A picture containing graphical user interface

Description automatically generated

COMP 6231 DISTRIBUTED SYSTEMS DESIGN

SUMMER 2021

DESIGN DOCUMENT FOR ASSIGNMENT 3

TEAM SMARTY PANTS

BINIT KUMAR - 40172005

ABDUL AZIZ RAZA SHAIK - 40182726

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

JULY 2021

Introduction

**Distributed Class Management System (DCMS) using Web Service:**

Distributed Class Management System is a distributed system used by center managers to manage information regarding the teachers and students across different centers.

The three centers locations are:

* Montreal (MTL)
* Laval (LVL)
* Dollard-des-Ormeaux (DDO)

The server for each center (called CenterServer) maintains a number of Records. The two types of Records are:

* TeacherRecord
* StudentRecord

A Record a unique RecordID starting with TR - TeacherRecord or SR – StudentRecord.

Fields in TeacherRecord – First Name, Last Name, Address, Phone, Specialization and Location.

Fields in StudentRecord – First Name, Last Name, Courses Registered, Status and Status Date.

Managers with a unique ManagerID can perform operations:

* createTRecord
* createSRecord
* getRecordCounts
* editRecord
* transferRecord

This application has a number of CenterServers(one per center) each implementing the above operations for that center and ManagerClients(one per center) invoking the manager’s operations at the associated CenterServeras necessary.

It has been developed using Web Services.

Description of the techniques used:

Web Services:

Web Services enable users from different parts of the world to communicate with each other, it is a software application using client server architecture to provide interoperability between diverse applications.

Standard Components of Web Services include SOAP, WSDL and UDDI.



**Architecture of a SOAP based Web Service**

We have also made use of various APIs to achieve some of the requirements of the system such as:

**Java Socket Programming:**

For communication among the instances of our center location servers: Montreal MTL, Laval LVL and Dollard-des-Ormeaux DDO.

**Java Multi-Threading:**

Using multiple threads our system achieves concurrency which enables Clients to invoke multiple methods concurrently, Servers able to handle requests on multiple invocations concurrently and separate threads to handle other operations like connection protocol etc.

**Logger for Java:**

For activity log to be maintained on both the client and server sides.

Data Structure used:

HashMap:

Using HashMap to store items in key/value pairs can accessing and updating various data items at the same time easier which in turn helps achieve concurrency and synchronization in our system. When a manager of a center location needs to create, edit or transfer a record, HashMap will not be locked completely, instead only the list containing the record will be locked. This makes multiple threads working on shared data items safer and prevent deadlock or any other undesirable effect in our DCMS. The performance aspect was vital to make a choice of this data structure.

How data is stored in Hash map

|  |  |  |
| --- | --- | --- |
| Record 1 | Unique record ID  Record Type = SR | Student Record |
|  |
| Record 2 | Unique record ID  Record Type = TR |  |
| Teacher Record |

Architecture of DCMS using Web Service:

Diagram

Description automatically generated

Separation of Client and Server:

By using the webservice to implement our system we have two different packages for client and server independent of each other.

DCMS has servers at 3 different locations where the managers can access and make changes to records.

A picture containing text

Description automatically generated

Architecture of the system:

The DCMS architecture has 3 locations (Montreal, Laval and DDO) which are the clients of the system.

The Managers at these locations have access to the database which consists of the records: Student Record and Teacher Record. The managers can perform functions such as createTRecord, createSRecord, getRecordCounts, editRecord and transferRecord as per requirement.

ManagerClient is the client program which handles these requests and attempts to create, edit or transfer the appropriate record with the corresponding server associated with the manager in the form of a hashmap and each server also maintains a log containing the history of all the operations that have been performed on that server in a separate text file. By invoking getRecordCounts, the manager of a center can also get the number of record (both teacher and student) present in their respective database.

This system has multiple centers and one CenterServer for each of them which make use of a centralized ManagerClient to invoke the required method from the repository, for each operation it finds the information about the requesting server and invokes the corresponding operation.

Diagram

Description automatically generated

TEST CASES EXECUTED:

@Binit – Update Test Cases part

Test Setup:

**Machine-1 Configuration**

From an MS-DOS system prompt (Windows), enter:

*start orbd -ORBInitialPort 7719*

note: 7719 is the orb port of DDO server

Execute RunDDO.java to run DDO server

**Machine-2 Configuration**

From an MS-DOS system prompt (Windows), enter:

*start orbd -ORBInitialPort 7717*

note: 7717 is the orb port of MTL server

Execute RunMTL.java to run MTL server

**Machine-3 Configuration**

From an MS-DOS system prompt (Windows), enter:

*start orbd -ORBInitialPort 7718*

note: 7718 is the orb port of LVL server

Execute RunLVL.java to run LVL server

**In any of the machine above:**

Execute RunClient.java to run manager’s client

# Test Scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Description | Expectation | Result |
| 1 | Manager creating Teacher records | Record should be saved in database | PASS |
| 2 | Manager creating Student records | Record should be saved in database | PASS |
| 3 | Managers requiring valid id to access the server Test Data: (MTLXXXX / LVLXXXX / DDOXXX) | Managers should be able to connect to the right server | PASS. |
| 4 | Managers not able to connect with invalid id | Invalid managers id will get an error message | PASS |
| 5 | Managers able to get all the record counts | The total count of all the records should be printed for each of the servers | PASS |
| 6 | Manager should be able to edit records | Manager should be able to modify records from the record id | PASS |
| 7 | Phone number validation in Teacher Records | Only numeric values in Phone number | PASS |
| 8 | Status Field validation in student record | Only Active and IActive status | PASS |
| 9 | Managers should not be allowed to edit any fields other than Address, Phone and Location in the Teacher Record. | Any fields other than Address, Phone and Location in the Teacher Record are not modified | PASS. |
| 10 | Managers should not be allowed to edit any fields other than Course Registered, Status and Status Date in the Student Record. | Any fields other than Course Registered, Status and Status Date in the Student Record are not modified | PASS |
| 11 | Transfer of record from one server to another | When transferring record using record id and server name, the new record is transferred to the new server | PASS |
| 12 | Deletion of record from current server after transferring the record to another server | After transferring record to another server, the record in the current server is being deleted successfully to avoid any duplicate data. | PASS |

CHALLENGES FACED:

* The most difficult and important part of the design and execution was creating the correct client-server architecture which would work exactly as given in the problem statement.
* Another challenge faced by us was to understand the specifics and limitations Web Service architecture and functionality of wsdl.
* Third difficulty we faced was to maintain concurrency in the system.
* Making use of wsgen and wsimport to build endpoint files automatically before publishing the service was a challenge.

UML Diagrams:

Use case diagram:

Diagram

Description automatically generated

Class diagram

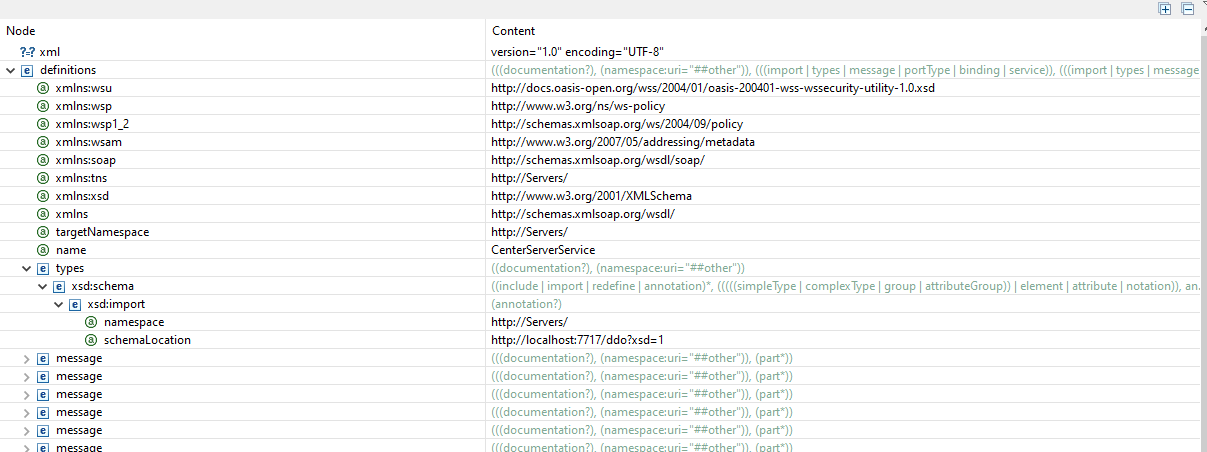
Diagram, engineering drawing

Description automatically generated



New Changes

wsdl file



* Every method includes manager id
* The generated package stub is under ca.concordia.dsd.stub package in the source code that is being using by managerclient.java

CenterServerImpl.java

Logo

Description automatically generated with medium confidence

Logo, company name

Description automatically generated

* CenterServerImpl class has @WebService and @WebMethod annotations

CenterServerPublisher.java

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

* wsgen class creates the necessary classes and wsimport to parses the URL to create files.

Graphical user interface, text, application

Description automatically generated

* Manager client uses generated stubs classes created using wsdl file of each server to create a connection to them.
* At the end it also starts the internal UDP server event loop that can transfer the record when client trigger transfer record command to another server via UDP.

References:

* George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair, "*Distributed Systems Concepts and Design"*, Fifth Edition, Addison-Wesley, 2012, ISBN: 0-13-214301-1.
* M. L. Liu, "*Distributed Computing: Principles and Applications*", Pearson Addison-Wesley 2004, ISBN 0-201-79644-9.