# Estudio de Interoperabilidad .NET/J2EE

**Leonardo Rodríguez, Andrés Vignaga, Felipe Zipitría**
Universidad de la República, Facultad de Ingeniería, Instituto de Computación
Montevideo, Uruguay
{lrodrigu,avignaga,fzipi}@fing.edu.uy

**Resumen**

El presente trabajo detalla un estudio realizado de cuatro posibles mecanismos para integrar aplicaciones desarrolladas en las plataformas J2EE y .NET. El objetivo de ese estudio fue generar el conocimiento necesario para solucionar el problema presentado por la integración de partes desarrolladas en .NET y J2EE de una misma aplicación. Al no existir un mecanismo predefinido para ello, se estudiaron varias alternativas disponibles en el mercado para realizar esta integración. Asimismo, se analizaron las ventajas y desventajas de cada una de las alternativas, de forma de contar con las pautas necesarias para optar por la mejor solución al problema puntual que originó este estudio.

**Palabras clave:** Interoperabilidad, Componentes, Transacciones, .NET, J2EE, Web Services, JCA

## 1. Introducción

La dinámica actual de las empresas origina su expansión, fusiones con otras empresas y aperturas a nuevos mercados por distintos medios, plantea exigencias muy altas para las aplicaciones empresariales que existen detrás para soportar tal estructura.

Algunas de las exigencias planteadas son:

- Escalabilidad y portabilidad de la aplicación (este requerimiento exige un mecanismo fácil y confiable para migrar plataformas de software y hardware a plataformas más performantes)
- Facilidad de expansión y mantenimiento.
- Facilidad de integración con aplicaciones y fuentes de datos existentes.
- La seguridad y confiabilidad son fundamentales.

Por estas exigencias, las diferentes ofertas de plataformas de desarrollo que existen en el mercado y la gran diversidad de funcionalidades que cada una de estos productos ofrecen, lleva a que cada vez sea más necesario combinar distintas plataformas para obtener una aplicación que se adapte lo mejor posible a las necesidades.

El caso particular que se desarrollará en el presente artículo es el de la integración de las plataformas J2EE (*Java 2 Enterprise Edition*) de Sun y .NET de Microsoft. Dado que son las plataformas más utilizadas actualmente en el mercado, se han realizado muchos esfuerzos para hacer esta integración posible. El artículo estudia cuatro alternativas para realizar la integración y realiza un análisis de las mismas de forma de tenerlo como referencia para decidir que alternativa utilizar frente a un problema dado.

En la sección 2 se estudiará el uso de Web Services como una tecnología emergente para la interoperabilidad de aplicaciones y en particular para integrar aplicaciones .NET con J2EE. En la sección 3 se estudiará una propuesta de Sun de una arquitectura para el acceso transparente a sistemas de información llamada JCA (*Java Connector Architecture*), en la sección 4 se estudia Ja.Net como una propuesta propietaria para conectar el mundo Java con el mundo .NET y en la sección 5 se estudio ActiveX Bridge como una propuesta de la familia de productos IBM WebSphere para la integración de Java con el mundo COM+ (*Component Object Model*) (modelo de componentes también utilizado por .NET). Por último en las secciones 6 y 7 se realiza un comparativo de las soluciones estudiadas y se concluye.

# 2. Interoperabilidad basada en Web Services

## 2.1 Descripción General

Los Web Services [10] son una tecnología emergente para permitir exponer servicios a través de la Web. Permite que aplicaciones Web interactúen dinámicamente con otras aplicaciones Web, utilizando para ello estándares abiertos como XML (*Extensible Markup Language*), UDDI (*Universal Description, Discovery, and Integration*) y SOAP (*Simple Object Access Protocol*). Las funciones que pueden ser realizadas por los web services pueden ir desde simples intercambios de información hasta complicados procesos de negocios. Las empresas vendedoras de servicios pueden encapsular su lógica y procesos de negocio mediante web services y exponerlas para que sus clientes las consuman a través de la Web.

Aparte de un avance tecnológico los web services plantean un cambio en los modelos de negocios de las empresas vendedoras de servicios, poniendo en jaque a los mecanismos actuales de venta de software y servicios computacionales. Exponiendo los servicios en la Web se tiene acceso a un espectro mucho mayor de clientes potenciales y facilita enormemente el acceso a dichos servicios.

Las ideas detrás de los Web Services no son nuevas, ya estaban presentes en tecnologías como RPC (*Remote Procedure Call*). Los Web Services permiten realizar invocaciones a procedimientos remotos, tanto en redes de porte pequeño como puede ser una Intranet empresarial como en redes de gran porte como Internet. Esto es posible porque están basados en un protocolo simple y liviano para realizar las invocaciones. El protocolo que permite realizar las invocaciones es SOAP. SOAP es un protocolo estándar creado por la W3C, basado en XML.

SOAP está compuesto por cuatro componentes fundamentales: un envoltorio que define un framework para describir los mensajes y cómo estos deben ser procesados, un conjunto de reglas para codificar (o serializar) instancias de tipos de datos definidos

por las aplicaciones, una convención de cómo representar invocaciones remotas y sus respuestas, y una convención para vincular el intercambio de mensajes con un protocolo de transporte. El protocolo de transporte que normalmente se utiliza con SOAP y el utilizado por los Web Services es http (*HyperText Transfer Protocol*).

Utilizar HTTP como protocolo para las invocaciones facilita el uso de la infraestructura Web ya existente en la actualidad en prácticamente toda empresa, para el intercambio de información o publicación de servicios de la empresa. Asimismo, este intercambio es posible realizarlo sin tener que agregar más protocolos a los ya existentes en los Firewalls corporativos para permitir estos flujos de información, lo cual sería necesario si se utilizara algún otro protocolo.

Los Web Services son mucho más que simplemente SOAP, este sólo define el protocolo de invocación remota de los métodos, pero aparte de éste también incorporan al WSDL (*Web Service Description Language*) como un lenguaje también basado en XML que permite describir los contratos de cada servicio. Este lenguaje fue desarrollado conjuntamente por Microsoft e IBM y sometido para su aprobación como un estándar.

También se incluye un protocolo para el descubrimiento de Web Services (Disco) el cual define el formato de un documento XML con la información de descubrimiento y un protocolo para que un usuario pueda recibir este documento, permitiendo descubrir los servicios de una URL conocida. Para los casos en que el usuario no conoce la URL, también incluye un mecanismos UDDI estándares para que los proveedores de servicios publiquen sus servicios y los consumidores los encuentren.

Al estar basados en estándares abiertos, no están restringidos a ninguna plataforma o lenguaje en particular, y por ello ofrecen un framework ideal para la integración de aplicaciones.

Existen otras tecnologías que permiten la integración de aplicaciones o componentes distribuidos; CORBA (*Common Object Request Broker Architecture*) ó DCOM (*Distributed Component Object Model*) son un ejemplo de ellas. Estas tecnologías normalmente son muy complejas (difíciles de implementar) y requieren de una gran cantidad de recursos para funcionar correctamente. Esto plantea requerimientos importantes de hardware tanto para el servidor como para el cliente y también sobre la infraestructura de la red de soporte. Por las razones mencionadas anteriormente, estas tecnologías son solo viables en redes privadas pequeñas y no es posible utilizarlas con componentes de bajos recursos como por ejemplo PDAs (*Personal Digital Assistant*) o teléfonos celulares. La ventaja que plantean los Web Services es que el protocolo que utilizan para realizar las invocaciones es un protocolo liviano y simple, lo que lo hace propicio para correr en redes de gran porte como Internet y pueden ser utilizados desde cualquier dispositivo.

Actualmente las plataformas .NET y J2EE son las principales plataformas que incluyen soporte de Web Services en forma nativa.

## 2.2 Transacciones

La aparición de los Web Services promete tener un gran impacto en la forma en que los grandes sistemas empresariales se comunican y colaboran. Los procesos de negocio ya no estarán restringidos a una sola empresa, estos pasarán a ser grandes procesos que implicarán la colaboración de varias empresas. La tecnología que permita realizar esta integración debe disponer de mecanismos de seguridad y transaccionalidad apropiados, para asegurar la confiabilidad y confidencialidad de la información por ellos manejada.

Actualmente la propuesta de Web Services no incluye soporte para transacciones. Numerosos esfuerzos se están llevando a cabo en esa dirección para lograr aumentar la confiabilidad de esta tecnología.

IBM está llevando adelante un proyecto llamado "Web Services Transactions Project", [11][12][13] el cual intenta brindar un solución al problema. Apuntan a resolverlo utilizando un framework de coordinación de propósito general como es el J2EE Activity Service [14], dada la gran variedad de modelos transaccionales que existen en la actualidad. La idea es que este framework sea lo suficientemente flexible como para permitir coordinar a todos los participantes con diferentes modelos transaccionales, utilice protocolos estándares como SOAP y no comprometa la autonomía de ninguno de los participantes.

Otra propuesta actualmente disponible en el mercado es la de Versata Inc., la cual dispone de una extensión [15] para servidores de aplicaciones J2EE llamada "Versata Logic Server" para el manejo de lógica de negocio. Soporta servidores de aplicaciones como IBM WebSphere o BEA WebLogic. Esta extensión también permite el manejo de transacciones en forma declarativa sobre los Web Services.

Asimismo existe una propuesta de un modelo transaccional basado en SOAP llamada SOAP-CTX [16], con el objetivo de ejecutar transacciones ACID (*atomicity, consistency, isolation, and durability*) sobre conexiones no confiables. Este fue especialmente diseñado pensando en protocolos como HTTP o SMTP (*Simple Mail Transfer Protocol*).

## 2.3 Integración J2EE y .NET

Claramente los Web Services presentan una propuesta muy interesante para la integración de aplicaciones en diferentes plataformas y diferentes lenguajes. Dado que los Web Services aún son una tecnología emergente, la cual no está completamente estable, los ideales de interoperabilidad entre plataformas no se cumplen en un 100% por lo cual es necesario complementar con determinadas herramientas para poder resolver estos problemas de interoperabilidad. Para la integración de plataformas como .NET y J2EE, existe una gran variedad de herramientas basadas en Web Services que permiten llevarla a cabo.

Algunas de estas herramientas son:

- **IBM Web Service Toolkit (WSTK)**: Provee de un runtime environment (tanto del lado del cliente como del servidor), demos, ejemplos y algunas herramientas adicionales para diseñar y ejecutar aplicaciones basadas en Web

Services [18]. El objetivo del producto es facilitar el desarrollo de aplicaciones basadas en Web Services. Trabaja como un plug-in para servidores de aplicaciones J2EE y actualmente en la versión 3.0. Los servidores soportados son: IBM WebSphere y Web Sphere MicroEdition, y Jakarta Tomcat 4.0. Estos servidores ya tienen soporte para Web Services, pero el WSTK le agrega algunas funcionalidades nuevas y aplicaciones ejemplo. Aparte de incluir todos los estándares como SOAP, WSDL, UDDI; el WSTK provee un conjunto de utilidades extras que ofrecen entre otras cosas servicios de identificación, notificación, medición de uso de Web Services y contratación de Web Services y servicios de acceso a datos similares a JDBC basado en Web Services [19].

- **GLUE**: Provee una plataforma tanto para desarrollar como para consumir Web Services para la plataforma Java. El mismo soporta y es completamente compatible con los estándares XML, SOAP, WSDL y UDDI. Incorpora una API muy completa que disminuye la complejidad del desarrollo y publicación de Web Services. Este producto es desarrollado por la empresa *The Mind Electric* [17]. Se destaca por su facilidad de uso (lo cual fue confirmado al implementar un prototipo para consumir Web Services de .NET desde Java), su confiabilidad y robustez.
- **Java XML Pack**: Es un paquete provisto por Sun que incorpora un conjunto de APIs para XML y APIs estándar para manejo de Web Services. Las APIs provistas soportan SOAP, XMLP (XML protocol) el cual es una especificación provista por W3C para la interacción entre objetos remotos basada en XML (está aún en construcción y se espera que finalmente sea basada en SOAP) y también soporte para WSDL y UDDI. El paquete está compuesto por cuatro APIs, JAXM para mensajería en general, basada en SOAP, JAXP para procesamiento de datos en formato XML, JAXR para registro de servicios (soporta UDDI) y JAX-RPC que permite realizar invocaciones a Web Services (basada en SOAP y con soporte WSDL).

## 2.4 Prototipos

Para probar la interoperabilidad entre J2EE y .NET, se intentó realizar pruebas con el IBM Web Services Toolkit y con GLUE. Inicialmente las pruebas apuntaron a probar la interoperabilidad de un aplicativo Java normal contra un aplicativo desarrollado en .NET. El éxito de estas pruebas permite luego, desde un EJB (*Enterprise Java Beans*), realizar las invocaciones necesarias mediante Web Services a la aplicación .NET.

Para el caso de IBM WSTK hubo problemas con su instalación, los cuales impidieron que se llegara a completar las pruebas. Estas pruebas se hicieron instalándolo sobre el servidor de aplicaciones IBM Web Sphere 4.0 sobre Windows 2000 Professional Edition.

En cambio para el caso de GLUE, las pruebas fueron realizadas con éxito y sin enfrentar mayores problemas. En las pruebas realizadas se utilizó GLUE 2.0 y fueron realizadas sobre Windows 2000 Professional Edition.

Inicialmente se creo un Web Service utilizando Visual Studio .NET, que por un lado dispone de un método que retorna el string tradicional “Hello World !!” y otro Web Service que dado dos números realiza su suma y retorna el resultado. Esos servicios se publicaron y quedaron prontos para ser utilizados.

Luego desde Java y utilizando GLUE se creo una aplicación que permite invocar estos Web Services y mostrar los resultados en pantalla. GLUE no requiere la creación de Stubs previo a la invocación de los métodos, estos son creados dinámicamente por él al momento de realizar las invocaciones. Esto permite disminuir y simplificar el trabajo necesario por los desarrolladores. Lo único que se debe crear previamente es una interfaz que corresponda a los métodos de los Web Services que se desea invocar. Esta interfaz es pasada como parámetro a GLUE, para que éste pueda devolver un proxy que la implemente. El resultado es que prácticamente en forma transparente, desde Java se puede obtener una referencia a un objeto que representa al objeto modelado por los Web Services del lado del servidor. Dicha interfaz puede ser generada en forma automática utilizando la herramienta "wsdl2java" partiendo del documento WSDL que describe al Web Service.

En la Figura 1 se puede ver la implementación del ambos Web Services en C#. Para marcar a un método como accesible desde la Web lo único que se necesita es poner el atributo "[WebMethod]" que así lo diga. En la Figura 2 se puede ver el código necesario para realizar la invocación a los Web Services usando GLUE. El mecanismo es muy simple, basta con realizar el binding a la URL en donde está localizado el documento WSDL que describe los servicios disponibles y así ya obtener una referencia a un proxy generado dinámicamente. Luego basta con invocar a los métodos del proxy devuelto, para realizar las invocaciones a los Web Services correspondientes.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace WSHelloWorld
{
  public class Service1 : System.Web.Services.WebService
  {
          public Service1()
          {
                  InitializeComponent();
          }
          #region Component Designer generated code
          private void InitializeComponent(){}
          #endregion

          protected override void Dispose( bool disposing ){}

          // http://localhost/WSHelloWorld/WSHelloWorld.asmx?WSDL
          [WebMethod]
          public string HelloWorld()
          {
                  return "Hello World";
          }

          // http://localhost/WSHelloWorld/WSHelloWorld.asmx?WSDL
          [WebMethod]
          public int add(int a, int b)
          {
                  return a+b;
          }
  }
}
```

**Figura 1:** Web Services implementados con .NET

```
// Prototipo:   Consumo de Web Services publicados por la plataforma
//              .NET desde Java utilizando GLUE.

import electric.registry.Registry;

public class TestIt{
  public static void main( String[] args ) throws Exception
  {
      // Realizando el bind con el correspondiente archivo WSDL.
      // El archivo WSDL se encuentra publicado en la dirección
      // abajo mencionada
      String url = "http://localhost/WSHelloWorld/WSHelloWorld.asmx?WSDL";
      IWSHelloWorld Hello = (IWSHelloWorld)Registry.bind(url,IWSHelloWorld.class);

      // Invocación al Web Service como si fuera un objeto Java local
      String str = Hello.HelloWorld();
      System.out.println("Prueba del Test Hello World");
      System.out.println("");
      System.out.println("Resultado: " + str);

      // Invocación al Web Service como si fuera un objeto Java local
      int res = Hello.add(10, 10);
      System.out.println("Prueba de la suma, con 10 + 10");
      System.out.println("");
      System.out.println("Resultado: " + res);
  }
}
```

**Figura 2**: Invocación de Web Services mediante GLUE

# 3. J2EE Connector Architecture

## 3.1 Descripción General

J2EE Connector Architecture (JCA) [1][2][3][4] es un estándar desarrollado por el Java Community Process apostando a solucionar el problema de integración entre servidores de aplicaciones y EIS (*Enterprise Information Servers*) existentes. Antes de la existencia de la JCA los vendedores de servidores de aplicaciones debían extender sus productos mediante mecanismos propietarios para permitir la interoperabilidad con determinados tipos de EIS. Los EJB desarrollados utilizando estos mecanismos propietarios perdían automáticamente la portabilidad hacia el resto de los servidores de aplicaciones J2EE disponibles en el mercado, y con esto las características fundamentales a la cual apunta la tecnología Java.

Entonces, con la introducción de JCA los EJBs desarrollados pueden acceder potencialmente a cualquier EIS mediante mecanismos homogéneos y bien definidos por la especificación JCA [1], logrando así no comprometer la portabilidad de los EJB desarrollados.

Como ejemplo de EISs podemos considerar a ERP (*Enterprise Resource Planning*), procesamiento de transacciones en mainframes, bases de datos, aplicaciones legadas implementadas en lenguajes distintos a Java y otras plataformas para desarrollo de aplicaciones empresariales (como puede ser .NET). Actualmente se encuentra disponible la especificación 1.0 de JCA, la cual se incorporó a la plataforma J2EE a partir de la versión 1.3.

La idea fundamental detrás de JCA es que los vendedores de servidores de aplicaciones extiendan sus productos para soportar la especificación JCA y los vendedores de EIS proveer de los llamados resource adapters (una suerte de drivers, se

entrará en mayor detalle en las siguientes secciones) para cada uno de sus EIS. Estos resource adapter se registran en el servidor de aplicaciones, quedando disponibles para ser utilizados por los componentes que residen en el mismo. Para acceder a los EIS, la especificación define un mecanismo homogéneo mediante un API (llamada Common Client Interface o CCI).

La especificación establece una serie de contratos que debe cumplir tanto los servidores de aplicaciones que implementen la JCA, como los resource adapters de cada EIS para definir mecanismos de escalabilidad, seguridad y permitir manejos transaccionales entre los mismos.

## 3.2 La Arquitectura

Para eliminar las heterogeneidades existentes entre los distintos EIS, la especificación define una serie de contratos a nivel del sistema que deben cumplir tanto el servidor de aplicaciones como los EIS. Al decir a nivel del sistema significa que van a ser coordinados entre ambos sistemas (servidor de aplicaciones y EIS) siendo completamente transparentes para las aplicaciones que corren en el servidor de aplicaciones.

Se establecen tres tipos de contratos:

- Manejo de conexiones: Con el objetivo de permitir el manejo homogéneo de las conexiones, proveer una interfaz común para el manejo de las conexiones y de un pool de conexiones a los efectos de permitir escalabilidad en las soluciones. También permitir un mecanismo genérico para que el servidor de aplicaciones pueda proveer distintas QoS (Quality of Service).
- Seguridad: Extiende los contratos de manejo de conexiones con detalles específicos a la seguridad, como puede ser permitir pasar las credenciales desde el servidor de aplicaciones al resource adapter.
- Manejo de transacciones: Permitir establecer un ambiente transaccional al trabajar con diferentes EIS de una forma completamente transparente.

La arquitectura de la JCA está basada en la existencia de los llamados resource adapters específicos de cada EIS, los cuales son componentes de software que implementan los contratos por el lado de los EIS, sirven de intermediarios entre el servidor de aplicaciones y los EIS (ver Figura 3). El servidor de aplicaciones por su lado, debe extenderse para soportar la JCA, implementando los contratos.

Múltiples resource adapters pueden registrarse ante un servidor a aplicaciones y también es posible que un resource adapter se registre en varios servidores de aplicaciones.

Para la comunicación entre los componentes y los resources adapters la especificación permite dos mecanismos:

- A través de la CCI (Common Client Interface). Interfaz especialmente diseñada para permitir un acceso uniforme a todos los EIS.
- A través de APIs específicas de cada resource adapter.

Se recomienda el uso de la primer opción, por tratarse de un mecanismo estádar.

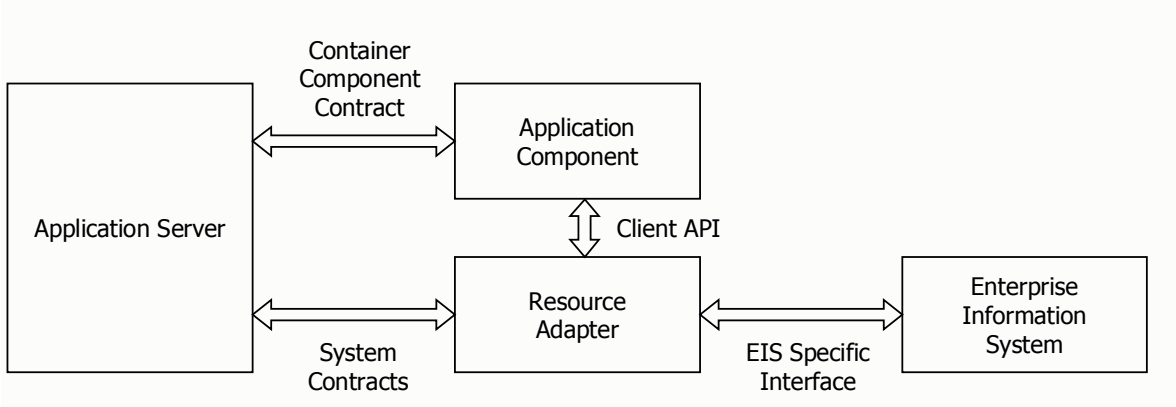


**Figura 3**: Arquitectura de la JCA

## 3.3 J2EE Connector Architecture y .NET

La J2EE Connector Architecture ofrece una solución interesante al problema de interoperabilidad existentes entre EJB de la plataforma J2EE y componentes COM de la plataforma Microsoft [3].

En la actualidad en el framework provisto por EJB y por COM las transacciones distribuidas son manejadas mediante la implementación del DTP (*Distributed Transaction Processing*) definido por los estándares X/Open. En este estándar existe el llamado TM (*Transaction Manager*) que es el encargado de crear y coordinar a todos los recursos participantes de las transacciones distribuidas. Cada uno de los repositorios de información tiene un representante llamado RM (*Resource Manager*) que dialoga con el TM (utilizando el protocolo XA) de forma de coordinar la transacción para ese repositorio en particular. Llegado el momento, el TM realizará el commit o el rollback de todos los participantes de la transacción simultáneamente.

Los servidores de aplicaciones J2EE ofrecen el servicio DTP a través del JTS (*Java Transaction Service*), de forma análoga del lado de Microsoft existe el DTC (*Distributed Transaction Coordinator*). A pesar de utilizar ambos el DTP, no es posible que el JTS se comunique directamente con un RM de COM, lo cual imposibilita tener transacciones que abarquen componentes EJB y COM simultáneamente.

Como fue presentado anteriormente, JCA fue diseñado especialmente para resolver problemas como éste, por lo cual utilizando JCA sería posible implementar un resource adapter específico para COM que junto a un servidor J2EE con soporte para JCA permita realizar transacciones distribuidas combinando componentes EJB y COM.

Actualmente JCA ha tenido una amplia aceptación en el mercado y existen varios servidores de aplicaciones J2EE con soporte para JCA. También existen gran cantidad de empresas abocadas al desarrollo de JCA Connectors o resource adapters. Algunas empresas como INTRINSYC [7] y Javector Software [3] han desarrollado resource adapters específicos para COM, permitiendo el manejo de transacciones, seguridad y pool de conexiones para COM.

Mediante el uso de los JCA es posible acceder a componentes COM desde cualquier plataforma, y de cualquier servidor de aplicaciones que soporte la especificación JCA. El connector desarrollado por INTRINSYC, no requiere la instalación de ningún software en especial del lado del servidor Microsoft, éste dialoga directamente utilizando el protocolo DCOM. En la Figura 4 se puede ver la arquitectura de la solución de INTRINSYC.

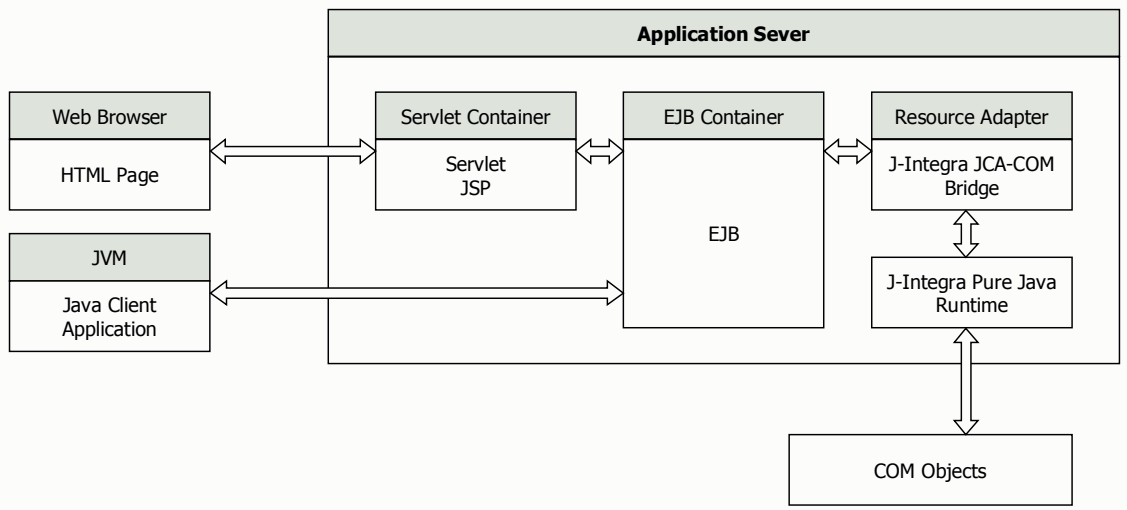


**Figura 4**: JCA Connector de INTRINSYC

Dispone de la biblioteca "J-Integra Pure Java Runtime" que permite tanto el acceso a COM a través del resource adapter, como de programas Java normales.

## 3.4 Servidores de Aplicaciones con soporte JCA

En la actualidad existen varios servidores de aplicaciones J2EE con soporte para J2EE Connector Architecture. En la Figura 5 se puede ver una lista de ellos. También existe una considerable cantidad de resource adapters disponibles en el mercado. Por más información ver [2].

| Vendedor / Producto | Versión JCA |
|---|---|
| BEA WebLogic Preview 7.0 | 1.0 |
| Borland Enterprise Server 5.0 | 1.0 |
| IBM WebSphere Technology for Developers | 1.0 |
| Macromedia JRun Server Technology Preview | 1.0 |
| Pramati Server 3.0 | 1.0 |
| SilverStream eXtend App Server 4.0 Beta | 1.0 |
| Sybase EAServer 4.1 | 1.0 |
| Trifork Application Server 3.0 | 1.0 |

**Figura 5**: Servidores de aplicaciones con soporte JCA

IBM WebSphere incluye soporte para JCA desde la versión 4.0, pero en modalidad BETA. A partir de la versión 4.1 de WebSphere el soporte de JCA ya es una de las funcionalidades finales incluidas por el producto.

# 4. Interoperabilidad basada en Ja.NET

## 4.1 Descripción General

Ja.NET es un producto ofrecido por la empresa Intrinsyc Software Inc. que provee un bridge entre el mundo Java y el mundo Microsoft .NET [5][6]. Permite el acceso a EJBs desde cualquier lenguaje provisto por .NET y también permite el acceso a componentes .NET desde cualquier aplicación Java, inclusive EJBs.

Esto lo logra haciendo uso de .NET remoting, el nuevo protocolo para objetos distribuidos ofrecido por Microsoft. .NET remoting fue diseñado para ser utilizado en ambientes Intranets donde el hay un alto acoplamiento entre los distintos componentes o también sobre ambientes Internets en donde existe un bajo acoplamiento entre los componentes.

.NET remoting es usado dentro de la plataforma .NET para permitir la comunicación de managed componentes (o componentes CLR) que estén en diferentes dominios de aplicación. Al hacer uso de .NET remoting Ja.NET permite que los componentes Java aparezcan como componentes CLR y que los componentes CLR aparezcan como componentes Java.

La versión actual de Ja.Net (1.0) no soporta transacciones distribuidas ni niveles de seguridad, pero promete que para futuras versiones estas funcionalidades van estar integradas.

Por las carencias que tiene esta propuesta a primera vista no tiene mayores ventajas sobre lo que se podría realizar utilizando Web Services. Pero en realidad si las tiene:

- Soporta múltiples protocolos de transporte. Por ejemplo permite transferencias binarias de alta velocidad sobre TCP.
- Permite activación y control de tiempo de vida de los objetos por parte del cliente.
- Soporta eventos y callback.
- Permite mapeo entre la jerarquía de clases y tipos .

## 4.2 Servidores de Aplicaciones con soporte para Ja.NET

Ja.Net incluye soporte para los servidores de aplicaciones BEA WebLogic, IBM WebSphere, Oracle 9i, iPlanet, Borland Enterprise server y JBoss.

# 5. Interoperabilidad basada en ActiveX Bridge

ActiveX bridge [8] es una tecnología agregada al servidor de aplicaciones IBM WebSphere a partir de la versión 4.0 [9] que permite que aplicaciones COM accedan a componentes EJB y otros servicios disponibles de la plataforma J2EE que se encuentren disponibles en el servidor WebSphere.

Esto se logra teniendo una JVM corriendo dentro del proceso ActiveX la cual es la que efectivamente se comunica con el servidor, y el componente COM a través del ActiveX bridge realiza las invocaciones correspondientes a esta JVM [8]. En la Figura 6 se ve reflejada la arquitectura propuesta.

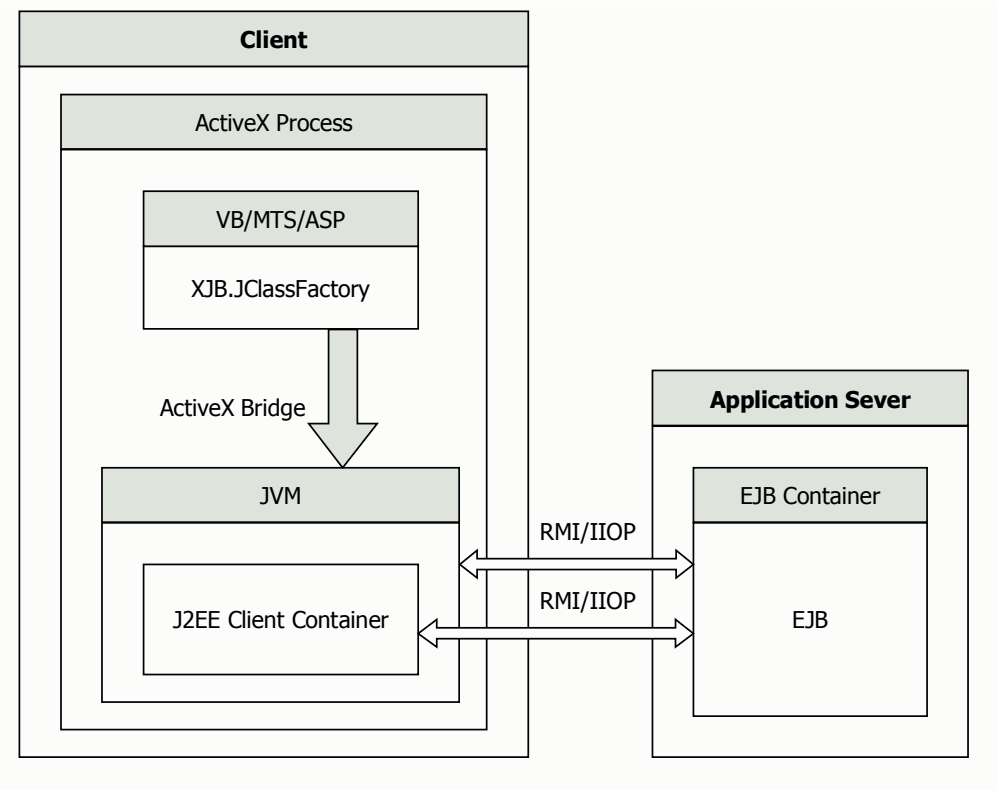


**Figura 6**: Arquitectura del ActiveX Bridge

El bridge además del acceso a componentes EJB permite el acceso a las APIs de Java, como pueden ser JNDI (*Java Naming and Directory Interface*), JDBC (*Java Database Connectivity*), JMS (*Java Messaging Service*), etc. El bridge soporta manejo de control de cargas y seguridad, pero no soporta transacciones distribuidas que involucren ambos ambientes. Otra carencia es que la comunicación es unidireccional, solo es posible invocar servicios o componentes Java desde COM, pero no es posible hacerlo en el otro sentido.

La documentación establece que el ActiveX bridge está soportado para ejecutar sobre Visual Basic, VBScript, ASP sobre plataformas NT o Windows 2000, pero debería poder hacerlo en todo lenguaje con soporte COM.

## 6. Comparación

A los efectos de poder comparar y optar por una solución particular frente a un problema particular, a continuación se presenta una lista de los puntos fuertes y débiles de cada solución analizada:

- Los Web Services cuentan con la ventaja de que están soportados en forma nativa por ambas plataformas, lo cual permite una solución portable, de bajo costo y fácil de implementar. Por otro lado los Web Services tienen la desventaja que aún no permiten manejar transacciones entre ambas plataformas, lo cual lleva a que la integración sea factible sólo para consultar información, no para la actualización. Una solución basada en Web Services permite la integración de cualquier plataforma, es algo mucho más general que la integración J2EE/.NET.
- Por otro lado la solución propuesta por JCA promete ser la más completa, ofrece una solución elegante a todo el problema de integración de ambas plataformas. Al permitir realizar transacciones que se extiendan en ambas plataformas permite realizar una integración completa. Al ser JCA es una especificación propuesta por Sun la cual es implementada por diferente Servidores de Aplicaciones hace que sea una solución portable. Se debe considerar que esta tecnología es relativamente nueva por lo cual la confiabilidad y robustez pueden representar un problema. Además la necesidad de disponer de un Resource Adapter para cada EIS puede llevar a que sea una solución cara.
- Las otras dos propuestas Ja.Net y ActiveX bridge plantean una solución propietaria generando una dependencia fuerte con un producto en particular. El uso de ActiveX bridge restringe completamente la portabilidad de los EJB que hagan uso de esta tecnología a los servidores de aplicaciones IBM WebSphere. Al igual que los Web Services, estas propuestas no son completas dado que no soportan transacciones.

## 7. Conclusiones

Existen varias alternativas para solucionar el problema de interoperabilidad entre las plataformas J2EE/.NET y cada una de ellas tiene sus puntos fuertes y débiles. Ninguna de ella muestra ser la solución por excelencia, por lo cual, frente a un problema particular de integración la elección del mecanismo a utilizar está fuertemente ligada a los requerimientos particulares que el problema imponga.

La elección de un mecanismo de integración se deben evaluar varios factores, portabilidad de la solución, confiabilidad y robustez, costo, soporte nativo, funcionalidades requeridas (ej. soporte transaccional). Los mecanismos más prometedores son los WebServices y JCA. Los WebServices aunque aún plantea una solución incompleta (aún no incorpora soporte de transacciones) prometen ser la solución más económica, estándar, portable y fácil de implementar. Por otro lado JCA promete ser una solución completa (soportando transacciones), estándar y portable.

## Referencias


[1] Java 2 Enterprise Edition, J2EE Connector Architecture Specification. Version 1.0, Java Community Process
Rahul Sharma

[2] J2EE Connector Architecture Product Information
http://java.sun.com/j2ee/connector/products.html

[3] Integrating EJB and COM Using the J2EE Connector Architecture
Mark Hansen and Peter Mamorski
Javector Software
http://e-serv.ebizq.net/obj/hansen_1.html

[4] J2EE Connector Architecture
http://java.sun.com/j2ee/connector/

[5] Ja.NET: Integration between Java and Microsoft .NET
Intrinsyc
http://www.intrinsyc.com/pdfs/whitepapers/janet_whitepaper.pdf

[6] Ja.NET
http://www.intrinsyc.com/products/bridging/janet.asp

[7] Bridging Enterprise JavaBeans TM and COM-based Enterprise Information Systems Via the J2EE Connector Architecture
Intrinsyc
http://www.intrinsyc.com/pdfs/jca-com_bridge.pdf

[8] WebSphere Application Server Enterprise Edition 4.0
A Programmer's Guide
International Business Machines
http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246504.html

[9] Integrating data and transactions for agile e-business
International Business Machines
http://www-1.ibm.com/partnerworld/pwhome.nsf/mktgsale/infra_wp.html

[10] WebServices.Org
http://www.webservices.org

[11] Transactional Attitudes: Reliable Composition of Autonomous Web Services
Thomas Mikalsen, Stefan Tai, Isabelle Rouvellou
IBM T.J. Watson Research Center, Hawthorne, New York, USA
http://www.research.ibm.com/AEM/pubs/wstx-WDMS-DSN2002.pdf

[12] Reliability of Composed Web Services From Object Transactions to Web Transactions
Thomas Mikalsen, Isabelle Rouvellou, Stefan Tai
IBM T.J. Watson Research Center, New York, USA
http://www.research.ibm.com/AEM/pubs/web_services_oopsla2001.pdf

[13] Web Services Transactions (WSTx)
International Business Machines
http://www.research.ibm.com/AEM/wstx.html

[14] J2EE Activity Service for Extended Transactions
http://jcp.org/jsr/detail/95.jsp

[15] Versata announces Web Services add-on
http://www.webservices.org/index.php/article/articleview/248/1/47

[16] SOAP Chained Transactions
http://www.webservices.org/index.php/link/category/22

[17] GLUE
The Mind Electric
http://www.themindelectric.com/glue/index.html

[18] Web Services Toolkit Overview
IBM Web Services Toolkit Development Team.
Parte de la documentación del IBM Web Service Toolkit
http://www.alphaworks.ibm.com/tech/webservicestoolkit

[19] IBM Web Service Tutorial. Implementing Web Services
James Snell
Parte de la documentación del IBM Web Service Toolkit
http://www.alphaworks.ibm.com/tech/webservicestoolkit