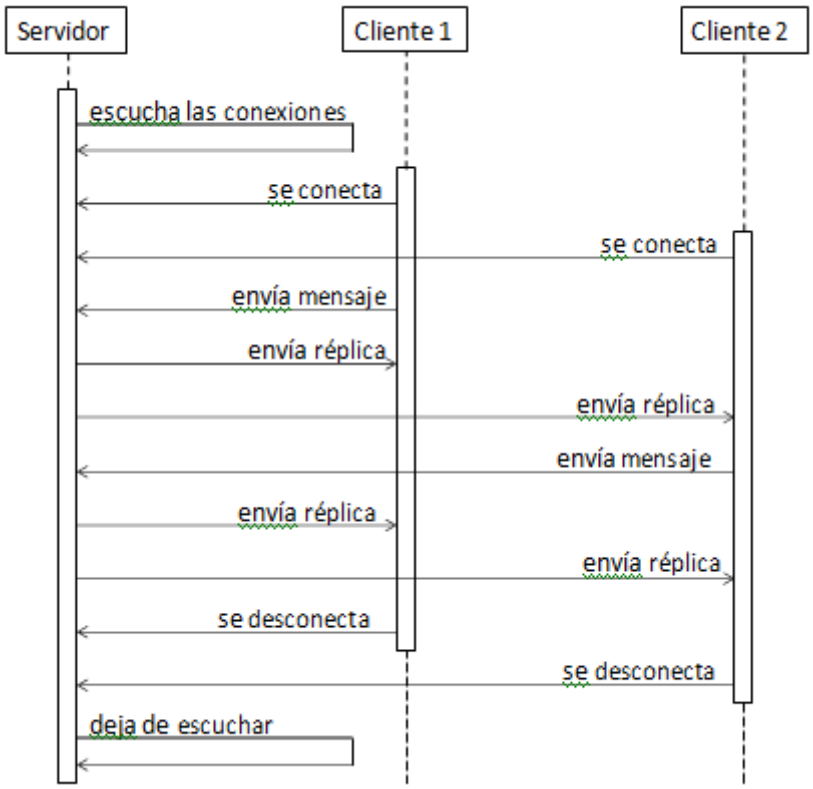
Como su nombre lo indica, la arquitectura cliente-servidor considera básicamente dos elementos: un Servidor que se refiere a un programa que provee algunos servicios a otros programas conectados a él a través de un canal de comunicación y un Cliente que es un programa que accede a un Servidor para utilizar sus servicios (Dollimore et al., 2005; Gomaa, 2013; Bruno, 2005).

Con base en lo anterior, en este artículo se considera una sección de fundamentos teóricos para describir el framework y la arquitectura utilizados. Posteriormente, en secciones separadas,se describe la metodología, la construcción del sistema y los resultados obtenidos, finalizando con la sección de conclusiones.

Fundamentos teóricos

La Arquitectura del Software es la rama de la Ingeniería de Software que se ocupa de organizar y comunicar los diferentes módulos que componen una aplicación de manera que éstos puedan operar entre sí. Existen diversas arquitecturas de software, siendo una de las más conocidas la Arquitectura Cliente-Servidor, utilizada comúnmente como base para el desarrollo de aplicaciones distribuidas, como es nuestro caso. En la figura 1 se muestra el diagrama de secuencias (UML) para un proceso de comunicación típico entre módulos bajo una arquitectura de este tipo (Dollimore et al., 2005; Gomaa, 2013; Bruno, 2005).

Figura 3 Proceso de comunicación bajo una arquitectura cliente-servidor



En este diagrama se pueden observar aquellos aspectos básicos en el proceso de comunicación entre módulos, descritos a continuación:

- El servidor comienza a operar, indicando a los clientes el puerto por el cual está "escuchando", es decir, el servidor les indica a los clientes el número de puerto por el cual pueden conectarse con él.

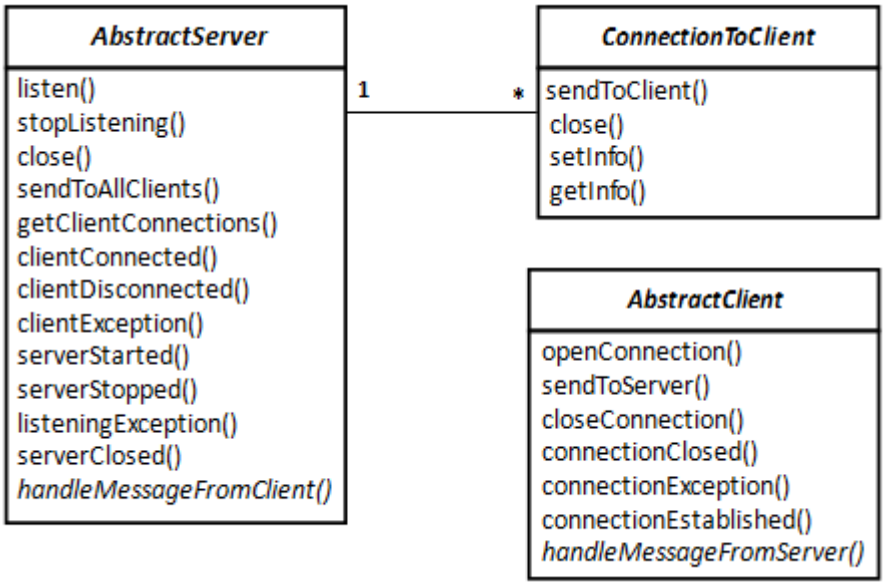
Una vez que el servidor está activo, los clientes (cualquier número de ellos) pueden conectarse con él a través del puerto indicado, validándose todos los errores posibles en el proceso de conexión.

- Los clientes conectados al servidor pueden iniciar un proceso de comunicación entre ellos a través del servidor mismo, por el cual pasarán todos los mensajes que se generen en dicho proceso.

- Finalmente, los clientes se desconectarán cuando lo deseen. El servidor hará lo propio una vez concluido el proceso de comunicación.

Respecto al Object Client-Server Framework (OCSF, por sus siglas en Inglés), éste consiste en tres clases: una clase para implementar el cliente y dos clases para implementar el servidor. Las clases del framework son ilustradas en la figura 2, en la cual se pueden observar sus métodos más importantes. La línea con el asterisco indica que pueden existir muchas instancias de la clase ConnectionToClient asociadas con el servidor (Raynal, 2013; Ashmore, 2014).

Figura 3.1 Elementos básicos de las clases del framework OCSF



Los programadores que utilizan el framework OCSF nunca deberían modificar sus clases, sino más bien deberían realizar lo siguiente para una aplicación:

- Crear subclases a partir de las clases abstractas del framework.

- En las subclases, escribir implementaciones de ciertos métodos declarados como abstractos.

- También en las subclases, sobrescribir ciertos métodos diseñados para ser sobrescritos.

- En diferentes partes de la aplicación, llamar los métodos públicos que conforman los servicios del framework, los cuales permiten al desarrollador controlar al cliente y al servidor.

A manera de ejemplo, al crear una subclase de la clase AbstractServer, elmétodo handleMessageFromClient llama al método sendToAllClients para poder publicar cualquier mensaje de un cliente dirigido a los demás clientes conectados al servidor, cuya tarea en este caso es retransmitir los mensajes a todos los clientes. Así, el código para logar esto sería el siguiente:

```
public void handleMessageFromClient
    (Object mensaje, ConnectionToClient cliente) {
            System.out.println("Mensaje recibido: "
              + mensaje + " de " + cliente);
            this.sendToAllClients(mensaje);
}
```

3.2 Metodología

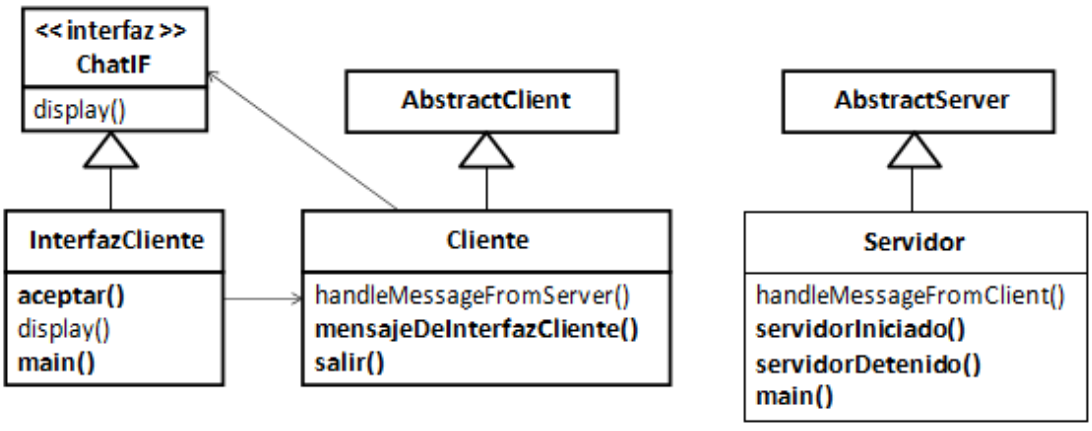
Dado que este trabajo consistió en construir diferentes versiones de un sistema de comunicación distribuido, la metodología para llevar a cabo esto se redujo a la ejecución secuencial de las siguientes actividades:

- Revisión de diferentes frameworks utilizados en el desarrollo de aplicaciones distribuidas.
- Definición del framework OCSF como plataforma de desarrollo del sistema de comunicación.
- Diseño de la arquitectura específica del sistema de comunicación basada en la arquitectura genérica cliente-servidor.
- Diseño y construcción de los módulos del sistema de comunicación: cliente, servidor e interfaz.
- Integración de los módulos: establecimiento de la comunicación entre ellos.
- Pruebas y depuración del sistema: aplicación en prácticas de Ingeniería de Software.

Construcción del sistema de comunicación

Para ilustrar la reutilización del framework OCSF en la construcción de nuestro sistema de comunicación, en la figura 3 se muestra las subclases creadas a partir de las clases abstractas del framework. Como se puede observar en la figura, las subclases creadas fueron Servidor, Cliente e Intefaz Cliente, para las cuales se definieron métodos específicos para lograr el propósito del sistema de comunicación.

Figura 3.2 Extensión del framewok OCSF para construir el sistema de comunicación distribuido



En el caso de la clase Servidor, como se mencionó con anterioridad, el método heredado a manera de servicio handle Message From Client de la clase Abstract Server es un método básico que permite publicar cualquier mensaje de un cliente dirigido a los demás clientes conectados un mismo servidor. Por su parte, el método main() permite crear una nueva instancia, iniciando así la operación del servidor, de manera que éste espera la conexión de clientes llamando al método listen() de la clase Abstract Server.

En el caso de la clase Cliente, en ésta se sobrescribe el método handle Message From Server(), el cual llama al método display() de la interfaz ChatIF, que a su vez resulta en una llamada al método display() de la clase Interfaz Cliente. Es importante señalar que la interfaz de usuario fue separada intencionalmente de la parte funcional del cliente con el propósito de poder darle mayor escalabilidad a nuestro sistema al momento de diseñar prácticas de laboratorio. Esta separación puede ser observada en la figura 3, donde se puede apreciar que la interfaz ChatIF juega un papel fundamental para lograr este propósito.

Cuando un cliente inicia, el método main() en la clase Interfaz Cliente se ejecuta, lo cual crea naturalmente una instancia de esta clase, pero también crea una instancia de Cliente (que opera como un segundo hilo), llamándose al método aceptar() para recibir mensajes del usuario. Este método se ejecuta en un ciclo hasta que el programa es interrumpido, mientras tanto envía todos los mensajes a la instancia de la clase Cliente con la ayuda del método mensaje De Intefaz Cliente(), el cual recibe los mensajes y los reenvía al servidor:

```
public void mensajeDeInterfazCliente  (String mensaje) {
   try {
            sendToServer(mensaje);
   }
   catch (IOException e) {
            cliente.display
              ("¡No se pudo enviar el mensaje al servidor!");
            salir();
   }
}
```

3.3 Resultados

En la figura 4 se puede observar la primera versión de nuestro sistema de comunicación, que a manera de ejemplo muestra tres clientes conectados a un servidor, el cual escucha por el puerto 2314, por lo que los clientes deberán conectarse por este puerto, de otra forma, no se establecerá la conexión con el servidor. Los clientes emulan una sesión de "chat", a decir, un profesor y dos estudiantes hablando de la entrega de una práctica de laboratorio.

Con base en esta versión inicial es posible generar una variedad de adecuaciones al sistema de comunicación, mismas que pueden aprovecharse para el diseño de prácticas para el reforzamiento del aprendizaje en temas relacionados con la Ingeniería de Software y los Sistemas Distribuidos, al ser necesario rediseñar la arquitectura del sistema y redefinir diversos diagramas UML (de clases, de secuencias, de estados, etc.) para lograr tales adecuaciones.

Como muestra de lo anterior, considérese la idea de que los clientes pudieran comunicarse a través de una interfaz gráfica, más amigable e intuitiva que la actual interfaz (tipo consola).

Esto es relativamente fácil de lograr gracias a la separación realizada desde un inicio en el lado del cliente, entre su parte funcional y la interfaz de usuario, así como a la existencia de la clase Frame de Java, que permite la creación de diversas aplicaciones en modo gráfico. De esta manera, para lograr una interfaz gráfica del cliente sería necesario que la subclase Cliente heredara algunos métodos de la clase Frame, modificándose ligeramente el diagrama de la figura 3, no obstante siendo siempre necesaria la clase ChatIF:

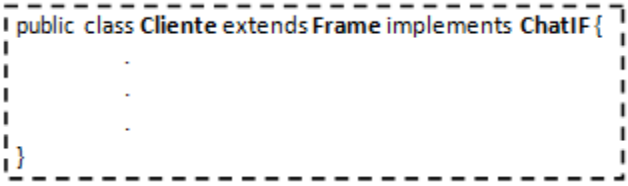


Figura 3.3 Primera versión del sistema de comunicación

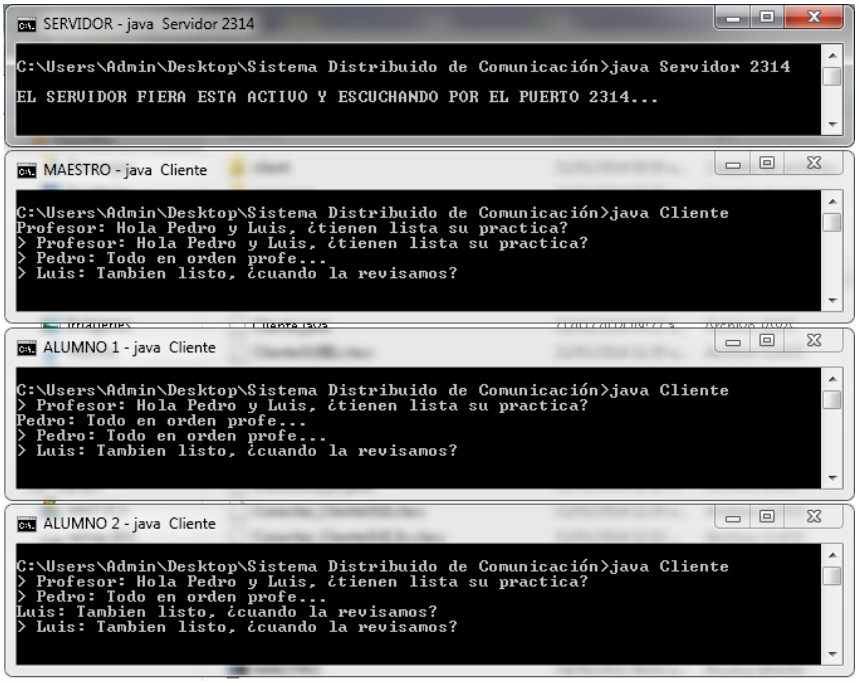
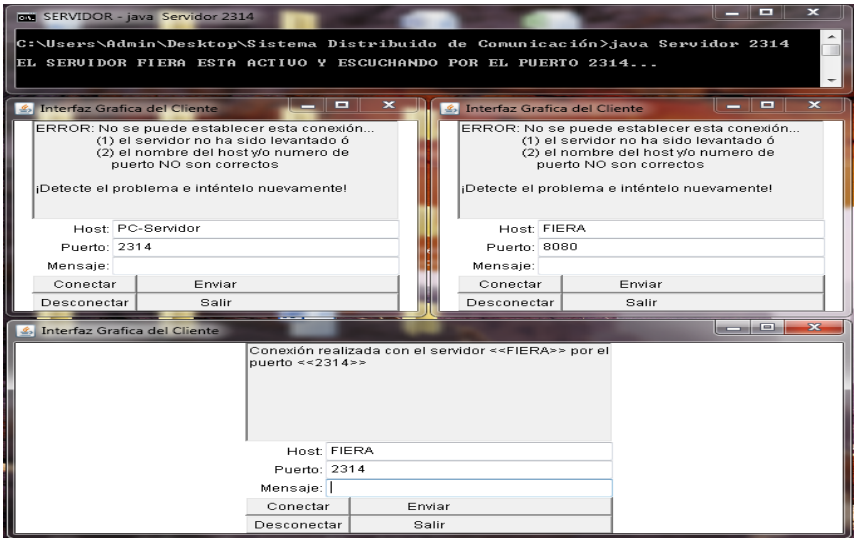


Figura 3.4 Sistema de comunicación operando mediante una interfaz gráfica para el cliente

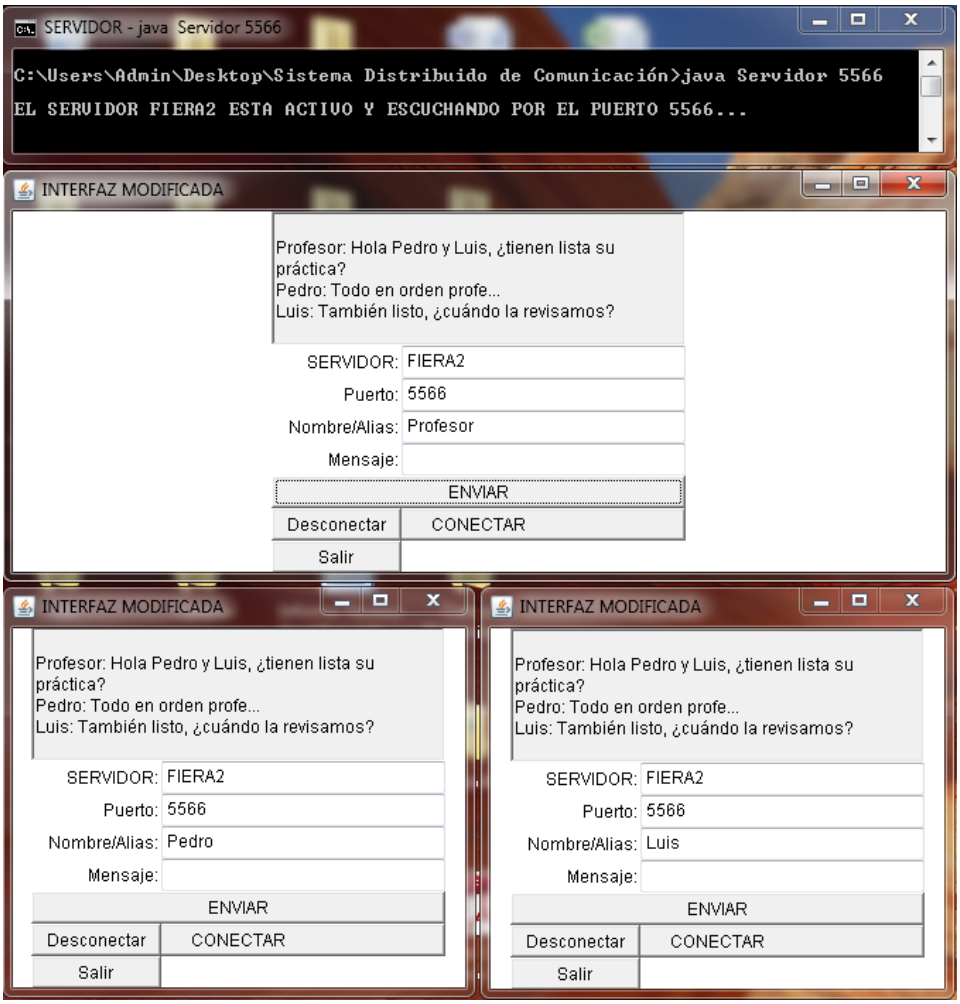


En la figura 5 se muestra una nueva versión de nuestro sistema de comunicación, con la cual el cliente puede comunicarse a través de una interfaz gráfica. Como se puede apreciar en la figura, la interfaz considera tres tipos de elementos: un área de diálogo, un conjunto cajas para entradas de datos y un conjunto de botones para realizar diversas acciones.

En el área de dialogo se despliegan los mensajes que ocurren en un sesión de comunicación entre un grupo de clientes, validándose los errores que puedan ocurrir al momento de iniciar la sesión. Por ejemplo, en la primera Intefaz del Cliente (arriba a la izquierda), el usuario trata de conectarse al servidor utilizando un nombre de Host equivocado, a decir, "PC-Servidor", ya que el nombre correcto es "FIERA". Por su parte, en la segunda Intefaz del Cliente (arriba a la derecha), el usuario trata de conectarse al servidor utilizando un número de Puerto equivocado, a decir, "8080", siendo el número correcto "2314". Finalmente, en la tercera Intefaz del Cliente (parte inferior de la figura), el usuario logra conectarse sin problemas al servidor, al utilizar los datos de conexión correctos.

En la figura 6 se muestra una modificación menor de la Intefaz Gráfica del Cliente, la cual consiste en el uso de un Nombre/Alias para poder identificar de quien proviene cada mensaje. Esta adecuación en la interfaz, aunque menor, puede considerarse una nueva versión del sistema de comunicación.

Figura 3.5 Interfaz gráfica del cliente ligeramente modificada



De esta manera, es posible realizar diversos cambios o adecuaciones en los diferentes módulos del sistema, menores o mayores, que invariablemente requieren de una serie de tareas propias de la Ingeniería de Software para su implementación, por lo que el aprendizaje del estudiante ocurre de manera natural.

3.4 Conclusiones

Se describió un framework para el desarrollo de aplicaciones distribuidas, tratándose el caso de un sistema de comunicación como un ejemplo interesante.

Se demostró mediante evaluaciones a estudiantes que el desarrollo de este tipo de aplicaciones puede ser de gran utilidad en la enseñanza y aprendizaje de temas relacionados con la Ingeniería de Software y los Sistemas Distribuidos, habiéndose logrado reducir los índices de reprobación prácticamente a la mitad con la ayuda de prácticas de laboratorio relacionadas con el desarrollo del sistema de comunicación.

Como trabajo futuro, se tiene contemplado reutilizar el framework para construir otras aplicaciones, quizás más sofisticadas, como sería el caso de un Asistente de Navegación Basado en GPS y otros casos similares.

Referencias

Ashmore, Derek C. (2014). *The Java EE Architect's Handbook: How to be a successful application architect for Java EE applications.* DVT Press, 2nd Edition.

Bruno, Eric. (2005). *Java Messaging (Programming Series).* Cengage Learning, 1st Edition. Dollimore, Jean; Kindberg, Tim and Coulouris, George. (2005). *Distributed Systems: Concepts and Design.* Addison Wesley, 4th edition.

Gomaa, H. (2013). *Designing Concurrent, Distributed, and Real-Time Applications with UML.* Addison-Wesley Professional, 1st Edition.

Guelfi, N., Reggio, G., & Romanovsky, A (2004). *Scientific Engineering of Distributed Java Applications.* Springer.

Lethbridge, Timothy C. and Laganiere, Robert. (2004). *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. McGraw-Hill, 2nd edition (2004).

Raynal, Michel. (2013). *Distributed Algorithms for Message-Passing Systems.* Springer.