# Software Engineering Lab 6 Report

## 1 Document Information

### 1.1 List of Authors

| Group/Team | Names |
|---|---|
| <Your Group> | <Hilary Ogalagu > |
| <Your team name> | <Author 2> |

### 1.2 Versions

| Version | Date | Author | Notes |
|---|---|---|---|
| 0.1 | 12.06.2021 | < Hilary Ogalagu > | Concepts for Distributed Object system, Client-Server Design and Demonstrator Description |
| 0.2 | 13.06.2021 | < Hilary Ogalagu > | RMI-based Demonstrator Description |
| 0.3 | 16.06 | < Hilary Ogalagu > | Modification of the RMI-based Demonstrator Description |
| 0.4 | 17.06 | < Hilary Ogalagu > | Continues remodification and updating lab 6 report |
|  |  |  | <further versions> |

## Content

## 2 Glossary

The following domain-specific terms are used throughout the document.

| Term | Abbreviation | Description | Reference |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Route on map | ROUTE | <how it is defined in other terms> | <Website or App> |
| etc. | | | |
| | | | |

## 3 Concepts for Distributed Object System

<Prep task #1 and #2 >

We want to distribute our application so we need interfaces which we extend remote  because it's a remote interface which methods throws RemoteException and these methods which be called on the proxy side and it is automatically executed on the Client side. So here the communication is important, so we have to analyze what is going on which direction. So that is why the interfaces has all the method of the classes that is needed to be distributed. So in the interfaces we include java Api's (**java.rmi.Remote**).

In the Server Class, we have two parts, the StartRegistry() because we do not need to start it from the command line but here by locateRegistry and create it and set a logger and then we have a  main program where we call the server and instantiate the object in order to make it public by **Naming.rebind.** The Registry keeps a reference to that object. And in the Client instead of instantiating the object we simply ask the Registry to give us a reference to that object. So the client does not instantiate but look up for the instantiate of the object from the server.

Now  we have the **UnicastRemoteObject** which are used to automatically create the proxy objects. So the classes that have the interfaces will be **UnicastRemoteObject (java.rmi.UnicastRemoteObject)** because we need proxy for those objects so called stubs.

Furthermore Serialization **(java.io.Serializable**) is used to serialize object of a class by implementing it and it is used for distribution of stubs. An object class must implement Serialization if it is to be passed to writeObject(). It is converted to a representation that can be sent over the Net.

## 4 Client-Server Design and Demonstrator Description

<If available take it from lab 5 report for explanation that is independent of implementation technology>

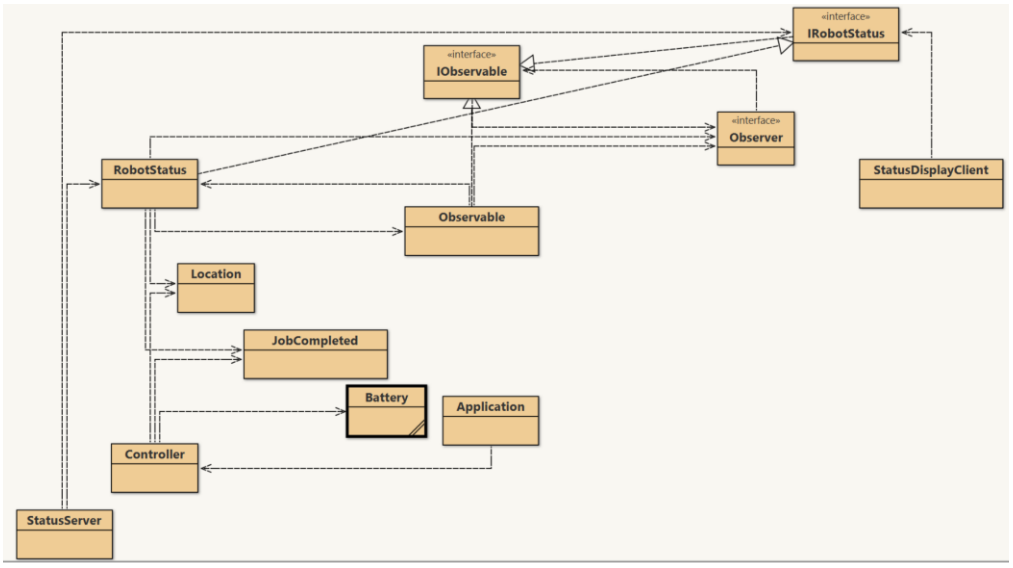
| **prototype client and server description** | |
|---|---|
| **Client** | **Server** |
| The client calls the Registry which have have already the server- stub and this why the client can directly connect with the server. When a method call is made at the Client-side, the sends message through the interface to the | The Server in the other hand implements the interface with the method call so when the client calls this method, the server gets the message and feeds back the Client with the right value in response to the method call. In |

| | |
|---|---|
| server which then in return feeds back the client with the right value. Here the Client represent our application while the server represents the Robot itself. | this case, the server represents the Robot and the Client the application. |

| **Classes** | **interfaces** | **Client- side** | **Server-side** | **stub** | **Description** |
|---|---|---|---|---|---|
| RobotStatus | IRobotStatus | IRobotStatus | Remote IRobotStatus | getUpdate() | The Server feeds the Registry with a server stub which when the client calls the registry and sends the stub to the client. |
| Location | | IRobotStatus | Remote IRobotStatus | getLocation() | The Server gets the method of the interface and when the client makes request with the same method call. The server then sends the stub to the client |
| JobCompleted | | IRobotStatus | Remote IRobotStatus | getJobCompleted | Same process like previously with the Location class. |
| Battery | | IRobotStatus | Remote IRobotStatus | getBattery | Here is also same process going on because the server communicate with the interfaces and sends the client the stub when the method call is made. |

## 5 RMI-based Demonstrator Description (Which partner)

<Prep task #3 and #4>



**task #3**

| **Interfaces for Implementation** | **Reasons** |
|---|---|
| IRobotStatus Interfaces | The IRobotStatus interface extends Remote and implements different methods of the class RobotStatus, which gets update and as well as method of getting the different status of the other classes that needs to be updated. |
| IObserver | IObserver interface extends Remote and gets update from the IObservable whenever there is a change in the State. |
| IObservable | IObservable interface extends Remote just like the previous interfaces and it is extended by the IRobot interface and observes the list of the observables. |

**task #4**

| **Class/Object/interface** | **Declared as java.rmi.Remote/ java.io.Serializable/ java.rmi.UnicastRemoteObject/exported by Naming.rebind** |
|---|---|
| IRobotStatus interface | java.rmi.Remote |
| Class Observable | java.io.Serializable |

| | |
|---|---|
| Class StatusServer | exported by Naming.rebind |
| Class StatusDisplayClient | java.rmi.UnicastRemoteObject |
| class RobotStatus | java.io.Serializable |

<Also comparison: name agreements and deviations in comparison with the corresponding sections of your lab 5 report>

| Observer Pattern: UC5: DisplStatus | | |
|---|---|---|
| **Classes** | **Agreements with Lab 5 report** | **Deviations with lab 5 report** |
| RobotStatus Class | This is in total agreement as it provides the state of the robot system and also implements the **Observable and also Serializable**. And also keeps the state of the Robot | No deviation. |
| Class Observable | This abstract class extends **Observerable** and implements **Serializable** and get updates of all the other classes. | Little implement deviation on Serialization. |
| Location | In Agreement | No deviation. |
| JobCompleted | In Agreement | No deviation. |
| Battery | In Agreement | No deviation. |
| Observer | This method is called whenever the observed object is changed | A new class was created. |
| IObservable | This has different methods that adds ,delete observer, notify observer as well as change. | A new interface was created to implement this. |

## 6 MOM-based Demonstrator Description (Which partner)

< Lab task #2>

| MOM-based demo |
|---|

| message type | message content description | sender | receiver | topic | queue |
|---|---|---|---|---|---|
| initial RobotStatus | Stores all the state of the robot like location, battery, jobcompleted. | Client | Server | Request for location | IRobotStatus |

## 7 Shortcomings and drawbacks

## References

<all used ressources>