CORBA I
An Introduction To CORBA
CptS 464/564 Jan 23, 2003

23rd January 2003

Lecture Overview

- Object Management Group and CORBA
- CORBA Overview
- Lab use
- CORBA Example

OMG and CORBA

- OMG (The Object Management Group) is a consortium of >800 companies with common interest in interoperable enterprise applications. OMG creates and maintains vendor-neutral standards in this area. In addition to CORBA, other OMG standards include UML (Unified Modelling Language), OMA (Object Management Architecture), OMG's web site is www.omg.org.
- CORBA (Common Object Request Broker Architecture): has been evolving since 1991. Goal is to provide hardware- and language- neutral facilities for systems built from distributed objects. OMG's role in CORBA is to set the specifications. Various vendors provide implementations of those specifications. Of course vendors are free to extend the functionality. The result is that as extensions become accepted further iterations are made on the standardization process.

CORBA Features

- Transparency
  - Hardware/OS/Programming Language/Vendor neutral

        * We'll see demonstrations today of two of these: idea, applications and services need not be written for the same hardware, in the same language, or even using tools from the same vendor.
    - Location transparency

- Object oriented
    - CORBA's model is one of distributed objects. In keeping with language-neutrality, however, CORBA's objects need not be mapped 1-1 onto language objects: one language object might implement many CORBA objects. Indeed, there are CORBA bindings for non OO languages.

- Language-independent definition of *interfaces*
    - Interface gives the names and types of object properties and methods, but does not give an implementation
    - CORBA's Interface Definition Language (IDL) has defined *mappings* into many other languages (C, C++, Java, Ada, COBOL, Python, ...). Among other things this allows automatic translation of IDL specifications into declarations, and even pieces of implementation in any of these languages.

- Overall goal: interoperability.

CORBA IDL Example

```
// messenger.idl
interface Messenger {
   boolean send_message (
        in      string user_name,
        in      string subject,
        inout   string message );
};
```

- Notice: method returns a typed result; method parameters may be either sent from the client to the server (in), sent from the server to the client (out), or both (inout).

CORBA IDL Example 2

```
module BankExample {
   interface Account {
      exception BadCheck {
         float fee;
        };
      float deposit(in float amount);
```

```
      float writeCheck(in float amount)
         raises (BadCheck);
    };
   interface AccountManager {
      Account openAccount(in string name);
    };
};
```

- Notice: module – namespace control; IDL supports exceptions

CORBA Object Architecture

- CORBA clients and servers are connected together over a software bus called the Object Request Broker (ORB). The "client" and "server" roles are not exclusive: an application may simultaneously be a client of some objects and a server for others.
- See CORBA Diagram

CORBA Services

- In addition to the basic object architecture and IDL, CORBA also includes specifications for a number of services such as Naming, Transactions, Events, Interface Repositories, etc. They are (of course) specified in IDL. We will look at some of them later in the semester.

LAB Info

- Location ETRL-301 - NIF lab
- Door codes available from https://helpdesk.eecs.wsu.edu (look near the bottom right corner of the page)
- Lab machines - Red Hat Linux 7.2
  - nif-c{6,7} 550MHz PIII
  - Nif-c{10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,35,39} 650MHz PIII, 320MB - located in ETRL-301for either walk-in or remote use via SSH version 2
  - nif-c{28,29,30,36,37} 650MHz PIII, 320MB - located in EME-16 and accessible only for remote access using SSH version 2
- Lab etiquette

  - The lab is shared by several other classes. I will be displeased by any of the following
    * Leaving trash on the floor or tables - there's a wastebasket in the room, and recycling sacks just down the hall
    * Leaving processes running on the machines after you log off: this is particularly easy to do in this class because you will be building servers that you'll often put into the background. Learn to use the *ps* and *kill* commands to clean up after yourself. Finally, we've noticed ongoing problems with *vim* sessions left running after people log out. *vim* has the unfortunate behavior of consuming 100% of the available cpu cycles when this occurs. Don't leave *vim* running when you log out.
    * Unplugging a lab machine in order to plug in your own laptop. The lab is monitored and this will be detected.

- Logon with your normal EECS account. Accounts are being set up for Tri-Cities students.
- For our CORBA system we will be using ORBacus donated by IONA Technologies. It is installed in /net/niflab/orbacus. You will need to put various directories below there on various path environment variables. An easy, if inelegant way to do so is to source the /net/niflab/orbacus/README file. You might want to copy the relevant lines to your .cshrc or .bashrc file and avoid the need. (sourcing the README file produces error messages but works).

```
nif-c10% source /net/niflab/orbacus/README
```

- Reference material: The ORBacus reference manual will be available for purchase at Cougar Copies in the CUB or the TC Copy Center beginning Monday. Cost should be about $10 including binding. Alternatively you can download the PDF file from the class web site: http://www.eecs.wsu.edu/~hauser/CS564/protected/OB-4.1.0.pdf. You will be asked for a userid and password. The userid is *student* and the password is *464-564*. The manual is nearly 400 pages, so the one from the copy centers may be a bargain if you want hardcopy. I encourage you to work through all the steps of the example in the first chapter.

CORBA Example

- /net/niflab/orbacus/cs564/Messenger
- Development steps and tools
- What goes into a CORBA client and server
- Interoperability demo

Development steps and tools

- Design the distributed object
- Specify the interface using IDL
- "Compile" the IDL
  - Produces *stubs* for use by clients, *skeletons* for use in servers, common header files, and optionally a sample server implementation.
- Write client code - makes use of the object on the server
- Write server code - most of the work will be here
- Compile and run using the tools and conventions of the target language. Notice that CORBA libraries will have to be findable during these steps

Step 1 - Design the object and create the IDL

For this example we will use the Messenger interface from above. Store it in a file called Messenger.idl

Step 2 - IDL Compilation

Assuming your PATH is set correctly

```
idl --impl --no-type-codes Messenger.idl
```

will create files needed in implementing a C++ client and server. –impl is optional and tells the idl compiler to produce a sample server object. Files produced are

- Messenger.cpp
- Messenger.h
- Messenger_impl.cpp – if –impl is specified
- Messenger_impl.h – if –impl is specified
- Messenger_skel.cpp
- Messenger_skel.h

Similarly

```
jidl --impl Messenger.idl
```

creates the files needed for client and server implementation in Java. The files are

- Messenger.java
- MessengerHelper.java
- MessengerHolder.java
- Messenger_impl.java – if –impl is specified
- MessengerOperations.java
- MessengerPOA.java
- _MessengerStub.java

WARNING: with jidl, specifying –impl overwrites any existing the existing *_impl.java file. Watch out, because this is where you will have done most of the work of implementing a server. Messenger_impl.h gives the best view as to how IDL types and methods were translated to C++

Remaining steps: populate Messenger_impl.java or Messenger_impl.cpp with code to implement the functionality you desire; Create a Client main program. Create a Server main program.

In the Messenger directory I've provided a sample Makefile and Make.rules file that will take care of what is needed to create the Messenger client and server.

Client

The client main program has to (refer to Client.cpp)

- initialize the ORB
- find a server object via an Interoperable Object Reference (IOR). It then instantiates a local C++ proxy for the remote object
- make a method call on the local (C++) proxy object

Server

The server main program has to (refer to Server.cpp)

- initialize the ORB
- Create a (C++) Messenger_impl object. Note that this is not yet a CORBA object

- Register the Messenger_impl object with the ORB's Root POA (to be explained). The object is now available as a CORBA object, but nobody else knows how to find it
- Write the IOR for the object to a file (there are other, better ways to make it available, to be discussed later)
- Wait for and process client calls

Messenger object implementation

The server depends on Messenger_impl, which is quite straightforward. See Messenger_impl.cpp. Apart from inheriting from some classes provided either directly by the ORB or generated from IDL (POA_Messenger, PortableServer::RefCountServantBase), the impl code is pretty much unaware of its use to implement a CORBA object. See Messenger_impl.{h,cpp}

Actions

Before next class, make sure you can log into the NIF lab machines. Take a look at the files making up the Messenger example. Work through the Hello example in the ORBacus reference manual.