Software Architecture Document

Version 1.1

for

A2 SOEN 423

Prepared by

Charles-Antoine Hardy	27417888	m.hardy.inc@gmail.com

Instructor:	Shivaraj Mallikarjun Alagond
Course:	SOEN 423
Date:	October 2018

Document history

Date	Version	Description	Author
A2	1.1	Update of the document for A2	CAH

Table of contents

1. Introduction

This document is describing the architecture of a SOEN 423 Assignment 2.

Purpose

This document shall provide a basic understanding of the architecture.

Class Diagram

I will not provide one large class diagram as it will be very confusing. Instead, multiple small class diagram has been put together to explain the software.

Model

The model contains object classes. The main classes described in the requirement: Record, Employee, Manager, Project, Location.

The Manager and Employee are direct children of the class Record. Each class has its own override for: 'toString()' , 'Equals' and 'hasCode()'. It's allowing us to write objects in the log file with 'toString()' and to compare objects easily.

(As the image is too large for a document, I've added it to the project Git repository.)

https://trello-attachments.s3.amazonaws.com/5ba8189ab9f30563a9787c4c/5bc91b7e608b24 1c4297a8a2/d99a0c80b2b7981c2eac913b18af511c/ObjectRelations.png

Making the *Employee* and *Manager* being children of *Record* is allowing to store List of records with manager and employee in the list.

Moreover, with CORBA system some model class have been duplicated by the code generator. It is creating a mess in the code.

Core

The core architecture of the RMI software is represented by this https://trello-attachments.s3.amazonaws.com/5ba8189ab9f30563a9787c4c/5bc91cd089cba20c3e6 77b89/98818b2b9c8887d8dbfde26c0e4a4b17/CoreCorba.png

The *ServerLauncher* class is launching the server component while the *client* class is using the registry.

The *ServerConfigurator* is creating the 3 servers. For each location (ENUM), the configurator is creating an *IStore* instance, the instance is inserted in an *IHRAction* instance which is also inserted in the UDP servers and in the registry. By doing so, the UDP server and registry are united.

As you can see, in the diagram the *IHRAction* contains an instance of *IStore* and the *ServerUDP* also contains an instance of *IStore*. It is allowing the UDP server to write into the log file of the server.

Each *IStore* instance is configured with a *storeName*. The name is the location name ("US", "UK", "CA") The name is used to create a folder or find the folder linked to the server. Each server has 3 default storage files created: **Log.txt, Record.txt,** and **Project.txt**.

With CORBA the communication is made through the DEMS class generated from *DEMS.idl* interface. Since the *IHRActions* inherits from *DEMSOperation*s the operations of the class are usable in a client like *HRManagerLauncher*.

ServerConfigurator - Code Snippet

https://github.com/Winterhart/java-rmi-simulation/blob/master/project/src/backEnd/ServerConfigurator.java

This is where the 'magic' happen. Each registry is bind to a *instanceHRAction* and each UDP servers is bound to the same instance. It's allowing UDP Server thread to communicate with current *HRAction* data. (e.g. HashMap<Integer, List<Record>>)

For A2, the same logic has been used. Moreover, I've added another UDP server to handle transfer record.

Storage

The storage is made of two class *IStore* (interface) and the implementation *Logger*. The storage is only made to insert/update/read/delete on a text file.
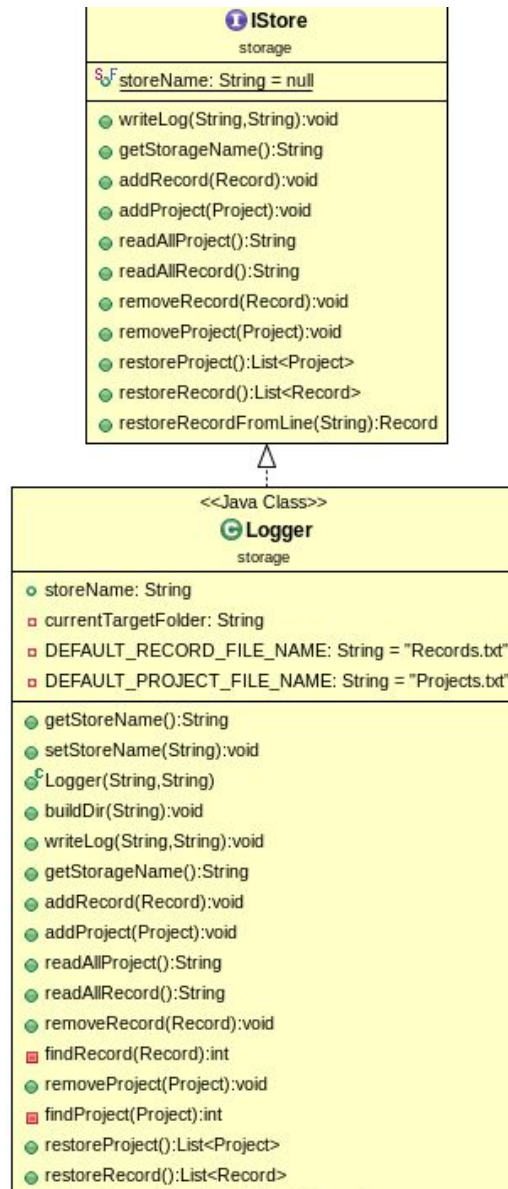
CORBA Interface

The interface has been implemented *IHRAction* and *HRAction.* Few helper methods were added to increase readability. With CORBA, the interface needs to inherit the DEMSOperations in order to be executed by the CORBA engine. Moreover, the class HRActions now needs to extend the DEMSPOA.
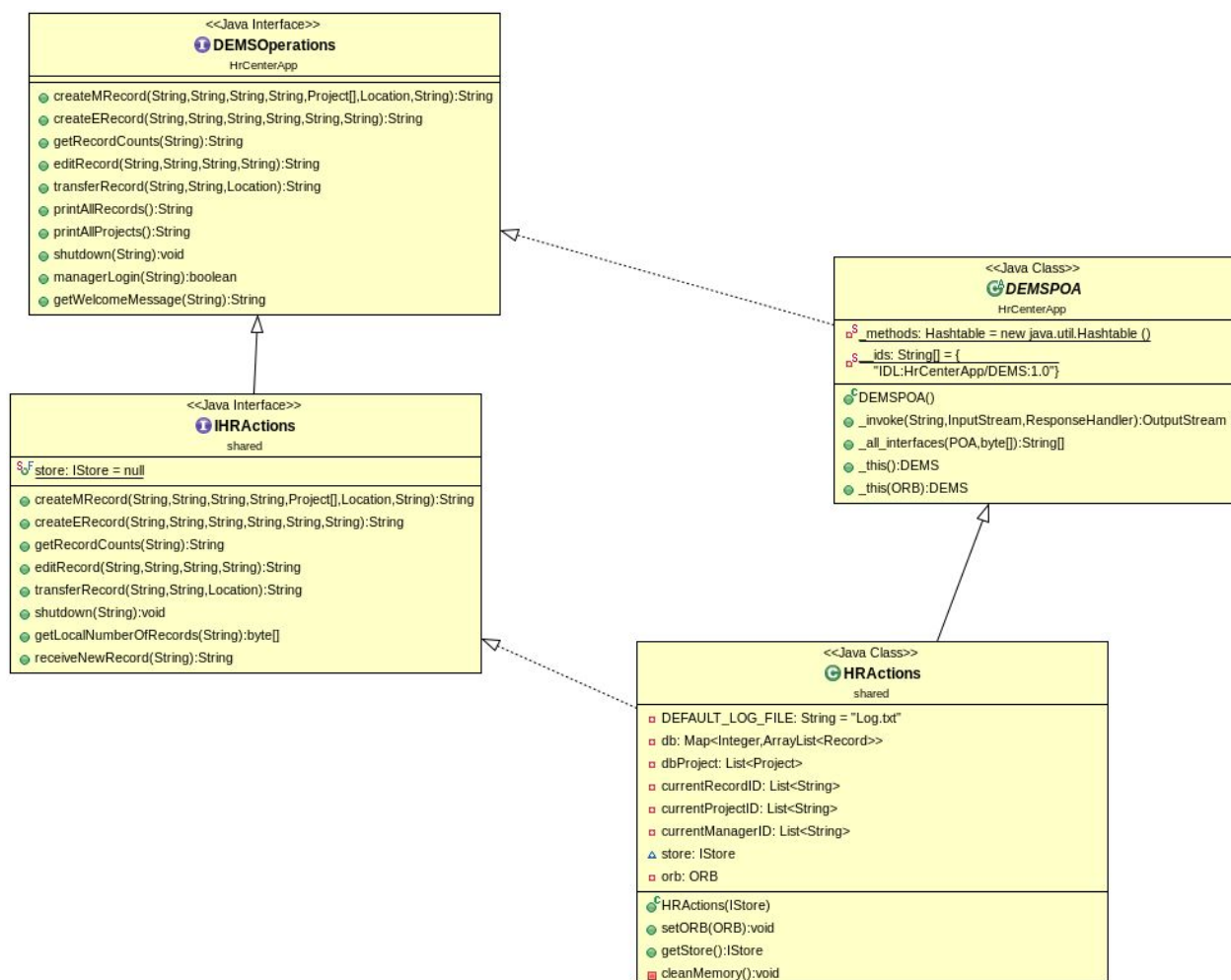
The method *getRecordCount* is counting the local record and then the other records. It is using the *PortConfiguration* object to get to the other UDP server. Once on the other UDP server, since they have an instance of the local HRAction, the UDP server simply call *localInstance.getLocalNumberOfRecords()*

Another complex method is the *editRecord* the method has to find out if the target is a project, manager or employee then it has to make sure it exists, finally, it is updating the value with an addition/deletion operation in storage.

UDP Server

As discussed, the UDP server is running in a separated thread. The server has an instance of its local HRAction called *localInstance.* Two UDP servers are available a Transfer Record server and a count record udp server.
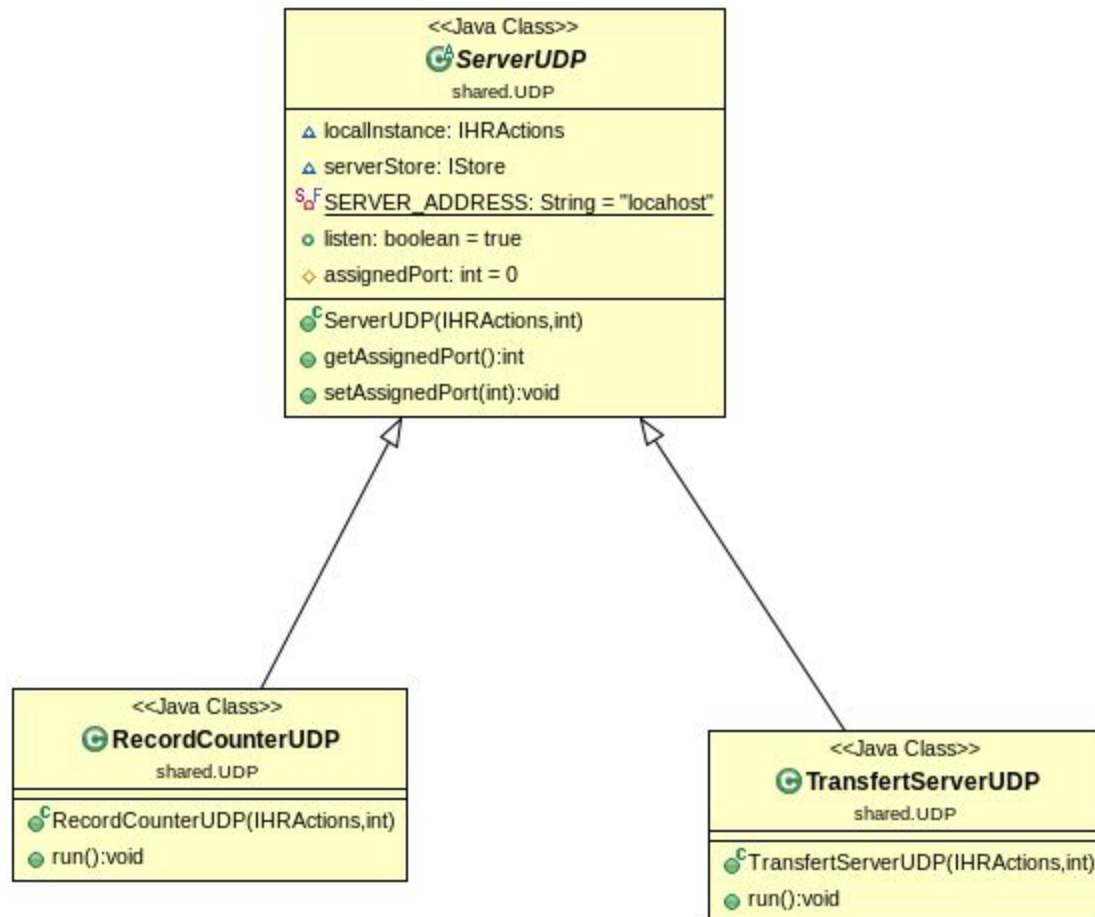
Config

A package folder has been added to the overall solution. Things like ports and directory are stored in it.

CLI

In the A2 a CLI has been added to facilitate the testing.

Repository

The project is being built in a private GitHub repository.
(https://github.com/Winterhart/java-rmi-simulation). The A2 will simply be done again on the same repository.