Introduction to Distributed Computing using CORBA

Rushikesh K. Joshi
Dept of Computer Science & Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai - 400 076, India.

Email: rkj@cse.iitb.ac.in
Why Do You Go for Distributed Computing?

• The information itself is inherently distributed due to the physical distributed nature of an organization

• Explicit distribution gives higher reliability, availability, performance etc.
What is the problem with traditional Single Address Space Computing?

- Objects have to be in the same address space, and hence an object cannot send a message to an object that is on a different machine.

You need to extend or enrich the traditional model for facilitating distributed computing.
Programming Paradigms
Distributed Computing

- Socket based programming
- Typed streams
- Remote Procedure Calls
- Programming Languages: SR, Lynx..
- Distributed Shared Memory
- Distributed Objects
A Distributed Object Computing Scenario

• *Server objects* and client programs located on different machines

• Client programs send messages to these server objects which are remote

• *Location Transparency*: Clients can send messages to these objects as if they are available locally
What is OMG and the CORBA Standard?

- **OMG**: The Object Management Group consisting of over 600 companies evolved the CORBA specs: Since 1989

- CORBA is a specification for the distributed object bus architecture defined by OMG

- OMG issues specifications, not products
Overview of the Object Management Architecture

Application Objects

Object Request Broker (ORB)

Object Services

Common Facilities

Application Objects
The ORB (object request broker)

ORB is the core of the Object Management Architecture

• Through ORB, objects written in different languages on different machines of different architectures running different operating systems can communicate to each other
Structure of the ORB

ORB is responsible for:

• mechanisms to find implementations for requests
• To prepare implementations to receive reqs
• To communicate data making up the reqs.
• ORB is not required to be implemented as a single component, but is defined by its interfaces
Commercial ORBs

There are commercial ORBs available

*Examples:*

- CORBAplus - Expertsoft
- Orbix - IONA
- Visibroker - Visigenic, now with Inprise
The Language of Application Development

• Client can be developed in one language, say C++

• Server can be developed in another language, say JAVA
Client

Observe the Location Transparency

// This is a client
// ...
main ()
{
    Library * iitb_lib ;
    //...
    //...
    iitb_lib = Library :: bind ("IITB_LIB");
    Book b = iitb_lib->list_a_book ("OOP");
}
Clients

- Have references to objects and invoke operations on them
- Clients know only the logical structure of the server objects (interfaces)
- Have no knowledge of implementations of the objects and object adapters used by these implementations
How do Clients invoke a Server Interface?

- May invoke server implementations through the *IDL generated stubs* (proxies)

  OR

- May invoke through the *Dynamic Invocation Interface*
The Interface Definition Language

• A server object declares its interface in the standard Interface Definition Language specified by the CORBA specification
• IDL separates the interface from implementation
• These interfaces are also commonly referred to as IDLs.
The Server

• The server object can register itself with the ORB and declare that it is available for accepting requests.

• It can also register its name which the clients use to get a handle for the server object.
An Example Server

//....
Class Library_Skeleton { ....}; // generated for you

Class Library_Impl : public Library_Skeleton {...};
main ( )
{
    Library_Impl *lib ;

    lib = new Library_Impl;

    orb->object_is_ready (lib);
    orb->implementation_is_ready (lib);
}
IDL: The Core of CORBA Spec
The Interface Definition Language

- IDL provides a language/OS independent interfaces to all objects, services and components on the CORBA bus
- The OMG IDL is purely declarative: that means, no implementation details are provided
- It is strongly typed.
- IDL specs can be written and invoked in any language that specifies CORBA bindings (C/C++/COBOL/Smalltalk)
Server implements an IDL and Client invokes interfaces defined by an IDL

- Implementation is in an implementation language
- Invocations are also in an implementation languages
- IDL to language mapping is necessary
- e.g. mappings for C/C++/COBOL/Smalltalk/Java
An Example IDL

Interface Account {

    void deposit (in float amount);
    void withdraw (in float amount, out float balance);

}
Inheritance

Interface Clock {
void setTime();
void start();
void stop();
};

Interface AlarmClock : Clock {
void setAlarm();
void stopAlarm();
void testAlarm();
};

Multiple inheritance is allowed
Inheritance..

• Inheritance of interface
• Components with both types of interfaces may exist
• Does not imply inheritance of implementation. The component implementing the derived may implement both interfaces entirely independently or may reuse an existing component
OMG IDL Features

- Modules
- interfaces
- operations
- attributes
- inheritance
- basic types

- Arrays
- sequences
- struct, enum, union
- typedef
- consts
- exceptions
Basic Types for use in IDL

- float
- double
- long
- short
- unsigned long
- unsigned short
- char
- boolean
- octet
- any
Direction of Parameters

• In (from client to server)
  object

• out (from server to client)

• inout (from and to client)
Exceptions

Interface Bank {
    exception Reject {
        string reason;  // a data member
    };
    exception TooMany {
    };  // to be returned when capacity exceeded

    Account newAccount (in string name) raises (Reject, TooMany);
};
One-way Operations

Interface Account {
  oneway void notice (in string notice);
};

Oneway operations do not block.
They cannot accept out and inout parameters.
They cannot have a raises clause.
Constructed Types: Structures for use in IDL

```cpp
struct PersonalDetails {
    string Name;
    short age;
};

interface Bank {
    PersonalDetails getPerDet (in string name);
};
```
Constructed Types: Arrays

- They can be multi-dimensional
- They must be of fixed size: known in the idl definition time

```
Account bankAccounts [100];

short matrix [10] [20];  // 2-d array
```
Constants

Interface Bank {
  const long MaxAccounts = 10000 ;
  ...
};

constants of types such as long, float, string can be declared
typedef short size;
size i;

typedef Account Accounts [100];
Accounts bankAccounts ;
Modules

Module Finance {
    interface Bank { ..... };
    interface Account { .... };
};

Modules are used to group interfaces into logical units.
Use full name of Account and Bank interfaces such as:

Finance::Account *a;
Preprocessor

- Macro substitution
- conditional compilation
- source IDL file inclusion

such as:

#include  #define  #if
#ifdef   #defined   ......

It is based on the C++ preprocessor
The IDL to Language Mapping

- Different languages (OO/non-OO) access CORBA objects in different ways

- Mapping covers:
  - Language specific data types
  - Structure of the client stub (only non-OO lang)
  - Dynamic invocation interface
  - Implementation skeleton
  - Object Adapters
  - Direct ORB interface
Mapping the Identifiers

• *Identifiers are mapped to same names*
  
  e.g. `add_book` in IDL is mapped to -->
  
  `add_book`

• *But if they are C++ keywords, an underscore is prefixed*
  
  e.g. `new` is mapped to -->`_new`
Mapping of Interfaces

• *Interfaces are mapped to classes*

*Interface Account { ... } becomes*

class Account : public virtual CORBA::Object { ..}

An IDL mapped C++ class cannot be instanciated
Mapping Scoped Names

\textit{Interface Bank} { \\
\textit{struct Details} { ..} \\
....} \textit{is mapped to} \\
\textit{class Bank} { \\
\textit{public:} \\
\textit{struct Details} {...} \\
};
Mapping Scoped Names

Module $M \{ \text{ Interface } A \{ .. \} \text{ Interface } B \{ .. \} \}$

is mapped to

namespace $M \{$
    class $A \{ .. \}$;
    class $B \{ .. \}$;
}

refer to them as $\Rightarrow M::A$ or $M::B$ etc.
Mapping the standard CORBA module

Is mapped to

namespace CORBA {
  ..
}

Use the members as follows:

CORBA::ORB_init (..);
Mapping the Basic Data Types

- **IDL**
  - short
  - long
  - unsigned short
  - unsigned long
- **C++**
  - CORBA::Short
  - CORBA::Long
  - CORBA::UShort
  - CORBA::Ulong
  - CORBA::Float
  - CORBA::Double
### Basic Data Types

<table>
<thead>
<tr>
<th>IDL</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>CORBA::Char</td>
</tr>
<tr>
<td>boolean</td>
<td>CORBA::Boolean</td>
</tr>
<tr>
<td>Octet</td>
<td>CORBA::Octet</td>
</tr>
<tr>
<td>any</td>
<td>CORBA::Any</td>
</tr>
</tbody>
</table>
Interface Repository

• Provides storage to store IDL information
• A program may refer to objects whose interface would be known at runtime
• This info may be used by the ORB to perform requests
• IR may be made to store other info about interfaces such as debugging info, browser routines etc
Implementation Repository

• Contains information that allows ORB to locate and activate the implementation of a required server object

• Also for storing server activation information such as the machine where a server would be started on a client’s request
Dynamic Invocation Interface

- Rather than calling a specific stub routine for an operation, it is possible to specify an object, operation on it, and parameters to it through a call or sequence of calls.

- Client must also supply the types of parameters passed.
Interoperability

• For supporting networks of objects distributed across multiple heterogeneous CORBA-compliant ORBs

--> InterORBability

• GIO\textit{P} : Standard transfer syntax and a set of message formats for communication between ORBs

• IIOP : The TCP/IP mapping of GIO\textit{P}
CORBA Services: Common Object Service Specification (COSS)

An ORB is just like a telephone exchange that connects objects. Applications require other services defined in terms of IDL.

OMG has brought out a COSS that includes services such as naming, events, life cycle, time, transactions, concurrency, persistence, query, security, licensing, relationships, properties, externalization and collection.
Common Facilities

• Newest area of OMG’s standardization

• ORB and Object Services are fundamental technologies, and common facilities extend them to application developers

• Horizontal and Vertical facilities

• e.g System management, compound documents, financial services

• May become most important area of OMG standards