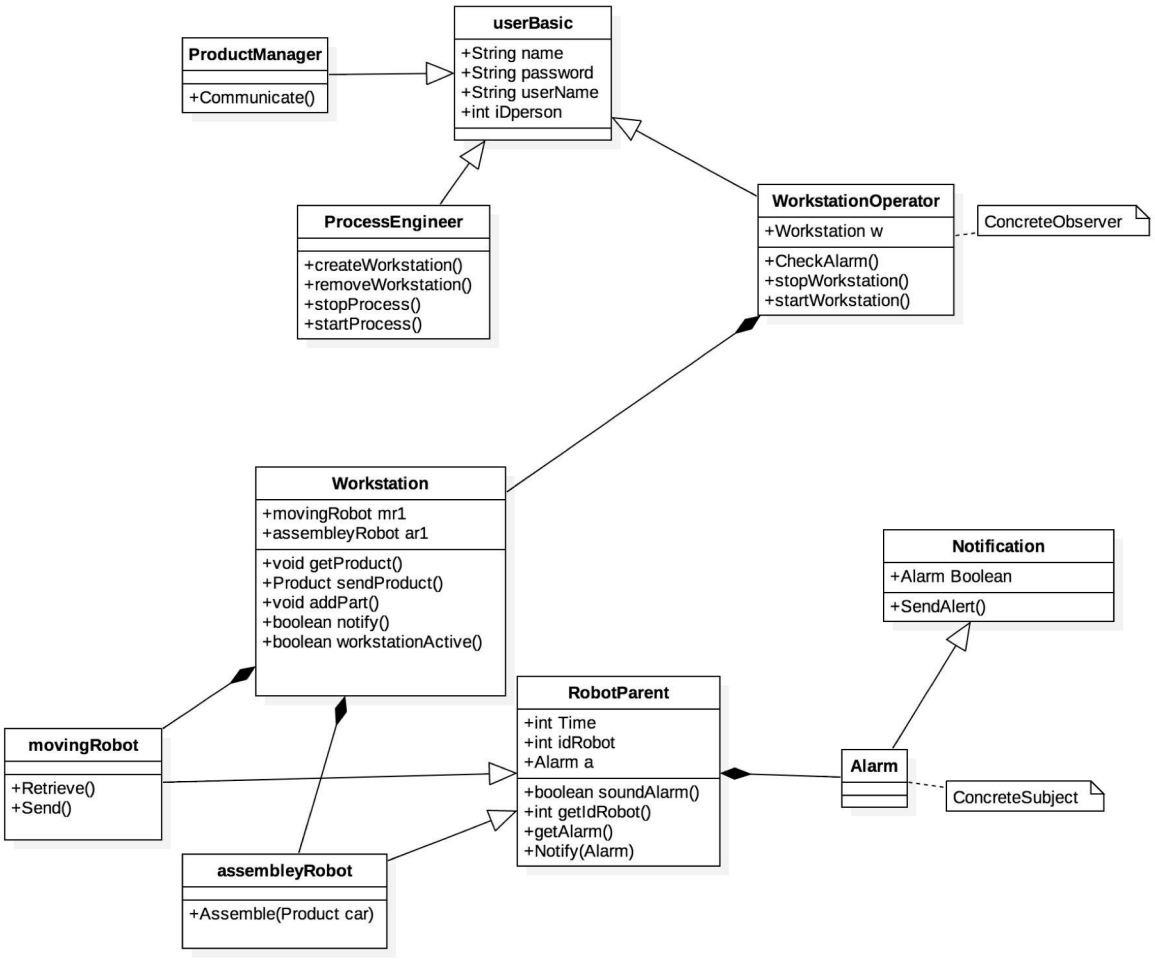
Ryerson University

CPS888 Warehouse Project

Iteration 2

Danushyan S., Farhad N., Matheesan M
4/3/2017

3.2 Class Diagram

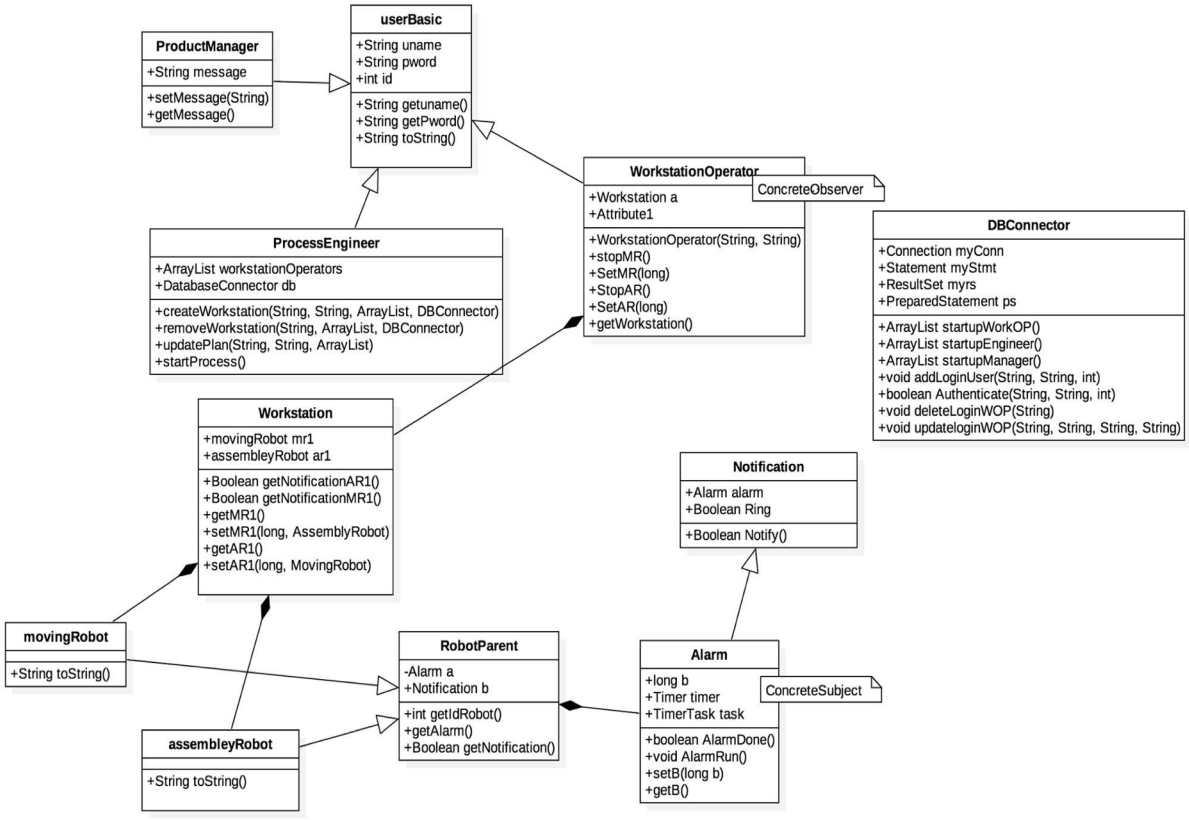


3.3 Design Pattern Discussion

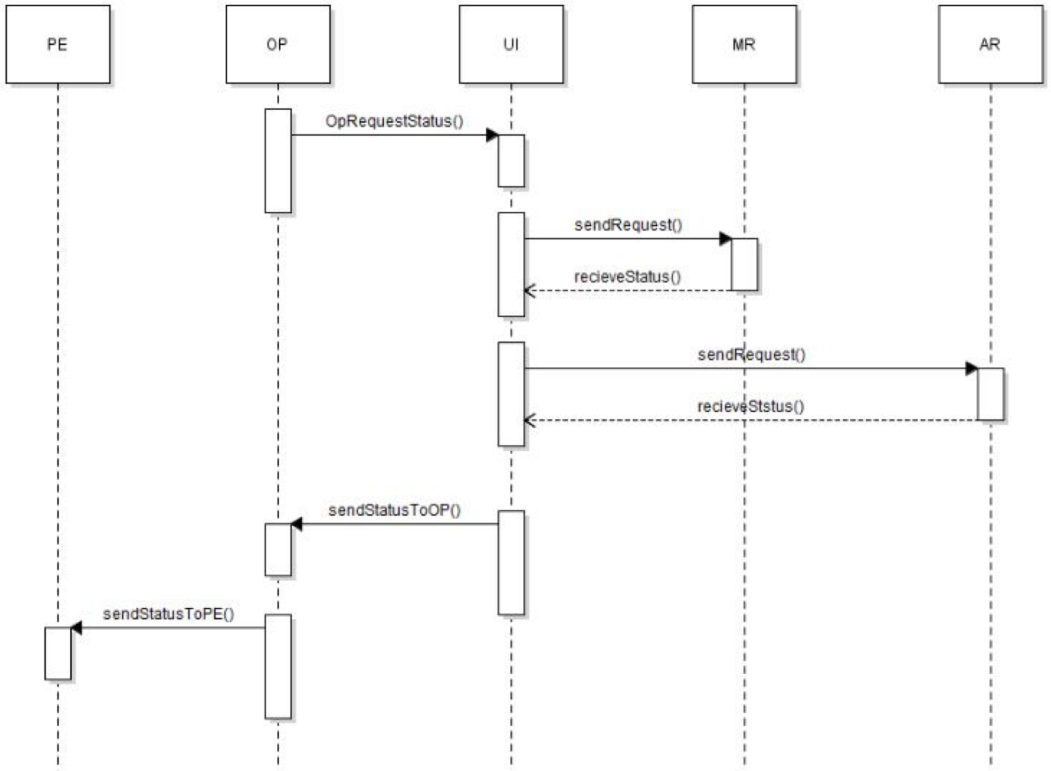
The most common components that were benefited by the Observer Pattern implementation was the notification/Main class. This class is in charge of sending important notices to all other components. These notices can take the form of updates and alerts. Each variation has a specific purpose and destination. The Subject will be the Main class while the Observers will be the OP, PE and PM along with the MR and AR in special circumstances. The way we implemented this in our code was to allow alarms to contain a variable, when less than 0 gives our notification class a Boolean true. This Boolean true allows us to update the status to true as alarm goes off. When the alarm goes off, the workstation operators all get a pop-up on the GUI asking them to insert a new alarm time so that they can fix the machine batteries and provide another check-up time to fix the machine later on again.

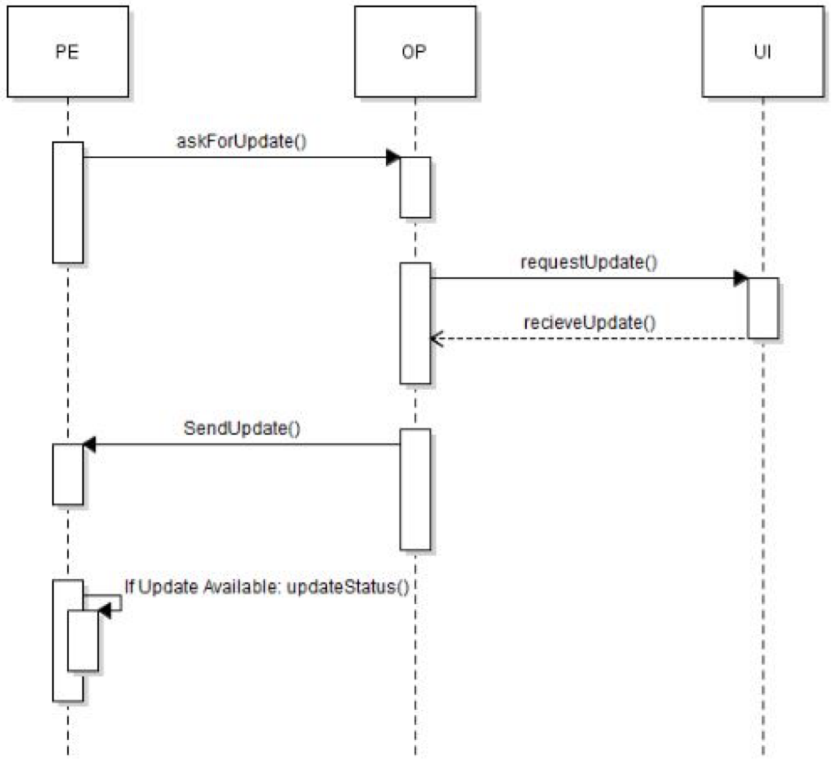
One pattern that we would have included in the class diagram would have been the MVC pattern. The reason being would be that we have a database connection class which allows us to control functions sent to the database, and receive packets of data from the database to be used in the back end. The backend would then send this information once processed to the frontend, which in our case would be the Java SWING GUI interface. The interface would display modified datasets obtained from the database connection class, which in turn received it from the database itself.

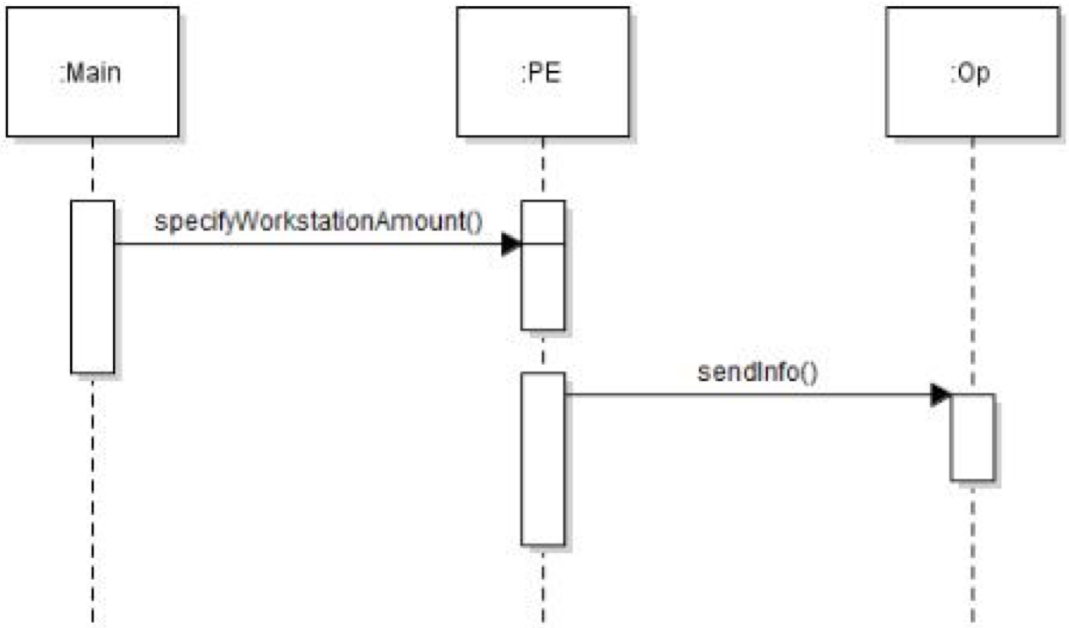
3.4 Updated Class Diagram

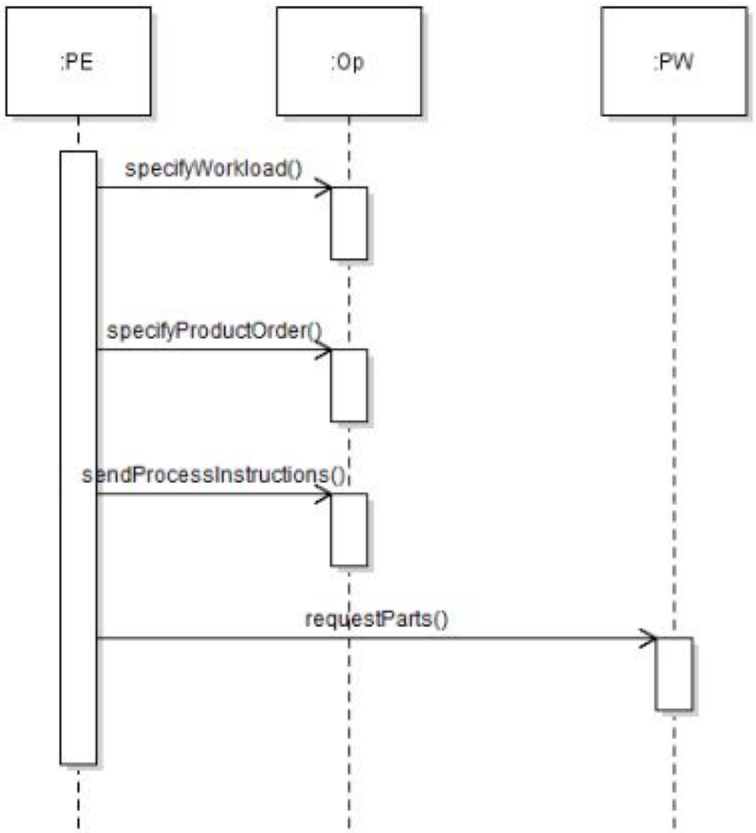


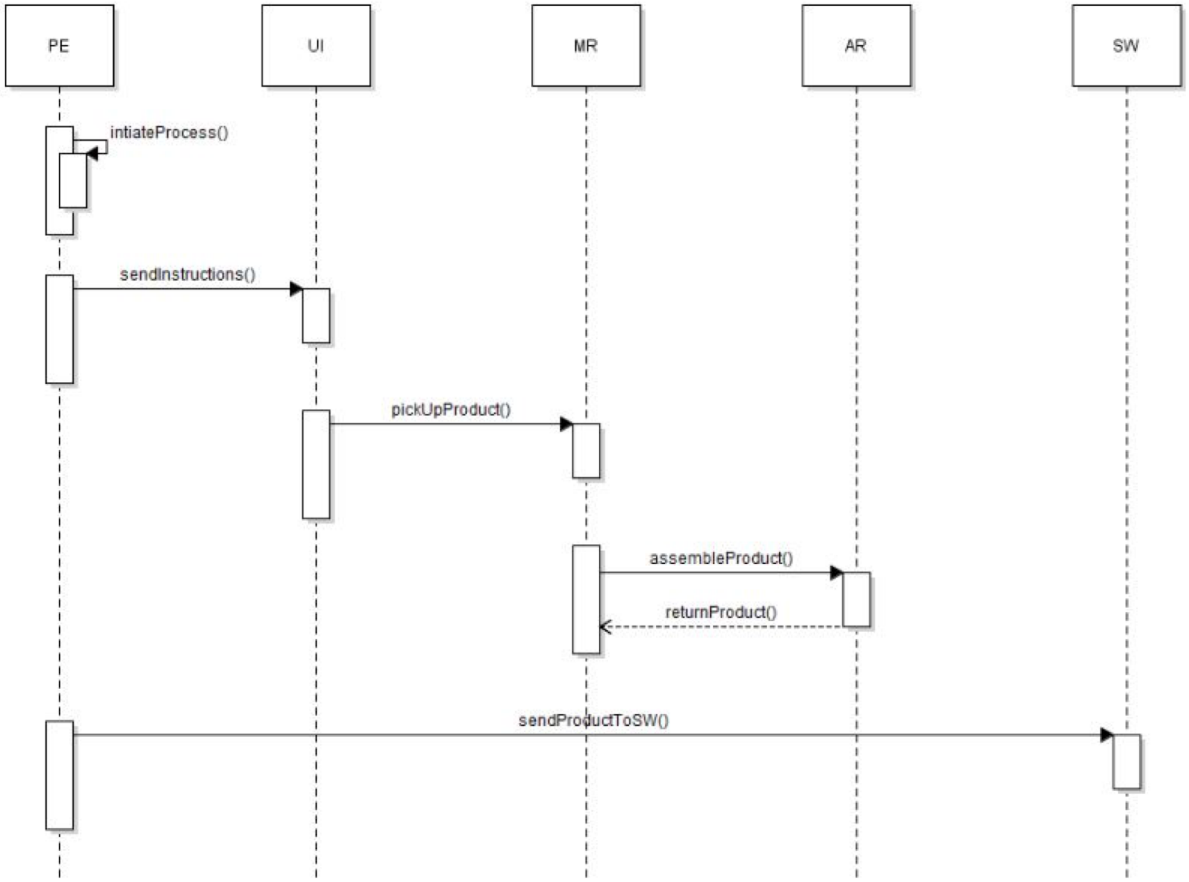
3.5 Sequence Diagrams

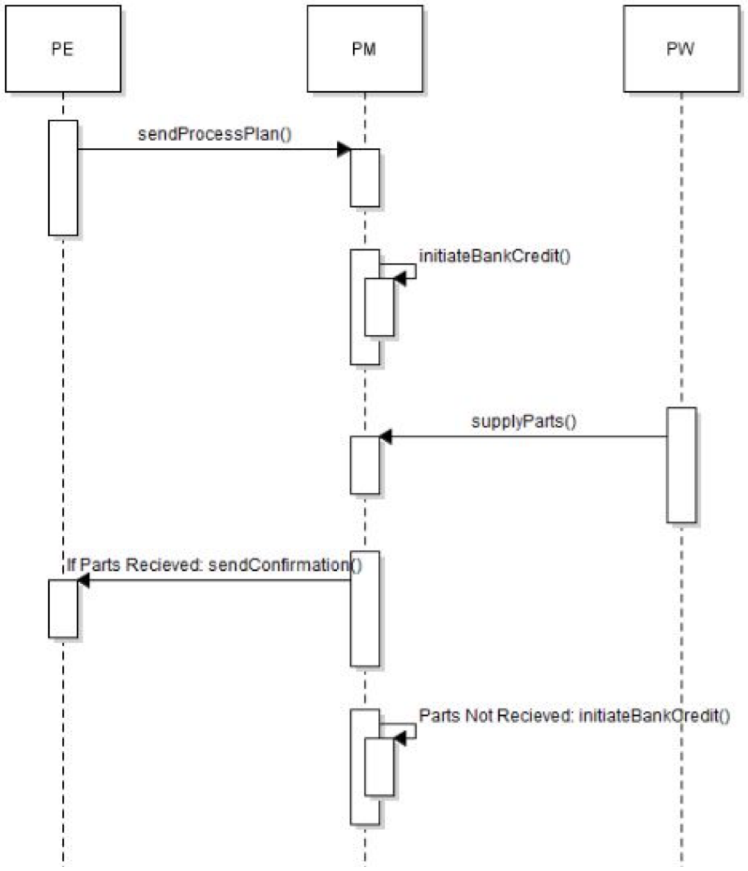


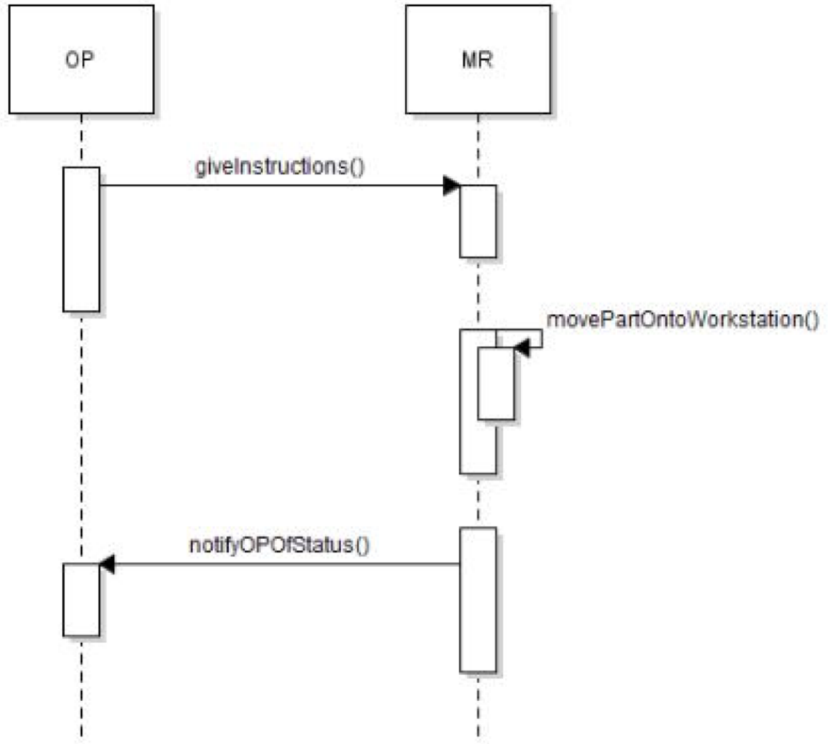


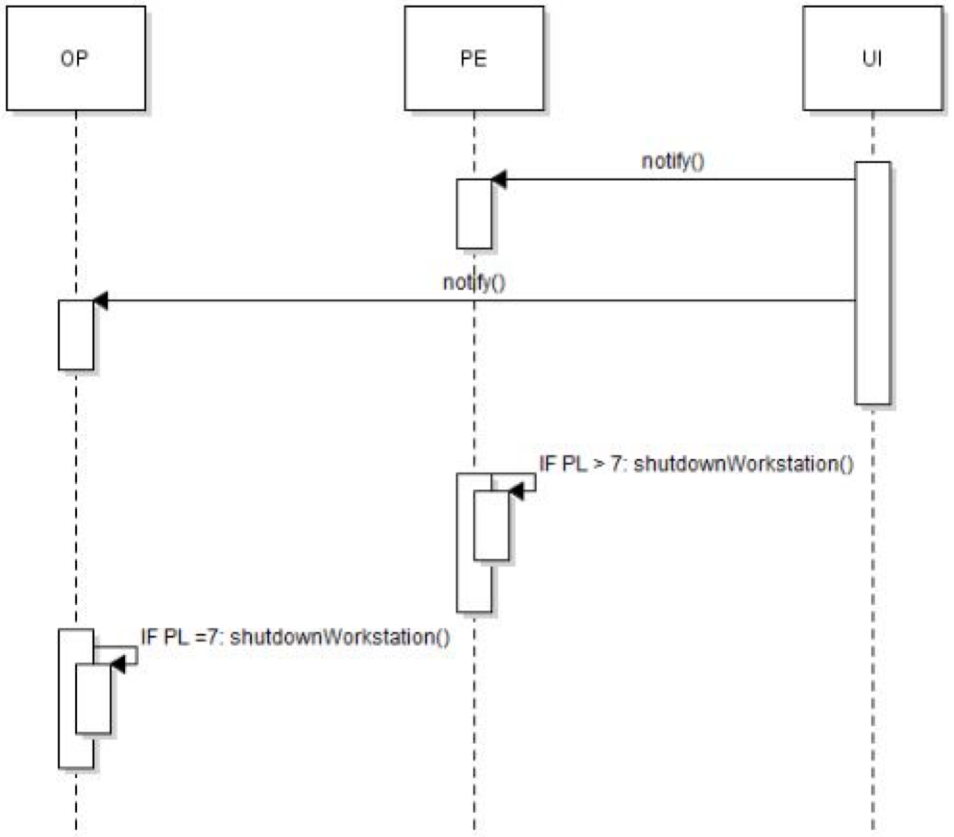

:Main
:PE
:Op
specifyWorkstationAmount()
sendInfo()

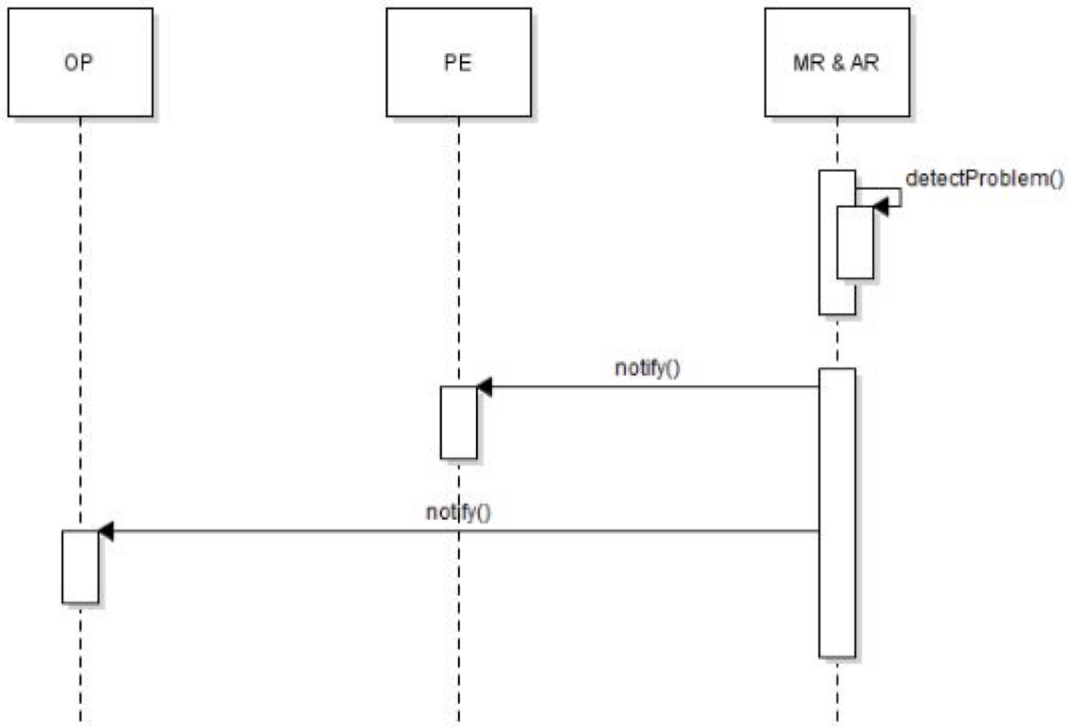

:PE
:Op
:PW
specifyWorkload()
specifyProductOrder()
sendProcessInstructions()
requestParts()


PE
UI
MR
AR
SW
intiateProcess()
sendInstructions()
pickUpProduct()
assembleProduct()
returnProduct()
sendProductToSW()


PE
PM
PW
sendProcessPlan()
initiateBankCredit()
supplyParts()
If Parts Recieved: sendConfirmation()
Parts Not Recieved: initiateBankOredit()


OP
MR
giveInstructions()
movePartOntoWorkstation()
notifyOPOfStatus()

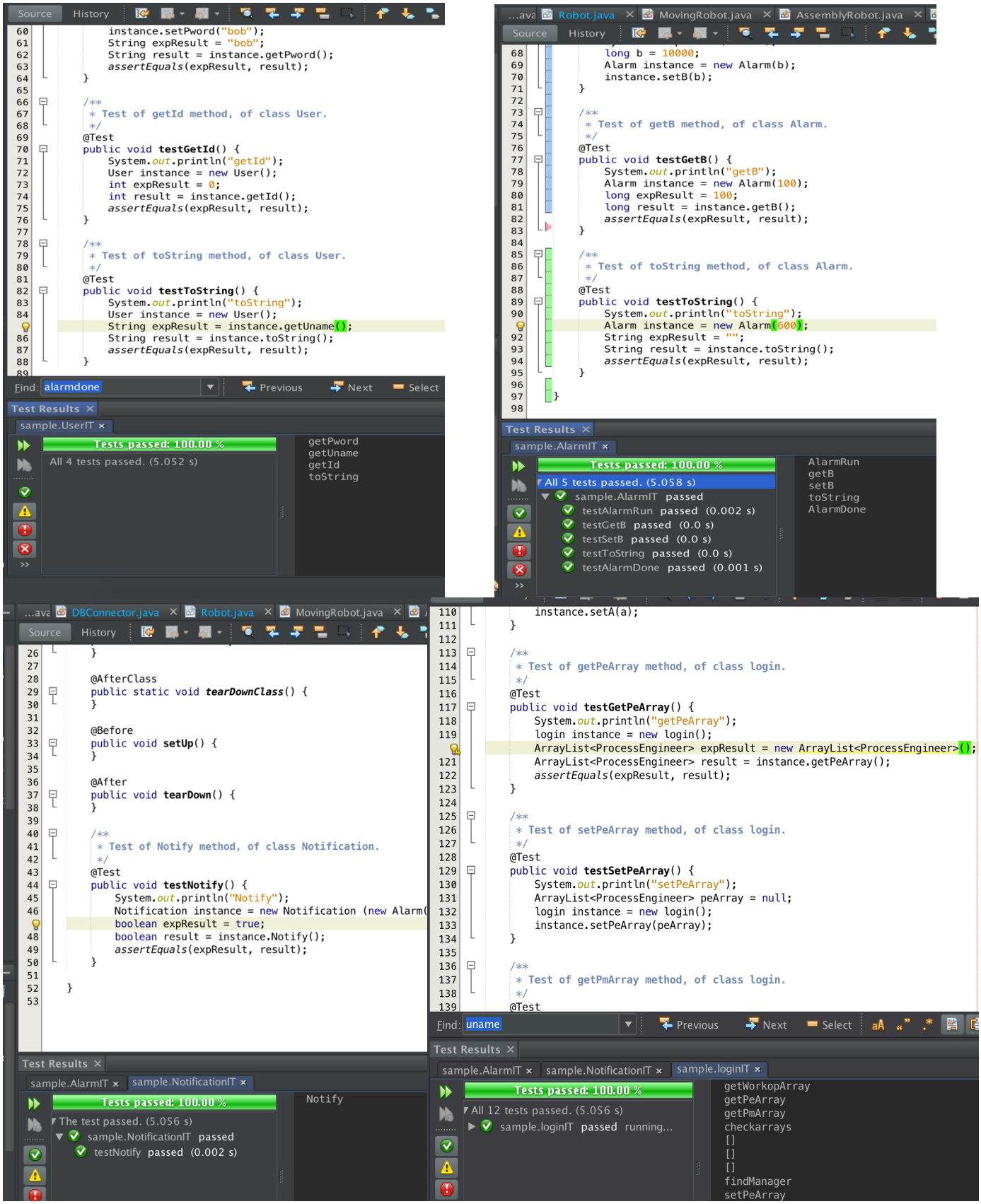

OP
PE
UI
notify()
notify()
IF PL > 7: shutdownWorkstation()
IF PL =7: shutdownWorkstation()


OP
PE
MR & AR
detectProblem()
notify()
notify()

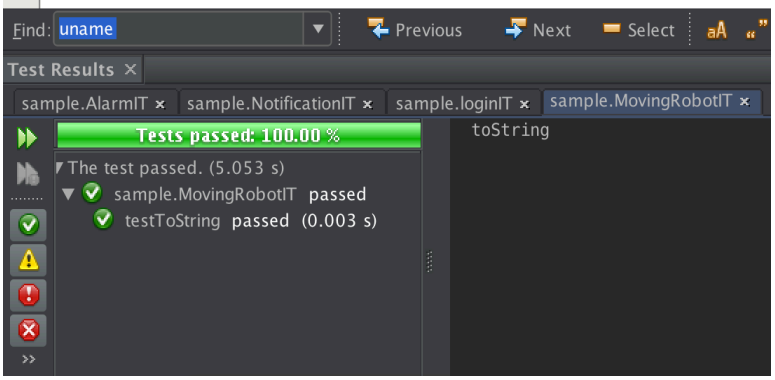
3.6.1 Test Plan

- Test Plan Identifier: Junit Testing
- Introduction: We used Junit testing provided to us by NetBeans. We tested all important classes and methods and did not test or retest similar working methods
- References: The project Management plan shows how we split our time on implementation of code. Testing the implementation of the code applied to everything but the GUI as GUI is front end and doesn't have many methods to test. Testing took half the time of implementing
- Test Items: Used Junit
- Features that were tested: Alarm, Database Connector, Login, Moving Robot, Notification, Process Engineer, Product Manager classes were all tested.
- Approach: Most of the time for simple methods such as set, get and return simple functioned values, I used black box.
    - For some complicated test cases forced me to use white box testing as I needed complex results to test the experimental and actual results with.
- I used standard AssertEquals and AssertTrue clauses to test the experimental and real values. Most values returned true or false for me to compare whether or not the tests passed.
- I'm testing based on the idea that all results whether experimental or real results, return a true value. If either of the results return false, then the test had failed.
- Estimate and schedule is provided in the PMP

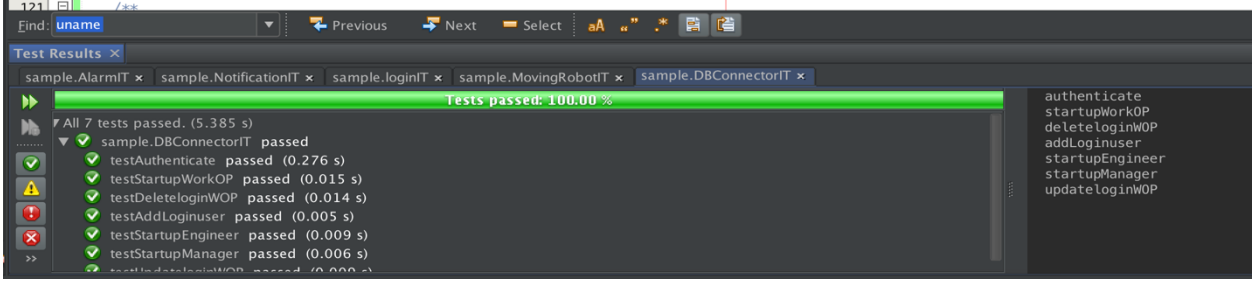
3.6 Test Cases



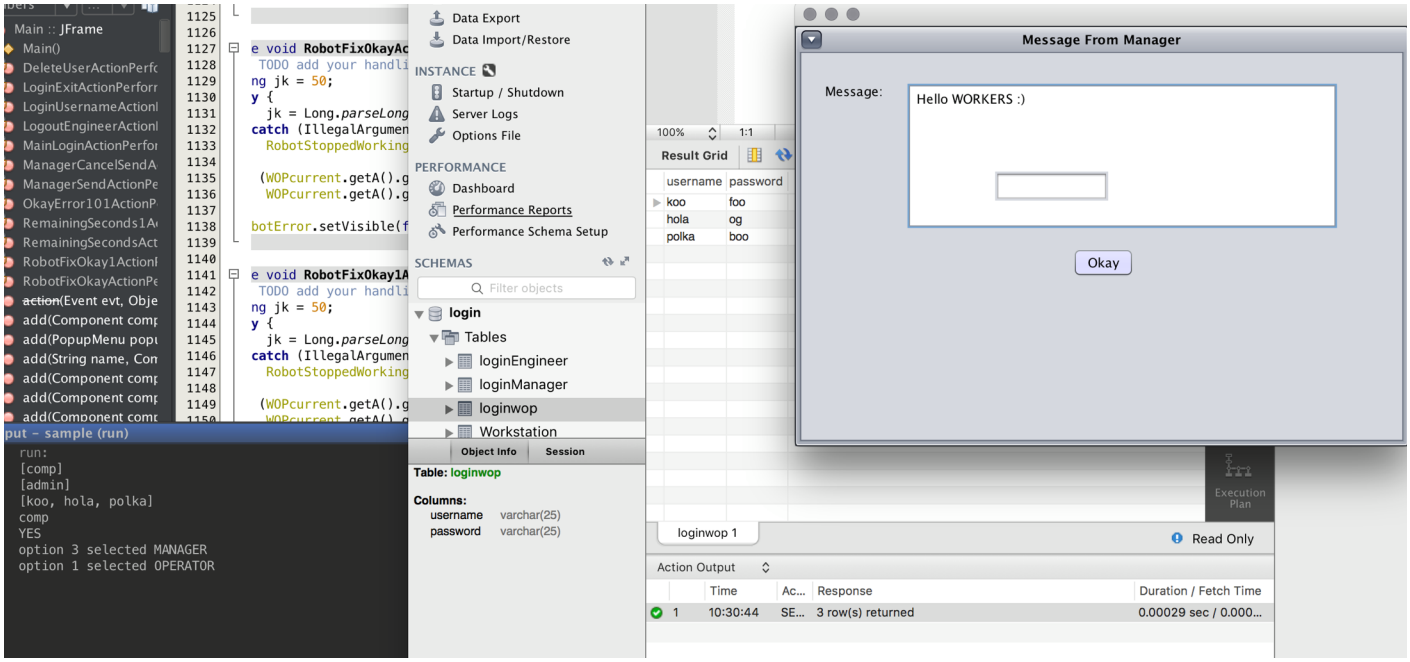
```
29      public static void tearDownClass() {
30      }
31
32      @Before
33      public void setUp() {
34      }
35
36      @After
37      public void tearDown() {
38      }
39
40      /**
41       * Test of toString method, of class MovingRobot.
42       */
43      @Test
44      public void testToString() {
45          System.out.println("toString");
47          MovingRobot instance = new MovingRobot(100);
48          String expResult = "Moving Robot ";
49          String result = instance.toString();
50          assertEquals(expResult, result);
51      }
52
53  }
```

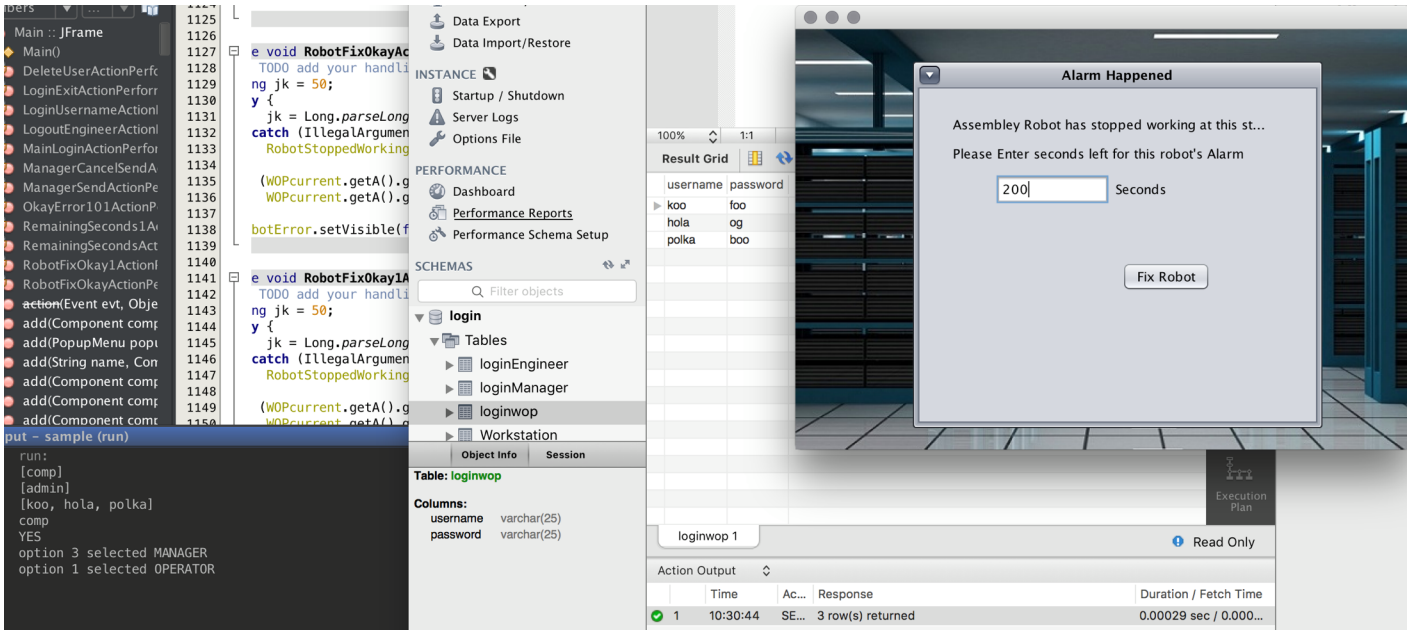


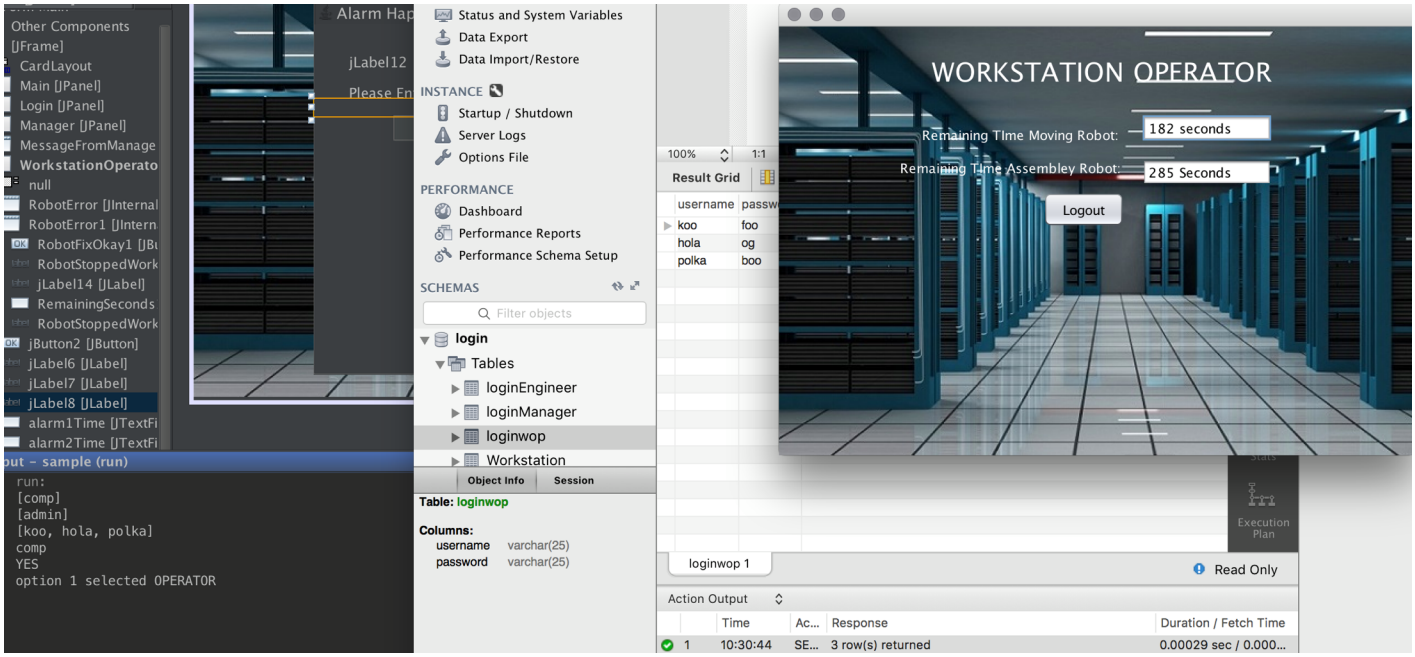

```
92          instance.addLoginuser(loginuser, loginpword, a);
93      }
94
95      /**
96       * Test of authenticate method, of class DBConnector.
97       */
98      @Test
99      public void testAuthenticate() {
100         System.out.println("authenticate");
101         String loginuser = "";
102         String loginpword = "";
103         int a = 0;
104         DBConnector instance = new DBConnector();
105         boolean expResult = false;
106         boolean result = instance.authenticate(loginuser, loginpword, a);
107         assertEquals(expResult, result);
108     }
109
110     /**
111      * Test of deleteloginWOP method, of class DBConnector.
112      */
113     @Test
114     public void testDeleteloginWOP() {
115         System.out.println("deleteloginWOP");
116         String deleteloginwop = "";
117         DBConnector instance = new DBConnector();
118         instance.deleteloginWOP(deleteloginwop);
119     }
120
121     /**
```

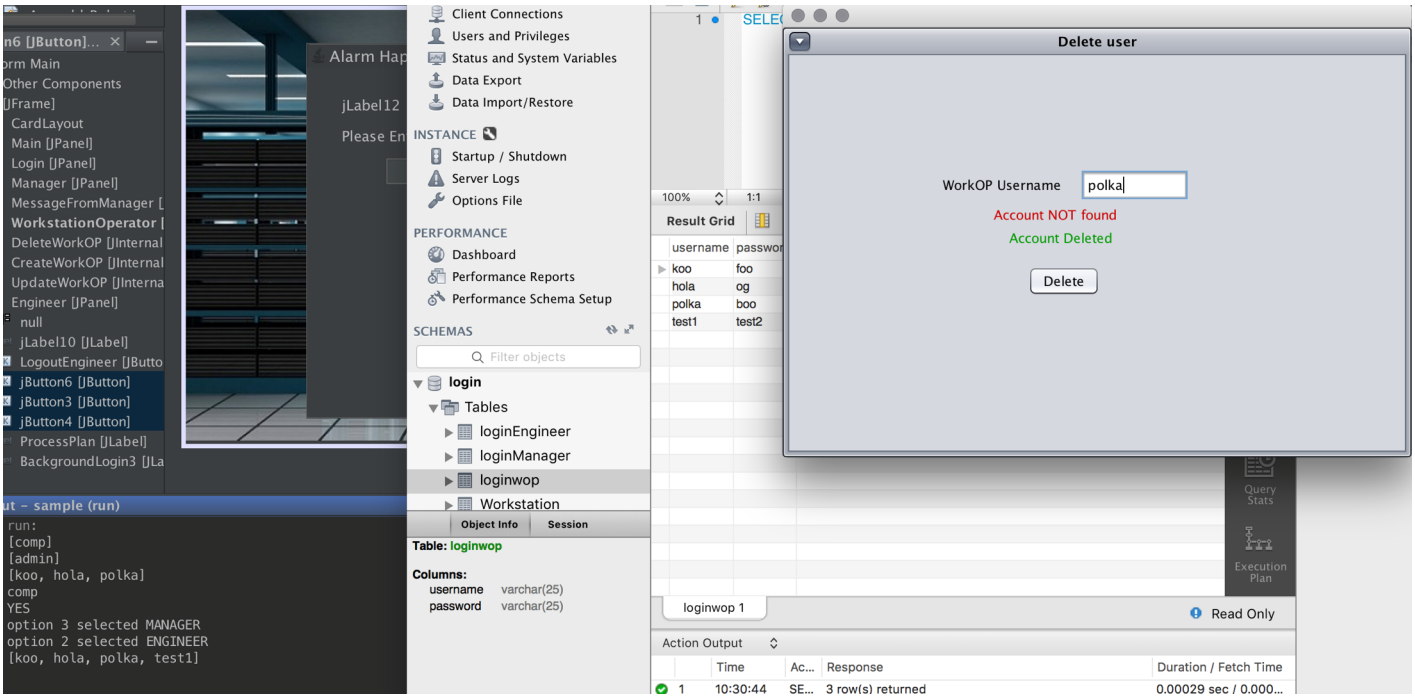


3.7 Sample Results






WORKSTATION OPERATOR
Remaining TIme Moving Robot:
182 seconds
Remaining Time Assembley Robot:
285 Seconds
Logout
run:
[comp]
[admin]
[koo, hola, polka]
comp
YES
option 1 selected OPERATOR
Table: loginwop
Columns:
username varchar(25)
password varchar(25)
Action Output
Time
Response
Duration / Fetch Time
1 10:30:44 SE... 3 row(s) returned 0.00029 sec / 0.000...


Delete user
WorkOP Username
polka
Account NOT found
Account Deleted
Delete
run:
[comp]
[admin]
[koo, hola, polka]
comp
YES
option 3 selected MANAGER
option 2 selected ENGINEER
[koo, hola, polka, test1]
Table: loginwop
Columns:
username varchar(25)
password varchar(25)
Action Output
Time
Response
Duration / Fetch Time
1 10:30:44 SE... 3 row(s) returned 0.00029 sec / 0.000...

3.8 Project Management Plan

Component	Cost	Time	Comments
Process Engineer	1 hour per module	5 hrs	• Changed degree of control extensively • Made sure that the PE didn't have control over the MR and AR • Acts as an advisor to the Op as opposed to a controller
Workstation Operator	0.5 hours per module	3 hrs	• Reduced the amount of actions available • Op can only solve problems and view status updates through the UI
Moving Robot	0.5 hours per module	2 hrs	• No change • Always meant to notify Op of move • Contains alarm function
Assembly Robot	0.5 hours per module	2 hrs	• Robot concrete class which contains alarm function • Belongs to the operator class
Notification/Main	2 hour per module	10 hour	• Takes a fundamental role in the observer pattern • Designed to send an object to the recipient • Every time something happens, Main alerts everything else
Alarm	0.5 hour per module	1 hour	• Subset of notification, or a type of notification • Similar attributes as updates but with an assigned priority level • Added a time variable of type long with a functioning countdown timer which allows the Op to reset to any number of seconds • Op must be logged in to make changes to the alarm
Product Manager	0.5 hours per module	1 hrs	• Mostly GUI • Acts as an email system w/ popups as emails
GUI	0.10 hours per module	15 hours	• Naming, all buttons, fields, and extracting backend to work with front end
DataBase	1 hour per module	3 hours	• Setting up MySQL and connecting java backend to server