**Design Documentation**

**For**

**<COMP 6231/2>**

Prepared by

<40040721 Luguang Liu>

<40052560 Zhongxu Huang>

<40024460 Chenwei Song>

<40025138 Jingye Hou>

## CORBA Architecture

## IDL with CORBA

To create a complete CORBA application, we use IDL (Interface Definition Language) to define interface which includes six methods (*createStudentRecord, createTeacherRecord, getLocalCount, getServerCount, editRecord, transferRecord*). Then use Java IDL complier to generate stubs and skeletons. After the compiler process, there will be six classes generated automatically as shown below.

** **

Implementation

Compile



Then we create RecordManagerImpl class to extend abstract class RecordManagerPOA which we write the implementation methods for the interface.

## Server Deployment in CORBA

In this assignment, we have three servers need to be bind to the CORBA object. In each server, we use function “init” to create pseudo-object. Then we use resolve\_initial\_reference to find POA root and activate it. We use servant\_to\_reference to register CORBA object. Finally, we use CORBA naming service to bind three servers to each *NameComponent* under the same naming context as shown below.



## Client Deployment in CORBA

In client side, we create CORBA pseudo-object by using init function and use CORBA naming service to find the server that it belongs to. Then we assign this server object to remote. Finally, we can use remote to execute the five functions that we have built in idl. The code is shown below:



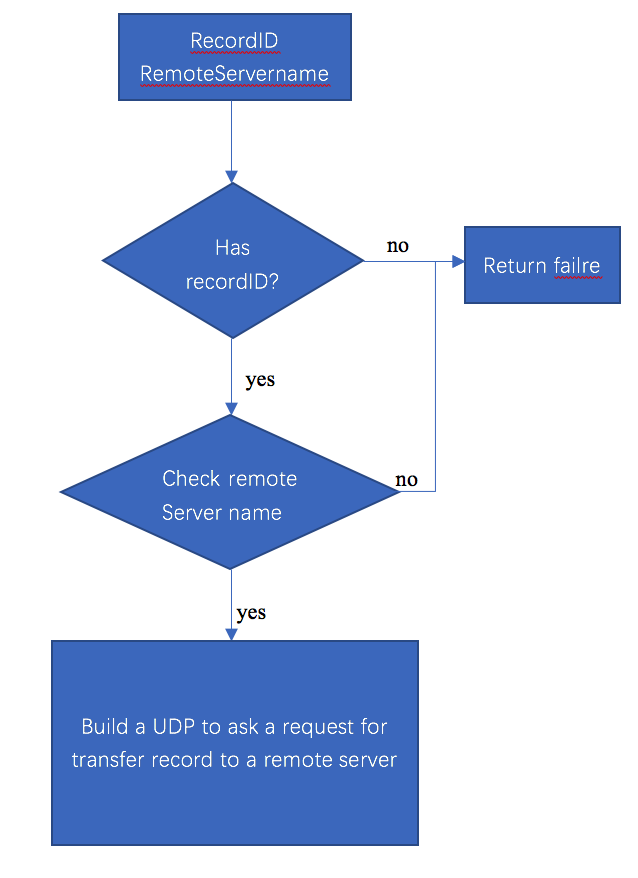
## The logic for implementing the function transferRecord

After a server receives tranferRecord requirement, it will do several actions as below:

First, it will check if it contains the record which is asked to be transferred to another server. For example, if the server “MTL” gets a transfer a record whose id is “SR10003” to the other remote server “LVL”, the server will iterate all its record and checks weather it contains the record or not.

Second, to make sure that the remote server name is not the local server name, we check if the local server name equals remote server name we will continue the next action, else we will return wrong message.

Third, if all the information above is right, we will start to build a UDP request to ask remote server to transfer record data. In the remote server side, there is a while loop to get a request from other side. If the request is to transfer record, it will check if the record is already there. If not, the server will create the record and then tell the server which sent the request that the record is transferred successfully. Finally, the server who sent the request will delete the transferred record in its list and tell client the message that the record is transferred successfully. Else, the remote server will tell the server who sent the request that you cannot transfer this record and the request server will tell client that reason that transferring record is failure.

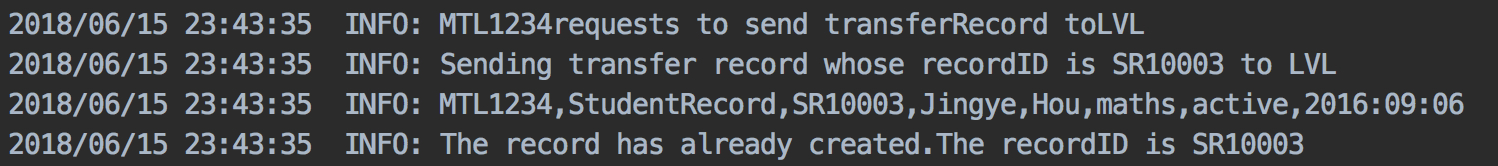


## Test case and Test scenarios

Scenario1. One client requests transferring record in the condition that the local server has the record and remote server does not has the record.

Expected: The action should be successful.

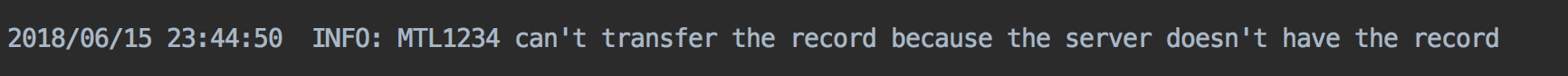
Test result:



Scenario2. One client requests transferring record in the condition that the local server does not have the record.

Expected: The request for transferring record should be rejected.

Test result:

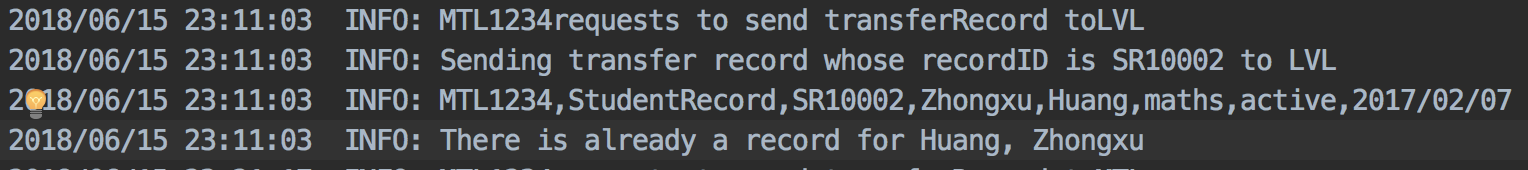


Scenario3. One client requests transferring record in the condition that the remote server has the same record.

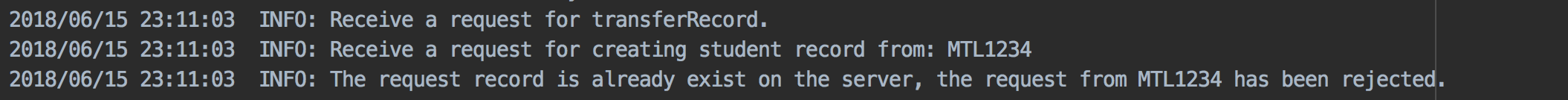
Expected: The request for transferring record should be rejected.

Test result:

In the request sender server:



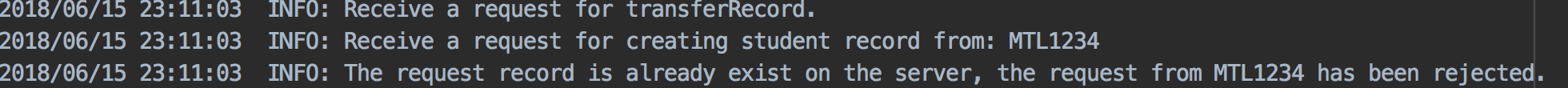
In the request receive server:



Scenario4. One client requests transferring Record in the condition that the remote server name is local server name.

Expected: The request for transferring record should be rejected.

Test result:



Scenario5. Multiple client request including editRecord, transferRecord and createRecord

REQUEST TRANSFER SR10003

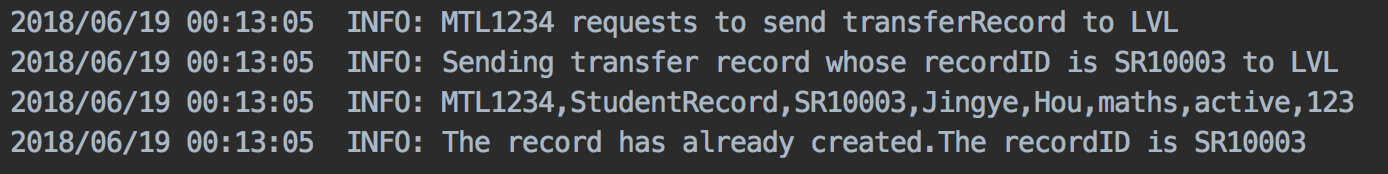
REQUEST EDIT SR10003

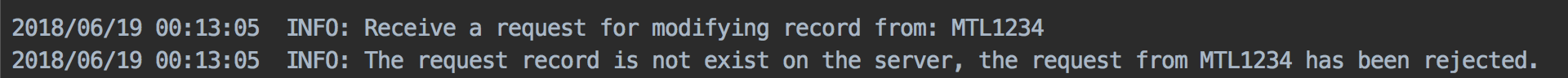
MTL SERVER

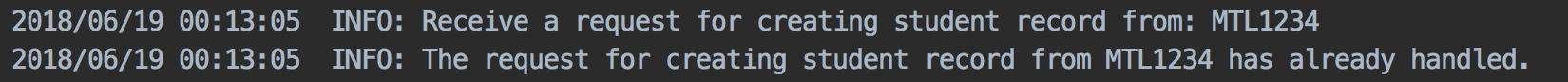
REQUEST CREATE SR10003

Expected: The 1st request for transfer should be accepted if the remote server don’t have the same record. Thus, the 2nd request for editing should be rejected since the record has been transferred. And the 3rd request should be accepted.

Test result:







Scenario6. Multiple client request including editRecord, transferRecord and createRecord

REQUEST EDIT SR10003

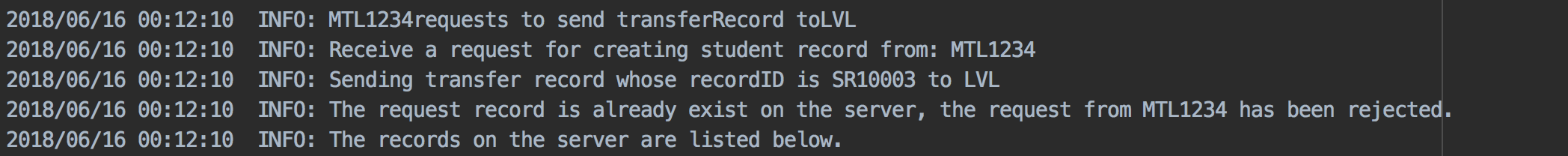
MTL SERVER

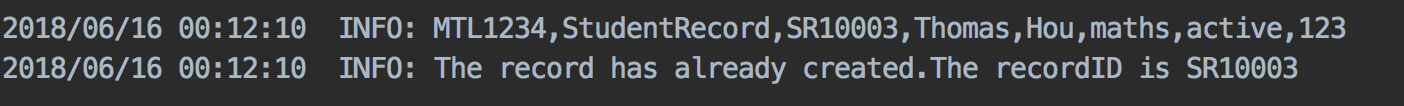
REQUEST TRANSFER SR10003

REQUEST CREATE SR10003

Expected: All the request will be accepted if the remote server don’t have the same record.

Test result: 





## Most difficult part and important part in this assignment

Difficult part: When we implement the CORBA and set up servers, we did not know how many ports should we create. Finally, we found a solution for it. We use naming server to create three different path for the three servers.

The important part is that we need to know the architecture of the CORBA and how it works. Secondly, we need to think of synchronization when we check that if the record is listed in the local server record list and delete record from the record list.

## Data structures, class and main method in the project

1. Database: HashMap <String, List<components>>

Key: String(A-Z) the first letter of last name

Value : Teacher: firstName, lastName, address, phone, specialization, location

Student: id, firstName, lastName, courseRegistered, status, statusDate

1. Method: createTRecord (String, String, String, String, String, String, String)

Attribute: managerID, firstName, lastName, address, phone, specialization, location

1. Class: teacherRecord (String, String, String, String, String, String)

Attribute:id, firstName, lastName, address, phone, specialization, location

Function: Getter and Setter methods, and editField()

1. Method: createSRecord (String, String, String, String, String,String)

Attribute: managerID, firstName, lastName, courseRegistered, status, statusDate

1. Class: StudentRecord(String, String, String, String,String,String)

Attribute: id, firstName, lastName, courseRegistered, status, statusDate

Function: Getter and Setter methods, and editField()

1. Class: Record(String, String, String)

Attribute: id, firstName, lastName

Function: Getter and Setter methods, and editField()

1. Method: GetCount()
2. Method: ServergetCount()
3. Method: transferRecord(String, String, String)

Atrribute: managerID, recordID, remoteServerName