A Software Interface Between System Dynamics and Agent-Based Simulations – Linking Vensim® and RePast®

Andreas Größler, Myrjam Stotz and Nadine Schieritz

Mannheim University
Industrieseminar, Schloss
D-68131 Mannheim, Germany

Tel.: +49 621 181-1583
Fax: +49 621 181-1579
E-Mail: agroe@is.bwl.uni-mannheim.de

Abstract

A software-based integration of agent-based modeling and system dynamics is presented. More precisely, it is described how RePast and Vensim can be coupled using standard procedure calls. In an example from supply chain management, agents modeled with RePast in an agent-based simulation context are provided with system dynamics models as internal schemata, which are built with Vensim. The prototypical application shows both, the technical simplicity of combining agent-based and system dynamics simulations when using RePast and Vensim, and the potential of a combination of the two approaches.

Key words: system dynamics, agent-based simulation, Vensim, RePast, supply chain management

To date mainly two approaches use computer simulation to investigate social and economic systems that are characterized by non-linearity, delays and feedback processes: system dynamics and agent-based simulation. Both concentrate on understanding and qualitative prediction of systems behavior. Although the two approaches have a large intersection regarding research topics they have been relatively unnoticed by each other (Phelan 1999). Therefore, an integration of both approaches concerning conceptual and technical issues seems to be a worthwhile endeavor.

Some pioneering work on the combination of system dynamics and agent-based simulation was conducted not earlier than two years ago by Akkermans (2001) and by Scholl (2001a, 2001b). More recently Pourdehnad, Maani and Sedehi (2002), Schieritz (2002) and Schieritz & Milling (2003) have concentrated on principal methodological issues when combining the two simulation approaches. In a paper by Schieritz & Größler (2003) a prototypical implementation of a combined system dynamics and agent-based simulation model is presented. In that paper an application from the area of supply chain management is discussed. Technically, agents in a supply chain are

modeled in eM-Plant whereas their decision structures (Anderson 1999) are represented by Vensim (i.e. continuous system dynamics) models.[1, 2]

In this paper we focus on a technically improved integration of both simulation approaches. Therefore, the rather "clumsy" implementation of agents in eM-Plant was dismissed and a specialized Java class library for programming agent-based simulations is used: RePast. However, the principal conceptual approach of integrating both simulation methods is kept: the overall simulation environment is modeled using agents technology; the internal decision structures of the agents are modeled as system dynamics models (using Vensim). We use a simple example from supply chain management to demonstrate the usefulness of this software integration.

An application integrating both simulation methods consists of a "master" program representing the overall simulation environment and the internal structures of the agents. The master program consists of Java classes that use the specialized RePast class libraries. Every simulation step the master program calls the behavior methods of the agents.

In an example from supply chain management two kinds of agents can be identified: a supplier and three manufacturers. The supplier's behavior is modeled in RePast whereas the manufacturers' behavior modes are represented by a system dynamics model built in Vensim. For each simulation step the manufacturers generate their order rates as the output of the simulation run of the system dynamics model. This data is passed to the supplier who delivers goods to the manufacturers as an input for the next simulation step. The communication between the manufacturer-agents and the Vensim model runs via Vensim DDL (see Figure 1).

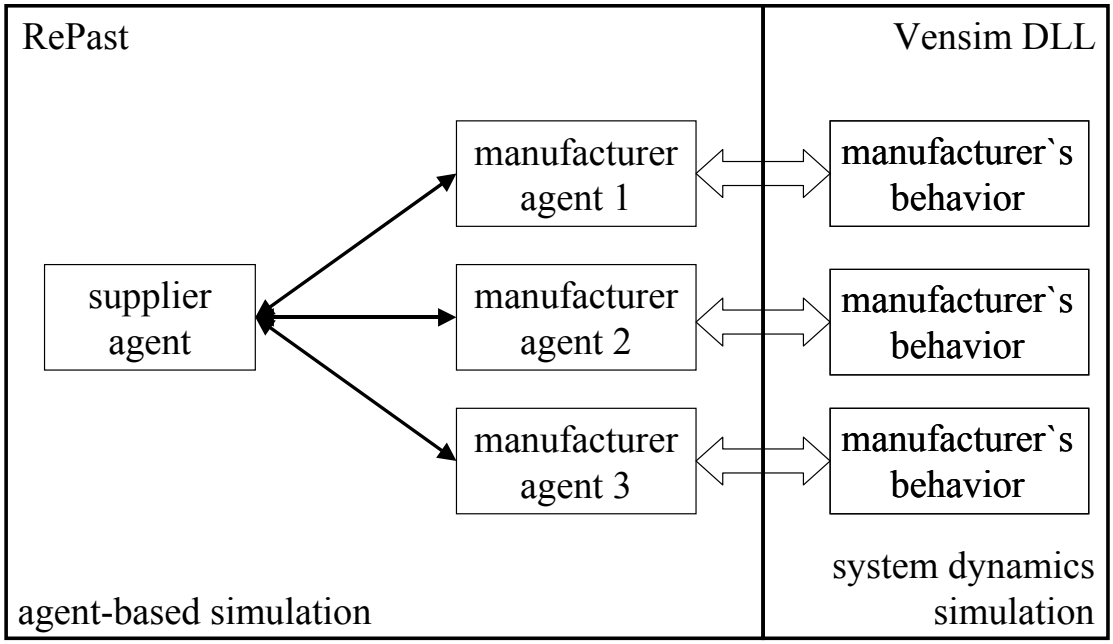

Figure 1: Example application with interaction between RePast and Vensim

In our example, the RePast master program consists of the following three parts:

1. The model class

This class manages the simulation context. Agents (manufacturers and supplier) are built, the simulation is started, simulation steps executed and their results displayed. Important methods in this class are buildModel(), buildSchedule(), getInitParam(), buildDisplay() and the main() method:

- private void buildModel() {

```
Manufacturer m1 = new Manufacturer(20,100,20,0,customerOrderRateM1);
Manufacturer m2 = new Manufacturer (30,200,50,0,customerOrderRateM2);
Manufacturer m3 = new Manufacturer (50,500,70,0,customerOrderRateM3);
manufacturer List.add(m1);
manufacturer List.add(m2);
manufacturer List.add(m3);
s = new Supplier(Manufacturer.manufacturerCount, capacity);
}
```

In buildModel() three manufacturers and one supplier are built. They are initialized with the values in brackets (e.g. initInventory,) and add to a list.[3]

- private void buildSchedule() {

```
ActionGroup firstGroup = new ActionGroup(ActionGroup.SEQUENTIAL);
BasicAction everyTickAction = new BasicAction() {
        public void execute() {
                int size = manufacturerList.size();
                s.setSuppliesReceived(manufacturerList);
                for(int i = 0; i < size; i++){
                Manufacturer m = (Manufacturer)manufacturerList.get(i);
                m.go();}
                s.getOrderRate(manufacturerList);
                graph.step();
        }
}
firstGroup.addAction(everyTickAction);
schedule.scheduleActionAt(1, firstGroup);
schedule.scheduleActionBeginning(2, everyTickAction);
}
```

The method buildSchedule() defines what actually happens during a simulation. Every simulation step the execute() function is called. Thus, in each simulation step the method setSuppliesReceived() of the supplier is executed, then the go()-method for all manufacturers and the getOrderRate() for the supplier are called. Finally the graph displaying the results is redrawn.

- Public String[ ] getInitParam(){

```
String[ ] params={"capacity","customerOrderRateM1",
"customerOrderRateM2","customerOrderRateM3"};
return params;
}
```

In this method it is defined which initial values can be set for a simulation run. After the start of the program a control panel is displayed in which the user can set these values.

In our example the user can set values for the capacity of the supplier (the amount of goods he can deliver) and the order rates of the manufacturers' customers.

- private void buildDisplay() {
```
graph.setYRange(95.0,130);
graph.setXRange(1,2);
graph.createSequence("Manufacturer 1", this, "getManufacturer0");
graph.createSequence("Manufacturer 2", this, "getManufacturer1");
graph.createSequence("Manufacturer 3", this, "getManufacturer2");
graph.setAxisTitles("Time", "Inventory");
}
```

The method buildDisplay() creates a graph to represent results of the simulation. In this case the values of the manufacturers' inventories are displayed.

- public static void main(String[ ] args) {
```
SimInit init = new SimInit();
Model model = new Model();
init.loadModel(model, null, false);
}
```

The main()-method is called first when executing the program. It builds and loads the model, a simulation run can be launched.

2. The manufacturer class

The manufacturer class describes the behavior of the manufacturing agents. In our example it manages the communication with Vensim, where the behavior is determined. Important methods of this class are loadVensimModel() and go():

- public void loadVensimModel(){
```
v = new Vensim();
VensimResult = v.command("SPECIAL>LOADMODEL|a:\\ Ordering.vmf" );
    if (VensimResult == 1)
    {System.out.println( "Vensim loaded model" ); }
}
```

This method loads via Vensim DLL the model *Ordering.vmf*. If the model is loaded successfully, "Vensim loaded model" is written to the standard output. It is executed each time when a new manufacturer is built (in the constructor of the manufacturer-class).

- public void go(){
```
VensimResult = v.command("GAME>GAMEINTERVAL|0.125");
VensimResult = v.command("MENU>GAME");
setValues();
VensimResult = v.command("GAME>GAMEON");
getValues();
}
```

Go() is a method that is called each time step of the simulation for each manufacturer. This method is responsible for the behavior of the agents. Via the Vensim DLL the system dynamics model *Ordering.vmf* built in Vensim is simulated (see Figure 2; modeled after Sterman [2000]). In each step of the agent-based simulation, also one step

of this simulation is run: the game interval is set to 0.125, the game is started, specific values of the manufacturer's Vensim model can be set, one step is simulated, and finally the results of this simulation step are returned.

In the supply chain management example the value for the goods delivered by the supplier (*Supplies Received*) is set before the next simulation step and *Orders Placed* as the output is returned and passed to the supplier.

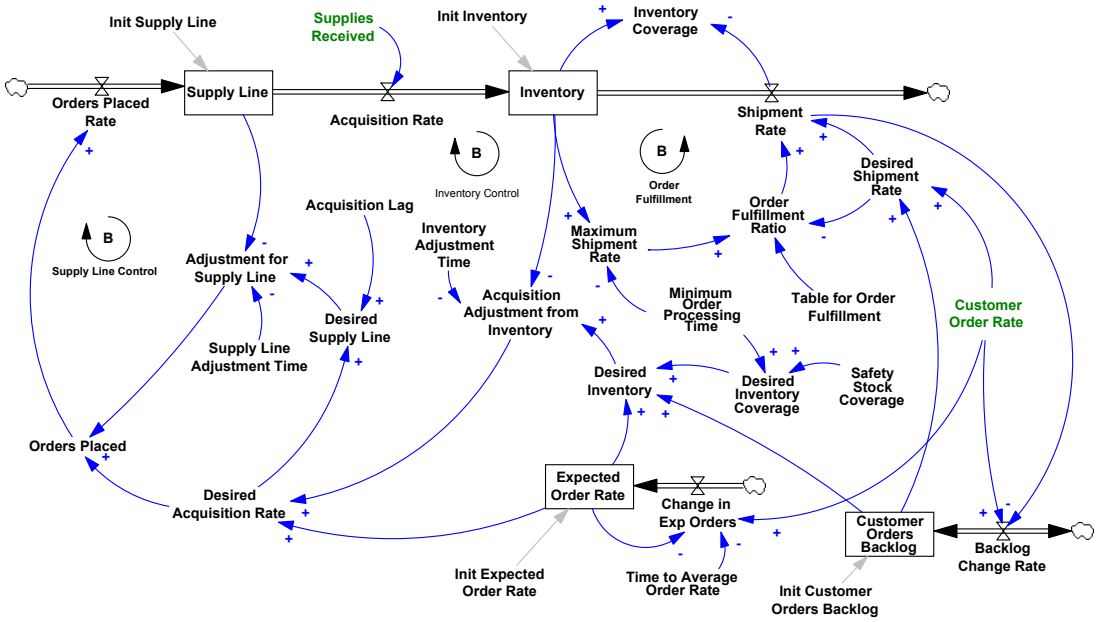

Figure 2: *Ordering.vmf*: behavior model of manufacturing agents

3. The supplier class

In the supplier class a very simplistic behavior of a supplier agent is described. This class receives the orders (*Orders Placed*) of the manufacturers (the output of the system dynamics simulation runs), and returns the amount of delivered goods (*Supplies Received*) as an input for the next simulation step. Important methods in this class are setSuppliesReceived() and getOrderRate():

- public void setSuppliesReceived(List list){
  for(int i = 0; i <manufacturerCount; i++){
  Manufacturer m = (Manufacturer)list.get(i);
  if (m.Simtime>0)
  { if(capacity < sumOrderBacklog)
  {m.suppliesReceived[0]
  =(orderBacklog[m.ID]/sumOrderBacklog)*capacity;}
  else m.suppliesReceived[0] = orderBacklog[m.ID];
  orderBacklog[m.ID] = orderBacklog[m.ID] - m.suppliesReceived[0];}
  }
}

SetSuppliesReceived() stores the amount of goods that are delivered to all manufacturers in the manufacturers' auxiliary variables suppliesReceived. The amount depends on how much the supplier can deliver (its capacity), how much the manufacturers have ordered and how many orders exist in the order backlog of the supplier. In the case that the supplier cannot deliver the full amount, its capacity is distributed proportionally according to the orders placed by the manufacturers. Orders that are not delivered are stored in the customer backlog of the supplier.

- public void getOrderRate(List list){
  sumOrderBacklog=0;
  for(int i = 0; i < manufacturerCount; i++){
  Manufacturer m = (Manufacturer)list.get(i);
  orderRate[m.ID]=m.orderRate[0];
  orderBacklog[m.ID] = orderBacklog[m.ID] + orderRate[m.ID];
  sumOrderBacklog = sumOrderBacklog + orderBacklog[m.ID]; }
  }

This method receives the amount of orders placed by all manufacturers. In this way it passes the output of the Vensim simulation (*Orders Placed*) to the supplier.

To run the simulation described in this paper, the user first has to install RePast, Java Development Kit JDK and Vensim including Vensim DLL. Having installed the needed runtime environment the Java class Model.class can be executed. Before a simulation is started, the user can set initial values of the simulation in the displayed settings window (see Figure 3). However, agents' properties can also be altered during simulation. Pushing the "Start" button in the window depicted in Figure 4 initiates the simulation and the results of the simulation are represented by a graph (see Figure 5).

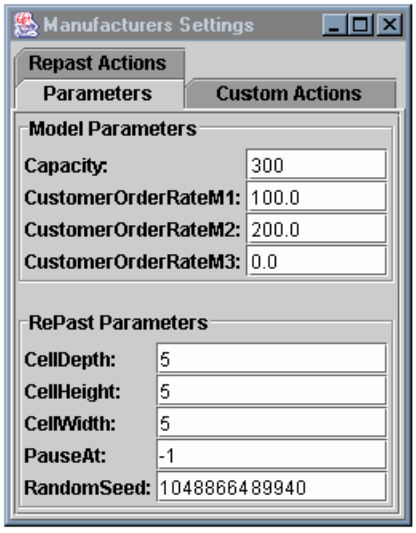

Figure 3: Default settings window for example application

Figure 4: Simulation control panel of RePast

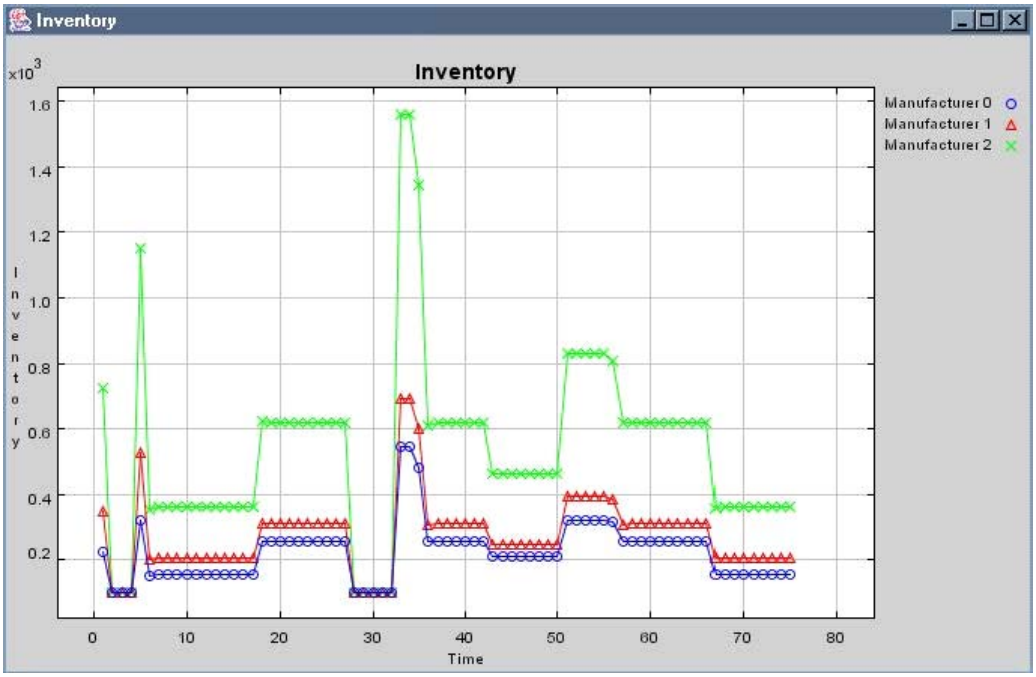

Figure 5: Exemplary output of simulation run

One drawback of the software interface presented here is that basic programming knowledge is needed to work with RePast. However, the example so far is only a prototype which is meant to be a paradigm of the technical procedure when integrating RePast and Vensim. From an application perspective, closer to reality problems can be identified in which the combination of agent-based and system dynamics modeling and simulation might be beneficial. For instance, in supply chain management dynamic reconfigurations of a chain's structure and its effects on the chain's performance can be investigated. The software solution presented in this paper provides a common technical platform to examine this and other problems that suggest the integration of the two simulation concepts.

References

Akkermans, HA. 2001. Emergent Supply Networks: System Dynamics Simulation of Adaptive Supply Agents. *Proceedings of the 34th Hawaiian International Conference on Systems Science*, Wailea.

Anderson, P. 1999. Complexity Theory and Organization Science. *Organization Science* 10(3): 219–232.

Phelan, SE. 1999. A Note on the Correspondence between Complexity and Systems Theory. *Systemic Practice and Action Research* **12**(3): 237–246.

Pourdehnad, J, Maani, KE, Sedehi, H. 2002. System Dynamics and Intelligent Agent-Based Simulation: Where is the Synergy? *Proceedings of the 20th International Conference of the System Dynamics Society*, Palermo.

Schieritz, N. 2002. Integrating System Dynamics and Agent-Based Modeling. *Proceedings of the 20th International Conference of the System Dynamics Society*, Palermo.

Schieritz, N, Größler, A. 2003. Emergent Structures in Supply Chains: A Study Integrating Agent-Based and System Dynamics Modeling. *Proceedings of the 36th Hawaiian International Conference on Systems Science*, Wailea.

Schieritz, N, Milling, PM. 2003. Modeling the Forest or Modeling the Trees: A Comparison of System Dynamics and Agent-Based Simulation. *Proceedings of the 21st International Conference of the System Dynamics Society*, New York.

Scholl, HJ. 2001a. Agent-based and System Dynamics Modeling: A Call for Cross Study and Joint Research. *Proceedings of the 34th Hawaiian International Conference on Systems Science*, Wailea.

Scholl, HJ. 2001b. Looking Across the Fence: Comparing Findings From SD Modeling Efforts With those of Other Modeling Techniques. *Proceedings of the 19th International Conference of the System Dynamics Society*, Atlanta.

Sterman, JD. 2000. *Business Dynamics – Systems Thinking and Modeling for a Complex World*, Boston.

Notes

1. One might consider the internal decision logic or cognitive structure of an agent as its “schema” or “mental model”.
2. Vensim, RePast, eM-Plant and Java are registered trademarks.
3. Complete program and model listings are available by the authors on request.

Back to the Top