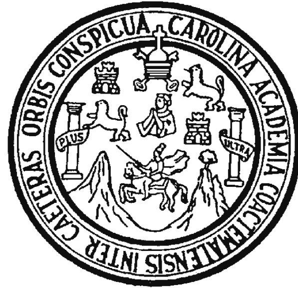
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB

Carlos Marcelo Santucci Marroquín
Asesorado por: Ing. Manuel Fernando López Fernández

GUATEMALA, OCTUBRE DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA

FACULTAD DE INGENIERÍA

POR

CARLOS MARCELO SANTUCCI MARROQUÍN

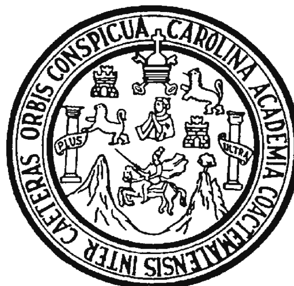
ASESORADO POR: ING. MANUEL FERNANDO LÓPEZ FERNÁNDEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympo Paiz Recinos
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADOR	Ing. Marlon Antonio Pérez Turk
EXAMINADOR	Ing. Édgar René Ornelyz Hoil
EXAMINADORA	Inga. Virginia Victoria Tala Ayerdi
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB

Tema que me fuera asignado por la coordinación de la carrera de Ingeniería en Ciencias y Sistemas, con fecha 10 de enero de 2003.

Carlos Marcelo Santucci Marroquín

Guatemala, 12 de agosto 2004

Ingeniero
Carlos Azurdia
Coordinador de Privados y Revisión de Tesis
Escuela de Ciencias y Sistemas

Estimado Ingeniero:

Por medio de la presente, me permito informarle que he asesorado el trabajo de graduación titulado: **ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB**, elaborado por el estudiante Carlos Marcelo Santucci Marroquín, y a mi juicio el mismo cumple con los objetivos propuestos para su desarrollo.

Agradeciéndole de antemano la atención que le preste a la presente, me suscribo de usted.

Atentamente,

Ing. Manuel Fernando López Fernández
Asesor

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Guatemala, 25 de agosto de 2004

Ingeniero
Luis Alberto Vettorazzi España
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Vettorazzi:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **Carlos Marcelo Santucci Marroquín,** titulado: **ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB**, y que a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

El Director de la Carrera de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor de tesis y del Licenciado en Letras, al trabajo de graduación titulado **ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB**, presentado por el estudiante **Carlos Marcelo Santucci Marroquín**, aprueba el presente trabajo y solicita la autorización del mismo.

ID Y ENSEÑAD A TODOS

Ing. Luis Alberto Vettorazzi España
Director
Ingeniería en Ciencias y Sistemas

Guatemala, 25 de septiembre de 2004

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **ESTÁNDARES CORBA Y SU USO EN LA CONSTRUCCIÓN DE APLICACIONES WEB**, presentado por el estudiante universitario **Carlos Marcelo Santucci Marroquín**, procede a la autorización para la impresión del mismo.

IMPRÍMASE:

Ing. Sydney Alexander Samuels
DECANO

Guatemala, 25 de septiembre de 2004

AGRADECIMIENTOS

A Dios

Por haberme dado la vida y la oportunidad de hacer lo que deseo.

A mis padres

José Fernando Santucci Lara y Gladys Fabiola de Santucci, por su amor, guía, ejemplo, comprensión y apoyo incondicional.

A Lucía Fernanda Gómez Solórzano

Por todo su amor, apoyo y comprensión.

A mis abuelos

Por toda su comprensión, apoyo y ejemplo.

A mi tío Carlos Fernando Marroquín

Por su ayuda durante los inicios de mi carrera.

A mis amigos y compañeros

En especial a Carlos, Omar, Otto, Arnoldo, José Luis, Daniel y Sergio.

Al Ing. Manuel Fernando López Fernández

Por su confianza y apoyo profesional para llevar a buen termino este documento.

A las empresas ICON y EEGSA

Por permitir mi desarrollo profesional, en especial al Lic. Hugo Cruz y al Ing. Guillermo Castañeda.

Y a la Universidad de San Carlos de Guatemala.

DEDICATORIA

A mis padres

Por haberme brindado su apoyo incondicional y permitirme tener mis propias metas y sueños.

A mis abuelos

Por su cariño y por darme la razón de superación en esta vida.

A mis hermanos

Giovanni y Guisseli, por su apoyo en todo momento.

A mi familia

Por su apoyo y confianza.

A mis compañeros

Por permitirme acompañarlos en el camino para construir una carrera profesional y siempre brindarme su apoyo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES VI
GLOSARIO VIII
RESUMEN XI
OBJETIVOS XII
INTRODUCCIÓN XIII

1. COMPUTACIÓN DISTRIBUIDA 1
1.1. Tendencias actuales de desarrollo 1
1.2. Aplicaciones distribuidas 2
1.2.1. Información distribuida 3
1.2.2. Computación distribuida 3
1.2.3. Usuarios distribuidos 3
1.2.4. Realidades fundamentales de las aplicaciones distribuidas 4
1.3. Grupo de administración de objetos (OMG) 6
1.3.1. Objetivo 6
1.3.2. Estructura 7
1.4. Aproximación estratégica para integrar la cadena de valor 8
1.4.1. El modelo *bussiness-to-customer* de comercio electrónico 10
1.4.2. *Business-to-Business* 11
1.4.3. Integración de la cadena de valor 12
1.4.4. Proceso de integración de negocios 13
1.4.5. Inicio de la integración de negocio 13
1.5. CORBA 14
1.6. Partes de CORBA 18

1.6.1. Interfaz dinámica de tiempo de ejecución 19
1.6.2. El proceso 20
1.6.3. Servicios de objetos 22
1.6.4. IDL 23
1.6.5. Interoperatividad de CORBA 23
1.7. Componentes del modelo CORBA 24
1.7.1. Cliente 25
1.7.2. Objeto 25
1.7.3. Sirviente 25
1.7.4. Núcleo ORB 26
1.7.5. Interfaz ORB 26
1.7.6. OMG IDL Stubs 27
1.7.7. Compilador IDL 27
1.7.8. Interfaz de invocación dinámica (DII) 27
1.7.9. Interfaz dinámica de *Skeleton* (DSI) 28
1.7.10. Adaptador de objetos 28
1.7.11. Interfaz de repositorio 28
1.7.12. Repositorio de implementación 29
1.7.13. Servicios CORBA 29
1.8. OMA (modelo de objetos) 30
1.8.1. Semántica de objetos 32
1.8.2. Peticiones 33
1.8.3. Creación y destrucción de objetos 35
1.8.4. Tipos 35
1.8.4.1. Tipos básicos 35
1.8.4.2. Tipos construidos 37
1.8.5. Interfaces 37
1.8.6. Tipos de valor 38
1.8.7. Interfaces abstractas 38

1.8.7.1. Operaciones 39
1.8.7.2. Parámetros 41
1.8.7.3. Excepciones 41
1.8.7.4. Contextos 41
1.8.7.5. Semántica de ejecución 42
1.8.7.6. Atributos 42
1.8.7.7. Implementación de objeto 43
1.8.7.8. Modelo de ejecución 43
1.8.7.9. Modelo de construcción 44
2. COMPONENTES DE *SOFTWARE* 47
2.1. Reutilización de componentes 49
2.2. Distribución de componentes 51
2.3. Desarrollo de componentes en Java 53
2.4. Consideraciones de construcción de aplicaciones del lado del servidor 55
2.5. Arquitectura de dos capas 57
2.6. Características de una arquitectura de dos capas 58
2.7. Arquitecturas de N capas 60
2.8. Características de las arquitecturas de N capas 61
2.9. Soluciones de arquitectura del lado del servidor 62
2.10. Plataforma *Sun Microsystem's* Java 2 63
3. PLATAFORMA J2EE 65
3.1. Tecnologías J2EE 65
3.2. *Enterprise JavaBeans* 67
3.3. Método de invocación remota 67
3.4. Nombramiento Java e interfaz de directorio 68
3.5. Conectividad de base de datos Java 69
3.6. API de transacción Java y servicio de transacción Java 69
3.7. Servicio de mensajes Java 70

3.8. *Servlets* Java y páginas de servidor Java 71
3.9. *Mail* Java 71
4. *ENTERPRISE BEANS* 73
4.1. *Beans* de sesión 73
4.2. *Beans* de sesión con estado (*Stateful Session Beans*) 74
4.3. *Beans* de sesión sin estado (*Stateless Session Beans*) 75
4.3.1. Elegir entre un *bean* con estado o uno sin estado 75
4.4. *Beans* de entidad (*Entity Beans*) 77
4.5. *Beans* de entidad persistentes administrados por *bean* 80
4.6. *Beans* de entidad persistentes administrados por el contenedor 80
4.7. Clase *Enterprise Bean* 81
4.8. Objeto EJB 82
4.9. Interfaz remota 84
4.10. RMI Java 85
4.11. Objeto *home* 87
4.12. Interfaz *home* 88
4.13. Descriptores de instalación 88
5. RESPONSABILIDADES DE UN SERVIDOR Y UN CONTENEDOR EJB 91
5.1. Manejo de recursos y ciclo de vida 92
5.2. Administración del estado 93
5.3. Transacciones 94
5.4. Seguridad 95
5.5. Persistencia 96
5.5.1. Serialización de objetos Java 96
5.5.2. Mapeo relacional de objetos 97
5.5.3. Bases de datos de objetos 98
5.6. Acceso remoto y transparencia de ubicación 99
5.7. Herramientas de instalación 100

5.8. Características especiales 101
6. APLICACIÓN 103
6.1. Elección de un servidor 103
6.1.1. *Websphere* 105
6.1.2. *Jboss* 105
6.1.3. *Weblogic* 106
6.1.4. 9iAS (OC4J) 107
6.2. Servidor elegido 108
6.3. Funcionamiento del servidor 110
6.4. Servidores lógicos y puertos de escucha 112
6.5. Archivos de configuración XML 114
6.5.1. Perfiles EJB y descriptores de instalación 115
6.5.2. Descriptores de aplicación *Web* 117
6.6. Descripción de la aplicación 122
6.7. Arquitectura de la aplicación 124
6.8. Descripción de las entidades del diagrama de clases 126
6.9. Convenciones de nombramiento 129
6.10. Elementos que componen el código de la aplicación 129
6.11. Descriptores de instalación 131
6.12. Funcionamiento de la aplicación 133
CONCLUSIONES 141
RECOMENDACIONES 143
BIBLIOGRAFÍA 144
APÉNDICE A 146

ÍNDICE DE ILUSTRACIONES

FIGURAS

1	Componentes del modelo CORBA	24
2	Esquema de funcionamiento de un *bean* de entidad	79
3	Diagrama de relación entre objetos EJB al interactuar con un cliente	84
4	Diagrama de funcionamiento de un contenedor EJB	92
5	Especificaciones soportadas por el servidor OC4J	110
6	Archivos de configuración del servidor OC4J	112
7	Archivos XML de instalación en J2EE	112
8	Esquema general de la aplicación	126
9	Esquema general de la creación de un EJB y su interacción con el cliente	127
10	Diagrama de clases de la aplicación de edición de páginas de cursos	129
11	Diagrama de bloques de la aplicación	131
12	Contenido del descriptor de instalación *web.xml*	132
13	Contenido del archivo de configuración de instalación *Ejb-jar.xml*	133
14	Contenido del archivo de configuración *application.xml*	134
15	Página de inicio de la aplicación	135
16	Página del listado de cursos	136
17	Página que muestra los datos de un curso y elección de operación	137
18	Página de modificación de los datos de un curso	137

19 Página de edición de páginas 138
20 Página para modificar los datos de una página 139
21 Página para modificar los datos de los títulos de una página 139
22 Página de modificación de párrafos de título pertenecientes a una página 140
23 Página que lista las FAQ asociadas a un curso 141
24 Página de edición de FAQ 141

TABLAS

I Métodos de la interfaz remota *javax.ejb.EJBObject* 85
II Características de configuración y sus respectivos archivos asociados 113
III Lista de elementos del descriptor de instalación *ejb-jar* 147

GLOSARIO

Buffer — Memoria intermedia para el almacenamiento de datos temporales en la comunicación entre un ordenador y un dispositivo externo.

Ciclo de vida — Es el proceso que se sigue para construir, entregar y hacer evolucionar el *software*, desde la concepción de una idea hasta la entrega del sistema.

Componente — Artefacto de *software* claramente identificable que ejecuta funciones específicas y posee una interfaz clara a través de la cual se puede integrar a otros sistemas.

Escalabilidad — Capacidad de un sistema de incrementar sus prestaciones en función del número de usuarios simultáneos que lo utilizan.

Firewall — Dispositivo que funciona denegando o aceptando cualquier tráfico que se produzca cerrando todos los puertos de comunicación de un ordenador.

Herencia Característica mediante la cual un componente puede utilizar los métodos heredados de otro objeto.

Interfaz Componente del *software* que permite que un usuario, componente o proceso, interactúe con él.

Mapeo Traducción de peticiones de conexión externas a servidores, procedimientos, funciones o máquinas internas.

Polimorfismo Capacidad de un objeto para tener métodos con el mismo nombre y diferente implementación.

Repositorio Componente que alberga a todos aquellos componentes que el *software* utiliza y almacena incluso después de ejecutarse, haciendo que los componentes sean persistentes en el tiempo.

Rutear Direccionamiento de la comunicación hacia el destino necesitado.

Serializar	Conversión de una estructura de datos a una serie de *bytes* susceptible de ser almacenada.
Sincrono	Comunicación en la cual el emisor se bloquea hasta que recibe respuesta.
Socket	Forma de comunicación con otros programas usando descriptores de fichero estándar.
Terminal tonta	Dispositivo que tiene entradas y salidas, el cual no posee capacidad de procesamiento

RESUMEN

El presente trabajo de graduación es de tipo investigativo, en el cual se incluirán las características principales del estándar EJB para el desarrollo de aplicaciones para internet, ya que con el cambio de los hábitos de compra, venta y comunicación de las personas en general y su creciente necesidad de información, han empezado a esperar que los negocios e instituciones les brinden un mayor poder de decisión al momento de realizar sus interacciones con ellas. Para que esto sea posible se debe contar con la tecnología necesaria para que esta interacción pueda realizarse.

El hacer accesible la tecnología implica, en muchos casos, la necesidad de comunicar sistemas heterogéneos con independencia de su ubicación física, por lo que es conveniente elegir alguna arquitectura de desarrollo de aplicaciones que se encuentren distribuidas por capas, de forma que el desarrollo y administración de las aplicaciones sea realizado sencillamente.

La aproximación tecnológica empleada en este trabajo es la del desarrollo de aplicaciones por medio de componentes, los cuales implementan un conjunto de interfaces definidas para realizar algún tipo de operaciones.

Este trabajo concluye con una breve comparación de cuatro servidores distintos que soportan EJB y la elección de uno de ellos para el desarrollo de una aplicación para internet, dividida en tres capas para administración y publicación de información general de cursos impartidos en alguna carrera que emplee tecnología EJB.

OBJETIVOS

General

Presentar el estándar CORBA y el estándar implementado EJB como alternativas para poder llevar a cabo el desarrollo de aplicaciones en ambiente *Web*, que faciliten el acceso e intercambio de información de forma distribuida.

Específicos

1. Mostrar las arquitecturas básicas utilizadas para el desarrollo de aplicaciones de forma distribuida.
2. Determinar las necesidades que los desarrolladores comúnmente tienen cuando desarrollan e implementan componentes en ambientes heterogéneos.
3. Conocer algunas soluciones de arquitectura que existen del lado del servidor.
4. Descripción de una arquitectura de componentes que permite el desarrollo de aplicaciones en la capa intermedia.
5. Conocimiento del servidor de aplicación EJB para el desarrollo e implementación de una aplicación ejemplo.

INTRODUCCIÓN

En este documento se muestra el estándar CORBA como una alternativa para el desarrollo y publicación de aplicaciones, en ambientes heterogéneos distribuidos, para el intercambio de información. El tratamiento de este tema se inicia con la descripción del Grupo de Administración de Objetos (OMG), cuya misión es la de desarrollar, adoptar, y promover estándares para el desarrollo y publicación de aplicaciones en ambientes heterogéneos distribuidos.

Luego se describe la arquitectura de administración de objetos (OMA), enfocándose en uno de sus componentes principales, la arquitectura de distribución de objetos comunes (CORBA), su núcleo y el lenguaje de definición de interfaz (OMG IDL), entre otros.

Posteriormente se procede a la descripción de las necesidades de arquitectura de un componente del lado del servidor, así como ciertas soluciones que existen desarrolladas cumpliendo con los estándares del OMG. Así también se presenta el estándar J2EE desarrollado por *Sun Micro Systems*, el cual toma la mayoría de las características del estándar CORBA, y llevando a cabo la definición de este nuevo estándar orientado al uso de tecnologías Java, sin perder compatibilidad con el estándar CORBA, cumpliendo con el protocolo de comunicación IIOP. Por último, se presenta una pequeña aplicación desarrollada con un producto que cumple con el estándar J2EE, presentando los pasos, configuraciones, así como una serie de consideraciones necesarias para desarrollar una aplicación que cumple con el estándar J2EE.

1. COMPUTACIÓN DISTRIBUIDA

1.1. Tendencias actuales de desarrollo

La computación de objetos distribuidos es la base de la siguiente generación de *software* de comunicación, la cual gira en torno a los *Object Request Brokers* (ORBs), los cuales automatizan muchas actividades de programación tediosas.

Cuatro tendencias le están dando forma al desarrollo comercial del *software*. La primera es que la industria del *software* se está alejando cada vez más de programar aplicaciones desde cero, y en cambio ahora se busca integrar aplicaciones por medio del uso de componentes reutilizables.

La segunda tendencia es que actualmente existe una gran demanda de aplicaciones distribuidas que proveen invocación de métodos de forma remota y/o una capa intermedia que simplifique la colaboración entre aplicaciones.

Y la tercera tendencia es que existen esfuerzos cada vez mayores para definir un marco estándar de infraestructura de *software*, que permitan a las aplicaciones interactuar a través de ambientes heterogéneos. Finalmente, la siguiente generación de aplicaciones tales como teleconferencias, vídeo por demanda, etc., requieren garantías de calidad de servicio (QoS) para latencia, ancho de banda y confiabilidad.

Una tecnología que soporta estas tendencias es la computación de objetos distribuidos de capa intermedia (DOC). La capa intermedia DOC facilita la colaboración de componentes de aplicación en ambientes heterogéneos y distribuidos.

La meta de la capa DOC es eliminar muchos de los aspectos tediosos, susceptibles de error y no portátiles que se presentan al desarrollar y actualizar servicios y aplicaciones distribuidas. En particular, la capa DOC automatiza muchas tareas comunes de programación de red, tales como ubicación de objetos, implementación de arranque (por ejemplo, activación de servidores y objetos), diferencias de tipos y tamaños entre una variedad de arquitecturas, recuperación de errores y seguridad.

1.2. Aplicaciones distribuidas

Los productos CORBA proveen un marco de trabajo para el desarrollo y ejecución de aplicaciones distribuidas. Aunque la distribución agrega un nuevo conjunto de dificultades, existen ocasiones en que no hay otra opción que desarrollar aplicaciones de forma distribuida, algunas por naturaleza, a través de múltiples computadoras debido a una o más de las siguientes razones:

- La información que utiliza la aplicación está distribuida.
- La computación es distribuida.
- Los usuarios de la aplicación están distribuidos.

1.2.1. Información distribuida

Algunas aplicaciones deben ejecutarse sobre múltiples computadoras debido a que la información a la que la aplicación debe acceder se encuentra distribuida entre las mismas, ya sea por razones administrativas o de propiedad.

En general, la información puede que se encuentre distribuida debido a que el propietario sólo permite que la información sea accedida de forma remota y que no sea almacenada localmente, o quizás la información no pueda estar en la misma ubicación y deba existir en múltiples sistemas heterogéneos.

1.2.2. Computación distribuida

Algunas aplicaciones necesitan correr en múltiples computadoras para aprovechar la ventaja en volumen de procesamiento que brindan varios procesadores trabajando en paralelo para resolver algún problema. Otras aplicaciones pueden ser ejecutadas en múltiples computadoras para aprovechar alguna característica especial de un sistema. En general, las aplicaciones pueden aprovechar la escalabilidad y heterogeneidad.

1.2.3. Usuarios distribuidos

Algunas aplicaciones se ejecutan en múltiples computadoras debido a que los usuarios de la aplicación se comunican e interactúan unos con otros por medio de la aplicación en lugares distintos sean estos remotos o locales.

De esta forma, cada usuario puede ejecutar una parte de la aplicación distribuida en su computadora pudiendo compartir objetos, los cuales puede que se estén ejecutando en su propia computadora o en uno o más servidores en ubicaciones remotas.

1.2.4. Realidades fundamentales de las aplicaciones distribuidas

Los desarrolladores de aplicaciones distribuidas deben enfrentarse a un número de problemas que pueden ser tratados con indiferencia en un ambiente local, donde toda la lógica se ejecuta en el mismo proceso de sistema operativo.

Existen algunas diferencias básicas entre los objetos que se encuentran en un mismo proceso y los objetos que interactúan a través de procesos y máquinas:

Comunicación: la comunicación entre objetos en el mismo proceso es más rápida en el orden de magnitudes que la comunicación entre objetos en diferentes máquinas. Una consideración importante es que se debe evitar diseñar aplicaciones distribuidas en la cuales dos o más objetos tienen interacciones muy estrechas, ya que si este es el caso lo mejor es que se encuentren situados en el mismo lugar.

Fallas en sistemas locales: los objetos fallan juntos mientras que en un ambiente distribuido los objetos fallan separadamente y la red puede causar particiones agregando problemas de sincronización en la comunicación. Cuando dos objetos se encuentran en el mismo lugar, fallan juntos; si el proceso en el cual se encuentran ejecutándose falla, entonces ambos fallan.

Pero si dos objetos se encuentran distribuidos, éstos pueden fallar independientemente. En este caso, el diseñador de los objetos debe preocuparse del comportamiento de cada uno de los objetos en el caso de que alguno de ellos falle. Similarmente, en un sistema distribuido, la red puede separar a los objetos pudiendo entonces correr independientemente asumiendo que el otro ha fallado.

Acceso concurrente: en un ambiente distribuido existe el acceso concurrente, mientras que en un ambiente local el acceso normalmente es concurrente cuando existen varios hilos de ejecución. El modo por defecto, para la mayoría de los programas locales, es el operar con un solo hilo de control, de esta forma, los objetos son accedidos en una forma secuencial bien definida por lo que no se necesita preocuparse del acceso concurrente.

Si se introducen múltiples hilos de control en un programa local, se debe considerar el posible orden de acceso a los objetos y el uso de mecanismos de sincronización para controlar el acceso concurrente a los objetos. En una aplicación distribuida, existen necesariamente múltiples hilos de control, ya que un objeto distribuido puede tener múltiples clientes concurrentes. Debido a esto se debe analizar el acceso concurrente a los objetos y utilizar los mecanismos de sincronización que sean necesarios para permitir el acceso.

Seguridad: un ambiente local normalmente tiende a ser más seguro mientras que su contraparte distribuida no lo es tanto, ya que cuando los objetos se encuentran colocados en el mismo proceso no es necesario el considerar aspectos de seguridad para autenticar la identidad de los objetos, mientras que cuando los objetos están en diferentes máquinas necesariamente se deben utilizar mecanismos de seguridad para poder determinar la identidad de otros objetos.

Sistemas de objetos distribuidos: son aquellos en los cuales todas las entidades son modeladas como objetos. Los sistemas de objetos distribuidos son un paradigma popular para aplicaciones distribuidas, ya que por medio de ellos la aplicación es modelada como un conjunto de objetos cooperativos y se ajusta naturalmente a los servicios de un sistema distribuido.

1.3. Grupo de administración de objetos (OMG)

Es a raíz de las tendencias anteriormente expuestas que han surgido varias iniciativas por parte de un grupo de compañías cuya actividad comercial e industrial no concierne únicamente al ámbito del desarrollo de *software* y telecomunicaciones. Es así como en abril de 1989 fue fundado el grupo de Administración de Objetos por once compañías incluyendo 3Com Corporation, American Airlines, Canon, Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems y Unisys Corporation. En octubre de 1989, el OMG inicio sus operaciones de manera independiente, como una corporación no lucrativa, con el compromiso de desarrollar especificaciones orientadas a la industria del *software* que sean técnicamente excelentes, viables comercialmente e independientes del desarrollador. El consorcio ahora agrupa a cerca de 800 miembros.

1.3.1. Objetivo

El OMG fue formado para crear un mercado de *software* basado en componentes, para lograr la introducción de *software* estandarizado de objetos. La agenda de la organización incluye el establecimiento de normas industriales y especificaciones detalladas de administración de objetos para proveer un marco común para desarrollar de aplicaciones con componentes.

El apego a estas especificaciones puede hacer posible el desarrollo de un ambiente computacional heterogéneo a través de las mayores plataformas de *hardware* y sistemas operativos. Las series de especificaciones del OMG detallan las interfaces necesarias para el procesamiento de objetos distribuidos. Es ampliamente popular su protocolo de Internet IIOP (Internet Inter-ORB Protocol), el cual está siendo usado como infraestructura tecnológica, por compañías como *Nestcape*, *Oracle*, *Sun*, IBM y otras.

Las especificaciones del OMG son usadas mundialmente para desarrollar e implementar aplicaciones distribuidas para mercados verticales como finanzas, telecomunicaciones, comercio electrónico, etc.

El OMG define la administración de objetos como el desarrollo de *software* que modele el mundo real a través de su representación por medio de objetos, los cuales proporcionan el encapsulado de atributos, relaciones y métodos de programas de *software* identificables.

Un beneficio clave de un sistema orientado a objetos es la habilidad que tiene para expandir su funcionalidad, extendiendo sus componentes existentes y agregando nuevos objetos al sistema. La administración de objetos redunda en el desarrollo más rápido de aplicaciones, mantenimiento más sencillo, enorme escalabilidad y programas reutilizables.

1.3.2. Estructura

El OMG está estructurado en tres partes principales: el comité de tecnología de plataforma (PTC), el comité de tecnología de dominio (DTC) y la junta de arquitectura.

La consistencia o la integridad técnica del trabajo producido en el PTC y DTC es administrado por la junta de arquitectura. Dentro de esta última y los comités de tecnología descansan todas las fuerzas de trabajo que dirigen el proceso tecnológico de adopción del OMG. Existen tres métodos principales para influenciar el proceso del OMG adicionales al impacto de la revisión general, comentarios y discusión abierta.

El primero es la habilidad para votar en puntos de trabajo o adopciones en las fuerzas de trabajo que finalmente son revisadas y votadas a nivel del comité de tecnología. El segundo consiste en poder votar en puntos de trabajo o adopciones en uno o ambos niveles. El tercero es la posibilidad de presentar tecnología para su adopción en uno o ambos niveles del comité de tecnología.

1.4. Aproximación estratégica para integrar la cadena de valor

Las relaciones, transacciones e interacciones entre clientes y negocios ha cambiado dramáticamente en los últimos años, y lo siguen haciendo para definir un nuevo modelo de negocios conocido como comercio electrónico.

Con el cambio en los hábitos de compra, los clientes y su creciente necesidad de información han empezado a esperar que los negocios les brinden poder de decisión o *empower* y, como resultado, los negocios se pueden beneficiar al permitirle a los clientes el que posean uno o más de los procesos de compra. No obstante, para que los clientes se ayuden a sí mismos a través del proceso de compra, los negocios deben proveerles la tecnología necesaria.

Adicionalmente al comercio electrónico *business-to-customer,* las transacciones *business-to-business* se estima son entre 7 y 10 veces más que el número de transacciones de *business-to-customer*. El punto clave entonces no es quién estará involucrado en el comercio electrónico de *business-to-business,* ya que el comercio electrónico no es un concepto nuevo. El punto clave es ¿cómo pueden integrar los negocios sus aplicaciones heterogéneas con tantos otros negocios de forma eficiente?.

Sin hacer caso a si se esta considerando llevar a cabo comercio electrónico de *business-to-customer* o de *business-to-business,* ambos requieren un enfoque de cadena de valor. La integración de los procesos y aplicaciones son parte de la cadena de valor y pueden ser orientados para incrementar el rédito, la satisfacción del cliente, nuevas oportunidades para ofrecer productos y servicios, menos defectos, mejor control, etc.

Una de las metas de la tecnología debe ser la de permanecer abierta y flexible para soportar los cambiantes objetivos de los negocios. No obstante, ¿cómo es posible permanecer abierto y flexible con tantas tecnologías pasadas, presentes y futuras? ya que tampoco es posible fijar las arquitecturas de los clientes y proveedores, a excepción de muy raros casos, pues normalmente se trabaja con la tecnología y procesos existentes del cliente. Con una amplia variedad de aplicaciones propias de una plataforma, con su funcionalidad completada, la integración de la cadena de valor[1] enfocada al comercio electrónico se debería realizar por medio de interfaces comunes de dominio.

---

[1] Cadena de valor: conjunto de actividades que una organización lleva a cabo para crear y distribuir productos y servicios, incluyendo actividades directas como la producción y actividades indirectas tales como recursos humanos y finanzas. La ventaja competitiva es alcanzada, de acuerdo al profesor Michael Porter, cuando una organización vincula las actividades en su cadena de valor de la manera más barata o experta de la que lo hacen sus competidores.

1.4.1. El modelo *bussiness-to-customer* de comercio electrónico

El surgimiento de Internet ha ofrecido una nueva forma de llevar a cabo los negocios que antes habrían tenido un costo prohibitivo e inaceptable para el cliente hace solo algunos años.

Con el cambio los hábitos de compra de los clientes y la creciente necesidad de información, los consumidores esperan que los negocios les brinden más poder de decisión, como resultado, los negocios pueden beneficiarse permitiendo que los clientes posean parte del proceso de compra y así eliminar los costos asociados. No obstante, para que los clientes puedan tener más poder de decisión en el proceso de compra, los negocios deben proporcionarles la tecnología que se los permita y tener un conocimiento completo de sus clientes, ya que si una vez le fueron presentadas ofertas genéricas, pueden ahora obtener sus propias ofertas.

Con los clientes pueden elegir estar más involucrados en el proceso de obtención. Los negocios de hoy en día se enfrentan a dificultades desafiantes, como el poseer completamente la experiencia de los clientes en medio del aumento en las presiones competitivas y los avances tecnológicos que hacen que sea más fácil para el cliente elegir entre distintas ofertas de negocio. Los productos que una vez fueron considerados como bienes especializados se están convirtiendo en producto de consumo, a medida que la distancia entre el cliente y el productor colapsa. Con diferenciadores tales como la satisfacción del cliente y el mercadeo directo, la carrera para integrar la cadena de valor se hace aún más crítica. La habilidad para controlar la experiencia completa de los clientes redundará en mayores niveles de satisfacción, posibilitando la retención de clientes y adquisición de nuevos, entre otras.

Para administrar la cadena de valor de forma adecuada y asegurar la confianza del cliente, los negocios también necesitarán garantizar una arquitectura robusta y una probada calidad de servicio a través de la cadena de valor completa.

La calidad de los servicios tales como una seguridad apropiada, control de transacciones, mensajes y muchos servicios más son requeridos durante la vida de una transacción.

1.4.2. Business-to-Business

Aunque hoy en día el énfasis se hace en capturar mercado estableciendo relaciones de negocio a cliente las transacciones de *business-to-business* ofrecen mayores oportunidades en términos de dinero y transacciones. Los estimados establecen los ingresos por transacciones de *business-to-business* en cerca de US$900 billones en contra de los US$80 billones que aproximadamente se registran por concepto de transacciones de *business-to-customer*.

El punto clave a considerar para la participación en el comercio electrónico de *business-to-business*, es como integrar las aplicaciones existentes con tantas otras heterogéneas de forma que sea ventajoso para todos los involucrados. Los requerimientos para las transacciones de *business-to-business* incluyen todos los que presentan las transacciones de *business-to-customer*, pero es necesario tomar en cuenta que presentan volúmenes más altos de transacciones, además de necesitar un mejor seguimiento de las transacciones así como una seguridad más alta debido a presiones internas y externas.

Las transacciones de *business-to-business* requerirán la habilidad para integrar aplicaciones o sistemas que fueron diseñadas, desarrolladas e implementadas por dos o más compañías diferentes.

Las compañías de *business-to-business* se hacen más complejas al considerar los efectos de la integración de las cadenas de valor de tantos proveedores y clientes, considerando que un proveedor para un negocio puede ser a su vez un cliente para otro.

1.4.3. Integración de la cadena de valor

Sin importar si se trata de desarrollar comercio electrónico de *business-to-business* o de *business-to-customer*, ambos requieren un enfoque sobre la cadena de valor. Ambas estrategias requieren la integración de procesos y aplicaciones con una variedad de proveedores y clientes. La integración de la cadena de valor permite ampliar los servicios para poseer la experiencia completa de los clientes, remover pasos del proceso que no agregan valor al negocio, reduciendo desperdicio de tiempo en el proceso, lo cual redunda en un ahorro.

Desde el punto de vista del negocio la integración de sistemas, elimina el desperdicio, reduce defectos, así como agrega otro gran número de beneficios que conllevan un valor real para los accionistas. El reto es cómo integrar tan diferentes tipos de procesos y sistemas técnicamente y a su vez asegurar una solución flexible que funcione con la tecnología en la que se ha invertido en el pasado, la que se usa hoy, y la tecnología que se pueda usar en un futuro.

1.4.4. Proceso de integración de negocios

Si se pudiera conocer las arquitecturas, estándares de diseño, estándares de codificación, y el proceso de implementación de algún proveedor, sería posible construir una interfaz para este proveedor en especial. Pero sucede que si se poseen más proveedores podría no ser posible desarrollar soluciones que funcionen con cualquiera de nuestros proveedores, considerando que se cuenta con presupuestos de gastos finitos.

Tampoco es posible el dictar las arquitecturas que deben emplear nuestros clientes o proveedores a excepción de muy raros casos. Es necesaria una forma segura de integrar la cadena de valor, por medio de un sistema de seguridad probada que maneje de forma segura las transacciones. Finalmente, se necesita la habilidad de almacenar permanentemente información para propósitos de dirección o históricos.

1.4.5. Inicio de la integración de negocio

En primera instancia, los servicios fundamentales deben estar listos antes de poder integrar múltiples aplicaciones. Afortunadamente, los beneficios asociados con la tecnología de objetos y el desarrollo de componentes permiten obtener soluciones prefabricadas, integrarlas en las aplicaciones ya existentes e implementarlas en poco tiempo. Por medio del uso de tecnología que ya se encuentra completa, un negocio puede empezar a integrar la cadena de valor de forma inmediata. No obstante, la integración de la cadena de valor involucra mas que únicamente tecnología, ya que en primer lugar existen puntos que deben ser satisfechos seguidos de una evaluación de la madurez del proceso de la cadena de valor completa.

1.5. CORBA

El paradigma que CORBA sigue es una combinación de dos metodologías existentes. La primera de ellas es la de la computación distribuida de cliente-servidor, la cual se encuentra basada en sistemas de paso de mensajes encontrados mayormente en ambientes basados en UNIX. La segunda es la metodología orientada a objetos. Los objetos CORBA son procesos que enlazan conjuntos de datos con otros objetos CORBA o clientes, esto lo consiguen por medio de adaptadores de objetos los cuales les permiten interactuar entre ellos a través del ORB.

Un adaptador de objetos es un objeto CORBA que conoce y utiliza los servicios de un ORB, esto permite al ORB realizar llamadas de métodos de objetos tales como crear, modificar y eliminar objetos, a la vez que proporciona seguridad para el ORB y el mapeo de referencias de objetos hacia implementaciones.

El propósito final es el de proporcionar otro nivel de abstracción para que los ORBs puedan tener una interfaz estándar con muchos objetos CORBA a través de diferentes plataformas y redes.

Los desarrolladores pueden utilizar CORBA para distribuir aplicaciones a través de redes cliente-servidor. En los primeros sistemas de computación se contaba con cientos de miles de líneas de código que corrían en un *mainframe*, y un conjunto de terminales tontas.. Ahora, sin embargo, es necesario contar con aplicaciones más robustas que sean capaces de comunicarse entre servidores y estaciones de trabajo.

Este nuevo enfoque brinda nuevos retos, ya que debe ser posible el mantener simple la distribución de aplicaciones, por lo cual es necesario contar con arquitecturas *plug-and-play* para poder distribuir aplicaciones cliente-servidor. Esto debe ser así para permitirle al programador escribir aplicaciones que corran independientemente a través de diferentes plataformas y redes.

Para poder lograr todo lo anterior es necesario contar con una arquitectura que pueda cumplir esos requerimientos, es por eso que la idea detrás de CORBA es la de ser un *software* intermediario que maneje y disperse solicitudes de acceso sobre conjuntos de datos.

Este *software* intermediario recibe el nombre de *Object Request Broker* (ORB). La función del ORB es la de interactuar y poder realizar peticiones a objetos dispares y entre los mismos objetos. Un ORB está situado en un *host* en medio de la información y la capa de aplicación.

Un ORB negocia los mensajes de los objetos con los objetos servidores y los conjuntos de datos. Por lo tanto, la meta de CORBA es hacer que la programación sea más fácil al lograr que las aplicaciones sean altamente portátiles. De esta forma las aplicaciones cliente pueden contactar objetos CORBA para obtener datos. Estos objetos pueden utilizar diferentes conjuntos de datos de diferentes objetos, pero no tienen su propio conjunto de datos. Esto permite que sean programas dependientes de datos, aunque operan basándose en otros conjuntos de datos.

El ORB maneja las interacciones entre estos objetos funcionando como una llamada a un procedimiento remoto orientado a objetos (RPC).

CORBA fue diseñado para ser más robusto y simple que las llamadas RPC y otras librerías ya existentes. Para lograr esta simplicidad, fue añadida una capa sobre la metodología de acceso de datos distribuida orientada a RPC, la cual es usada para servir como comunicación entre procesos con relaciones cliente servidor para obtener acceso a conjuntos de datos.

Los conjuntos de datos se encuentran por tanto separados y son independientes del desarrollador de aplicaciones. El desarrollador hace solicitudes de datos a través de objetos CORBA, y el ORB maneja el mensaje accediendo al objeto de datos a través de la definición del objeto.

La presencia de la metodología orientada a objetos en CORBA es el resultado de la necesidad y no de una elección. La programación orientada a objetos es una forma diferente y conveniente de explicar y desarrollar un programa.

En CORBA son utilizadas tres características de la programación orientada a objetos.

La primera de ellas es que el polimorfismo de objetos esta permitido, el ORB hace que diferentes objetos y sus conjuntos de datos asociados sean independientes y que puedan ser reutilizados por diferentes aplicaciones.

Segundo, se utiliza el encapsulado de datos, el cual consiste en que cada aplicación cliente no conoce nada acerca de la información a la que está accediendo, lo único que hace es pedir a través del ORB, el cual obtiene la información para algún objeto de la aplicación cliente.

Tercero, se proporciona herencia. Si una descripción de objeto es diseñada para tener una interfaz con un ORB, cualquier objeto derivado de ese objeto padre conserva la interfaz de su padre.

La transparencia del servidor también es mantenida dentro de CORBA. Los ORB pueden acceder y hacer llamadas a diferentes objetos CORBA en varias máquinas. Si una aplicación cliente se encuentra corriendo en un servidor, el ORB de ese servidor es capaz de localizar datos en un lugar diferente en ese servidor o en una máquina diferente.

CORBA permite que los clientes realicen invocaciones de operaciones sobre objetos distribuidos, sin tener que considerar los siguientes aspectos:

Ubicación de objetos: un objeto CORBA puede estar ubicado ya sea en el cliente o distribuido dentro de un servidor remoto, sin afectar su implementación o uso.

Lenguaje de programación: idealmente no debería presentar ningún problema el llevar a cabo la programación de aplicaciones en cualquier lenguaje de programación, entre los lenguajes soportados por CORBA se encuentran C, C++, Java, Ada95, COBOL y *Smalltalk* entre otros.

Plataforma de sistema operativo: CORBA corre sobre Win32, UNIX, VMS y en sistemas de tiempo real tales como VxWorks, Chorus y LynxOS.

Protocolos de comunicación e interconexión: los protocolos e interconectores en los que CORBA corre incluyen TCP/IP, IPX/SPX FDDI, ATM, *Ethernet*, *Fast Ethernet* y memoria compartida.

Hardware: CORBA protege a las aplicaciones de efectos secundarios debidos a la diversidad del hardware tales como formas de almacenamiento, tipos de datos y rangos.

1.6. Partes de CORBA

La OMG ha definido cuatro grandes partes de CORBA que son el ORB, los servicios de objeto, las facilidades comunes y los objetos de aplicación.

La parte más importante de la especificación de la OMG es el lenguaje de definición de interfaz (IDL). En el núcleo de todos los objetos CORBA se encuentra la interfaz IDL, la parte más importante de la implementación de una aplicación. La interfaz IDL para cada objeto le permite a un objeto interactuar con el resto del mundo comunicando los métodos y parámetros de ese objeto a otros objetos a través del ORB.

La implementación de la parte RPC de CORBA ocurre a través de los *stubs* y *skeletons*. Existen dos tipos de *stubs* uno para el proceso cliente y otro para el proceso servidor. El *stub* para el proceso de servidor es llamado *skeleton*. Estos *stubs* actúan como plantillas en el cliente y el servidor, para que los tipos de parámetros y los métodos sean consistentes. Los *stubs* son generados del mapeo de IDL para algún lenguaje dado, esto es que los *stubs* y *skeletons* son los IDL compilados para los objetos. Dentro de estos *stubs* se encuentran detallados el procesamiento de señales y el manejo de excepciones necesarias para que la comunicación ocurra.

1.6.1. Interfaz dinámica de tiempo de ejecución

CORBA también especifica una interfaz de invocación dinámica, la cual le permite a los objetos CORBA el no conocer sino hasta en tiempo de ejecución la información de interfaz de otro objeto con el cual se podrían comunicar y solicitarle servicios. Los vínculos a los objetos son manejados por el ORB mientras que la aplicación se encuentra ejecutándose, a esta interfaz del lado del servidor se le conoce como interfaz dinámica de *Skeleton* (DSI).

Un objeto o cliente realiza solicitudes en tiempo de ejecución a una referencia de objeto específica, pasando una selección de operación y todos los parámetros necesarios. Los parámetros aparecen como una lista vinculada dinámicamente y se encuentran sujetos a chequeo de tipos en tiempo de ejecución.

El DSI es utilizado por el ORB para enviar solicitudes a objetos que están implementados independientemente y de los cuales no tiene conocimiento en tiempo de ejecución de su implementación.

Los desarrolladores pueden utilizar DSI para definir interfaces desde sus objetos hacia interfaces desconocidas en tiempo de desarrollo. Aunque la aproximación del DSI es más flexible, es más difícil de implementar que su contraparte estática IDL, una vez desarrollado, el DSI es siempre más lento que un IDL estático.

1.6.2. El proceso

El ORB es el negociador de datos de este marco de trabajo. Un ORB conoce información acerca de las interfaces con ciertos objetos. Los IDL para los objetos que conoce están localizados en una lista dinámica llamada interfaz de repositorio.

Un ORB tiene la capacidad de conocer a otros objetos que residen en diferentes sistemas de servidores. Cuando un ORB es consultado, éste intenta encontrar una coincidencia entre el objeto de datos solicitado y su interfaz de repositorio. Entonces, a través de su implementación de repositorio, el ORB intenta enviar un mensaje al objeto que le fue solicitado.

Si el objeto de datos o su servidor no está corriendo, el ORB obtiene una referencia de cómo y dónde iniciar el objeto en la implementación del repositorio o en una base de datos asociada.

Una vez que el ORB tiene la referencia correcta, intenta iniciar el objeto y, acto seguido, procede a enviar el mensaje original. Si este proceso falla o el ORB no sabe del objeto solicitado el ORB le devuelve un mensaje de error apropiado a el objeto o aplicación que efectuó la llamada.

El ORB remueve la complejidad de una red distribuida de programación para los desarrolladores. De aquí que no se necesite preocuparse de cómo implementar y probar rutinas de llamada de procesos de bajo nivel para trasladar información de una aplicación a otra, en lugar de esto, simplemente se implementa una referencia IDL para el objeto y sus métodos de respuesta.

La interfaz de repositorio especificada por el OMG es guardada dentro de un servidor CORBA y mantenida por el proceso ORB. La interfaz de repositorio es un listado de todos las IDL de objetos que el ORB necesita conocer. Las IDL de los objetos no deben encontrarse forzosamente dentro del servidor dentro del cual se encuentra corriendo el ORB. Pueden ubicarse en diferentes servidores, pero deben estar registrados en una interfaz de repositorio para poder utilizarse.

El repositorio permite la existencia de objetos persistentes, sin importar el estado de un objeto. Por lo tanto, las aplicaciones no necesitan estar compiladas o vinculadas con la información de otros objetos CORBA, y no obstante son capaces de enviar peticiones y emplear información a través de la DII. El repositorio también posibilita que el ORB localice y active las variadas implementaciones de los objetos.

Los objetos CORBA no necesariamente se deben encontrar corriendo. Por lo tanto, un ORB debe saber como activar un objeto en caso de que sea necesario. Usualmente, la activación de un objeto requiere una tabla de símbolos o de otra base de datos la cual posea la ruta hacia el objeto o servidor y cualquier otra información necesaria. Tal información podría ser la última vez en la que un objeto o servidor fue accedido.

Si otro objeto CORBA realiza una solicitud a un objeto CORBA que no se encuentra corriendo, el repositorio de implementación permite la activación del objeto solicitado y el respectivo paso de la solicitud original.

1.6.3. Servicios de objetos

Los servicios de objetos se refieren a servicios fundamentales proporcionados por los objetos mismos que soportan interacciones comunes entre otros objetos y aplicaciones, los cuales siguen las especificaciones de servicios de objetos comunes (COSS). Los actuales servicios incluyen nombramiento, manejo de eventos, persistencia, ciclo de vida, concurrencia, externalización y transacciones.

- El servicio de nombramiento de objeto mapea un nombre entendible para un humano a un objeto relativo a su contexto. Este servicio no depende de ningún otro servicio.

- Los servicios de manejo de eventos se refieren a la comunicación asincrona entre distintos objetos.

- Los servicios de persistencia de objetos aseguran que un objeto sobreviva a su creador. De esta forma, si el objeto es llamado tiempo, más tarde los métodos se encontrarán en su lugar para que el objeto puede activarse y continúe siendo utilizado. Esto también permite que el estado exacto de un objeto sea retenido en cualquier momento.

- Los servicios de ciclo de vida de un objeto determinan los métodos para la creación y finalización de un objeto.

- Los servicios de concurrencia de objetos son usados para tener vistas distribuidas de un objeto dado. Las vistas distribuidas permiten que un objeto sea utilizado en un ambiente global de nombres.

- La externalización de servicios de objetos se refiere a la recolección de objetos y su transporte como paquetes seriales.

- Los servicios de transacción de objetos permiten que múltiples objetos compartan una transacción.

Las facilidades comunes son servicios e interfaces de alto nivel, y además están relacionadas para extender los servicios de objetos existentes. Los objetos de aplicación se refieren a los objetos construidos por un programador para utilizar el marco de trabajo CORBA.

1.6.4. IDL

CORBA es definido y mapeado para un lenguaje en particular tal como Java a través del IDL. Esta definición consiste de métodos y parámetros que componen la interfaz de objeto.

El IDL describe las interfaces que los objetos clientes utilizan cuando quieren referenciar una implementación de objeto. Cada interfaz IDL es definida completamente para el objeto.

1.6.5. Interoperatividad de CORBA

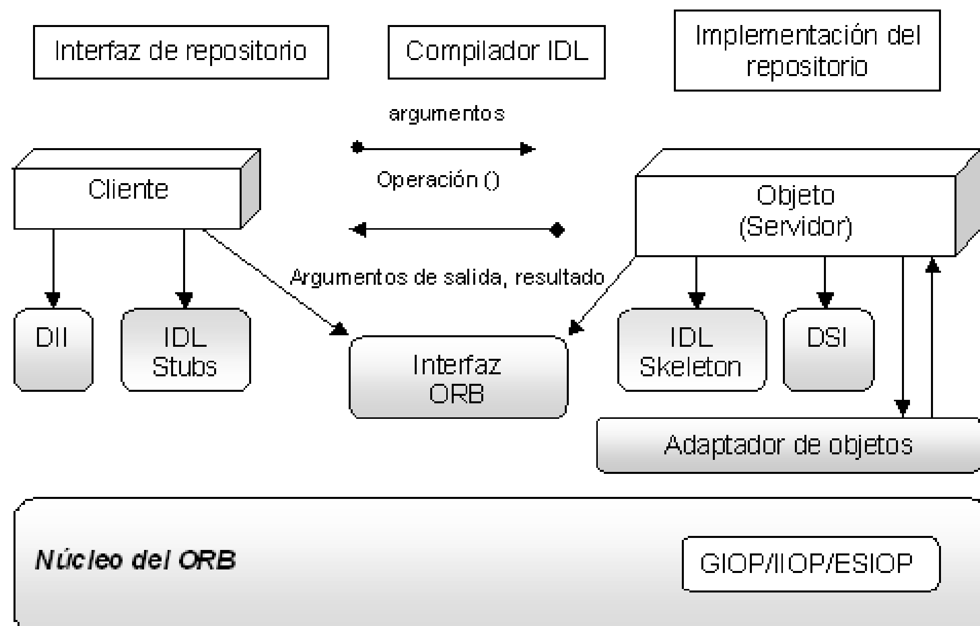
El estándar de interoperatividad CORBA fue desarrollado para permitir que diferentes ORB se pudieran comunicar. La interoperatividad de CORBA provee una infraestructura que hace que distintos ORB sean compatibles y se asienta en la capa de transporte del modelo OSI.

El protocolo general Inter-ORB (GIOP) especifica una condición principal la cual es que los diferentes ORB deben cumplir con el protocolo TCP/IP. Esta forma especializada del GIOP es referida como el protocolo de Internet Inter-ORB (IIOP). A través de la especificación CORBA, los métodos y servicios CORBA permiten que la metodología orientada a objetos sea llevada a una red distribuida. A través de IIOP, CORBA está estandarizado a un nivel más bajo de red, por lo que puede ser utilizado entre múltiples redes heterogéneas.

1.7. Componentes del modelo CORBA

El modelo CORBA se encuentra conformado por trece componentes, los cuales se describen en la siguiente figura.

Figura 1. Componentes del modelo CORBA



1.7.1. Cliente

Un cliente es un rol que obtiene referencias a objetos e invoca operaciones sobre ellos para llevar a cabo tareas de aplicación. Los objetos pueden ser remotos o encontrarse relativamente cerca del cliente, sin que esto afecte el funcionamiento de la aplicación, de esta forma, los clientes pueden acceder a los objetos remotos como si se tratase de objetos locales.

1.7.2. Objeto

En CORBA, un objeto es una instancia de una interfaz de definición de lenguaje (IDL). Cada objeto es identificado por una referencia de objeto, la cual asocia uno o más caminos a través de los cuales un cliente puede tener acceso a un objeto en un servidor cualquiera. Un identificador de objeto asocia al objeto con su implementación, la cual recibe el nombre de sirviente, un identificador siempre es único dentro del espectro de un adaptador de objeto. Durante su vida, un objeto puede tener uno o más sirvientes asociados con él para implementar su interfaz.

1.7.3. Sirviente

Este componente implementa las operaciones definidas por una interfaz OMG IDL. En los lenguajes orientados a objetos como C++ y Java los sirvientes son implementados usando una o más instancias de clase. En los lenguajes no orientados a objetos tales como C, los sirvientes se encuentran implementados utilizando funciones y estructuras.

Un cliente nunca interactúa con los sirvientes directamente, sino que siempre lo hace a través de objetos identificados por medio de referencias de objetos.

1.7.4. Núcleo ORB

Cuando un cliente invoca una operación de un objeto el núcleo ORB es el responsable de entregar la petición al objeto y devolver una respuesta, si es que ésta existe de vuelta al cliente. Un núcleo ORB se encuentra implementado como una librería de enlace en tiempo de corrida dentro de aplicaciones de cliente y servidor. Para los objetos que se ejecutan remotamente, un núcleo ORB se comunica con ellos por medio de una versión del protocolo general Inter-ORB (GIOP), tal como el protocolo de Internet Inter-ORB (IIOP) el cual corre sobre la capa de transporte del protocolo TCP/IP.

1.7.5. Interfaz ORB

Un ORB es una abstracción que puede ser implementada de varias maneras, por ejemplo, con uno o más procesos o un conjunto de librerías. Para aislar a las aplicaciones de los detalles de la implementación la especificación CORBA define una interfaz con el ORB, la cual provee operaciones estándar para inicializar y apagar el ORB, convertir referencias de objetos en *strings* y, a la inversa, crear listas de argumentos para peticiones hechas por medio de la interfaz de invocación dinámica DII.

1.7.6. OMG IDL Stubs

Los *stubs* y *skeletons* IDL sirven como unión entre los clientes, los sirvientes y el ORB. Los *stubs* proporcionan una interfaz de invocación estática (SSI), que convierte los parámetros de la aplicación a un nivel común de representación de mensajes. Contrariamente, los *skeletons* implementan el patrón de adaptación y reconvierten la representación a nivel de mensaje a parámetros con tipo que tienen significado para una aplicación.

1.7.7. Compilador IDL

Un compilador IDL transforma las definiciones OMG IDL en *stubs* y *skeletons,* que son generados automáticamente en un lenguaje de programación, tal como C++ o Java. Adicionalmente, proveen transparencia del lenguaje de programación los compiladores IDL eliminan fuentes comunes de errores y proporcionan oportunidades para llevar a cabo optimizaciones del compilador.

1.7.8. Interfaz de invocación dinámica (DII)

La interfaz de invocación dinámica le permite a los clientes generar solicitudes en tiempo de ejecución, lo cual es altamente útil cuando una aplicación no tiene conocimiento en tiempo de compilación de la interfaz a la que accederá. La DII también permite a los clientes el realizar llamadas aplazadas sincronas, las cuales desligan la petición y la respuesta en dos porciones para evitar que el cliente se bloquee hasta que el servidor responde.

1.7.9. Interfaz dinámica de Skeleton (DSI)

La DSI es análoga a la DII, pero para el sirviente. Permite que un ORB devuelva respuestas a sirvientes que no tienen conocimiento en tiempo de compilación de la interfaz que implementan. Los clientes no necesitan saber si el servidor ORB emplea *skeletons* estáticos o dinámicos, de igual forma los servidores no necesitan saber si los clientes utilizan la DII o SII para realizar peticiones.

1.7.10. Adaptador de objetos

Un adaptador de objetos es un componente compuesto que asocia a los sirvientes con los objetos, crea referencias a objetos, demultiplexa solicitudes entrantes para los sirvientes y colabora con la IDL de *skeleton* para despachar la operación adecuada al sirviente. Los adaptadores de objetos permiten a los ORBs el soportar varios tipos de sirvientes que poseen requerimientos similares, esto permite que el ORB sea más pequeño y simple así mismo que soporte un gran rango de granularidades de objetos, tiempos de vida, políticas, estilos de implementación, etc.

1.7.11. Interfaz de repositorio

La interfaz de repositorio proporciona información en tiempo de corrida acerca de las interfaces IDL. Por medio del empleo de esta información, es posible para un programa el encontrar un objeto para el cual no conocía una interfaz en tiempo de compilación, y aun así determinar qué operaciones válidas es posible solicitar al objeto y hacerle las peticiones.

La interfaz de repositorio también proporciona una ubicación común para almacenar información adicional asociada con interfaces para otros objetos, tales como librerías de tipos para *stubs* y *skeletons*.

1.7.12. Repositorio de implementación

El repositorio de implementación contiene información que le permite a un ORB activar servidores para procesar sirvientes. La mayoría de la información dentro del repositorio de implementación es específica del ORB o de un ambiente de sistema operativo. Adicionalmente, el repositorio de implementación proporciona una ubicación común par almacenar información asociada con los servidores tales como controles administrativos, ubicación de recursos, seguridad y modos de activación.

1.7.13. Servicios CORBA

Otra parte importante de el estándar CORBA es la definición de un conjunto de servicios distribuidos para soportar la integración e interoperatividad de objetos distribuidos. Los servicios conocidos como servicios CORBA o COS se encuentran definidos sobre el ORB. Esto significa que se encuentran definidos como objetos estándar CORBA con interfaces IDL, a veces referidos como servicios de objeto. Existen varios servicios CORBA, entre los que se encuentran:

- Ciclo de vida de objeto. Define cómo son creados, removidos, movidos y copiados los objetos.

- Nombramiento. Define cómo los objetos pueden tener nombres simbólicos amigables.

- Eventos. Permiten la comunicación entre objetos distribuidos.

- Relaciones. Permiten tipos arbitrarios de relaciones ilimitadas entre objetos CORBA.

- Externalización. Coordina la transformación de objetos CORBA desde y hacia dispositivos externos.

- Transacciones. Coordina el acceso atómico a objetos CORBA.

- Control de Concurrencia. Provee un servicio de bloqueo de objetos CORBA para hacer posible el acceso serializado.

- Propiedad. Hace posible la asociación de pares nombre valor con objetos CORBA.

- *Trader*. Soporta la localización de objetos CORBA basándose en propiedades que describen el servicio ofrecido por el objeto.

- Consultas. Soporta consultas sobre objetos.

1.8. OMA (modelo de objetos)

El modelo de objetos provee una presentación organizada de los conceptos de objetos y su terminología. Define un modelo parcial para la computación que agrupa las características clave de los objetos.

El modelo de objetos de la OMG es abstracto y se encuentra basado en el hecho de que no está directamente realizado sobre ninguna tecnología. Un modelo concreto de objetos puede diferir de un objeto abstracto en las siguientes características:

- Puede hacer más elaborado al modelo abstracto al hacerlo más específico, por ejemplo, definiendo la forma de los parámetros solicitados o el lenguaje usado para especificar los tipos.

- Puede restringir el modelo eliminando entidades o imponiendo restricciones adicionales a su uso.

Un sistema de objetos es una colección de objetos que aísla a los solicitantes de servicios (clientes) de los proveedores de servicios por medio de una interfaz de encapsulado bien definida. En particular, los clientes se encuentran aislados de las implementaciones de servicios en su forma de representaciones de datos y código ejecutable.

El modelo de objetos describe conceptos que tienen sentido para los clientes, incluyendo conceptos tales como creación de objetos e identidad, peticiones y operaciones, tipos y firmas. Luego describe los conceptos relacionados con las implementaciones de objetos, incluyendo conceptos tales como métodos, mecanismos de ejecución y activación.

El modelo de objetos es más específico y directivo en la definición de conceptos significativos para el cliente. La discusión de la implementación de un objeto es más sugestiva, con la intención de permitir la máxima libertad para diferentes tecnologías de objetos, para proveer diferentes formas de implementación los mismos.

Existen otras características de sistemas de objetos que están fuera del alcance del modelo de objetos. Algunos de estos conceptos son aspectos de la arquitectura de aplicación, otros están asociados con modelos específicos a los cuales la tecnología de objetos está asociada. Algunos ejemplos de estos conceptos excluidos son objetos compuestos, vínculos, copia de objetos y transacciones.

También están fuera del alcance del modelo de objetos, detalles de las estructuras de control, ya que no dice nada sobre si los clientes son multi-hilo o mono-hilo y no especifica cómo están programados los ciclos de eventos ni cómo son creados, destruidos o sincronizados los hilos.

El modelo de objetos es un ejemplo de un modelo clásico, donde un cliente envía un mensaje a un objeto, éste lo interpreta y decide cuál servicio ejecutar. En el modelo clásico, un mensaje identifica a un objeto con cero o más parámetros actuales. Tal como en la mayoría de modelos de objetos, un primer parámetro es requerido, el cual identifica la operación a ser realizada; la interpretación del mensaje por parte del objeto comprende la selección de un método basado en la operación específica. Operacionalmente, la selección del método podría ser realizada, ya sea por el objeto o por el ORB.

1.8.1. Semántica de objetos

Un sistema de objetos provee servicios a los clientes. Un cliente de un servicio es una entidad capaz de realizar una petición de servicio. La semántica de objetos define los conceptos relevantes para los clientes.

1.8.2. Peticiones

El termino petición es ampliamente empleado para referirse a una secuencia completa de eventos que transcurre entre su inicio por parte del cliente, y el último evento casualmente asociado con tal iniciación. Por ejemplo:

El cliente recibe la respuesta final asociada con la petición formulada al servidor. El servidor lleva a cabo la operación asociada en el caso de una petición de una sola vía. La secuencia de eventos asociados con la petición termina en un fallo, de algún tipo. El inicio de una petición es un evento. La información asociada con una petición consiste de una operación, un objeto meta, cero o más parámetros actuales, y un contexto opcional de petición.

Una forma de petición es una descripción o modelo que puede ser evaluado o ejecutado múltiples veces para provocar el envío de peticiones. Una forma de petición alternativa consiste en llamadas a una interfaz dinámica para crear una estructura de invocación, agregar argumentos a la estructura de invocación, y llevar a cabo la invocación.

Un valor es cualquier cosa que pueda ser un parámetro legítimo en una petición. Más particularmente, un valor es una instancia de un tipo de datos OMG IDL. No obstante también existen valores que no son de objetos, así como valores que hacen referencia a objetos.

Una referencia de objetos es un valor que se denota de manera confiable a un objeto en particular. Específicamente, una referencia de objeto identificará al mismo objeto cada vez que la referencia sea usada en una petición.

Un objeto puede ser denotado por múltiples y distintas referencias de objetos, puede tener parámetros que son utilizados para pasar datos al objeto objetivo y puede tener también un contexto el cual provea de información adicional sobre la petición.

Una petición de parámetros puede ser usada para ser ejecutada en beneficio del cliente. Un posible resultado de la ejecución del servicio es el retorno al cliente de los resultados, si los hay, definidos para la petición. Si una condición anormal ocurre durante la ejecución de una petición, una excepción es devuelta. La excepción puede portar parámetros adicionales propios de tal excepción.

Los parámetros de solicitud son identificados por posición. Un parámetro puede ser un parámetro de entrada o de salida, o uno de entrada-salida. Una petición puede devolver un único valor, así como los resultados guardados en los parámetros de salida y en los parámetros de entrada-salida.

Los siguientes situaciones aplican para todas las peticiones:

- Para cualquier asociación de parámetros no está garantizado que será removida o preservada.

- El orden en el cual los parámetros asociados son escritos no se encuentra garantizado.

- El valor de resultado y los valores almacenados en los parámetros de entrada, salida y de entrada-salida está indefinido si una excepción es devuelta.

1.8.3. Creación y destrucción de objetos

Los objetos pueden ser creados y destruidos. Desde el punto de vista de un cliente, no hay ningún mecanismo especial para crear o destruir un objeto. Éstos son creados o destruidos como resultado de peticiones. El resultado de la creación de un objeto es revelada al cliente en forma de una referencia de objeto que denota el nuevo objeto.

1.8.4. Tipos

Un tipo es una entidad identificable con un predicado asociado, esto es, una función con un argumento matemático y un resultado de tipo *boolean* definido para ciertas entidades. Una entidad que satisface este tipo es llamada un miembro de este tipo. Los tipos son usados en firmas para restringir un posible parámetro o para caracterizar un posible resultado. La extensión de un tipo es un conjunto de entidades que satisfacen este tipo en un momento particular del tiempo. En otras palabras, un tipo de objeto es satisfecho solo por referencias de objeto.

1.8.4.1. Tipos básicos

- Enteros de complemento, 16-*bit*, 32-*bit*, y 64-*bit* con o sin signo.

- Números de punto flotante, precisión simple (32-*bit*), precisión doble (64-*bit*), doble-extendido (mantisa de al menos 64 bits, un *bit* de signo y un exponente de al menos 15 bits).

- Números decimales de punto fijo de más de 31 dígitos significativos.

- Caracteres definidos en ISO *Latin*-1 (8859.1) y otros conjuntos de caracteres sencillos o de múltiples *bytes*.

- Tipo *boolean* toma los valores *True* (verdadero) y *False* (falso).

- Un opaco que pueda ser detectado con un tamaño de 8 bits, garantizado para no sufrir conversiones de transferencia entre sistemas.

- Tipos enumerados, consistentes de una secuencia de identificadores ordenados.

- Tipo *string*, el cual consiste de un arreglo de caracteres de longitud variable. La longitud del *string* es un entero no negativo y está disponible en tiempo de corrida. La longitud puede tener un máximo definido.

- Un tipo contenedor, el cual representa un posible tipo básico o construido.

- Caracteres que pueden representar caracteres de cualquier conjunto de caracteres amplios.

- Arreglos de caracteres, consistentes en una longitud, disponible en tiempo de corrida, y un arreglo de longitud variable de caracteres.

1.8.4.2. Tipos construidos

- Un tipo registro *struct*, el cual consiste de un conjunto ordenado de pares (nombre, valor).

- Un tipo unión, que consiste de un discriminador cuyo valor exacto siempre está disponible seguido por una instancia de un tipo apropiado para el valor del discriminador.

- Un tipo secuencia, el cual consiste en un arreglo de longitud variable de un solo tipo, estando disponible su longitud en tiempo de corrida.

- Un tipo arreglo, que consiste de un arreglo multidimensional de un solo tipo.

- Un tipo valor, el cual especifica el estado así como el conjunto de operaciones que una instancia de este tipo debe soportar.

1.8.5. Interfaces

Una interfaz es una descripción de un conjunto de operaciones posibles que un cliente puede solicitar de un objeto, por medio de la interfaz. Provee una descripción sintáctica de cómo un servicio proporcionado por un objeto que soporta esta interfaz, es accedido por medio de este conjunto de operaciones.

Un objeto cumple con una interfaz si provee su servicio a través de operaciones de la interfaz, de acuerdo a la especificación de las operaciones.

El tipo de interfaz es un tipo de objeto, tal que una referencia de objeto satisfaga el tipo sí y solo sí el objeto referente también satisface esta interfaz. La herencia de interfaces provee el mecanismo de composición para permitir que un objeto soporte múltiples interfaces. La interfaz principal es simplemente la interfaz más específica que el objeto soporta y consiste de todas las operaciones en cerradura transitiva del grafo de herencia de la interfaz.

1.8.6. Tipos de valor

Un tipo de valor es una entidad que comparte muchas de las características de las interfaces y estructuras. Es una descripción de un conjunto de operaciones que un cliente puede solicitar y de los estados accesibles para tal cliente. Las instancias de un tipo de valor son siempre implementaciones locales concretas en algún lenguaje de programación. Un tipo de valor, además de las operaciones y estados definidos para sí misma, puede también heredar de otro tipo de valores y a través de la herencia múltiple soportar otras interfaces.

1.8.7. Interfaces abstractas

Una entidad abstracta es una entidad en la cual el tiempo de corrida puede representar ya sea una interfaz regular o un tipo de valor. A diferencia de un tipo de valor abstracto, no implica semántica de paso por valor, y a diferencia de un tipo de interfaz regular, no implica paso de valores por referencia. En vez de esto, el tipo de entidad en tiempo de corrida determina cual de estas semánticas emplear.

1.8.7.1. Operaciones

Una operación es una entidad identificable que denota una primitiva de servicio indivisible que puede ser solicitada. El acto de solicitar una operación es referida como una operación de invocación.

Una operación es identificada por un identificador de operación y posee una firma que describe los valores legítimos de los parámetros solicitados y los valores retornados. En particular, una firma consiste de:

- Una especificación de parámetros que es requerida en una petición de esa operación.

- Una especificación del resultado de la operación.

- Una identificación de las excepciones de usuario que pueden surgir por un llamado a la operación.

- Una especificación de información de contexto adicional que puede afectar la llamada.

- Una indicación de la semántica de ejecución que el cliente debería esperar de la llamada de una operación.

Las operaciones son potencialmente genéricas, esto significa que una operación puede ser invocada de manera uniforme en objetos con distintas implementaciones, posiblemente resultando en un comportamiento o resultado diferente.

La forma general, para una operación de firma es:

[unavia] <op_tipo_spec> <identificador> (param1, ..., paramL) [activa(excep1,...,excepN)] [contexto(nombre1, ..., nombreM)]

donde:

- La palabra clave "unavia" indica que la sintaxis correcta es esperada en las peticiones de esta operación; la semántica por defecto es una si la operación retorna exitosamente los resultados o más de una en caso de que una excepción sea devuelta.
- El **<***op_tipo_spec***>** es el tipo de resultado.
- El **<***identificador***>** provee un nombre para la operación de la interfaz.
- Los parámetros necesarios para la operación están marcados con los modificadores *in* (dentro), *out* (fuera), o *inout* para indicar la dirección en la cual la información fluye.
- La expresión activa indica cuáles excepciones definidas por el usuario pueden ser señaladas para terminar la invocación de esta operación, si no se provee una expresión de este tipo, ninguna excepción definida por el usuario será indicada.
- La expresión contexto indica que información de contexto estará disponible para la implementación del objeto.

1.8.7.2. Parámetros

Un parámetro puede ser caracterizado por su modo o su tipo. El modo indica ya sea que el parámetro debería ser pasado del cliente hacia el servidor *in*, o desde el servidor hacia el cliente *out* o ambos *inout*. El tipo del parámetro restringe el posible valor que puede ser pasado en las direcciones indicadas por el modo.

1.8.7.3. Excepciones

Una excepción es una indicación de que una operación no fue llevada a cabo exitosamente. Una excepción puede ir acompañada por información específica adicional, la cual es una forma especializada de registro y como registro puede consistir en cualquiera de los tipos descritos anteriormente.

1.8.7.4. Contextos

Un contexto de petición provee información específica adicional que puede afectar el desempeño de una petición, ya que en base a una mayor o menor cantidad de información se puede generar un resultado con un mayor o menor grado de precisión.

1.8.7.5. Semántica de ejecución

Dos estilos de semántica de ejecución se encuentran definidos en el modelo de objeto:

At-most-once (al menos una): si una petición de operación regresa exitosamente, fue ejecutada exactamente una vez, mientras que si retorna una indicación de excepción, esto indica que fue ejecutada más de una vez, característica que es útil para poder controlar de mejor manera el flujo de las aplicaciones.

Best-effort (mejor esfuerzo): una operación de mejor esfuerzo es una operación únicamente de solicitud que no puede retornar ningún valor y el solicitante nunca se sincroniza al terminar ésta en caso de que la misma consiga terminar. La semántica de ejecución que puede ser esperada y está asociada con una operación. Esto evita que un cliente y una implementación de un objeto asuman diferentes semánticas de ejecución.

1.8.7.6. Atributos

Una interfaz puede tener atributos. Un atributo es lógicamente equivalente a la declaración de un par de accesos de función, uno para obtener el valor del atributo y otro para fijar el valor del atributo. Un atributo puede ser sólo de lectura, en cuyo caso solamente la función de acceso está definida.

1.8.7.7. Implementación de objeto

La implementación de un objeto conlleva las actividades necesarias para efectuar el comportamiento de los servicios requeridos. Estas actividades pueden incluir el cómputo de los resultados de la petición y la actualización del estado del sistema. En el proceso se pueden hacer solicitudes adicionales. La implementación del modelo consiste de dos partes el modelo de ejecución y el modelo de construcción.

El modelo de ejecución describe cómo los servicios son ejecutados, mientras que el modelo de construcción describe cómo es que los servicios están definidos.

1.8.7.8. Modelo de ejecución

Un servicio solicitado es ejecutado en un sistema computacional ejecutando código que opera sobre ciertos datos. Los datos representan un componente del estado del sistema computacional. El código lleva a cabo el servicio solicitado, el cual puede cambiar el estado del sistema. Este código ejecutado para llevar a cabo un servicio recibe el nombre de método, el cual es una descripción inmutable de un computo que puede ser interpretado por un motor de ejecución.

Un método posee un atributo inmutable llamado formato de método, que define un conjunto de motores de ejecución que pueden interpretar el método.

Un motor de ejecución es una máquina abstracta no un programa, que puede interpretar métodos con ciertos formatos, haciendo que los cómputos descritos sean llevados a cabo. Un motor de ejecución define un contexto dinámico para la ejecución de un método. La ejecución de un método recibe el nombre de activación de método.

Cuando un cliente lleva acabo una petición, un método del objeto objetivo es llamado, los parámetros de entrada pasados por el solicitante son trasladados al método y los parámetros de entrada, entrada-salida y el valor de resultado son enviados de vuelta al solicitante.

La ejecución de un servicio solicitado hace que se ejecute un método que pueda operar sobre el estado persistente del objeto. Si la forma persistente del método no es accesible para el motor de ejecución, puede que sea necesario copiar en primer lugar el método o estado dentro del contexto de ejecución. Este proceso recibe el nombre de activación y el proceso inverso recibe el nombre de desactivación.

1.8.7.9. Modelo de construcción

Un objeto computacional debe proveer de mecanismos para que se lleven a cabo las peticiones. Estos mecanismos incluyen definiciones del estado del objeto, definiciones de métodos y definiciones de cómo la infraestructura del objeto habrá de seleccionar los métodos a ejecutar, además de cómo seleccionará las partes relevantes del objeto para hacerlas accesibles a los métodos.

Una implementación de objeto, es una definición que provee la información necesaria para crear un objeto y permitirle proveer un conjunto apropiado de servicios. Una implementación típicamente incluye definiciones de los métodos que operan sobre el estado del objeto.

2. COMPONENTES DE *SOFTWARE*

Un componente de *software* consiste en código que implementa un conjunto de interfaces definidas. Los componentes no son aplicaciones completas, es decir, no tienen la capacidad para correr por su cuenta. En lugar de esto pueden ser utilizados como piezas para resolver un problema más grande.

La idea del uso de componentes de *software* es muy poderosa, ya que de esta manera una compañía puede comprar un conjunto de módulos bien definidos para resolver un problema y a la vez puede combinarlos con otros componentes para resolver problemas más complejos. Por ejemplo, si se considera un componente de *software*, el cual computa el precio de mercancías y se le proporciona información sobre un conjunto de productos, en base a esa información éste encuentra el precio total de la orden.

No obstante, el problema de la obtención del precio se puede tornar un poco más complicado. Por ejemplo, el caso de una orden de partes de computadora, la cual se encuentra compuesta por memoria y discos duros. El componente de precio de mercancías podría encontrar el precio correcto basándose en un conjunto de reglas de precio tales como:

- Precios base de un módulo de memoria o un disco duro.

- Cantidad de descuentos que un cliente recibe por ordenar más de 10 módulos de memoria.

- Descuentos conjuntos que un cliente recibe por ordenar ambos memoria y disco duro.

- Descuentos locales que se dan dependiendo de el lugar donde el cliente resida.

- Costos adicionales entre los que se encuentran los cargos de envío e impuestos.

Estas reglas para obtener precio no se aplican únicamente para la venta de partes de computadora. Otros negocios, tales como los de cuidado de la salud y líneas aéreas, entre otras, usan reglas de cálculo de precio parecidas, aunque en un diferente contexto.

Obviamente, sería un desperdicio de recursos si cada compañía que necesitara del cálculo de sus precios de forma compleja tuviera que escribir su propio y sofisticado componente de calculo de precios. Lo anterior hace que tenga lógica que una empresa cree un componente genérico de cálculo de precio, el cual pueda ser vuelto a usar por diferentes clientes.

Los componentes reutilizables son bastante tentadores porque promueven el desarrollo rápido de aplicaciones. No obstante, para que un componente tenga éxito resolviendo un problema, ambos desarrollador y el cliente que usa el componente deben estar de acuerdo en la sintaxis para las llamadas de los métodos del componente, para que de esta forma, conforme el desarrollador libere nuevas versiones de componentes, no existan problemas si el cliente obtiene las nuevas versiones y las utiliza en su aplicación existente. Esto se logra debido a que la interfaz de todo componente separa la interfaz de su implementación.

La interfaz de un componente define las convenciones de solicitud de peticiones del componente con el código que lo llama, esto es que la interfaz define métodos y parámetros que el componente acepta. Es de esta forma como se enmascara la implementación del componente.

En lo que respecta a la implementación del componente, consiste en la programación que un componente provee la cual consiste de algoritmos específicos, lógica y datos, los cuales son privados del componente y deben estar ocultos para todo el código cliente que llame al componente.

2.1. Reutilización de componentes

La reutilización es una actividad, no un objeto. Se trata de una actividad muy corriente en la vida cotidiana y no es más que utilizar algo de nuevo. Toda persona normalmente se encuentra familiarizada con el reciclaje y con la manera de hacer que una cosa sirva para varias otras siempre que esto sea posible. Ahora bien, en el contexto de la creación de sistemas que hagan uso intensivo del *software*, la reutilización es simplemente cualquier procedimiento que produce un sistema mediante la reutilización de algo procedente de algún esfuerzo de desarrollo anterior. La única cuestión entonces es determinar qué se llega a reutilizar y cuál es el procedimiento que en caso de seguirse dará lugar a una reutilización que tenga éxito.

En el contexto de la ingeniería del *software*, la reutilización es a la vez una vieja y nueva idea, ya que los programadores han reutilizado ideas, objetos, argumentos, abstracciones y procesos desde los primeros días de la computación, pero los enfoques de la reutilización han sido siempre a la medida.

En la actualidad, es preciso construir, en periodos muy breves de tiempo, sistemas basados en computadoras sumamente complejos y que sean de muy alta calidad lo cual demanda un enfoque más organizado de la reutilización.

Un rápido estudio de los posibles beneficios de la reutilización del *software* revelará que se puede ganar mucho más que un simple ahorro de costos durante el desarrollo de un producto de *software*. Por ejemplo, la reutilización de un componente de *software* del cual se sabe que es fiable introduce un menor riesgo que el diseño y nueva codificación de ese mismo componente para una nueva aplicación, proporciona por tanto la ventaja de que aumenta la fiabilidad del *software* desarrollado.

Además, los asuntos de eficiencia también se pueden abordar de forma más efectiva si se puede centrar la atención en la optimización de un conjunto de componentes reutilizables, en lugar de tener que optimizar continuamente nuevas versiones de módulos ya existentes.

Cuando la reutilización predomina durante el desarrollo de una aplicación, el enfoque de construcción suele conocerse a veces con el nombre de desarrollo basado en componentes o *software* por componentes. Cuando se utiliza este enfoque, se emplea algo que recibe el nombre de biblioteca de componentes reutilizables, la cual podría considerarse como un inventario clasificado de todos los componentes y la funcionalidad que poseen para facilitar la tarea de elegir cuáles son necesarios para el desarrollo de aplicaciones basadas en componentes.

2.2. Distribución de componentes

La distribución de componentes consiste en determinar cual será la ubicación física que tendrán los componentes, es decir, qué porcentaje o cuáles de ellos se encontraran en el cliente o en el servidor. Aun cuando no existen reglas absolutas que describan la distribución de componentes de aplicación entre el cliente y el servidor, suelen seguirse los siguientes lineamientos generales:

- El componente de presentación/interacción suele ubicarse en el cliente. La disponibilidad de entornos basados en ventanas y la potencia de cómputo necesaria para una interfaz gráfica de usuario hace que este enfoque sea eficiente en términos de coste.

- Cuando es necesario compartir la base de datos entre múltiples usuarios conectados a través de una LAN, lo que se acostumbra normalmente es que la base de datos se ubica en el servidor. El sistema de gestión de la base de datos y la capacidad de acceso a la base de datos también se asignan al servidor, junto con la base de datos física.

- Los datos estáticos que se utilicen como referencia deberían ser asignados al cliente, esto sitúa los datos más próximos al usuario que tiene necesidad de ellos y minimiza el tráfico de red innecesario y la carga del servidor.

- El resto de los componentes de aplicación se distribuye entre cliente y servidor, basándose en la distribución que optimice las configuraciones del cliente, el servidor y la red que los conecta.

La decisión final acerca de la distribución de componentes debe estar basada no solamente en la aplicación individual, sino en la mezcla de todas las aplicaciones que estén funcionando en el sistema para que el rendimiento total del sistema sea el adecuado sin que algunas aplicaciones sufran debido a las necesidades de alguna aplicación.

Una arquitectura de componentes completa prepara el camino para lo siguiente:

- Desarrolladores que escriben componentes reutilizables.

- Proveedores que proporcionen contenedores que provean un ambiente de ejecución para servicios y componentes.

- Proveedores que proporcionen herramientas de mantenimiento, las cuales proporcionen complementos necesarios para los mismos componentes.

Esta aproximación permite a los proveedores proporcionar un conjunto de servicios que la mayoría de los componentes necesitarán, ahorrando de esta manera tiempo valioso de desarrollo e instalación. Además, quienes normalmente desarrollan estos productos son profesionales expertos en proporcionar este tipo de servicios. De esta forma, los usuarios ahorran tiempo comprando los componentes en lugar de construirlos. Adicionalmente, el desarrollo en general se ve fortalecido, ya que los desarrolladores son expertos en los dominios para los cuales se encuentran escribiendo los componentes por lo que la calidad de los mismos debe ser superior de acuerdo a la experiencia que van adquiriendo.

2.3. Desarrollo de componentes en Java

En el presente trabajo se emplea la arquitectura de desarrollo de componentes que es utilizada en Java, en la cual se utilizan primeramente *JavaBeans*. Éstos son partes específicas de una aplicación y son utilizados para construir aplicaciones completas.

Un *JavaBean* no es un componente instalable y la razón de esto es que no son aplicaciones completas. Debido a que no pueden ser instalados, los *JavaBeans* no necesitan un medio ambiente en el cual vivir, pues no necesitan un contenedor para que les cree una instancia, los destruya y les provea servicios debido a que la aplicación misma está compuesta por *JavaBeans*.

En el caso del estándar *Enterprise JavaBeans*, éste define un componente de arquitectura para componentes instalables que reciben el nombre de *Enterprise JavaBeans*, los cuales son componentes de aplicación listos para ser instalados. Pueden ser instalados solos o unidos con otros componentes en sistemas de aplicación más extensos. Estos deben ser instalados en un contenedor que provee servicios en tiempo de ejecución a los componentes tales como los servicios para crear instancias de componentes.

Los *Enterprise JavaBeans* son muy similares a otros dos tipos de componentes Java, como son los *applets* y *servlets*. Los *applets* pueden ser instalados en una página *Web*, donde el visor de *applets* del navegador provee un contenedor de tiempo de corrida para los *applets*. Los *servlets* pueden ser instalados en un servidor *Web*, donde el motor de *servlet* del servidor *Web* proporciona un contenedor de tiempo de ejecución para los mismos.

La verdadera diferencia entre *applets*, *servlets* y *Enterprise JavaBeans* es el dominio del cual cada tipo de componente debe ser parte.

Los *applets* son programas portátiles que pueden ser descargados rápidamente y ser ejecutados. Por ejemplo, un *applet* puede ser descargado de un servidor *Web* a un navegador *Web*, el cual típicamente despliega una interfaz de usuario. Mientras que los *servlets* son componentes de red que pueden ser usados para extender la funcionalidad de un servidor *Web* pues están orientados a recibir una petición y proporcionar una respuesta de vuelta al cliente, esto los hace ideales para llevar a cabo tareas *Web*, tales como proporcionar una interfaz HTML para un catálogo de comercio electrónico.

Ambos *applets* y *servlets* se ajustan bien para manejar operaciones del lado del cliente, tales como proporcionar una interfaz gráfica de usuario aunque no necesiten una forzosamente, realizar alguna otra lógica de presentación y operaciones livianas de lógica del negocio. El lado del cliente puede estar constituido por un navegador *Web* para el caso de *applets* que proporcionan interfaces utilizando clases Java. El lado de un cliente puede ser también un servidor *Web* cuando se trata de *servlets* que proporcionan interfaces HTML, en ambas situaciones, los componentes se encuentran tratando directamente con el usuario final.

Por otro lado, los *Enterprise JavaBeans* no están ideados para ser usados del lado del cliente sino que están diseñados para realizar operaciones del lado del servidor, tales como ejecutar algoritmos complejos o realizar grandes volúmenes de transacciones. El lado del servidor tiene diferentes tipos de necesidades, a diferencia de las de un rico ambiente gráfico.

Los componentes del lado del servidor necesitan correr en un ambiente de alta disponibilidad, tolerante a fallos, transaccional, multiusuario y seguro. Un servidor de aplicaciones provee el ambiente necesario para los *Enterprise JavaBeans*, y provee el contenedor en tiempo de ejecución necesario para manejarlos.

2.4. Consideraciones de construcción de aplicaciones del lado del servidor

El construir una aplicación del lado del servidor no es una tarea sencilla, debido a que se presentan muchas consideraciones tales como la escalabilidad, la facilidad de mantenimiento, la seguridad y la confiabilidad, entre otras.

Todas estas consideraciones deben ser debidamente analizadas, debido a que en este tipo de aplicaciones es posible llegar a contar con una gran cantidad de clientes, dependiendo de la aplicación del lado del servidor, por lo que puede ser una catástrofe que el o los servidores centrales colapsen, se tornen muy lentos o permitan el acceso hostil de personas al sistema. Por lo tanto, las aplicaciones del lado del servidor necesitan estar bien escritas y probadas además de que deben correr en ambientes robustos, esto es, que los servidores que hospeden las aplicaciones deben contar con las capacidades de procesamiento, memoria y comunicación que permitan brindar un servicio efectivo y eficiente a los clientes.

Cualquier aplicación con arquitectura cliente-servidor que se encuentre bien desarrollada posee una partición lógica de *software* en capas. Cada capa posee una diferente responsabilidad con respecto al total de la aplicación y en cada capa pueden existir uno o más componentes.

Un sistema dividido en capas es un sistema bien diseñado ya que cada capa es responsable de una tarea distinta, existen varias divisiones en capas pero la división en capas más comúnmente utilizada es la siguiente:

- Capa de presentación
- Capa de la lógica del negocio
- Capa de datos

Capa de presentación: contiene componentes que trabajan con las interfaces de usuario y manejan su interacción del usuario realizando funciones de validación de datos y control del flujo de la aplicación. Esta capa podría estar compuesta en una aplicación *Web* por *servlet*, *applets*, ASP, etc.

Capa de la lógica del negocio: se encuentra constituida por componentes que trabajan juntos para resolver problemas de negocio. Ellos normalmente se dedican a llevar a cabo operaciones con la información proporcionada o solicitada por el usuario, que cumple con las políticas y reglas de funcionamiento del negocio.

Capa de datos: es utilizada por la capa del negocio para tener un estado de persistencia permanente, es decir, está formada por componentes que controlan y proporcionan la conectividad a base de datos, que es donde se encuentran físicamente almacenados los datos. Bajo la capa de datos pueden encontrarse una o más bases de datos en las cuales se almacena la información, existiendo normalmente un distinto componente encargado de operar sobre cada una de ellas.

La ventaja de dividir una aplicación en capas lógicas radica en el hecho de que, de esta forma, se hace posible, por ejemplo, el agregar una diferente presentación para la aplicación, minimizando el impacto que este cambio pudiera ocasionar en la capas de lógica o de datos.

La separación física de las anteriores capas es algo totalmente diferente, pues en una aplicación que usa una arquitectura de dos capas. Dos de las capas pueden encontrarse separadas físicamente en distintas máquinas, formando dos capas separadas físicamente, aunque bien podrían encontrarse coexistiendo en una única máquina.

Sin importar el tipo de arquitectura empleado, las capas pueden encontrarse separadas una de otra por algún límite físico, tal como límites de máquinas, límites en el alcance de los procesos o por límites corporativos.

2.5. Arquitectura de dos capas

Una arquitectura de dos capas combina la capa lógica con alguna de las otras dos capas, ya sea ésta la de presentación o la de datos. Para el caso en el que se combina la capa de presentación con la capa de negocio, se dibuja un límite entre la capa de negocio y la capa de datos. Para este caso, si se considera la primera capa como un cliente y la segunda como un servidor, lo que se tiene es un cliente pesado y un servidor delgado.

En esta arquitectura, la aplicación cliente normalmente se comunica con la capa de datos a través de un puente de datos API, tal como el de conectividad de base de datos abierta ODBC (por sus siglas en inglés) o por medio de conectividad de base de datos Java o JDBC (por sus siglas en inglés). Esto separa al cliente de la base de datos en uso.

2.6. Características de una arquitectura de dos capas

Una arquitectura de dos capas normalmente posee las siguientes características, aunque no necesariamente deba o pueda presentar todas ellas:

- Los costos de instalación son altos cuando los *drivers* deben ser instalados y configurados en cada uno de los clientes, ya que podrían llegar a ser cientos o miles de máquinas.

- El costo del cambio de *driver* es alto, ya que el cambiar el *driver* de base de datos por otro requiere de la reinstalación en cada máquina cliente.

- Cambiar el esquema de la base de datos presenta un alto costo si los clientes acceden directamente la base de datos vía JDBC u ODBC. Lo cual significa que los clientes están tratando directamente con el esquema de base de datos, por lo que si se decide cambiar el esquema de la base de datos se hace necesario el reinstalar cada cliente.

- El costo de un cambio en el tipo de base de datos es alto. Los clientes están atados a un API de base de datos, que sin importar que sea relacional o de objetos hace que, si se decide cambiar de tipo de base de datos, no sólo sea necesario el reinstalar cada cliente sino además cambiar el código cliente para que se adapte al nuevo tipo de base de datos.

- El migrar la lógica del negocio puede representar un alto costo, debido a que cualquier cambio obliga a una nueva compilación del programa y su correspondiente instalación en cada uno de los clientes.

- Los costos de la conexión a base de datos son altos, debido a que cada cliente necesita establecer su propia conexión a la base de datos y a que están limitadas en número.

- El desempeño de la red sufre a causa de que cada vez que la capa de lógica realiza una operación de base de datos, es necesario hacer un cierto número de viajes redondos a través de la red, lo cual puede obstaculizar la red reduciendo el ancho de banda del que los otros usuarios disponen.

Si se coloca parte de la capa de lógica dentro de la capa de datos se puede pensar en la primera capa como en una capa cliente y en la segunda capa como un servidor. Con esta arquitectura, lo que se obtiene es un cliente delgado y un servidor pesado. Esto se lleva a cabo colocando parte de la capa lógica dentro de la base de datos por medio de módulos ejecutables dentro de la base de datos llamados procedimientos almacenados.

No obstante que el colocar parte de la lógica dentro de la base de datos aumenta el desempeño e incrementa la escalabilidad de instalación, muchos de los problemas listados anteriormente aún prevalecen, y aunque el desarrollo de procedimientos almacenados es un paso adelante, también agregan algunos problemas, por ejemplo, los lenguajes utilizados en los procedimientos almacenados son propietarios lo cual ata a los cliente a una base de datos en particular.

2.7. Arquitecturas de N capas

Una arquitectura de N capas agrega una o más capas al modelo anterior. En una instalación de N capas, las capas de presentación, lógica de negocio y de datos están separadas en sus respectivas capas. Con cuatro o más capas, se descompone cada una de las capas aún más, permitiendo que varias partes del sistema sean escalables independientemente.

Un ejemplo concreto de una arquitectura de N capas es una instalación *Web* de tres capas, la cual normalmente se descompone de la siguiente manera:

Capa de presentación que corre en el espacio de uno o más servidores *Web*, consistente en *servlets* Java, *scripts* para hacer adaptable la presentación por medio de páginas activas de servidor, páginas de servidor Java, etc. y una lógica de trabajo que las une.

La capa de lógica de negocio corre dentro del espacio de uno o más servidores de aplicación. Éstos son necesarios para proveer un ambiente en el cual los componentes de lógica de negocio puedan correr. El servidor de aplicaciones también administra estos componentes eficientemente y les provee un cierto número de servicios, por ejemplo, proporciona una capa de acceso a la base de datos para los componentes de negocio permitiéndoles tener persistencia, también es responsable de hacer que los componentes de negocio se encuentren disponibles para ser utilizados.

La capa de datos consiste en una o más bases de datos y puede contener lógica relacionada a los datos en forma de procedimientos almacenados.

2.8. Características de las arquitecturas de N capas

- Los costos de instalación son bajos ya que los *drivers* de base de datos ya se encuentran instalados, y configurados en el lado del servidor en lugar de en las máquinas clientes.

- Es mucho más barato instalar y configurar *software* en un ambiente del lado del servidor que instalar y configurar en máquinas cliente.

- Los costos por cambio de base de datos son más bajos, ya que los clientes no tienen acceso directo a las bases de datos sino a través de la capa media de acceso a datos. Esto permite migrar esquemas de base de datos, cambiar los *drivers* o incluso cambiar el medio de almacenamiento, de manera trasparente para los clientes.

- Es posible asegurar partes de la aplicación con *firewalls*. Muchos negocios deben brindar protección adecuada a su información, por lo tanto se puede colocar un *firewall* entre la capa de presentación y la capa de lógica del negocio para brindar mayor protección.

- La bajas en el rendimiento están localizados, es decir, que si una capa esta sobrecargada las otras tres capas pueden aún funcionar apropiadamente. Por ejemplo, para el caso de un ambiente *Web* los usuarios pueden ser capaces de ver la página principal de un sitio aun cuando el servidor de aplicación esté sobrecargado.

- Los errores están localizados, ya que si ocurre uno éste normalmente está ubicado en una sola capa. Las otras capas pueden aún funcionar apropiadamente mientras se resuelve el problema, siempre que esto sea posible.

2.9. Soluciones de arquitectura del lado del servidor

Existen multitudes de servidores de aplicación, los cuales proveen un ambiente de ejecución en el cual los componentes se pueden ejecutar y a los cuales se les puede proveer los servicios de capa intermedia, tales como alternabilidad de recursos y seguridad, de forma confiable y escalable.

Cada servidor de aplicación normalmente provee los componentes y servicios en una forma no uniforme y propietaria. Lo cual significa que, una vez elegido un servidor, el código está atado a la solución de ese proveedor. Esto reduce ampliamente la facilidad de transportación así como reduce y obstaculiza el comercio de componentes debido a que un cliente puede no ser capaz de combinar un componente escrito para un servidor de aplicación con otro escrito para un servidor de aplicación diferente. A raíz de esto es que la necesidad de una arquitectura de componentes del lado del servidor ha surgido.

Esta arquitectura necesita construir una interfaz entre el servidor de aplicación, el cual contiene los componentes y los componentes mismos, y permite administrar los componentes de manera transportable en lugar de propietaria. El objetivo es permitir a los desarrolladores de componentes el enfocarse en la lógica del negocio de los problemas por resolver y evitar el preocuparse de problemas adicionales como el manejo de recursos, redes, seguridad y demás. La meta es lograr el desarrollo rápido de componentes que corran del lado del servidor, lo cual permite a los desarrolladores liberarse de la capa de red preexistente mientras escriben componentes portátiles del lado del servidor, permite intercambiar los componentes entre varios servidores, sin tener que cambiar el código o recompilar los componentes.

2.10. Plataforma *Sun Microsystem's* Java 2

El lenguaje Java es adecuado para ser usado del lado del servidor. No obstante del lado del cliente Java tiene varios problemas, tales como inconsistencia con interfaces de usuario en distintas plataformas, baja velocidad en estas interfaces de usuario y versiones incorrectas de máquinas virtuales corriendo en las máquinas clientes.

Las aplicaciones del lado del servidor corren en un ambiente controlado, lo cual significa que la versión correcta del lenguaje de programación Java siempre será utilizado. La velocidad también es un problema de menor importancia del lado del servidor, debido a que típicamente un 80 por ciento o más de una aplicación de múltiples capas transcurre en la base de datos o a nivel de red. Java también es un lenguaje muy conveniente para escribir componentes del lado del servidor, ya que el mercado de servidores está dominado por máquinas UNIX. Lo cual significa que un lenguaje multiplataforma agrega un gran valor, debido a que un desarrollador puede escribir un componente e instalarlo en cualquier ambiente de servidor existente.

La plataforma Java 2 versión empresarial (J2EE) provee independencia de la plataforma, portabilidad, multiusuario, seguridad y una plataforma estándar para realizar instalaciones de componentes escritos en Java. Esta plataforma simplifica muchas de las complejidades que rodean a la construcción de una aplicación basada en componentes del lado del servidor y es muy análoga a el DNA de Windows. La diferencia más notable es que J2EE es una especificación, mientras que el DNA de Windows es un producto.

J2EE especifica las reglas que los desarrolladores deben respetar al escribir *software*, de esta forma, los desarrolladores implementan especificaciones J2EE con productos que cumplen con J2EE, además debido a que J2EE es una especificación, no se encuentra atada a un solo proveedor.

3. PLATAFORMA J2EE

La plataforma J2EE proporciona una aproximación al desarrollo tradicional de componentes de capa intermedia y una arquitectura adecuada para el desarrollo rápido de aplicaciones. CORBA, por otro lado, ofrece un conjunto más amplio de características de capa intermedia con las cuales poder trabajar. No obstante para usar los servicios CORBA, es necesario programar APIs complejas de capa intermedia, lo cual aumenta la curva de aprendizaje de CORBA. Es por esto que J2EE es más adecuado para el desarrollo rápido de aplicaciones.

Actualmente se realizan revisiones cerradas, lo cual significa que se encuentran bien definidas, de modo que si se desea desarrollar una aplicación utilizando la plataforma J2EE, se puede elegir la versión que desee con la seguridad de que es posible encontrar los componentes adecuados a la versión seleccionada y que son compatibles con aquellos componentes construidos por otros desarrolladores para la versión elegida. Existe, además, un paquete de prueba para verificar si un desarrollador está implementando J2EE de forma adecuada y cumpliendo con todo el estándar. Se cuenta también con una referencia para realizar implementaciones de forma que los desarrolladores puedan cumplir con los estándares J2EE.

3.1. Tecnologías J2EE

La plataforma Java 2 es un conjunto de servicios de capa intermedia, que hacen que sea más sencilla la construcción de aplicaciones para los desarrolladores, e incluye las siguientes tecnologías:

- *Enterprise JavaBeans* (EJB). Define cómo están escritos los componentes del lado del servidor y provee un contrato entre los componentes y los servidores de aplicación que los administran. Promueve el desarrollo de un mercado de componentes donde los proveedores pueden vender componentes reutilizables que son obtenidos para resolver problemas de negocio.

- Método de invocación remota (RMI) y RMI-IIOP, RMI. Permite la comunicación entre procesos y provee otros servicios de comunicación relacionados. RMI-IIOP es una extensión portátil de TMI que puede utilizar el protocolo Inter-ORB de Internet (IIOP) que puede ser usado para obtener integración con CORBA.

- Nombramiento Java y una interfaz de directorio (JNDI). Identifica las ubicaciones de los componentes y otros recursos a través de la red.

- Conectividad de base de datos Java. Es un puente de base de datos que permite operaciones de base de datos relativamente portátiles.

- API de transacciones Java (JTA) y servicio de transacciones Java (JTS). Estas especificaciones permiten que los componentes sean soportados en un ambiente confiable de transacción.

- *Servlets* Java y páginas de servidor Java (JSP). Ambos son componentes de red, los cuales son ideales para operaciones orientadas a solicitudes y respuestas, tales como interactuar como clientes a través de HTTP.

- Java IDL. Es la implementación de CORBA que permite la integración con otros lenguajes y facilita a los objetos soportar el rango completo de servicios CORBA.

3.2. Enterprise JavaBeans

Es una arquitectura que permite crear aplicaciones escalables. Provee una compleja capa media sin costo para los desarrolladores, es decir, consiste en un conjunto de funciones comunes que la mayoría de las aplicaciones necesitan. Los componentes pueden ser escritos desde cero o pueden ser escritos utilizando componentes existentes que proveen caminos bien definidos para realizar la migración de sistemas heredados sin tener que abandonarlos.

Utilizando EJB no es necesario conocer bastante acerca de capas medias para construir componentes diseñados para correr en arquitecturas de múltiples capas escalables. De esta forma, los componentes obtienen servicios de capa intermedia de manera implícita y transparente del servidor EJB ya que el servidor de aplicación maneja implícitamente transacciones, persistencia, seguridad, manejo de estados, concurrencia, etc.

3.3. Método de invocación remota

RMI es un mecanismo para invocar remotamente métodos en otras máquinas. Está estrechamente integrado con el lenguaje Java. EJB utiliza RMI como su API de comunicación entre componentes y sus clientes. RMI permite comunicarse en forma distribuida de manera casi idéntica a programar un *applet* independiente o una aplicación.

Abstrae los problemas de redes, haciendo que sea más sencillo el comunicarse en red. También posee otras funciones tales como descarga de clases, activación automática de objetos remotos, y recolección de basura de manera distribuida para limpiar espacio ocupado por objetos remotos no utilizados.

3.4. Nombramiento Java e interfaz de directorio

El nombramiento Java y la interfaz de directorio son un estándar para servicios de nombramiento y directorio. Los *Enterprise JavaBeans* dependen del JNDI para buscar componentes distribuidos a través de la red. Es una tecnología clave requerida para que el código cliente sea capaz de conectarse con un componente EJB. JNDI depende de la noción de un servicio de directorio.

Un directorio de servicio almacena información acerca de donde residen los recursos, así como otros datos tales como el nombre de usuario y su clave. En EJB, cuando un cliente solicita acceso a un componente, los servicios de directorio son empleados para localizar y recuperar un servicio de componentes para el cliente.

Históricamente, existen varios tipos de servicios de directorio, así como protocolos para tener acceso a ellos. Por ejemplo, NDS de Novell y el estándar de Internet LDAP, son estándares competidores así que cada uno tiene una diferente forma de acceso. Y cada servicio de directorio almacena información de forma propietaria.

El servicio de nombramiento y la interfaz de directorio (JNDI) resuelven este problema colocando un puente entre diferentes servicios de directorio.

Con JNDI se pueden escribir código portátil de nombramiento y servicios de directorio. Esto es posible ya que JNDI abstrae el código de cualquier servicio de directorio particular.

3.5. Conectividad de base de datos Java

El paquete de conectividad de base de datos Java es una extensión al estándar Java que permite a los programadores codificar para una API unificada de base de datos relacional. Utilizando JDBC se puede representar conexiones a base de datos, enviar sentencias SQL, procesar resultado de base de datos, etc.

Los clientes programan al API JDBC, el cual está implementado por un *driver* JDBC y es un adaptador que sabe como comunicarse de forma propietaria con una base de datos particular. JDBC contiene soporte para *pooling* de conexiones a base de datos aumentando la independencia entre el código y la base de datos.

3.6. API de transacción Java y servicio de transacción Java

Una transacción es una unidad de trabajo que debe ser ejecutada completa si ésta se encuentra dentro del marco de la aplicación. JTA es una interfaz de transacción de alto nivel que las aplicaciones utilizan para controlar las transacciones, mientras que JTS es un conjunto de interfaces de transacción de bajo nivel que hacen posible que múltiples proveedores colaboren al momento de ejecutar una transacción distribuida, en un ambiente heterogéneo. La implementación completa de la especificación JTA 1.0.1, consiste de dos partes:

- La interfaz de transacción. Permite demarcar una transacción, con lo cual el trabajo realizado por varios componentes puede ser unido por una transacción global. Esta es una forma de marcar grupos de operaciones para constituir una sola transacción global.

- Interfaz de recursos XA. Se basa en la interfaz X/Open/XA que permite el manejo de transacciones distribuidas. Esto involucra la coordinación de una transacción a través de más de un recurso, tal como una base de datos o una cola. Con J2EE los desarrolladores no necesitan preocuparse de la programación de transacciones explicitas con JTA ya que el trabajo es hecho a través de las API JDBC y EJB, manejadas por el contenedor y configuradas por el descriptor de instalación de aplicación. Los desarrolladores de aplicación pueden enfocarse en el diseño de transacciones en lugar de su implementación.

3.7. Servicio de mensajes Java

Es un mecanismo para poder intercambiar mensajes entre programas Java. Esta es la forma en que Java soporta la comunicación asíncrona de forma que tanto el emisor como el receptor no necesiten conocer el uno del otro. Por lo tanto, cada uno puede operar de forma independiente. JMS soporta dos modelos de mensajes:

- Punto a punto el cual se basa en colas de mensajes. Un emisor de produce un mensaje a una cola. Un consumidor de mensajes se puede adjuntar a la cola para poder escuchar los mensajes. Los mensajes pueden ser enviados a una sola cola y son usados por un solo consumidor.

- Publicación y suscripción es un modelo en el cual los emisores envían mensajes sobre un tema y todos los consumidores registrados para ese tema los reciben. En este caso, muchos consumidores pueden recibir el mismo mensaje.

3.8. *Servlets* Java y páginas de servidor Java

Los s*ervlets* son componentes de red que pueden usarse para extender la funcionalidad de un servidor *Web*. S*ervlets* están orientados hacia recibir peticiones y enviar respuestas ya que reciben peticiones de un cliente y luego generan una respuesta hacia el mismo. Esto los hace ideales para realizar tareas *Web*, tales como generar interfaces HTML.

Las páginas de servidor Java son muy similares a los s*ervlets*, de hecho, los *scripts* de las páginas JSP son compiladas dentro de un *servlet*. La mayor diferencia entre los *scripts* JSP y los *servlets* es que los *scripts* JSP no son código Java puro. Se utilizan cuando se quiere mantener la apariencia de la instalación separada físicamente y que sea más fácil darle mantenimiento.

3.9. *Mail* Java

El objeto de correo Java define un conjunto de interfaces para las cuales se escribe el código de la aplicación, y estas interfaces protegen el código de protocolos o servicios específicos de correo. Esto hace al código de correo portátil entre plataformas, así como a través de protocolos de correo. Además posee un conjunto de clases para simplificar el desarrollo de aplicaciones y posee unos cuantos proveedores de servicio que implementan los protocolos de correo más populares.

4. ENTERPRISE BEANS

Un *Enterprise bean* es un componente de lado del servidor que puede ser instalado en un ambiente de múltiples capas. Un *Enterprise bean* puede contener uno o más objetos Java debido a que un componente puede no ser más que un simple objeto. Sin importar la composición de un *Enterprise bean*, los clientes de un *bean* se relacionan con una sola interfaz del *Enterprise bean*. Esta interfaz, así como el mismo *Enterprise bean,* deben ajustarse a la especificación de los *Enterprise JavaBeans*, la cual requiere que los *beans* expongan unos cuantos métodos obligatorios que le permiten al contenedor administrar los *beans* de manera uniforme, sin importar en que contenedor el *bean* se encuentre corriendo.

Hay que hacer notar que el cliente de un *Enterprise bean* puede ser un *servlet*, un *applet* o algún otro *Enterprise bean*. Para el último caso, una solicitud de un cliente podría resultar en una cadena de *beans* siendo llamados. Esto puede representar una gran ventaja, ya que es posible dividir una tarea muy compleja entre varios *beans*, permitiéndole al *bean* el llamar a una gran variedad de *beans* para que manejen las sub-tareas.

4.1. *Beans* de sesión

Un *bean* de sesión representa un trabajo hecho para un cliente que lo ha solicitado. Son componentes reutilizables que contiene la lógica para los procesos de negocio. Implementan la lógica del negocio, sus reglas y su flujo de trabajo. Por ejemplo, un *bean* de sesión puede realizar cálculos de cuotas, toma de pedidos, compresión de video, transacciones bancarias, etc.

Los *beans* de sesión reciben este nombre debido a que su existencia es tan larga como el tiempo que dure la sesión que sostenga el cliente que llamó al *bean*, es decir, que si un cliente llama a un *bean* para que realice una operación de cálculo de precios, el servidor de aplicación es responsable de crear una instancia para el *bean* de sesión que fue invocado.

El servidor EJB es el responsable de manejar el tiempo de vida de los *beans*. Esto es, que el cliente no crea directamente las instancias de los *beans*, ya que el servidor de EJB lo hace automáticamente. Y es también el servidor EJB quien se encarga de destruir a los *beans* a su debido tiempo. Esto permite que los *beans* puedan recibir múltiples llamadas y que sean reutilizados por muchos clientes. Existen dos subtipos de *beans* de sesión que son los *beans* de sesión con estado y los *bean* de sesión sin estado.

4.2. ***Beans* de sesión con estado (*Stateful Session Beans*)**

Como se dijo anteriormente, los *beans* de sesión representan los procesos del negocio. Algunos de estos pueden ser llevados a cabo con una única solicitud a un método, por ejemplo el calcular el precio de un grupo de artículos, o la verificación de un número de cuenta de tarjeta de crédito. Otros procesos de negocio son más extensos y pueden conllevar muchas solicitudes a métodos y realizar múltiples transacciones. Un ejemplo de un proceso de negocio que se extiende para muchas llamadas a métodos es el de una tienda de comercio electrónico. Mientras que un usuario visita un sitio de comercio electrónico, el usuario puede agregar o quitar productos de su carretilla de compras. Lo cual implica un proceso de negocio que se divide en varias llamadas a métodos. La consecuencia de este proceso de negocio es que el componente debe controlar el estado del usuario, de solicitud en solicitud.

Por lo tanto un *bean* de sesión con estado es un *bean* que ha sido diseñado para atender procesos de negocio que conllevan múltiples solicitudes de procesos y transacciones. Para lograrlo, los *beans* de sesión con estado guardan el estado de cada cliente individual. Si un *bean* de sesión con estado cambia durante la llamada de un método, ese mismo estado será el que el cliente encuentre la próxima vez que el cliente realice una nueva solicitud.

4.3. *Beans* de sesión sin estado (*Stateless Session Beans*)

Un proceso de una única solicitud es aquel que no requiere que su estado sea mantenido a través de múltiples llamadas. Los *beans* sin estado son componentes que pueden manejar este tipo de solicitudes. Son proveedores de métodos anónimos, y se les llama de esta forma debido a que no se mantienen al tanto de la historia del cliente.

Un ejemplo de este tipo de procesos podría ser un complejo motor que resuelve complejas operaciones matemáticas de acuerdo a una entrada que bien podría consistir en comprimir datos de audio o vídeo. El cliente podría pasar un *buffer* de datos no comprimido así como el factor de compresión deseado y, una vez concluido el proceso, el *bean* se encontraría disponible para que algún otro cliente solicite sus servicios sin importar quien haya sido el último que los solicitó.

4.3.1. Elegir entre un *bean* con estado o uno sin estado

Al decidir cual usar, lo primero que se debe considerar es si se requiere que se guarde el estado, siendo entonces el componente ideal a utilizar.

Cuando se utilizan *beans* de sesión con estado, el guardar el estado limita el nivel de tolerancia a fallos, ya que si, por ejemplo, sucediera un error inesperado a nivel de sistema tal como la caída de un *bean*, pérdida de la comunicación o un reinicio de sistema el estado no es conservado.

Si se estuviera en un modelo sin sesión lo que se podría hacer para satisfacer las solicitudes sería redirigir transparentemente la solicitud hacia otro componente para que éste atienda la solicitud del cliente. En un ambiente con conservación de estado es muy poco lo que se puede hacer para redirigir la solicitud del cliente debido a que el estado se pierde cuando la falla ocurre.

No obstante, vale la pena acotar que algunos servidores de alto rendimiento están agregando servicios de recuperación para *beans* de sesión con estado. Estos servicios permiten que aun para este tipo de *beans* sea posible recuperar su estado de forma transparente.

Si se tiene un proceso que requiere que se mantenga el estado, existe otra forma alternativa para resolver el problema que consiste en pasar por parámetro el estado completo del cliente durante las solicitudes que el cliente realiza. El pasar el estado del cliente de esta forma puede llevar a una severa degradación del desempeño de la aplicación, lo cual es especialmente cierto si el cliente se encuentra localizado remotamente y si el estado que se pasa como parámetro es largo, aunque de esta forma se puede alcanzar tolerancia a fallos, la escalabilidad general del sistema puede verse comprometida debido al costo de la latencia de la red.

4.4. *Beans* de entidad (*Entity Beans*)

Otra parte fundamental de un negocio es la información permanente que el proceso del negocio utiliza. Para el caso de un componente de consulta de saldos los datos que utiliza es la información de cuentas del banco.

Para el caso de un componente de toma de órdenes que lleva a cabo el proceso de negocio de control de órdenes de productos, tales como crear una nueva orden de una nueva computadora que debe ser enviada al cliente, genera información que consta del número de orden así como del detalle de productos asociada a la orden.

En cada uno de los escenarios anteriores, los componentes de proceso de negocio manipulan información que se encuentra en algún medio de almacenamiento tal como una base de datos relacional. Un *bean* de entidad es un componente que representa información persistente. Los *beans* de entidad representan objetos de datos reales, tales como clientes, productos o empleados y no contienen lógica de procesos de negocio ya que modelan información.

Los *beans* de sesión manejan los procesos del negocio. Los *beans* de sesión pueden usar *beans* de entidad para representar la información que utilizan.

El valor que los *beans* de entidad proporcionan es que dan una vista orientada a objetos en memoria de la información almacenada en algún dispositivo de almacenamiento. La forma tradicional en que las aplicaciones trabajan con la información es por medio de el acceso a tablas que se encuentran contenidas en base de datos relacionales, leyendo y escribiendo la información conforme lo vayan necesitando.

Mientras que, por otro lado, los *beans* de entidad, son representaciones de la información subyacente. Es posible manejar la información almacenada en una base de datos relacional como si fueran objetos reales. Es posible leer la información de una base de datos y colocarla en memoria como un *bean* de entidad, para que de esta forma se pueda manipular al *bean* ejecutando métodos sobre él.

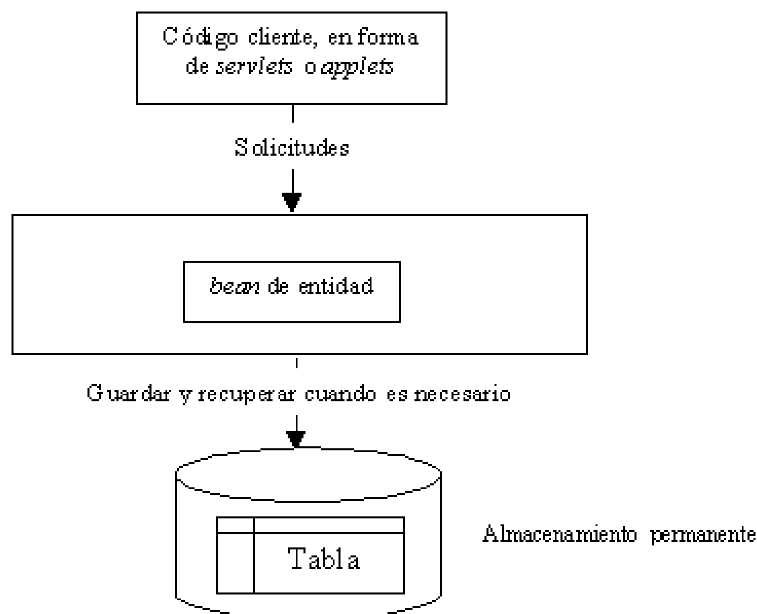
Lo anterior puede permitir el efectuar una llamada a un método denominado retiro sobre un objeto de entidad cuenta, el cual retiraría dinero restando la cantidad deseada a un miembro privado del *bean* llamado saldo y una vez que el objeto cuenta es almacenado la base de datos ya contiene el nuevo saldo de la cuenta. Por lo tanto, los *beans* de entidad permiten combinar la funcionalidad de la información persistente con el conveniente encapsulado de objetos. En esencia, el trabajo que realiza un *bean* de entidad es implementar la lógica de acceso a la información en una arquitectura de múltiples capas.

Debido a que los *beans* de entidad modelan información permanente son persistentes. Sobreviven a fallos críticos, tales como caídas de servidor, ya que son sólo representaciones de la información almacenada en dispositivos de almacenamiento tolerantes a fallas.

Si una máquina se cae, el *bean* de entidad puede ser reconstruido en memoria nuevamente simplemente leyendo la información de la base de datos. Debido a que la base de datos sobrevive a caídas del sistema los componentes que la representan lo hacen también, esta es la diferencia principal entre los *beans* de sesión y los *beans* de entidad, ya que los *beans* de entidad tienen un ciclo de vida mucho más largo que la sesión de un cliente.

Los *beans* de entidad se diferencian de los *beans* de sesión en otro aspecto importante, y es que múltiples clientes pueden utilizar los *beans* de entidad simultáneamente por medio del uso de transacciones. Los *beans* de entidad son muy útiles cuando se tiene un sistema heredado de base de datos, ya que son muy buenos para proporcionar acceso a información persistente.

Figura 2. Esquema de funcionamiento de un *bean* de entidad



De hecho, los registros de la base de datos podrían haber existido antes de que la compañía decidiera utilizar una solución orientada a objetos, debido a que una estructura de base de datos puede ser independiente del lenguaje. EJB saca partido de esto y permite la transformación de la información de una base de datos en objetos. La carga de la transformación de la información puede recaer sobre el *bean* o el contenedor puede realizar la transformación automáticamente.

4.5. *Beans* de entidad persistentes administrados por *bean*

Los *beans* de entidad son componentes persistentes debido a que su estado es guardado en un medio secundarios de almacenamiento. Si se utiliza tecnología de mapeo de objetos relacional, se puede tomar un objeto en memoria y hacerle un mapeo en una base de datos relacional.

Se puede recuperar esos registros más tarde para reconstruir el objeto en memoria y utilizarlo de nuevo. Otro posible esquema consiste en utilizar una base de datos orientada a objetos como forma de almacenamiento, la cual almacene objetos en lugar de registros de tipo relacional. Un *bean* de entidad administrado por *bean* es un *bean* de entidad que debe ser hecho persistente a mano. En otras palabras, el desarrollador debe escribir el código para transformar el contenido de la memoria en una forma de almacenamiento subyacente tal como una base de datos relacional o una base de datos orientada a objetos.

4.6. *Beans* de entidad persistentes administrados por el contenedor

Este tipo de persistencia la proporciona automáticamente el contenedor EJB evitando el tener que escribir todos los métodos que permitan realizar las operaciones de almacenamiento, búsquedas, recuperación y eliminación. En lugar de esto se solicita al contenedor que maneje la persistencia de los *beans*. Esto proporciona en teoría independencia de la base de datos, permitiéndole al desarrollador el cambiar el almacén de datos por otro, ya que no se escribe código que para ninguna API de base de datos.

4.7. Clase *Enterprise Bean*

La especificación de *Enterprise JavaBeans* define los contratos entre las diferentes partes involucradas en una instalación, para que un *bean* puede trabajar dentro de un contenedor y para que pueda trabajar con cualquier cliente de ese *bean*. Para que pueda trabajar con cualquier cliente se debe adherir a una interfaz bien definida.

En EJB se proporciona la implementación de los componentes en una clase *Enterprise bean,* la cual es simplemente una clase Java que se ajusta a una interfaz bien definida y que sigue ciertas reglas.

Una clase *Enterprise bean* contiene los detalles de implementación de un componente, y aunque no existen reglas fijas en EJB, una implementación de un *bean* de sesión es muy diferente de una implementación de un *bean* de entidad. En una implementación de un *bean* de sesión la clase contiene típicamente procesos de lógica de negocio, mientras que en el caso de un *bean* de entidad la clase contiene lógica relacionada a la información del negocio.

La especificación EJB define unas cuantas interfaces estándar que toda clase *bean* debe implementar. Estas interfaces fuerzan a una clase *bean* a que exponga ciertos métodos que todos los *beans* deben proporcionar tal y como lo define el modelo de componentes EJB. El contenedor utiliza estos métodos para administrar los *beans* y para alertarlos de eventos de importancia. La interfaz básica que todo *bean* debe implementar es la javax.ejb.EnterpriseBean la cual se define de la siguiente manera:

```
Public interface javax.ejb.EnterpriseBean extends java.io.Serializable {}
```

Esta interfaz sirve como un marcador de interfaz. El implementarla indica que la clase que se esta construyendo es una clase *Enterprise JavaBean*. Un aspecto interesante de esta definición es el hecho de que extiende java.io.Serializable lo cual significa que todos los *Enterprise beans* pueden ser convertidos al tipo *bit-blob* y compartir todas las propiedades de los objetos serializables.

Ambos *beans* de sesión y de entidad tienen interfaces más específicas que extienden a la interfaz javax.ejb.EnterpriseBean. Todos los *beans* de sesión deben implementar la clase javax.ejb.SessionBean y todos los *beans* de entidad deben implementar la interfaz javax.ejb.EntityBean.

4.8. Objeto EJB

Cuando un cliente quiere utilizar una instancia de una clase *Enterprise bean*, nunca llama al método directamente de una instancia *bean,* en lugar de esto la llamada es interceptada por el contenedor de EJB y luego delegada a la instancia a la que se le hace la solicitud. Esto es así por varias razones:

- Una clase *Enterprise bean* no puede ser llamada a través de una red directamente debido a que no es una clase que se encuentre habilitada por sí misma para comunicarse a través de la red. En lugar de esto, el contenedor EJB maneja las comunicaciones a través de la red envolviendo al *bean* en un objeto habilitado para comunicarse a través de la red. De esta forma, quien recibe las peticiones de los clientes es el objeto quien a su vez delega las peticiones a la clase *bean*. Esto permite que el desarrollador no tenga que preocuparse de todas las consideraciones y problemas que el comunicarse a través de la red conlleva.

- Al interceptar las solicitudes, el contenedor de EJB puede llevar a cabo tareas de administración que son necesarias, las cuales incluyen lógica de transacciones, seguridad, creación de instancias, y cualquier otra tarea que el contenedor requiera.

- El contenedor de EJB puede seguir la huella de los métodos que invoca desplegar un gráfico de utilización en una interfaz de usuario administrador, reunir datos para poder llevar a cabo balanceo de carga. No existe un requerimiento que indique que el contenedor debe realizar estas tareas, pero debido a que el contenedor EJB intercepta todas las llamadas a métodos, existe oportunidad para que los contenedores las realicen.

El contenedor de EJB actúa como una capa de direccionamiento entre el código cliente y el *bean*. Esta capa de direccionamiento se manifiesta como un solo objeto que recibe el nombre de objeto EJB.

El objeto EJB es un objeto que conoce principalmente de redes, transacciones y seguridad. Es un objeto inteligente que sabe como llevar a cabo la lógica intermedia que el contenedor requiere antes de que una llamada a un método sea atendida por una instancia de un *bean*, de esta manera, un objeto EJB actúa como la unión entre el cliente y el *bean* exponiendo cada método de negocio que el mismo *bean* exponga.

Los objetos EJB pueden ser considerados como partes físicas del contenedor, ya que todos los objetos EJB tienen código específico de cada contenedor dentro de ellos. Esto se debe a que cada contenedor maneja la capa intermedia de forma diferente y provee diferentes calidades de servicio.

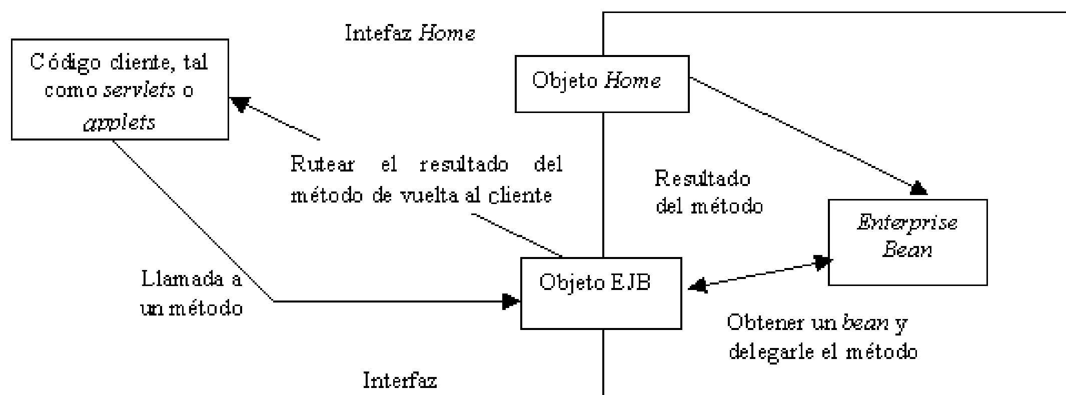
Debido a que cada objeto EJB es distinto, cada desarrollador de contenedores proporciona herramientas que generan los archivos clase para los objetos EJB automáticamente.

4.9. Interfaz remota

Ya que los clientes invocan métodos sobre objetos EJB en lugar de hacerlo sobre los *beans* directamente, los objetos EJB deben clonar cada método de negocio que las clases *bean* exponen. Esto es posible ya que las herramientas emplean una interfaz especial que los proveedores de *beans* proporcionan. Esta interfaz duplica todos los métodos de la lógica del negocio que corresponden a los que las clases *bean* exponen y recibe el nombre de interfaz remota.

Las interfaces remotas deben cumplir con ciertas reglas especiales que la especificación EJB define, por ejemplo, todas las interfaces deben derivar de una interfaz común que proporciona *Sun Microsystems* la cual tiene el nombre de *javax.ejb.EJBObject*.

Figura 3. Diagrama de relación entre objetos EJB al interactuar con un cliente



Los métodos que la interfaz javax.ejb.EJBObject proporciona se pueden apreciar en la siguiente tabla:

Tabla I. Métodos de la interfaz javax.ejb.EJBObject

Método	Explicación
getEJBHome()	Obtiene la referencia al objeto *home* correspondiente.
getPrimaryKey()	Devuelve la llave primaria de un objeto EJB. La llave primaria sólo se utiliza con los *beans* de entidad.
Remove()	Destruye un objeto EJB. Cuando un cliente termina de utilizar un objeto EJB, debe de llamar a este método. De esta forma, los recursos que este objeto estaba utilizando pueden ser reutilizados.
GetHandle()	Obtiene una referencia a un objeto EJB. Una referencia a un EJB puede ser utilizada por un cliente para guardarla y utilizarla luego para utilizar los métodos del objeto EJB.
IsIdentical()	Comprueba si dos objetos son idénticos

El código cliente que desea trabajar con los *beans* realiza llamadas a los métodos en la interfaz javax.ejb.EJBObject. Adicionalmente a los métodos antes mostrados, la interfaz remota duplica los métodos de los *beans*.

4.10. RMI Java

La interfaz *java.rmi.Remote* es parte del método remoto de invocación Java RMI (por sus siglas en inglés). Cualquier objeto que implementa la interfaz *java.rmi.Remote* es un objeto remoto el cual es llamado desde una distinta máquina virtual Java, de esta forma es que se llevan a cabo las llamadas remotas en Java.

Debido a que el objeto EJB implementa la interfaz remota de los *beans*, también implementa indirectamente a la interfaz java.rmi.Remote. Esto significa que todos los objetos EJB son objetos con comunicación de red y por lo tanto pueden ser llamados desde cualquier otra máquina virtual Java o desde cualquier otra máquina situada en cualquier otro lugar de la red. Por lo tanto la interfaces remotas EJB son sólo interfaces remotas Java con la excepción de que las interfaces remotas EJB deben ser construidas de acuerdo a la especificación EJB.

Las interfaces remotas EJB deben cumplir con las reglas de interfaz remota RMI, es por esto que cualquier método que es parte de un objeto remoto que puede ser llamado a través de distintas máquinas puede generar una excepción remota.

Una excepción remota es una excepción *java.rmi.RemoteException* e indica que algo inesperado sucedió en la red mientras se llamaba a algún método entre máquinas virtuales.

Las interfaces remotas también deben cumplir con las convenciones de paso de parámetros, pues no todo puede pasar por la red en una llamada entre máquinas virtuales. Los parámetros que se pasan entre métodos deben ser de tipos validos para el RMI, entre ellos están los primitivos, los objetos serializados y los objetos remotos RMI.

EJB también hereda un beneficio significativo del RMI, ya que en RMI la ubicación física de un objeto remoto que se está llamando se encuentra enmascarado para quien realiza la llamada.

EJB garantiza la transparencia de ubicación de componentes distribuidos. La transparencia de ubicación es necesaria en instalaciones de múltiples capas. Esto significa que el código cliente es portátil y que no se encuentra atado a una instalación de múltiples capas específicas.

4.11. Objeto *home*

El código cliente que se relaciona con objetos EJB nunca interactúa con los *beans* directamente. Un cliente no puede obtener una instancia de un objeto EJB directamente debido a que los objetos EJB pueden existir en una máquina distinta de la que se encuentra el cliente. De manera similar, EJB promueve la transparencia de ubicación para que los clientes nunca deban tener conciencia de donde es que residen en realidad los objeto EJB.

Para obtener una referencia a un objeto EJB, el código cliente solicita un objeto EJB de una fábrica de objetos EJB. Esta fábrica es responsable de crear instancias de objetos EJB. La especificación EJB llama a tal fábrica un objeto *home*.

Las responsabilidades principales de los objetos *home* son:

- Crear objetos
- Encontrar objetos existentes
- Remover objetos EJB

Al igual que los objetos EJB, los objetos *home* son propietarios de cada contenedor EJB.

4.12. Interfaz *home*

Un objeto EJB define primero como desea ser inicializado exponiendo un método de inicialización que puede poseer o no diferentes tipos de parámetros, especificando de esta forma una interfaz *home*. Las interfaces *home* definen los métodos para crear, destruir y encontrar objetos EJB.

EJB define algunos métodos requeridos para todas la interfaces *home*, estos métodos requeridos se encuentran definidos en la interfaz *jevax.ejb.EJBHome* que las interfaces *home* deben extender. La interfaz *javax.ejb.EJBHome* deriva de la interfaz *java.rmi.Remote*, lo cual significa que las interfaces *home* también lo hacen, por lo tanto, los objetos *home* también son objetos con comunicación a través de la red y pueden ser llamados a través de máquinas virtuales.

4.13. Descriptores de instalación

Los descriptores de instalación permiten a los contenedores EJB el proporcionar servicios implícitos de capa intermedia a los componentes *Entreprise java beans*. Un servicio implícito de capa intermedia es un servicio al que los *beans* pueden tener acceso sin tener que escribir ninguna línea de código de una API de capa intermedia pues los obtienen automáticamente.

Para informarle al contenedor de las necesidades de capa intermedia, se deben declarar los requerimientos de capa intermedia en un archivo descriptor de instalación. Por ejemplo, se puede utilizar un descriptor de instalación para indicar como se debe manejar el ciclo de vida, la persistencia, el control de transacciones y los servicios de seguridad.

Un descriptor de instalación puede ser utilizado para especificar los siguientes requerimientos de un *bean*:

- Administración del *bean* y requerimientos de ciclo de vida. Estos parámetros del descriptor de instalación le indican al contenedor como debería de administrar los *beans*. Por ejemplo, se especifica el nombre de la clase *bean*, si el *bean* es de sesión o de entidad y la interfaz *home* que genera los *beans*.

- Requerimientos de persistencia (para *beans* de entidad únicamente). Cuando se utiliza un *bean* de entidad se utilizan los descriptores de instalación para informarle al contenedor de cómo el *bean* maneja su persistencia, si lo hace por sí solo o si se lo delega al contenedor.

- Requerimientos transaccionales. Especificar sus requerimiento de transacción para los *beans* no necesita escribir ninguna línea de código para comunicarse con alguna API de transacciones.

- Requerimientos de seguridad. Los descriptores de instalación pueden contener entradas de control de acceso las cuales utilizan los *beans* y el contenedor para controlar el control de acceso a ciertas operaciones. Por ejemplo, es posible especificar a quien se le permite usar cuales *beans* y aun quien tiene derecho de llamar a algún método de algún *bean* en particular. También se puede especificar qué roles de seguridad son los que los *beans* deben correr, lo cual es útil cuando los *beans* necesitan realizar operaciones seguras.

5. RESPONSABILIDADES DE UN SERVIDOR Y UN CONTENEDOR EJB

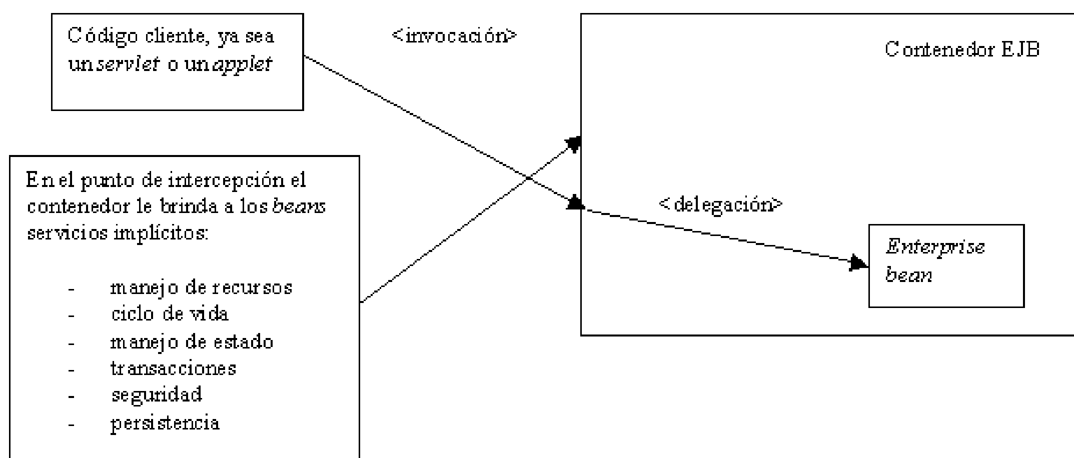
Un contenedor provee los servicios implícitos para los componentes EJB y los contenedores viven dentro del ambiente de un servidor EJB.

Los contenedores EJB son responsables del manejo de los *beans* y pueden interactuar con los *beans* llamando a los métodos de administración como sea necesario. Estos métodos de administración solo los invoca el contenedor y le permiten alertar a los *beans* cuando suceden los eventos de la capa intermedia, por ejemplo, cuando un *bean* de entidad esta por ser almacenado persistentemente.

La responsabilidad más importante para un contenedor EJB es proporcionar un ambiente en el cual los *Enterprise beans* pueden correr permitiendo que sus clientes los puedan invocar remotamente. En esencia, los contenedores EJB actúan como intermediarios entre los clientes y los *beans*. Son responsables de conectar a los clientes con los *beans*, llevar acabo la coordinación de transacciones, proveer persistencia y de administrar el ciclo de vida del *bean*, entre otras tareas tal como se aprecia en la figura 5.

La clave para entender los contenedores EJB es el considerarlos como entidades abstractas, ya que ni los *beans* ni los clientes especifican explícitamente el código para comunicarse con la API del contenedor EJB. En lugar de esto es el contenedor quien implícitamente maneja la sobrecarga de una arquitectura de componentes distribuida.

Figura 4. Diagrama de funcionamiento de un contenedor EJB



5.1. Manejo de recursos y ciclo de vida

Una arquitectura de múltiples capas ve mejorada su escalabilidad cuando un servidor maneja inteligentemente los recursos entre una gran variedad de componentes instalados. Los recursos pueden ser hebras, conexiones por *sockets* y conexiones a base de datos entre otros. Por ejemplo, las bases de datos pueden ser reutilizadas por los servidores de aplicación y reutilizados entre varios componentes heterogéneos. En el ambiente de un contenedor este último es responsable de proveer el manejo de todos los recursos.

Conforme las solicitudes de clientes llegan el contenedor instancia, destruye y reutiliza dinámicamente los *beans* de forma apropiada, por ejemplo, si un cliente solicita un *bean* que aun no existe en memoria el contenedor de EJB puede crear una nueva instancia en beneficio del cliente. Por otro lado, si un *bean* ya existe en memoria puede no ser apropiado crear una nueva instancia de *bean* cuando el sistema ya cuenta con poca memoria.

Lo más lógico es reutilizar un *bean* de un cliente para atender a otro cliente, también tiene sentido el destruir algunos *beans* que no se encuentren en uso, esto recibe el nombre de *pooling* de instancia. En resumen, el contenedor de EJB es responsable de coordinar el manejo de recursos completo así como del manejo del ciclo de vida de los *beans*.

5.2. Administración del estado

Para entender la necesidad del manejo del estado de un *bean* hay que observar el comportamiento que presentan los clientes o el código cliente ya que normalmente toman bastante tiempo para hacer las llamadas a los métodos de los componentes.

Una forma de ejemplificar este comportamiento es con el caso de un usuario que visita una página HTML, normalmente un usuario hace clic en un botón que ejecuta alguna regla de negocio por medio de un componente y luego espera hasta que aparezca de nuevo la página y procede, entonces, a leer la nueva página generada. Mientras esto sucede el servidor de aplicación podría utilizar el componente con el que se encuentra interactuando el cliente para atender a otro cliente distinto.

Esto es lo que sucede en EJB cuando un *bean* no posee estado, pues puede ser reasignado a otros clientes dinámicamente por el contenedor EJB. Esto es posible debido a que se almacena el estado del cliente para que le pueda ser reasignado a un *bean* empleando la información guardada. Esta reutilización resulta en un gran ahorro de recursos ya que sólo unas cuantas instancias de *beans* se necesitan para atender a varios clientes. Esto sucede ya que el contenedor de EJB se encuentra llevando a cabo un monitoreo en memoria constantemente de la actividad de llamadas recibidas por los *beans*.

Por otro lado, si el *bean* si posee estado las cosas se complican más pues el contenedor de EJB debe proporcionar un manejo transparente para los componentes con estado.

El manejo de estado es necesario cuando se desea reutilizar un componente con estado para dar servicio a varios clientes. Si se considera el escenario en el cual un cliente no ha utilizado un *bean* con estado en un largo tiempo, cuando un nuevo cliente se conecta al contenedor y solicita un componente el contenedor podría haber alcanzado su límite para crear instancias de componentes, en este caso en el contenedor puede que exista un componente que no ha sido utilizado por lo que el servidor podría serializar su estado convirtiéndolo en una cadena binaria y luego guardarlo en disco.

Una vez que el estado del cliente original ha sido guardado, el *bean* puede ser reasignado a un nuevo cliente y puede mantener el estado para ese nuevo cliente exclusivamente. Más adelante, si el cliente original realiza una petición, su estado puede ser recuperado del disco y ser utilizado de nuevo posiblemente en un distinto objeto en memoria.

5.3. Transacciones

Las transacciones son una forma segura de tener a múltiples objetos trabajando en operaciones distribuidas de objetos. Una transacción es una serie de operaciones que aparentan ejecutarse como una sola operación (operación atómica).

Las transacciones permiten que varios usuarios compartan la misma información garantizando que cualquier conjunto de datos que actualicen será escrito íntegramente y sin traslapes de modificación con otros clientes.

Utilizar transacciones de forma apropiada asegura que la información de una base de datos sea mantenida consistente. Las transacciones también aseguran que las operaciones de dos componentes de base de datos son independientes unas de otras.

Las transacciones previenen que desastres afecten la base de datos. Sin transacciones, una base de datos se puede tornar corrupta, lo cual es inaceptable en aplicaciones de misión crítica.

El contenedor de EJB maneja las operaciones de transacción subyacentes, coordinando esfuerzos entre los participantes de la transacción. El valor agregado que proporciona EJB es que las transacciones pueden ser llevadas a cabo implícita y automáticamente.

5.4. Seguridad

En una instalación crítica la seguridad siempre es un problema a considerar. El papel de los contenedores de EJB en la seguridad es el de manejar la validación de las peticiones de tarea que los usuarios desean realizar. Esto es logrado por medio de Listas de Acceso de Control o ACL (por sus siglas en Inglés).

Una ACL es un lista de usuarios y permisos, si el usuario posee los permisos correctos puede llevar a cabo la operación deseada. El paquete de desarrollo Java provee un modelo robusto de autenticación identificando si el usuario es quien dice ser a la vez que determina si el usuario posee los permisos necesarios para llevar a cabo la operación deseada.

Los contenedores de EJB agregan transparencia y seguridad sin tener que acceder a una API de seguridad, los *Enterprise beans* pueden correr como una cierta identidad de seguridad.

5.5. Persistencia

Los *beans* de entidad son objetos que representan datos que se encuentran en una capa subyacente que normalmente suele tratarse de una base de datos ya sea relacional u orientada a objetos. Los contenedores de EJB pueden proporcionar persistencia de forma transparente para los componentes.

5.5.1. Serialización de objetos Java

Cuando se trabaja con objetos Java, en muchos casos se hace necesario capturar el estado de algún objeto con el cual se este trabajando para poder almacenarlo en algún medio físico.

Una forma de poder capturar el estado de un objeto es por medio de la serialización del objeto. La serialización de objetos puede ser una forma sencilla de manejar un objeto gráfico en forma compacta.

Cuando se serializa un objeto gráfico, el objeto es convertido en una cadena de *bytes* para que de esta forma se puedan realizar cualquier tipo de operaciones sobre la cadena de la forma que sea necesaria o almacenar la cadena en un medio de almacenamiento que bien podría ser una base de datos, un sistema de archivos, etc.

No obstante para un manejo sofisticado de la persistencia la serialización de objeto no es suficiente, ya que si, por ejemplo, se tiene un millón de objetos de cuenta bancaria en un sistema de archivos almacenados en forma serializada y se quiere obtener los números de cuenta cuyo saldo es mayor a Q1000.00, lo que se tendría que hacer sería tomar todos los objetos desconvertirlos subirlos a memoria y revisar a cada uno de los objetos para ver si cumplen la condición, ahora bien, considerando este caso no existe ninguna manera eficiente de obtener los números de cuenta con la serialización de objetos.

En general, el hacer consultas sobre objetos almacenados utilizando serialización de objetos representa un costo demasiado alto, es por esto que se le utiliza en dominios restringidos tales como los de comunicaciones en red y persistencia simple.

5.5.2. Mapeo relacional de objetos

Otra forma de almacenar objetos Java es por medio del uso de bases de datos relacionales, para que en lugar de serializar cada objeto este sea dividido en sus partes constitutivas y, de esta forma, sea almacenado por aparte. Para lograr esto se puede utilizar JDBC o SQL/J para llevar a cabo el mapeo del objeto dentro de la base de datos relacional. Cuando se desea extraer un objeto de la base de datos, primero se debe crear una instancia para esa clase, leer los datos de la base de datos y luego llenar los atributos del objeto con los datos extraídos de la base de datos.

El mapeo de objetos hacia bases de datos relacionales es una tecnología que recibe el nombre de mapeo relacional de objetos, y consiste en el acto de convertir y desconvertir objetos en memoria en información relacional. El mapeo relacional de objetos es un mecanismo de persistencia mucho más sofisticado que la simple serialización de objetos. Al convertir los objetos en información relacional se pueden realizar consultas arbitrarias sobre la información.

5.5.3. Bases de datos de objetos

Un sistema de administración de base de datos de objetos ODBMS (por sus siglas en inglés), es un almacén que guarda objetos completos. En una base de datos de objetos los objetos son habitantes de primera clase en la base de datos, esto quiere decir que no existe ninguna capa de mapeo relacional de objetos ya que los objetos son almacenados íntegramente dentro de la base de datos. Debido a esto, no es necesario el realizar programas para comunicarse con ninguna API relacional.

La mayoría de las bases de datos de objetos proveen facilidades para consultar la información de los objetos por medio del lenguaje de consulta de objetos OQL (por sus siglas en inglés). OQL es una interfaz de alto nivel que permite el consultar características de los objetos de forma arbitraria, también agrega una capa de abstracción a las consultas relacionales de base de datos.

Adicionalmente a las consultas OQL, las bases de datos de objetos soportan relaciones entre objetos. Es posible establecer relaciones entre los objetos y navegar de manera transparente a través del modelo de objetos.

La navegación transparente hace que sea más fácil navegar por el modelo de objetos y tiene un buen desempeño comparado con los *joins* de SQL que son necesarios para realizar operaciones equivalentes sobre las bases de datos relacionales.

Las bases de datos de objetos también tienen un desempeño bastante predecible y escalabilidad. Ofrecen buena integridad y seguridad, a la vez que proporcionan un excelente almacenamiento para objetos complejos.

5.6. Acceso remoto y transparencia de ubicación

El acceso remoto consiste en la conversión de un componente nativo de una red en un componente que puede ser llamado remotamente. Los *Enterprise JavaBeans* aíslan al proveedor de componentes de todos los problemas y consideraciones de red.

Los *beans* son escritos como componentes aislados sin capacidad de conexión a red, pero una vez que son instalados en un contenedor EJB se convierten en componentes distribuidos. Los contenedores EJB utilizan las interfaces del método remoto de ubicación Java RMI (por sus siglas en inglés) para proporcionar el acceso remoto.

El beneficio de las tecnologías de comunicación distribuida tales como RMI es que los clientes no se tienen que preocupar por la ubicación física del componente al cual están accediendo. El componente se puede encontrar al otro lado del mundo, en una red de área local o en la máquina del mismo cliente. Podría incluso residir el espacio de direcciones de código del cliente (en un servidor de aplicación que soporte *servlets* Java así como *Enterprise beans* en una misma JVM).

Sin importar cual sea el caso, el cliente no tiene que preocuparse de donde es que se encuentra el componente realmente, esto recibe el nombre de ubicación transparente.

La transparencia de ubicación es de beneficio ya que no se escriben los *bean* para aprovechar alguna configuración de instalación específica pues no se codifican ubicaciones físicas de máquinas. Esta es una parte esencial de la reutilización de componentes que pueden ser instalados en una variedad de configuraciones con múltiples capas.

Además, la transparencia de ubicación también le permite al proveedor del contenedor el proporcionar valor agregado, tal como la habilidad para quitar una máquina de la red temporalmente para darle mantenimiento, instalarle nuevo *software*, o actualizar los componentes que la máquina posee.

Durante el mantenimiento, la transparencia de ubicación le permite a otra máquina reemplazar a la anterior para proporcionar los componentes a los clientes, ya que el cliente no depende de direcciones fijas en la red para solicitar los componentes.

5.7. Herramientas de instalación

Cada contenedor de EJB normalmente incluye un conjunto de herramientas. Éstas están hechas para integrar *beans* al ambiente de un contenedor EJB. Las herramientas generan código de ayuda Java tal como *stubs*, *skeletons*, clases de acceso a datos, y otras clases que puedan ser específicas para el contenedor.

Los desarrolladores de *beans* no necesitan preocuparse de problemas específicos sobre cómo trabaja el contenedor, ya que estas herramientas generan automáticamente su propio código Java automáticamente. Estas herramientas son las responsables de transformar a los EJB en componentes completamente distribuidos del lado del servidor.

5.8. Características especiales

Más allá de las actividades normales de un contenedor, los contenedores especializados pueden proporcionar cualidades adicionales de servicio que no son requeridas por la especificación. Estas cualidades ayudan a diferenciar a los desarrolladores de contenedores y permiten la innovación, aunque también agregan un elemento de peligro, ya que un *bean* podría en determinado momento depender de cierta característica de servicio que le podría impedir correr en otros contenedores.

El balanceo de carga es una selección de componentes en beneficio de los clientes para que los componentes puedan residir en muchos contenedores en muchos servidores EJB. Dado el hecho de que una instalación de N capas puede variar ampliamente en cuanto a la ubicación física de los componentes, el algoritmo de balanceo de carga es implementado independientemente.

El hecho de que el algoritmo de balanceo de carga sea implementado de forma independiente permite una gran dosis de creatividad por parte del proveedor del contenedor EJB, por lo tanto, algunos contenedores podrían proporcionar una forma de llevar a cabo un balance de carga parametrizable entre componentes distribuidos.

Por otro lado, podría existir un contenedor que proporcione crecimiento dinámico de los recursos administrados y de los componentes, por ejemplo, si los clientes muestran menor actividad durante las noches que durante el día podría ser que redujera el tamaño de la pila de recursos por la noche permitiendo utilizar los recursos liberados para otros procesos.

Otros ejemplos de funciones especiales que los contenedores pueden incluir están:

- Recuperación de estado.
- *Clustering*.
- Reinstalación dinámica de componentes en tiempo de ejecución.
- Soporte para una variedad de mecanismos de persistencia, incluyendo bases de datos relacionales o base de datos orientadas a objetos.
- Monitoreo.
- Transacciones distribuidas.
- Ambiente de desarrollo visual integrado.
- Servicios complejos de persistencia.
- Facilidades para XML integradas.

6. APLICACIÓN

6.1. Elección de un servidor

Para realizar la elección del servidor empleado para llevar a cabo el desarrollo e implementación de la aplicación realizada en el presente trabajo, fueron elegidos cuatro servidores de entre una amplia gama de servidores de aplicación que soportan EJB.

El criterio inicial para su elección fue su nivel de popularidad en el mercado de este tipo de servidores, justificando este criterio con el razonamiento de que sería una tarea demasiado ardua el probar cada uno de los servidores de aplicación que proporcionan soporte para EJB. Los cuatro servidores para prueba elegidos en base al criterio mencionado anteriormente fueron:

- *Weblogic Server* 6.1 desarrollado por BEA Systems, Inc.
- *WebSphere Advanced Single Server* v3.5 desarrollado por IBM *Corporation.*
- *Jboss* v2.4.6 desarrollado por *Jboss Group*.
- 9iAS *Standard* v1.0.2.1 desarrollado por Oracle Corporation.

El precio de los servidores fue ignorado, pues todos pueden descargarse sin costo alguno de sus respectivos sitios de Internet y ser usados para propósitos que no sean de índole comercial. Otra razón para no considerar el precio aun cuando pueda existir la posibilidad de que se dé uso a la aplicación, es que se encuentra escrita en Java, cumpliendo con el estándar J2EE, por lo cual es altamente transportable entre los distintos servidores.

Las versiones fueron obtenidas para el sistema operativo Windows NT 4.0, ya que la máquina de prueba y desarrollo fue una máquina IBM-PC compatible, con las siguientes características de procesamiento, memoria y almacenamiento:

- Procesador AMD/K62 de 500 Mhz.
- 256 MB RAM.
- Disco duro de 4 GB.

Debido a que se poseía un tiempo limitado para poder llevar a cabo la prueba de los distintos servidores de aplicación y al limitado poder de procesamiento de la máquina de prueba, los aspectos que más se tomaron en cuenta fueron la facilidad con que el servidor de aplicación se podía instalar, así como que tan sencillo y rápido se podían echar a andar pequeños programas de ejemplo. Así mismo, se consideró el desempeño que tenia el servidor de acuerdo a las características de la máquina.

A continuación se encuentran descripciones generales de las experiencias que se obtuvieron con cada uno de los servidores, las cuales varían entre uno y otro debido a que cada servidor posee distintas características y necesidades de configuración para su instalación, así como para el desarrollo y ejecución de programas de prueba.

No se pretende que las características a continuación expuestas constituyan una lista de todas las eventualidades posibles que se podrían dar durante el curso de las instalaciones de los productos ni durante su uso, ya que cada instalación puede ser diferente y presentar un grado de dificultad mayor o menor de acuerdo a las características propias del entorno donde se desee llevar a cabo la instalación.

6.1.1. Websphere

La instalación de *Websphere* no es sencilla y con suerte puede que no sea necesario realizarla más de una vez, no obstante, una vez instalado es funcional. Para el desarrollo de programas de ejemplo se utilizó el paquete *Visual Java Age* (VAJ) que es el paquete recomendado para ser utilizado con este servidor.

Este paquete se encuentra bastante bien integrado con *Websphere*, ya que fue posible crear EJB funcionales con pocos pasos. No obstante después de varias pruebas, se descubrió que poseen ciertas incompatibilidades, ya que al instalar el parche 2 para *Websphere* 3.5 el cual soporta *Servlets* 2.2 y JSP 1.1, dejó de funcionar adecuadamente la herramienta VAJ ya que no posee soporte para estas dos tecnologías en estas versiones.

Por tal motivo se intento utilizar otro producto de desarrollo, para solventar el problema de compatibilidad, pero debido a la gran integración existente entre el *Websphere* y VAJ, muchas de las características que se habían podido utilizar ya no son funcionales a nivel del entorno de desarrollo.

6.1.2. Jboss

El proceso de instalación es bastante sencillo y una característica importante de *Jboss* es que no utiliza el *registry* de *Windows* ni algún otro mecanismo específico de configuración, ya que toda esta información la almacena en archivos de texto.

El desempeño obtenido fue adecuado de acuerdo a las capacidades de procesamiento de la máquina en la que se realizaron las pruebas.

El mayor problema que posee es que gran parte de la documentación no se encuentra al intentar consultarla y, si se encuentra, normalmente no está actualizada, lo cual afecta a los desarrolladores que como en este caso no poseen experiencia en el desarrollo de aplicaciones en este servidor en particular.

Jboss no implementa la especificación de *servlets*, en lugar de esto se encuentra integrado con otros dos proyectos de código abierto, que son “Jakarta Tomcat” y “Jetty” para poder proporcionarlo, lo cual permite el correrlos sobre la misma máquina virtual o sobre una distinta. Esta característica es muy interesante si se desea realizar algún desarrollo de bajo costo, no obstante lo anteriormente expuesto, existe un fuerte argumento en contra de *Jboss* y es que para el caso de una aplicación con demasiados clientes este no soporta *clustering*, así que la única forma de escalar para *Jboss* por el momento es verticalmente.

6.1.3. Weblogic

Su instalación fue bastante sencilla y al momento de escribir este trabajo es el servidor preferido en el mercado. Este producto proporciona treinta días de uso en su versión de prueba, lo cual puede constituir una limitante en el caso de no contar con suficiente tiempo para llevar a cabo las pruebas durante el periodo en que el producto funciona.

A pesar de que posee un ambiente de administración bastante amigable, muchas veces las tareas de administración se vuelven problemáticas, ya que aunque permite la instalación de aplicaciones automáticamente en algunas ocasiones no funcionan completamente o se ejecutan con errores. El principal problema que presenta radica en el hecho de que utiliza más archivos de configuración de los que especifica el estándar J2EE definido por *Sun Micro Systems*.

Un punto fuerte a su favor es que posee una gran cantidad de documentación referente a la tecnología EJB, no obstante los ejemplos no son del todo claros y no brindan orientación de forma precisa en cuanto a la definición de los archivos de instalación, los cuales normalmente constituyen el más grande obstáculo a salvar para poder ejecutar programas de prueba.

6.1.4. 9iAS (OC4J)

La instalación fue la más rápida de todos los servidores probados con su configuración original. Pueden existir problemas al momento de llevar a cabo la instalación, ya que no posee un instalador automático que realice la instalación y establezca las variables de entorno necesarias para su ejecución. La configuración fue bastante sencilla, ya que esta se encuentra en archivos de texto pese a que no se utilizó ninguna herramienta de administración para poder llevar a cabo estas configuraciones.

Posee una adecuada cantidad de documentación referente a la tecnología de EJB, la cual, en general, fue bastante clara y concisa. Puede ser usado en conjunto con Apache para que éste le brinde servicios de *proxy*, enviándole las peticiones que le correspondan directamente.

6.2. Servidor elegido

El servidor que se eligió finalmente fue el 9iAS de *Oracle*, ya que fue el que presentó el menor grado de dificultad al momento de instalarlo, así mismo fue el servidor en el que la instalación y ejecución de los programas de prueba fue más sencilla, cumpliendo con los dos criterios iniciales que se establecieron como los más importantes a la hora de poder elegir el producto.

A continuación se muestra una descripción más detallada de las características del servidor elegido:

- El contenedor se encuentra escrito completamente en Java y se ejecuta con la máquina virtual (JVM) del *Kit* estándar de desarrollo (JDK). Puede correr sobre la JDK estándar del sistema operativo que exista en la máquina y no requiere el uso de la JVM Oracle9i.

- OC4J posee su propio servidor *Web*, el cual es recomendable configurar para que maneje las aplicaciones J2EE detrás de Apache, aprovechando el servicio *proxy* que este último puede prestarle.

- Liviano, ya que requiere menos de 10MB de espacio en disco y 12.2 MB de memoria al inicio.

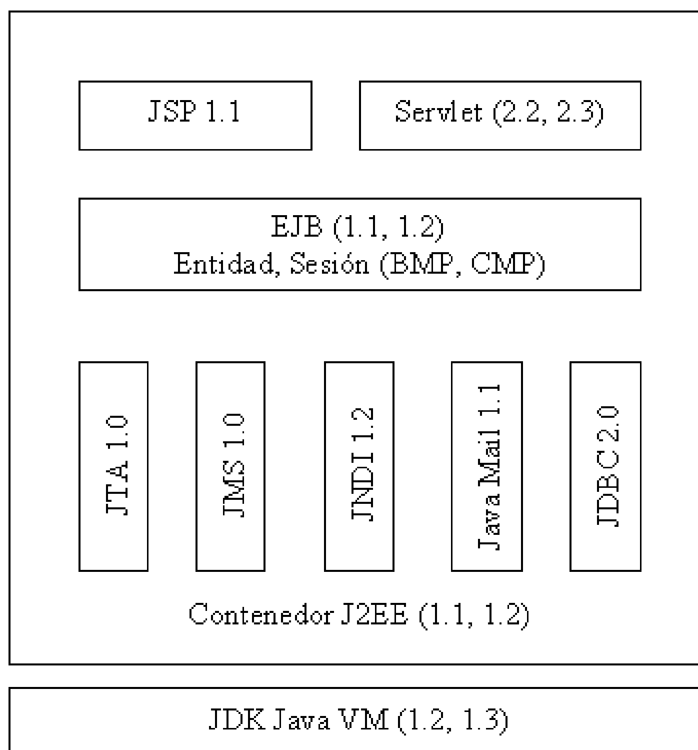
- Rápida instalación. Su instalación toma menos de dos minutos con la configuración original.

- Está certificado para correr sobre la JDK 1.2.2_07 y JDK 1.3.x.x. Y basa sus mejoras de desempeño y nuevas características en cada nueva versión de la JDK para cada sistema operativo y plataforma de hardware.

- Se encuentra disponible para todos los sistemas operativos y plataformas estándar, entre ellas Solaris, HP-UX, AIX, Tru64, Windows NT y Linux.

OC4J es un contenedor completo de J2EE 1.2 que incluye un traductor de JSP, un motor de *servlets* y un contenedor de *Enterprise JavaBeans* (EJB). También soporta el servicio de mensajes Java (JMS) y otras especificaciones más, como se muestra en la siguiente figura.

Figura 5. Especificaciones soportadas por el servidor OC4J



6.3. Funcionamiento del servidor

Cada instancia de servidor puede contener múltiples sitios *Web*, así como múltiples aplicaciones J2EE. Cada sitio *Web* posee un *listener* HTTP el cual es utilizado para proporcionar servicio a clientes que usan navegadores de Internet.

Cada aplicación es una aplicación estándar J2EE definida en un archivo EAR. Una aplicación puede tener componentes de aplicación *Web* así como EJB. Las aplicaciones *Web* son puestas a disposición de los clientes al enlazarlas a una URL en un sitio *Web*. Las aplicaciones que contienen únicamente EJB no se encuentran enlazadas a una URL, pero son accesibles en el servidor por medio de RMI. Las relaciones de cada uno de estos componentes están descritas en los archivos de configuración XML.

La configuración del servidor OC4J consiste en archivos básicos que son los archivos de configuración del sitio *Web* y el archivo de balanceo de carga. La configuración de una aplicación J2EE consiste de archivos XML para la instalación de la aplicación y la configuración del cliente. Los archivos de configuración del servidor OC4J son específicos para el servidor y apuntan a la ubicación de archivos claves de configuración de J2EE. Los parámetros que contienen no se relacionan al usuario de las aplicaciones sino al servidor mismo. Los archivos de configuración de la aplicación J2EE son específicos de J2EE permiten instalar las aplicaciones. Los archivos de configuración OC4J y de instalación se aprecian en las siguientes figuras.

Figura 6. Archivos de configuración del servidor OC4J

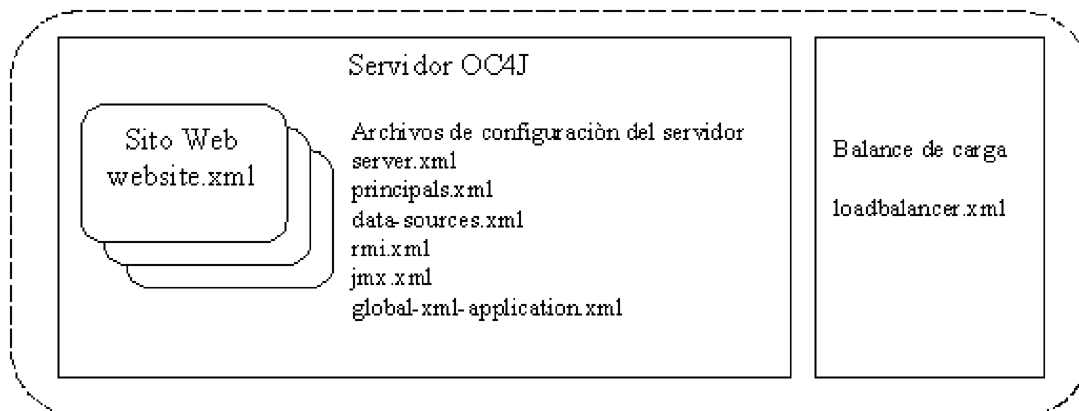
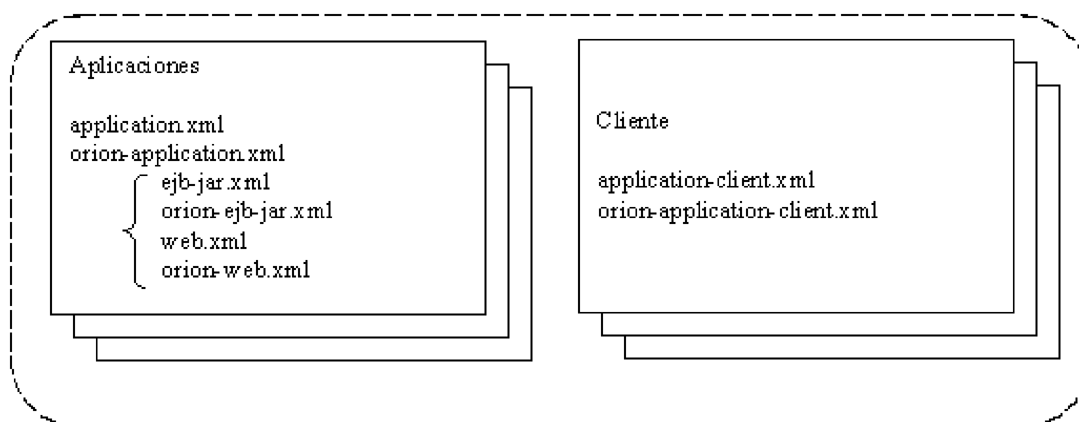


Figura 7. Archivos de instalación XML de J2EE



En la siguiente tabla se pueden apreciar las características de configuración y los archivos asociados.

Tabla II. Características de configuración y sus respectivos archivos asociados

Características/Componentes	Archivos de configuración XML
Configuración del servidor OC4J	server.xml
Seguridad OC4J	principals.xml
Desempeño de servidores OC4J	loadbalancer.xml
Soporte JDBC	Data-sources.xml
Soporte RMI dentro de OC4J	Rmi.xml
Soporte JMS dentro de OC4J	Jms.xml
Sitios *Web* definidos dentro de OC4J	*web-site.xml Cada sito *Web*, se encuentra definido dentro de su propio archivo *web-site.xml, para lo cual cada archivo debe nombrarse quitando el asterisco y reemplazándolo por el nombre deseado
Aplicaciones J2EE	application.xml y orion-application.xml
Aplicaciones *Web* J2EE	global-web-application.xml es un archivo de configuración específico para todos los servlets de todos los sitios *Web*. web.xml y orion-web.xml para cada aplicación *Web*.

6.4. Servidores lógicos y puertos de escucha

Cada instalación de OC4J puede contener varios servidores *Web* lógicos, cada uno con su propio puerto de escucha y un contenedor lógico de EJB con su puerto de escucha RMI. El archivo de configuración del *Web server* web-site.xml indica el puerto HTTP de escucha (por defecto 8888) al cual se puede acceder con URLs de la forma http://<host>:<web-server-port>.

El archivo rmi.xml define el puerto de escucha RMI (por defecto 23791). El servidor RMI puede ser accedido desde programas EJB clientes con URLs de la forma ormi://host:<rmi-server-port>.

Pasos para la instalación y configuración del servidor de aplicación Oracle9iAS:

- Descargar la última versión del Oracle9iAS para la plataforma elegida, del sitio http://ias.us.oracle.com/ o http://technet.oracle.com/.

- Descomprimir el archivo OC4J. Estos pasos asumen que el archivo es descomprimido en c:\ en un sistema Windows NT. El descomprimir el archivo crea una jerarquía de directorios bajo el directorio OC4J\ (OC4J_HOME).

- Si se desea habilitar JSP, así como otras tecnologías que necesitan acceso a un compilador Java, se debe copiar el archivo tools.jar al directorio OC4J.

- Configurar el puerto HTTP en el cual el servidor de aplicaciones escuchara las peticiones de sus clientes. Por defecto, el puerto que tiene configurado es el 80, pero si en la máquina que se está instalando ya se cuenta con otro servidor de aplicaciones el cual se encuentre escuchando sobre el mismo puerto, se hace necesario el cambiar el puerto que utilizara el Oracle9iAS, para que no existan conflictos. Esto se hace abriendo el archivo config\default-web-site.xml y editando la línea <web-site display-name="Default OC4J WebSite"> para que quede como <web-site port="8080" display-name="Default OC4J WebSite">. El número de puerto a utilizar puede ser cualquiera siempre y cuando sea valido y no se encuentre siendo utilizado.

- Para arrancar el servidor de aplicaciones Oracle9iAS, se ejecuta java –jar OC4J.jar. Si todo funciona correctamente al cabo de un corto tiempo el servidor muestra OC4J/x.x.x initialized, donde x.x.x es el número de versión del servidor. También se puede probar intentando acceder a la dirección http://localhost:<numero-de-puerto> donde número de puerto es el puerto elegido en el paso anterior.

Para instalar los *drivers* JDBC de *Oracle*, se deben descargar los *drivers* adecuados a la versión de base de datos con la que se pretende trabajar. Luego se procede a copiar el archivo del *driver* elegido en el directorio OC4J_HOME/lib.

6.5. Archivos de configuración XML

Un descriptor de instalación es un archivo que define la siguiente clase de información:

- Información estructural tal como el nombre del EJB, clase, *home* e interfaces remotas, tipo de *bean*, ambiente, referencias, seguridad, así como información basada en el tipo del *bean*.

- Información de ensamble de aplicación la cual consiste en referencias a otros EJB, seguridad, permisos sobre métodos y atributos del contenedor de transacciones.

6.5.1. Perfiles EJB y descriptores de instalación

La especificación de un descriptor de instalación es una tarea requerida que un proveedor de *beans* debe llevar a cabo. El proveedor de *beans* crea un archivo descriptor de instalación utilizando las convenciones del lenguaje XML, de acuerdo a la sintaxis definida en le especificación 1.1 de *Enterprise Java Beans*.

Todo descriptor de instalación EJB JAR válido, debe contener la siguiente declaración de tipo:

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

Un instalador o ensamblador de aplicación puede modificar sólo cierta información de un descriptor de instalación, tal como el nombre del EJB, valores de los parámetros de ambiente y descripciones.

Pasos comúnmente utilizados para la creación de un descriptor de instalación ejb-jar, con sus elementos básicos:

1. Crear un archivo de texto y nombrarlo ejb-jar.xml

2. Crear la sentencia DOCTYPE, todo descriptor de instalación ejb-jar contiene una referencia a la ubicación y versión del documento de definición de tipo (DTD).

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

3. Crear el cuerpo principal del archivo ejb-jar.xml. Todas las etiquetas deben ir encerradas dentro de etiquetas <ejb-jar>.

4. Definir la función u objetivo del EJB.

```
<description>
        Sistema de control de proyectos
</description>
```

5. Crear el cuerpo principal de la definición de cada EJB

```
<enterprise-beans></enterprise-beans>
```

6. Definir el nombre del EJB

```
        <display-name>
                nombre
        </display-name>
```

7. Definir el identificador del *bean*

```
        <ejb-name>
                nombre
        </ejb-name>
```

8. Definir el nombre de la interfaz *home*

    <home>
        nombre de la interfaz *home*
    </home>

9. Definir el nombre de la interfaz remota

    <remote>
        nombre de la interfaz remota
    </remote>

10. Nombre de la clase del *bean*

    <ejb-class>
        nombre de la clase
    </ejb-class>

6.5.2. Descriptores de aplicación *Web*

Los archivos descriptores de aplicaciones *Web* están escritos utilizando XML y su finalidad es la de poder definir los componentes y características operativas de una aplicación *Web* para que puedan ser instaladas en un servidor.

Normalmente una aplicación Web esta compuesta por servlets y/o páginas JSP que pueden realizar llamadas a Enterprise Java Beans. Estos descriptores son usados para instalar aplicaciones Web en servidores EJB.

Existen pasos comúnmente utilizados para escribir descriptores de instalación de aplicaciones *Web*, los cuales están sujetos a variación de acuerdo a las características o propiedades que se deseen incluir dentro de la aplicación *Web*:

1. Crear un archivo de texto y nombrarlo *Web*.xml

2. Crear la sentencia DOCTYPE, todo descriptor de instalación de aplicación *Web* contiene una referencia a la ubicación y versión del documento de definición de tipo (DTD). Aun cuando este elemento hace referencia hacia una URL externa en java.sun.com, todo servidor posee su copia propia de este archivo, para que no sea necesario que el servidor tenga acceso a Internet. No obstante se debe incluir este encabezado.

Se debe utilizar cualquiera de las siguientes dos sentencias DOCTYPE:

- Si se están usando las características de la especificación de *servlets* 2.3, tales como filtros o eventos de aplicación se debe utilizar la siguiente sentencia DOCTYPE.

```
<!DOCTYPE Web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/Web-app_2_3.dtd">
```

- Si no se necesitan las características de la especificación de *servlets* 2.3 se debe usar la siguiente.

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/Web-app_2_2.dtd">
```

3. Crear el cuerpo principal del archivo *Web*.xml. Todas las etiquetas deben ir encerradas dentro de etiquetas <*Web*-app>

```
<Web-app>
    cuerpo del archivo
</Web-app>
```

4. Definir los atributos de instalación en tiempo de instalación, estas etiquetas proporcionan información para las herramientas de instalación o para las herramientas de administración del servidor.

```
<diplay-name>nombre de la aplicación
</display-name>

<description>  descripción de la aplicación
</description>
```

5. Definir los parámetros de contexto, los cuales declaran el contexto de inicialización de la aplicación *Web*. Cada parámetro de contexto se define, utilizando la etiqueta <context-param>

```
<context-param>
   <param-name>
         nombre del parámetro
   </param-name>
```

<param-value>

valor definido para el parámetro

</param-value>

</context-param>

6. Instalación de *servlets*, para instalar un *servlet*, se le debe dar un nombre, especificar el archivo de clase o JSP utilizado para implementar su función así como definir otras propiedades específicas. Se debe listar cada *servlet* de la aplicación dentro de las etiquetas <servlet></servlet>. Después de crear todas las definiciones para cada *servlet*, se deben incluir etiquetas que definan la referencia del *servlet* a una URL específica.

<servlet>

<servlet-name>

contiene el nombre canónico del *servlet*, es decir, el nombre con el que se le puede hacer referencia en cualquier otro lugar en el descriptor.

</servlet-name>

<description>

contiene una descripción de la función que desempeña el *servlet*.

</description>

<servlet-class>

contiene el nombre de la clase del *servlet*

</servlet-class>

ó

<jsp-file>
contiene el nombre del archivo .jsp
</jsp-file>

<init-param> contiene parámetros especificados en la forma pares nombre/valor

<param-name>
nombre del parámetro
</ param-name>

<param-value>
valor del parámetro
<param-value>
<init-param>
</servlet>

7. Mapeo del *servlet* a una dirección URL, una vez que se han declarado los *servlets* o *jsp* deben ser definidas sus referencias hacia direcciones URL para convertirlas en recursos HTTP públicos.

<servlet-mapping>
<servlet-name>
nombre del *servlet* al cual se desea que apunte la dirección URL el cual debe corresponder a un nombre asignado al *servlet* en la etiqueta
</servlet-name>

<url-pattern>
define el patrón de la dirección URL asociada
</url-pattern>
</servlet-mapping>

8. Establecer referencias hacia recursos EJB

<ejb-ref> define referencias hacia objetos EJB
<ejb-ref-name>
contiene el nombre del EJB utilizado en la aplicación,
este nombre es referenciado al árbol JNDI
</ejb-ref-name>
<ejb-ref-type>
tipo del objeto esperado
</ejb-ref-type>
<home>
nombre de la interfaz *home* del EJB
</home>
<remote>
nombre de la interfaz remota del EJB
</remote>
</ejb-ref>

6.6. Descripción de la aplicación

La aplicación consiste en un sitio *Web* que cual permite llevar a cabo la creación, administración y acceso de páginas HTML para los cursos impartidos en la escuela de ciencias y sistemas de la facultad de ingeniería.

Este sitio provee de una interfaz para la creación de forma dinámica y en línea de las páginas de los cursos de manera que la actualización sea fácil de realizar a la vez que mantiene toda la información centralizada para que los estudiantes puedan acceder a las páginas desde un mismo sitio, ya que normalmente las páginas de los cursos son creadas de manera independiente por parte de cada catedrático.

La creación de las páginas se basa en un sencillo esquema, en el cual una página está compuesta de uno o más títulos los cuales a su vez pueden tener asociados uno o más párrafos. Este esquema fue definido de esta forma para que la creación de las páginas sea intuitiva y fácil de realizar.

El sistema cuenta con tres perfiles de usuario, los cuales son:

- Administrador: el usuario que posee este perfil puede llevar a cabo las tareas de administración del sistema, la cual consiste principalmente en la creación, actualización y eliminación de los cursos impartidos en la carrera de ciencias y sistemas.

- Auxiliar/Catedrático: este usuario puede modificar el contenido de las páginas de los cursos que posea asignados, pudiendo agregar, modificar o eliminar los títulos y sus respectivos párrafos.

- Alumno: el usuario alumno puede asignarse cursos, para poder tener acceso a las páginas de estos cursos, a la vez que puede administrar los datos de su cuenta en el sistema.

6.7. Arquitectura de la aplicación

La aplicación está diseñada para que los clientes accedan a ella por medio de una interfaz construida con páginas HTML generadas dinámicamente por el servidor, de acuerdo a las solicitudes que el cliente realice. Estas páginas pueden ser vistas utilizando cualquier navegador de Internet, aunque se recomienda que sean vistas por medio de Microsoft Internet Explorer 5.0 o mayor o bien con Netscape Navigator 5.0 o mayor.

Los pasos que sigue toda solicitud enviada por el cliente es la siguiente:

1. El cliente especifica en el navegador la dirección a la que quiere acceder, que en este caso es la dirección en la que se encuentra configurado el servidor para escuchar las peticiones.

2. Si se trata de la petición inicial, el *servlet* solicita al objeto generador de HTML que construya el HTML de la página inicial, en la cual se le solicita al cliente su usuario, contraseña y el perfil bajo el cual desea ingresar al sistema.

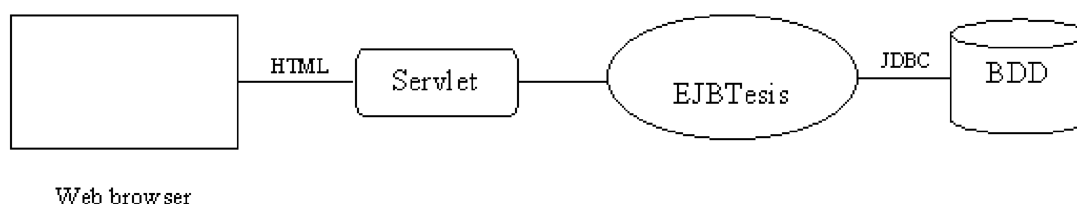
3. Una vez recibida la información solicitada en la pantalla anterior, el *servlet* se comunica con el EJB para que este a su vez llame al objeto de acceso a base de datos el cual determina si la información proporcionada por el usuario es válida. Si la información proporcionada resulta no ser válida, se le indica al usuario con una página de error, brindándole la oportunidad de que vuelva a intentarlo.

4. Una vez que la información ha sido determinada como válida, las peticiones enviadas por el cliente son recibidas por un *servlet* el cual se encarga de llevar a cabo toda la interacción con el cliente.

El *servlet* analizará todas las solicitudes del cliente para determinar qué es lo que está solicitando. Una vez que ha determinado lo que el cliente desea, puede ser que envíe una petición adecuada al objeto EJBTesis para todos aquellos casos en los que la solicitud del cliente involucre acceso a base de datos, o bien que él mismo realice alguna operación sobre información que ya posea en memoria para proporcionar la respuesta adecuada al cliente.

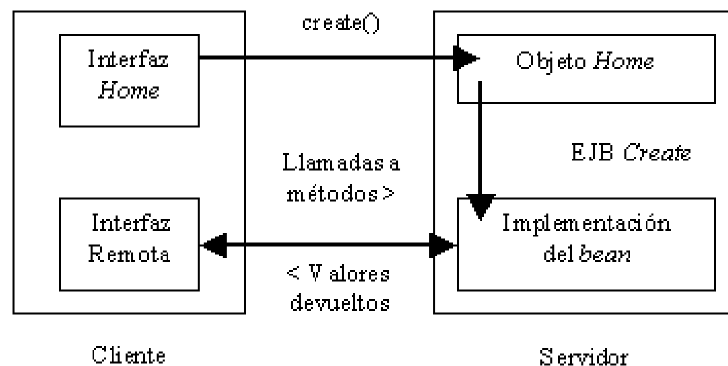
Para todos aquellos casos en los que la solicitud del cliente involucre alguna operación sobre la base de datos, el EJB se encarga de llamar al método adecuado del *bean* de acceso de datos para que, una vez obtenida la respuesta, del *bean* de acceso de datos el EJB traslade el resultado al *servlet* que recibe las peticiones del cliente. De esta manera, el cliente a su vez solicite al *bean* generador de HTML que éste se encargue de generar el código HTML que recibirá como respuesta el cliente en el navegador de internet, tal como se aprecia en la siguiente figura que presenta el esquema general de la aplicación.

Figura 8. Esquema general de la aplicación



A continuación se presenta el esquema general de la interacción entre un cliente y un objeto EJB cuando es creada una nueva instancia de un objeto EJB. Una vez creado el objeto, el cliente puede solicitar los servicios que proporcione el EJB por medio de su interfaz remota.

Figura 9. Esquema general de la creación de un EJB y su interacción con el cliente



6.8. Descripción de las entidades del diagrama de clases

A continuación se describen brevemente las entidades del modelo entidad relación, utilizado en la aplicación:

- Usuario. Contiene la información acerca de los usuarios registrados para utilizar la aplicación, así como su tipo (perfil), el cual determina el acceso a ciertas opciones del total que componen la aplicación.

- Asignación. Sirve para guardar la información acerca de los cursos que posee asignados un usuario determinado.

- Curso. Almacena la información referente a los diferentes cursos que son impartidos en la carrera de ciencias y sistemas para este caso en particular.

- FAQ. Guarda la información de las preguntas más frecuentes formuladas en un curso.

- Respuesta. Contiene la información acerca de las respuestas a las preguntas más frecuentes.

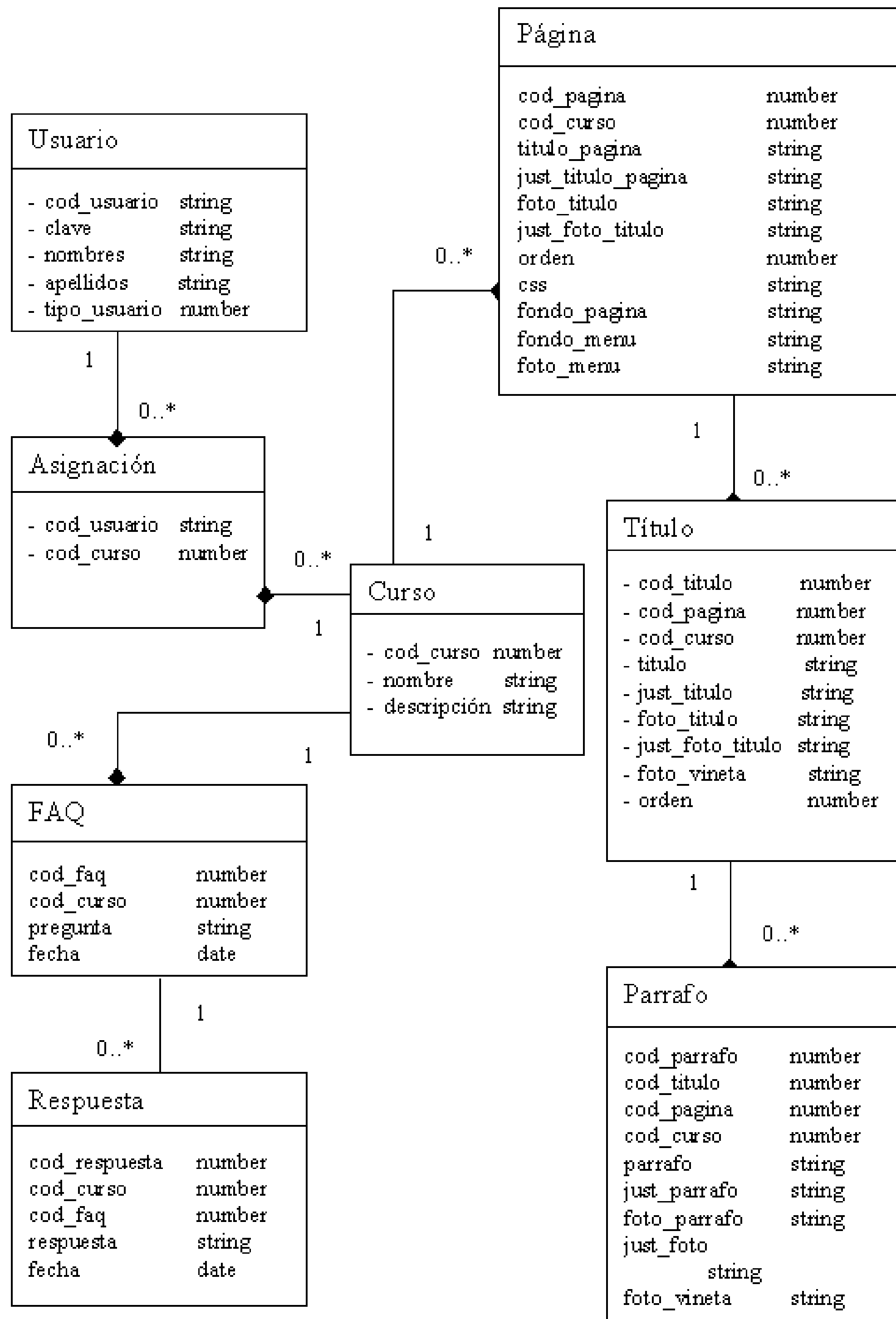
- Página. Guarda la información pertinente a las páginas de los cursos.

- Título. Contiene la información de los títulos asociados a las páginas de los cursos.

- Párrafo. Almacena la información referente a los títulos que pertenecen a las distintas páginas de los cursos.

A continuación se aprecia la figura del modelo entidad relación de la aplicación.

Figura 10. Diagrama de clases de la aplicación de edición de páginas de cursos



6.9. Convenciones de nombramiento

A continuación se muestra la lista de convenciones que se definieron para nombrar las variables dentro del código Java de la aplicación:

- m_nombre variable miembro de una clase Java.
- l_nombre variable local en el ámbito de un método específico.
- p_nombre parámetro enviado de un método.

6.10. Elementos que componen el código de la aplicación

La aplicación se encuentra compuesta por 15 archivos, los cuales poseen las siguientes funciones:

- Dos contienen el código de las dos interfaces EJB la interfaz *home* se encuentra en el archivo SistemaHome.java, mientras que la interfaz remota se encuentra en Sistema.java.

- Un archivo de la clase EJB SistemaBean.java, el cual contiene el código que sen encuentra declarado en la interfaz remota.

- Un archivo almacena la configuración de la conexión a la base de datos ConnectionParams.java.

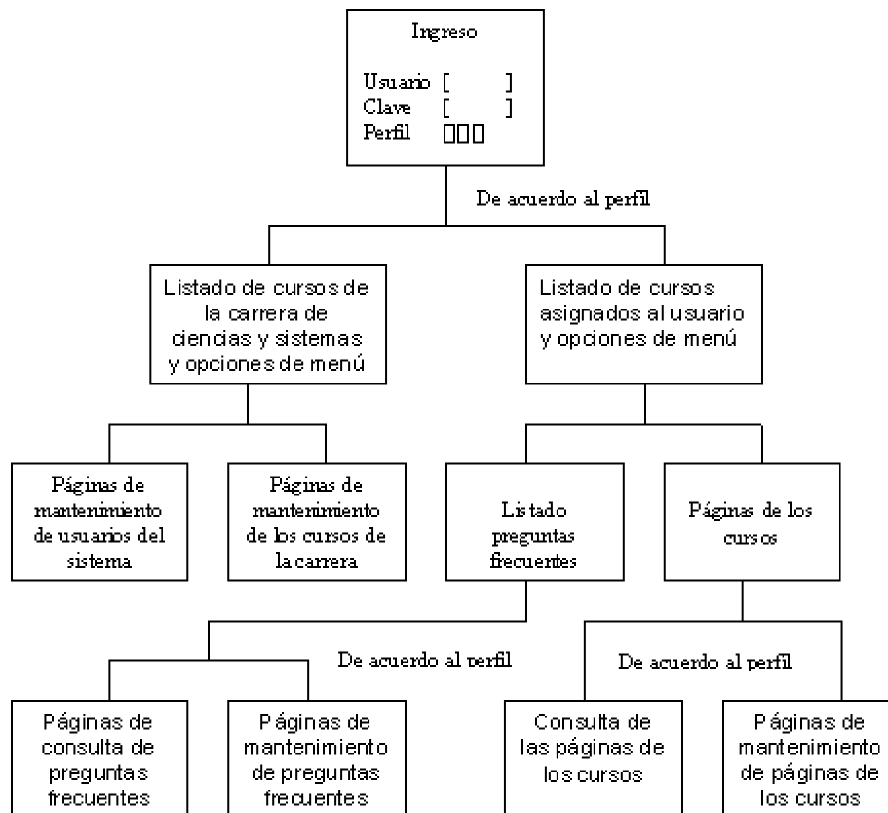
- Ocho archivos de clases para representar y almacenar la información referente a las entidades del modelo, siendo estos archivos ClassTitulo.java, ClassPagina.java, ClassParrafo.java, Class.javaFaq, ClassRespuesta.java, ClassCurso.java, ClassUsuario.java, ClassAsignacion.java.

- Clase para realizar operaciones en la base de datos TesisJDBC.java.

- Un archivo que contiene el código del *servlet* que sirve para llevar a cabo la interacción con el cliente siendo el archivo EJBAplicacionServlet.java.

- Un archivo que posee la clase que genera el código HTML que recibe el cliente como resultado de su interacción con el sistema. Esto se encuentra en el archivo EJBServletHTML.java.

Figura 11. Diagrama de bloques de la aplicación



6.11. Descriptores de instalación

En general, toda aplicación que se instale empaquetada dentro de un servidor de EJB cuenta con tres descriptores de instalación principales que son application.xml, *Web*.xml y Ejb-jar.xml. El archivo application.xml contiene la información para que el servidor pueda instalar la aplicación. El archivo *Web*.xml posee la información sobre el módulo *Web*, mientras que la información sobre el módulo de EJB se encuentra dentro del archivo Ejb-jar.xml.

Figura 12. Contenido del descriptor de instalación *Web*.xml

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
  <display-name>Sistema de Control de Proyectos</display-name>
  <servlet>
    <servlet-name>EJBTesis.web.EJBAplicacionServlet</servlet-name>
    <description>Servlet que llama a los demás Beans</description>
    <servlet-class>EJBTesis.web.EJBAplicacionServlet</servlet-class>
  </servlet>

  <ejb-ref>
    <ejb-ref-name>ejb/SistemaHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>EJBTesis.ejb.SistemaHome</home>
    <remote>EJBTesis.ejb.Sistema</remote>
  </ejb-ref>

  <servlet-mapping>
    <servlet-name>EJBTesis.web.EJBAplicacionServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Figura 13. Contenido del archivo de configuración de instalación Ejb-jar.xml

```
<?xml versión="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <description>Sistema de Control de Proyectos</description>
  <enterprise-beans>
    <session>
      <display-name>Bean Aplicacion</display-name>
      <ejb-name>EJBTesis.ejb.Sistema</ejb-name>
      <home>EJBTesis.ejb.SistemaHome</home>
      <remote>EJBTesis.ejb.Sistema</remote>
      <ejb-class>EJBTesis.ejb.SistemaBean</ejb-class>
      <session-type>Stateful</session-type>
    </session>
  </enterprise-beans>

  <assembly-descriptor>
    <security-role>
      <description>Users</description>
      <role-name>users</role-name>
    </security-role>
  </assembly-descriptor>
</ejb-jar>
```

Figura 14. Contenido del archivo de configuración application.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">

<application>

  <display-name>Sistema de Control de Proyectos</display-name>

  <module>
    <ejb>EJBTesis-ejb.jar</ejb>
  </module>

  <module>
    <web>
      <web-uri>EJBTesis-web.war</web-uri>
      <context-root>/EJBTesis</context-root>
    </web>
  </module>

</application>
```

6.12. Funcionamiento de la aplicación

El objetivo principal de la aplicación es el de proporcionar un medio sencillo para desarrollar páginas de los cursos impartidos en la carrera de ingeniería en Ciencias y Sistemas.

La página inicial del sistema es la que permite que el usuario indique el nombre de usuario, clave y perfil con el que desea ingresar a la aplicación, como se aprecia la figura 16. Cada usuario posee un usuario diferente para cada uno de los perfiles que posea asignados para utilizar la aplicación.

Figura 15. Página de inicio de la aplicación

Una vez que el usuario ha ingresado los anteriores datos y el sistema los ha validado, se le muestra el listado de los cursos que posee asignados. Muestra si ingresó a la aplicación bajo el perfil de estudiante o los cursos que imparte, si lo hizo bajo el perfil de catedrático/auxiliar o el listado completo de los cursos que se encuentran disponibles, y si ingresó bajo el perfil de administrador.

Todos los listados presentan las estructura que a continuación se aprecia en la figura, el cual consiste en un título descriptivo para saber de que listado se trata, así como de dos flechas, que permiten el desplazamiento a través del listado de diez en diez resultados.

Se proporciona un vínculo por cada fila del listado para que el usuario pueda acceder hacia otra pantalla en la que pueda ver o realizar operaciones sobre la información presentada en el listado. Si el usuario cuenta con los privilegios para crear datos relacionados al listado, aparece el botón de nuevo, para tal efecto, como se aprecia en la siguiente figura.

Figura 16. Página del listado de cursos

Universidad de San Carlos

Home | Mi cuenta | Salir

Cursos Asignados
Asignaciones

Listado de Cursos

Curso	Nombre	Descripción
3	Organización Computacional	Electrónica básica
4	Software Avanzado	Ingeniería de software
6	Matemática Básica I	Ecuaciones de primer grado, geometría
9	Matemática Básica Dos	Introducción al Cálculo
10	Matemática Intermedia	Métodos de solución de integrales
11	Matemática Aplicada I	Cálculo en R3
12	Matemática Aplicada II	Ecuaciones diferenciales
30	Fisica Básica	Introducción a la física

Nuevo

Acerca de EJB | Contacto

A continuación se muestra la pantalla a la que se accede si se hace clic sobre el vínculo del código del curso. En esta pantalla se aprecia la información del curso elegido en modo de vista y si el usuario posee privilegios suficientes aparecen en la parte superior e inferior botones para que pueda llevar a cabo las operaciones de edición, eliminación o creación de nuevos registros o cursos en este caso.

Figura 17. Página que muestra los datos de un curso y elección de operación

En la siguiente figura se puede aprecia la página que permite la edición de los datos de un registro del listado de cursos.

Figura 18. Página de modificación de los datos de un curso

Todos los listados presentan la misma lógica de navegación y funcionamiento, a excepción de los listados de páginas y FAQ, pues en el primero, al poseer perfil de catedrático/auxiliar, puede crear páginas mediante el creador de páginas que se muestra en la figura 20, mientras que si el usuario posee perfil de estudiante, puede ver únicamente las páginas de cada curso sin poder modificar el contenido.

Figura 19. Página de edición de páginas

Universidad de San Carlos

Home | Mi cuenta | Salir

Cursos

Microprocesadores

El Microprocesador

El CPU de una computadora contiene la inteligencia de la máquina; es donde se realizan los cálculos y las decisiones. El complejo procedimiento que transforma datos nuevos de entrada en información útil de salida se llama procesamiento. Para llevar a cabo esta transformación, la computadora usa dos componentes: el procesador y la memoria. El procesador es el cerebro de la computadora, la parte que interpreta y ejecuta las instrucciones. El procesador casi siempre se compone de varios circuitos integrados o chips, estos están insertados en tarjetas de circuitos, módulos rígidos rectangulares con circuitos que los unen a otros chips y a otras tarjetas de circuitos. El microprocesador moderno contiene unos 20 millones de transistores y cada chip terminado es el producto de procesos más complicados que los que se utilizaron en el Proyecto Manhattan para construir la bomba atómica. Y no obstante, pese a un proceso de manufactura extraordinariamente refinado, los microchips se producen en volumen a razón de más de 1,000 millones de unidades por año.

Por medio de los botones de función es posible el llevar a cabo modificaciones en la información de las páginas de los cursos. En las tres figuras siguientes se pueden apreciar los mantenimientos de los datos de las páginas, títulos y párrafos respectivamente.

Figura 20. Página para modificar los datos de una página

Universidad de San Carlos

Home | Mi cuenta | Salir

Cursos

Mantenimiento de Páginas

Título de Página: Organización Computacional

Justificación de Título: Izquierda Centrado Derecha

Foto de Título: orga.gif

Orden: 1

CSS:

Fondo Página:

Fondo Menú:

Foto Menú:

Guardar Cancelar

Acerca de EJB | Contacto

Figura 21. Página para modificar los datos de los títulos de una página

Universidad de San Carlos

Home | Mi cuenta | Salir

Cursos

Mantenimiento de Títulos

Título:

Justificación: Izquierda Centrado Derecha

Foto de Título:

Justificación de Título: Izquierda Centrado Derecha

Foto Viñeta:

Orden:

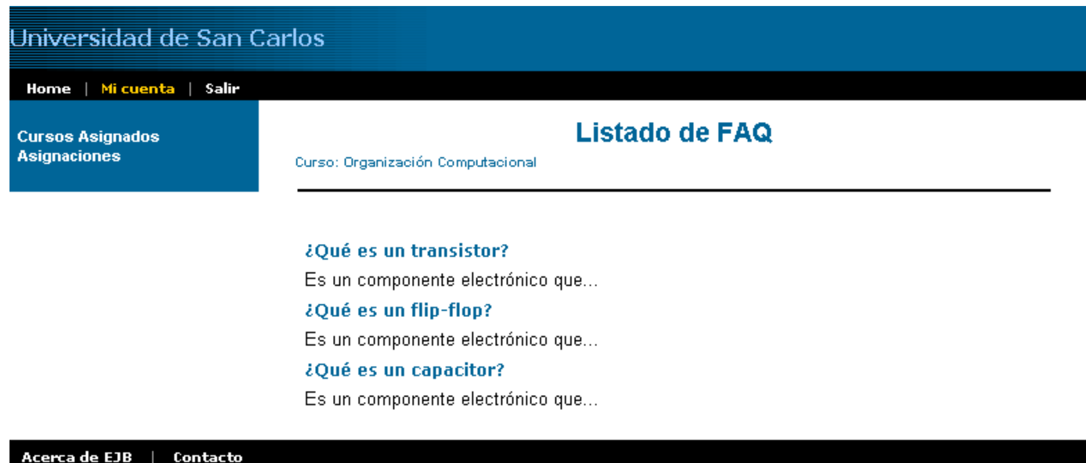
Guardar Cancelar

Acerca de EJB | Contacto

Figura 22. Página para modificar los párrafos de un título pertenecientes a una página

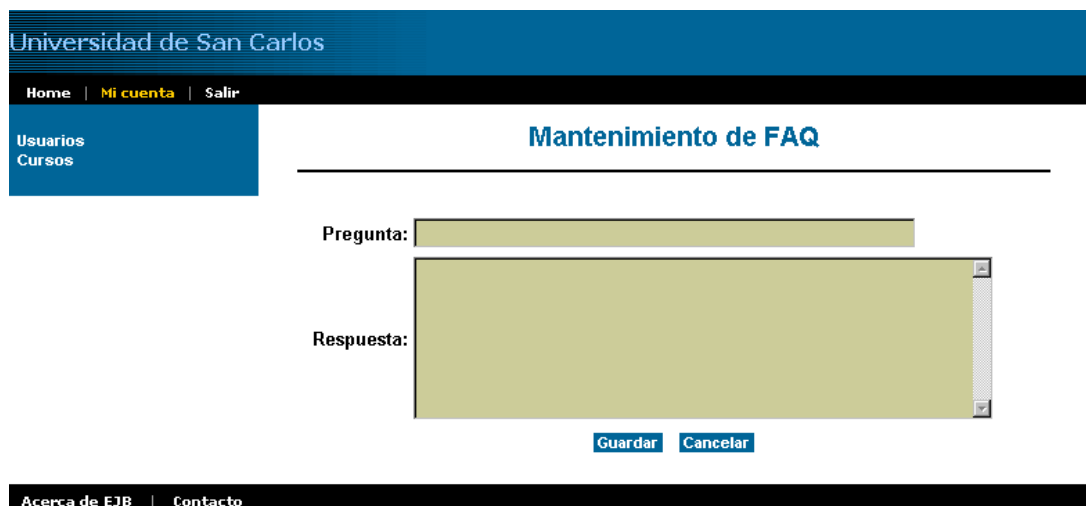
En la figura 24 se aprecia la forma en que se presenta el listado de FAQ, en el cual se listan las preguntas más frecuentes sobre el curso que haya sido elegido por el usuario y, una vez más, sólo un usuario con el perfil de catedrático/auxiliar puede modificar los datos de las FAQ.

Figura 23. Página que lista las FAQ asociadas a un curso



En la siguiente figura se muestra la página que permite la modificación de los datos de las FAQ del curso que se encuentre elegido.

Figura 24. Página de edición de FAQ



CONCLUSIONES

1. De todas las necesidades que los desarrolladores tienen cuando lo hacen de forma distribuida, la que más dificultades presenta en la mayoría de los casos es el lograr comunicar componentes heterogéneos, independientemente de su ubicación física.

2. Las arquitecturas básicas para el desarrollo de aplicaciones distribuidas son las de dos y tres capas, debido a que son las más sencillas de implementar.

3. Las arquitecturas para el desarrollo de aplicaciones distribuidas van desde el uso de dos capas hasta el uso de n capas, aunque para que la arquitectura se pueda considerar que se encuentra del lado del servidor, la mayor cantidad de procesamiento se debe encontrar ubicada en uno o más servidores, de manera que la administración sea más sencilla.

4. La plataforma EJB proporciona una aproximación al desarrollo tradicional de componentes de capa intermedia, una arquitectura adecuada para el desarrollo rápido de aplicaciones. CORBA, por otro lado, ofrece un conjunto más amplio de características de capa intermedia con las cuales poder trabajar. No obstante para usar los servicios CORBA, es necesario programar APIs complejas de capa intermedia, lo cual aumenta la curva de aprendizaje de esta tecnología.

5. El problema de ensamblar componentes heterogéneos es lograr que todos trabajen juntos. EJB define interfaces estándar para que los componentes sean instalables en un contenedor, pero no especifica cómo los componentes deben interactuar.

RECOMENDACIONES

1. Si se va a llevar a cabo el desarrollo de una aplicación nueva, la cual necesite de componentes distribuidos con comunicación entre ellos, lo más aconsejable es desarrollarla con EJB, ya que proporciona todos los servicios de capa comunicaciones y de red necesarios para poder comunicarlos.

2. Cuando se poseen sistemas heredados, los cuales poseen componentes con los que es necesario comunicarse de la forma más directa posible y estos no se encuentran escritos en Java o C++, lo más conveniente es el empleo de CORBA para poder establecer la comunicación, aunque esto haga que el índice de dificultad sea más grande.

3. Si se desea desarrollar aplicaciones distribuidas del lado del servidor, se debe analizar concienzudamente cuál es el grado de distribución de la carga que se necesita para que el desempeño de la aplicación sea lo más eficiente posible.

4. Para las compañías que intentan reducir el tiempo de desarrollo, aumentar la productividad de sus desarrolladores, así como evitar muchos de los problemas asociados con la integración de sistemas, constituye una buena alternativa el hacer uso de servidores de aplicación para resolver sus problemas, ya que estos ofrecen características tales como balanceo de carga, seguridad, *clustering* que pueden hacer que las compañías cumplan sus objetivos.

BIBLIOGRAFÍA

BEA *Weblogic Server*. http://www.weblogic.com (febrero de 2004)

Oracle Technology Network. http://technet.oracle.com (febrero de 2004)

Enterprise JavaBeans. http://java.programacion.net/javabeans/pasos.htm (febrero de 2004)

Linux DCE, CORBA *and* DCOM *Guide*. http://www.gnucash.com (febrero de 2004)

Grupo de administración de objetos. http://www.corba.org (febrero de 2004)

Guía de DCE, CORBA y COM. http://linux.org/linux/corba.html (febrero de 2004)

Web Cornucopia. http://www.execpc.com (febrero de 2004)

Introducción a CORBA con Visibroker y C++ Builder. http://community.borland.com (febrero de 2004)

Programando con CORBA. http://www.blackmagic.com (marzo de 2004)

Jguru. http://www.jguru.com (marzo de 2004)

Conexión de desarrollo Java. http://developer.java.sun.com (marzo de 2004)

Introducción técnica al XML. http://www.xml.com/pub/a/98/10/guide0.html (marzo de 2004)

Tutoriales para utilizar la plataforma Java 2 y la tecnología XML. http://developerlife.com (marzo de 2004)

XML, Java, y el futuro de Internet.
http://www.xml.com/pub/a/w3j/s3.bosak.html (marzo de 2004)

IBM. http://www.ibm.com (marzo de 2004)

APÉNDICE A

Tabla III. Lista de elementos del descriptor de instalación ejb-jar

Elemento	Descripción	Role EJB
Assembly-descriptor	Contiene la información de aplicación y ensamble	Ensamblador de aplicación
Cmp-field	Describe un campo manejado por el contendor	Proveedor del *bean*
Container-transaction	Especifica cómo maneja las transacción el contenedor cuando se realizan invocaciones a los métodos de EJB	Ensamblador de aplicación
description	Proporciona un texto que describe al elemento padre	Proveedor de *bean* o ensamblador de aplicación
ejb-class	Especifica el nombre de la clase EJB	Proveedor de *bean*
ejb-jar	Elemento raíz del descriptor de instalación.	Todos
ejb-link	Es utilizado en el elemento ejb-ref para especificar que una referencia EJB esta vinculada a otro EJB en el archivo EJB JAR	Ensamblador de aplicación
ejb-name	Especifica el nombre del EJB	Ensamblador de aplicación
ejb-ref	Declara una referencia hacia otro *home* EJB	Proveedor de *bean*
ejb-ref-name	especifica el nombre de una referencia EJB	Proveedor de *bean*
ejb-ref-type	Especifica el tipo esperado de un EJB referenciado	Proveedor de *bean*
enterprise-beans	Declara uno o más EJBs	Proveedor de *bean*
Entity	Declara un *bean* de entidad	Proveedor de *bean*
env-entry	Declara parámetros de ambiente EJB	Proveedor de *bean* o ensamblador de aplicación
env-entry-name	Especifica el nombre de un parámetro de ambiente	Proveedor de *bean* o ensamblador de aplicación

env-entry-type	Especifica el tipo Java esperado del parámetro de ambiente que es esperado por el código del EJB	Proveedor de *bean* o ensamblador de aplicación
env-entry-value	Especifica el valor de un parámetro de ambiente	Proveedor de *bean* o ensamblador de aplicación
field-name	Especifica el nombre de un campo manejado por el contenedor	Proveedor de *bean*
home	Especifica el nombre de la interfaz *home* del EJB	Proveedor de *bean*
method	Denota un método de una interfaz *home* o remota o un conjunto de métodos	Ensamblador de aplicación
method-intf	Le permite a un elemento de método el diferenciar entre métodos con el mismo nombre y firma, que están definidos en la interfaz remota y la interfaz *home*	Ensamblador de aplicación
method-name	Especifica el nombre de un método EJB, o el carácter (*), el cual es usado cuando un elemento denota todos los métodos de las interfaces *home* y remota	Ensamblador de aplicación
method-param	Especifica el nombre del tipo Java de un parámetro de un método	Ensamblador de aplicación
method-params	Contiene una lista de los nombres de los tipos Java de los parámetros de los métodos	Ensamblador de aplicación
method-permission	Especifica uno o más roles de seguridad, que tienen permisos para llamar a uno o más métodos EJB	Ensamblador de aplicación
persistence-type	Especifica el tipo de persistencia del *bean* (*bean* o *contained-managed*)	Proveedor de *bean*
prim-key-class	Especifica el nombre de la clase de llave primaria	Proveedor de *bean*
primkey-field	Especifica el nombre del campo de llave primaria para un *bean* con persistencia manejada por el contenedor	Proveedor de *bean*
reentrant	Especifica si el *bean* es reentrante o no	Proveedor de *bean*

remote	Especifica el nombre de la interfaz remota del *bean*	Proveedor de *bean*
res-auth	Especifica si el código EJB se firma programáticamente al administrador de recursos, o si el contenedor firmará al EJB en beneficio de este último.	Proveedor de *bean*
res-ref-name	Especifica el nombre de la referencia de fábrica	Proveedor de *bean*
res-type	Especifica el tipo de fuente de datos que se espera sea implementado por la fuente de datos	Proveedor de *bean*
resource-ref	Declara la referencia de un EJB hacia un recurso externo	Proveedor de *bean*
role-link	Enlaza un *role* de seguridad a un *role* de seguridad definido	Ensamblador de aplicación
role-name	Especifica el nombre de un *role* de seguridad	Ensamblador de aplicación
security-role	Define un *role* de seguridad	Ensamblador de aplicación
security-role-ref	Declara una referencia a un *role* de seguridad en el código EJB	Proveedor de *bean*
Session-type	Especifica el tipo de un *bean* de sesión	Proveedor de *bean*
Session	Declara un *bean* de sesión	Proveedor de *bean*
transaction-type	Especifica el manejo de la transacción (manejada por el *bean* o por el contenedor)	Proveedor de *bean*
Trans-attribute	Especifica cómo debe manejar el contenedor las transacciones al delegar la invocación de un método a un método de negocio de un EJB	Proveedor de *bean*