CptS 464/564 Project #1
Travel Planner

Given: Tuesday, September 7, 2003
Due: 2:50 PM Tuesday, September 21, 2003
Weight: 5% of Final Grade

Overview

In this project, you will create a simple client and two simple servers. The client will be able to communicate with each of the servers. The purpose of this project is to give you an idea of how CORBA actually works and get you familiar with ORBacus. This first project shouldn't cost you more than five or ten hours but just get you started a little bit, and get over the initial learning curve in using CORBA in general and ORBacus in particular.

Problem setting:

Design and implement a simple CORBA system, using ORBacus. We will have three kinds of objects in our system: DistanceList, Converter, and Enquirer. Please use these class/interface names.

- The Enquirer (the client) is an interactive program that prompts for a city name, sends a request for the distance to the city (starting from Pullman) to the DistanceList object which returns the distance in miles. Using the result from the DistanceList the Enquirer asks the Converter to convert this distance to the distance in km. The Enquirer then prints both distances and prompts for another city name. The Enquirer is a client program that uses CORBA objects, but is not itself a CORBA object.

- The DistanceList object (a servant) receives a request from the Enquirer for the distance to a particular city and returns the distance in miles. It has a list of the following distances: Colfax 15, Dusty 35, Spokane 75, and Oslo 5000. Of course you may add others as well.

- The Converter object (a servant) receives a request from the Enquirer with a distance given in miles and converts this distance into km and returns it.

The IDL interfaces in the Planner module are given by DistanceList.idl and Converter.idl:

```
// DistanceList.idl
module Planner {
      interface DistanceList {
            long getDistance(in string city);
      };
};

// Converter.idl
module Planner{
      interface Converter {
                        void convert(inout unsigned long
                        distance);
      };
};
```

Notice that the DistanceList servant provides its result as a return value while the Converter uses an in-out parameter to return its result.

Argument checking:
If the DistanceList object doesn't have the distance to the city that the user asked for, then it returns -1.

Expected output:

```
-- Welcome to the trip planner  --
Which city do you want to go to: Dusty
Distance to Dusty: 35 miles or 56 km.

Have a nice trip and drive safely.

-- Welcome to the trip planner  --
Which city do you want to go to: Seattle
I haven't heard of Seattle that must be a small city.
```

Additional 564 Work

Students enrolled in CptS 564 should add a user-level exception to DistanceList server instead of returning –1. The server should check if the city name passed in is one of those from the table of known cities. If it is not, then the exception should be thrown by the server. That exception should of course be caught by the client (that is, assuming you want full credit).
The exception component of the assignment is only 20% of your grade, so we strongly suggest you get the base program going first, and then add the exception. Once you get the rest going, it should at most take a few hours.

Implementation language

You can use either Java or C++ (ask about others).

Note for Java:
If you are working on your home computer with Java version 1.4 or above and ORBacus you have to do the following to run the applications.

Windows:
```
java -Xbootclasspath/p:%ORBACUS%/OB.jar;%JAVA_HOME%/lib/rt.jar
application
```

***nix/Mac:**
```
java -Xbootclasspath/p:${ORBACUS}/OB.jar:${JAVA_HOME}/lib/rt.jar
application
```

Implementation steps:

It is strongly recommended that you carefully read and study the notes of the September 2 and September 7 lectures before you start. The steps below are very much like those in the handout.
Note: if you are using java, replace `*.cpp with *.java`

1. Design and code the IDL See the above specification. The files `DistanceList.idl` and `Converter.idl` have been specified above (use them). Don't forget to fix your `PATH` and `LD LIBRARY PATH` before you start compiling. Refer to the `/net/niflab/orbacus/README`.
2. Run the IDL compiler. In pages 1 & 2 the objects are specified for you as well as the IDL files. Compiling the IDL files will result in several files being generated for use by the client and the servers. The files generated are listed in Appendix I.
3. Create the client code (Enquirer). Name the file which contains your client-side code as `EnquirerClient.cpp`. Remember that the Enquirer will need to access two servers. (Hint: you might want to use two IOR files. In the Messenger example the client, `Client.cpp`, uses only one IOR file, `Messenger.ref`).

4a. Create code to implement the C++ objects. This is where you plug in your city list and other mechanism for the given servers. The example servant implementation classes, `DistanceList_impl.cpp` and `Converter_impl.cpp`, generated by using the `-impl` option (C++ only, Java make them yourself) in the IDL compiler can be used here. You have to implement the necessary server-side functions `getDistance` and `convert` in `DistanceList_impl.cpp` and `Converter_impl.cpp` classes repectively.

4b. Create the server code (DistanceList & Converter). Here you will create the two server files. Name them as `DistanceListServer.cpp` and `ConverterServer.cpp`. The major pieces of a server file that you will need to build are listed below. (Note: Messenger examples `Server.cpp` is annotated with these steps #4B1, #4B2...)
    1. Initialize the ORB in the two server files.
    2. Create Portable Object Adaptors (POA) in both the files.
    3. Create (C++) DistanceList_impl and Converter_impl language objects

4. Instantiate CORBA objects from the language ones created above.
5. Write the IOR for the servant objects to two files.
6. Activate the POA managers (and the POA) for both the files.
7. Wait for incoming requests.

   You can gather/borrow details from the Messenger example in `/net/niflab/orbacus/cs564/Messenger` directory. The Messenger example files (C++) that are of interest to us in this directory are listed in Appendix II.

4. Compile the Code. As a starting point you can use the Makefile and Make.rules from the Messenger example. Use the existing files as a guide to what pieces go together to make up a client and server executables and how to build the pieces. Building the Makefile for this project may be the hardest part.
5. Run the code. When you have got everything right, make the client call each of the servers and print to the screen the information it gets.

Turn-ins:

The project is due at 2:50PM on Tuesday September 21. This means that all files must be stored under your EECS home directory by this time and they must remain untouched after this time. Hardcopy hand-in will occur during class on September 21st. In class you are to hand in:

1. Hardcopy of all the files you created or modified by hand. Note, this does not include files that are generated by the IDL compiler. The files that you will be turning in are listed in the Appendix III.
2. Hardcopy sample output when your program is correctly running. (That is, the client sends request to the two servers and gets responses from both). You only need to copy the output from the screen (or send it to a file) and print it out. Graduate students sample output must illustrate the raising and handling of an exception.

Do not modify or otherwise change the last accessed dates on your created files after the submission date, in case we need to check the files.

Appendix-I

The files generated when you compile the idl files are listed below.

- `DistanceList.cpp`
- `DistanceList.h`
- `DistanceList _impl.cpp`
- `DistanceList _impl.h`
- `DistanceList _skel.cpp`
- `DistanceList _skel.h`
- `Converter.cpp`
- `Converter.h`
- `Converter_impl.cpp`
- `Converter_impl.h`

- `Converter_skel.cpp`
- `Converter_skel.h`

The impl files are generated if you use the `-impl` option in the idl compiler. The `-impl` generates example servant implementation classes that you can use.

Appendix-II

The Messenger directory at `/net/niflab/orbacus/cs564/Messenger` also contains java files. The C++ files of interest to us are listed below.

- `Makefile`
- `Make.rules`
- `Messenger.idl`
- `Client.cpp`
- `Messenger.cpp`
- `Messenger.h`
- `Server.cpp`
- `Messenger_impl.cpp`
- `Messenger_impl.h`
- `Messenger_skel.cpp`
- `Messenger_skel.h`

The server and client executables are also present in this directory. However you can make them in your directory after copying the necessary files.

Appendix-III

The files to be turned in are listed below.
Note: if you are using java, replace `*.cpp with *.java`

- `Makefile`
- `Make.rules`
- `EnquirerClient.cpp`
- `DistanceListServer.cpp`
- `ConverterServer.cpp`
- `DistanceList_impl.cpp`
- `Converter_impl.cpp`

Also turn-in any additional files that you have modified or created.