CORBA
Common Object Request Broker Architecture

Based partially on Notes by D. Hollinger and Java Network Programming and Distributed Computing Chapter 12
Review of some benefits of Java RMI

- A network of heterogeneous machines is seen as a homogeneous network of Java virtual machines.
- Transparency of access to remote objects: remote method invocation looks like local method invocation, modulo exception handling.
CORBA

• The notion of having objects distributed across the network has been around for a while.
• The Object Management Group (OMG) was formed in 1989 to create a set of standards that would facilitate the development of distributed object-oriented applications.
Buzzword Warning

• CORBA is a rich source of acronyms and buzzwords.

• OMG is now the largest standards body that has ever existed (on this planet).

• First buzzword: *Middleware* - software that hides the details of network programming from programmers, so they can worry about the application. CORBA is middleware.
Important message from our sponsor

IDL does not provide a complete definition of OMA, nor does it facilitate the use of DII in conjunction with an ORB. Only with the aid of a BOA or alternative OAs as described in the RFPs and RFIs will it be possible to make use of any IIOP compliant system.
Object Management Group

- OMG creates specifications, not implementations.
- Some Key Specifications:
  - OMA: Object Management Architecture.
  - CORBA: Common Object Request Broker Architecture.
OMA Object Model

- Objects provide services.
- Clients makes a request to an object for a service.
- Client doesn’t need to know where the object is, or anything about how the object is implemented!
- Object interface must be known (public) - provides signature for each object method.
Object References

- Clients don’t *have* objects, they just have object references.

- Object references can be persistent (saved for use later).
Accessing Remote Methods

- Clients can call remote methods in 2 ways:
  - Static Invocation: using stubs built at compile time.
  - Dynamic Invocation: actual method call is created on the fly. It is possible for a client to discover new objects at run time and access the object methods.
Interface Definition Language

- IDL is the language used to describe object interfaces, the same as Java interfaces in RMI.
- IDL is a declarative language, it only describes object interfaces.
- IDL is language neutral - there are mappings for many object oriented languages (C++, Smalltalk, Java).
Inheritance

• IDL supports interface inheritance
  – all operations are effectively virtual.

• IDL doesn’t say anything about implementation!
Interface Repository

• An IR provides persistent storage of IDL interface declarations.
• IR serves 2 purposes:
  – Support dynamic invocation interface (DII).
Object Adapters

• Object Adapters provide a layer between object method requests and the servers that service the requests. Functions include:
  – generation of object references
  – starting up the actual server program(s)
  – handling security
Basic Object Adapter

- Simplest object adapter, can support a number of different implementations:
  - one server that always is running
  - one program that can handle requests for multiple objects.
  - one program per object implementation.
  - one program for each object method.
Portable Object Adapter

• POA newer than BOA.
• Supports additional services:
  – Persistent objects.
  – Threads.

• So far there is no TOA, ZOA or OOA
Object Request Broker

• The ORB is an abstract entity that acts as the middleman in all remote method invocations.
• The ORB finds a server that can handle a method invocation, passes the request to the server, receives the response and forwards it to the client.
• The functions handled by an ORB are actually implemented in both client and server.
A Description of The ORB

http://www.omg.org/corba/whatiscorba.html

"The (ORB) is the middleware that establishes the client-server relationships between objects. Using an ORB, client can transparently invoke a method on a server object, which can be on the same machine or across a network."
A Picture from OMG

www.omg.org/gettingstarted/corbafaq.htm

Figure 1: A request passing from client to object implementation
ORB Differences

• The Specification of the functionality of an ORB is not a complete implementation description.
• Many of the details are left up to the implementor.
• Every Vendor does things differently.
• You write code to work with a specific ORB.
Inter-ORB Protocol

• There is support for connecting ORBs.

• The most significant support is the Internet Inter-Orb Protocol (IIOP)
  – Specifies the details of communication (bridging) between ORBs.
Multiple ORB Picture

www.omg.org/gettingstarted/corbafaq.htm

Figure 2: Interoperability uses ORB-to-ORB communication

Copyright © 2000 Object Management Group
Call Semantics
(part of Corba Object Model - 1.2.8.5 Execution Semantics)

"Two styles of execution semantics are defined by the object model:

• **At-most-once**: if an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at-most-once.

• **Best-effort**: a best-effort operation is a request-only operation (i.e., it cannot return any results and the requester never synchronizes with the completion, if any, of the request)."
General Layout of IDL File

module identifier {
  type, constant & exception declarations

  interface identifier : base {
    attribute declarations
    type identifier(parameters)
    raises exception;
    type identifier(parameters)
    raises exception;
    ...
  }
}
module MyAnimals {
    interface Dog:Pet,Animal {
        attribute integer age;
        exception NotInterested(string explanation);
        void Bark(in short how_long)
            raises(NotInterested);
        void Sit(in string where)
            raises(NotInterested);
    }
    interface Cat:Animal {
        void Eat();
    }
}

IDL for Bank Account Example

interface bank_account {
    exception overdrawn_exception {};
    void deposit(in float amount);
    void withdraw(in float amount)
        raises (overdrawn_exception);
    float balance();
};
Java Mapping to IDL Datatypes

- Void
- Boolean
- Char
- Byte
- Short
- Int
- Long
- Float
- Double
- java.lang.String

- void
- boolean
- wchar
- octet
- short
- long
- long long
- float
- double
- string / wstring
Parameter passing in IDL

- Parameters to a method can be declared of three different types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>Used for input only</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>Contents may be modified</td>
</tr>
<tr>
<td><strong>inout</strong></td>
<td>May be used for input and may also be modified</td>
</tr>
</tbody>
</table>
IDL Exception Handling

• It is still possible to define different exception types.
• However, there is no class hierarchy of exceptions like in Java.
• So, it is not possible to catch a related subset of possible exceptions within a single `catch(NetworkException ne)`.

A “feature”?
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