



Agent-based Modelling and Simulation *”Repast Symphony”*

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy

bucchiarone@fbk.eu

30 September 2020

- CAS are composed by interacting, *autonomous agents*.
- Agents have *properties* and *behaviors*.
- Agents *interacts* with and *influence* each other.
- Agents *learn* from their experiences.
- Agents *adapt their behaviors* to they are better suited to their environment(s)

North, MJ, NT Collier, J Ozik, E Tatara, M Altaweel, CM Macal, M Bragen, and P Sydelko, "*Complex Adaptive Systems Modeling with Repast Symphony*", Complex Adaptive Systems Modeling, Springer, Heidelberg, FRG (2013). <https://doi.org/10.1186/2194-3206-1-3>.

Antonio Bucchiarone. Collective Adaptation through Multi-Agents Ensembles: The Case of Smart Urban Mobility. ACM Trans. Auton. Adapt. Syst. **14(2)**: 6:1-6:28 (2019)

- Agent-based modeling has been used successfully to model complex adaptive systems.
- ABM is used in many disciplines
 - Biology, Supply chains, economics, military planning, consumer market analysis, etc..
- ABM tools
 - StarLogo, NetLogo, Swarm, MASON, EcoLab, Ascape, GAMA
 - Recursive Porous Agent Simulation Toolkit (REPAST)
 - **Repast Symphony**

- Free and open source toolkit to handle large-scale agent simulation application development.
- Growing and large community.
- Multiple users have applied Repast to a wide variety of applications.
 - Socials systems, evolutionary systems, market modeling, industrial analytics.

<https://sourceforge.net/projects/repast/>

https://repast.github.io/repast_symphony.html

<https://repast.github.io/docs.html>



North, MJ, NT Collier, J Ozik, E Tatara, M Altaweel, CM Macal, M Bragen, and P Sydelko, "Complex Adaptive Systems Modeling with Repast Symphony", Complex Adaptive Systems Modeling, Springer, Heidelberg, FRG (2013). <https://doi.org/10.1186/2194-3206-1-3>.

- Repast Symphony uses *Eclipse* as its primary development environment.
- Repast Symphony leverage Eclipse's plug-in architecture to provide a set of development options.
- Repast Symphony Eclipse plug-ins provide *tools*, *views*, and *perspectives* for creating a range of Repast-specific model components:
 - general Repast projects
 - ReLogo projects
 - Flowcharts, and
 - ReLogo agents
- The plug-ins allow Repast models to be *executed*, *debugged*, and *packaged* into self-contained installers for deployment.

- It uses a highly modular architecture.
- Each module is an independent “plug-in” that can be connected or disconnected with a few lines of XML.

Plugin or plugin sets	Function
Simphony Application Framework	Provides the basic runtime interface plug-in system and user interface tools
Eclipse Set	Provides model specification and programming tools
Core Set	Provides central simulation functions, such as scheduling
ReLogo Set	Provides a simple modeling language and structure
GIS	Provides GIS modeling and visualization
Freeze Dryer Set	Provides persistence in XML and text formats
Batch Run Set	Provides parameter sweeps and stochastic iteration
Deployment	Packages user models for self-contained release
Charts	Provides interactive model graphing in the runtime interface
Model Integration	Provides tools for embedding legacy models
2D Visualization Set	Provides tools for interactive two-dimensional (2D) model viewing
3D Visualization Set	Provides tools for interactive three-dimensional (3D) model viewing
Terracotta	Provides distributed model execution
System Dynamics	Provides tools for ordinary differential and difference equations
Third-Party Application Set	Supports a set of independent plug-ins for: <ul style="list-style-type: none">● Geographic Resources Analysis Support System (GRASS),● Java Universal Network/Graph Framework (JUNG) network analysis,● *ORA network analysis,● Pajek network visualization,● R statistics,● iReport and Jasper Reports enterprise reporting,● Spreadsheets,● Structured Query Language (SQL) analysis within running simulations,● VisAD scientific visualization, and● Weka data mining.

- How to develop models of CAS with Repast Symphony.
- **Scenario:**
 - *Innovation Diffusion Environment.*
 1. *Group of people* randomly circulates.
 2. People occasionally *communicate ideas* when they encounter one another.
 3. When an idea is transferred, a new link is made between the transferring parties, and then any existing network lines are forgotten.
 4. People occasionally develop new ideas.
 5. When they do so, they forget all their old links.
 6. The quality of ideas is rated and tracked over time.

What is the quality of ideas over time?

- Must have real individual behaviors;
- Adapt, change, learn;
- Form dynamically changing relationships;
- Form organizations;
- Have spatial interactions;
- Have arbitrarily large populations; or
- When structural change is an output, not an input.

- What are the modeling question(s)?
- Who are the stakeholders?
- What output are needed?
- What input data are available?
- Who are the agents?
- What are the agent behaviors?
- What is the agent environment?

- What are the modeling question(s)?

How does the quality of ideas change over time in the innovation network?

- Who are the stakeholders?

The stakeholders in this demonstration case are simply the modelers themselves.

- What Outputs Are Needed?

The output will be the history of the quality of innovation. This output will be displayed as a time series chart.

- What Input Data Are Available?

For this model, the input will be the probability of developing a new innovation on a given time step.

- Who are the agents?

The agents are individual people in the innovation network.

- What are the Agent Behaviors?
 1. **The agents will randomly circulate on a continuous two-dimensional surface.**
 2. **When an agent with an innovation comes close enough to another agent, they will transfer their ideas.**
 3. **When an idea is transferred, a new link is made between the transferring parties.**
 4. **Any existing network links in the recipient will be deleted.**
 5. **Agents develop at the rate given by the input innovation probability.**
 6. **When they do so, they delete all their links.**
 7. **Each idea is represented using a number.**
 8. **The value of the number represents the relative quality of the idea.**
 9. **The value also automatically maps into a display color for the agents and its links.**

- What is the Agent Environment?

The agent environment is a simple, continuous 2D surface with support for a network between agents.

Package Explorer ✕

▼ Innovation

▶ src Model code

output

▶ shapes Set of optional icons that may be assigned to agents.

▼ src

▼ innovation.relogo

▶ UserGlobalsAndPanelFactory.groovy

▶ UserLink.groovy

▶ UserObserver.groovy

▶ UserPatch.groovy

▶ UserTurtle.groovy

Automatically generated
model components

▶  UserGlobalsAndPanelFactory.groovy

Controls that will appear in the model runtime window

```
package innovation.relogo


import repast.simphony.relogo.factories.AbstractReLogoGlobalsAndPanelFactory

public class UserGlobalsAndPanelFactory extends AbstractReLogoGlobalsAndPanelFactory{
    public void addGlobalsAndPanelComponents(){

        // add the needed control buttons
        addStateChangeButton("setup");
        addStateChangeButton("go");

        //add the model input
        addInput("innovationProbability",0.001)

    }
}
```

►  UserObserver.groovy

How to initialize the model and what happens at each simulation tick.

```
package innovation.relogo


import static repast.simphony.relogo.Utility.*;

class UserObserver extends ReLogoObserver{

    // define the model builder
    def setup() {
        // remove any existing turtles
        clearAll()
        //build a new set of turtles
        createTurtles(20){
            setShape("person")
            setColor(gray())
            forward(1)
        }
        //distribute an initial innovation
        ask(turtles().first()){
            innovation = yellow()
            setColor(innovation)
        }
    }

    //define the model time step routine
    def go() {
        //give each turtle a chance to act
        ask(turtles()){
            act()
        }
    }
}
```

1. clear all the model of any existing turtles.
2. create a set of 20 turtles, set the shape of each turtle, set the initial color and move the turtle agent one step forward.
3. retrieve the first turtle and set its *innovation* parameter and color to yellow.

▶  UserObserver.groovy

How to initialize the model and what happens at each simulation tick.

```
package innovation.relogo

import static repast.simphony.relogo.Utility.*;

class UserObserver extends ReLogoObserver{

    // define the model builder
    def setup() {
        // remove any existing turtles
        clearAll()
        //build a new set of turtles
        createTurtles(20){
            setShape("person")
            setColor(gray())
            forward(1)
        }
        //distribute an initial innovation
        ask(turtles().first()){
            innovation = yellow()
            setColor(innovation)
        }
    }

    //define the model time step routine
    def go() {
        //give each turtle a chance to act
        ask(turtles()){
            act()
        }
    }
}
```

1. ask each turtle agent to execute its “act” method once each time the simulation is stepped.

►  UserTurtle.groovy

It contains all the Turtle agent behaviors

```
package innovation.relogo

import static repast.simphony.relogo.Utility.*;

class UserTurtle extends ReLogoTurtle{

    // Define the innovation tracker.
    int innovation = 0
    // Define the activity.
    def act() {
        // Move forward randomly.
        left(random(15))
        right(random(15))
        forward(1.1)
        // Spontaneously create innovations.

        if (randomFloat(1.0) > (1.0-innovationProbability)) {
            // Develop the new innovation.
            innovation = 10 * random(11) + 25
            setColor(innovation)
            // Clear out my old links.
            ask (myLinks()) { die() }
            }// Check for an innovation.

        if (innovation > 0) {
            // Diffuse an innovation from "myself" to
            // nearby turtles ("self ").
            ask (other(turtlesHere())) {
            // Get the innovation.
                innovation = myself().innovation
                setColor(myself().getColor())
                // Clear out the old links.
                ask (myLinks()) { die() }
                // Build a new link.
                UserLink newLink =
                createLinkWith(myself())
                newLink.setColor(myself().getColor())
            }
        }
    }
}
```

1. Act moves the turtle forward by 1.1 units in a direction that is randomly determined.
2. *spontaneous innovations* are randomly created within the turtle by generating a random integer between zero and one and checking this value against the input threshold value (1.0 -innovationProbability).
3. If a *new innovation* is to be created, then the “innovation” value is set based on the random value, the turtle’s icon color is set to the new innovation value, and the turtle’s existing links are destroyed.

►  UserTurtle.groovy

It contains all the Turtle agent behaviors

```
package innovation.relogo

import static repast.simphony.relogo.Utility.*;

class UserTurtle extends ReLogoTurtle{

    // Define the innovation tracker.
    int innovation = 0
    // Define the activity.
    def act() {
        // Move forward randomly.
        left(random(15))
        right(random(15))
        forward(1.1)
        // Spontaneously create innovations.

        if (randomFloat(1.0) > (1.0-innovationProbability)) {
            // Develop the new innovation.
            innovation = 10 * random(11) + 25
            setColor(innovation)
            // Clear out my old links.
            ask (myLinks()) { die() }
            }// Check for an innovation.

        if (innovation > 0) {
            // Diffuse an innovation from "myself" to
            // nearby turtles ("self ").
            ask (other(turtlesHere())) {
                // Get the innovation.
                innovation = myself().innovation
                setColor(myself().getColor())
                // Clear out the old links.
                ask (myLinks()) { die() }
                // Build a new link.
                UserLink newLink =
                createLinkWith(myself())
                newLink.setColor(myself().getColor())
            }
        }
    }
}
```

4. In the event that a new innovation is created, the turtle asks for other turtle agents nearby (located in the same patch) to which the new innovation will be transferred.

5. The innovating turtle sets its neighboring turtles' "innovation" and "color" to the new "innovation" value, destroys the neighboring turtles' existing links, and generates a new link between the turtle and its neighbor.

- Grading system:
 - 50% Oral exam
 - 50% Lab assignments
- Implementation of distributed algorithms, to be completed during the course
 - 2 assignments (middle November, early January)
 - Based on <https://repast.github.io/> (Java, agent-based modeling)
 - Group-based (1-3 students)

<https://bucchiarone.bitbucket.io/>

<https://docs.google.com/spreadsheets/d/1mdYwWRUZQWHROQ0huvplyPjjUAaZDnTMi1ve6uCAYAA/edit?usp=sharing>