Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds

MULTI-METHOD MODELING

Andrei Borshchev

The AnyLogic Company
49 Nepokorennykh ave
St.Petersburg, 195220, RUSSIA

ABSTRACT

Frequently, the problem cannot completely conform to one of the three existing modeling paradigms (discrete event, system dynamics, or agent based modeling). Thinking in terms of a single-method modeling language, the modeler inevitably either starts using workarounds (unnatural and cumbersome constructs), or just leaves part of the problem outside the scope of the model (treats it as exogenous). If our goal is to capture business, economic, and social systems in their interaction, this becomes a serious limitation. In this paper we offer an overview of most used multi-method (or multi-paradigm) model architectures, discuss the technical aspects of linking different methods within one model, and consider examples of multi-method models. The modeling language of AnyLogic is used throughout the paper.

1 INTRODUCTION

The three modeling methods, or paradigms, that exist today, are essentially the three different viewpoints the modeler can take when mapping the real world system to its image in the world of models.

- The **system dynamics** paradigm suggests to abstract away from individual objects, think in terms of aggregates (stocks, flows), and the feedback loops.
- The **discrete event modeling** adopts a process-oriented approach: the dynamics of the system are represented as a sequence of operations performed over entities.
- In an **agent based model** the modeler describes the system from the point of view of individual objects that may interact with each other and with the environment.

Depending on the simulation project goals, the available data, and the nature of the system being modeled, different problems may call for different methods. Also, sometimes it is not clear at the beginning of the project which abstraction level and which method should be used. The modeler may start with, say, a highly abstract system dynamics model and switch later on to a more detailed discrete event model. Or, if the system is heterogeneous, the different components may be best described by using different methods. For example in the model of a supply chain that delivers goods to a consumer market the market may be described in system dynamics terms, the retailers, distributors, and producers may be modeled as agents, and the operations inside those supply chain components – as process flowcharts.

Frequently, the problem cannot completely conform to one modeling paradigm. A "single-method-minded" modeler inevitably either starts using workarounds (unnatural and cumbersome language constructs), or just leaves part of the problem outside the scope of the model. If our goal is to capture business, economic, and social systems in their interaction, this becomes a serious limitation. In this paper we offer an overview of most used multi-method model architectures, discuss the technical aspects of linking different methods within one model, and consider two examples of multi-method models:

- Consumer market and supply chain
- Epidemic and clinic

2 MULTI-METHOD MODEL ARCHITECTURES

The number of possible multi-method model architectures is infinite, and many are used in practice. Popular examples are shown in the Figure 1. In this section we briefly discuss the problems where these architectures may be useful.

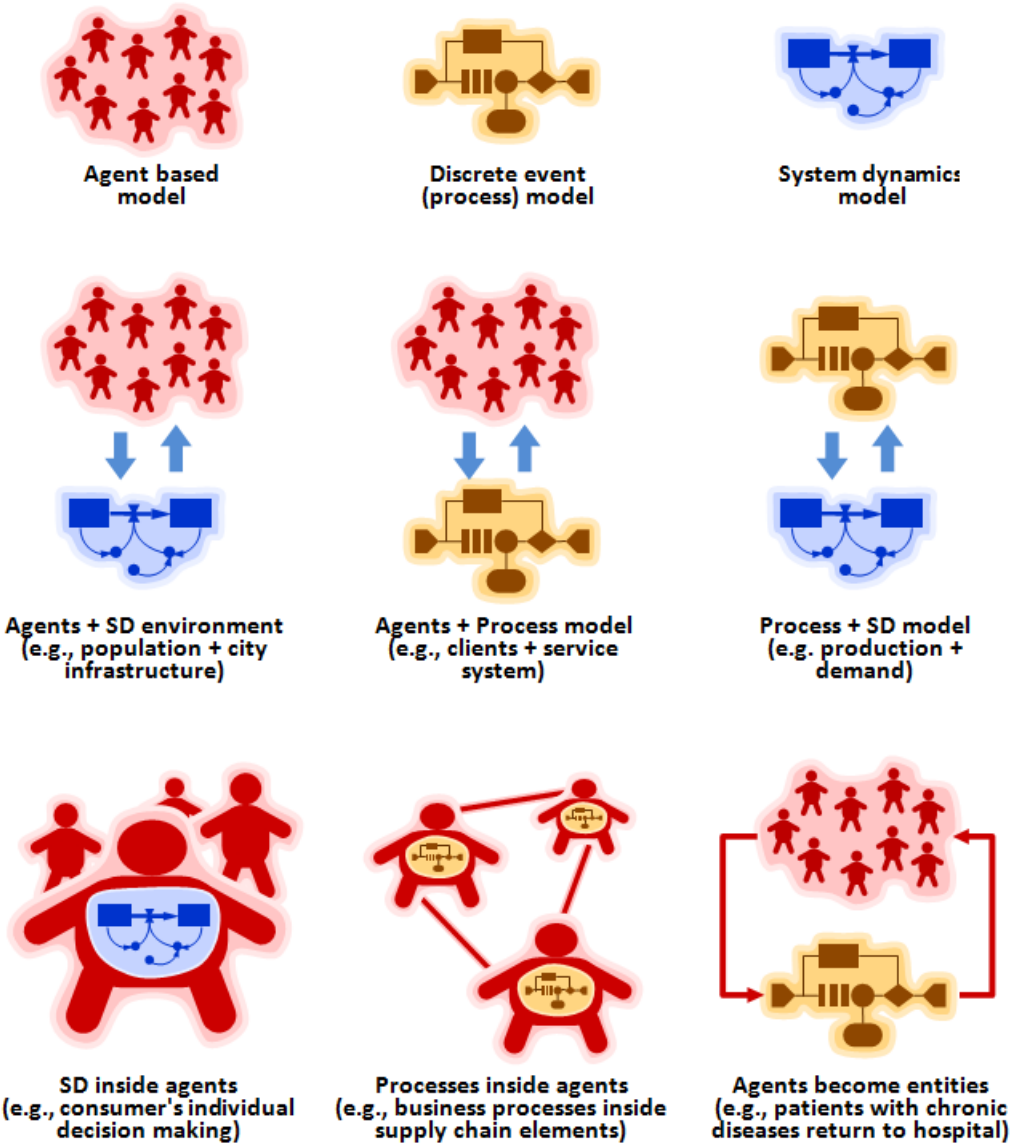

Figure 1: Popular multi-method model architectures

Agents in an SD environment. Think of a demographic model of a city. People work, go to school, own or rent homes, have families, and so on. Different neighborhoods have different levels of comfort, including infrastructure and ecology, cost of housing, and jobs. People may choose whether to stay or move to a different part of the city, or move out of the city altogether. People are modeled as agents. The dynamics of the city neighborhoods may be modeled in system dynamics way, for example, the home prices and the overall attractiveness of the neighborhood may depend on crowding, and so on. In such a

model agents' decisions depend on the values of the system dynamics variables, and agents, in turn, affect other variables.

The same architecture is used to model the interaction of public policies (SD) with people (agents). Examples: a government effort to reduce the number of insurgents in the society; policies related to drug users or alcoholics.

Agents interacting with a process model. Think of a business where the service system is one of the essential components. It may be a call center, a set of offices, a Web server, or an IT infrastructure. As the client base grows, the system load increases. Clients who have different profiles and histories use the system in different ways, and their future behavior depends on the response. For example, low-quality service may lead to repeated requests, and, as a result, frustrated clients may stop being clients. The service system is naturally modeled in a discrete event style as a process flowchart where requests are the entities and operators, tellers, specialists, and servers are the resources. The clients who interact with the system are the agents who have individual usage patterns.

Note that in such a model the agents can be created directly from the company CRM database and acquire the properties of the real clients. This also applies to the modeling of the company's HR dynamics. You can create an agent for every real employee of the company and place them in the SD environment that describes the company's integral characteristics (the first architecture type).

A process model linked to a system dynamics model. The SD aspect can be used to model the change in the external conditions for an established and ongoing process: demand variation, raw material pricing, skill level, productivity, and other properties of the people who are part of the process.

The same architecture may be used to model manufacturing processes where part of the process is best described by continuous time equations – for example, tanks and pipes, or a large number of small pieces that are better modeled as quantities rather than as individual entities. Typically, however, the rates (time derivatives of stocks) in such systems are piecewise constants, so simulation can be done analytically, without invoking numerical methods.

System dynamics inside agents. Think of a consumer market model where consumers are modeled individually as agents, and the dynamics of consumer decision making is modeled using the system dynamics approach. Stocks may represent the consumer perception of products, individual awareness, knowledge, experience, and so on. Communication between the consumers is modeled as discrete events of information exchange.

A larger-scale example is interaction of organizations (agents) whose internal dynamics are modeled as stock and flow diagrams.

Processes inside agents. This is widely used in supply chain modeling. Manufacturing and business processes, as well as the internal logistics of suppliers, producers, distributors and retailers are modeled using process flowcharts. Each element of the supply chain is at the same time an agent. Experience, memory, supplier choice, emerging network structures, orders and shipments are modeled at the agent level.

Agents temporarily act as entities in a process. Consider patients with chronic diseases who periodically need to receive treatment in a hospital (sometimes planned, sometimes because of acute phases). During treatment, the patients are modeled as entities in the process. After discharge from the hospital, they do not disappear from the model, but continue to exist as agents with their diseases continuing to progress until they are admitted to the hospital again. The event of admission and the type of treatment needed depend on the agent's condition. The treatment type and timeliness affect the future disease dynamics.

There are models where each entity is at the same time an agent exhibiting individual dynamics that continue while the entity is in the process, but are outside the process logic – for example, the sudden deterioration of a patient in a hospital.

2.1 The choice of the model architecture and methods

The choice of the model architecture, the abstraction level(s), and the method(s) always depends on the problem you are solving. Among other things, this choice should be governed by the criterion of *naturalness*. Compact, minimalistic, clean, beautiful, easy to understand and explain – if the internal texture of your model is like that, then your choice was most probably right. The design patterns and model examples further in this paper are given in the AnyLogic modeling language (see, for example, Borshchev and Filippov 2004). This is a multi-method object-oriented language designed on the basis of "no workarounds" principle. This language allows you to create model architectures of arbitrary type and complexity, including all previously mentioned.

3 TECHNICAL ASPECT OF COMBINING DIFFERENT MODELING METHODS

In this section we will consider the techniques of linking different modeling methods together. Important feature of the modeling language we are using is that all model elements of all methods, be they SD variables, statechart states, entities, process blocks, exist in the "same namespace": any element is accessible from any other element by name (and, sometimes, "path" – the prefix describing the location of the element in the model hierarchy). The following examples are all taken from the real projects and purged of all unnecessary details. This set, of course, does not cover everything, but it does give a good overview of how you can build interfaces between different methods.

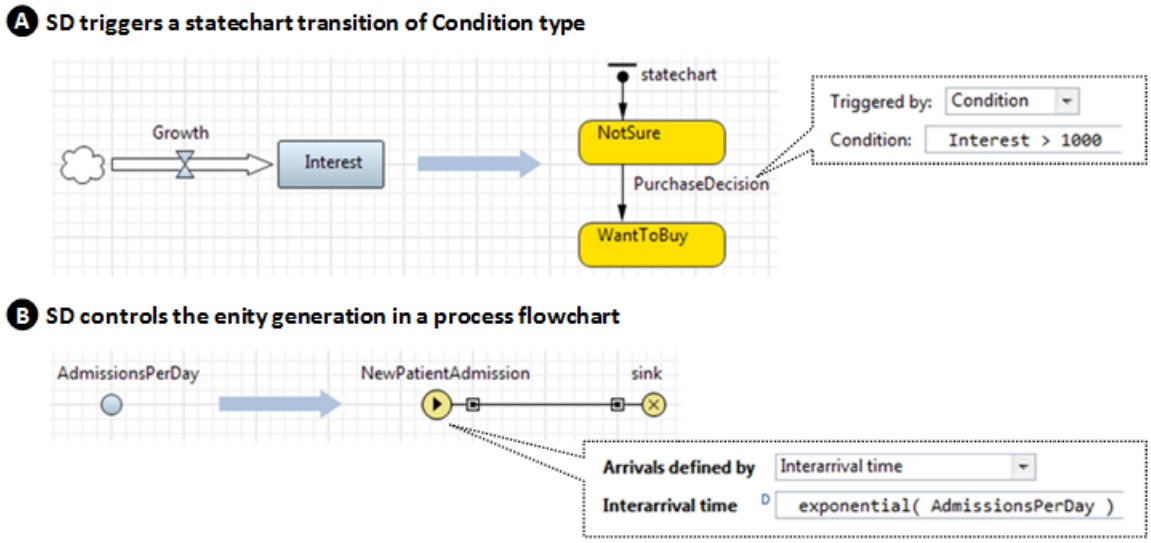

Figure 2: SD impacts discrete elements of the model

3.1 System dynamics impacts discrete elements

The system dynamics model is a set of continuously changing variables. All other elements in the model work in discrete time (where any changes are associated with events). SD itself does not generate any events, so it cannot actively make an impact on agents, process flowcharts, or other discrete time constructs. The only way for the SD part of the model to impact a discrete element is to let that element watch on a condition over SD variables, or to use SD variables when making a decision. The Figure 2 shows some possible constructs.

In the Figure 2 case A an SD variable triggers a statechart transition. Events (low-level constructs that allow scheduling a one-time or recurrent action) and statechart transitions are frequent elements of agent behavior. Among other types of triggers, both can be triggered by a condition – a Boolean expression. If

the model contains dynamic variables, all conditions of events and statechart transitions are evaluated at each integration step, which ensures that the event or transition will occur exactly when the (continuously changing) condition becomes true. In the figure the statechart is waiting for the *Interest* stock to rise higher than a given threshold value. In the Figure 2 case B the flowchart source block *NewPatientAdmissions* generates new entities at the rate defined by the dynamic variable *AdmissionsPerDay*, which may be a part of a stock and flow diagram.

3.2 Discrete elements impact system dynamics

Consider the Figure 3. In case C, the SD stock triggers a statechart transition, which, in turn, modifies the stock value. Here, the interface between the SD and the statechart is implemented in the pair condition/action. In the state *WantToBuy*, the statechart tests if there are products in the retailer stock, and if there are, buys one and changes the state to *User*.

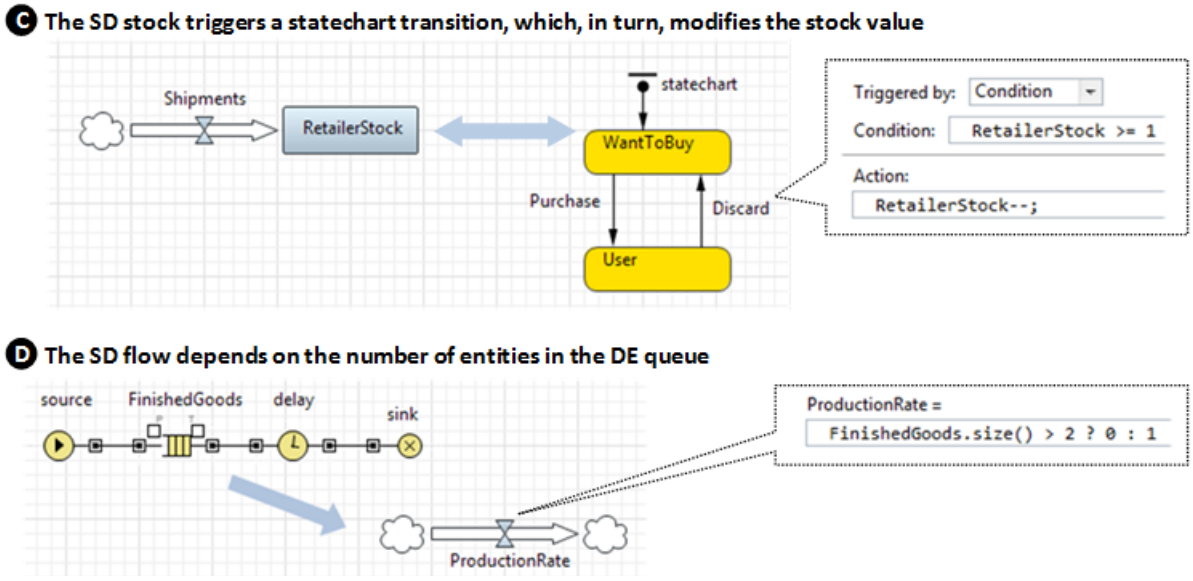

Figure 3: Discrete elements of the model impact SD

You can freely change the values of the system dynamic stocks from outside the system dynamics part of the model. This does not interfere with the differential equation solving: the integrator will just start with the new value. However, trying to change the value of a flow or auxiliary variable that has an equation associated with it, is not correct: the assigned value will be immediately overridden by the equation, so assignment will have no effect

DE objects can be referenced in a SD formula. In Figure 3 case D, the flow *ProductionRate* switches between 0 and 1, depending on whether the finished products inventory (the number of entities in the queue *FinishedGoods* returned by the function `size()`) is greater than 2 or not. Again, one can close the loop by letting the SD part control the production process.

3.3 Interaction of agent based and discrete event parts of the model

In the Figure 4 case E a server in the DE process model is implemented as an agent. Imagine complex equipment, such as a robot or a system of bridge cranes. The behavior of such objects is often best modeled "in agent based style" by using events and statecharts. If the equipment is a part of the manufacturing process being modeled, you need to build an interface between the process and the agent representing the equipment.

In this example, the statechart is a simplified equipment model. When the statechart comes to the state *Idle*, it checks if there are entities in the queue. If yes, it proceeds to the *Working* state and, when the work is finished, unblocks the *hold* object, letting the entity exit the queue. The *hold* object is set up to block itself again after the entity passes through.

The next entity will arrive when the equipment is in the *Idle* state. To notify the statechart, we call the function `onChange()` upon each entity arrival (see the On enter action of the queue). This is necessary because, unlike in the models with continuously changing SD elements, in the models built of purely discrete elements events and transitions triggered by a condition, do not monitor the condition continuously.

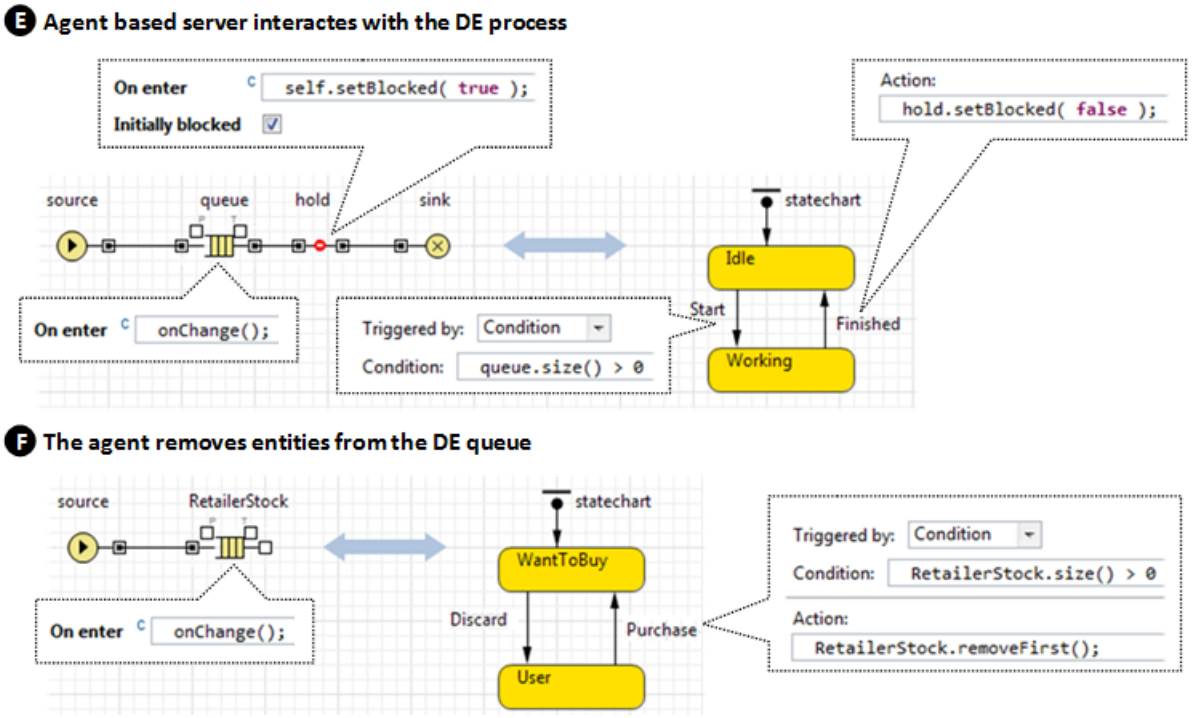

Figure 4: Interaction of AB and DE parts of the model

In the case F the agent removes entities from the DE queue. Here, the supply chain is modeled using discrete event constructs; in particular, its end element, the retailer stock, is a Queue object. The consumers are outside the discrete event part, and they are modeled as agents. Whenever a consumer comes to the state *WantToBuy*, it checks the *RetailerStock* and, if it is not empty, removes one product unit. Again, as this is a purely discrete model, we need to ensure that the consumers who are waiting for the product are notified about its arrival – that's why the code `onChange()` is placed in the On enter action of the *RetailerStock* queue.

In this simplified version, there is only one consumer whose statechart is located "on the same canvas" as the supply chain flowchart. In the full version there would be multiple agents-consumers and, instead of calling just `onChange()`, the retailer stock would notify every consumer.

4 EXAMPLES OF MULTI-METHOD MODELS

4.1 Consumer market and supply chain

We will model the supply chain and sales of a new product in a consumer market in the absence of competition. The supply chain will include the delivery of the raw product to the production facility, produc-

tion, and the stock of the finished products. The QR inventory policy will be used. Consumers are initially unaware of the product; advertizing and word of mouth will drive the purchase decisions. The product has a limited lifetime, and 100% of users will be willing to buy a new product to replace the old one. The full version of the model is available at RunTheModel.com.

We will use discrete event methodology to model the supply chain, and system dynamics methodology, namely, a slightly modified Bass diffusion model (Bass 1969), to model the market. We will link the two models through the purchase events.

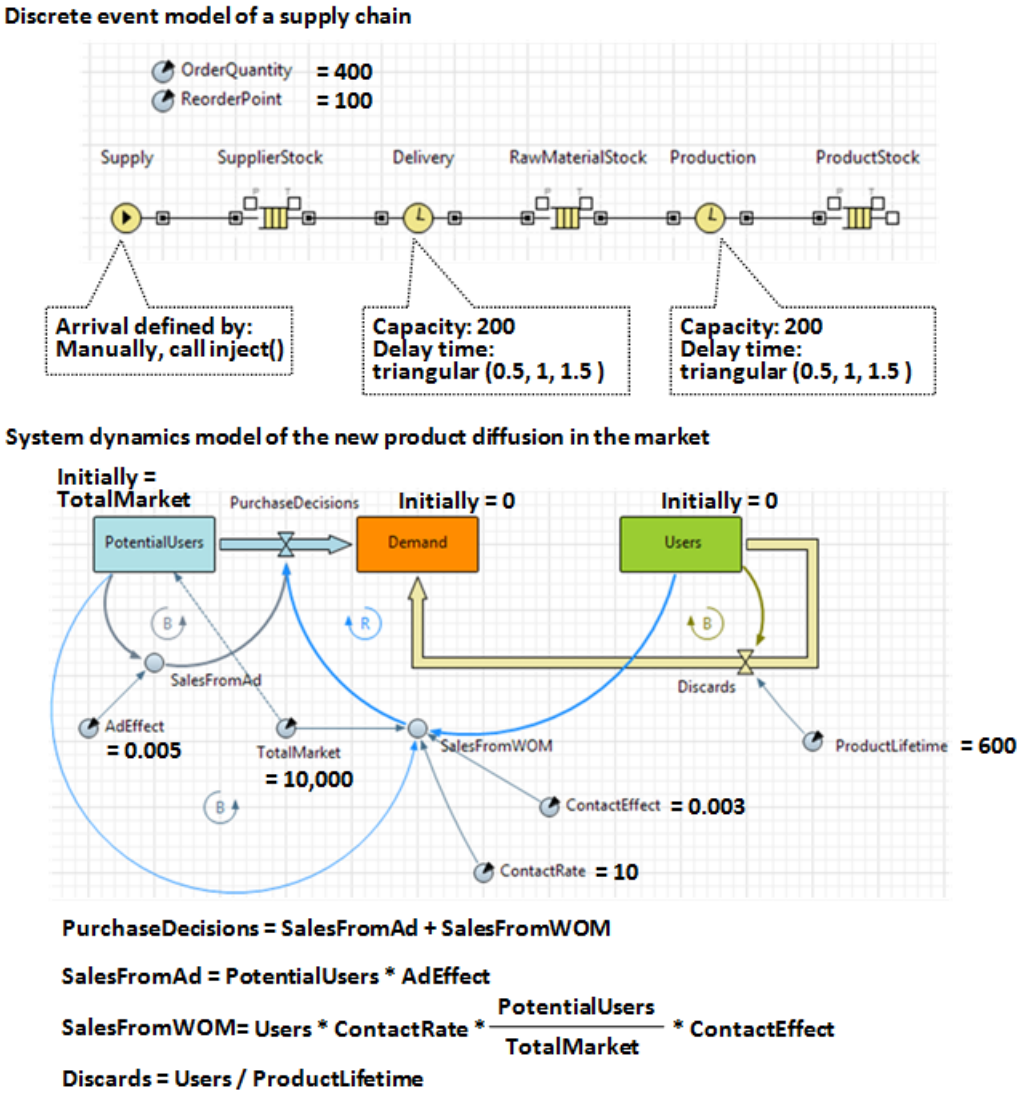

Figure 5: DE model of the supply chain and SD model of the market (so far, not linked)

The supply chain flowchart (top of the Figure 5) includes three stocks: the supplier stock of raw material, the stock of raw material at the production site, and the stock of finished products at the same location. Delivery and production are modeled by the two Delay objects with limited capacity. The *Supply* block of the flowchart is not generating any entities unless explicitly asked to do so (the inventory policy is not yet present at this stage). To load the supply chain with some initial product quantity we will add

this Startup code: `Supply.inject( OrderQuantity );`. If we run this model, at the beginning of the simulation, four hundred items of the product are produced and accumulate in the *ProductStock*.

The market is modeled by a system dynamics stock and flow diagram as shown in the bottom of the same Figure 5. The SD part is located on the same canvas where the flowchart was created earlier. The difference of this market model from the classical Bass diffusion model with discards (Sterman 2000) is that the users, or adopters, stock of the classical model is split into two: the *Demand* stock and the actual *Users* stock. The adoption rate in this model is called *PurchaseDecisions*. It brings *PotentialUsers* not directly into the *Users* stock, but into the intermediate stock *Demand*, where they wait for the product to be available. The actual event of sale, i.e., "meeting" of the product and the customer who wants to buy it, will be modeled outside the system dynamics paradigm. If we run this part of the model alone, the potential clients will gradually make their purchase decisions (triggered by advertizing), building up the *Demand* stock.

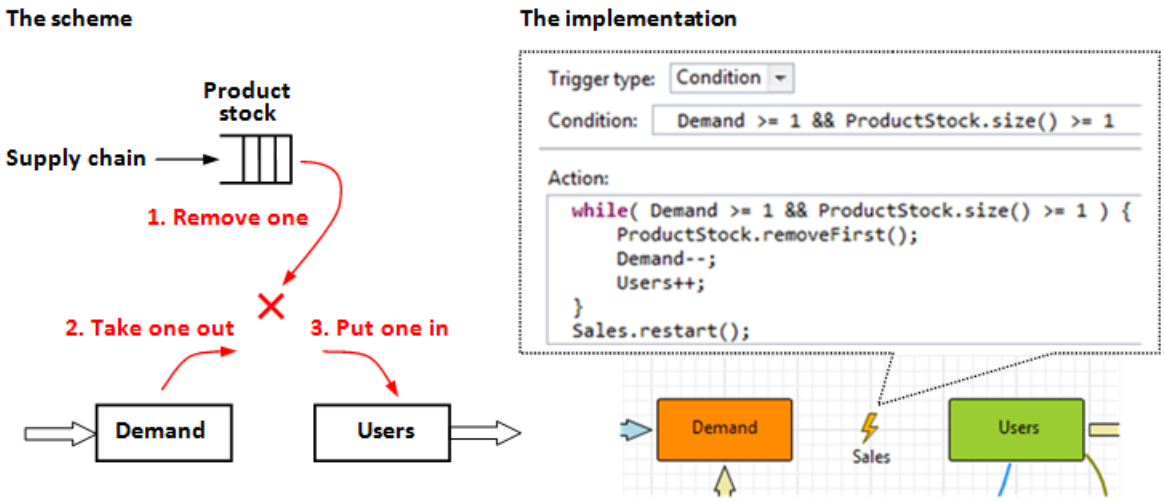

Figure 6: Linking the supply chain and the market

How do we link the supply chain and the market? We want to achieve the following:

- If there is at least one product item in stock and there is at least one client who wants to buy it, the product item should be removed from the *ProductStock* queue, the value of *Demand* should be decremented, and the value of *Users* should be incremented, see the Figure 6 on the left.

We, therefore, have a condition and an action that should be executed when the condition is true. The construct that does exactly that is the condition-triggered event. The implementation of the scheme shown in same figure on the right. The condition-triggered events work this way: in the presence of continuous dynamics in the model the condition of the event is evaluated at each numeric micro-step. Once the condition evaluates to true, the event's action is executed.

In the Action code of the event we put the sale into a `while` loop, because a possibility exists that two or more product items may become available simultaneously, or the *Demand* stock may grow by more than one unit per numeric step. Therefore, more than one sale can potentially be executed per event occurrence. By default, the condition event disables itself after execution. As we want it to continue monitoring the condition, we explicitly call `restart()` at the end of the event's action.

With the *Sales* event in place, the sales start to happen, and the 400 items produced in the beginning of the simulation disappear in about a week. The *Users* stock increases up to almost 400; it then slowly starts to decrease according to our limited lifetime assumption. And, since we have not implemented the inventory policy yet, no new items are produced. This is the last missing piece of the model. We will in-

clude the inventory policy in the same Sales event; the inventory level will be checked after each sale. The following piece of code is added to the Action of the *Sales* event:

```
//apply inventory policy
int inventory = //calculate inventory
      ProductStock.size() + //in stock
      Production.size() + //in production
      RawMaterialStock.size() + //raw product inventory
      Delivery.size() + //raw product being delievered
      SupplierStock.size(); //supplier's stock
if( inventory < ReorderPoint ) //QR policy
   Supply.inject( OrderQuantity );
```

Now, the supply chain starts to work as planned, see the Figure 7. During the early adoption phase the supply chain performs adequately, but as the majority of the market starts buying, the supply chain cannot keep up with the market. In the middle of the new product adoption (days 40-100), even though the supply chain works at its maximum throughput, still the number of waiting clients remains high. As the market becomes saturated, the sales rate reduces to the replacement purchases rate, which equals the Discard rate in the completely saturated market, namely *TotalMarket / ProductLifetime* = 16.7 sales per day. The supply chain handles that easily.

An interesting exercise would be to make the supply chain adaptive. You can try to minimize the order backlog and at the same time minimize the inventory by adding the feedback from the market model to the supply chain model.

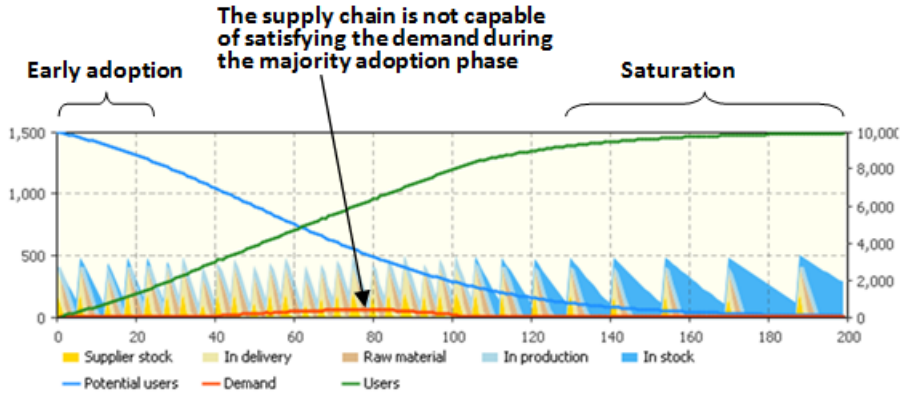

Figure 6: The supply chain dynamics pattern changes as the market gets saturated

4.2 Epidemic and clinic

We will create a simple agent based epidemic model and link it to a simple discrete event clinic model. When a patient discovers symptoms, he will ask for treatment in the clinic, which has limited capacity. We will explore how the capacity of the clinic affects the disease dynamics. This model was suggested in 2012 by Scott Hebert, a consultant at AnyLogic North America. The full version of the model is available at RunTheModel.com.

We will model patients as agents (individual objects); their initial number is 2,000. The patients will be connected with a distance-based network: two patients are connected if they live at most 30 miles away.

The behavior of a patient will be modeled by a statechart (see the Figure 7) structured according to the classical SEIR compartmental epidemiology model. The patient is initially in the *Susceptible* state, where he can be infected. Disease transmission is modeled by the message "Infection" sent from one patient to another. Having received such a message, the patient transitions to the state *Exposed*, where he is already infectious, but does not have symptoms. After a random incubation period, the patient discovers

symptoms and proceeds to the *Infected* state. We distinguish between the *Exposed* and *Infected* states because the contact behavior of the patient is different before and after the patient discovers symptoms: the contact rate in the *Infected* state is significantly lower. The internal transitions in both states model contacts. We model only those contacts that result in disease transmission; therefore, we multiply the base contact rate by *Infectivity*, which, in our case, is 7%.

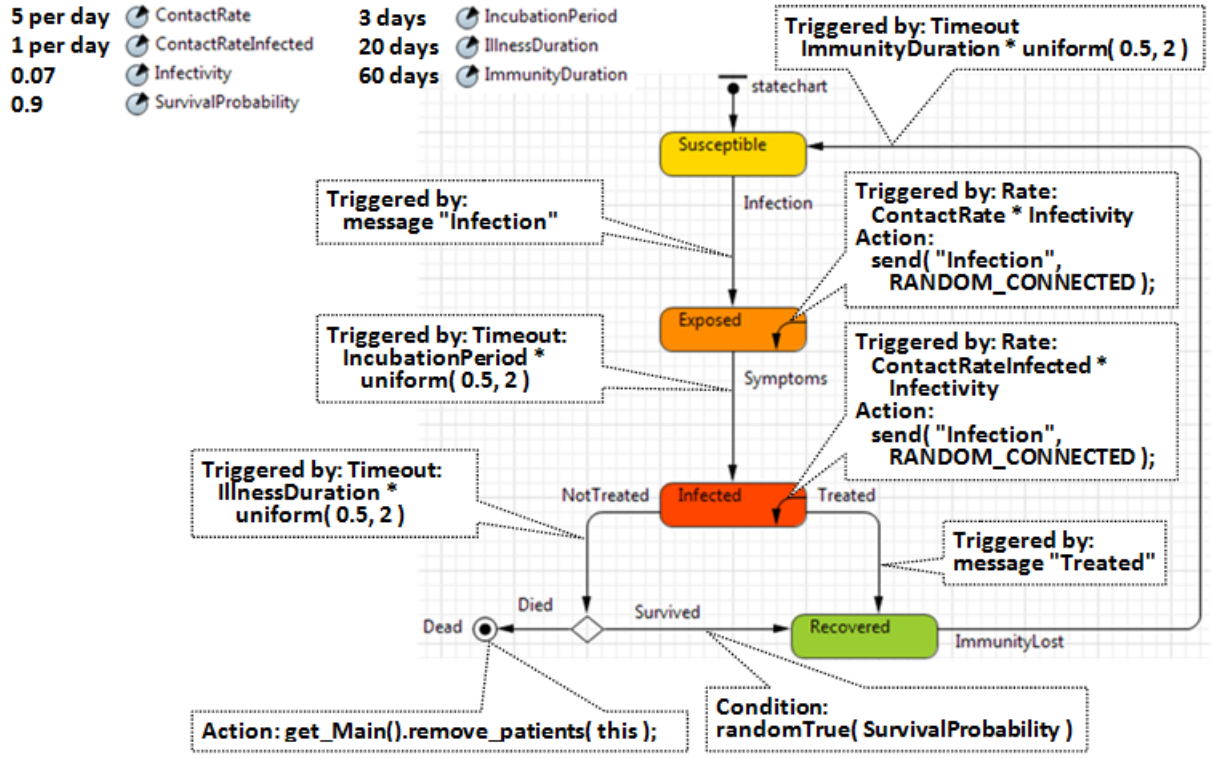

Figure 7: Patient behavior: a statechart built according to the SEIR epidemic model

There are two possible exits from the *Infected* state. The patient can be treated in a clinic (and then, he is guaranteed to recover), or the illness may progress naturally without intervention. In the latter case, the patient can still recover with a high probability, or die. If the patient dies, it deletes himself from the model, see the Action of the *Dead* state. The completion of treatment is modeled by the message "Treated" sent to the agent. In the absence of the clinic model, this message is, of course, never received.

The recovered patient acquires a temporary immunity to the disease. We reflect this in the model by having the state *Recovered*, where the patient does not react to the message "Infection" that may possibly arrive. At the end of the immunity period the transition *ImmunityLost* takes the patient back to the *Susceptible* state. We need to create the initial entry of the infection into the population. This can be done at the top level of the model, by, for example, sending the message "Infection" to a few randomly chosen people in the beginning of the simulation.

If you run the epidemic model at this stage, you will see the dynamics like shown in the Figure 8. The epidemic does not end after the first wave, because the immunity period is not long enough.

The next step is to add the clinic, and let the patients be treated there. Our clinic will be modeled via a very simple discrete event model: the Queue for the patients waiting to be treated and the Delay modeling the actual treatment. We will put the process flowchart (see the bottom right of the Figure 9) at the top level of the model. Unlike in classical discrete event models, however, the entities in this process are not

generated by a Source object, but are injected by the agents via an Enter object. The communication scheme between the patients-agents and the clinic process is shown in the Figure 9 on the left.

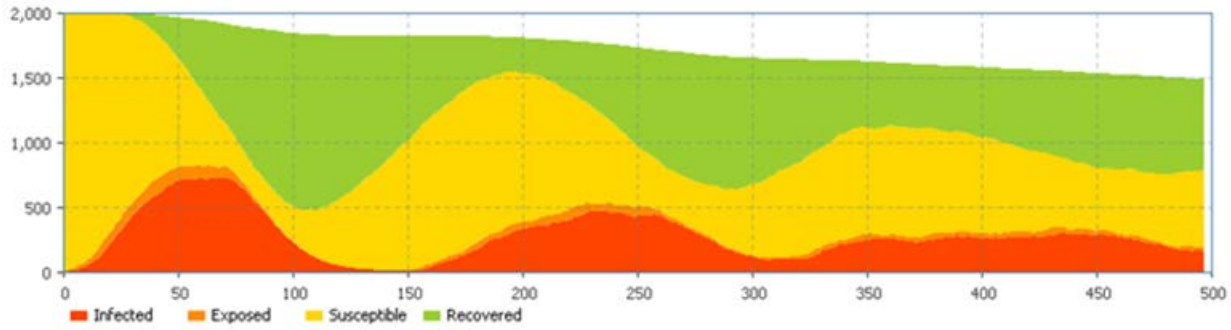

Figure 8: The natural dynamics of the epidemic (patients are not treated)

Once the patient discovers symptoms, he creates an entity – let's call it "treatment request" – and injects the entity into the clinic process. Once the treatment is completed, the entity notifies the patient by sending him a message "Treated" that causes the patient to transition to the Recovered state. If, however, the patent is cured or dies before the treatment is completed, he will discard his treatment request by removing it from whatever stage in the process it is.

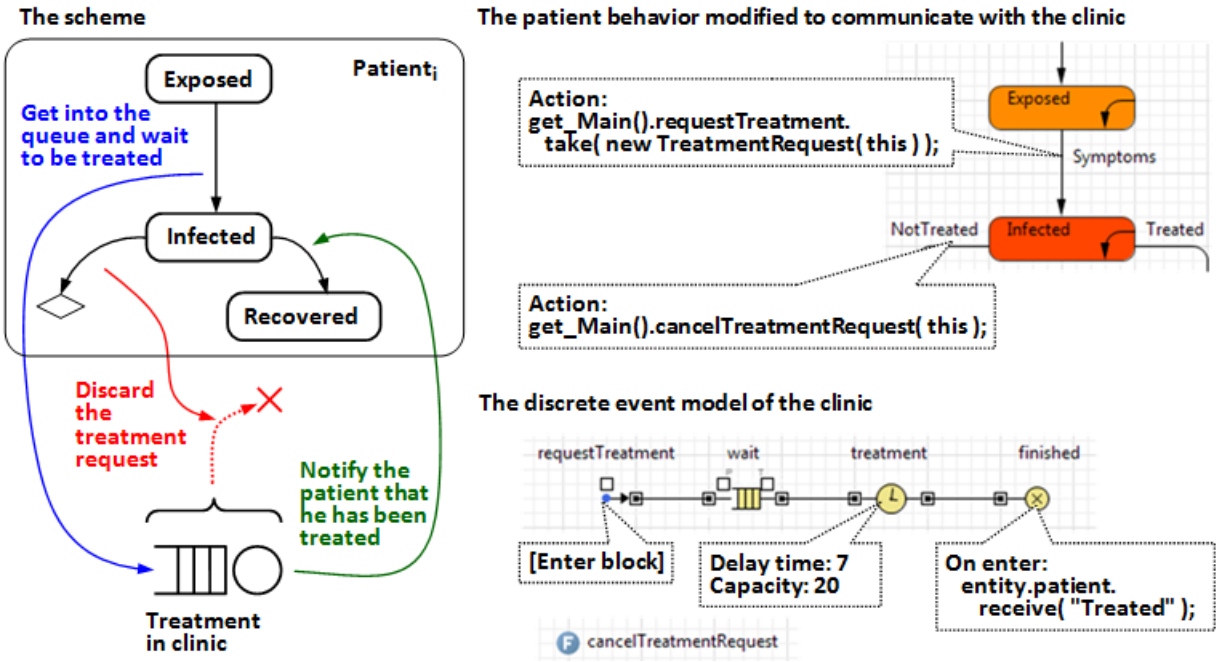

Figure 9: The model of the clinic linked to the agent based model of epidemic

With the clinic added, the model shows a different dynamic, or, to be more precise, a different range of dynamics. The oscillations are still possible, but a possibility also exists that the epidemic will end after the first wave, see the Figure 10. One may experiment with different clinic capacities to figure out the number of beds needed in order to treat everybody on time and prevent the further waves of the epidemic.

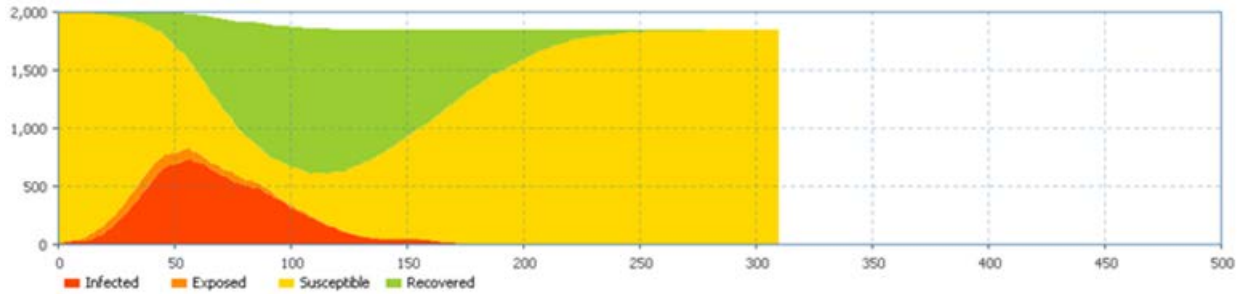

Figure 10: The epidemic dynamics with patients being treated in the clinic

5 DISCUSSION

When developing a discrete event model of a supply chain, IT infrastructure, or a contact center, the modeler would typically ask the client to provide the arrival rates of the orders, transactions, or phone calls. He would then be happy to get some constant values, periodical patterns, or trends, and treat arrival rates as variables exogenous to the model. In reality, however, those variables are outputs of another dynamic system, such as a market, a user base. Moreover, that other system can, in turn, be affected by the system being modeled. For example, the supply chain cycle time, which depends on the order rate, can affect the satisfaction level of the clients, which impacts repeated orders and, through the word of mouth, new orders from other customers. The choice of the model boundary therefore is very important.

The only methodology that explicitly talks about the problem of model boundary is system dynamics (Sterman 2000). However, the system dynamics modeling language is limited by its high level of abstraction, and many problems cannot be addressed with the necessary accuracy. With multi-method modeling you can choose the best-fitting method and language for each component of your model and combine those components while staying on one platform.

REFERENCES

Bass, F. 1969. *A new product growth model for consumer durables.* Management Science 15 (5): p215–227.

Borshchev, A, and A. Filippov. 2004. *From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools.* In Proceedings of the 22nd International Conference of the System Dynamics Society, Oxford, England.

Compartmental models in epidemiology. Wikipedia article. http://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology.

Sterman, J. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Irwin/McGraw-Hill.

AUTHOR BIOGRAPHY

ANDREI BORSHCHEV is CEO and co-founder of The AnyLogic Company (former name XJ Technologies). He received his Ph.D. in Computer Science and Complex Systems Modeling from St.Petersburg Polytechnic University. Since 1998 Andrei has been leading the development of AnyLogic software product. His interests include applied agent based modeling, multi-method simulation modeling, hybrid discrete/continuous simulation, design of languages for dynamic simulation. Andrei has conducted numerous lectures, seminars, and training sessions on simulation modeling and AnyLogic. He is a member of the System Dynamics Society, and a constant participant and speaker at International System Dynamics Conference, Winter Simulation Conference, and INFORMS Annual Meeting. His e-mail address is andrei@anylogic.com and the company website is http://www.anylogic.com.