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.

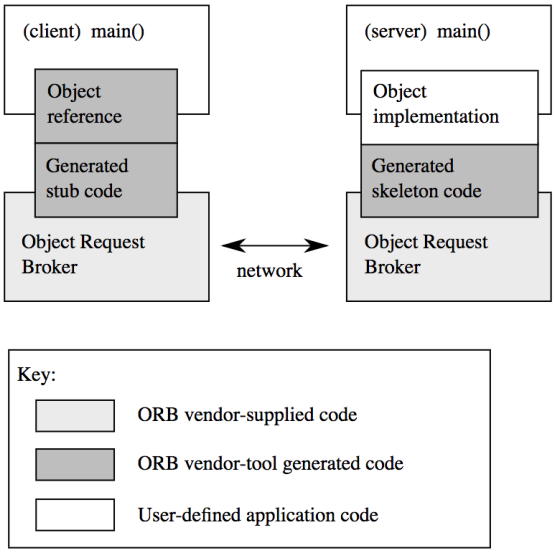


figure 1.

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.

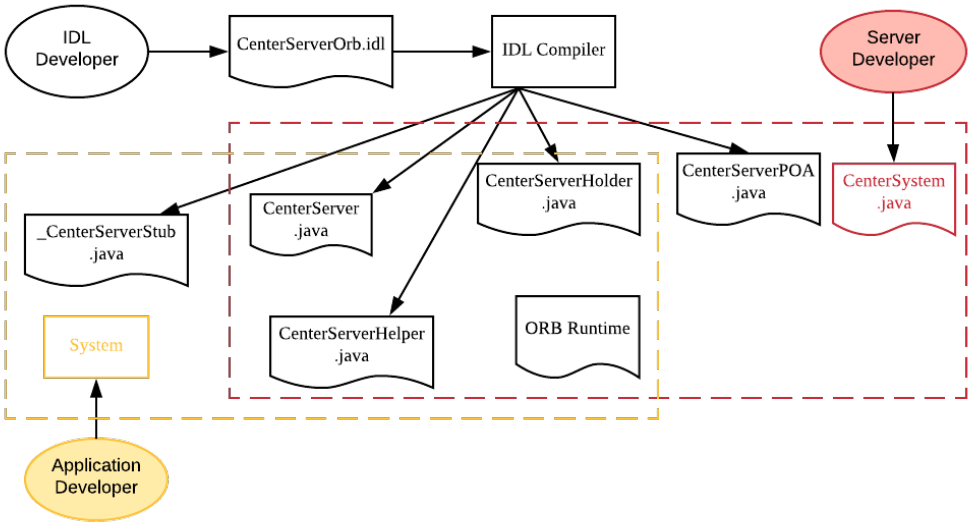


figure 2. IDL-to-Java Translation

1.2 System Architecture

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.
There is a method named ***transferRecord(stub: CenterServer, managerId: String)*** in class Client. However, this method is different from the one above and with different method signature. It plays a role that providing a prompt for users to choose the transfer operation and when this method is called by client, it will trigger the method ***transferRecord(managerID, recordID, remoteCenterServerName)*** who has actual industrial function in order to finish the transfer operation.

1.3 Class Diagram

The following diagram *(figure 3.)* 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.

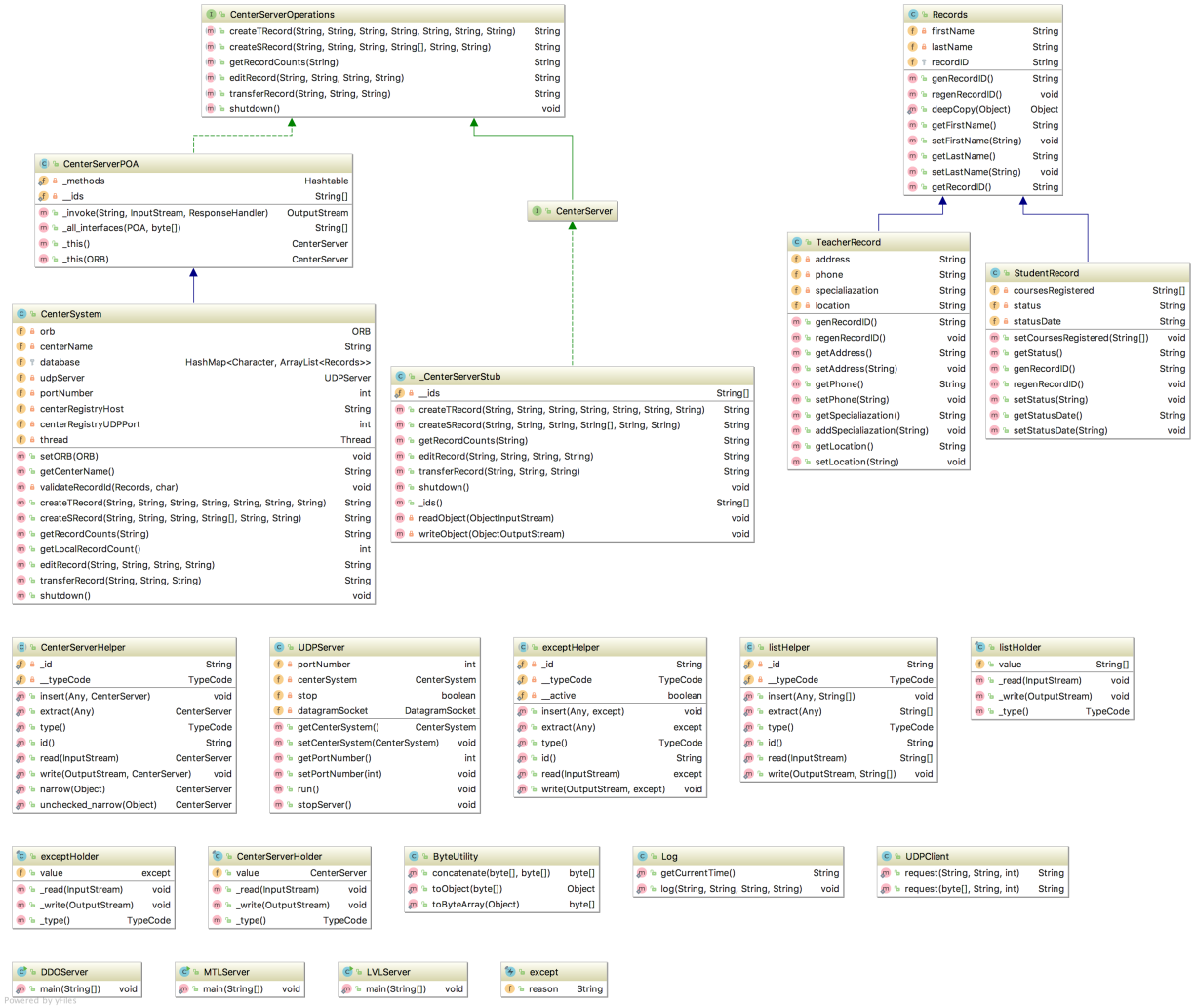


figure 3. Class diagram

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 ***Client*** 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 TR00065
Create Successful, recordId is TR18787
Create Successful, recordId is TR46543
Create Successful, recordId is SR44533
Record number is LVL:1 MTL:3 DDO:0
```

(output)

```
2018-06-17 20:32:37 | LVL0001 | createSRecord | Create successfully! Record ID is SR44533
2018-06-17 20:32:37 | LVL0000 | getRecordCounts | Successful
```

(log LVL.txt)

```
2018-06-17 20:32:37 | getRecordCounts | LVLServer | Successful
```

(log LVL0000.txt)

```
2018-06-17 20:32:37 | createSRecord | LVLServer | Create successfully! Record ID is SR44533
```

(log LVL0001.txt)

```
2018-06-17 20:32:37 | MTL0002 | createTRecord | Create successfully! Record ID is TR00065
2018-06-17 20:32:37 | MTL0000 | createTRecord | Create successfully! Record ID is TR18787
2018-06-17 20:32:37 | MTL0003 | createTRecord | Create successfully! Record ID is TR46543
```

(log MTL.txt)

```
2018-06-17 20:32:37 | createTRecord | MTLServer | Create successfully! Record ID is TR18787
```

(log MTL0000.txt)

```
2018-06-17 20:32:37 | createTRecord | MTLServer | Create successfully! Record ID is TR00065
```

(log MTL0002.txt)

```
2018-06-17 20:32:37 | createTRecord | MTLServer | Create successfully! Record ID is TR46543
```

(log MTL0003.txt)

2.3 Transfer a record to a remote server

Based on the former operations and enter a valid manager ID, 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.

```
MTL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Test concurrently edit and transfer the same record.
7> Exit.
5
Please input the transfer record id:
TR00065
Please input the destination to transfer:
LVL
TR00065 is stored in the LVL | TR00065 is removed from MTL
```
(output)

```
2018-06-17 20:57:39 | MTL0000 | transferRecord:TR00065 | TR00065 is stored in the LVL | TR00065 is removed from MTL
```
(log MTL.txt)

```
2018-06-17 20:57:39 | transferRecord:TR00065 | MTLServer | TR00065 is stored in the LVL | TR00065 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 supposed 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> Test concurrently edit and transfer the same record.
7> Exit.
5
Please input the transfer record id:
TR12345
Please input the destination to transfer:
MTL
No such record Id for this manager
```
(output)

```
2018-06-17 20:59:48 | LVL0001 | tranferRecord:TR12345 | No such record Id for this manager
```
(log LVL.txt)

```
2018-06-17 20:59:48 | tranferRecord:TR12345 | LVLServer | No such record Id for this manager
```
(log LVL0001.txt)

2.5 Transfer an invalid record

In our project, the record ID generated by this system should have a prefix as identifier of teacher record with ***TR*** or student record with ***SR*** and a following 5 digits of random numbers. This time we enter an invalid record with 6 following digits under the operations of a valid manager ID, where it supposed to report a failure of no such record ID for the manager. 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> Test concurrently edit and transfer the same record.
7> Exit.
5
Please input the transfer record id:
TR123456
Please input the destination to transfer:
LVL
No such record Id for this manager
```
(output)

```
2018-06-17 21:01:28 | DDO0000 | tranferRecord:TR123456 | No such record Id for this manager
```
(log DDO.txt)

```
2018-06-17 21:01:28 | tranferRecord:TR123456 | DDOServer | No such record Id for this manager
```
(log DDO0001.txt)

2.6 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> Test concurrently edit and transfer the same record.
7> Exit.
5
Please input the transfer record id:
TR46543
Please input the destination to transfer:
TRT
 No such Center to transfer
```

(output)

```
2018-06-17 21:06:07 | MTL0000 | tranferRecord:TR46543 |  No such Center to transfer
```

(log MTL.txt)

```
2018-06-17 21:06:07 | tranferRecord:TR46543 | MTLServer |  No such Center to transfer
```

(log MTL0000.txt)

2.7 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> Test concurrently edit and transfer the same record.
7> Exit.
5
Please input the transfer record id:
TR46543
Please input the destination to transfer:
LVL
No such record Id for this manager
```

(output)

```
2018-06-17 21:08:20 | DDO0000 | tranferRecord:TR46543 | No such record Id for this manager
```

(log DDO.txt)

```
2018-06-17 21:08:20 | tranferRecord:TR46543 | DDOServer | No such record Id for this manager
```

(log DDO0000.txt)

2.8 Concurrently edit and transfer a same record

There is an option for the client to test concurrency of editing and transferring a same record at the same time. Enter 6 to execute the test function defined in ***Client***. This function will trigger two threads for editing and transferring respectively. In this test scenario, the testing order of these two operations is firstly edit then do the transfer operation. This option of operation supposed to both successfully edit and transfer the same record and report results of these two operations. Output and logs are shown as following.

```
Please input your manager ID:
LVL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Test concurrently edit and transfer the same record.
7> Exit.
6
Please input your manager ID:
Record updated
SR44533 is stored in the DDO | SR44533 is removed from LVL
```

(output)

```
2018-06-17 20:35:04 | LVL0000 | transferRecord:SR44533 | SR44533 is stored in the DDO | SR44533 is removed from LVL
2018-06-17 20:35:04 | LVL0000 | edit: firstName | Record updated
```

(log LVL.txt)

```
2018-06-17 20:35:04 | transferRecord:SR44533 | LVLServer | SR44533 is stored in the DDO | SR44533 is removed from LVL
2018-06-17 20:35:04 | edit: firstName | LVLServer | Record updated
```

(log LVL0000.txt)

2.9 Fail to concurrently edit while transfer a same record

In this test scenario, we will test the same operations as 2.8 but in a different order of operations. I changed the order that firstly transfer the record then edit the same one, where there is expected an error reported in this system due to no such record ID. Output and logs are shown as following.

```
Please input your manager ID:
LVL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Test concurrently edit and transfer the same record.
7> Exit.
6
Please input your manager ID:
SR88141 is stored in the DDO | SR88141 is removed from LVL
No such record Id for this manager
```

(output)

```
2018-06-18 23:44:22 | LVL0000 | transferRecord:SR88141 | SR88141 is stored in the DDO | SR88141 is removed from LVL
2018-06-18 23:44:22 | LVL0000 | edit: firstName | No such record Id for this manager
```

(log LVL.txt)

```
2018-06-18 23:44:22 | transferRecord:SR88141 | LVLServer | SR88141 is stored in the DDO | SR88141 is removed from LVL
2018-06-18 23:44:22 | edit: firstName | LVLServer | No such record Id for this manager
```

(log LVL0000.txt)

2.10 Server is stopped unexpectedly

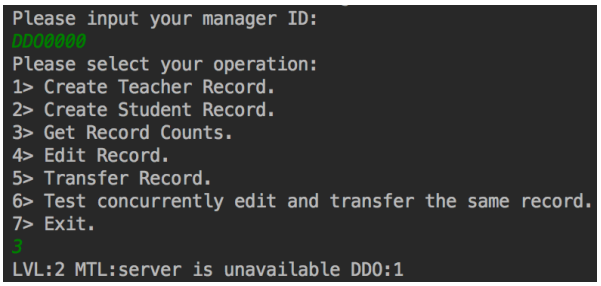
At last, we try to force stop a server and execute some communication among the servers and client. In this test, I force stopped the MTL server, and then try to do the operation of getting the record counts which will trigger the expected communication. We expect that, firstly, the client can report an error of unavailable server and, then, the working server can report an order of processing with itself as the first in the sequence and the closed server as the last in that. Here are the outputs of tests on both LVL server and DDO server after force stopping the MTL server.

```
Please input your manager ID:
LVL0000
Please select your operation:
1> Create Teacher Record.
2> Create Student Record.
3> Get Record Counts.
4> Edit Record.
5> Transfer Record.
6> Test concurrently edit and transfer the same record.
7> Exit.
3
LVL:2 MTL:server is unavailable DDO:1
```

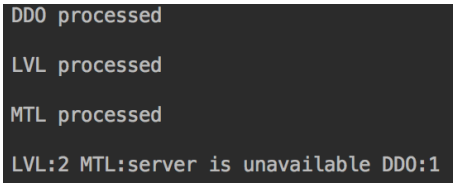
(“Client” output)

```
LVL processed

DDO processed

MTL processed

LVL:2 MTL:server is unavailable DDO:1
```

(“LVLServer” output)



("Client" output)



("DDOServer" output)

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