www.monografias.com

Componentes

INTRODUCCIÓN.

En las primeras épocas de la computación las computadoras operaban independientemente una de otra sin tener comunicación entre ellas. Las aplicaciones de software eran comúnmente desarrolladas para un propósito especifico. Compartir los datos entre sistemas era mínimo y se hacías de una manera muy fácil, se transportaban los medios de almacenamiento (tarjetas, cintas, discos, etc.) de un lado a otro. El próximo paso fue conectar las computadoras a través de una red usando protocolos propietarios, luego los protocolos fueron estandarizados. Luego llego la era de los sistemas abiertos y la integración de sistemas por los cuales un cliente podía elegir varios componentes de hardware de diferentes vendedores e integrarlos para crear una configuración necesaria con costos razonables.

Así siguió el paso de los años. Nuevas técnicas de desarrollo de software se fueron sucediendo una tras otra, desde la programación estructurada y modular hasta la programación orientada a objetos, siempre buscando reducir costos y aumentar la capacidad de reuso. Si bien la programación orientada a objetos fomentó el reuso y permitió reducir costos, no se ha logrado aún el objetivo último: comprar componentes de varios proveedores e integrarlos para formar aplicaciones que a su vez se integren para formar sistemas completos.

Para lograr la integración total de componentes realizados por terceras partes es necesario llegar a un acuerdo común en el que se establezcan los mecanismos necesarios para que esa integración se haga efectiva. Será necesario especificar de manera independiente al lenguaje de programación en el que se desarrolló el componente cuáles son sus puntos de acceso (funciones), luego será necesario establecer los mecanismos de comunicación entre componentes, que podrían estar ejecutándose en una máquina remota.

En este sentido, y buscando satisfacer esa necesidad de mecanismos estándar e interfaces abiertas, son tres los esfuerzos que más han sobresalido. Por un lado, Microsoft ha introducido en el mercado sus tecnologías COM, DCOM y COM+. Otro participante es Sun Microsystems, que ha presentado Java Beans. El tercero es el Object Management Group, un consorcio integrado por varias industrias importantes, que ha desarrollado CORBA (Common Request Broker Architecture).

1. COM / DCOM

Microsoft Distributed COM (DCOM) extiende COM (Component Object Model) para soportar comunicación entre objetos en ordenadores distintos, en una LAN, WAN, o incluso en Internet. Con DCOM una aplicación puede ser distribuida en lugares que dan más sentido al cliente y a la aplicación.

Como DCOM es una evolución lógica de COM, se pueden utilizar los componentes creados en aplicaciones basadas en COM, y trasladarlas a entornos distribuidos. DCOM maneja detales muy bajos de protocolos de red, por lo que uno se puede centrar en la realidad de los negocios: proporcionar soluciones a clientes.

Atualmente DCOM viene con los sistemas operativos Windows 2000, NT, 98 y también está disponible una versión para Windows 95 en la página de Microsoft. También hay una

implementación de DCOM para Apple Macintosh y se está trabajando en implementaciones para plataformas UNIX como Solaris.

1.1 La arquitectura DCOM

DCOM es una extensión de COM, y éste define como los componentes y sus clientes interactuan entre sí. Esta interacción es definida de tal manera que el cliente y el componente puede conectar sin la necesidad de un sistema intermedio. El cliente llama a los métodos del componente sin tener que preocuparse de niveles más complejos.

En los actuales sistemas operativos, los procesos están separados unos de otros. Un cliente que necesita comunicarse con un componente en otro proceso no puede llamarlo directamente, y tendrá que utilizar alguna forma de comunicación entre procesos que proporcione el sistema operativo. COM proporciona este tipo de comunicación de una forma transparente: intercepta las llamadas del cliente y las reenvía al componente que está en otro proceso.

Cuando el cliente y el componente residen en distintas máquinas, DCOM simplemente reemplaza la comunicación entre procesos locales por un protocolo de red. Ni el cliente ni el componente se enteran de que la unión que los conecta es ahora un poco más grande.

Las librería de COM proporcionan servicios orientados a objetos a los clientes y componentes, y utilizan RPC y un proveedor de seguridad para generar paquetes de red estándar que entienda el protocolo estándar de DCOM.

1.2 Los Componentes y su reutilización

Muchas aplicaciones distribuidas no están desarrolladas

Al existir infraestructuras de hardware, software, componentes, al igual que herramientas, se necesita poder integrarlas y nivelarlas para reducir el desarrollo y el tiempo de trabajo y coste. DCOM toma ventaja de forma directa y transparente de los componentes COM y herramientas ya existentes. Un gran mercado de todos los componentes disponibles haría posible reducir el tiempo de desarrollo integrando soluciones estandarizadas en las aplicaciones de usuario. Muchos desarrolladores están familiarizados con COM y pueden aplicar fácilmente sus conocimientos a las aplicaciones distribuidas basadas en DCOM.

Cualquier componente que sea desarrollado como una parte de una aplicación distribuida es un candidato para ser reutilizado. Organizando los procesos de desarrollo alrededor del paradigma de los componentes permite continuar aumentando el nivel de funcionalidad en las nuevas aplicaciones y reducir el tiempo de desarrollo.

Diseñando para COM y DCOM se asegura que los componentes creados serán útiles ahora y en el futuro.

1.3 Independencia de la localización

Cuando se comienza a implementar una aplicación distribuida en una red reak, aparecen distintos conflictos en el diseño:

- Los componentes que interactuan más a menudo deberían estar localizados más cerca.
- Algunos componentes solo pueden ser ejecutados en máquinas específicas o lugares específicos.
- Los componentes más pequeños aumentan la flexibilidad, pero aumentan el tráfico de red.
- Los componentes grandes reducen el tráfico de red, pero también reducen la flexibilidad.

Con DCOM, estos temas críticos de diseño pueden ser tratados se forma bastante sencilla, ya que estos detalles no se especifican en el código fuente. DCOM olvida completamente la localización de los componentes, ya esté en el mismo proceso que el cliente o en una máquina en cualquier lugar del mundo. En cualquier caso, la forma en la que el cliente se conecta a un componente y llama a los métodos de éste es identica. No es solo que DCOM no necesite cambios en el código fuente, sino que además no necesita que el programa sea recompilado. Una simple reconfiguración cambia la forma en la que los componentes se conectan entre sí.

La independencia de localización en DCOM simplifica enormemente las tarea de los componentes de aplicaciones distribuidas para alcanzar un nivel de funcionamiento óptimo. Supongamos, por ejemplo, que cierto componente debe ser localizado en una máquina específica en un lugar

determinado. Si la aplicación tiene numerosos componentes pequeños, se puede reducir la carga de la red situándolos en la misma LAN, en la misma máquina, o incluso en el mismo proceso. Si la aplicación está compuesta por un pequeño número de grandes componentes, la carga de red es menor y no es un problema, por tanto se pueden poner en las máquinas más rápidas disponibles independientemente de donde esten situadas.
Con la independencia de localización de DCOM, la aplicación puede combinar componentes relacionados en máquinas "cercanas" entre si, en una sola máquina o incluso en el mismo proceso. Incluso si un gran número de pequeños componentes implementan la funcionalidad de un gran módulo lógico, podrán interactuar eficientemente entre ellos.

1.4 Independencia del lenguaje de programación

Una cuestión importante durante el diseño e implementación de una aplicación distribuida es la elección del lenguaje o herramienta de programación. La elección es generalmente un termino medio entre el coste de desarrollo, la experiencia disponible y la funcionalidad. Como una extensión de COM, DCOM es completamente independiente del lenguaje. Virtualmentem cualquier lenguaje puede ser utilizado para crear componentes COM, y estos componentes puede ser utilizado por muchos más lenguajes y herramientas. Java, Microsoft Visual C++, Microsoft Visual Basic, Delphi, PowerBuilder, y Micro Focus COBOL interactuan perfectamente con DCOM.
Con la independencia de lenguaje de DCOM, los desarrolladores de aplicaciones puede elegir las herramientas y lenguajes con los que estén más familiarizados. La independencia del lenguaje permite crear componentes en lenguajes de nivel superior como Microsoft Visual Basic, y después reimplementarlos en distintos lenguajes como C++ o Java, que permiten tomar ventaja de características avanzadas como multihilo.

1.5 Independencia del protocolo

Muchas aplicaciones distribuidas tienen que ser integradas en la infraestructura de una red existente. Necesitar un protocolo específico de red, obligará a mejorar todos los cliente, lo que es inaceptable en muchas situaciones. Los desarrolladores de aplicaciones tienen que tener cuidado de mantener la aplicación lo más independiente posible de la infraestructura de la red.
DCOM proporciona esta transparencia: DCOM puede utilizar cualquier protocolo de transporte, como TCP/IP, UDP, IPX/SPX y NetBIOS. DCOM proporciona un marco de seguridad a todos estos protocolos.
Los desarrolladores pueden simplemente utilizar las características proporcionadas por DCOM y asegurar que sus aplicaciones son completamente independiente del protocolo.

2. CORBA

CORBA es un Middeware o marco de trabajo estándar y abierto de objetos distribuidos que permite a los componentes en la red interoperar en un ambiente común sin importar el lenguaje de desarrollo, sistema operacional, tipo de red, etc. En esta arquitectura, los métodos de un objeto remoto pueden ser invocados “transparentemente” en un ambiente distribuido y heterogéneo a través de un ORB (Object Request Broker). Además del objetivo básico de ejecutar simplemente métodos en objetos remotos, CORBA adiciona un conjunto de servicios que amplían las potencialidades de éstos objetos y conforman una infraestructura sólida para el desarrollo de aplicaciones críticas de negocio.
CORBA es la respuesta del “Grupo de Gestión de Objetos” (Object Management Group – OMG) a la necesidad de interoperabilidad ante la gran proliferación de productos hardware y software, es decir permitir a una aplicación comunicarse con otra sin importar el tipo de red, protocolo, sistema operacional o lenguaje de desarrollo.
CORBA automatiza muchas tareas comunes y “pesadas” de programación de redes tales como registro, localización y activación de objetos; manejo de errores y excepciones; codificación y decodificación de parámetros, y protocolo de transmisión.
En un ambiente CORBA, cada *Implementación de Objeto*, define bien su *Interface* a través una especificación normalizada conocida como IDL (Interface Definition Language) a través de la cual

en forma *Estática* (en el momento de compilación) o en forma *Dinámica* (en el momento de ejecución) un *Cliente* que requiera el servicio de una *Implementación de Objeto*, puede ser ejecutada. Las invocaciones a métodos remotos son enviados por los clientes llamando objetos locales llamados "Stubs" (generados por un compilador de IDL - Estático), el cual intercepta dichas invocaciones y continua el proceso de llevar y retornar automáticamente dicha invocación. La *Implementación del objeto*, no tiene que conocer el mecanismo por el cual un *Cliente* le ha invocado un servicio.
Cuando el *Cliente* y una *Implementación de Objeto* están distribuidos por una red, ellos usan el protocolo GIOP/IIOP suministrado por la arquitectura para lograr la comunicación.
La forma de cómo una *Implementación de Objeto* (desarrollada por un programador de aplicaciones) se conecta a un ORB, es a través de un *Adaptador de Objetos*. Este adaptador recibe las peticiones por la red e invoca los servicios a la implementación correspondiente.
Actualmente CORBA ya se han resuelto los problemas fundamentales de interoperabilidad y comunicación entre objetos [Omg95a] [Omg99a] [Vin98] y se han definido y especificado un conjunto de servicios comunes requeridos para la construcción de las aplicaciones [Omg95b] [Omg95c] [Omg98c], pero donde hay gran actividad es en la especificación de objetos comunes por dominio de aplicación o conocidas en CORBA como Interfaces de Dominio. Allí se trabajan en áreas como Telecomunicaciones, Medicina, Finanzas, Manufactura, etc. [Omg98a] [Omg98b] [Omg99b] [Omg99c].
CORBA esta fundamentado en dos modelos: Un modelo de Objetos, el cual agrega todas las características de Orientación por Objetos como Tipos de Datos, Abstracción, Polimorfismo y Herencia y un modelo de referencia o arquitectura conocida como OMA (Object Management Architecture).

2.1 Servicios Middleware

Para resolver los problemas inherentes a sistemas heterogéneos y distribuidos, que dificultan la implementación de verdaderas aplicaciones empresariales, los proveedores de software están ofreciendo interfaces de programación y protocolos estándares. Estos servicios se denominan usualmente *servicios middleware*, porque se encuentran en una capa intermedia, por encima del sistema operativo y del software de red y por debajo de las aplicaciones de los usuarios finales. En esta sección se describen las características principales de los servicios middleware [2].
Un *servicio middleware* es un servicio de propósito general que se ubica entre plataformas y aplicaciones. Por plataformas se entiende el conjunto de servicios de bajo nivel ofrecidos por la arquitectura de un procesador y el conjunto de API´s de un sistema operativo. Como ejemplos de plataformas se pueden citar: Intel x86 y Win-32, SunSPARCStation y Solaris, IBM RS/6000 y AIX, entre otros.
Un servicio middleware está definido por las API´s y el conjunto de protocolos que ofrece. Pueden existir varias implementaciones que satisfagan las especificaciones de protocolos e interfaces. Los componentes middleware se distinguen de aplicaciones finales y de servicios de plataformas específicas por cuatro importantes propiedades:

- Son independientes de las aplicaciones y de las industrias para las que éstas se desarrollan.
- Se pueden ejecutar en múltiples plataformas.
- Se encuentran distribuidos.
- Soportan interfaces y protocolos estándar.

Debido al importante rol que juegan una interfaz estándar en la portabilidad de aplicaciones y un protocolo estándar en la interoperabilidad entre aplicaciones, varios esfuerzos se han realizado para establecer un estándar que pueda ser reconocido por los mayores participantes en la industria del software. Algunos de ellos han alcanzado instituciones como ANSI e ISO; otros han sido propuestos por consorcios de industrias como ser la Open Software Foundation y el Object Management Group y otros han sido impulsados por industrias con una gran cuota de mercado. Este último es el caso de Microsoft con su Windows Open Services Architecture.

CORBA es el estándar propuesto por el OMG. EL OMG fue fundado en 1989 y es el más grande consorcio de industrias de la actualidad, con más de 700 compañías que son miembros del grupo. Opera como una organización no comercial sin fines de lucro, cuyo objetivo es lograr establecer *todos* los estándares necesarios para lograr interoperabilidad en todos los niveles de un mercado de objetos.

Originalmente los esfuerzos de la OMG se centraron en resolver un problema fundamental: cómo lograr que sistemas distribuidos orientados a objetos implementados en diferentes lenguajes y ejecutándose en diferentes plataformas interactúen entre ellos. Más allá de los problemas planteados por la computación distribuida, problemas más simples como la falta de comunicación entre dos sistemas generados por compiladores de C++ distintos que corren en la misma plataforma frenaron los esfuerzos de integración no bien comenzados. Para opacar aún más el escenario, distintos lenguajes de programación ofrecen modelos de objetos distintos. Los primeros años de la OMG estuvieron dedicados a resolver los principales problemas de *cableado*. Como resultado se obtuvo la primer versión del Common Object Request Broker, publicado en 1991. Hoy en día, el último estándar aprobado de CORBA está por la versión 2.3, y la versión 3.0 está a punto de ser lanzada.

Desde sus principios, el objetivo de CORBA fue permitir la interconexión abierta de distintos lenguajes, implementaciones y plataformas. De esta forma, CORBA cumple con las cuatro propiedades enumeradas como deseables de los servicios middleware. Para lograr estos objetivos, la OMG decidió no establecer estándares binarios (como es el caso de COM); todo está estandarizado para permitir implementaciones diferentes y permitir que aquellos proveedores que desarrollan CORBA pueden ofrecer valor agregado. La contrapartida es la imposibilidad de interactuar de manera eficiente a nivel binario. Todo producto que sea compatible con CORBA debe utilizar los costosos protocolos de alto nivel.

CORBA está constituido esencialmente de tres partes: un conjunto de interfaces de invocación, el ORB (object request broker) y un conjunto de adaptadores de objetos (objects adapters). CORBA va más allá de simples servicios middleware, provee una infraestructura (framework) para construir aplicaciones orientadas a objetos. Las interfaces definen los servicios que prestan los objetos, el ORB se encarga de la localización e invocación de los métodos sobre los objetos y el object adapter es quien liga la implementación del objeto con el ORB.

Para que las interfaces de invocación y los adaptadores de objetos funcionen correctamente, se deben cumplir dos requisitos importantes. En primer lugar, las interfaces de los objetos deben describirse en un lenguaje común. En segundo lugar, todos los lenguajes en los que se quieran implementar los objetos deben proveer un *mapeo* entre los elementos propios del lenguaje de programación y el lenguaje común. La primer condición permite generalizar los mecanismos de pasaje de parámetros (marshaling y unmarshaling). La segunda permite relacionar llamadas de o a un lenguaje en particular con el lenguaje de especificación común. Este lenguaje común fue una parte esencial de CORBA desde sus orígenes y es conocido como el OMG IDL: Interface Definition Language. Existen mapeos del OMG IDL a C, C++, Java y Smalltalk,

2.2 Arquitectura de CORBA

CORBA define una arquitectura para los objetos distribuidos. El paradigma básico de CORBA es el de un pedido servicios de un objeto distribuido. Todo definido por el OMG está en términos de este paradigma básico.

Los servicios que un objeto proporciona son dados por su interfaz . Los interfaces se definen en la lengua de la definición de interfaz de OMG (IDL). Los objetos distribuidos son identificados por las referencias del objeto, que son mecanografiadas por los interfaces de IDL.

La figura abajo representa gráficamente una petición. Un cliente lleva a cabo una referencia del objeto a un objeto distribuido. La referencia del objeto es mecanografiada por un interfaz. En la figura debajo del objeto la referencia es mecanografiada por Conejo interfaz. El corredor de la petición del objeto, u ORB, entrega la petición al objeto y vuelve cualquier resultado al cliente. En la

figura, a salto la petición vuelve una referencia del objeto mecanografiada por AnotherObject interfaz.

2.3 CORBA como estándar para los objetos distribuidos

Una de las metas de la especificación de CORBA es que los clientes y las puestas en práctica del objeto son portables. La especificación de CORBA define el interfaz de un programador del uso (API) para los clientes de un objeto distribuido así como un API para la puesta en práctica de un objeto de CORBA. Esto significa que el código escrito para un producto de CORBA del vendedor podría, con un mínimo de esfuerzo, ser reescrito para trabajar con el producto de un diverso vendedor. Sin embargo, la realidad de los productos de CORBA en el mercado es hoy que los clientes de CORBA son portables pero las puestas en práctica del objeto necesitan alguna reanudación virar hacia el lado de babor a partir de un producto de CORBA a otro.

CORBA 2,0 agregó interoperabilidad como meta en la especificaciones. El detalle de En, CORBA 2,0 definen el protocolo del un de la red, llamado IIOP (el Internet Enterrar-orbe protocolo), permite del que un clientes usando un productos de CORBA del cualquier vendedor para comunicarse hacen trampas el los objetos usando un producto de CORBA del vendedor de otro de cualquier. El trabaja de IIOP un del del través Internet, el o más exacto, un través de la cualquier puesta en práctica de TCP/IP.

Interoperability es más importante en un sistema distribuído que la portabilidad. IIOP se usa en otros sistemas que no intentan proporcionar el API de CORBA ni siquiera. En particular, IIOP se usa como el protocolo de transporte para una versión de Java RMI (para que llamó "RMI encima de IIOP"). Desde que EJB se define por lo que se refiere a RMI, puede usar IIOP también. Los varios servidores de la aplicación disponible en el uso del mercado IIOP pero no expone el API de CORBA entero. Porque ellos todos usan IIOP, programas escritos entre sí al interoperate de la lata de estos API diferente y con programas escritos al API de CORBA.

2.4 CORBA y el desarrollo basado en componentes

Como se mencionó anteriormente, el desarrollo basado en componentes que se puedan comprar y poner en marcha sin mayores dificultades (Plug & Play) es una meta a alcanzar que facilitaría el reúso de software. En esta sección se hace una pequeña introducción al desarrollo basado en componentes y el rol que le compete a CORBA en este ámbito.

Muy frecuentemente el término componente se utiliza sin precisar su significado, dando por sentado que todos lo conocen. Pero no siempre es así, y al utilizar el término componente puede suceder que no siempre se esé hablando de lo mismo. Es deseable entonces comenzar con una definición de componente.

Un componente ha sido definido en la European Conference on Object Oriented Programming (ECOOP) de 1996 como una “una unidad de composición con interfaces contractuales especificadas y dependencias de contexto explícitas.”[7] Un componente de software puede ser desarrollado independientemente y utilizado por terceras partes para integrarlo mediante composición a sus sistemas.”[1] Los componentes son para crear software utilizando composición, por eso es esencial que sean independientes y que se presenten en formato binario, permitiendo así distintos vendedores e integración robusta.

Para que un componente pueda ser integrado por terceras partes en sus sistemas, éste deber ser suficientemente autocontenido y debe proveer una especificación de lo que requiere y provee. En otras palabras, los componentes deben encapsular su implementación e interactuar con otros componentes a través de interfaces bien definidas.

Un componente no es un objeto. A diferencia de los objetos, los componentes no tienen estado. Esto quiere decir que un componente no puede distinguirse de una copia de sí mismo. Un componente puede tomar la forma de un archivo ejecutable o una biblioteca dinámica que usualmente cobra vida a través de objetos, pero no es este un requisito indispensable. De hecho, los primeros componentes conocidos (aunque en su momento no se los haya definido así) fueron las bibliotecas de prodecimientos. Sin embargo, los objetos, con sus características de encapsulación y polimorfismo, facilitan la construcción e integración de componentes.

Y al hablar de objetos vale la pena distinguir aquí los objetos de las clases. Una clase es una definición de propiedades y funcionalidades ha ser provistas por los objetos. A partir de una clase es posible instanciar objetos. Los componentes pueden contener una o más clases y serán los clientes de los componentes quienes soliciten la creación de las instancias de estas clases.
Como escenario de ejemplo para mostrar cómo sería la construcción de software basado en componentes, se podría pensar en una librería. Una librería necesita un sistema que le permita llevar el stock de sus libros, contabilizar sus ventas y gestionar los pedidos a sus proveedores. Quien estuviera a cargo de este proyecto de desarrollo podría adquirir un componente de facturación que provee las siguientes clases (entre otras):

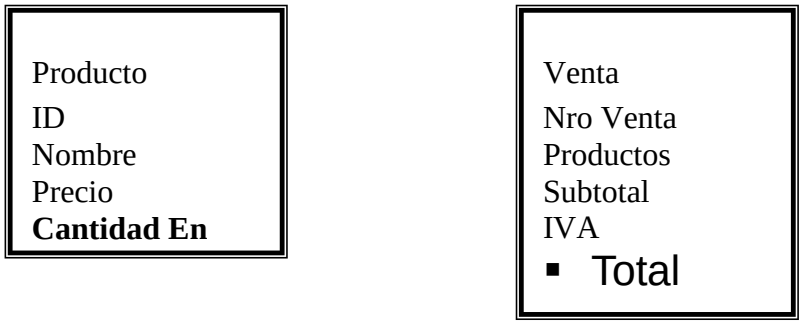


Figura 1: Clases provistas por el componente Facturación

El fabricante del componente de Facturación no sabe de antemano en que ambientes se va a utilizar su componente. Por lo tanto, decide no ofrecer una implementación de la clase producto. Sólo provee provee su interfaz para que quién utilice el componente sea quien finalmente la implemente de acuerdo a sus necesidades. Sí en cambio ofrece la implementación completa de la clase venta. La especificación de la clase venta podría indicar que:

- *Nro de Venta* es un número entero que se asigna automáticamente al crearse una nueva instancia de un objeto Venta.
- *Productos* es un conjunto de productos (existirá en otro lado la definición de conjunto con sus operaciones de inserción y eliminación). Al agregar un producto a la venta se debe especificar su cantidad. Esta cantidad será deducida de la cantidad en stock de ese producto.
- Etc.

El programador del sistema para la librería decide crear una clase Libro que implementa la interfaz de producto (un objeto Libro se comportará como si fuera un objeto producto cuando sea necesario), y así podrá crear ventas de libros.
Si bien este es un ejemplo simple, es un escenario al que se desearía poder llegar con el desarrollo basado en componentes.
Hasta aquí todo suena muy lindo. Pero tomando el lugar del programador que debe integrar el componente a su aplicación, surgen algunas incógnitas que debería poder resolver. El fabricante del componente solamente ha entregado un archivo ejecutable que se debe iniciar en la máquina en la que se ejecuta la aplicación y la interfaz definida en un lenguaje de definición de interfaces (IDL) estándar. El programador desea ahora:

- **Mapear la interfaz***:* ¿Cómo hace el programador para mapear las clases que el componente provee al lenguaje en que realizará su implementación final? Seguramente el lenguaje de programación elegido provee mecanismos para definir clases, pero es necesario que la definición que se haga de la clase en ese lenguaje corresponda a la definición que dio el fabricante del componente.
- **Crear objetos**: ¿Cómo hace el programador para crear una instancia del objeto Venta? Es necesario que exista un mecanismo para indicar al componente que cree una instancia del

objeto Venta. Una vez creada la instancia ¿Cómo se logra acceder a sus propiedades o métodos?

- **Transparencia**: El componente sería de poca utilidad si su utilización no fuera transparente. Si para cada llamada al componente el programador debiera utilizar un servicio del sistema operativo de llamada entre procesos (IPC), o peor aun si el componente es remoto, un servicio de llamada remota a procedimientos (RPC), está claro que dejaría el componente de lado pues es más trabajo utilizarlo que hacer un programa desde cero.

Estas son sólo algunas de las cuestiones que el programador debería poder resolver para poder utilizar el componente. En el caso de que el programador llegara a comprar otro componente, es seguro que desea que los mecanismos de utilización sean uniformes para no tener que resolverlas nuevamente. Los servicios middleware que provee CORBA buscan resuelven estos problemas.

2.5 CORBA Services

OMA esta construida sobre un fundamento y arquitectura CORBA que desarrolla la visión de la OMG de componentes de software plug-and-play.

Los CORBA Services especifican servicios básicos casi todos los objetos necesitan.

Para cada componente de OMA, la OMG provee una especificación formal descrita en OMG IDL (como invocar cada operación dentro de un objeto) y su semántica en lenguaje ingles (que hace cada operación y las reglas de comportamiento). Un proveedor no tiene que suministrar obligatoriamente ningún servicio adicional al ORB, pero si este lo ofrece, deberá esta de acuerdo a la especificación que para este servicio tiene la OMG.

Los CORBA Services proveen servicios a nivel de aplicación fundamentales para las aplicaciones orientadas por objetos y componentes en entornos distribuidos. La OMG ha definido alrededor de 15 servicios de objetos. Los cuales son:

- Nombres
- Trader
- Notificación
- Eventos
- Transacciones
- Seguridad
- Ciclo de vida
- Propiedades
- Persistencia
- Consulta
- Relaciones
- Concurrencia
- Externalización
- Licenciamiento
- Tiempo
- Colección

De éstos 15 se destacan los siguientes servicios clave:

- Acceso a referencias de objetos a través de la red, soportada por el servicio de Nombres y de Trader.
- Notificación de eventos importantes o cambios de estado en un objeto, soportado por el servicio de Eventos y de Notificación.
- Soporte para semántica transaccional (two-phase commit y rollback) soportado por el servicio de Transacciones.
- Soporte para seguridad, soportada por el servicio de Seguridad.

Nombres: Permite a componentes descubrir otros componentes en un ambiente distribuido, básicamente es un servicio de localización que asocia identificadores a manejadores que proveen una forma de contactar el componente deseado en un sistema distribuidos. Este asocia nombres - organizados jerárquicamente - a objetos.

Ciclo de vida: Básicamente es un servicio de configuración, define servicios y convenciones para crear, borrar, copiar y mover componentes en un sistema distribuido.

Eventos: Implementa un modelo de comunicación desacoplado, basado en el paradigma publicación/suscripción. En este modelo, uno o más publicadores pueden enviar mensajes relacionados con un tópico especifico, mientras que un grupo de suscriptores reciben los mensajes asincrónicamente. Con este mecanismos, los publicadores pueden generar evento sin tener que

conocer la identificación de sus consumidores y viceversa. Hay dos acercamientos, modelo Push y modelo Pull, en el modelo Push los publicadores toman la iniciativa de iniciar la comunicación, en el modelo Pull, los suscriptores requieren eventos de los publicadores.
Transacciones: gestiona interacciones entre objetos distribuidos estableciendo puntos de Commit y delimitación de transacciones. Este servicio soporta varios modelos y permite interoperabilidad entre diferentes arquitecturas de red y modelos de programación.
Seguridad: Controla la identificación de los elementos del sistema distribuido (usuarios, objetos, componentes, etc) que permite verificar que un Cliente esta autorizado a acceder los servicios de una Impementación remota. Adicionalmente permite la comunicación segura sobre enlaces de comunicación inseguros, ofreciendo confidencialidad e integridad de la información transmitida.
Tiempo: Suministra información del tiempo y permite la definición de llamadas periódicas.
Licenciamiento: Controla la utilización de objetos específicos, inhibiendo uso inadecuado de derechos y propiedad intelectual.
Propiedades: provee la habilidad de dinámicamente asociar propiedades a los objetos los cuales pueden ser consultados o modificados por otros elementos.
Relaciones: Permite la definición del papel ejecutado por objetos CORBA en una relación.
Consulta: Permite a los usuarios y objetos invocar operaciones de consulta sobre colecciones de objetos.
Persistencia: provee mecanismos para retener y mantener el estado de objetos persistentes.
Concurrencia: permite la coordinación de múltiples accesos a recursos compartidos a través de la provisión de varios modelos de locks y permite resolución flexible de conflictos.
Externalización: define convenciones y protocolos para representar el estado de información relacionado a un objeto en la forma de una secuencia de datos, permitiendo a esta información ser almacenada o transferida a otras localizaciones.

2.6 CORBA Facilities Horizontales

Esta facilidades CORBA tanto horizontales como verticales, son diseñadas para completar la arquitectura entre los Servicios básicos de CORBA y las aplicaciones especificas de industria. Con una arquitectura completa, las compañía compartirán datos a nivel de aplicación y funcionalidades para la integración de diferentes sistemas de información. Las facilidades representan un punto medio entre una aplicación particular y los servicios y ORB.
La OMG ha definido como Facilidades horizontales las siguientes:

- Interface de usuario
- Administración de información
- Administración de sistemas
- Administración de tareas

Hoy en día estas especificaciones han sido adheridas a ORBOS (ORB y Servicios) y ya no esta como un grupo aparte.
Específicamente a nivel de facilidades verticales la OMG ha definido las siguientes:

- Especificación para la administración de sistemas XCMF.
- Facilidad para colar de impresión.

2.7 CORBA Facilities Verticales o de Dominio

La potencialidad que representa el lenguaje IDL para la especificación de un objeto u componente ha permitido trabajar en la normalización de intereses comunes para un sector de mercado particular y que una vez se llegue a un acuerdo en cuanto a estas especificaciones, sería estándar dentro de este mercado.
Para esto se ha creado el Domain Technology Committee (DTC) y esta a su vez esta organizada en una serie de Domain Task Forces (DTF) los cuales escriben documentos de requerimientos (RFI y RFP) para nuevas especificaciones y evalúan y recomiendan especificaciones candidatas. Basados en las recomendaciones de los DTF, el DTC conduce un proceso formal de votación para asegurar que cumple todos los requerimientos del sector y no solo de quien haya propuesto.

Posteriormente estas recomendaciones requieren ser enviadas a la Junta de Directores de la OMG para hacerla una especificación oficial. Actualmente hay 8 DTF:

- Objetos de negocio
- Finanzas y seguros
- Comercio Electrónico
- Manufactura
- Salud o Medicina
- Telecomunicaciones
- Transportes
- Investigación de ciencias de la vida

También bajo la OMG pero que no tienen DTF se encuentran dos Grupos de Interés Especial:

- Utilities (principalmente energía eléctrica)
- Estadística

Seis especificaciones ya han sido adoptadas oficialmente como estándares de dominio vertical, ellos son:

- Facilidad Currency del DTF de Finanzas
- Un conjunto de Habilitadores para la administración de datos de productos, del DTF de manufactura.
- Servicio de Identificación de Personas (PIDS) de CORBAMed
- Servicio de Consulta Lexicon de CORBAMed
- Control y Administración de flujos de Audio/Vídeo, del DTF de telecomunicaciones
- Servicio de Notificación, del DTF de Telecomunicaciones.

2.8 Nuevas especificaciones de CORBA

Después que fue completada y normalizada la versión 2.0 en 1996, la OMG continuo trabajando en la incorporación de aspectos importantes que deben ser tenidos en cuenta en un sistema distribuido y ha ampliado su modelo de objetos para incorporar aspectos como: múltiples interfaces por objeto, paso de objetos por valor, modelo de componentes, soporte para tiempo real y tolerancia a fallos entre otros [Omg00a]. CORBA 3.0 se refiere a una serie de nuevas especificaciones que unidas dan una nueva dimensión a CORBA. Esta nuevas especificaciones están divididas en tres categorías:

- Integración con Internet.
- Control de Calidad de Servicio
- Arquitectura de componentes CORBA

Por otro lado, han aumentado las especificaciones de mercados verticales, o lo que en CORBA se conoce como Dominios Verticales y mediante la utilización de IDL, existen muchos mercados estandarizando en CORBA (Finanzas, Seguros, Comercio electrónico, Medicina, Manufactura, Telecomunicaciones, Transportes, Investigación y Objetos de negocio).

2.9 Integración con Internet

Esta integración esta siendo desarrollada por la especificación “firewall”. La especificación CORBA 3 de firewalls define firewalls a nivel de transporte, de aplicación y conexiones bidireccionales GIOP útiles para callbacks y notificación de eventos.

Los firewalls de transporte trabajan a nivel de TCP, para lo que la IANA ha reservado los puertos bien conocidos 683 para IIOP y 684 para IIOP sobre SSL. También hay una especificación de CORBA sobre SOCKS.

En CORBA es frecuente que un objeto invoque un método (callback) o notifique de algún suceso a su cliente, por esto el objeto puede comportarse como cliente, por esto generalmente se requiere abrir una nueva conexión TCP en sentido inverso el cual puede ser detenido por un firewall. Bajo esta especificación, una conexión IIOP se le permite llevar invocaciones en el sentido inverso bajo ciertas condiciones que no comprometan la seguridad de la conexión. Más información acerca de esta especificación puede ser encontrada en [Omg98e].

Modelo de componentes de CORBA (CORBABeans)
CORBA Components [Omg00d] extiende el modelo de objeto de la OMG para incluir un número de características importantes en un sistema distribuido. La noción de componente puede no corresponder al modelo uno a uno ni de un objeto CORBA, esta centrado más en la relación de un Componente con un conjunto de interfaces.
Los componentes tienen identificadores de instancias, así como propiedades que son externamente accedidas y que soporta mecanismos de notificación (servicio de eventos) y validación cuando alguna propiedad cambia.
Los componentes de CORBA no solo serán trasladados a los lenguajes ya soportados, sino también a otros modelos de componentes como los Java Beans. Igualmente se esta trabajando en una especificación para un lenguaje de scripting que facilite el nivel de usuario la utilización de componentes.

3. Common Gateway Interface (CGI)
Common Gateway Interface es una interfaz al servidor Web que permite extender la funcionalidad de éste. Con CGI se puede interactuar con los usuarios que acceden a un sitio en particular. En un nivel teórico, los CGI permiten extender las capacidades del servidor para interpretar las entradas obtenidas del browser (navegador) y regresar la información apropiada de acuerdo a la entrada del usuario. En un nivel práctico, CGI es una interfaz que facilita la escritura de programas para que se comuniquen fácilmente con el servidor.
Usualmente, si se desea extender las capacidades del servidor Web, se tendría que modificar el servidor manualmente. Esta no es una solución deseable puesto que requiere un entendimiento de bajo nivel de programación de red sobre el Internet y el protocolo World Wide Web. También requiere editar y recompilar el código fuente del servidor o escribir un servidor dedicado para cada tarea. Por ejemplo, si se quiere extender el servidor para que actúe como una compuerta Web-a-e-mail que tomara la entrada del usuario desde el browser y lo reenvia a otro usuario. Se tendría que insertar código en el servidor que interpretara la entrada desde el browser, re-enviar la entrada al otro usuario y regresar una respuesta al browser sobre una conexión de red.
Primeramente, ésta tarea requiere accesar el código del servidor, algo que no siempre es posible. En segundo lugar, es difícil y requiere excesivo conocimiento técnico. Tercero, funciona solamente para un servidor específico. Si se quiere mover el servidor Web a una plataforma diferente, se tendrá que re-iniciar o al menos desperdiciar mucho tiempo al trasladar el código a esa plataforma.
CGI proporciona una solución portable y simple a estos problemas. El protocolo CGI define una forma estándar para que los programas se comuniquen con el servidor Web. Sin mucho conocimiento especial, se puede escribir un programa en cualquier lenguaje de computación que interactúe con el servidor Web. Este programa trabajará con todos los servidores Web que entiendan al protocolo CGI.
La comunicación CGI esta manejada sobre la entrada y salida estándar, lo que significa que si se conoce como imprimir en pantalla y leer datos utilizando el lenguaje de programación, se puede escribir una aplicación sobre el servidor Web. Programar las aplicaciones CGI es casi equivalente a programar cualquier otra aplicación . Por ejemplo, si se desea programar un programa "Hola México", se utiliza las funciones de imprimir en pantalla del lenguaje y el formato definido para programas CGI para imprimir en pantalla el mensaje apropiado.

- **Selección del lenguaje**

Debido a que CGI es una "interfaz común", no está restringida a ningún lenguaje de computación en particular. Una pregunta importante es ¿qué lenguaje de programación se utiliza para programar CGIs? Se puede utilizar cualquier lenguaje que realice lo siguiente:

- Imprimir a la salida estándar.
- Leer desde la entrada estándar.
- Leer desde variables de ambiente.

La mayoría de los lenguajes de programación y muchos lenguajes desempeñan estas tres actividades y se pueden utilizan cualquiera de ellas.
Los lenguajes caen bajo una de las siguientes clases: compilar o interpretar. Un lenguaje compilador tal como C o C++ tiende a ser más pequeño y más rápido, mientras que el lenguaje intérprete tal como Perl o Rexx requieren cargar, algunas veces, un intérprete grande antes de comenzar. Adicionalmente, se puede distribuir código binario (código compilado en lenguaje máquina) sin el código fuente, si el lenguaje utilizado es del tipo compilador. La distribución de scripts interpretados normalmente significa distribuir el código fuente.
Antes de escoger el lenguaje, se deben considerar las prioridades. Se necesita balancear la ganancia de velocidad y eficiencia de un lenguaje de programación contra la facilidad de programación de otro.
Existen alternativas importantes para aplicaciones CGI. Ahora, muchos servidores incluyen una programación API que hace más fácil programar directamente extensiones al servidor en contra parte a separar aplicaciones CGI. Los servidores APIs tienden a ser más eficientes que los programas CGI. Algunas aplicaciones se manejan por nuevas tecnologías desarrolladas en la parte del cliente (en lugar del servidor) tales como Java.

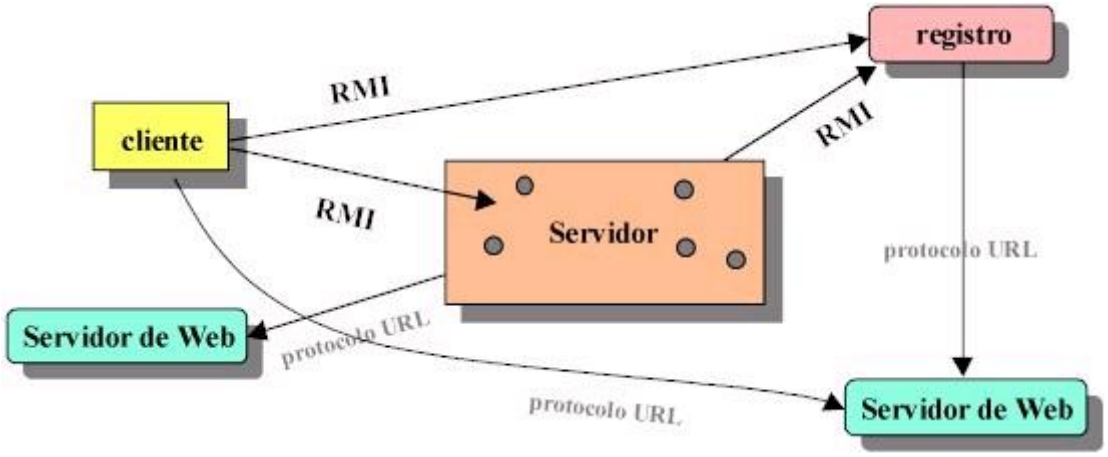
4. Java en Computación Distribuida

Java es una arquitectura neutral, orientada a objetos, portable y un lenguaje de programación de alto desempeño que proporciona un ambiente de ejecución dinámica, distribuida, robusta, segura y multi-hilos. La principal ventaja de Java para computación distribuida radica en la capacidad de descargar el ambiente. En términos de una arquitectura de objeto distribuido totalmente nueva, Java proporciona las siguientes opciones: Java Remote Method Invocation (RMI), Java IDL y la empresa JavaBEan. La especificación RMI es un API que nos permite crear objetos escritos puramente en lenguaje de programación Java, cuyos métodos se invocan de una Máquina Virtual Java diferente (JVM Java Virtual Machine). La tecnología Java IDL para objetos distribuidos facilitan que los objetos interactuen a pesar de estar escritos en lenguaje de programación Java u otro lenguaje tal como C, C++, COBOL, entre otros.

4.1 Java RMI

Básicamente RMI proporciona la capacidad para llamadas a métodos sobre objetos remotos, los cuales convierten al componente de transporte del objeto en arquitectura de objeto distribuido. También proporciona mecanismos para el registro y persistencia del objeto. Ofrece servicios distribuidos tales como Java IDL que proporciona una forma de conectar, transparentemente, a los clientes Java a los servidores de red utilizando la industria estándar: Lenguaje de Definición de Interfaces (IDL Interface Definition Language). Las aplicaciones RMI están, a menudo, compuestas de dos programas separados: un servidor y un cliente. Una aplicación común del servidor crea algunos objetos remotos, realiza referencias para accederlos y se encuentra en espera de que los clientes invoquen los métodos sobre éstos objetos remotos. Una aplicación del cliente tiene una referencia remota a uno o más objetos remotos en el servidor y entonces invoca al método sobre ellos. RMI proporciona los mecanismos a través de los cuales el servidor y el cliente se comunican e intercambian información. Las aplicaciones utilizan uno o dos mecanismos para obtener referencias a objetos remotos. Una aplicación registra sus objetos remotos con la facilidad de denominación del RMI, o bien la aplicación pasa y regresa la referencia a los objetos remotos como parte de su operación normal. Los detalles de comunicación entre objetos remotos está a cargo del RMI; para el programador, la comunicación remota se asemeja a la invocación del método Java. Dado que RMI permite que un solicitante pase objetos a objetos remotos, RMI proporciona los mecanismos necesarios para cargar un código de objeto, así como también de transmitir sus datos. RMI soporta su propio protocolo de transporte denominado Protocolo de Mensajes Remotos Java (JRMP Java Remote Messaging Protocol) para definir el conjunto de formatos de mensajes que permiten que los datos pasen a través de una red de computadoras a otra.

La siguiente ilustración representa una aplicación distribuida RMI que utiliza el registro para obtener una referencia a un objeto remoto. El servidor llama al registro para asociar (o ligar) un nombre con un objeto remoto. El cliente busca el objeto remoto por su nombre en el registro del servidor y entonces invoca un método sobre él. La ilustración también muestra que el sistema RMI utiliza un servidor Web para cargar los códigos en bytes de las clases, del servidor al cliente y del cliente al servidor, para los objetos cuando se les necesita.



RMI pasa objetos por su tipo verdadero permitiendo que nuevos tipos sean introducidos en una máquina virtual remota por consiguiente extendiendo el comportamiento de una aplicación de manera dinámica. RMI trata un objeto remoto de manera diferente de un objeto local cuando el objeto se le pasa de una máquina virtual a otra. En lugar de hacer una copia de la implementación del objeto en la máquina virtual receptora. El solicitante invoca un método en el stub local que tiene como responsabilidad cargar la llamada al método en el objeto remoto. Un stub para objetos remotos implementa el mismo conjunto de interfaces remotas que el mismo objeto remoto implementa. Esto permite que un stub se acople a cualquiera de las interfaces que el objeto remoto implemente. Sin embargo, esto también significa que solamente aquellos métodos, definidos en una interface remota, estén disponibles para su solicitud en la máquina virtual receptora.

Java IDL

Java IDL esta basado en CORBA (Common Object Request Brokerage Architecture). Una característica clave de CORBA es un lenguaje-neutral IDL (Interface Definition Language). Cada lenguaje que soporta CORBA tiene su propio traductor IDL y como su nombre lo dice, Java IDL soporta la traducción para Java. Para soportar la interacción entre objetos en programas separados, Java IDL proporciona un ORB (Object Request Broker). El ORB es una librería clase que facilita la comunicación de bajo nivel entre las aplicaciones Java IDL y otras aplicaciones CORBA.

¿RMI o IDL?

- Java RMI es una solución 100% Java para objetos remotos que proporciona todas las ventajas de las capacidades de Java "una vez escrito, ejecutarlo en cualquier parte". Los servidores y clientes desarrollados con Java RMI se muestran en cualquier lugar en la red sobre cualquier plataforma que soporta el ambiente de ejecución de Java. Java IDL, en contraste, está basado en una industria estándar para solicitar, de manera remota, a objetos escritos en cualquier lenguaje de programación. Como resultado, Java IDL proporciona un medio para conectar

aplicaciones "transferidas" que aún cubren las necesidades de comercio pero que fueron escritos en otros lenguajes.

- Actualmente Java RMI y Java IDL emplean protocolos diferentes para la comunicación entre objetos sobre diferentes plataformas. Java IDL utiliza el protocolo estándar de CORBA IIOP (Internet Inter-Orb Protocol). Al igual que IDL, IIOP facilita la residencia de objetos en diversas plataformas y escritos en diversos lenguajes para interactuar de manera estándar. Actualmente Java RMI utiliza el protocolo JRMP (Java Remote Messaging Protocol), un protocolo desarrollado específicamente para los objetos remotos de Java.
- En Java IDL, un cliente interactúa con un objeto remoto por referencia. Esto es, el cliente nunca tiene una copia actual del objeto servidor en su propio ambiente de ejecución. En su lugar, el cliente utiliza stubs en la ejecución local para manipular el objeto servidor residiendo sobre la plataforma remota. En contraste, RMI facilita que un cliente interactúe con un objeto remoto por referencia, o por su valor descargándolo y manipulándolo en el ambiente de ejecución local. Esto se debe a que todos los objetos en RMI son objetos Java. RMI utiliza las capacidades de serialización del objeto, que proporciona el lenguaje Java, para transportar los objetos desde el servidor al cliente. Java IDL, dado que interactúa con objetos escritos en cualquier lenguaje, no toma ventaja de "una vez escrito, ejecutarlo en cualquier parte", característica del lenguaje de programación Java.

5. Comparación de Arquitecturas

- **Bases**
  - La base para las arquitecturas de objeto distribuido presentadas, es la arquitectura cliente-servidor.
  - La arquitectura utiliza un tipo de protocolo de transporte (IIOP, JRMP, DCOM o HTTP) para enviar mensajes a través de las computadoras en una red.
  - Utilizan un tipo de invocación de método remoto.
- **Neutralidad de la Arquitectura vs. Transparencia en la Comunicación**
  - CORBA proporciona transparencia en la comunicación, transparencia local/remota e independencia de la plataforma. Java RMI proporciona una arquitectura neutral, transparencia en la comunicación pero no proporciona transparencia local/remota. DCOM proporciona soporte para solamente plataformas Windows. CGI, por su parte, proporciona transparencia en la comunicación, transparencia local/remota, independencia de la plataforma y una arquitectura neutral.
- **Abastracción**
  - CORBA proporciona una solución abstracta de cómputo distribuido, que se puede implementar a través de plataformas, sistemas operativos y lenguajes. Mientras que DCOM, CGI y los Componentes Distribuidos Java especifican detalles de implementación.
- **Independencia del Lenguaje**
  - Los estándares de CORBA, CGI y DCOM se utilizan en distintos lenguajes mientras que Java RMI se utilizan solamente con lenguaje Java. Sin embargo, Java IDL se utiliza en diversos lenguajes.
  - Java RMI también es un ORB en el sentido genérico, esto es nativo al lenguaje Java.
- **Complejidad y Madurez**
  - CORBA es más maduro que las otras tres tecnologías y están disponibles diversas implementaciones de éste estándar.

    - CORBA utliza IDL debido a lo cual, el desarrollo de aplicaciones es más complejo que DCOM, CGI y los Componentes Java.
- **Tecnología**
    - A pesar que Java y CGI's proporciona medios para cómputo distribuido es aún una tecnología de programación mientras que CORBA es una tecnología de integración. DCOM es también una tecnología de integración pero sólamente bajo plataformas Windows.
- **Servicios Distribuidos**
    - CORBA proporciona un conjunto más completo de servicios distribuidos que Java y DCOM. Aunque Java esta creciendo con API's para proporcionar éstos servicios.
- **Paso por Valor/Referencia**
    - CORBA actualmente no soporta el paso de objetos por valor mientras que DCOM, Java RMI y CGI's (en desarrollo a través de URN) si los soporta.
- **Simplicidad en el desarrollo de aplicaciones**
    - Las aplicaciones distribuidas se producen fácilmente empleando DCOM y Java puesto que existen herramientas disponibles para el desarrollo de la aplicación. CORBA carece de esta facilidad y así es difícil construir aplicaciones CORBA, lo mismo sucede con CGI's.
- **Herencia**
    - DCOM proporciona soporte para objetos que tienen interfaces múltiples pero no soporta herencia en la interfaz. CORBA soporta herencia en la interfaz pero no soporta objetos que tienen interfaces múltiples.
- **Recolección de Basura (Garbage Collection GC)**
    - DCOM y los objetos Java tienen recolección de basura, lo cual no tiene CORBA ni CGI's.
- **Otros**
    - CORBA realiza la tarea de registro de objetos, la generación de referencia a objeto y la inicialización del skeleton de manera implícita. En DCOM éstas tareas son manejadas por el programador explícitamente o bien, se realizan dinámicamente durante el tiempo de ejecución.
    - El protocolo DCOM esta fuertemente atado a RPC, CORBA no lo está.
    - Las tecnologías desarrolladas en la parte cliente tales como Java, no son probables que reemplacen a los CGI porque existen ciertas aplicaciones que las del lado del servidor están mejor adaptados para realizarse.

Muchas de las limitaciones de CGI son limitaciones de HTML o HTTP. Dado que los estándares del Web, en general, que involucran éstas limitaciones no afectan las capacidades de CGI.

Cuadro Comparativo

Arquitectura	Independencia del Lenguaje	Independencia del Lenguaje	Tecnología	Paso por Valor/Referencia	Garbage Collection	Independencia de Plataforma
CORBA	si	integración	por referencia	no	si	
DCOM	si	integración (Windows)	por valor	si	no, sólo para Windows	
JAVA	sólo con lenguaje Java	programación	por valor	si	si	

CGI	si	programación	ambas	no	si	

Según: http://delta.cs.cinvestav.mx/~gmorales/seminario2000/ArtLuis/ArtLuis.html;

BIBLIOGRAFIA

http://www.microsoft.com/Com/ Página oficial de Microsoft
http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomarch.asp Arquitectura DCOM
http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomtec.asp Resumen Técnico de DCOM
http://www.cvc.uab.es/shared/teach/a20383/practiques/ Introducción al modelo COM
http://www.gsi.dit.upm.es/~jcg/is/curso97-98/grupos/y3/html_doc/indice2.htm OLE/COM/DCOM
http://www.dis.eafit.edu.co/areas/telematica/online/corba/intro/ Objetos distribuidos CORBA/RMI/DCOM
http://www.gsyc.inf.uc3m.es/~jjmunoz/lro/9798/copia/%257Eatrigo/datos/indice.html Programación COM/DCOM
http://club.idecnet.com/~chavesj/dcom/ DCOM for children
http://www.codeguru.com/activex/index.shtml Código fuente ActiveX/COM/DCOM
http://www.cs.concordia.ca/~teaching/comp690j/dcomTutorial/comTutorial.html Tutorial DCOM/COM
http://www.codeproject.com/com Código fuente COM/DCOM
http://journal.iftech.com/articles/dcom_1/ Excelente Tutorial de COM/DCOM
http://www.cs.wustl.edu/~schmidt/submit/Paper.html DCOM y CORBA (comparación)
http://shrike.depaul.edu/~tliu/ds520/link.htm Links ActiveX y DCOM
http://www.dalmatian.com/com_dcom.htm DCOM
http://swt.informatik.uni-jena.de/~stolle/f/CompSem2000/works/COM-paper_html/ Introducción a COM, DCOM y COM+
http://sern.ucalgary.ca/Courses/CPSC/547/W2000/webnotes/COM/COM.html ActiveX, Com y DCOM
http://tochna.technion.ac.il/project/LearnDCOM/html/LearnDCOM.html COM/DCOM/ActiveX
http://www.elai.upm.es/spain/Investiga/GCII/areas/administracion/DCOM.htm Introducción DCOM
http://delta.cs.cinvestav.mx/~gmorales/seminario2000/ArtLuis/ArtLuis.html; comparación entre arquitecturas

Yamile Ramirez Herrera
yamileastrid@hotmail.com