Notes on Naval Simulation

Murat M. Gunal

Industrial Engineering Department
Turkish Naval Academy
Tuzla, Istanbul, Turkey

Draft v.0.2

July - 2010

Genoa, ITALY

Contents

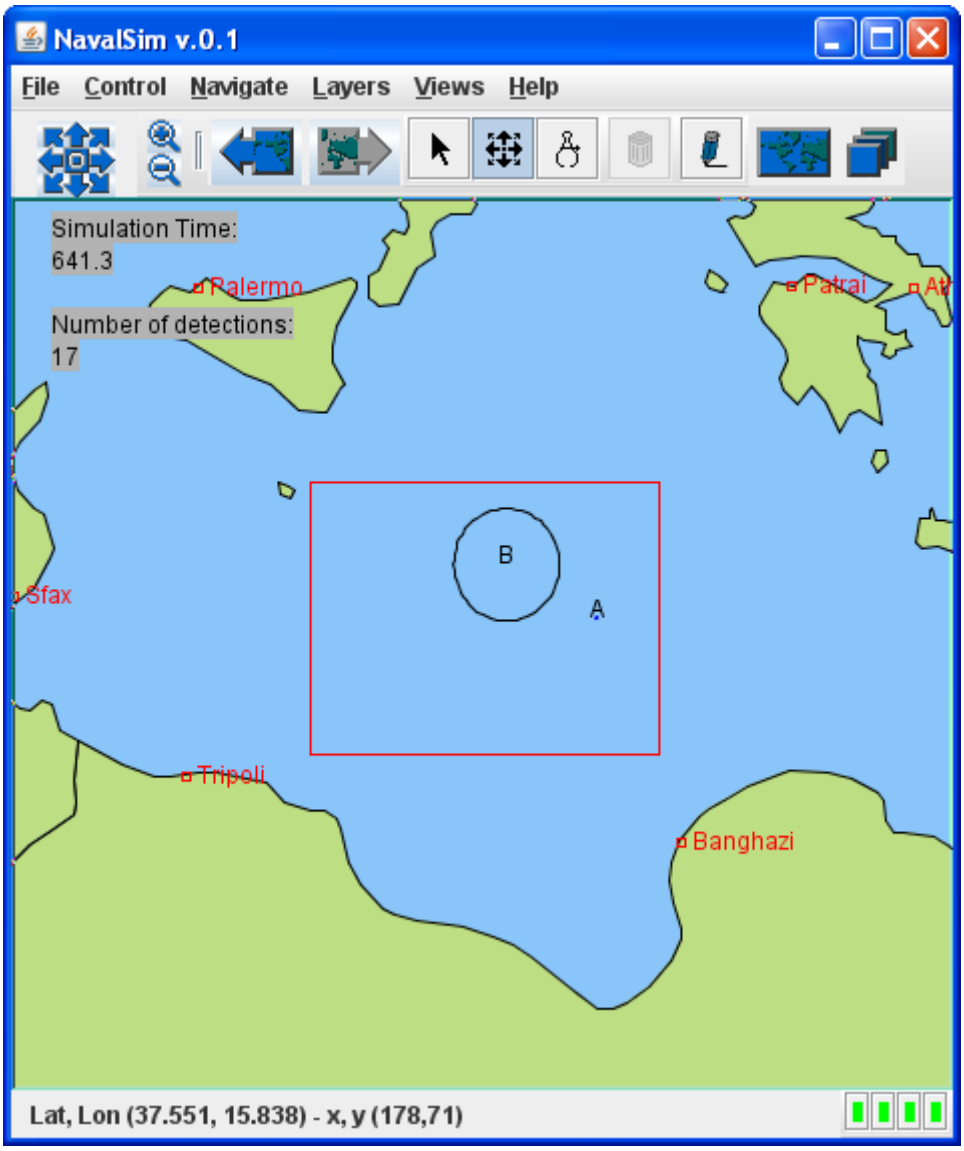
1 INTRODUCTION

The purpose of this note is to show how a basic model of naval operations can be built. This note is written as a tutorial rather than a text to explain the concepts in naval simulation modelling. The readers are assumed that they can understand Java programming language.

Using this tutorial, it is aimed that the reader can build the model which its user interface is shown in Figure 1-1. In this model the following packages are used;

- OpenMap (http://www.openmap.org)
- Simkit (http://diana.nps.edu/Simkit/)

Figure 1-1 Screen shot of the model.



The user interface in Figure 1-1 is the default interface of OpenMap package, with some minor additions. This model uses graphical features of OpenMap package. You extend its default appearance by adding new features to it. For example the A and B letters, and circles around them, are moving simulation entities, and the texts "Simulation Time" and "Number of Detections" on the top left corner are simulation texts. By writing Java code, you programme these simulation objects.

The model presented in this notes is based on conceptualisation of movement and detection in Simkit (Buss and Sanchez, 2005).

1.1 OpenMap and Simkit

The OpenMap package is a Geographical Information System (GIS) package written in Java. It is open source and therefore full source code can be downloaded from the web site. In addition, the package comes with the API which the programmer can use class libraries and extend them if needed.

The Simkit package is a Discrete Event Simulation (DES) code library also written in Java. It is developed by Buss (2001). Although it is a generic simulation package, it performs better in modelling defence systems such as naval operations. This package is taught at graduate level simulation course in the U.S. Naval Postgraduate School (NPS). There are ample notes and examples on the website on how a Simkit model can be built.

There is no direct link between these two pieces of software. However, as this tutorial shows it is possible to link them two and use together. This link is first established in a masters thesis in NPS (Mack, 2000) and showed that they can work well together. Simkit and a different GIS (GeoKit) is used in another study (Yildirim et.al,2009) where military deployment operations are analysed. Buss and Ahner (2006) is one of the other applications of Simkit.

Distinction of this modelling exercise is that it uses ready-made classes of Simkit and OpenMap. Therefore, it allows fast and minimum effort modelling. However, the modeller can still extend the work here, for example by adding rules of engagement to the vessels.

Since OpenMap and Simkit are written in Java, we will develop our model in Java too. There are a few Java compilers in the market,

such as Borland's, Microsoft's, and Sun's. You may use any of these compilers to build this model, however in this tutorial, Eclipse (ref.), a free Java programming environment is used.

1.2 The Model Scope

The model in this exercise is a fictional case. It is a simple search and detection scenario on the sea. There is a searcher (B) and there is a target (A), shown in Figure 1-1. They only move in an area which its boundaries are known (the red rectangle). The target moves linearly and patrols along the middle line of the rectangle. The searcher, however, moves linearly but in random piecewise routes, e.g. starting and ending locations of each route is randomly determined.

2 BACKGROUND INFORMATION

In this section a general overview on Geographical Information Systems and simulation software is given.

2.1 Geographical Information Systems (GIS)

A GIS is software to deal with the earth's geography. There are many GIS packages and tools in the market. ARCView, Google Maps, GeoKIT, NASA World Wind, Digital Nautical Chart, Falcon View, and OpenMap are some of the best known GIS software. Among these software some of them require licence, in which case you need to pay for it, but some others are freely available and open source, such as OpenMap. OpenMap is a free and open source software written in Java programming language. This is the main purpose why this GIS is chosen in this modelling exercise, first it is free and open source, and second it is written in Java.

GIS software actually manipulates geographical digital data. What makes a GIS software valuable is the data supplied to it. It is like a satellite navigation on a car, it is useless without the map data. Whatever type of data is supplied to a GIS, it shows that data to the user. If, for example, you are interested in vehicle routing on roads, then the road data, which is generally represented by vectors, must be given to the software.

Three types of digital map data are mentioned here.

- Vector maps
- Elevation maps
- Raster maps

A vector map consist of vectors, or lines in a way, which their start and end points are known. Think about shore lines or political boundaries, or rivers, or roads in a town. You can draw these shapes with lines, either straight or curved on a paper. Curved lines are more difficult to represent on computer since they need more parameters than a straight line needs. For a straight line, or say for multiple straight and connected lines, you need to keep starting point and ending point of each line. By a "point" we meant a point on earth which means a pair of latitude and longitude. On a 2 dimensional computer screen it is easy to covert world coordinate system (Lat, Lon pair) to screen coordinate system (X, Y pair).

There are different format of elevation data. Most common one is the Digital Terrain Elevation Data (DTED). DTED simply keep grid based elevation data, that is the earth is divided into squares and for each square there is an elevation value. Then the problem is what the dimensions of these squares are. This is a matter of resolutions and in standards there are three levels of resolution; in DTED level 0 (DTED0), a square is X by X, in DTED level 1 (DTED1) it is X by X, and finally in DTED level 2 (DTED2) it is X by X. (ref.).

Raster maps are real maps which are scanned and converted to digital format. It is convenient to see a real map on the screen but in digital format it is not of much use, since the data is actually a scanned image.

On these three types of map data you can also keep additional text based information, such as the name of a location (a point on earth), the length of a road (a vector).

As mentioned earlier a GIS software generally manipulates the data supplied to it. It can handle different data at once. The data generally displayed as "layers", which you can turn on and off. For example you can show a vector map together with DTED map. In this note, a simulation model is also represented as a layer in OpenMap. Therefore you can observe simulation with the map data.

2.2 Simulation Software

There are many simulation software in the market and it is difficult to compare them here. Most of these software, one way or another, has capabilities for simulating naval operations. In a simple way, for example if you want to model a moving vessel on sea, the naval vessel is not much different than a truck on the road. It must follow a route and interact with its environment.

You could model the system described here by using any other COTS simulation software, such as Arena and MicroSaintSharp. However Simkit is more powerful than the others since it has ready made objects for combat simulations.

3 BUILDING A BASIC NAVAL MODEL

As discussed in the previous sections, we will first setup the GIS tool and then write a simulation layer for our simulation model. Later the model will be built.

3.1 OpenMap

Before writing any code in Java, you need to setup OpenMap. The setup requires editing the OpenMap input file and the batch file for running the application.

3.1.1 Input file (openmap.properties)

"openmap.properties" file is the input file for OpenMap GIS package. This file is a text file which you can edit by using any text editor, such as Windows' Notepad. You can customize the user interface of OpenMap by adding and deleting appropriate lines in this file.

To add your SimulationLayer, you must add the following two lines at the end of this file.

```
#----------------------------
myLayer.class=NavalSim.SimulationLayer
myLayer.prettyName=My Simulation Layer
```

These two lines indicate where your layer's class file is located, and what label (My Simulation Layer) appears in the Layers menu on the user interface. Note that in this input file, the sign "#" is the comment character, in other words, the sentence beginning with this sign is not read by the programme.

After indicating the location of your class file, you must then tell OpenMap that you want to show this layer. To do this you need to add "myLayer" word in the "openmap.layers" string, as shown below.

```
openmap.layers=myLayer date SimExtra cities
shapePolitical
```

In this statement you also see other layers such as date, SimExtra, cities. These layer are shown in the same order as its written here. This is important since the mouse control is given to the top layer.

Note that as class and prettyName properties are written for myLayer, each of these layers have also property lines in the input file. A layer may also have extra properties that are entered in the input file. MyLayer, or the Simulation Layer, has only these default two properties. If you want to parameterize this layer in the future, then you need to write “getProperties” method in you layer class.

OpenMap is a GIS software and it can only work with the GIS data, such as vector maps, Digital Terrain Elevation Data (DTED) maps, raster maps. These maps are actually files in your computers file system and it is your task to tell OpenMap where these files are located.

3.1.2 Running the application

Although a Java environment, such as Eclipse, is advised to be used in this modelling exercise, it is worthy to explain how your Java application can be run on a PC, without any programming environment. To run a Java application, such as this Naval Simulation model, Java Runtime Environment (JRE) must be installed on the computer. JRE provides the runtime files needed.

To run the OpenMap user interface, the easiest way is to write a batch file for DOS environment. A batch file is a text file which includes DOS commands. A batch file to run the application is shown below.

```
rem Java Virtual Machine
set JAVABIN=java.exe

rem NavalSim folder
set NavalSim_HOME=C:\MyJavaProjects\NavalSim

rem OpenMap folder
set OPENMAP_HOME=%DenizSim_HOME%\lib\openmap.jar

rem Classpath for Java
set CLASSPATH=%OPENMAP_HOME%;
%DenizSim_HOME%\inputs\;
%DenizSim_HOME%\bin\;
%DenizSim_HOME%\lib\;
%DenizSim_HOME%\lib\simkit.jar;

rem Run the application
%JAVABIN% -mx256m -Dopenmap.configDir=%DenizSim_HOME% -
Ddebug.showprogress com.bbn.openmap.app.OpenMap
```

“rem” in a batch file is like a comment character, which means that a line starting with “rem” will not be taken into account and will not be processed. The statement starting with “set” command is to

set the environmental variables, such the first set line tells the operating system that JAVABIN variable's value is java.exe. Likewise you can set NavalSim_HOME and OPENMAP_HOME variables. In these two variables you basically set where your executables, or class files, are located.

Note that the libraries openmap.jar and simkit.jar are required by the model, and therefore we explicitly must write where these files are located. Additionally, we give some other folders, such as \inputs\ and \bin\ to tell the application where our other files are located (e.g. in the bin folder, there are class files of our NavalSim application).

Finally, we call java.exe (%JAVABIN% line) and give the application home as a parameter. This will eventually call the openmap application's main executive method.

3.1.3 Writing a Simulation Layer for OpenMap

After setting up the input file you can start writing a layer class for OpenMap. The class definition and the imports are shown in Code Box 3-1.

The Java imports are required Java libraries in this layer. The other two parts, OpenMap and Simkit, are the other libraries that this class use. Note that to import these two libraries, you need to map library class files in your programming environment. A simple way of doing this is to map openmap.jar and simkit.jar files. A jar file is like a winzip file which effectively contains multiple .class files, or compiled Java files, in a hierarchical way.

In Eclipse, to map the openmap.jar and simkit.jar files, you choose the project properties and then Java Build Path and Libraries tab.

Code Box 3-1 Class definition of SimulationLayer for OpenMap

```
package NavalSim;
/*
 * Java imports
 */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.geom.Point2D;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JPanel;
/*
 * OpenMap imports
 */
import com.bbn.openmap.Layer;
import com.bbn.openmap.event.LayerStatusEvent;
import com.bbn.openmap.event.MapMouseListener;
import com.bbn.openmap.event.ProjectionEvent;
import com.bbn.openmap.omGraphics.OMGraphicList;
import com.bbn.openmap.omGraphics.OMRaster;
import com.bbn.openmap.omGraphics.OMText;
import com.bbn.openmap.omGraphics.OMCircle;
import com.bbn.openmap.proj.Projection;
/*
 * Simkit imports
 */
import simkit.SimEvent;
import simkit.SimEventListener;

public class SimulationLayer extends Layer implements SimEventListener,
                                                       MapMouseListener,
                                                       ActionListener {

}
```

In the class definition part, you name the class (SimulationLayer) as you name in the input file. Note that Java is case sensitive, this means that SimulationLayer is different that simulationlayer. This layer "extends" Layer class, in order words we use predefined methods of a Layer, but we may re-write some of these methods if needed. This layer also implements a number of interfaces. For example, SimEventListener is used to implement the "processSimEvent" method, which will be discussed later.

The constructor of this class is shown in Code Box 3-2. The "scn" variable is the link between the model and the map layer. The rest of the constructor is the code required for showing graphics on the map. For example text1 and 2 are the letters A and B on the map, circle1 and 2 are the circles around them.

Code Box 3-2 SimulationLayer constructor.

```
public SimulationLayer(){
            scn=new TheModel();

            grafikList=new OMGraphicList();

            Point2D p1=scn.getLocationMover(0);
            Point2D p2=scn.getLocationMover(1);

            text1=new OMText((float)
                  p1.getX(),(float)p1.getY(),"A",1);
            text2=new OMText((float)
                  p2.getX(),(float)p2.getY(),"B",1);

            circle1=new OMCircle((float)
                  p1.getX(),(float)p1.getY(),1,1);
            circle1.setLinePaint(Color.BLUE);

            circle2=new OMCircle((float) p2.getX(),(float)p2.getY(),
                        scn.nmToDeg(1,30.0f));

            grafikList.add(text1);
            grafikList.add(text2);
            grafikList.add(circle1);
            grafikList.add(circle2);
}
```

On the map, locations of moving objects, the texts and circles, are provided from the model. The line scn.getLocationMover returns the location of the entity with the id number given as the parameter. Once p1 and p2 is known, the rest is to update the graphics accordingly; finally, the updated graphics are added to the graphic list. The list is later projected and redrawn on the map.

The constructor's role is to display the movers in their initial locations.

The animation, or the change of movers' locations occur in the processSimEvent method (Code Box 3-3). When a simulation event occurs this method is called. Type of the event is the variable e and the getEventName method reveals the name of this event. For example in the first if statement the ping event is handled. The ping event is an easy way to animate entities in simkit. This will be discussed later in this section. Likewise when a detection event occurs, the number of detections will be incremented by one.

When the user changes the zoom level, or add/delete a layer on the map, then the projectionChanged event is triggered. In this case the code in Code Box 3-4 is executed. There are two things in this method; first, the contents of the grafiklist, which contains the

mover icons and the letters A and B, are projected on the map. Secondly, the graphics are repainted. When the repaint method is called the paint method is executed, where the grafiklist contents are rendered on the map.

Finally the last part of our simulation layer is the control panel of the model. There could be a menu item or a panel for controlling the simulation. However for showing how a layer panel can be created, a getGUI method is written. In the Code Box 3-5, related user interface can be created, This method actually controls the simulation model. The pingThread2 class is the “ping” event creator in the simulation. Its parameters are used to control the ping events’ interval (the time between two ping events in the simulation) and the animation ping intervals (screen update frequency). The first is to add “ping” events in the event list, and the second is the real time between ping events.

The following five lines are to create sim. event listener links. Eventually, these lines say that this layer will listen to the simulation events created by pt, movers, and sensors. Where will these events be handled? In the processSimEvent function. There is only one JButton object in this basic interface. When this button is clicked we want to run the model. Therefore an actionlistener is created and in this code we start the model and as well as pinging.

Code Box 3-3 ProcessSimEvent function.

```
public void processSimEvent(SimEvent e) {
            fireStatusUpdate(LayerStatusEvent.START_WORKING);

            if (e.getEventName().equals("Ping")) {

            OMText tempText1=(OMText)
                  grafikList.getOMGraphicAt(0);
            OMText tempText2=(OMText)
                  grafikList.getOMGraphicAt(1);

            OMCircle tempCirc1=(OMCircle)
                  grafikList.getOMGraphicAt(2);
            OMCircle tempCirc2=(OMCircle)
                  grafikList.getOMGraphicAt(3);

            tempText1.setLat((float)
                  scn.getLocationMover(0).getX());
            tempText1.setLon((float)
                  scn.getLocationMover(0).getY());

            tempCirc1.setLatLon(
                  (float)scn.getLocationMover(0).getX(),
                  (float)scn.getLocationMover(0).getY());

            tempText2.setLat(
                  (float)scn.getLocationMover(1).getX());
            tempText2.setLon(
                  (float)scn.getLocationMover(1).getY());
            tempCirc2.setLatLon(
                  (float)scn.getLocationMover(1).getX(),
                  (float)scn.getLocationMover(1).getY());

            tempText1.generate(proj);
            tempText2.generate(proj);
            tempCirc1.generate(proj);
            tempCirc2.generate(proj);

            }

            if (e.getEventName().equals("Detection")) {
            System.out.println("Detection at:"+
                  getSimTime());
            detectionCounter++;
            }

            if(proj != null){
            ((OMGraphicList)grafikList).project(
                  (Projection)proj, true);
            }
            repaint();
            fireStatusUpdate(LayerStatusEvent.FINISH_WORKING);
}
```

Code Box 3-4 projectionChanged and paint methods.

```
public void projectionChanged(ProjectionEvent e) {
            proj = e.getProjection();
             System.out.println("projection Changed");
            ((OMGraphicList)grafikList).project(
                  e.getProjection(), true);
            repaint();
}


public void paint(java.awt.Graphics g) {
        if(grafikList.size() > 0){
          grafikList.render(g);
        }
        fireStatusUpdate(LayerStatusEvent.FINISH_WORKING);
}
```

Code Box 3-5 GetGUI method.

```
//A GUI for the layer
public java.awt.Component getGUI() {
      JPanel returnPanel = new JPanel();

      final PingThread2 pt = new PingThread2(0.1, 100, false);
      pt.addSimEventListener(this);
      scn.mover[0].addSimEventListener(this);
      scn.mover[1].addSimEventListener(this);
      scn.sensor[0].addSimEventListener(this);
      scn.sensor[1].addSimEventListener(this);

      runButton.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent e)
        {
             scn.startTheModel();
             pt.startPinging();
        }
      });

      returnPanel.add(runButton);

      return returnPanel;
}
```

There is also a one-line function to communicate with the model's simulation time. The getSimTime function gets the simulation time via scn variable.

3.2 Simkit

The simkit model is in the TheModel class and it includes a constructor (Code Box 3-6) and a method (Code Box 3-7). Class variables are arrays of movers, sensors, and a referee. These are to keep our entities in the simulation model. Note that all of these classes are Simkit classes which are included in the Simkit library.

Constructor of the model class starts with adding a sensor, a mover, and a mediator class to the SensorTargetMediatorFactory. This is to create a top level viewer to the model. This class will keep an eye on the interactions between movers and sensors, e.g. the time a mover enters a sensor's detection range, the location when a mover exits from a sensor's range.

We need to know initial locations of our movers and sensors. Points1 and 2 variables are OpenMap's 2D arrays and they simple keep a pair of double variables. Note that this may mean X and Y coordinate of a point on the screen, as well as latitude and longitude of a location in Cartesian coordinate system. In our case it is the latter. The default starting locations are in the central Mediterranean.

In our simulation model we have two movers. We initiate two movers in the mover array. Our movers are of type "UniformLinearMover". This type of mover assumes that a mover entity moves according to v=x/t equation, no acceleration, and no curves, turn etc. Please refer to Buss and Sanchez (2005) for more detail. The last parameter in the initialization is the speed of the mover. The important point here is that the speed is in distance travelled in unit time. The distances in our model are measured on a map, it is not pixels on the screen, and therefore the distances are converted to degrees on the map. In this class, a method to do the calculation is given:

public float nmToDeg(int latOrLon, float distance)

In our fictional scenario, the searcher moves randomly. To create this effect we initiate a random variate with two elements; to get a uniformly distributed random variable between 33 and 36 (this the latitude bounds of the area) and a uniformly distributed random variable between 15 and 20 (and longitude bounds). The rv variable is used in the random mover manager.

Code Box 3-6 The Model constructor.

```
      public Point2D.Double[] points1;
      public Point2D.Double[] points2;

      Mover[] mover;
      Sensor[] sensor;
      MoverManager[] manager;
      SensorTargetReferee ref;

public TheModel(){

      SensorTargetMediatorFactory.addMediator(
            CookieCutterSensor.class,
            UniformLinearMover.class,
            CookieCutterMediator.class
      );

      points1 =  new Point2D.Double[2];
      points2 =  new Point2D.Double[2];

      points1[0] = new Point2D.Double(34.5, 15.0);
      points1[1] = new Point2D.Double(34.5, 20.0);

      points2[0] = new Point2D.Double(36.0, 17.5);
      points2[1] = new Point2D.Double(33.0, 16.0);

      mover = new Mover[] {
            new UniformLinearMover("Ship 1", points1[0], 0.1),
            new UniformLinearMover("Ship 2", points2[0], 0.2)
      };

      sensor = new Sensor[] {
            new CookieCutterSensor(nmToDeg(1,0.0f), mover[0]),
            new CookieCutterSensor(nmToDeg(1,30.0f), mover[1])
      };

      RandomVariate[] rv = new RandomVariate[] {
      RandomVariateFactory.getInstance("Uniform", new Object[] { new
                  Double(33.0), new Double(36.0) } ),
      RandomVariateFactory.getInstance("Uniform", new Object[] { new
                  Double(15.0), new Double(20.0) } )
      };

      manager = new MoverManager[] {
            new PatrolMoverManager(mover[0], points1),
            new RandomLocationMoverManager(mover[1], rv)
      };

      for (int i = 0; i < manager.length; ++i) {
           manager[i].setStartOnReset(true);
      }

        ref = new SensorTargetReferee();
        for (int i = 0; i < mover.length; ++i) {
            ref.register(mover[i]);
        }

        for (int i = 0; i < sensor.length; ++i) {
            ref.register(sensor[i]);
        }
}
```

Mover Managers are to control the movements of the movers. There are a number of mover managers already coded in Simkit, such as Patrol Mover Manager and Random Location Mover Manager. Since in this example we have one mover doing patrolling and another one doing random moves, we initiate them accordingly.

Finally, we can tell the movers that when the model is reset they can start moving. We also need to say that movers and sensors are registered with the referee.

getLocationMover and getSimTime methods are auxiliary methods. The first one is required by the animation on the map. The map objects (OMText and OMCircle) know the location of movers in the simulation by calling this method. Likewise, the simulation time can be asked at anytime by calling getSimTime.

Code Box 3-7 Other methods in theModel class.

```
public void startTheModel() {

        //SimplePropertyDumper dumper = new SimplePropertyDumper();
        //ship1.addPropertyChangeListener(dumper);
        //ship2.addPropertyChangeListener(dumper);

      //Schedule.reset();
        Schedule.setVerbose(true);

        Schedule.setSingleStep(false);

        Schedule.stopAtTime(1000.0);
        for (int i = 0; i < manager.length; ++i) {
            manager[i].start();
        }
}

public Point2D getLocationMover(int i){
      return mover[i].getLocation();
}

public double getSimTime(){
      return Schedule.getSimTime();
}
```

startTheModel method is called externally. In this method we set some parameters in the model, for example verbose mode, single step mode, and when the model will stop. After these lines we can start the movements by calling mover managers' start method.

There are two missing parts in this simple model; first the data collection facility. In the Code Box 3-7 the commented lines are to give an indication of how this can be done. Simkit also provides data collection classes such as SimplePropertyDumper, with property listeners, e.g. dumper listens property changes of ship1. The second missing part is the multiple replications. As with other simulation models which use random numbers, this model must also run multiple times, however as it is now, it allows one model replication. To do multiple replications, there must be a loop which in each iteration the model resets and then runs with different random number stream.

REFERENCES

Buss A.H. (2001) Basic Event Graph Modeling. Technical Notes, Simulation News Europe, April 2001: 1-6.

Buss A.H. and P. Sanchez (2005) Simple Movement and Detection in Discrete Event Simulation. Proceedings of the 2005 Winter Simulation Conference, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds. pp.992-1000.

Buss A.H., D.K.Ahner (2006) Dynamic Allocation Of Fires And Sensors (Dafs):A Low-Resolution Simulation For Rapid Modeling. Proceedings of the 2006 Winter Simulation Conference. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

Mack, P. (2000) THORN: A Study in Designing a Usable Interface for a Geo-Referenced Discrete Event Simulation. MSc Thesis, Naval Postgraduate School. USA.

Yildirim U.Z., B.C. Tansel, I.Sabuncuoglu (2009) A multi-modal discrete-event simulation model for military deployment, Simulation Modelling. Practice and Theory, Volume 17, Issue 4, April 2009, Pages 597-611, ISSN 1569-190X, DOI: 10.1016/j.simpat.2008.09.016.