UNIVERSITÀ DEGLI STUDI DI TORINO
FACOLTÀ DI ECONOMIA

DIPARTIMENTO DI SCIENZE ECONOMICHE E FINANZIARIE G. PRATO

THE USE OF PTHISTOGRAM CLASS TO DRAW HISTOGRAMS AND BAR CHARTS IN A SWARM PROJECT

Dario Landini

1 The Ptolemy project and the *ptplot* library

Ptolemy is a project developed by the *Department of Electrical engineering and Computer Science* of the *University of California, Berkeley,* to simulate and solve complex problems in real time (more informations can be found at: *http://ptolemy.eecs.berkeley.edu/*).

The Ptolemy project contains the *ptplot* package, a graphical pakage developed in Java. In particular we are referring to the java archives *plot.jar* and *gui.jar*. These libraries include many classes to draw graphics and it is possible to use them to improve the output of *Swarm* models.

The libraries can be downloaded from:

http://ptolemy.eecs.berkeley.edu/java/ptplot5.1p1/ptolemy/plot/doc/index.htm

according to the copyright published at the beginning of each class[1]

These functions are used in *Java Virtual Enterprise (JVE)*[2] model to draw the waiting lists of units and the quantities stored in warehouses.

[1]

[2]Developed by Pietro Terna and his students at the University of Turin, Italy.

Our aim is to show the way to link the *Ptolemy* plot aspects with *Swarm*, via the following example.

2 How to use PTHistogram to draw graphs

The example is the PTHistogram class of JVE project.

The class has three constructors, to define:

- the title of graph
- the label of X axis
- the label of Y axis
- the right dimension of X axis
- the top dimension limit
- the series of data to be drawn:
  1. one in the first constructors
  2. two in the second
  3. three in the third one

With the constructors we are also defining:

- the dimension of the bars and how they have to be centered on the axis[3]:

```
setBinOffset(1)
setBinWidth(0.5);
```

[3]The value 1 in the method *setBinOffset* has been chosen because it fixes the bars at the centre of the value; according to the relation: $x - \frac{w}{2}$; $x + \frac{w}{2}$, when 1 substitutes w, the bar is exactly divided in the middle.

- the legend of the graph; it could be useful whenever you need to specify more than one dataset (represented by a number starting from 0) to show on the same value more than one bar. For example, in our case, when you need to represent two or three series of data on the same graph. So we can have:

```
addLegend(0, "Queues in u."); // if we use just the first
constructor
..... or .....
addLegend(0, "Queues in u.");
addLegend(1, "Quant. in w."); // if we use the second
constructor with two series of data, etc...
```

- the dimension of the window and its position on the screen, according to the use of javax.swing.JFrame properties:

```
myFrame.setBounds(10,300,windowWidth,windowHigh);
```

- the choice of showing the buttons for printing, resizing, redefining the graph during the simulation:

```
setButtons(true);
```

To draw the data value it's necessary just a method, called **addPoints**, which has the aim to scroll the lists passed by the constructor:

```
for (int i=0;i<fL.getCount();i++){
      //put in unit the object of the first list imported
      unit = (Unit) fL.atOffset(i);
```

then for asking them about the number of data contained (useful to put the right ticks on the X axis)

```
Integer f = (new Integer (unit.unitNumber));
      String singleLabels = f.toString();
      //add the labels on x axis
      addXTick(singleLabels , i);
```

and, at the end, for drawing to points necessary to represent the required lenght of the bar:

```
for(int j=0;j<unit.getWaitingListLength();j++){
                addPoint(0, (double) i);
            }
```

as for the legend, also for addPoint it is necessary to specify to which dataset it is referreing on (in this case 0).

This process has to be be repeated any time you need to show a new dataset on the same graph.

To work in this way it's necessary to set at the beginning of the method :

```
clear(false);
```

which clears the graph, but not changes its format. At the end of the method:

```
repaint();
```

which executed the clear instruction.

3 PTHistogram in Swarm

It's very simple to use PTHistogram in Swarm, according to the standard of the *ObserverSwarm* you have to:

- build the object in the observer method *buildObjects()*, choosing with a conditional *if/else* to which constructor you are referring:

```
if(vEFrameModelSwarm.getUseWarehouses()==true){
ptHistogram = new PTHistogram("Orders", "Virtual Enterprise",
            "Count", vEFrameModelSwarm.totalUnitNumber, 20.0,
            vEFrameModelSwarm.unitList,
            vEFrameModelSwarm.warehouseList);
}
else ptHistogram = new PTHistogram("Orders", "Virtual Enterprise",
                 "Count", vEFrameModelSwarm.totalUnitNumber, 20.0,
                 vEFrameModelSwarm.unitList);
```

- execute the method *addPoints* of PTHistogram class in the *schedule* of the project, so add in the *buildActions()*:

```
displayActions.createActionTo$message
        (ptHistogram,
          SwarmUtils.getSelector(ptHistogram, "addPoints"));
```

4 Attachments

In attachment you will find:

1. the class PTHistogram.java, which can be used like described in the paragraph 3.
2. the libraries *plot.jar* and *gui.jar*.
3. a directory, which contains the *.gif* files to show the buttons on the histogram. It has to be copied and pasted in the current directory of the project.