DISTRIBUTED CLASS MANAGEMENT SYSTEM DESIGN DOCUMENTATION

version 3.0

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 Web Services

In this program, we implemented Web Services for communication among servers and clients in the system. In Web Services architecture, there are three crucial roles: Service registry, Service provider, Service consumer (as shown below in *figure 1*.). The service provider is the entity that implements a web service and can be known as server; the service consumer is the users of a web service and utilizes that by opening a network connection and sending requests, also, can be treated as clients; the service registry is a logically centralized directory of web services that allows services be published and consumers to find an appropriate existed service.
To be more specific, we implied SOAP web services, where SOAP means Simple Object Access Protocol, that is a messaging protocol for exchanging structured information in the web services.

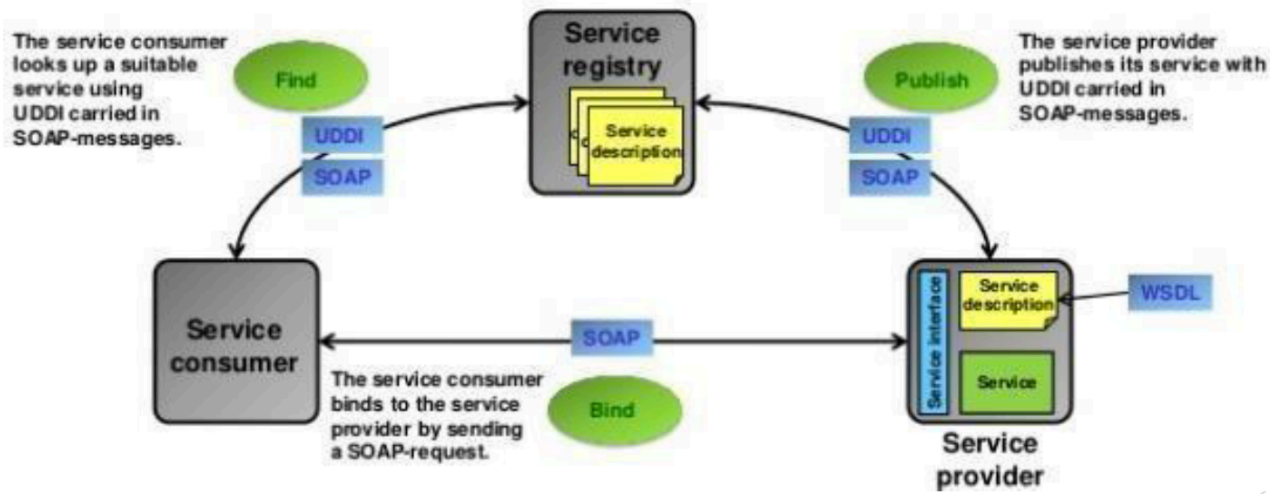

figure 1. Web Services Architecture

1.2 System Architecture and Class Diagram

The following diagram *(figure 2.)* shows the implementations of this system via UML class diagram with relations of basic inheritances and implementations.
In our project, we define an end-point interface called "***webservice.CenterServer***" in a typical format of "***package-name.interface-name***". In this interface, all the required functions of a service provider (aka server) are declared as ***@WebMethod***. The detailed functions of those methods are implemented in the class ***CenterServerImpl***, which we could read from the class diagram below and all the entities of this service have an instance of which.
Besides, the relations among other classes remain the same as the old version of this system.

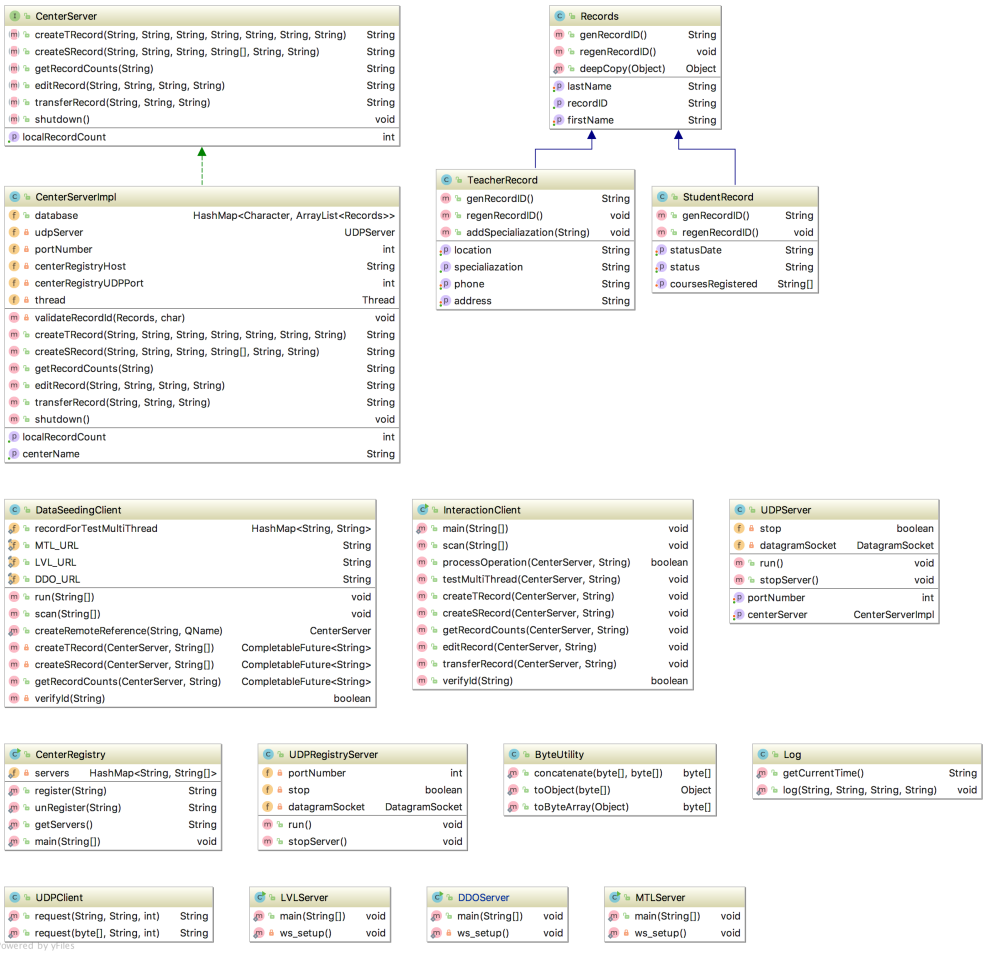

figure 2. Class diagram

1.3 Design and Implementation

Functional methods required are defined in the ***CenterServer*** interface, the detailed implementation for each method is indicated below.

1.3.1 createTRecord(String managerId, String firstName, String lastName, String address, String phone, String specialization, String location)

This method is implemented for creating a teacher record. We create a teacher record based on the passed arguments, then take the first character of the ***lastName*** of that teacher record as the key for the records ***Hashmap***. The records ***Hashmap*** is the shared data with synchronization where the new record no matter what type of record it is will be put in this shared data, and the value for each key is an ***ArrayList*** which contains all the records with the same first character of their ***lastName***. In addition, Log creation operation will be invoked to generate logs. The return value of this method is the new-generated ***recordID*** for this new teacher record.

1.3.2 createCRecord(String managerId, String firstName, String lastName, String[] courseRegistered, String status, String statusDate)

This method is implemented for creating a student record, and the implementation principles are similar to the method that creates a teacher record except that this method returns a new-generated ***recordID*** for the new student record.

1.3.3 getRecordCounts(String managerId)

In this method, the concurrent implementation via Java8 parallel stream is used. Firstly, we get the lists of registered servers. Then, generates the record counts from the lists got via UDP in parallel. Finally, returns the

generated counts of records. In addition, invoking log creating operation to create the result of getting record counts on relevant log file.

1.3.4 editRecord(String managerId, String recordID, String fieldName, String newValue)

This method is implemented to edit an existed record. According to the passed arguments, it will check if the argument of ***recordID*** is valid from the shared data called database at first in this program. Only if the ***recordID*** is valid, the following operations for editing a record will continue, otherwise, it will return relevant failed information as a result and in log file as well.
If the ***recordID*** is valid, correct properties of specific record type will be checked as next. Only if the property (aka argument ***fieldName***) and the type of record are matched, the following operations for editing will continue, otherwise, it will return relevant failed information as a result and in log file as well.
When the ***fieldName*** is specified as "location", the parameter ***ableModified*** will be used for checking if the value of ***newValue*** is valid. The default value of ***ableModified*** is true, while the checking processes will change value of this parameter with true for valid location prefix and false for invalid prefix which is not able to be modified. The record will be edited successfully if ***ableModified*** is true with succeeded result and log file, otherwise, it will return relevant failed information as a result and in log file as well.

1.3.5 transferRecord(String managerID, String recordID, String remoteCenterServerName)

This method is implemented to transfer an existed record from its original server to a remote server. According to the passed arguments, it will check if the argument of ***recordID*** and target remote center server are valid from the shared data called database at first in this program. Only if the they are both valid, the following operations for transferring a record will continue, otherwise, it will return relevant failed information as a result and in log file as well.
There is a parameter ***ableToTransfer*** which will be used for checking if the ***remoteCenterServerName*** is valid in this program. The value of ***ableToTransfer*** will be true for valid location prefix and false for invalid prefix which means the record is not able to be transferred. The record will be transferred successfully if ***ableToTransfer*** is true with succeeded result and log file, otherwise, it will return relevant failed information as a result and in log file as well.

1.3.6 getLocalRecordCount()

This method is implemented to return a count of local record of an object instance for server.

1.3.7 shutdown()

In this method, the stop operation method will be invoked via UDP requests to stop a server.

2 Test Scenario

This section will describe test scenarios that cover all the operation significant functions of this system and the execution result shown on logs.

2.1 Execute and register servers

Execute and register the servers. Results shown below.

```
Is Published : true
press stop to shut down!
```
(***DDO*** output)

```
Is Published : true
press stop to shut down!
```
(***LVL*** output)

```
Is Published : true
press stop to shut down!
```
(***MTL*** output)

2.2 Concurrently create seeding data

Execute the class ***InteractionClient*** to create several in different servers concurrently as the seeding data records by invoking method in class ***DataSeedingClient*** inside 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 TR03654
Create Successful, recordId is TR62752
Create Successful, recordId is TR52261
Create Successful, recordId is SR54083
Please input your manager ID:
Record number is LVL:1 MTL:3 DDO:0
```

(output)

```
2018-07-06 19:11:43 | LVL0001 | createSRecord | Create successfully! Record ID is SR54083
2018-07-06 19:11:43 | LVL0000 | getRecordCounts | Successful
```

(log LVL.txt)

```
2018-07-06 19:11:43 | getRecordCounts | LVLServer | Successful
```

(log LVL0000.txt)

```
2018-07-06 19:11:43 | createSRecord | LVLServer | Create successfully! Record ID is SR54083
```

(log LVL0001.txt)

```
2018-07-06 19:11:43 | MTL0000 | createTRecord | Create successfully! Record ID is TR03654
2018-07-06 19:11:43 | MTL0002 | createTRecord | Create successfully! Record ID is TR62752
2018-07-06 19:11:43 | MTL0003 | createTRecord | Create successfully! Record ID is TR52261
```

(log MTL.txt)

```
2018-07-06 19:11:43 | createTRecord | MTLServer | Create successfully! Record ID is TR03654
```

(log MTL0000.txt)

```
2018-07-06 19:11:43 | createTRecord | MTLServer | Create successfully! Record ID is TR62752
```

(log MTL0002.txt)

```
2018-07-06 19:11:43 | createTRecord | MTLServer | Create successfully! Record ID is TR52261
```

(log MTL0003.txt)

2.3 Create a teacher record

Based on the former operations and enter a valid manager ID, then enter 1 to create a new teacher record following the prompts on console. Output and logs are shown as following.

```
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.
1
Please input teacher's first name:
Tina
Please input teacher's last name:
Lin
Please input teacher's address
1234 ddo
Please input your teacher's specialization:
cs
Please input teacher's location:
MTL
Please input teacher's phone:
123456789
Teacher record with id: TR35357 was created
```

(output)

```
2018-07-06 19:18:56 | DDO0000 | createTRecord | Create successfully! Record ID is TR35357
```

(log DDO.txt)

```
2018-07-06 19:18:56 | createTRecord | DDOServer | Create successfully! Record ID is TR35357
```

(log DDO0000.txt)

2.4 Create a student record

Based on the former operations and enter a valid manager ID, then enter 2 to create a new student record following the prompts on console. 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.
2
Please input student's first name:
firstname
Please input student's last name:
lastname
Please input student's status:
status
Please input your student's statusDate:
statusdate
Please input student's courses(split with space):
comp6231 comp6411
Student record with id: SR26172 was created
```
(output)

```
2018-07-06 19:23:25 | LVL0000 | createSRecord | Create successfully! Record ID is SR26172
```
(log LVL.txt)

```
2018-07-06 19:23:25 | createSRecord | LVLServer | Create successfully! Record ID is SR26172
```
(log LVL0000.txt)

2.5 Get record counts

Based on the former operations and enter a valid manager ID, then enter 3 to get record counts. It is supposed to prompt record counts for every server. Output and logs are shown as following.

```
Please input your manager ID:
MTL0003
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:3 DDO:1
```
(output)

```
2018-07-06 19:31:20 | MTL0003 | getRecordCounts | Successful
```
(log MTL.txt)

```
2018-07-06 19:31:20 | getRecordCounts | MTLServer | Successful
```
(log MTL0003.txt)

2.6 Successfully edit a record

Based on the former operations and enter a valid manager ID, then enter 4 to edit record. With correct manager ID and correct record ID, the operations should be accomplished successfully and the program will prompt a succeed information on console. 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.
4
Please input your record id:
TR03654
Please input the field name you want to change:
phone
Please input new value:
phone
Record updated
```
(output)

```
2018-07-06 19:28:21 | MTL0000 | edit: phone | Record updated
```
(log MTL.txt)

```
2018-07-06 19:28:21 | edit: phone | MTLServer | Record updated
```
(log MTL0000.txt)

2.7 Edit a non-existed record

Based on the former operations and enter a valid manager ID, then enter 4 to edit record. But this time, we try a non-existed record and it is supposed to report a failure of no such record Id. Results 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.
4
Please input your record id:
TR00000
Please input the field name you want to change:
phone
Please input new value:
phone
No such record Id for this manager
```

(output)

```
2018-07-06 20:16:24 | MTL0000 | edit: phone | No such record Id for this manager
```

(log MTL.txt)

```
2018-07-06 20:16:24 | edit: phone | MTLServer | No such record Id for this manager
```

(log MTL0000.txt)

2.8 Edit an existed record with unmatched field

This time we try to edit a field that is not matched with its record type and here we expect a report of failure of wrong field name. 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.
4
Please input your record id:
TR45812
Please input the field name you want to change:
status
Please input new value:
status
fieldName doesn't match record type
```

(output)

```
2018-07-06 20:29:54 | MTL0000 | edit: status | fieldName doesn't match record type
```

(log MTL.txt)

```
2018-07-06 20:29:54 | edit: status | MTLServer | fieldName doesn't match record type
```

(log MTL0000.txt)

2.9 Edit an existed record with invalid new value

This time, we try to edit the location field to an invalid location and here we expect a report of failure of invalid new value. Output and logs are shown as following.

```
Please input your manager ID:
MTL0002
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.
4
Please input your record id:
TR48042
Please input the field name you want to change:
location
Please input new value:
location
The new value is not valid!
```

(output)

```
2018-07-06 20:39:01 | MTL0002 | edit: location | The new value is not valid!
```
(log MTL.txt)

```
2018-07-06 20:39:01 | edit: location | MTLServer | The new value is not valid!
```
(log MTL0002.txt)

2.10 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.

```
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:
TR45812
Please input the destination to transfer:
LVL
TR45812 is stored in the LVL | TR45812 is removed from MTL
```
(output)

```
2018-07-06 20:47:19 | MTL0000 | transferRecord:TR45812 | TR45812 is stored in the LVL | TR45812 is removed from MTL
```
(log MTL.txt)

```
2018-07-06 20:47:19 | transferRecord:TR45812 | MTLServer | TR45812 is stored in the LVL | TR45812 is removed from MTL
```
(log MTL0000.txt)

2.11 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)

```
2018-07-06 20:49:08 | LVL0001 | tranferRecord:TR12345 | No such record Id for this manager
```
(log LVL.txt)

```
2018-07-06 20:49:08 | tranferRecord:TR12345 | LVLServer | No such record Id for this manager
```
(log LVL0001.txt)

2.12 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:
MTL0002
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:
TR48042
Please input the destination to transfer:
TRT
 No such Center to transfer
```
(output)

```
2018-07-06 20:53:38 | MTL0002 | tranferRecord:TR48042 |  No such Center to transfer
```
(log MTL.txt)

```
2018-07-06 20:53:38 | tranferRecord:TR48042 | MTLServer |  No such Center to transfer
```
(log MTL0002.txt)

2.13 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:
TR48042
Please input the destination to transfer:
LVL
No such record Id for this manager
```
(output)

```
2018-07-06 20:56:57 | DDO0000 | tranferRecord:TR48042 | No such record Id for this manager
```
(log DDO.txt)

```
2018-07-06 20:56:57 | tranferRecord:TR48042 | DDOServer | No such record Id for this manager
```
(log DDO0000.txt)

2.14 Concurrently edit and transfer a same record with edit earlier than transfer

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 ***InteractionClient***. 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:
MTL0003
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
The record id list below are belongs to your server. Please input the record id as your test case.
TR35837
TR71564
TR55254
TR55254
Please input the field name you want to change:
address
Please input new value:
new
Please input the destination to transfer:
LVL
Record updated
Please input your manager ID:
TR55254 is stored in the LVL | TR55254 is removed from MTL
```
(output)

```
2018-07-06 21:14:18 | MTL0003 | edit: address | Record updated
2018-07-06 21:14:18 | MTL0003 | transferRecord:TR55254 | TR55254 is stored in the LVL | TR55254 is removed from MTL
```
(log MTL.txt)

```
2018-07-06 21:14:18 | edit: address | MTLServer | Record updated
2018-07-06 21:14:18 | transferRecord:TR55254 | MTLServer | TR55254 is stored in the LVL | TR55254 is removed from MTL
```
(log MTL0003.txt)

2.15 Concurrently edit and transfer a same record with transfer earlier than edit

In this test scenario, we will test the same operations as 2.14 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.

```
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.
6
The record id list below are belongs to your server. Please input the record id as your test case.
TR61106
TR61106
Please input the field name you want to change:
phone
Please input new value:
phone
Please input the destination to transfer:
DDO
Please input your manager ID:
TR61106 is stored in the DDO | TR61106 is removed from MTL
No such record Id for this manager
```

(output)

```
2018-07-06 21:00:50 | MTL0000 | transferRecord:TR61106 | TR61106 is stored in the DDO | TR61106 is removed from MTL
2018-07-06 21:00:50 | MTL0000 | edit: phone | No such record Id for this manager
```

(log MTL.txt)

```
2018-07-06 21:00:50 | transferRecord:TR61106 | MTLServer | TR61106 is stored in the DDO | TR61106 is removed from MTL
2018-07-06 21:00:50 | edit: phone | MTLServer | No such record Id for this manager
```

(log MTL0000.txt)

2.16 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 LVL 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 servers indicated with the port number of each server in an order of 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, as well as a timeout about 3 seconds occurs.

```
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.
3
MTL:0 LVL:server is unavailable DDO:0
```

(***InteractionClient*** output)

```
MTL on port 8180 processed

DDO on port 8182 processed

LVL on port 8181 processed

MTL:0 LVL:server is unavailable DDO:0
```

(***MTLServer*** output)

```
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.
3
MTL:0 LVL:server is unavailable DDO:0
```

(***InteractionClient*** output)

```
DDO on port 8182 processed

MTL on port 8180 processed

LVL on port 8181 processed

MTL:0 LVL:server is unavailable DDO:0
```
(***DDOServer*** output)

3 Important and Difficult Parts

In this assignment, we mainly refactor our project from ***CORBA*** to Web Service method. So, the most important part is how to create stub in web service. In web service, we use ***URI*** to locate the resource. So, the package translates the service implementation to ***wsdl*** file which is middle file. At the same time, we push it to a special ***URI***. The ***URI*** defines the port number. Then, the client can get the stub from that special ***URI*** and invoke the functions in server.
Specially, we need notice the utility of annotation:
"***@Webservice***": This is used to mark center service and its implementation.
"***@WebMethod***": used to mark functions would be translated to ***wsdl*** and pushed to ***URI***.

4 Reference

[1]. https://moodle.concordia.ca/moodle/pluginfile.php/3157537/mod_resource/content/2/Tutorial_6_Web_Services.pdf
[2]. https://en.wikipedia.org/wiki/SOAP