

1 LowLevel

The low level provides the high level with an interface to execute tasks (*our project statement will go here*).

Lowlevel
Methods +LowLevel(); +boolean executeTask(Task t); +Status getStatus();
Attributes -TaskExecutor taskExecutor; //responsible for translating a task into actions and issuing //the actions to the hardware -TaskMonitor taskMonitor; //responsible for monitoring the status of the hardware system //and detecting any errors that occur -HardwareInterface hardwareInterface; //provides communication between the low level //and the simulation

1.1 LowLevel()

This method initializes a new low level object.

Pseudo Code:

```
public LowLevel()  
{  
    taskExecutor = new TaskExecutor();  
    taskMonitor = new TaskMonitor();  
    hardwareInterface = new HardwareInterface();  
}
```

1.2 executeTask(Task t)

This method takes a task given to it by the high level and translates it into actions that are sent to the hardware interface.

Pseudo Code:

```
public boolean executeTask(Task t)  
{  
    return taskExecutor.executeTask( t );  
}
```

1.3 Status getStatus()

This method sends status information back to high level.

Pseudo Code:

```
public Status getStatus()  
{  
    return taskMonitor.getStatus();  
}
```

2 TaskExecutor

The TaskExecutor is responsible for translating a given task into actions that are then sent to the hardware interface.

TaskExecutor
Methods +TaskExecutor(); +boolean executeTask(Task t);
Attributes #ActionSequenceSelector actionSequenceSelector; #ActionSequenceExecutor actionSequenceExecutor;

2.1 TaskExecutor()

This method is responsible for initializing a new TaskExecutor object.

Pseudo Code:

```
public TaskExecutor()  
{  
    actionSequenceSelector = new ActionSequenceSelector();  
    actionSequenceExecutor = new ActionSequenceExecutor();  
}
```

2.2 boolean executeTask(Task t)

This method takes a give task and breaks it up into actions sequences which are then sent to the hardware interface.

Pseudo Code:

```
public boolean executeTask(Task t)  
{  
    ActionSequence as;  
    as = actionSequenceSelector.getActionSequence( t );  
    return actionSequenceExecutor.executeActionSequence( as );  
}
```

3 ActionSequenceSelector

The ActionSequenceSelector selects action sequences for a given task. This class provides a point where ActionSequences can be intelligently selected.

ActionSequenceSelector
Methods

```
+ActionSequenceSelector();  
+ActionSequence getActionSequence(Task t);  
# ActionSequence[] retrieveActionSequences(Task t);
```

Attributes

```
#ActionSequenceDataStore actionSequenceDataStore;
```

3.1 ActionSequenceSelector()

This method initializes a new ActionSequenceSelector object.

Pseudo Code:

```
public ActionSequenceSelector()  
{  
    initialize connection to data store.  
}
```

3.2 ActionSequence getActionSequence(Task t)

This method returns an action sequence that accomplishes the given task. If more than one action sequence exist for the given the task, the most suitable action sequence is selected.

Pseudo Code:

```
public ActionSequence getActionSequence(Task t)  
{  
    ActionSequence[] sequences;  
    ActionSequence as;  
    sequences = retrieveActionSequences( t );  
  
    intelligent selection happens here and a single action sequence  
    is returned  
  
    return as;  
}
```

3.3 ActionSequence[] retrieveActionSequences(Task t)

This method queries the ActionSequenceDataStore for all action sequence that will fullfill the given task

Pseudo Code:

```
protected ActionSequence[] retrieveActionSequences(Task t)  
{  
    query data store for all action sequences for given task  
}
```

4 ActionSequenceDataStore

The ActionSequenceDataStore interfaces with the action sequence data store.

```
ActionSequenceDataStore
```

Methods +ActionSequence[] getActionSequences(Task t);

4.1 ActionSequence[] getActionSequence(Task t)

This method queries the data store for every action sequence that is related to the given task.

Pseudo Code:

```
public ActionSequence[] getActionSequence(Task t)
{
    query data store for list of action sequences for the give task
}
```

5 ActionSequenceExecutor

The ActionExecutor issues actions to the simulation interface.

ActionSequenceExecutor extends Thread
Methods +ActionSequenceExecutor(); +boolean executeActionSequence(ActionSequence as); +void run(); +void suspendExecution(); +void resumeExecution(); +void stopThread();
Attributes #String timeStamp; #ActionSequence as; #boolean isRunning; #boolean isActions; #boolean isExecuting;

5.1 ActionSequenceExecutor()

This method initializes a new ActionSequenceExecutor object.

Pseudo Code:

```
boolean executeActionSequence(ActionSequence as)
{
    timeStamp = new String;
    //other initialization if anything
}
```

5.2 boolean executeActionSequence(ActionSequence as)

This method takes an action sequence and waits for each action in the sequence to complete.

Pseudo Code:

```
boolean executeActionSequence(ActionSequence as)
{
    this.as = as;
    start executing each action
    while( isActions )
        Sleep();
    return true if every thing is successful else false.
}
```

5.3 void run()

This method is responsible for actually executing each action in the action sequence. This method can be suspended and resumed.

Pseudo Code:

```
public void run()
{
    while( isRunning )
    {
        while( isActions && isExecuting )
        {
            get next action
            time stamp action and add any arguments
            execute next action
            wait for response
        }
    }
}
```

5.4 void suspendExecution()

This method suspends execution of the ActionSequenceExecutor.

Pseudo Code:

```
public void suspendExecution()
{
    isExecuting = false;
}
```

5.5 void resumeExecution()

This method resumes execution of the ActionSequenceExecutor.

Pseudo Code:

```
public void resumeExecution()
{
    isExecuting = true;
}
```

5.6 *void stopThread()*

This method stops the ActionSequenceExecutor.

Pseudo Code:

```
public void stopThread()
{
    isRunning = false;
}
```

6 HardwareInterface

The HardwareInterface provides interface for communication between the low level system and the simulated hardware.

abstract HardwareInterface
Methods +HardwareInterface(); +void executeAction(Action a);
Attributes #InterruptReader interruptReader; #DataReader dataReader; #BufferedWriter out;

6.1 *public HardwareInterface()*

This method initializes a new HardwareInterface object. A connection is made to the simulation and the appropriate read threads are created.

Pseudo Code:

```
public HardwareInterface()
{
    initialize connection to simulation

    interruptReader = new InterruptReader(stderr //from simulation );
    dataReader = new DataReader( stdin //from simulation );

    interruptReader.start();
    dataReader.start();
}
```

6.2 *void executeAction(Action a)*

This method executes a give action.

Pseudo Code:

```
public void executeAction(Action a)
{
    out.write(":at-time " + a.toString() );
}
```

7 InterruptReader

The InterruptReader is responsible for reading interrupts sent from the simulation which are sent via standard error stream.

InterruptReader extends Thread
Methods +InterruptReader(BufferedReader b); +void run(); +void start(); +void stopThread(); +boolean isRunning();
Attributes #BufferedReader read;

7.1 *InterruptReader(BufferedReader b)*

7.2 *void run()*

7.3 *void start()*

7.4 *void stopThread()*

7.5 *boolean isRunning()*

8 DataReader

The DataReader is responsible for reading messages for the TaskMonitor and TaskExecutor sent by the simulation. The messages are sent by means of standard input.

DataReader extends Thread
Methods +DataReader(BufferedReader b); +void run(); +void start(); +void stopThread(); +boolean isRunning();
Attributes #BufferedReader read;

8.1 *DataReader(BufferedReader b)*

8.2 *void run()*

8.3 *void start()*

8.4 *void stopThread()*

8.5 *boolean isRunning()*

9 Action

Action
Methods +String toString();
Attributes #int day; #int milliseconds; #int deviceId; #int command; #double args[];

9.1 *public String toString()*

This method converts an action into string form.

Pseudo Code:

```
public String toString()
{
    String actionString = new String("(" + day + " "
        + milliseconds
        + " ) ( "
        + deviceId
        + " "
        + command );

    for(int x = 0; x < args.length(); x++)
    {
        actionString += args[x];
    }

    actionString += ")";

    return actionString;
}
```

10 Task

Up for discussion with ANTS-PSR.

11 ActionSequence

An action sequence is a list of actions.

ActionSequence
Methods +ActionSequence(Action action[]); +Action peek(); +pop();
Attributes #Stack actionStack;

11.1 ActionSequence(Action action[])

This method creates a new ActionSequence object. The object is passed an array of actions.

Pseudo Code:

```
public ActionSequence(Action action[])
{
    actionStack = new Stack();
    for(int x = (action.length() - 1); x >= 0; x--)
    {
        actionStack.push( (Object) action[x] );
    }
}
```

11.2 Action peek()

This method returns the next action to be executed.

Pseudo Code:

```
public Action peek()
{
    return (Action) actionStack.peek();
}
```

11.3 void pop()

This method removes the next action.

Pseudo Code:

```
public void pop()
{
    actionStack.pop();
}
```

12 FeedBack

FeedBack

13 Status

This is up for discussion with ANTS-PSR.