

CORBA

R. Sean Lindsay
Senior Software Engineer
Mercator Software

Contributors
Matt Mick, Information Technology, Boston University
Bart Hanlon, VP of Engineering, Mercator Software

Overview

- Introduction
- The Object Management Group (OMG)
- Common Object Request Broker Architecture (CORBA)
- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Object Adapters (BOA/POA)
- CORBA versus RMI
- Summary

Early Distributed Computing

- Language specific remote procedure calls
 - Tightly coupled to protocol
 - Tightly coupled to language semantics
 - Often highly proprietary
- Homogenous environment
- Typically one to one cardinality
 - Not highly scalable

Shortcomings

- Proprietary
- Tightly coupled to language
- Lack extensibility
- Lack scalability
- Lack transparency

Distributed Computing Evolution

- Object-Oriented design and reuse
 - OO thriving in non-distributed programming
- Multi-tiered architectures
- Legacy application extension
- Enterprise level application integration
- TCP/IP
- Internet computing

Object Management Group

Object Management Group, est. 1989

- Initially 8 companies, over 750 today
- One of the largest industry consortiums
- Does not develop implementations
- Formal process for defining specifications
 - Members submit proposals, develop implementations
- Emphasizing cooperation and compromise
 - Most specs are an amalgamation of ideas
- Non-profit

OMG Goals

“...promote the theory and practice of object-oriented technology in software development...”

“...promote the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments...”

Object Management Architecture

- Framework within which all OMG adopted technology fits
- Two fundamental models on which CORBA is based

“...to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based...”

OMA Models

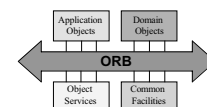
- Core Object Model
 - Abstract definition
 - Details how Object Request Broker (ORB) facilitates distributed application development
- Reference Object Model
 - Architectural framework
 - ORB at center, interface definitions
 - Framework for future technology adoption

OMA Core Object Model

- Design portability
 - Interface based interaction
 - Access does not rely on location or language of implementation
- Interoperability (I14Y)
 - Ability to invoke operations regardless of location, platform, or language of implementation

OMA Reference Model

- Defines interfaces to infrastructure and object services
- Guide for developers and vendors
- Defines five main components
 - Object Request Broker
 - Object Services
 - Common Facilities
 - Domain Interfaces
 - Application Interfaces



Common Object Request Broker Architecture

What is CORBA?

- Architecture for interoperable distributed computing
 - Based on the OMG's Object Management Architecture
- Internet Interoperability Protocol (IIOP)
- Language mappings (OMG IDL)
- Integrated and reusable services

CORBA Timeline

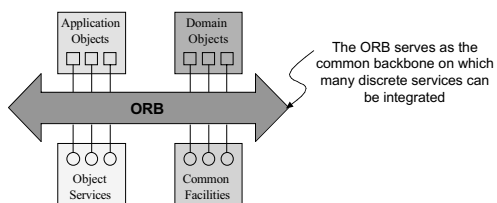
- Object Management Group founded 1989
- CORBA 1.x (91-93)
 - Experimental, architecture focused, IDL
- CORBA 2.0 (8/96)
 - Interoperability and COM integration
- CORBA 2.1 (8/97)
 - Security, language mappings

CORBA Timeline

- CORBA 2.2 (2/98)
 - DCOM interoperability, POA, IDL/Java
- CORBA 2.3 (1999)
 - Objects by value, RMI/IIOP, language binding enhancements
- CORBA 3.0 (?)
 - Multiple interface support, component model, scripting support

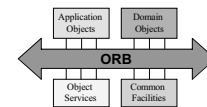
CORBA Architecture

- Based on OMA Reference Model



CORBA Model Components

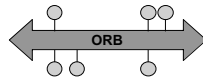
- Object Request broker
- Object Services
- Common Facilities
- Domain Interfaces
- Application Interfaces



Object Request Broker (ORB)

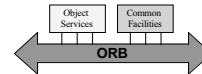
- Enables object to transparently make and receive requests and responses

"...like a telephone exchange, providing the basic mechanism for making and receiving calls..."



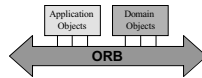
Object Services & Facilities

- Collection of interfaces and objects
- Services
 - Support functions for implementing and using objects
 - i.e. life cycle
- Facilities
 - Services that many applications may share, but are not fundamental
 - i.e. system management



Domain & Application Objects

- Domain Interfaces
 - OMG Domain Special Interest Groups (SIGs)
 - Market vertical specific services
- Application Objects
 - Vendor provided or custom object implementation
 - Top layer in Reference Model, not standardized by OMG



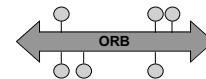
Internet Inter ORB Protocol (IIOP)

- Transport protocol
 - Defines inter ORB communication
 - Runs on top of TCP/IP
 - Defines CORBA messages
- IIOP is a specification
 - Vendors must implement to be "CORBA-compliant"
 - Allows for multi-vendor interoperability

Object Request Broker

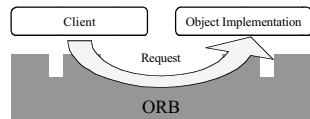
What is an ORB?

- A distributed object *bus*
- Hides transport mechanisms
 - Location
 - Method invocation
 - Marshalling
- OMG Interface Definition Language (IDL) provides the language independent semantics



Object Request Broker (ORB)

- Abstracts remote request and response mechanisms
- Transport for *distributing* method invocations



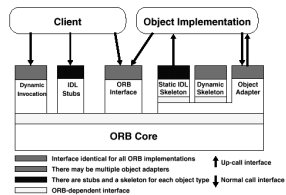
ORB Usage

- ORB is a singleton
- ORB initialization
 - Single static call to `init()` an ORB

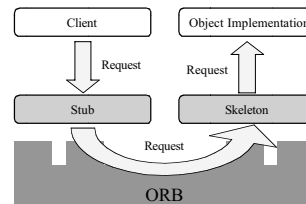
```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
```

- After initialization, you register objects with the ORB using an Object Adapter

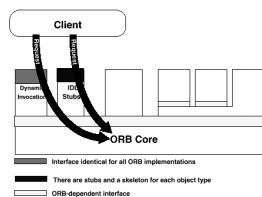
ORB Request Interfaces



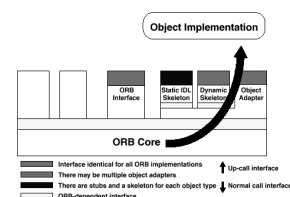
Proxy-based Invocation



Client Requesting



Server Receiving



Static Invocation

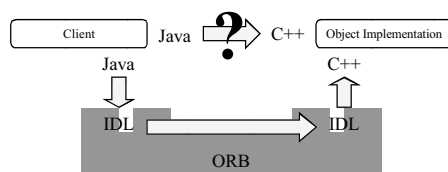
- Proxy objects generated to support distributed invocation
 - Interface defined using IDL
 - Stub and Skeleton classes
 - Language specific
 - Network and marshalling support
- Interface defined statically at compile time
 - Used when client is built

Dynamic Invocation Interface

- Alternative to static Stub/Skeleton calls
 - Don't need Stubs when client is built
- Structure a generic invocation structure and submit to DII
- Asynchronous (*deferred synchronous*) calls
- Slower than static but more flexible
- Similar to Java Reflection

ORB Abstraction

- How is this possible in a heterogeneous environment?



Interface Definition Language

Interface Definition Language (IDL)

- Specification language
- Language independent interface
 - Declare interfaces to object methods
 - IDL maps to many high-level programming languages
- Design paradigm
 - Code to interface specified in the IDL regardless of implementation

OMG Language Mappings

- Mapping IDL to programming language
 - Many OMG standard mappings
 - C
 - C++
 - Smalltalk
 - Ada '95
 - COBOL
 - Java

Key IDL Language Elements

- Module
- Interface
- Attribute
- Operation
- Argument
- Exception
- Struct
- Typedef
- Sequence
- Any

Sample IDL Definition

```
// Quote system module
module QuoteSystem ← specifies the scope/package QuoteSystem
{
  // Specify a data structure for quote
  struct Quote
  {
    string value;
  } ← define the struct for a Quote value

  // Specify interface to quote server
  interface QuoteServer
  {
    // Specify an stock exchange name attribute
    string exchange; ← data member

    // Unknown symbol exception
    exception UnknownSymbolException { string message; };

    // Lookup symbol
    Quote getQuote (in string symbol)
    raises (UnknownSymbolException);
  };
};
```

↑ return type

↑ argument (direction and type)

↑ declares that method throws an exception

defines a QuoteServer object's interface

Modules & Interfaces

- Module
 - Maps to a **package** in Java
 - Name space scoping
 - Module can contain multiple interfaces
- Interface
 - Maps to a set of related classes & interfaces

```
module QuoteSystem
{
  interface QuoteServer
  {
    ...
  };
};
```

↓

QuoteSystem.QuoteServer

Struct

- Structure
 - Maps to a class in Java
 - Construct to hold logical blocks of data
 - Accessors and mutators
 - Generated for all data elements within structure

```
struct Quote
{
  string value;
};
```

⇒

```
public final class Quote
{
  public String value;
};
```

Attribute

- Maps to variable accessor and mutator methods
- In Java, maps to overloaded functions
 - Not JavaBean style get()/set(...) ☹
- Variables must be declared by developer
 - Not automatically generated by IDL compiler

```
string exchange; ⇒ String exchange();
void exchange(String arg);
```

Operations & Arguments

- Operation maps to a method
- Arguments for operations
 - Specify *direction*
 - IN (read in by method)
 - OUT (set by the method for return to caller)
 - INOUT (read and modified by the method)

```
Quote getQuote (in string symbol)
↓
public Quote getQuote (String symbol)
```

Exception

- Maps to a Java exception
 - In IDL, no inheritance of exceptions
- Operation
 - *raises* instead of *throws* exceptions

Quote `getQuote (in string symbol) raises UnknownSymbolException;`

↓
public Quote getQuote (String symbol)
throws UnknownSymbolException;

IDL to Java Mapping

Primitive Types:

IDL Type	Java Type
float	float
double	double
long, unsigned long	int
long long, unsigned long long	long
short, unsigned short	short
char, wchar	char
boolean	boolean
octet	byte
string, wstring	java.lang.String

IDL to Java Mapping

Complex Types:

IDL Type	Java Type
any	set of related classes
interface	set of related classes
sequence	array
struct	final class

Others:

IDL Type	Java Type
module	package
exception	exception class (inheriting from org.omg.CORBA.UserException)

Additional Notes

- IDL is case sensitive
 - Identifiers can't differ only by case

```
boolean foobar
interface FooBar
```
- No overriding or overloading of methods
 - Not all languages have these features
- Comments

```
// comment
/* comment */
```

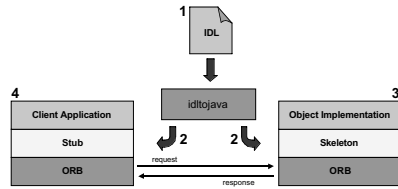
Developing CORBA Objects

- Define interface using IDL
- Process IDL to create *stub* and *skeleton* code
- Write code that implements the object (servant) and server to host it
- Write code that uses the object (client)

IDL Compilation

- IDL compilation
 - Generates code
 - Encapsulates underlying network code, marshalling
 - Compiled to language dependent interfaces
- Stub (client side)
 - Proxy, reference to a “remote” object
- Skeleton (server side)
 - Manage interaction between proxy and server implementation

Development Steps



steps:

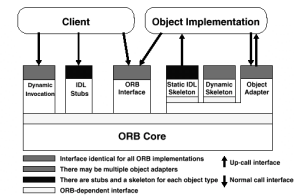
- 1 write the IDL file
- 2 compile with idltojava (stubs/skeleton generated automatically)
- 3 write object implementation (servant)
- 4 write client application

Object Adapters

Object Adapters

- Generate and interpret object references
- Activate and deactivate object implementations
- Handle method invocations via skeletons
- Basic Object Adapter (BOA)
- Portable Object Adapter (POA)

Object Adapters



Basic Object Adapter (BOA)

- Defines how objects are activated/deactivated
 - Initializing server objects:
 - `BOA.obj_is_ready(/* the object ref */);`
 - Registers the object with the ORB
 - `BOA.impl_is_ready();`
 - Tells the BOA/ORB to begin listening for requests
- Underspecified
 - Initially unclear which features would be required on various platforms

Portability Considerations

- Basic Object Adapter (BOA)
 - Vendors have taken liberty with the BOA specification
 - Various features are vendor dependent and non-portable
 - Implementation experiences used to define more complete and portable Object Adapter
- CORBA 2.2 introduced the Portable Object Adapter (POA)

Portable Object Adapter (POA)

- Replaces BOA
 - Most commercial implementations still use BOA
- Expanded scope of OA to include
 - Activation policies
 - Threading models
 - Object life cycle (transient/persistent)
 - Pre/post invocation capabilities

Interoperable Object Reference (IOR)

- “Shareable” reference to a CORBA object
- Compatible with all CORBA-compliant ORBs
- Analogy: URL for object instances
- Location independent
 - 1) Save an IOR
 - 2) Go to another location
 - 3) Load the saved IOR
 - 4) Establish communication with the same object

Creating a CORBA Object

- Initialize ORB and BOA
- Instantiate the object
- Export the object
 - BOA.obj_is_ready()
- Optionally register the object
 - NamingContext.rebind()
- Tell the ORB to begin receiving requests
 - BOA.impl_is_ready()

Activating the QuoteServer

```
public static void main(String[] args) {
    try {
        // Initialize object request broker
        ORB orb = ORB.init(args, null);
        // Initialize basic object adapter
        BOA boa = orb.BOA_init();
        // Create a new QuoteServer ...
        QuoteServer quoteServer = new QuoteServer();
        // ... and export the object
        boa.obj_is_ready(quoteServer);
        // Object Request Broker Initialized
        System.out.println("QuoteServer ORB initialized");
        // Wait for incoming requests
        boa.impl_is_ready();
    } catch (SystemException e) {
        System.err.println(e);
    }
}
```

Accessing a CORBA Object

- ORB uses Interoperable Object References (IORs)
- Object might be local, client uses *proxy*
- Client must acquire first object reference
 - Naming/Trading service
 - Proprietary bind
 - Proprietary URL service
 - Some other proprietary means

Registering the QuoteServer

```
// Create a new QuoteServer ...
QuoteServer quoteServer = new QuoteServer();
.
.
.
// Obtain reference for our nameservice
org.omg.CORBA.Object object =
    orb.resolve_initial_references("NameService");
// Since we have only an object reference, we must
// cast it to a NamingContext. We use a helper
// class for this purpose
NamingContext namingContext = NamingContextHelper.narrow(object);
// Add a new naming component for our interface
NameComponent name[] = { new NameComponent("QuoteServer", "") };
// Now notify naming service of our new interface
namingContext.rebind(name, quoteServer);
```

Using a CORBA Object

- Initialize the ORB on the client
- Get a reference to the remote object
 - IOR
 - Actual reference (e.g. from bind())
- Invoke methods

Accessing the QuoteServer

```
// Create an object request broker
ORB orb = ORB.init(args, null);
// Obtain object reference for name service ...
org.omg.CORBA.Object object =
    orb.resolve_initial_references("NameService");
// ... and narrow it to a NameContext
NamingContext namingContext = NamingContextHelper.narrow(object);
// Create a name component array
NameComponent name[] = { new NameComponent("QuoteServer", "") };
// Get a QuoteServer object reference ...
org.omg.CORBA.Object objectReference = namingContext.resolve(name);
// ... and narrow it to get a QuoteServer
QuoteServer quoteServer = QuoteServerHelper.narrow(objectReference);
// invoke methods on reference
Quote noveraQuote = quoteServer.getQuote("MCTR");
```

Services & Facilities

Services

- Naming
- Events
- Life Cycle
- Relationships
- Externalization
- Transactions
- Concurrency Control
- Licensing
- Query
- Properties
- Security (IOP/SSL)
- Collections
- Trading
- Time

Naming & Trading Services

- | | |
|--------------------------|--------------------------------------|
| • Naming | • Trading |
| – “White pages” | – “Yellow pages” |
| – Federated namespace | – Federated namespace |
| – Name to object mapping | – Attribute support |
| • No attributes | – Limited commercial implementations |
| – Limited protocol | |

Facilities

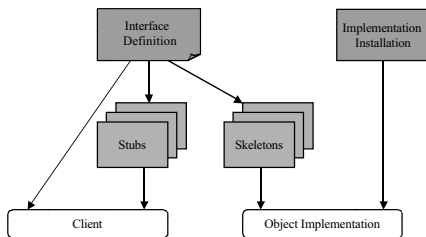
- Internationalization (I18N)
- Distributed Document Component
 - Based on OpenDoc
- Systems Management
- Time
- Data Interchange
- Mobile Agent
- Printing

CORBA versus RMI

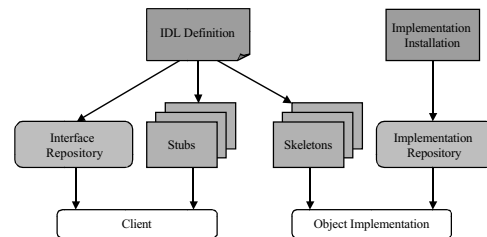
CORBA vs. RMI

CORBA	RMI
Platform independent	JVM specific
Protocol independent (IIOP)	Proprietary protocol (JRMP)
Language independent (IDL)	Java specific
Objects by value (3.0)	Objects by value (serialization)

RMI Deployment



CORBA Deployment



Summary

CORBA Features

- Language independence
- Location transparency
- Reuse of facilities & services
- OMG IDL defined interfaces
- Stub & Skeleton generation
- Server activation

CORBA Resources

- Object Management Group (OMG)
 - www.omg.org
 - www.corba.org
- Discussion groups
 - comp.object.corba
 - comp.lang.java.corba
- Implementers
 - Inprise: www.inprise.com
 - Iona: www.ionac.com
 - Sun: java.sun.com/products/jdk/1.2/