Agent-Based Modeling in Repast

Giuseppe Narzisi

Courant Institute of Mathematical Sciences

Feb 21 2008
Lecture outline

• Agent-Based Modeling
• RePast toolkits and its features
• How to build a model
  – SimpleModel methods
  – Model parameters
  – Schedule
  – Spaces and Topology
• Example
  – Game of Life
• GIS integration
• Example:
  – PLAN C
Agent-Based Modeling

• ABM is a computational methodology that allows the analyst to create, analyze, and experiment with artificial worlds populated by agents that interact in non-trivial ways
  • Bottom-up
  • Computational
  • Builds on CAs
Complex Adaptive systems

A **CAS** is a network exhibiting aggregate properties that emerge from primarily local interaction among many, typically heterogeneous agents mutually constituting their own environment.

- Emergent properties
- Large numbers of diverse agents
- Local and/or selective interaction
- Adaptation through selection
- Endogenous, non-parametric environment
Why use agent-based modeling?

ABM helps fill the gap between formal but restrictive models and wide-ranging but imprecise qualitative frameworks

- Contingency and counterfactuals
- Mechanisms in space and time
- Intangible concepts
What is RePast?

- **Recursive Porous Agent Simulation Toolkit**
- Repast is an open-source software framework for creating agent-based simulations using Java
- Initially developed by the Social Science Research Computing at the University of Chicago
- Further developed by the RePast Organization for Architecture and Development (ROAD) and Argonne National Laboratory

- We will use RePast 3
- Repast Simphony (recently released)
  - Offers point-and-click development environment (we do not need it !)
Why RePast?

• Alternatives: Swarm, Ascape, NetLogo...
• "RePast is at the moment the most suitable simulation framework for the applied modeling of social interventions based on theories and data" (2004): http://jasss.soc.surrey.ac.uk/7/1/6.html
• Modeled on Swarm but easier to use and better documented
• Important criteria:
  – abstraction, ease of use and user-friendliness
  – flexibility and extensibility
  – performance and scalability
  – support for modeling, simulation & experimentation
  – Interoperability (GIS, statistical packages, …)
What does RePast offer?

• Skeletons of **agents** and their environment
• Graphical **user interface**
• **Scheduling** of simulations
• **Parameters** management
• Behavior **display**
• **Charting**
• Data collection
• Batch and **parallel runs**
• Utilities and sample models
How to build a model

• Extending the `SimpleModel` class with our own model class:

```java
import uchicago.src.sim.engine.SimpleModel;

public class MyModel extends SimpleModel {
    ...
}
```
import uchicago.src.sim.engine.SimpleModel;

public class MyModel extends SimpleModel {
    public static final int TFT = 1;
    public static final int ALLD = 3;

    private int a1Strategy = TFT;
    private int a2Strategy = ALLD;

    ...  

    public void setup() {
        super.setup();
        a1Strategy = TFT;
        a2Strategy = ALLD;
    }

    public void buildModel() {
        super.buildModel();
        Agent a1 = new Agent(a1Strategy);
        Agent a2 = new Agent(a2Strategy);
        agentList.add(a1);
        agentList.add(a2);
    }
}
Running the simulation

```java
import uchicago.src.sim.engine.SimpleModel;

public class MyModel extends SimpleModel {
    ...

    public void setup() {...

    public void buildModel() {...

    public void preStep() {...

    public void step() {
        super.step();
        for (Iterator it = agentList.iterator(); it.hasNext();) {
            Agent agent = (Agent) iterator.next();
            agent.play();
        }
    }

    public void postStep() {...
}
```
import uchicago.src.sim.engine.SimpleModel;

public class MyModel extends SimpleModel {
    public static final int TFT = 1;
    public static final int ALLD = 3;

    private int a1Strategy = TFT;
    private int a2Strategy = ALLD;

    public MyModel() {
        name = "Example Model";
        params = new String[] {"A1Strategy"};
    }

    public void setA1Strategy(int strategy) {
        this.a1Strategy = strategy;
    }

    public int getA1Strategy() {
        return a1Strategy;
    }
}
SimpleModel methods

- `setStoppingTime(long time)`
  set the time at which the current simulation run will stop
- `setRngSeed(long seed)`
  set the seed for the random number generator (default: 1)
- `getNextIntFromTo(int from, int to)`
  returns the next random int between from and to (inclusive)
- `getNextDoubleFromTo(double from, double to)`
  returns the next random double between from and to (exclusive)
- `atPause()`
  will be executed whenever your simulation is paused
- `atEnd()`
  will be executed whenever your simulation ends

For the full list, consult the RePast API:
How to use a schedule

- Schedule object is responsible for all the state changes within a Repast simulation

```java
schedule.scheduleActionBeginning(1, new DoIt());
schedule.scheduleActionBeginning(1, new DoSomething());
schedule.scheduleActionAtInterval(3, new ReDo());

tick 1: DoIt, DoSomething
tick 2: DoSomething, DoIt
tick 3: ReDo, DoSomething, DoIt
tick 4: DoSomething, DoIt
tick 5: DoIt, DoSomething
tick 6: DoSomething, ReDo, DoIt
```
Scheduling actions on lists

- An action can be scheduled to be executed on every element of a list:

```java
public class Agent {
    public void step() {
    }
}
schedule.scheduleActionBeginning(1, agentList, "step");
```

- is equivalent to:

```java
public void step() {
    for (Iterator it = agentList.iterator(); it.hasNext();)
    {
        Agent agent = (Agent) it.next();
        agent.step();
    }
}
schedule.scheduleActionBeginning(1, model, "step");
```
Different types of scheduling

• `scheduleActionAt(double at, ...)`: executes at the specified clock tick
• `scheduleActionBeginning(double begin, ...)`: executes starting at the specified clock tick and every tick thereafter
• `scheduleActionAtInterval(double in, ...)`: executes at the specified interval
• `scheduleActionAtEnd(...)`: executes the end of the simulation run
• `scheduleActionAtPause(...)`: executes when a pause in the simulation occurs
GUI features

Graphical user interfaces offer:
• Customized parameter panel
• Dynamic graphs
• Graphical displays
• Probes
Separating GUI and batch modes

- Model extends SimpleModel
- ModelGUI extends Model
- ModelBatch extends Model
Subclassing a GUI model

```java
class Model extends SimpleModel {
    model variables
    setup()
    buildModel()
    step()
    main()
}

class ModelGUI extends Model{
    GUI variables (graph)
    setup() {
        super.setup();
        params = ...
        delete old graphs
    }
    buildModel() {
        super.buildModel();
        create graph
    }
    step() {
        super.step();
        update graph
    }
    ...
    main()
}
```
Types of plots and charts

- **Time series**
  user defined variable(s) over time

- **Histogram**
  bar chart showing a variable’s distribution

- **Scatter plot**
  snapshot of two variables
Showing time series

- **Main class:** OpenSequenceGraph
  `uchicago.src.sim.analysis.OpenSequenceGraph`

- **Extension:** NetSequenceGraph
  Specialized for network statistics (path length, cluster coefficient, density, ...). Note: `agentList` should implement the `Node` interface!

- **Declaration:**

  ```java
  public class ModelGUI extends Model {
      private OpenSequenceGraph graph;
      ...
  }
  ```
Showing time series (cont.)

- Initialization:

```java
public void setup() {
    super.setup();
    if (graph != null)
        graph.dispose();
}
```

- Instantiation:

```java
public void buildModel() {
    super.buildModel();

    graph = new OpenSequenceGraph("Graph", this);
    graph.setXRange(...);
    graph.addAxisTitles(...);
    graph.display();
}
```
Showing time series (cont.)

• Updating:

```java
public void step() {
    super.step();

    graph.step();
}
```

• Adding variables:

```java
graph.createSequence("X", this, "getXValue");
```
Three crucial questions:

1. **Variation**: What are the actors’ characteristics?
2. **Interaction**: Who interacts with whom, when and where?
3. **Selection**: Which agents or strategies are retained, and which are destroyed?

(see Axelrod and Cohen. 1999. *Harnessing Complexity*)
In each time period, a player interacts with four other random players.
The players are arranged on a fixed torus and interact with four neighbors in the von-Neumann neighborhood.
Fixed Random Network

The players have four random neighbors in a fixed random network. The relations do not have to be symmetric.
Adaptation through imitation

Neighbors at $t$

Imitation
Types of spaces

- **Boundaries**
  - Grid
  - Torus

- **Cell’s shape**
  - Rectangular
  - Hexagonal

- **Cell’s content**
  - One object
  - Collection of agents

- **GIS environment**
Classes

- **Object2DGrid**
  A discrete two-dimensional grid whose cells may contain an object.

- **Object2DTorus**
  A discrete two-dimensional torus whose cells may contain an object.

- **Multi2DGrid**
  A two-dimensional grid whose cells can contain more than one Object. The order of the Objects in each cell is undefined.

- **OrderedMulti2DGrid**
  A two-dimensional grid whose cell can contain more than one Object. The order of the Objects in each cell is first in, first out.

- **Diffuse2DGrid**
  A discrete approximation of two-dimensional diffusion. The space itself is a toroidal (donut-shaped) grid whose cells contain doubles.
Usage

• Random arrangement:

```java
Object2DGrid space = new Object2DGrid(spaceWidth, spaceHeight);
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);
}
```

Random arrangement

One occupant per cell
Neighborhood

- Moore

getMooreNeighbors(int x, int y, int xExtent, int yExtent, boolean nulls)

- Von Neumann

getVonNeumannNeighbors(int x, int y, int xExtent, int yExtent, boolean nulls)
Game of life
origins

• John Conway was interested in a problem presented in the 1940s by John von Neumann: *self-replicating machines*
• John von Neumann found a very complicated rules on a rectangular grid

• Conway devised the *Game of Life*. 
**Game of Life**

- First practical CA invented by John Conway in the late 1960s
- Later popularized by Martin Gardner

**Simple rules:**

- A dead cell with 3 live neighbors comes to life
- A live cell with 2 or 3 neighbors stays alive
- Otherwise the cell dies (loneliness or overcrowding)
Until recently, coupled GIS models of humans-environment interactions were rare. Many GIS based biophysical models have been developed (soil erosion, hydrology, etc.). Urban CA models are also common. The need to include social science data in agent models, as well as create models of spatially intelligent agents, is essential. A major barrier to building integrated models lies in the static structure of GIS databases.
Repast and GIS

- Two classes are needed
  - One is reading and writing data.
  - The other is working with the GIS to coordinate the display of the GIS with updates to ABMS data.
  - In the Repast-GIS integration, these tasks are generally broken up into two different classes, a data class, and a display class.

- There are two data classes:
  - GeotoolsData and OpenMapData

- There are two main GIS systems
  - ESRI ArcMap and OpenMap

- There are two display classes
  - EsriDisplay and OpenMapDisplay

- You need to use the correct data class for the display class
  - GeotoolsData ↔ EsriDisplay
  - OpenMapData ↔ OpenMapDisplay
Multiple layers

- Properties
  - Topography, land cover, zoning

- Networks
  - Transportation, social, and communication

- Diffusion models
  - Information transfer, positive and negative spatial spillovers, transport of pollutants, species migration
PLAN C
Planning with Large Agent-Networks against Catastrophes
Agents and Environment

• Multiple computational agents with unique parameter sets:
  – Hospitals.
  – Individuals (persons).
  – On-Site Responders.
  – Ambulances.
  – Catastrophe event type.

• Multiple communication channels for information exchange between similar and differing agents.

• Personality trait modeling (worry, distress, trust, & compliance).

• Dose-response health progression model.

• “Geographic realism”: integration of topographic and transportation constraints.
Person agent

- Selfish and boundedly rational, with stochastic personality traits emulating panic behavior:
  - **State:**
    - Headed to original destination or to a hospital.
  - **Parameters:**
    - Current health level ($H_i$).
    - Disability factor ($D_i$).
    - Current “amount” of medication/treatment.
    - Access to a communication device.
    - Probability of communication.
  - **Knowledge:**
    - Location of some hospitals and (±) their capacity.
    - Time of last update of this information.
    - List of $N$ most recently visited nodes.
  - **Personality:**
    - Degree of worry ($W_i$).
    - Level of compliance ($C_i$).
    - Level of distress ($S$).
LRTA* with ignore-list for route finding

• The Learning Real-Time algorithm (LRTA). (Korf, 1990) interweaves planning and execution in an on-line decision-making setting.

• Default:
  – If all neighbors of the current node \((i)\) are in the ignore list, pick one randomly.

• Else:
  – *Look-Ahead:* Calculate \(f(j) = k(i,j) + h(j)\) for each neighbor \(j\) of the current node \(i\) that is not in the ignore-list. Here, \(h(j)\) is the agent's current estimate of the minimal time-cost required to reach the goal node from \(j\), and \(k(i,j)\) is the link time-cost from \(i\) to \(j\),
  – *Update:* Update the estimate of node \(i\) as follows:

\[
h(i) = \max \left\{ h(i), \min_{j \in \text{Next}(i)} f(j) \right\}
\]

*Action Selection:* Move towards the neighbor \(j\) that has the minimum \(f(j)\) value.
Hospital agent

• Attributes
  – **State**: available, critical or full;
  – **Facts**: resource level (representing both recoverable resources like doctors, nurses and beds, and irrecoverable resources like drugs and saline), reliability of communication device (information up-date rate);
  – **Knowledge**: locations and current capacities of known hospitals;
  – **Triage Behavior**: health-levels below which a person is considered critical, non-critical or dischargeable.
Hospital agent

- Probability of admission:
  \[ P_r(\text{admit}) = \frac{Eff_H \times rate_p \times tick_{size}}{1 + |H|_{occ}} \]

- Hospital efficiency:
  \[ Eff_H = \frac{1}{1 + \left( \frac{|H|_{occ} + S_H}{|H| + |ED|} \right)} \]

- Average time in ED:
  \[ Avg_{t}^{ED} = base_t \times (1 + |Ed|_{occ} + \frac{S_{ED}}{|ED|}) \]
Real-time Computer GIS graphics and overview

Real Time Date Collection and Analysis

User Friendly Flexible, real-time Interface
Demo: Sarin attack in NYC
Food poisoning scenario in Washington Square Area, NYC