Sun Microsystems

Enterprise JavaBeans™ to CORBA Mapping

This is the specification of the standard mapping of Enterprise JavaBeans™ 1.0 architecture to CORBA.

Please send technical comments on this specification to:

ejb-spec-comments@sun.com

Please send product and business questions to:

ejb-marketing@sun.com

Rohit Garg

Version 1.0

March 23, 1998 3:32 pm

Contents

1 Introduction

Enterprise JavaBeans^{TM} (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 1.0.

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 EJBHome objects
- *Mapping of Transactions* - defines the mapping of the EJB transaction support to the OMG Object Transaction Service (OTS) v1.1 [6]
- *Mapping of Security* - defines the mapping of the security features in EJB to CORBA security

1.3 Acknowledgments

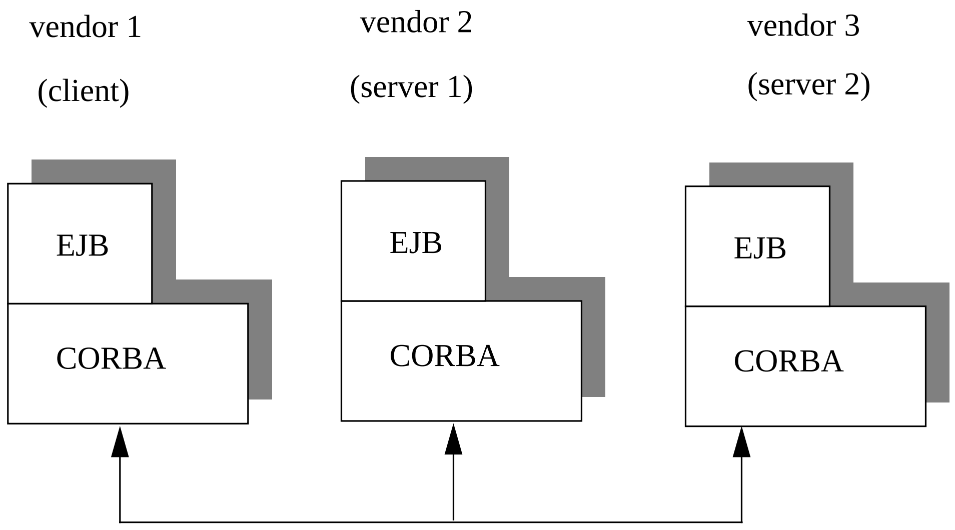
Vlada Matena, Graham Hamilton, Anil Vijendran, Sanjeev Krishnan, Shel Finkelstein, and David Heisser have provided invaluable input to this specification.

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.

That is, this 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 EJB server.
- A client program can mix calls to CORBA and EJB objects within 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

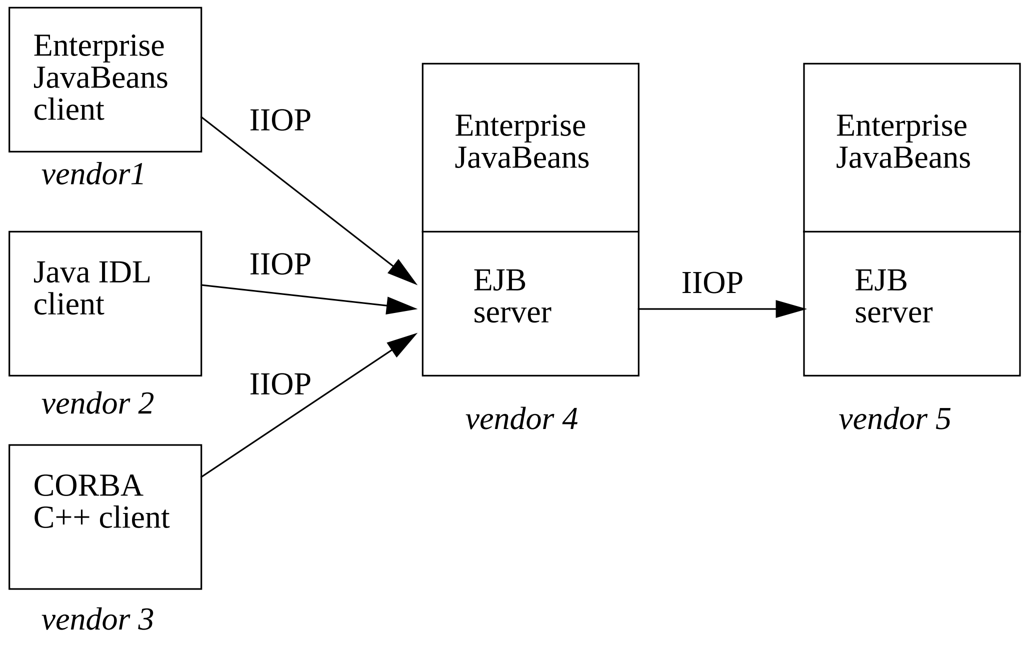
From the discussion above, there are two types of CORBA clients to an EJB Server:

- **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 *javax.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 used implicitly by the runtime).

An Enterprise JavaBean running in a CORBA based EJB server is also an EJB/CORBA client to other EJBs.

- **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 ensures that both types of clients interoperate with a CORBA-based EJB server by producing the same bits on the wire.



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

For each Enterprise JavaBean, that is deployed in the EJB Server, there are two remote interfaces - the bean's EJBHome interface, and the bean's EJBObject interface. The Java Language to IDL Mapping [11] and the Objects by Value [13] OMG specifications describe precisely how these remote interfaces are mapped to IDL. This mapping to IDL is typically implicit when Java RMI over IIOP is used to access the EJBs.

3.1.1 Mapping of Handle

The Handle object is a value object. Its standard mapping is defined in terms of an Interoperable Object Refernce (IOR).

```
// Handle
value Handle {};
value StandardHandle : Handle {
    public string EJBObjectIOR;
};
```

3.1.2 Mapping of EJBMetaObject

The EJBMetaData is a value object. Its standard mapping is defined as follows:

```
// EJBMetaData
value EJBMetaData {};
value StandardEJBMetaData : EJBMetaData {
    public string  EJBHomeIOR;
    public string  homeClassName;
    public string  remoteClassName;
    public string  keyClassName;
    public boolean isSession;
};
```

3.1.3 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].

1.plain insecure IIOP, SECIOP, or IIOP over SSL

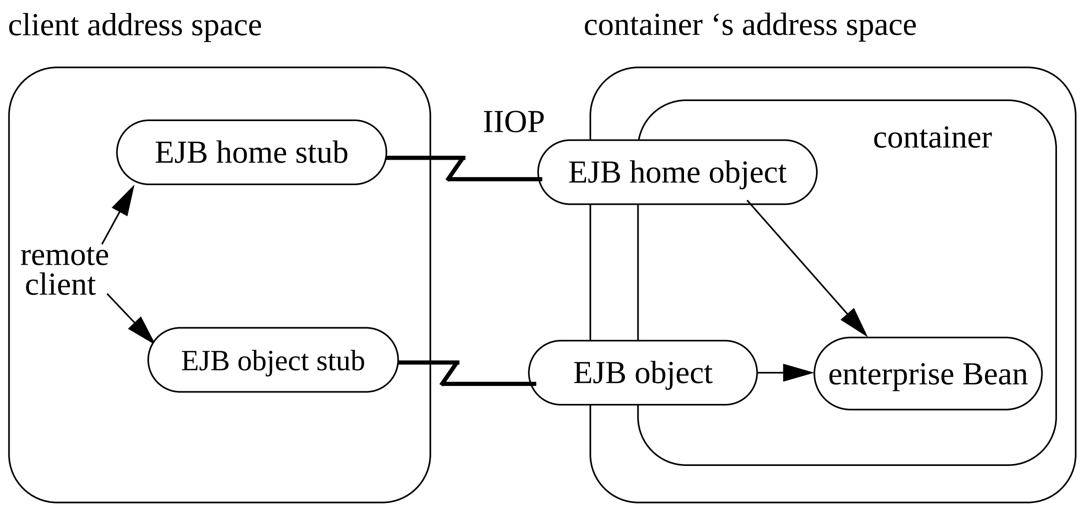
- For an entity enterprise bean, the IDL interface for the enterprise bean's home interface must inherit from the *CosTransactions::TransactionalObject* IDL interface.

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 20 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.



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

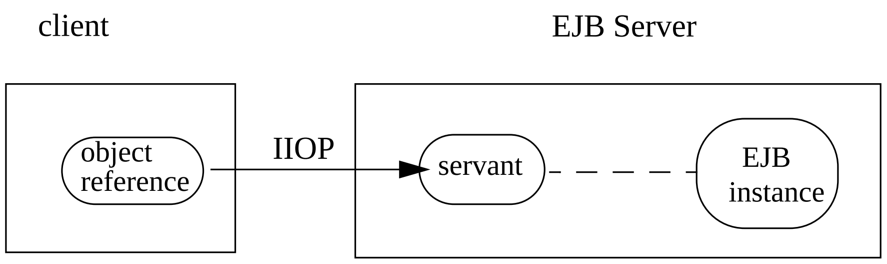
An example of the complete IDL for the remote interfaces in the Enterprise JavaBeans is listed in "Enterprise JavaBeans IDL" on page 17.

3.3 CORBA Object and Enterprise JavaBean 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

2.This is necessary to ensure the implicit propagation of the transaction context from the client to the enterprise bean.

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.



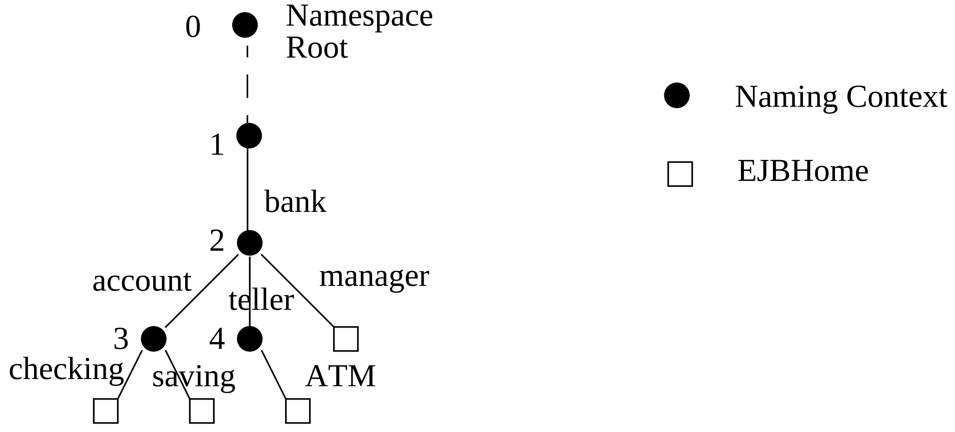
4 Mapping of Naming

A CORBA based EJB runtime is required to use the OMG COS NameService for publishing and resolving the EJBHome interface objects. The COS NameService can either be accessed directly using the COS Naming API or using the JNDI API with the standard COS Naming service provider [12].

4.1 COS Namespace Layout

Each bean's deployment descriptor contains a property called the `BeanHomeName` 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:



In the figure above, there are 5 naming contexts:

- naming context 0 is the root of the COS name space. It is obtained using ORB's resolve_initial_references method with "NameService" as the argument
- naming context 1 is where the ejb-jar is "installed". Note that we do not mandate any relationship between naming contexts 0 and 1. If they are not the same, then the client pretty much has to be configured with a path between them.
- naming context 2 is bound to context1 with name `bank`; naming contexts 3, and 4 are bound to context 2 with names `account` and `teller` respectively

The following rules should be used to construct a CosNaming::Name (to use in CosNaming APIs) from a container path.

- the container path is parsed from left to right
- each '/' separated name is the *id* field of a CosNaming::NameComponent, and

- the *kind* field of the NameComponent is always an empty string (““)

5 Mapping of Transactions

A CORBA based EJB runtime is required to use OMG Object Transaction Service (OTS) [6] for transaction support. The following sections describe the mapping of the transaction concepts in EJB to OMG (OTS).

5.1 Transaction Propagation

The rules for the mapping of enterprise bean's remote interfaces to CORBA IDL (See Subsection 3.1.3) 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
TX_NOT_SUPPORTED	-	-
	T1	suspend T1 invoke bean resume T1
TX_BEAN_MANAGED	-	resume bean's transaction invoke bean suspend bean's transaction
	T1	suspend T1 resume bean's transaction invoke bean suspend bean's transaction resume T1

TX_REQUIRED	-	begin T2
		invoke bean
		end[a] T2

	T1	inherits T1

Transaction Attribute	Client’s transaction	operation on TransactionCurrent on the Server
TX_SUPPORTS	-	-
	T1	inherits T1

TX_REQUIRES_NEW	-	begin T2
		invoke bean
		end T2

	T1	suspend T1 begin T2 invoke bean end T2 resume T1
TX_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):

- *TX_NOT_SUPPORTED*

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

- *TX_BEAN_MANAGED*

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

- *TX_REQUIRED*

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

- *TX_SUPPORTS*

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

- *TX_REQUIRES_NEW*

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

- *TX_MANDATORY*

```
if (Current.getStatus() == StatusActive) {
        return instance.method(args);
} else {
        throw org.omg.CORBA.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/97-03-08.pdf*)

[12] Java Naming and Directory Service Providers (*http://java.sun.com/products/jndi/serviceproviders.html*)

[13] CORBA Objects by Value (*http://www.omg.org/docs/orbos/98-01-18.pdf*)

Appendix B: Enterprise JavaBeans IDL

B.1 ejb.idl

```
// ejb.idl

#include “java.lang.ExceptionValue.idl”

module javax {

    module ejb {

        // CreateException
        value CreateException : ::java:lang:Exception {};
        exception CreateEx {
            CreateException the_value;
        };

        // DuplicateKeyException
        value DuplicateKeyException : ::java:lang:Exception {};
        exception DuplicateKeyEx {
            DuplicateKeyException the_value;
        };

        // EJBException
        value EJBException : ::java:lang:Exception {};
        exception EJBEx {
            EJBException the_value;
        };

        // FinderException
        value FinderException : ::java:lang:Exception {};
        exception FinderEx {
            FinderException the_value;
        };

        // ObjectNotFoundException
        value ObjectNotFoundException : ::java:lang:Exception {};
        exception ObjectNotFoundEx {
            ObjectNotFoundException the_value;
        };

        // RemoveException
        value RemoveException : ::java:lang:Exception {};
        exception RemoveEx {
            RemoveException the_value;
        };

        // Handle
        value Handle {};
        value StandardHandle : Handle {
            public string EJBObjectIOR;
        };
        // EJBMetaData
        value EJBMetaData {};
        value StandardEJBMetaData : EJBMetaData {
            public string  EJBHomeIOR;
            public string  homeClassName;
            public string  remoteClassName;
            public string  keyClassName;
            public boolean isSession;
        };

        interface EJBHome;

        // EJBObject
        interface EJBObject {

            EJBHome getEJBHome();

            Any getPrimaryKey();

            void remove() raises (RemoveEx);

            Handle getHandle();

            boolean isIdentical(in EJBObject object);
        };


        // EJBHome
        interface EJBHome {

            EJBMetaData getEJBMetaData();

            void remove(Any primaryKey) raises (RemoveEx);

            void remove(Handle handle) raises (RemoveEx);
        };
    };
};
```

B.2 java.lang.Exception.idl

```
// java.lang.Exception.idl

module java {
    module lang {
        value Exception {
            public ::CORBA::WStringValue detailMessage;
        };
    };
};
```

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 deployed as TX_REQUIRED.

C.1 Bean Home Interface

```
package trading;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;

/**
 * TraderHome is a remote interface for the Trader Home.
 */
public interface TraderHome extends EJBOHome {

    Trader create(String user)
        throws RemoteException, CreateException;
}
```

C.2 Bean Remote Interface

```
package trading;

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

/**
 * Trader is a remote interface for the Trader server bean.
 */
public interface Trader 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;
}
```

C.3 Bean’s Remote Exceptions

```
package trading;

/**
 * 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.4 Generated IDL

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

module trading {

    value TooManySharesException: ::java::lang::ExceptionValue {
        int maxAllowed;
    };

    exception TooManySharesEx {
        TooManySharesException the_value;
    };

    interface TraderHome : ::javax::ejb::EJBHome {
     Trader create(in string user)
         raises(::javax.ejb.CreateEx);
    };

    interface Trader : ::javax::ejb::EJBObject,
        ::org::omg::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);
    };
};
```

C.5 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 Home Interface
CosName home_name(...);

// locate the Trader Home
CORBA_Object_var obj = initial->resolve(home_name);
trading_TraderHome_ptr home = trading_TraderHome::_narrow(obj);

// create a session bean
trading_Trader_ptr trader = home->create(“user1”);

// 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);

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

...
```