DISTRIBUTED CLASS MANAGEMENT SYSTEM DESIGN DOCUMENTATION

version 2.0

COMP6231 | Team 7

Contents

1 Techniques, System Architecture and Data Structure

This section will illustrate the significant technique used in this system, the architecture and data structure of this system via UML class diagram.

1.1 Techniques

1.1.1 CORBA

In this system, we applied CORBA architecture to allow distributed objects to interoperate in a heterogeneous environment and support remote communication. We used Java IDL (explanation on section 1.1.2 Java IDL) to implement CORBA. The following diagram *(figure 1.)* is an illustration of the auto-generation of the infrastructure code from an interface defined using the CORBA IDL.

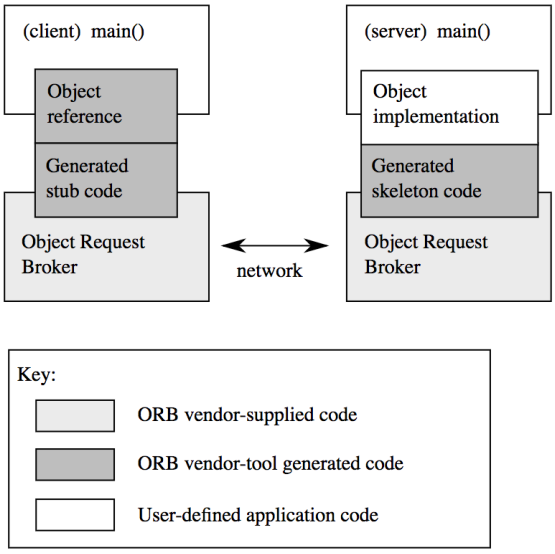

figure 1.

1.1.2 Java IDL

Java IDL is an implementation of CORBA. Its facility includes a CORBA ORB, an IDL-to-Java compiler, and a subset of CORBA standard services. In this system, we use the IDL complier to generate the IDL files automatically. The following diagram *(figure 2.)* briefly describes the relationships among significant components in the system.

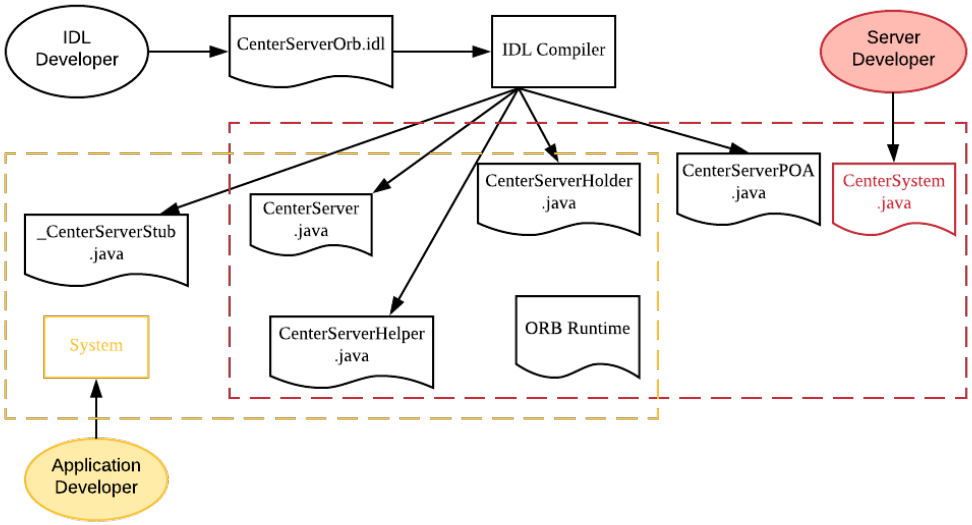

figure 2. IDL-to-Java Translation

1.2 System Architecture

This diagram *(figure 3.)* indicate the architecture of this system. We implemented the new technique and method based on the former version of this system. The interface ***CenterServer*** was moved to the package ***CenterServerOrb*** where there are all of the IDL files of this system. The new method ***transferRecord(managerID, recordID, remoteCenterServerName)*** was defined in interface ***CenterServerOperations***, which is in the package ***CenterServerOrb***, and implemented in servers.

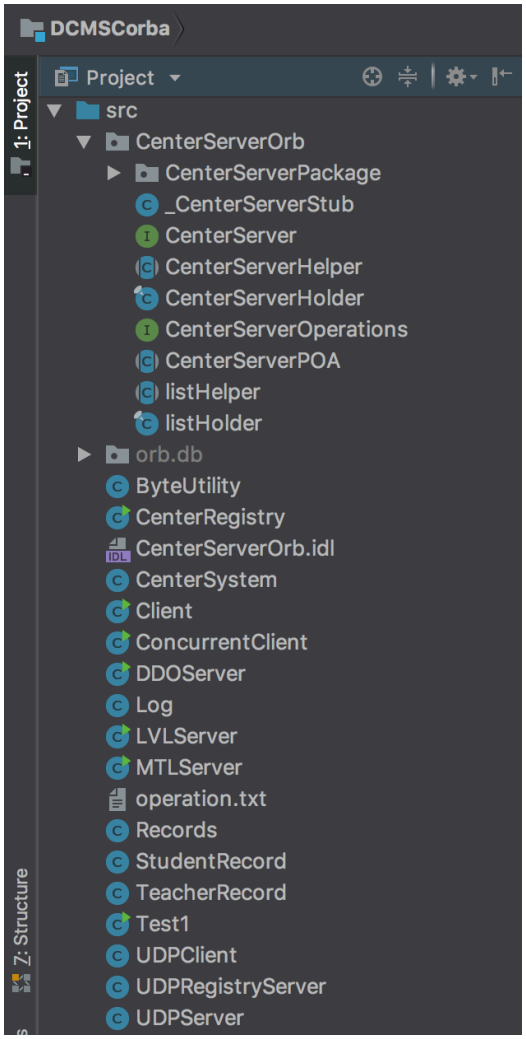

figure 3. System Architecture

1.3 Class Diagram

The following diagram *(figure 4.)* shows the implementations of this system via UML class diagram with relations of basic inheritances and implementations.

We can clearly read from the class diagram that there is an interface called ***CenterServerOperations***, which is in the package of IDL files, is extended by an interface ***CenterServer*** and implemented by ***CenterServerPOA*** that is in the IDL files package as well. The class ***_CenterServerStub*** implements interface ***CenterServer***. The class ***CenterSystem*** implements the methods of operations and extends the class ***CenterServerPOA***.

Besides, the relations among other classes remain the same as the old version of this system.

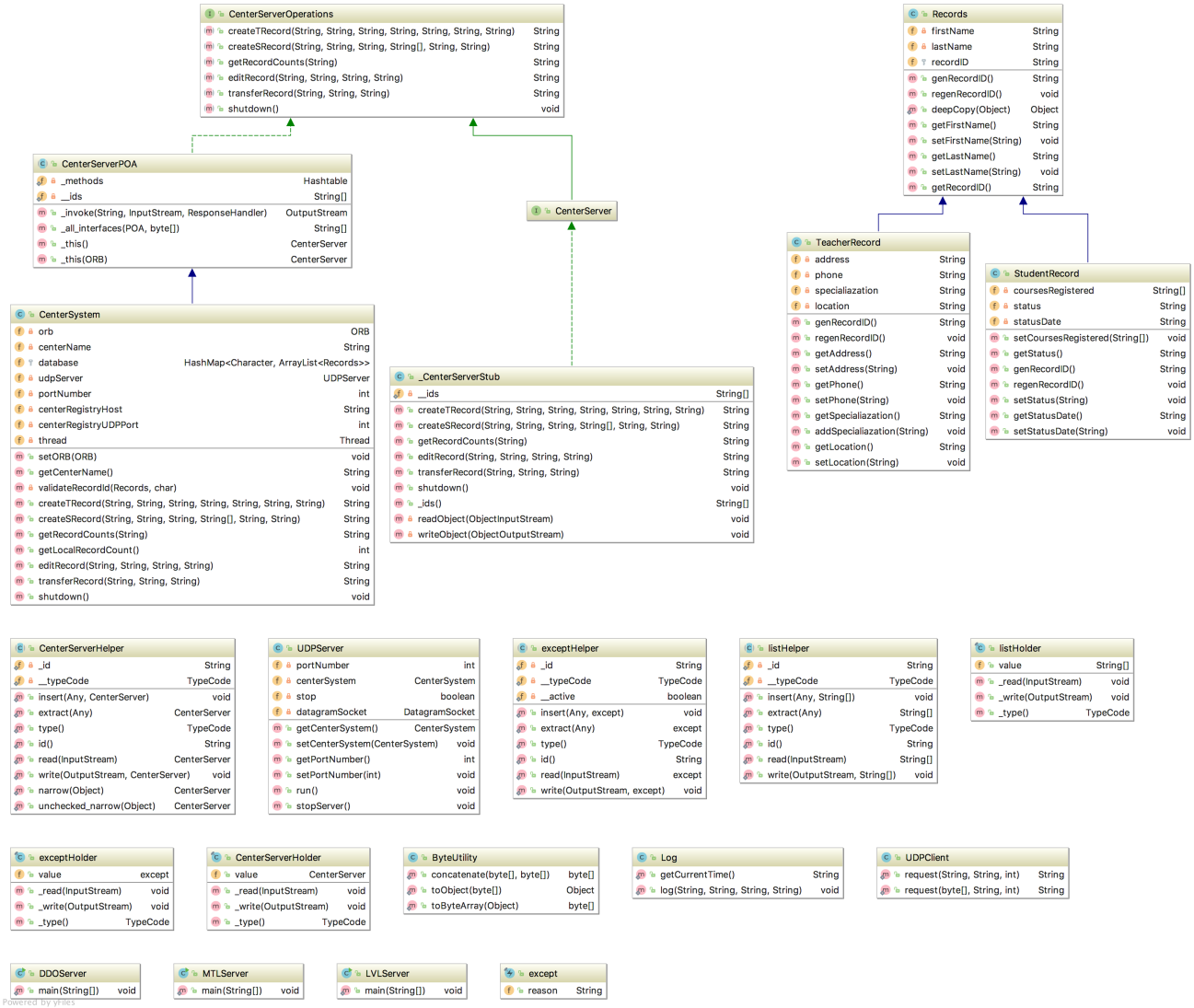

figure 4. Class diagram

2 Test Scenario

This section will describe some test scenarios that cover the operation of the new added method and the execution result shown on logs, which include: i) successfully transfer a record to a remote server; ii) fail to transfer a record in a server to another remote server in several situations; iii) edit an existed record and do the transfer operation simultaneously.

2.1 Execute and start orbd and servers

Start the ***orbd*** before executing any of the others and then the servers. Results shown below.

```
 centerRegistry is waiting for servers requests
press stop to shut down!
received data: register:MTL:Penelopes-Mac.local:8180
request ops: register
MTL successfully registered
received data: register:LVL:Penelopes-Mac.local:8181
request ops: register
LVL successfully registered
received data: register:DDO:Penelopes-Mac.local:8182
request ops: register
DDO successfully registered
```

```
MTL is launched
input s to shut down!
```

```
LVL is launched
input s to shut down!
```

```
DDO is launched
input s to shut down!
```

(output)

2.2 Concurrently create seeding data

Execute the class ***ConcurrentClient*** to create several records in different servers concurrently as the seeding data in our project, then, the system will generate logs for every manager. Test data and output, as well as logs are shown as below.

```
MTL0000|createTRecord|firstName|lastName|address|specialiazation|location|phone
MTL0002|createTRecord|firstName|lastNae|address|specialiazation|location|phone
MTL0003|createTRecord|firsName|lastName|address|specialiazation|location|phone
LVL0001|createSRecord|firstName|lastName|status|statusDate|comp6231 comp5411
LVL0000|getRecordCounts
```

(Test data)

```
Create Successful, recordId is TR50676
Create Successful, recordId is TR46340
Create Successful, recordId is TR00455
Create Successful, recordId is SR63107
Record number is LVL:1 MTL:3 DDO:0
```

(output)

```
time | managerId | operation | result
2018-06-15 01:14:21 | LVL0001 | createSRecord | Create successfully! Record ID is SR63107
2018-06-15 01:14:21 | LVL0000 | getRecordCounts | Successful
```

(log LVL.txt)

```
time | operation | location | result
2018-06-15 01:14:21 | getRecordCounts | LVLServer | Successful
```

(log LVL0000.txt)

```
time | operation | location | result
2018-06-15 01:14:21 | createSRecord | LVLServer | Create successfully! Record ID is SR63107
```

(log LVL0001.txt)

```
time | managerId | operation | result
2018-06-15 01:14:21 | MTL0002 | createTRecord | Create successfully! Record ID is TR46340
2018-06-15 01:14:21 | MTL0000 | createTRecord | Create successfully! Record ID is TR50676
2018-06-15 01:14:21 | MTL0003 | createTRecord | Create successfully! Record ID is TR00455
```

(log MTL.txt)

```
time | operation | location | result
2018-06-15 01:14:21 | createTRecord | MTLServer | Create successfully! Record ID is TR50676
```

(log MTL0000.txt)

```
time | operation | location | result
2018-06-15 01:14:21 | createTRecord | MTLServer | Create successfully! Record ID is TR46340
```

(log MTL0002.txt)

```
time | operation | location | result
2018-06-15 01:14:21 | createTRecord | MTLServer | Create successfully! Record ID is TR00455
```

(log MTL0003.txt)

2.3 Transfer a record to a remote server

At first, execute the class ***Client*** for further operations. Then enter 5 to do the operation of transferring the record. Enter the record id indicated in the correct log file and the record can be successfully transfer from its original server to a remote server. Output and logs are shown as following.

```
Please input your manager ID:
MTL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
5
Please input the transfer record id:
TR50676
Please input the destination to transfer:
DDO
TR50676 is stored in the DDO | TR50676 is removed from MTL
```

(output)

```
2018-06-15 03:40:21 | MTL0000 | transferRecord:TR50676 | TR50676 is stored in the DDO | TR50676 is removed from MTL
```

(log MTL.txt)

```
2018-06-15 03:40:21 | transferRecord:TR50676 | MTLServer | TR50676 is stored in the DDO | TR50676 is removed from MTL
```

(log MTL0000.txt)

2.4 Transfer a non-existed record

This time we enter a non-existed record under the operations of a valid manager ID, where it supposes to report a failure of no such record ID for the manager. Output and logs are shown as following.

```
Please input your manager ID:
LVL0001
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
5
Please input the transfer record id:
TR12345
Please input the destination to transfer:
MTL
No such record Id for this manager
```

(output)

```
2018-06-15 03:44:02 | LVL0001 | tranferRecord:TR12345 | No such record Id for this manager
```

(log LVL.txt)

```
2018-06-15 03:44:02 | tranferRecord:TR12345 | LVLServer | No such record Id for this manager
```

(log LVL0001.txt)

2.5 Transfer to wrong server

In this operation, entering a non-existed server other than MTL, LVL, or DDO and we expect there should be a report of failure because of the system cannot get the correct server. Output and logs are shown as following.

```
Please input your manager ID:
MTL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
5
Please input the transfer record id:
TR24305
Please input the destination to transfer:
TRT
 No such Center to transfer
```

(output)

```
2018-06-15 17:26:21 | MTL0000 | tranferRecord:TR24305 |  No such Center to transfer
```

(log MTL.txt)

```
2018-06-15 17:26:21 | tranferRecord:TR24305 | MTLServer |  No such Center to transfer
```

(log MTL0000.txt)

2.6 Transfer from wrong manager ID

In this operation, entering a correct manager ID and a correct record ID, but they are not matched with each other. We expect there should be a report of failure because of the manager and the record do not match. Output and logs are shown as following.

```
Please input your manager ID:
DDO0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
5
Please input the transfer record id:
TR00455
Please input the destination to transfer:
LVL
No such record Id for this manager
```

(output)

```
time | managerId | operation | result
2018-06-15 03:58:43 | DDO0000 | tranferRecord:TR00455 | No such record Id for this manager
```
(log DDO.txt)

```
time | operation | location | result
2018-06-15 03:58:43 | tranferRecord:TR00455 | DDOServer | No such record Id for this manager
```
(log DDO0000.txt)

2.7 Edit a record and transfer the same one at the same time

2.7.1 Ensure operation works individually

At last, we try to test the concurrency of several clients operate with a same record.
To ensure that the edit operation works well, we test this operation firstly. The following pictures show that editing function works as normal and, according to section 2.3, the transfer operation works as well, then we can test the concurrency of this system. Output and logs are shown as following.

```
Please input your manager ID:
LVL0001
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
4
Please input your record id:
TR00455
Please input the field name you want to change:
phone
Please input new value:
123456
Record updated
```
(client1 output)

```
2018-06-15 04:12:12 | LVL0001 | edit: phone | Record updated
```
(log LVL.txt)

```
2018-06-15 04:12:12 | edit: phone | LVLServer | Record updated
```
(log LVL0001.txt)

2.7.2 Test concurrency

At first, we enter the edit operation via client1, and stop following steps before editing. Output shown below.

```
Please input your manager ID:
LVL0001
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
4
Please input your record id:
TR00455
Please input the field name you want to change:
```
(client1 output)

Then we execute Client one more time as client2.
Finish the operations which are the same as those in section 2.3 in order to successfully transfer the same record as that handled by client1 from its original server to a remote server. Output is shown as following client2 output, where we can see that it is successful.
After the operations of transfer this record, let us get back to client1 and try to finish the remaining steps of editing. Here it supposes to reports a failure that it cannot find the record because it has been transferred to another server. Output and logs are shown as following.

```
Please input your manager ID:
LVL0001
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Exit.
5
Please input the transfer record id:
TR00455
Please input the destination to transfer:
MTL
TR00455 is stored in the MTL | TR00455 is removed from LVL
```
(client2 output)

```
Please input the field name you want to change:
phone
Please input new value:
999999
No such record Id for this manager
```
(client1 output)

```
2018-06-15 04:13:16 | LVL0001 | transferRecord:TR00455 | TR00455 is stored in the MTL | TR00455 is removed from LVL
2018-06-15 04:13:39 | LVL0001 | edit: phone | No such record Id for this manager
```
(log LVL.txt)

```
2018-06-15 04:13:16 | transferRecord:TR00455 | LVLServer | TR00455 is stored in the MTL | TR00455 is removed from LVL
2018-06-15 04:13:39 | edit: phone | LVLServer | No such record Id for this manager
```
(log LVL0001.txt)

3 Important and Difficult Parts

In this assignment, we mainly realize "***transferRecord***" function. As usual, we put it in ***CenterSystem***. In the realization process, we should consider several significant points.

3.1 The order of transferring and removing operation

In our project, we decide to find the record and transfer it first. Then, if the transfer process success, we will remove it in the hash map of original server and write log. In this way, we avoid the terrible case that record is removed from database but the transfer fails.

3.2 How to store the record in the hash-map of destination

In our project, we still use UDP to send the operation request. We bind one UDP server with each server to receive request. When a UDP server receive the request of transfer, it will invoke the database(hash-map) of relevant server and store the record in it.

3.3 How to ensure the database of a server only being modified by one process

In this function, we both synchronize the "sender" server's database and the "receiver" server's database so that when this process start, the databases of these servers will not mess up with other server's databases.

4 Reference

[1]. https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture#/media/File:Orb.svg
[2]. https://pdfs.semanticscholar.org/presentation/8c00/e30bf77d8f90d820aca488aba3a22e78f222.pdf