SOEN 423 Assignment 3 - Report

Nathan MacInnes 1957341

Tuesday, November 18, 2014

The following report outlines the Distributed Reservation Management System (DRMS) web service using JAX-WS in Eclipse.

The WebService

First the Library Server (*LibraryServerImplBase*) implementation was taken from Assignment 2 and a Library Server (*LibraryServer)* interface was extracted from it. @WebMethod was added above all methods that were in the interface and @WebService was added to the LibraryServer interface. @WebService(endpointInterface="libraryserver.LibraryServer") was added to the Library Server Implementation class in order to generate the WSDL file.

Next a WSDL file was generated using the following command,

```
wsgen -verbose -cp . libraryserver.LibraryServerImplBase -wsdl
```

Once the WSDL file was generated in the /bin folder of the project, a class was created in order to publish the WebService. An endpoint was created on a different port for each library server,

```
public class PublishWS {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Endpoint concordiaEndpoint = Endpoint.publish("http://localhost:7777/
libraryserver", new LibraryServerImplBase("localhost", 4441, "concordia"));
		Endpoint mcgillEndpoint = Endpoint.publish("http://localhost:7778/
libraryserver", new LibraryServerImplBase("localhost", 4442, "mcgill"));
		Endpoint queensEndpoint = Endpoint.publish("http://localhost:7779/
libraryserver", new LibraryServerImplBase("localhost", 4443, "queens"));

		System.out.println(concordiaEndpoint.isPublished());
		System.out.println(mcgillEndpoint.isPublished());
		System.out.println(queensEndpoint.isPublished());
	}
}
```

Once the class was created, the PublishWS class was run, publishing the endpoints for each server.

With the endpoints published, the corresponding WebService Client classes can be created based on the published WSDL using,

```
wsimport -keep -d . -p libraryclient http://localhost:7777/libraryserver?wsdl
```

The wsimport generates the required classes for each of the WebMethods as defined in the interface, along with the ObjectFactory and LibrarySeverImplBaseService classes that allow communication between a client and the WebService.

In order to allow access to the Library Web Service, a LibraryClientServerProxy class was created to act as an intermediary between the server client classes instantiations and the clients using the WebService.

The LibraryClientServerProxy class contains a static LibraryServer representing each library server (concordia, mcvill, queens). There are two methods contained in it for the clients to use, lookupServers() that starts the web services as defined in the LibraryServerImplBaseService class for each of the endpoints and lookupServer that returns the appropriate LibraryServer based on a String query from the Client.

Each server object contains its own UDP Server. Allowing for message passing for the getAllNonreturners() and the reserveInterLibrary() methods. The UDP Server method (udpServer) extends thread to allow for multiple instantiations and is synchronized. The synchronized udpServer() method is called in the LibraryServerImplBase classes constructor.
In the udpServer() method differentiates between which method the request is wanting to call by accepting the request DatagramPacket and checking the first character. If the character is 0 the getNonReturners() method is called. The remainder of the packet contains the method arguments, the bytes are converted as stings and are passed into the getNonReturners() method. There reply generated

is sent back with the first character of the reply representing the nature of the result being sent back.

The synchronized getAllNonReturners() method uses the UDP protocol in order to communicate between the other two library web services. The method first checks that the username and password for the administrator is correct. Once the administrator is authenticated, the byte array that will be sent to the remote servers is set up containing a String of arguments, the first character is 1, that tells the remote servers that the getNonReturners() method is being called. next is the server arguments, in our case it is the string “14”.
Next the host address is resolved as “localhost”, followed by checking the local server up port and setting the remote ports to the remaining ports.

```
if (this.udpPort == 4441) {
                            remoteServerPorts[0] = 4442;
                            remoteServerPorts[1] = 4443;
                      } else if (this.udpPort == 4442) {
                            remoteServerPorts[0] = 4441;
                            remoteServerPorts[1] = 4443;
                      } else if (this.udpPort == 4443) {
                            remoteServerPorts[0] = 4442;
                            remoteServerPorts[1] = 4441;
                      } else {
                            System.out.println("server not found");
                      }
```

Once the host is known, the arguments are known and the ports are known, we can call the remote servers. First the DatagramPacket must be created containing the arguments, the arguments length, the host, and the remote port. Then the packet is sent and a DatagramPacket is created to accept the reply. Once the reply is received it is converted to a string and added to all the results of the other servers and returned to the client.

The reserveInterLibrary() method functions using UDP in the same way that the getNonReturners functions. One difference between the two is that if the Book is found locally, or on the first server, the book is borrowed and returned to the client, instead of sending a UDP to the next server.

The Loan class on the server abstracts the action of loaning away from the LibraryCatalog and the Book object.

The create account method create a StudentAccount object that then saves the object into an array list that is saved into a hash table with the key equal to the the first

character in the username. The method fails if the account already exists or the username or password does not correspond to the criteria.
The reserveBook method allows registered students to take a book out on loan. The student authenticates with a username and password. If the user is accepted by the server the book is looked up in the BookCatalog object in a hash table with a key value pair of <“BookName”, Book> The borrow fails if the there are no book instances left in the catalog or the book does not exist in the catalog.

Client Side

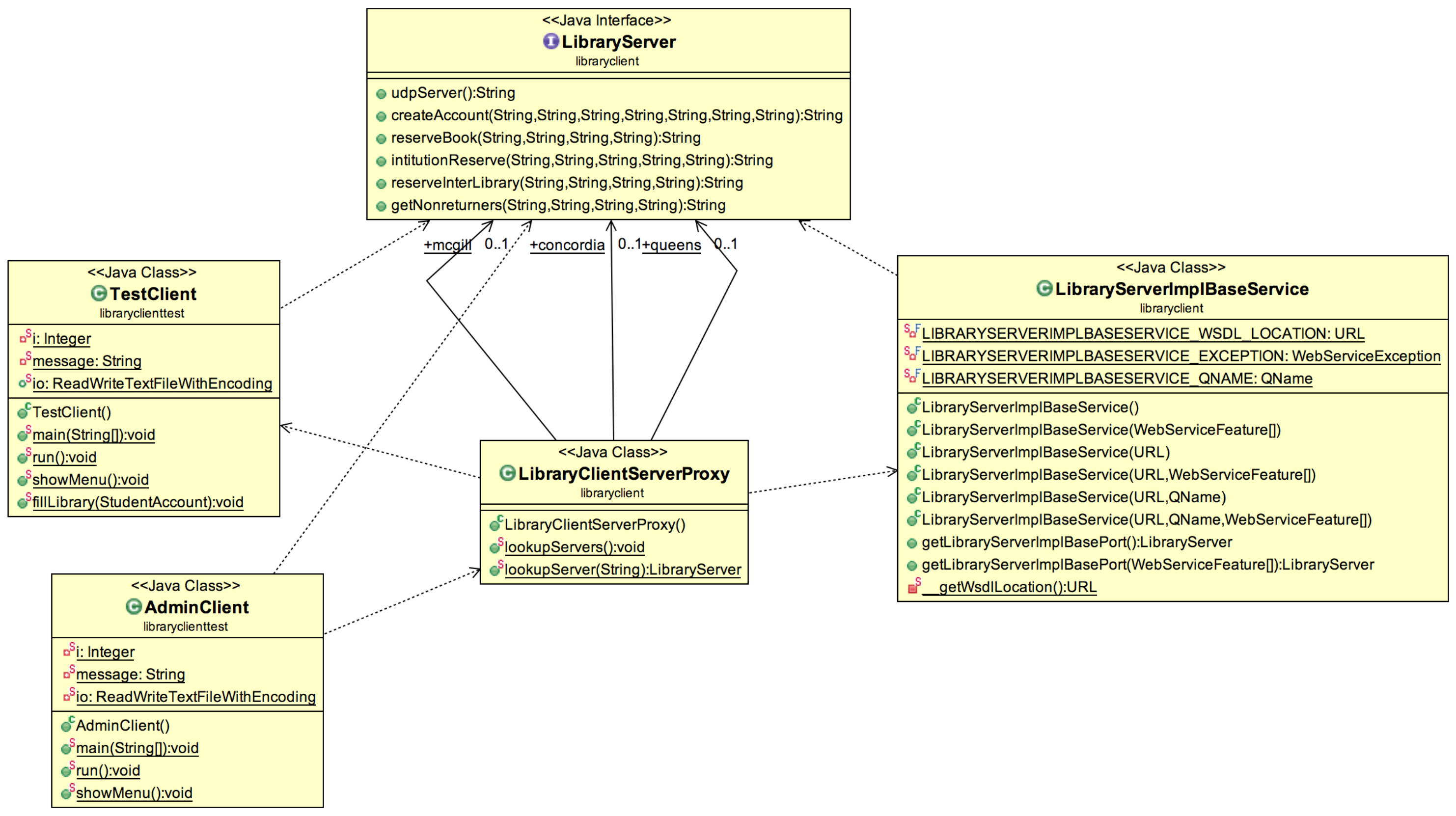
The client side provides and menu and a means for clients to communicate with the server.

A StudentClient was created to allow students to perform the createAccount method, the reserveBook method and the interLibraryLoan method. When the method is called it connects to the server corresponding to the user’s educational institution and performs the method on the web service that they request.

An AdminClient is created and connects to their respective educational institution once authenticated. They are presented with a menu that allows them to perform the getAllNonreturners() method.

For an overview of the whole system class diagram, view the image in the project src folder.

Client side communication to LibraryService interface.



<<Java Interface>>
LibraryServer
libraryclient
udpServer():String
createAccount(String,String,String,String,String,String,String):String
reserveBook(String,String,String,String):String
intitutionReserve(String,String,String,String,String):String
reserveInterLibrary(String,String,String,String):String
getNonreturners(String,String,String,String):String
+mcgill
0..1
+concordia
0..1
+queens
0..1
<<Java Class>>
TestClient
libraryclienttest
i: Integer
message: String
io: ReadWriteTextFileWithEncoding
TestClient()
main(String[]):void
run():void
showMenu():void
fillLibrary(StudentAccount):void
<<Java Class>>
LibraryClientServerProxy
libraryclient
LibraryClientServerProxy()
lookupServers():void
lookupServer(String):LibraryServer
<<Java Class>>
LibraryServerImplBaseService
libraryclient
LIBRARYSERVERIMPLBASESERVICE_WSDL_LOCATION: URL
LIBRARYSERVERIMPLBASESERVICE_EXCEPTION: WebServiceException
LIBRARYSERVERIMPLBASESERVICE_QNAME: QName
LibraryServerImplBaseService()
LibraryServerImplBaseService(WebServiceFeature[])
LibraryServerImplBaseService(URL)
LibraryServerImplBaseService(URL,WebServiceFeature[])
LibraryServerImplBaseService(URL,QName)
LibraryServerImplBaseService(URL,QName,WebServiceFeature[])
getLibraryServerImplBasePort():LibraryServer
getLibraryServerImplBasePort(WebServiceFeature[]):LibraryServer
__getWsdlLocation():URL
<<Java Class>>
AdminClient
libraryclienttest
i: Integer
message: String
io: ReadWriteTextFileWithEncoding
AdminClient()
main(String[]):void
run():void
showMenu():void