Un modelo de interoperabilidad de objetos

Champredonde, Raúl[1]

Koldobsky, Mariano[2]

Director: De Giusti, Armando[3]

Laboratorio de Investigación y Desarrollo en Informática[4]

Departamento de Informática

Facultad de Ciencias Exactas

Universidad Nacional de La Plata

Resumen

En los últimos años, gran parte de la investigación sobre orientación a objetos ha tendido hacia los llamados objetos interoperables. Esto es, (aunque tal vez algo exagerado) la máxima expresión de los objetos, que les permite interactuar independientemente de la aplicación a la que pertenecen, del lenguaje de desarrollo, del sistema operativo y de la plataforma de hardware.

Este trabajo presenta un modelo que define una arquitectura de objetos que facilita la resolución de las cuestiones relacionadas a la interoperabilidad y portabilidad.

[1] L.I.D.I. Ayudante Diplomado Dedicación Semiexclusiva del Dpto. de Informática, Fac. de Cs. Exactas, U.N.L.P. rchampre@info.unlp.edu.ar

[2] Becario L.I.D.I. Auxiliar docente del Dpto. de Informática, Fac. de Cs. Exactas, U.N.L.P. mariano@info.unlp.edu.ar

[3] Director del L.I.D.I. Investigador Principal del CONICET. Profesor Titular Dedicación Exclusiva, Dpto. de Informática, Fac. de Cs. Exactas, U.N.L.P. degiusti@info.unlp.edu.ar

[4] L.I.D.I. Laboratorio de Investigación y Desarrollo en Informática, Dpto. de Cs. Exactas, U.N.L.P.
Calle 50 y 115 - 1er piso - (1900) La Plata - Buenos Aires
Tel/Fax: 54-21-22-7707
Tel: 54-21-4-2738
E-Mail: lidi@info.unlp.edu.ar

Introducción

Interoperabilidad de objetos

Objetos interoperables se llama a aquellos que pueden sobrepasar los límites usuales impuestos por los lenguajes de programación, los espacios de direcciones de los procesos, los sistemas operativos y las plataformas de hardware. Son el resultado de la convergencia de ciertas tendencias de las tecnologías de software, esto es, la evolución continua de la programación orientada a objetos en las áreas de objetos independientes del lenguaje, procesamiento distribuido y tecnologías de documentos compuestos.

Si bien la interoperabilidad de objetos es relativamente nueva, el procesamiento distribuido no lo es. Las tecnologías de procesamiento distribuido son las que, de alguna manera, proveen las bases para la interoperabilidad de objetos.

Gran parte de las realizaciones en objetos interoperables por el momento no son más que especificaciones, aunque ya hay ciertas tecnologías demostrables que implementan parcialmente interoperabilidad.

Como ejemplos de especificaciones e implementaciones de objetos interoperables más populares hasta el momento pueden mencionarse CORBA (Common Object Request Broker Architecture) de Object Management Group, SOM (System Object Model) de IBM, COM (Components Object Model) de Microsoft, PDO (Portable Distributed Objects) de NextStep, TalAE (Taligent Application Environment) de Taligent, ADB (Appware Distributed Bus) de Novell. *[BETZ94] [BETZ95]*

Estándares en objetos interoperables

Hasta el momento, las principales especificaciones de objetos interoperables son las del Object Management Group (OMG), IBM y Microsoft. *[ORFA95] [BETZ94]*

OMG es un consorcio, formado por más de 300 compañías de hardware y software, el cual es autor de la especificación de Common Object Request Broker Architecture (CORBA). CORBA especifica la arquitectura de un Object Request Broker (ORB), cuyo trabajo es permitir y regular la interoperabilidad entre objetos y aplicaciones.

El ORB de CORBA es una parte de una visión mayor llamada el Object Management Architecture (OMA), la que define una arquitectura de servicios y relaciones, junto con los modelos de objetos y referencias.

El System Object Model (SOM) de IBM es una de las principales implementaciones que respetan las especificaciones de CORBA.

Por otro lado, está Object Linking and Embedding (OLE) de Microsoft construido sobre la base del llamado Component Object Model (COM), el cual realiza algunas de las mismas funciones de un ORB pero a diferente escala y utilizando otras técnicas. La idea de Microsoft acerca de un modelo de objetos difiere bastante de la del resto de las compañías que incursionaron en objetos interoperables.

Un factor común a todos los pretendidos estándares de objetos interoperables es la restricción establecida por la necesidad de reutilizar lo hecho hasta el momento.

CORBA Y OMA

CORBA es la especificación de una arquitectura e interfaces que permiten que las aplicaciones hagan requerimientos a objetos en forma independiente y transparente a los lenguajes, sistemas operativos o ubicación. *[OMG91] [YATE94] [VINO93]*

En realidad CORBA es una parte de la llamada Object Management Architecture (OMA) la cual es la visión completa que tiene OMG de un ambiente distribuido.

CORBA enfoca sólo a la interacción de objetos y a los mecanismos que ella necesita, mientras que OMA *[OMG92a]* define una arquitectura extensa de servicios y relaciones dentro de un ambiente además de los modelos de objetos y referencias. *[OMG92b]*

OMA está construido sobre los servicios del ORB definido por CORBA. Éste provee el modelo de interacción de objetos para la arquitectura.

La especificación de CORBA describe el modelo de objetos de OMG, el cual sostiene que CORBA, al igual que todo OMA, es el clásico "clientes envían mensajes a servidores, y un mensaje identifica un objeto y cero o más parámetros".

El modelo de OMG separa estrictamente la interfaz de la implementación, ocupándose sólo de la primera. Esta aproximación surge de la necesidad que tienen los modelos de definir interfaces entre componentes sin preocuparse por el lenguaje de implementación.

Los objetos son identificados por referencias, las cuales son un tipo definido por la implementación que garantiza que una referencia identificará siempre al mismo objeto cada vez que sea utilizada.

La creación y destrucción de objetos se realiza dinámicamente como respuesta a un requerimiento.

Se permite herencia múltiple aunque, claro está, se refiere solamente a herencia de interfaces.

La herencia entre interfaces de objetos se especifica sintácticamente mediante el lenguaje de definición de interfaces (IDL: Interface Definition Languaje) de OMG.

El modelo de OMG tiene un concepto fuerte de tipos, los cuales son usados para restringir y caracterizar operaciones.

Arquitectura del ORB

El trabajo del ORB es manejar la interacción entre objetos clientes y servidores, lo cual incluye todas las responsabilidades de un sistema de procesamiento distribuido. Para eso la especificación de CORBA define una arquitectura de interfaces independientes de la implementación.

La arquitectura de CORBA consiste de tres componentes específicas: la interfaz con los clientes, la interfaz con la implementación y el corazón del ORB.

La interfaz con los clientes es la que provee a éstos las interfaces al ORB y a los objetos servidores. La misma está compuesta por la interfaz de invocación dinámica, la interfaz de *stubs* de IDL y la interfaz de servicios del ORB.

La interfaz de invocación dinámica es un mecanismo de especificación de requerimientos en tiempo de ejecución. La interfaz dinámica es necesaria cuando no se conoce el tipo de interfaz en tiempo de compilación.

En general, la interfaz de stubs consiste de pequeñas partes de código interfaz al lenguaje de máquina, las cuales son generadas de acuerdo a las definiciones del IDL y enlazadas dentro del programa cliente. Los stubs representan mapeos entre el lenguaje del cliente y la implementación de ORB, permitiendo que éste sea utilizado por clientes escritos en cualquier lenguaje para el que exista un mapeo.

Los servicios del ORB proveen funciones que pueden ser accedidas directamente por el código del cliente. Por ejemplo, la recuperación de una referencia a un objeto.

La interfaz con la implementación también tiene tres componentes. Uno de ellos son los servicios del ORB, el cual es compartido con la interfaz con los clientes. Los otros dos componentes son la interfaz de esqueletos IDL y el Object Adapter.

La interfaz de esqueletos IDL junto con la interfaz de stubs son discutidas luego. Por el momento diremos que a través de la interfaz de esqueletos el ORB llama a los esqueletos de los métodos de la implementación como respuesta a un requerimiento de un cliente.

El Object Adapter es el medio a través del cual las implementaciones de servidores acceden a la mayoría de los servicios del ORB. Estos servicios incluyen generación e interpretación de referencias a objetos, invocación de métodos, seguridad, activación, mapeo de referencias a implementaciones y registración de objetos.

El IDL

El propósito de un IDL es permitir la expresión de interfaces en forma independiente al lenguaje. [OMG94] [VALD95]

Este tipo de lenguaje abstracto y simbólico de especificación de interfaces es anterior a CORBA y a los sistemas orientados a objetos en general, ya que desde los primeros sistemas de llamados a procedimientos remotos (RPC: Remote Procedure Call), estos lenguajes son conocidos como IDLs.

El objetivo del IDL se logra por medio de un mapeo entre la sintaxis del IDL y cada uno de los lenguajes usados para la implementación de objetos clientes y servidores.

Los clientes y servidores no necesitan ser implementados en el mismo lenguaje, y de hecho se presupone que no lo están.

CORBA IDL es un lenguaje que recuerda en muchas de sus construcciones a las de C++. El IDL tiene las mismas reglas léxicas que C++ e introduce algunas palabras claves nuevas propias de las necesidades de los sistemas distribuidos.

SOM / DSOM

SOM provee un modelo de objetos que soporta herencia, encapsulamiento y polimorfismo. SOM respeta totalmente lo especificado por CORBA (CORBA-compliant). *[CAMP95] [VALD95]*

Implementa un ORB altamente optimizado que provee interoperabilidad entre lenguajes y soporta compatibilidad binaria en la herencia y el uso de los objetos.

Las clases SOM son definidas usando el CORBA IDL.

SOM extiende lo especificado por CORBA proveyendo herencia y polimorfismo de implementación, y metaclases que son manipuladas como objetos de primer orden. Soporta además tres tipos de resolución de métodos, así como la creación de clases y adición de métodos dinámicos en tiempo de ejecución.

DSOM no implementa un modelo de objetos o un runtime, sino que es un framework construido usando SOM.

SOM está implementado desde 1991 en OS/2 2.0, y actualmente se encuentra en AIX, Windows y Mac System 7. En los próximos meses se implementará en otras plataformas UNIX y en Novell Netware.

Actualmente, se encuentran versiones interoperables del framework distribuido (DSOM) para SOM de AIX, OS/2 Warp y Windows.

SOM es también la base para OpenDoc, tecnología de aplicación compuesta de Component Integration Laboratories (CIL) y para Taligent Application Environment (TalAE) de Taligent (consorcio formado por Apple, IBM y Hewlett Packard).

COM

COM de Microsoft es, a pesar de ser la misma una de las empresas participantes de OMG, un modelo alternativo a CORBA. *[WILL95] [ORFA95]*

El modelo de objetos de COM respeta el principio de encapsulamiento de la orientación a objetos. Los clientes no manipulan los objetos directamente. Los objetos exponen a sus clientes, varios conjuntos de punteros a funciones, conocidos como "interfaces".

A diferencia de CORBA, las clases de COM no son clases en el sentido de orientación a objetos. Las interfaces de COM no tienen estado y no pueden ser instanciadas para crear objetos. Las interfaces de COM son simplemente conjuntos de funciones agrupadas.

Como CORBA, COM separa la interfaz de las implementaciones y requiere que todas las interfaces sean declaradas usando un IDL. El IDL de Microsoft, basado en DCE (Distributed Computing Environment), no es CORBA-complaint.

COM soporta la herencia de una manera bastante distinta a la tradicional de orientación a objetos. El mecanismo es la agregación o composición de objetos, con total acceso a sus subobjetos.

La versión actual de COM se encuentra embebida en el corazón de OLE 2.0. Actualmente, junto con Digital Equipment, Microsoft se encuentra especificando la nueva versión de COM distribuido, que va a estar finalizada cuando Microsoft libere la versión beta de la próxima generación de Windows NT llamado Cairo.

A diferencia del COM actual, el nuevo COM va a proveer transparencia local/remota entre componentes a través de la red, construida sobre el DCE RPC.

El modelo de objetos propuesto

Introducción

El objetivo de este modelo es proveer interoperabilidad de objetos sin las precondiciones determinadas por las herramientas existentes, lo que redunda en una mayor flexibilidad y sencillez en favor del modelo.

El modelo de objetos propuesto es un modelo clásico en lo que se refiere a la herencia, polimorfismo, binding dinámico, etc. Se basa en la idea de que cada objeto es una entidad independiente que se comunica mediante mensajes con los demás objetos.

Cada objeto tiene un espacio de direccionamiento propio e independiente del espacio de los demás, e interactúa con ellos sólo por medio de los métodos definidos en su interfaz. De esta manera, desaparece la relación entre objetos dada por la pertenencia de los mismos a una aplicación.

Además de un espacio de direcciones propio e independiente, un objeto puede ejecutarse en cualquier máquina de una red heterogénea.

Los objetos atributos, por el simple hecho de ser objetos, también tienen su propio espacio de direcciones y por lo tanto pueden ejecutarse en cualquier máquina. En particular, un objeto puede tener atributos que se ejecuten en máquinas distintas a la que corre él mismo.

De la misma manera, un objeto que se ejecute en una computadora puede ser heredero de otro que corra sobre otra computadora.

Estos últimos dos conceptos potencian notablemente el modelo dando a cada objeto capacidades de ejecución que exceden los límites tradicionales establecidos por las arquitecturas de hardware.

Este modelo incorpora, entre otras, las siguientes características: herencia múltiple, identificación única de objetos, mensajes sincrónicos y asincrónicos.

Uno de los elementos importantes para un modelo de objetos interoperables es la identificación de instancias de objetos. En este caso se ha optado por una identificación única universal.

Los objetos se ejecutan en paralelo y, si bien hasta el momento los modelos de objetos se han arreglado con mensajes similares a un llamado a procedimiento, este modelo provee mensajes sincrónicos y asincrónicos dando de esta manera mayor flexibilidad a la comunicación y sincronización entre objetos. Los mensajes son manejados de acuerdo a su prioridad. Por ejemplo, un mensaje originado por una interrupción tendrá prioridad por sobre cualquier otro tipo de evento.

La arquitectura

Los componentes básicos que integran la arquitectura propuesta se muestran en la figura 1.

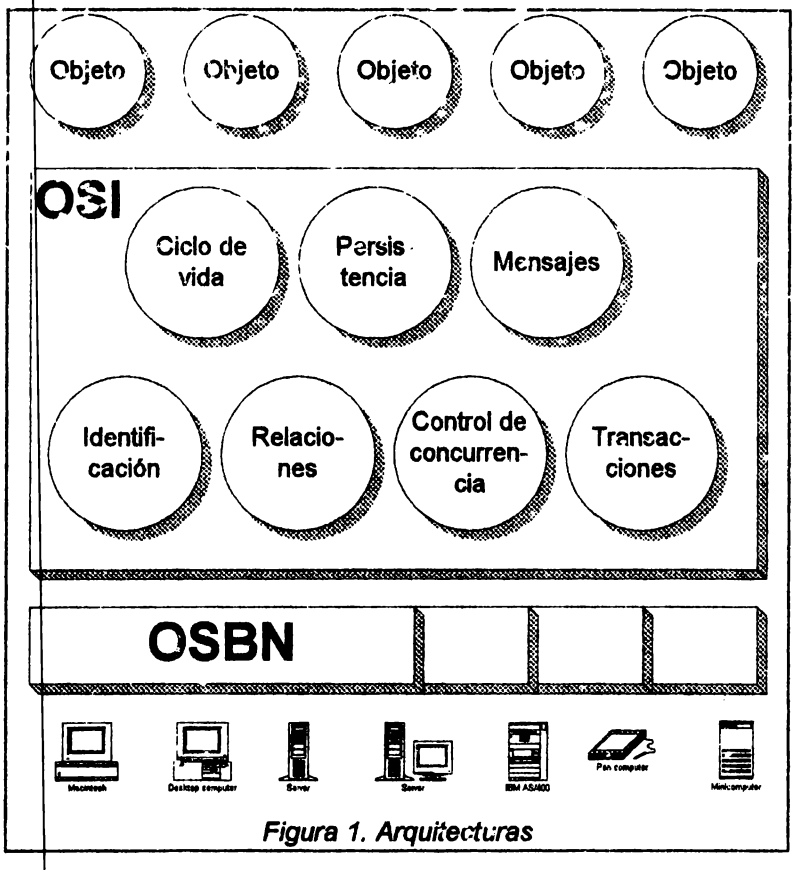

Figura 1. Arquitecturas

El OSBN es la capa no portable relacionada directamente con las arquitecturas de hardware. Por eso habrá una implementación para cada plataforma.

El mismo está compuesto por los objetos necesarios para el manejo del hardware de cada plataforma a la cual se desee portar. Existe también un conjunto de objetos que interactúan con los de bajo nivel y proveen la base para una implementación del modelo de objetos previamente descripto.

Los objetos de bajo nivel componen una capa acotada, a la que llamamos Objetos de Servicios de Bajo Nivel (OSBN), la cual es dependiente de la plataforma, pero necesaria para proveer portabilidad a los objetos de más alto nivel.

Sobre el OSBN, se encuentra un conjunto de objetos que proveen los servicios de identificación, persistencia, ciclo de vida, administración de mensajes, transacciones, relaciones entre objetos, control de concurrencia. Este conjunto de objetos es llamado Objetos de Servicios de Interoperabilidad (OSI).

El servicio de Identificación administra las identificaciones de los objetos locales e interactúa con los servicios de Identificación remotos para lograr identificar objetos residentes en cualquier lugar de la red.

El servicio de Persistencia permite que instancias de objetos sean almacenadas para su posterior recuperación.

La construcción y destrucción de instancias son administradas por el servicio de Ciclo de vida. La construcción de un objeto se realiza en forma dinámica ante un requerimiento mientras que la destrucción es por medio de un mecanismo similar al garbage collector.

El servicio de Mensajes es el encargado de asegurar que los objetos reciban los requerimientos que se les solicita, teniendo en cuenta el tipo y la prioridad de los mensajes.

La coordinación entre componentes recuperables es llevada a cabo por el servicio de Transacciones.

El servicio de Relaciones lleva el control de los enlaces entre objetos para mantener un seguimiento de las instancias que permita determinar si tienen referencias o no, e interactúa con el servicio de Ciclo de vida.

El servicio de Control de concurrencia administra la política de scheduling entre los diferentes objetos y la suspensión momentánea de su ejecución en espera de sincronismo u otro tipo de evento.

Sobre estos servicios, viven las instancias de objetos que se comunican entre sí a través de ellos. Cabe aclarar que la representación de las instancias de objetos de la figura 1, no sólo se refiere a las instancias locales a un procesador, sino al conjunto total de instancias alcanzables en una red heterogénea.

En el caso de arquitecturas multiprocesador, se ve a cada procesador como uno más de una red heterogénea de procesadores. Por eso se monta un OSI con una identificación única sobre cada procesador más que sobre cada computadora.

Especificación de los Objetos de Servicio de Interoperabilidad

Servicio de Identificación

Objetivo

Mantener las identificaciones de las clases locales y remotas, y las identificaciones de las instancias locales.

Tareas

Mantenimiento de colecciones de identificaciones únicas de objetos locales.

Mantenimiento de colecciones de identificaciones de objetos remotos conocidos.

Consulta de identificaciones.

Búsqueda de identificaciones de objetos remotos que no están en la colección de identificaciones de objetos remotos conocidos.

Atención de pedidos remotos de identificación de objetos locales.

Atributos

Colección de identificaciones locales y el nombre de la clase asociada

Colección de identificaciones y nombres de clases remotas instanciadas por una instancia local. (Nc es cacheable)

Colección volátil de identificaciones y nombres de clases remotas conocidas (en algún momento han sido instanciadas por una instancia local, pero actualmente no lo son) (Es cacheable)

Colección de identificaciones de instancias locales

Colección de identificaciones de los servicios locales

Colección volátil de identificaciones de servicios remotos.

Métodos

AgregarClaseLocal (NombreClase): IdClase

Agrega el nombre de la clase NombreClase en la colección de clases locales junto con una identificación única que crea para esa clase. Retorna la identificación IdClase que creó.

BorrarClaseLocal (NombreClase)

Borra la clase NombreClase de la colección de clases locales.

ConsultarClasePorID (IdClase): NombreClase, Resultado

Consulta local o remotamente y devuelve, si puede, el Nombre de la clase IdClase. IdClase es la identificación única de una clase dentro de una máquina en particular (IdClase + IdInstanciaOSI + IdMáquina). Si por alguna razón no encuentra la clase IdClase indica la excepción en Resultado.

ConsultarIdPorClase (NombreClase, IdDominio, Criterio): IdClase, Resultado

Busca en la colección de identificaciones de clases locales el NombreClase. Si no lo encuentra, busca en el dominio de identificación IdDominio. Si la encuentra devuelve la identificación de la clase en IdClase y la agrega a la colección de clases remotas conocidas. En cualquier caso indica el resultado de la operación en Resultado. En Criterio se indica si la búsqueda se detiene cuando encuentra una local, cuando encuentra la primera, la más cercana, la más rápida, etc.

ConsultarIdPorClase (NombreClase, IdDominio): ColeccionIdClase, Resultado

Busca en todo el dominio IdDominio las identificaciones de la clase NombreClase y devuelve una colección de identificaciones de clases ColeccionIdClase. Agrega todas las identificaciones que encontró en la colección de identificaciones de clases remotas conocidas. Devuelve el resultado de la operación en Resultado

AgregarClaseRemota (NombreClase, IdClase)

Agrega la clase remota NombreClase junto a la identificación IdClase en la colección de clases remotas en uso.

AgregarInstanciaLocal (IdClase): IdInstancia

Crea una identificación única para una instancia de la clase IdClase y lo agrega a la colección de identificaciones de instancias locales.

BorrarInstanciaLocal (IdInstancia)

Elimina la instancia IdInstancia de la colección de identificaciones de instancias locales.

ConsultarInstanciasLocales : ColecciónIdInstancias

Devuelve una colección de identificaciones de todas las instancias locales.

ConsultarServicios : ColecciónServiciosLocales

Devuelve una colección de identificaciones de los servicios locales a los que se puede tener acceso remotamente.

ConsultarServicioPorNombre (NombreServicio) : IdServicio

Devuelve la identificación del servicio NombreServicio.

AgregarServicioLocal (NombreServicio, IdServicio)

Agrega en la colección de servicios locales el servicio de nombre NombreServicio e identificación IdServicio.

AgregarServicioRemoto (NombreServicio, IdServicio)

Agrega en la colección de servicios remotos el servicio de nombre NombreServicio e identificación IdServicio.

Servicio de Mensajes

Objetivo

Encargado de rutear los mensajes.

Tareas

Rutear mensajes entre objetos independientemente de si los mismos son locales o remotos.

Cuando recibe un mensaje de instanciación, interactúa con los servicios de identificación, ciclo de vida y relaciones para crear la instancia. Rutea el mensaje a la nueva instancia poniéndoselo en la cola y devuelve la identificación de la instancia al objeto creador.

Cuando recibe un mensaje asincrónico a una instancia existente se lo encola.

Cuando recibe un mensaje sincrónico además de rutear le avisa al servicio de control de concurrencia que duerma la instancia llamadora.

Queda claro que un mensaje de instanciación debe ser un mensaje sincrónico.

Cuando le pide al servicio de ciclo de vida que cree una instancia luego deberá pedirle al de relaciones que cree la relación (con el IdInstancia que la devuelve el mensaje de creación de instancia solicitada al ciclo de vida).

Un mensaje tenga o no identificación de instancia destino, el servicio de mensajes debe fijarse si hay relación entre fuente y destino. Si no la hay la debe crear.

Métodos

AtenderMensaje (IdInstanciaFuente, IdInstanciaDestino, NombreClase, Mensaje, Sincronía, TimeOut, Parámetros) : IdInstancia

Si la identificación de la instancia destino IdInstanciaDestino es nula envía el mensaje AtenderInstanciación y recibe la identificación de la instancia IdInstancia.

Encola el mensaje Mensaje en la cola de la instancia IdInstanciaDestino con los parámetros Parámetros e indica al servicio de control de concurrencia que la instancia

IdInstanciaDestino tiene mensaje en la cola. En caso de que Sincronía indique que el mensaje es sincrónico le avisa al servicio de control de concurrencia que duerma la instancia IdInstanciaFuente con un time out TimeOut.

AtenderInstanciación (IdInstanciaFuente, NombreClase, TimeOut, Parámetros) : IdInstancia

Le pide al servicio de Identificaciones la identificación IdClase de la clase NombreClase. Si la instancia a crear será local le pide al servicio de Ciclo de Vida que cree una instancia de la clase IdClase. Si la instancia a crear será remota le pide al servicio de Ciclo de Vida remoto que cree una instancia de la clase IdClase. En ambos casos le pide al servicio de Relaciones que cree una relación entre IdInstanciaFuente e IdInstancia, le encola el mensaje de inicialización a la instancia creada y devuelve el IdInstancia.

ResponderMensajeSincrónico (IdInstanciaFuente,

IdInstanciaDestino) : Parámetros, Respuesta

Un mensaje sincrónico es respondido utilizando este mensaje, que completa la llamada entre IdInstanciaFuente e IdInstanciaDestino. En caso de time out el servicio de control de concurrencia envía este mensaje con una indicación de time out en Respuesta, y el de mensajes le envía un Nack al IdInstanciaFuente.

Nota: Cada objeto tiene un atributo cola de mensajes. El algoritmo de selección depende de la clase de cola que se instancie en ese atributo. Cuando llega un mensaje para una instancia, el servicio de mensajes le dice al de control de concurrencia que esa instancia tiene mensajes en la cola como para que este lo pase a estado ready.

Servicio de Ciclo de Vida

Objetivo

Crear y destruir instancias de clases.

Tareas

Crear una instancia interactuando con los OSBN. Para esto le pide una identificación al servicio de identificaciones.

Destruir una instancia.

Métodos

CrearInstanciaDeClase (IdClase) : IdInstancia, Resultado

Dada la identificación de clase IdClase le pide al servicio de identificación una identificación de instancia por medio del mensaje AgregarInstanciaLocal e interactúa con los OSBN para levantar el código de la clase. Retorna la identificación de la instancia y el resultado de la operación.

MatarInstanciaLocal (IdInstancia)

Mata la instancia IdInstancia

Servicio de Relaciones

Objetivo

Mantener las relaciones entre objetos independientemente de si son locales o remotos.

Garbage Collector de relaciones cortadas.

Tareas

Agregar relaciones.

Eliminar relaciones.

Nota: Hay tres tipos de relaciones:

1) padre-atributo: el padre es fuente y el atributo destino. Un atributo no puede ser destino de otro objeto que no sea su padre.

2) objeto local a método: el objeto poseedor del método es fuente y el objeto local al método es destino.

3) objeto con objeto: el que crea la relación es el fuente y el otro es el destino. El destino puede ser destino de otras relaciones.

Para los tres casos una instancia se mata cuando se corta la última relación de la cuales destino. Las instancias de los OSI no se matan aunque sean último destino. Tampoco los del OSBN.

Atributos

Colección de relaciones

Métodos

AgregarRelación (IdInstanciaFuente, IdInstanciaDestino)

Agrega una relación entre las instancias IdInstanciaFuente e IdInstanciaDestino a la colección de relaciones.

BorrarRelación (IdInstanciaFuente, IdInstanciaDestino)

Si la instancia IdInstanciaDestino es local y es la última relación de la cual esa instancia es destino le avisa el servicio de ciclo de vida que elimine la instancia.

Si la instancia IdInstanciaDestino es remota le envía el mensaje BorrarRelación al servicio de relaciones remoto.

En ambos casos elimina la relación de la colección de relaciones.

Servicio de Control de Concurrencia

Objetivo

Administrar los estados de las instancias

Implementar la política de scheduling de CPU

Tareas

Administrar las colas de scheduling

Atributos

Cola de objetos corriendo

Cola de objetos dormidos

Pulsos

Métodos

SeleccionarObjetoADarCPU : IdInstancia

Selecciona de la cola de objetos corriendo el objeto a darle la CPU, según la prioridad y el método de selección utilizado.

DormirObjeto (IdInstancia, TimeOut)

Pasa una instancia IdInstancia de la cola de objetos corriendo a la cola de objetos dormidos con su TimeOut en relación a los Pulsos absolutos.

DespertarObjeto (IdInstancia)

Pasa la instancia IdInstancia de la cola de objetos dormidos a la cola de objetos corriendo.

Pulso

Mensaje de un OSBN clock. Incrementa Pulsos. Analiza la cola de objetos dormidos para ver si se produjo un time out. Si se produjo llama a DespertarObjeto y le envía un ResponderMensajeSincronico al servicio de Mensajes con una indicación de time out en Respuesta.

AgregarInstancia (IdInstancia)

Encola en la cola de objetos corriendo la instancia IdInstancia.

Comportamiento de los OSI ante requerimientos habituales

Instanciación de una clase

Supongamos que un objeto A desea instanciar uno de sus atributos de clase C, digamos B. A le envía un mensaje a B. El servicio de Mensajes atrapa el mensaje, examina sus parámetros para ver si tiene la identificación de la instancia destino. Como obviamente no la tiene por tratarse de una instanciación, le pide al servicio de Identificaciones que ubique a la clase C.

Si la puede ubicar localmente el servicio de Mensajes le pide al de Ciclo de Vida que cree la instancia B de la clase C; si la ubica en forma remota se lo pedirá al servicio remoto de Ciclo de Vida.

El servicio de Ciclo de Vida le pide al servicio de Identificaciones una identificación para la instancia B, la crea y le responde al servicio de Mensajes con la identificación de la instancia B.

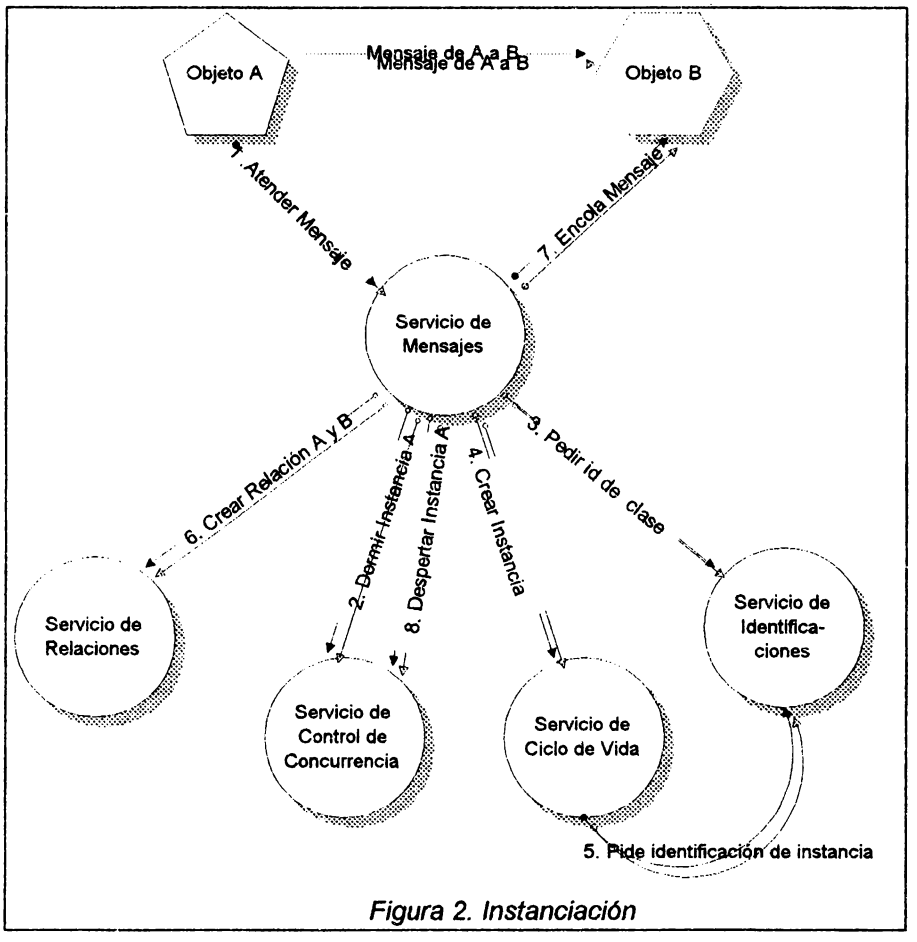

Figura 2. Instanciación

El servicio de Mensajes le pide al de Relaciones que cree la relación entre A y B, le encola el mensaje de inicialización (todos los objetos deben tener al menos un método de inicialización) y devuelve la identificación de B.

Destrucción de una instancia

Si un objeto envía un mensaje de destrucción a otro, el servicio de Mensajes le solicita al servicio de Relaciones que elimine la relación entre la instancia fuente y destino.

Cuando el servicio de Relaciones detecta que una instancia no tiene relaciones le solicita al servicio de Ciclo de Vida que elimine la instancia.

El servicio de Ciclo de Vida le dice al servicio de identificaciones que elimine la identificación de la instancia.

Conclusiones

Se ha presentado un modelo de objetos entre cuyas principales características se pueden mencionar la interoperabilidad, la portabilidad, la independencia de la implementación de productos previos.

El concepto básico que proporciona una gran potencia al modelo es la independencia del espacio de direcciones de los objetos. Esto permite que los mismos puedan ejecutarse en cualquier computadora de una red heterogénea. Un ejemplo de esto sería un atributo de un objeto que se ejecuta en una computadora remota.

Referencias

[BETZ94] Betz, M., *Interoperable Objects. Laying the foundation for distributed-object computing*. Dr. Dobb's Journal, October 1994.

[BETZ95] Betz, M., *OMG's CORBA*. Dr. Dobb's Special Report, Winter 1994/1995

[CAMP95] Campagnoni, F.R., *IBM's System Object Model*. Dr. Dobb's Special Report, Winter 1994/1995.

[OMG91] *The Common Object Request Broker: Architecture and Specification*, Revision 1.1, December 6, 1991. OMG TC Document 91.12.1.

[OMG92a] *Object Management Architecture Guide*, revision 2.0, Second Edition, September 1, 1992. OMG TC Document 92.11.1.

[OMG92b] *Object Services Architecture*, revision 6.0, October 1992. OMG TC Document 92.8.4.

[OMG94] *OMG RFP Submission: IDL C++ Language Mapping Specification*, August 1994. OMG Document 94.8.2.

[ORFA95] Orfali, R.,Harkey, D., *Client/Server with Distributed Objects*. Byte, April 1995.

[VALD95] Valdes, R., *Implementing Interoperable Objects*. Dr. Dobb's Special Report, Winter 1994/1995.

[VINO93] Vinoski, S., *Distributed Object Computing with CORBA. C++ Report*. August 1993.

[WILL95] William, S., Kindel, C., *The Component Object Model. The foundation for OLE services*. Dr. Dobb's Special Report, Winter 1994/1995.

[YATE94] Yates,R.B., *Estándares para uso de objetos en sistemas distribuidos*. Segundas Jornadas Nacionales sobre Tecnología de OO, Buenos Aires, Noviembre 1994.