Sun Microsystems

Enterprise JavaBeans^{TM} to CORBA Mapping

This is a draft specification of the standard mapping of Enterprise JavaBeans to CORBA.

Sun Microsystems Inc. Proprietary and Confidential document

Please send technical comments on this specification to:

ejb-spec-comments@sun.com

Please send product and business questions to:

ejb-marketing@sun.com

Version 0.8

Rohit Garg

December 9, 1997 3:03 pm

Contents

1	Introduction	4
2	Goals	5
3	Mapping of Distribution	7
4	Mapping of Naming	9
5	Mapping of Transactions	10
6	Mapping of Security	13
Appendix A:	References	14
Appendix B:	Enterprise JavaBeans IDL	15
Appendix C:	Example Application	18

1 Introduction

Enterprise JavaBeans (EJB) [1] is a component architecture for development and deployment of object-oriented distributed enterprise-level Java applications. Applications written using Enterprise JavaBeans are scalable, transactional, and multi-user secure. These applications can be written once, and then deployed on any EJB-enabled server platform.

We expect that many EJB servers will be based on the CORBA/IIOP [2] industry standard. To ensure interoperability among the CORBA-based implementations from multiple-vendors, we have defined a standard mapping of EJB to CORBA. This document corresponds to the EJB specification version 0.80.

We thank the reviewers for their time and effort in helping us to improve the specification.

1.1 Target Audience

The target audience for this specification are the vendors of transaction processing platforms, vendors of enterprise application tools, and other vendors who want to use the CORBA/IIOP standard to provide support for Enterprise JavaBeans in their products.

1.2 Mapping Overview

The EJB to CORBA mapping is divided into four areas:

- *Mapping of Distribution* - defines the relationship between an Enterprise JavaBean and a CORBA object, and the mapping of the Java RMI remote interfaces defined in the EJB specification to OMG IDL.
- *Mapping of Naming* - specifies how COS Naming is used to locate the Container objects
- *Mapping of Transactions* - defines the mapping of the EJB transaction support to the OMG Object Transaction Service v1.1 [6]
- *Mapping of Security* - defines the mapping of the security features in EJB to CORBA security

1.3 Acknowledgments

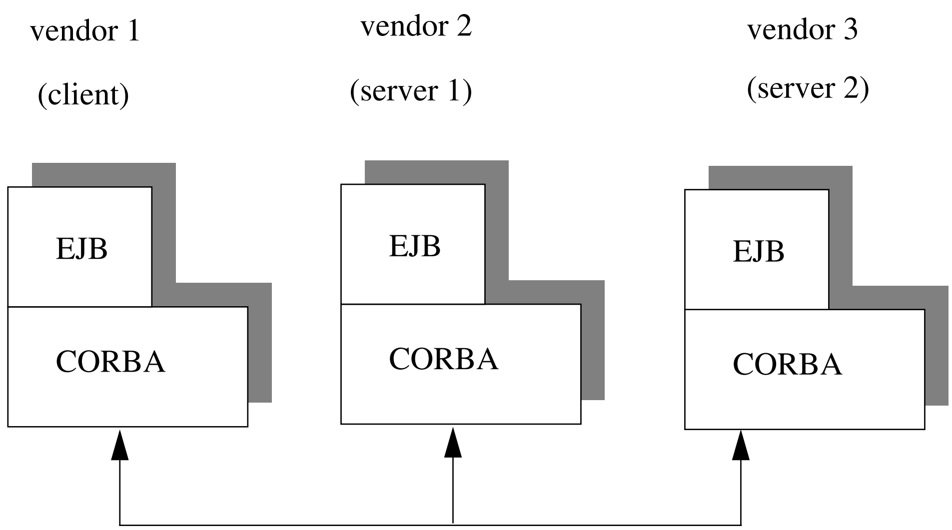
Vlada Matena, Graham Hamilton, Anil Vijendran, and Sanjeev Krishnan have provided invaluable input to this specification.

2 Goals

The primary goal of this specification is to define "on-the-wire" interoperability so that multiple CORBA based implementations of the EJB specification can interoperate on the network.

The specification makes it possible to provide the following EJB/CORBA interoperability:

- A CORBA client (written in any CORBA supported language) can access Enterprise JavaBeans deployed in a CORBA based transaction server
- A program can mix calls to CORBA and EJB objects in a transaction.
- A transaction can span multiple EJB objects that are located on multiple CORBA-based EJB servers, provided by different vendors.



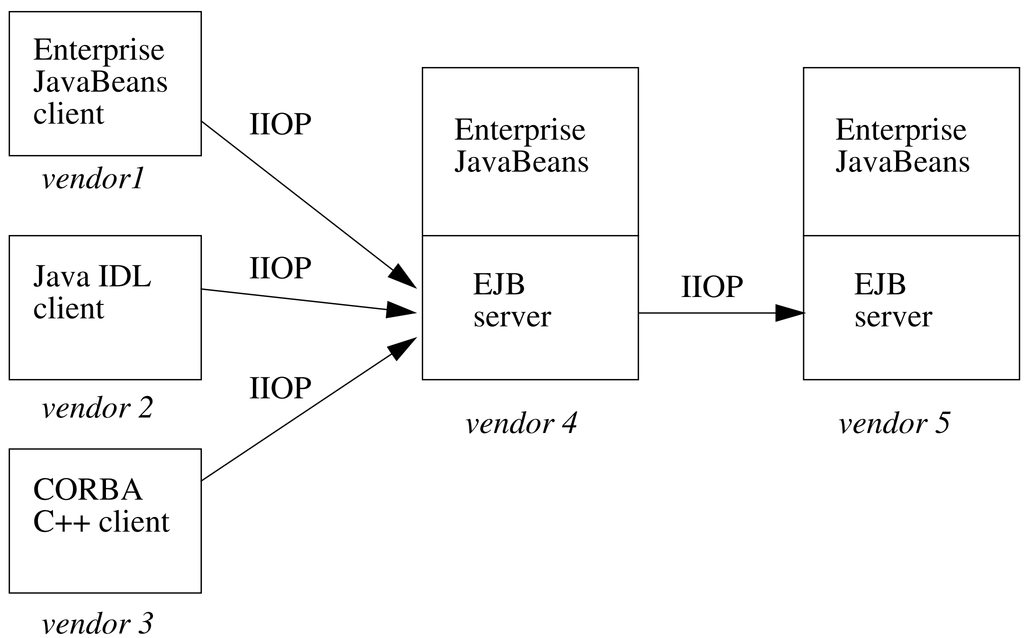
2.1 Types of CORBA Clients

There are two types of CORBA clients to an EJB Server:

- **EJB/CORBA Client** - A Java client that uses the EJB APIs. The client uses JNDI to locate objects, Java RMI over IIOP to invoke remote methods, and the *java.jts.CurrentTransaction* interface to demarcate transaction boundaries. The use of CORBA IDL is implicit (i.e. the programmer writes only Java code and the corresponding CORBA IDL is generated automatically by tools).

- **Plain CORBA Client** - A client written in any language that uses a language specific bindings of the CORBA IDL. Such a client uses COS Naming to locate objects, CORBA IDL to invoke remote methods, and OTS to demarcate transactions. The use of CORBA IDL is explicit (i.e. the programmer creates an IDL file and runs an IDL compiler to generate stubs for a given language).

The mapping needs to ensure that both types of clients interoperate with a CORBA-based EJB server.



3 Mapping of Distribution

Even though CORBA does not mandate using IIOP, lately, CORBA and IIOP have become synonymous. This document assumes that the EJB/CORBA compliant implementations are using IIOP[1] as the communication protocol.

3.1 Mapping Java Remote Interfaces to IDL

Each enterprise bean is associated with a Java remote interface. The mapping of an enterprise bean's remote interface to CORBA IDL is specified by the standard Java to IDL mapping [12] specification.

> *This mapping is possible because the EJB specification restricts the types used by the enterprise bean's remote interface to the types supported by the standard Java to IDL mapping.*

3.1.1 Mapping of *ejb.java.EJBObjet* interface

The IDL interface generated from the bean's remote interface must inherit from the *EnterpriseJavaBeans::EJBObject* IDL interface.

3.1.2 Marking of transaction-enabled enterprise bean objects

An enterprise bean whose transaction attribute is set to *BEAN_MANAGED*, *SUPPORTS*, *REQUIRED*, *REQUIRES_NEW*, or *MANDATORY* is said to be *transaction-enabled*.

The CORBA mapping of a transaction-enabled enterprise bean must follow these rules:

- The IDL interface for the enterprise bean's remote interface must inherit from the *CosTransactions::TransactionalObject* IDL interface[2].
- For an entity enterprise bean, the IDL interface for the enterprise bean's factory and finder interfaces must inherit from the *CosTransactions::TransactionalObject* IDL interface.

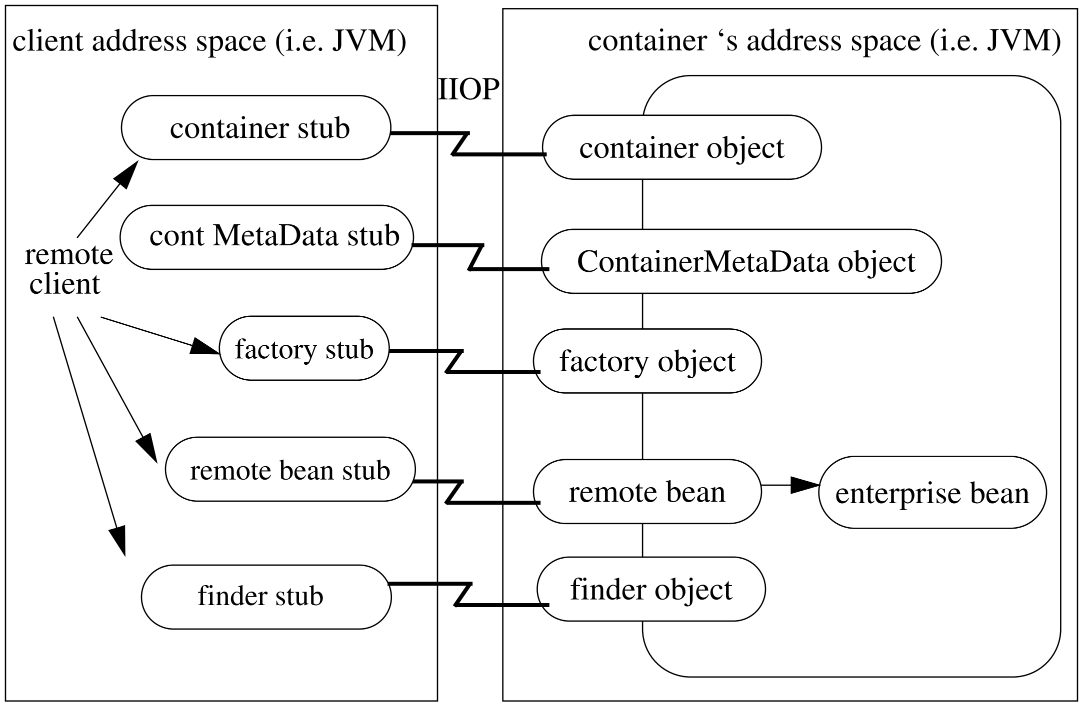
The IDL interface for an enterprise bean whose transaction attribute is set to *NOT_SUPPORTED* must not inherit from the *CosTransactions::TransactionalObject* IDL interface.

See "Generated IDL" on page 19 for an example of mapping the Java Remote interfaces in EJB to IDL using the above rules.

3.2 Client Side Stubs

The following figure illustrates the runtime objects used in a typical distributed EJB-enabled CORBA environment.

1.plain insecure IIOP, SECIOP, or IIOP over SSL
2.This is necessary to ensure the propagation of a transaction context from the client to the enterprise bean.

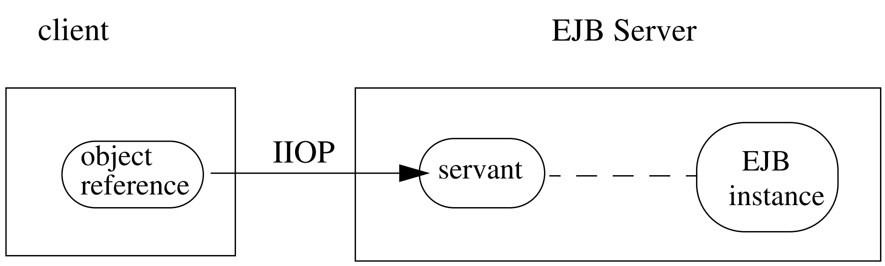


Depending on the client type, the client stubs are either RMI/IIOP stubs, or plain IDL stubs defined by the language-specific CORBA mappings.

An example of the complete IDL for the remote interfaces in the Enterprise JavaBeans is listed in "Enterprise JavaBeans IDL" on page 15.

3.3 CORBA Object and Enterprise Bean Relationship

As a server-side implementation technique, the CORBA runtime may use a servant implementing the enterprise bean's CORBA IDL to field a method invocation, and delegate the method invocation to the appropriate enterprise bean. One way to achieve this is use TIE based skeletons (as defined in the CORBA v2.x) that are initialized with the enterprise bean instance. Since the architecture of the stubs and skeletons does not relate to on-the-wire interoperability, it is not specified in this document.



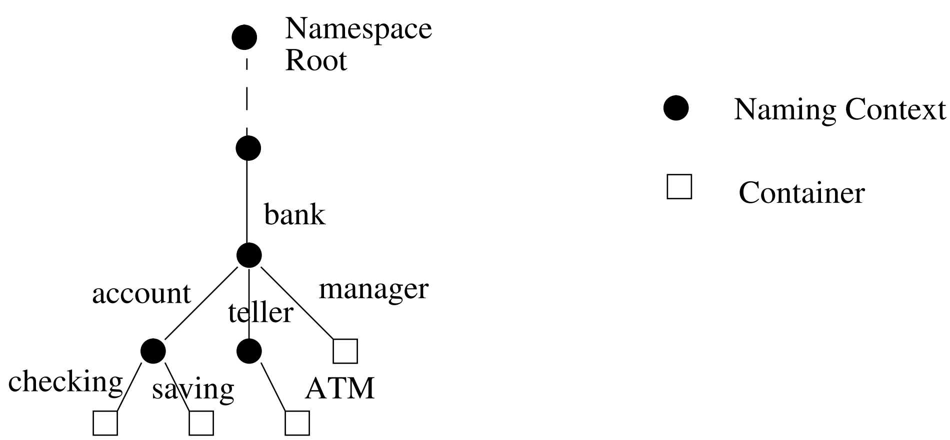
4 Mapping of Naming

An EJB Container is published in the COS Namespace by the EJB Server for each Enterprise Bean. To obtain a reference to a Container object, an EJB/CORBA client uses JNDI API, whereas a plain CORBA client uses the COS Naming API directly.

4.1 COS Namespace Layout

Each bean's deployment descriptor contains a property called the `containerName` that specifies the pathname in the name space at which to bind the enterprise bean's container object.

For example, if an ejb-jar contains four enterprise beans with container names: *bank/account/checking*, *bank/account/saving*, *bank/teller/ATM*, and *bank/manager* then the COS namespace may look as shown in the following figure:



5 Mapping of Transactions

The following sections describe the mapping of the transaction support in EJB to OMG Object Transaction Service (OTS) [6].

5.1 Transaction Propagation

The rules for the mapping of enterprise bean's remote interfaces to CORBA IDL (See Subsection 3.1.2) ensure that the IDL interface of the remote interface for a transaction-enabled enterprise bean inherits from the *CosTransactions::TransactionalObject* IDL interface.

Inheritence from the *CosTransactions::TransactionalObject* inheritence ensures that the ORB and OTS will propagate the client's transaction context (if the client is associated with any transaction context) to the enterprise bean object.

5.2 Container support for transactions

Every client method invocation on an enterprise bean object is interposed by the bean's container. The interposition allows the container to perform declarative transaction management.

The following table shows what actions are taken by the EJB server runtime for different values of the enterprise bean's transaction attribute:

Table 1: Declarative Transaction Attribute Management

Transaction attribute	Client's transaction	operation on TransactionCurrent on the Server
NOT_SUPPORTED	-	-
	T1	suspend T1 invoke bean resume T1
BEAN_MANAGED	-	-
	T1	T1
REQUIRED	-	begin T2 invoke bean end[a] T2
	T1	inherits T1
SUPPORTS	-	-
	T1	inherits T1

Transaction attribute	Client's transaction	operation on TransactionCurrent on the Server
REQUIRES_NEW	-	begin T2 invoke bean end T2
	T1	suspend T1 begin T2 invoke bean end T2 resume T1
MANDATORY	-	throw TRANSACTION_REQUIRED
	T1	inherit T1

a.end transaction refers to either commit or abort based on the outcome from the method invocation on the bean.

The following pseudo-code illustrates how a CORBA-based container should implement the required semantics of the enterprise bean's transaction attribute (the pseudo-code does not show exception handling):

- *NOT_SUPPORTED*

```
clientTransaction = Current.suspend();
result = instance.method(args);
Current.resume(clientTransaction);
return result;
```

- *BEAN_MANAGED*

```
return instance.method(args);
```

- *REQUIRED*

```
if (Current.getStatus() == StatusActive) {
        return instance.method(args);
} else {
        Current.begin();
        result = instance.method(args);
        Current.commit();
        return result;
}
```

- *SUPPORTS*

```
return instance.method(args);
```

- *REQUIRES_NEW*

```
clientTransaction = Current.suspend();
Current.begin();
result = instance.method(args);
Current.commit();
Current.resume(clientTransaction);
return result;
```

- *MANDATORY*

```
if (Current.getStatus() == StatusActive) {
        return instance.method(args);
} else {
        throw TRANSACTION_REQUIRED;
}
```

5.3 Client-side Demarcation

A CORBA client will typically use the OTS Current interface to demarcate transaction boundaries. An EJB CORBA-based infrastructure must propagate the client's transaction context to the transaction-enabled enterprise beans.

6 Mapping of Security

The main security concern in the EJB specification is **access control**, which requires the server ORB to determine the client's identity. Each bean also has an identity (specified in the bean's deployment descriptor,) which is used for ACL checking when the bean itself acts as a client to another bean, or when the bean invokes protected resources.

The client identity is based on the actual security/communication protocols used by the ORBs:

- *plain IIOP* - The client identity is the *CORBA::Principal* that comes over the wire as part of the IIOP Request Message. The Principal can be mapped by the ORB to the underlying operating system userid.
- *Common Secure IIOP* (CSI)[7] - The client identity is defined by the specific mechanism (GSSKerberos, SPKM, CSI-ECMA) used with SECIOP (Secure IIOP.)
- *IIOP over SSL* [8] - The client's identity is the X.500 distinguished name obtained using SSL client authentication.

  Note that when CSI or IIOP/SSL are used, then the CORBA::principal is deprecated. For real secure interoperability the ORB should implement CSI specification, or the CORBAsecurity/SSL specification.

Appendix A: References

[1] Enterprise JavaBeans Specification.

[2] CORBA/IIOP version 2.1 Specification (*http://www.omg.org/corba/corbaiiop.htm*)

[3] CORBA COS Security Service (*http://www.omg.org/corba/sectrans.htm#sec*)

[4] CORBA Interoperability (http://www.omg.org/docs/interop/96-05-01.ps)

[5] Naming Service (*http://www.omg.org/corba/sectrans.htm#nam*)

[6] Transaction Service (*http://www.omg.org/corba/sectrans.htm#trans*)

[7] Common Secure IIOP (CSI) (*http://www.omg.org/docs/orbos/96-06-20.ps)*

[8] CORBAsecurity/SSL Interoperability (*http://www.omg.org/docs/orbos/97-02-04.ps*)

[9] IDL Java Mapping 1.0 (*http://www.omg.org/docs/orbos/97-03-01.ps*)

[10] Java to IDL Mapping RFP (*http://www.omg.org/docs/orbos/orbos/97-03-08.ps)*

[11] Java to IDL Mapping, Joint Initial Submission, IBM, Netscape, Oracle, Sun, and Visigenic (*http://www.omg.org/docs/orbos/97-08-06.ps*)

Appendix B: Enterprise JavaBeans IDL

```
// ejb.idl

#include "java.lang.ExceptionValue.idl"

module java {

    module ejb {

        // NotDestroyableException exception
        value NotDestroyableExceptionValue:
                  ::java::lang::ExceptionValue {};
        exception NotDestroyableException {
            NotDestroyableExceptionValue value;
        };

        // InvalidKeyException exception
        value InvalidKeyExceptionValue:
                  ::java::lang::ExceptionValue {};
        exception InvalidKeyException {
            InvalidKeyExceptionValue value;
        };

        // NoObjectWithKeyException exception
        value NoObjectWithKeyExceptionValue:
                  ::java::lang::ExceptionValue {};
        exception NoObjectWithKeyException {
            NoObjectWithKeyExceptionValue value;
        };

        // Handle
        value Handle { state {}; };

        // PrimaryKey
        value PrimaryKey { state {}; };

        // Class
        value Class { state {}; };

        // forward references
        interface Container;
        interface ContainerMetaData;
        interface Factory;
        interface Finder;

        // Container
        interface Container {

            void destroy__(in Handle handle)
                raises(NotDestroyableException);

            void destroy__PrimaryKey(in PrimaryKey primaryKey)
```

```
            raises(NotDestroyableException);

        ContainerMetaData getContainerMetaData();

        Factory getFactory();

        Finder getFinder();

        };

    // ContainerMetaData
    interface ContainerMetaData {

        string getClassName();

        Container getContainer();

        Class getEnterpriseBeanClass();

        Class getPrimaryKeyClass();
    };

    // EJBObject
    interface EJBObject {

        void destroy() raises(NotDestroyableException);

        string getClassName();

        Container getContainer();

        Handle getHandle();

        PrimaryKey getPrimaryKey();

        boolean isIdentical(in EJBObject bean);
    };

    // Factory
    interface Factory {

        string getClassName();

        Container getContainer();
    };

    // Finder
    interface Finder {

        EJBObject findByPrimaryKey(in PrimaryKey primaryKey)
           raises (InvalidKeyException, NoObjectWithKeyExcep-
tion);

        string getClassName();
```

```
                Container getContainer();
            };
        };
    };
```

ISSUE: Using the current (draft proposal) Java RMI to IDL Mapping the module scoping is such that the Java stubs and skeletons are generated with the `java.ejb` package prefix. There needs to be a way of overriding this package name. One possible solution is to use a compiler directive #pragma javaPackage (available in the Java IDL compiler) to prefix all generated packages.

Appendix C: Example Application

This sections shows the IDL that is generated from the remote interface of a session bean and the pseudocode for a C++ client that invokes methods on it in the context of a client initiated transaction. The bean is marked as *transactionEnabled*=true.

C.1 Bean Remote Interface

```
package trading.interfaces;

import java.ejb.EJBObject;
import java.rmi.RemoteException;

/**
 * TraderRemote is a remote interface for the Trader server bean.
 */
public interface TraderRemote extends EJBObject {

    /* Buy shares of specified stock. */
    public void buy(String stockSymbol, int shares, double price)
     throws RemoteException, TooManySharesException;

    /* Sell shared of specified stock. */
    public void sell(String stockSymbol, int shares, double price)
     throws RemoteException, TooManySharesException;

    /* Submit orders. */
    public void submitOrders();
}
```

C.2 Bean's Remote Exceptions

```
package trading.interfaces;

/**
 * TooManySharesException is thrown if a client attempts to trade
 * more shares than the Trader server bean permits.
 */
public class TooManySharesException extends Exception {
    public TooManySharesException() {
     super();
    }

    public TooManySharesException(String s) {
     super(s);
    }

    int getMaxAllowed();

    private int maxAllowed;
}
```

C.3 Generated IDL

```
#include “ejb.idl”
#include “java.lang.ExceptionValue.idl”

module trading {

    value TooManySharesExceptionValue:
              ::java::lang::ExceptionValue {
        state {
            private int maxAllowed;
        };
    };

    exception TooManySharesException {
        TooManySharesExceptionValue value;
    };

    interface TraderRemote : ::java::ejb::EJBObject,
         CosTransactions::TransactionalObject {

        void buy(in string stockSymbol, in long shares,
                 in double price)
            raises (TooManySharesException);

        void sell(in string stockSymbol, in long shares,
                  in double price)
            raises (TooManySharesException);

        void submitOrders();

    };
};
```

Note in the generated IDL above that the TraderRemote IDL interface inherits from two IDL interfaces as required by the interface mapping rules described here.

C.4 C++ CORBA Client Pseudocode

```
...

// initialize the ORB
CORBA_ORB_ptr orb = CORBA_ORB_init(argc, argv, NULL);

// get initial reference for the Name Service
CORBA_Object_ptr iobj =
     orb->resolve_initial_reference(“NameService”);
CosNaming_NamingContext_ptr initial =
     CosNaming_NamingContext::_narrow(iobj);

// construct the CosNaming::Name for the Bean Container
CosName container_name(......);
```

```
// locate the container
CORBA_Object_var cobj = initial->resolve(container_name);
EnterpriseJavaBeans_Container_ptr container =
     EnterpriseJavaBeans_Container::_narrow(cobj);

// get the trader factory
trading_TraderFactory_var fobj = trading_TraderFactory::_narrow(
     container.getFactory());

// create a session bean
trading_TraderRemote_ptr trader = factory->create( ... );

// invoke on the trader bean under a client initiated transaction
try {
    txn_crt.begin();

    trader->buy("SUNW", 100, 52.00);
    trader->sell("ABCC", 50, 35.90);
    trader->submitOrders();

    txn_crt.commit(FALSE);

} catch (...) {
    txn_crt.rollback();
}

...
```