

Interoperabilidad entre Servidores de Aplicaciones Heterogéneos

Raul Ruggia

InCo-FIng-UDELAR
Montevideo, Uruguay
ruggia@fing.edu.uy

Jorge Besil

InCo-FIng-UDELAR
Montevideo, Uruguay

Carla Pais

InCo-FIng-UDELAR
Montevideo, Uruguay
cpais@fing.edu.uy

Dario Sande

InCo-FIng-UDELAR
Montevideo, Uruguay
dsande@fing.edu.uy

Resumen

La integración de sistemas, tanto en redes locales como a través del Web, ha generado fuertes requerimientos para resolver la interoperabilidad entre plataformas heterogéneas, especialmente a nivel de servidores de aplicaciones.

Por otro lado, las plataformas actualmente dominantes para el desarrollo de sistemas con arquitectura de varios niveles (presentación, lógica, almacenamiento), son CORBA, Microsoft .NET, J2EE, con una fuerte tendencia hacia las dos últimas.

En este artículo se analizan diferentes opciones de interoperabilidad entre estas plataformas tecnológicas, en conexiones de tipo cliente/servidor, tanto sincrónicas como asíncronas. Asimismo se presenta un sistema para ensayos de interoperabilidad, que permite realizar pruebas de este tipo en forma sistematizada, y sobre la cual se realizaron varios de los análisis descriptos.

Palabras clave

Interoperabilidad, J2EE, CORBA, Microsoft .NET, Web Services, RMI-IIOP.

INTRODUCCIÓN

En los últimos años han cobrado importancia los sistemas cooperativos e integradores, que apuntan a brindar servicios con valor agregado reutilizando sistemas existentes [11]. Aún más, en muchos casos se propone especialmente que esta interacción se realice a través del Web [5].

Estos sistemas integradores deben ser capaces de interactuar con otros sistemas en plataformas diferentes, en particular de servidor de aplicaciones. Por lo tanto, para desarrollar estos sistemas resulta imprescindible resolver la interoperabilidad entre sistemas basados en plataformas heterogéneas.

Los mecanismos de interoperabilidad entre plataformas heterogéneas pueden ser muy variados por diferentes razones, entre ellas: la variedad de las plataformas a conectar, las diferentes formas de interacción (p.ej. sincrónica y asíncrona), la existencia de requerimientos sobre la interacción (p.ej. atomicidad transaccional, niveles de robustez y performance, compatibilidad con lenguajes u otras platafor-

mas), si se utilizan mecanismos estándares previstos para la interacción u opciones ad-hoc.

En este artículo se aborda una parte de esta problemática, y se analizan opciones de interoperabilidad en el siguiente marco: *Interoperabilidad J2EE-CORBA y J2EE-NET, en modalidades sincrónica y asíncrona, priorizando el uso de mecanismos estándares previstos para tal función.*

Esta selección de casos tuvo como objetivo concentrar el análisis en las situaciones que se identifican como más probables en el futuro inmediato, así como para las que existen productos industrializados. Asimismo se consideró de importancia que el mecanismo de interoperabilidad permitiera realizar la integración de los sistemas a través de Internet.

El conjunto complementario al de los casos tratados es sumamente amplio, pero se destacan: la interoperabilidad entre CORBA y Microsoft [29], el uso de tecnologías de EAI (Enterprise Application Integration) e *Integration Brokers* [10], productos propietarios para la integración de servidores particulares, y el uso de *middleware* básico [24].

Asimismo, y como forma de contar con un ambiente que permita trabajar en forma sistemática sobre esta área, se desarrolló un sistema que permite la realización de ensayos de interoperabilidad entre servidores de aplicaciones heterogéneos. Los casos de interoperabilidad analizados se implementaron en el contexto de este sistema.

Este artículo presenta el análisis de las opciones de interoperabilidad entre los servidores de aplicaciones heterogéneos. Asimismo se presenta una descripción del sistema para la realización de ensayos, llamado "Matriz de Interoperabilidad". Una descripción más detallada puede encontrarse en [3].

El resto del artículo se estructura de la siguiente forma. En la próxima sección se presentan el conocimiento existente relativo al problema abordado. Luego se presentan los análisis de los casos de interoperabilidad, incluyendo la descripción de la arquitectura implementada para los mismos. En la siguiente sección se describe la plataforma de ensayos

implementada. El artículo termina con Conclusiones y las Referencias.

CONOCIMIENTO EXISTENTE.

A continuación se describen varios conceptos y técnicas ya desarrollados.

CORBA [6] (Common Object Request Broker Architecture) es un estándar para objetos distribuidos que ha sido desarrollado por el OMG (Object Management Group). CORBA proporciona los mecanismos a través de los cuales los objetos hacen peticiones y reciben respuestas, definidas como ORBs (Object Request Broker) de forma transparente para el sistema. El ORB de CORBA proporciona interoperabilidad entre diferentes objetos, eventualmente en diferentes lenguajes y en sistemas operativos distintos.

IIOP (Internet Inter.-ORB Protocol) es el protocolo de comunicación utilizado por CORBA, que hace posible la comunicación entre ORBs CORBA (eventualmente heterogéneos) que soporten aplicaciones distribuidas.

Notification Service es la especificación de OMG para el servicio de gestión de eventos de CORBA.

J2EE [14] (Java 2 Enterprise Edition) es una especificación diseñada por Sun para plataforma de sistemas con arquitectura de múltiples capas basados en Java. Esta especificación incluye la de componentes en el servidor (Enterprise Java Beans), servicio de mensajería (Java Message Service), y Message Driven Beans que combina las funcionalidades de EJB y JMS. J2EE provee herramientas para: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI), funciones de correo electrónico (JavaMail), aplicaciones Web (JSP y Servlets), etc. J2EE es una especificación, sobre la cual se desarrollan productos concretos [2][19] que pueden cubrir diferentes partes de dicha especificación.

RMI (Remote Method Invocation) es el mecanismo de invocación remota de Java, y permite crear aplicaciones Java en las cuales es posible invocar métodos de objetos remotos Java desde otra máquina virtual, eventualmente desde otro host. **JRMP** (Java Remote Method Protocol) es el protocolo utilizado por RMI, que permite la invocación de métodos sobre objetos remotos.

RMI sobre IIOP [18] combina las características de RMI e IIOP permitiendo que Java RMI utilice IIOP como protocolo de comunicación, en lugar de JRMP.

.NET [23] es un conjunto de tecnologías propuesta por Microsoft, que entre otras características permite el desarrollo y ejecución de sistemas multi-lenguaje con arquitectura de múltiples capas. El paquete de .NET incluye el Frame-

work.NET, el cual implementa el soporte de ejecución CLR (Common Language Runtime) y compilación de .NET, así como Visual Studio, que es la herramienta de desarrollo. .NET muestra una fuerte orientación hacia XML y Servicios Web. Una característica importante es que permite la interacción multi-lenguaje.

J2EE Connector Architecture.

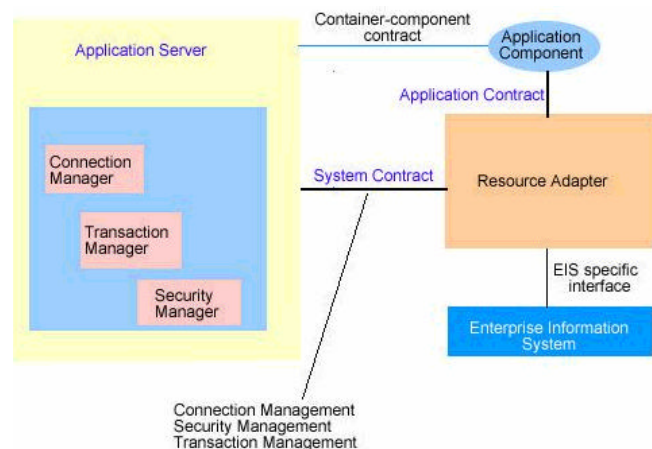
J2EE Connector Architecture (JCA) [13] es una especificación desarrollada por el Java Community Process con el objetivo de resolver la integración entre servidores de aplicaciones J2EE y sistemas externos tipo Enterprise Information System (EIS). Actualmente se encuentra en la versión 1.5 de JCA, la cual se incorporó a a partir de la versión 1.3 de J2EE.

Usando JCA, los EJBs que se ejecutan en el servidor de aplicaciones J2EE pueden acceder a los sistemas externos mediante mecanismos homogéneos y bien definidos, b- grandando transparencia y portabilidad con respecto al EIS.

La especificación de JCA incluye tres tipos de *Contracts* que homegeinizan la interfase hacia los sistemas externos. Estos *Contracts* son: (i) *Connection Management contract*, que provee los mecanismos de pool de conexiones para soportar la escalabilidad en aplicaciones que acceden al EIS; (ii) *Transaction Management contract*, que soporta acceso transaccional al EIS, y (iii) *Security contract*, que extiende los *contracts* de manejo de transacciones para realizar un acceso seguro al EIS.

La conexión entre el Servidor de Aplicaciones y el EIS se realiza a través de los llamados *Resource Adapters*, que implementan los *contracts* anteriores en el EIS, sirviendo de intermediarios entre el Servidor de Aplicaciones y el EIS. La Figura 1 muestra la arquitectura de JCA.

Figura 1: J2EE Connector Architecture [13].



La JCA permiten a los EIS proveer conexiones estandarizadas para la invocación de sus métodos desde sistemas desarrollados en servidores de aplicaciones J2EE. De esta forma, al implementar conectores según la JCA, los servidores de aplicaciones se aseguran conectividad con diferentes EISs.

Para que un EIS pueda conectarse a un servidor de aplicaciones J2EE que soporta JCA, resulta suficiente que el EIS provea un *resource adapter*. Por otro lado, un servidor de aplicaciones puede utilizar múltiples *resource adapters*, por ejemplo uno por cada EIS.

Actualmente existen, registrados en Sun, diez Servidores de Aplicaciones J2EE que soportan JCA, y una cantidad interesante de productos externos que implementan *resource adapters*.

Los conectores JCA han sido pensados para resolver el acceso a sistemas legados, especialmente sistemas transaccionales en mainframes, en una modalidad sincrónica request/reply, lo cual se refleja en la mayoría de los conectores existentes.

La JCA permite asociar propiedades de seguridad, transaccionalidad y pool de conexiones del servidor J2EE con las de EIS.

Sin embargo, la versión actualmente implementada de JCA (1.0) presenta limitaciones, entre las cuales se destacan: (i) No cuenta con mecanismos nativos para trabajar con metadata, lo cual obliga al desarrollo ad-hoc de estos módulos, comprometiendo el valor de la JCA para el acceso a sistemas externos e información heterogénea. (ii) No cuenta con soporte nativo para XML, lo cual limita en forma importante la interfase con sistemas que generan o leen datos en este formato. (iii) Solo soporta acceso sincrónico y unidireccional.

Estas tres limitaciones están anotadas para ser resueltas en las próximas versiones de JCA (1.5 y 2.0) [20], tendiendo a incorporar conexiones asíncronas y bi-direccionales.

En cuanto a su uso en la interoperabilidad con .NET, si bien no se han desarrollado productos orientado a este tipo de conexión, la JCA ofrece una opción, a desarrollar, para la interoperabilidad entre Servidores de Aplicaciones J2EE y Microsoft. Esto es debido a que ambas plataformas utilizan el estándar DTP (*Distributed Transaction Processing*) definido por X/Open.

Web Services.

Web Services [34] es un conjunto de estándares basados en XML (SOAP, WSDL, UDDI), que permite la interoperabi-

lidad entre sistemas con HTTP como protocolo de transporte.

La comunicación se basa en el intercambio de paquetes SOAP (Simple Object Access Protocol) [30]. SOAP es un protocolo liviano basado en XML que permite el intercambio de información. SOAP está compuesto por cuatro componentes básicos: un "envoltorio" que define cómo describir los mensajes y cómo procesarlos, reglas para serializar (codificar) instancias de datos, una especificación de cómo representar invocaciones remotas y sus respuestas y una convención para relacionar mensajes con un protocolo de transporte (habitualmente HTTP).

Por su parte, el WSDL (Web Service Description Language) permite especificar los contratos de cada servicio en una sintaxis basada en XML. Esta especificación permite conocer por parte de los consumidores los métodos exportados, así como los parámetros y datos retornados.

UDDI: provee un mecanismo de páginas amarillas que permite la localización de servicios a través de Internet.

Actualmente las plataformas J2EE y .NET son las principales que incorporan Web Services en forma prácticamente nativa.

Los Web Services se caracterizan por permitir conexiones tanto sincrónicas request/reply como asíncronas, conectando aplicaciones no necesariamente J2EE y usar HTTP como protocolo, lo cual lo hace muy apropiado para conexiones en Internet. Por el momento no incluye mecanismos nativos de seguridad y control de transacciones, este último en desarrollo (ver en Conclusiones).

INTEROPERABILIDAD ENTRE SERVIDORES HETEROGENEOS.

Los diferentes casos de interoperabilidad pueden caracterizarse mediante tres dimensiones básicas:

- Servidor de Aplicaciones (p.ej. CORBA, J2EE, .NET).
- Rol del sistema (cliente o servidor).
- Tipo de interacción (sincrónica, asíncrona).

Asimismo, para cada caso, se pueden analizar propiedades tales como: si es un mecanismo nativo, si sigue un estándar o es ad-hoc, si traslada contexto transaccional del cliente al servidor, nivel de robustez.

En este artículo se analizan diferentes casos y se describen las experiencias de interoperabilidad consistentes en cruzamientos de invocaciones cliente/servidor, sincrónicas y asíncronas, entre las plataformas CORBA, J2EE y Microsoft,

utilizando los productos CORBA Orbacus, J2EE JBoss, y Microsoft .NET.

La Tabla 1 resume las opciones analizadas para invocaciones sincrónicas. Se marca con (*) las opciones implementadas.

Tabla 1. Opciones de Interoperabilidad Sincrónica

Cliente → Servidor ↓	Microsoft .NET	CORBA	J2EE
MS .NET	Nativa (*)	No analizado	Web Services (*). JCA. Adaptadores propietarios.
CORBA	No analizado	Nativa (*)	RMI sobre IIOP (*).
J2EE	Web Services (*). Adaptadores propietarios.	RMI sobre IIOP (*).	Nativa (*)

La Tabla 2 resume las opciones analizadas para invocaciones asíncronas. Se marca con (*) las opciones implementadas.

Tabla 2. Opciones de Interoperabilidad Asíncrona

Cliente → Servidor ↓	Microsoft .NET	CORBA	J2EE
MS .NET	Nativa (*)	No analizado	Web Services + AXIS (*).
CORBA	No analizado	Nativa (*)	Gateway (*)
J2EE	Web Services (*)	Gateway (*)	Nativa (*)

Interoperabilidad entre J2EE y Microsoft .NET.

La interoperabilidad entre J2EE y .NET puede ser resuelta de diferentes formas.

Las dos opciones más estandarizadas son los Web Services y los J2EE Connector Architecture (JCA), descritos en la sección anterior.

Otras opciones son los adaptadores propietarios, por ejemplo ActiveX Bridge y Ja.NET. ActiveX Bridge permite el acceso desde componentes COM a EJBs en el servidor de aplicaciones IBM WebSphere [9]. Ja.NET es un producto que provee un puente entre el mundo Java y .NET y es propietario de INTRINSYC [16][12].

En el contexto planteado de conexión J2EE-.NET, los Web Services presentan las siguientes ventajas: (i) Puede utilizarse en conexiones sobre HTTP, lo cual lo hace utilizable para conexiones a través de Internet; (ii) permite interacción tanto sincrónica como asíncrona; (iii) son multilinguaje; (iv) tienen un nivel de desarrollo y compromiso empresarial muy importante en múltiples plataformas. Esto se traduce, por ejemplo, en la disponibilidad de herramientas de desarrollo.

Como desventaja principal de los Web Services, se destaca que no incluye, al momento, control de transacciones y de seguridad.

Por su parte los JCA, presentan como ventaja principal que incluyen mecanismos de control de transacciones y seguridad, compatibles con los de J2EE.

Como desventajas de JCA se señalan: (i) desarrollo prácticamente nulo en plataforma Microsoft y especialmente .NET. (ii) Mecanismo fuertemente basado en Java.

En resumen, en la disyuntiva de elegir entre Web Services y JCA, se deben tener en consideración los siguientes factores: (i) *Uso exclusivo en red local vs. abierto a Internet*. Mientras que en el primer caso los JCA serían una opción aplicable, el segundo lleva a usar HTTP, y por lo tanto Web Services. (ii) *Sistemas centrados en Java y/o J2EE vs. multi-plataforma/lenguaje*. El primer caso sería abordable con JCA, mientras que el segundo lleva a usar Web Services como forma de hacer portable y transparente los mecanismos de interoperabilidad.

Los otros tipos de conectores presentan como ventajas el hecho de estar desarrollados específicamente para resolver la interacción J2EE con Microsoft, y como desventaja principal la falta de estandarización. En [27] se realiza una descripción detallada de estas opciones. En [31] se presentan comentarios sobre JCA y su posicionamiento frente a los Web Services.

A continuación se describen la experiencia realizada en base a Web Services, tanto para interoperabilidad sincrónica como asíncrona. La Figura 2 muestra el esquema general de esta interacción.

Figura 2: Interoperabilidad a través de Web Services



Interoperabilidad sincrónica sobre Web Services.

La opción de Servidor J2EE y Cliente .NET se implementó publicando un Web Service sobre el servidor J2EE. Para esto se exportó una clase de J2EE como un Web Service.

Para realizar la implementación se utilizó la herramienta JBuilder 7, que genera en forma automática los componentes necesarios. Entre esos componentes generados está el archivo WSDL, que permite importar las funcionalidades del Web Service desde un cliente, lo cual es fácilmente realizable desde la plataforma .NET.

Asimismo se agregó una Web Reference, partiendo del archivo WSDL especificado, y automáticamente se generó un Proxy, que son un conjunto de clases locales que encapsulan la invocación al Web Service. Este procedimiento funcionó aún cuando se tuviera un pasaje de datos complejos (en este caso un array de objetos). Cuando .NET detecta que un objeto retornado por un Web Service no es básico (String, Float, etc) genera clases locales a partir del archivo WSDL que representan a los objetos retornados.

La opción de Servidor .NET y Cliente J2EE se realizó mediante procedimiento análogo pero usando la herramienta Microsoft Visual Studio, y obteniéndose resultados similares.

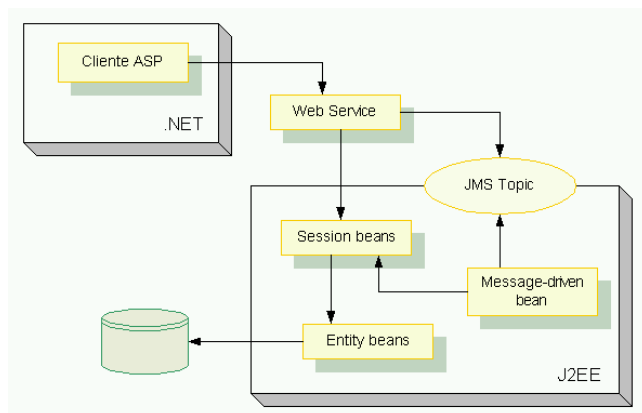
El Web Service generado en .NET se importó desde J2EE. La creación del Web Service y el hecho de consumirlo desde un EJB, incluso ante el pasaje de objetos complejos, es casi automático, así como la creación de objetos locales que encapsulan dicha complejidad.

Interoperabilidad asincrónica con Web Services.

La interoperabilidad asincrónica entre J2EE y .NET presenta algunas dificultades, debido a la diferencia de modelos en la comunicación asincrónica [33]. Mientras que J2EE modela el asincronismo en el servidor, Microsoft lo hace en el cliente. Esto hace que existan diferencias importantes en las funciones que implementan clientes y servidores en las diferentes plataformas. Este hecho es independiente del mecanismo utilizado, aunque en la práctica se experimentó la conexión con Web Services.

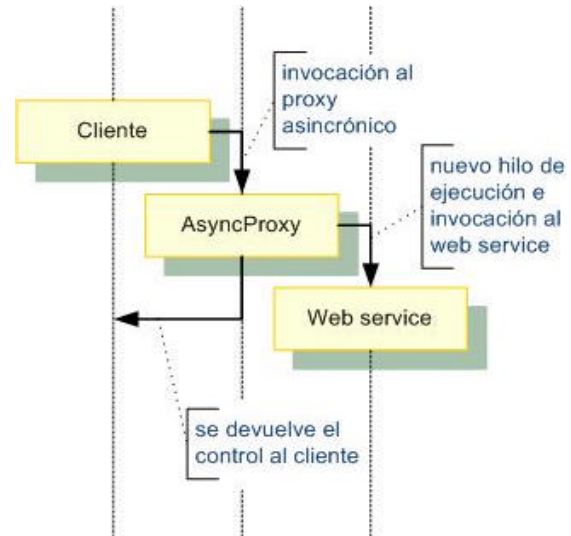
La opción de Servidor J2EE y Cliente .NET se implementó en forma directa, a través de la publicación de un Web Service en J2EE interactuando con JMS y Message Beans (Figura 3).

Figura 3: Arquitectura para Cliente.NET-Servidor J2EE



La opción de Servidor J2EE y Cliente .NET se implementó con el asincronismo en el cliente. Para esto se siguió las propuestas de [33] y [7], que permiten implementar un mecanismo cliente similar al de Microsoft. Se generó un proxy partiendo de un archivo WSDL, que encapsula las invocaciones a los métodos de un Web Service. Cada método es expuesto por este proxy en forma síncrona y asíncrona, y en el segundo caso el proxy abre un nuevo hilo de ejecución para esperar por el valor retornado por el Web Service, devolviendo el control en forma inmediata al cliente (Figura 4).

Figura 4: Invocación asincrónica a un Web Service con asincronismo en el Cliente.



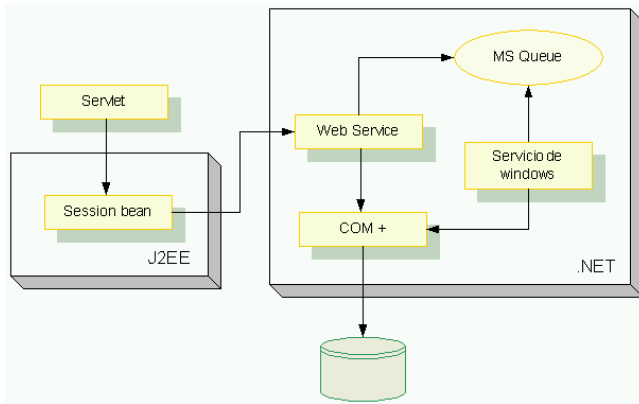
Arquitectura implementada.

La implementación de la opción Servidor J2EE y Cliente .NET sobre la cual se realizaron las pruebas descritas, se muestra en la Figura 3.

Resulta importante destacar que los Session Beans, Entity Beans, Message-Driven Beans y las clases auxiliares usadas en el caso de cliente y servidor homogéneos J2EE, no debieron ser modificadas para interoperar con .NET. Únicamente fue necesario implementar un Web Service que, al ser invocado desde un cliente ASP, permite el acceso a las funcionalidades de la aplicación.

Por su parte, la configuración Servidor .NET y Cliente J2EE, se implementó a través de un servlet que instancia a un Session Bean, y éste utiliza los métodos provistos por el servidor de Web Service, implementado en la plataforma Microsoft (Figura 5).

Figura 5: Arquitectura General para la combinación Cliente J2EE-Servidor .NET



Interoperabilidad entre J2EE y CORBA.

En el análisis de las opciones de interoperabilidad entre J2EE y CORBA, se partió de la hipótesis que las aplicaciones desarrolladas sobre plataforma CORBA debían ser multilenguaje (no solo Java).

Para resolver la problemática de interoperabilidad entre estas plataformas se identificaron dos variantes: (i) utilizar un gateway, (ii) utilizar RMI sobre IIOP.

Utilizando un "gateway" se hace uso de la facilidad que nos presenta CORBA al proveer interoperabilidad a través de diferentes lenguajes, entre ellos, Java. De esta forma, la utilización de un "gateway" desarrollado en Java (Figura 6), permitirá la comunicación entre cualquier componente J2EE y cualquier componente CORBA, eventualmente desarrollado en cualquier lenguaje.

Figura 6: Interoperabilidad J2EE-CORBA a través de un Gateway



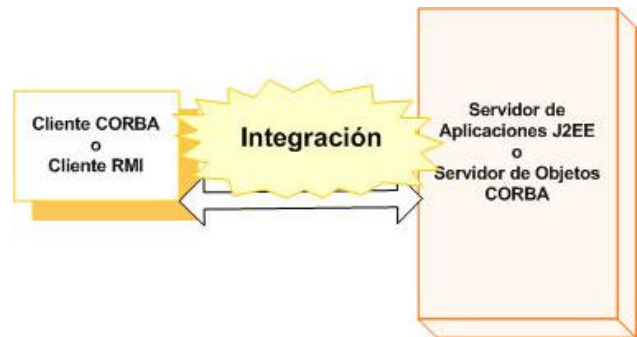
Este mecanismo es flexible en cuanto a la implementación, dado que el gateway puede implementar su propio método de llamado a los objetos del Servidor de Aplicaciones, introduciendo características como asincronismo.

Sin embargo, resulta en una solución ad-hoc para cada caso, lo cual requiere mayor trabajo de desarrollo específico a la aplicación, y por lo tanto mantenimiento posterior.

Interoperabilidad directa, basada en IIOP y RMI sobre IIOP. En este caso la interoperabilidad es implementada por

el propio servidor de aplicaciones J2EE, dando soporte a los mecanismos de comunicación de CORBA, como ser IIOP y RMI sobre IIOP [18]. La Figura 7 ilustra este caso.

Figura 7: Interoperabilidad directa entre J2EE y CORBA



Java y J2EE utilizan el mecanismo de RMI para resolver las invocaciones entre aplicaciones distribuidas. El protocolo de comunicación nativo de RMI es el JRMP (Java Remote Method Protocol). Por su parte, CORBA provee interoperabilidad a través del lenguaje y a través de los diferentes proveedores de ORB. Esta interoperabilidad es alcanzada con IIOP.

Por otro lado, Java RMI y CORBA resuelven de forma análoga las comunicaciones entre Cliente y Servidor, ya que ambos utilizan *Stubs* y *Skeletons* para sus comunicaciones a través de la red. Un *Stub* es un objeto proxy que reside del lado de los clientes que se encarga de delegar todas las invocaciones a métodos remotos, a su correspondiente *Skeleton* que reside del lado del servidor. El *Stub* es el objeto con el cual interactúan los clientes. Un *Skeleton* es un objeto proxy que reside en el servidor y se encarga de delegar los requerimientos que llegan a través de la red al objeto remoto requerido. Esos pedidos a través de la red han sido enviados por el *Stub* que reside del lado del cliente.

La diferencia entre Java RMI y CORBA radica en el protocolo utilizado para resolver las comunicaciones Cliente/Servidor. Es aquí donde surge *RMI sobre IIOP*, lo cual permite que Java RMI utilice IIOP como protocolo de comunicación. RMI sobre IIOP combina la facilidad de programación de RMI con la interoperabilidad de CORBA. IIOP es el protocolo para aplicaciones distribuidas escritas o bien en IDL o en Java RMI. Por lo tanto, IIOP permite interoperar entre aplicaciones escritas puramente en Java, que utilizan RMI, con aplicaciones escritas en cualquier lenguaje CORBA compatible.

Esta interoperabilidad directa basada en *RMI sobre IIOP* permite que, utilizando RMI/IIOP, un cliente CORBA invoque directamente a un componente J2EE como si fuera un objeto CORBA remoto, y viceversa (Figura 9).

Esta opción resulta eficiente y transparente en la implementación, ya que se omite la capa intermedia. El cliente CORBA realiza un llamado CORBA abstrayéndose del servidor. De la misma forma ocurre con un cliente RMI. Aunque resulta la opción más simple de implementar, esta tarea no es trivial ya que requiere manejo de código, hacia uno y otro lado, bastante complejo. Esto, según la escala y el diseño, puede hacer que el mantenimiento se compleje.

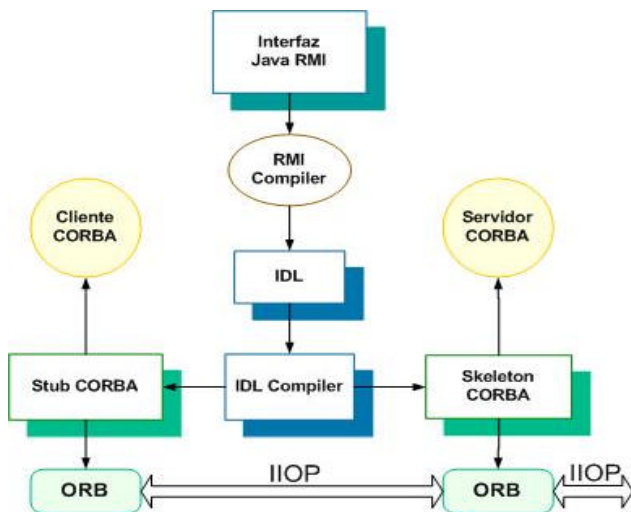
Con respecto al uso de JCA para interoperabilidad J2EE-CORBA, la interacción usando RMI/IIOP permitirá resolver adecuadamente los casos de invocación sincrónica, por lo que no se justifica el uso de conectores de tipo JCA. No ocurre lo mismo con los casos de invocación asíncrona, donde no existe una conexión comparable entre los mecanismos de manejo asíncrono de información de ambas plataformas. En este caso, dadas las limitaciones de JCA descritas antes, tampoco resulta de interés optar por esta tecnología.

Interoperabilidad sincrónica con RMI-IIOP.

Primeramente se debe aclarar que el Servidor de Aplicaciones J2EE utilizado (JBoss) da soporte a los mecanismos de comunicación con CORBA, tanto mediante IIOP como RMI sobre IIOP, lo cual permitió resolver la interoperabilidad sincrónica de forma directa. Para ello es necesario configurar JBoss de forma específica para permitir este tipo de comunicación.

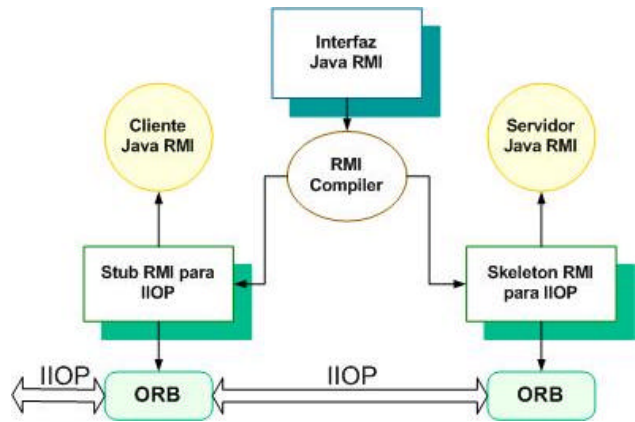
Para este propósito son necesarias algunas herramientas de la Plataforma J2SE, entre ellas el RMI Compiler. Esta herramienta puede generar varias salidas a partir de interfaces RMI, entre ellas: Interfaces IDL, y Stubs IIOP para dichas interfaces RMI. Al obtener interfaces IDL a partir de interfaces RMI se podrán generar los Stubs y Skeletons IIOP necesarios para CORBA (Figura 8).

Figura 8: Interoperabilidad usando RMI e IIOP



Por otro lado, al obtener los Stubs IIOP para las interfaces RMI se podrá acceder a los objetos a través de RMI con IIOP como protocolo de comunicación (Figura 9).

Figura 9: RMI sobre IIOP



Uniendo las Figuras 8 y 9 se puede observar, a grandes rasgos, el esquema a interoperabilidad basado en IIOP y RMI sobre IIOP.

Aplicando el procedimiento descrito, cualquier cliente CORBA podrá acceder a objetos Java utilizando IIOP. Más concretamente, un EJB podrá ser utilizado por cualquier cliente CORBA. En esta tarea JBoss genera los skeletons necesarios para que cualquier cliente, CORBA o RMI, pueda utilizarlos.

Para el caso inverso, en el que un cliente Java RMI accede a objetos CORBA, deberá realizarse toda la secuencia indicada en las Figuras 8 y 9, ya que estos casos escapan al alcance del Servidor de Aplicaciones J2EE. Sin dudas, esta es una tarea tediosa, ya que requiere desarrollar código tanto para el cliente como para el servidor. Por otro lado, más allá de tratarse de objetos CORBA, se deben escribir sus interfaces RMI, lo que implica un trabajo adicional, que puede ser más o menos complejo dependiendo de la aplicación. En esta situación se observa una clara carencia en las herramientas de desarrollo disponibles.

Interoperabilidad asíncrona con gateway.

Para resolver la interoperabilidad asíncrona se requirió la utilización de un Gateway como mecanismo de comunicación. La interoperabilidad entre un cliente CORBA y JMS no es posible de forma directa, en la medida de que un cliente RMI no puede comunicarse directamente con un Canal de Eventos CORBA (Notification Services).

Una implementación más directa (con el gateway en el cliente o servidor mismos) hubiera sido posible para clientes

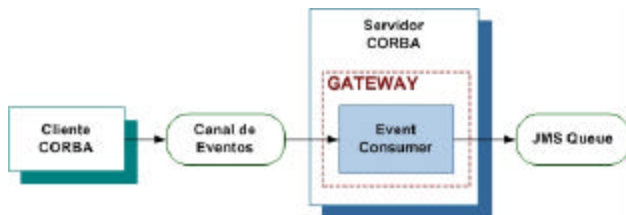
CORBA implementados en Java. Pero, dado que se apuntó a analizar las opciones de interoperabilidad independientemente del lenguaje utilizado en CORBA, la opción elegida fue la implementación de un *gateway* general. Claramente el *gateway* deberá estar desarrollado en Java, pero éste será sólo un intermediario.

Existen varias opciones para la implementación del *gateway* y es aquí donde se puede recalcar el carácter totalmente asincrónico de la solución. Si bien es posible implementar el *gateway* como un objeto Java común, invocado desde el cliente para comunicarse con JMS o un Canal de Eventos, dependiendo del caso, esta solución perdería el carácter asincrónico desde el lado del cliente. Por ello, en este trabajo, se utilizaron como *gateways* los propios mecanismos asincrónicos de cada plataforma. Es decir, un MDB para el caso de Cliente RMI-Servidor CORBA, y un Consumidor de Eventos CORBA para el caso de un Cliente CORBA-Servidor J2EE. De esta forma el cliente se libera inmediatamente después que envía el mensaje.

La implementación del *gateway* dependerá del sentido de la interoperabilidad.

El *gateway* CORBA → JMS se encargará de colocar en una Cola JMS todos los mensajes que lleguen al Canal de Eventos CORBA (Figura 10).

Figura 10: Gateway CORBA → JMS



En este caso, un cliente CORBA que desee utilizar un topic de JMS deberá comunicarse con el *gateway* para resolverlo. Este *gateway* Java será capaz de establecer la comunicación con JMS.

Para un cliente RMI el caso es análogo, el *gateway* CORBA será quien resuelva la comunicación con el Canal de Eventos CORBA. El cliente RMI deberá comunicarse con el *gateway* para resolver la comunicación con el Canal de Eventos CORBA (Figura 11).

Figura 11: Gateway JMS → CORBA

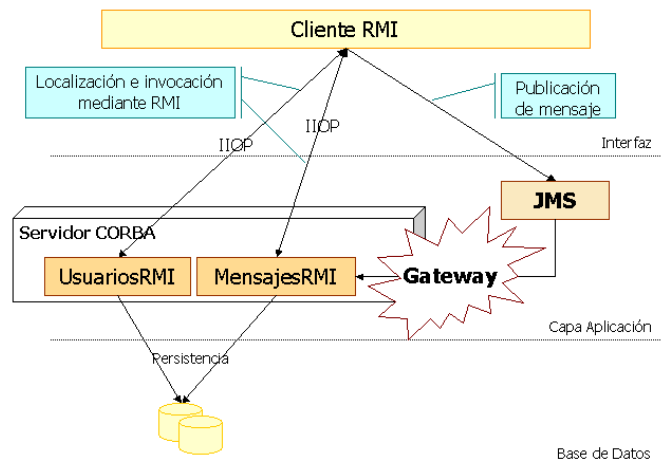


En ambos casos, el *gateway* deberá ocuparse de la conversión necesaria de un evento CORBA a un mensaje JMS y viceversa. Esta conversión no es muy compleja, pero dependerá del tipo de mensajes JMS que se estén enviando [21]. En la implementación realizada se utilizaron mensajes de texto con lo cual la conversión no representó una problemática para el trabajo.

Arquitectura implementada.

La arquitectura implementada para el caso Servidor CORBA y Cliente RMI se muestra en la Figura 12.

Figura 12: Arquitectura para Cliente RMI - Servidor CORBA



Los objetos *UsuariosRMI* y *MensajesRMI* son los objetos CORBA que implementan las distintas operaciones de la aplicación en el servidor CORBA.

El cliente es quien realiza las invocaciones a las operaciones de los objetos *UsuariosRMI* y *MensajesRMI*, de forma síncrona o asincrónica dependiendo de la operación. Estos clientes utilizan RMI como mecanismo de localización de los objetos. Para ello los objetos son publicados en el servicio de nombres CORBA integrado con el Servidor de Aplicaciones JBoss, y el JacORB Name Service.

La arquitectura implementada para el caso Servidor J2EE y Cliente CORBA se muestra en la Figura 13.

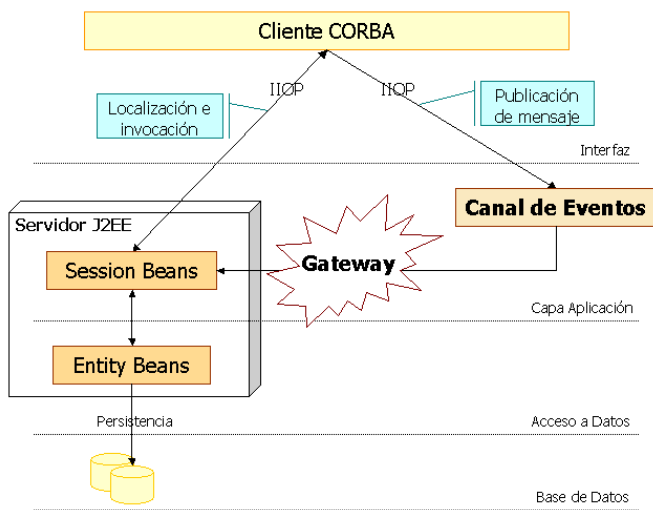
Los Entity Beans encapsulan el acceso y persistencia de los datos. Los Sessions Beans implementan las distintas operaciones de la aplicación en el servidor J2EE.

El cliente es quien realiza las invocaciones a las operaciones sobre los Sessions, de forma síncrona o asincrónica dependiendo del caso.

El Servidor de Aplicaciones J2EE, JBoss, ofrece el servicio de nombres CORBA de JacORB, JacORB Name Service, y el servicio JNDI para Java. Esto hará posible que un cliente

CORBA acceda a aquellos Entity y/o Sessions Beans que tengan interfaces remotas.

Figura 13: Arquitectura para Cliente CORBA - Servidor J2EE



LA PLATAFORMA DE ENSAYOS.

Los análisis implementados fueron realizados en el contexto de una plataforma de ensayos que asiste en la realización los mismos.

Esta plataforma fue implementada como un servicio, y está basada en las matrices descritas en las secciones anteriores, por lo que lo hemos llamado "*Matriz de Interoperabilidad*".

Este servicio permite realizar pruebas de interoperabilidad en forma sistemática basada en invocaciones de tipo cliente/servidor sobre la base de una aplicación de correo electrónico simplificado.

Los servidores de aplicaciones actualmente operativos en el servicio *Matriz de Interoperabilidad* son las siguientes:

CORBA-ORBacus: Basada en ORBacus (CORBA 2.4), el cual da soporte a varios de los servicios utilizados: ORBNotify (Notification Services), ORBacus IMR (Servicio de dirección de servidores). Como servidor de nombres para CORBA se utiliza JacORB Name Services ya que permite la interacción con el Servidor de Aplicaciones J2EE. Se utilizan las herramientas de Java 1.4.0 Standard Edition con JBoss/IIOP para integración con J2EE.

J2EE-JBoss: Basada sobre Jboss, con soporte para EJB 2.0. El protocolo de comunicación es JRMP. JMS es utilizado como servicio de mensajería, y JNDI es el servicio de nombres. Para resolver la interoperabilidad con CORBA JBoss

da soporte para IIOP y el servicio de nombres de JacORB. Para el servidor de Servlets se utilizó Tomcat 4.0, y como herramienta de desarrollo JBuilder 7.0 y 8.0.

Microsoft .NET: En Windows 2000 Server, utilizando .NET Framework 1.0. El servidor Web es Internet Information Server 5.0. El servicio de colas de mensajes es Microsoft Message Queue (ambos provistos por W2000). Se utiliza un servicio Windows para implementar la recepción de mensajes, con un componente para persistir los datos. La herramienta de desarrollo es Visual Studio .NET.

A los efectos de contar con un repositorio único de datos, se utilizó ODBC.NET y el driver ODBC provisto por PostgreSQL [26], para acceder a la base de datos que se encuentra en Linux (Redhat 8.0).

La plataforma descrita está desarrollada de forma de que los diferentes casos de interacción puedan ser ejecutados a través de una interfaz Web. La Figura 14 muestra la página inicial, en la cual se observa la matriz.

Figura 14: Interfaz a usuario de la plataforma de ensayos "Matriz de Interoperabilidad".



La *Matriz de Interoperabilidad* tiene en sus filas y columnas las distintas opciones de cliente y servidor, respectivamente, de las plataformas ensayadas. Cada celda de la matriz representa la interoperabilidad entre las plataformas involucradas, estando activas las marcadas con un ✓. Dentro de cada celda se han implementado diferentes operaciones sincrónicas y asíncronas.

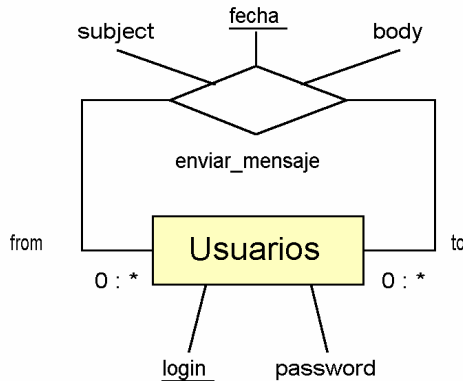
Como forma de probar un ejemplo más de interoperabilidad, la autenticación de usuarios al sistema se realiza invocando un Web Service desarrollado en .NET, desde un sitio fue implementado sobre la plataforma J2EE.

Para lograr flexibilidad frente a la aparición de nuevas plataformas u operaciones a implementar, este sitio se crea dinámicamente a partir de una Metadata. Asimismo se implementaron funcionalidades de administración que permiten agregar nuevos servidores de aplicaciones a la plataforma.

La invocación a operaciones multi-plataforma puede realizarse tanto on-line como batch a través de un archivo de comandos (script).

La *Matriz de Interoperabilidad* está basada en una aplicación que implementa el envío de mensajes entre usuarios. Un usuario, con atributos 'login' y 'password', envía y recibe cero o más mensajes. Los mensajes tendrán una 'fecha' de envío, un 'subject' y un 'body'. La Figura 15 muestra el modelo de datos de esta aplicación.

Figura 15: Aplicación base en la Matriz de Interoperabilidad



Brevemente, las funcionalidades de la aplicación implementada son:

- Alta de usuarios.
- Validar usuario/contraseña.
- Eliminar usuario.
- Enviar mensaje de un usuario a otro. Operación asíncrona con *write*, con colas de mensajes.
- Eliminar mensajes de un usuario. Operación sincrónica con *write*.
- Consultar mensajes de un usuario. Operación sincrónica de *read*.
- Invocación de "lotes de operaciones", y obtención de los resultados en un archivo de Log.

Estas funcionalidades permiten probar los diferentes mecanismos de interoperabilidad en un caso reducido que simplifica el agregado de nuevos servidores de aplicaciones.

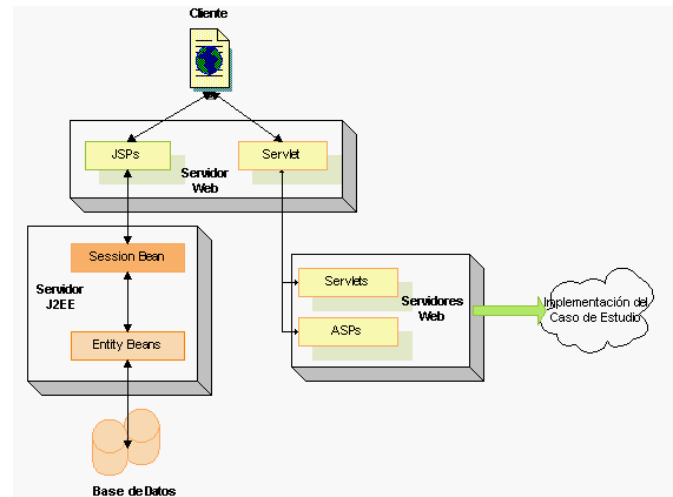
La arquitectura de implementación de plataforma de ensayos se muestra en la Figura 16.

Los clientes son browsers de HTML, que acceden o bien a los JSPs que despliegan las páginas de interfaz, o bien al Servlet que redirecciona los pedidos a los Servlets o ASPs, según las plataformas seleccionadas.

En el Servidor J2EE se implementa un Session Bean para acceso remoto a las funcionalidades de la Plataforma de

Ensayos. Los Entity Beans encapsulan el acceso a los datos y son accedidos únicamente por el Session Bean.

Figura 16: Arquitectura de la Matriz de Interoperabilidad



CONCLUSIONES Y TRABAJOS FUTUROS.

En este artículo se analizaron diferentes opciones de interoperabilidad entre servidores de aplicaciones CORBA, J2EE y .NET, respondiendo a los requerimientos cada vez más importantes de conectar plataformas heterogeneas a nivel de servidor de aplicaciones.

Los principales aportes del trabajo realizado consisten en el análisis con experimentación práctica de las diferentes opciones de interoperabilidad, así como la implementación de la plataforma de ensayos.

Como conclusiones generales se observa que, si bien aún restan una cantidad importante de problemas por resolver en la interoperabilidad entre servidores de aplicaciones, existen líneas de trabajo importantes y estables en las principales organizaciones y empresas tendientes a resolverlos, y con fuertes tendencias a lograr mecanismos abiertos.

Dentro de las carencias en los mecanismos de interoperabilidad se destacan las de gestión de transacciones y de mecanismos de seguridad homogéneos. Asimismo existe muy poca experiencia en cuanto a robustez y escalabilidad de estos mecanismos.

Con respecto a los aspectos técnicos, se observa que la **interoperabilidad sincrónica**, bi-direccional, entre J2EE y CORBA, así como entre J2EE y .NET, es resoluble y factible de implementar mediante mecanismos sistemáticos.

Un elemento a destacar es la simetría en la resolución de los mecanismos de interoperabilidad entre los diferentes pares de plataformas: *Dado un par de plataformas, la interoperabilidad tomando uno u otro como cliente o servidor, se resuelven de la misma forma.* Esta propiedad tiene un impacto importante en la claridad y mantenibilidad de las implementaciones.

La **interoperabilidad asíncrona** presenta dificultades en ambos casos J2EE-.NET y J2EE-CORBA.

En el caso J2EE→.NET se plantean problemas derivados de la existencia de modelos diferentes del concepto de "interacción asíncrona" en J2EE y Microsoft, el cual es resoluble utilizando complementos de software en el cliente J2EE.

Para resolver la interoperabilidad asíncrona entre J2EE y CORBA se requiere un gateway, que implemente la interacción entre Java JMS-MDB y un canal de eventos CORBA.

Perspectivas y Trabajo Futuro.

Tal como dijo anteriormente, hay una gran actividad en torno a la mejora de los mecanismos de interoperabilidad. Entre estos se destacan las extensiones propuestas para J2EE-JCA y para Web Services.

Las extensiones a la J2EE-JCA [20] apuntan a proveer: integración asíncrona y bi-direccional con EIS, mayor integración con JMS y soporte de Metadata y manejo de XML para invocación de funciones en el EIS (CCI, Common Cliente Interfase).

Las extensiones en curso a los Web Services corresponden principalmente mecanismos de transacciones. El concepto de transacción es fundamental para asegurar la consistencia de las operaciones en la plataforma distribuida. Por lo tanto, esta carencia limita drásticamente el uso de los Web Services como mecanismo de interoperabilidad.

La actual especificación de Web Services no incluye la noción de transacción, en el sentido de operación que cumple con las propiedades ACID (*Atomicity, Consistency, Isolation, Durability*).

Esta carencia, y la extensión del uso del mecanismo de Web Services, ha motivado el desarrollo extensiones a la especificación a la actual, siendo una de las principales la de WS-Transactions que está siendo impulsada por BEA, IBM, y Microsoft [4].

En esta iniciativa se diferencian los conceptos de *transacciones atómicas* y *transacciones del negocio*. Mientras las primeras corresponden a las tradicionales ACID, las segundas están más orientadas a compensar las acciones en caso

de anulación en lugar de realizar un *rollback* como las del primer tipo.

En lo relativo a trabajo en el Instituto de Computación, se está avanzando en el estudio de los mecanismos de transacciones sobre Web Services [8], y se espera incluir nuevos servidores de aplicaciones en la *Matriz de Interoperabilidad*.

REFERENCIAS.

- [1] Andoh A., Nash S.. RMI over IIOP. <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-iiop.html>. Java World, Diciembre 1999. Visitado Agosto 2002.
- [2] Apache Tomcat. <http://jakarta.apache.org/tomcat/index.html>. Visitado Octubre 2002.
- [3] Besil J., Pais C., Sande D.. *Interoperabilidad entre Servidores de Aplicaciones*. Proyecto de Grado de Ingeniería en Computación, InCo Fac. Ingeniería. Iniciado en 2002.
- [4] Cabrera F., Copeland G., Cox B., Freund T., Klein J., Storey T., Thatte S.. Specification: Web Services Transaction (WS-Transaction). August 2002. <http://www.ibm.com/developerworks/library/ws-transpec/>. Accedida en Junio 2003.
- [5] Congreso Iberoamericano de Telemática. Sitio Web de edición 2003 en <http://www.fing.edu.uy/cita2003/>.
- [6] CORBA. <http://www.omg.org/>. Visitado Julio 2002
- [7] Creating a simple web service with Axis. Tutorial de Jbuilder. Visitado en Enero 2003. http://info.borland.com/techpubs/jbuilder/jbuilder8/webvcs/publishbean/publish_bean_tutorial.html.
- [8] Ezquerro M., Pereira S., Revello S., Silva L.. *Transacciones distribuidas en la Web*. Proyecto de Grado de Ingeniería en Computación, InCo Fac. Ingeniería. Iniciado en 2003. <http://www.fing.edu.uy/~pgtdistr/>.
- [9] IBM WebSphere. <http://www-3.ibm.com/software/info1/websphere>.
- [10] *Intelligent EAI*. <http://www.intelligenteai.com/>
- [11] *International Conference on Cooperative Information Systems (CoopIS)*. Sitio Web de la edición 2003 en <http://www.cs.rmit.edu.au/fedconf/coopis/2003/>.
- [12] Intrinsic, EAI Products. http://www.intrinsic.com/products/enterprise_applications.asp.
- [13] J2EE Connector Architecture. SUN. <http://java.sun.com/j2ee/connector/>. Accedida en Junio 2003.

- [14] J2EE. Sun Web Site. <http://java.sun.com/j2ee>. Visitado Julio 2002.
- [15] J2SE. Sun Web Site. <http://java.sun.com/j2se>. Visitado Enero 2003.
- [16] Ja.NET - Intrinsic . <http://ja.net.intrinsic.com/>.
- [17] JacORB Web Site. <http://www.jacorb.org>. Visitado Febrero 2003.
- [18] Java RMI over IIOP. Sun Web Site. <http://java.sun.com/products/rmi-iiop/>. Visitado Agosto 2002.
- [19] JBoss Web Site. <http://www.jboss.org/index.html>. Visitado Junio 2002.
- [20] Jeyaraman R.. J2EETM Connector Architecture 1.5. Java Specification Requests 112. November 2002. <http://www.jcp.org/en/jsr/detail?id=112>. Visitado en Junio 2003.
- [21] JMS and CORBA Notification Service Interworking. <http://java.isavvix.com/whitepapers/1028730371009.pdf>. Visitado Agosto 2002.
- [22] Microsoft Developer Network. <http://msdn.microsoft.com>. Visitado Julio 2002.
- [23] Microsoft.NET. Microsoft Web Site. <http://www.microsoft.com/net/>. Visitado Julio 2002.
- [24] Middleware Resource Center. <http://www.middleware.org/>.
- [25] ORBacus For C++ and Java 4.1.0. Orbacus Web Site. http://www.iona.com/products/orbacus_home.htm. Visitado Agosto 2002.
- [26] PostgreSQL. <http://www.postgresql.org/>. Visitado Julio 2002.
- [27] Rodríguez L., Vignaga A., Zipitría F.. Estudio de Interoperabilidad entre .NET/J2EE. Reporte Técnico, InCo-Pedeciba Informática. 2002. <http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0208.pdf>.
- [28] Roman E. *Mastering Enterprise JavaBeans*. Second Edition.. ISBN: 0-471-41711-4.
- [29] Rosen M., Curtis D.. *Integrating CORBA and COM Applications*. Wiley Computer Publishing. 1998.
- [30] Simple SOAP. Microsoft Web Site. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xml10152001.asp>.
- [31] Taft D. K.. JCP Gets Java Services-Ready. EWeek. January 7, 2002. Visitado en Junio 2003. http://www.eweek.com/print_article/0,3668,a=20685,00.asp.
- [32] The Server Side Web Site. <http://www.theserverside.com/home/index.jsp>. Visitado Junio 2002.
- [33] Verma M.. Platform Interoperability: Sun ONE and Microsoft .NET, Achieving Asynchronous Communications. Sun Web Site. http://sunonedev.sun.com/building/tech_articles/asyn_c_paper.html. Visitado Febrero 2003.
- [34] Web Services. W3C Web Site. <http://www.w3.org/2002/ws/>. Visitado Agosto 2002.