CptS 464/564 Project #2

Assigned: Tuesday, February 18, 2003
Due: **Tuesday, March 11, 2003**
Weight: 5% of final grade

Objectives and Overview

The purpose of this assignment is to give you further practice with CORBA distributed programming, to gain more experience with CORBA exceptions, to gain experience with servers that use other servers, and to introduce the CORBA Naming Service. Graduate students will also investigate use of caching in a CORBA client.

Resources that you may find helpful:

- `/net/niflab/orbacus/cs564/Project2` (the IDL files for this project)
- `/net/niflab/orbacus/cs564/MessengerEx` (The messenger example with exceptions)
- `/net/niflab/orbacus/cs564/MessengerName` (The messenger example using a Name Service for IOR lookup; **look at the README file for directions on running the example**.)
- Chapters 6 and 9 of the ORBacus reference manual (For info on using the Name Service)

The Library project involves implementing a few key features of a circulation and inter-library loan system for three very(!) small public libraries. The basic idea is that a patron may query any of the libraries for any book: if it is available at any of the libraries then the patron can borrow it through inter-library loan. There are several considerations:

- We'll assume that patrons contact their "home" library. If the book is available at the home library it should be borrowed from there.
- Patrons look up availability information for many books that they don't borrow. Therefore looking up a book and borrowing it are two different actions, but both may involve inter-library communications of different kinds.
- We'll assume that the libraries have agreed on a common numbering scheme for their books, and that there is some external way of mapping from title/author/etc. information to book numbers.

```
// Library.idl
module Project2  {

  interface Library {
    typedef long BookIDType;

    typedef short Date;
      // Dates are used for due dates, and are treated abstractly.
      // A date is just the number of days from the start of
      // running of the system.

    struct BookInfo {
      BookIDType bookID;
      string bookTitle;
    };

    exception CheckedOut {
      Date dueDate;
    };

    exception NotInCollection {
      // no further data needed
      // raised when an operation is attempted on a book that
      // is not in the library's collection
    };

    exception NotMine {
      // no further data needed
      // raised when an attempt is made to check in a book to
      // the wrong library
    };

    // Methods used by clients on their local library.
    // The local library may make further calls on
    // other libraries to resolve a client's request

    boolean queryBook(in BookIDType bookID, out string libName)
            raises (NotInCollection);
      // returns true if available (i.e. checked in),
      // false if it is part of the collection of some library but not available
      // sets libName to the name of a library that has the book in its collection
      // raises NotInCollection if the book ID is not known at any library

    string checkoutBook(in BookIDType bookID) raises (NotInCollection, CheckedOut);
      // if successful, returns the library name from which the book was checked out
      // raises NotInCollection if the book is not part of any library's collection
      // raises CheckedOut if the book is in some library's collection
      // yet it is checked out everywhere
      // The due date in the exception must be the earliest due date at any library.

    void returnBook(in string libraryName, in BookIDType bookID)
         raises (NotInCollection);
      // raises NotMine if the named library doesn't own the book,
      // or its copy is not checked out

    BookInfo getBookInfo(in BookIDType bookID) raises (NotInCollection);
      // get the info for a book; raise the exception if it is not in any collection
```

```
    // Methods used by library servers to talk to one another
    // The library serving these calls never makes further calls
    // in order to resolve them. The listed exceptions are raised
    // if the call cannot be satisfied locally.

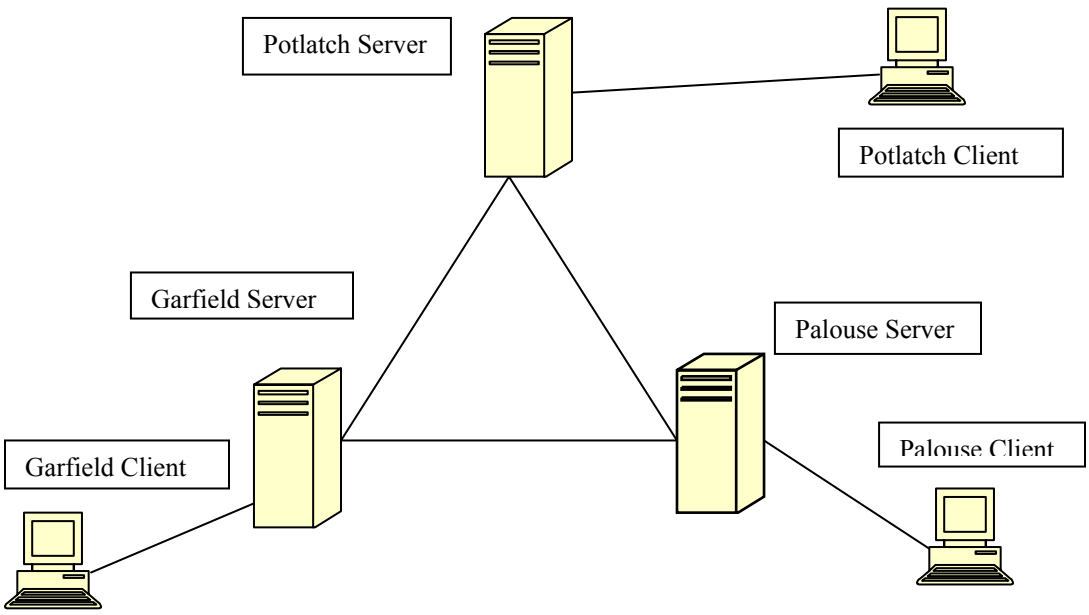
    boolean queryBookLocal(in BookIDType bookID);
      // Tests whether a book is available (i.e., checked in)
      // in the library serving the call. If so, returns true; returns false
      // if it is part of the local collection but not checked in.
      // raises NotInCollection if it is not part of the local collection

    void checkoutBookLocal(in BookIDType bookID)
         raises (NotInCollection, CheckedOut);
      // Checks out a book from the library serving the call.

    void returnBookLocal(in BookIDType bookID) raises(NotMine);
      // Returns a book to the library serving the call.

    BookInfo getBookInfoLocal(in BookIDType bookID) raises (NotInCollection);
      // Gets info for a book from the library serving the call.

  };
};
```



A typical scenario proceeds as follows: a patron looks up a book number at the local library by making a method call on the library's server. If the book is part of the collection and available for checkout the patron is told this information. Otherwise, the library server looks up the book number at each of the other libraries in turn, stopping

and returning the information that the book is available by inter-library loan if the book is available at any of the other libraries. If the book is not available the patron is informed of this fact. When the patron decides to check out a book, it makes a method call on the local library's server. The checkout will occur from the library's collection, if the book is available, otherwise the library server will try to check the book out of each of the other libraries in turn, stopping when it succeeds.

Attempting to check out a book that is not part of the collection or which is already checked out causes an exception. Looking up a book that is checked out does not cause an exception. We will discuss the reasons behind this design decision in class. Full details are in the comments in the IDL file.

The three libraries all implement the same interface as described by the following IDL:

Naming Service

In this project we introduce a more user-friendly approach to finding CORBA objects. Instead of using IOR string, we will use a name server which translates human-sensible names to object references.

A naming service is running in the nif lab for your use. It can be located by resolving an initial reference to "NameService" (see the MessengerName example). Each of your servers will need to register itself with the Naming Service using a well-known name. To avoid conflicts, use yourLoginName_libraryName as the name of each. If you do not follow this convention you may find your clients connecting to somebody else's servers – which would be fairly disastrous for both of you.

Libraries and Book Collections

The three libraries are named "Palouse", "Garfield", and "Potlatch". In `/net/niflab/orbacus/cs564/Project2` you will find files `Palouse.txt`, `Garfield.txt` and `Potlatch.txt` that contain lists of the collections of the three libraries. Your library server may read in the collection files, or you could build them into the code. You should have *one* server executable and *one* client executable. Each will take a command-line parameter designating the library name it serves, in the case of the server, and the local library name, in the case of the client. In order to make the CheckedOut exception sensible you'll need to assign different checkout periods for the different libraries. You may assume that the current date is 0 at the start of each run of your servers.

You may make the client interactive, or build in the sequence of operations it performs against the local library.

Graduate Students

During the process of looking up and deciding to check out a book, a client makes several related calls on its local library server, which causes it to make several calls on the other library servers. Some of these calls will be obviously useless based on results of previous calls. You are to identify some subset of these useless calls and design and implement a caching mechanism in your library server that eliminates them.

(Some CORBA implementations allow such caches to be implemented below the level of the generated proxy objects; ORBacus does not, so you will have to implement above the level of the proxies. Nevertheless, maintain the interface from the actual client code to the remote objects to the greatest extent possible.)

The caching is worth only 20% of the grade, so make sure that you get the base assignment working first before designing and implementing the proxies.

Simplify!

The point of this project is to work with CORBA. To that end you may make as many simplifying assumptions as you like, as long as you can meet all of the given design constraints. For example, using data structures that limits the total size of a library collection to a small number is fine.

Turn-in

Reminder: DO YOUR OWN WORK! This is ***not*** a collaborative project.

Turn in hardcopy of all the files you create or edit yourself – this does not include files completely generated by the IDL compiler. Turn in a printout showing the run-time execution of your system. The printout must show/verify every method of the interface being called in both normal and exceptional cases. It must also show each exception being raised and caught. Graduate students must illustrate that their caching code works as intended.

One method of obtaining the necessary evidence is to embed print statements in your code. For example: print out the name of each method and its arguments when each procedure is called; include print statements in exception catching code; etc.

All Pullman students will demo their systems to the TA. During your demo you will be expected to run your code and explain how it works. You must be able to explain in detail all the code you have written yourself and explain, in a general sense, code that you've copied from the example programs. Demo times will be arranged following the due date. We'll have to figure out how to do something resembling a demo for the Tri-Cities students (but you won't have to come to Pullman!).

Store all of the files associated with the project somewhere in your EECS home directory. Do not change the modification times on the files after the due date in case we want to look at them. (You do not need to ensure that the files are accessible to us unless we ask.)

Helpful Hints

1) Pay attention to string and structure storage management. Be careful not to reference data after it has gone out of scope or been deleted.
2) The naming service should be running in the lab at all times. How do your programs find it? Start them with command line switches as follows

    a. `yourprogramname –ORBInitRef NameService=corbaloc::nif-s1:26000/NameService yourprogramswitchesandarguments`

3) If for some reason the shared naming service isn't running, you can start one yourself. Please kill it when you log out. With the orbacus directories on your path (as usual)
    a. `nameserv –OAport someportnumber &`
    b. in this case change the `ORBInitRef` switch to `-ORBInitRef NameService=corbaloc::localhost:someportnumber/NameService`
4) You'll need to look carefully at the MessengerName example: additional header files are required to use the naming service, as is an additional shared library. Be sure to add the library to the final link step in your Makefile or Make.rules.