**< WEEKLY REPORT FOR WEEK 7 >**

Name: Oh SuJin

Project: Multimodal Sensor Interfacing, Acquisition and Visualization

**I) Project Work Summary**

**Finished:**

* Simple Swing GUI for iFarm
* Integrating Leap Motion DAQ unit with the Drools assessment unit

**Ongoing:**

* Improving Drools rule engine
* Java-Unity integration

**II) Tasks Assigned**

**III) Detailed Activities / Accomplishments**

**DAY1**

* Problem with Drools.
* There was initial attempt to use Drools (Rule Engine) as an assessment unit for iFarm, where Java constantly feeds Drools with meaningful features such as tip velocity, for each time frame. Since Drools can be written in simple if-then statement, it is easy to understand and amend the assessment criteria and rules if required in the future. This makes iFarm more business-friendly and scalable.
* However, it was later found out that Leap motion sends up to 300 frames of tracking data per second and Drools is not designed to handle such constant stream of data. Due to time constraint, assessment is done within Java instead for the timebeing.

**DAY2&3**

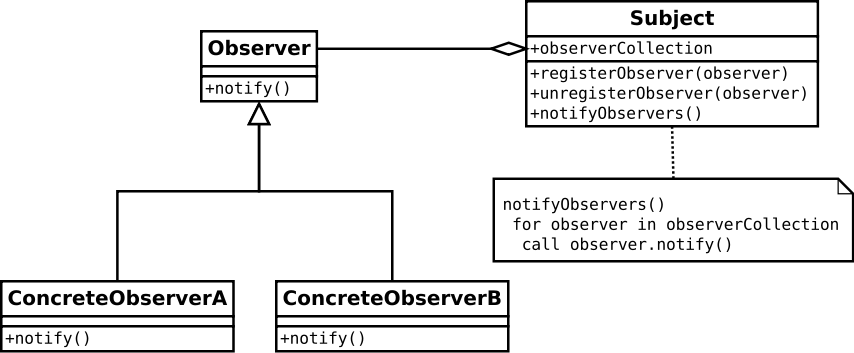
* I had difficulty integrating Drools with existing Leap motion data acquisition unit.
* Using**Observer Pattern** was considered.
* The *observer* pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.
* The object which is being watched is called the *subject*. The objects which are watching the state changes are called *observers* or *listeners*.
* The observer pattern is very common in Java. For example, you can define a listener for a button in a user interface. If the button is selected, the listener is notified and performs a certain action.

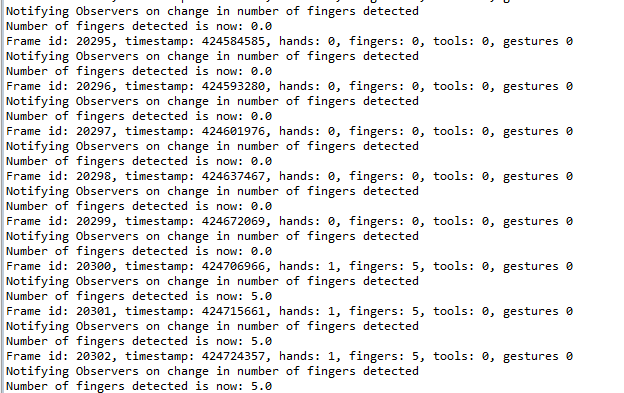
But the observer pattern is not limited to single user interface components. For example, you could have a part A in your application which displays the current temperature.

Another part B displays a green light if the temperature is above 20 degree celsius. To react to changes in the temperature, part B registers itself as a listener to Part A.

If the temperature in part A is changed, an event is triggered. This event is sent to all registered listeners, as, for example, part B. Part B receives the changed data and can adjust its display.

The following example code shows such a listener implementation for a button.





**DAY 4&5**

* Start designing simple GUI for iFarm presentation on week 10.
* Knowledge in java swing GUI is required.
* This simple GUI will display basic tracking information and contains buttons to select desired exercise.

