**Mobile Computing Project 2 - Reactive Engine**

**Problem Definition:**

To implement a reactive engine for ATLAS. The engine should be capable of displaying the basic events and actions available to the users. The users can build on these events and actions to create complex events and actions. There will be rules which are of the form R=Event, Condition, Action. On the triggering of an event, if a particular condition is true, the specified action should be taken.

**DESIGN AND IMPLEMENTATION**

The project is divided into two modules

1. GUI With Parsing functionality
2. Reactive Engine

**GUI Parsing Tool:**

The GUI parsing tools takes the input command from the user and performs the following actions.

LIST, LIST event, LIST action, LIST condition, LIST rule: The following commands display both the basic events and the user defined events, basic actions and user defined actions, conditions and rules.

BASIC, BASIC event, BASIC action: The following commands display basic events and basic actions.

DEFINE event: This command gives the user the flexibility to define his desired events. This command checks if the events from the new event is derived from are already present. If any of the f the event is not present it throws an error.

DEFINE action: This command gives the user the flexibility to define his desired actions. This command checks if the actions from the new action is derived from are already present. If any of the action is not present it throws an error.

DEFINE condition: This command enables the user to define new conditions.

DEFINE rule: This command helps to define new rules. This command checks if the event, condition and action that make the rule are valid. If they are not, it throws an error.

RUN: Starts the reactive engine.

SET: Helps to define conditions on the fly.

STOP: Stops the Reactive Engine.

**REACTIVE ENGINE:**

The reactive engine is responsible for the evaluation of events and conditions, checking if a rule is satisfied and if it is, the appropriate actions should be triggered.

**Architecture:**

The events, actions, conditions and rules are all stored as classes. These classes are entered into concurrent hashmaps for concurrent and threadsafe access by multiple threads. Each new event/action/condition or rule is stored as an object of the particular class in the hashmaps.

When data is received, the evaluation is triggered. Each event in the maps is evaluates for its truth values. The evaluation takes care of precedence of operators. A shift-reduce style algorithm is used to achieve this. For events of the form E\*seconds\*E, a separate evaluation is required. The event objects store timestamps on them with which we can determine if the event has occurred within the required duration. Once the events are evaluated, each event object contains its truth value and a timestamp when the event was updated.

Once the events are updated, the rules are retrieved and checked if the corresponding events and conditions are true. If so, the corresponding action is fetched and performed. In case of combination of two or more actions, they are performed sequentially in the order which they appear.

**WORKING FEATURES:**

Commands:

1. BASIC
2. LIST { action | event | rule | condition }
3. DEFINE {action | event | condition | rule }
4. SET
5. RUN
6. STOP
7. LOAD

Event Evaluation:

1. Simple events : e1=H3(78)
2. Complex events: e1=e2+e3\*e4
3. Composite Events: e1=e2\*seconds\*e3

Conditions:

1. SET

Actions:

1. Simple actions : a1=servo\_R8(100)
2. Combination of actions : a1=a2;a3;a4

**NON – WORKING FEATURES:**

GUI input may not detect all possible errors. Most of the validations have been covered.

The time spaced events are not bound exactly by the time limit, this is due to the dependence on when the data received callback method is called by the emulator.

**CHALLENGES:**

Building the time spaced event evaluation was a challenge as the token “\*sec\*” also contains the and operators “\*”. Hence this had to be handled specially to make it work. Also tracking and calculating accurately the time difference of the events was a challenge.

Building the GUI parser was a challenge considering the Grammar and the user freedom.

Getting the values of the sensors and moving the servo were relatively simple given the object oriented nature of the framework and the supplied packages.