Topics to be covered

- Remote Procedure Calls.
- Conceptual overview of CORBA
- CORBA IDL
- Understanding the Broker-OA and BOA
- Interoperability
- Applications of CORBA
Remote Procedure Calls

Client Program calls functions on the Server (remote), the following functions take place.

- Client Calls Remote Procedure → Local Host is generated by the RPC Package called Client Stub → This Packages Messages & dispatches it to the Server
- Server Stub on Server Host Receives Message → Calls Server Procedure
- Server Function returns to Server Stub → this Packages Values and sends it back to Client Stub
- Client Stub takes return values and returns it to Client Application.
Typical RPC Interaction Diagram
Object Management Group

- The **object management group (OMG)** was formed in 1989. It is a group of companies which defined an object oriented architecture for applications which form the basis for CORBA. There are 4 components in OMG OMA:
  - **Object request broker**: ORB provides the communication hub for all objects. Analogous to the HARDWARE BUS.
  - **Object services**: provides set of standard functions to create objects, track object and object references.
  - **Common facilities**: common facilities provide a set of general purpose application capabilities for use by many different application.
  - **Application object**: provide a set of objects that perform specific tasks for end users. They are essentially object oriented applications.
Enter CORBA

- CORBA (Common Object Request Broker Architecture) is a specification for a standard object architecture. CORBA provides the ability to:
  - Access Distributed Information and Resources from within popular desktop applications
  - Make existing business data systems available as Network Resources

What Problems does CORBA Solve??

- Augment popular desktop tools and applications with custom functions and capabilities
- Change or evolve network based systems to reflect new topologies or new resources
CORBA and Distributed Object Computing

- CORBA is based on distributed object computing-enable applications to share each others objects.
- CORBA uses a broker or intermediary, to handle messages called requests. Broker separates the interface and the implementation.

Various Methods of Distributed Computing:

- Using RPC (Procedures are typically synchronous in nature)
- Coding to a Network Application Programming Interface (API) - support both asynchronous and synchronous communication

CORBA enhances distributed computing:

- Allowing Flexible, changing relationship between clients and servers
- Allowing Servers to be more than single process.
- Supporting both Communication styles
- Adding an intermediary called the broker
ORB Middleware
Client sends request to Sender
ORB (Object Request Broker)

It acts like an Object Bus through which all the distributed objects move from server to client. It has the following functions:

1. To find the Objects implementation
2. To prepare the Object’s implementation to receive request
3. To communicate data to the requestor

The interface of the client is independent of the language used in implementation.

ORB is language neutral

Use of *any language* to create clients that invoke remote methods through the ORB. ‘*any language*’ means that you need a language mapping defined between the implementation language and CORBA’s Interface Definition Language (IDL).
Important components of CORBA Architecture that works in tandem with ORB to make it a full-fledged and a popular middleware. These components are as follows:

- Interface Definition Language (IDL)
- Internet Inter ORB Protocol (IIOP)
- Client Stub / Server Skeleton
- Client Side - Dynamic Invocation Interface
- Server Side - Dynamic Skeleton Interface
- Client Side – Interface Repository
- Server Side – Implementation Repository
Complete CORBA(ORB) Architecture
OBJECT IDENTIFIERS

- OID (Object Identifiers) are handles to objects.
- In CORBA the OID exists as Object References
- Instead of Passing the Object in CORBA over the wire we pass that handle over the wire.
- Results in saving Bandwidth
The CORBA’s Interface Definition Language is the language used to describe the interfaces that client objects call and also the interfaces that object implementations provide.

IDL is purely a descriptive language/definition language.

IDL is CORBA’S object contract language
CORBA Interfaces

- CORBA interface is a collection of 3 things
  - OPERATIONS
  - ATTRIBUTES
  - EXCEPTIONS

ACCESS TO ATTRIBUTES are through
- ACCESSOR (get operation)
- MUTATOR (a set operation)
CORBA IDL contd..

- IDL compiler is composed of
  - FE (Front End or the IDL Compiler or IDL FE)
  - BE (Back End or IDL BE)
- IDL FE $\rightarrow$ Preprocessing
- IDL BE $\rightarrow$ Produces the translation(or mapping) and is Target Language Specific creates source code for C, C++, Rexx, COBOL
interface Box
{
    attribute short content;
};

Signatures: If any signatures or attribute changes the types are different and cannot be the same interface.

interface Window
{
    boolean open ();
    boolean close();
}
IDL contd..

- **Parameter Passing Modes**
  - In
  - Out
  -InOut

    ```
    interface Account
    {
        void debit( in money amount , out money balance);
        void credit(in money amount , out money balance);
        void adjustment(inout money amount);
    };
    structure money { long units; short fractions;, currency currency_type};
    ```
Stubs

Compiling the IDL results in 2 items:

- **Client Stub**: Implementation of the `ObjRef`
- Implementation Skeleton
- **Stub** is the Local Object the Client Makes Requests
- **Stub** is the real objects local (client side proxy)
Skeletons

- Skeleton is the Implementation-side equivalent of the stub
- Framework for constructing the objects actual implementation.
- Coordinates Invocation of the object with the Object Adapter.
- Object Adapters access the objects methods using an implementation definition object
Object Reference

- An Object Reference is an instance of a stub
- Objref is an instance of the stub.
- Object References may be made persistent
- Allowing persistent references causes architectural difficulties
- An ORB is the extent of it’s objects references.
Object Interface

- All CORBA Objects derive from the Object Interface.
- Any time an Object Reference is present the Client has access to all the operation in the object reference.

The Object interface has 12 operations:

1) Get Implementation: The Get Implementation takes no parameters and returns an implementation definition.
2) Get Interface: returns an interface definition.
3) Is Nil: built in assertion about whether a particular object exists or not.
4) Duplicate: allows a client to make a copy of an object reference
5) Release: This allows Storage Reclamation when the use of the reference is complete
6) **Is A**: operation returns a bool. It takes in a Logical Type Id which is a string that denotes a Repository ID.

7) **Nonexistent**: returns a bool and takes no parameters. It return true when the ORB unequivocally knows the implementation no longer exists.

8) **Is Equivalent**: checks the operation against parameter object reference. Two references which are identical are equivalent.

9) **Hash**: returns an unsigned long and has one parameter an unsigned long Object References contain enough information for an ORB to route requests. There is a globally unique identifier for the look up service and a key where ORB goes to determine what to do with requests. CORBA specs state that the value never changes and the ASH finds this out.

(In short a inexpensive method for finding Is Equivalent)
Dynamic Invocation Interface

Client

Implementation

Stub  Dll interface  ORB interface  BOA interface  Skeleton

ORB (Object Request Broker)
DII contd..

- DII makes objects useful objects that are unknown at compile time.
- Knowing objects at Compile time can be more efficient
- A static client must have compile time knowledge of an object to use to it’s operations
- Static Clients can still use the Dynamic approach by the way of the Request object
- This is specified in the Create Request object operation.
Create Request

- **Status** `create_request( in Context ctx, in Identifier operation, in NVList arg_list, inout NamedValue result, out Request request, in Flags req_flags );`

- Context: to define a runtime context
- Identifier is a name
- Named Value is the pair of Identifier and the Any Value
- Request: is a pseudo object
- Request flags: are of the flag type and contain parameter passing information like CORBA::ARG_OUT or CORBA::ARG_INOUT
Request Object Interface

Interface Request {
    Status add_arg(
        in Identifier name;
        in TypeCode arg_type,
        in void  *value
        in long  len,
        in Flags arg_flags);
    Status Invoke(in Flags invoke_flags)
    Status delete();
    Status send(in Flags invoke_flags)
    Status get_response(in Flags response_flags)
};
The Request Object has 5 operations all of which return a Status.

- **Add Argument**: Gives Specification for the object to created
- **Invoke**: Instructs Request to execute the operation
- **Delete**: Releases objects resources
- **Send**: instructs request to execute operation(Flags parameter)
- **Get Response**: Flag is Set to CORBA::RESP_NO_WAIT that call should not block if no response occurs.
The ORB

- Object Request Brokers are in a general sense are a class of facilities of which CORBA is an instance.

- ORB interface operation include:
  - Operations that convert object references to String and Back
  - Operation to create Named Value List
  - Operation to create Named Value List for a specific operation.
  - To get the Default context
  - To initialize the OA(BOA)
Fundamental Operations

- **ORB Initialize**: not implemented on any object that returns ORB as a result also known as the BOOTSTRAP mechanism with which ORB initializes itself.

  ```c
  ORB ORB_init(inout arg_list arv, in ORBid orb_identifier);
  ```

- **Persistent Object References**: Sending object references by Email was the main motivation behind this.

  ```c
  string object_to_string(in Object obj);
  string string_to_object(in string str);
  ```
Object Adapters

- Suggests the style of implementation of the object.
- CORBA has a general object adapter called the Basic Object Adapter (BOA).
- BOA supports different styles of Activation tight coupling to the ORB.
- BOA is involved in the various parts of an objects life cycle like creation, destruction, activation and deactivation.
Activation has four distinct styles:

- **Shared Server**: Objects share one process
- **Unshared Server**: Separate Process per object
- **Per Method Server**: Separate Process per Request
- **Persistent Server**: Active from System Start up and does not require BOA activation
Typical CORBA interaction diagram

Client  Stub  ORB  BOA  Skeleton  Implementation

1  2  3  4  5
Library Object Adapters (LAO): may concern bringing a shared local library on first use and never afterward. Can be used for WAN development.

Load Balancing Object Adapter: resource aware OA’s might communicate with each other in such a way that they could forward requests for starting an implementation at a location which has more available resource.

Mobile Object Adapters: knows more about the object location than the BOA. It may queue requests for roaming objects.
Repositories IR interface repositories

- Client
- Implementation
- Stub
- DLL interface
- ORB Interface
- BOA Interface
- Skeleton

ORB
Interface repository containment hierarchy
Interoperability
Interoperability contd..

- Inter ORB Bridges require 4 things
  - Way to represent null references
  - Way to signify type so that strongly based systems preserve integrity
  - Way to determine protocols in use.
  - Way that each correspondent gains access into others private session keys. (required for decrypt and encrypt the incoming and outgoing protocols respectively)

- GIOP and IIOP provide this through Interoperable Object Reference)IOR internals are available to the ORB builder.
General Inter-ORB Protocol

- Application
- Presentation
- Session
- Transport
- Network
- DataLink
- Media

- Stubs/Skeleton
  - GIOP
    - IIOP
    - ?
    - ?
  - TCP
  - PPP
  - SNA

CORBA Interoperability
Internet Inter ORB Protocol

- IIOP maps the GIOP on the TCP/IP session
- IIOP specifies a profile. This profile is the instruction for the GIOP to use the IIOP protocol for its connection.

```cpp
module IIOP
{
  struct Version
  {
    char major;
    char minor;
  };
  struct ProfileBody
  {
    Version iiop_version;
    string host;
    unsigned short port;
    sequence<octet> object_key;
  };
};
```
Environment Specific Inter-orb Protocol

Application
Presentation
Session
Transport
Network
DataLink
Media

Stubs/Skeleton
GIOP
GIOP
IIOP
ESIOP
DCE-CIOP
TCP
Caveat: Requirements & Historical Limitations of CORBA for Real-time Systems

Requirements
• Location transparency
• Performance transparency
• Predictability transparency
• Reliability transparency

Historical Limitations
• Lack of QoS specifications
• Lack of QoS enforcement
• Lack of real-time programming features
• Lack of performance optimizations
Applying RT CORBA to Image Processing

Goals
• Examine glass bottles for defects in real-time

System Characteristics
• Process 20 bottles per sec
  • i.e., ~50 msec per bottle
• Networked configuration
• ~10 cameras
An Example Distributed Application

- Consider an application where cooperating drones explore a surface & report its properties periodically —e.g., color, texture, etc.
- This is a simplification of various autonomous vehicle use-cases

- Drones aren’t very “smart,”
  - e.g., they can fall off the “edge” of the surface if not stopped
- Thus, a controller is used to coordinate their actions
  - e.g., it can order them to a new position
Topics not Covered

- Security
- Event Services Push and Pull
- Transaction Services
- Persistence Service
- Licensing Service
- Concurrency Control
References

- The CORBA Reference Guide : Alan Pope
- Understanding CORBA : Randy Otte Paul Patrick and Mark Roy
- CORBA Fundamentals and Programming
- Core Java 2 Volume 2 Cay S Horstman Gary Cornell
- Web site : www.omg.org for specifications
- TCP/IP illustrated R Stevens