CORBA
Common Object Request Broker Architecture Part II

Based partially on Java Network Programming and Distributed Computing Chapter 12
Also on Sun's Java Online Documentation
And OMG's Technical Documentation at www.omg.org

Topics

- Writing your own CORBA service in Java
  - The AddressBook example revisited.
- Java packages supporting CORBA.
- CORBA Services
- CORBA's Event Service.
- CORBA Deployment Issues

A Running Example: An Address Book Service

- Recording and looking up people by their names and emails.
- Language-neutral IDL interface definition
  - An `address_book_system` module with a single `address_book` interface.
- Servant, server, and client implementations in Java, using and extending code generated by `idlj -fall foo.idl` command.
- Example taken from:

  http://www.javacoffeebreak.com/articles/javaidl/javaidl.html

Writing your Own CORBA Service in Java

- Create IDL schema containing modules and interfaces
- Map your IDL to Java –generate code for stubs and skeletons.
- Write the Servant code – implement operations in `FooOperations.java` extending `_FooImplBase.java`.
- Write the Server code – the servant can also include the Server code as the `main` method.
- Write the Client code

Writing the IDL schema

- You need to define
  - Modules
  - Interfaces
  - Exceptions
- Do not expose implementation details.
- Analogous to designing a network protocol.
- Accommodate different client and server implementations.
  - Different teams/programming languages/O.Ss.
  - Make your code extensible.

Mapping your IDL schema to Java

- Use `idlj` tool included in the JDK (in Java 2 SDK v1.3 or higher, otherwise download `idl2java` tool separately).
- Command-line options include:
  - `-fall` --generate code for client and server
  - `-fclient` --generate code for client only
  - `-fserver` –generate code for server only.

Writing the Servant code

- This is your service implementation
- Implement operations in generated `FooOperations.java` extending generated `_FooImplBase.java`.

Writing the Server code

- Import CORBA related packages:

```
org.omg.CORBA.*
org.omg.CosNaming.*
org.omg.CosNaming.NamingContextPackage.*
```

- Create an ORB
  - `ORB.init`
- Create a servant
- Connect it to the ORB
- Export the ORB to a name service
- Wait for requests and process them

Writing the Client code

- Import CORBA related packages:
  `org.omg.CORBA.*`
  `org.omg.CosNaming.*`
- Create an ORB
  - `ORB.init`
- Get a reference to the name service
  - Similar to server code.
- Look up a servant
- Narrow it to your interface type
- Use the remote object!

Like a cast operation!

org.omg.CORBA

- ORB Class
  - A fully functional Object Request Broker (ORB).
- Exceptions
- Holder classes
- Helper classes

org.omg.CORBA.ORB Class

- When an application requires a CORBA environment it needs a mechanism to get an ORB object reference and possibly an OA object reference (such as the root POA).
- This serves two purposes.
  - It initializes an application into the ORB and OA environments.
  - It returns the ORB object reference and the OA object reference to the application for use in future ORB and OA operations.

org.omg.CORBA.ORB Class

These are the ORB methods that provide access to the ORB:

- init()
- init(String [] args, Properties props)
- init(Applet app, Properties props)

- Using the `init()` method without parameters initiates a singleton ORB, which can only give typecode creation *anys* needed in code generated in Helper classes by `idlj`.

org.omg.CORBA.ORB Class

Applications require a portable means by which to obtain their initial object references.

References are required for the root POA, POA Current, Interface Repository, and various Object Services instances.

org.omg.CORBA.ORB Class

Initial references are obtained via:

- resolve_initial_references(String name)
- list_initial_services()
- register_initial_reference(String id, org.omg.CORBA.Object obj)

OMG does not want to mandate that the Naming Service be made available to all applications in order that they may be portably initialized. Consequently, the simplest naming context is a flattened naming space.

omg.org.CORBA Exceptions

- System exceptions –raised by the ORB libraries to signal systemic error conditions like:
  - Server-side system exceptions, such as resource exhaustion or activation failure.
  - Communication system exceptions, for example, losing contact with the object, host down, or cannot talk to ORB daemon (orbd).
  - Client-side system exceptions, such as invalid operand type or anything that occurs before a request is sent or after the result comes back.
- User-defined exceptions
  - Defined in IDL interfaces by `raises` keyword.

org.omg.CORBA System Exceptions

Unchecked system exceptions inherit from `org.omg.CORBA.SystemException` – extends `java.lang.RuntimeException`:

BAD_CONTEXT BAD_INV_ORDER BAD_OPERATION BAD_PARAM BAD_TYPECODE COMM_FAILURE DATA_CONVERSION FREE_MEM IMP_LIMIT INITIALIZE INTERNAL INTF_REPOS INVALID_TRANSACTION INV_FLAG INV_IDENT INV_OBJREF INV_POLICY MARSHAL NO_IMPLEMENT NO_MEMORY NO_PERMISSION NO_RESOURCES NO_RESPONSE OBJECT_NOT_EXIST OBJ_ADAPTER PERSIST_STORE TRANSACTION_REQUIRED TRANSACTION_ROLLEDBACK TRANSIENT UNKNOWN

org.omg.CORBA User Exceptions

- Inherit from `org.omg.CORBA.UserException` –which extends `java.lang.Exception`.
- As opposed to system exceptions, the compiler checks that programmer code deals with these exceptions:
  - By appropriate `try/catch` blocks, or
  - By declaring that a method `throws` such exceptions.

org.omg.CORBA Holder Classes

- Support for out and inout parameter passing modes requires the use of additional *holder classes*, because Java does not support out or inout parameters.
- Holder classes are needed as a means of passing a parameter that can be modified.
- To support portable stubs and skeletons, holder classes also implement the org.omg.CORBA.portable.Streamable interface.

org.omg.CORBA Holder Classes

- Holder classes are named by appending "Holder" to the name of the type.
  - E.g., a holder class for the interface named Account in Java would be named AccountHolder.
- Holder classes are available for all of the basic IDL datatypes in the org.omg.CORBA package.
  - E.g., there are already-defined classes for LongHolder, ShortHolder, FloatHolder, and so on.

org.omg.CORBA Holder Classes

- Each holder class has:
  - a constructor from an instance
  - a default constructor
  - a public instance member, value which is the typed value.
  - a method for reading an input stream and assigning the contents to the type's value field
  - a method for writing the value of the value field to an output stream
  - a method for getting the typecode of the type

org.omg.CORBA Helper Classes

- Helper files supply several static methods needed to manipulate the type. These include:
  - *Any* insert and extract operations for the type
  - getting the repository id
  - getting the typecode
  - reading and writing the type from and to a stream
  - implement the ValueHelper interface (if it is a user-defined value type)

org.omg.CORBA Helper Classes

- The helper class for a mapped IDL interface or abstract interface also includes narrow operation(s).
- The static narrow method allows an `org.omg.CORBA.Object` to be narrowed to the object reference of a more specific type.
- The IDL exception `CORBA.BAD_PARAM` is thrown if the narrow fails because the object reference does not support the requested type.

org.omg.CORBA Helper Classes

- Helper classes fall into two broad categories:
  - helpers for value types –can be serialized as parameters to remote methods, or return values.
  - helpers for non value types.
- When OMG IDL is mapped to the Java programming language, a "helper" class is generated for each user-defined type.
  - This generated class will have the name of the user-defined type with the suffix Helper appended.

org.omg.CORBA Behind-the-Scenes

- Classes Created by an ORB
  - Contains classes that are created by an ORB and contain information used in request operations.
- Classes That Deal with Requests
- Interfaces That Serve as Constants
  - The IDL-to-Java mapping mandates that IDL enums are mapped to a Java class with the enumerated values represented as public static final fields.
- Interface Repository Interfaces and Classes
  - The purpose of the Interface Repository is to identify the interfaces stored in it so that they can be accessed by an ORB.

org.omg.CORBA Classes Created by an ORB

- TCKind -- indicates the kind (datatype) for a TypeCode object
- TypeCode -- indicates a datatype and possibly other information
- Any -- contains a value and its typecode
- NamedValue -- contains a name, an Any object, and an argument mode flag. NamedValue objects contain information about method arguments, method return values, or a context.

org.omg.CORBA **Classes Created by an ORB**

- ContextList -- a list of strings that describe the contexts that need to be resolved and sent with an invocation
- ExceptionList -- a list of TypeCodes for exceptions that may be thrown by a method
- Environment -- a container for the exception thrown during a method invocation
- Context -- a list of NamedValue objects used to pass auxiliary information from client to server
- NVList -- a list of NamedValue objects, used to pass arguments or get results

org.omg.CORBA **Classes That Deal with Requests**

- Object –
  - the base class for all CORBA object references
- Request –
  - the main class in the DII, which contains methods for adding arguments to the request, for accessing information about the method being invoked (the method name, its arguments, exceptions it throws, and so on), and for making invocations on the request

org.omg.CORBA **Classes That Deal with Requests**

- DynamicImplementation –
  - the base class for server implementations using the DSI. It has the method invoke, which is used by an implementation of this class to determine the state of a ServerRequest object and to set its result or exception
- ServerRequest –
  - captures the explicit state of a request for the Dynamic Skeleton Interface

org.omg.CORBA **Interfaces That Serve as Constants**

- IDL constants defined outside of an IDL interface are mapped to a Java interface for each constant.
  - This is why several interfaces in the org.omg.CORBA package consist of a single field, value, which is a short.
  - This field is a constant used for such things as an error code or value modifier.
    - For example, the value field of the interface BAD_POLICY is one of the possible reasons for the exception PolicyError to be thrown. To specify this error code, you would use BAD_POLICY.value.

org.omg.CORBA **Interface Repository**

- The purpose is to identify the interfaces stored in the repository so that they can be accessed by an ORB.
  - Generated by the `idlj` compiler from the OMG IDL interface `ir.idl`.
- Each module, type, interface, attribute, operation, parameter, exception, constant, etc, described completely by the Interface Repository API.
- An ORB does not require that there be an interface repository, and Java IDL does not include one.
  - Java 2 does not include an implementation of an interface repository.

CORBA Services

- Services are similar to high-level libraries that support specific functionality.
- Especially created for management of distributed and remote objects.
- Attempt to standardize “common” object services APIs for interoperability.

CORBA Service Design Principles

- Designed to do one thing well
  - therefore, they are only as complicated as they need to be.

But specifications are hundreds of pages long!

- Designed to be composable
  - can be combined together to provide for more interesting and powerful functionality.
- Generic
  - services do not depend on the type of the client object, or on the type of data passed in requests.

CORBA Service Design Principles

- Local or Remote implementations
  - structured as CORBA objects with OMG IDL interfaces for local or remote access.
- Different Quality of Service hidden in implementations
  - interface is the same. (e.g. fast, unreliable event delivery; or slower guaranteed delivery).
- Use of callback interfaces.

CORBA Service Design Principles

- No Global Identifier Space
  - rather names are unique within "contexts".
- Finding a Service is Orthogonal to Using it.
  - Enables complex object resolution strategies
  - E.g., to support load balancing in a heavily used web service.

What CORBA Services are There?

- Collection Service
- Concurrency Service
- Enhanced View of Time
- Event Service
- Externalization Service
- Naming Service
- Licensing Service
- Life Cycle Service
- Notification Service

What CORBA Services are There? The list continues…

- Persistent State Service
- Property Service
- Query Service
- Relationship Service
- Security Service
- Time Service
- Trading Object Service
- Transaction Service

Popular CORBA Services Description

- Object life cycle
  - Defines how CORBA objects are created, removed, moved, and copied
- Naming
  - Defines how CORBA objects can have friendly symbolic names
- Events
  - Decouples the communication between distributed objects

Popular CORBA Services Description

- Relationships
  - Provides arbitrary typed n-ary relationships between CORBA objects
- Externalization
  - Coordinates the transformation of CORBA objects to and from external media
- Transactions
  - Coordinates atomic access to CORBA objects

Popular CORBA Services Description

- Concurrency Control
  - Provides a locking service for CORBA objects in order to ensure serializable access
- Property
  - Association of name-value pairs with CORBA objects
- Trader
  - Finding of CORBA objects based on properties describing the service offered by the object
- Query –supports queries on objects.

The CORBA Events Service

- See tutorial at:

  http://www.cs.wustl.edu/~schmidt/PDF/coss4.pdf

CORBA Deployment Issues

- CORBA is a specification; it is a guide for implementing products. Several vendors provide CORBA products for various programming languages.

CORBA Products that Support Java

- **The Java 2 ORB**
  - The Java 2 ORB comes with Sun's Java 2 SDK. It is missing several features.
- **VisiBroker for Java**
  - A popular Java ORB from Inprise Corporation.
  - VisiBroker is also embedded in other products. For example, it is the ORB that is embedded in the Netscape Communicator browser.

CORBA Products that Support Java

- **OrbixWeb**
  - Popular Java ORB from IonaTechnologies.
- **WebSphere**
  - Popular application server with an ORB from IBM.
- **Netscape Communicator**
  - Netscape browsers have a version of VisiBroker embedded in them.
  - Applets can issue request on CORBA objects without downloading ORB classes into the browser. They are already there.

CORBA Products that Support Java

- **Various free or shareware ORBs**
  - CORBA implementations for various languages are available for download on the web from various sources.

ORB-Dependent Implementations

- Different ORB products have differences in the way they initialize an ORB, a BOA, a POA.
- This makes service implementations **not** portable across multiple ORB products.

You need to write some server code specifically for a given ORB implementation.

Not all ORBs implement all services. Not always in the same way…