Introduction to Java and Agent-Based Economic Platforms (CF-904)

3. Simulations with JAS

Mr. Simone Giansante
Email: sgians@essex.ac.uk
Web: http://privatewww.essex.ac.uk/~sgians/
Office: 2.522

SimModel super class (1)

- The Model and the Observer classes have to extend the super class “SimModel”

```
public class Model extends SimModel
{ .....
}
```

- It will require two unimplemented methods:
  - “setParameters()” for initializing variables and create a graphic probe

```
public void setParameters()
 {
            numAgents = 10;
            Sim.openProbe(this, "parameters");
}
```

SimModel super class (2)

- "buildModel()" for building the objects of the simulations

```
public void buildModel()
 {
        for (int i = 0; i < numAgents; i ++)
        {
                Agent agent = new Agent (i);
        }
        .......
}
```

Schedule events (1)

- The package ***jas.events*** contains the events hierarchy, used by the event list to schedule events in time.

Class Summary	
EventFactory	Create instances of any type of Event and manages a garbage list.
EventList	The event list manages a time ordered list of events.
RealTimeEventList	It extends the simple event list.
SimCollectionEvent	It is able to inform all elements within a collection about an event.
SimEvent	An abstract class that any event must override.
SimGroupEvent	A special implementation of the SimEvent familiy.
SimMultiCastEvent	This event is able to inform different kind objects at the same time.
SimSimpleEvent	The simpler implementation of SimEvent class.
SimSystemEvent	System events are directly processed by the simulation engine.

http://jaslibrary.sourceforge.net/docs/index.html

Schedule events (2)

- The two common methods for scheduling events are:
  - *eventList.scheduleSimple(long atTime, int withLoop, java.lang.Object object, java.lang.String method)*

It will execute "*method*" of "*object*" every "*withLoop*" steps starting from the step *"atTime"*.

```
eventList.scheduleSimple(0, 1, this, "step");
```

  - *eventList.scheduleCollection(long atTime, int withLoop, java.util.Collection elements, java.lang.Class objectType, java.lang.String method)*

It will execute "*method*" for all objects "*objectType*" collected in "*elements*" every "*withLoop*" steps starting from the step *"atTime"*.

```
eventList.scheduleCollection(0, 1, agentList, getObjectClass("Agent"), "step");
```

Observer and GUIs (1)

- All graphic objects are managed by the Observer class (that extends SimModel)
- The Observer has to create an instance to the Model class in order to plot results of your simulation. In order to syncronize the Observer with the Model, we have to create an istance to the model in the following way:

```
model = (Model) Sim.engine.getModelWithID("Model");
```

The method "*getModelWithID*" returns the object *Model* that has beed already created and collected by the the JAS' engine

Observer and GUIs (2)

- The package jas.graphics.plot contains statistical plotters, which are able to work with jas.statistics interfaces.

Class Summary	
CollectionBarPlotter	A bar chart plotter dynamically showing elements in a CrossSection object.
IndividualBarPlotter	A bar chart plotter showing elements manually added by user.
TimeSeriesPlotter	A time series plotter is able to trace one or more data sources over time.

Observer and GUIs (3)

- CollectionBarPlotter
  - It is a bar chart plotter that dynamically shows elements in a CrossSection object
  - A cross section is a collection of values each of them representing the status of a given variable of an element of a collection of agents).
    - **CrossSection.Double**
    - **CrossSection.Integer**
    - **CrossSection.Float**
    - **CrossSection.Long**

Observer and GUIs (4)

- **You can use a CrossSection object to collect values from a collection of objects, i.e. your agentList**

CrossSection.Double

```
public CrossSection.Double(java.util.Collection source,
                           java.lang.Class objectClass,
                           java.lang.String valueName,
                           boolean getFromMethod)
```

Create a basic statistic probe on a collection of objects.

Parameters:

`name` - Name of the statistic object.
`source` - A collection of generic objects.
`objectClass` - The class of the objects contained by collection source.
`valueName` - The name of the field or the method returning the variable to be probed.
`getFromMethod` - Specifies if valueName is a method or a property value.

Example from HeatBugs:

```
CrossSection.Double UnHappiness = new CrossSection.Double(model.heatBugList,
    HeatBug.class, "unhappiness", false);
```

The CrossSection "*UnHappiness*" will collect the value of the variable "*unhappiness*" of all the objects in the "*heatBugList*"

Observer and GUIs (5)

- **You can create a CollectionBarPlotter and add a CrossSection object in the following way:**

```
CollectionBarPlotter plotter = new CollectionBarPlotter(“name of the plot");
plotter.addSeries (“name of the series”, crossSectionObject);
```

Observer and GUIs (6)

- **IndividualBarPlotter**
  - A bar chart plotter showing elements manually added by user

```
IndividualBarPlotter plotter = new IndividualBarPlotter (“name of the plot");
plotter.addSource (java.lang.String legend, java.lang.Object target,
      java.lang.String variableName, boolean getFromMethod)
```

where:

- *legend* is the name of the source
- *target* is the object where the value is collected
- *variableName* is the name of the variable/method
- *getFromMethod* has to be true if *variableName* is a method and false if *variableName* is a variable

Observer and GUIs (7)

- **TimeSeriesPlotter**
  - A time series plotter is able to trace one or more data sources over time.

```
TimeSeriesPlotter plotter = new TimeSeriesPlotter (“name of the plot");
plotter.addSeries (java.lang.String legend, java.lang.Object target,
      java.lang.String variableName, boolean getFromMethod)
```

where:

- *legend* is the name of the source
- *target* is the object where the value is collected
- *variableName* is the name of the variable/method
- *getFromMethod* has to be true if *variableName* is a method and false if *variableName* is a variable

Observer and GUIs (8)

- **In order to add your plotter in the JAS graphic interface you have to use the method *addSimWindows* in the following way:**

```
addSimWindow(plotter);
```

- **In order to update you plotter over time you have to use the following event:**

```
eventList.scheduleSimple(atTime, withLoop, , plotter, Sim.EVENT_UPDATE);
```

where:

- *atTime* identify the step of the first updating
- *withLoop* is the frequency of the updating
- *plotter* is the Plotter Object
- Sim.*EVENT_UPDATE* refer to a JAS function for the updating

Statistic interfaces

- Plotters are able to work with jas.statistics interfaces.

  http://jaslibrary.sourceforge.net/files/JAS-Statistics-Howto.0.1.pdf