

# **Notification Service for EJB™**

---

Seamless Notification-to-EJB interworking

**A Borland White Paper**

*By Ke Jin, VisiBroker Development*

November 2002

---

**Borland®**

## **Contents**

<b>Introduction.....</b>	<b>3</b>
<b>EJB™ is defined on top of CORBA® and IIOP®.....</b>	<b>3</b>
<b>Is JMS a "golden hammer?".....</b>	<b>4</b>
<b>Why not the Notification-to-JMS workaround? .....</b>	<b>6</b>
<b>The best workaround is not to workaround.....</b>	<b>8</b>
<b>Typed Channel RMI-to-RMI/EJB scenario .....</b>	<b>10</b>
<b>Structured channel CORBA-to-RMI/EJB scenario .....</b>	<b>12</b>

## Introduction

Enterprise JavaBeans™ (EJB™) 2.0 is CORBA®/IIOP® interoperable. Therefore, Notification-to-EJB is a plain intra-CORBA interworking, which is seamlessly supported by VisiNotify,™ an add-on service to Borland® Enterprise Server, VisiBroker® Edition. This contrasts with the poor solution of mapping one CORBA entity—Object Management Group™ (OMG™) Notification—into a non-CORBA entity (Java™ Message Service (JMS)) to interwork with another CORBA entity (EJB) via a CORBA-to-non-CORBA integration workaround (Message Driven Beans (MDB)).

## EJB is defined on top of CORBA and IIOP

EJB is one of the most well-known distributed platforms for developing and deploying component-based, object-oriented business applications. The Java™ 2 Platform, Enterprise Edition (J2EE™)/EJB is designed to be CORBA/IIOP compatible from the ground up. J2EE 1.3 requires OMG RMI-over-IIOP to be one of the native protocols for Java RMI (J2EE 1.3 section 7.2). EJB 2.0 further mandates RMI-over-IIOP to be its only interoperable protocol (EJB 2.0 section 19.2). Additionally:

- IIOP transaction propagation is recommended by the J2EE 1.3 specification. And Java Transaction Service (JTS) is actually an implementation of the OMG Object Transaction Service (J2EE 1.3 section 4.3.1).
- The interoperability of JNDI is based on the OMG COS Naming Service (J2EE 1.3 section 7.2.2).
- EJB security interoperability is based on Conformance Level 0 of the OMG CORBA Security Interoperability Version 2 (CSIv2) (EJB 2.0 section 19.8.2).

This CORBA/IIOP compatible architecture establishes the J2EE/EJB platform on top of a widely accepted, open interoperable infrastructure, and automatically inherits its rich assets. Specifically, using RMI-over-IIOP as its native protocol, J2EE/EJB naturally becomes part of

the CORBA platform and seamlessly interworks with OMG distributed services, such as Security, Transaction, Naming, and Event/Notification.

However, not all Event/Notification Service implementations can seamlessly interwork with RMI-over-IIOP or EJB 2.0 applications. A seamless RMI-over-IIOP or EJB interworking requires the Event/Notification Service implementation to support the following:

- Valuetypes or Java serializable objects in event messages.
- A practically useable, less restrictive OMG Typed Channel.
- Channel to handle mismatches between IDL interface operations and their corresponding Java remote interface invocations.

Without the above support, J2EE/EJB event-driven applications could not use the native solution, namely the OMG Event/Notification Service, for de-coupled asynchronous communications but would have to use other non-native alternatives.

## **Is JMS a “golden hammer?”**

A non-native alternative for J2EE/EJB event-driven applications is the Java Message Service (JMS) along with the Message Driven Bean (MDB). JMS is designed as a portable Java API allowing applications to integrate with traditional Message Oriented Middleware (MOM) and MDB is its EJB client abstraction. There are some proven MOM products, and they support more communication models, semantics, and QoS policies than the OMG Event/Notification Service. Therefore, JMS/MDB were quickly accepted in the J2EE community as the de facto solution for event-driven applications.

Nevertheless, JMS is neither an interoperable messaging standard nor a native asynchronous interworking architecture for J2EE/EJB. JMS is only a portable API for MOM-to-Java integration. The JMS is not CORBA/IIOP compatible, nor does it define its own interoperable protocol. It has no common services, such as security and firewall tunneling. Using it to support a primary communication model would imply a proprietary infrastructure to co-exist with the open CORBA/IIOP infrastructure in J2EE/EJB. The resulting heterogeneous

infrastructures would defeat the J2EE/EJB design goals and compromise the advantages of having an open, interoperable, and unified infrastructure. For instance, as shown in Figure 1, applications may have to use two distinct authentication services, thereby forcing users to login twice.

Similarly, MDB is an ad hoc bean model designed for MOM-to-EJB integration via the JMS. The MDB does not comply with the RMI/EJB object model. Namely, the syntax of a MDB is not abstractly defined by a Java remote interface but buried in its informal, type unsafe, manually written message pack/unpack code. Using JMS/MDB to realize a primary communication model would also defeat EJB design goals and create many inconsistencies and inconveniences including:

- A bean developer has to manually pack/unpack messages and pay attention on synchronization mode instead of only focusing on business logic.
- A bean designed for a synchronous model could not be used or reused in an asynchronous model. Namely, business logic's reusability depends on system configuration.
- Although it is not uncommon in business processes that an object needs to handle both synchronous requests and asynchronous messages simultaneously, two distinct beans have to be developed and deployed in EJB, as shown in Figure 1.

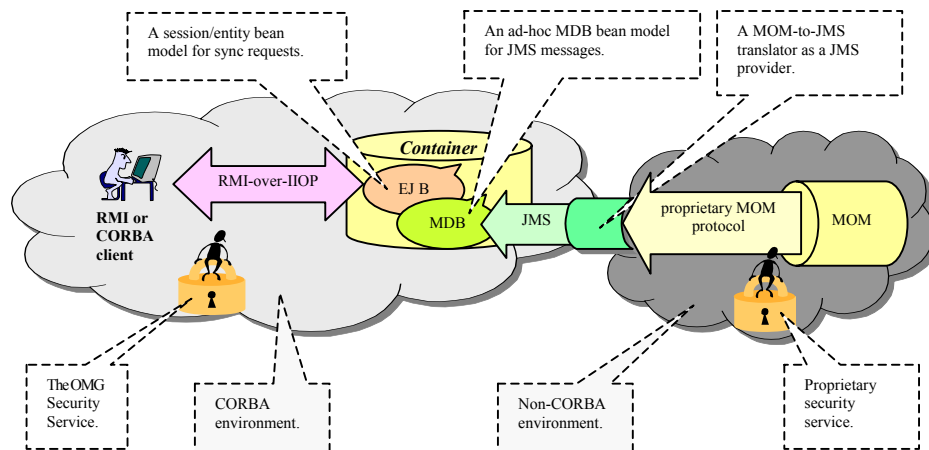


Figure 1. MOM-to-JMS/MDB. Heterogeneous infrastructures and bean models.

Conclusively, JMS/MDB should be used in situations that they are designed for, specifically integrating a traditional MOM. In turn, a traditional MOM should be used if it has already been used in a legacy system, or if it provides desired feature(s) not available elsewhere. Neither JMS nor any traditional MOM is the interoperable messaging standard for J2EE/EJB. For ordinary asynchronous communication, J2EE applications should use the native interoperable infrastructure of the J2EE/EJB platform, namely a RMI-over-IIOP based service if its semantic functions and QoS policies are adequate.

## Why not the Notification-to-JMS workaround?

The idea of Notification-to-JMS interworking simply translates OMG Structured Events into JMS messages. This equivalently wraps the OMG Notification Service as a JMS provider. This approach partially allows asynchronous communication to use the CORBA/IIOP infrastructure in the J2EE/EJB platform and also allows OMG Structured Events to be delivered to JMS/MDB applications, as illustrated in Figure 2.

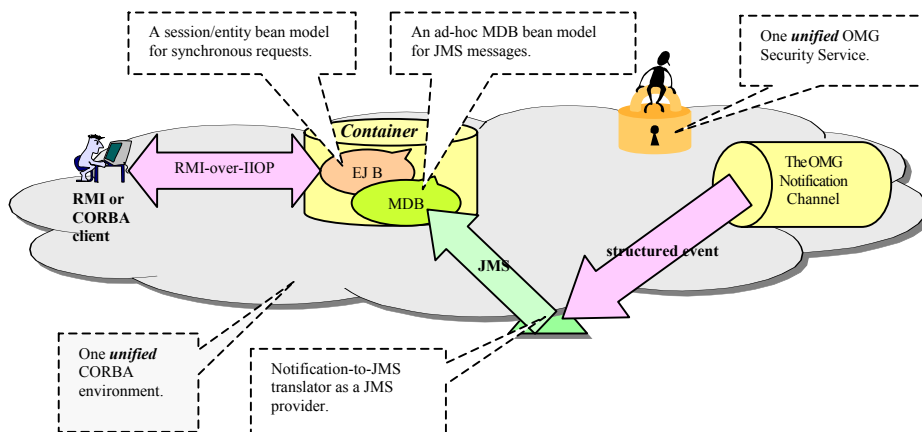


Figure 2. Notification-to-JMS/MDB. Unified infrastructure but heterogeneous bean models.

However, this idea has the following problems:

- **It is complicated for bean developers to an impractical degree.** Not all OMG IDL data types can be mapped into JMS. For instance, it is pretty common that IDL structure, union, and sequence are used as filterable\_data property values<sup>1</sup>, which have no corresponding mappings in JMS. Either the translation flattens them into multiple primitive data property components, or it moves them into the message body. The first option substantially increases the JMS consumer or MDB complexity. The second approach would lead to very complicated translation rules.
- **It is not a solution.** For Notification-to-EJB interworking, as EJB is CORBA interoperable, straightforward intra-CORBA interworking scenarios should be pursued. It is logically absurd to first map one CORBA system (namely, the Notification Service) into a non-CORBA system, and then interwork with another CORBA system (namely, the EJB) via a CORBA-to-non-CORBA integration workaround.
- **It is putting the cart before the horse.** JMS is not an interoperable messaging standard of Java but only a portable API for traditional MOM-to-J2EE integration. JMS mapping should be considered if it improves portability and/or interoperability. The OMG Event/Notification Service is inherently portable<sup>2</sup> and interoperable in J2EE/EJB. In this situation, Notification-to-JMS mapping only compromises portability, interoperability, and reusability, which makes it completely unnecessary and even harmful.
- **It has inherent performance weakness.** This mapping is Structured Event oriented. As such, it inherently falls short on performance, scalability, type safety, and formal event definition. It also means that, even if used as a JMS-to-JMS solution, it will unlikely match the performance and scalability of native MOM-based JMS. Besides, this mapping introduces another user level of event translation and performance overhead<sup>3</sup>.
- **It still does not support Java serializable objects in CORBA Structured Events or in JMS ObjectMessages.**

---

<sup>1</sup> For instance, TMN notifications defined by ITU-T and TMF use either TimeBase::UtcT or CosNaming::Name in filterable\_data.

<sup>2</sup> Actually, it is more portable than JMS because it is language independent.

<sup>3</sup> For instance, JMS property iteration is more expensive than directly accessing and scanning the filterable\_body name-value pair sequence.

## The best workaround is not to workaround

As said previously, the OMG Event/Notification Service is inherently portable and interoperable with J2EE/EJB. Therefore, it already provides a natural solution for asynchronous and message communication in J2EE/EJB environments.

ORB and EJB platform vendors usually only support one of these two technologies and ignore the other. Borland, however, considers the ORB to be the core infrastructure of EJB. Borland is not only the leading ORB vendor but also a leading EJB platform vendor. Borland Enterprise Server, the Borland platform for EJB, is built entirely on top of the VisiBroker infrastructure. Therefore, Borland Enterprise Server is inherently CORBA/IIOP compliant and the VisiBroker Edition of Borland Enterprise Server, the add-on service VisiNotify, and other common object services<sup>4</sup> are all built to be J2EE/EJB compatible.

VisiNotify is an OMG Event/Notification Service implementation. With its full IIOP compliance, VisiNotify can seamlessly support J2EE/EJB applications without any workaround. As a general overview, Typed or Structured Events can be directly delivered to ordinary EJB session and entity beans via standard RMI-over-IIOP with VisiNotify (see Figure 3). Note that the Notification-to-JMS and JMS-to-MDB mapping bridges originally shown in Figure 2 are completely eliminated.

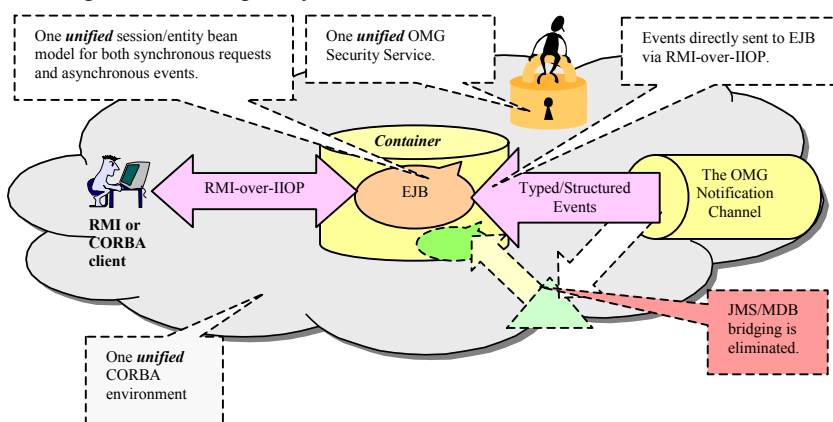


Figure 3. Direct Notification-to-EJB. Unified infrastructures and bean model.

<sup>4</sup> Such as VisiTransact,™ VisiSecure,™ Naming Service, GateKeeper.

Specifically, VisiNotify supports two J2EE/EJB application scenarios in the current release, namely the Typed Channel scenario and Structured Channel scenario. A third scenario is currently under development. These two scenarios are straightforward and fully compliant to both CORBA/IIOP and J2EE/EJB standards. Rather than introducing another foreign proprietary infrastructure, the synchronous and asynchronous communications in J2EE/EJB are now handled by the same CORBA/IIOP infrastructure, which brings the following advantages:

- **A unified set of common distributed services can be used in both synchronous and asynchronous communications.** Users are not forced to use two separate sets of services (such as firewall, proxy, security, and authentication).
- **It allows J2EE/EJB applications to use the OMG Typed Event Service** and to take advantage of type safety, efficiency, formal event definition, and other features.
- **It allows the structured channel to be used for integrating the existent Structured Event system with EJB.**
- **It has no restrictions on data types.** All valid IDL types can be used in Typed or Structured Events.
- **It introduces no data level message translation.** The structure to structure valuebox translation is performed at the CDR level instead of on individual data element<sup>5</sup>. Therefore, it is much more efficient than the JMS wrapper solution.
- **It respects the EJB abstract object model.** The bean designated to receive an asynchronous message can be defined by RMI interfaces with user-defined type safe operations.
- **The same bean can be used to handle synchronous and asynchronous messages independently or simultaneously.** It is not necessary to write and deploy two distinct beans.

There is another paper from Borland, “Using EJB with the Notification Service: Challenges and Benefits,” which gives more detailed information on this subject and turnkey examples; see <http://www.borland.com>. Also, Borland is developing a third Notification-to-EJB

---

<sup>5</sup> Borland is developing a third scenario of Notification-to-J2EE/EJB. In this scenario, this “to-valuebox” translation will be eliminated.

interworking scenario, which is compatible with and more elegant and flexible than the two scenarios mentioned here. Specifically, the third scenario will allow Untyped, Structured, Sequence, RMI/IDL-typed, and OMG IDL-typed events to be sent directly to an EJB session or entity beans without any translation.

## Typed Channel RMI-to-RMI/EJB scenario

This scenario is suitable for pure RMI/EJB configurations, where all Typed Event suppliers are RMI clients<sup>6</sup> and all Typed Event consumers are either RMI objects or ordinary EJB session/entity beans as described below and shown in Figure 4:

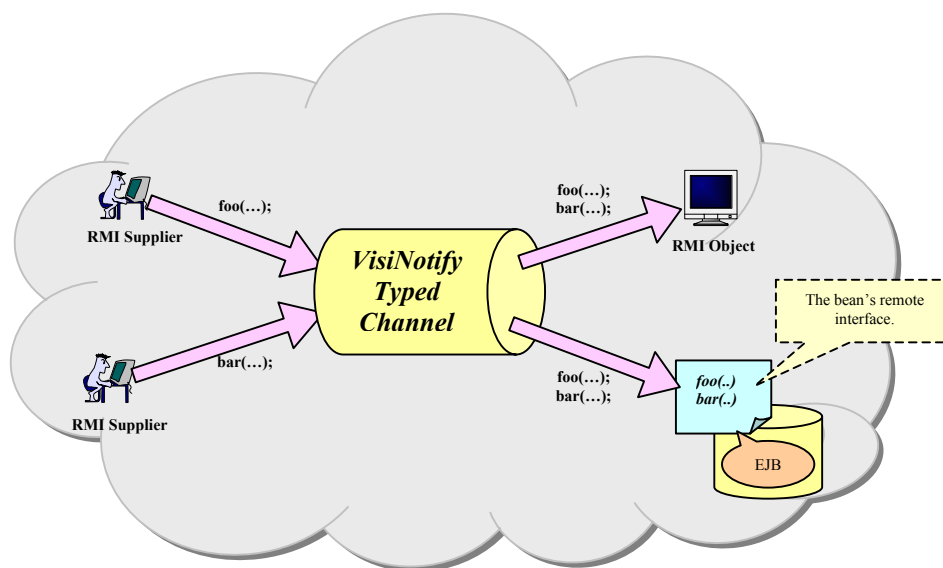


Figure 4. Typed channel RMI-to-RMI/EJB scenario.

<sup>6</sup> It is fully possible that some of the suppliers are CORBA clients, i.e. they are using OMG IDL stubs and are written in other non-Java languages. However, the IDL definitions they use are generated from the RMI remote interface using java-to-IDL mapping. Therefore, these clients are still categorized as RMI clients.

- Typed Event interfaces are defined by Java remote interfaces instead of OMG IDL interfaces. For instance:

```
public interface MyRMITypedEvent extends javax.ejb.EJBObject {
    public void foo(int a) throws RemoteException;
    public void bar(String s) throws RemoteException;

    ... // other operations
}
```

- There is no special requirement to the bean itself. It could be a ordinary session or entity bean. Its home interface should have a parameter-less create() method. For instance:

```
public class MyRMITypedEventBean implements javax.ejb.SessionBean {
    ... // private members

    public void foo(int a) { ... }
    public void bar(String s) { ... }

    ... // other operations
}
```

- References of RMI objects or remote interfaces of beans are connected to the given VisiNotify Typed Channel as the <I> interfaces. This can be done programatically or use the VisiNotify subtool utility.
- RMI client (supplier) applications obtain typed push proxies from the Typed Channel and call get\_typed\_consumer() on them to retrieve <I> interfaces.

```
// get the <I> interface
omg.org.CORBA.object obj = proxy.get_typed_consumer();
```

- Then, these RMI client (supplier) applications narrow down the <I> interfaces into the desired RMI remote interfaces and invoke RMI methods on them to send typed events.

```
// narrow down to the stub
MyRMITypedEvent consumer = MyRMITypedEventHelper.narrow(obj);
// sending events
consumer.foo(123);
consumer.bar("hello");
...
```

- The VisiNotify Typed Channel forwards these RMI invocations to connected consumers' <I> interfaces. The bean has no knowledge about the synchronization mode of the invocation. In fact, the same foo() or bar() operations in the mentioned example can be invoked directly by a RMI client or asynchronously via a Notification Channel.

Obviously, this scenario is identical to an ordinary CORBA Typed Channel scenario, except it is now used in a pure Java RMI/EJB environment. The <I> interfaces are no longer CORBA interfaces defined by OMG IDL but rather RMI remote interfaces defined by RMI/IDL.

## Structured channel CORBA-to-RMI/EJB scenario

This scenario allows ordinary EJB session/entity beans to receive Structured Events from VisiNotify. All suppliers in this scenario are CORBA Structured Event clients, while some consumers are CORBA Structured Event objects and other consumers are ordinary EJB session/entity beans described below and shown in Figure 5, including:

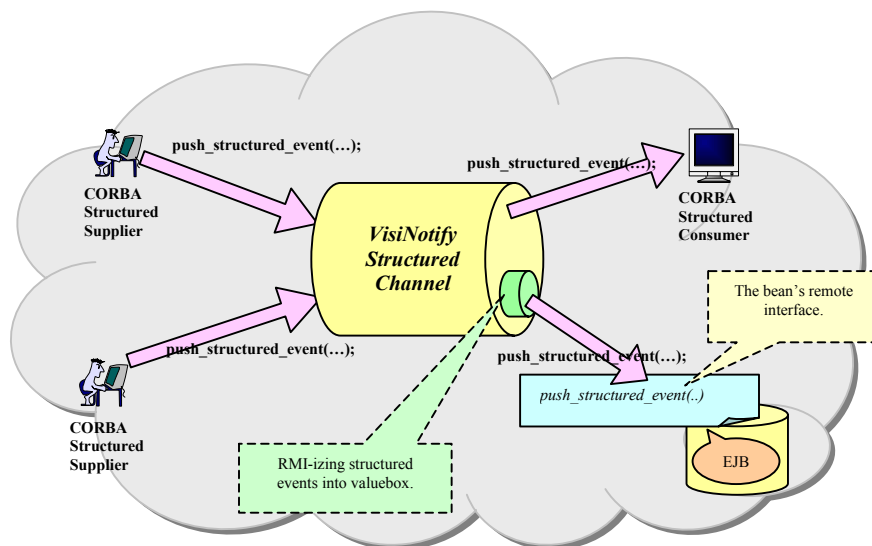


Figure 5. Structured channel CORBA-to-RMI/EJB scenario.

- The EJB bean implements, among other operations, a

`push_structured_event(StructuredEvent)` method. For instance:

```
import omg.org.CosNotification.*;
public class MyEventBean implements javax.ejb.SessionBean {
    ... // private members

    // the structured event method
    public void push_structured_event(StructuredEvent ev){...}

    ... // other operations
}
```

- The remote interface of the bean also declares this method with the restriction of not being overloaded. For instance:

```
public interface MyEvent extends javax.ejb.EJBObject {
    // the structured event method
    public void push_structured_event(
        omg.org.CosNotification.StructuredEvent ev)
        throws RemoteException;

    ... // other operations
}
```

- Having this bean deployed and connecting its remote interface to the VisiNotify Structured Channel using the VisiNotify subtool utility.
- On forwarding CORBA Structured Events to EJB through their remote interfaces, VisiNotify handles the mismatch between the OMG IDL `push_structured_event` (in `StructuredEvent`) and its RMI counterpart. This is referred to as “OMG IDL structured event to RMI/IDL structured event translating” or “RMI-izing” for short.

**Made in Borland®** Copyright © 2002 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other marks are the property of their respective owners. Corporate Headquarters: 100 Enterprise Way, Scotts Valley, CA 95066-3249 • 831-431-1000 • www.borland.com • Offices in: Australia, Brazil, Canada, China, Czech Republic, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Korea, the Netherlands, New Zealand, Russia, Singapore, Spain, Sweden, Taiwan, the United Kingdom, and the United States. • 13451