UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:

INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
INGENIERA DE SISTEMAS

TEMA:

ANÁLISIS Y APLICACIÓN DE PROGRAMACIÓN ORIENTADA A AGENTES E INTERACCIÓN DE MULTIAGENTES CON PLATAFORMA JADE.

AUTORAS:

EVELYN ESTEFANIA OBANDO TOASA
DORIS MARLENE TITUAÑA CHICAIZA

TUTOR:

JOSE LUIS AGUAYO MORALES

Quito, julio del 2016

CESIÓN DE DERECHOS DE AUTOR

Nosotras Evelyn Obando con documento de identificación N°: 1719354613 y Doris Tituaña, con documento de identificación N°: 1719309476, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autoras del trabajo de titulación intitulado: ANÁLISIS Y APLICACIÓN DE PROGRAMACIÓN ORIENTADA A AGENTES E INTERACCIÓN DE MULTIAGENTES CON PLATAFORMA JADE, mismo que ha sido desarrollado para optar por el título de: Ingenieras de Sistemas, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autoras nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

.....

Evelyn Estefania Obando Toasa

1719354613

.....

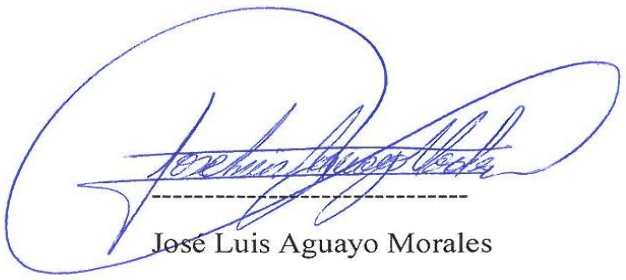
Doris Marlene Tituaña Chicaiza

1719309476

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el proyecto de titulación: ANÁLISIS Y APLICACIÓN DE PROGRAMACIÓN ORIENTADA A AGENTES E INTERACCIÓN DE MULTIAGENTES CON PLATAFORMA JADE realizado por las estudiantes Evelyn Estefania Obando, Doris Marlene Tituaña Chicaiza obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, julio 2016



José Luis Aguayo Morales

1709562597

AGRADECIMIENTO

El presente proyecto de titulación, agradecemos a la Universidad Politécnica Salesiana por darnos la oportunidad de estudiar en tan prestigiosa institución y ser profesionales de dicha institución.

A nuestro tutor, Ing. José Luis Aguayo por su esfuerzo y dedicación quien, con sus conocimientos, experiencia, paciencia, buen humor y su motivación ha logrado que podamos concluir este proyecto con éxito.

También nos gustaría agradecer a todos los profesores que han formado parte de nuestra carrera profesional porque han aportado con sus conocimientos para nuestra formación.

De igual manera agradecer a los lectores del proyecto Ing. Washington Ramírez, Ing. René Arévalo por su visión crítica acerca del tema, por su profesionalismo, por sus consejos, que ayudan a formarte como investigador.

DEDICATORIA

Dedico este proyecto de titulación a mis padres Gonzalo e Hilda por ser los pilares fundamentales en mi vida, quienes con su esfuerzo y sacrificio me apoyan y guían con mucho amor gracias por darme la vida.

Con todo mi cariño a Gonzalo quien, con su apoyo, consejos y principalmente por estar siempre presente en cada etapa de mi vida, es un ejemplo a seguir a Byron por cuidarme y protegerme desde niña y a pesar del tiempo que estuvo lejos de mí siempre estuvo pendiente, por ayudarme con los recursos necesarios para empezar mi carrera, por buscar mi bienestar, gracias por todo hermanos.

A mis sobrinos David, Brenda, Sofía y Alex quienes han formado parte de mi vida y me han dado mucha felicidad, por ser lo mejor que mis hermanos me han dado. A mi cuñada Alexandra por ser la persona que siempre está presente en los buenos y malos momentos de la familia.

Especialmente agradezco a dos personas muy especiales que siempre creyeron en que llegaría a ser una profesional a mi tío Aníbal te recordaré siempre y Clarita gracias por siempre estar conmigo.

Gracias a todos ustedes por ayudarme a forjar lo que soy mis valores, principios, carácter y coraje para conseguir mis objetivos y ser una profesional.

Evelyn

Dedico este proyecto de titulación principalmente a mis padres Rosalino Tituaña y Olga Chicaiza, quienes a lo largo de mi vida, han velado por mi bienestar y educación siendo mi apoyo incondicional en todo momento, por darme la vida y permitirme crecer en un hogar lleno de amor y comprensión, por ser el pilar fundamental para cumplir todos mis sueños.

A todos mis hermanos por estar siempre presente en mi vida, por sus consejos y palabras de aliento las cuales me ayudaron a seguir.

A mis sobrinos los cuales con su presencia llenaron mi vida de motivación, inspiración y felicidad y de manera especial a mi hija Dana quien es mi inspiración para seguir adelante. Todo este trabajo ha sido posible gracias a ellos.

Doris

ÍNDICE

INTRODUCCIÓN 1
CAPÍTULO 1 5
ESTADO DEL ARTE 5
1 Agente 5
1.1 Características de los agentes 5
1.2 Clasificación de los agentes 6
1.3 Ciclo de vida de un agente 7
1.4 Sistema Multi-agente (SMA) 8
1.4.1 Características de los SMA 8
1.5 Estándar FIPA 9
1.6 Comunicación FIPA-ACL 10
1.6.1 Elementos de FIPA-ACL 11
1.7 Netbeans 13
1.7.1 Características 14
1.7.2 Profile Project 14
1.8 Plataforma Jade (Framework de desarrollo de Agentes Java). 16
1.8.1 Características Jade 16
1.8.2 Entorno Jade 16
1.8.3 Main Container 17
1.8.4 Dummy Agent 20
1.8.5 Sniffer Agent. 21
1.8.6 Introspector Agent. 21
1.9 Comportamientos de JADE 22
1.9.1 Métodos de un Comportamiento 24
1.10 Programación Orientada a Objetos 25
1.11 Elementos de la programación orientada a objetos (POO) 25
1.11.1 Objeto 25
1.11.2 Clase 25
1.11.3 Método 25
1.12 Programación Orientada a Agentes (POA) 26

1.13 Programación Orientada a Objetos vs Programación Orientada a Agentes 26
1.13.1 Unidad básica 27
1.13.2 Parámetros que definen el estado de la unidad básica 27
1.13.3 Proceso de cálculo 27
1.13.4 Tipo de mensaje 28
1.13.5 Restricción del método 28
CAPÍTULO 2 29
METODOLOGÍA ORIENTADA A AGENTES 29
2.1 Herramienta AUML (Agent Unified Modeling Language) 29
2.2 Arquitectura AUML 30
2.3 Casos de Estudio 30
2.3.1 Diagrama de Interacción 31
2.3.2 Diagrama Protocolos de interacción. 32
2.3.3 Diagrama de Tareas 34
2.3.4 Diagrama de Secuencias 35
2.4 Diagramas UML Objetos 37
2.4.1 Diagrama de Caso de Uso 37
2.4.2 Diagrama de Clases 39
2.4.3 Diagrama de Secuencia 40
CAPÍTULO 3 43
DESARROLLO DE LA APLICACIÓN 43
3.1 Herramientas de desarrollo 43
3.1.1 Java Development Kit 43
3.1.2 JDK en Windows 44
3.2 Instalación plugins y herramientas de programación 44
3.2.1 Instalación de Netbeans 44
3.2.2 Configuración de JADE 44
3.3 Desarrollo y ejecución de la aplicación 44
3.3.1 Programación Orientada a Agentes 44
3.3.2 Comunicación de Agentes 49
3.3.3 Programación Orientada a Objetos 52
CAPÍTULO 4 57
ANÁLISIS DE RESULTADOS 57

4.1 Gráficas Monitor Agentes 57
4.1.1 Variación *Heap Size* y *Used Heap* 58
4.1.2 *Surviving Generations* 58
4.1.3 Variación *Surviving Generations* 59
4.1.4 *Threads, Loaded Classes* 59
4.1.5 Variación *Threads, Loaded Classes* 59
4.1.6 Variación *Heap Size* y *Used Heap* en la creación del agente vendedor. 60
4.1.*7 Variación Threads, Loaded Classes* 60
4.1.8 Variación *Heap Size* y *Used Heap* en la creación del agente comprador. 61
4.1.9 *Variación Threads, Loaded Clases* 61
4.1.10 Variación *Heap Size y Used Heap* al final de la aplicación. 62
4.1.11 Variación *Threads, Loaded Classes* 62
4.2 Gráficas Monitor Objetos 63
4.2.1 Variación *Heap Size* y *Used Heap* 63
4.2.2 Variación *Surviving Generations* 64
4.2.3 *Variación Threads, Loaded Classes* 65
4.2.4 Variación *Heap Size* y *Used Heap* en la creación del vendedor 65
5.2.5 Variación *Heap Size* y *Used Heap* en la creación del comprador 66
4.3 Gráficas Resultantes 67
4.3.1 Gráfica de *Used Size* 67
4.3.2 Gráfica Resultante de *Surviving Generations* 68
4.3.3 Gráfica Resultante Hilos 69
4.3.4 Gráfica Resultante Loaded Classes 70
4.4 Gráficas CPU Agentes 71
4.4.1 Variación de *Heap Size* y *Used Size* 71
4.4.2 Variación de *Surviving Generations* 71
4.4.3 Variación Threads, Loaded Classes 72
4.5 Gráficas CPU Objetos 72
4.5.1 Variación *Heap Size* y *Used Heap* 72
4.5.2 Variación *Surviving Generations* 73
4.6 Gráficas Resultantes 74
4.6.1 Gráfica de *Used Size* 74
4.6.2 Gráfica Resultante de *Surviving Generations* 75

4.6.3 Gráfica Resultante Hilos 76
4.6.4 Gráfica Resultante Clases Cargadas 78
4.7 Gráficas Memoria Agentes 79
4.7.1 Variación de *Heap Size* y *Used Size* 79
4.7.2 Variación de Surviving Generations 79
4.7.3 Variación *Threads, Loaded Classes* 80
4.8 Gráficas Memoria Objetos 80
4.8.1 Variación *Heap Size* y *Used Heap* 80
4.8.2 Variación de *Surviving Generations* 81
4.8.3 Variación *Threads, Loaded Classes* 81
4.9 Gráficas Resultantes 82
4.9.1 Gráfica de *Used Size* 82
4.9.2 Gráfica Resultante de *Surviving Generations* 82
4.9.3 Gráfica Resultante Hilos 84
4.9.4 Gráfica Resultante Clases Cargadas 85
4.10 Prueba de Hipótesis 85
CONCLUSIONES 93
GLOSARIO 96
REFERENCIAS 98

ÍNDICE DE TABLAS

Tabla 1. Elementos de FIPA-ACL 12
Tabla 2. Comportamientos Agentes en JADE 23
Tabla 3. Características de la Programación Orientada a Objetos vs la Programación Orientada a Agentes 26

ÍNDICE DE FIGURAS

Figura 1. Representación de los estados de un agente. 7
Figura 2. Arquitectura FIPA. 10
Figura 3. Componentes del Lenguaje de Comunicación FIPA-ACL. 11
Figura 4. Gráficas del Profiler. 15
Figura 5. Monitoreo remoto de agentes. 17
Figura 6. Estructura de Jade. 18
Figura 7. Estructura de los elementos de JADE. 19
Figura 8. Dummy Agent. 20
Figura 9. Agente Sniffer. 21
Figura 10. Introspector Agent. 22
Figura 11. Diagrama de Interacción. 32
Figura 12. Diagrama Protocolos de Interacción. 33
Figura 13. Diagrama de Tareas. 34
Figura 14. Diagrama de Secuencia. 36
Figura 15. Diagrama Caso de Uso. 39
Figura 16. Diagrama de Clases. 40
Figura 17. Diagrama de Secuencias. 42
Figura 18. Pantalla principal. 45
Figura 19. Creación Agente Vendedor. 46
Figura 20. Ingreso información Libro. 46
Figura 21. Creación Agente Comprador. 47
Figura 22. Proceso de búsqueda del libro a comprar. 48
Figura 23. Información del Agente Vendedor. 48
Figura 24. Información de la Compra. 49
Figura 25. Creación del Agente Sniffer. 50
Figura 26. Comunicación entre Agentes. 51
Figura 27. Estructura del mensaje ACL. 51
Figura 28. Pantalla Principal. 52

Figura 29. Pantalla Instrucciones del Usuario. 52
Figura 30. Pantalla registro del vendedor. 53
Figura 31. Pantalla registro catálogo. 54
Figura 32. Pantalla registro comprador. 54
Figura 33. Pantalla Compra del Libro. 55
Figura 34. Resultado Exitoso de la búsqueda. 56
Figura 35. Resultado erróneo de la búsqueda. 56
Figura 36. Gráficas Monitor Agentes. 57
Figura 37. Variación valores *Heap Size* y *Used Size.* 58
Figura 38. Variación *Surviving Generations.* 59
Figura 39. *Threads, Loaded Classes.* 59
Figura 40. Variación valores *Heap Size* y *Used* Size. 60
Figura 41. *Threads, Loaded Classes.* 60
Figura 42. Variación *Heap Size* y *Used Heap.* 61
Figura 43. *Threads, Loaded Classes.* 61
Figura 44. Variación *Heap Size* y *Used Heap* 62
Figura 45. *Threads, Loaded Classes.* 62
Figura 46. Graficas Monitor Objetos. 63
Figura 47. Variación de valores *Heap Size* y *Used Heap.* 64
Figura 48. Variación *Surviving Generations.* 64
Figura 49. *Threads, Loaded Classes.* 65
Figura 50. Variación Heap Size y Used Heap en la creación del vendedor. 66
Figura 51. Variación *Heap Size* y *Used Heap* en la creación del comprador. 66
Figura 52. Gráfica Final Heap Size agentes vs objetos. 67
Figura 53. Gráfica final de *Surviving Generations* agentes vs objetos. 68
Figura 54. Gráfica final hilos agentes vs objetos. 69
Figura 55. Gráfica final *loaded classes* agentes vs objetos. 70
Figura 56. Variación valores *Heap Size* y *Used Size.* 71
Figura 57. Variación *Surviving Generations.* 71
Figura 58. *Threads, Loaded Classes.* 72
Figura 59. Valores *Heap Size* y *Used Heap* inicio de la aplicación. 73

Figura 60. Variación de Surviving Generations. 73
Figura 61. Variaciòn de Threads,Loaded Classes. 74
Figura 62. Gráfica Final Heap Size agentes vs objetos. 75
Figura 63. Gráfica final de *Surviving Generations* agentes vs objetos. 76
Figura 64. Gráfica Final hilos Agentes vs Objetos. 77
Figura 65. Gráfica Final hilos Agentes vs Objetos. 78
Figura 66. Variación valores *Heap Size* y *Used Size*. 79
Figura 67. Variación *Surviving Generations*. 79
Figura 68. *Threads, Loaded Classes*. 80
Figura 69. Valores Heap Size y Used Heap Inicio de la Aplicación. 80
Figura 70. Variación Surviving Generations. 81
Figura 71. Variación de threads y loaded classes. 81
Figura 72. Gráfica Final Heap Size Agentes vs Objetos. 82
Figura 73. Gráfica Final *Surviving Generations* agentes vs objetos. 83
Figura 74. Gráfica Final hilos Agentes vs Objetos. 84
Figura 75. Gráfica Final hilos Agentes vs Objetos. 85
Figura 76. Prueba de hipótesis 87
Figura 77. Prueba de hipótesis. 88
Figura 78. Prueba Hipótesis. 90
Figura 79. Prueba de hipótesis objetos. 92

RESUMEN

En la década del noventa, los agentes comenzaron a implementarse debido a sus características como autonomía, movilidad y colaboración que los hacían atractivos para ser implantados en ambientes abiertos. El paradigma de agentes tuvo sus fundamentos en algunas tecnologías conceptuales como Frames y el paradigma orientado a objetos. La Programación Orientada a Objetos (POO) y la Programación Orientada a Agentes (POA) son tecnologías utilizadas para el desarrollo de software

Este proyecto de investigación se realizó con el propósito de determinar las diferencias existentes entre la Programación Orientada a Objetos (POO) y la Programación Orientada a Agentes (POA). Se realizó un análisis para obtener información, examinar detalladamente sus partes, conocer sus características y extraer conclusiones, se estudió el uso y el manejo de la plataforma JADE para el desarrollo de agentes implementada en Java, se investigó una metodología para el desarrollo de agentes. Se desarrolló una aplicación demo (escritorio) de subasta de libros con POA y con POO. Utilizando un conjunto de datos se realizaron gráficas que evidencian las diferencias entre POO y POA.

Los resultados obtenidos en la prueba de hipótesis demuestran que más del 3% de los estudiantes no conocen la nueva tecnología de programación, la programación orientada agentes consume más del 5% de memoria respecto a la programación orientada a objetos. Finalmente se determinó que es factible el uso de agentes en el desarrollo de software por su autonomía, proactividad y comunicación con otros agentes.

ABSTRACT

This technical project is carried out in order to determine the differences between Object Oriented Programming (POO) and Agents Oriented Programming (POA) as these two technologies are very useful for software development, we will focus more on the use of agents and the POA.

So maked an analysis to obtain information, detailed examination of parts, know their characteristics and draw conclusions will study the use and management of the JADE platform for the development of agents implemented in Java and will study a methodology for development of agents which may be to structure, plan and control the development agents process. Develop a demo application (desktop) auction of books with POA and POO. Analyze the efficiency of POO and POA using a data set and finally with the data preliminarily obtained show graphs enabling the difference between OOP and POA.

On the other hand we will study the interaction between agents which make up the multi-agent systems this is done through the FIPA-ACL protocol that is responsible for establishing communication through messages agents such as (Inform, Request, Query-ref etc.) for communicating are also used behaviors too.

With the results we can determine that it is feasible the use of agents in the development of software as it is helpful for his capacity of autonomy, proactivity and communication with other agents.

INTRODUCCIÓN

La tecnología orientada a objetos no es nueva y presenta inconvenientes como: limitaciones para el programador, curvas de aprendizaje largas, dependencia del lenguaje, determinación de las clases, tamaño excesivo en las aplicaciones resultantes, herencia, velocidad de ejecución (Fernández V.).

Los agentes constituyen un nuevo enfoque para la realización de sistemas computacionales ya que son activos y su comportamiento es autónomo, presentan características como: reactivos, móviles y sociales tienen su respectiva clasificación y cuentan con un ciclo de vida. La programación orientada a agentes (POA) está motivada por la necesidad de manejar arquitecturas abiertas que cambien continuamente lo cual ofrece una nueva perspectiva en el diseño de software. (Ahogado & Reinemer, 2002).

El presente proyecto de investigación tiene por objetivo analizar y aplicar la programación orientada a agentes e interacción de multi-agentes mediante el estándar FIPA (Foundation for Intelligent Physical Agents) con esto se logrará determinar las diferencias existentes entre la programación orientada a objetos (POO) y la programación orientada a agentes (POA). Lo cual se evidenciará con una aplicación de escritorio en versión demo (Pedra, 2014) de subasta de libros con Programación Orientada a objetos y Programación Orientada a Agentes.

En el presente documento se resumirá la información sobre agentes, tipos de agentes, ciclos de vida, estados, herramientas de los agentes, plataforma Jade, arquitectura de la plataforma, estándares de comunicación FIPA.

La programación orientada a agentes está recibiendo una gran atención en los últimos años (Carrillo, 2005) y por lo tanto las personas están comenzando a interesarse en utilizar esta tecnología para aplicarlos en sus respectivas áreas de interés (Iglesias, 1998).

Problema de estudio

La Programación orientada a Objetos es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos; su uso se popularizó a principios de la década de los años 1990 (trahc.mx).

Sin embargo, según (Manso, 2002), (Jimenez, 2013), la tecnología orientada a objetos no es nueva y presenta inconvenientes como:

- Limitaciones para el programador un gran porcentaje de programadores no están familiarizados con los conceptos de dicha tecnología. En otras palabras, la lógica de la programación estructurada sigue siendo predominante en la mayoría de los desarrolladores de software.
- Curvas de aprendizaje largas un sistema orientado a objetos involucra la conceptualización de todos los elementos de un programa. Toda la comunicación entre los objetos debe realizarse en forma de mensajes.
- Determinación de las clases una clase se utiliza para crear nuevos objetos por los cual se deben crear la mínima cantidad de clases específicas para la aplicación que se está desarrollando.
- Tamaño excesivo en las aplicaciones resultantes al momento de ejecutar las aplicaciones demanda de gran capacidad de almacenamiento y memoria en el computador.
- Herencia cuando se heredan clases a partir de clases existentes; se heredan de forma implícita todos los miembros de dicha clase aun cuando no todos se necesiten, lo que produce aplicaciones muy grandes que no siempre encajan en los sistemas.

Justificación

El desarrollo de este tema se enfoca en el análisis de las diferencias existentes entre la programación orientada a objetos (POO) y la programación orientada a agentes (POA),

explicando las ventajas de usar agentes, como las que cita (Fernandez), (Navarrete, 2008, pág. 5), (Soza Pollman, 2014):

- Los agentes están sujetos a un ciclo de vida que definen los estados que puede presentar como: iniciado, activado, suspendido, en espera, tránsito y finalizado.
- Los agentes pueden ser clasificados en: agentes colaborativos, agentes móviles, agentes de interfaz, agentes de información, agentes inteligentes, agentes software; logrando así interactuar entre ellos dependiendo de sus necesidades.
- Entre sus características principales de los agentes se tienen:
    - reactivos es capaz de responder en cualquier entorno que se encuentre.
    - móviles tiene la capacidad de trasladarse a través de una red telemática.
    - sociales puede comunicarse con otros agentes a través de algún tipo de lenguaje de comunicación.

La POA está motivada por la necesidad de manejar arquitecturas abiertas que cambien continuamente lo cual ofrece una nueva perspectiva en el diseño de software; para evidenciar esto se realizará una aplicación demo en la cual se mostrará el uso de agentes y la interacción entre ellos (multi-agentes) (Ahogado & Reinemer, 2002).

JADE: Es una plataforma software para el desarrollo de agentes.

FIPA: Lenguaje de comunicación entre agentes.

FIPA-ACL: Agente Comunicación de Idiomas.

AMS: Sistema de Gestión de Agentes.

IIOP: Protocolo basado en objetos puede mejorar los tipos de aplicaciones o servicios o bases de datos que se construyen y comunicarse en la web.

Objetivo General

Analizar y aplicar la programación orientada a agentes e interacción de multi-agentes mediante el estándar FIPA.

Objetivos específicos

- Analizar la programación orientada a agentes.
- Estudiar el funcionamiento de la plataforma Jade.
- Investigar una metodología para el desarrollo de agentes.
- Desarrollar una aplicación demo con POO Y POA.
- Mostrar el resultado de las métricas para sustentar la diferencia existente entre la POO y la POA.

Alcance

Este proyecto pretende dar a conocer la programación orientada a agentes (POA) usando agentes y multi-agentes para lo cual se realizarán los siguientes procedimientos:

- Realizar un análisis previo de la Programación Orientada a Agentes. Aquí se obtendrá información relevante, examinar detalladamente separando o considerando por separado sus partes, para conocer sus características o cualidades, o su estado, y extraer conclusiones.
- Estudiar el uso y el manejo de la plataforma JADE que es una herramienta de desarrollo de agentes implementada en Java que soporta la coordinación de múltiples agentes FIPA. JADE proporciona una implementación estándar del lenguaje de comunicación FIPA-ACL que facilita la comunicación entre agentes y permite la detección de servicios que se proporcionan en el sistema.
- Estudiar una metodología para el desarrollo de agentes con la cual se podrá estructurar, planear y controlar el proceso de desarrollo de agentes.
- Desarrollar una aplicación demo de escritorio que realiza una subasta de libros con Programación Orientada a objetos y Programación Orientada a Agentes (se crearán agentes vendedores y agentes compradores, se registrará la información del libro en el catálogo, se mostrará la interacción entre agentes la comunicación mediante FIPA y el envío de mensajes).
- Analizar la eficiencia de la POO y POA utilizando un conjunto de datos.

- Con los datos obtenidos preliminarmente se realizará métricas que permitan evidenciar la diferencia entre POO y POA.

CAPÍTULO 1

ESTADO DEL ARTE

1 Agente

Existen algunas definiciones de agentes como, por ejemplo:

- "Un agente es un programa de ordenador que actúa autónomamente en nombre de una persona u organización" (Aguilar & Abraham, 2009).
- "Un agente es esencialmente un componente especial de software que tiene autonomía, que provee una interface interactiva con un sistema cualquiera se comporta como un agente humano, trabajando para algunos clientes con sus propios objetivos" (Manotas, 2010).

Por lo tanto, se considera que un agente es un proceso que cuenta con autonomía, es proactivo, reactivo y puede ser móvil. Por lo general los agentes no actúan solos, sino que pueden interactuar y comunicarse con otros agentes por medio de mensajes.

1.1 Características de los agentes

Las principales características de agentes se detallarán a continuación según (Soza Pollman, 2014):

- Un agente es social: porque tiene la capacidad de interactuar con otros agentes para alcanzar sus objetivos.
- Un agente es autónomo: porque tiene la capacidad de operar por sí mismo, sin la necesidad de una guía humana para controlar su estado interno o para realizar sus acciones.
- Un agente es proactivo: porque tiene la capacidad de tener una conducta dirigida por objetivos tomando la iniciativa para alcanzar sus metas.

1.2 Clasificación de los agentes

Según Navarrete la clasificación es:

- Agentes Colaborativos

Los agentes colaborativos se enfatizan en la autonomía y la cooperación para ejecutar tareas por ellos mismos.

- Agentes Móviles

Los agentes móviles son capaces de salir de los límites de las redes locales y acceden a computadores en redes remotas o incluso Internet.

- Agentes de Interfaz

Los agentes de interfaz trabajan sobre el aprendizaje y la autonomía. Su objetivo es el de servir a las necesidades del usuario.

- Agentes de Información

Los agentes de información surgen de la necesidad de recolectar, administrar y clasificar grandes volúmenes de información, como es el Internet.

- Agentes Reactivos

Los agentes reactivos son una clase especial de agentes, que actúan de manera reactiva a un evento o estímulo producido dentro del entorno.

- Agente Inteligente

Los agentes inteligentes permiten usar la inteligencia artificial como método de aprendizaje y razonamiento para su ejecución.

- Agente Software

Los agentes software permiten trabajar en entornos virtuales (Internet) dependiendo de su dominio (Navarrete, 2008, pág. 5).

1.3 Ciclo de vida de un agente

El ciclo de vida de un agente, según lo define Fernández es:

El ciclo de vida de un agente sigue el ciclo propuesto por FIPA, es decir, cumple con la propuesta de estándar de interoperabilidad entre agentes más aceptada (Fernandez, s.f.).

Representación de los estados de un agente

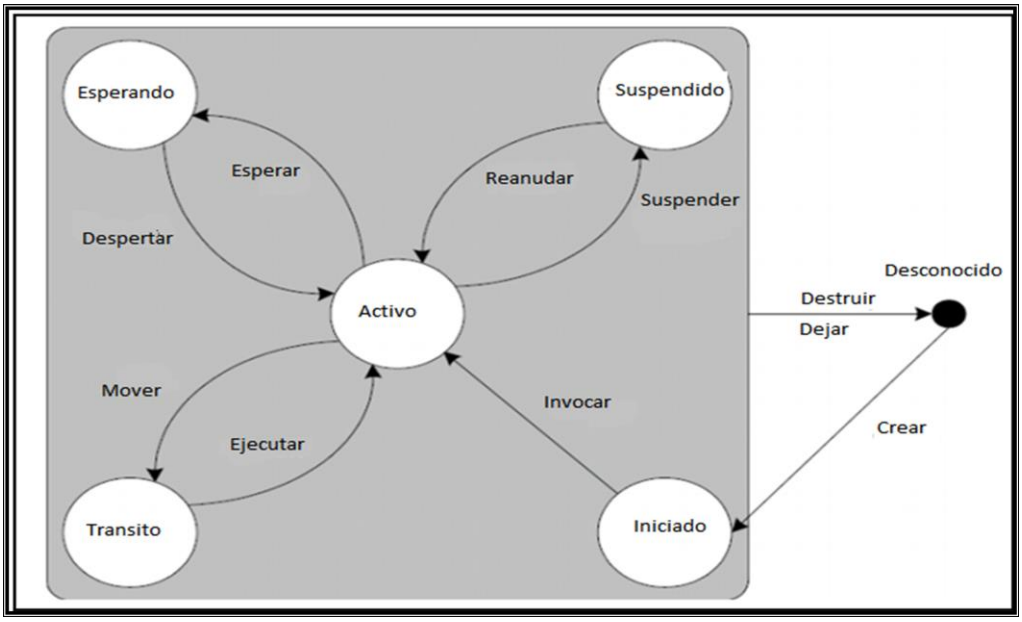


Figura 1. Representación de los estados de un agente.
El ciclo de vida de un agente o el ciclo de FIPA muestra los estados y las acciones que provocan el cambio de estado
Fuente: (Garamendi, 2004).

Las siguientes características mencionadas por Fernández detallan los principales estados que presentará un ciclo de vida de un agente.

- Iniciado: el agente esta creado, pero todavía no se ha registrado en el (AMS), no tiene nombre, ni dirección y tampoco se puede comunicar con otros agentes.

- Activo: el agente está registrado en el AMS, tiene un nombre y una dirección y puede acceder a todas las opciones de JADE.

- Suspendido: el agente está parado. Su hilo de ejecución está detenido y no ejecuta ningún comportamiento.

- En espera: el agente está bloqueado esperando por algo. Su hilo de ejecución está dormido en un monitor de java y se despertará cuando se cumpla una cierta condición (cuando reciba un mensaje).

- Desconocido: el agente ha sido eliminado. El hilo de ejecución ha terminado y se ha eliminado del registro del AMS.

- Tránsito: un agente móvil entra en este estado mientras está migrando a una nueva localización. El sistema sigue guardando los mensajes en el buffer hasta que el agente vuelve a estar activo (Fernandez, s.f.).

1.4 Sistema Multi-agente (SMA)

"Los Sistemas Multi-agente son típicamente complejos distribuidos y concurrentes que se compone de entidades que interactúan y muestran comportamientos emergentes" (Essaidi, Ganzha, & Paprzycki, 2012).

Los sistemas multi-agentes (SMA) son sistemas que describen a los agentes en un entorno social, en el cual dichos agentes cooperan para lograr tanto sus metas individuales como colectivas. El buen funcionamiento de un SMA se basa en el establecimiento de protocolos eficientes de comunicación y coordinación (Bravo, Aguilar, & Rivas, 2001).

1.4.1 Características de los SMA

Algunas de las características mencionadas por Romero son las siguientes:

- Cada agente no tiene información completa ni capacidad para resolver el problema.

- Tienen puntos de vista limitados.
- No hay un sistema de control global.
- Los datos están descentralizados.
- Computación asíncrona (Romero, 2009-2010).

1.5 Estándar FIPA

FIPA es un estándar que permite mantener la comunicación entre agentes a través de una estructura de mensajes, según Navarrete FIPA "se denomina Arquitectura Abstracta porque define los elementos comunes a diferentes sistemas subyacentes" (Navarrete, 2008).

Esta arquitectura abstracta está compuesta por los siguientes elementos según Alonso.

- Un directorio de agentes (páginas blancas): se encarga de registrar a los agentes y la dirección o ruta a donde enviar los mensajes.
- Un directorio de servicios (páginas amarillas): se encarga de registrar las capacidades que los agentes ponen a disposición (servicios).
- Un mecanismo de transporte de mensajes (routing): se encarga del envío de los mensajes entre agentes dentro y fuera de una plataforma.
- Un lenguaje de comunicación de agentes (ACL): define los mensajes que pueden utilizar los agentes para comunicarse (Alonso, 2015).

Arquitectura FIPA

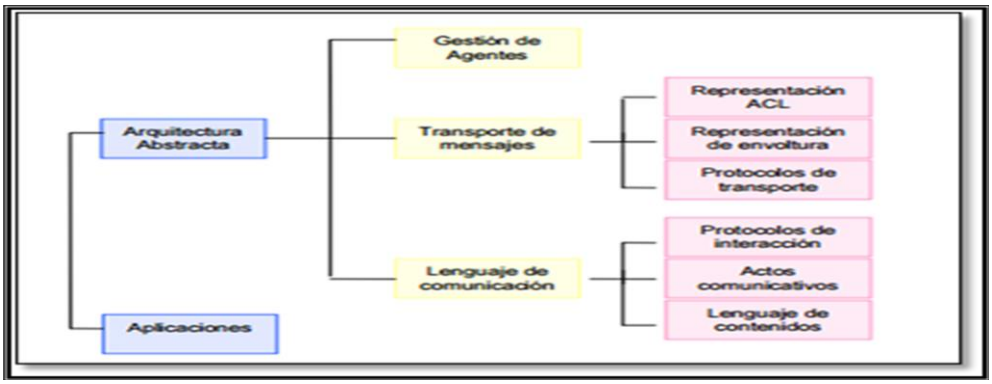


Figura 2. Arquitectura FIPA.
En la figura se muestra la arquitectura de FIPA que es una arquitectura abstracta y cubre todos los aspectos de un entorno de agentes.
Fuente: (Navarrete, 2008, pág. 24).

1.6 Comunicación FIPA-ACL

FIPA-ACL permite mantener una comunicación interna entre los agentes. FIPA es un protocolo de la plataforma JADE mientras que ACL es un Lenguaje de Comunicación de Agentes.

Un mensaje FIPA-ACL representa la intención de realizar alguna acción, es decir, cada mensaje puede ser visto como una acción del emisor, que trata de modificar el "estado mental" del receptor (Navarrete, 2008).

Componentes del Lenguaje de Comunicación FIPA-ACL

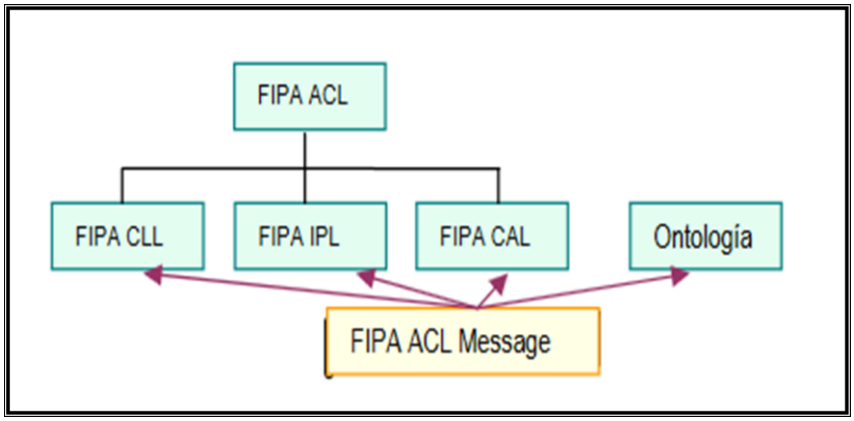


Figura 3. Componentes del Lenguaje de Comunicación FIPA-ACL.
Proceso comunicativo entre los estados de FIPA obteniendo un mensaje ACL.
Fuente: (Navarrete, 2008).

La figura muestra que los mensajes FIPA-ACL están formados por:

- Lenguaje de Contenidos FIPA (Content Language Library CLL): el cual es utilizado para expresar el contenido de una comunicación entre agentes.
- Actos Comunicativos FIPA (Communicative Act Library CAL): son la acción que realiza el mensaje (inform,request etc).
- Protocolos de Interacción FIPA (Interaction Protocol Library (IPL): son patrones de intercambio de mensajes que se presentan entre agentes.

Además de que presentan una ontología que es un conjunto de símbolos con una interpretación asociada que puedan ser compartidos por una comunidad de agentes y esta puede ser opcional.

1.6.1 Elementos de FIPA-ACL

En la tabla que se muestra a continuación se podrán enlistar los principales elementos de FIPA-ACL mencionados por (Ramirez, 2008).

Tabla 1.

Elementos de FIPA-ACL

Elementos	Descripción
Accept-proposal	Aceptación de una propuesta para realizar cierta acción.
Agree	Acuerdo para realizar cierta acción.
Cancel	Informa a un agente que el emisor no tiene intención de que el segundo efectúe cierta acción.
Call-for-proposal	Pide candidaturas para realizar cierta acción.
Confirm	Informa al receptor que cierta proposición es cierta, siempre que sea cierto que el receptor tiene incertidumbre sobre ella.
Disconfirm	Contrario de Confirm.
Failure	Informa que se intentó cierta acción, pero falló.
Inform	Informa al receptor de que cierta proposición es cierta.
Inform-if	Informa si el agente que recibe la acción cree que la sentencia es verdad, informara de manera afirmativa y si no indicara que es falsa.
Inform-ref	Comunica al receptor una descripción referencial de un objeto.
Not-understood	Informa al receptor que recibió cierta petición de acción pero que no la entendió.
Propagate	El receptor recibe un mensaje y además debe propagarlo al resto de la lista de receptores.
Propose	Se remite una propuesta para realizar cierta acción bajo ciertas precondiciones.

Proxy	El emisor quiere que el receptor (broker) seleccione agentes objetivos a partir de cierta descripción y les envíe cierto mensaje.
Query-if	Pregunta a otro agente si la proposición es cierta o no.
Query-ref	Preguntar a otro agente por el objeto referenciado en una expresión
Refuse	Se niega a realizar una acción determinada.
Reject-proposal	Acción de rechazar una propuesta para realizar alguna acción durante una negociación.
Request	El emisor desea que el receptor realice cierta acción.
Request-when	Se desea que el receptor realice una acción tan pronto cuando cierta proposición sea cierta.
Request-whenever	Solicitar al receptor que realice alguna acción de forma continua.

Nota: Con los elementos mencionados anteriormente se podrá mantener una comunicación exitosa entre los agentes.

Fuente: (Ramirez, 2008).

1.7 Netbeans

NetBeans es un entorno de desarrollo muy completo y profesional. Contiene muchas funcionalidades, para distintos tipos de aplicaciones y para facilitar al máximo la programación, la prueba y la depuración de las aplicaciones que se desarrollan. También incorpora un editor propio (Hernández).

En el podemos realizar las tareas asociadas a la programación como son:

- Editar el código.
- Compilar.
- Ejecutar.

- Depurar.

1.7.1 Características

Las características más relevantes según Gimeno & González son:

- Simplifica algunas de las tareas que sobre todo en proyectos grandes son tediosas.
- Nos asiste (parcialmente) en la escritura de código, aunque no libera de aprender el lenguaje de programación.
- Ayuda en la navegación de las clases predefinidas en la plataforma.
- Aunque puede ser costoso su aprendizaje, los beneficios superan las dificultades (Gimeno & González , 2011).

1.7.2 Profile Project

Profiler es una herramienta de monitoreo de Java con todas las funciones integradas en el IDE NetBeans. Las características incluyen la CPU, memoria, hilos y clases cargadas, heap, lo relevante de esto es que se puede ver el comportamiento de la aplicación y obtener indicadores de cómo y cuantos recursos consume, cuantos objetos se crean, permitiendo a los desarrolladores sean más productivos en la solución de problemas de rendimiento y de memoria (netbeans.org).

Monitorización de una aplicación: se puede obtener información sobre varias propiedades importantes de la JVM, al ejecutar el IDE lanza la aplicación y la ventana de perfiles se abre en el panel izquierdo. La ventana de perfiles contiene los controles que le permiten hacer lo siguiente (netbeans.org).

- Controlar la tarea de perfiles.
- Ver el estado de la tarea actual de perfiles.
- Visualización Resultados de perfiles.
- Administrar los resultados de descripción instantáneas.

- Ver las estadísticas básicas de telemetría.

Puede utilizar la ventana de telemetría general para obtener rápidamente una visión general de los datos de seguimiento en tiempo real como lo indica en (netbeans.org):

- *Heap size*: cuando un programa Java inicia Java Virtual Machine consigue un poco de memoria de sistema operativo es decir que cada vez que se crea un objeto con nuevo se asigna memoria desde Heap y cuando el objeto muere, la memoria se remonta al espacio de almacenamiento dinámico en Java.
- *Used Heap*: tamaño de memoria utilizada según el criterio mencionado anteriormente.
- *Surviving Generations*: métricas que pueden detectar posibles fugas de memoria. Una generación es un conjunto de instancias creadas dentro del mismo intervalo de GC. Sobrevivir a una generación es una generación que sobrevive al menos una recolección de basura.
- *Threads*: hilos o procesos que existen en la aplicación.
- *Loaded Classes*: clases cargadas y ejecutadas en la aplicación.

Gráficas del Profiler

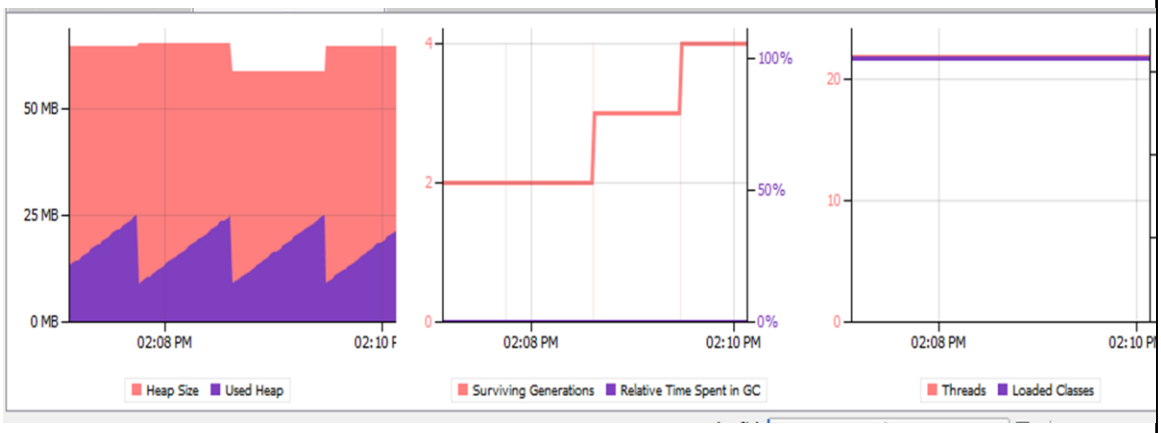


Figura 4. Gráficas del Profiler.

Muestra el rendimiento de la CPU en un intervalo de tiempo según las interacciones que se realizan en la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

La Figura 4 muestra que la memoria utilizada es aproximadamente 25B al ejecutar la aplicación durante un intervalo de tiempo por lo cual la generación de supervivencia inicia en 2 un (50%) y aumenta gradualmente hasta llegar a 4 un (100%) y finalmente se muestra los hilos y clases aproximadamente más de 20.

Analizar el rendimiento de la CPU: El IDE monitoriza el rendimiento de la CPU en un tiempo de ejecución de la aplicación y procesa los resultados en tiempo real. Se obtendrá datos sobre el rendimiento de la aplicación, incluyendo el tiempo para ejecutar los métodos y el número de veces que se invoca el método (netbeans.org).

Analizar el uso de memoria: le da los datos a los objetos que han sido asignados en la aplicación, tales como el número, tipo y ubicación.

1.8 Plataforma Jade (Framework de desarrollo de Agentes Java).

Jade es una plataforma de desarrollo que utiliza Java para la creación de agentes o sistemas multi-agente. Jade utiliza el estándar FIPA para el intercambio de mensajes entre agentes ya que cada agente es representado por un hilo (thread).

1.8.1 Características Jade

A continuación, se detallarán algunas características de JADE según Garamendi.

- Los agentes tienen nombres únicos y proporcionan servicios.
- Utilizan mensajes asíncronos, FIPA-ACL (Agent Comunication Language) como lenguaje de comunicación.
- Cada agente se implementa como un thread.
- Distintas plataformas pueden comunicarse vía protocolos http o iiop.
- Incorpora un conjunto de agentes auxiliares (Garamendi, 2004).

1.8.2 Entorno Jade

Jade necesita de ciertos componentes para su funcionamiento los cuales son detallados a continuación por Garamendi:

- Los agentes necesitan del entorno de ejecución para poder “vivir”.
- Contenedor: instancia del entorno de ejecución.
- Plataforma: capa que oculta el entorno donde se ha decidido ejecutar la aplicación, es un conjunto de contenedores activos.

JADE = Main Container + Agentes auxiliares (RMA, Dummy agent, Sniffer agent, DF, AMS) (Garamendi, 2004).

Monitoreo remoto de agentes

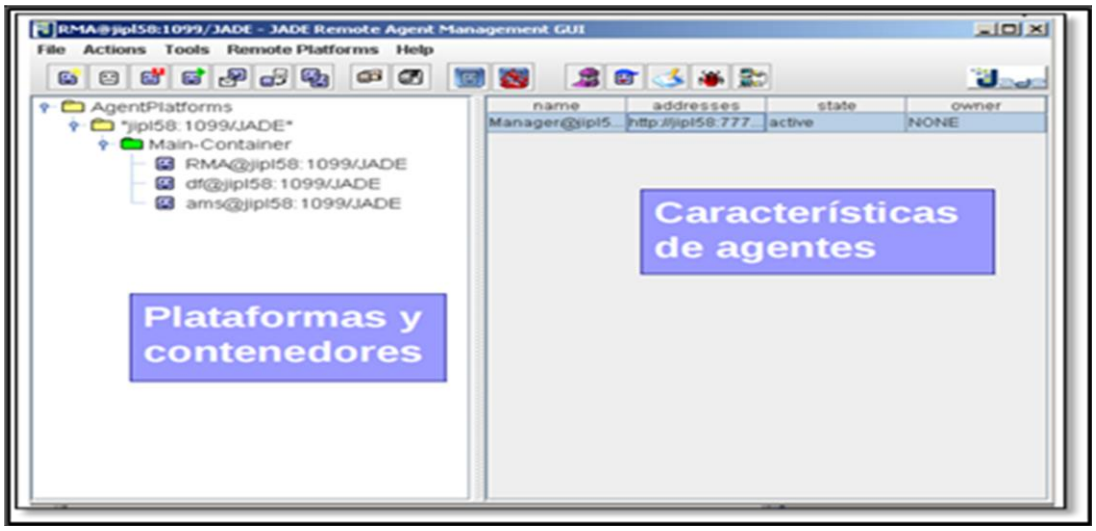


Figura 5. Monitoreo remoto de agentes.

En esta figura se muestra la plataforma y contenedores donde se muestran los agentes auxiliares de JADE bajo Main Container.
Fuente: (Garamendi, 2004).

1.8.3 Main Container

Contenedor principal el cual almacena los agentes auxiliares y los agentes que serán creados con su respectivo nombre dirección y puerto.

Estructura de Jade

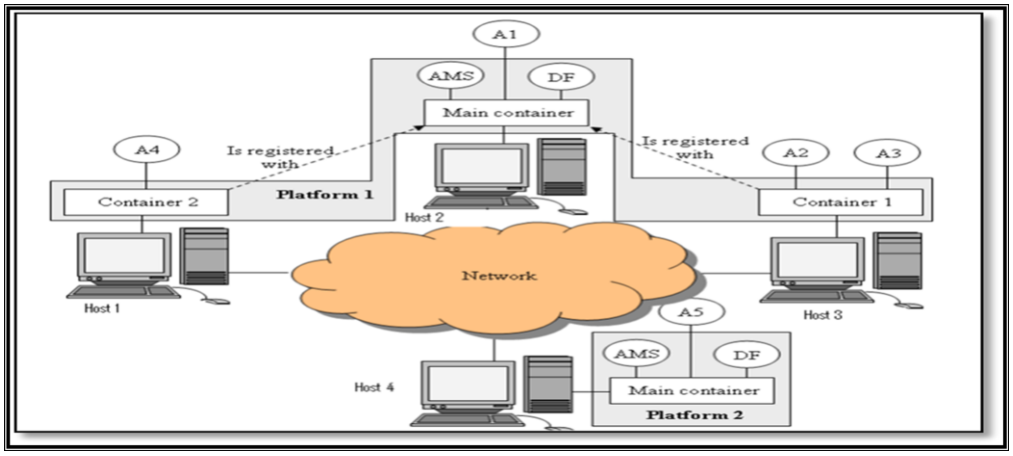


Figura 6. Estructura de Jade.

En la imagen se visualiza la plataforma de Jade la cual tiene un contenedor principal llamado main donde se encuentra el AMS y DF.
Fuente: (Morales, 2009).

1.8.3.1 RMA (Agente de Monitoreo Remoto)

Es el gestor de la plataforma del cual Garamendi dice que RMA permite:

- Iniciar, suspender, reiniciar agentes.
- Matar agentes o contenedores.
- Mandar mensajes.
- Clonar agentes.
- Añadir o quitar plataformas remotas (Garamendi, 2004).

1.8.3.2 AMS (agente de administración de sistema)

Es el agente que supervisa toda la plataforma, "Es el punto de contacto para todos los agentes que necesitan interactuar con las páginas blancas de la plataforma, así como administrar su ciclo de vida. Cada agente debe registrarse con el AMS para obtener un Identificador de agente Jade (AID) válido." (Manotas, 2010).

1.8.3.3 DF (el facilitador de directorio)

La definición mencionada por Aranda dice “que en la interfaz del facilitador de directorio (DF) se registran los servicios que ofrece cada agente, pudiéndose dar más de uno por agente”.

Los servicios que ofrece el DF son:

- Ver descripciones de los agentes registrados.
- Registrar y eliminar agentes.
- Modificar registros.
- Buscar descripciones (Aranda, 2008).

La ventana de la plataforma JADE con los agentes auxiliares y el contenedor principal se ve en la figura 7.

Estructura de los elementos de JADE.

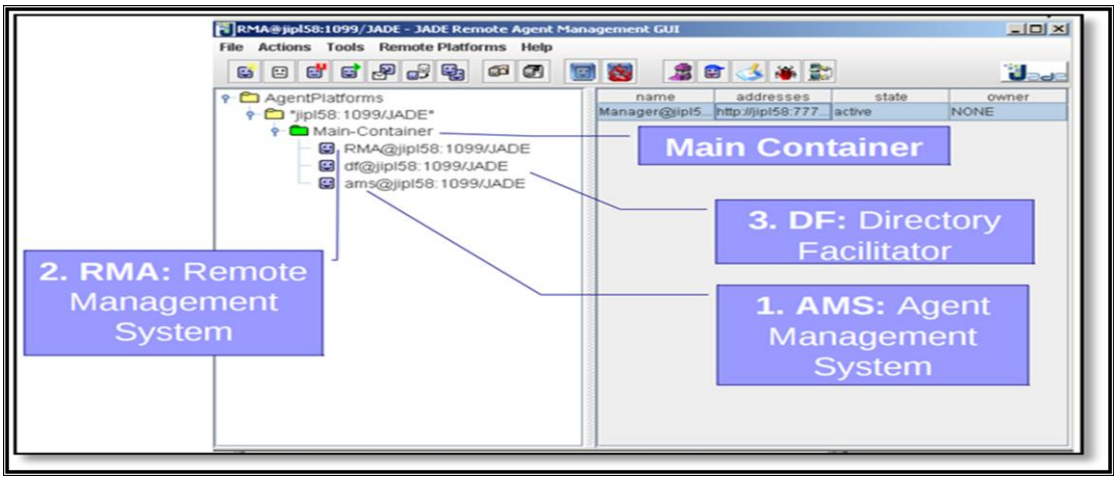


Figura 7. Estructura de los elementos de JADE.

Los componentes de la plataforma Jade se visualizan en la parte izquierda de la figura, donde el componente raíz es la Plataforma de agentes *(AgentPlatforms*), bajo este se halla la dirección de la máquina local donde se está ejecutando la plataforma con su respectivo puerto, y al final se halla el contenedor principal (Main- Container) del cual se derivan los agentes auxiliares.
Fuente: (Aranda, 2008).

1.8.4 Dummy Agent

Para Garamendi, el agente Dummy actúa como intermediario en la comunicación entre agentes, ya que: Compone los mensajes ACL de terceros y los envía entre ellos o también estos mensajes pueden ser almacenados y empleados posteriormente. El agente dummy puede ser llamado desde su ícono o desde el agente auxiliar RMA (Garamendi, 2004).

Dummy Agent

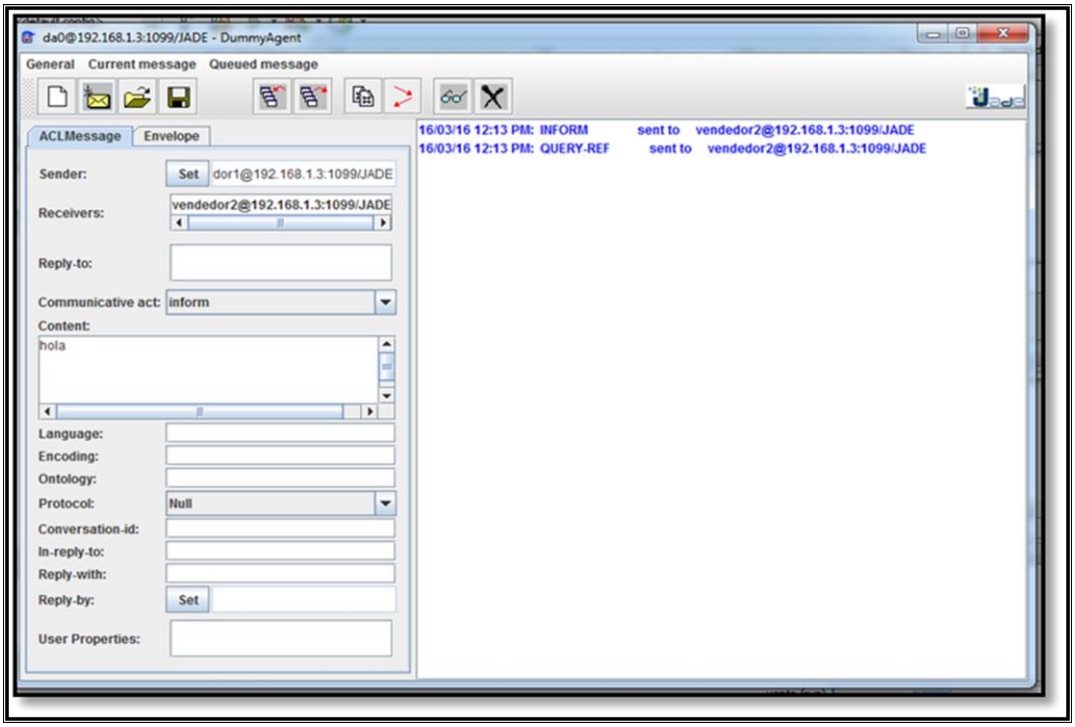


Figura 8. Dummy Agent.

En esta figura se observa el envío de mensajes ACL entre agentes mediante la utilización de Dummy Agent para lo cual se debe seleccionar el agente que envía y el agente que recibe el mensaje, seleccionar el acto comunicativo y el protocolo los cuales pueden ser: (Inform, Query-Ref, Request) entre otros.
Fuente: (Garamendi, 2004).

1.8.5 Sniffer Agent.

“Herramienta para rastrear los mensajes intercambiados entre agentes a través de la plataforma. Permite seleccionar el agente o los agentes que están bajo observación y capturar sus mensajes de entrada y salida (Fernández J. , 2013).

Sniffer Agent

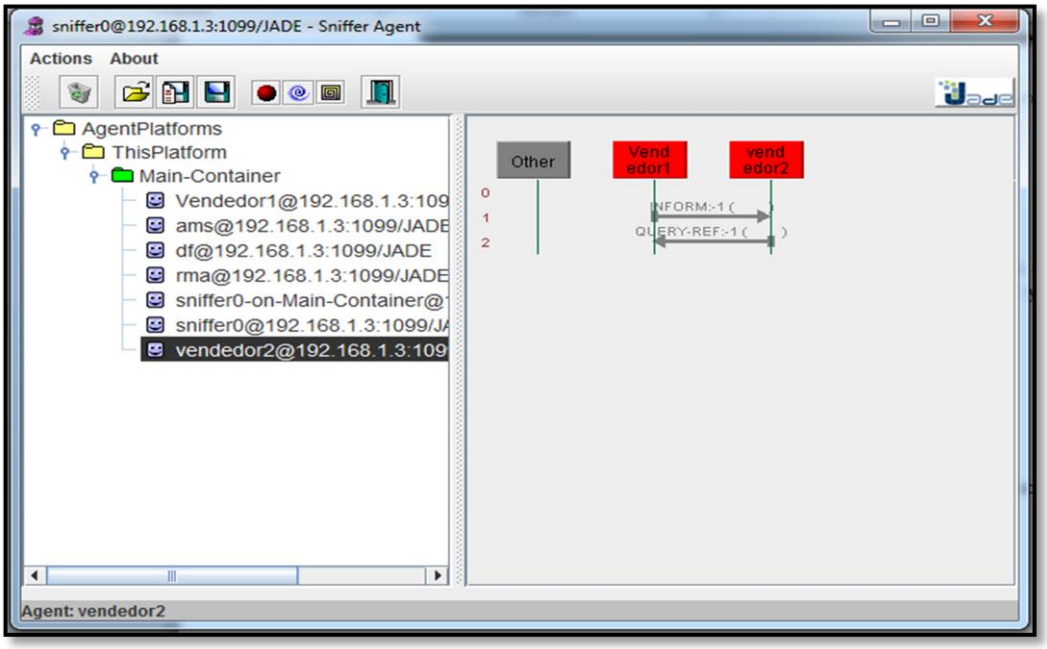


Figura 9. Agente Sniffer.

Muestra la interacción que se produce ente los agentes en este caso agente vendedor1 y agente vendedor2, es decir la secuencia del envío de los mensajes entre ellos.
Fuente: (Fernández J. , 2013).

1.8.6 Introspector Agent.

Monitoriza y controla el ciclo de vida de un agente en ejecución, así como su cola de recepción de mensajes (Fernández J. , 2013).

Introspector Agent

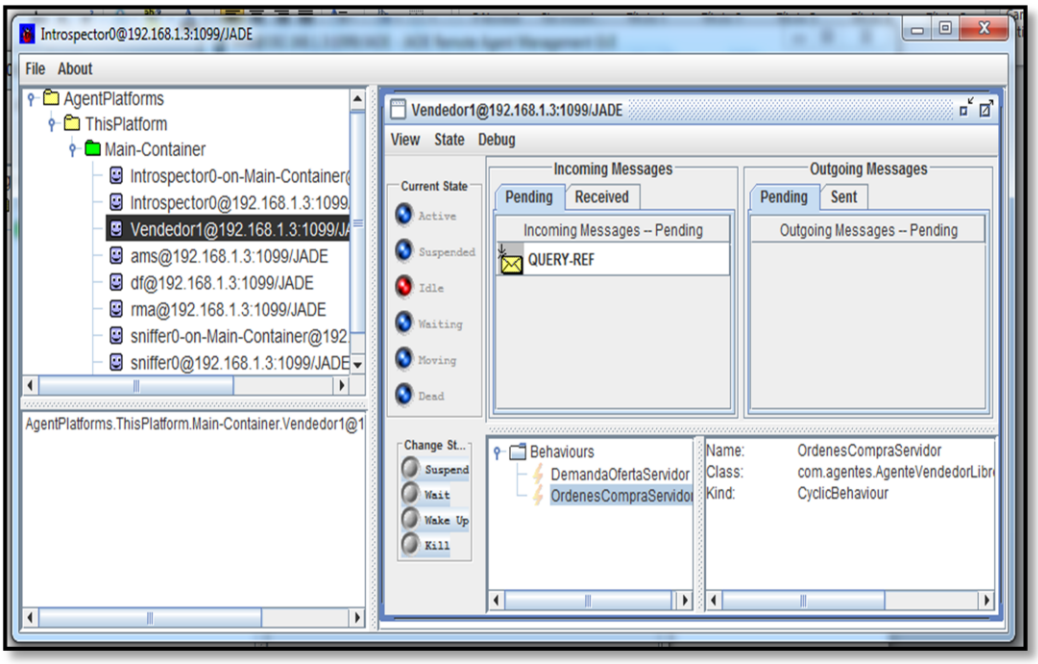


Figura 10. Introspector Agent.
Para la monitorización se debe seleccionar un agente de la lista para este caso agente vendedor1 el cual muestra la información de los mensajes y su respectivo comportamiento.
Fuente: (Fernández J. , 2013).

1.9 Comportamientos de JADE

Según la investigación realizada por Oviedo los comportamientos de JADE "se utilizan básicamente para el envío y recepción de mensajes" (Oviedo, 2010), mientras que Aranda dice que "los agentes realizan las acciones y las percepciones mediante los comportamientos de JADE" (Aranda, 2008).

En el estudio realizado por Venturini se detallan los principales comportamientos de Jade que se presentan en la tabla 2 (Venturini, 2012, pág. 219).

Tabla 2.

Comportamientos Agentes en JADE

Agente	Comportamiento	Tipo de Comportamiento	Función
Localizador	DetectClient	DetectClient	Detecta la presencia de un agente cliente.
Intermediario	ReplyToRegister	CyclicBehaviour	Recibe mensajes del agente cliente para ser registrado.
termediario	WarnProvider	OneShotBehaviour	Advierte al agente proveedor sobre la presencia de un agente usuario.
Usuario	ReceiveMsg AskForAgreements MakeRecommendation ResponseRecommendation AskForReputation ResponseReputation	CyclicBehaviour OneShotBehaviour CyclicBehaviour OneShotBehaviour Behaviour CyclicBehaviour	Recibe mensajes de los proveedores u otros agentes clientes. El cliente realiza consultas al proveedor.
Proveedor	Register RequestNegotiation	OneShotBehaviour CyclicBehaviour	Registrar un proveedor en el sistema. Creado para aceptar o rechazar alguna

	OfferRequestServices	Behaviour	propuesta del cliente. Ofrece productos/ servicios a los clientes y se queda a la espera de alguna consulta para finalmente concretar o no la negociación.
Administrador de usuarios	UserFiltering	OneShotBehaviour	Creado para filtrar agentes usuarios con intereses similares.

Nota: En esta tabla se podrá visualizar varios tipos de comportamientos con una breve descripción de su funcionamiento.

Fuente: (Venturini, 2012, pág. 219).

1.9.1 Métodos de un Comportamiento

Entre los comportamientos de Jade, todas las clases que hereden Behaviour, presentarán dos métodos principales que son el método action () y el método done ().

Método action ()

"Este método define la acción a ser ejecutada cuando se inicie el comportamiento. Debe incluir el código de las acciones a realizar cuando se ejecute el comportamiento" (Oviedo, 2010).

Método done ()

"Es invocado cuando finaliza la ejecución del método action (). Este método determina si el comportamiento ha sido completado o no. Devuelve un valor booleano: true si ha terminado, o false en caso contrario" (Oviedo, 2010).

1.10 Programación Orientada a Objetos

Según Alvarez “Es una forma especial de programar, más cercana a como se expresa las cosas en la vida real que otros tipos de programación” (Alvarez, 2001).

1.11 Elementos de la programación orientada a objetos (POO)

En la programación orientada a objetos se tiene elementos que son necesarios al momento de desarrollar las aplicaciones y que deben ser de esencial conocimiento para el programador.

1.11.1 Objeto

“Un objeto no es más que un conjunto de variables (datos) y métodos (funciones) relacionados entre sí. Los objetos en programación se usan para modelar objetos o entidades del mundo real.” (Izquierdo).

1.11.2 Clase

“Una clase es una plantilla que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo” (Izquierdo).

1.11.3 Método

Según Mestras “Los métodos son acciones que puede realizar un objeto y pueden devolver un valor al acabar su ejecución” (Mestras, 2007).

1.12 Programación Orientada a Agentes (POA)

Para Ahogado & Reinemer "La POA utiliza un conjunto de componentes llamados agentes que se caracterizan entre otras cosas por su autonomía, pro actividad y su habilidad para comunicarse para el desarrollo de aplicaciones. Está motivada por la necesidad de manejar arquitecturas abiertas que evolucionen continuamente, logrando así satisfacer nuevos requerimientos" (Ahogado & Reinemer, 2002).

1.13 Programación Orientada a Objetos vs Programación Orientada a Agentes

En la tabla 3 se detallarán las características de la Programación Orientada a Objetos vs la Programación Orientada a Agentes según (Azawi & Ayesh, 2013).

Tabla 3.
Características de la Programación Orientada a Objetos vs la Programación Orientada a Agentes

Características	POO	POA
Unidad básica.	Objeto.	Agente.
Unidad básica de estados.	Sin restricciones.	Creencias, capacidades,objetivos.
Proceso de cálculo.	Métodos de Respuesta y Paso de Mensajes.	Métodos de respuesta y paso de mensajes.
Tipo de mensajes.	Sin restricciones.	inform,request,offer,refuse.
Limitaciones en	Ninguna.	Registro del protocolo, páginas

métodos.		amarillas y directorios.

Nota: En esta tabla se visualiza las diferencias de las características entre Programación Orientada a Objetos (PPO) y Programación Orientada a Agentes.
Fuente: (Azawi & Ayesh, 2013).

1.13.1 Unidad básica

Los objetos son abstracciones de cosas mientras que los agentes son abstracciones de seres inteligentes y autónomos con las creencias, metas, deseos, etc. (Azawi & Ayesh, 2013).

1.13.2 Parámetros que definen el estado de la unidad básica

Los agentes contienen creencias, planes, objetivos, y se comunican entre sí a través de actos comunicativos (mensajes) tales como: Información (inform), Petición (request), Oferta (offer), Rechazo (refuse) (Azawi & Ayesh, 2013).

1.13.3 Proceso de cálculo

El mensaje de un objeto solicita una operación y el mensaje guarda un formato propio. El intermediario de mensajes tiene la tarea de hacer coincidir cada mensaje con una llamada a un método. Los objetos se comunican a través de interfaces de métodos fijo mientras que los agentes se comunican a través de un Agente de Comunicación (ACL) (Azawi & Ayesh, 2013).

La comunicación basada en agentes utiliza el lenguaje de comunicación (ACL) con un formato preciso e inequívoco (Azawi & Ayesh, 2013).

1.13.4 Tipo de mensaje

Objetos son convencionalmente pasivos, con sus métodos se invocan llamadas para envío de mensajes (Azawi & Ayesh, 2013).

Los agentes son comúnmente considerados como entidades autónomas, porque pueden trabajar para su propio conjunto de responsabilidades internas. Además, los agentes son entidades interactivas que son capaces de utilizar formas dinámicas de mensajes, así como informar a los agentes de eventos particulares (Azawi & Ayesh, 2013).

1.13.5 Restricción del método

Las clases de objetos están diseñados para ser predecibles con el fin de facilitar la reutilización de componentes.

Los agentes están diseñados comúnmente para determinar su comportamiento en base a las metas individuales y estados, así como los estados de las conversaciones en curso con otros agentes.

En implementaciones orientadas a objeto sólo permite que un objeto haga lo que las interfaces soporten. En Objetos, no existe ninguna disposición para que un objeto pueda "anunciar" sus interfaces.

A diferencia de implementaciones con agentes que emplean mecanismos como registro del protocolo, páginas amarillas y directorios; bajo un modelo de comunicación asíncrono (Azawi & Ayesh, 2013).

CAPÍTULO 2

METODOLOGÍA ORIENTADA A AGENTES

La metodologia INGENIAS, según el estudio realizado por Pavon es ” una metodologia que proporciona soluciones de ingenieria para el desarrollo de sistemas multi-agentes”. El estudio de la metodologia Ingenias precenta cinco aspectos importantes en su desarrollo:

I).Modelo de organización, permitirá estructurar un sistema multi-agente dependiente de los roles. Los roles especifican los objetivos o metas para los que fue creado el agente.

II). Modelo de agente, en este aspecto los agentes presentan: Responsabilidades las cuales permiten a los agentes cumplir con su objetivo, control el cual les permite ver que el objetivo sea cumplido y estado mental el cual proporciona creencias, deseos e intenciones al agente.

III).Modelo de objetivos y tareas, permitira la identificación de objetivos generales y la descomposición en objetivos más específicos (tareas) que se asignan a los agentes.

VI). Modelo de interacción, en este aspecto se podra visualizar las interacciones que existen entre los agentes y los roles (los objetivos).

V). Modelo de entorno, especifica las entidades y recursos con los que interactúa un sistema multi-agente (Pavon, s.f.).

2.1 Herramienta AUML (Agent Unified Modeling Language)

El lenguaje de modelado unificado de agentes (AUML) es una herrameinta basada en el estudio de Programacion Orientada a Agentes (POA) y surge de la necesidad de analizar a los agentes.

Según el estudio de Prieto define los sigientes conceptos de UML y AUML como:

UML es un lenguaje gráfico para visualizar, especificar y costruir un modelo de sistema basado en la Programacion Orientada a Objetos.

AUML es un lenguaje gráfico para la visualizar, especificar y costruir los modelos de sistemas basados en la programacion orientada a agentes (Prieto, 2013).

2.2 Arquitectura AUML.

AUML según González presenta tres tipos de especificaciones mencionadas como:

- Junta de Arquitectura (Arquitecture Board): responsable final de aprobar los planes de los grupos de trabajo y de los comités técnicos.
- Grupos de Trabajo (Working Groups): parte informal de las especificaciones y se encarga de la identificación de campos de aplicación y prueba.
- Comités Técnicos (Technical Committees - TCs): se encargan de la producción de especificaciones formales (González, 2005).

2.3 Casos de Estudio

Para el desarrollo de este proyecto técnico se usará el siguiente caso de estudio que constará en analizar las etapas existentes en la elaboración de una aplicación demo (Escritorio).

La aplicación tiene como objetivo realizar una subasta de libros en la cual se crearán agentes vendedores y estos agentes ingresarán el título y el precio del libro, que serán registrados en el catálogo, se crearán agentes compradores los cuales ingresarán el libro que desean comprar, se realiza una búsqueda interna en el caso de que encuentre el libro objetivo se desplegará la información de los agentes vendedores registrados, el título del libro con el precio más bajo el cual será vendido, caso contrario se desplegará un mensaje de que el libro no se encuentra registrado en el catálogo.

Con el desarrollo de este ejemplo se visualizarán las interacciones de los agentes los cuales se comunicarán mediante el protocolo FIPA-ACL, el mismo que será visualizado a través de la plataforma Jade.

La herramienta que se utilizó para la elaboración de los diagramas de esta aplicación fue UML-Diagrammer.

2.3.1 Diagrama de Interacción

El diagrama de interacción muestra la comunicación entre entidades. Para el desarrollo de este diagrama se seguirán los siguientes pasos que son:

I).En el primer paso, se identificarán los agentes que actúan con sus respectivos roles: *Agente vendedor* con su rol vender, *Agente comprador* con su rol comprar.

II). En el segundo paso, se identificarán las interaciones de los agentes, las cuales corresponden al protocolo fipa-query y fipa-request: la interacción entre *registro titulo-libro y precio* con el protocolo fipa-query mientras que la interacción *Solicitar libro* se lo hizo con el protocolo fipa_request.

III). En el tercer paso, se identificarán los objetivos que cada interacción persigue: el protocolo fipa-query tiene como objetivo la interacción *Registro en catalogo* mientras que el protocolo fipa-request tiene como objetivo la interacción *Libro objetivo a buscar*.

Los siguientes pasos mencionados, en el diagrama de interacción podrán ser visualizados en la Figura11 que se muestra a continuación.

Diagrama de Interacción

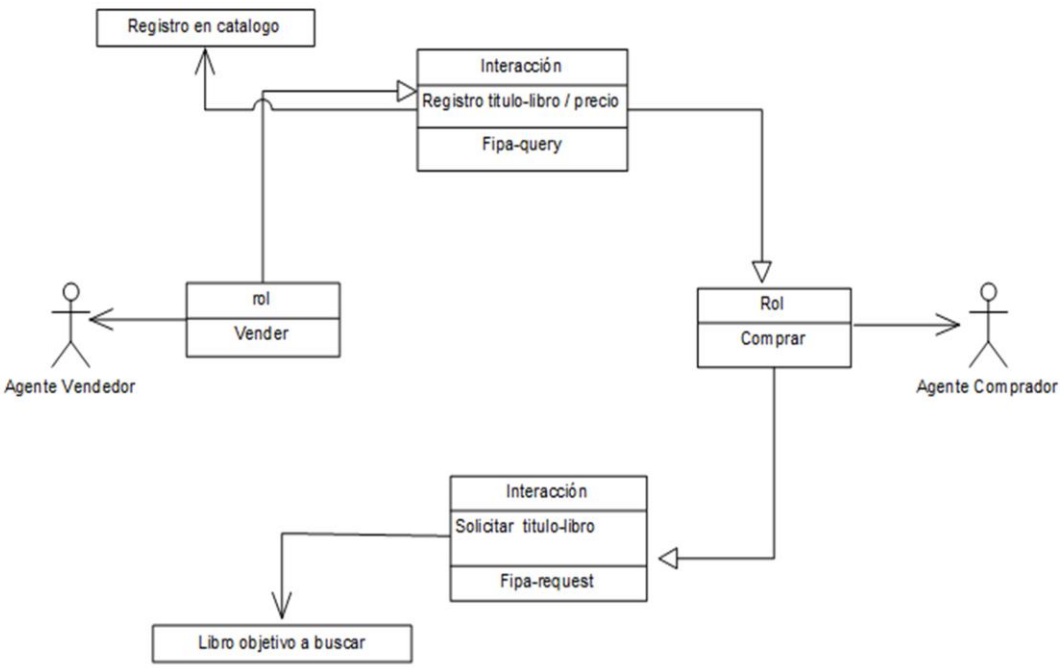


Figura 11. Diagrama de Interacción.
La figura muestra la interacción que existe entre el agente vendedor y el agente comprador, identificando el objetivo (rol) principal de cada agente.
Elaborado por Evelyn Obando & Doris Tituaña.

2.3.2 Diagrama Protocolos de interacción.

El diagrama de protocolos tomará en cuenta los siguientes pasos:

I).En el primer paso, se incluirá los roles(objetivos) identificados en al diagrama de interacciones, formando asi un árbol de elementos de cada uno de ellos.

II). En el segundo paso, se enlistarán las tareas identificadas para cada rol(objetivo) del agente : el rol *vender* tiene como tareas *Ingresar titulo-libro* e *Ingresar precio*, mientras que el rol *comprar* tiene como tareas *libro a comprar* y *nombre comprador*.

III). En el tercer paso,se procede a identificar las interacciones que forman el árbol de elementos de *Catálogo*, cabe mencionar que c*atalogo* es un proceso interno que se ejecutará en la aplicación*: Pregunta precio libro,Pregunta informacion libro, Titulo*

libro,Precio, Responde libro, Verifica disponibilidad libro, Verifica vendedor, Confirma comprador.

VI). En el cuarto paso, se procede a identificar cada una de las relaciones existentes entre las tareas y las interaciones de *Catálogo*, manteniendo así el objetivo de la aplicación. Los siguientes pasos mencionados, en el diagrama de protocolos de interacción, podrán ser visualizados en la Figura12 que se muestra a continuación.

Diagrama Protocolos de Interacción

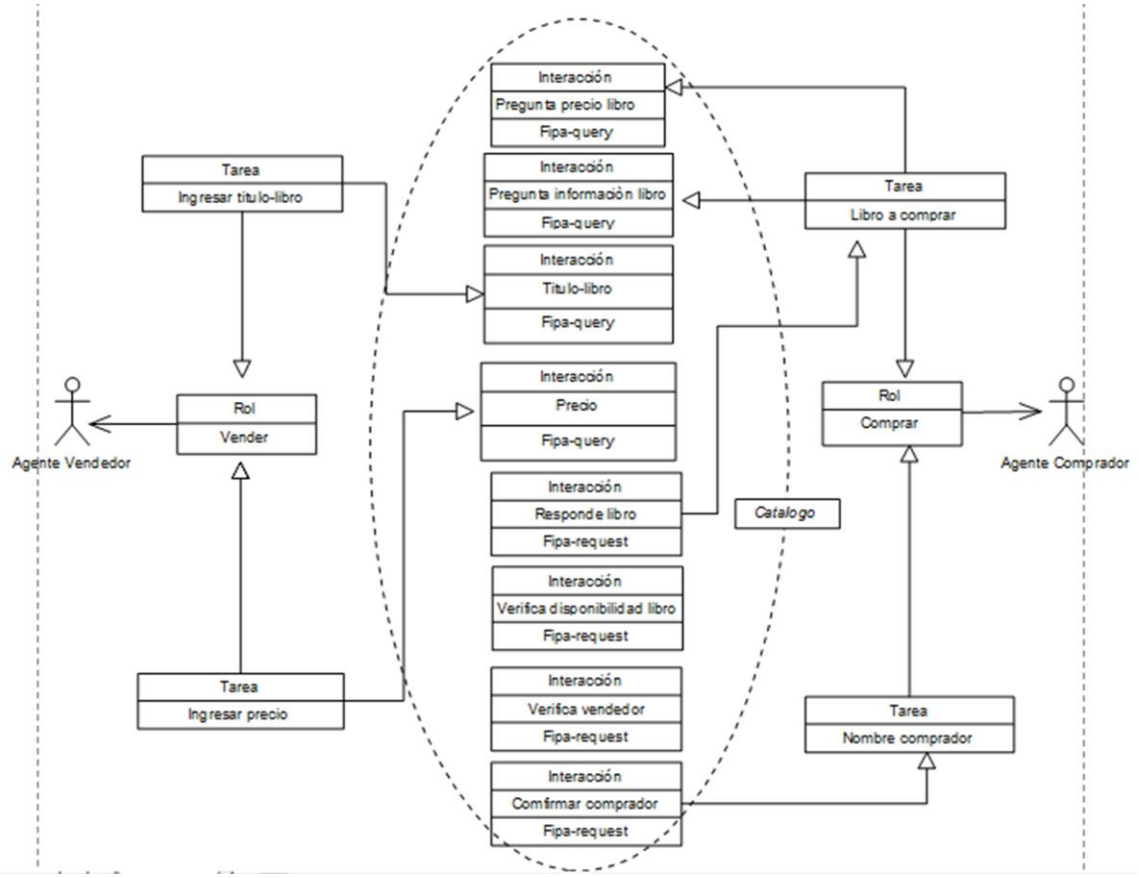


Figura 12. Diagrama Protocolos de Interacción.
La figura muestra el diagrama de protocolos de interacción para los agentes vendedor y comprador. Además muestra las interacciones de *Catálogo* que actúa de forma interna en el proceso.
Elaborado por Evelyn Obando & Doris Tituaña.

2.3.3 Diagrama de Tareas

El diagrama de tareas surge de las interacciones (tareas) que fueron identificadas en el diagrama de protocolos de interacción.

I). En el primer paso, se inserta las tareas ya identificadas anteriormente, definiendo las entradas de cada una de las tareas.

II). En el segundo paso, las entradas de las tareas servirán como cuerpo del mensaje, las cuales tienen relaciones con: las tareas *Ingresa titulo-libro* e *Ingresa precio* son tareas específicas del *agente vendedor* con el rol *vender*. La tarea *libro a comprar* es específica del agente comprador.

Los pasos mencionados anteriormente se visualizarán en la Figura13 que se muestra a continuacion.

Diagrama de Tareas

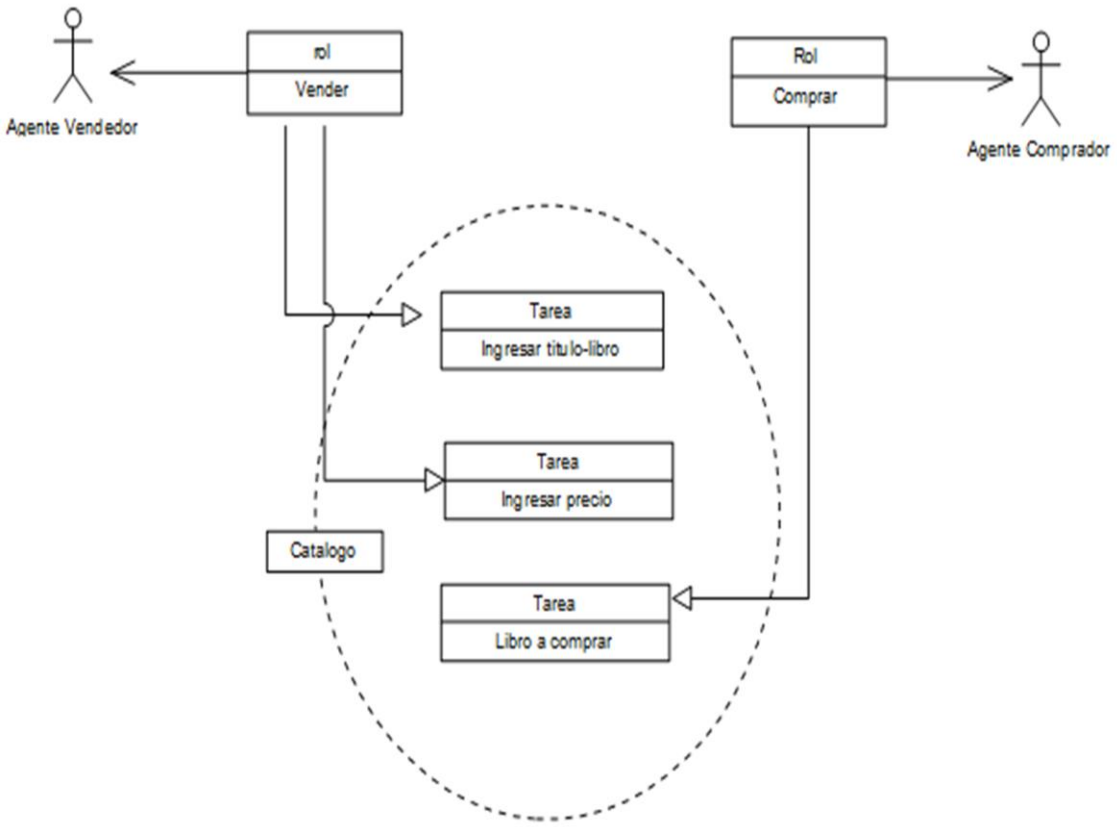


Figura 13. Diagrama de Tareas.

La figura muestra una representación específica de las tareas que debe cumplir cada rol (objetivo). Elaborado por Evelyn Obando & Doris Tituaña.

2.3.4 Diagrama de Secuencias

"El diagrama de secuencias muestra la interacción de agentes y la secuencia de mensajes intercambiados entre ellos" (Prieto, 2013).

Para desarrollar el diagrama de secuencias se tomará en cuenta los siguientes pasos:

I). En el primer paso, se identifican los agentes (actores) *agente vendedor, agente comprador* y las clases que forman la aplicación como son: *AgenteVendedorLibro.java, AgenteCompradorLibro.java, GuiVentaLibro.java y catalogo* (agente interno).

II). En el segundo paso, se procede a ingresar la información, el *agente vendedor* ingresa su *nombre,* el *titulo del libro,* el *precio del libro* utilizando la clase *AgenteVendedorLibro.java*, a su vez esta clase inicia su comunicación con *catalogo* registrando el *título del libro* y su *precio,* mientras que *el agente comprador* ingresa su *nombre* y el *nombre del libro a comprar* utilizando clase *AgenteCompradorLibro.java*.

III). En el tercer paso, el agente comprador realiza la petición del libro que desea comprar es decir el libro objetivo.

VI). En el cuarto paso, se realiza un proceso interno para verificar la existencia del libro, si existe el libro en *catalogo* responde el *título del libro* y el *precio* al agente vendedor , si no existe el libro no realiza ninguna acción.

V). En el quinto paso, una vez vendido el libro al agente comprador el agente finaliza (muere) y el agente comprador puede seguir ingresando libros si así lo desea.

Los pasos mencionados, podrán ser visualizados en la Figura14 que se muestra a continuación.

Diagrama de Secuencia

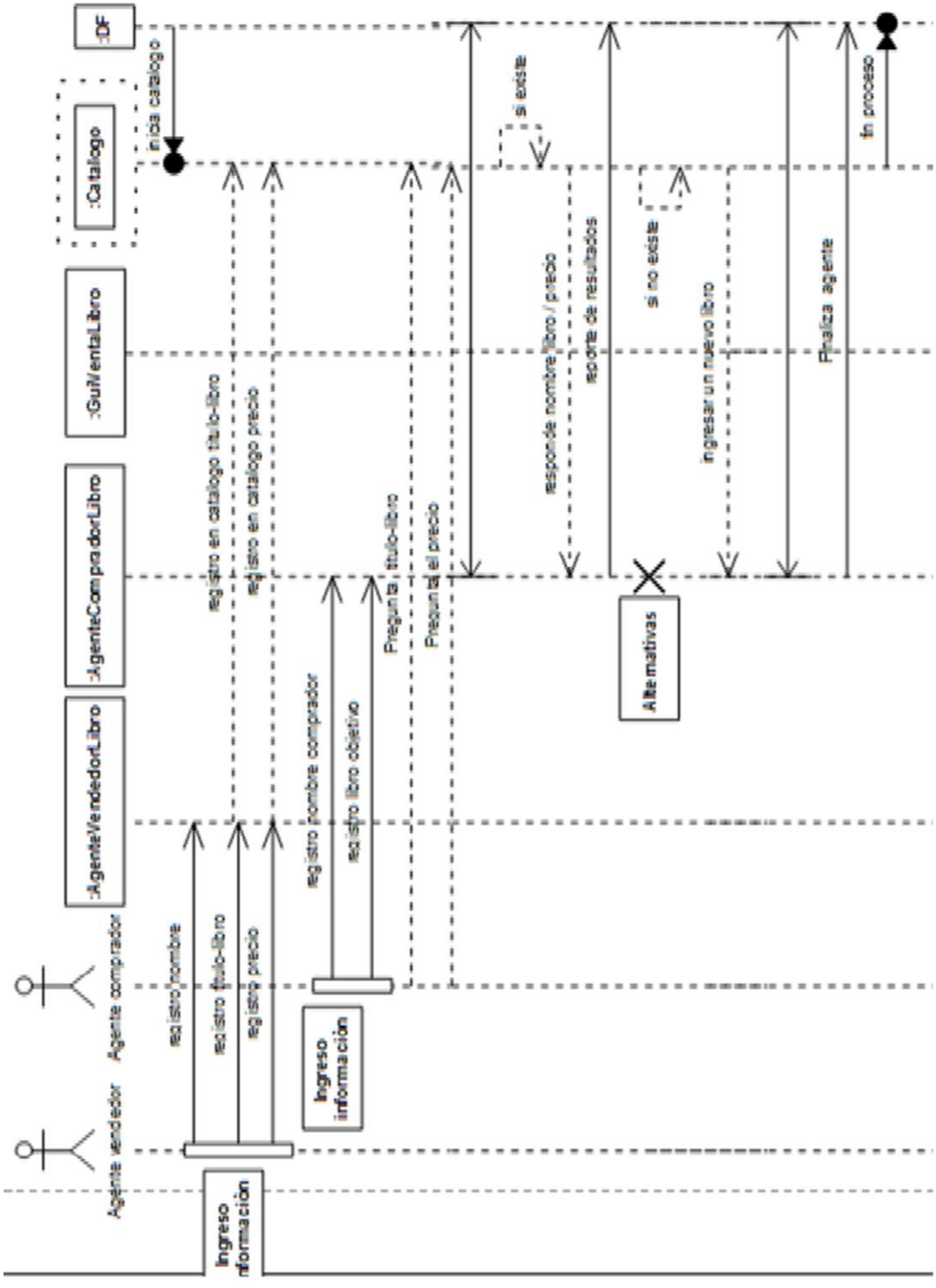


Figura 14. Diagrama de Secuencia.
En esta figura se muestra el diagrama de comunicaciones basado en los elementos relacionados entre los agentes vendedor y comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

La Figura 14 se muestra el envío de mensajes entre los agentes creados y las clases que forman la aplicación. En la primera interacción se podrá ver el ingreso de información entre el *agente vendedor* con la clase *AgenteVendedorLibro* en el cual se ingresa el nombre del comprador, el titulo del libro y el precio del libro, una vez realizado el ingreso se activa *Catalogo* que es un proceso interno en la aplicación. En la segunda interacción se podrá ver el ingreso de información entre le *agente comprador* y la clase *AgenteCompradorLibro* en el cual se ingresa el nombre del comprador y el libro a buscar, este agente a su vez mantiene una comunicación con *Catalogo*, para verficar si el libro que ingreso existe responde el nombre del libro y el precio caso contrario el comprador deberá ingresar un nuevo libro, formando un proceso de alternativas, las mismas que son internas en la aplicación. Al finalizar este proceso se podrá observar el nombre del vendedor el cual registro el libro, el nombre del comprador, el nombre del libro y el precio.

2.4 Diagramas UML Objetos

2.4.1 Diagrama de Caso de Uso

Según Vega define el estudio de un caso de uso como:

Una unidad funcional coherente de un sistema, subsistema o clase, en el cual se podrá identificar sus principales elementos en el esquema del diagrama.

- Actor: un actor podría ser cualquier cosa que se comunica con el sistema y que es externo a él.
- Caso de Uso: muestran las relaciones entre los casos de uso de un sistema y sus actores.
- Relaciones: especifican los tipos de relaciones que pueden presentar los casos de uso como extensión (extend), inclusión (include) (Vega, 2010).

Para el desarrollo del diagrama, se podrá seguir los siguientes pasos:

I). En el primer paso, se deberá identificar los actores que actúan como roles en el sistema, el actor *vendedor* podrá ingresar al caso de uso *catálogo*, el actor *comprador* podrá ingresar al caso de uso *registro_compra*.

II): El segundo paso, consiste en identificar las operaciones existentes conocidas como (casos de uso) que serán ejecutadas por los actores identificados, en este diagrama los casos de uso existentes son*: catalogo, inf_catalogo, in_catalogo, registro_compra, libros, libross, resultado*.

III): En el tercer paso, se identificarán las relaciones existentes entre los elementos tales como:

- El actor *vendedor* tiene una relación con el caso de uso *catálogo*.
- El caso de uso *catalogo* presenta dos relaciones.
    - La relación (include), especifica que el caso de uso *catalogo* necesita del caso de uso *inf_catalogo* para poder cumplir su tarea.
    - La relación (extend), especifica que el caso de uso *in_catalogo* presenta opciones alternativas para el caso de uso *inf_catalogo*.
- El actor *comprador* tiene una relación con el caso de uso *registra_compra*.
- El caso de uso *registra_compra* presenta tres relaciones.
    - La relación (include), en *libros* indica que necesita del caso de uso *registra_compra* para poder cumplir con su tarea.
    - La relación (include), en *resultados* indica que necesita del caso de uso *registra_compra* para poder cumplir con su tarea.
    - La relación (extend), en *libros* indica que presenta algunas alternativas para el caso de uso *libross.*

Los pasos mencionados para realizar el diagrama de casos de uso podran ser visualizados en la Figura 15 que se muestra a continuacion.

Diagrama Caso de Uso

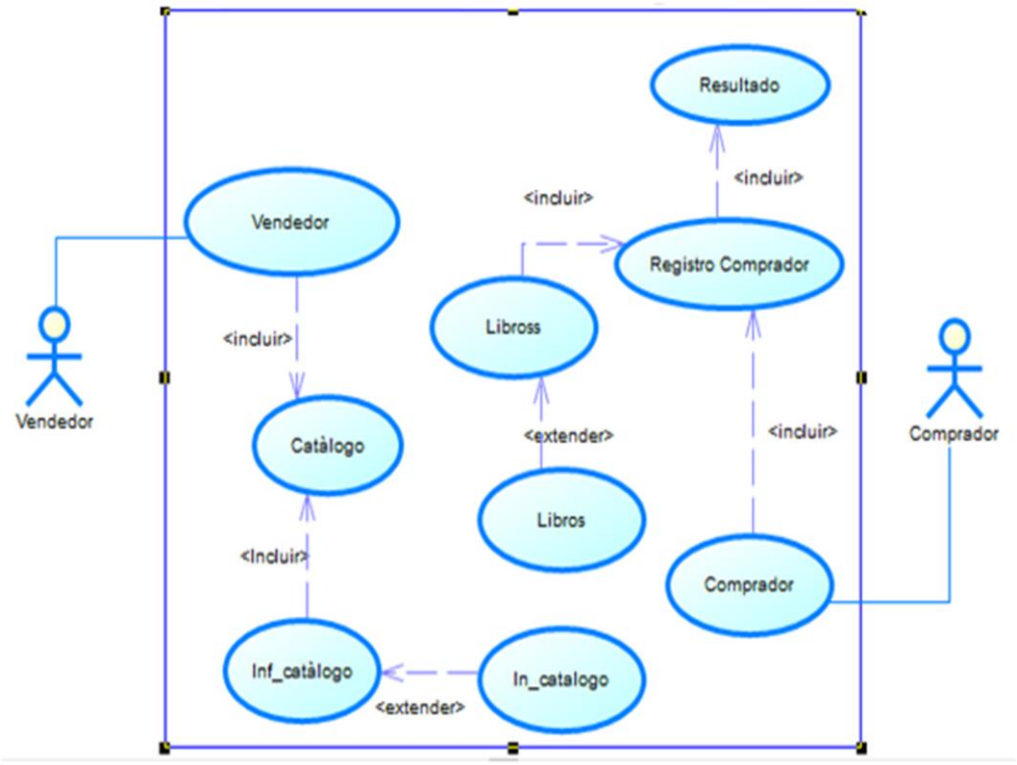


Figura 15. Diagrama Caso de Uso.

La figura muestra los casos de usos que se utilizan para el desarrollo de la aplicación en objetos. Elaborado por Evelyn Obando & Doris Tituaña.

2.4.2 Diagrama de Clases

Riesco menciona que un diagrama de clases es "una descripción de conjuntos de objetos que comparten los mismos atributos, operaciones, métodos y relaciones. Las clases son gráficamente representadas por cajas con compartimentos para: Nombre de la clase, los atributos y las operaciones a realizar" (Riesco, s.f.).

En la Figura 16 se podrá visualizar el diagrama de clases respectivo para esta aplicación en la cual se identificará cada una de las clases existentes, permitiendo formar grupos de atributos para su funcionamiento. Tomando en cuenta tres puntos necesarios como:

- Herencia, la misma que indica que una clase hereda los métodos y atributos por otra clase.

- Agregación, la cual permite identificar enteros, reales, secuencia de caracteres.
- Asociación, la cual permite relacionar clases u objetos que colaboran entre ella.

Diagrama de Clases

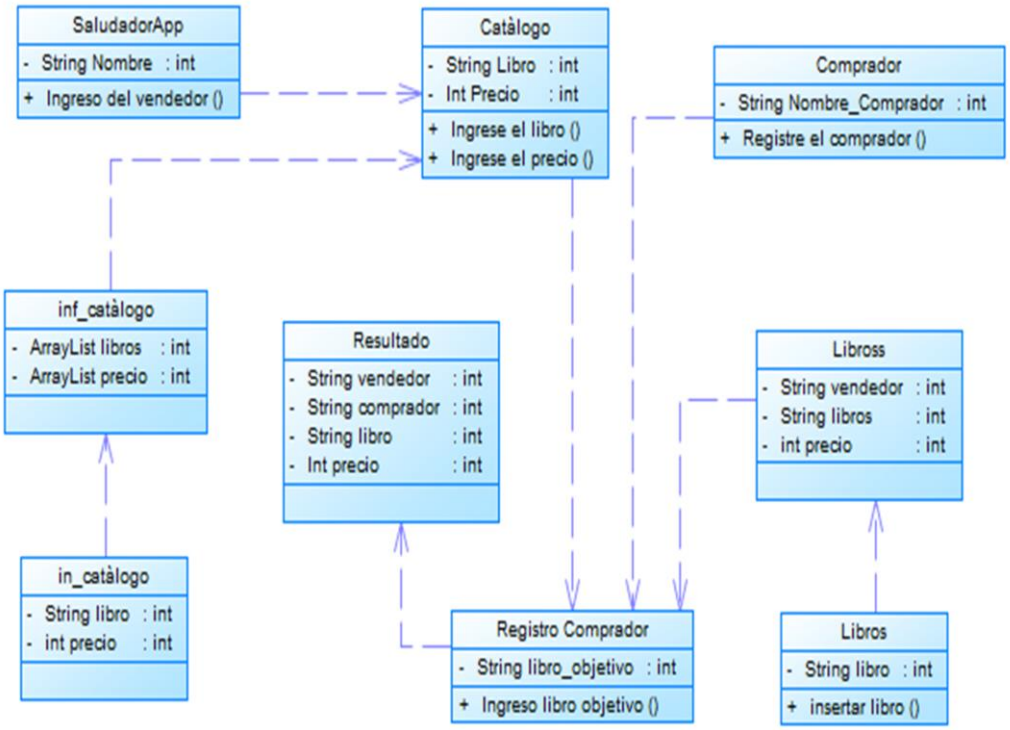


Figura 16. Diagrama de Clases.
La figura muestra el diagrama de clases, en este se detallan las interacciones de las clases utilizadas en la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

2.4.3 Diagrama de Secuencia

Los diagramas de secuencia son definidos por Gutierrez como, "diagramas que muestran la forma en que un grupo de objetos se comunican entre sí a lo largo del tiempo. Consta de objetos, mensajes entre estos objetos y una línea de vida del objeto representada por una línea vertical" (Gutierrez, 2011).

Para el desarrollo del diagrama de secuencia se realizarán algunos pasos, los cuales nos permitirán visualizar la comunicación existente entre los objetos:

I). En el primer paso, insertan los actores, que fueron identificados en el diagrama de casos de uso

II). En el segundo paso, se toma en cuenta los roles de las clases identificadas en el diagrama de clases

III). En el tercer paso, se colocan las flechas que representan los mensajes de comunicación entre los objetos como:

- El actor *vendedor* inicia su comunicación con el objeto *vendedor*.
- El actor *comprador* inicia su comunicación con el objeto *comprador*.
- El objeto *vendedor* inicia su comunicación con el objeto *catálogo*.
- El objeto *comprador* inicia su comunicación con el objeto *registro_compra*.

IV). En el cuarto paso, se coloca las flechas de activación las cuales indicaran los lapsos de tiempo que ejecutará una acción.

- El objeto *catalogo* inicia su activación entre el objeto *inf_catalogo* y el objeto *in_catalogo*.
- El objeto *registro_compra* inicia su activación entre el *objeto libros* y el objeto *libro*.

V). En el quinto paso, una vez realizada la consulta del libro a buscar, presenta un reporte de información del libro solicitado. Los pasos mencionados para realizar el diagrama de secuencias podran ser visualizados en la Figura 17 que se muestra a continuación.

Diagrama de Secuencias

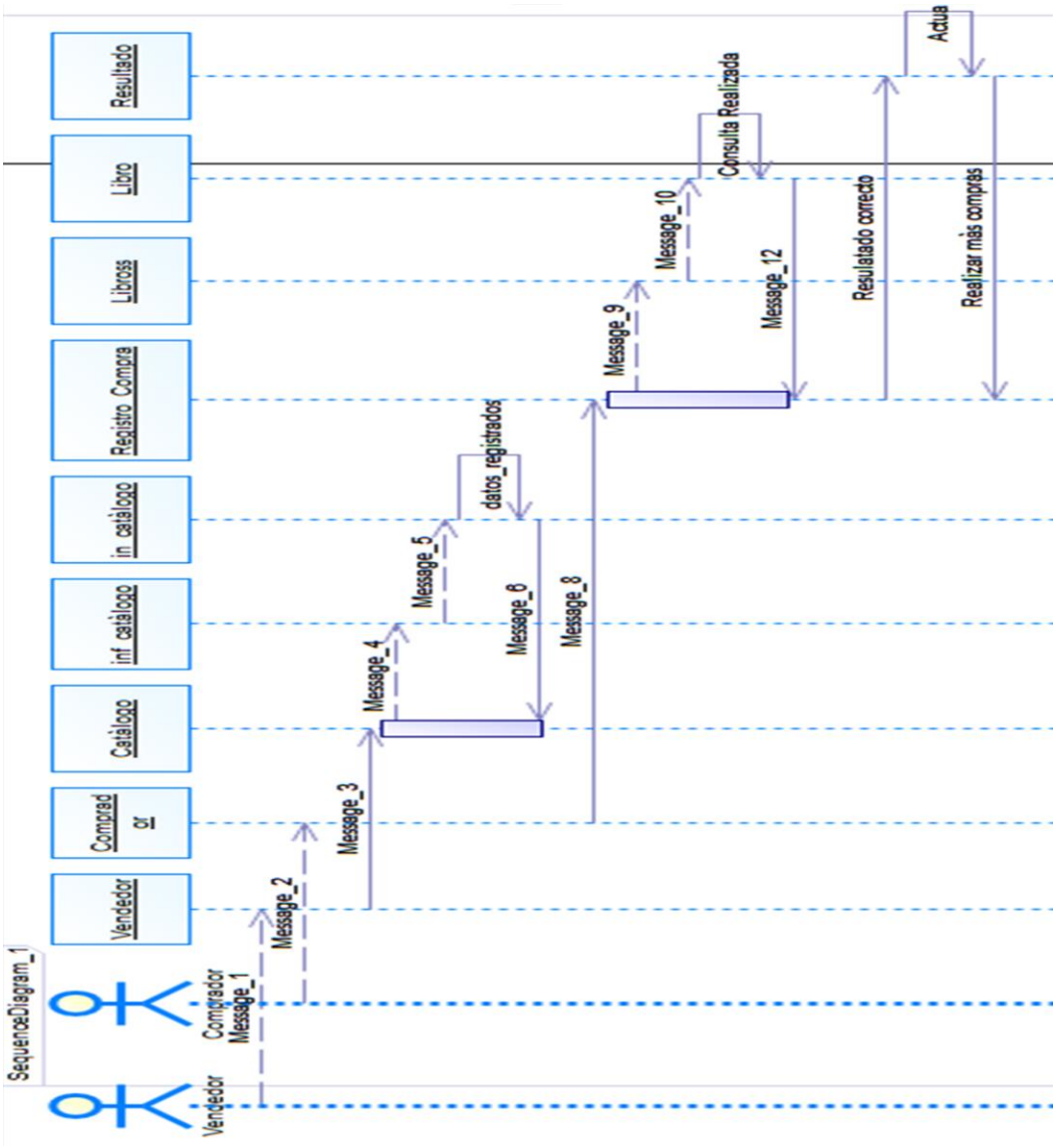


Figura 17. Diagrama de Secuencias.
La figura muestra la comunicación entre las clases utilizadas en la aplicación y entre los actores vendedor y comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

La Figura 17 muestra la comunicación que existe entre los actores vendedor y comprador. En la primera interacción el actor *Vendedor* inicia su comunicación con la clase *Vendedor* a su vez el actor *Comprador* inicia su comunicación con la clase *Comprador*. En la segunda interacción la clase *Vendedor* inicia una comunicación con la clase *Catalogo,* esta clase mantiene una relación interna con las clases *inf_catalogo* y *in_catalogo* las mismas que permitirán el ingreso del nombre del vendedor, el título del libro y el precio. En la tercera interacción la clase *Comprador* inicia su comunicación con la clase *Registro_Compra,* en esta clase se realizará el proceso,en el cual se consultará si el nombre del libro ingresado por el comprador existe o no, si el libro existe se enviará el nombre del vendedor el cual registro el libro, el nombre del comprador , el nombre del libro y el precio, caso contrario el comprador deberá ingresar un libro nuevo.

CAPÍTULO 3

DESARROLLO DE LA APLICACIÓN

3.1 Herramientas de desarrollo

Se detallarán las herramientas que se utilizó para el desarrollo de la aplicación

3.1.1 Java Development Kit

Más conocido como JDK, es un software que provee herramientas de desarrollo para la creación de programas en java. Puede instalarse en una computadora local o en una unidad de red (León Paredes, 2012).

Los programas que incluye el JDK son:

- Appletviewer: es un visor de applet para generar sus vistas previas.
- Javac:es el compilador de JAVA.

- java: es el intérprete de JAVA.
- javadoc: genera la documentación de las clases java de un programa.

3.1.2 JDK en Windows

Para que el JDK funcione en Windows además de instalarlo se deberán configurar las diferentes variables de entorno:

- JAVAPATH: path del directorio donde está instalado el JDK.
- CLASSPATH: son las librerías o clases de usuario.
- PATH: variable donde se agrega la ubicación de JDK.

3.2 Instalación plugins y herramientas de programación

Se requiere tener instalado algunos plugins para que las herramientas sean utilizadas adecuadamente.

3.2.1 Instalación de Netbeans

Se procede a descargar e instalar netbeans según las características de la máquina, un requisito previo es tener instalado jdk si no lo tiene se procede a descargar e instalar. Se debe configurar las variables de entorno.

3.2.2 Configuración de JADE

Para la instalación de la plataforma Jade se debe descargar de la página oficial, copiar la carpeta en la unidad C, compilar usando comandos en el símbolo del sistema y dentro del directorio jade.

3.3 Desarrollo y ejecución de la aplicación

3.3.1 Programación Orientada a Agentes

Esta aplicación consta de 3 clases: AgenteCompradorLibros.java, AgenteVendedorLibros.java, GuiVentaLibros.java.

La aplicación está diseñada para crear agentes vendedores y compradores, pero para la demostración se realizó la creación de un (1) agente vendedor y un (1) agente comprador, se procede a detallar los pasos del manejo de la aplicación:

1: Ejecutar la aplicación para que arranque la Plataforma Jade.

Pantalla Principal

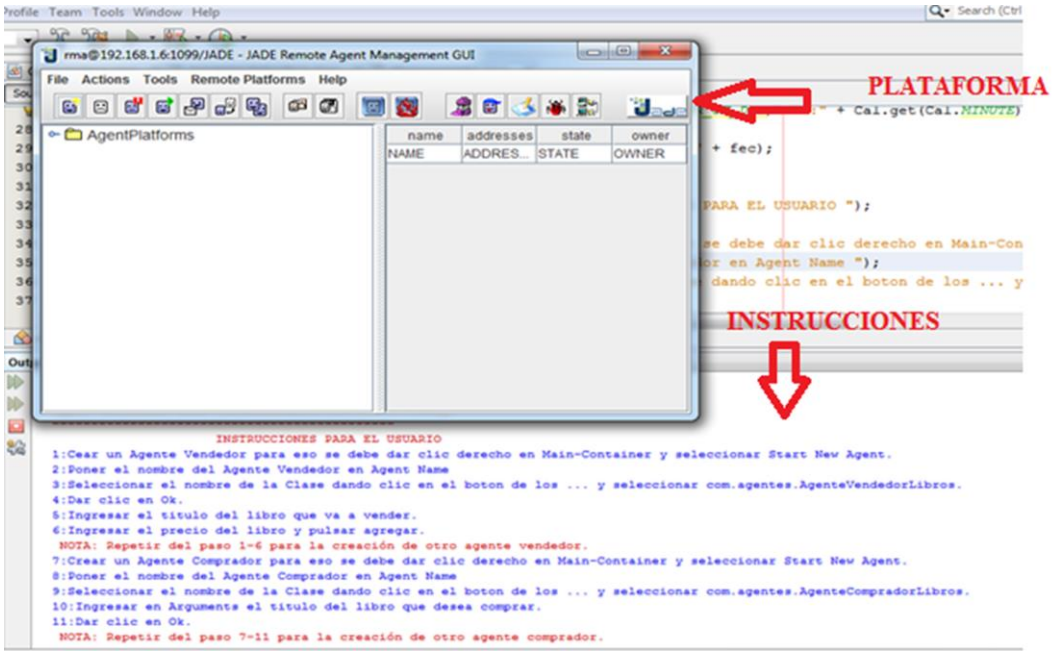


Figura 18. Pantalla principal.
En esta figura se muestra el arranque de la plataforma Jade para iniciar la creación de los agentes, a la vez en consola se muestra las instrucciones para el usuario.
Elaborado por Evelyn Obando & Doris Tituaña.

2: Crear el agente vendedor, para lo cual se debe ingresar el nombre y seleccionar la respectiva clase, que en este caso es AgenteVendedorLibros.java.

Creación Agente Vendedor

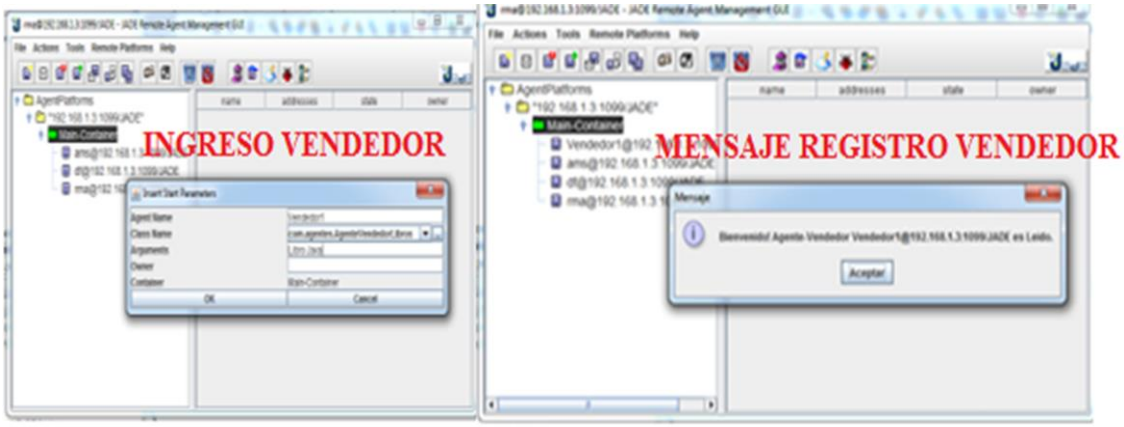


Figura 19. Creación Agente Vendedor.
En esta figura se muestra la creación del agente vendedor desde el *main container* y el mensaje de bienvenida con su respetivo nombre, dirección y puerto: *Vendedor1@192.168.1.3:1099/JADE*.
Elaborado por Evelyn Obando & Doris Tituaña.

3: Una vez que se presiona aceptar el agente *Vendedor1* debe ingresar la información del libro que desea vender en este caso será *Titulo del Libro*: Libro Java (la primera letra debe ser Mayúscula y el resto minúsculas) y *Precio*: 70 (Este valor debe ser un número entero).

Ingreso información Libro

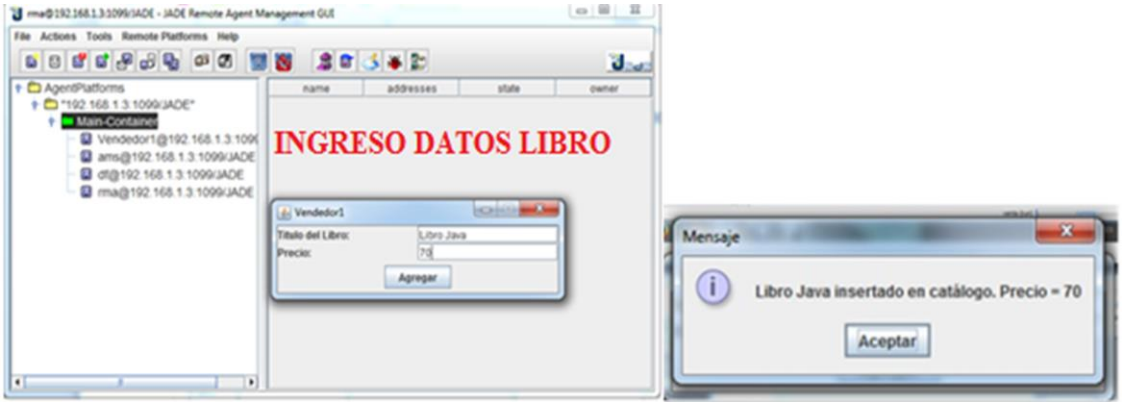


Figura 20. Ingreso información Libro.
En esta figura se muestra el ingreso de la información del libro que es insertado en catálogo.
Elaborado por Evelyn Obando & Doris Tituaña.

El mismo procedimiento anterior se realizará para la creación de otros Agentes Vendedores si se lo requiere.

4: Creación Agente Comprador, desde el *main container*, ingresa el nombre, selecciona la respectiva clase que es AgenteCompradorLibros.java y en *Arguments* el libro que desea comprar.

Creación Agente Comprador

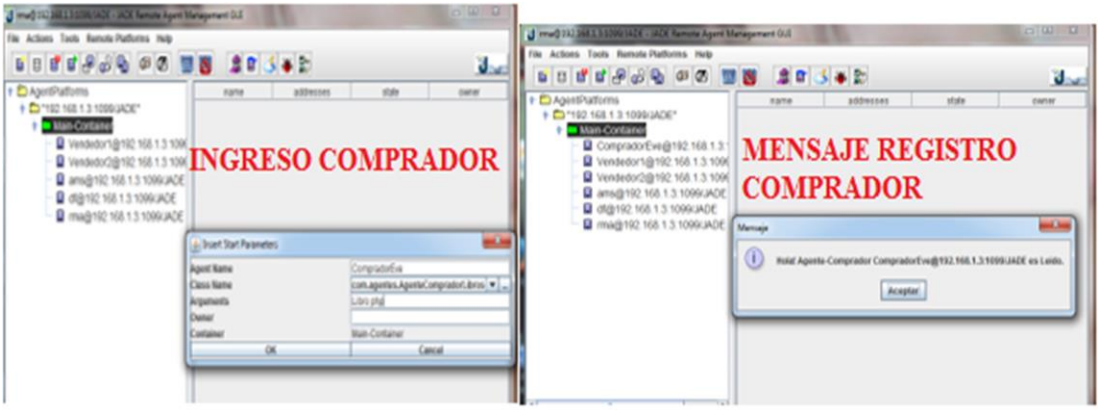


Figura 21. Creación Agente Comprador.

En esta figura se muestra el ingreso del agente comprador en *Arguments* ingresa el libro objetivo y el despliegue del mensaje de registro del agente comprador *CompradorEve@192.168.1.3:1099/JADE*. Elaborado por Evelyn Obando & Doris Tituaña.

5: Proceso de búsqueda del libro objetivo es un proceso interno y el usuario no lo puede visualizar

Proceso de búsqueda del libro a comprar

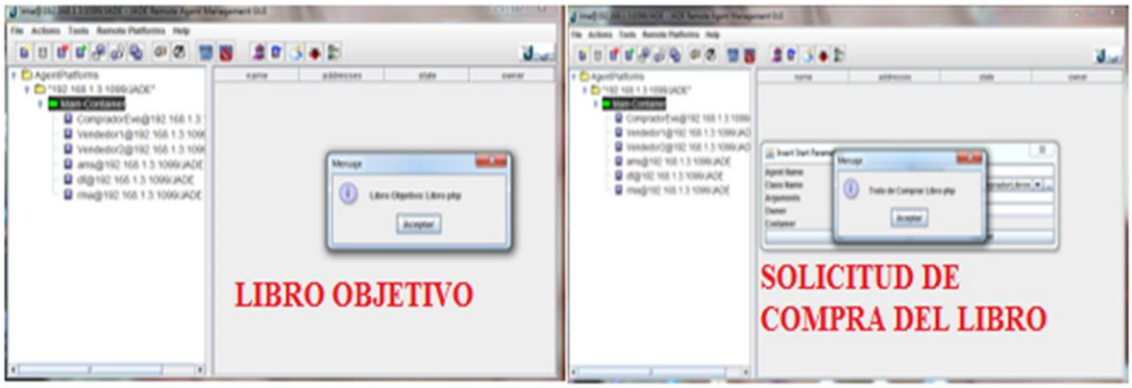


Figura 22. Proceso de búsqueda del libro a comprar.

En esta figura se muestra el nombre del libro que se desea comprar, después de un tiempo de espera de 60 segundos que se lo programó en la aplicación me despliega un mensaje de *Trato comprar* el libro objetivo que ingreso el agente comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

6: Una vez encontrado el libro objetivo se despliega la información de los agentes vendedores registrados.

Información del Agente Vendedor

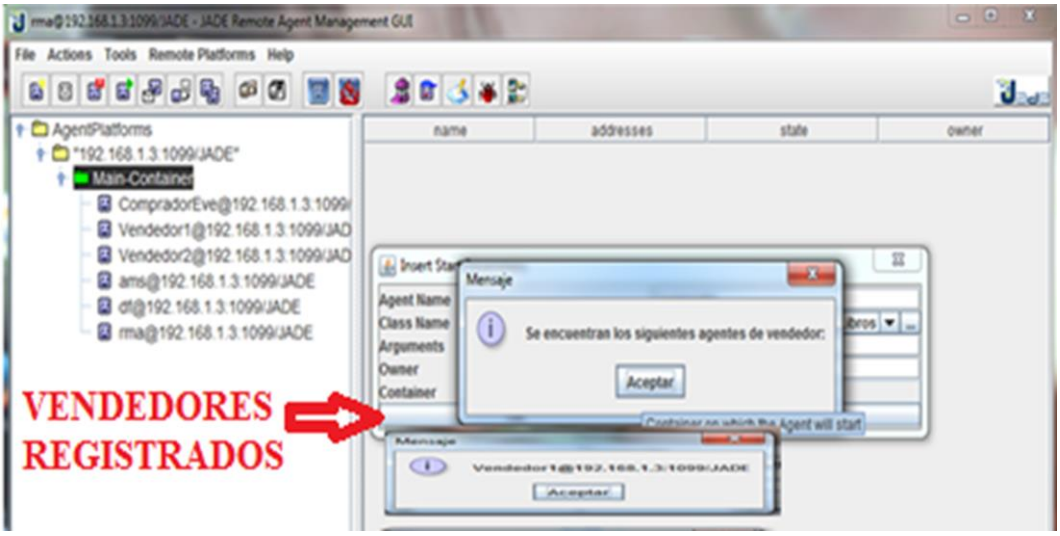


Figura 23. Información del Agente Vendedor.
En esta figura se muestra la información del agente vendedor registrado con su dirección y el puerto.
Elaborado por Evelyn Obando & Doris Tituaña.

7: Al terminar la compra exitosamente se despliega el nombre del agente comprador, el nombre del agente vendedor y el precio.

Información de la Compra

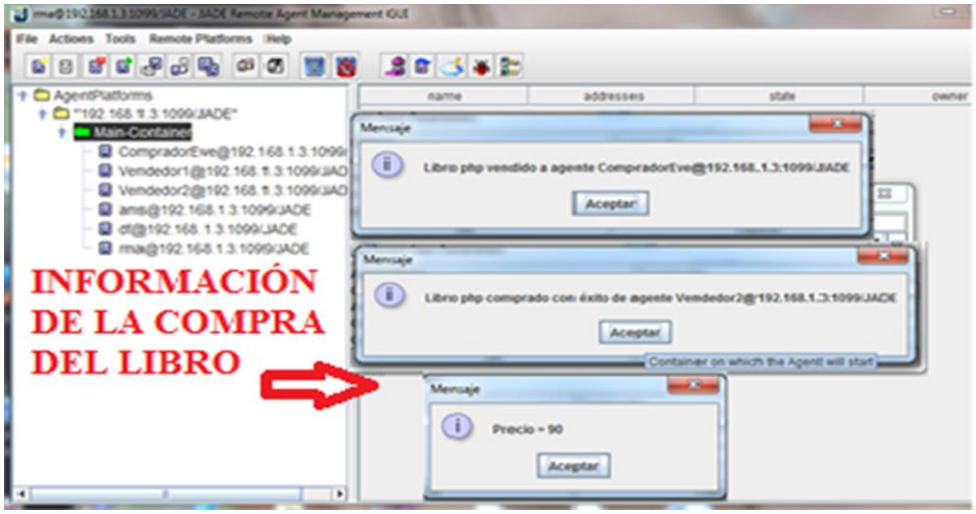


Figura 24. Información de la Compra.
En esta figura se muestra que el libro objetivo fue vendido con éxito, muestra el nombre del comprador, nombre vendedor y precio.
Elaborado por Evelyn Obando & Doris Tituaña.

Una vez realizada la venta del libro el agente comprador finaliza.

3.3.2 Comunicación de Agentes

Se podrá visualizar la comunicación que realizan los agentes en el proceso de compra del libro mediante el protocolo FIPA-ACL, para realizar este proceso se debe iniciar el Sniffer seleccionando los agentes que van a realizar la comunicación en este caso (Agente vendedor y agente comprador).

Creación del Agente Sniffer

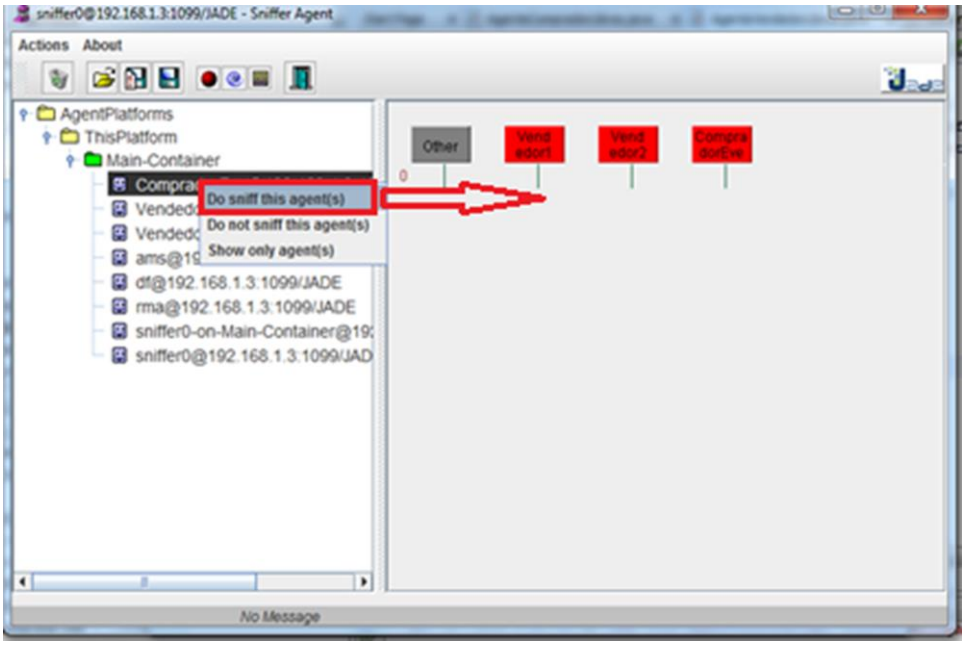


Figura 25. Creación del Agente Sniffer.
En esta figura se muestra el inicio del agente Sniffer para realizar la comunicación entre agentes, debemos seleccionar los agentes que se van a comunicar en este caso agente vendedor y agente comprador dando clic derecho en la opción (Do sniff this agent).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 25 se visualizará la comunicación entre los agentes mediante mensajes ACL con todas las interacciones que realizan hasta llegar a la meta.

Comunicación entre Agentes

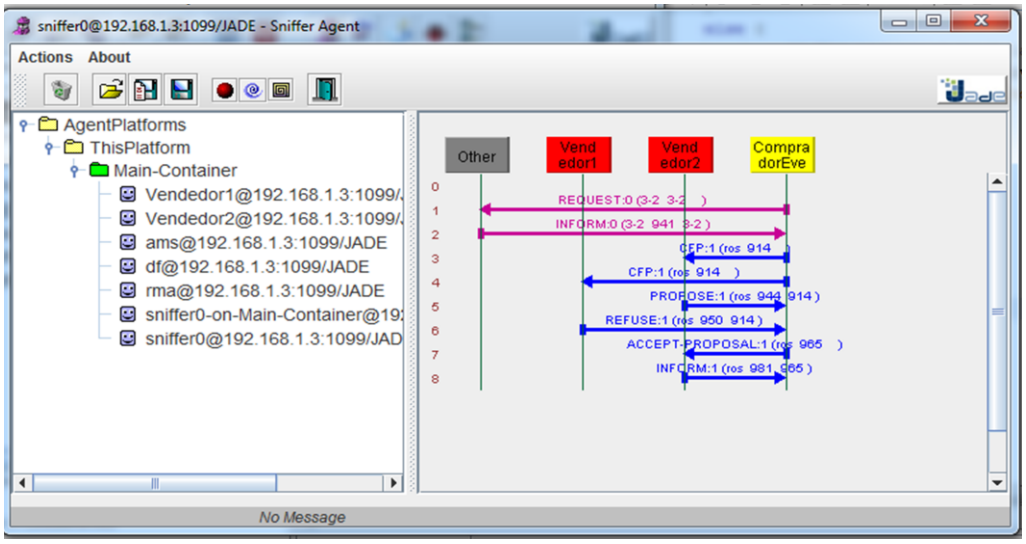


Figura 26. Comunicación entre Agentes.
En esta figura se muestra la comunicación entre los agentes los actos comunicativos son (*Request,inform,cfp,propose,refuse,accept-proposal*).
Elaborado por Evelyn Obando & Doris Tituaña.

Para visualizar los parámetros del mensaje ACL se da clic sobre el correspondiente acto comunicativo (flecha) y se despliega la ventana que se muestra en la figura 27

Estructura del mensaje ACL

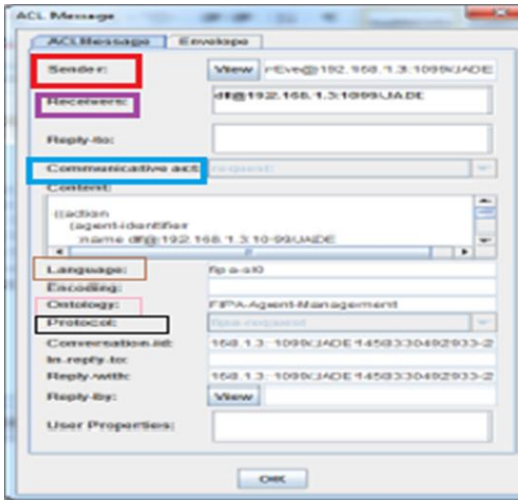


Figura 27. Estructura del mensaje ACL.
En la figura se muestran los parámetros del mensaje ACL para este caso *(Sender*: agente que va a enviar el mensaje, *Receiver*: agente que va a recibir el mensaje, *comunication act*: *request, content, lenguage*: *fipa*, *ontology*: FIPA-*Agent-Management*).
Elaborado por Evelyn Obando & Doris Tituaña.

3.3.3 Programación Orientada a Objetos

La pantalla que se muestra en la Figura 28, indicará la portada inicial de la aplicación, permitiendo al usuario visualizar tres opciones definidas como: *Instrucciones del Usuario*, *Registro Vendedor*, *Registro Comprador*.

Pantalla Principal

Figura 28. Pantalla Principal.
En esta figura se muestran las opciones de la aplicación, permitiendo el registro de vendedores y compradores.
Elaborado por Evelyn Obando & Doris Tituaña.

Pantalla Instrucciones del Usuario.

La pantalla que se muestra en la Figura 29, indicará una serie de instrucciones las cuales facilitarán el uso de la aplicación para el usuario.

Pantalla Instrucciones del Usuario

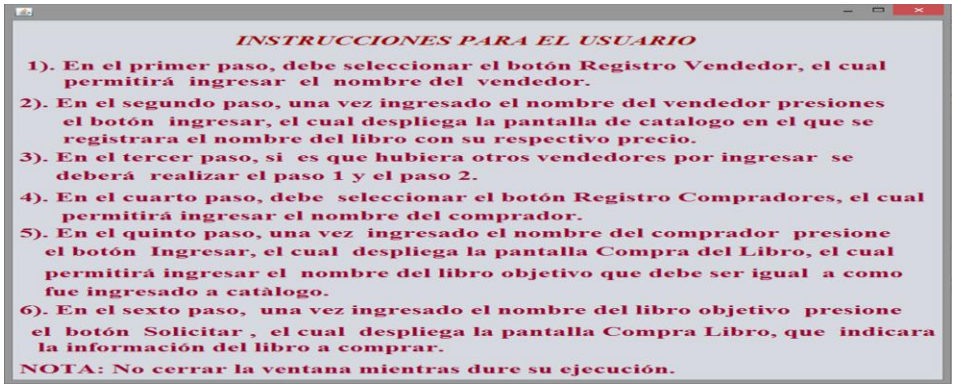


Figura 29. Pantalla Instrucciones del Usuario.
En esta figura se muestran las instrucciones que permiten a los usuarios el uso correcto de la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

Pantalla registro del vendedor.

La pantalla que se muestra en la Figura 30, permite el ingreso del vendedor. En caso que existieran más vendedores a ser registrados, se deberá regresar a la pantalla principal y seleccionar nuevamente la opción *Registro Vendedor*. A su vez esta pantalla indica al usuario una guía del ingreso de los datos.

Pantalla registro del vendedor

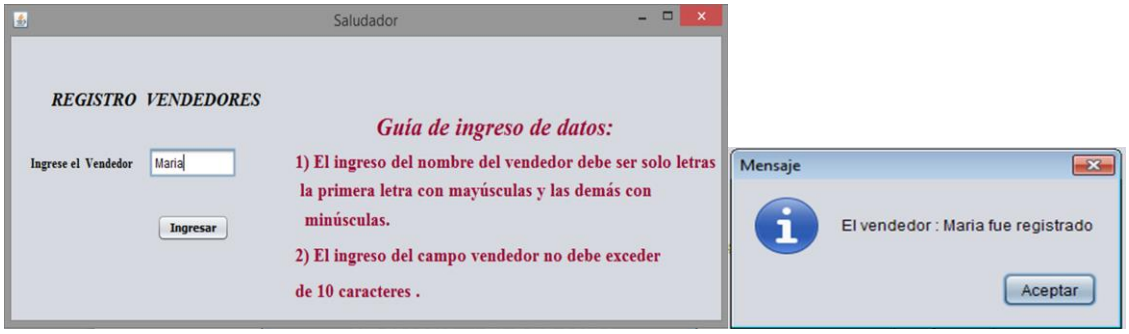


Figura 30. Pantalla registro del vendedor.
En esta figura se muestra la pantalla que permitirá ingresar el nombre del vendedor, una vez ingresado nos despliega el mensaje de registro.
Elaborado por Evelyn Obando & Doris Tituaña.

Pantalla registro catálogo.

La pantalla que se muestra en la Figura 31, permitirá al vendedor registrado ingresar el título del libro y el precio, cabe mencionar que dicho vendedor podrá registrar los libros que desee, se visualizará una tabla que indicará los ingresos realizados. Además, esta pantalla se encuentra la guía del ingreso de los datos.

Pantalla registro catálogo

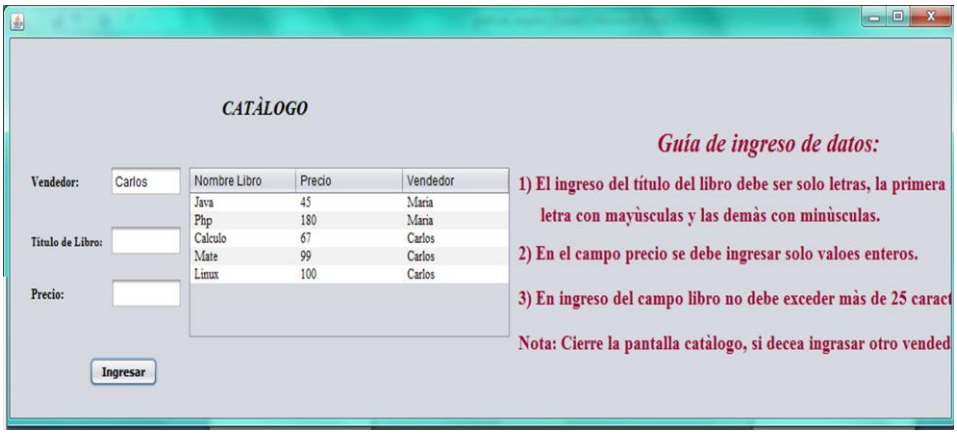


Figura 31. Pantalla registro catálogo.
En esta figura se muestra la pantalla que permitirá el ingreso del título del libro y el precio los cuales serán posteriormente consultados por el comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

Pantalla registro comprador.

La pantalla que se muestra en la Figura 32, permitirá ingresar el nombre del comprador. En el caso que existan más compradores para ser registrados se deberá regresar a la pantalla principal y seleccionar nuevamente la opción *Registro Comprador*. A demás esta pantalla permitirá observar una guía para el ingreso de los datos.

Pantalla registro comprador

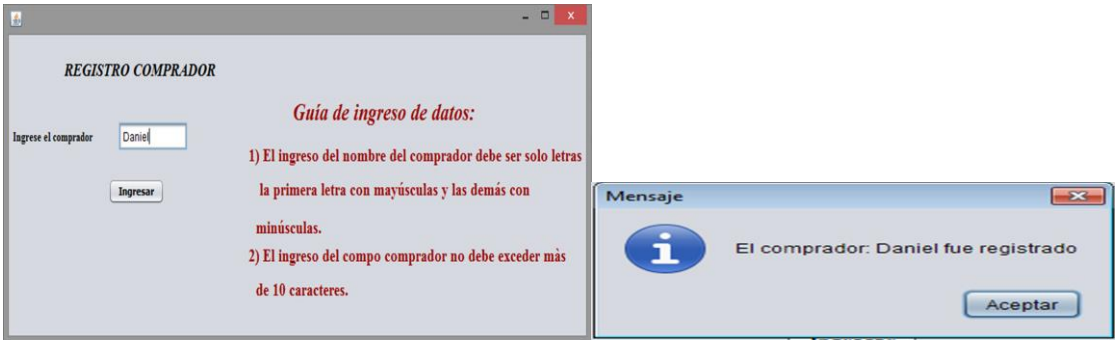


Figura 32. Pantalla registro comprador.
En esta figura se muestra la pantalla que permite el ingreso del nombre del comprador a con la respectiva guía de ingreso de datos, una vez ingresado el comprador nos despliega un mensaje de registro.
Elaborado por Evelyn Obando & Doris Tituaña.

Pantalla Compra del Libro.

La pantalla que se muestra en la Figura 33, le permitirá al comprador ingresar el nombre del libro objetivo es decir el libro que desea comprar. Cabe mencionar que el ingreso del libro objetivo debe ser igual a como fue ingresado en la pantalla *Registro catálogo*, para ello el comprador visualizará en consola un listado de los libros que están desplegados. Además, esta pantalla permitirá observar una guía del ingreso de los datos.

Pantalla Compra del Libro

Figura 33. Pantalla Compra del Libro.
En la siguiente figura muestra la pantalla que permitirá ingresar el libro objetivo, la respectiva guía de ingreso de datos.
Elaborado por Evelyn Obando & Doris Tituaña.

Resultado Exitoso de las Búsqueda.

La pantalla que se muestran en la Figura 34, permite visualizar el resultado exitoso de la búsqueda realizada por el comprador donde indicará: el título del libro, precio, el nombre del vendedor que ingreso el libro y el nombre del comprador como un resultado final.

Resultado Exitoso de las Búsqueda

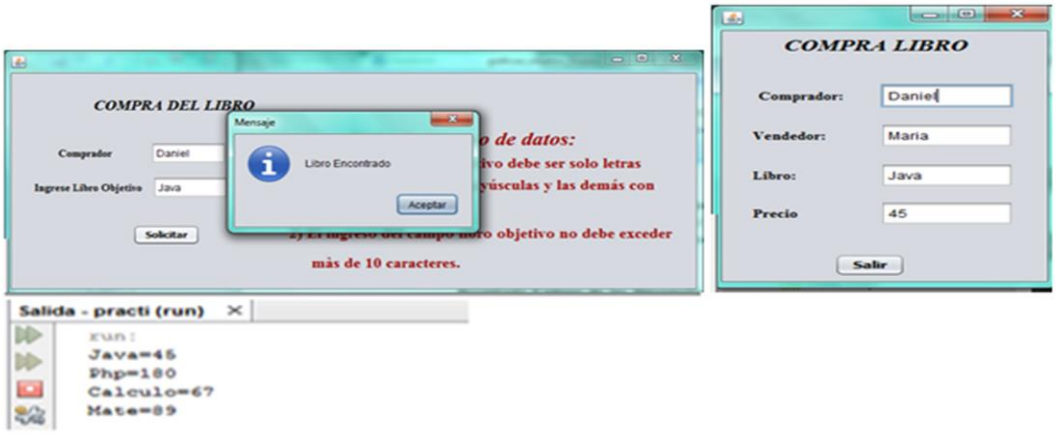


Figura 34. Resultado Exitoso de la búsqueda.
En esta figura muestra la pantalla que indica el resultado exitoso de la búsqueda realizada por el comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

Resultado erróneo de la búsqueda.

La pantalla que se muestra en la Figura 35, permitirá visualizar si el ingreso del libro objetivo no se encuentra en el listado de los libros registrados y desplegados en consola, se obtendrá un mensaje el cual especifica que el libro ingresado no existe en el *Catálogo*.

Resultado erróneo de la bùsqueda

Figura 35. Resultado erróneo de la búsqueda.
En esta figura se muestra la pantalla que permite visualizar si el ingreso del libro objetivo no se encuentra registrado despliega un mensaje de *ERROR*.
Elaborado por Evelyn Obando & Doris Tituaña.

CAPÍTULO 4

ANÁLISIS DE RESULTADOS

En este capítulo se analizarán los resultados obtenidos de las aplicaciones desarrolladas con Programación Orientada a Agentes (POA) vs Programación Orientada a Objetos (POO). Utilizamos el *Profiler* de Netbeans para obtener datos y poder realizar gráficas comparativas.

4.1 Gráficas Monitor Agentes

En la Figura 36 se visualizarán los valores obtenidos al momento de ejecutar la aplicación es decir al arranque de la plataforma, se tienen cinco parámetros los cuales se utilizarán para la interpretación de los resultados, estos son: *Heap Size, Used heap, Surviving Generations, Threads, Loaded Classes.*

Gráficas Monitor

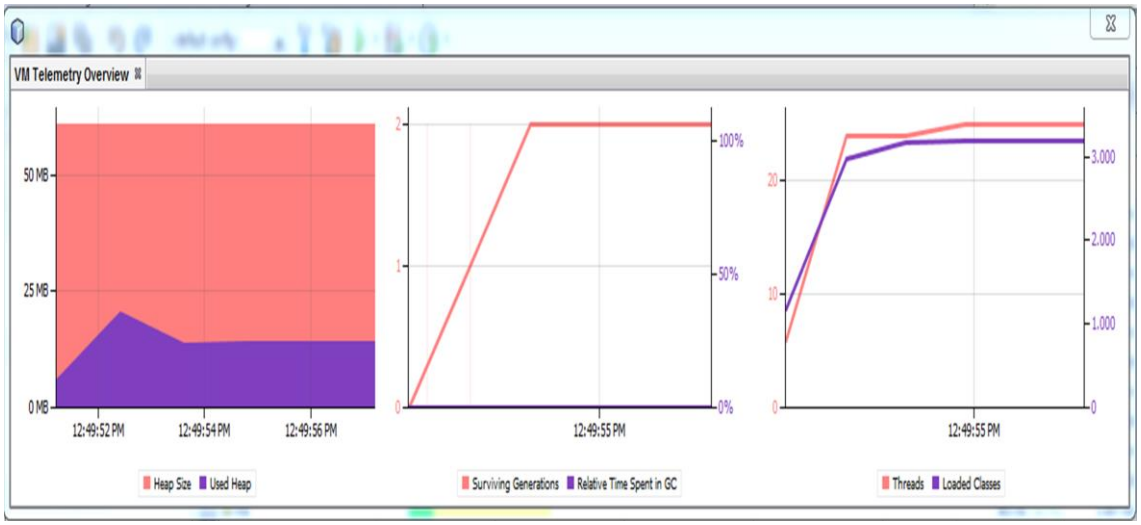


Figura 36. Gráficas Monitor Agentes.
En esta figura se muestran las gráficas de monitor que son evaluadas en dos ejes el eje x (hora) y el eje y (Mb) al arrancar la plataforma Jade.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.1 Variación *Heap Size* y *Used Heap*

Al iniciar el programa se consigue un poco de memoria de sistema operativo es decir que cada vez que se crea un objeto o un agente se asigna memoria desde *Heap* y cuando el objeto muere se libera espacio, la variación de valores para esta aplicación de agentes se muestra en la Figura 37.

Variación valores *Heap Size y Used Size*

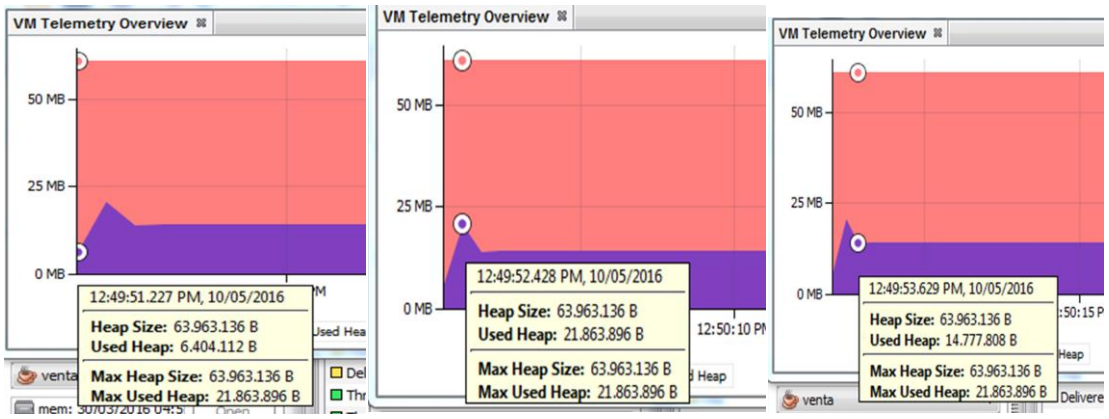


Figura 37. Variación valores *Heap Size* y *Used Size*.
En esta figura se muestra la variación en los valores de *Used Size* en el punto inicial(6,4MB), medio(21,8Mb) y final(14,7Mb). El valor de *Heap Size* se inicia y se mantendrá en 63.963Mb.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.2 *Surviving Generations*

La métrica que puede detectar posibles fugas de memoria es la generación de supervivencia (*Surviving Generations*). Una generación es un conjunto de instancias creadas dentro del mismo intervalo.

4.1.3 Variación *Surviving Generations*

Variación *Surviving Generations*

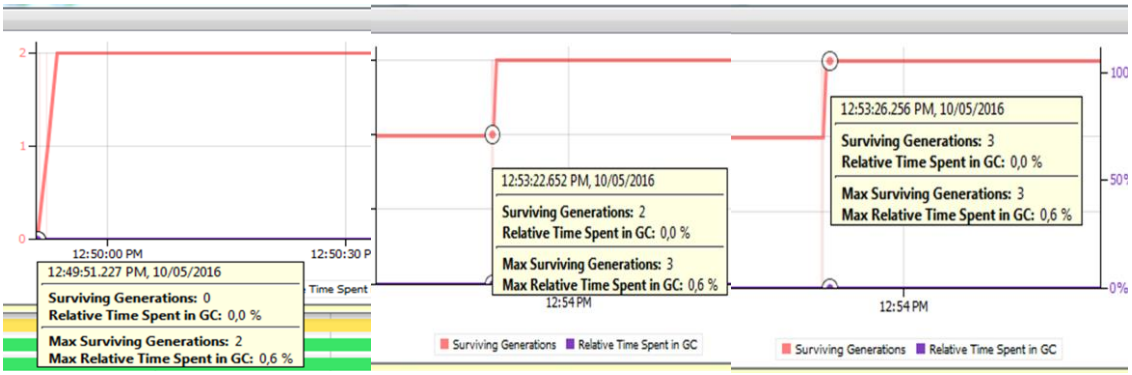


Figura 38. Variación *Surviving Generations.*
En esta figura se muestra la variación en los valores de *Surviving Generations* en el punto inicial es decir al arranque de la plataforma tiene un valor de (0) en el proceso de creación de agentes tanto vendedor como comprador cambia a (2) y al finalizar los agentes varía a (3).
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.4 *Threads, Loaded Classes*

Hilos, procesos, clases cargadas y ejecutadas que existen en la aplicación.

4.1.5 Variación *Threads, Loaded Classes*

Threads, Loaded Classes

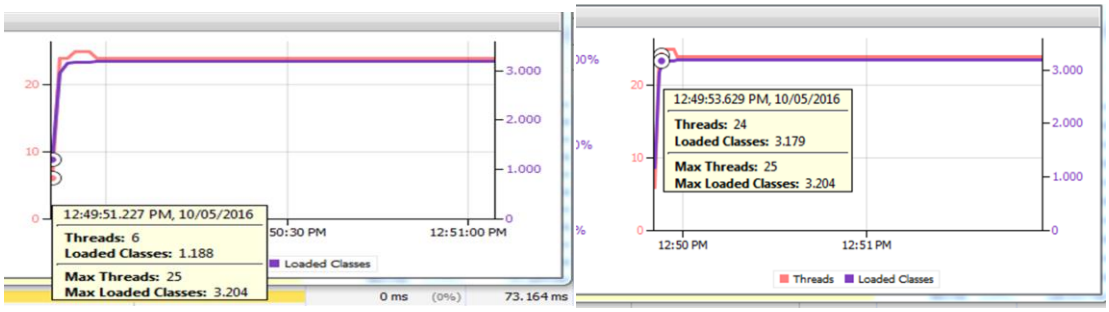


Figura 39. *Threads, Loaded Classes.*

En esta figura se muestra la variación en los valores de *Threads y Loaded Classes* en el punto inicial es decir al arranque de la plataforma tiene un valor de (6) para hilos y (1.188) para clases cargadas, llega a un punto final con un valor de (24) para hilos y (3.179) para clases cargadas.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.6 Variación *Heap Size* y *Used Heap* en la creación del agente vendedor.

Variación valores *Heap Size* y *Used Size*

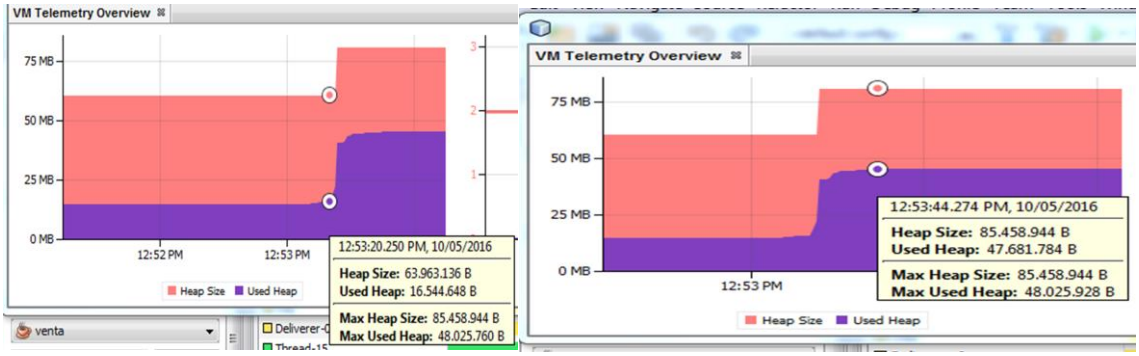


Figura 40. Variación valores *Heap Size* y *Used* Size.

En esta figura se muestra la variación en los valores de *Used Size* en el punto inicial(16,5B) y final(47,6Mb). El valor de *Heap Size* inició en 63.963Mb y luego permaneció constante.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.*7 Variación Threads, Loaded Classes*

Threads, Loaded Classes

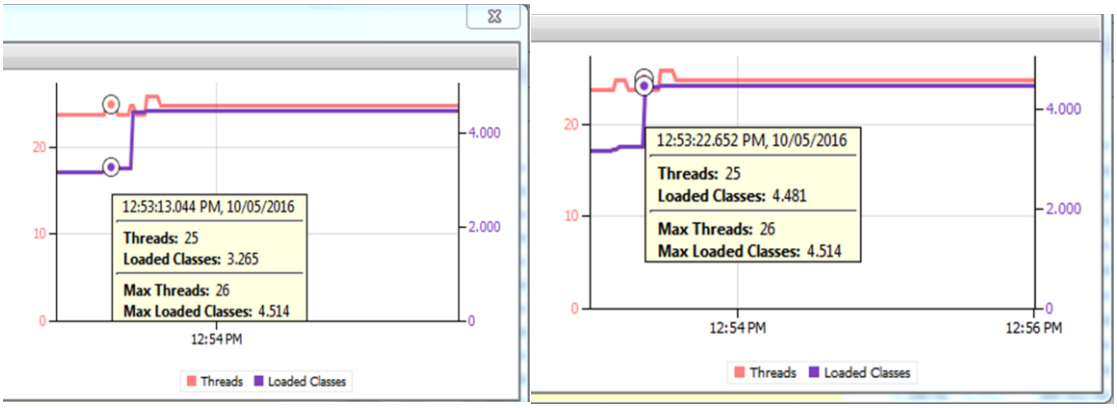


Figura 41. *Threads, Loaded Classes.*

En esta figura se muestra la variación de los valores de *Threads y Loaded Classes*, al crear el agente vendedor tiene un valor de (25) para hilos y (3.265) para clases cargadas, llega a un punto final con un valor de (25) para hilos y (4.481) para clases cargadas.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.8 Variación *Heap Size* y *Used Heap* en la creación del agente comprador.

Variación *Heap Size* y *Used Heap*

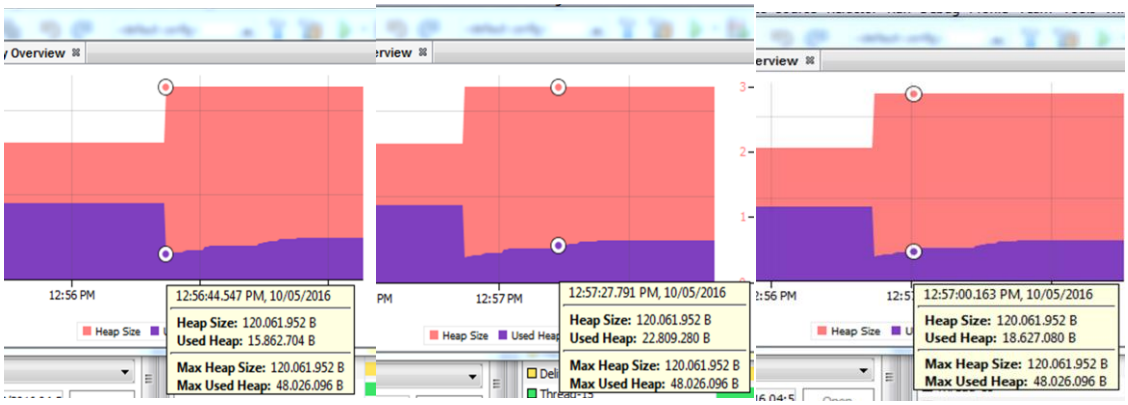


Figura 42. Variación *Heap Size* y *Used Heap*.
En esta figura se muestra la variación en los valores de *Used Size* en el punto inicial(15,8MB) medio (18,6MB) y final(22,8Mb). El valor de *Heap Size* se incrementó a 120Mb.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.9 *Variación Threads, Loaded Clases*

Threads, Loaded Classes

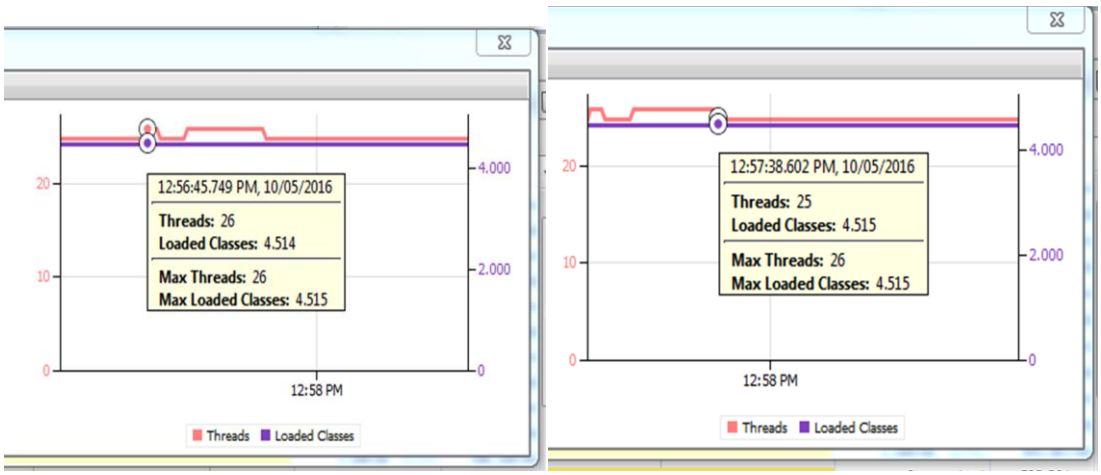


Figura 43. *Threads, Loaded Classes.*

En esta figura se muestra la variación en los valores de *Threads y Loaded* Classes al crear el agente comprador tiene un valor de (26) para hilos y (4.514) para clases cargadas y cuando muere el agente comprador, llega a un punto final con un valor de (25) para hilos y (4.515) para clases cargadas.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.10 Variación *Heap Size y Used Heap* al final de la aplicación.

Variación *Heap Size y Used Heap*

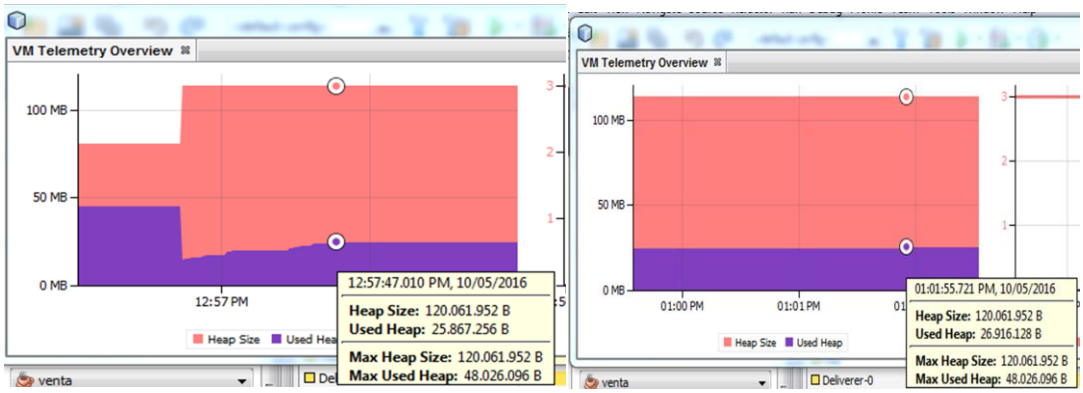


Figura 44. Variación *Heap Size* y *Used Heap*.

En esta figura se muestra la variación en los valores de *Used Size* en el punto medio (25,8MB) y final(26,9Mb). El valor de *Heap Size* se mantiene en 120Mb.
Elaborado por Evelyn Obando & Doris Tituaña.

4.1.11 Variación *Threads, Loaded Classes*

Threads, Loaded Classes

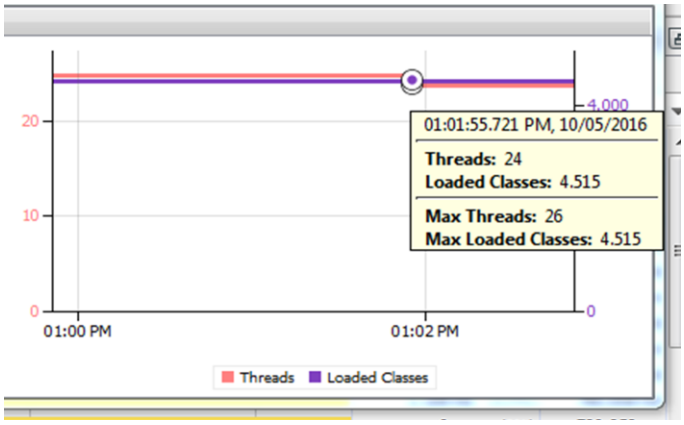


Figura 45. *Threads, Loaded Classes.*

En esta figura se muestra la variación en los valores de *Threads y Loaded* Classes al finalizar los agentes comprador y vendedor con un valor de (24) para hilos y (4.515) para clases cargadas.
Elaborado por Evelyn Obando & Doris Tituaña.

4.2 Gráficas Monitor Objetos

En la Figura 46 se visualizarán los valores obtenidos al momento de ejecutar la aplicación, se tienen cinco parámetros los cuales se utilizarán para la interpretación de los resultados estos son: *Heap Size,Used heap ,Surviving Generations, Threads, Loaded Classes*.

Gráficas Monitor Objetos

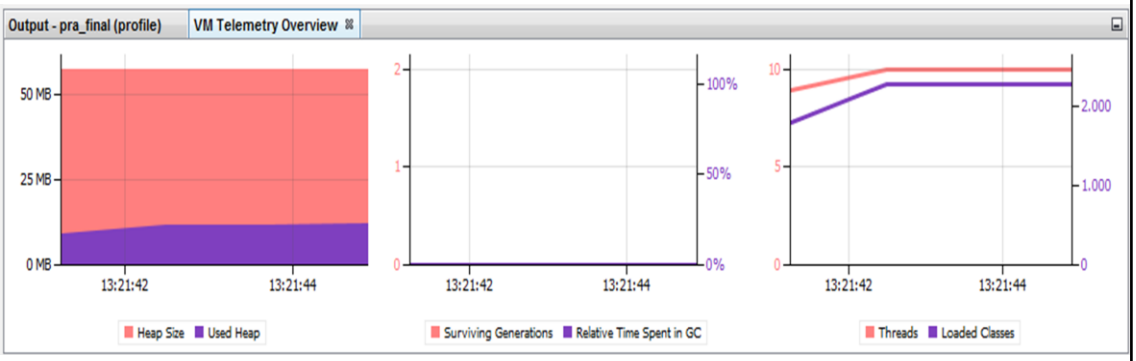


Figura 46. Graficas Monitor Objetos.
En esta figura se muestra las gráficas de monitor de objetos que son evaluadas en dos ejes el eje x(hora) y el eje y(Mb) al arrancar la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

4.2.1 Variación *Heap Size* y *Used Heap*

Al iniciar el programa se consigue un poco de memoria de sistema operativo es decir que cada vez que se crea un objeto o un agente se asigna memoria desde Heap y cuando el objeto muere, se libera espacio de memoria y la variación de valores para la aplicación de agentes se muestra en la Figura 47.

Heap Size (es un valor constante de (60.2 Mb) con el cual inicia el arranque de la aplicación, en caso que el valor del Used Heap este por igualar este valor el sistema propio de Netbeans incrementa su valor) y Used Heap (es el valor que permitirá registrar los cambios en las variaciones).

Variación de valores *Heap Size* y *Used Heap*

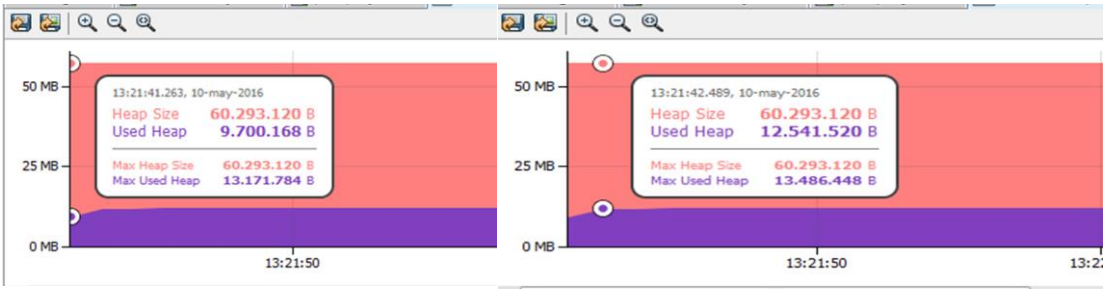


Figura 47. Variación de valores *Heap Size* y *Used Heap*.
En esta figura se muestra la variación en los valores de Used Size en el punto inicial (9,7B) y final (12,5B). El valor de Heap Size se inicia y se mantendrá en 60.293B.
Elaborado por Evelyn Obando & Doris Tituaña.

4.2.2 Variación *Surviving Generations*

Variación *Surviving Generations*

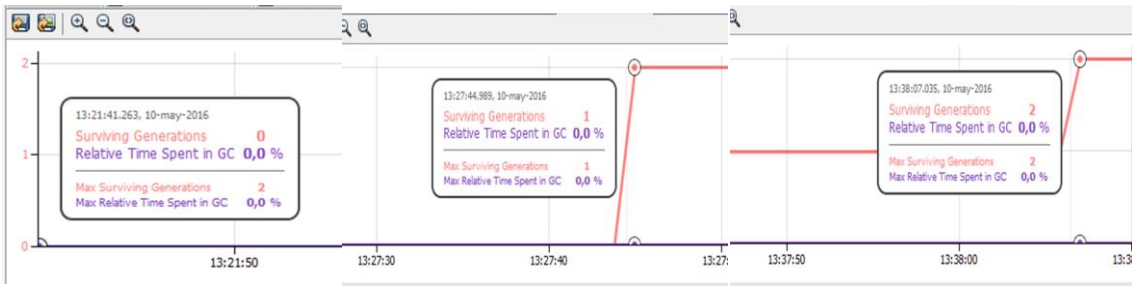


Figura 48. Variación *Surviving Generations.*

En esta figura se muestra la variación de los valores de *Surviving Generations* en el punto inicial es decir al ejecutar la aplicación tiene un valor de (0) en el proceso de registro de vendedor cambia a (1) y al finalizar el proceso de compra del libro cambia a (2).
Elaborado por Evelyn Obando & Doris Tituaña.

4.2.3 *Variación Threads, Loaded Classes*

Threads, Loaded Classes

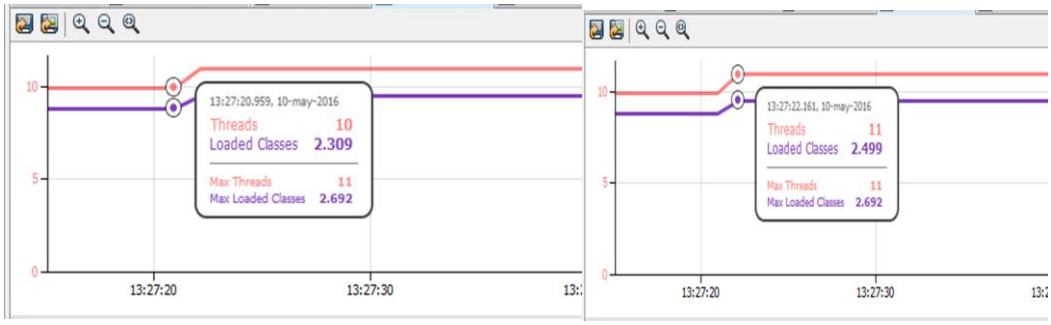


Figura 49. *Threads, Loaded Classes.*

En esta figura se muestra la variación en los valores de *Threads y Loaded* Classes al ejecutar la aplicación inicia con un valor de (10) para hilos y (2.309) para clases cargadas, y se mantiene con un valor de (11) para hilos y (2.499) para clases cargadas hasta que se ejecute otro proceso.
Elaborado por Evelyn Obando & Doris Tituaña.

4.2.4 Variación *Heap Size* y *Used Heap* en la creación del vendedor

En la Figura 50 se visualizarán los valores de las gráficas de *Used Heap*. La gráfica que se encuentra al lado derecho de la figura presenta el punto más alto con un *Used Heap* de (15.6 B) al momento de ingresar un vendedor, mientras que la gràfica que se encuentra en el lado izquierdo de la figura presenta un descenso en el *Used Heap* de (6.4 B) una vez registrado el vendedor permitiendo mantener distintas variaciones en la ejecuciòn de la aplicación. Un punto de vista a mencionar, es que el *Used Heap* va incrementando su valor una vez ya realizado todos los ingresos deseados esto permite ver que mientras la aplicación està en ejecuciòn el *Used Heap* llena casi todo el espacio permitido por Netbeans.

Variación *Heap Size* y *Used Heap* en la creación del vendedor

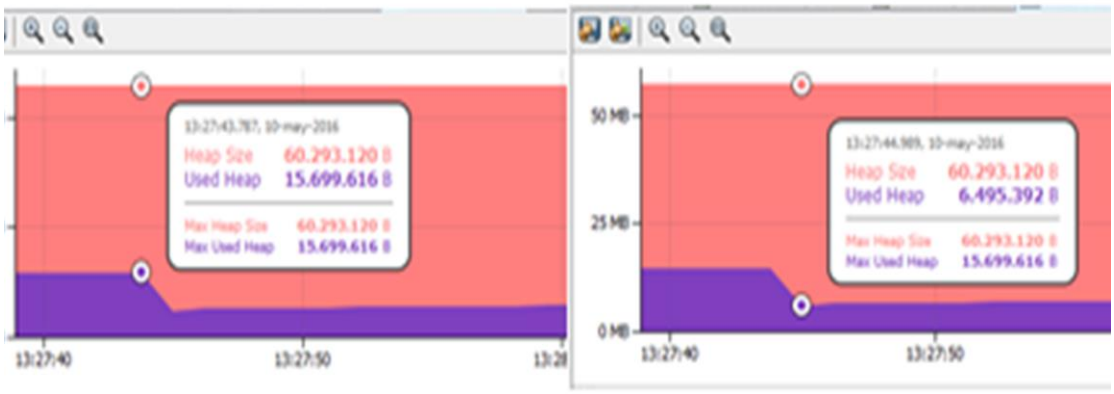


Figura 50. Variación *Heap Size* y *Used Heap* en la creación del vendedor.

La figura muestra el valor de la variación en el *Used Heap* cuando se registra un vendedor y la información del libro.
Elaborado por Evelyn Obando & Doris Tituaña.

5.2.5 Variación *Heap Size* y *Used Heap* en la creación del comprador

En la figura 51 se visualizará el valor obtenido en la gráfica *Used Heap* que es de (13.3B) al momento de ingresar el comprador.

Variación *Heap Size* y *Used Heap* en la creación del comprador

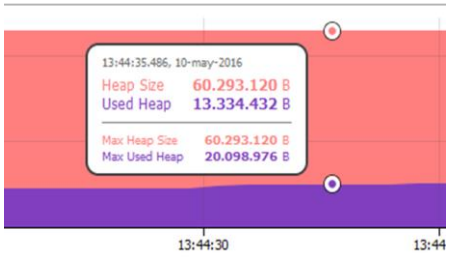


Figura 51. Variación *Heap Size* y *Used Heap* en la creación del comprador.
La figura muestra el valor de elevación registrado en el *Used Heap* cuando se registra un comprador.
Elaborado por Evelyn Obando & Doris Tituaña.

4.3 Gráficas Resultantes

4.3.1 Gráfica de *Used Size*

Una vez obtenido los valores de *Used Size* de Agentes y Objetos se procede a realizar la gráfica general utilizando Excel como se ve en la Figura 52.

Gráfica Final *Heap Size* Agentes vs Objetos

HORA	OBJETOS (B)	AGENTES(B)
12:50:00	9,7	6,4
12:50:10	12,5	21,8
12:50:20	12,5	14,7
12:50:30	15,6	16,5
12:50:40	6,4	47,6
12:53:20	14,7	15,8
12:55:00	20,1	18,6
12:58:00	6,1	22,8
13:00:00	6,9	24,2
13:00:10	13,3	25,8
13:00:20	20,5	26,9

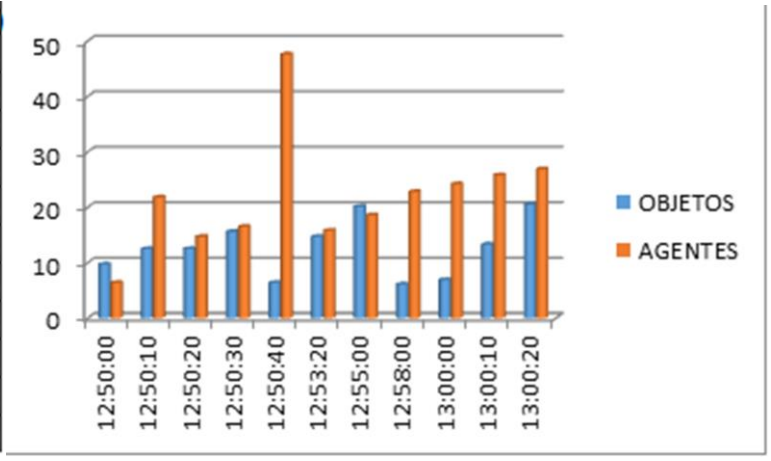


Figura 52. Gráfica Final *Heap Size* agentes vs objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo y en un rango de (0-100Mb).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 52 se visualiza de color azul a los valores de objetos y de tomate a los valores de agentes. En el primer intervalo objetos presenta (9,7B) y agentes (6,4B) por lo que se determina que objetos consume más recursos, debido a que las aplicaciones en objetos requieren más memoria del sistema operativo, a partir del segundo intervalo en adelante agentes consume más recursos que objetos, esto se debe a que al ejecutar la aplicación arranca simultáneamente la plataforma Jade dentro de la cual se crearán los agentes.

Nota: Se observa en la Figura 52, los valores 14.6 (B) y 20.1 (B) que son los valores registrados al momento de ingresar los vendedores en la aplicación de objetos, mientras

que el valor 20.5 (B) representa el valor obtenido al finalizar todos los ingresos, cabe mencionar que este último valor puede seguir incrementando mientras no finalice toda la aplicación.

4.3.2 Gráfica Resultante de *Surviving Generations*

Una vez obtenido los valores de *Surviving Generations* de agentes y objetos se procede a realizar la gráfica general como se ve en la Figura 53.

Gráfica Final Generación de *Surviving Generations* Agentes vs Objetos

HORA	OBJETOS(B)	AGENTES(B)
12:50:00	0	0
12:50:10	1	2
12:50:20	2	3
12:50:30	2	3
12:50:40	2	3
12:53:20	2	3
12:55:00	2	3
12:58:00	2	3
13:00:00	2	3

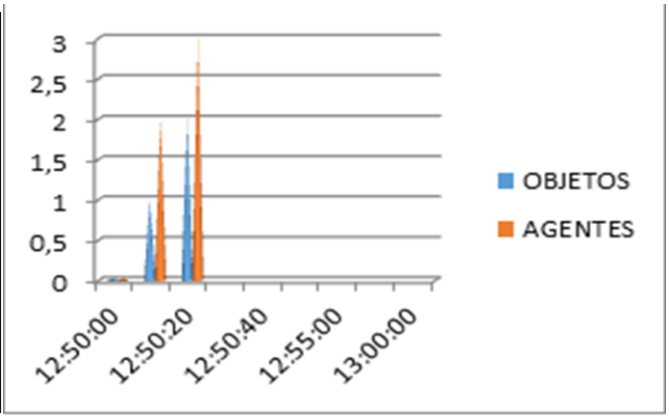


Figura 53. Gráfica final de *Surviving Generations* agentes vs objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en eje x y en el eje y en un intervalo numérico de (0-3).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 53 se visualiza de color azul a los valores de objetos y de tomate a los valores de agentes.

El valor de supervivencia en el primer intervalo, presenta para objetos un valor numérico de (0) y para agentes (0) debido a que se inicia la ejecución de las aplicaciones, a partir del segundo intervalo es decir en la creación de vendedores y compradores objetos presenta un valor de (1) y agentes de (2) es decir que predomina agentes sobre objetos debido a que al crear los agentes se utiliza la plataforma jade la cual ya se encuentra consumiendo recursos y memoria mientras que objetos no necesita

ninguna plataforma al ejecutar la aplicación, en el último intervalo objetos tiene un valor de (2) y agentes de (3) los cuales se mantienen hasta finalizar las aplicaciones.

4.3.3 Gráfica Resultante Hilos

Gráfica Final hilos Agentes vs Objetos

HORA	OBJETOS	AGENTES
12:50:00	10	6
12:50:10	11	24
12:50:20	11	25
12:50:30	11	26
12:50:40	11	25
12:53:20	11	24
12:55:00	11	24
12:58:00	11	24
13:00:00	11	24

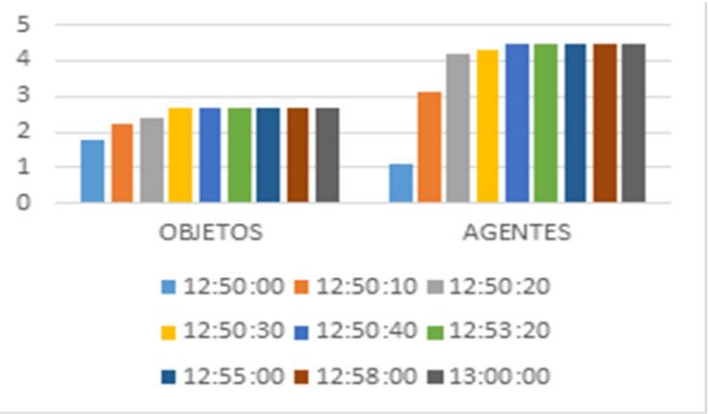


Figura 54. Gráfica final hilos agentes vs objetos.

En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en el eje x y en un intervalo numérico en el eje y.
Elaborado por las autoras de la investigación

En la Figura 54 se visualizan barras de diferentes colores tanto para objetos como para agentes, en el primer intervalo en objetos presenta un valor de hilos que es numérico de (10) y agentes (6) , objetos predomina ya que presenta más clases cargadas, a partir del segundo intervalo en objetos presenta un valor de (11) los cuales se mantiene hasta el final de la aplicación mientras que agentes presenta un valor de (24) al arrancar la plataforma, un valor de (25) al crear el agente vendedor, un valor de (26) al crear el agente comprador, al finalizar la compra toma un valor de (25) y al finalizar el agente vendedor toma un valor de (24) el mismo que se mantiene hasta finalizar la aplicación. Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

4.3.4 Gráfica Resultante Loaded Classes

En la Figura 55 se visualiza de color azul a los valores de objetos y de tomate a los valores de agentes, en el primer intervalo en objetos presenta un valor para clases cargadas que es numérico de (1,8) y agentes (1,1) , objetos predomina ya que presenta más procesos, a partir del segundo intervalo en objetos presenta un valor de (2,2) al registra el vendedor, un valor de (2,4) al registrar el comprador y un valor de (2,7) al finalizar la compra el cual se mantiene hasta el final de la aplicación, mientras que agentes presenta un valor de (3,1) al crear el agente vendedor, un valor de (4,2) al crear el agente comprador, un valor de (4,3) al finalizar la compra del libro y un valor de (4,5)al finalizar el agente vendedor el cual se mantiene hasta el cierre de la plataforma. Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

Gráfica final *loaded classes* Agentes vs Objetos

HORA	OBJETOS	AGENTES
12:50:00	1,8	1,1
12:50:10	2,2	3,1
12:50:20	2,4	4,2
12:50:30	2,7	4,3
12:50:40	2,7	4,5
12:53:20	2,7	4,5
12:55:00	2,7	4,5
12:58:00	2,7	4,5
13:00:00	2,7	4,5

Figura 55. Gráfica final *loaded classes* agentes vs objetos.

En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en eje x y en un intervalo numérico en el eje y.
Elaborado por Evelyn Obando & Doris Tituaña.

4.4 Gráficas CPU Agentes

4.4.1 Variación de *Heap Size* y *Used Size*

Variación valores *Heap Size y Used Size*

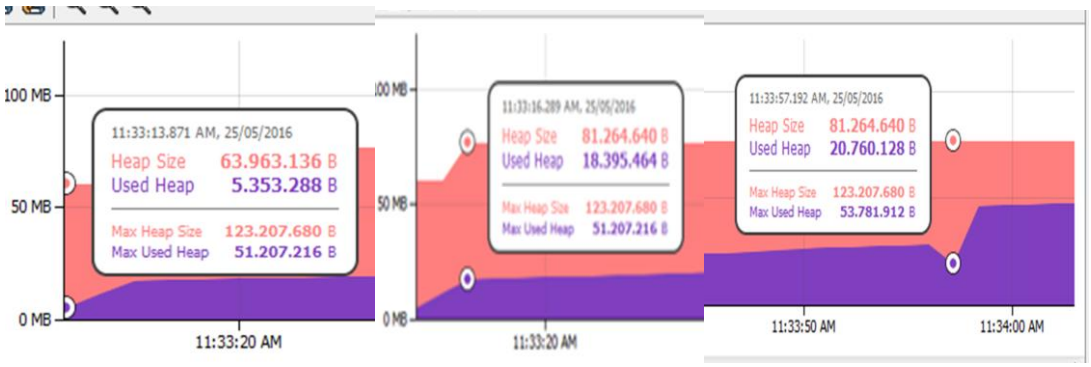


Figura 56. Variación valores *Heap Size* y *Used Size*.
En esta figura se muestra la variación en los valores de *Used Size* en el punto inicial (5,3B), medio (18,39B) y al crear el agente vendedor (20,76 B). El valor de *Heap Size* se inicia en (63,96B) y se mantendrá en (81.26B).
Elaborado por Evelyn Obando & Doris Tituaña.

4.4.2 Variación de *Surviving Generations*

Variación *Surviving Generations*

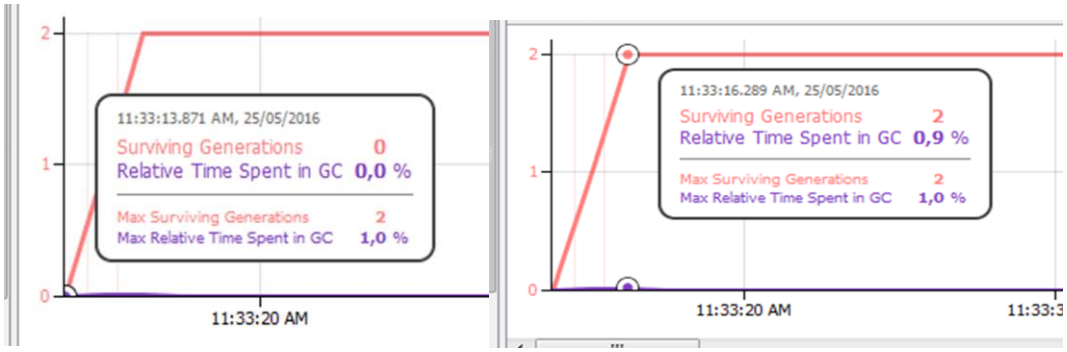


Figura 57. Variación *Surviving Generations.*

En esta figura se muestra la variación de los valores de *Surviving Generations* en el punto inicial es decir al ejecutar la aplicación tiene un valor de (0), al realizar el proceso de creación de agente vendedor tiene un valor de (2) el cual se mantiene hasta el cierre de la plataforma.
Elaborado por Evelyn Obando & Doris Tituaña.

4.4.3 Variación Threads, Loaded Classes

Threads, Loaded Classes

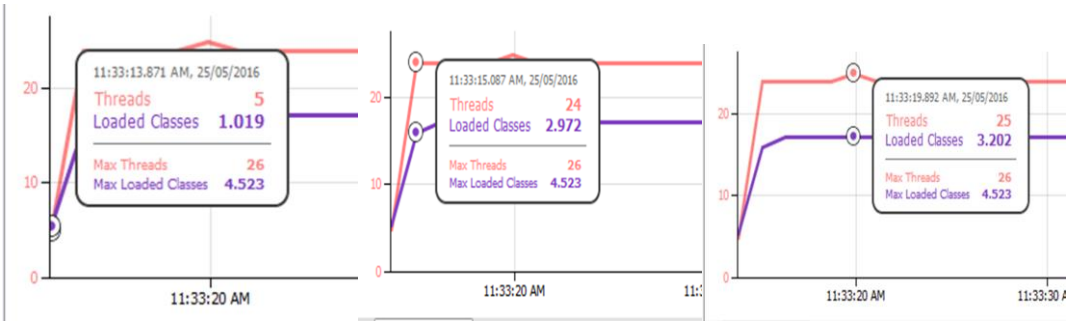


Figura 58. *Threads, Loaded Classes.*

En esta figura se muestra la variación en los valores de *Threads y Loaded Classes* al arranque de la plataforma (5) hilos y (1.01) clases cargadas, al crear el agente vendedor tiene un valor de (24) para hilos y (2.97) para clases cargadas, llega a un punto final con un valor de (25) para hilos y (3.20) para clases cargadas.
Elaborado por Evelyn Obando & Doris Tituaña.

4.5 Gráficas CPU Objetos

4.5.1 Variación *Heap Size* y *Used Heap*

En la Figura 59 se visualizarán los principales valores obtenidos en el momento el cual inicia la aplicación.

Valores *Heap Size y Used Heap* inicio de la Aplicación

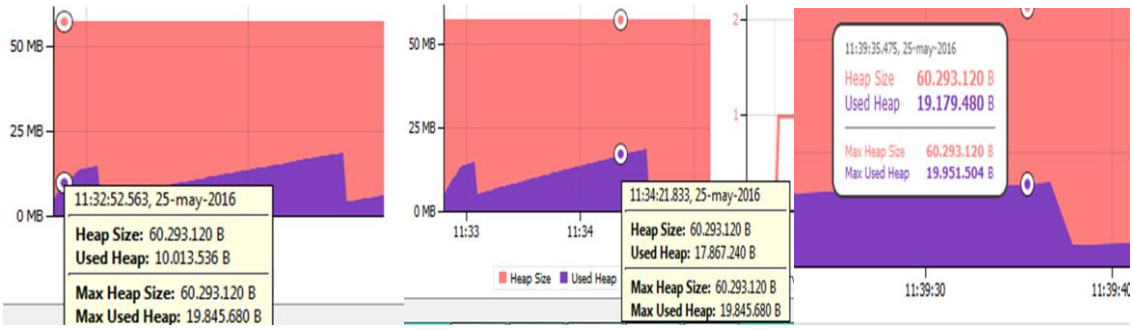


Figura 59. Valores *Heap Size* y *Used Heap* inicio de la aplicación.

En esta figura se muestran las variaciones en los valores de *Used Size* en los tres principales puntos en la cual arranca la aplicación como son: el punto inicial (10,1B) en el punto medio (17,8B) y en el punto final (19,17Mb). El valor de *Heap Size* es un valor constante de (60.29B), este valor puede presentar ciertas variaciones dependiendo del alcance que presente el *Used Size*.
Elaborado por Evelyn Obando & Doris Tituaña.

4.5.2 Variación *Surviving Generations*

En la Figura 60 se visualizarán los valores de supervivencia generados en la aplicación al iniciar su ejecución.

Variación *Surviving Generations*

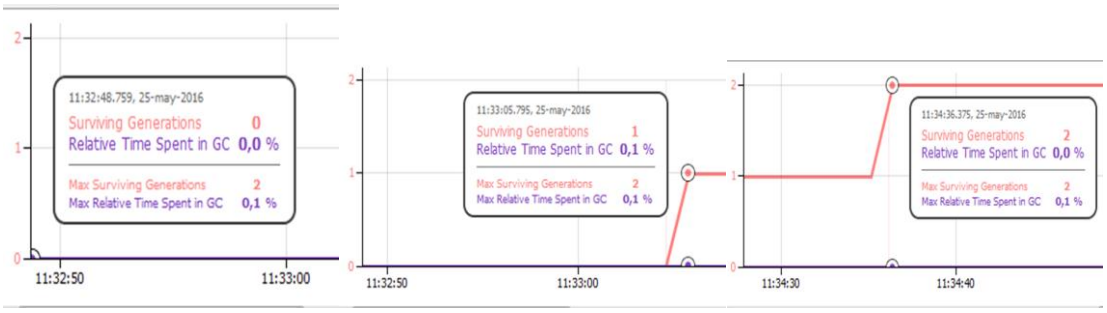


Figura 60. Variación de *Surviving Generations.*

En esta figura se muestra la variación en los valores de *Surviving Generations* en el punto inicial es decir al ejecutar la aplicación tiene un valor de (0) en el proceso de registro de vendedor y cambia a (1) y al finalizar el proceso de compra del libro cambia a (2), el último valor registrado se mantiene constante durante todo el proceso de la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

4.5.3 Variacion de *Threads,Loaded Classes*

La Figura 61 muestra las distintas variaciones que presenta la aplicación al ser ejecutada.

Variaciòn de *Threads,Loaded Classes*

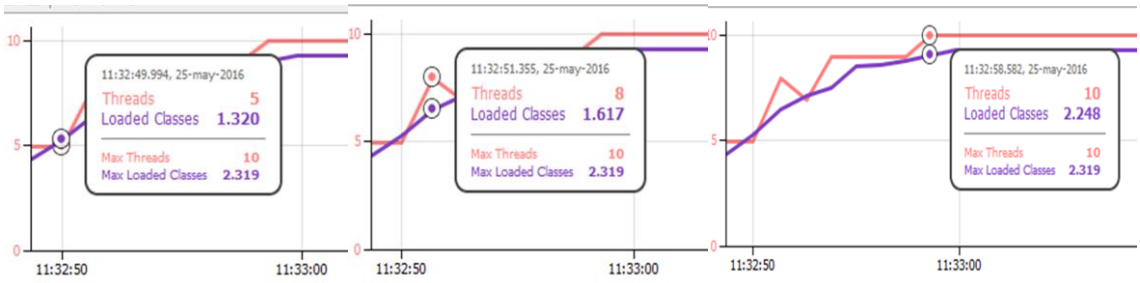


Figura 61. Variaciòn de *Threads,Loaded Classes.*

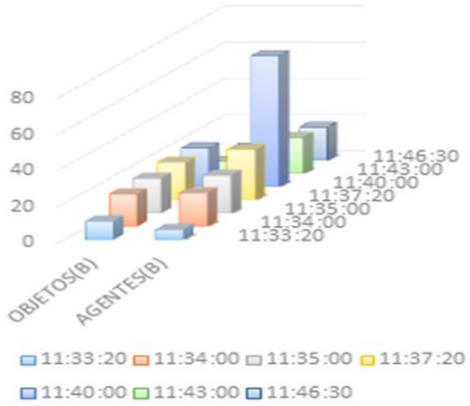
En esta figura se muestran los hilos y clases cargadas al momento de iniciar la aplicación, en el punto inicial se registran (5) hilos y (1.3) clases, en el punto medio se observa una variación de (8) hilos y (1.6) clases mientras que en el siguiente punto se registra una variación de (10) hilos y (2.2) clases, una vez registrado este último punto se mantendrá constante mientras se presente un cambio en la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

4.6 Gráficas Resultantes

4.6.1 Gráfica de *Used Size*

Una vez obtenido los valores de *Used Size* de Agentes y Objetos se procedió a realizar la gráfica general utilizando Excel como se muestra en la Figura 62.

Gráfica Final *Heap Size* Agentes vs Objetos



HORA	OBJETOS(B)	AGENTES(B)
11:33:20	10	5,3
11:34:00	17,8	18,3
11:35:00	19,1	20,7
11:37:20	20,7	27,8
11:40:00	21,22	72,14
11:43:00	6,5	19,72
11:46:30	6,5	18,1

Figura 62. Gráfica Final *Heap Size* agentes vs objetos.
En esta figura se observan los valores obtenidos tanto para agentes como para objetos en un intervalo de tiempo y en un rango de (0-100B).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 62 se visualizan barras de diferentes colores tanto para agentes. En el primer intervalo objetos presenta (10B) y agentes (5,3B) por lo que se determina que objetos consume más recursos debido a que las aplicaciones en objetos requieren más memoria del sistema operativo, a partir del segundo intervalo en adelante agentes predomina a objetos esto es debido a que al ejecutar la aplicación arranca simultáneamente la plataforma Jade en la cual se crearán los agentes.

4.6.2 Gráfica Resultante de *Surviving Generations*

Una vez obtenido los valores de *Surviving Generations* de agentes y objetos se proceden a realizar la gráfica general como se podrá visualizar en la Figura 63.

Gráfica Final de *Surviving Generations* agentes vs objetos

HORA	OBJETOS	AGENTES
11:33:20	0	0
11:34:00	1	2
11:35:00	2	2
11:37:20	2	2
11:40:00	2	2
11:43:00	2	2
11:46:30	2	2

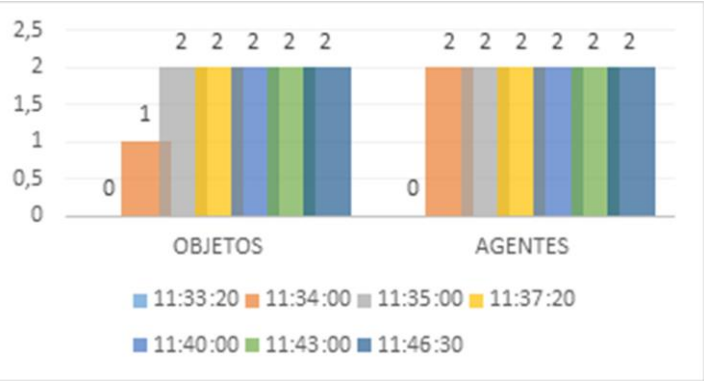


Figura 63. Gráfica final de *Surviving Generations* agentes vs objetos.
En esta figura se observan los valores obtenidos tanto para agentes como para objetos en un intervalo de tiempo en el eje x y en el eje y en un intervalo numérico de (0-3).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 63 se visualizan barras de diferentes colores tanto para objetos como para agentes.

El valor de supervivencia en el primer intervalo, objetos un valor numérico de (0) y para agentes(0) debido a que se inicia la ejecución de las aplicaciones, a partir del segundo intervalo es decir en la creación de vendedores y compradores objetos presenta un valor de(1) y agentes de (2) es decir que predomina agentes sobre objetos esto es debido a que al crear los agentes se utiliza la plataforma jade la cual ya se encuentra consumiendo recursos y memoria mientras que objetos no presenta ninguna plataforma estos valores se mantienen hasta finalizar las aplicaciones.

4.6.3 Gráfica Resultante Hilos

Una vez obtenido los valores de Generación de supervivencia de agentes y objetos se proceden a realizar la gráfica general como se podrá visualizar en la Figura 64.

Gráfica Final hilos Agentes vs Objetos

HORA	OBJETOS	AGENTES
11:33:20	5	5
11:34:00	8	24
11:35:00	10	25
11:37:20	11	26
11:40:00	11	25
11:43:00	11	24
11:46:30	11	24

Figura 64. Gráfica Final hilos Agentes vs Objetos.

En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en el eje x y en un intervalo numérico en el eje y.
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 64 se visualizan barras de diferentes colores tanto para objetos como para agentes, en el primer intervalo en objetos presenta un valor de hilos que es numérico de (5) y agentes(5), los dos presentan el mismo valor al iniciar la aplicación a partir del segundo intervalo en objetos presenta un valor de (8) y agentes un valor de (24), al crear el vendedor objetos presenta un valor de (10) mientras que agentes un valor de (25), al crear el comprador objetos presenta un valor de (11) el cual se mantiene hasta el final de la aplicación mientras que agentes presenta un valor de (26), al finalizar la compra del libro presenta un valor de (25) y al finalizar el agente vendedor toma un valor de (24). Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

4.6.4 Gráfica Resultante Clases Cargadas

Gráfica final clases cargadas Agentes vs Objetos

HORA	OBJETOS	AGENTES
11:33:20	1,3	1,06
11:34:00	1,6	3,1
11:35:00	2,2	3,2
11:37:20	2,7	4,5
11:40:00	2,7	4,5
11:43:00	2,7	4,5
11:46:30	2,7	4,5

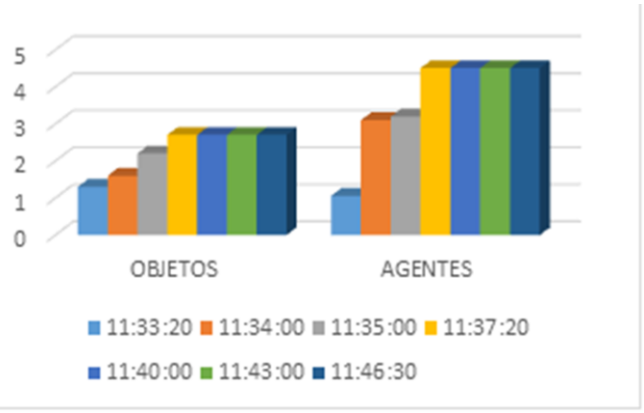


Figura 65. Gráfica Final hilos Agentes vs Objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en el eje x y en un intervalo numérico en el eje y.
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 65 se visualizan barras de diferentes colores tanto para objetos y agentes, en el primer intervalo en objetos presenta un valor para clases cargadas que es numérico de (1,3) y agentes (1,06), agentes predomina ya que presenta más procesos, a partir del segundo intervalo en objetos presenta un valor de (1,6) al registra el vendedor, un valor de (2,2) al registrar el comprador y un valor de (2,7) al finalizar la compra el cual se mantiene hasta el final de la aplicación, mientras que agentes presenta un valor de (3,1) al crear el agente vendedor, un valor de (3,2) al crear el agente comprador, un valor de (4,5) el cual se mantiene hasta el cierre de la plataforma. Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

4.7 Gráficas Memoria Agentes

4.7.1 Variación de *Heap Size* y *Used Size*

Variación valores *Heap Size y Used Size*

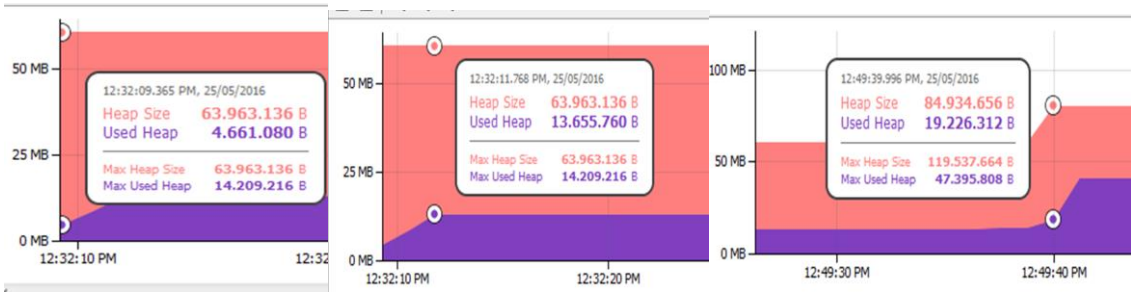


Figura 66. Variación valores *Heap Size* y *Used Size*.
En esta figura se muestra la variación en los valores de *Used Size* en el punto inicial (4,6B), medio (13,65B) y al crear el agente vendedor (19,22 B). El valor de *Heap Size* se inicia en (63,96B) y se mantendrá en (84.93B).
Elaborado por Evelyn Obando & Doris Tituaña.

4.7.2 Variación de Surviving Generations

Variación *Surviving Generations*

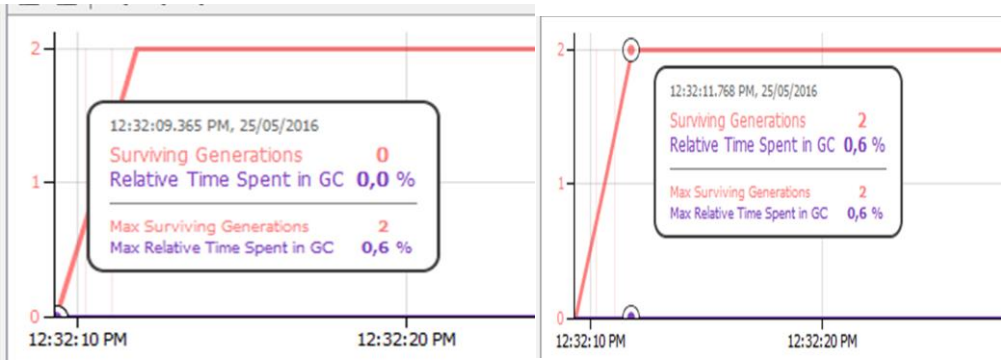


Figura 67. Variación *Surviving Generations*.

En esta figura se muestra la variación de los valores de *Surviving Generations* en el punto inicial es decir al ejecutar la aplicación tiene un valor de (0) en el proceso de registro de vendedor cambia a (1) y al finalizar el proceso de compra del libro cambia a (2).
Elaborado por Evelyn Obando & Doris Tituaña.

4.7.3 Variación *Threads, Loaded Classes*

Threads, Loaded Classes

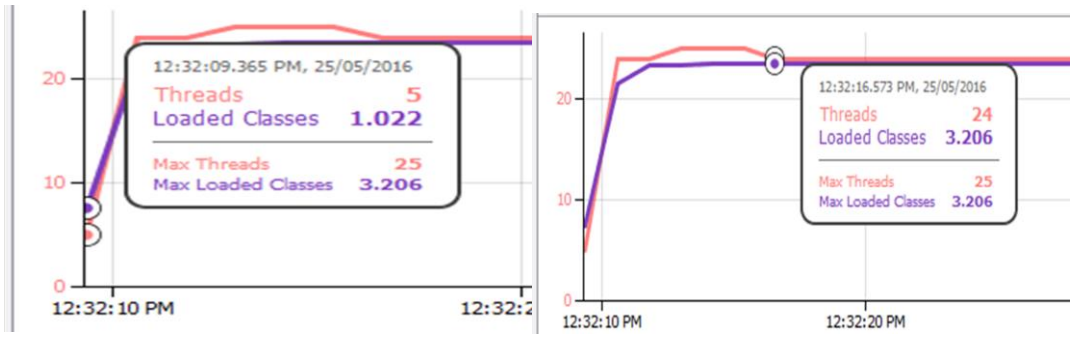


Figura 68. *Threads, Loaded Classes*.

En esta figura se muestra la variación en los valores de *Threads y Loaded Classes* al arranque de la plataforma (5) hilos y (1.02) clases cargadas, al crear el agente vendedor tiene un valor de (24) para hilos y (3.20).
Elaborado por Evelyn Obando & Doris Tituaña.

4.8 Gráficas Memoria Objetos

4.8.1 Variación *Heap Size* y *Used Heap*

Valores *Heap Size* y *Used Heap* inicio de la aplicación

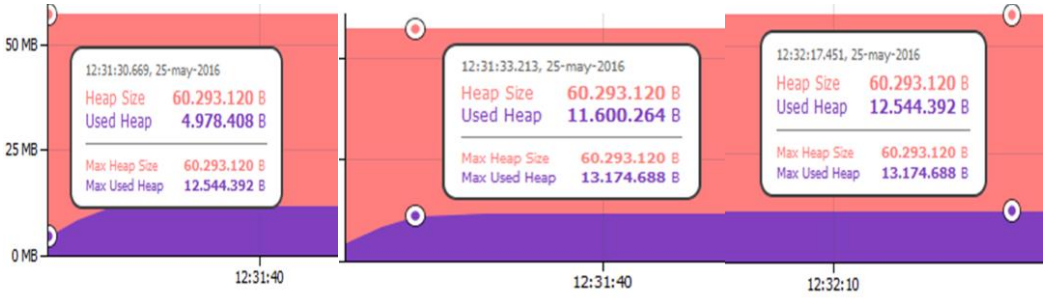


Figura 69. Valores Heap Size y Used Heap Inicio de la Aplicación.
En esta figura se muestran las variaciones en los valores de *Used Size* en los tres principales puntos en la cual arranca la aplicación como son: el punto inicial (4,9B) en el punto medio (11,60B) y en el punto final (12.54B). El valor de *Heap Size* es un valor constante de (60.29 Mb), este valor puede presentar ciertas variaciones dependiendo del alcance que presente el *Used Size.*
Elaborado por Evelyn Obando & Doris Tituaña.

4.8.2 Variación de *Surviving Generations*

Variación *Surviving Generations*

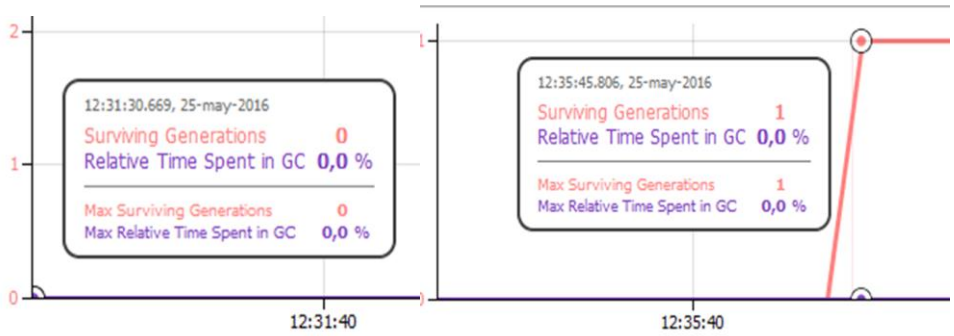


Figura 70. Variación *Surviving Generations.*

En esta figura se muestra la variación en los valores de *Surviving Generations* en el punto inicial es decir al ejecutar la aplicación tiene un valor de (0) en el proceso de registro de vendedor y cambia a (1) este último valor registrado se mantiene constante durante todo el proceso de la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

4.8.3 Variación *Threads, Loaded Classes*

Variación de threads y loaded classes.

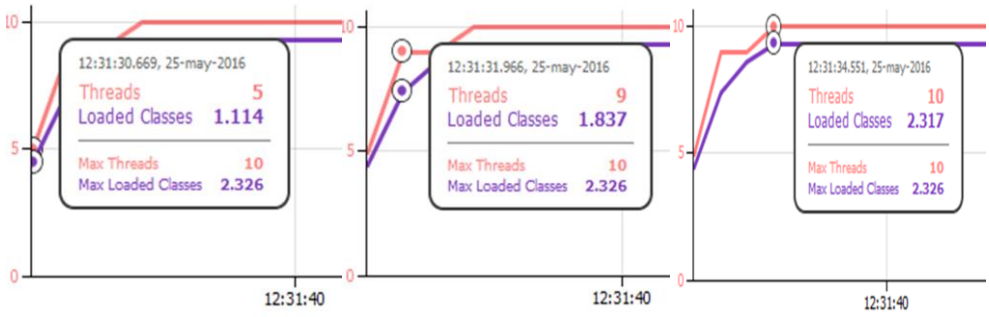


Figura 71. Variación de *threads* y *loaded classes*.

En esta figura se muestran los hilos y clases cargadas al momento de iniciar la aplicación, en el punto inicial se registran (5) hilos y (1.1) clases, en el punto medio se observa una variación de (9) hilos y (1.8) clases mientras que en el siguiente punto se registra una variación de (10) hilos y (2.3) clases, una vez registrado este último punto se mantendrá constante mientras se presente un cambio en la aplicación.
Elaborado por Evelyn Obando & Doris Tituaña.

4.9 Gráficas Resultantes

4.9.1 Gráfica de *Used Size*

Una vez obtenido los valores de *Used Size* de agentes y objetos se procede a realizar la gráfica general utilizando Excel como se podrá visualizar en la Figura 72.

Gráfica Final *Heap Size* Agentes vs Objetos

HORA	OBJETOS(B)	AGENTES(B)
11:33:20	4,9	4,6
11:34:00	11,6	13,6
11:35:00	12,5	19,22
11:37:20	15	43,1
11:40:00	6,3	20,7
11:43:00	15,1	19
11:46:30	19,3	16,2

Figura 72. Gráfica Final Heap Size Agentes vs Objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo y en un rango de (0-100B).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 72 se visualizan de color azul a los objetos y de color tomate a los agentes. En el primer intervalo objetos presenta (4,9B) y agentes (4,6B) por lo que se determina que objetos consume más recursos debido a que los objetos requieren más memoria del sistema operativo, a partir del segundo intervalo en adelante agentes predomina a objetos esto es debido a que al ejecutar la aplicación arranca simultáneamente la plataforma Jade en la cual se crearán los agentes.

4.9.2 Gráfica Resultante de *Surviving Generations*

Una vez obtenido los valores de *Surviving Generations* de agentes y objetos se proceden a realizar la gráfica general como se visualiza en la Figura 73.

Gráfica final *Surviving Generations* agentes vs objetos

HORA	OBJETOS	AGENTES
11:33:20	0	0
11:34:00	1	2
11:35:00	1	2
11:37:20	1	2
11:40:00	1	2
11:43:00	1	2
11:46:30	1	2

Figura 73. Gráfica Final *Surviving Generations* agentes vs objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en eje x y en el eje y en un intervalo numérico de (0-3).
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 73 se visualiza de color azul a los valores de objetos y de tomate a los valores de agentes.

El valor de supervivencia en el primer intervalo objetos presenta un valor numérico de (0) y agentes (0) debido a que se están ejecutando las aplicaciones, a partir del segundo intervalo es decir en la creación de vendedores y compradores objetos presenta un valor de (1) y agentes de (2) los cuales se mantienen hasta finalizar las aplicaciones, es decir que predomina agentes sobre objetos esto es debido a que al crear los agentes se utiliza la plataforma jade la cual ya se encuentra consumiendo recursos y memoria mientras que objetos no presenta ninguna plataforma al ejecutar la aplicación.

4.9.3 Gráfica Resultante Hilos

Gráfica Final hilos Agentes vs Objetos

HORA	OBJETOS	AGENTES
11:33:20	5	5
11:34:00	9	24
11:35:00	10	25
11:37:20	11	26
11:40:00	11	25
11:43:00	11	24
11:46:30	11	24

Figura 74. Gráfica Final hilos Agentes vs Objetos.

En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en eje x y en un intervalo numérico en el eje y.
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 74 se visualizan barras de diferentes colores tanto para objetos como para agentes, en el primer intervalo en objetos presenta un valor de hilos que es numérico de (5) y agentes(5), los dos presentan el mismo valor al iniciar la aplicación a partir del segundo intervalo en objetos presenta un valor de (9) y agentes un valor de (24), al crear el vendedor objetos presenta un valor de (10) mientras que agentes un valor de(25), al crear el comprador objetos presenta un valor de (11) el cual se mantiene hasta el final de la aplicación mientras que agentes presenta un valor de (26), al finalizar la compra del libro presenta un valor de (25) , al finalizar el agente vendedor toma un valor de (24). Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

4.9.4 Gráfica Resultante Clases Cargadas

Gráfica final clases cargadas Agentes vs Objetos

HORA	OBJETOS	AGENTES
11:33:20	1,3	1,06
11:34:00	1,6	3,1
11:35:00	2,2	3,2
11:37:20	2,7	4,5
11:40:00	2,7	4,5
11:43:00	2,7	4,5
11:46:30	2,7	4,5

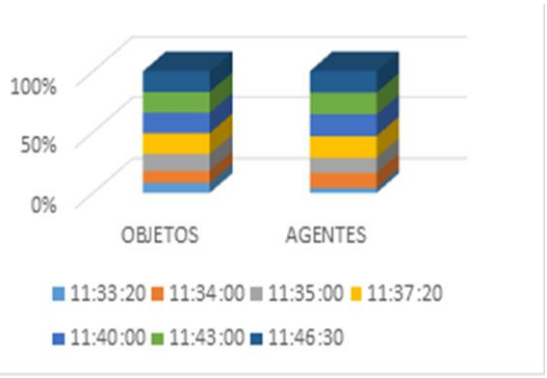


Figura 75. Gráfica Final hilos Agentes vs Objetos.
En esta figura se observa una tabla con los valores obtenidos previamente tanto para agentes como para objetos en un intervalo de tiempo en eje x y en un intervalo numérico en el eje y.
Elaborado por Evelyn Obando & Doris Tituaña.

En la Figura 75 se visualizan cuadros de diferentes colores tanto para objetos y agentes, en el primer intervalo en objetos presenta un valor para clases cargadas que es numérico de (1,3) y agentes (1,06), agentes predomina ya que presenta más procesos, a partir del segundo intervalo en objetos presenta un valor de (1,6) al registra el vendedor, un valor de (2,2) al registrar el comprador y un valor de (2,7) al finalizar la compra el cual se mantiene hasta el final de la aplicación, mientras que agentes presenta un valor de (3,1) al crear el agente vendedor, un valor de (3,2) al crear el agente comprador, un valor de (4,5), el cual se mantiene hasta el cierre de la plataforma. Cabe mencionar que agentes presenta mayores variaciones debido al uso de la plataforma Jade.

4.10 Prueba de Hipótesis

Una prueba de hipótesis es una prueba estadística que se utiliza para determinar si existe suficiente evidencia en una muestra de datos para inferir que cierta condición es válida para toda la población (www.minitab.com).

1) Estudiantes de la Universidad Politécnica Salesiana están interesados en utilizar un nuevo lenguaje de programación. Tras realizar una encuesta a los estudiantes de la carrera de Ingeniería en Sistemas, se toma la muestra de 400 estudiantes, de los cuales, 12 conocían del lenguaje de programación. A un nivel de significación del 1% ¿apoya el estudio las siguientes hipótesis?

   a. Más del 3% de los estudiantes no conoce el nuevo lenguaje de programación.
   b. Menos del 2% los estudiantes no conoce el nuevo lenguaje de programación

Datos:

n =400

x = 12

$\alpha = 1\% = 0{,}01$

$$z_{prueba} = \frac{\frac{x}{n} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$$

Donde:

x = ocurrencias

n = observaciones

$\frac{x}{n}$= proporción de la muestra

p_0= proporción propuesta

Solución:

a)

$po = 3\% = 0{,}03$

$Hipótesis\ Nula: H_o: p = p_0$

$Hipótesis\ Alternativa: H_1: p > p_0$

a = 0,01 $\Longrightarrow z_{tabla} = 2{,}326$

$$Zprueba = \frac{\frac{x}{n} - po}{\sqrt{\frac{po(1-po)}{n}}} = \frac{\frac{400}{10} - 0{,}03}{\sqrt{\frac{0{,}03(1-0{,}03)}{400}}} = \frac{123.97}{0.00685} = -0{,}586210382$$

Prueba de hipótesis

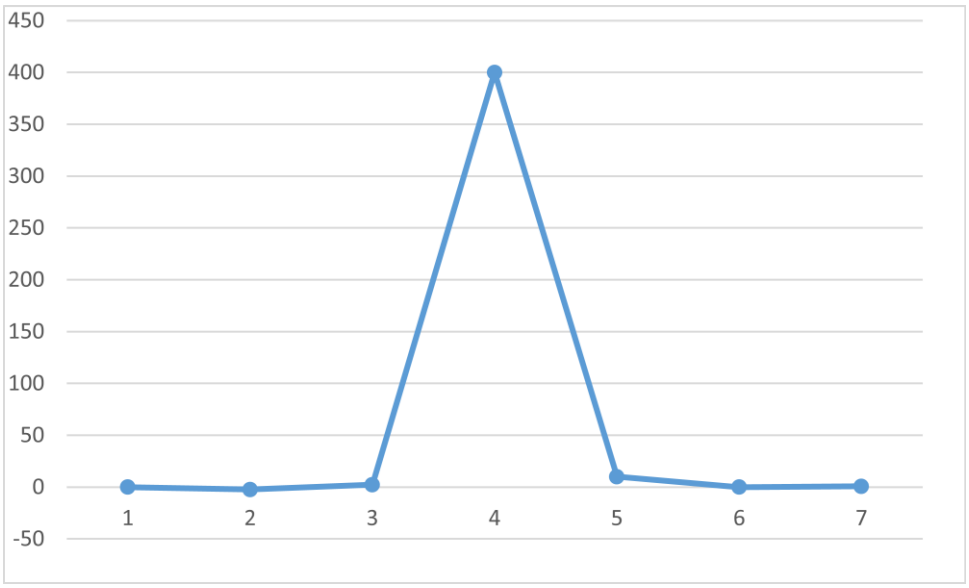


Figura 76. Prueba de hipótesis
H_0 es aceptada, ya que z_{prueba} (-0.586) es menor que z_{tabla} (2,326), por lo que es cierto que más del 3% de los estudiantes no conoce el nuevo lenguaje de programación.
Elaborado por Evelyn Obando & Doris Tituaña.

Solución:

b)

$po = 2\% = 0{,}02$

$Hipótesis\ Nula: H_O: p = p_0$

$Hipótesis\ Alternativa: H_1: p > p_0$

a = 0,01 $\Longrightarrow z_{tabla} = 2{,}326$

$$Zprueba = \frac{\frac{x}{n} - po}{\sqrt{\frac{po(1-po)}{n}}} = \frac{\frac{400}{10} - 0{,}02}{\sqrt{\frac{0{,}02(1-0{,}02)}{400}}} = \frac{39{,}98}{0.007} = 0{,}714285714$$

Prueba de hipótesis

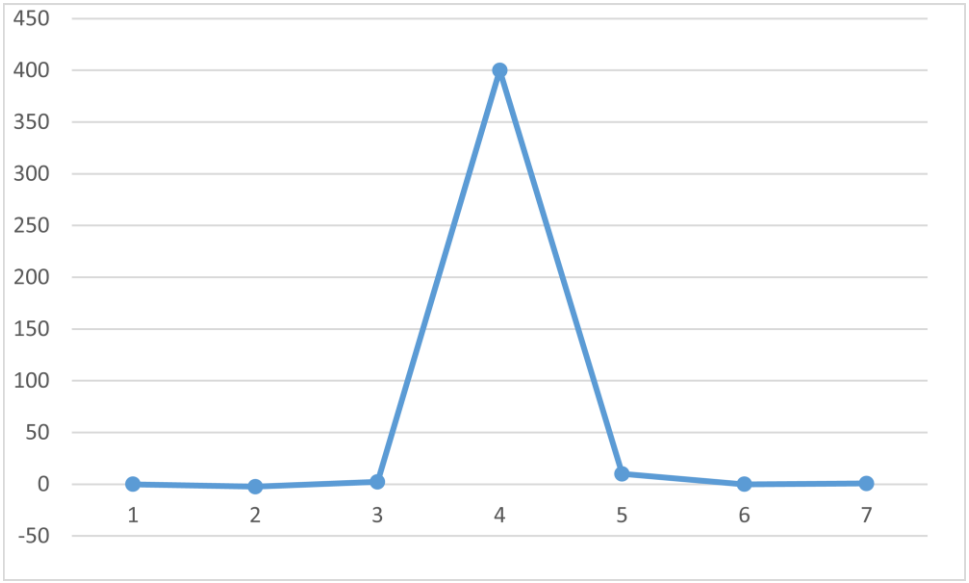


Figura 77. Prueba de hipótesis.
H_0 es aceptada, ya que z_{prueba} (0.714) es menor que z_{tabla} (2,326), por lo que es cierto que menos del 2% de los estudiantes conoce el nuevo lenguaje.
Elaborado por Evelyn Obando & Doris Tituaña.

2: Estudiantes de la Universidad Politécnica Salesiana necesitan saber que lenguaje de programación utiliza más recursos de memoria. Tras tomar los valores obtenidos anteriormente en la Figura 72 se va a comparar la Programación Orientada a Objetos (POO) con la Programación Orientada Agentes (POA). A un nivel de significación del 1% ¿apoya el estudio las siguientes hipótesis?

a. Programación Orientada Agentes consume más del 5% de memoria.

b. Programación Orientada a Objetos Menos del 5% de memoria.

HORA	OBJETOS(B)	AGENTES(B)
11:33:20	4,9	4,6
11:34:00	11,6	13,6
11:35:00	12,5	19,22
11:37:20	15	43,1
11:40:00	6,3	20,7
11:43:00	15,1	19
11:46:30	19,3	16,2

Valor promedio de Objetos =12,1B

Valor promedio de Agentes =19,48B

Valor desviación estándar de Objetos =5,08B

Valor desviación estándar de Agentes =11,74B

Datos:

n =3

x = 19.48

$$p = \frac{3}{19.48} = 0.15$$

$$\alpha = 1\% = 0{,}01$$

$$z_{prueba} = \frac{\frac{x}{n} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$$

Donde:

x = ocurrencias

n = observaciones

$\frac{x}{n}$= proporción de la muestra

p_0= proporción propuesta

Solución:

a)

$po = 5\% = 0{,}05$

$Hipótesis\ Nula: H_O: p = p_0$

$Hipótesis\ Alternativa: H_1: p > p_0$

a = 0,01 $\Longrightarrow z_{tabla} = 2{,}326$

$$Zprueba = \frac{\frac{x}{n} - po}{\sqrt{\frac{po(1-po)}{n}}} = \frac{\frac{3}{19.48} - 0{,}05}{\sqrt{\frac{0{,}05(1-0{,}05)}{3}}} = 51.20$$

Prueba Hipótesis

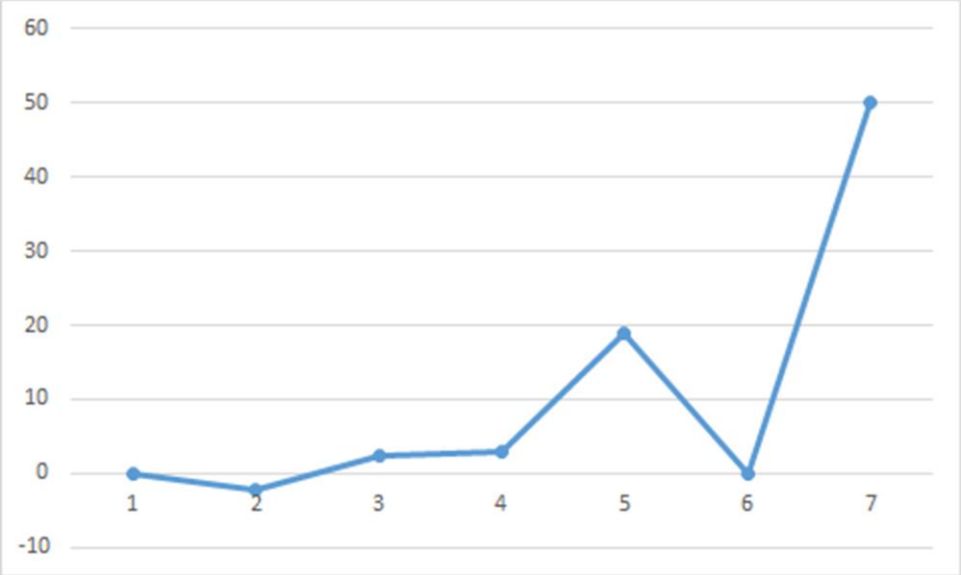


Figura 78. Prueba Hipótesis.

H_0 es rechazada, ya que z_{prueba} (51.20) es menor que z_{tabla} (2,326), por lo que es cierto que la programación orientada agentes consume más del 5% de memoria.
Elaborado por Evelyn Obando & Doris Tituaña.

Datos:

n =3

x = 12.1

$$p = \frac{3}{12.1} = 0.24$$

$$\alpha = 1\% = 0{,}01$$

$$z_{prueba} = \frac{\frac{x}{n} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$$

Donde:

x = ocurrencias

n = observaciones

$\frac{x}{n}$= proporción de la muestra

p_0= proporción propuesta

Solución:

b)

$po = 5\% = 0{,}05$

$Hipótesis\ Nula: H_o: p = p_0$

$Hipótesis\ Alternativa: H_1: p > p_0$

a = 0,01 $\Longrightarrow z_{tabla} = 2{,}326$

$$Zprueba = \frac{\frac{x}{n} - po}{\sqrt{\frac{po(1-po)}{n}}} = \frac{\frac{3}{12.1} - 0{,}05}{\sqrt{\frac{0{,}05(1-0{,}05)}{3}}} = 31.41$$

Prueba de hipótesis

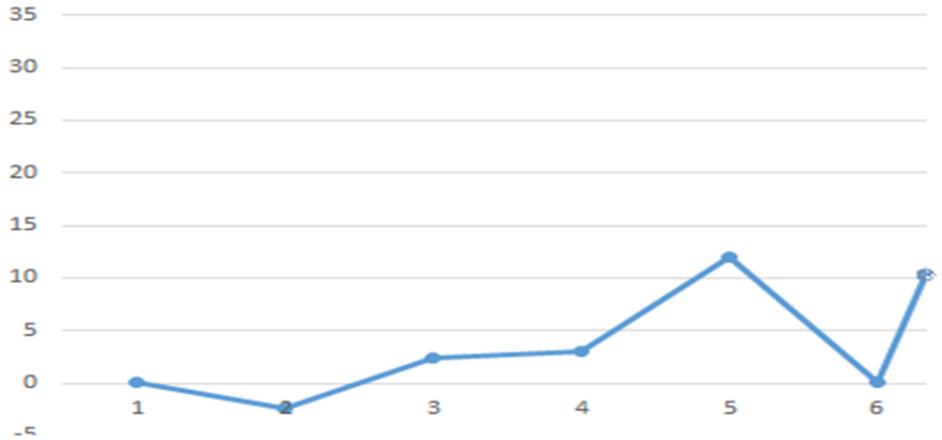


Figura 79. Prueba de hipótesis objetos.

H_0 es rechazada, ya que z_{prueba} (31.40) es menor que z_{tabla} (2,326), por lo que es cierto que la programación orientada a objetos consume más del 5% de memoria.
Elaborado por Evelyn Obando & Doris Tituaña.

CONCLUSIONES

1. Una vez analizada la Programación Orientada a Agentes (POA) se concluye que los agentes de software tienen sus propios hilos de control, metas y reglas individuales, son considerados como entidades autónomas, con un conjunto de responsabilidades internas, son entidades interactivas capaces de comunicarse con otros agentes mediante mensajes, son capaces de iniciar una acción independientemente de cualquier otra entidad. Los agentes son diseñados para determinar su comportamiento basado en metas y estados individuales.

2. Jade es una herramienta de desarrollo de agentes y sistemas multi-agentes que incluye un entorno de ejecución, librería de clases y herramientas gráficas, cada entorno de ejecución es llamado Main Container el cual siempre está activo en la plataforma está basado en el estándar FIPA. Jade funciona con un conjunto de agentes auxiliares: RMA (Gestor de la plataforma), Dummy agent (permite interactuar con agentes), Sniffer agent (Muestra las interacciones que se producen es decir el contenido de cada mensaje), DF (proporciona servicios de páginas amarillas), AMS (proporciona el nombre del servicio), un agente es considerado un hilo que tiene un ciclo de vida. Con la ayuda de la plataforma Jade se crearon los respectivos agentes para la aplicación.

3. La metodología Ingenias proporciona soluciones de ingeniería para desarrollar sistemas multi-agentes a través de lenguajes visuales que facilitan su diseño, además mantiene varios aspectos necesarios para su comprensión tales como: agentes, los cuales desempeñan los roles es decir buscan satisfacer el objetivo por cual fueron creados, organización, que define la estructura de un sistema multi-agente, objetivos, los mismos que son conocidos como las tareas generales es decir el rol de un agente, interacciones, este aspecto permitirá identificar las relaciones que mantienen los agentes y los roles en el diagrama.

4. Al realizar la aplicación con (POA) se evidencia que el uso de agentes no es muy eficiente con respecto a objetos, debido al uso excesivo de memoria, cantidad de hilos y clases cargadas, por el contrario los agentes son de ayuda en la POA ya que los agentes son creados para cumplir un objetivo específico, son capaces de tomar decisiones propias y a su vez realizan procesos internos mediante comportamientos, en los cuales no interviene el programador mientras que objetos son considerados pasivos porque sus métodos son invocados solo cuando entidades externas les envían mensajes.

5. Al realizar la aplicación con (POO) se utilizó un número de clases excesivas para lograr obtener una aplicación similar a agentes, los objetos no presentan un ciclo de vida es decir que están activos mientras el programador no finalice la aplicación, realizan llamadas a los métodos, las clases en objetos son diseñadas para facilitar la reutilización de los componentes.

6. Con los datos obtenidos en las gráficas de las métricas: monitor, CPU, memoria y los respectivos valores de (*Heap Size y Used Size*, *Surviving Generations*, *Threads*, *Loaded Classes*) se pudo observar que agentes tiene un valor más alto en las métricas con respecto a objetos ya que utiliza la plataforma Jade desde el inicio hasta el final de la aplicación la cual es necesaria para la creación de agentes.

7. Los multi-agentes se derivan de la interacción de un conjunto de agentes los cuales muestran su estado, son descentralizados es decir que no hay un agente de control, para lograr una interacción exitosa los agentes requieren la habilidad de cooperar, coordinarse y negociar unos con otros, pueden ser utilizados para resolver problemas que son difíciles para un agente individual.

8. La comunicación entre agentes se realiza mediante mensajes que representan actos de habla y que se codifican en un lenguaje de comunicación de

agentes(ACL), el contenido del mensaje se expresa en un lenguaje de contenido(SL) que hace referencia a una ontología (semántica), mientras que objetos realiza la comunicación mediante mensajes pasivos es decir invocaciones a los métodos.

GLOSARIO

JADE: Es un entorno que simplifica la implementación de sistemas multi-agente mediante una capa de soporte (middleware) que respeta las especificaciones FIPA (Marchetti & Garcìa, 2010, pág. 2)

FIPA: Lenguaje de comunicación entre agentes. (Aranda, 2008)

FIPA-ACL: Agente Comunicación de Idiomas. (Aranda, 2008)

RMA: Monitoreo Agente Remoto, Permite gestionar, monitorear y controlar los agentes de una forma remota. (Garamendi, 2004)

AMS: Garantiza que cada agente en la plataforma tenga un único nombre (Aranda, 2008).

DF: Agente que proporciona el servicio de páginas amarillas. (Aranda, 2008)

JADE-LEAP: Plataforma de agentes compatibles con entornos Java Móviles (Cuesta, 2008, pág. 5).

RMI: Es una técnica no invasiva que utiliza el fenómeno de la resonancia magnética para obtener información sobre la estructura y composición del cuerpo a analizar (Moral, 2013)

FRAMEWORK: Es una plataforma software para el desarrollo de agentes (Marchetti & Garcìa, 2010).

INTEROPERABILIDAD: Es la capacidad que tiene un sistema, cuyas interfaces son totalmente conocidas, para funcionar con otros sistemas (Chevrel, s.f.).

ABSTRACTA: Es algo que está en el universo de las ideas, los pensamientos, pero que no se puede concretar en algo material, que se pueda tocar (Alvarez M. , 2014).

ESTÁNDAR: Es un documento establecido por consenso, aprobado por un cuerpo reconocido, y que ofrece reglas, guías o características para que se use repetidamente. (PMI.org)

INSTANCIA: Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro, menos la clase Object que es la madre de todas. (upv)

DEPURACIÓN: El debugging o depuración es el proceso metodológico para encontrar y reducir bugs (errores) o defectos en un programa informático o en una pieza de hardware (alegsa.com.ar).

THREAD: Son procesos ligeros, con línea de flujo de control propia pero que comparte el espacio de direcciones del programa. Los threads hacen posible la ejecución concurrente de código (Drake & Barros, pág. 2).

TRACEAR: Determinar la ruta a un destino mediante el envío de paquetes de eco de Protocolo de mensajes de control de Internet al destino (Microsoft Windows, 2016)

CREENCIAS: Son expresiones que califican la verdad de los juicios para lo cual se utilizan operadores modales.

OPERADORES MODALES: Son expresiones que califican la verdad de los juicios.

PLANES: Son secuencias de acciones que un agente puede realizar.

ONTOLOGÍAS: Son acuerdos, en un contexto social, para cubrir una serie de objetivos. (GRUBER, 1993).

REFERENCIAS

Aguilar, J., & Abraham, B. (2009). Sistema multiagente para el manejo de incidentes de seguridad. *Revista Ciencia e Ingeniería-Universidad de los Andes*, 12.

Ahogado, D., & Reinemer, A. (2002). *Nuevos Paradigmas de Desarollo de Software-La Programación Orientada a Agentes.*

Alonso, J. B. (2015). *Sistemas Multiagentes-Universidad Politecnica Catalunya-Departamento de Ciencias de la Computación*.

Alvarez, M. (24 de 7 de 2001). *Qué es la programacion orientada a objetos.* Obtenido de desarrolloweb.com: http://www.desarrolloweb.com/articulos/499.php

Aranda, G. (1 de Julio de 2008). *Taller de Programaciòn de Agentes con JADE,Escuela Politécnica Superior " La Rábida" (Gonzalo Aranda,Tècnicas de Difusiòn de la Tecnologìa Multiagente en la Universidad de Sevilla)*. Obtenido de http://jmas.us.es/: http://jmas.us.es/doc/2010/jmas2010.Aranda.pdf

Azawi, R., & Ayesh, A. (8 de Mayo de 2013). *COMPARING AGENT-ORIENTED PROGRAMMING VERSUS OBJECT ORIENTED PROGRAMMING.* Obtenido de the 6th International Conference on Information Technology: https://www.researchgate.net/publication/261288542_COMPARING_AGENT-ORIENTED_PROGRAMMING_VERSUS_OBJECT-ORIENTED_PROGRAMMING

Bravo, C., Aguilar, J., & Rivas, F. (2001). Diseño de una arquitectura de automatización industrial basada en sistemas multi-agentes. *Revista Ciencia e Ingeniería-Universidad de los Andes*, 16.

Carrillo, J. L. (2005). *ANÁLISIS DE METODOLOGÍAS PARA DESARROLLO DINTRANETE AGENTES, CASO PRÁCTICO: BUSCADOR INTELIGENTE PARA UNA.* Obtenido de ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO-TESIS MASTER EN INFORMATICA: http://dspace.espoch.edu.ec/bitstream/123456789/4219/1/20T00013.pdf

Essaidi, M., Ganzha, M., & Paprzycki, M. (2012). *Software Agents, Agent Systems and their Applications* . IOS Press.

Fernández, J. (4 de 2 de 2013). *Un poco de Java, Jade.* Obtenido de Java Agent DEvelopment Framework (JADE): https://unpocodejava.wordpress.com/?s=JADE

Fernandez, M. (s.f.). *Departamento de Ciencias de la Computación Universidad de Chile(Mario Fernández, Semantic Web + Agents = Future Web)*. Obtenido de http://dcc.uchile.cl/: http://users.dcc.uchile.cl/~mfernand/index.php?slug=agents

Fernández, V. (s.f.). *SIMULADOR DEL ESTÁNDAR TCP DEL IETF EN C++*. Obtenido de Universidad de Sevilla-Biblioteca de Ingeniería: http://bibing.us.es/proyectos/abreproy/11144/fichero/PROYECTO%252FCAPITULO+3.PROGRAMACION+ORIENTADA+A+OBJETOS.pdf

Garamendi, J. (2004). Introducción a JADE-Java Agent DEvelopment Framework. *Curso de doctorado Sistemas- Multiagente Universidad Politécnica de Valencia*. España.

Gimeno, J. M., & González , J. (2011). *Introducción a Netbeans.* Obtenido de http://ocw.udl.cat/enginyeria-i-arquitectura/programacio-2/continguts-1/1-introduccioi81n-a-netbeans.pdf

González, D. (2005). *AUML Sistemas multi-agentes*. Obtenido de Escuela Superior de Ingenieria Informatica-Universidad de Vigo -España: http://trevinca.ei.uvigo.es/~pcuesta/sm/alumnos2005/AUML.pdf

GRUBER, T. (1993). *"A translation approach to portable ontologies"*. Obtenido de http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html

Gutierrez, D. (Mayo de 2011). *UML Diagrama de Secuencias ,Universidad de los Andes Venezuela*. Obtenido de http://www.codecompiling.net/files/slides/UML_clase_06_UML_secuencia.pdf

Hernández, L. (s.f.). *Uso de NetBeans.* Obtenido de Facultad de Informática - Universidad Complutense: https://www.fdi.ucm.es/profesor/luis/fp/devtools/NetBeansUso.html

Iglesias, C. (enero de 1998). *DEFINICIÓN DE UNA METODOLOGÍA PARA EL DESARROLLO DE SISTEMAS MULTIAGENTE.* Obtenido de Tesis Doctoral-Universidad Politécnica de Madrid-Departamento de Ingeniería de Ssitemas Telemáticos: http://www.upv.es/sma/teoria/agentes/tesiscif.pdf

Izquierdo, L. R. (s.f.). *Luis R. Izquierdo.* Obtenido de Introducción a la Progrmación Orientada a Objetos : http://luis.izqui.org/resources/ProgOrientadaObjetos.pdf

Jimenez, C. O. (2013). *PROGRAMACION ORIENTA A OBJETOS.* Obtenido de UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD-ESCUELA DE CIENCIAS BASICAS TECNOLOGÍA E INGENIERIA-COLOMBIA: http://datateca.unad.edu.co/contenidos/301403/MODULO_PROGRAMACION_ORIENTA_A_OBJETOS_2013.pdf

León Paredes, S. E. (Octubre de 2012). *Repositorio Digital Espe.* Recuperado el 08 de 03 de 2016, de http://repositorio.espe.edu.ec/handle/21000/5901

Manotas, A. d. (2010). DISEÑO DE UN SISTEMA MULTI-AGENTE PARA MONITOREO DE REDES UTILIZANDO JADE Y JPCA. *Revista Inge-CUC* , 10.

Manso, M. Á. (2002). *Programación Orientada a Objetos* . Obtenido de Universidad Politécnica de Madrid-Curso de Informatica : http://pdi.topografia.upm.es/m.manso/docencia/Informatica_plan92/Curso-2002-2003/poo.pdf

Marchetti, T., & Garcìa, A. (2010). Plataformas para Desarrollo de Sistemas Multiagente. *Desarrollo en Inteligencia Artificial,Departamento de Ciencias,Universidad Nacional del Sur*. Bahía Blanca, Argentina.

Mestras, J. P. (2007). *Universidad Complutense Madrid.* Obtenido de Fundamentos de la Programación Orientada a Objetos-Dep. Ingeniería del Software e Inteligencia Artificial: https://www.fdi.ucm.es/profesor/jpavon/poo/1.1.Objetos%20y%20Clases.pdf

Morales, P. C. (25 de Mayo de 2009). *Sistema Multiagentes*. Obtenido de Sistema Multiagentes: http://es.slideshare.net/pcuestaesei/la-herramienta-de-desarrollo-de-agentes-jade

Navarrete, E. (agosto de 2008). "DESARROLLO DE UNA AGENDA MULTIAGENTE ORIENTADA A LA WEB PARA LA GESTIÓN DE REUNIONES UTILIZANDO LA PLATAFORMA JADE – JAVA EN LA ESCUELA POLITÉCNICA DEL EJÉRCITO SEDE LATACUNGA" . *Tesis para obtener el título de Ingeniería en Informática - Escuela Politécnica del Ejército Sede Latacunga*. Latacunga, Cotopaxi , Ecuador.

netbeans.org. (s.f.). *Profiler.* Obtenido de https://profiler.netbeans.org/

Oviedo, D. (5 de Abril de 2010). *Introducción al desarrollo en la Plataforma JADE,Universidad de Sevilla*. Obtenido de Departamento de Tecnoloìas Electrònicas: https://www.dte.us.es/Members/oviedo/SMID/TP1

Pavon, J. (s.f.). *INGENIAS: Desarrollo dirigido por modelos de SMA.* Obtenido de Dep. de Ingeniería del Software e Inteligencia Artificial-Universidad Complutense Madrid-España: http://jmas.us.es/doc/6.Pavon_JMASweb.pdf

Pedra, M. (2014). *Qué significa la nomenclatura del Software* . Obtenido de http://www.marcelopedra.com.ar/blog/2014/04/07/que-significa-la-nomenclatura-del-software-beta-vl-retail-rtm-shareware-etc/

Prieto, M. (26 de Junio de 2013). *Metodologia AUML*. Obtenido de https://prezi.com/yxsejlmqfwag/auml/

Ramirez, K. (Abril de 2008). Obtenido de Lenguajes de Comunicación- Protocolos y Plataforma JADE. Tecnologías para el Desarrollo de Sistemas Multi-Agente-Universidad de Costa Rica.

Riesco, D. (s.f.). *UML Diagramas de Clases y Objetos*. Obtenido de Software Engineering Group, Universidad Nacional de San Luis,San Luis, Argentina: http://sel.unsl.edu.ar/licenciatura/ingsoft2/UML-DiagramaClaseObjeto.pdf

Romero, D. M. (2009-2010). Sistemas MultiAgente. *Programa de doctorado Informática Industrial-Departamento de Tecnología Electrónica – Universidad de Sevilla* . Sevilla, España.

Soza Pollman, H. (Septiembre de 2014). *Ingeniare. Revista Chilena de Ingeniería.* Recuperado el 10 de Marzo de 2016, de Evaluando características del agente software: http://www.redalyc.org/articulo.oa?id=77231339014

trahc.mx. (s.f.). *PROGRAMACIÓN ORIENTADA A OBJETOS POO.* Obtenido de TraCH STUDIO total web: http://trahc.mx/programacion-orientada-a-objetos-poo.html

Vega, M. (10 de 2010). *Casos de Uso UML*. Obtenido de Departamento de Lenguaje y Sistemas Informàticos,Universiad de Granada,Granada,España: https://lsi.ugr.es/~mvega/docis/casos%20de%20uso.pdf

Venturini, V. (Mayo de 2012). Sistema Multi-Agente basado en Contexto, Localización y Reputación para dominios de Inteligencia Ambiental. *Tesis Doctoral,Universidad Carlos III de Madrid,Doctorado en Ciencia y Tecnología Informática*. Madrid, España.

www.minitab.com. (s.f.). *Soporte técnico de Minitab 17.* Obtenido de Ejemplo de cómo se realiza una prueba de hipótesis básica: http://support.minitab.com/es-mx/minitab/17/topic-library/basic-statistics-and-graphs/hypothesis-tests/basics/what-is-a-hypothesis-test/