OBJECTSPACE VOYAGER
CORE TECHNOLOGY

THE

AGENT ORB

FOR JAVA

Voyager CORBA Integration Technical Overview

This paper presents a technical overview of the ObjectSpace Voyager™ Core Technology (Voyager) CORBA integration. This integration is part of Voyager Version 2.0 Beta 1, released on December 1, 1997. Version 2.0 Beta 2 will be released in January 1998, and the 2.0 production version is scheduled for release in the first quarter of 1998.

This paper presents an overview of features in the CORBA technology integration and addresses the use of this integration through a series of code examples and accompanying descriptions. For more information about the CORBA integration, read Part 4 of the *ObjectSpace Voyager Core Technology User Guide*, Version 2.0 Beta 1.

OBJECTSPACE

© 1997 ObjectSpace, Inc. • All rights reserved • Version 2.0 Beta 1 • December 1997

Contents

Overview **1**

Voyager as a CORBA Client **2**
Creating a CORBA Server 3
Creating a Voyager Client 4
Running the CORBA Server 5
Running the Voyager Client 5

Voyager as a CORBA Server **6**
Creating a Voyager Server 7
Creating a CORBA Client 8
Running the Voyager Server 9
Running the CORBA Client 9

CORBA-Enabling a Java Class **10**
Creating a Voyager Server 11
Creating a CORBA Client 12
Running the Voyager Server 13
Running the CORBA Client 13

Advanced Services with Voyager and CORBA **14**
Creating a CORBA Server 14
Creating a Voyager Client 15
Running the CORBA Server 16
Running the Voyager Client 16

Overview

A major strength of the ObjectSpace Voyager™ Core Technology (Voyager) is its natural binding to the Java™ language. Any Java class can be remote-enabled without modification, and remote objects behave in a way that is intuitive to Java programmers. The CORBA Java binding defined by the OMG does not have these properties. The ObjectSpace goal is not to simply make Voyager a CORBA object request broker (ORB), for Voyager is more powerful and easier to use for distributed Java development than related CORBA technologies. It is important that Voyager developers be able to access both CORBA objects and DCOM objects in a simple and seamless manner while preserving the power, elegance, and ease of use found in the Voyager natural Java language binding. Due to its powerful Java-centric architecture, Voyager provides in many ways a better solution for CORBA-enabling a Java program than any other CORBA solution available today.

Highlights of the Voyager CORBA integration are described below:

- Voyager can take any IDL file and automatically create its equivalent Java interface and Voyager virtual reference class. It can also take any Java interface or `.class` file and automatically create its equivalent IDL file, allowing a Java class to be CORBA-enabled without modification.
- Voyager can obtain a virtual reference to a CORBA object, and CORBA can obtain a remote reference to a Voyager object.
- A Voyager developer can send messages to an object via a virtual reference without knowing whether the object is a CORBA object or a Voyager object. If the object is in a CORBA ORB, Voyager automatically uses IIOP to communicate with the CORBA object.
- Virtual references to Voyager objects are automatically converted to CORBA references when sent to a CORBA ORB, and CORBA references are automatically converted to virtual references when sent to Voyager.
- Voyager supports in, inout, and out parameters via the standard holder mechanism.
- Because virtual references hide the details of the underlying ORB, Voyager's advanced services, such as Space™, multicast messaging, futures, and dynamic invocation, can be used with CORBA objects, Voyager objects, or a mixture of both CORBA and Voyager objects.
- The CORBA integration adds only about 100KB to the total size of Voyager.

The rest of this paper describes the integration in more detail and provides examples. All code examples were prepared using the Visigenic's Visibroker for Java.

Voyager as a CORBA Client

This section demonstrates how a Voyager client can connect to a Bank object located in a CORBA server, create a remote Account, and then access the Account.

The IDL definitions for the interfaces used in this and later examples are shown below:

Interface Bank.idl

```
#include "Account.idl"

module finance
 {
 interface Bank
  {
  finance::Account openAccount();
  };
 };
```

Interface Account.idl

```
#include "OverdrawnException.idl"

module finance
 {
 interface Account
  {
  void deposit( in long amount );
  long getBalance();
  void withdraw( in long amount ) raises( finance::OverdrawnException );
  };
 };
```

Interface OverdrawnException.idl

```
module finance
 {
 exception OverdrawnException
  {
  string reason;
  };
 };
```

This examples assumes that a CORBA programmer creates implementations of the above IDL definitions and constructs an instance of Bank in a remote CORBA ORB. To connect to the Bank object, Voyager must obtain its standard object reference (also known as an IOR). This can be done in several ways, such as using a naming service; for simplicity, assume that the IOR is written to a file in a known location that the Voyager client can read.

Creating a CORBA Server

The following program creates a simple CORBA server that exports a Bank object by storing its IOR in a known location. Note that this server is in no way coupled to Voyager.

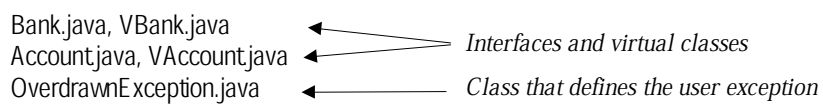
Class Server.java

```
import org.omg.CORBA.*;
import finance.*;
import java.io.*;

public class Server
 {
 static org.omg.CORBA.ORB orb;

 public static void main( String[] args )
  {
  try
   {
   //initialize the ORB and create a Bank server
   orb = org.omg.CORBA.ORB.init();
   org.omg.CORBA.BOA boa = orb.BOA_init();
   BankImpl bank = new BankImpl( "fort knox" );
   boa.obj_is_ready( bank );

   //write IOR to file for later use by Voyager client
   RandomAccessFile file = new RandomAccessFile( args[0], "rw" );
   file.writeUTF( orb.object_to_string( bank ) );
   file.close();
   System.out.println( "IOR is written to " + args[0] + ", ready" );

   boa.impl_is_ready();
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Before creating a Voyager client, process the IDL files using the vcc utility:

```
vcc Bank.idl Account.idl OverdrawnException.idl
```

The vcc utility automatically creates a Java interface and a functional virtual reference for each of the IDL interfaces found in the files, as well as a Java class for the exception.



The Voyager client reads the IOR of the CORBA Bank implementation from the file, converts it to a virtual reference using VObject.forObjectAt(), and then communicates with the CORBA object as if it were local. Note that when openAccount() returns a new Account,

the CORBA reference is automatically converted to a Voyager virtual reference and returned as a Java Account interface. When an exception occurs on the server side, it is automatically rethrown on the client side.

Creating a Voyager Client

The following program creates a Voyager client that retrieves the IOR exported by the previous Server program and uses regular Voyager lookup and messaging to access and modify the associated Bank and Account classes. Note that this client is in no way coupled to CORBA.

Class Client.java

```
COM.objectspace.voyager.*;
import finance.*;
import java.io.*;

public class Client
 {
 public static void main( String[] args )
  {
  try
   {
   //read Bank IOR from the file
   RandomAccessFile file = new RandomAccessFile( args[0], "r" );
   String ior = file.readUTF();
   file.close();
   System.out.println( "IOR read from file, client ready" );

   //create virtual reference to CORBA Bank and use it
   Bank bank = (Bank) VObject.forObjectAt( ior );
   Account account = bank.openAccount();
   account.deposit( 1000 );
   System.out.println( "balance = " + account.getBalance() );
   account.withdraw( 2000 );
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }

  Voyager.shutdown();
  }
 }
```

Running the CORBA Server

Run the Server.java program. The following output displays:

```
>java Server ..\..\IOR.dat
IOR is writterto ..\..\IOR.dat, server is ready
>_
```

Running the Voyager Client

Now run the Client.java program.

```
>java Client ..\..\IOR.dat
IOR read from file, client ready
voyager(tm) 2.0 beta 1, copyright objectspace 1997
address = 208.6.239.137:1251, root = \
balance = 1000
finance.OverdrawnException
>_
```

Voyager as a CORBA Server

This section demonstrates how a CORBA client can connect to a Bank object located in a Voyager server, create a remote Account, and then access the Account. It is a mirror image of the previous example.

The definitions for the Java Bank and Account interfaces and the OverdrawnException class used in this example are listed below:

Interface finance\Bank.java

```
package finance;

public interface Bank
 {
 public Account openAccount();
 }
```

Interface finance\Account.java

```
package finance;

public interface Account
 {
 public void deposit( int amount );
 public int getBalance();
 public void withdraw( int amount ) throws OverdrawnException;
 }
```

Class finance\OverdrawnException.java

```
package finance;

public class OverdrawnException extends java.lang.Exception
 {
 public java.lang.String reason;

 public OverdrawnException()
  {
  }

 public OverdrawnException( java.lang.String reason )
  {
  this.reason = reason;
  }
 }
```

This examples assumes that a Voyager programmer creates concrete implementations of the above interfaces. BankImpl is the implementation of the Bank interface in the following

example. Using vcc as described in the *ObjectSpace Voyager Core Technology User Guide,* the programmer creates a virtual reference to the class, in this case VBankImpl. This virtual reference allows the programmer to very simply create and register the Bank interface server in the following example.

To connect to the Bank server, CORBA must obtain its IOR. This can be done in several ways; for simplicity, assume that the IOR is written to a file in a known location the CORBA client can read.

Creating a Voyager Server

The following program creates a Voyager server that exports a Voyager Bank object by writing its IOR to a known location. Note that except for the required step of retrieving the object's IOR, this code is in no way coupled to CORBA.

Class Server.java

```
import COM.objectspace.voyager.*;
import finance.*;
import java.io.*;

public class Server
 {
 public static void main( String[] args )
  {
  try
   {
   //create the Bank server
   VBankImpl bank = new VBankImpl( "localhost" );
   bank.liveForever();

   //write CORBA IOR to a well-known file for use by a CORBA client
   RandomAccessFile file = new RandomAccessFile( args[0], "rw" );
   file.writeUTF( bank.toCORBA() );
   file.close();
   System.out.println( "IOR is written to " + args[0] + ", ready" );
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Before creating a CORBA client, process the .java files using the vcc utility with the –I (uppercase I) option:

```
vcc -I Bank.java Account.java OverdrawnException.java
```

This command automatically generates the following IDL files for use by the CORBA client:

```
Bank.idl
Account.idl
OverdrawnException.idl
```

The CORBA client then reads the IOR of the Voyager Bank server from the file, converts it to an object reference using string_to_object(), and then communicates with the Voyager object as if it were local. Note that when openAccount() returns a new Account, the implementation of the Account exists in the Voyager server and is returned to the client as a CORBA object reference for the Account. When an exception occurs on the server side, it is automatically rethrown on the client side.

Creating a CORBA Client

The following program creates a CORBA client that accesses the Bank exposed in the previous Voyager Server by reading its associated IOR from the known location. Note that this code is in no way coupled to Voyager.

Class Client.java

```
import org.omg.CORBA.*;
import finance.*;
import java.io.*;

public class Client
 {
 static org.omg.CORBA.ORB orb;

 public static void main( String[] args )
  {
  try
   {
   //initialize the ORB
   orb = org.omg.CORBA.ORB.init();
   org.omg.CORBA.BOA boa = orb.BOA_init();

   //read the CORBA IOR for the Bank from the file
   RandomAccessFile file = new RandomAccessFile( args[0], "r" );
   String ior = file.readUTF();
   file.close();
   System.out.println( "IOR read from file, client ready" );

   //create a CORBA client using the IOR and use it
   org.omg.CORBA.Object object = orb.string_to_object( ior );
   Bank bank = BankHelper.narrow( object );
   Account account = bank.openAccount();
   account.deposit( 1000 );
   System.out.println( "balance = " + account.getBalance() );
   account.withdraw( 2000 );
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Running the Voyager Server

Run the Server.java program. The following output displays:

```
>java Server ..\..\IOR.dat
voyager(tm) 2.0 beta 1, copyright objectspace 1997
address = 208.6.239.137:1253, root = \
IOR is writterto ...\\IOR.dat, server is ready
>_
```

Running the CORBA Client

Now run the Client.java program.

```
>java Client..\..\IOR.dat
IOR read from file, client ready
balance = 1000
exception OverdrawnException{string reason="not enough money
    at finance.AccountImpl.withdraw(AccountImpl.java:20)
    thrown remotely from 208.6.239.137:1253";}
>_
```

CORBA-Enabling a Java Class

This section shows how a Java class can be CORBA-enabled without modification of source code. This is a powerful feature of the Voyager CORBA integration and is a feat unmatched by any other Java CORBA ORB. Using this feature, programmers can easily CORBA-enable third-party classes. This section also shows how Voyager virtual references can be used to hide the kind of ORB that holds the destination object, providing dynamic protocol selection and seamless language binding for developing applications using a mixture of ORB technology.

In the following example, a Voyager server constructs a local Customer object and a local Bank object. IORs to both objects are written to a file that is accessible to the CORBA client. The CORBA client performs these steps:

1. Connects to the Voyager Customer, the Voyager Bank, and a local CORBA Bank.
2. Opens an account in both banks.
3. Adds the accounts to the Voyager Customer.
4. Asks the Voyager Customer to calculate its total balance, which causes the Customer to ask the Voyager Account and the CORBA Account for their respective balances.

The Java code for the Customer class is listed below. Note that it contains no code specific to Voyager or CORBA.

Class finance\Customer.java

```
package finance;

import java.util.Vector;

public class Customer
 {
 Vector accounts = new Vector();

 public void addAccount( Account account )
  {
  accounts.addElement( account );
  }

 public int getTotal()
  {
  int total = 0;

  for( int i = 0; i < accounts.size(); i++ )
    total += ( (Account) accounts.elementAt( i ) ).getBalance();

  return total;
  }
 }
```

Use the vcc utility with the -L option to CORBA-enable the Customer class.

```
vcc -L Customer
```

This creates both a CORBA Customer.idl file and a Voyager virtual reference class VCustomer for the Customer. The original Java class does not require any modification.

Creating a Voyager Server

The program below creates a Voyager server that exports a Customer object by writing its autogenerated IOR to a known location.

Class Server.java

```
import COM.objectspace.voyager.*;
import finance.*;
import java.io.*;

public class Server
 {
 public static void main( String[] args )
  {
  try
   {
   //create the Voyager Bank server and a Customer
   VBankImpl bank = new VBankImpl( "localhost" );
   bank.liveForever();
   VCustomer customer = new VCustomer( "localhost" );
   customer.liveForever();

   //write CORBA IORs for the Bank and Customer to a well-known file
   RandomAccessFile file = new RandomAccessFile( args[0], "rw" );
   file.writeUTF( bank.toCORBA() );
   file.writeUTF( customer.toCORBA() );
   file.close();
   System.out.println( "IORs are written to " + args[0] + ", ready" );
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Creating a CORBA Client

The program below creates a Voyager client that accesses the exported Bank and Customer objects. Note that the CORBA code can treat the Customer object as if it were custom-written for CORBA.

Class Client.java

```
import org.omg.CORBA.*;
import finance.*;
import java.io.*;

public class Client
 {
 static org.omg.CORBA.ORB orb;

 public static void main( String[] args )
  {
  try
   {
   //initialize the ORB
   orb = org.omg.CORBA.ORB.init();
   org.omg.CORBA.BOA boa = orb.BOA_init();

   //read the IORs from the file
   RandomAccessFile file = new RandomAccessFile( args[0], "r" );
   String bankIOR = file.readUTF();
   String customerIOR = file.readUTF();
   file.close();
   System.out.println( "IORs read from file, client ready" );

   //create CORBA client interfaces to the Voyager Bank and Customer
   Bank bank1 = BankHelper.narrow( orb.string_to_object( bankIOR ) );
   Customer customer =
    CustomerHelper.narrow( orb.string_to_object( customerIOR ) );

   //create a local CORBA Bank implementation
   BankImpl bank2 = new BankImpl( "fort knox" );
   boa.obj_is_ready( bank2 );

   //create accounts, one in Voyager, one in CORBA, sum across them both
   Account account1 = bank1.openAccount();
   account1.deposit( 1000 );
   Account account2 = bank2.openAccount();
   account2.deposit( 2000 );
   customer.addAccount( account1 );
   customer.addAccount( account2 );
   System.out.println( "customer total = " + customer.getTotal() );
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Running the Voyager Server

Run the Server.java program. The following output displays:

```
>java Server ..\..\IOR.dat
voyager(tm) 2.0 beta 1, copyright objectspace 1997
address = 208.6.239.137:1267, root = \
IORs are writterto ...\\IOR.dat, server is ready
>_
```

Running the CORBA Client

Now run Client.java program.

```
>java Client..\..\IOR.dat
IORs read from file, client ready
customer total = 3000
>_
```

Advanced Services with Voyager and CORBA

This section shows how Voyager's advanced services can be used to communicate with both CORBA objects and Voyager objects.

Voyager's Space, multicast messaging, futures, and dynamic invocation features all operate on virtual references and are independent of the ORB that contains the target object. The following code demonstrates the use of the Voyager advanced services on a mixture of CORBA and Voyager objects. For additional information on advanced Voyager features, refer to the *ObjectSpace Voyager Core Technology User Guide*, available from the ObjectSpace Web site at www.objectspace.com.

Creating a CORBA Server

The following program creates a CORBA server that exports a Bank object as a CORBA object.

Class Server.java

```
import org.omg.CORBA.*;
import finance.*;
import java.io.*;

public class Server
 {
 static org.omg.CORBA.ORB orb;

 public static void main( String[] args )
  {
  try
   {
   //initialize the ORB and create a CORBA Bank server
   orb = org.omg.CORBA.ORB.init();
   org.omg.CORBA.BOA boa = orb.BOA_init();
   BankImpl bank = new BankImpl( "fort knox" );
   boa.obj_is_ready( bank );

   //write CORBA IOR of Bank server to a file for use by a Voyager client
   RandomAccessFile file = new RandomAccessFile( args[0], "rw" );
   file.writeUTF( orb.object_to_string( bank ) );
   file.close();
   System.out.println( "IOR is written to " + args[0] + ", ready" );
   boa.impl_is_ready();
   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

Creating a Voyager Client

The following program creates a Voyager client that multicasts to a mixture of Voyager and CORBA objects simultaneously. Note that no changes to the Bank or Server classes are required to support this powerful Voyager feature.

Class Client.java

```
import COM.objectspace.voyager.*;
import COM.objectspace.voyager.space.*;
import finance.*;
import java.io.*;

public class Client
  {
  public static void main( String[] args )
    {
    try
      {
      //read the IOR for the CORBA Bank server
      RandomAccessFile file = new RandomAccessFile( args[0], "r" );
      String ior = file.readUTF();
      file.close();
      System.out.println( "IOR read from file, client ready" );

      //connect to CORBA bank, open account
      Bank bank1 = (Bank) VObject.forObjectAt( ior );
      Account account1 = bank1.openAccount();

      //create Voyager bank, open account
      Bank bank2 = new VBankImpl( "localhost" );
      Account account2 = bank2.openAccount();

      //Use Space to multicast to Voyager and CORBA accounts
      VSubspace subspace = new VSubspace( "localhost" );
      subspace.add( (VObject) account1 );
      subspace.add( (VObject) account2 );
      VAccount accounts = new VAccount( subspace );
      accounts.deposit( 1000 ); //Multicast to both accounts.
      System.out.println( "sleep for 3 seconds..." );
      try{ Thread.sleep( 3000 ); } catch( Exception e ) {}
      System.out.println( "account1 = " + account1.getBalance() );
      System.out.println( "account2 = " + account2.getBalance() );

      //demonstrate use of Future to CORBA object
      Result result = ( (VAccount) account1 ).getBalance( new Future() );
      System.out.println( "account1 = " + result.readInt() );

      //demonstrate use of dynamic invocation on CORBA object
      Sync sync = new Sync();
      sync.setSignature( "deposit(I)" );
      sync.writeInt( 3000 );
      System.out.println( "use DII to deposit $3000 into account1" );
      ((VObject) account1).send( sync );
      System.out.println( "account1 = " + account1.getBalance() );
```

```
    }
   catch( Exception exception )
    {
    System.err.println( exception );
    }

   Voyager.shutdown();
  }
 }
```

Running the CORBA Server

Run the Server.java program. The following output displays.

```
>java Server ..\..\IOR.dat
IOR is written to ..\..\IOR.dat, server is ready
>_
```

Running the Voyager Client

Now run Client.java.

```
>java Client ..\..\IOR.dat
IOR read from file, client ready
type = finance.Bank
voyager(tm) 2.0 beta 1, copyright objectspace 1997
address = 208.6.239.137:1275, root = \
sleep for 3 seconds...
account1 = 1000
account2 = 1000
account1 = 1000
use DII to deposit $3000 into account1
account1 = 4000
>_
```

For additional technical information on ObjectSpace products and programs or for information on how to order and evaluate ObjectSpace technology, contact us today!

14850 Quorum Drive, Suite 500
Dallas TX 75240

972.726.4100
1.800.OBJECT.1
Fax: 972.715.9099

E-mail: sales@objectspace.com
Web: www.objectspace.com

Dallas • Austin • Chicago •
San Francisco • Washington DC

Java is a trademark of Sun Microsystems.
ObjectSpace Voyager and Space are trademarks of ObjectSpace, Inc.
All other trademarks are the property of their respective companies.