Including events in DynRisk models

There are essentially four main categories of events:

- “Send data” events
- “Receive data” events
- “Send and receive data” events
- “Trigger action” events

A “Send data” event adds all the input values it gets from its predecessors, and sends this sum to its target application. Nothing (i.e., the number zero) is passed on to its successors.

A “Receive data” event sends a request for a certain result (a single real number) to its target application and passes this result on to its successors.

A “Send and receive data” event adds all the input values it gets from its predecessors, and sends this sum together with a request for a certain result (a single real number) to its target application and passes this result on to its successors. The received result is typically some function of the input sum.

A “Trigger action” event simply asks its target application to carry out a certain action, e.g., recalculate a spreadsheet. Nothing (i.e., the number zero) is passed on to its successors.

When you include events in a model, a good way to start is to find out which of the above categories the different events belong to, and in which order their respective messages should be sent.

To illustrate this, we consider a simple example. Suppose you want your model to feed random numbers into a certain cell in a spreadsheet. For each such number, you want the spreadsheet to be recalculated. Finally after each recalculation, you want to transfer the value of a specific cell back to DynRisk.

As a “random number generator” we use a node which we call “X”. The number we get back from the spreadsheet, will be passed on to a node called “Y”.

We observe that there are three different messages we need to send to the spreadsheet:

- “Send X”
- “Recalculate”
- “Receive Y”

[If the spreadsheet is configured to do automatic recalculation, we could of course skip the second message, but for the purpose of the argument, we assume that this is not the case here. Indeed, in cases where you feed many values into a large complex spreadsheet, you may save a lot of time by switching off automatic recalculation, and then send a recalculate message after the last number has been transferred.]

To accomplish this, we will use three events, named “Send X”, “Recalculate” and “Receive Y” respectively. Together with the two nodes, “X” and “Y”, we then have a model with five objects.

Now, to get correct results, it is obvious that the three messages need to be sent in the following order: “Send X”, “Recalculate”, “Receive Y”. Moreover, before we can send the value of “X”, we need to calculate this value, and before we can calculate “Y”, we need to receive this value from the spreadsheet.

When DynRisk calculates a model, the model objects are always calculated in an order which respects the directions of the edges. If there is a directed path from one object to another, then the first object will always be calculated before the second.

This means that to get correct results in our case, we need to add edges to the model such that we get a directed path starting at “X”, passing through “Send X”, “Recalculate” and “Receive Y” in that order, and ending at “Y”.

Considering the categories of the three events, we see that “Send X” is a “Send data” event, “Recalculate” is a “Trigger action” event, and “Receive Y” is a “Receive data” event. This implies that we have the following data flow through our model:

- “X” -> “Send X” : The value of “X”
- “Send X” -> “Recalculate” : 0
- “Recalculate” -> “Receive Y” : 0

• “Receive Y” -> “Y” : The value of “Y”

We notice that the values passed along the two intermediate edges, are just dummy values. They are not used at all in the calculations. We still need these edges to ensure that everything happens in the right order.

Finally, note that we cannot skip “Y”. Why? Because DynRisk does not allow events to be stored on file. Thus, we have to pass the value over to a node.

In fact, if we skipped “Y” in the diagram, and then tried to run a simulation on the model, then only “X” would be simulated. No messages will ever be sent to the spreadsheet!

In order to determine which objects that need to be calculated during a simulation, DynRisk starts out with the objects which are selected to be stored on file. If “Y” is deleted from the model, only “X” could be stored. DynRisk then continues by following the edges backwards through the model, and includes all objects found along the way. In our case, “X” has no predecessors. Thus, without “Y”, only “X” will be included in the simulation.