CprE 450/550X
Distributed Systems and Middleware

Distributed Object-based Systems (CORBA)

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

March 4, 2003

Readings for Today's Lecture

- References
  - Chapter 9 of "Distributed Systems: Principles and Paradigms"
  - http://www.corba.org/
  - http://www.omg.org/gettingstarted/
  - http://www.omg.org/gettingstarted/readingroom.htm
  - "Understanding CORBA"
  - "Examples of Writing CORBA Applications", http://www.cs.wustl.edu/~schmidt/PDF/corba-apps4.pdf
  - "Introduction to Distributed Object Programming with CORBA ", http://www.cs.wustl.edu/~schmidt/PDF/corba4.pdf

3

Outline

- Role of CORBA and need for object oriented distributed computing
- A simple CORBA architecture
- CORBA client-server example
- Coding with IDL
- Complete CORBA architecture and its various components
- Some CORBA products and vendors

4

CORBA and OMG

- CORBA (Common Object Request Broker Architecture) is a **standard** for distributed objects being developed by the Object Management Group (OMG) that provides the mechanisms by which objects transparently make requests and receive responses
- CORBA provides interoperability between applications built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments
- The OMG is a consortium of software vendors and end users

5

CORBA and Distributed Computing

- Access distributed information and resources from within popular desktop applications
- Make existing business data and systems available as network resources
- CORBA's model of object oriented computing makes reuse of software components and application development easier
- CORBA enables applications in a heterogeneous distributed environment to access and share each other's objects

6

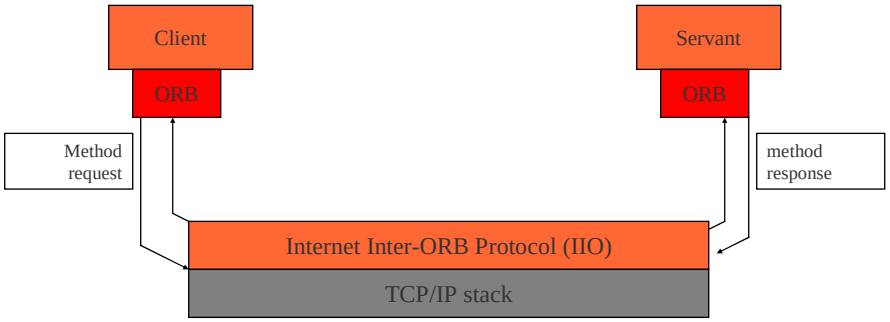
Middleware

- Middleware is a type of distributed system software which connects different kinds of applications and provides distribution transparency to its connected applications
- It is used to bridge heterogeneities that occurred in the system
- Middleware insulates applications from the lower-level details and complexities of the software on which the system depends

CORBA has been called a communications middleware

7

Simple CORBA Architecture



8

ORB (Object Request Broker)

- Uses Object Reference to identify and locate objects
  - ***Object Reference*: A handle to an object that a client must hold in order to access the object**
- Delivers request to objects
- Returns output values back to client
- Services necessary to accomplish the tasks are completely transparent to the client

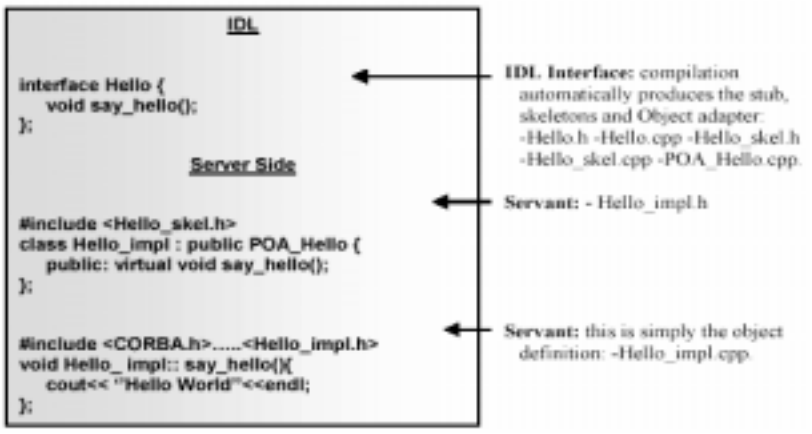
9

CORBA Application Development

- Steps in developing a CORBA server and client
  - **Design your application interface and specify them in OMG IDL (Interface Definition Language)**
  - **Run the IDL specs through IDL compiler of language of your choice, say C++, to generate client-side stub and server-side skeleton**
  - **Implement server side interfaces using C++ classes (called servants)**
  - **Implement the server program that instantiates the servants**
  - **Compile the server program along with the skeleton code using a C++ compiler**
  - **Implement the client program**
  - **Compile the client program along with the stub code**

10

A Sample Client – Server Program



Client – Server Program (Cont.)

```
#include <CORBA.h>...<Hello_impl.h>
int run (CORBA :: ORB_Ptr);
int main (){
    CORBA :: ORB_var orb;
    orb = CORBA :: ORB_init ();
    status = run (orb);
    ..........
}

int run (CORBA :: ORB_Ptr orb){
....create references to and initializes POA..
....create pointers to servant....
    CORBA::String s = orb -> object_to_string();
    const char* refFile = "Hello.ref";
....use I/O stream to make file....
    orb -> run();
}
```

Server: include our servant and CORBA packages: -Server.cpp
Forward declaration.
Creates and initialize orb.
Return a pointer.
Start the orb.

object_to_string function creates a string file (IOR) that must be used by the client to find the server implementation.

Client – Server Program (Cont.)

Client Side

```
#include <CORBA.h>...<Hello.h>
int main (){
...similar to server side...
...code to initialize and destroy ORB...
}

int run (CORBA :: ORB_Ptr orb){
    const char* refFile = "Hello.ref";
....use I/O stream to read file....
    CORBA::Object_var obj = orb ->
    string_to_object();
...now can call function as normal....
    obj-> say_hello();
};
```

Client: include CORBA packages: -Client.cpp.

Read IOR and create object reference.

The function can now be called as normal even if it is in another machine or in another process.

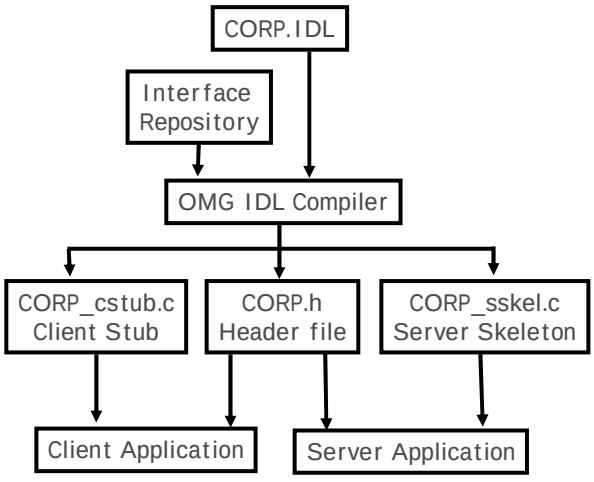
13

Interface definition Language (IDL)

- Separates object implementation from interface
- Basically a declarative language, similar in appearance to C++
- A means by which the object implementation tells clients what operations are available and how to invoke them.
- Mapped to a particular programming language (C , C++, Java)
- IDL compilation produces stubs/skeletons
  - **stub - local function call for the client**
  - **skeleton - server side of the object implementation**
- Client - Server communication is facilitated by stubs & Skeletons

14

Coding with IDL



Coding with IDL (cont.)

```
//File CORP.IDL

Module CORP
{
    typedef long BadgeNum;
    typedef long DeptNum;
    enum DismissalCode {DISMISS_FIRED, DISMISS_QUIT}

    Interface Employee
    {
        void promote(in char newjobclass);
        void dismiss(in DismissalCode reason,
                in string description);
    }

    ......
}
```


Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining an object attribute in ODL

Module CORP
{
    typedef long BadgeNum;
    typedef long DeptNum;
    enum DismissalCode {DISMISS_FIRED, DISMISS_QUIT}

    struct DeptInfo
        {
            DeptNum id;
            string name;
        }

    Interface Department
    {
        atttribute DeptInfo DeptID;
    }

    ......
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining an read-only object attribute in ODL

Module CORP
{
Interface Employee;

    struct DeptInfo
        {
                DeptNum id;
                string name;
        }

    Interface Department
    {
        atttribute DeptInfo DeptID;
        readonly attribute Employee manager_obj;
    }
    Interface Employee
    {
        attribute EmpData personal_data;
        readonly attribute Department department_obj;
    }
    ……
}
```


Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: single inheritance

Module CORP
{
    struct PersonalData    {
                           string lastname;
                           string firstname;
                           string phone;
        }
    typedef PersonalData EmpPersonalData;
    struct EmpData {
                           BadgeNum id;
                           char job_class;
                           float hourly_rate;
        }
    Interface Employee
    {
        attribute EmpData personal_data;
        readonly attribute Department department_obj;
        void promote(in char new_job_class);
        void dismiss(……);
        void transfer(……);
    }
    Interface Manager: Employee
    {
        void approve_transfer(……);
    }
    ……
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: multiple inheritance

Module CORP
{
    Interface Employee
    {
        ……
    }
    Interface Manager: Employee
    {
        ……
    }
    Interface Peronnel: Employee
    {
        ……
    }

    Interface PeronellManager: Personnel, Employee
    {
        ……
    }
    ……
}
```


Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining inheritance in ODL: inheritance across modules

Module CORP
{
    Interface PeronellManager: Personnel, Employee
    {
        ……
    }
    ……
}

Module ENGINEERING
{
    Interface EmployeeLocator
    {
        void FindEngineer(in CORP::BadgeNum id,
                          out CORP::PersonalData info);
    }
    Interface PersonnelManager: CORP::PersonnelManger
    {
    }
}
```

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining User-defined Exceptions

Module CORP
{
    enum DenyApprovalReasons {REASON, CODES};
    exception DENY_APPROVAL
        {
            DenyApprovalReasons reason;
        }
    Interface Manager: Employee
    {
        void approval_transfer (in Employee employee_obj,
                                in Department current_department,
                                in Department new_department)
                                raises (DENY_APPROVAL);
    }
    ……
}
```

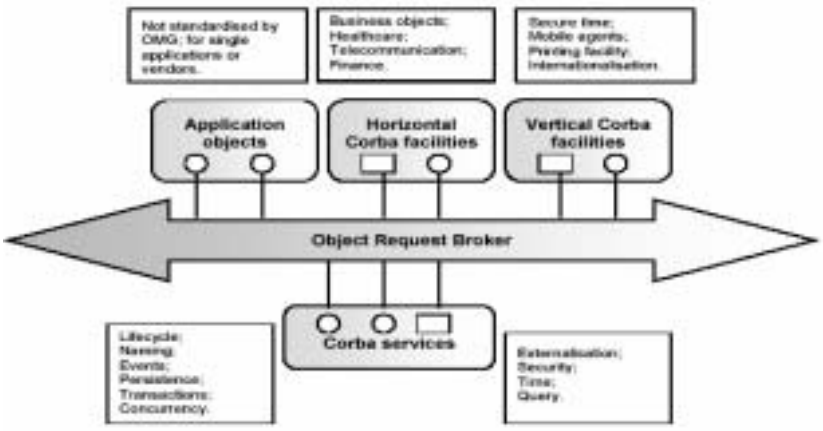

Coding with IDL (cont.)

```
//File CORP.IDL    ---- Defining context objects

Module CORP
{
    Interface Manager: Employee
    {
        void approval_transfer (in Employee employee_obj,
                                in Department current_department,
                                in Department new_department)
                                raises (DENY_APPROVAL)
                                context("division");
    }
    ……
}
```

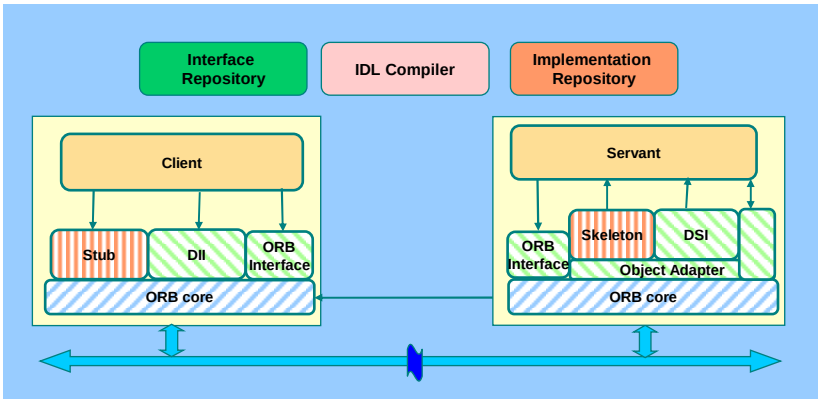
23

The Object Management Architecture



24

CORBA Components



25

Static and Dynamic Invocation Interface

- Static Invocation Interface (SII)
  - **Client knows interface operations in advance**
  - **Client is compiled with the relevant stub**
  - **During invocation, the proxy object understands the parameters in an operation and marshals them into the request**
- Dynamic Invocation Interface (DII)
  - **A client may not always have the stub available at compile time**
    - **Bridges, Proxy servers**
  - **Allows clients to discover operations parameters using Interface Repository and create requests dynamically**
  - **More flexible but less efficient. Also, more complicated and less typesafe**

26

Interface Repository (IFR)

- A service that provides persistent objects that represent the IDL information in a form available at runtime
- Provides type information necessary to issue requests using the DII
- Also stores additional information like debugging info , libraries of stubs or skeletons etc

27

Static and Dynamic Skeleton Interface

- Static Skeleton Interface (SSI)
  - **Similar to SII, but on server side**
  - **Knows the operation types at compile time**
  - **Performs request demarshaling and dispatching**
- Dynamic Skeleton Interface (DSI)
  - **Similar to DII, but on server side**
  - **Generic skeleton interface for all objects**

28

Object Adaptor (OA)

- Implementations must be registered with the OA
- When a client requests a service from an object, the OA maps the request to the appropriate implementation
- Activate and deactivate objects
- Objects can be implemented as C++ classes or C functions
- Allowing varied methods of implementation facilitates integration of legacy applications
- Two types – **BOA** and **POA**

29

Interoperability

- GIOP (General Interoperability Protocol)
  - **Abstract protocol for communication between different ORB products**
  - **Specifies message types**
    - **Request, Reply, LocateRequest, LocateReply, CancelRequest, CloseConnection, MessageError**
  - **Specifies data format**
    - **CDR (common data representation)**
- IIOP (Internet Inter-ORB Protocol)
  - **Mapping of GIOP over TCP/IP**
  - **IIOP - IOR contains a host name and port number as endpoint info**

30

CORBA Vendors and Applications

- CORBA vendors
  - **WUSTL TAO**
  - **IONA Orbix**
  - **Inprise Visibroker**
  - **BEA ObjectBroker**
  - **Expersoft CORBAplus**
  - **Peerlogic DAIS**
  - **OIS ORBexpress**
  - **AT&T OmniORB**
- Applications of CORBA technology
  - **Telecom**
    - **Motorola - Ground station control for IRIDIUM Global Cellular Network built on Orbix**
    - **Ericsson - TMN-based Cellular Management Operations Systems (CMOS) built using CORBA**
  - **Healthcare**
    - **Artemis - software system for sharing and managing distributed patient records. Orbix as underlying middleware**
  - **Finance**
    - **Charles Schwab - SchwabLink Web - online trading and research service uses CORBA/IIOP standards**

Next lecture:

Dr. Manimaran Govindarasu will give an invited talk on Real-time CORBA.