Análisis de Modelos de Componentes

Dr. Alfonso Miguel Reyes
Instituto Mexicano del Petróleo, CIC-IPN.
Lic. Elizabeth Acosta Gonzaga
Investigadora del CIDETEC-IPN.

Este artículo describe un análisis de dos de los Modelos de Componentes más usados en la industria del software: COM (Component Object Model) y EJB (Enterprise Java Beans), con la finalidad de obtener una estructura general de un Modelo de Componentes, sus elementos, características y procesos comunes, que permitan establecer un marco de referencia a futuros trabajos de modelos de componentes, así como abrir líneas de investigación sobre estos elementos que coadyuven a una mejor estandarización e interoperabilidad entre modelos diferentes.

Componentes de Software

Un componente de software es definido como [MLUM99, p. 8]:

"Un componente de software es una abstracción estática con enchufes, parte de una estructura, con una arquitectura de software que determina las interfaces que los componentes pueden tener y las reglas que gobiernan su composición".

La característica estática se debe a la instanciación que sufre para ser usado. El componente tiene enchufes, los cuales son usados para proporcionar servicios, pero también para requerir de ellos. Los enchufes son el principal prerrequisito para composición. Un enchufe es una interfaz, pero qué tipo es exactamente ésta, y cómo se enchufan conjuntamente, depende de una estructura de componentes a otra.

Un componente no es usado en forma aislada, sino en el contexto de una arquitectura de software que determina cómo los componentes son enchufados conjuntamente. La estructura, la arquitectura de software y las reglas de composición pueden variar entre los diferentes modelos de componentes.

Szypersky [CSZY97, p. xiii] define a un componente como:

"Una unidad binaria de producción, adquisición e instalación independiente, que interactúa con otros para formar un sistema funcionando".

Un componente de software finalmente es la implementación de un modelo conceptual, producto de una descomposición funcional realizado por el arquitecto de un sistema para obtener la estructura que satisfaga una aplicación de software [PHOS00, p. 429].

Modelos de Componentes.

Recientemente han emergido Modelos de Componentes como una manera para construir aplicaciones fácil y rápidamente, creando una simple estructura para combinar, reusar y transformar componentes de software [RHTS98, p.127].

Las características generales de un modelo de componentes pueden ser definidas como [PHOS00, p. 7]:

1. Un componente es una construcción autocontenida que tiene un uso definido, tiene una interfaz en tiempo de ejecución, puede ser instalado automáticamente y es construido con el conocimiento de un código pegamento específico.

2. El código pegamento es un software que proporciona una bien definida y bien conocida interfaz en tiempo de ejecución para soportar una infraestructura en la cual el componente se fijará. Una interfaz en tiempo de diseño sólo es necesaria pero no suficiente porque ésta no existe en tiempo de ejecución, a menos que sea implementada por alguna pieza de software, esto es, la infraestructura.

3. Un componente es construido por composición y colaboración con otros componentes.

4. Un código pegamento y los correspondientes componentes son diseñados para uso de una persona con conjunto definido de habilidades y herramientas.

Un modelo de Componentes es definido como:

Una estructura metodológica para desarrollar aplicaciones de software usando componentes de software, en el que se definen los atributos, características, mecanismos, herramientas y procesos empleados para lograr la composición de los diferentes elementos de software denominados componentes.

Un atributo es una propiedad del componente, la cual conserva indistintamente y es parte de la definición del modelo. Los atributos constituyen los elementos que definen al modelo.

Las características son el conjunto de cualidades que distinguen a un modelo de componentes de otro y se obtienen en función del ambiente que define el componente, de sus atributos y capacidades.

Los mecanismos de composición definen la forma de realizar la composición de los diferentes elementos del componente.

Las herramientas de composición definen los mecanismos que permiten extender las habilidades del desarrollador, para lograr la creación de componentes.

Un proceso define la forma cómo es creado y usado un componente, lo que impone un estilo de desarrollo e implementación de aplicaciones.

Cada modelo de componente es implementado a través de uno o varios frameworks (estructuras), los cuales constituyen las plataformas de desarrollo de aplicaciones con componentes. Éstos incorporan los mecanismos necesarios para lograr su distribución y composición a través de la red. Por lo que un Modelo de Componentes es un framework horizontal y vertical al mismo tiempo, y también es una herramienta que facilita el desarrollo de aplicaciones distribuidas.

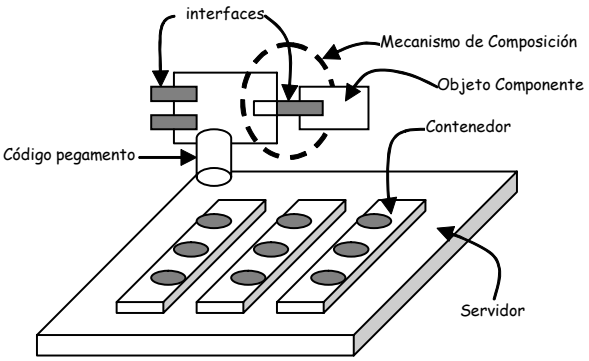

Figura 1. Diagrama de un Modelo de Componentes.

La **figura 1** muestra el diagrama de un Modelo de Componentes.

Atributos de un Modelo de Componentes

Un atributo es una propiedad del componente. Los atributos constituyen parte de los elementos que definen al modelo y son inherentes a éste.

Nombres

Un nombre indica un recurso del sistema, el cual puede ser un dato, componente, evento, etc., cuyas características son significativas, relevantes y asociadas en un contexto específico. Un Modelo de Componentes debe resolver la forma de nombrar los diferentes recursos de aplicaciones en un contexto distribuido.

Interfaces

Una interfaz es un atributo que permite definir puntos de conexión en un componente, para acoplar diferentes elementos. Ésta define una frontera o límite entre dos componentes que permite intercambiar información o interoperar. Szypersky proporciona la siguiente definición [CSZY97, p.174]:

"*Una colección de definiciones de operaciones, cada una con una firma y posiblemente un tipo de retorno. La firma de la operación define el número, tipos y modos de paso de parámetros*".

Servidor/Contenedor

Un servidor/contenedor proporciona un contexto de desarrollo, instalación, ejecución y ciclo de vida de los componentes. De desarrollo, porque ofrece mecanismos para que los componentes puedan interactuar entre ellos; de ejecución, al proporcionar mecanismos que permiten instanciar los componentes requeridos por las aplicaciones en ejecución; de instalación porque ofrece los mecanismos necesarios para registrar los nuevos componentes y de ciclo de vida, al permitir llevar la administración de la creación, uso y destrucción de componentes.

El Servidor implementa los servicios de más bajo nivel del modelo de componentes, como son aquellos asociados con el uso de servicios de las plataformas de middleware. El contenedor implementa los servicios de más alto nivel, como son los asociados con la administración y ciclo de vida de los componentes.

Fábricas de componentes

Un atributo distintivo de un modelo de componentes es la forma como implementa la instanciación de sus componentes, ya que no existe un

mecanismo como el *"new"* de los lenguajes de programación orientados a objetos; éstos deben implementar lo que se denomina "fábricas de componentes", la cual define un objeto que permite crear otro objeto [RHIG96, p.126].

Código Pegamento o Enchufe

El código pegamento o enchufe define la forma cómo un componente es enchufado o pegado al servidor/contenedor, lo que permite que pueda ser utilizado y enlazado por otros componentes. Cada modelo de componentes define una forma particular de realizarlo.

Objeto Componente

Un objeto componente o componente constituye la definición de la unidad mínima de instalación, administración e instanciación en un modelo de componentes, la cual es registrada por el Servidor/Contenedor y corresponde básicamente a la definición de la interfaz del componente, la que a su vez define una unidad de instanciación.

Características de un Modelo de Componentes.

Un modelo de componentes puede tener varias características [MLUM99, p. 9], aunque el número de éstas puede ser tan grande como el número de modelos de componentes existentes. Sólo se analizan los más generales.

1. Escala y Granularidad.- Define el tamaño del componente.

2. Código Binario o Fuente.- Indica la forma final del código del componente.

3. Homogeneidad o Heterogeneidad.- Indica la aceptación en la composición de elementos de otros modelos en forma directa, sin intervención por parte del desarrollador en el diseño de adaptadores o transformadores.

4. Caja Blanca o Caja Negra.- Manifiesta la forma ó capacidad como el componente es exportado.

5. Con Estado o sin Estado.- Indica la posibilidad de modificar los estados de los componentes, sobre todo los referentes a los atributos del componente sin necesidad de codificación.

6. Metacomponentes.- Indica la capacidad del componente para describir su composición, y los elementos que la integran en forma dinámica en tiempo de ejecución, por lo que se dice es reflectivo. Un metacomponente define un componente con capacidades reflectivas.

7. Administración de Versiones.-Indica los mecanismos que permitan identificar y hacer compatibles diferentes versiones de un mismo componente.

8. Tipos.- Identifica mecanismos de uso ó verificación de tipos. De interés especial es la consideración que un tipo es una interfaz con un contrato simplificado [CSZY97, p. 77].

9. Distribución.- Es la capacidad del modelo con mecanismos necesarios para lograr su distribución y composición a través de la red.

El Modelo de Objetos de Componentes (COM, Component Object Model)

COM es resultado del proceso evolutivo de tecnología de Microsoft, ya que unifica diferentes criterios utilizados en diferentes productos: bibliotecas DLL (Dynamic Linking and Loading), DDE (Dynamic Data Exchange), objetos OLE (Object Linking and Embedding), objetos OLE2, etc. [DROG97, p. 12].

Un componente es una abstracción con enchufes, los cuales le permiten agregarse o incorporarse a otros, utilizando algún mecanismo de composición. COM especifica la arquitectura de un Modelo de Componentes que permite la composición de elementos denominados objetos componentes. Los enchufes son denominados interfaces, las cuales definen los puntos de conexión entre diferentes elementos de un componente, además son el medio de composición de los elementos.

Los objetos componentes son los elementos básicos de composición, éstos representan unidades de implementación de los servicios ofrecidos por el componente. Un objeto componente es construido para satisfacer un requerimiento del sistema o aplicación. Un requerimiento de un sistema puede ser tan simple que un solo componente sea necesario para su satisfacción o tan general que es dividido en diferentes subrequerimientos, lo que obliga a los diseñadores a desarrollar diferentes niveles de abstracción. Cada nivel de abstracción puede ser representado por uno o varios componentes, los cuales a su vez son utilizados como elementos de composición posteriormente.

En su definición más sencilla un servidor es un proceso proveedor de servicios que son demandados por un conjunto de procesos denominados clientes. En COM un servidor es un mecanismo de agrupación de componentes.

La **figura 2** muestra el diagrama del Modelo de Componentes COM.

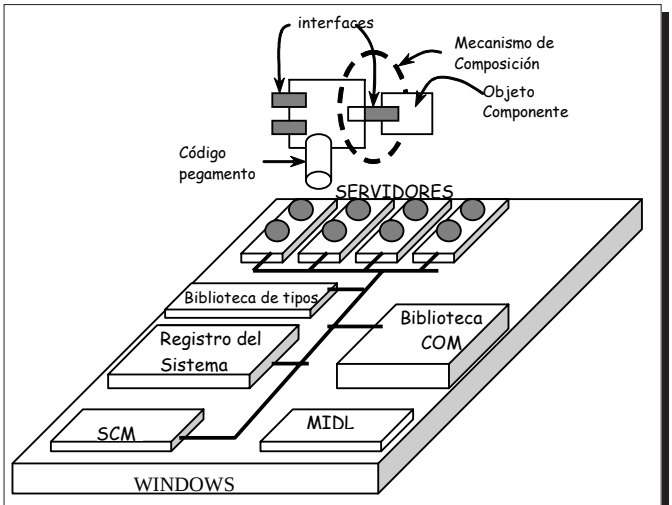

Figura 2. Modelo de componentes COM.

ATRIBUTOS

INTERFACES

Una interfaz proporciona una conexión entre dos diferentes objetos, ya que permite que un objeto use una función implementada en otro objeto, por lo que la interfaz los encadena, como se muestra en la **Figura 3**.

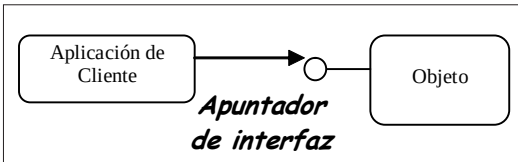

Figura 3. Interfaz.

Una de las características del modelo de componentes COM es la independencia del lenguaje de programación, por la utilización de un Lenguaje de Definición de Interfaces Microsoft (MIDL por sus siglas en ingles), el cual permite definir interfaces que son independientes de un lenguaje de programación, con la estructura mostrada en la **Figura 4**.

La interfaz debe tener al menos dos atributos: ***Object*** para indicar que es una interfaz COM y ***el nombre físico de la interfaz, e****l cual es representado por un Identificador Global Unico (GUID por sus siglas en Inglés) de 128 bits*, obtenido por medio de la función ***HRESULT CoCreateGuid (GUID *pguid).***

INTERFAZ IUNKNOWN

Las interfaces usadas en COM deben ser derivadas de la ***interfaz Iunknown***, la cual es definida por el archivo UNKNOWN.H o UNKNOWN.IDL. Ésta constituye la base de composición de los componentes COM, ya que contiene tres funciones miembros que le permiten navegar a través de los componentes y manejar su ciclo de vida: ***QueryInterface(), AddRef(), Release().***

La **figura 5** muestra la definición de la interfaz IUnknown.

```
Interface IUnknown
{
    virtual HRESULT STDMETHODIMP QueryIntrface(REFIID riid, void *ppv) = 0;
    virtual ULONG STDMETHODIMP AddRef(void) = 0;
    virtual ULONG STDMETHODIMP Release(void) = 0;
}
```

Figura 5. Interfaz Unknown.

MÉTODO QUERYINTERFACE

La interfaz IQueryInterface comprueba que el componente y el proceso que lo llama son realmente compatibles. *QueryInterface* es la parte más importante de COM ya que las interfaces que un componente soporta son las que *QueryInterface* regresa un apuntador de interfaz.

MÉTODOS ADDREF() Y RELEASE()

AddRef y Release implementan una técnica de administración de memoria llamada ***conteo de referencias***. Un componente COM mantiene un número llamado cuenta de referencia. Cuando un cliente toma una interfaz desde el componente, la cuenta de referencias es incrementada y cuando el cliente finaliza el uso de la interfaz, la cuenta de referencia es decrementada. Cuando la cuenta de referencias llega a cero, la instancia de la clase es borrada.

```
[ atributo1, atributo2,.....,atributon, ..]
interface IEstaInterfaz : IInterfazBase
{
        typedef1;
        typedef2;
        .
        .
        method1;
        method2;
}
```

Figura 4. Estructura de un MIDL.

NOMBRES

COM no define un servicio de nombres para administración de contextos de nombres, los cuales permiten resolver los problemas asociados con la identificación de nombres en un contexto distribuido. En este caso ofrece algoritmos para la generación de identificadores únicos (GUID), los cuales son asociados a nombres de interfaces, los que a su vez identifican a un componente.

Los GUID's residen en el registro del sistema (system registry), la cual puede ser explorada con el programa ***regedit.***

SERVIDORES

Un servidor COM es un archivo binario que agrupa el código de métodos para uno o más objetos COM.

Un servidor puede ser empacado como una librería de enlace dinámico o un archivo ejecutable normal. Cada máquina que soporta COM tiene un SCM (Service Control Manager), que es el encargado de los servicios de activación de los objetos COM (servidor) y encadenar el apuntador de interfaz inicial (IUnknown) [DBOX98, p.100-114]. Cada objeto COM corre dentro de un servidor. Un solo servidor puede soportar múltiples objetos COM como se muestra en la **figura 6**, existiendo tres formas como un cliente puede accesar a objetos COM proveídos por un servidor: Servidor en proceso, Proxy de objeto local y Proxy de objeto remoto.

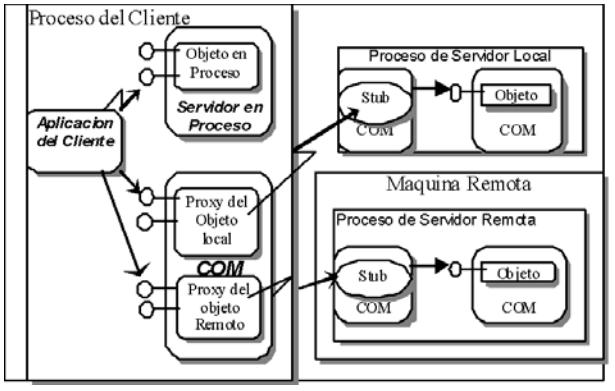

Figura 6. Servidor y objetos COM.

Código Pegamento o Enchufe

Este tipo de código se encuentra compuesto por una clase objeto, los puntos de entrada del servidor y las funciones de registro en el sistema.

Fábricas de Clases

Existen dos interfaces que pueden ser usadas como fábricas de clases, *IClassFactory* e *IclassFactory2*. Éstas tienen dos métodos: *CreateInstance* y *LockServer,* las cuales permiten crear una instancia de una clase COM. LockServer evita la caída o descarga de un servidor cuando este aún puede ser utilizado.

Objeto Componente

Un objeto COM no es un objeto en el sentido estricto de orientación a objetos, las interfaces COM no tienen estado y no pueden ser instanciadas para crear un objeto único. Una interfaz es simplemente un grupo de funciones relacionadas, los clientes de COM obtienen un apuntador para acceder a las funciones de una interfaz.

Una sola interfaz define a un objeto componente, aunque puede soportar cualquier número de interfaces, como se muestra en la **figura 7**.

Bibliotecas de Tipos

Una biblioteca de tipos define un archivo binario de encabezados independiente de los lenguajes de programación. Éste es creado por el compilador MIDL a partir de la definición del archivo IDL del componente. Este archivo contiene los nombres de las clases e interfaces que son implementadas con el servidor, número y tipo de los parámetros para cada método, así como los GUID's para cada clase e interfaces.

Mecanismos de Composición

La forma de reutilización en COM es a partir de la definición de nuevas interfaces que en esencia crean nuevos componentes COM, las cuales pueden extender interfaces existentes, agregando nueva funcionalidad requerida. COM soporta dos mecanismos de composición: la contención o inclusión y la agregación de interfaces. En ambos casos el componente externo controla el ciclo de vida de los componentes internos e IUnknown representa las interfaces de los componentes internos.

La contención/delegación/inclusión en COM permite que un componente externo delegue o reenvíe las invocaciones de funciones o métodos ofrecidas por el componente, las cuales se encuentran en componentes internos. La **Figura 8** muestra un ejemplo de este tipo de mecanismo.

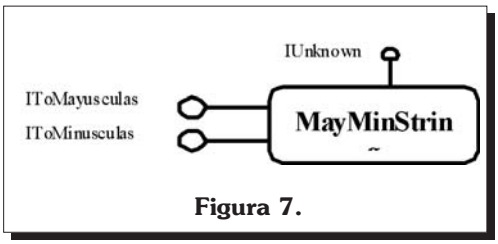

Figura 7.

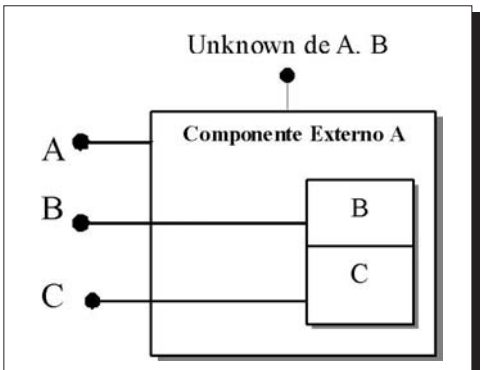

Figura 8. Contención/delegación/ inclusión.

En la agregación en lugar de reenviar cada llamada, el componente externo *IUnknown* expone directamente los apuntadores de las interfaces de los componentes internos a sus clientes, es decir el componente externo expone las interfaces de los componentes internos como suyas,

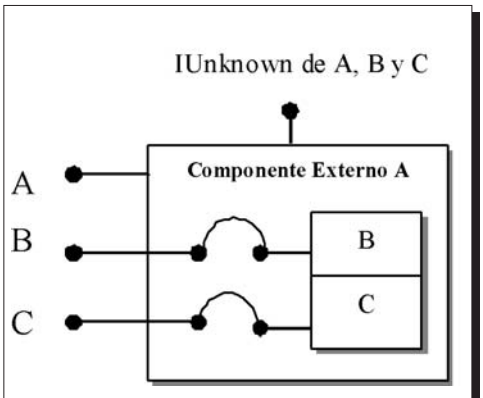

Figura 9.

como se muestra en la **Figura 9**.

Procesos

Los procesos básicos del modelo COM son: Creación y Uso de componentes COM. El proceso de creación de un componente COM es el siguiente:

1. Crear un proyecto DLL nuevo.
2. Crear un archivo de interfaz usando IDL.
3. Declarar una clase C++ que implemente la interfaz COM.
4. Implementar las interfaces declaradas.
5. Crear una clase objeto.
6. Crear los puntos de entrada requeridos por el DLL.
7. Crear las funciones de registro en el sistema.
8. Crear el cliente.

La creación del Cliente realiza los siguientes pasos:

1. Iniciando / Cargando un Servidor.
2. Instanciando un Componente.
3. Uso del apuntador de la interfaz del componente o referencia a éste.

Características

La siguiente tabla resume las características del modelo de componentes COM.

Característica		Comentarios
1	Escala y Granularidad	De granularidad pequeña a grande, a través de composición de interfaces.
2	Código Binario o Fuente	Código Binario
3 4	Homogeneidad o Heterogeneidad	Los componentes son heterogéneos en el lenguaje de programación, y en plataformas de hardware solo aquellas compatibles con Windows. Existe composición con otros modelos de componentes a través de técnicas de interoperabilidad, no en forma directa.
5	Caja Blanca o Caja Negra	Caja negra
6	Con Estado o sin Estado	Sin Estado.
7	Meta componentes	No
8	Administración de Versiones	No
9	Tipos	Si
10	Distribución	Si, con el uso de DCOM

JavaBeans Enterprise

JavaBeans Enterprise define un Modelo de Componentes como una arquitectura para el desarrollo y la implementación de aplicaciones distribuidas de negocios basados en componentes [SMIC99, p.15]. La arquitectura define como elemento base lo que denomina un bean empresarial, equivalente a un componente. Para entender el concepto de un componente EJB (Enterprise Java Beans), es necesario comprender el modelo de proceso de desarrollo de aplicaciones utilizando EJB. Este modelo supone cuatro fases de desarrollo:

1. Desarrollo de componentes EJB conteniendo la lógica del negocio.
2. Desarrollo de servidores/contenedores de aplicaciones para uso de plataformas de instalación y acoplamiento de componentes EJB.
3. Desarrollo de aplicaciones integrando y acoplando diversos componentes EJB en un servidor/contenedor.
4. Instalación y mantenimiento de las aplicaciones.

El proceso de desarrollo reconoce seis papeles o responsabilidades que adoptan los diseñadores de las aplicaciones, con los siguientes roles:

1. Proveedor de Beans.
2. El Proveedor de Contene dores.
3. El Proveedor de Servido res.
4. El Ensamblador de Aplica ciones.
5. El Instalador.
6. El Administrador del Siste ma.

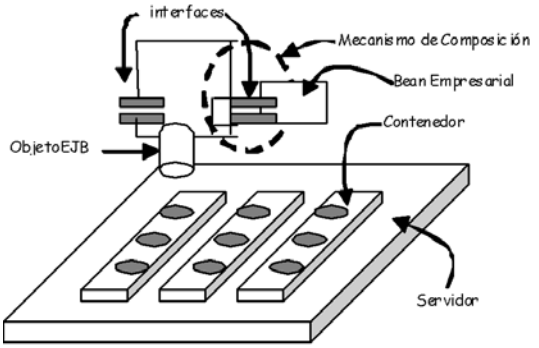

Figura 10. Diagrama del Modelo de Componentes EJB.

La **figura 10** muestra un diagrama del Modelo de Componentes EJB.

Atributos

Nombres

La forma de dar solución al problema de nombres en el modelo de componentes EJB, depende del proveedor del servidor/contenedor, ya que éste define el tipo de servicios de middleware que son soportados. Dos modelos son soportados ampliamente: El servicio de Nombres de CORBA (Common Object Request Broker Architecture) y el servicio de nombres de JNDI (Java Naming and Directory Interface).

JNDI es un sistema para clientes Java basado en acceso a estructuras de directorios de diferentes proveedores de sistemas distribuidos (LDAP (Lightweight Directory Acces Protocol), NIS (Network Information Services), SLP (Service Location Protocol), CORBA Name Services, etc.).

El Servicio de Nombres de CORBA permite formar contextos de nombres en una estructura jerárquica de árbol, la cual permite la navegación una vez obtenido un contexto inicial.

Interfaces

EJB usa el modelo de interfaces usadas en el lenguaje de programación Java, por lo que todos los componentes EJB son derivados de clases especiales.

La clase Bean Empresarial constituye el modelo de implementación del bean base definida a través de una interfaz, la cual proporciona los enchufes por los que el componente se incorpora a un contenedor y se conecta a los clientes [EROM, p.72]. Las clases bean empresariales son derivadas a partir de la interfaz *javax.ejb. EnterpriseBean,* además incluye otros dos tipos de interfaces: Remota y Hogar

INTERFAZ REMOTA

La Interfaz Remota conjunta todos los métodos que el componente expone a los clientes. Ésta se deriva de la interfaz *javax.ejb.EJBObject,* y se implementa por un objeto Hogar como se muestra en la Figura 11, posee un conjunto de métodos que permite la administración de los elementos del componente.

INTERFAZ HOGAR

La interfaz Hogar proporciona la definición de los métodos para crear, encontrar y remover los elementos del componente, se deriva de la interfaz *javax.ejb.EJBHome,* la cual es implementada por un objeto EJB como se muestra en la **Figura 11**.

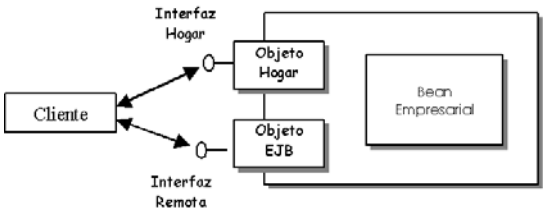

Figura 11. Interfaz Hogar, derivada de la interfaz *javax.ejb.EJBHome*.

CONTENEDOR/SERVIDOR EJB

El concepto de contenedor y servidor en EJB no es claramente separado [EROM99, p. 46], ya que el modelo EJB los define como una sola entidad. Un servidor EJB proporciona un ambiente para ejecución de uno ó más contenedores, éste administra los recursos de bajo nivel del sistema, asignándoselos a los contenedores como son requeridos. El contenedor proporciona una base donde los beans pueden ejecutarse; es responsable de administrar los beans que están corriendo en él. Los contenedores son responsables de conectar los clientes con los beans, realizar la coordinación de las transacciones, proveer persistencia, administrar el ciclo de vida, etc. La **figura 12** muestra el diagrama de un Servidor/Contenedor EJB.

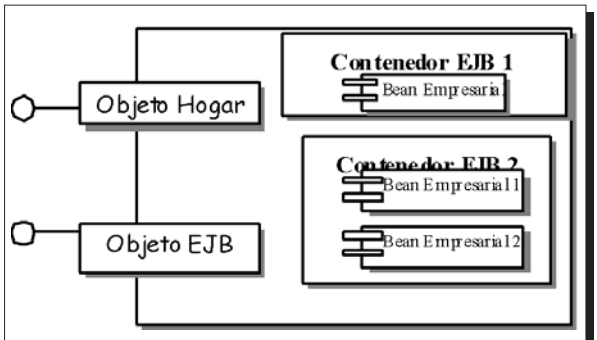

Figura 12. Diagrama de un Servidor/Contenedor EJB.

Un Servidor/Contenedor tiene la responsabilidad de administrar: las transacciones distribuidas, la seguridad, los recursos y ciclo de vida, la persistencia, la accesibilidad remota, el soporte multicliente y localizar en forma transparente nombres.

CONTEXTOS

Un contenedor almacena información relacionada con la operación de los beans en un objeto llamado ***objeto contexto EJB.*** Un objeto contexto representa un camino por el cual los beans pueden realizar llamadas de regreso al contenedor, para conocer su estado actual y modificarlo si es necesario. La información guardada en los contextos pertenece a los objetos hogar, objetos EJB, interacciones del bean, seguridad, y propiedades ambientales de la instalación del bean. Los beans poseen también un contexto propio.

CÓDIGO PEGAMENTO: OBJETOS EJB

Los clientes no pueden instanciar directamente un componente, ya que la invocación es atrapada por el contenedor y ***delegada*** a la instancia del bean. Un componente no puede ser llamado directamente a través de la red; ya que no posee habilidades de red, en este caso el contenedor realiza el trabajo correspondiente, además de guardar toda la información de uso de los beans contenidos, fungiendo como una capa de indirección entre el cliente y el bean. Esta capa de indirección es lo que se denomina ***Objeto EJB*** y es un representante que conoce toda la operación del componente, sirviendo como pegamento (GLUE) entre el cliente y el componente, como se muestra en la **Figura 13**.

FÁBRICAS DE CLASES: OBJETOS HOGAR.

Un cliente no puede instanciar un bean directamente, entonces: ¿Cómo adquiere referencias a los elementos

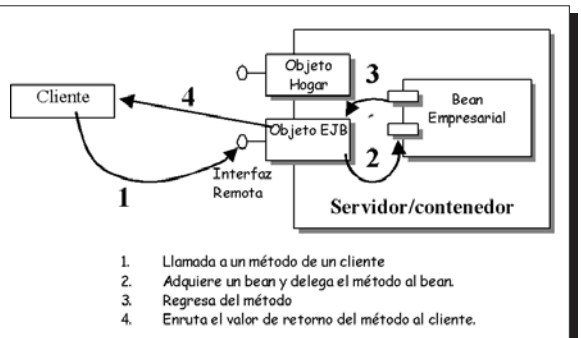

Figura 13. Capa de indirección u *Objeto EJB*.

del componente y cómo los destruye?. El mecanismo empleado es usando fabricas de clases, las cuales permiten la instanciación de objetos. Estas fabricas de clases en EJB son denominadas Objetos Hogar y tienen como responsabilidades: Crear, Localizar y Remover elementos.

Objeto Componente: Bean Empresarial (BE).

En el desarrollo de una aplicación se distinguen dos tipos de elementos que realizan diferentes tipos de pruebas, uno está orientado al desarrollo de transacciones cuyo objetivo es el procesamiento de información en general, y es llamado *componente de lógica de aplicación*; el otro se encuentra orientado a operaciones relacionados con el almacenamiento o manejo de datos por parte de la aplicación, *denominado componente de datos persistentes.* Un Bean Empresarial de Sesión representa un componente de lógica de aplicación, éste puede ser con estado o sin estado. Los beans de sesión con estado tienen la capacidad de registrar la información de los clientes, con los que están interactuando. Los beans de sesión sin estado no registran información de los clientes con los que interactuan.

Un Bean Empresarial de Entidad representa un componente de datos persistentes, usado para modelar datos. *Una instancia de un bean de entidad es la vista en memoria de la base de datos.* Los datos de un bean de entidad (o instancia de datos) son el conjunto físico de datos, almacenados en una base de datos.

Un bean de entidad es de larga duración, que sobrevive a las fallas, requiere la implementación de dos operaciones por parte del bean: *ejbLoad() y ejbStore()* para uso con una Base de Datos; *ejbActivate() y ejbPassivate(),* para optimizar el uso de recursos durante el apilamiento de operaciones; *ejbCreate(), ejbRemove(), y ejbFind*()* para administración de recursos del bean y ciclo de vida.

Un Bean Empresarial de Sesión ó Entidad se encuentra compuesto básicamente de los siguientes elementos:

1. Clase bean Empresarial.
2. Interfaz Remota.
3. Objeto EJB.
4. Interfaz Hogar.
5. Objeto Hogar.
6. Descriptor de Instalación.
7. Manifiesto.
8. Archivo Ejb-jar.

Un Bean Empresarial de Entidad agrega una Clase Llave Primaria, que es el identificador único del bean.

Los elementos 1, 2, 3, 4, 5 ya fueron descritos.

Descriptor de Implementación

Es un archivo que permite describir los requerimientos de servicios de middleware del componente.

Propiedades Específicas del Bean

Un archivo de propiedades específicas del componente puede ser incluido. Éstas son leídas en tiempo de ejecución para afinar las funciones del bean.

Archivo Ejb-jar

El archivo Ejb-jar constituye la última fase del proceso de desarrollo de un componente EJB, ya que es la fase de empacamiento del componente en un formato .JAR

Mecanismos de Composición

El mecanismo básico de composición de EJB es el de contención.

Procesos

El procedimiento general de construcción de bean de sesión sin estado es el siguiente:

1. Declaración de la interfaz remota
2. Implementación del bean Empresarial
3. Construcción de la interfaz Hogar
4. Escribir el Descriptor de Instalación.
5. Construir el archivo EJB-jar.
6. Escribir el código del cliente.

Un bean de sesión con estado requiere de mayores recursos, ya que es necesario guardar la información generada con la interacción con cada uno de sus clientes. Como la cantidad de clientes realizando peticiones a un bean con estado puede ser mayor a la capacidad disponible del bean, es necesario un mecanismo que permita maximizar el aprovechamiento de sus recursos, extendiéndolos de una manera virtual. El bean usa ***ejbPassivation()***, que permite liberar recursos al contenedor al almacenar en dispositivos secundarios la conversación del bean, y ***ejbactivation()***, que permite recuperar la conversación de un bean de un almacén secundario, como se muestra en la **figura 14**.

El procedimiento para escribir un bean de entidad es similar el de sesión, lo que cambia son los paráme-

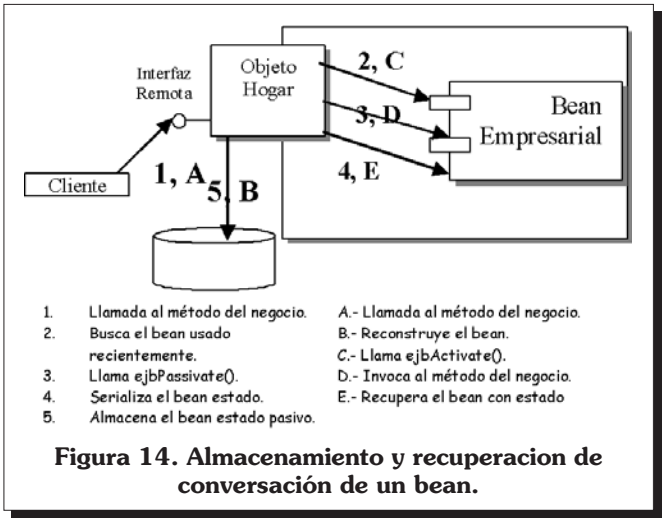

Figura 14. Almacenamiento y recuperacion de conversación de un bean.

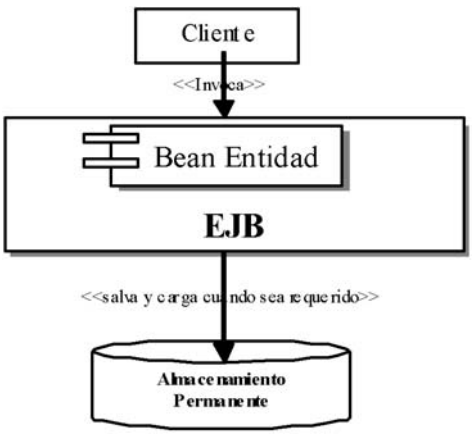

Figura 15. Interacción de un cliente con el Bean y el almacenamiento persistente.

tros que deben utilizarse en la herramienta de instalación para dotar de persistencia al bean, generalmente una base de datos, la cual debe estar trabajando. La **figura 15** muestra la forma como un cliente interactúa con el Bean y el almacenamiento persistente.

Un Bean Entidad a diferencia de un bean de sesión puede ser compartido por varios clientes. Además que un bean entidad debe modelar las características de transaccionalidad, concurrencia y recuperación, requeridas por unidades de información.

Una característica importante del modelo EJB es el Código Pegamento, el cual es generado en forma automática por el Servidor/Contenedor durante la instalación del Bean Empresarial, a partir de la información de la interfaz remota y hogar. Ésta es la razón por la que no aparece en el procedimiento de construcción del bean.

Característica	Comentarios
1. Escala y Granularidad	De granularidad pequeña a grande, a través de composición Beans Empresariales
2. Código Binario o Fuente	Código Bytecode
3. Homogeneidad o Heterogeneidad	Los componentes son homogéneos en el lenguaje de programación, y heterogéneos en plataformas de hardware. Existe composición con otros modelos de componentes a través de técnicas de interoperabilidad, no en forma directa.
4. Caja Blanca, Negra ó Gris	Caja Gris
5. Con Estado o sin Estado	Con Estado.
6. Meta componentes	Si
7. Administración de Versiones	No
8. Tipos	Sí
9. Distribución	Si, usando RMI ó CORBA

Características

La tabla anterior resume las características del modelo de componentes EJB.

Conclusiones

El análisis de los Modelos de Componentes construyó un marco de referencia para:

1. Una estructura común.
2. Identificar elementos comunes de un Modelo de Componentes.- En los siguientes años surgirán Modelos de Componentes con diferentes características, por lo que es necesario tener un marco de referencia para identificar los elementos comunes de análisis.
3. Comparación de Modelos de Componentes.
4. Facilitar toma de decisiones acerca de la selección de un Modelo de Componentes adecuado a dominios de aplicaciones especificas.
5. Abrir líneas de investigación en elementos comunes de los modelos, que faciliten la estandarización e interoperabilidad entre diferentes modelos.
6. COM y EJB son dos modelos de componentes con elementos comunes de diseño e implementación diferente. Ambos carecen de un Servicio de Eventos.

Bibliografía

CSZY97 Clemens Szyperski; Component Software, Beyond Object-Oriented Programming. Editorial Addison Wesley, 1997.

MLUM99 Marcus Lumpe. A pi-Calculus Based Approach for Software Composition. Bern University Ph. D. Thesis, 1999.

DBOX98 Don Box. Essential COM. Editorial Addison Wesley, 1998.

EROM99 Ed Roman. Mastering Enterprise JavaBeans. Editorial Wiley, 1999.

SMIC99 Sun Microsystems. Enterprise JavaBeans, Specification, v1.1. 24-Nov-1999.

DROG97 Dale Rogerson. Inside COM. Editorial Microsoft Press, 1997.

AGOR00 Alan Gordon. The COM and COM+, Programming Primer. Microsoft Technologies Series. Editorial Prentice Hall, 2000.

RHTS98 Reaz Hoque and Tarun Sharma. Programming Web Components. Editorial McGraw Hill, 1998.

PHOS00 Peter Herzum and Oliver Sims. Business Component Factory. John Wiley and Sons Inc., 2000.

RHIG96 Robert H. High Jr. Component Model for Managed Objects in Large-Scale Distributed

CUC'96 Component-Based Software Engineering. Editorial Cambridge University Press, 1998.

Agradecimientos:
Al Instituto Mexicano del Petróleo por las facilidades brindadas para este trabajo, en especial al Ing. Tomás Ramírez Maldonado.