Conciencia Tecnológica
ISSN: 1405-5597
contec@mail.ita.mx
Instituto Tecnológico de Aguascalientes
México

Luna Rosas, Fco. Javier
Diseñando un Servicio de Balanceo de Carga Dinámico con Agentes en Ambientes Distribuidos Orientados a Objetos (CORBA)
Conciencia Tecnológica, núm. 21, enero, 2003, pp. 31-36
Instituto Tecnológico de Aguascalientes
Aguascalientes, México

Diseñando un Servicio de Balanceo de Carga Dinámico con Agentes en Ambientes Distribuidos Orientados a Objetos (CORBA).

Fco. Javier Luna Rosas
fjluna@verona.fi-p.unam.mx

División de Estudios de Posgrado Facultad de Ingeniería (DEPFI-UNAM), Cd. Universitaria, Apdo. Postal 70-256, C.P. 04510, México, D.F. Tels. (525) 5622-30-12 y (49) 9723515, Fax(525) 5616-10-73.
Instituto Tecnológico de Aguascalientes, Av. A. López Mateos 1801 Ote. esq. Av. Tecnológico, Fracc. Ojocaliente, CP. 20256, Aguascalientes, Ags.

Resumen

Balancear Carga significa como distribuir procesos entre procesadores conectados por una red, para equilibrar la carga de trabajo entre ellos [7]. Los algoritmos de planeación distribuida globales pueden ser divididos en dos grandes grupos: algoritmos de Balanceo de *Carga Dinámica y* algoritmos de Balanceo de *Carga estática.* Los algoritmos de Balanceo de *Carga Estática*, también referenciados como planeación de tareas; obtienen la localización de todos sus requerimientos antes de comenzar su ejecución. Los algoritmos de Balanceo de *Carga Dinámica* intentan equilibrar la carga en tiempo de ejecución. Este reporte propone una nueva forma de Balancear la Carga Dinámicamente en ambientes distribuidos orientados a objetos, el artículo se centra sobre las especificaciones de Balanceo de Carga basadas en el *Middleware* utilizando el estándar de *CORBA,* define una arquitectura propia para Balancear la Carga D*inámicamente* dentro de *CORBA*, en la arquitectura implementamos y comparamos varios algoritmos que son bien conocidos y obtenemos sus resultados.

1.- Taxonomía del Balanceo de Carga.

Los mecanismos de Balanceo de Carga distribuyen equitativamente la carga de trabajo de los clientes entre un conjunto de servidores para mejorar el tiempo de respuesta global del sistema. Los mecanismos de Balanceo de Carga generalmente se encuentran en uno de los niveles siguientes[14].

- A Nivel de la Red.
- A Nivel del Sistema Operativo.
- A Nivel Middleware.

Balanceo de Carga Basado en Red: los servidores de nombres de dominio y los ruteadores IP que sirven a una gran cantidad de máquinas host proveen este tipo de Balanceo de Carga. Por ejemplo cuando un cliente resuelve un hostname, el DNS puede asignar una dirección IP a cada requerimiento basado en las condiciones de carga actual, el cliente entonces contacta el servidor designado, sin que el cliente se entere, un servidor diferente puede ser designado en la próxima resolución de nombres. El Balanceo de carga en esta capa es algo limitado, ya que no toman en cuenta el contenido de los requerimientos del cliente.

Balanceo de Carga Basado en Sistemas Operativos. los sistemas operativos distribuidos, proveen este tipo de Balanceo de Carga a través de un conjunto de computadoras, compartición de carga (la compartición de carga no debe ser confundida con el Balanceo de Carga, por ejemplo, el procesamiento de recursos puede ser compartido entre procesadores, pero no necesariamente balanceados) y mecanismos de migración de procesos. El agrupamiento es una forma efectiva de lograr alta disponibilidad y alto desempeño combinando muchas computadoras para mejorar el tiempo de respuesta global. Los procesos pueden entonces ser distribuidos transparentemente entre las computadoras de una agrupación. El agrupamiento generalmente emplea compartición de carga y migración de procesos. Los mecanismos de migración de procesos Balancean la Carga a través de procesadores o más generalmente a través de nodos de red transfiriendo el estado de los procesos entre los nodos.

Balanceo de Carga Basado en Middleware: las interacciones globales son acopladas por las arquitecturas llamadas Middleware. La arquitectura más común de Middleware para aplicaciones orientadas a objetos distribuidas es Common Object Request Broker Architecture (CORBA) [13] y Distributed Component Object Model (DCOM) [6]. El Balanceo de Carga ejecutado en el Middleware a menudo se basa por sesión o por requerimiento.

2. Arquitectura.

CORBA provee soluciones para muchos cambios en sistemas distribuidos, tales como predectibilidad, seguridad, transacciones y tolerancia a fallas, pero

CORBA aún carece de soluciones o mecanismos estándar para intentar Balancear la Carga. Un servicio de balanceo de carga CORBA debe de atar un requerimiento del cliente a una replica cada vez que se realiza una decisión de Balanceo de Carga, respaldando los mecanismos que podemos implementar para realizar Balanceo de Carga en CORBA podemos clasificar el atado de los requerimientos de los clientes de acuerdo a su granulidad de la siguiente forma [14]:

- **Por sesión**. Los requerimientos del cliente continuaran siendo enviados a la misma replica que fue diseñada (en el contexto de CORBA se define como el periodo de tiempo durante el cual el cliente es conectado a un servidor dado, es usualmente definido por el periodo de vida de los clientes.
- **Por requerimiento.** Cada requerimiento del cliente será enviado a una replica diferente, esto es, se realizara un atado a la replica cada vez que el requerimiento es invocado.
- **Por demanda.** Los requerimientos del cliente pueden ser re – enviados a otra replica cuando el Balanceador de Carga lo considera necesario.

Nuestra arquitectura se basa en el atado de requerimientos por demanda. La figura 1 muestra como los clientes reciben una referencia de objetos al Balanceador de Carga, usando el mecanismo estándar de CORBA (LOCATION_FORWARD()), el Balanceador de Carga puede re - enviar el requerimiento del cliente inicial a la replica apropiada del servidor. Los clientes CORBA continuaran utilizando la nueva referencia de objetos obtenida como parte del mensaje LOCATION_FORWARD() para comunicarse con la replica directamente hasta que sean nuevamente re - enviados o terminen su ejecución.

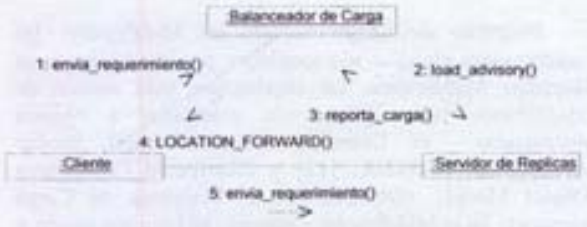

Figura 1. Arquitectura Adaptativa.

Por otra parte Russell, considera como agente inteligente, a todo agente que *percibe* su ambiente mediante sensores y que responde o actúa en tal ambiente por medio de *efectores (acciones)* [16]. Lo primero que tiene que hacer un agente de software para tomar una acción inteligente es percibir que esta a su alrededor, para tener una idea del estado del mundo que lo rodea, una vez que nuestro agente inteligente a percibido que eventos han ocurrido el próximo paso es tomar una acción. Un agente de software exhibe *inteligencia* si este actúa racionalmente. Este utiliza conocimiento información y razonamiento para tomar acciones razonables en busca de metas. Bigus dice que hay tres dimensiones que debemos de tomar en cuenta para medir las capacidades de un agente de software [1].

- Operación (agency). Es el grado de autonomía que el agente de software tiene, en representación de los usuarios, aplicaciones, sistemas de computadoras o de otros agentes inteligentes.
- Inteligencia. Se refiere a la habilidad de un agente para capturar y aplicar conocimiento de dominio específico y procesar la solución de problemas.
- Movilidad. Un agente es móvil si se puede mover entre sistemas en una red.

La figura 2 representa la estructura de un agente inteligente sencillo en forma esquematizada. En ella se pueden observar cómo las reglas de condición acción permiten al agente establecer la conexión entre percepciones y acciones.

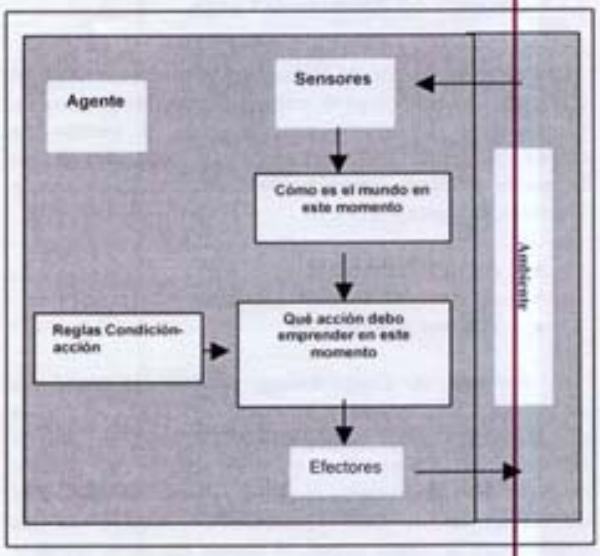

Figura 2. Diagrama Esquematizado de un Agente Inteligente Simple.

En general podemos decir que la arquitectura del Balanceador de Carga esta compuesta de los siguientes agentes:

1) Localizador de Replicas. Este agente identifica que replica recibirá que requerimiento. El Localizador de Replicas también conecta los clientes a las replicas identificadas. El Localizador de Replicas es implementado

usando el mecanismo estándar de CORBA llamado Portable Object Adapter (POA) [13] tal como servant locators implementa el patrón interceptor. El Localizador de Replica envía cada requerimiento que recibe a la replica que el Analizador de Carga selecciona.

2) Monitor de Carga. Este agente monitorea la carga sobre una replica dada, reporta la carga de la replica al Balanceador de Carga, responde mensajes que el Balanceador de Carga envía e informa a las replicas cuando ellas deben aceptar requerimientos vs. re-enviarlos de regreso al Balanceador de Carga.

3) Analizador de Replicas. Este agente decide que replica recibirá el próximo requerimiento del cliente. El Localizador de Replicas obtiene una referencia a una replica del Analizador de Carga y entonces envía el requerimiento a esa replica. El Analizador de Carga también permite seleccionar una estrategia de Balanceo de Carga en tiempo de corrida, tratando de mantenerse siempre simple y flexible.

4) Balanceador de Carga. Es un medidor de carga que integra los agentes que describimos anteriormente.

3.- Análisis, Diseño y Construcción del Balanceador de Carga.

En esta sección se hace el análisis, diseño y construcción del Balanceador de Carga. El desarrollo se apoya en el Proceso de Desarrollo de Software del Modelo Unificado [2], en el Lenguaje de Modelado Visual UML [2,15], en el Lenguaje de Programación Orientado a Objetos JAVA [9, 11] y en la Tecnología Middleware CORBA que permite la comunicación de objetos en sistemas distribuidos [12,13, 18].

La figura 3 ilustran tres agentes cooperativos que integran nuestro Balanceador de Carga. El Balanceador de Carga debe asegurar que la carga a través de las replicas este balanceada, esos pasos incluyen forzar a la replica para re - enviar el cliente de regreso al Balanceador de Carga cuando la carga es alta, y obligar a la replica para que acepte el requerimiento cuando la carga es nominal.

El Monitor de Carga implementa una estrategia para monitorear la carga sobre un recurso dado. La interfaz reporta la carga al Balanceador de Carga, o bien el Balanceador de Carga obtiene la carga del Monitor de Carga.

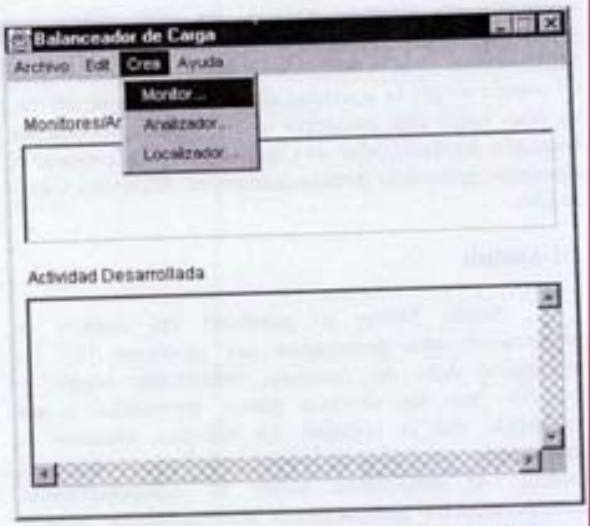

Figura 3. Interfaz Gráfica del Balanceador de Carga.

La figura 4 ilustra la interfaz gráfica del Monitor de Carga, este actúa como mediador entre el Balanceador de Carga y una replica dada. Este patrón asegura que haya un débil acoplamiento entre el servidor de carga y la replica del servidor. Con la capacidad de mediador, el Monitor de Carga responde a los requerimientos de Balanceo de Carga que le envía el Balanceador de Carga, dependiendo del tipo de requerimientos, la replica continuará aceptando estos requerimientos o bien los retornara de regreso al Balanceador de Carga. El Balanceador de Carga pregunta al Monitor de Carga por la carga de la replica actual mediante un mensaje load_advisory que coloca el estado de la replica a uno de los siguientes valores: nominal o alta.

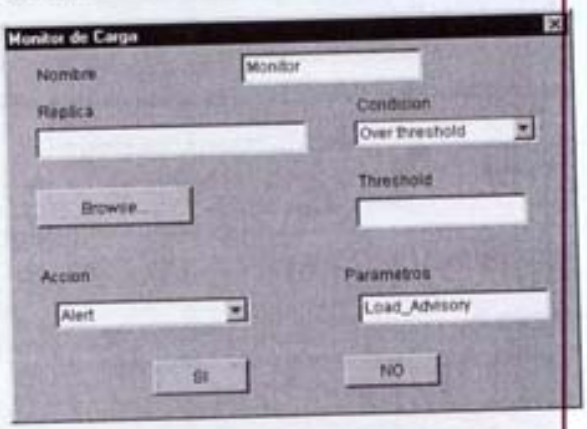

Figura 4. Monitor de Carga.

La estrategia de selección de la Replica será realizada mediante el Analizador de Carga, esto es el Analizador de Carga implementará diferentes Casos de Uso de Balanceo de Carga sin manejar solamente uno solo. Hay veces en que es necesario Balancear la Carga

solamente en algunas replicas, en tales casos una estrategia simple de Balanceo de Carga debe ser suficiente, en otras ocasiones se manejan periodos largos, tal como recoger la actividad durante la carga de un dia, en tales casos una estrategia más compleja debería ser necesaria. El Analizador de Carga deberá implementar el algoritmo apropiado dinámicamente en diferentes Casos de Uso.

3.1 Análisis.

Según Mellor el propósito del análisis es proporcionar una descripción del problema [2]. La descripción debe ser concreta, consistente, legible y revisable, por las diversas partes interesadas y ser comparada con la realidad. En nuestros términos es proporcionar un modelo de la forma en que se comporta el sistema. Al enfocarnos sobre tal comportamiento identificamos los puntos claves que representan alguna actividad primaria en el sistema o con frecuencia denotan el mapeo de entradas y salidas que representan las transformaciones que el sistema hace en su ambiente.

Para efectuar el análisis, Booch[2], establece que el análisis de requerimientos, y el análisis del dominio del sistema, son dos pasos que deben ser liberados durante esta etapa. Por una parte el análisis de requerimientos del sistema debe proporcionar la escritura básica de las funciones que el sistema deberá realizar y por su parte el análisis del dominio del sistema proporciona las estructuras lógicas que son claves para el sistema. En nuestro ejemplo especificamos el análisis de requerimientos utilizando un modelo de Casos de Uso, los Casos de Uso nos representan la funcionalidad que debe tener el sistema [2]. La Figura 5 muestra un caso de uso que permite el registro de requerimientos en el Balanceador de Carga.

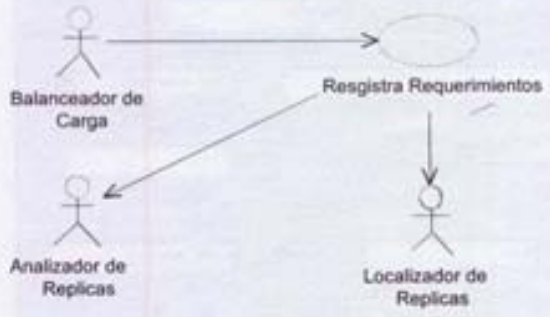

Figura 5. Registro de Requerimientos.

Por otra parte el análisis del dominio lo realizamos utilizando diagramas de estructura estática. Un diagrama de estructura estática muestra el conjunto de clases y objetos que son parte de un sistema, junto con sus relaciones existentes entre estas clases y objetos [3].

2.2 Diseño.

El propósito del diseño es crear una arquitectura para implementar el comportamiento que requiere el modelo del análisis [2]. Para tal propósito creamos nuestros escenarios, un escenario es usado para describir como los casos de uso se realizan mediante interacciones entre sociedades de objetos, para capturar los escenarios utilizamos dos tipos de diagramas de interacción: diagramas de secuencia que describen las interacciones de los objetos a través del tiempo y los diagramas colaborativos que muestran la interacción de objetos organizada a través de objetos y sus relaciones. La Figura 6 muestra el diagrama de secuencia para controlar y ejecutar los requerimientos en el Balanceador de Carga:

1) Un cliente obtiene una referencia de objeto a la replica e invoca una operación. El cliente transparentemente invoca el requerimiento sobre el Balanceador de Carga.
2) Después de recibir el requerimiento del cliente, el Balanceador de Carga despacha el requerimiento al componente Localizador de Replicas.
3) El componente Localizador de Replicas pregunta al Analizador de Carga por una replica del servidor apropiada.
4) El Localizador de Replicas transparentemente cambia el requerimiento del cliente a la replica elegida.
5) El cliente entonces continuara enviando requerimientos directamente a la replica seleccionada hasta que el Balanceador de Carga detecte altas cargas sobre las replicas.
6) El Balanceador de Carga monitorea la carga de las replicas. Dependiendo de la politica de carga reportada el Monitor de Carga reportara la carga al Balanceador de Carga, o bien el Balanceador de Carga preguntara al Monitor de Carga por la carga de las replicas.
7) Así como el Balanceador de Carga colecciona la carga, el Analizador de Carga analiza la carga de la replica.
8) Si una replica llega a estar sobre-cargada el Balanceador de Carga puede dinámicamente re – enviar al cliente a otra replica menos cargada.
9) El monitor de carga diseña un mensaje de control para la replica. Dependiendo del contenido del mensaje diseñado por el Balanceador de Carga es mensaje de control causara que la replica acepte o re – envíe el requerimiento.
10) En este punto el Balanceador de Carga se cicla y comienza nuevamente.

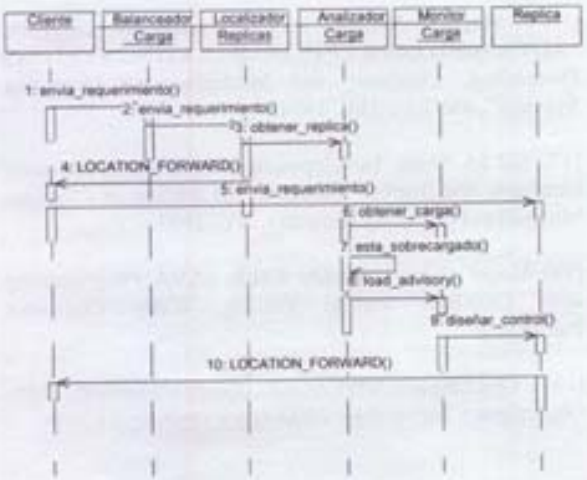

Figura 6. Control de Requerimientos.

4. Resultados.

Para llevar a cabo el balanceo de carga dinámico, nos basamos en el algoritmo de Bryant and Finkel (también conocido como el algoritmo de paridad), el algoritmo de Barak y Shiloh (también llamado algoritmo del vector) y el algoritmo de Stankovic y Sidhu (también llamado algoritmo de ofrecimiento y aceptación) [17]. Evaluamos los algoritmos en una red LAN (LAN-Base 100) con cinco máquinas simétricas (Pentium II, 350Mhz y 32 Mb en RAM), se hicieron seis evaluaciones cada evaluación procesa 1000 objetos, que son balanceados dinámicamente en nuestra arquitectura con los algoritmos previamente mencionados, los objetos tienen un periodo de vida (LifeTime) de 1.4 seg. Además cada evaluación mantiene diferentes características tal como se describen a continuación:

- *Evaluación#1*: 5 replicas sin concurrencia de objetos.
- *Evaluación#2*: 5 replicas con 1 replica concurrente, ejecutando objetos de 1.0, 1.5 y 2.0seg, más el LifeTime de los objetos principales.
- *Evaluación#3*: 5 replicas con 2 replicas concurrentes, ejecutando objetos de 1.0, 1.5 y 2.0seg, más el LifeTime de los objetos principales.
- *Evaluación#4*: 5 replicas con 3 replicas concurrentes, ejecutando objetos de 1.0, 1.5 y 2.0seg, más el LifeTime de los objetos principales.
- *Evaluación#5*: 5 replicas con 4 replicas concurrentes, ejecutando objetos de 1.0, 1.5 y 2.0seg, más el LifeTime de los objetos principales.
- *Evaluación#6*: 5 replicas con 5 replicas concurrentes, ejecutando objetos de 1.0, 1.5 y 2.0seg, más el LifeTime de los objetos principales.

Una vez concluidas las evaluaciones, se sacaron los promedios de cada evaluación, con el propósito de mantener un parámetro de tiempo de cada algoritmo (Ver Figura 7 y Tabla 1.).

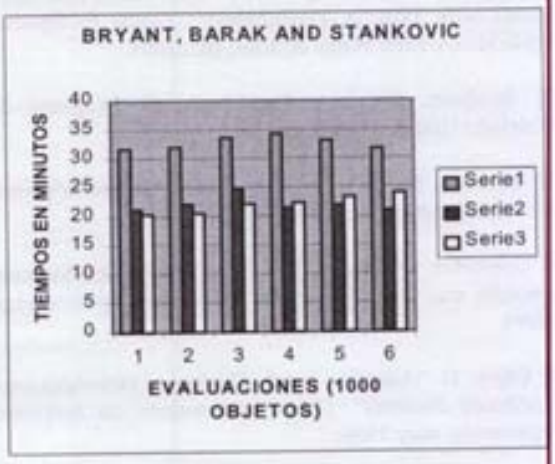

Figura 8. Promedios de las Evaluaciones.

Evaluaciones (1000 Objetos)			
#	Bryant	Barak	Stankovic
1	31.69	21.2653	20.3606
2	32.0169	22.1003667	20.5428667
3	33.5415333	24.7937667	22.1045767
4	34.2260733	21.4683667	22.3698
5	33.0727033	21.8970467	23.2709
6	31.7052833	20.9701	23.9585833

Tabla1. Tiempos Promedio de las Evaluaciones.

5.- Conclusión.

El uso combinado de *Tecnologías Orientadas a Objetos, Tecnologías Distribuidas y la Tecnología Web*(OT, DT y WT) cambia la forma en que los sistemas son diseñados. Los objetos han sido agregados a las redes e integrados con Middleware, a menudo llamados *Object Request Broker (ORB's)*. Los objetos han sido usados en sistemas distribuidos para representar unidades de distribución, movimiento y comunicación. La adición de WT proporciona portabilidad en la aplicación. Existe en efecto un nuevo paradigma de computación, en el cual inter-operan los objetos este nuevo paradigma es la *Tecnología de Objetos Distribuida (DOT)*. La DOT nos da la oportunidad de movernos rápido y hacia el futuro en la distribución y globalización de las aplicaciones. En nuestra arquitectura de Balanceo de Carga la DOT nos da la oportunidad de mantener *portabilidad, interoperabilidad y transparencia* entre plataformas.

Bibliografía.

[1] Bigus P. Joseph and jennifer, Constructing Intelligent Agents with Java. A Programmer´s Guide to Smarter Applications. John Wiley & Sons, Inc. 2001.

[2] Jacobson, Booch y Rumbaugh. El Lenguaje de Modelado Unificado. Addsin Wesley 1999.

[3] Jacobson, Booch y Rumbaugh. The Unified Software Development Process. Addsin Wesley 1999.

[4] Coulouris, Dollimore, Kindberg. Distributed Systems Concepts and Design, Second Edition. Addison-wesley (1996).

[5] Eager, D. "Adaptive Load Sharing in Homogeneous Distributed Systems". IEEE Transactions on Software Engineering, may 1986.

[6] G. Eddon and H. Eddon. Inside distributed COM. Microsoft Press, 1998.

[7] Goscinski, Andrzej., Distributed Operating Systems. The Logical Design. Addison-Wesley 1992.

[8] Hisao Kameda, Jie Li, Chonggun Kim and Yongbing Zhang, Optimal Load Balancing in Distributed Computer System. Springer-Verlag, London 1997.

[9] Jaworski Jamie. Java 1.2 Al Descubierto. Primera Edición Prentice Hall, SAMS.

[10] Krueger, P., and R. Finkel, "An Adaptive Load Balancing Algorithm for a Multicomputer", Technical Report 539, University of Wisconsin-Madison, Apr. 1984.

[11] Lea, Doug. Concurrent Programming in Java, Design Principles and Patterns. Addison-Wesley 1997.

[12] Orfali Robert, Harkey Dan. Client/Server Programming with JAVA and CORBA. Second Edition, Wiley Computer Publishing.

[13] OMG (Object Management Group). The Common Object Request Broker: Architecture and Specification. Technical Report, http:// www.omg.org(2001).

[14] Ossama Othman, O´Ryan Carlos and Schmidt. Strategies for CORBA Middleware-Based Load Balancing. IEEE Distributed Systems Online http://computer.org/dsonline/0103/features/oth103_1.html

[15] Quatrani, Terry. Visual Modeling with Rational Rose and UML. Addison-Wesley 1998.

[Russell96] Russell Stuart, Norving Peter. Inteligencia Artificial un Enfoque Moderno. Prentice may 1996.

[16] Singhal, Mukesh and Shivaratri G. Niranjan, "ADVANCED CONCEPTS IN OPERATING SYSTEMS Distributed, Database, and Multiprocessor Operating Systems", McGraw-Hill, 1994.

[17] Sol M. Shatz. Development of Distributed Software Concepts and Tools. University of Illinois at Chicago. Macmillan Publishing company, NY. 1993.

[18] Vogel andreas, Duddy Keith. JAVA Programming with CORBA. Second Edition, Wiley Computer Publishing.

[19] ObjectSpace VOYAGER (Core Technology 2.0). ObjectSpace Technology 2000. www.objectspace.com.