Introduction to RepastJ

Henrique Lopes Cardoso
(DEI/FEUP)
September 2009
MAS Software: JADE vs. Repast

- **JADE** is meant for building agent systems, but has no simulation infrastructure.
- **Repast** is meant for agent-based simulations and not for developing agent systems.
- **JADE** is not a simulation framework: it has no scheduler, nor a notion of a "clock"; you would have to code any simulation infrastructure.
- **Repast** is not for building MAS, and is not FIPA compliant.
- The agents within **Repast** are not the same kind of agents within **JADE**: both systems are designed for two entirely different purposes.
Repast

- Agent-based modeling
  - Computational methodology that aims at creating, analyzing and experimenting with artificial worlds populated by agents that interact in non-trivial ways

- Recursive Porus Agent Simulation Toolkit (REPA ST)
  - Open-source framework for creating agent-based simulations
  - Focus on modeling social behavior, but not limited to social simulation
  - Three flavors: RepastJ, RepastPy, Repast.Net

- Note
  - this presentation is based on Repast 3, not Repast Simphony
Repast Tools and Features

- Toolbar for controlling simulations
- GUI for manipulating parameters
Repast Tools and Features

- Data collection and output
- Displaying agent interaction
Repast Spaces

- **Purposes:**
  - Contain a collection of agents (spatial agents)
  - Define the spatial relationships of agents relative to each other

- **Cellular**
  - **Boundaries:**
    - Grid
    - Torus
      - No edges: if you keep going in the same direction, you come back to where you were
      - No spatial boundaries, always have the same number of neighboring cells
  - **Shapes:** rectangular, hexagonal
  - **Functionality:**
    - Inserting/removing items (agents) via coordinates
    - Retrieving list of items in neighboring cells
      - Von Neumann or Moore neighborhood (rectangular)
      - rings (hexagonal)
    - Determining the min/max item in neighborhood
      - application-specific
Repast Spaces

- Network
  - Nodes and edges (weighted graph)

- GIS
GUI/Batch Simulations

- **Batch mode**
  - Run simulation in background and get results

- **GUI mode**
  - Show variables in control panel
  - Greater user interaction
  - Run-time visualization through space displays or charts
  - Probes: view and manipulate single agents
Repast Simulations

- **Discrete event simulator**
  - Unit of time: *tick*
  - Events are scheduled for specific ticks: ticks are simply hooks on which events may be scheduled
  - The relative order of execution of events is respected
    - tick1, tick2, tick5, ...

- **Repast simulation**
  - Collection of agents
  - Model that sets up and controls the execution of agents’ behaviors according to a *schedule*

- **Schedule**
  - Execution of agent behaviors (actions)
  - Model actions, e.g. display updates, data recording, ...
Building a Repast Simulation

- Model class
  - Sets-up and controls the schedule, the collection of agents, display surfaces, data collectors, simulation parameters, ...
  - Implements the SimModel interface

- Agent class
  - To display in a space, must implement the Drawable interface
  - For probing, must define getters/setters and optionally implement the CustomProbeable interface

- Space
  - Spatial placing of agents
  - To display: space --> display --> display surface
Building a Simple Repast Model

uchicago.src.sim.engine.SimpleModel extends SimModelImpl

```
public void setup() {
    // prepare simulation
}

public void buildModel() {
    // create agents and add them to agentList
    // create space, displays, data recorders, charts
}

public void preStep() { }  // optional

public void step() {
    // action to execute at each tick, e.g.:
    // - iterate through all agents and have them
    //   execute behavior
    // - record data
    // - update displays
}

public void postStep() { }  // optional
```

Simpler but less powerful

**Alternative:** autoStep

- Agents in `agentList` must implement `Stepable`
- At each tick, the simulation will automatically go through every agent and invoke `Stepable.step()`

**Shuffle**

- With autoStep, if `shuffle` is set the simulation will shuffle agents at each tick:
  ```java
  SimUtilities.shuffle(agentList);
  ```
importuchicagosrc.sim.engine.SimInit;
importuchicagosrc.sim.engine.SimpleModel;

public class MyHelloWorldModel extends SimpleModel {
    private int numberOfAgents;

    public MyHelloWorldModel()
    {
        name = "My Hello World Model";
    }

    public void setup()
    {
        super.setup();
        numberOfAgents = 3;
        autoStep = true;
        shuffle = true;
    }

    public void buildModel()
    {
        for(int i=0; i<numberOfAgents; i++)
            agentList.add(new MyHelloWorldAgent(i));
    }

    protected void preStep()
    {
        System.out.println("Initiating step " +
                          getTickCount());
    }

    protected void postStep()
    {
        System.out.println("Done step " +
                          getTickCount());
    }
}

importuchicagosrc.sim.engine.Stepable;

public class MyHelloWorldAgent implements Stepable {
    private int id;

    public MyHelloWorldAgent(int id)
    {
        this.id = id;
    }

    public void step()
    {
        System.out.println(id + " Hello World!");
    }
}
Hello World

Run: 1
Initiating step 1.0
0 Hello World!
2 Hello World!
1 Hello World!
Done step 1.0
Initiating step 2.0
2 Hello World!
0 Hello World!
1 Hello World!
Done step 2.0
Initiating step 3.0
1 Hello World!
2 Hello World!
0 Hello World!
Done step 3.0
Initiating step 4.0
1 Hello World!
2 Hello World!
0 Hello World!
...
Building a Complex Repast Model

uchicago.src.sim.engine.SimModelImpl implements SimModel

```
public void begin() {
    buildModel();
    buildDisplay();
    buildSchedule();
}

private void buildModel() {
    // create and store agents
    // create space, data recorders
}

private void buildDisplay() {
    // create displays, charts
}

private void buildSchedule() {
    // build the schedule
}
```

More complicated but can take fuller advantage of Repast’s capabilities
private static final boolean BATCH_MODE = true;

public static void main(String[] args) {
    boolean runMode = !BATCH_MODE;   // BATCH_MODE or !BATCH_MODE

    // create a simulation
    SimInit init = new SimInit();

    // create a model
    MyModel model = new MyModel();

    // load model into simulation:
    init.loadModel(model, null, runMode);
}

void loadModel( SimModel model, String fileName, boolean isBatch);
Using Parameters

- Define setters/getters

```java
public void setNumberOfAgents(int numberOfAgents) {
    this.numberOfAgents = numberOfAgents;
}

public int getNumberOfAgents() {
    return numberOfAgents;
}
```

- Include in `getInitParam()`

```java
public String[] getInitParam() {
    return new String[] { "numberOfAgents" };
}
```

- Use:
  - GUI mode: parameters will be available in the settings window
  - Batch mode: parameter file defines values to be used
Scheduling

- setting up method calls on objects to occur at certain times

What
- sub-classes of the BasicAction class
  ```java
  class MyAction extends BasicAction {
    public void execute() {
      // do something
    }
  }
  ```
- arbitrary method calls
  - Repast itself creates the BasicAction

When
- Invoke specific methods from Schedule class to schedule actions to occur:
  - every tick beginning at some specific tick
  - once at a specific tick
  - at intervals
  - at a pause in the simulation
  - at the end of the simulation
- Typical examples:
  - Agent behavior and DisplaySurface.updateDisplay() at every tick
  - Snapshots at intervals
Schedule Methods

- $<BA> = \textbf{BasicAction} \text{ action | Object o, String methodName}$

```
public BasicAction scheduleActionBeginning(double beginning, <BA>);
public BasicAction scheduleActionAt(double at, <BA>);
public BasicAction scheduleActionAtInterval(double interval, <BA>);
public BasicAction scheduleActionAtPause(<BA>);
public BasicAction scheduleActionAtEnd(<BA>);
```

- Variations:
  - schedule order within the same tick (random or last); see also \textbf{ActionGroup} for sequential execution
  - duration
  - object lists, shuffling
Using Spaces

- Create space:
  
  ```java
  Object2DGrid space = new Object2DGrid(spaceWidth, spaceHeight);
  ```

- Placing agents in space:
  ```java
  for (int i = 0; i < numAgents; i++) {
      int x, y;
      do {
          x = Random.uniform.nextIntFromTo(0, space.getSizeX() - 1);
          y = Random.uniform.nextIntFromTo(0, space.getSizeY() - 1);
      } while (space.getObjectAt(x, y) != null);
      MyAgent agent = new MyAgent(x, y, space);
      space.putObjectAt(x, y, agent);
      agentList.add(agent);
  }
  ```

- Moving agent: change x and y coordinates, remove from previous position in space and add to new position
  ```java
  space.putObjectAt(x, y, null); // not multi-space (only 1 agent per cell)
  x = newX;
  y = newY;
  space.putObjectAt(x, y, this); // assuming this is done from inside the agent
  ```
public void setup() {
    
    dsurf = new DisplaySurface(this, "Simulation Title");
    registerDisplaySurface("Simulation Title", dsurf);

    space.putObjectAt(...)
    agentList.add(...)

    ...}

private void buildModel() {
    
    agentList = new ArrayList();
    space = new Object2DGrid(width, height);

    space.putObjectAt(...)
    agentList.add(...)

    ...}

private void buildDisplay() {
    
    Object2DDisplay agentDisplay = new Object2DDisplay(space);
    agentDisplay.setObjectList(agentList);

    dsurf.addDisplayableProbeable(agentDisplay, "Agents");
    addSimEventListener(dsurf);
    dsurf.display();
}
Data Recording

- Sources of data:

  ```java
  public interface NumericDataSource {
    public double execute();
  }
  public interface DataSource {
    public Object execute();
  }
  ```

- **DataRecorder**
  - creating:
    ```java
    public DataRecorder(String fileName, SimModel model);
    ```
  - adding data sources:
    ```java
    public void addNumericDataSource(String name, NumericDataSource s);
    public void createNumericDataSource(String name, Object feedFrom, String methodName);
    ```
  - using: schedule method invocations
    - record, writeToFile
Creating Charts

- **Sources of data:**
  - **OpenSequenceGraph**: plots some variables versus time ticks
    - creating:
      ```java
      public interface Sequence {
          public double getSValue();
      }
      ```
    - adding sequences:
      ```java
      public OpenSequenceGraph(String title, SimModel model);
      public Sequence addSequence(String name, Sequence sequence);
      public Sequence createSequence(String name, Object feedFrom,
                                       String methodName);
      ```
    - variations: color, markstyle
    - using: schedule method invocations
      - record, updateGraph, step, takeSnapshot, writeToFile

- **NetSequenceGraph, OpenHistogram, Histogram, Plot**
Demo: ColorPickingModel

- A set of agents in a torus space
- Each agent is assigned a color and a position
- At each tick, each agent:
  - makes a random move: walk or jump
  - picks a color of a random neighbor
- Eventually a few number of colors (1?) will prevail...

Files:
- ColorPickingModel.java
- ColorPickingAgent.java
More Information

- Repast Howto Documentation

- How to build a Repast model
  - [http://www.perfectknowledgedb.com/Tutorials/H2R/HowTo00.htm](http://www.perfectknowledgedb.com/Tutorials/H2R/HowTo00.htm)

- Repast: A Software Toolkit for Agent-Based Social Science Modeling
  - [http://www.econ.iastate.edu/tesfatsi/repastsg.htm](http://www.econ.iastate.edu/tesfatsi/repastsg.htm)