Enterprise Architect

User Guide Series

Simulation and Behavior

Author: Sparx Systems & Stephen Maguire

Date: 16/01/2019

Version: 1.0

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Introduction

Enterprise Architect is a powerful platform that provides a unique set of developer tools integrated into a sophisticated development environment. In addition to the standard facilities available to work with static code, there is a wide range of facilities that support the simulation and automatic code generation from the models.

In an era dominated by digital disruption and the need for organizations to be agile and responsive to dynamic business changes, these facilities provide a powerful insurance policy that will ensure an organization becomes a true digital enterprise.

Enterprise Architect holds a unique position and, regardless of whether you use Enterprise Architect's powerful standard code engineering features or another Integrated Development Environment such as Eclipse or Visual Studio, these facilities provide productivity gains not matched in other tools.

Model Simulation brings your behavioral models to life with instant, real-time behavioral model execution. Coupled with tools to manage triggers, events, guards, effects, breakpoints and simulation variables, plus the ability to visually track execution at run-time, the Simulator is a powerful means of 'watching the wheels turn' and verifying the correctness of your behavioral models. With Simulation you can explore and test the dynamic behavior of models. In the Corporate, Unified and Ultimate editions, you can also use JavaScript as a run-time execution language for evaluating guards, effects and other script-able pieces of behavior.

With extensive support for triggers, trigger sets, nested states, concurrency, dynamic effects and other advanced simulation capabilities, Enterprise Architect provides a remarkable environment in which to build interactive and working models that help explore, test and visually trace complex business, software and system behavior. With JavaScript enabled, it is also possible to create embedded COM objects that will do the work of evaluating guards and executing effects - allowing the simulation to be tied into a much larger set of dependent processes. For example, a COM object evaluating a guard condition on a State Transition might query a locally running process, read and use a set of test data, or even connect to an SOA web service to obtain some current information.

As Enterprise Architect uses a dynamic, script driven Simulation mechanism there is no need to generate code or compile your model before running a simulation. It is even possible to update simulation variables in real time using the Simulation console window. This is useful for testing alternate branches and conditions 'on the fly', either at a set Simulation break point or when the Simulation reaches a point of stability (for example, when the Simulation is 'blocked').

In the Professional version of Enterprise Architect, you can manually walk through simulations - although no JavaScript will execute - so all choices are manual decisions. This is useful for testing the flow of a behavioral model and highlighting possible choices and processing paths.

Dynamic Model Simulation

Using the Model Simulator, you can simulate the execution of conceptual model designs containing behavior. When you start a simulation, the current model Package is analyzed and a dynamic simulation process spawned to execute the model. As the simulator analyzes and works with UML constructs directly, there is no requirement to generate intermediary code or compile simulation 'executables'. This results in a very rapid and dynamic simulation environment in which changes can be made and tested very rapidly.

To get up and running with simulation, the only steps required are:

- Build a behavioral diagram (State or Activity for manual or dynamic execution, Sequence for manual interaction only)

- Optional: load the 'Simulation Workspace' layout - a fast way of bringing up all the frequently used Simulation windows

- Click on the simulator Play button

If the diagram contains any external elements (those not in the same Package as the diagram) you will have to create an Import connector from the diagram's Package to the Package containing the external elements. You can do this by dragging both Packages from the Project Browser onto a diagram and using the Quicklinker arrow to create the connector between them.

The Professional edition provides a quick and simple way to verify your design's behavior for logical correctness by manually stepping through a diagram. In the Corporate edition and above it is possible to:

- Dynamically execute your behavioral models

- Assess guards and effects written in standard JavaScript

- Define and fire triggers into running simulations

- Define and use sets of triggers to simulate different event sequences

- Auto-fire trigger sets to simulate complex event histories without user intervention

- Update simulation variables 'on the fly' to change how simulations proceed

- Create and call COM objects during Simulation to extend the Simulation's reach and input/output possibilities

- Inspect Simulation variables at run time

- Set a script 'prologue' for defining variables, constants and functions prior to execution

- Use multiple Analyzer Scripts with differing 'prologues' for running the Simulation under widely differing conditions

In the Unified and Ultimate Editions it is also possible to simulate BPMN models.

Platforms and Editions Available

Platform/Edition	Details
Models and Platforms Supported	The Model Simulator currently supports the execution of UML Activity, Interaction and StateMachine models and BPMN Business Processes on the simulation platforms: • UML Basic • BPMN
Edition Support	Model Simulation is available at different levels across the range of editions of Enterprise Architect: • Professional - Manual Simulation only • Corporate and above - Adds dynamic JavaScript evaluation

	• Unified and Ultimate - Adds BPMN simulation

How it Looks

Enterprise Architect has a special way of displaying model information during simulation. This helps focus attention on the executing or active nodes.

During a Simulation, Enterprise Architect will dynamically track and highlight the active nodes within your model. If a node in another diagram is activated, that diagram will be automatically loaded and the current node highlighted. It is possible to modify the diagram while the simulation is running; however, the changes made are not recognized until the current simulation is ended and a new one begun.

Highlighting of the active node (s) during simulation

- In the example here, the currently active node (VehiclesGreen) is highlighted in normal Enterprise Architect colors
- All possible transitions out of the current node are rendered at full strength
- The elements that are possible targets of the current active node's outgoing transitions are rendered in a semi-faded style so they are readable and clearly different to the other elements within the diagram
- All other elements are rendered in a fully faded style to show they are not targets of the next Simulation step

As the Simulation progresses (especially if automatically run), this highlighting helps focus the attention on the current item and its visual context.
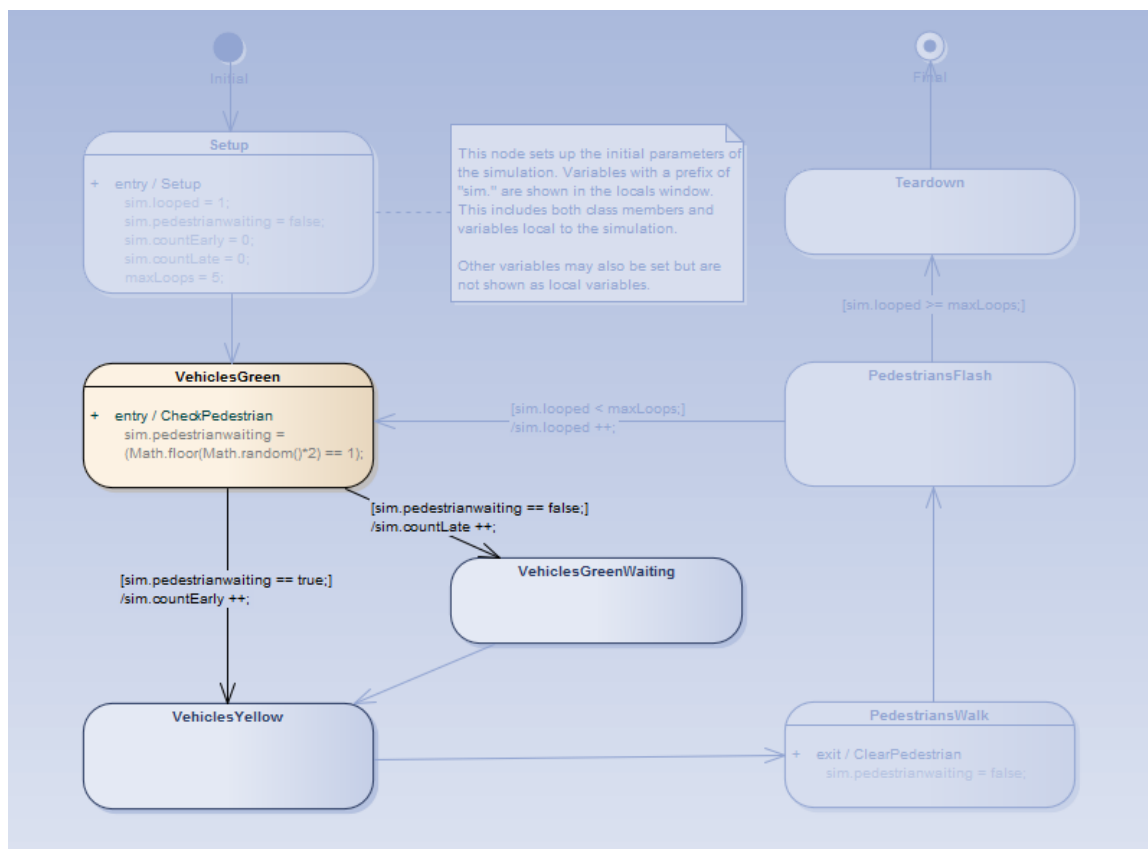
Simulation Windows
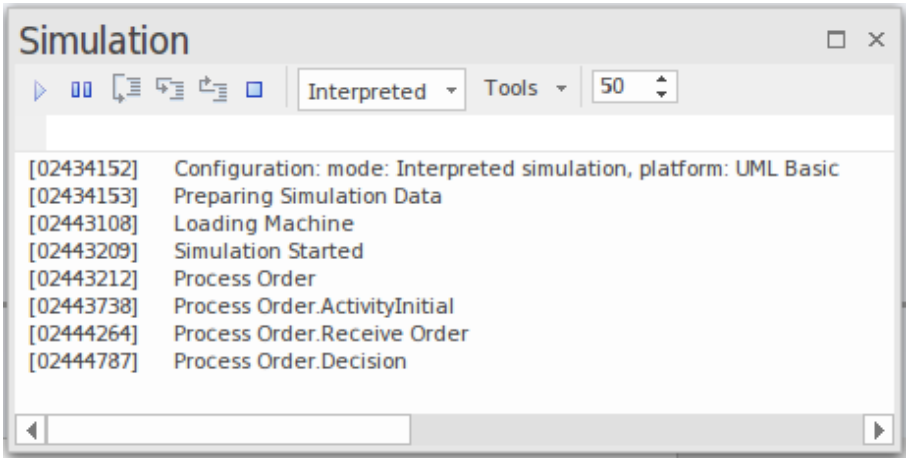
When executing a Simulation in Enterprise Architect it is possible to set break-points, fire triggers, examine variables, record a trace of execution, set simulation speed, view the call stack and visually trace the active nodes as the simulation proceeds.
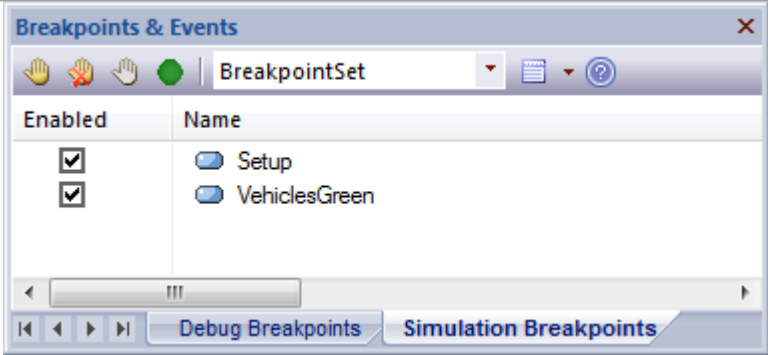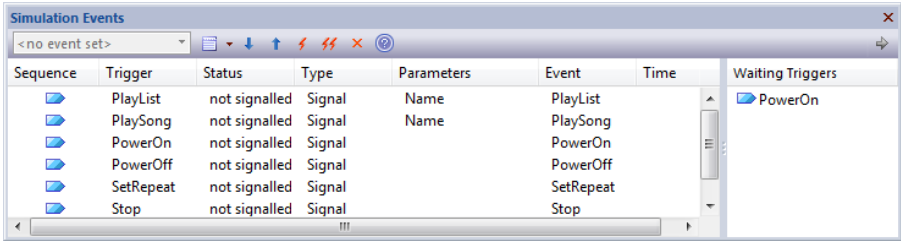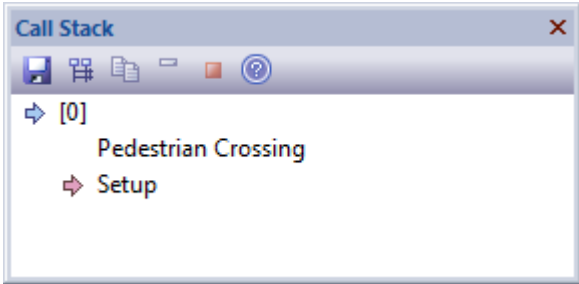
When a Simulation runs, some aspects such as the output and console input are found in the Simulator window itself, while others such as the local variables and call-stack use the standard Execution Analyzer windows. The table provides an overview of the main Windows used during Simulation.
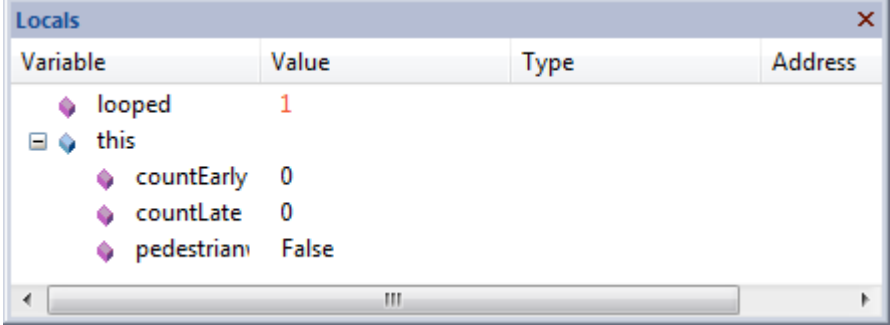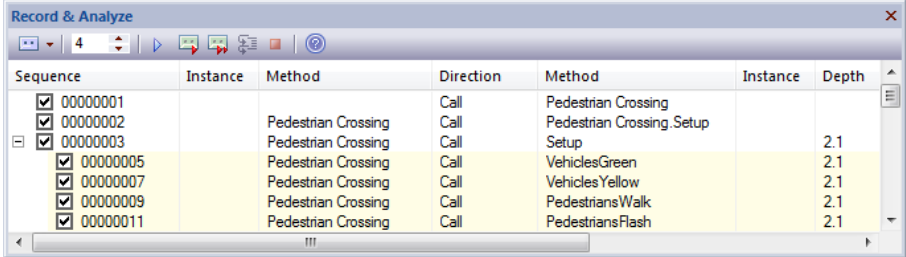
Access

Ribbon	Simulate > Dynamic Simulation > Simulator > Open Simulation Window

Windows

Window	Purpose
Execution and Console	The Simulation window provides the main interface for starting, stopping and stepping your Simulation. During execution it displays output relating to the currently executing step and other important information. See the *Run Model Simulation* topic for more information on the toolbar commands.
	Note the text entry box just underneath the toolbar. This is the Console input area - here you can type simple javascript commands such as: *this.count = 4;* to dynamically change a Simulation variable named "count" to 4. In this way you can dynamically influence simulation at run-time.
Breakpoints & Events Window	The Simulation process also makes use of the 'Simulation Breakpoints' tab of the Breakpoints & Markers window ('Simulate > Dynamic Simulation > Breakpoints'). Here you set execution breakpoints on specific elements and messages in a Simulation. See the *Simulation Breakpoints* topic for more details.

Simulation Events Window	The Simulation Events window ('Simulate > Dynamic Simulation > Triggers') provides tools to manage and execute triggers. Triggers are used to control the execution of StateMachine transitions.
Call Stack Window	During the Simulation the Call Stack window ('Simulate > Dynamic Simulation > Call Stack') displays information about the Threads and current execution context of the Simulation.
	The Simulator supports multi-threaded Simulations and will include a Thread entry for every active and paused thread of execution. For each thread, the Call Stack window will show the start or entry context (such as a StateMachine element) plus the current active element within that thread. If the current active element is the entry point of a composite activity or sub-machine state, the stack will also include the current active element within that sub-context (and all further nested, active composite, sub-states as well).
Simulation Local Variable Window	The Simulator uses the standard Locals window ('Simulate > Dynamic Simulation > Local Variables') to show all current Simulation variables when the simulation is single stepping or paused at a break point. Note that it is possible to dynamically update these variables using the Simulator Console.

	Locals
	Variable: looped, Value 1; this; countEarly 0; countLate 0; pedestrian False
Recording	During execution of your Simulation, a recording is kept of all activity and displayed in the Record & Analyze window ('Execute > Analyze > Recorder > Open Recorder'). This is similar to how the normal call recording works in the Visual Execution analyzer - although at this time, no ability to generate Sequence diagrams from recordings is possible.

Set Up Simulation Script

You can use Simulation scripts to provide fine control over how a simulation starts. In general, you do not need to set up a Simulation script unless:

- You want to run an interpreted simulation that requires variables to be initialized before the simulation commences; this is useful for setting up global variables and defining functions

- (In the Corporate Edition and above) You do not want to apply the default behavior of interpreting the Guards (that is, you prefer to use a Manual execution), or

- You want to have multiple ways of running the same diagram

For most diagrams it is possible to initialize a script for a simulation simply by setting variables in the first element or connector after the Start element. For State Charts, this is the Transit connector exiting the initial element, and for Activity models this is the first Action element.

As an alternative, you can use Simulation scripts to initialize settings before a simulation starts. This is useful for setting up different sets of initial values using multiple Analyzer Scripts, so that you can run your simulation under a range of pre-set conditions.

To configure a simulation execution script, first select the Package in the Project Browser, Package Browser, Diagram List or Model Search. You can then use the Execution Analyzer window to add a new Script for that selected Package. You will use the 'Simulation' page of the 'Execution Analyzer' dialog to configure the relevant properties.

Access

Show the Execution Analyzer window using one of the methods outlined here.

On the Execution Analyzer window, either:

- Locate and double-click on the required script and select the 'Simulation' page or

- Click on ![icon] in the window Toolbar and select the 'Simulation' page

Ribbon	Code > Configure > Analyzer > Edit Analyzer Scripts
	Execute > Run > Analyzer > select and run script
Context Menu	Project Browser | Right-click on Package | Execution Analyzer
Keyboard Shortcuts	Shift+F12

Configure a Simulation Script

Option	Action
Platform	For UML Activity, Interaction or StateMachine simulation, click on the drop-down arrow and select 'UML Basic'.
	For BPMN diagrams, click on the drop-down arrow and select 'BPMN'.
Entry Point	Click on the [...] button and select the:
	- Entry point for the simulation, and
	- Activity, Interaction or StateMachine to simulate

	If you do not specify an entry point, the simulator attempts to work through the entire Package.
Evaluate Guards and Effects using JavaScript	(In Corporate and higher editions) Leave the checkbox unselected to perform a manual simulation, where you select the next State to transition to and the point where a decision must be made.
	Select the checkbox to execute the code for Effect behavior in the simulation. The simulation executes JavaScript code in these places:
	• State entry/exit/do operations
	• Transition guard/effect
	• BPMN Activity Loop Conditions and Sequence Flow Condition Expressions
	With the exception of the guard, all of these should be one or more valid JavaScript statements, including the semi-colon.
	The guard must be a valid boolean expression, also terminated with a semi-colon.
	Variables that are members of 'sim' or 'this' are listed in the Locals window when a simulation breakpoint is reached.
	sim.count = 0;
Input	When JavaScript is enabled, you can type script commands in this field that will execute prior to the simulation being run.
Post Processing Script	Using a Post Simulation Script, you can run JavaScript after the simulation ends. Type in the qualified name of a script from the model script control.
	For example, if you have a script named 'MyScript' in the Script Group 'MyGroup', type in the value 'MyGroup.MyScript'.
OK	Click on this button to save your changes.

Notes

- Usually all simulation elements and relationships reside within the Package configured for simulation; however, you can simulate diagrams that include elements from different Packages, by creating Package Import connectors from the configured Package to each 'external' Package (or alternatively, for a BPSim model, a Dependency connector from the configured Package to each external **element**)
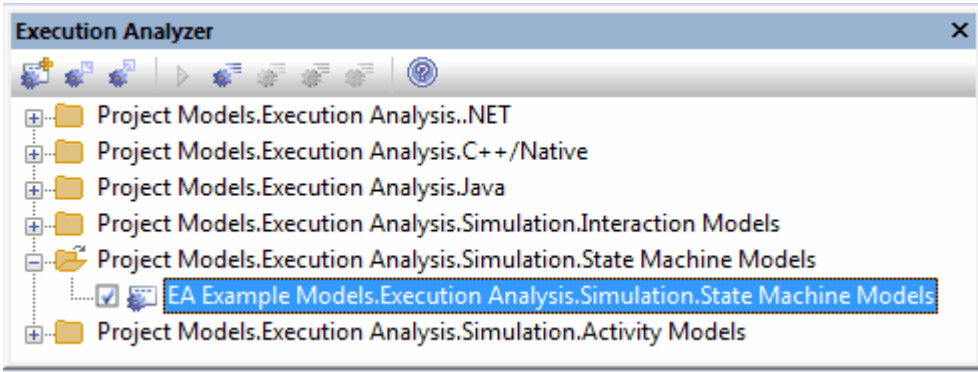
Activate Simulation Script

An Execution Script is configured for a model Package defining the simulation parameters. The most common reason for activating an execution script is when multiple simulation scripts are configured against a Package and you want to run a specific one.

Access

Ribbon	Code > Configure > Analyzer > Edit Analyzer Scripts Execute > Run > Analyzer > select and run script
Context Menu	Right-click on Package | Execution Analyzer
Keyboard Shortcuts	Shift+F12

Activate a Simulation script for execution

Step	Action
1	In the Execution Analyzer window, select the required execution script. This makes it the current default for your open model, so that clicking on the Simulation Run button will automatically invoke this Simulation script.
2	Click on the checkbox to the left of the script to activate it.
3	Select the 'Simulate > Simulator > Open Simulation Window' ribbon option to execute the simulation.

Run Model Simulation

A simulation executes the model step-by-step, enabling you to validate the logic of your behavioral model. The current execution step is automatically highlighted in the model's diagram to make it easy to understand the various processes and state changes as they occur during the simulation.

There are several ways to start a model simulation:

- When the active diagram can be simulated, the Run button on the main Simulation window will process the current diagram, either by running an existing script or defining a new temporary one

- When the active diagram can not be simulated, the Run button on the main Simulation window will run the simulation for the active Execution Analyzer script

- By right-clicking on a Simulation script in the Execution Analyzer window and selecting the 'Start Simulation' option

- By right-clicking on a suitable diagram and selecting one of the 'Execute Simulation' options

There are visual cues during execution. When the simulation is running, Enterprise Architect will actively highlight each active node for each executed step. In addition, all outgoing transitions and control flows will be highlighted, showing the possible paths forward. Elements at the end of possible paths forward will be de-emphasized to half-strength and any other remaining elements will be 90% 'grayed out'. This provides a very dynamic and easy to follow execution that continually refocuses attention on the execution context.

Access

Ribbon	Simulate > Dynamic Simulation > Open Simulation Window

Edition Specific Details

In the Professional edition, if a branch is encountered in the execution, the simulator prompts you to choose the appropriate path to take in your execution.

In the Corporate, Unified and Ultimate editions, in which JavaScript is enabled, the Simulation will automatically evaluate all guards and effects and dynamically execute the Simulation without user intervention. If the Simulation becomes blocked due to no possible paths forward evaluating to True (or multiple paths evaluating to True) you can modify Simulation variables on the fly using the console input of the Simulation Execution window.

Run a Simulation Using the Toolbar

Icon	Action
▷	Start the simulator for the current diagram or, if the current diagram cannot be simulated, run simulation using the activated simulation script.
II	Pause the simulation.
🗐 🗐 🗐	When the simulation is paused, step over, step in and step out to control the simulator's execution at the required step in the model simulation.
■	Stop the simulation.
Manual ▾ Manual Interpreted Executable	Click on the drop-down arrow and select the type of simulation to run: • 'Interpreted' - Perform dynamic execution of a simulation (Corporate and Extended Editions) • 'Manual' - Step through a simulation manually (the only option available in the Professional edition) • 'Executable' - Select when running the simulation on an Executable StateMachine
Tools ▾	Click on the drop-down arrow and select from a menu of options for performing specific operations on the simulation script and output, such as Build, Run, Generate and View Breakpoints.
60 ▴▾	Vary the execution rate of the simulation, between 0% and 100%; at: • 100%, the simulation executes at the fastest possible rate • 0% the simulator breaks execution at every statement

Notes

- The Simulation tool only becomes active when a valid simulation Execution Script is activated
- You can set a Simulation script as the current default by setting its checkbox in the Execution Analyzer window

Simulation Breakpoints

The 'Simulation Breakpoints' tab of the Breakpoints & Events window enables you to interrupt and inspect the simulation process.

When dynamically executing a simulation (in the Corporate and above editions) the process will proceed automatically - if you want to stop execution at some point to examine variables, inspect call stacks or otherwise interact with the simulator, you can set a breakpoint on a model element in much the same way as you would with a line of source code. When the simulator reaches the breakpoint execution is halted and control returned to Enterprise Architect.

Access

Ribbon	Simulate > Dynamic Simulation > Breakpoints > Simulation Breakpoints

Breakpoints

The simulation executes the model step-by-step, enabling you to validate the logic of your behavior model; the simulation halts when it reaches an element defined as a breakpoint.

The UML elements that can be defined as breakpoints include: Actions, Activities, States and most other behavioral nodes (such as decision, initial, or final).
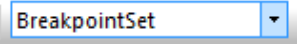
The UML relationships that can be defined as breakpoints include Interaction Messages.

The breakpoints are stored as Breakpoint Sets for a given Enterprise Architect project.

Elements that are included in a simulation and that have breakpoints are marked by a green circle off the top left corner of the element, whilst the simulation is in progress. If the simulation is not running, the green circles are not displayed.

When JavaScript is enabled, all Simulation variables will be displayed in the Locals window - and it is possible to modify Simulation variables using the Simulation window's console input field (underneath the Toolbar).

Toolbar Buttons

Item	Description
	Enables all breakpoints defined in the current Breakpoint Set for the simulation session.
	Deletes all breakpoints defined in the current Breakpoint Set for the simulation session.
	Disables all breakpoints defined in the current Breakpoint Set for the simulation session.
	Adds a breakpoint for the selected element or Sequence message to the current Breakpoint Set.
BreakpointSet	Changes the selected **Breakpoint Set** for use in the simulation session.

	Performs Breakpoint Set commands:
	• **New Set**: Create a new Breakpoint Set
	• **Save As Set**: Saves the current Breakpoint Set under a new name
	• **Delete Selected Set**: Deletes the current Breakpoint Set
	• **Delete All Sets**: Deletes all Breakpoint Sets saved for the diagram

Objects and Instances in Simulation

As a given business, system or mechanical process executes, the Activities and Actions within it might generate objects of a specific type and perform operations on those objects, perhaps even consuming or destroying them. You can simulate the creation, use and consumption of such objects using a simulation model that represents the objects and actions with model elements such as Classes, Instance Objects, attributes, operations and Ports (ActionPins and ObjectNodes). The model can also create, act on and destroy several different objects at different stages as part of the same process. Representing model data or objects in simulation makes the simulation more accurately reflect the real process.

Object Concepts

Term	Description
SimType	The type of simulation element, such as Class, Enumeration or Interface. These can be classifiers of objects in a simulation.
SimObject	An object that is an instance of (is classified by) a SimType element.
Attribute	A property of a SimType element, or of a specified node such as an ActivityNode.
Operation	A behavior of a SimType element, or of a specified node such as an ActivityNode.
Port	A Port of a Class or Object, an ActionPin of an Action, or an ObjectNode of an Activity. Ports of classifiers are a type, whilst a Port of an object is a realization of the type.
Parameter/ Activity Parameter	Parameters of Operations; Activity Parameters are, specifically, parameters of ActivityNodes.
Slot	A realization of an attribute in an object. A Slot has a run time value that can be initialized by the run state value of the Slot. If these values don't exist, the system uses the initial values of the attributes.
Runtime Environment	All objects exist in the JavaScript runtime environment, so you can use JavaScript to create or change simulation objects and simulation variables.
Display Variables	All simulation objects, simulation variables or events are identified on the Locals window while they are in effect. In some cases, to show the variables you might need to add break points to the model to pause processing while the variable exists. As all objects and variables are shown, global variables that exist outside the simulation but that are significant to it - such as the parent Class and Activity elements within which a process is defined - are automatically also represented as default object variables. So too is the anticipated output of the Activity, as a return variable.

Create Objects in a Simulation

In a simulation model, you can create Classes and either create instances of them (Global Objects) to represent objects that exist in the process, or define Actions to generate one or more Objects at any point during the process.

You have three options for creating Objects in a simulation model:

- Manually create the Object
- Dynamically create an Object through a CreateObject Action element
- Use the JavaScript function sim.CreateObject ("name") as the 'Effect' of an Action element, to again create an Object dynamically

Having created an Object dynamically you can also instantiate any inner objects of that Object, such as an Activity on a Class, and act on the properties of that inner object.

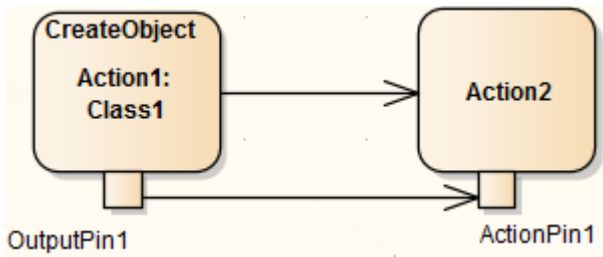## Manually Create an Object

Simply create an Object element on a diagram in the model, either by:

- Dragging an Object element from the 'Object' pages of the Diagram Toolbox and setting its classifier, or
- Dragging a classifier element from the Project Browser and pasting it into the diagram as an instance

In the simulation model you can then set up the Object properties themselves (such as setting run-states to re-set the initial value of an attribute) or the behaviors of Actions to act on the Object (such as passing it along a process flow) and observe what happens to the Object in a simulation.

Create an Object through a CreateObject Action

If your process generates objects in runtime, you can simulate this using a CreateObject Action.

Step	Action
1	On your Activity diagram, drag an 'Action' icon from the Diagram Toolbox, and select the 'Other | CreateObject' context menu option to define it as a CreateObject Action element.
2	Set the classifier of the CreateObject Action to the Class of which the Object will be an instance. (Advanced | Set Classifier). Create an Action Pin on the CreateObject Action, of kind output.
3	Create or select the next Action in the processing sequence, and add an Action Pin of kind input. Connect the two Actions with a Control Flow connector, and the two Action Pins with an Object Flow connector.

4	Perform a Simulation on the diagram. When the CreateObject Action is executed, it creates an Object having the properties of the classifier, and stores it in its Output Pin. The Object itself is passed through the Object Flow connection to the Input Pin of Action 2, where its properties can be listed in the Locals window for the simulation.

Create Object Using JavaScript

You also can create simulation objects dynamically using a JavaScript command in the 'Effect' field of the Action element. The command is:

    sim.newObject = sim.CreateObject("ClassName");

or

    sim.newObject = new SimObject("ClassName"); (natural JavaScript)

That is: 'Simulate the creation of an Object based on Class <name>'. The classifying Class would exist in the same Package as the Action.

As for the CreateObject Action element, the Object is created during the simulation and can be passed down to and processed by 'downstream' elements. In this example, the created Object is identified as sim.object1 and in Action 2 it is accessed and one of its attributes given a different value (also by JavaScript as an Effect of the Action).
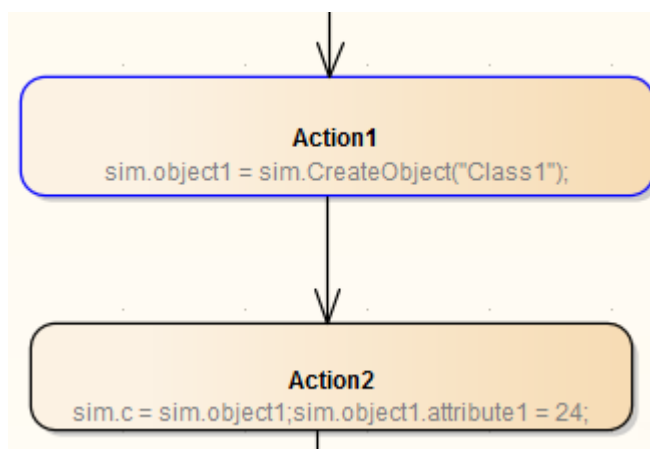
Instantiate Inner Objects

As described earlier, you can create an Object using either JavaScript or a CreateObject Action. Similarly, you can instantiate inner objects using JavaScript or a CallBehavior Action.

In this example, using JavaScript, the simulation first creates a test object based on Class1. Class 1 has an Activity element and diagram, with an Activity Parameter 1 set to the integer 5 and an Activity Parameter 2 set to the string 'test'. The value of Activity Parameter 1 is captured as a buffer value 'buf'.
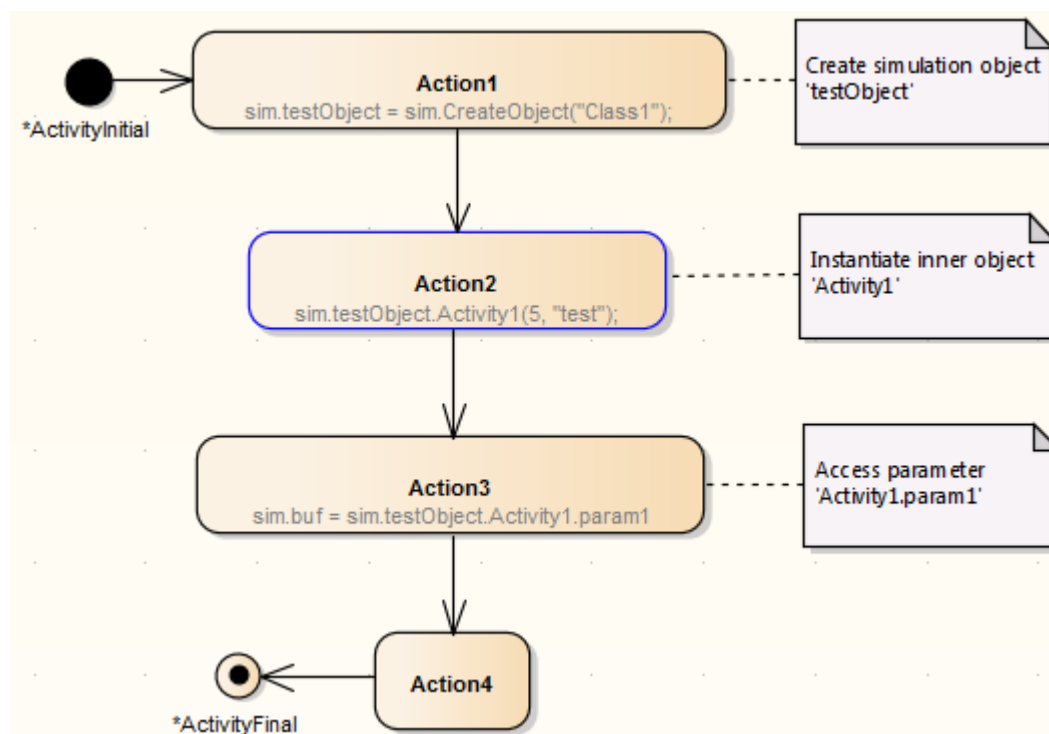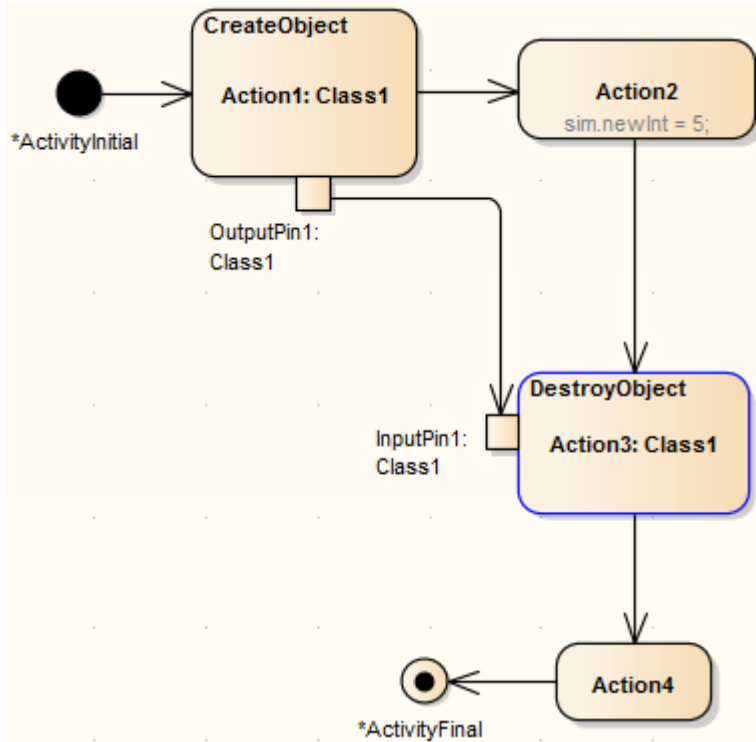
Destroy Objects in a Simulation

Having created or generated Objects in your simulation model, you can define Actions to destroy those objects at any point during the process. All simulation objects are destroyed automatically when the simulation completes.

You have two options for destroying the Objects in your simulation model:

- Dynamically destroy the Objects through a DestroyObject Action element
- Dynamically destroy the Objects using JavaScript in an Action element

The result of the deletion can be observed in the change of local variables, on the Local window.

Destroy an Object through a DestroyObject Action

Step	Action
1	On your Activity diagram, drag an 'Action' icon from the Diagram Toolbox, and select the 'Other | DestroyObject' context menu option to define it as a DestroyObject Action element.
2	Set the classifier of the DestroyObject Action to the Class of which the Object is an instance. (Advanced | Set Classifier). Create an Action Pin on the DestroyObject Action, of kind *input*.
3	Connect the Input Action Pin to an Object Flow connector from the last Action that operated on the Object. In this example, the last Action that operated on the Object is the Action that created it.
4	Perform a Simulation on the diagram. The process passes the Object name or value into the Input Action Pin as a parameter. When the DestroyObject Action is executed, it deletes the Object having that name or value from the model. In the example, the instance of Class1 is specifically destroyed before Action4 is processed, but the results

	of Action2 are unaffected.

Destroy an Object using JavaScript
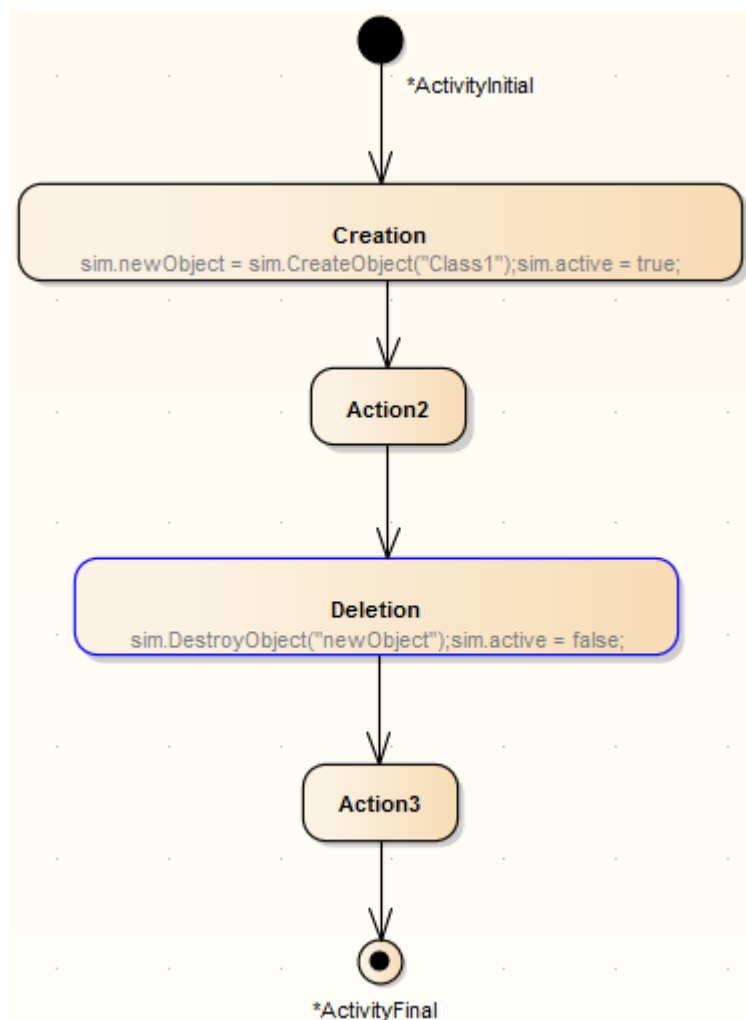
In the 'Properties' dialog of the Action element, in the 'Effect' field on the 'Effect' page, type either:

   sim.DestroyObject ("objectname")

or

   delete sim.objectFullName

For example:

Notes

- In either case, you can also destroy a global object (one that is created outside the process flow) by identifying the Object to the Action performing the destruction; in the case of the DestroyObject Action, by passing the Object name from a Port on the Object to the Input Pin on the Action through an Object Flow connector

Dynamic Simulation with JavaScript

The Corporate and suite editions of Enterprise Architect have the capability of using JavaScript to evaluate guards, effects and other aspects of behavior within the Simulation context. This provides for a fully automated, intelligent execution of your State or Activity model, with fine control over breakpoints, execution speed and simulation variables.

JavaScript can be written that uses any variables. To enable you to display the variable values to the user interface, two objects are defined that have their members shown in the Local Variables window. These are sim and this; for example:

- sim.logger
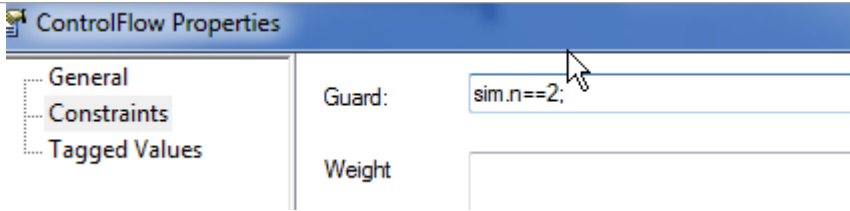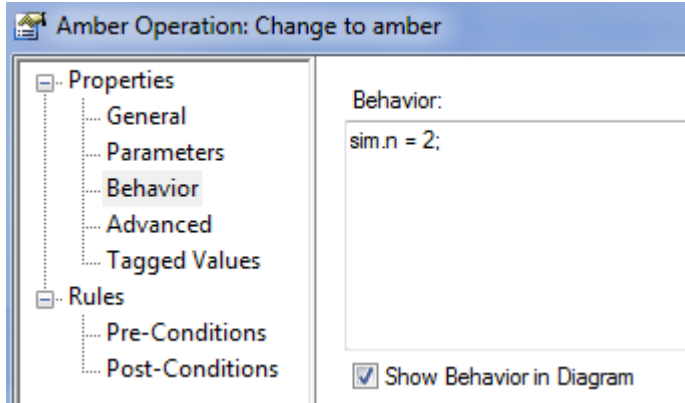- sim.Customer.name
- this.count
- this.Account.amount

All these variables will be shown in the Locals window.

The recommended convention is to add any global or control variables not declared in the owning Class to the sim object. In contrast it would be normal to add attributes of the owning classifier to the this object.

Some examples of where and how you can set Simulation behavior using JavaScript are shown here. See the EAExample.eap model that comes with Enterprise Architect for further examples. Also see the *Learning Center* for further information on setting up and working with JavaScript in Simulations.

Using JavaScript

Setting	Action
Analyzer Script Input	If you enter JavaScript code into the Execution Analyzer window 'Input' field, this code will be injected into the Simulation and executed before the Simulation starts. This is useful for establishing COM variables, global counters, functions and other initialization. Platform: UML Basic Options: ☑ Evaluate Guards and Effects using JavaScript Input sim.n=1;
Transition and Control Flow Guards	This is the workhorse of the Simulation functionality. As Enterprise Architect evaluates possible paths forward at each node in a Simulation, it tests the Guards on outgoing transitions and control flows and will only move forward if there is a single true path to follow - otherwise the Simulation is considered 'blocked' and manual intervention is required. You must use the '==' operator to test for equality.

Element Behavior	Entry and Exit behavior can be defined for States. Such code will execute at the appropriate time and is useful for updating Simulation variables and making other assignments. You can also simulate the behavior of Classes via their Object Instances, and Activities in your model.

Using COM	One very important feature of the implementation of JavaScript in Enterprise Architect's simulator is that it supports the creation of COM objects. This provides the ability to connect the running Simulation with almost any other local or remote process and either influence the Simulation based on external data, or potentially change data or behavior in the external world based on the current Simulation state (for example, update a mechanical model or software simulation external to Enterprise Architect). The syntax for creating COM objects is shown here:
	
	`this.name="Odd Even";`
	
	`var logger = new COMObject("MySim.Logger");`
	
	`logger.Show();`
	
	`logger.Log("Simulation started");`

Signalled Actions	It is possible to raise a signalled event (trigger) directly using JavaScript. The BroadcastSignal() command is used to raise a named trigger that could influence the current state of a simulation. For example, this fragment raises the signal (trigger) named "CancelPressed".
	
	`BroadcastSignal("CancelPressed");`
	
	Note that a trigger named CancelPressed must exist within the current simulation environment and must uniquely have that name.
	
	You can also call the signal using its GUID. For example:
	
	`BroadcastSignal("{996EAF52-6843-41f7-8966-BCAA0ABEC41F}");`

IS_IN()	The IS_IN(state) operator returns True if the current simulation has an active state in any thread matching the passed in name. For example, to conditionally control execution it is possible to write code such as this:

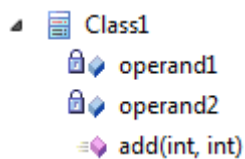|	
	if (IS_IN("WaitingForInput")) BroadcastSignal("CancelPressed") Note that the name must be unique within all contexts.
Trace()	The Trace(statement) method allows you to print out trace statements at any arbitrary point in your simulation. This is an excellent means of supplementing the Simulation log with additional output information during execution.

Call Behaviors

In the course of simulating a process, you can enact the behaviors defined in an operation of either a Class (through its simulation Object) or an Activity in the model. In each case, you use JavaScript to call the behavior.

Invoke the Behavior of a Class

A Class in your model defines a behavior that you want to simulate. This behavior is defined in the Behavior page of an Operation of the Class.

For example, the Class is intended to add two integers, through the Operation add. The integers in this case are parameters of the operation, defined by attributes of the Class, operand1 and operand2.

▲ ▤ Class1
    🔒◆ operand1
    🔒◆ operand2
    ≡◆ add(int, int)

Step	Action

1	In the Properties window for the operation, select the 'Behavior' tab and edit the 'Behavior' field to apply the JavaScript simulation objects (*this* or *sim*) to the behavior definition.
	
	In the example:
	
	    this.operand1=operand1;
	
	    this.operand2=operand2;
	
	    return operand1+operand2

2	Drag the Class onto your simulation Activity diagram and paste it as an Instance.
	
	In the example, the Object is called 'calculator'. For clarity, the element shown here is set to display inherited attributes and operations, and the behavior code, on the diagram.
	
	calculator: Class1
	::Class1
	- operand1: int
	- operand2: int
	::Class1
	+ add(int, int): int
	     this.operand1= operand1;
	     this.operand2 = operand2;
	     return operand1+ operand2;

3	On the simulation diagram, for the appropriate Action element, open the 'Properties' dialog and on the 'Effect' page type in the JavaScript to capture and simulate the Object's behavior. In the example, the JavaScript defines a value that will be provided by simulating the behavior of the operation from the Object, as performed on two provided integers. That is: sim.result=sim.calculator.add(7,9)
4	Run the simulation, and observe its progress in the Locals window. Ultimately the Class behavior is reflected in the result of the simulation.

	In the example: result = 16.

Invoke the Behavior of an Activity

An Activity element can have a behavior, defined by an operation in that element. As a simple example, an Activity might have an operation called Get Result, with the behavior return "ON";.

You can simulate this behavior in the Activity's child diagram (that is, internal to the Activity), with a JavaScript statement in the appropriate Action element's 'Effect' field. In the example, this might be:
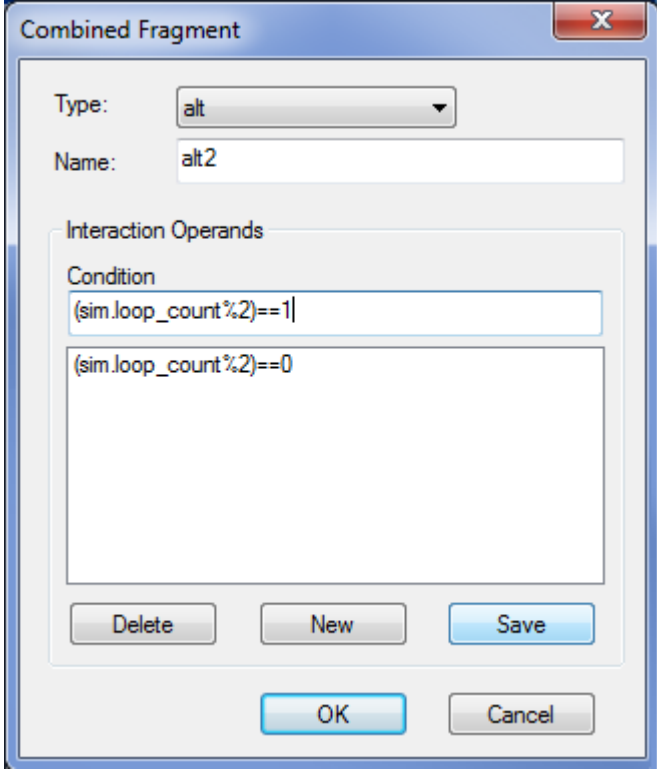
    sim.result=this.GetResult();

The statement invokes the parent Activity's operation GetResult and assigns the outcome of that operation's behavior to sim.result. You can observe the progress of the simulation and the result of simulating the behavior in the Locals window, where (in this example) the value result "ON" will ultimately display.

Interaction Operand Condition and Message Behavior

When you simulate the behavior of a Sequence diagram, you can use a Condition for the CombinedFragment Interaction Operand, to control the flow during the course of the simulation.

A Message in Sequence diagram can link to an Operation, so the behavior of the Operation can also be used during the course of the simulation.

Interaction Operand Conditions

Field/Column	Description
Operand Condition	Interaction Operand Conditions are conditional statements that are evaluated whenever the simulator has to determine which path to take next. Operand Conditions typically have these characteristics: - Defined in the 'Combined Fragment' dialog - Written in JavaScript - Can refer to variables defined during simulation
Adding Operand Conditions	To add an Operand Condition: 1. Double-click on the CombinedFragment element to open the 'Combined Fragment' dialog. 2. Click on the New button. 3. In the 'Condition field', type the JavaScript for the condition. 4. Click on the Save button.
	During execution the Simulator evaluates any Operand Condition within the

Evaluation Semantics	CombinedFragment; the CombinedFragment type and the outcome of the evaluation can determine the path that the simulation continues on.

Guards and Effects

Guards and Effects are used to control the flow of the simulation and to execute additional actions or effects during the course of a simulation.

Guards and Effects

Concept	Detail
Guards	Guards are conditional statements that are evaluated whenever the simulator has to determine the path to take next. Guards typically have these characteristics: • Defined on transitions and control flows to govern how simulation proceeds • Written in JavaScript • Can refer to variables defined during simulation
Adding Guards	Guards are defined on the Transition or Control Flow in the 'Properties' dialog for the selected connector. A Guard is typically a piece of JavaScript that will evaluate to either True or False. For example, here is a conditional statement that refers to a current variable (Balance) being greater than zero. Note the use of the prefix *this* to indicate that the variable is a member of the current Class context.
Evaluation Semantics	During execution the Simulator will examine all possible paths forward and evaluate any guard conditions. This evaluation could establish that: • A single valid path forward evaluates to True; the Simulator will follow that path • Two valid paths exist; the Simulator will block, waiting for some manual input via the console window to resolve the deadlock • No valid path exists; the same response as when two paths are found - the Simulator waits for the user to change the execution context using the console window • No paths evaluate to True but a default (unguarded path) exists; the Simulator will take the unguarded path
Effects	Effects are defined behaviors that are executed at special times: • On entry to a state • On exit from a state • When transitioning from one state to another (transition effect) Effects can either be a section of JavaScript code or a call to another Behavior element in the current simulation.
JavaScript Effects	A JavaScript effect might resemble this example, in which the Balance variable is decremented:

	Guard: [_____] ☐ Effect is a Behavior Effect: this.Balance--;
Call Behavior Effects	In this example the effect is a call behavior effect. In this case, it calls into a model the Activity named Decrement Balance that is defined elsewhere. The simulation will then enter into that diagram/behavior and continue to execute until returning to the point at which the effect was invoked. Guard: [_____] ☑ Effect is a Behavior Effect: Decrement Balance
Order of Execution of Effects	In complex simulations that might involve transitioning out of deeply nested states into other deeply nested states in a different parent context, it is important to consider these rules concerning the order of execution: • All exit actions (effects) encountered leaving a nested context are executed in order of most deeply nested to least deeply nested • All actions (effects) defined on transitions are executed next • Finally, all entry effects are executed from the least deeply nested context to the most deeply nested So the basic rule is: all exit actions, followed by all transition actions, and finally all entry actions.
Note on JavaScript Variables	JavaScript variables to be accessed and referred to during Simulation execution belong to either: • sim (for example, sim.pedestrianwaiting) - typically used for global simulation variables, or • this (for example, this.CustomerNumber) - typically used to refer to owning Class attributes This is important to let the JavaScript engine know you are referring to a Simulation variable and not a simple local variable used during, for example, basic calculations. You can create Simulation variables of arbitrary scope and depth - for example, this.customer.name is a legitimate qualified name.

Triggers

Triggers represent signals and events that can activate transitions leaving the current state(s). A trigger might represent a real world signal or event such as:

- A button being pressed
- A message being received
- A pedal being depressed
- A switch being thrown
- A state in a concurrent region being entered or exited

For a trigger to have an effect

- Transitions have to be defined that will fire when the simulation receives the signal / event
- The current simulation state(s) or its parent(s) must have an outgoing transition that accepts that trigger
- The transition activated must be unguarded or have a guard that will evaluate to true

Managing Triggers

Action	Detail
Creating Triggers	Triggers are either created as an instance of a Signal element or as an anonymous event. Triggers are connected to Transitions in the 'Transition Properties' dialog as shown here. In this example a Trigger named 'Pushdown' has been defined based on the Signal 'Signal_Pushdown'. • Omitting the Type and Specification details results in a simple anonymous Trigger. • If parameters are needed, these are defined on the Signal and must be supplied at the time the event fires A trigger will appear in the Project Browser as illustrated here:
Using Triggers	Triggers are deployed by connecting them to transitions, as in the earlier example,

and are used during simulation by 'firing' them into the running simulation as required.

When using triggers these points should be taken into account:

- A 'triggered' transition can not take place until its effective trigger is signalled or fired

- When a trigger is received it will activate all current waiting transitions dependent on that trigger (that is, the trigger is broadcast)

- Triggers are evaluated on all transitions for all parents of a current child state; this allows a parent state to exit all child states if necessary

- Once used in a simulation, a trigger is consumed and must be re-fired if needed again

- Sets of triggers can be saved and either manually or automatically fired to facilitate automated model simulation under different event models

Firing Triggers	Firing triggers means to signal or activate a trigger within the current simulation. This could activate zero, one or many waiting transitions depending on the state and concurrency of the current simulation.

Firing triggers can be achieved in many ways. The most efficient is the 'Waiting Triggers' list.

During the course of model simulation, if the simulator reaches an impasse due to required triggers not being available (fired), the list of all possible candidate triggers is shown in the 'Waiting Triggers' list of the Simulation Events window.

Double-clicking a trigger in this list will fire it into the simulation. Other ways to fire a trigger include:

1. Double-click an un-signalled trigger in the Events window.

You can also use the context menu on these events to either signal an un-signalled event, or to re-signal an event that has already been fired previously.

2. Use the context menu of the Transition required to fire and select the 'Signal Trigger in Simulation' menu option.

Action Behavior By Type

You can vary the behavior initiated by an Action element by defining (or even redefining) its type. In simulation, you can apply and observe a number of different behaviors using the Actions in the types and groups described in this table.

Action Types

Type	Description
Object Actions	Object Actions operate on an object in a specific way, such as creating, destroying or reading the object. They include: • CreateObject • DestroyObject and • Read Self
Variable Actions	Variable Actions have an association variable in the form of the Tagged Value variable with the value of the name of an object in run-time. They provide the variable not only as an object but also as a property (such as an attribute or Port) of an object. They include: • ReadVariable • WriteVariable • ClearVariable • AddVariableValue • RemoveVariable
StructuralFeature Actions	StructuralFeature Actions operate on a structural feature, namely an attribute of an Activity or of the classifier of an object. They include: • ReadStructuralFeature • WriteStructuralFeature • ClearStructuralFeature • AddStructuralFeatureValue • RemoveStructuralFeatureValue
Invocation and Accept Event Actions	Invocation and Accept Event actions define the Triggers and Signals of an event. They include: • SendSignal • BroadcastSignal • AcceptEvent • SendObject • CallBehavior • CallOperation • AcceptCall
Miscellaneous Actions	The ValueSpecificationAction evaluates a value; it must have an input value and some evaluating code as its behavior or effect.

Structured Activity Simulation

One of the more complex structures in a behavioral model is a Structured Activity, which models a series of actions either in a nested structure or in a process of assessment and execution. The assessment types of Structured Activity are the Conditional Node and Loop Node, both of which you can simulate quite easily.

Conditional Node

A Conditional Node essentially consists of one or more pairs of Test / Body partitions, each pair being referred to as a Clause. The Test partition is composed of Activity diagram elements that test a condition, and if that condition is met further Activity diagram elements in the Body partition are executed to produce a result.

If there is one Clause, the Conditional Node either outputs the result of the Body partition, or no result. If there is more than one Clause, control flows from one Test to the next until either a condition is met and a Body partition is executed to produce a result, or all Tests fail.

Simulation currently supports use of the 'Is Assured' checkbox setting in the 'Condition' tab of the Element Properties window. The other two checkbox settings are ignored. If the 'Is Assured' checkbox is:

- Selected, at least one Test must be satisfied, so its Body is executed and a result output; if no Test is satisfied and no result output, the Conditional Node is blocked and processing cannot continue beyond it

- Not selected, a Test can be satisfied and a result output, but if no Test is satisfied and no result output, processing can still continue beyond the Condition Node

You can simulate a range of possible paths and outcomes by typing JavaScript sim. statements that define or lead to specific Test results and Body results, in the 'Effect' fields of the Action elements within each partition of each Clause. These sim. statements must identify the full path of the Conditional Node, Clause and Output Pin being set. For example, in a test to see if a person qualifies as a senior citizen:

if (sim.Person.age >=65)

sim.AgeCondition.Clause1.Decider1=true;

else

sim.AgeCondition.Clause1.Decider1=false;

The Condition Node is called AgeCondition, the test is in Clause1 and the OutputPin for that test is Decider1.

Loop Node

A Loop Structured Activity Node commonly represents the modeling equivalents of While, Repeat and For loop statements. Each Loop Node has three partitions:

- Setup - which initiates variables to be used in the loop's exit-condition; it is executed once on entry to the loop

- Test - which defines the loop exit-condition

- Body - which is executed repeatedly until the Test produces a False value

You define the Loop Nodes by dragging Activity diagram elements from the Toolbox pages into the Setup, Test and Body partitions. The Body partition can contain quite complex element structures, defining what the Loop Node actually produces in the process.

The Loop Node has a number of Action Pins:

- Loop Variable (Input) - the initial value to be processed through the Loop

- Loop Variable (Output) - the changing variable on which the Test is performed

- Decider - an Output Pin within the Test partition, the value of which is examined after each execution of the Test to determine whether to execute the loop Body

- Body Output - the output value of the processing in the Body partition, which updates the Loop Variable Output Pin

for the next iteration of the loop, and

- Result - the value of the final execution of the Test partition (which is the value passed back from the last execution of the Body partition)

You can simulate the effects of different actions and outputs through the loop, by typing JavaScript sim. statements that define or lead to specific Test results and Body results, in the 'Effect' fields of the Action elements within each partition. These sim. statements must identify the path of the Loop Node and Output Pin being set. For example, in an Action in the Test partition:

sim.LoopNode1.decider = (sim.LoopNode1.loopVariable>0);
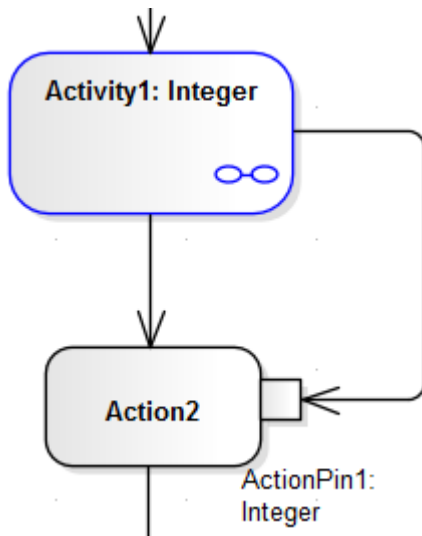
Activity Return Value Simulation

An Activity is likely to produce a return value as the output of the process it represents. You can simulate how that return value is passed on to the next stage in the process, under three scenarios:

- The Activity simply produces a return value, which is passed directly to the next Action
- The Activity has one or more Activity Parameters - represented on a diagram by Activity Nodes - that accept input values to or hold output values from the child Actions of the Activity, and the output Activity Parameter collects and passes on the return value
- The Activity is instantiated by a CallBehavior Action that replicates the behavior of the Activity and passes the return value onwards

Activity Return Value Pass Out

(This method is unique to Enterprise Architect simulation, mimicking the effect of an Activity Parameter without one having to exist.)

The Activity has a return value, which is transferred from the Activity element to an Action Pin on the next Action in the process via an Object Flow connector.

You can simulate this by setting a simple JavaScript statement to set the return value in the Activity's child element (such as this.return=12;) and, running the simulation, see the value transferred to the Action Pin in the Locals window.

Activity Parameter Pass Out

If the Activity has an Activity Parameter, its value passes to the corresponding Activity Node and then, via an Object Flow connector, to the Input ActionPin of the next Action in the process, as shown:
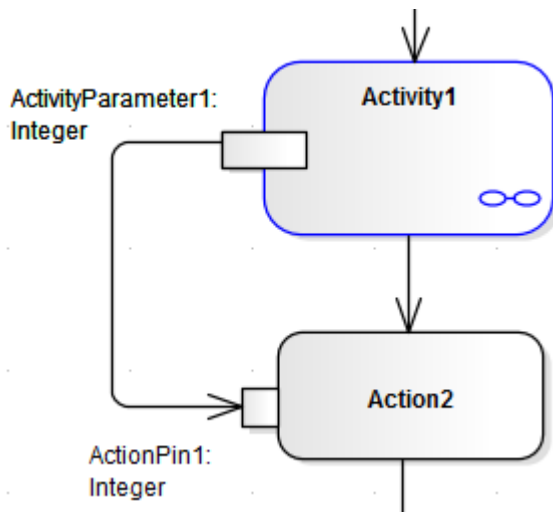
In the Locals window, you can either observe the Parameter's default value pass through to the ActionPin, or you can use JavaScript in the Activity's child Actions to simulate an update of the value within the Activity. For example:

   this.ActivityParameter1=20;

CallBehavior Action

An Activity might be used a number of times in a process, in which case you might want to generate a separate instance of the Activity each time. You can do this using a CallBehavior Action to create an object of the Activity and execute its behavior. The input and output Activity Parameters are bound to corresponding input and output Action Pins (arguments) on the CallBehavior Action.
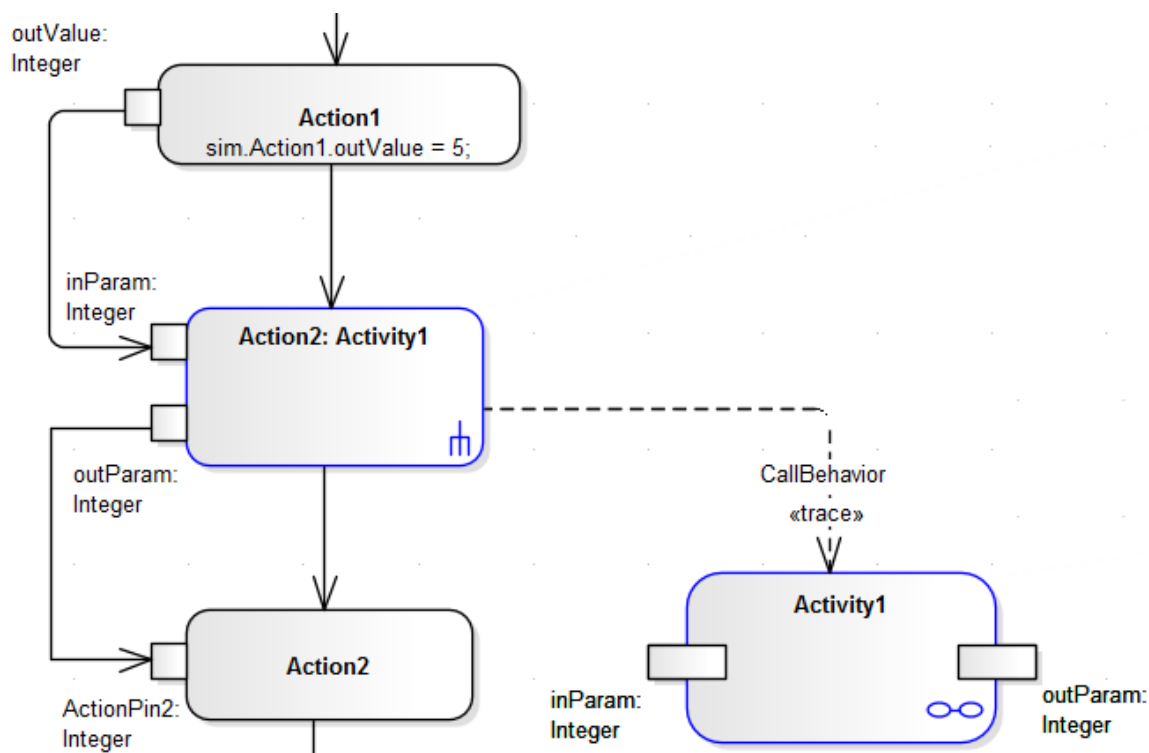
When you simulate the part of the process containing the Activity, you provide an input value (as in Action 1) that passes into the input Action Pin on the CallBehavior Action, which creates an Object of the Activity. The CallBehavior executes the behavior of the Activity, using the input Action Pin to act as the input Activity Parameter, and the Output Action Pin to receive the return as the output Activity Parameter. The Activity return value is then passed to an Action

Pin on the next Action, using an Object Flow connector. You can provide JavaScript statements in the Actions of the Activity to act on the input value and generate a return value, such as:

sim.buf=this.inParam;   and

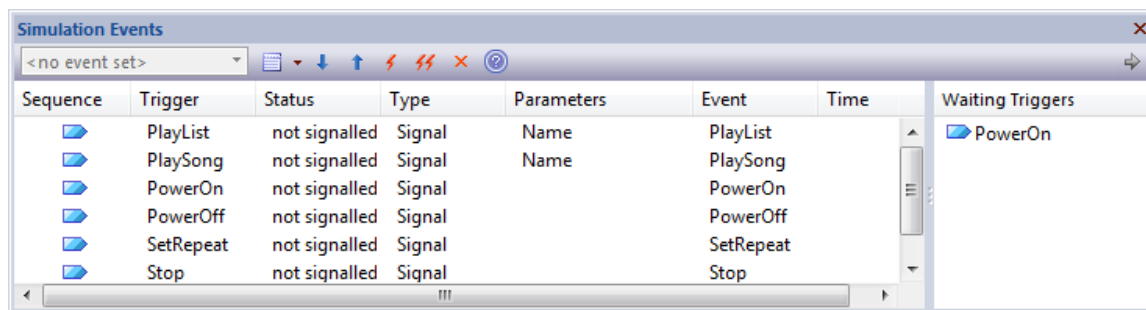this.outParam=sim.buf + 11:

Simulation Events Window

The Simulation Events window is where you manage triggers and sets of events in a simulation. Its main functions are to:

- Add, delete and re-sequence a set of triggers for a simulation
- Display a list of fired, lost and waiting events for the current running simulation
- Provide options to fire any arbitrary trigger into the current simulation
- Provide a convenient 'Waiting Triggers' list of triggers that the simulation is waiting on
- Save trigger sets for later use in both manual and automated simulations
- Accept triggers dragged from the Project Browser into the current list
- Enter trigger parameters for a waiting trigger prior to firing

As triggers are consumed in the simulation, their status and position is logged in the main body of the Simulation Events window.

You can save the log of fired triggers as a trigger set or event set to reapply in another Simulation run, which you can execute manually or automatically. See the topic *Trigger Sets and Auto-Firing* for more information on building and using Trigger sets.

This image illustrates the Simulation Events window during execution.

Access

Ribbon	Simulate > Core > Events

Column Details

Field/Column	Action
Sequence	During and after the simulation, indicates the position in the sequence in which a trigger was fired or is expected to be fired. Note that if a trigger is fired out of sequence, it will be moved to the bottom of the signalled events section.
Trigger	The name of the trigger - identifies the Trigger used to initiate the event.
Status	Indicates the status of the Trigger. Values can be: • used - the trigger has been fired and processing has passed on

	• lost - the trigger has been fired in the list, but it had no effect • signalled - a trigger was fired and consumed by one or more transitions • not signalled - the trigger has not yet been fired
Type	Indicates the type of trigger. Currently only supports: • Signal • (no type) an anonymous trigger
Parameters	For a Signal Trigger, initially shows the parameters required for firing by the Signal specification. For example a "Login" signal might include username and password parameters - and each triggered invocation can use different parameters. Each time the simulation fires the trigger, the system will prompt you for values. You can also edit the values directly in the list when the trigger is set to not signalled. Parameters are very useful for testing the conditional logic in your simulation and to simulate a variety of inputs and data coming in from outside the simulation.
Event	For a: • Signal Trigger, identifies the Signal specification • For anonymous Triggers has no value
Time	The simulation time at which the trigger was signalled. Note that this is an absolute (real world) time, and not a relative simulation event time.
Waiting Triggers	Lists the Triggers available for selection from the current state(s), including those where more than one trigger is possible at a single transition. Double-click on a trigger to add and signal it as the next trigger in the current event sequence. You can show and hide this panel by clicking on the gray arrow just above the panel.

Toolbar Items

Option	Action
Set1 ▾	Use this drop list to select and work with previously defined trigger sets. Before running a simulation, select a previously-defined trigger set to use for the next simulation run. You elect to not use a trigger set by selecting the <no event set> option.
▾	Click to create and delete trigger sets: • Save Set - Save the current trigger list as a new trigger set; the system prompts you for a name for the new set • Save Set As - Create a copy of the current set under a new set name • Delete Selected Set - Delete the current trigger set • Delete All Sets for Diagram - Delete all saved trigger sets for the current diagram
↓	Move the selected trigger one line down in the firing sequence of triggers.

		This option is not available if there are no signaled triggers below the selected line.
	↑	Move the selected trigger entry one line up in the firing sequence of triggers. This option is not available if there are no signaled triggers above the selected line.
	⚡	Click to fire the selected trigger. You can also fire the trigger by double-clicking on it.
	⚡⚡	Click to toggle auto-firing on and off. Auto-firing will fire the un-signalled triggers in your trigger set sequentially. If your set matches a valid execution path, then the simulation will run automatically. Out of sequence or unused triggers will be 'lost'. A breakpoint pauses the auto-firing and you will need to click on the next trigger to resume auto-firing the simulation.
	✕	Delete the selected trigger(s) from the list.

Context Menu Options

Option	Action
Signal Selected	Signal, or fire, the selected not signalled trigger.
Remove Selected	Remove a not signalled trigger from the sequence.
Re-Signal Selected	Fire a used or signalled trigger again.
Set All to Unsignalled	Set all used or signalled triggers to not signalled.
Clear Trigger List	Clear all triggers from the window, regardless of their status.

Waiting Triggers

When a simulation reaches a point where any change of state (for any thread) requires a Trigger to proceed, the simulation is effectively paused and control returns to the system. The simulation is now effectively waiting for some form of event (a real world signal) to proceed. The Waiting Triggers list is useful in helping to determine which Trigger should be manually signaled.

Access

Ribbon	Simulate > Dynamic Simulation > Triggers

The Waiting Triggers list on the Simulation Events window is:

- Populated on each Simulation cycle with any Triggers that would have an immediate effect if signalled

- Populated with a discrete set (any duplicates are not shown as a Trigger is effectively broadcast to all transitions at once)

- Activated by double-clicking on the Trigger of interest

- Includes all possible triggers - including those activating transitions on parents of currently nested states

This example shows that the current simulation has hit a point where two possible Triggers can influence the flow of execution.
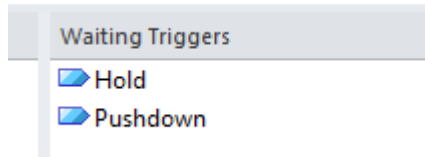
Due to the nature of Triggers and their effects, the list can refer to each of these example situations equally validly:

- A single state has two outgoing transitions that are respectively waiting for Hold and Pushdown; firing one of these will activate the associated transition in the simulation

- A single state has two or more possible triggers for the same transition, such as a security camera being switched on by a motion detector, sound detector or heat detector

- Two (or more) threads (concurrent regions) each have a state waiting on either Hold or Pushdown; firing one of these triggers will result in the thread(s) waiting on that trigger to proceed while the other thread(s) will remain blocked

- A child state is waiting on one of the triggers while a parent state is waiting on the other; firing a trigger will result in the associated transition being fired and either the child or parent proceeding accordingly

- Any combination of these

Re-Signal Triggers

It is possible to re-signal a Trigger as a shortcut of dragging in additional Trigger instances for signalling.

Access

Display the Simulation Events window, then right-click on a Trigger within that window and select the 'Re-Signal Selected' option.
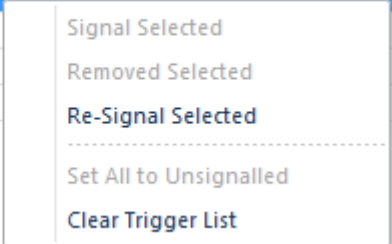
Ribbon	Simulate > Dynamic Simulation > Triggers > right-click on existing trigger > Re-Signal Selected

Untitled

The Simulation Events window contains a list of Triggers that have already fired. By right-clicking on a Trigger that you want to signal again, you can use the context menu to cause the re-signal to happen.

This image demonstrates resignalling in action. When a signal is re-signalled, a new copy is made and placed at the end of the signalled triggers list, where it is automatically fired again.
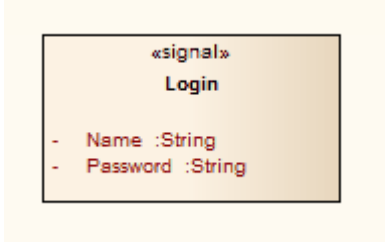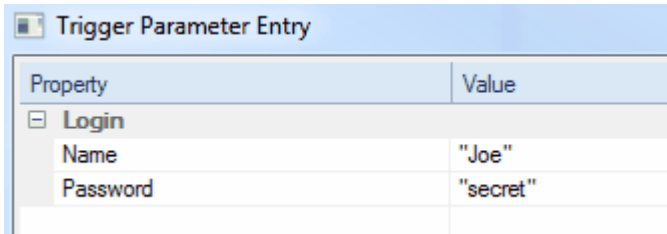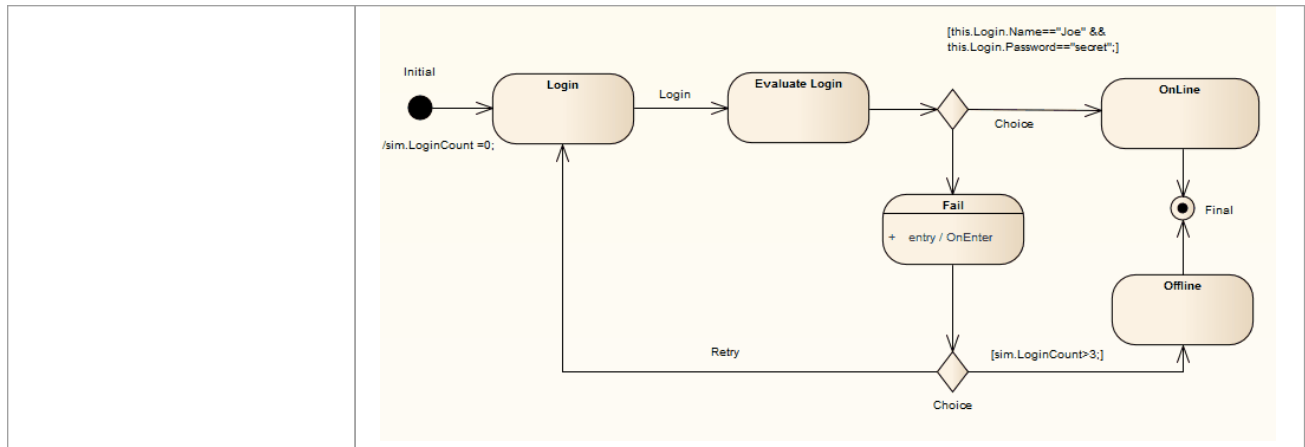
Trigger Parameters

Trigger parameters are arguments passed into the simulation along with a trigger when it is fired. They allow for complex behavior to be specified decision to be made based on variables and data passed into a simulation at run-time by a fired trigger (event).

Parameters

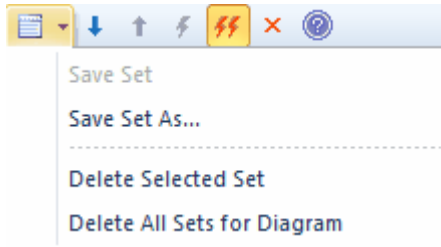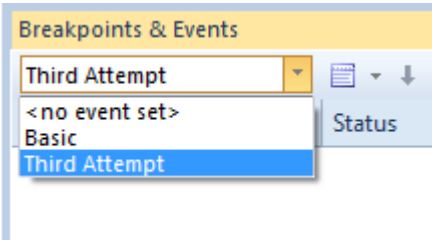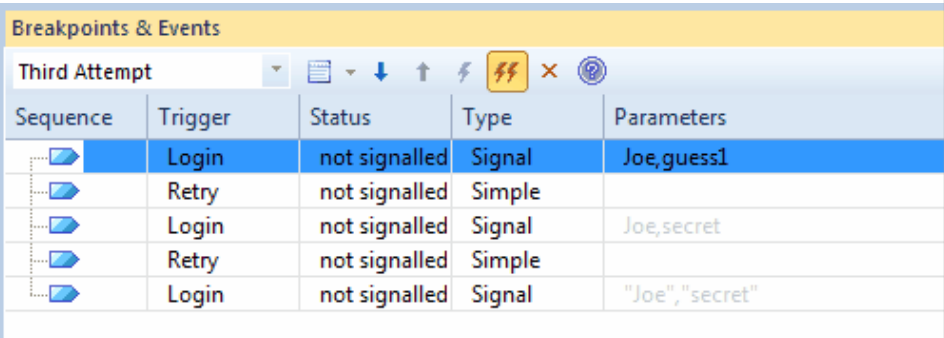| Parameter	Detail
Introduction	To use trigger parameters you: • First create a Signal element with the appropriate attributes that will become your parameters at run time • On a suitable transition in your diagram, create a trigger that is based on the signal created earlier • At run-time, will be prompted to enter suitable parameters - they are then passed in along with the trigger
Signals	A Signal element is a template or specification from which actual triggers can be built. This example has two arguments, a Name and a Password. These will be filled in at execution time either manually or as part of a pre-defined trigger set.
Trigger Parameters	The Trigger parameters 'prompt' that asks for suitable values for each parameter. Note that you need to enclose strings in double quotes, otherwise the interpreter will think you are referring to other variables.
Example Diagram	This is an example diagram that makes use of trigger parameters. At the Evaluate Login state, the simulation examines the variables passed in as trigger parameters and makes a decision to either accept the credentials or deny them.

Trigger Sets and Auto-Firing

Trigger Sets are a powerful means of automating and streamlining the execution, testing and validation of simulation models. By re-using sets of triggers (with or without parameters) it is possible to quickly and efficiently walk through many simulation scenarios, either manually or automatically using the 'auto-firing' tool.

Access

Ribbon	Simulate > Dynamic Simulation > Breakpoints > Simulation Breakpoints

About Trigger Sets

Aspect	Details
Trigger Sets	• Stored with an associated diagram • Made up of a list of Triggers in a set sequence • May include Trigger parameters where necessary • May be used manually by double-clicking Triggers to fire as required • May be used as part of the 'auto-fire' behavior to automate execution • Managed from the Simulation Events Window
Managing Sets	Trigger sets can be created by manually dragging triggers into the active triggers list and then using the 'Manage Trigger Sets' drop menu to save a new set. It is also possible to save a set of triggers built up during a single simulation setting as a new set. This is convenient for creating multiple test paths through a simulation, based on saving the manually fired triggers for each test case. You can also delete a set and delete all sets for the current diagram. It is also possible to load a set, modify parameters and/or order of firing and save the set with a new name. This is a convenient method for rapidly creating a suite of simulation test scripts.
Using Sets	To use a trigger set you first select it by name from the trigger set drop list as in this example image. Once selected it loads the Trigger List window with the defined trigger set. Note that the special item *<no event set>* means no set is currently selected. At the start of each simulation, if a set is selected, it will be loaded afresh for the next run.

If *<no event set>* is selected, the trigger list will be cleared.

Once you have selected a trigger set and the list of triggers loaded you have two options:

- Fire the triggers as required manually
- Use the auto-fire feature to fully automate the simulation

	Auto-Firing	Auto-firing is a convenient way of streamlining your simulations. Once you have loaded a trigger set, if you select the Auto-fire button then Enterprise Architect will automatically pick up waiting triggers when it reaches an impasse in the simulation. In practice, this means that trigger sets matching exactly a path through the simulation will automatically run without your intervention. As you can save any number of trigger sets with different pathways and trigger parameters, you can effectively and quickly test and work with many different scenarios.
Auto-Firing Rules		When a simulation runs with auto-firing enabled, Enterprise Architect will wait until a point is reached where the simulation is 'blocked' or stable, waiting on one or more triggers to advance the simulation. At that time, the first unfired trigger in the list will be picked up and fired into the simulation. The outcome depends on the relevance and perhaps on the parameters of the trigger. • If the trigger matches a 'waiting' trigger it is immediately consumed and the simulation advances • If the trigger matches no 'waiting' trigger or possible parent transition, then the trigger is 'lost' and the simulation remains in the current state; this corresponds to a scenario such as a user pressing an 'on' button several times in succession - there is no effect other than that occasioned by the first press

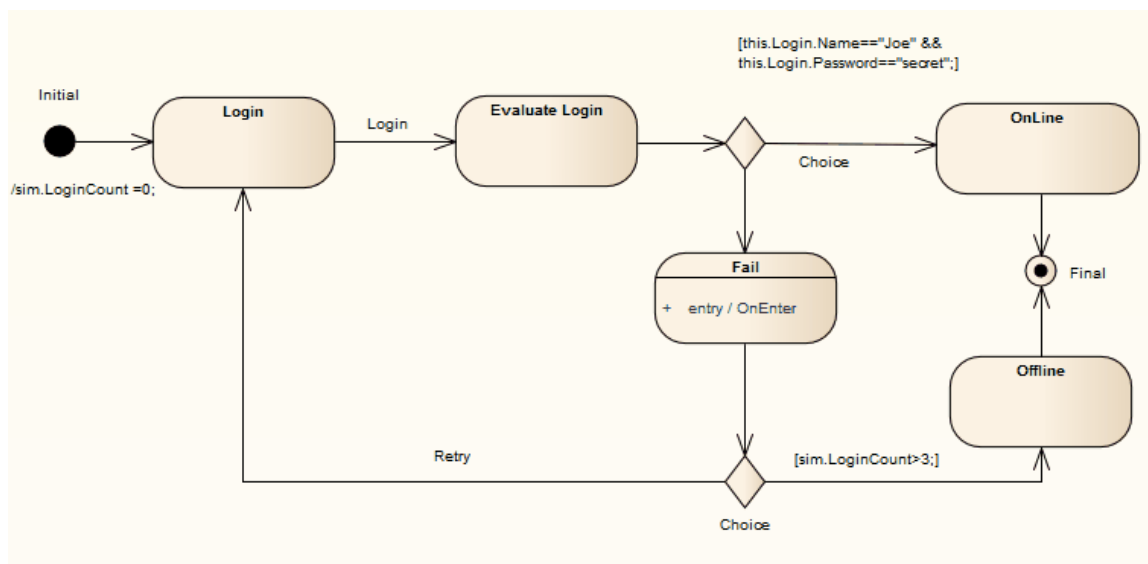# Using Trigger Sets to Simulate an Event Sequence

As a simple example of how useful trigger sets are, consider this example trigger set and associated diagram.

In this example we simulate a simple "three strikes and you are out" login process, taking a user name and password. The success path is waiting for the name "Joe" and the password "secret" (note - it is very important that parameters referencing strings are enclosed in quotes, otherwise the interpreter thinks the name refers to another variable within the simulation).

- Pass 1 tries Joe and guess1 - which fails

- Pass 2 tries Joe and secret, but as they are referring to variables, not strings - this fails as well

- Pass 3 shows the correct way of referencing trigger parameters and the simulation will succeed


Here is a simple diagram simulating a login process requiring a username and password pair.

Multi-threading - Forks and Joins

The Model Simulator provides the ability to handle multi-threaded simulations using Fork and Join nodes.

- In the example, the current execution point has forked into two threads, each with its own active node

- As this example progresses, the lower branch will wait at the Join node until the top branch has completed all its Actions

- Once the two threads merge back into one, the Simulation will continue as a single thread until completion

- When automatically stepping, each thread will be seen to execute a single step during one simulation 'cycle' - although when single stepping or at a breakpoint, the behavior is to alternate step between threads as each thread receives processing time

- Note that the Call Stack window will show two active threads and one 'paused' thread in the example; once the threads merge there will be a return to single threaded execution

- Also note that the Local variables are shared (global) between all threads; if you want to Simulate private variables on a thread you must create new Simulation variables at the start of each thread - pre-loading such variables with existing global data
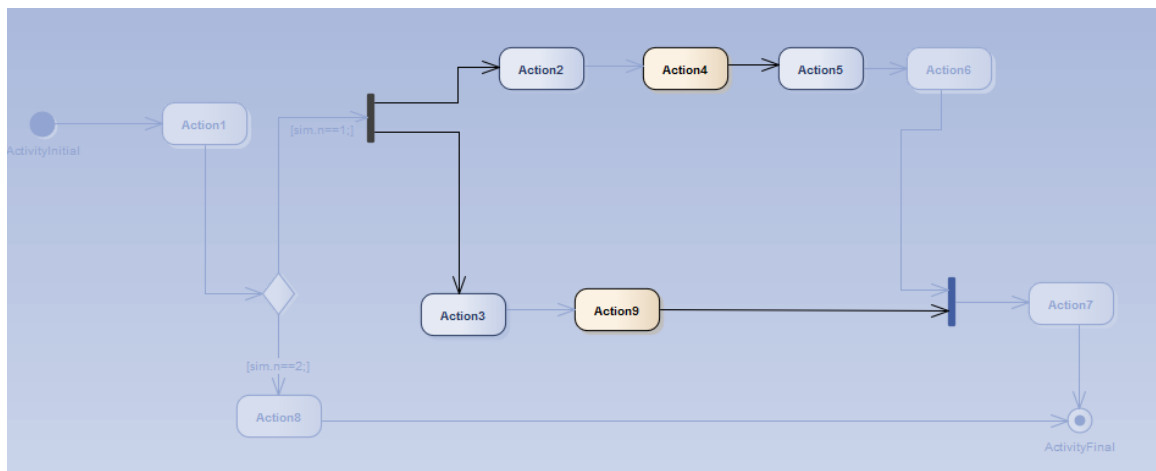
Multi-threading - Concurrent State Regions

Concurrent regions within a State represent state changes and processing that occurs in parallel inside one overall parent State. This is especially useful when one region raises events or modifies simulation variables that another region is dependent on. For example, one region could contain a simulated timer which raised events on set intervals that invoked state changes in the States within other regions.

Concurrent regions are essentially the same as Forks and Joins with similar logic and processing rules.

In the example:

- When the transition to SalesProcess is taken, each region is concurrently activated

- Credit is checked, the order totaled and the goods required packed up

- However, in the event that the Credit Check fails, this triggers the transition to the Sale Cancelled state; note that when this occurs, the entire parent state and all owned regions are immediately exited, regardless of their processing state

- If the Credit Check succeeds, the region moves to the final state and once the other regions have all reached their own final state, the parent state can then be exited
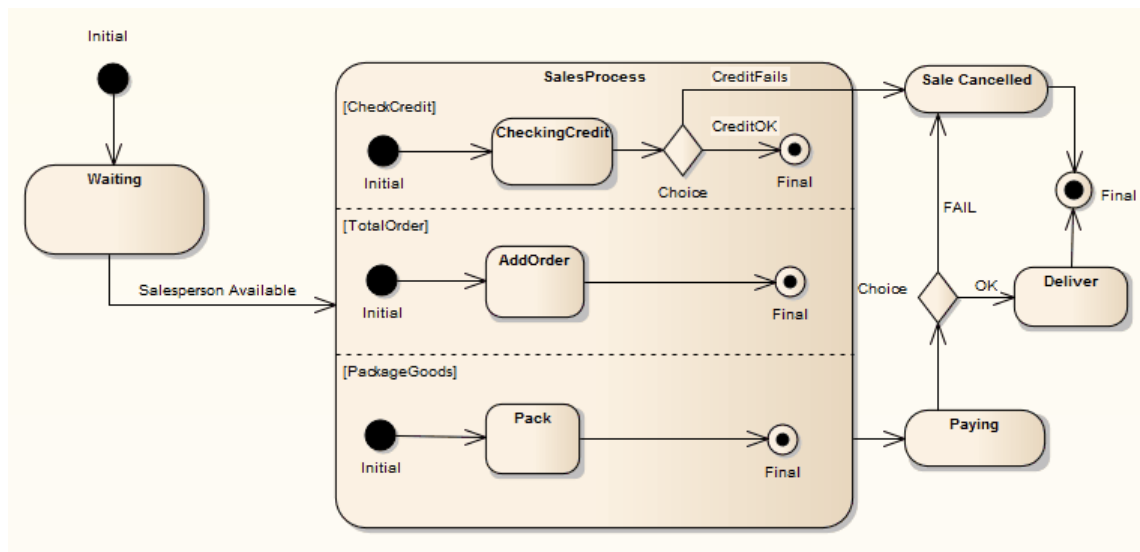
Using Composite Diagrams

If you want to simulate processing that includes a branch represented on a different diagram (for example, to reduce complexity on the main diagram, or to hide areas of processing that are only actioned under an exception), you can use a Composite element to represent and access the branch on its child Composite diagram. When you run the simulation and it reaches the Composite element, it opens the child diagram and processes it before returning (if appropriate) to the main processing path. This is an excellent method of following the processing path in a complex process, representing sections of the process with Composite Activity elements that expand the actual processing in their respective child diagrams. You can have several Composite elements representing different stages or branches of the process.

One aspect to watch for (and that would be revealed by a failure in the simulation) is to have multiple threads that process simultaneously on separate diagrams. The simulation cannot pass to a new diagram if it is also following another thread on the current diagram.

This diagram provides an overview of an ATM cash withdrawal process:
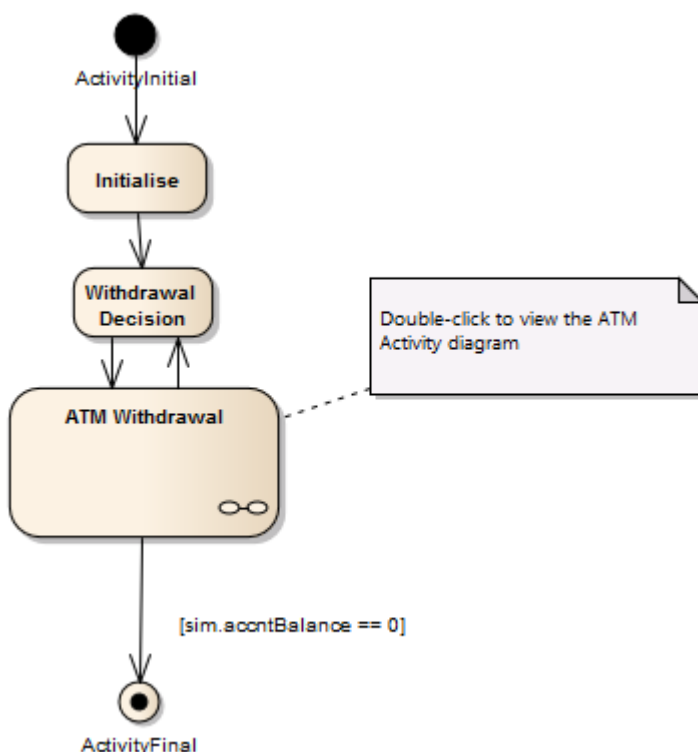
The ATM Withdrawal Activity is a Composite element. If you double-click on it, you open and display the child diagram, which is a more detailed breakdown of the withdrawal process. Similarly, a simulation will open and process the child diagram.
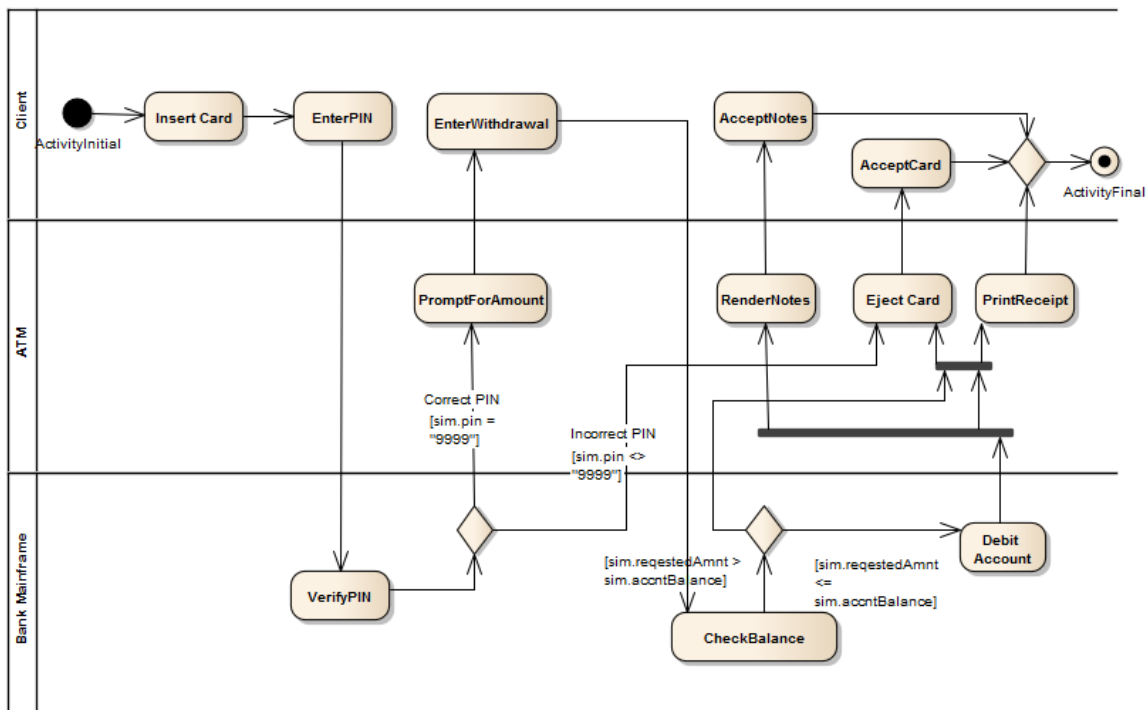
Win32 User Interface Simulation

Enterprise Architect supports the simulation of dialogs and screens created with the Win32® User Interface profile, to integrate user interface design with defined system behavior. Dialogs can be programmatically referenced and invoked using JavaScript commands within a behavioral model such as a StateMachine, providing a fully customizable and fully interactive execution of your behavioral model.

Button controls can be used to broadcast signals, firing a trigger when the button is clicked. Signal arguments can be filled from the dialog input fields; for example, to capture and send a username and password for evaluation.

Dialogs designed using the Win32 User Interface profile (and existing within the same Package branch as the behavioral model being executed) will be created as new windows in the background at the beginning of simulation. Various properties that can affect the appearance and behavior of each dialog and control can be customized at design time via Tagged Values provided by the Win32 User Interface profile.

To interact with a dialog via JavaScript during simulation, the 'dialog' simulation-level keyword is used, followed by a period and the name of the dialog. Properties and methods can then be accessed; for example, to show the dialog, or to set the text value of an 'Edit Control':

    dialog.Login.Show=true;

    dialog.Login.Username.Text="admin";

Examples

To view an example of the Win32 User Interface Simulation, open the EAExample model and locate the diagram:

    Example Model > Model Simulation > State Machine Models > Customer Login > Customer > Customer Login

Common Properties

These common properties and methods are available on most supported Win32 UI Control types.

Property/Method	Description
Enable	Boolean Enables or disables user interaction.
Move To (x,y,width,height)	Move the window to the specified coordinates and set the window height and width.
Show	Boolean Show or hide the dialog. When this property is set to False, the dialog is moved off-screen.
Text	String Set the title of the dialog or window.

JavaScript Functions

| Function	Description
BroadcastSignal (string Signal)	Sends a signal to the simulation event queue. Parameters: • Signal: String – the name of the Signal to be broadcast
UIBroadcastSignal (string Signal, array Parameters)	Sends a signal with additional parameters to the simulation event queue. Parameters: • Signal: String – the name of the Signal to be broadcast • Parameters: Array – additional parameters to be supplied for this Signal Example: UIBroadcastSignal("Login",{Name: dialog.Login.Username.Text, Password: dialog.Login.Password.Text});
ShowInterface (string InterfaceName, boolean Show)	Deprecated. See the **Show** property on the 'Dialog' control. For example: dialog.HelloWorld.Show = true;
InterfaceOperation (string InterfaceName, string ControlName, string OperationName,[string arg1],[string arg2])	Deprecated. Operations can be referenced directly from the control. For example: dialog.HelloWorld.ListControl.InsertItem("Test", 2);
GetInterfaceValue (string InterfaceName, string ControlName, string OperationName,[string arg1],[string arg2])	Deprecated. Properties can be referenced directly from the control. For example: dialog.HelloWorld.EditControl.Text;

Notes

- Controls must be within a dialog; any controls outside a dialog will not be interpreted
- Dialogs and controls must be on a Win32 User Interface diagram
- Simple UI controls and Basic UI controls can also be used in a simulation, but are limited in functionality compared to Win32 UI controls
- Dialog names and Control names must be unique; if multiple controls of the same name exist, the simulation will not be able to differentiate between them
- Spaces in dialog names and Control names are treated as underscores
- Dialog names and Control names are case sensitive

Supported Win32 UI Controls

This table identifies all of the Win32 UI Controls available in Enterprise Architect for user interface design and simulation.

Access

Ribbon	Design > Diagram > Toolbox : 🔍 > Specify 'User Interface - Win32' in the 'Find Toolbox Item' dialog
Keyboard Shortcuts	Alt+5 : 🔍 > Specify 'User Interface - Win32' in the 'Find Toolbox Item' dialog

Win32 UI Controls

Control	Description
Button	Button controls are a common way to allow user interaction during runtime; for example, an OK button in a login screen. A Button can respond to a click event, defined by adding an 'OnClick' Tagged Value. In response to a click event, a button can be used to, for example, send a signal, causing a trigger to fire during runtime. Customizable design properties: • Client Edge • Default Button • Disabled • Flat • Horizontal Alignment • Modal Frame • Multiline • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Transparent • Vertical Alignment • Visible Tagged Values: • OnClick – specifies a JavaScript command to be executed in response to a click event on this Button Properties: • Enable

	• Show • Text Operations: • MoveTo
Check Box	Customizable design properties: • Auto • Client Edge • Disabled • Flat • Horizontal Alignment • Left Text • Modal Frame • Multiline • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Vertical Alignment • Visible Tagged Values: • OnCheck – specifies a JavaScript command to be executed in response to a change in the value of this checkbox Properties: • Checker – integer value [0|1] • Enable • Show • Text
Combo Box	Customizable design properties: • Auto • Client Edge • Data – semi-colon delimited string of values to populate the combo box at runtime; for example, "yes;no;maybe" • Disabled • Has Strings • Lowercase • Modal Frame • Right Align Text • Right To Left Reading Order • Sort • Static Edge • Tabstop • Type • Uppercase • Vertical Scroll

	• Visible
	Operations
	• AddString (string)
	• DeleteAll ()
	• DeleteItem (number) – delete item at specified index
	• DeleteString (string) – deletes all items matching string
	• GetCount ()
	• GetString (number)
	• InsertItem (number, string)
	• InsertString (number, string)
	• SetString (number, string)
	Properties:
	• Enable
	• Selection – index of the currently selected item
	• Show
Dialog	Customizable design properties:
	• Absolute Align
	• Application Window
	• Border - Resizing or Dialog Frame only
	• Center
	• Client Edge
	• Center Mouse
	• Clip Siblings
	• Disabled
	• Horizontal Scrollbar
	• Left Scrollbar
	• Local Edit
	• Maximize Box
	• Minimise Box
	• No Activate
	• Overlapped Window
	• Palette Window
	• Right Align Text
	• Right To Left Reading Order
	• Set Foreground
	• System Menu
	• System Modal
	• Title Bar
	• Tool Window
	• Topmost
	• Transparent
	• Vertical Scrollbar
	• Visible
	• Window Edge

	Properties: • Enable • Show • Text Operations: • MoveTo
Edit Control / Rich Edit Control	Customizable design properties • Align Text • Auto HScroll • Auto VScroll • Border • Client Edge • Disabled • Lowercase (Edit Control only) • Modal Frame • Multiline • Number • Password • Read Only • Right Align Text • Right To Left Reading Order • Static Edge • Tabstop • Transparent • Uppercase (Edit Control only) • Visible • Want Return Properties: • Enable • Show • Text
Group Box	Customizable design properties: • Client Edge • Disabled • Flat • Horizontal Alignment • Modal Frame • Right Align Text • Static Edge • Tabstop • Visible Properties: • Enable

	• Show • Text
List Box	Customizable design properties: • Border • Client Edge • Disable No Scroll • Disabled • Left Scrollbar • Modal Frame • Right Align Text • Selection • Sort • Static Edge • Tabstop • Vertical Scroll • Visible Operations: • AddString (string) • DeleteAll () • DeleteItem (number) – delete item at specified index • DeleteString (string) – deletes all items matching string • GetCount () • GetString (number) • InsertItem (number, string) • InsertString (number, string) • SetString (number, string) Properties: • Enable • Selection – index of the currently selected item • Show
List Control	Customizable design properties: • Alignment • Always Show Selection • Border • Client Edge • Disabled • Edit Labels • Left Scrollbar • Modal Frame • No Column Header • No Scroll • Single Selection • Sort

	• Static Edge • Tabstop • View • Visible Tagged Values: • Columns – string to initialize column names and sizes for this List Control, separated by semi-colons: for example, "Column1;100;Column2;150;" Operations: • AddString (string) • DeleteAll () • DeleteItem (number) – delete item at specified index • DeleteString (string) – deletes all items matching the string • GetCount () • GetString (number, number) • InsertItem (number, string) • InsertString (number, string) • SetString (number, number, string) Properties: • Enable • Selection – index of the currently selected item • Show
Progress Control	Customizable design properties: • Border • Client Edge • Disabled • Marquee • Modal Frame • Smooth • Static Edge • Tabstop • Vertical • Visible Tagged Values: • Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100" Properties: • Enable • Pos • Range • Show • Step
Radio Button	Customizable design properties: • Auto • Client Edge

<table>
<tr><td></td><td>

- Disabled
- Flat
- Group
- Horizontal Alignment
- Left Text
- Modal Frame
- Multiline
- Static Edge
- Tabstop
- Vertical Alignment
- Visible

Tagged Values:

- OnChangeSelection – specifies a JavaScript command to be executed in response to a change in selection of this radio button

Properties:

- Checker – integer value [0|1]
- Enable
- Selection – integer value
- Show

</td></tr>
<tr><td>Slider Control</td><td>

Customizable design properties:

- Auto Tick
- Border
- Client Edge
- Disabled
- Enable Selection Range
- Modal Frame
- Orientation
- Point
- Static Edge
- Tabstop
- Tick Marks
- Transparent
- Transparent Background
- Tooltips
- Visible

Tagged Values:

- Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100"

Properties:

- Enable
- PageSize
- Pos
- Range
- Show

</td></tr>
<tr><td></td><td></td></tr>
</table>

Spin Control	Customizable design properties: • Alignment • Arrow Keys • Auto Buddy • Client Edge • Disabled • Modal Frame • No Thousands • Orientation • Set Buddy Integer • Static Edge • Tabstop • Visible • Wrap Tagged Values: • Range – string specifying minimum and maximum values for this control, separated by a semi-colon: for example, "1;100" Properties: • Enable • Pos • Range • Show
Static Text / Label	Customizable design properties: • Align Text • Border • Client Edge • Disabled • End Ellipsis • Modal Frame • Path Ellipsis • No Wrap • Notify • Path Ellipsis • Right Align Text • Simple • Static Edge • Sunken • Tabstop • Visible • Word Ellipsis Properties: • Enable • Show • Text

Tab Control	Customizable design properties: • Border • Bottom • Buttons • Client Edge • Disabled • Flat Buttons • Focus • Hot Track • Model Frame • Multiline • Right Align Text • Static Edge • Style • Tabstop • Tooltips • Visible Tagged Values: • Tabs – string specifying names of each tab for this control, separated by a semi-colon: for example, "Tab 1;Tab 2;Tab 3;" Properties: • Enable • Show
Tree Control	Customizable design properties: • Always Show Selection • Border • Check Boxes • Client Edge • Disable Drag Drop • Disabled • Edit Labels • Full Row Select • Has Buttons • Has Lines • Horizontal Scroll • Left Scrollbar • Lines At Root • Modal Frame • Right Align Text • Right To Left Reading Order • Scroll • Single Expand • Static Edge

	• Tabstop
	• Tooltips
	• Track Select
	• Visible
	Operations:
	• Delete () - delete the specified TreeItem
	• InsertItem (string) - dotted path of the new tree item to be inserted; any parent items in this dotted path that do not yet exist will be created automatically
	• InsertString (string) - See *InsertItem*
	• TreeItem (string) - dotted path of the tree item to be accessed; use the *Text* property to set text for this tree item, or use the *Delete* operation to delete this item from the tree
	Properties:
	• Enable
	• Selection – string containing dotted path of the selected tree item
	• Show
	• Text – get or set text for a specified TreeItem
	Examples:
	dialog.MyDialog.MyTreeControl.InsertItem("Root.Parent.Child");
	dialog.MyDialog.MyTreeControl.TreeItem("Root.Parent.Child").Text = "Modified";
	dialog.MyDialog.MyTreeControl.Selection = "Root.Parent";
	dialog.MyDialog.MyTreeControl.TreeItem("Root.Parent.Modified").Delete();

Win32 Control Tagged Values

Various properties that can affect the appearance and behavior of each Win32 dialog and control can be customized at design time via Tagged Values provided by the Win32 User Interface profile.

Tagged Values

Some control types support the addition of special Tagged Values that modify their behavior.

Controls such as Buttons, Check Boxes and Radio Buttons can react to GUI events and execute a JavaScript command. To allow a control to respond to an event, create a new Tagged Value with an appropriate name; for example, 'OnClick', then type the JavaScript command into the value.

Tab Controls can use a 'Tabs' Tagged Value to define the tabs that will appear within this control when it is simulated.

Slider Controls, Spin Controls and Progress Controls can use a 'Range' Tagged Value to define the default minimum and maximum values accepted by the control during simulation.

Tag	Description
Columns	**Applies to:** List Control **Use:** Initializes column names and widths for a List Control. Each column name and width is separated by a semi-colon; for example, "Column1;100;Column2;150;".
OnClick	**Applies to:** Button **Use:** Identifies the JavaScript command to be executed in response to a click event on a Button control.
OnCheck	**Applies to:** Check Box **Use:** Identifies the JavaScript command to be executed in response to a change in the value of a Check Box control.
OnChangeSelection	**Applies to:** Radio Button **Use:** Identifies the JavaScript command to be executed in response to a change in the value of a Radio Button control.
Range	**Applies to:** Slider Control, Spin Control, Progress Control **Use:** Specifies the default minimum and maximum values for the control, separated by a semi-colon: for example, "1;100".
Tabs	**Applies to**: Tab Control **Use:** Specifies the name of each tab to be created for the Tab Control, separated by a semi-colon: for example, "Tab 1;Tab 2;Tab 3;".

BPMN Simulation

BPMN simulation is a method for visualizing and validating the behavior of your BPMN Business Process diagrams. With visual indications of all currently executing activities and the possible activities that can be executed next, you will easily be able to identify and resolve potential issues with the process you have modeled.

Simulating BPMN models is similar to simulating standard UML Behavioral models, except that BPMN:

- Uses some different element types (such as Gateway instead of Decision) and

- Operates on scripting placed, generally, in the appropriate 'Tagged Value' field associated with the connectors and elements, instead of in the 'Properties' fields (and, if you prefer, rather than in the 'Execution Analyzer Build Scripts' dialog); the scripting is written in JavaScript

Working with BPMN Simulation

Activity	Detail
Create a BPMN Simulation Model	When you create a BPMN model suitable for simulation, you take into consideration how you represent the start point, the flow and the conditions to be tested.
Compare UML Activities to BPMN Processes	The simulation of BPMN Business Process models has a number of differences to the simulation of UML Activity diagrams.

Notes

- BPMN simulation is available in the Corporate, Unified and Ultimate editions of Enterprise Architect

Create a BPMN Simulation Model

As part of the process of developing a simulation model, consider which of the three options for performing the simulation you prefer to apply:

- Execute a simulation script to initialize the variables for the diagram - select 'BPMN' as the Platform, execute the simulation as 'As Script' and select the script; you would then define the conditions and decisions as JavaScript declarations within the Tagged Values of the elements and connectors on the diagram, either before you start the simulation or during the simulation

- Do not use a script, but initialize the variables within the first Activity and, again, modify the conditions and decisions within the Tagged Values of the elements and connectors, then execute the simulation as 'Interpreted'; you can then re-initialize the variables during simulation, as well as the conditions

- Execute the simulation as 'Manual' and manage the flow and conditions manually at each step

Create a BPMN diagram suitable for simulation

Step	Action
1	Create a Business Process or BPEL diagram from the BPMN 2.0 technology. If you create a BPEL diagram Enterprise Architect displays specialized dialogs to streamline the creation of compliant models.
2	We recommend that you create a Start Event to clearly show where your simulation starts. You have several choices for the Event Type; the choice does not influence the simulation of your model. If no Start Events are defined, the simulation will start from an Activity that has no incoming Sequence Flows.
3	Add all of the Activities that are involved in the Process being modeled. You have several choices for the Task Type; the choice does not influence the simulation of your model. The behavior of Activities can be further decomposed by specifying an Activity Type of Sub-Process and selecting Embedded or CallActivity. Standard Loops are also supported.
4	Add Sequence Flows between your activities. In the 'BPEL properties' dialog you can enter the condition that must be satisfied (True) before the Sequence Flow will be followed. You can also set the conditionType to 'Default' to ensure that this flow will be taken if all other branches fail the condition specified. If you are not working with a BPEL diagram, you use the conditionExpression and conditionType Tagged Values.
5	Add End Events for any conditions that will cause the process or active execution path to end. You have several choices for the Event Type; of these only the Terminate type will influence the execution. In simulations with multiple active nodes, it causes the entire process to terminate instead of just the thread that reaches that node.

Notes

- To include Activities that are in Packages external to the Package being simulated, either draw a:
    - Package Import connector from the Package containing the diagram
      being simulated to each external Package, or
    - Dependency connector from the Package containing the diagram
      being simulated to each Activity in the external Packages

Initialize Variables and Conditions

For a BPMN simulation model, you can initialize your variables in an Execution Analyzer script. You can also initialize these variables in the Tagged Values of the first Activity element of the process, which gives you greater flexibility in adding and changing variables as the simulation proceeds. Similarly, you can define the conditions and values to apply at the various decision points (Gateways) in the process, in the Tagged Values of the Sequence Flow connectors.

If you want to incorporate a user-interface into your simulation process, using Win32, you again use Tagged Values to identify the dialog or prompt to display, in the Activity element just prior to the point at which the value or decision is processed.

For the simulation of UML diagrams, variables inside the 'sim' object and 'this' object are displayed in the Local Variables window.

Access

Display the Tagged Values window, using one of the methods outlined here.

Ribbon	Start > Explore > Properties > Tagged Values
	Show > Portals > Window > Properties > Tagged Values
Keyboard Shortcuts	Ctrl+Shift+6

Initialize Variables

1. On the diagram, click on the first Activity element in the process.

2. In the Tagged Values window, click on the drop-down arrow of the taskType 'value' field, and select 'Script'.

3. In the script 'value' field, type in the appropriate JavaScript code, such as:

    sim.loan=true; sim.status="undefined";

Define Conditions

1. On the diagram, click on a Sequence Flow connector that issues from a Gateway element.

2. In the Tagged Values window, click on the drop-down arrow of the conditionType 'Value' field, and select 'Expression'.

3. In the conditionExpression 'Value' field (<memo>*) click on the [...] button to display the Tagged Value Note window. Type in the appropriate JavaScript code, such as:
    sim.status=="Hold"

4. Click on the OK button. The statement text displays as a label of the connector.

Incorporate Win32 User Interface

1. On the diagram, click on the Activity element that represents where the decision is made.

2. In the Tagged Values window, click on the drop-down arrow of the 'taskType value' field, and select 'Script'.

3.   In the 'script value' field, type in the appropriate JavaScript code, such as:
      dialog.Screen1.Show=True;
      (This statement displays the dialog Screen1. You can temporarily hide the dialog by changing 'Show' to False.)

Comparison of UML Activities and BPMN Processes

The execution and simulation of BPMN models have a number of differences from the execution and simulation of UML Activity diagrams. The mapping of similar concepts, and the differences between the two methods of expressing the behavior of a system, are presented here.

Comparison of UML Activities and BPMN Processes

UML Activity	BPMN Business Process
The starting point is defined by an Initial Node. No method of specifying why the Activity was started is available.	The starting point is defined by a Start Event. This implies a specific cause for the Activity to start, although it could be unspecified.
The basic behavior unit in an Activity is the Action element. UML provides many different forms of Actions, although the simulation makes use of a small subset of these.	The basic behavior unit in an Activity is the Activity element. A number of different Task Types are available. These typically describe different methods of execution (for example Manual) as opposed to what happens.
A Control Flow is used to connect the elements on an Activity diagram. A distinguishing feature is that only a single Control Flow can be followed from any node, except for an explicit Fork Node. To restrict flow on a Control Flow, add a Guard.	A Sequence Flow is used to connect the elements on a Business Process diagram. These differ from UML Activity diagrams in that all valid sequence flows are taken by default. To restrict flow on a Sequence Flow set the conditionType Tagged Value to 'Expression' and create the script in the conditionExpression Tagged Value.
A Decision node is used to explicitly model a decision being made. A Merge node, which uses the same syntax is used when the potential flows are combined back into one.	A Gateway node set to 'Exclusive' is used when a single path must be selected. It is also used to combine the potential flows again. A direction can be specified as 'Converging' or 'Diverging' to explicitly select between the two modes.
A Fork node is used to concurrently execute multiple nodes, while a Join node, using the same syntax is used to wait for all incoming flows to become available and leave with a single flow.	A Gateway node set to 'Parallel' is used to explicitly model concurrent execution of multiple nodes. It is also used to wait for all incoming flows to become available and leave with a single flow. A direction can be specified as 'Converging' or 'Diverging' to explicitly select between the two modes.
There is no allowance for concurrently executing	A Gateway node set to Inclusive is used to explicitly model the situation where all outgoing flows with a true condition are executed concurrently.

only some outputs from a node for UML Activities. If you needed this you add later Control Flows with the appropriate Guards.	
A Call Behavior Action is used when behavior needs to be further decomposed by referring to an external activity.	Activity elements are set as an CallActivity Sub-Process when behavior needs to be further decomposed by referring to an external activity.
Activity Action Call Behavior Action.	Activity elements are set as an Embedded Sub-Process when behavior needs to be further decomposed without referring to an external activity.

Business Process Simulation (BPSim)

The open BPSim specification provides a rich set of material on how to configure and assign resources to activities/tasks, how to raise events, decision making and other real-world capabilities. Once configured according to the BPSim specification, a business process model (constructed in BPMN) can be sent to a suitable BPSim simulation engine and run according to the process defined in the BPMN model, using the configuration data attached in the BPSim information.

The BPSim specification is very detailed and offers the interested modeler and business strategist an unprecedented flexibility in assigning operating information to a model and then assessing the quality of the solution based on information received back from the Simulation engine. This section describes in detail the various screens and options available when configuring a model for BPSim execution.

Sparx Systems provide a BPSim-capable simulator (for separate purchase) - the **MDG BPSim Execution Engine**. This Add-In integrates with the BPSim and BPMN models defined in Enterprise Architect, providing the capability to run and store the results from multiple simulations and to perform convenient comparisons across each configuration's result set.

Installing the Sparx Systems **MDG BPSim Execution Engine** and registering the licence for it are pre-requisites for accessing and using the built-in BPSim configuration facilities. Once you have set up a BPSim configuration, you can export the BPMN model along with its BPSim data in a standard form that can be consumed by the Sparx Systems BPSim Execution Engine and by any other standards-compliant BPSim engines.

BPMN Model with BPSim Configuration
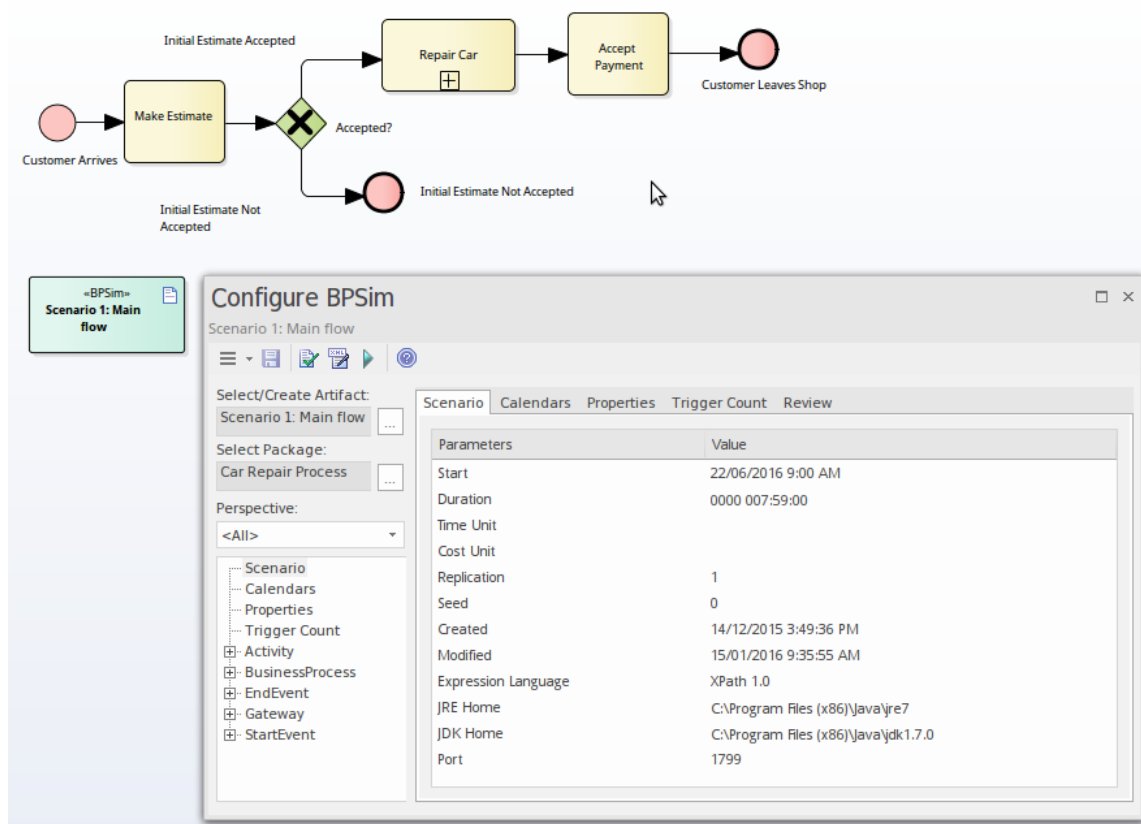
The BPMSim configuration screens have been divided into multiple Perspectives, each of which focuses on one aspect of the set up process at a time. The three Perspectives are:

- Control Perspective, which examines how activity flows through the process, moderated by the likelihood of a sequence of events and the priorities of certain events
- Time (Temporal) Perspective, which examines how the duration of one or more phases in the processing of an

Activity influences the business process

- Resource Perspective, which examines the involvement of types and roles of workers and other resources, their required numbers, their costs and their availability

A fourth Perspective (<All>) presents all of the options of the other three Perspectives.

You can also maintain multiple versions of a configuration and easily compare the differences between versions to see how each configuration will vary the flow of the proposed simulation or process execution. You might, for example, establish a baseline configuration and then create multiple 'what-if?' configurations that vary one or more parameters. Once run through a simulation engine you can examine each result and decide on the relative merits of each configuration. One useful facility to apply here is the simple inheritance of common, unchanged data in one configuration by another configuration that contains only the data being varied.

Users can combine the BPSim and Charting facilities to quickly vary, simulate and compare aspects of a Business Process model, and show the differences between the simulations in one of the many Chart formats.

If you are working across multiple projects, you can export and import the BPSim configurations between them. The configuration automatically carries with it the BPMN 2.0 model on which it is based.

The Enterprise Architect Business Process Simulation configuration tool is based on the BPSim Framework developed by the Workflow Management Coalition (WfMC).

Install the BPSim Execution Engine

You can purchase and download the MDG BPSim Execution Engine from the 'Products' page of the Sparx Systems website (sparxsystems.com/products). As part of this process, you will receive a registration key by email, to enable you to use the system beyond a 14-day trial period.

The MDG BPSim Execution Engine installer does not include JRE or JDK, therefore you must install JRE version 1.7 or higher to use the simulation and, if your BPSim Configuration contains any property parameters, you must also install JDK version 1.7 or higher.

You do not need to do any further configuration of the engine unless you have multiple versions of JRE/JDK on your system and you want to specify which version the execution engine should use. In this case, apply these environment variables as shown:

1. Click on the Windows 'Start' icon and select the 'Computer' option.

2. From the banner menu, select the 'System properties' option.

3. From the side panel, select the 'Advanced system settings' option.

4. On the 'Advanced' tab of the 'System Properties' dialog, click on the Environment Variables button.

5. On the 'Environment Variables' dialog, in the 'System variables' panel, click on the New button.

6. On the 'New System Variable' dialog, complete the fields with the values shown:

    For JRE: Variable name: MDG_BPSIM_JRE_HOME
                Variable value: C:\Program Files\Java\jre7

    For JDK: Variable name: MDG_BPSIM_JDK_HOME
                Variable value: C:\Program Files\Java\jdk1.7.0_51

7. Click on the OK button.

8. You must re-start your machine for the new variables to take effect.

Notes

- If you click on a business process element or connector in a diagram or in the Project Browser, it is highlighted and selected in the Configure BPSim window

- The Business Process that you simulate can contain elements from more than one Package; to include the external elements in the simulation, you must create a Package diagram containing the 'parent' Package and either the

'external' Packages containing the external elements, or the external elements themselves; create a:
    - Package Import connector from the parent Package to each external Package, or
    - Dependency connector from the parent Package to each external element

BPSim Configuration

A Business Process Simulation (BPSim) configuration is represented and contained in a Business Process Simulation Artifact element, which you can create on a diagram in any Package in the same project as the BPMN model that you are working with.

Create a Business Process Model

Each BPSim configuration is created specifically for and from an existing Business Process, defined in BPMN. Therefore you will need to create or import the BPMN model on which the configuration is to be based, before you use the Business Process Simulation Artifact.

This example diagram can be found and worked on in the EAExample model, in:

   Analysis and Business Modeling > BPMN 2.0 Examples > Process Diagrams > Shipment Process of a hardware retailer
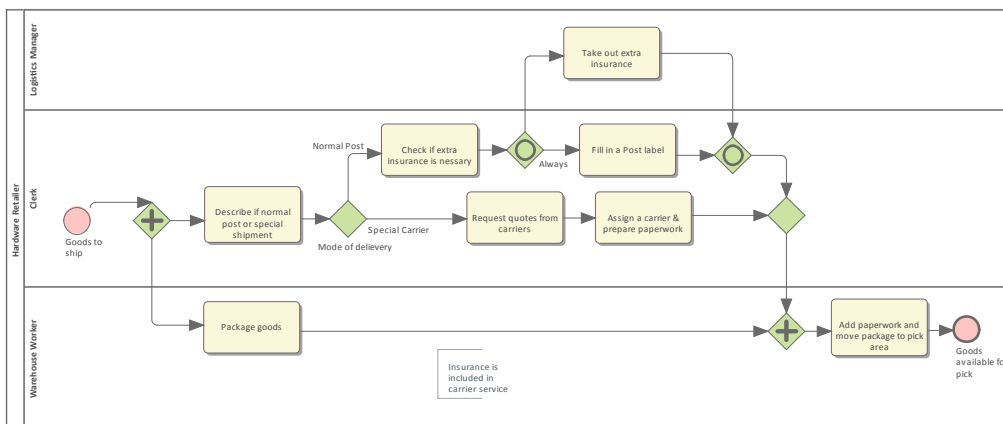
Create a Business Process Simulation Artifact

Open a diagram in which to create the Artifact, and display the Diagram Toolbox (press Alt+5). Expand the common 'Artifacts' page, and drag the 'Business Process Simulation' icon onto the diagram.

When you set up the Artifact, consider whether you might create separate Artifacts and use Generalization connectors between them so that one Artifact inherits the data you have defined in another Artifact. This way, you do not have to keep re-defining the whole configuration in every Artifact you create.

Double-click on the element and give it an appropriate name, such as (for the example) 'Shipment Process - Resource'.

Configure BPSim Window Overview

Right-click on the Artifact element (either in the diagram or in the Project Browser) and select the 'Configure BPSim' option. The Configure BPSim window displays for the Artifact.
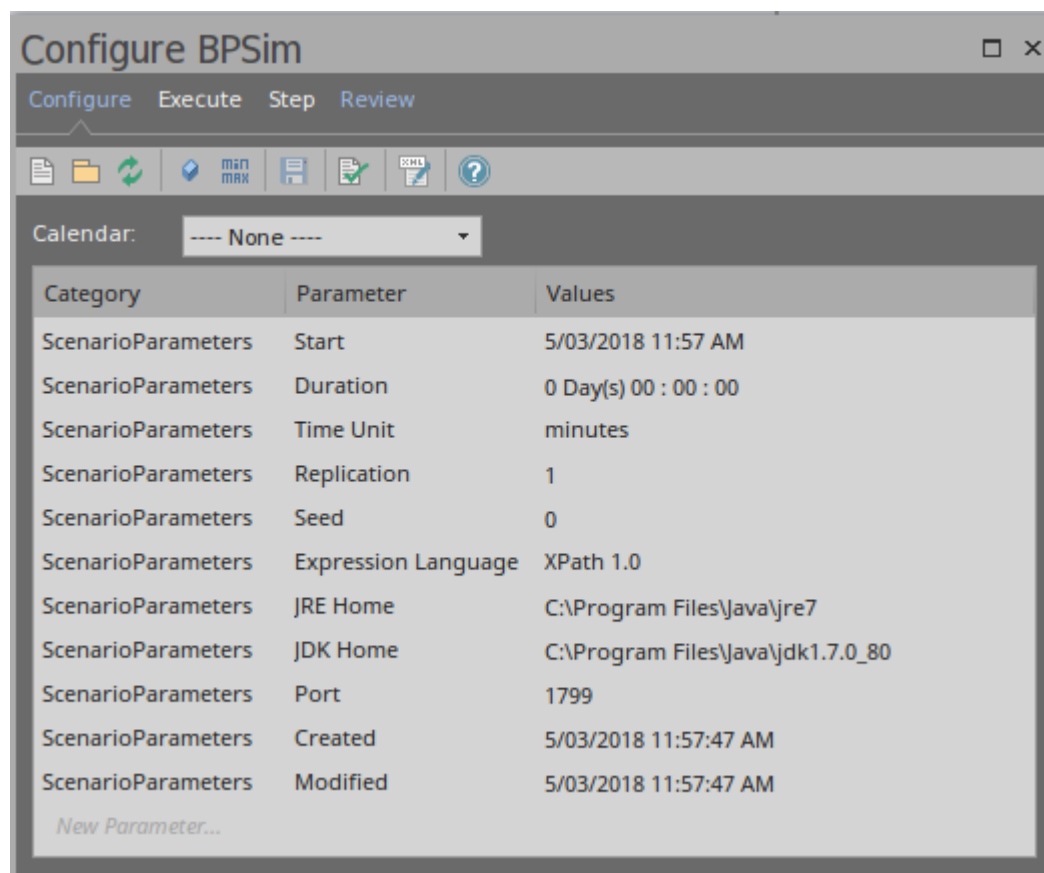
This window contains 4 tabs: Configure, Executive, Step and Review.

- Configure: Configure BPSim parameters for each BPMN Element; Define Property Parameters, Calendars and Scenario Parameters.

- Execute: Execute the BPMN model with BPSim configuration.

- Step: Step Over / Step In to provide an insight look to the execution process, including token status, property values and resource allocations per time/step.

- Review: Review / Compare configuration artifact(s), Standard / Customized simulation result report(s)

BPSim - Configure Page

The BPSim Artifact will be configured to a Package. All BPMN elements under this Package or its sub-Packages will be loaded. By default, the Package containing this Artifact will be configured when loaded into this window.

This window is context sensitive. When an element is selected on a diagram or the Project Browser, the list will show current configurations for the element; also, the combo boxes will only show available parameters for the element.

When the BPSim Artifact is the context element, the list will show the ScenarioParameters.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Configure Page

Toolbar Options

Option	Description
	Click on this button to select or create a DMNSimConfiguration element.
	Click on this button to set a Package for the BPSim Artifact. All BPMN elements under this Package or its sub-Packages will be loaded.
	Click on this button to reload BPMN elements from the configured Packages. For example, when some BPMN elements are modified, run this command to reload the Package so that the changes will be taken into account for BPSim Simulation.
	Click on this button to define Properties, which can be used as Property Parameters on BPMN elements.
	Click on this button to define Calendars, which can be used to configure element parameters.
	Click on this button to show or hide the 'Result Request' column. Result Request configuration is required for a Custom simulation. The execution report will only contain results that are requested.
	Click on this button to save the BPSim Simulation window information to a BPSim Artifact element.
	Click on this button to validate the BPMN model and the BPSim Configurations. Error or Warning messages might display in the System Output window if they are generated.
	Click on this button to export the BPMN model with the BPSim configuration. This exported BPMN file conforms to the BPMN and BPSim specifications and can be used by third-party BPSim Execution engines.

Scenario Parameters

A scenario is composed of a collection of element parameters. The scenario itself defines parameters used by all elements as global settings.

Name	Description
Start	The date and time at which the process starts to take effect.
	You can edit this by overtyping the values or, for the date, by selection from a drop-down calendar.
Duration	The length of time the process takes.
	The 'Duration' parameter is a required value. It must be long enough to accommodate a complete simulation; for example, if a process (and hence its simulation) takes three hours to complete, the 'Duration' parameter must be set to a value greater than three hours.
	You can edit this by overtyping the appropriate segment in the format 'days hours:minutes:seconds'.
Time Unit	The base unit in which periods of time are expressed in this scenario. All numeric and floating values representing time should be considered as being expressed in this unit, unless overridden locally.
	You can edit this by clicking on the drop-down arrow and selecting the unit.

Cost Unit	The currency unit of any costs recorded in the process. All numeric and floating values representing a cost should be considered as being expressed in that currency code, unless overridden locally. You can edit this by clicking on the drop-down arrow and selecting the unit abbreviation.
Replication	The number of replications of the scenario to be executed. Defaults to 1. You can edit this by simply typing a value in.
Seed	A random seed to be used to initialize a pseudo-random number generator. You can edit this by simply typing a value in.
Expression Language	XPath 1.0 and Java - XPath 1.0 is the default language. If Java is specified as the expression language, JDK Home must be set. You can edit this by clicking on the drop-down arrow and selecting the language.
DMN Module	When business rule tasks are used in the BPMN Model, you can implement these tasks as a DMN Model. You might first create a DMN Model and generate a DMN Server in Java, then click on the [...] button to specify the generated DMN Server file.
JRE Home	The Enterprise Architect BPSim Execution Engine runs in a Java Environment, therefore a JRE Home has to be specified. Click on the [...] button to choose a directory; for example, C:\Program Files\Java\jre7. You can edit this by clicking on the [...] button again to browse the directory.
JDK Home	When the expression language is Java, the Enterprise Architect BPSim Execution Engine will generate Java code and compile with javac as the vendor extension. So a JDK Home must be specified. Use the [...] button to choose a directory (such as C:\Program Files\Java\jdk1.7.0_80). You can edit this by again clicking on the [...] button to browse the directory.
Port	The Port number that Enterprise Architect used to communicate with the BPSim Execution Engine. The default port number is 1799.
Created	Read only field. The timestamp when the BPSim Artifact was created.
Modified	Read only field. The timestamp when the BPSim Artifact was last modified.

Control Parameters

To begin defining Control Parameters:

- Select the element on the diagram, then choose 'Control' in the 'Category' column in the list
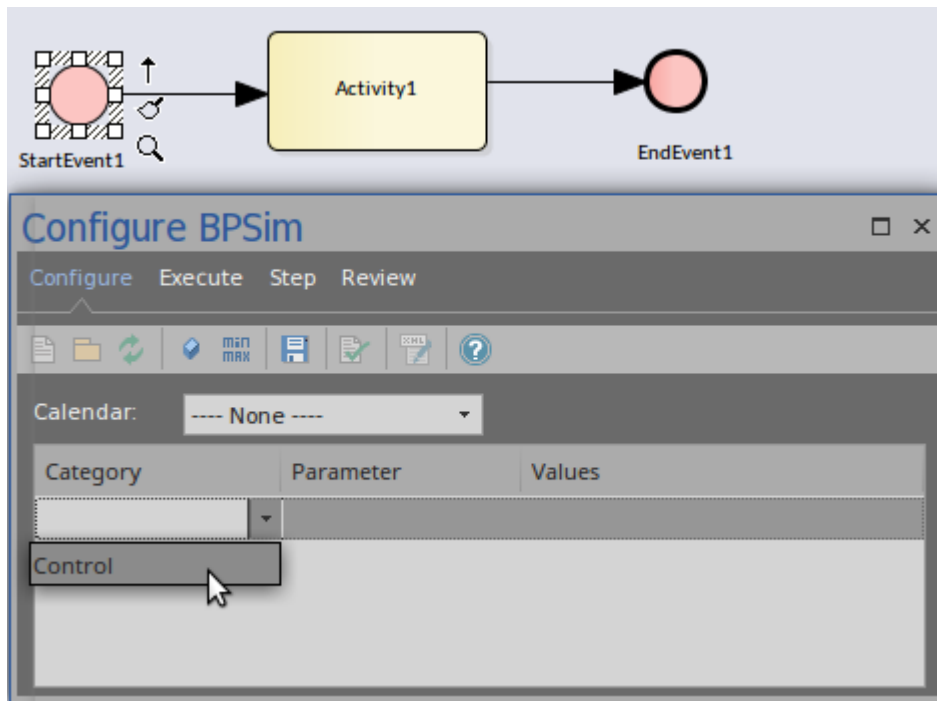
- After the category selection, the combo box in the 'Parameter' column will be filled with all the available parameters for the selected element
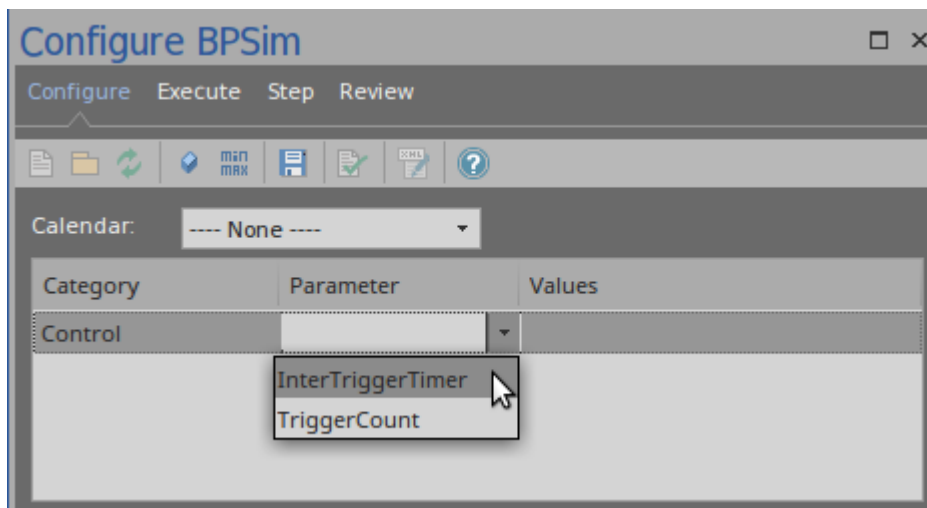
- After the parameter selection, you can enter the parameter value in the field or use the [...] button to open the 'Parameter Value' dialog
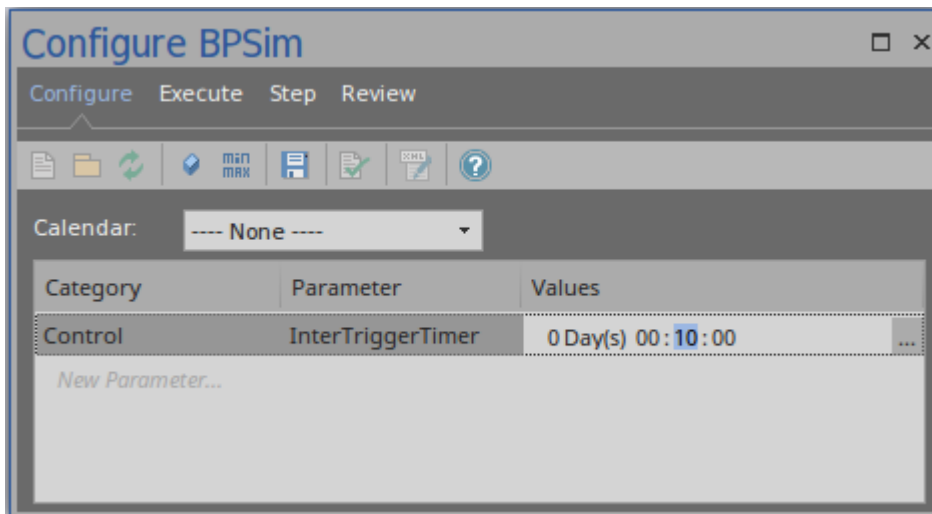
Temporal Parameters
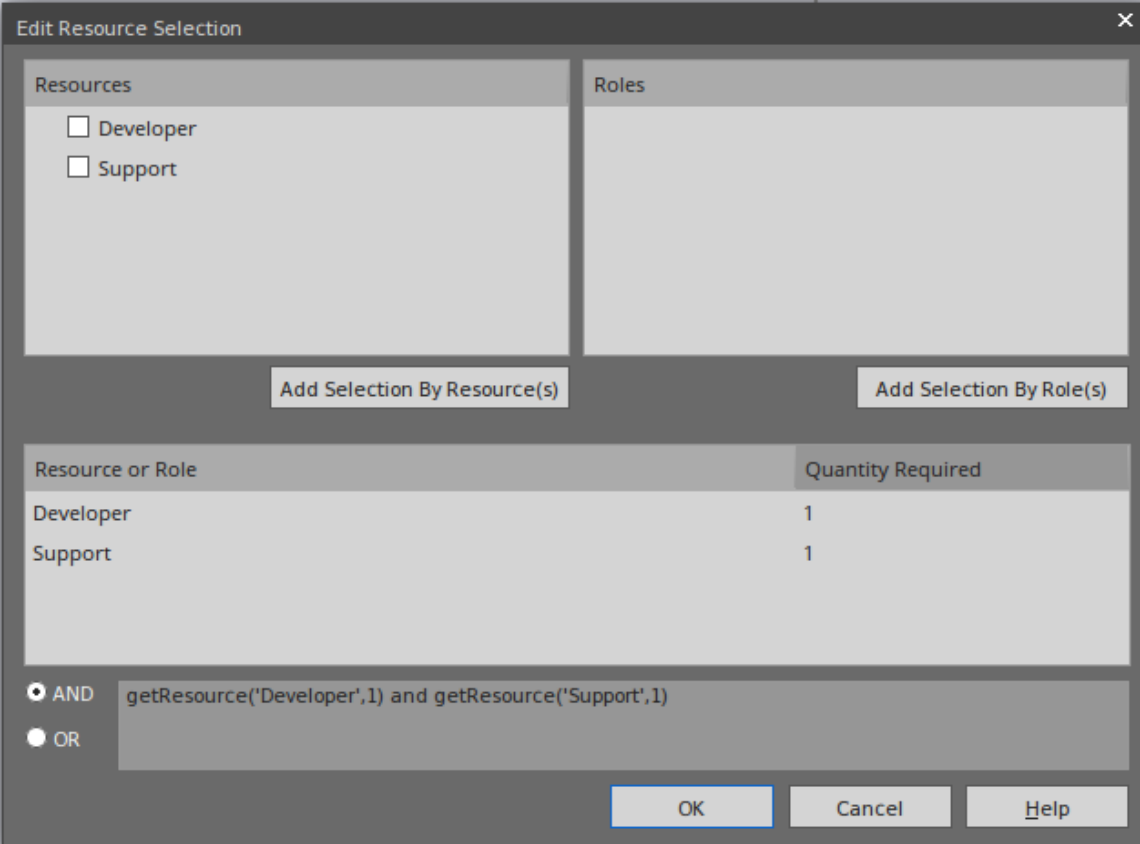
To begin defining Time Parameters:

- Select the element on the diagram, then choose 'Time' from the 'Category' column in the list

- After you select the category, the combo box in the 'Parameter' column will be filled with the available parameters for the selected element

- After you select the parameter, you can enter the value in the field or use the ⟨ ... ⟩ button to open the 'Parameter Value' dialog

- You can toggle the 'Result Request' column by clicking on the min/max button on the toolbar to customize the simulation result by requiring certain results
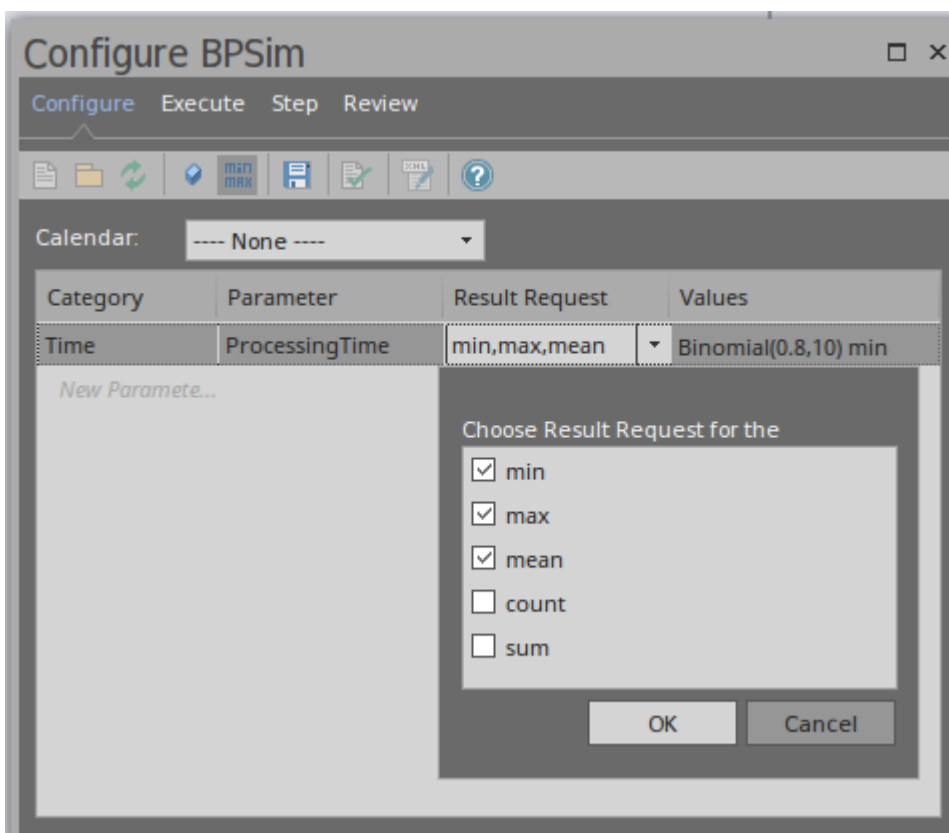
Resource Parameters

To begin defining the Resource Parameters:

- Select the BPMN Task element on the diagram, then choose 'Resource' from the 'Category' column and 'Selection' from the 'Parameter' column

- Click on the [...] button to open the 'Edit Resource Selection' dialog

- The top left list will be filled with the defined Resource elements; use the Add Selection By Resource(s) button to confirm the selection
- The top right list will be filled with the defined Roles on the resource elements; use the Add Selection By Role(s) button to confirm the selection
- Edit the numbers in the 'Quantity Required' column
- Click on the appropriate radio button to set the logical relationship to AND or OR for the selection
- The final expression for Resource selection is composed and displayed in the text field

Property Parameters

To begin defining the Property Parameters:

- Click on the Property button on the toolbar
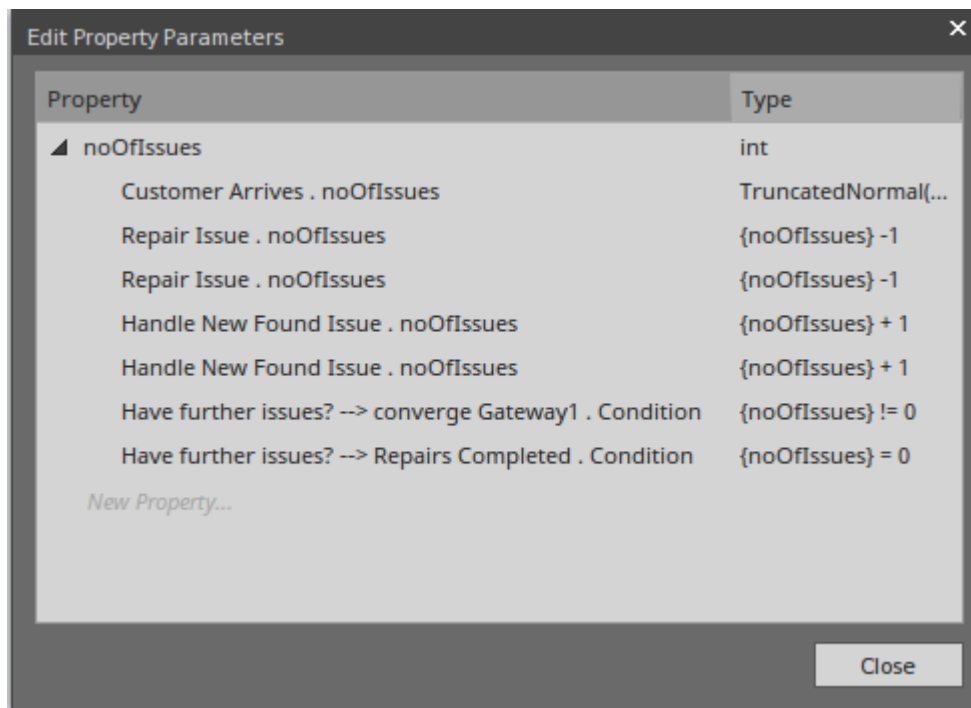
The defined properties and their references are listed.

You can add a new property, delete the selected property, or update a property's name and type.

- With the defined properties, you can set property parameters on BPMN elements:
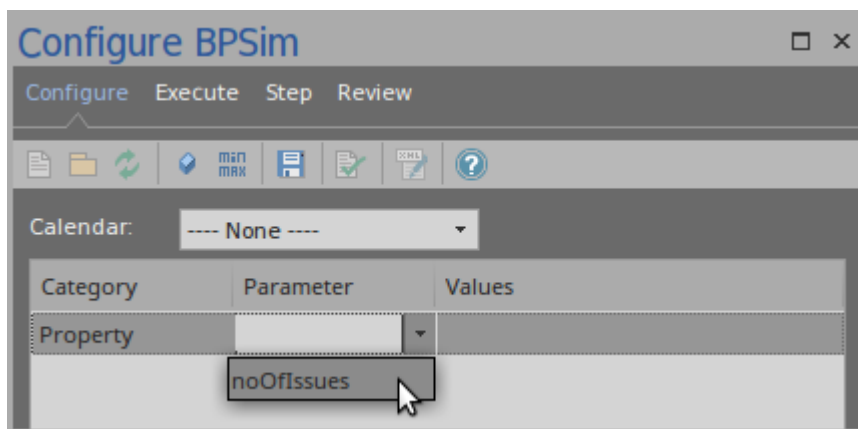
Choose 'Property' as a category and a defined property for 'Parameter'.

- Click on the [...] button on the 'Values' column to display the 'Parameter Value' dialog
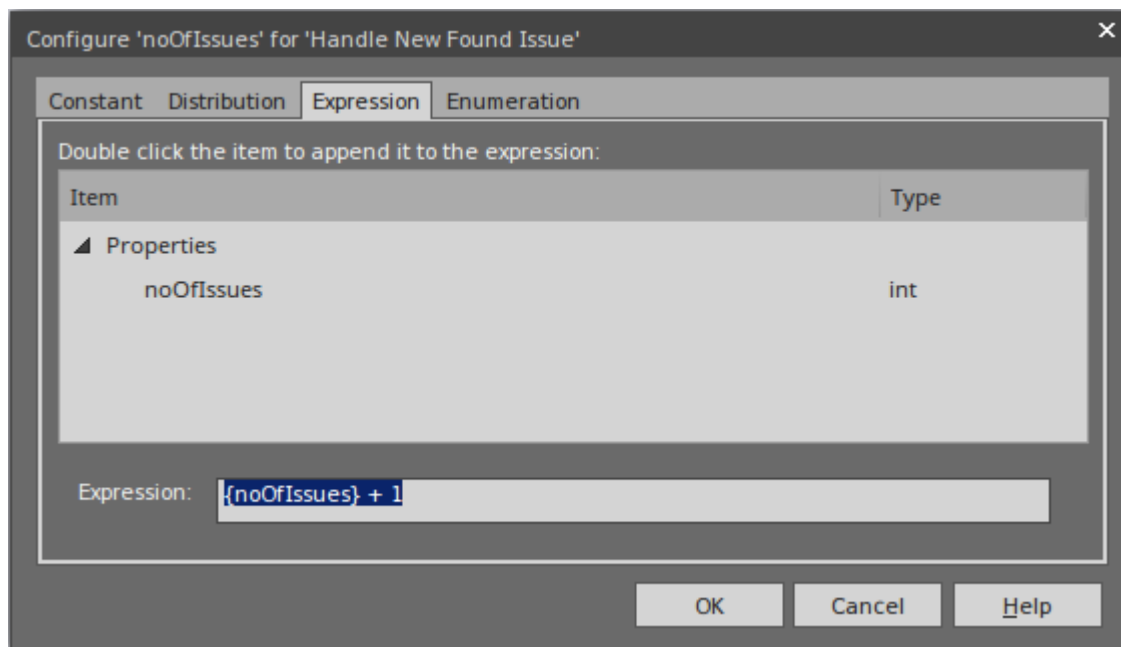
Navigate to the 'Expression' tab; all the defined properties are listed and you can double-click on an item to enclose the property in curly brackets and append it to the edit control. In this example, we compose '{noOfIssues} + 1'

Calendars

With calendars, you can define any number of special periods of time that can influence the process, such as working days, shifts, holidays or periodic events (for example, stock-taking, inventory or auditing).
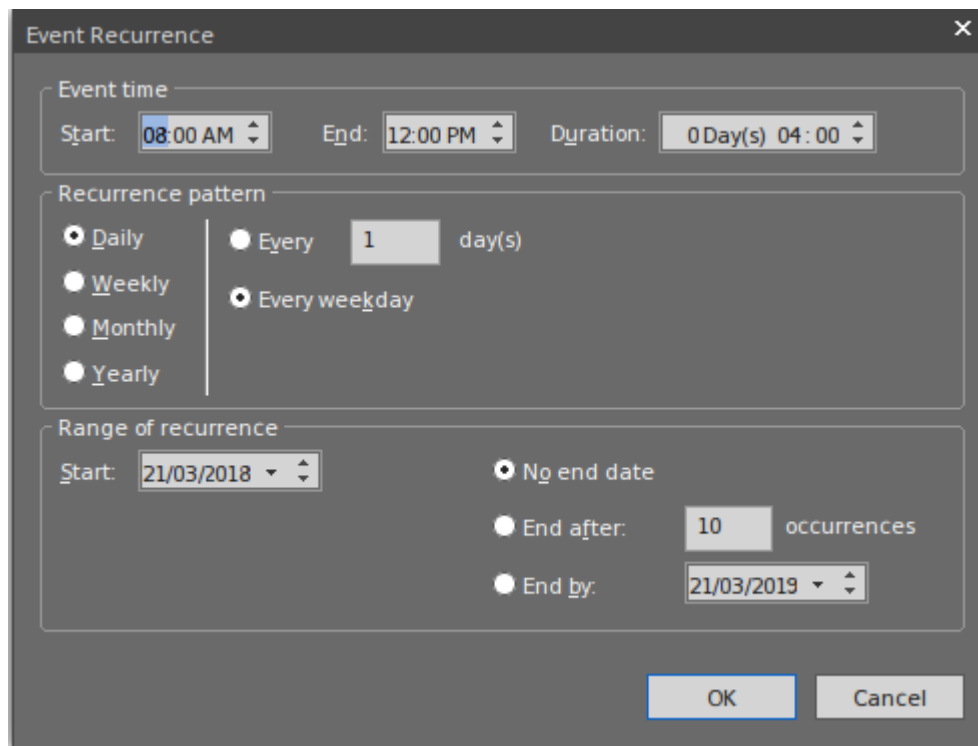
To begin defining the Calendars:

- Click on the Calendar button on the toolbar; the defined calendars will be listed
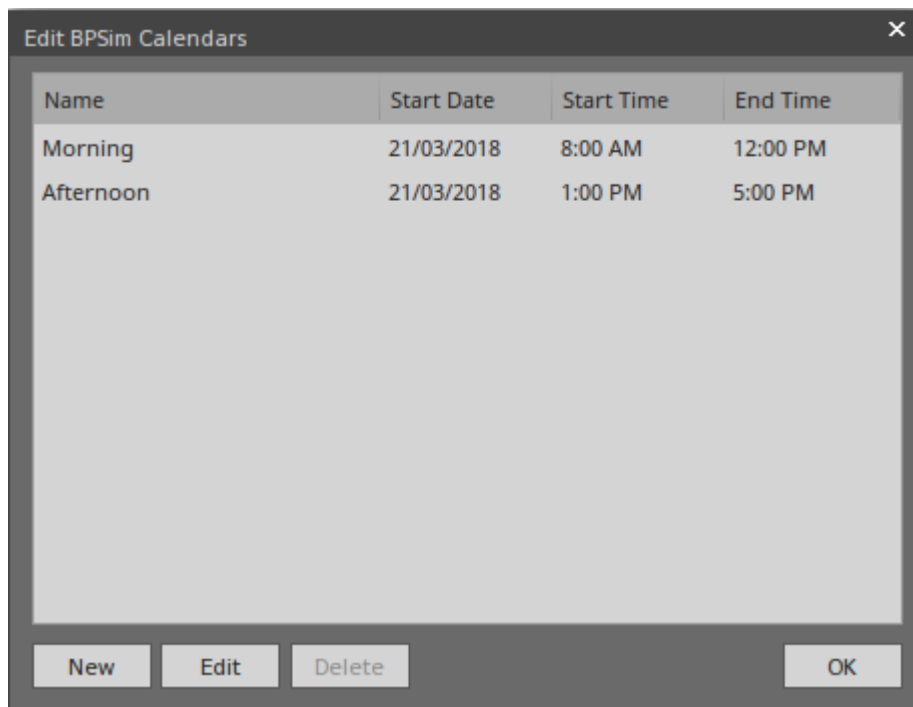
You can add a new calendar, or edit or delete a selected calendar; to add a new calendar period:

1. Click on the New button to display the 'Event Recurrence' dialog.

2. In the 'Event time' panel, the 'Start' and 'End' fields both default to the current time. The 'Start' field is the anchor; a change to either the 'End' field or the 'Duration' field automatically updates the other field, in reference to the 'Start' field. Click on the hour and minute segments of each field (and, for the 'Duration' field, the 'Day(s)' segment) separately, and use the 'spin' arrows to set the start time and the end time or duration of the period.

3. In the 'Recurrence pattern' panel, select the radio button for the interval at which the calendar period recurs. Each option displays an appropriate set of fields in the right of the panel for refining that interval to every day/week/month or every two/three/four days/weeks/months, on a particular day of the week, or day or date of the month, or day or date in the year. Select the checkboxes or values in drop-down lists as appropriate.

4. In the 'Range of recurrence' panel, select the date on which the calendar period takes effect and select the appropriate radio button to define when the period ceases to apply - never, after a set number of occurrences, or on a specific date. You can select an end date either from a drop-down calendar or using the 'spin' arrows on each segment of the date.

5. Click on the OK button to set the calendar period.

As you define calendar periods, they are listed in order of the start date and/or time, earliest first.

- With defined calendars, you can configure parameters on a selected calendar
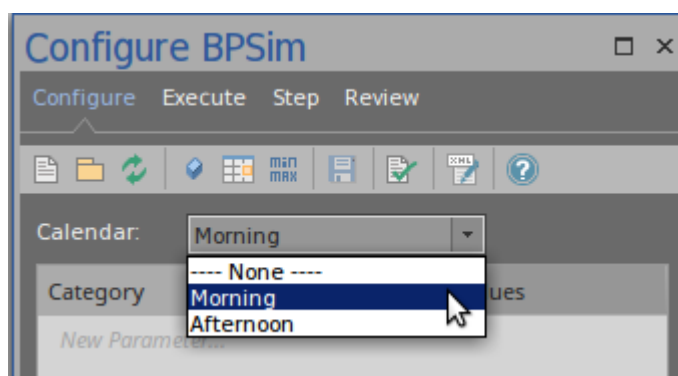
Validation

After configuring BPSim parameters for some BPMN Elements, run the Validation model. Any BPMN or BPSim

errors/warnings will be displayed in the System Output window. Fix the issues according to messages.

Then we are ready for the next topic: BPSim Execute Page.

BPSim - Execute Page

After the configuration, you can choose a Standard simulation or a Customized simulation. The execution will generate a result report and a list of records used to replay (step through) the simulation.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager > Execute page

Toolbar Options

Option	Description
	Click on this button to execute the BPMN file with the BPSim configuration and generate a standard report.
	Click on this button to execute the BPMN file with the BPSim configuration and generate a customized report based on the 'Result Request' settings.
	Click on this button to stop the execution and exit the simulation mode.
	Click on this button open the generated report in the 'Review' page.

Execution

When you click on the Standard / Customized Execute button, the BPMN model with BPSim Configuration will be exported and loaded into the Execution engine.
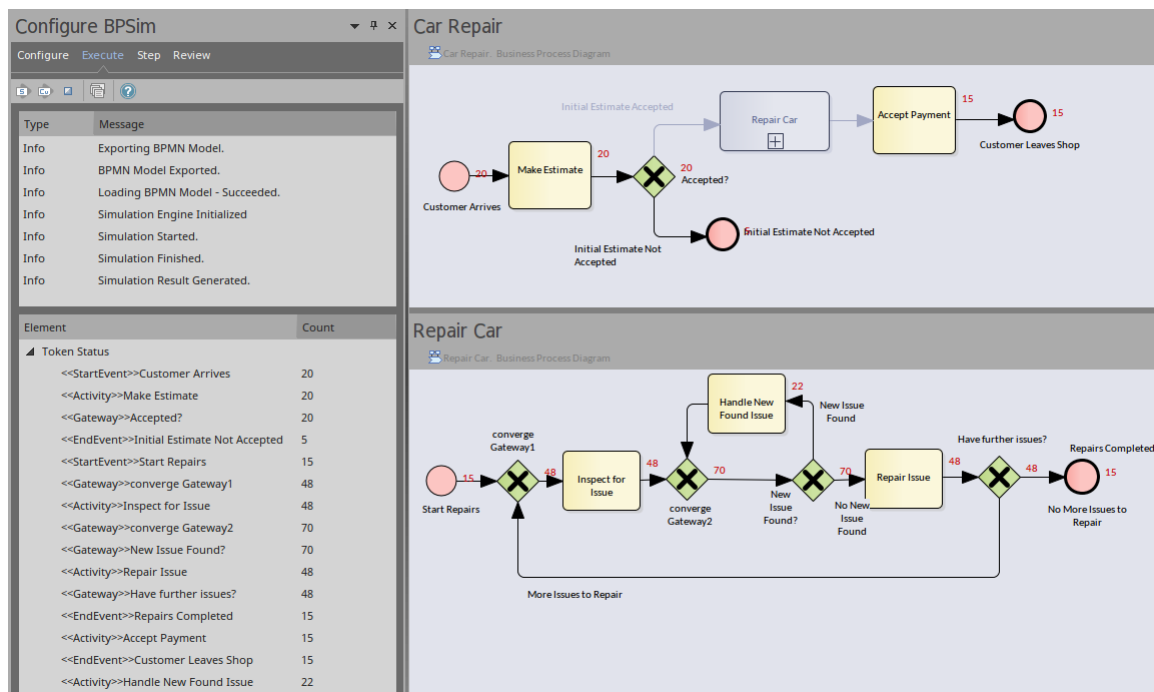
- The token status list will flash with runtime values

- The diagram will flash with runtime token counts

In this example, BPMN elements under the Process 'Car Repair' and sub-process 'Repair Car' get triggered as new customers arrive at regular intervals.

BPSim - Step Page

After successfully running the execution, you get an execution report that tells you the process status in general, such as the average time of a task, the total waiting time of customers and how many issues are repaired.

In addition, you can inspect the process from various angles. For example:

- From the timestamp - what was the status of this process at 9:30 AM?
- From the token - what did the 3rd customer do in the shop?
- From the property - how does the number of issues decrease and increase for the 2nd car?
- From multiple threads - can I see customers walk in and simulate automatically on the diagram?
- From the resources - when is a support busy or idle? Why is a customer waiting for 40 minutes?

All of these kinds of question can be answered on the 'Step Simulation' page.

Access

Ribbon	Simulate >Process Analysis > BPSIM > Open BPSIM Manager > Step tab

Toolbar Options

Option	Description
▷ ▾	Click on this button to simulate the process automatically based on the execution result. Click on the menu option 'Set Speed for Replay' and adjust the simulation speed. For example, input '60' means 60 times faster than actual; 1 minute in real life will be simulated in 1 second.
▋▋	Click on this button to pause the auto-replay simulation.
▢	Click on this button to exit the simulation mode.
⤇	Click on this button to 'Step over' to the next timestamp. Each 'Step over' can contain multiple 'steps'.
⤇	Click on this button to play a single step. This represents a single movement of a token in the process.
▱ ▾	Click on this button to generate a Timing Diagram for the simulation. You can choose from the menu, either 'Generate a single timeline for each token' or 'Generate multiple timelines for each token'.
⛁ ▾	Click on this button to export the filtered records on this step page to a CSV file. You can choose which tab to export.

Tokens Page

After running the execution, this page will be filled with token information during the simulation; the sequence of entries is in order of triggering time.


- 'Apply Filter' in the 'Token ID' column will only show records for a specific token
- One click on the Step in button will play one record in the list
- Double-clicking on a record is equivalent to 'Step to' that record from the beginning
- If time parameters are set on the elements, 'Step over' will run to the last record of the next time event
- When a record in the list is played, the simulation snapshot will show on the diagram

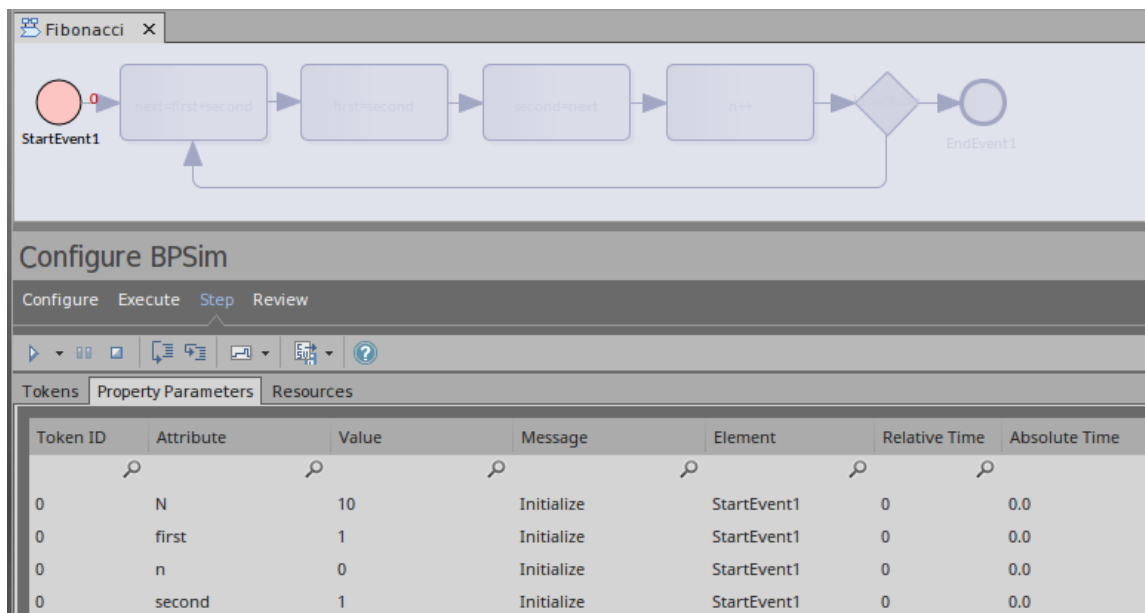## Property Parameters Page

While the records on the 'Tokens' page are played, the 'Property Parameters' page will show the runtime value of properties at the timestamp.
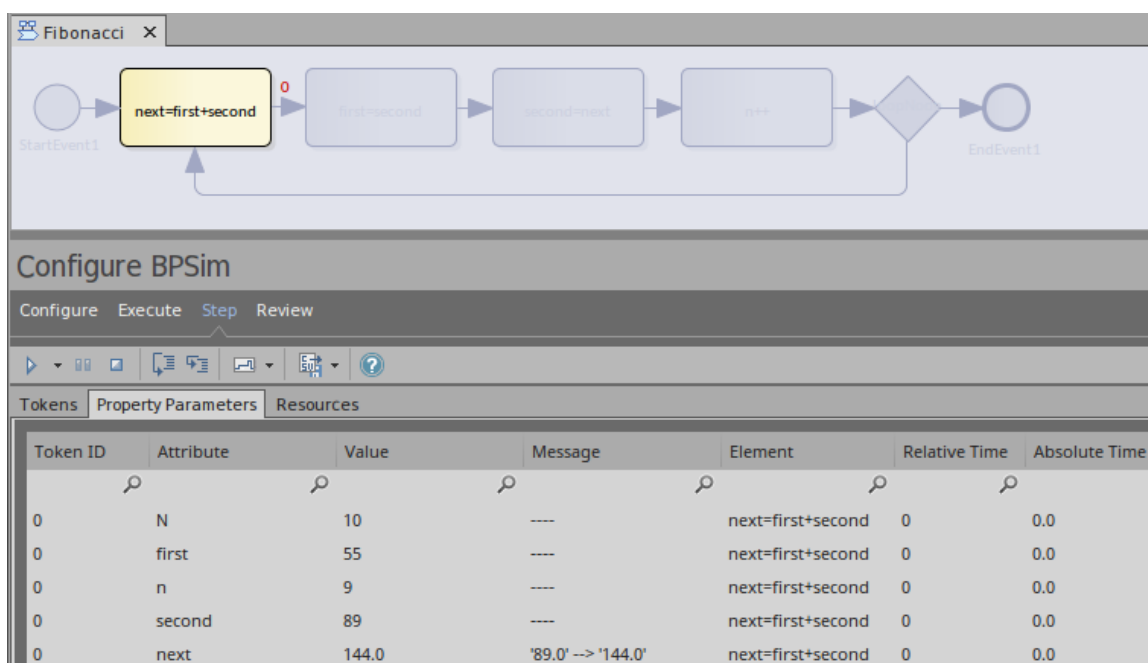
For example, a BPMN process to calculate Fibonacci numbers might be modeled in this way:
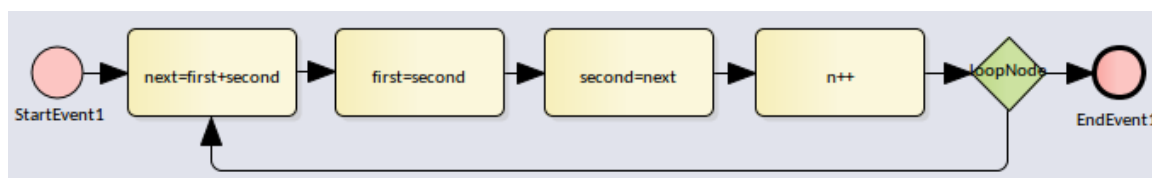
After defining property parameters, configuring BPSim parameters for each element and executing the model, we are ready for the step simulation:

The 'Message' column indicates that properties 'N, first, n, second' are initialized.

Keep clicking on the Step in button; the properties in the list will change their values. This figure shows that on entering task 'next = first + second', the value of property 'next' changes from 89 to 144.

Resources Page

While the records on the 'Tokens' page are played, the 'Resources' page will show the runtime resource available, quantities and allocation/release events at the timestamp.
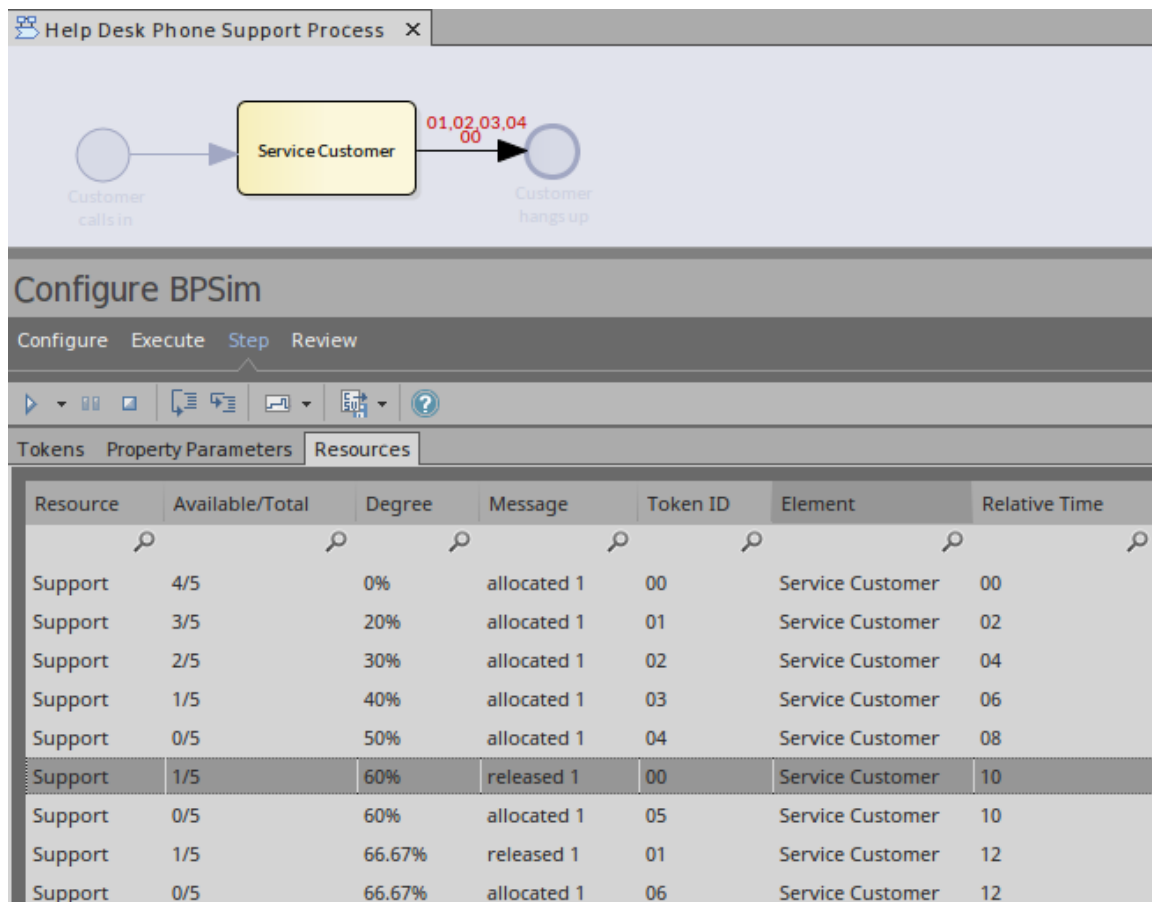
Generate Timing Diagram

When the time parameters are configured on the BPMN elements, Enterprise Architect will be able to generate a Timing diagram for the simulation process.

- Generate a single timeline for each token
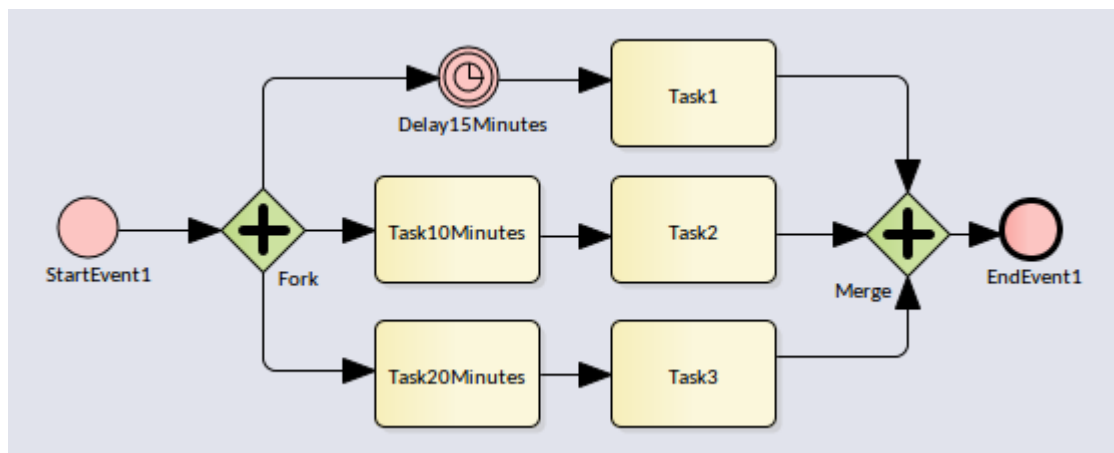
Use this option for a 'single threaded' process; for example, no parallel gateway or event sub processes

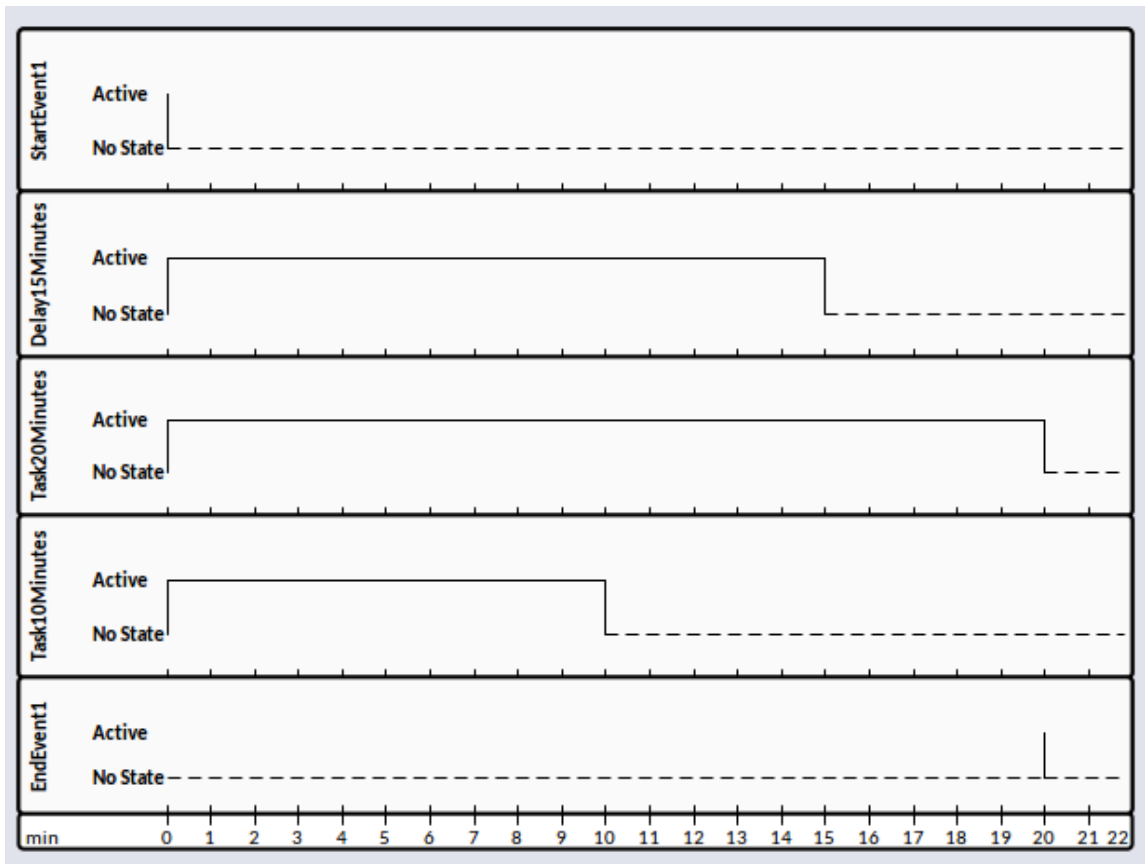- Generate multiple timelines for each token

Use this option in cases when the 'Generate a single timeline for each token' option does not apply.

For example:

Execute this model and click on 'Generate multiple timelines for each token' the generated Timing diagram resembles

this:

BPSim - Review Page

This review page contains three tabs:

- Configuration Summary
- Standard Results Report
- Custom Results Report
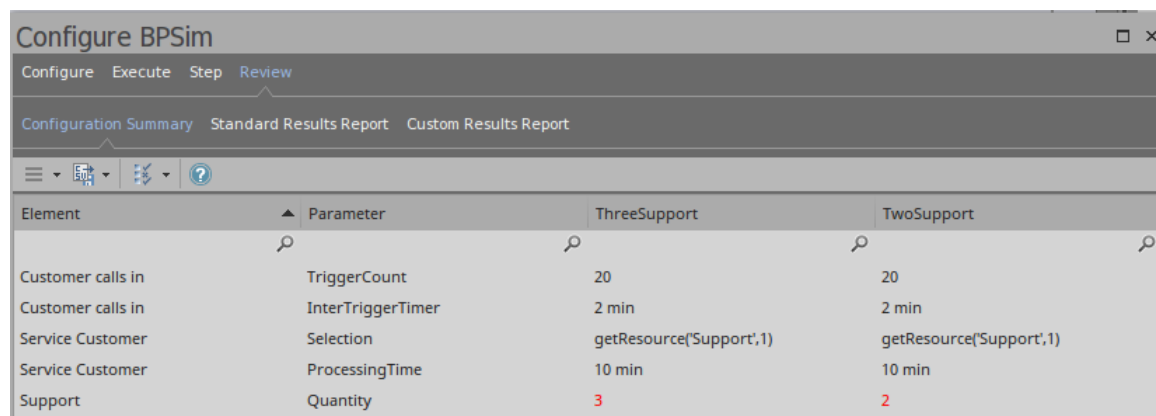
These tabs work in a similar way: add one Artifact for review or multiple Artifacts for comparison. This makes it easy for you to do what-if analyses.

Access

Ribbon	Simulate > Process Analysis > BPSIM > Open BPSIM Manager	Review Page

What-If Analysis

In the Help Desk Support example, we can compare two Artifacts and their corresponding results.

We only show the different parameter values.

Configure BPSim

Configure   Execute   Step   Review

Configuration Summary   Standard Results Report   Custom Results Report

Element	Parameter	ThreeSupport- Result	TwoSupport- Result
Help Desk Phone Support Process	Maximum Time	34	64
Help Desk Phone Support Process	Standard Deviation Time	7.72	17.23
Help Desk Phone Support Process	Average Time	21.4	37
Service Customer	Average Number Of Tokens Waiting For Resource	3.17	5.29
Service Customer	Average Time In Task	21.4	37
Service Customer	Average Time Waiting For Resource	11.4	27
Service Customer	Maximum Number Of Tokens Waiting For Resource	8	12
Service Customer	Maximum Time In Task	34	64
Service Customer	Maximum Time Waiting For Resource	24	54
Service Customer	Total Time In Task	428	740
Service Customer	Total Time Waiting For Resource	228	540
Support	Degree Of Utilisation	92.59%	98.04%
Support	Number Started Immediately	3	2
Support	Total Time Available	216	204
Support	Total Time Idle	16	4
Support	Average Number Available	0.22	0.04

In this example, we see that when the number of support staff decreases from 3 to 2, the average time waiting for resources increases from 11.4 minutes to 27 minutes.

Using the Parameter Value Dialog

The 'Parameter Value' dialog helps you to define values for a wide range of parameters throughout the BPSim Configuration. It supports the definition of simple fixed values through to distributions and expressions that yield a derived value. Not all types of value or derivation are appropriate to all types of parameter.

The dialog name is taken from the object name and the name of the parameter being defined; for example, Configure 'Processing' for 'Activity1'.

Access

With a BPSim Artifact loaded in the Configure BPSim window, select a BPMN Element on the diagram or Project Browser, then click on […] in the 'Values' field. (If the parameter is not already created, choose Category and Parameter in the list to create a new one.)

Constant Tab

Use this tab to define a specific value for the parameter - a numeral, text string or time, for example.
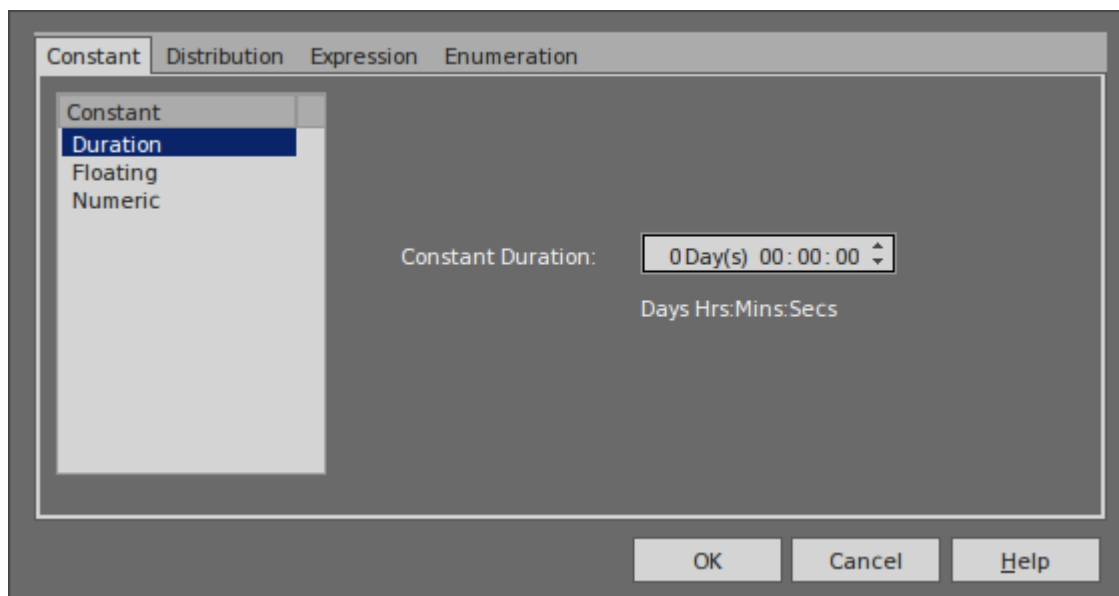
In the 'Constant' panel, select the type of constant:

- Floating
- Numeric
- String
- DateTime
- Boolean, or
- Duration

Appropriate fields display to the right of the panel; type the value and, if required, the unit in which the value is expressed (for example, a unit of time or of currency). For some types of parameter a drop-down list is available from which you can select a value.

Distribution Tab

On this tab, you can apply a statistical sampling method to obtain the parameter value; for each type of distribution available, the appropriate fields display for you to enter the parameters of the distribution. All distributions require you to identify the unit of expression.

The distribution parameters are not necessary for the business process you are developing, but (if you are deriving values from a distribution) are required for the simulation.

You can select from these types of distribution:

- **Beta** - a continuous probability distribution providing 'real' values within a short range, commonly 0 to 1

- **Weibull** - a continuous probability distribution providing 'real' values, commonly used for object lifetime analysis

- **Gamma** - a continuous probability distribution providing 'real' values, useful for modeling exponentially distributed random variables

- **Binomial** - an 'integer' distribution, providing values based on the number of trials and the probability of a certain outcome

- **Erlang** - provides 'real' values based on the Erlang K value and the mean of the distribution

- **Normal** - provides 'real' values based on the mean and standard deviation of the distribution

- **LogNormal** - a continuous probability distribution of 'real', random variables whose logarithm is normally distributed

- **Poisson** - a discrete ('integer') probability distribution that expresses the probability of a given number of events occurring independently in a fixed interval of time or space (volume, distance or area)

- **NegativeExponential** - provides 'real' values based on the mean of the distribution

- **Triangular** - provides 'real' values based on the mode of the distribution and the minimum and maximum values of a range

- **TruncatedNormal** - provides 'real' values based on the mean and standard deviation of points within the minimum and maximum values of a range

- **Uniform** - provides 'real' values between the minimum and maximum values in a range

Expression Tab

On this tab, you type in an XPATH 1.0 expression to combine explicit values, operators and functions to be processed at runtime to provide a value. Each property parameter of an expression must be enclosed in braces - {xxx}.

Example 1: In order to represent c = a + b + 10, we assign this expression to a property 'c':

{a} + {b} + 10

where 'a' and 'b' are properties defined in the BPSim model.

Example 2: In order to represent $c = t - p * (a - b)^2$, we assign this expression to a property 'c':

{t} - {p} * Math.pow({a} - {b}, 2.0)

Note: When simulating a model with this expression, please select 'Java' as the language in order to use the java built-in function Math.pow().

Enumeration Tab

On the 'Enumeration' tab, you can define an enumeration to supply a collection of constant values. You would have obtained these values from real-world, historical data or from analysis and simulation of a model. Every time the parameter is evaluated, the next enumeration value is returned.

As you define each numeration value, click on the Save button to add it to the list of possible values, and click on the New button to clear the data fields ready to enter another value. For some types of enumeration value you might be asked to define the unit in which the value is expressed. The types of enumeration you can define include:

- String
- Floating
- Numeric
- Duration
- DateTime
- Boolean

The BPSim User Interface Components

The BPSim Perspectives have four pages, or tabs, of data in common. These are the 'Scenario', 'Calendars', 'Properties' and 'Review' tabs, the first three of which are also reflected in the data hierarchy on the left of the window. The data specific to each Perspective is discussed in separate topics.

The Save button, Validate Button, Export button and Run button are also available across the configuration, to save all changes made, to validate the configuration, to export the configuration and to execute the Sparx Systems Business Process Simulation engine (if available) on the current configuration.

Because these tabs are in common, you might prefer to populate them with data in a 'base' version of the configuration and use inheritance to make that data available to other configurations. This helps reduce the work in defining the data in the first place, and avoids errors in repeatedly entering the same information in different configurations - you only have to enter it once.

The Scenario Tab

This tab defines the general characteristics of the scenario that the configuration represents, such as:

- The date and time at which the process takes effect; you set and edit this by overtyping the values or, for the date, by selection from a drop-down calendar
- The length of time the process takes; overtype the appropriate segment with the number of years:days:hours:minutes:seconds:milliseconds
- The base unit in which periods of time are expressed in this scenario (unless overridden); click on the drop-down arrow and select the unit
- The currency unit of any costs recorded in the process; click on the drop-down arrow and select the unit abbreviation
- The number of times the scenario as a whole will be repeated in an execution cycle, which you simply type in
- The 'seed' or starting number for any sequence of randomly-generated numbers in the process, which you simply type in
- The date and time on which the scenario was created
- The date and time at which the scenario was last modified
- The JRE Home directory, which you can browse for and select
- The JDK Home directory, which you can browse for and select
- The number of the Port through which Enterprise Architect communicates with the BPSim Execution Engine; this defaults to 1799

The Properties Tab

This tab is used for defining **global** properties; that is, properties of the scenario, or properties that have even broader application, such as 'Number of Issues' or 'Net Cost'.

A number of objects throughout the configuration have their own 'Property' tab, for features specific to that object.

To create a global property

1.   Click on the *New Property* text and type in the name of the property.

2.   Click on the [...] icon in the 'Value' field. The 'Parameter Value' dialog displays, from which you select the appropriate tab to define the value itself or the expression to derive the value. When you have defined the value, click on the OK button.

3.   In the 'Unit' field, if necessary, type the unit of expression of the value you have defined. For example, Hours or $K.

4.   In the 'Calendar' field, if this property can be modified by a defined calendar period, click on the drop-down arrow and select the appropriate period.

Result Requests

In the lower panel of the 'Properties' tab, and of the object parameter tabs throughout the configuration, most of the possible types of parameter appropriate to the tab are listed. For many of these parameters, you can click on the drop-down arrow in the 'Result Request' column and select the type of processing of that parameter you want to check for in the simulation results, such as a count of the number of times it was used, or the sum, mean, minimum or maximum value used, in the context of a specific activity or of the Business Process as a whole. Certain parameters, such as 'Priority' and 'Probability', would not have any meaning in this context and either are not listed or cannot be set in the 'Result Request'.

You would then execute a **customized** simulation, which will only show the parameters you have selected to review.

You can also specify whether the parameter is a 'Key Performance Indicator' (KPI) or part of a 'Service Level Agreement' (SLA), by selecting the appropriate checkbox against the parameter.

The Review Tab

The 'Review' tab presents a summary of the saved changes that have been made to the object parameters and properties in the configuration, during the current work session. You cannot change the data captured on this tab.

BPMN Validation

It is possible to validate your BPMN Model and BPSim Configuration. Click on the Validate button in the configuration to validate the model and configuration; all errors and warnings will be reported in the System Output window.

Alternatively, you can select a root Package containing a BPMN model and BPSim Configuration, and then select the 'Design > Package > Edit > Validation > Validate Current Package' ribbon option.

The BPSim Control Perspective

The Control Perspective targets how activity flows through the process, moderated by the likelihood of a sequence of events and the priorities of certain events when they occur.

When you initially select 'Control' in the 'Perspective' field, the hierarchy on the left of the screen groups all elements in the Business Process (including the Business Process itself) by type and, for each element that has one or more connections to other elements, lists those connections by name and/or by the name of the target element. The Configure BPSim window also initially displays the 'Trigger Count' tab.

Access

Use one of the methods outlined in this table to open the Configure BPSim window and load a Business Process Simulation Artifact element, using the 'Control' perspective.

Ribbon	Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ☰ ▾ > Select Artifact > [select a Business Process Simulation Artifact] > Set 'Perspective' to 'Control' Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ☰ ▾ > [select a Package] > Create Artifact : Add New > [provide a name] > Set 'Perspective' to 'Control' Simulate > BPMN/BPSim > Manage > Find BPSim Configuration Artifacts > double-click an Artifact in search results > Set 'Perspective' to 'Control'
Context Menu	In Project Browser or diagram | Right-click Business Process Simulation Artifact | Configure BPSim | Set 'Perspective' to 'Control'

The Control Tab

The 'Control' tab displays when you click on the name of a Start Event, Intermediate Event or connector in the hierarchy. Through this tab you can define a small number of parameters, dependent on the nature of the object; some objects cannot take control parameters, some can take only certain types, and others a wider range.

To define a control parameter, click on the drop-down arrow after the *New Parameter* text in the 'Type' column and select from the list of available parameters. An object that cannot take parameters has the text *No parameter items* in the 'Type' column.

If you have set any applicable calendar periods, for each parameter you can click on the drop-down arrow in the 'Calendar' field and select the period during which the parameter takes effect. You can set multiple parameters of one type if the value varies according to time or chronological event.

The Trigger Count Tab

The 'Trigger Count' tab lists the elements in the Business Process, grouped by stereotype. Against each element, in the 'Trigger Count' column, is a checkbox that you can select to capture the number of times the element is accessed during a simulation. It provides a quick and simple way to specify the actions to specifically monitor in a customized simulation. You can also use the 'Result Request' column in the 'Properties' or 'Control' tabs to capture the count of accesses for each

element individually (and those settings update and are updated by the 'Trigger Count' tab), but the tab helps you set or clear the count requests in one place. You do need to use the 'Result Request' column to capture the number of times a pathway (connector) *between* elements is accessed.

On the 'Trigger Count' tab you can, if you prefer, use the arrows in the column heads and/or the Filter bar to list the elements in a different sequence by Stereotype, Name or selected/cleared 'Trigger Count' checkbox.

Control Parameters

Parameter	Description
Condition	This parameter simply defines some type of condition that must be satisfied before processing continues. It can be either a simple constant, a statistically-derived value, a value calculated from an expression, or an enumerated value. In the 'Value' field, click on the [...] button and work on the 'Value' dialog.
Probability	This parameter assigns a weighting to the object or connector so that when there is a choice of paths, the simulation will process that object or connector a fixed proportion of the times. Probability is expressed as a decimal portion of 1.0, so in the 'Value' field you type a value such as 0.25 (25% of the time, the simulation will process this object). You should check that the probabilities of all paths in the same choice add up to 1.0.
InterTriggerTimer	This parameter defines the time interval between triggers for the action that the object or connector represents. This interval can be either a simple constant, a statistically-derived value, a value calculated from an expression, or an enumerated value. In the 'Value' field, click on the [...] button and work on the value dialog as for a global property.
TriggerCount	The TriggerCount sets the number of times the object or connector should be processed during one execution of the simulation, so that the processing cycle imitates the typical demand for a particular chain of actions. It is generally set for Start Event elements.
	Note that this TriggerCount **sets** the number of times the element or connector **is to be** processed in a normal or customized simulation, whilst the TriggerCount in the 'Trigger Count' tab and 'Result Requests' **gets** the number of times the element or connector **has been** processed in a customized simulation.

The Priority Tab

The 'Priority' tab displays when you select an Activity element. The tab identifies whether the activity can be interrupted by another activity, and what the relative priority is of the activity if there is a choice as to which activity is executed first.

Parameter	Description
Interruptable	This parameter defines whether the activity can be interrupted, and takes a value of True or False. You can create more than one instance with different values for different calendar periods.
Priority	This parameter defines the priority of the activity relative to other activities that

	might be executed at the same time. In the 'Value' field, click on the [...] button and work on the value dialog to set either a simple constant or a derived value for the priority. Take into consideration whether the competing activities are interruptable and what their priority values are.
	You can set different priorities for the same activity, corresponding to different calendar periods.

The BPSim Resource Perspective

The 'Resource Perspective' targets the involvement of types and roles of workers and other resources in a process, reviewing the required numbers, combinations, costs and availability. Before you define how the resources can be deployed in the process, you must create the appropriate BPMN 2.0 Resource elements within your model. These are then listed in the 'Resource' folder in the Configure BPSim window hierarchy. You can also use ResourceRoles that you might have defined in your model, but it is quicker and easier to create and use any roles you require within the BPSim configuration.

Access

Use one of the methods outlined in this table to open the Configure BPSim window and load a Business Process Simulation Artifact element, using the 'Resource' perspective.

Ribbon	Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ≡ ▾ > Select Artifact > [select a Business Process Simulation Artifact] > Set 'Perspective' to 'Resource' Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ≡ ▾ > [select a Package] > Create Artifact : Add New > [provide a name] > Set 'Perspective' to 'Resource' Simulate > BPMN/BPSim > Manage > Find BPSim Configuration Artifacts > double-click an Artifact in search results > Set 'Perspective' to 'Resource'
Context Menu	In Project Browser or diagram | Right-click Business Process Simulation Artifact | Configure BPSim | Set 'Perspective' to 'Resource'

The Resource Tab

The 'Resource' tab is available when you click on a Resource element in the 'Resource' folder. On this tab, you define how many instances of each resource are available for each calendar period that you have defined, and any roles the resource might be specialized into. You can provide multiple instances of each resource parameter, depending on how the resource availability varies across any defined calendar periods.

When you have finished defining the resource availability, click on the Save button. The 'Resource' tab has no 'Result Request' facility.

Parameter	Description
Availability	In the 'Type' column, select this parameter. In the 'Value' column, click on the drop-down arrow and select either 'True' (the resource is available) or 'False' (this resource is not available). You can also define mechanisms for establishing availability, using the 'Parameter Value' dialog, as long as they result in a value of True or False. In the 'Calendar' column, click on the drop-down arrow and select the calendar period to which this availability applies.
Quantity	In the 'Type' column, click on the drop-down arrow and select this parameter.

	In the 'Value' column, either type in a simple numerical value or click on the drop-down arrow and use the 'Parameter Value' dialog to define a constant or derived value for the quantity of this resource that is available for assignment. In the 'Calendar' column, click on the drop-down arrow and select the calendar period during which this quantity of the resource is available.
Role	To add a role to the panel, click on the *New Role* text and type in the role name. At this point the new roles remain as parameters within the BPSim configuration and are not reproduced in the model.

The Cost Tab

The 'Cost' tab is available when you select an Activity element or a Resource element in the Configure BPSim window hierarchy. To add a parameter, click on the drop-down arrow after the 'New Parameter' text in the 'Type' column. You can have multiple costs if the cost varies according to the calendar period in which it is incurred.

Parameter	Description
Fixed Cost	This parameter defines the fixed cost of calling the resource or activity into use, automatically expressed in the unit of currency defined on the 'Scenario' tab. For example, the call-out fee for an electrician. In the 'Value' column, either type in the cost or click on the drop-down arrow and define how the cost is derived. A derived cost must resolve to a floating or numeric value. If necessary, in the 'Calendar' column click on the drop-down arrow and select the calendar period during which the fixed cost applies.
Unit Cost	This parameter defines the unit cost of using the resource or running the activity, automatically expressed in the unit of currency and per the unit of time defined on the 'Scenario' tab. For example, the hourly rate of the electrician. In the 'Value' column, either type in the cost or click on the drop-down arrow and define how the cost is derived. A derived cost must resolve to a floating or numeric value. If necessary, in the 'Calendar' column click on the drop-down arrow and select the calendar period during which the unit cost applies.

The ResourceRole Tab

The 'ResourceRole' tab is available when you select an Activity element or Business Process element in the 'BPSim Configuration' hierarchy. Its main purpose is to define the resources for an Activity, but if necessary you can assign resources that act within the Business Process definition itself.

You can apply the resources as defined within the model. Alternatively, you can define a combination of resources and roles that is not identified in the model, which overrides the original definition. For example, the model might show that an Activity 'Take Customer Call' is associated with the resource 'Front Office'. In the BPSim configuration you can qualify that relationship to indicate that the Activity actually uses 50 Front Office staff. But you might instead override the model relationship to indicate that the work can be done by 50 resources from either Front Office or Support, if (for example) the Support resources have the role of Switchboard Operators.

The resource selection for the Activity is also displayed as an expression, at the bottom of the tab.

Panel	Description
Resource Roles defined in BPMN	In this panel, you assign a resource to the selected Activity in the BPMN model, as shown: 1. Click on the Add button, which adds a ResourceRole element to the 'Type' column and to the Activity in the model. 2. Click on the 'BPMN Resource' column and on the [...] button, and select a Resource from the Tagged Values browser (which assigns the resource to the 'ResourceRef' Tagged Value of the new ResourceRole element). 3. If necessary, type a role for the resource in the 'Role(Name)' column (preferably using the roles you defined for the resource on the 'Resource' tab). This value is reflected in the ResourceRole element defined for this Activity. 4. In the 'Quantity' column, the quantity of this resource that the Activity uses on each iteration defaults to 1. If necessary, overtype this number with a different value. 5. Click on the Save button. Note that the quantity is reflected in the 'Advanced Override for Single Resource Role' panel. 6. If you need to add another type of resource, or a different role of the same resource, you can repeat steps 1 to 5 **or** (preferred) use the Advanced Override. Note that the quantity reflected in the 'Advanced Override for Single Resource Role' panel is for one selected ResourceRole entry.
Advanced Override for Single Resource Role	In this panel, you can define a *combination* of resources to use in place of a relationship defined in the model. All resources currently defined in the model are automatically listed in the panel. The override applies to a single selected ResourceRole entry in the 'Resource Roles defined in BPMN' panel; if you have more than one entry in this panel, set the 'Quantity' value for the other entries to 0. 1. Select the radio button to indicate whether the combination of resources you are defining is an AND relationship (so many of Resource X and so many of Resource Y at the same time), or an OR relationship (0 to n of Resource X OR 0 to n of Resource Y at any time). 2. Double-click in the 'Quantity Required' column and type the required number of each resource for the activity. 3. Click on the Save button. Note that the 'Resource Selection Expression for Activity' panel at the foot of the tab shows the expression that will apply this combination of resources. If a resource is defined in the 'Resource Roles defined in BPMN' panel, a change to the quantity of that resource is reflected in that panel, but the addition of a subsequent resource in the 'Advanced Override for Single Resource Role' panel changes the value of the 'Quantity' field to 'Override'. If you want to make the **role** of a resource significant in the combination, you must have at least one resource identified in the upper panel. This adds a 'Role' item to the end of the list in the 'Advanced Override for Single Resource Role' panel. 1. Click on the 'Role' item and on the [...] icon in the 'Resource or Role' column. The 'Roles for Selection' dialog displays, listing the roles available for each resource in the configuration (as you defined on the 'Resource' tab, previously). 2. Select the checkbox against the required role (or more than one role, if appropriate) and click on the Save button. The roles are shown in the 'Resource or Role' column. 3. In the 'Quantity Required' column, type in the required number of resources of this role or these roles. 4. Click on the Save button. The 'Role' item moves underneath the appropriate resource name, and another empty 'Role' item is added to the end of the list.

| | The 'Resource Selection Expression for Activity' panel is updated to show the resource roles you have specified. |
| | If you do not want to keep a resource or role in the defined combination, set its 'Quantity Required' value to 0. It will remain at this value while you remain on this Activity, but when you return to the Activity after working elsewhere the field is clear. |

The BPSim Time Perspective

The 'Time' (or 'Temporal') Perspective helps you to examine how the duration of one or more phases in the processing of an Activity influences the business process. In this Perspective you only set parameters on Activity elements, but you can monitor (through the 'Result Request' column) the occurrence, minimum, maximum, mean and sum duration of a parameter for the Business Process as a whole or for the Activities involving a defined Resource.

Access

Use one of the methods outlined in this table to open the Configure BPSim window and load a Business Process Simulation Artifact element, using the 'Temporal' perspective.

Ribbon	Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ☰ ▾ > Select Artifact > [select a Business Process Simulation Artifact] > Set 'Perspective' to 'Temporal' Simulate > BPMN/BPSim > Manage > Open BPSim Manager > click toolbar icon ☰ ▾ > [select a Package] > Create Artifact : Add New > [provide a name] > Set 'Perspective' to 'Temporal' Simulate > BMPN/BPSim > Manage > Find BPSim Configuration Artifacts > double-click an Artifact in search results > Set 'Perspective' to 'Temporal'
Context Menu	In Project Browser or diagram | Right-click Business Process Simulation Artifact | Configure BPSim | Set 'Perspective' to 'Temporal'

The Time Tab

On this tab, for a selected Activity element, you define the duration of a number of stages through which the Activity might pass. For each stage you:

- Click on the drop-down arrow in the 'Type' field and select the type as a parameter

- Click on the ⌞...⌟ button in the 'Value' field and define the duration (either as a fixed value or as a derived value) and unit of measurement, and

- If appropriate, click on the drop-down arrow in the 'Calendar' field and select a calendar period to which this duration applies

Parameter	Description
Processing Time	The time spent actually performing the Activity. This can be strictly defined, or more broadly set to represent the duration of the Activity (the duration being the time it takes to complete the Activity; in strict use, the sum of the 'Setup Time', 'Processing Time', 'Validation Time' and 'Rework Time'). For this and the other six parameters, you use the 'Parameter Value' dialog to define a constant or derived value in the appropriate unit of time; a derived value must resolve to a NumericParameter, a FloatingParameter or a DurationParameter.
Queue Time	The delay between the Activity being offered and the Activity being processed; a

	refinement of the lag time.
Rework Time	The time spent redoing or correcting the work of the Activity; a refinement of the duration.
Setup Time	The time expended between the element being offered and the element being allocated to perform the actual work; a refinement of the duration.
Transfer Time	The time spent traveling from the previous processing step; a refinement of the lag time.
Validation Time	The time taken to review or inspect the work done; a refinement of the duration.
Wait Time	The time between the element being allocated and its action actually starting. This can be strictly defined, or more broadly used to represent the lag time of the Activity (the lag time being the time between the action of one Activity stopping and the action of the next Activity starting; in strict use the sum of the 'Transfer Time', 'Queue Time' and 'Wait Time').

Notes

- In the BPSim configuration you can set values for seven different time parameters; however, the BPSim Simulation engine, during a simulation, amalgamates the values of these parameters into one ProcessingTime parameter
  You can vary the initial values and value generation methods of the individual parameters to observe the effect each has on the eventual processing time, or to define a simulation to be exported into a different tool for analysis and/or execution
- An example of the behavior of the Time parameters during simulation is provided in the EAExample model

Configuration Inheritance

When you define a BPSim configuration it can contain a lot of data, some relating to each of the three separate Perspectives, and some being base data for the configuration grouped on three pages or tabs of the 'Configuration' dialog. A major objective of performing simulations of a business process using BPSim is to observe the effects of varying the values of certain parameters, whilst leaving others unchanged. It would be very useful, then, to maintain all of the standard data in a single place where it is created only once, and work on the variable data in one or more separate places that refer back to the base data. This would also mean that if you changed any of the standard data, the changes would be reflected in all the configurations that refer to it.

In BPSim, you can achieve this arrangement using inheritance between configuration Artifact elements.

In essence, you would define the standard data in a base configuration, and the data you are manipulating (and only that data) in configurations called, say, Resource Scenario 1, Resource Scenario 2, Timing Scenario A, or Control Scenario B. You would then create Generalize connectors from the scenario configuration Artifacts (the sources) to the base configuration Artifact (the target).
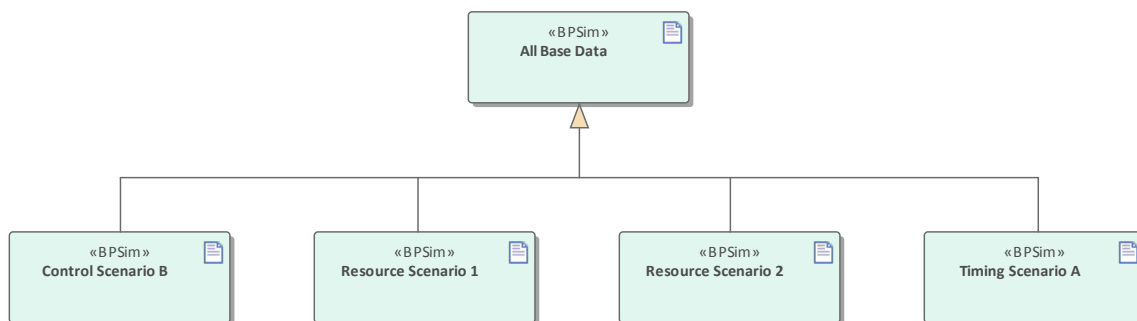
When you run a simulation on a scenario configuration, it will inherit the data defined in the base configuration and the simulation will perform as if it were processing a single configuration containing all the data. If the scenario configuration contains data that is also defined in the base, the (changed) values in the scenario take precedence.

Comparisons

To keep track of how the base and scenario configurations have been set up, and what the differences are between them, you would run a 'BPMN Simulation Configuration' report on one of them and drag the other(s) into the same report.

Whether or not you select the 'Show Only Different Items' option or the 'Highlight Different Items' option, you will see all the base configuration data in one column and a much smaller volume of scenario data in another. Where the data occurs in both columns, you can compare values and check what the scenario values have been changed to.

Similarly, when you run simulations on both configurations, you can display the 'BPMN Simulation Report View' and compare the results of the simulations. In this case if you select either the 'Show Only Different Items' or the 'Highlight Different Items' option, you will see a much smaller data set where the scenario values differ from the base configuration values, because in the **simulation** the scenario will have inherited all of the base configuration data and will produce substantially the same output.

Using the MDG BPSim Execution Engine

The MDG BPSim Execution Engine is an Add-In that you can purchase to execute the simulations that you have defined using the Business Process Simulation (BPSim) facility in Enterprise Architect. Installation of the Engine is a prerequisite for accessing and using the facilities of BPSim.

Access

Open the Configure BPSim window, load a Business Process Simulation Artifact and click on ▶ on the toolbar.

Ribbon	Simulate > BPMN/BPSim > Manage > Open BPSim Manager : click ≡ ▾ on the toolbar > Select Artifact > [select a Business Process Simulation Artifact] : click on ▶
	Simulate > BPMN/BPSim > Manage > Open BPSim Manager : click ≡ ▾ on the toolbar > [select a Package] > Create Artifact : Add New > [provide a name] : click on ▶
	Simulate > BPMN/BPSim > Manage > Find BPSim Configuration Artifacts > double-click an Artifact in search results : click on ▶
Context Menu	Right-click on a Business Process Simulation Artifact element | Configure BPSim : ▶
Other	Right-click on a Business Process Simulation Artifact element | Simulate BPMN with BPSim

Execute and Control a Simulation

Click on the Run button and select either:

- 'Standard Simulation'   or
- 'Customized Simulation'

These options trigger the same processing, except that while 'Standard Simulation' generates a report on all built-in parameters set in the simulation, 'Customized Simulation' extracts results for only the parameters you have specifically flagged using the 'Result Request' columns in the configuration.

The simulation executes, displaying processing messages in the top section of the dialog, and the elements and parameters processed with the runtime values used from the configuration.

While the simulation is in progress, you can use the Pause button and Resume button to temporarily halt processing and then resume, or you can click on the Stop button to cancel the simulation altogether.

The results of the simulation are written to an Artifact element added to the Business Process parent Package. A Standard simulation writes to a <<BPSimReport>>-stereotyped Artifact, whilst a Customized simulation writes to a <<BPSimCustomReport>>-stereotyped Artifact.

Track Property Values

As well as the built-in parameters, you can define your own process-specific Property parameters (attributes) in the configuration. When the simulation has finished, and if you have defined property parameters, the Attributes button is enabled. When you click on this button the 'BPSim PropertyParameter Values' dialog displays, through which you can track how the run-time values of all the property parameters accrue or change through the business process.

Review a Simulation

When the simulation has finished processing, click on the Open Result button. The 'BPMN Simulation Report View' tab opens in the main work area, showing the results for the built-in parameters in the current simulation (but not for the user-defined property parameters). If you have already run a simulation of another configuration based on the same business process, that is also displayed in the report as an additional column. Otherwise, you can click on the report Artifact element and drag it onto the report tab, to compare the runtime values of the built-in parameters under two (or more) configurations.

To make it easier to view the data from the report, you can drag the 'BPMN Simulation Report View' tab out of the main view so that it becomes a floating window, and enlarge the window to a suitable size.

Click on the expansion boxes against the parameters you want to check. You can also expose and filter the information using right-click context menu options.

You can represent specific differences between the results from separate simulations as Charts. The simulation result Artifacts (<<BPSimReport>>*name* - Result elements) must exist before you can set up the Chart Artifacts. There is a template Chart Artifact for Standard simulations and one for Customized simulations.

BPMN Simulation Report Options

Option	Description
Collapse All	Select this option to collapse the parameter hierarchy down to just the parent tab names.
Expand All	Select this option to expand the parameter hierarchy out to the lowest value type.
Show Only Different items	(When you have two or more simulations shown.) Select this option to restrict the display to those parameters where the values differ between the simulations. Click on the option again to deselect it.
Highlight Different Items	(When you have two or more simulations shown and where some of their parameter values are different.) Displays the differing parameter values in red. This option is disabled if you select the 'Show Only Different Items' option.
Show Only Non Empty Items	Select this option to filter the display to show only parameters that have a specific value other than 0.
Remove Model	(When you have selected a specific result, which identifies the simulation in the report.) Select this option to remove the simulation column from the report.

BPSim Execution Engine - Simulation language

The MDG BPSim Execution Engine supports simulation on XPath 1.0 or Java, where the appropriate language is defined as the Expression Language in the simulation configuration. It also supports the use of process instance data in BPSim Property Parameters, where the actual value is only determined during execution.

XPath 1.0 Operators

These operators can be used in BPSim expression parameters.

Operator	Description
|	The Union operator, used for resource acquisition. Example: getResource('w1',1) | getResource('w2',1)
+	Addition. Example: 4 + 6
-	Subtraction. Example: 6 - 4
*	Multiplication. Example: 6 * 4
div	Division. Example: 8 div 4
=	Equality. Example: 4 = 4 (True)
!=	Not Equal. Example: 5 != 3
<	Less than. Example: 6 < 9
<=	Less than or equal to. Example: x <= 6
>	Greater than. Example: 9 > 6
>=	Greater than or equal to. Example: n >= 7
or	Alternative. Example: n = 6 or n <= 6

and	Combination.
	Example: n = 5 and m < 8
mod	Modulus division.
	Example: 5 mod 2
getProperty	Get a property value.
	Example: getProperty ("amount")
getResource	Get a resource assignment.
	Example: getResource ('w1',1)

Note

The Expression Language can be set in the Configure BPSim window, on the 'Scenario' tab; the two options 'XPath 1.0' and 'Java' are available.

If you select 'Java', you must set the property 'JDK Home' to a valid JDK directory.

Tip: You can use {PropertyParameterName} as a short form of getProperty('PropertyParameterName'), which is useful when writing the value for the expressions; for example:

   {n} < {N}   instead of   getProperty('n') < getProperty('N')

The short form of the getProperty operator can be used in both XPath 1.0 and Java.
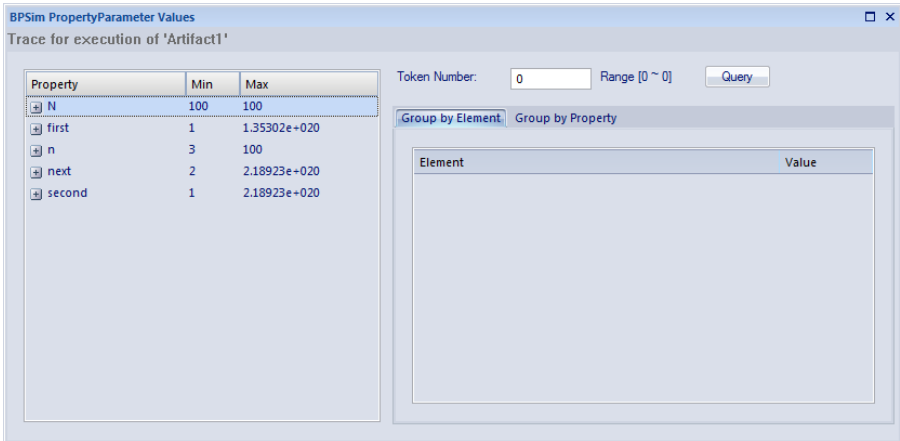
BPSim Property Parameters

In Enterprise Architect release 13.0 and higher, BPSim property parameters can hold process instance data to which no value is assigned until run time. You can define the property parameter type on the 'Properties' page of the Configure BPSim window; the supported types are 'int', 'double' and 'string'.
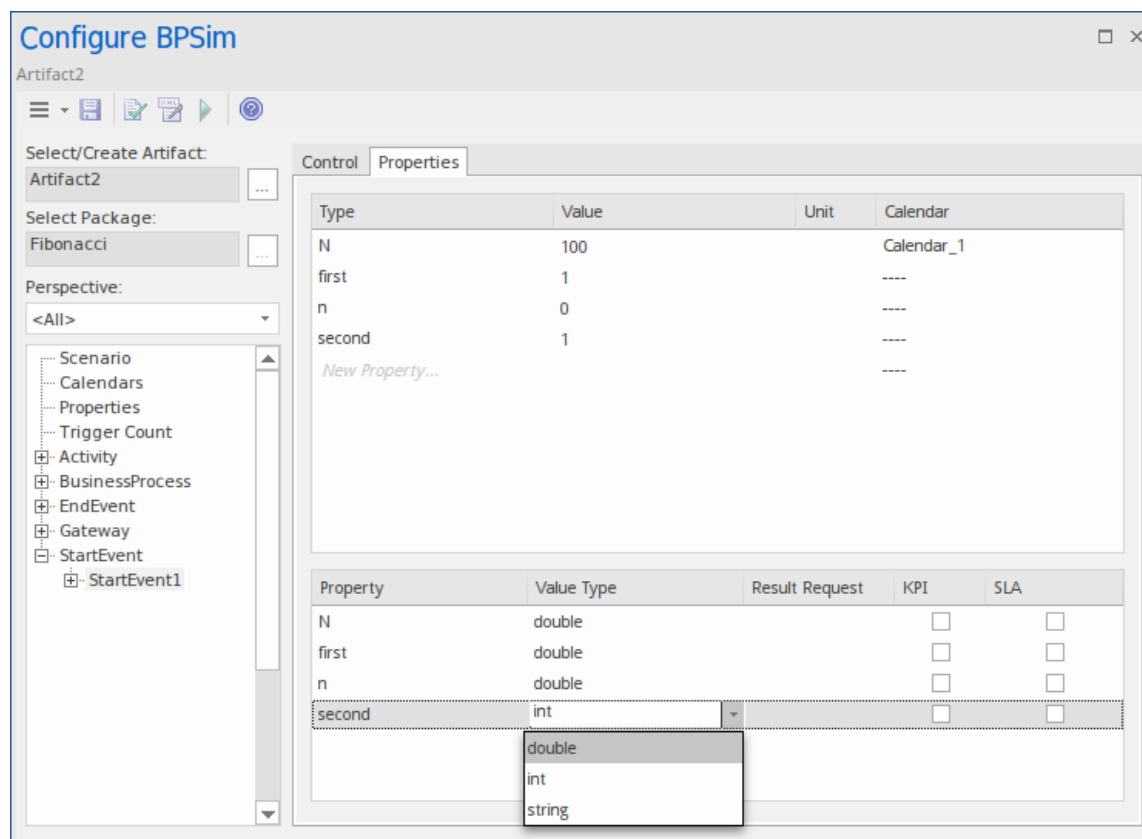
Tracking Property Parameter Values

The Business Process Simulator (BPSim) helps you to model and test the operating details of a business process, such as the resourcing of activities and tasks, the intervention of events, and the impact of decision points and the decisions made at those points. You add these process-specific property parameters, or attributes, to the BPSim configuration and, as you run simulations on the model according to the configuration, the BPSim engine helps you to capture the run-time values of the property parameters for every iteration of the simulation, and to filter the results to examine specific pathways or decision points. This gives you an incredibly detailed insight into what might actually happen in your business process under a specific condition or combination of conditions, either to generate a result or to show what processing path produces that result.

Access

Context Menu	Right-click on a defined Business Process Simulation configuration Artifact |Simulate BPMN with BPSim... : Run (select simulation type) : Attributes
	(The Attributes button is not available if the simulation configuration does not contain any property parameters)

BPSim PropertyParameter Values dialog fields

Option	Action
Property	This table lists the properties defined for the process, and shows the minimum and maximum possible values for each property for the whole process.
	If you click on the expansion box for a property, the table shows the minimum and maximum values for the property at each activity or event (element) during the process.
Token Number	Type the number of the 'token' to examine; this number must lie within the range shown to the right of the field. A 'token' is an independent trigger, such as a customer or an order entering a business and initiating the business process under review. There can be any number of customers or orders each having no relationship with any other customer or order, and each potentially following a

	different route through the business process. Where there is only one possible instance of one possible trigger event, such as an on/off switch being thrown, the token is regarded as being 0.
Query	Click on this button to initiate a query on the token simulation, to populate the 'Group by Element' and 'Group by Property' tabs.
Group by Element	Displays the results from the perspective of how a property parameter's value changes within a selected element. The tab shows a list of the elements in the process and, for each element, the value of each property applied in the element on each iteration of the simulation. Using the 'Toggle Filter Bar' option on the header bar, you can refine the display to show only a particular property and see how often it is used by the element and with what values.
Group by Property	Displays the results from the perspective of how the value of each property changes during the whole process. The tab shows a list of properties that are applied during the process and, for each property, the value in each activity (element) on each iteration of the process.

Examples

In the EAExample Model, you can study two examples of generating information on property parameters from a simulation of a BPMN business process model. These will demonstrate how you define property parameters in the configuration, based on the model. You can initially just run a simulation on each example, and examine the output as described here. You can then examine the business processes and configurations themselves, and change or add to the property parameters provided.

The examples are described in the *Tracking Property Values - Examples* topic. Briefly, they are:

- 'Fibonacci' - a very simple recursive business process that calculates a series of Fibonacci numbers through ten iterations; you can see how the property parameters increment in each iteration through the elements of the process (in Example Model > Model Simulation > BPSim Models > Fibonacci)

- 'Car Repair' - a more complex and realistic process that represents what might happen when a series of individual 'walk-in' customers bring vehicles into a car repair shop for estimation and repair (in Example Model > Model Simulation > BPSim Models > Car Repair Process)

There is also a small example of the behavior of the Time Parameters (in Example Model > Model Simulation > BPSim Models > Time Parameter).

Notes

- If a BPSim configuration contains result requests and a Custom simulation is performed on it, the 'BPMN Simulation Report' shows only the built-in parameters requested in the configuration; in contrast, the 'BPSim PropertyParameter Values' dialog lists all property parameters regardless of any result request settings or type of simulation

# Tracking Property Parameter Values - Examples

To help you learn about the facility for generating information on property parameters from a simulation of a BPMN business process model, Sparx Systems provide two examples that you can explore in the EAExample model. These are the:

- Fibonacci process - a very simple example to help you become familiar with the parameter tracking facilities
- Car Repair process - a more complex example that you can manipulate to see how a real-life process might be investigated

At the end of this topic is a section that briefly discusses how you might work with an integer-based process containing parameters that are initialized by 'real' distributions, and a section describing the example of the behavior of the Time Parameters.

The Fibonacci Example

This is a very simple recursive business process that calculates a series of Fibonacci numbers through ten iterations; you can see how the property parameters increment in each iteration through the elements of the process. Open Example Model > Model Simulation > BPSim Models > Fibonacci.
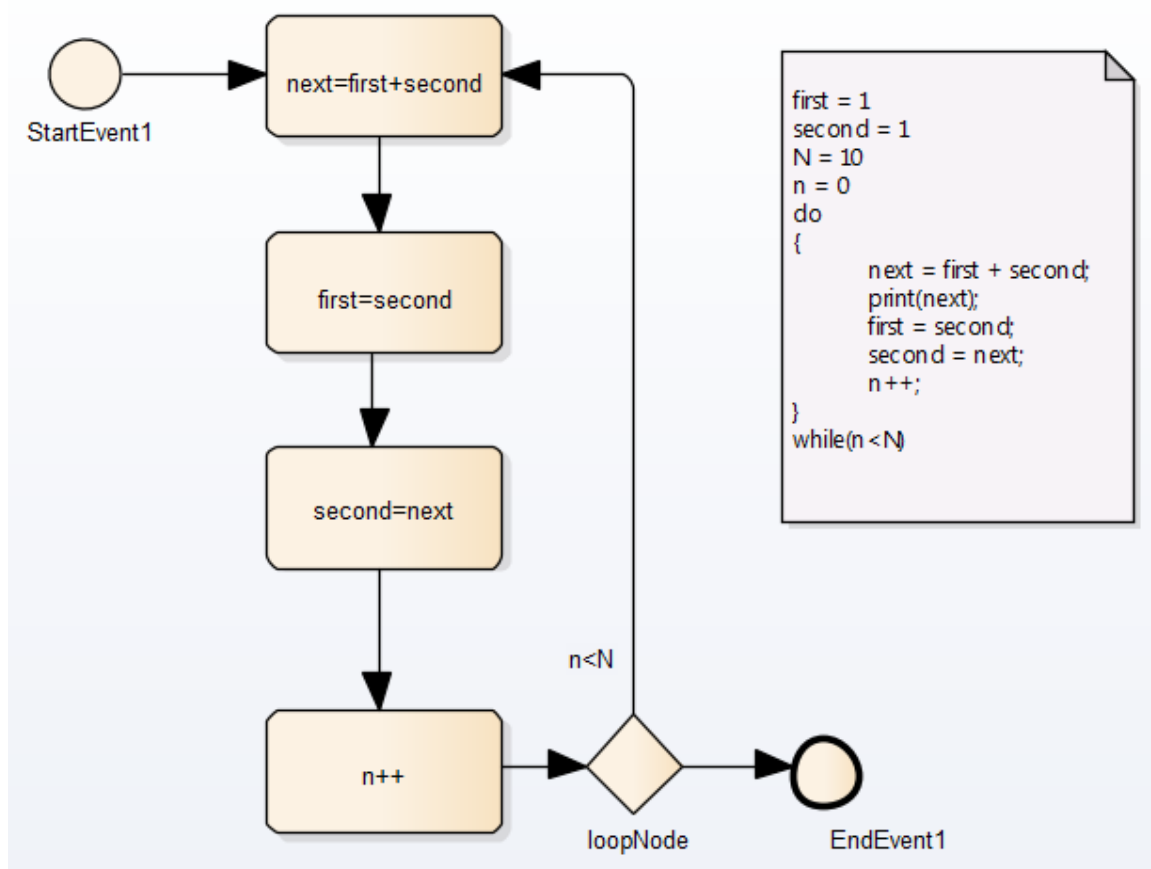
The pseudocode of the process is shown in the Notes element on the diagram. The statement 'print(next)' will output the number series 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.
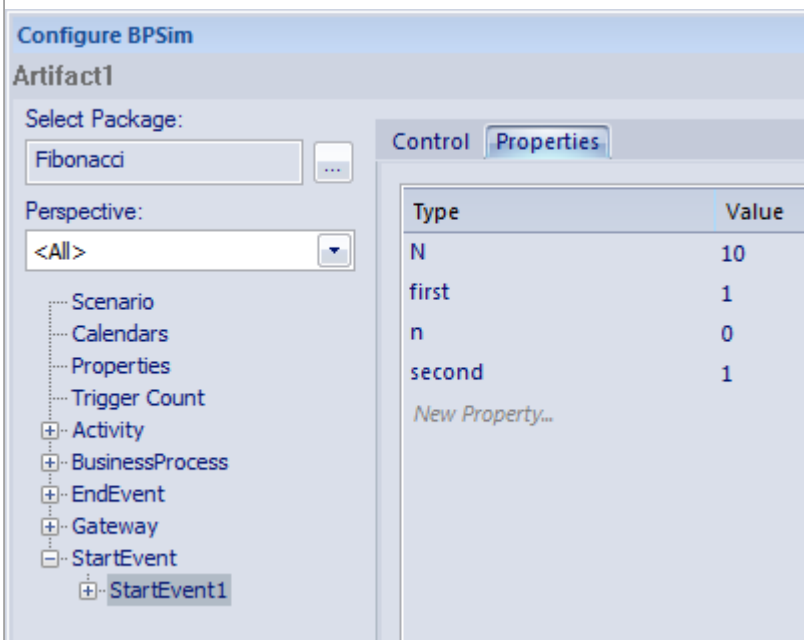
The BPSim configuration for this process is set up as described here.

Step	Action
1	
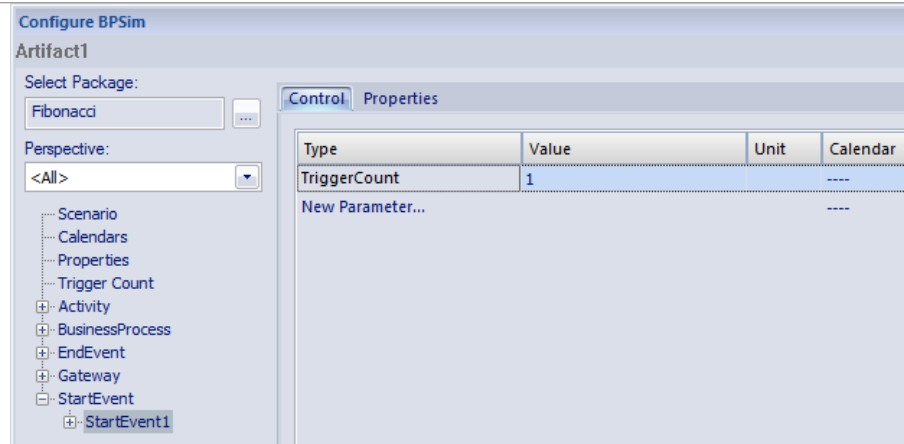
On the 'Control' tab for the StartEvent element, set 'TriggerCount' to '1', and on the 'Properties' tab create and initialize the properties:

- 'N' as '10'

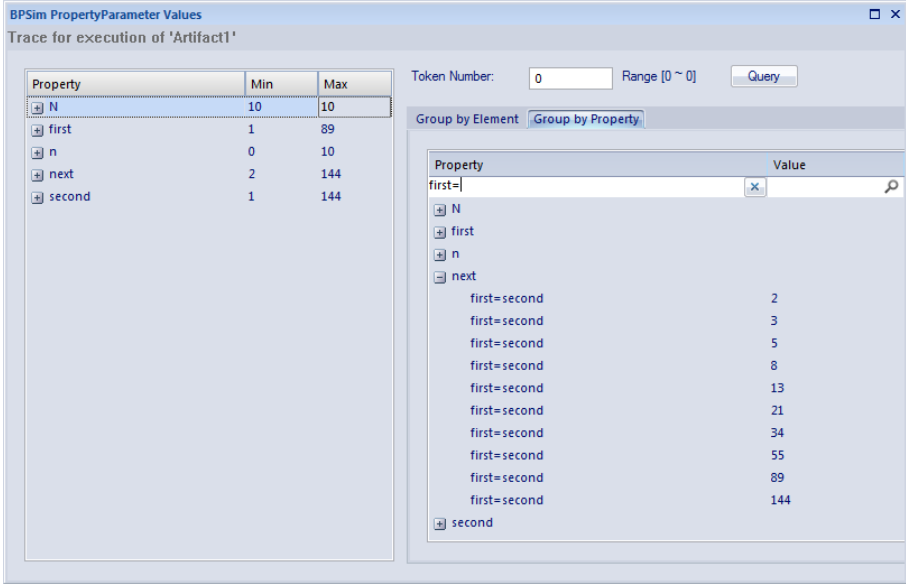- 'first' as '1'

- 'second' as '1'

- 'n' as '0'

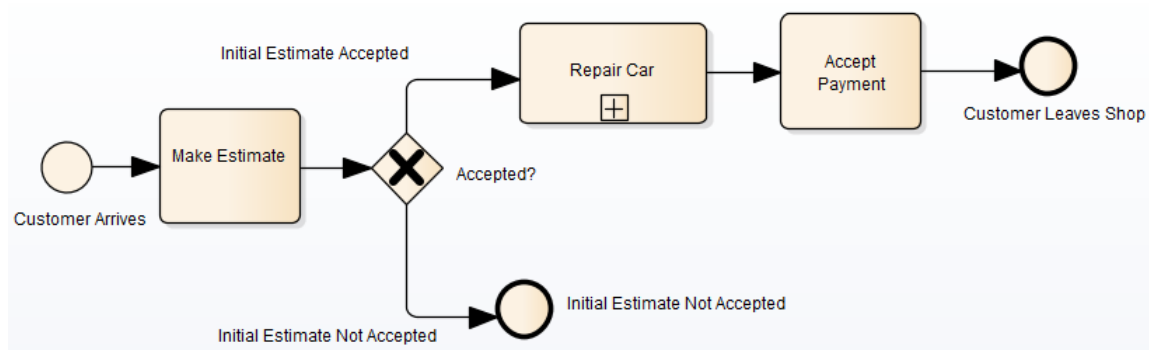| 2	Now define the properties for each of the Activities in the process, on the 'Properties' tab. Note that the values for these properties are derived from **Expressions**, the components of which must be enclosed in braces - {xxx}. For the Activity:
	- next=first+second - set the property 'next' and define the value as the Expression {first} + {second}

- first=second - set the property 'first' and define the value as the Expression {second}
- second=next - set the property 'second' and define the value as the Expression {next}
- n++ - set the property 'n' and define the value as the Expression {n} + 1

| 3 | Set the 'Condition' property parameters for the two Sequenceflow connectors issuing from the 'loopNode' Gateway element, on the 'Control' tab. |

Expand the Gateway | loopNode element and for the link to:

- next=first+second - set the Control parameter to 'Condition' and define the value as the Expression {n}<{N}

- EndEvent1 - set the Control parameter to 'Condition' and define the value as the Expression {n}=>{N}

| 4 | Having completed the configuration, click on the Run button on the Configure BPSim window and on the 'BPSim Simulation Controller' dialog, selecting a Standard simulation. |

When the simulation is complete, click on the Attributes button.

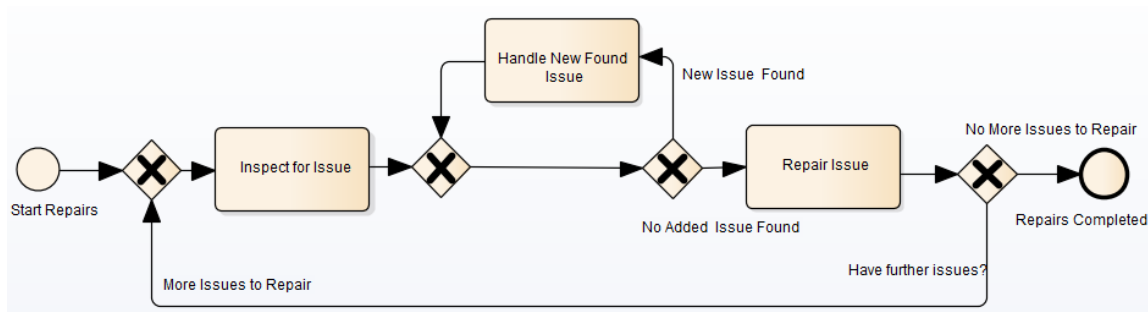|	
	On the 'BPSim PropertyParameter Values' dialog, set the 'Token number' field to '0' and click on the Query button.
5	Now examine the values of the 'next' property on entering the first=second Activity in each iteration of the simulation. Click on the 'Group by Property' tab and expand the 'next' item. The list of values is long, so right-click on the column headers and select the 'Toggle Filter Bar' option. Under the 'Property' column heading, type 'first='. This filters the list to show only the property parameter values on entering the first=second Activity.

The Car Repair Example

This more complex example is based on a realistic model of a car repair process, where a number of individual customers request an estimate for repair and either proceed with the repair or decline to continue; you can see how the property parameters vary as different decisions are made during the process. Open Example Model > Model Simulation > BPSim Models > Car Repair Process.

The overall process is represented by this diagram:

The *Repair Car* Activity is a composite element that contains this sub process diagram:

Step	Action
1	In the Project Browser, expand the 'BPSim' child Package underneath the 'Car Repair Process' Package, and double-click on the 'Scenario 1: Main Flow' Artifact. The Configure BPSim window displays. On the 'Scenario' tab look at the 'Duration' field; this has been set to 2 days and 12 hours (that is, 60 hours).
2	In the process hierarchy on the left of the window, expand the 'Start Event' category and click on 'Customer Arrives'. Select the 'Control' tab and look at the 'InterTriggerTimer' parameter, which has the value 24 minutes; that is, a customer arrives every 24 minutes (so over the 60 hour duration, 150 customers pass through the repair shop).
	Each customer enters the repair shop with one or more issues to be evaluated and repaired. The number of issues each customer presents can be randomly generated using one of the Distributions supported by BPSim. As the issue number is counted in discrete units (rather than measured on a continuous scale) we would use an 'integer' distribution. If you select the 'Properties' tab for the 'Customer Arrives' Start Event, you will see that the 'noOfIssues' property value is initialized from a Poisson distribution with a mean of 3.
3	Now expand the decision Gateway 'Accepted?' and its connectors, in the process hierarchy. 'Initial Estimate Accepted' has a Control parameter 'Probability', set to 0.67. The alternative connector, 'Initial Estimate Not Accepted' has a similar Control parameter 'Probability' set to 0.33. That is, we expect an average of one issue in three to be withdrawn - or not pursued - by the customer.
4	Further into the process, when an issue is being assessed on the vehicle, there is a possibility of another issue being discovered.
	In the list of Gateway elements, the last 'unnamed element' has two paths: 'New Issue Found' and 'No Added Issue Found'. Click on each of these and look at the 'Control' tab; the 'Probability' parameter for 'New Issue Found' is set to 0.25 and, for 'No Added Issue Found', to 0.75. So on average, for every four issues reported and assessed, one new one is discovered.
	The 'New Issue Found' path takes the process to the 'Handle New Found Issue' Activity, which adds 1 to the number of issues to be processed for the current customer. Expand the Activity group, and click on the 'Handle New Found Issue' element and on the 'Property' tab. You will see that the property 'noOfIssues' here has the Expression value {noOfIssues} **+ 1**.
5	When a problem with the vehicle is resolved, the 'Repair Issue' Activity deducts 1 from the number of issues to be repaired for the current customer. Click on the 'Repair Issue' element in the Activity group and on the 'Property' tab. You will see that the property 'noOfIssues' here has the Expression value {noOfIssues} **- 1**.
6	The value from the 'Repair Issue' Activity is tested at the 'Have Further Issues?' Gateway.
	Click on the 'More Issues to Repair' connector and on the 'Control' tab; the Condition parameter for following this path is set to the Expression value {noOfIssues} **> 0**; flow passes to the Gateway prior to the 'Inspect for Issue' Activity.
	Similarly, if you click on the 'No More Issues to Repair' connector and on the 'Control' tab, the Condition parameter for following that path is set to the Expression value {noOfIssues} **=< 0**, and flow passes to the

		'Repairs Completed' End Event.
		Now that you have examined the process flow and configuration settings, you can run a simulation and review the results.
7		On the Configure BPSim window, click on the Run button, and then on the 'BPSim Simulation Controller' dialog click again on the Run button, selecting the 'Standard' simulation (however, the type of simulation makes no difference for reviewing property parameters).
		On the 'BPSim Simulation Controller' dialog, you can review the Token status (and see that an additional customer manages to enter the shop at the very last minute) but it is difficult to see exactly how this summary data resulted. Click on the Attributes button to obtain the detailed property parameter values information on the 'BPSim PropertyParameter Values' dialog.
8		On the left hand side of the dialog is a summary of the minimum and maximum values for the property parameter (attribute) for each element in the process. For example, for the 'Customer Arrives' element the 'noOfIssues' parameter has a minimum of 0 and a maximum of 8, as generated by the Poisson (3) distribution.
		In the 'Token Number' field, type in a number (N) between 0 and 150 to select for the Nth customer who entered the repair shop. Click on the Query button to obtain the property parameter values used in the process for that customer. Review the results on each of the two tabs:
		• On the 'Group by Element' tab, see how the attribute's value changes in each element; for example, for customer 24 the 'noOfIssues' parameter is initialized with a value of 4 by the random distribution, and the 'Inspect for Issue' Activity is called six times with the parameter value being adjusted to 3 for three of those calls before cycling to 1, and the 'Handle New Found Issue' Activity is called twice with the parameter value at 3 both times
		• On the 'Group by Property' tab see how the parameter value changes as the process cycles through the Activities to completion, starting at 4, being adjusted between 3 and 4 a number of times and then decrementing to 0 at the end
9		Continue to explore the results as required, selecting different customers (Tokens). You can also return to the BPSim configuration and change the parameter initializations and add new ones, or change the decision points, to experiment with the process.

Responding to real numbers in the simulation of an integer-based process

In some cases, you might need to generate property parameter values using a distribution that returns 'real' numbers when the activities in the process operate with integers, or when you want to see what impact forcing integer values has on the process.

One mechanism to apply in such cases is to set conditions to avoid absolute numbers. So, for example, you might have a counter that decrements by 1, that is initialized to a 'real' number. If you set a condition to 'value==0' (equals 0) or 'value !=0' (does not equal 0), the two conditions might never be True or might always be True, respectively, causing an infinite loop. To avoid that, in the conditions you would use operators such as:

'value > 0'

'value < 0'

'value >= 0'

'value <= 0'

Another mechanism is to edit the code template used by the BPSim engine, to intercept and replace the real numbers provided to specific parameters with integers, as shown:

1. Select the 'Code > Configure > Options > Edit Code Templates' ribbon option.

2. In the Code Template Editor, in the 'Language' field, click on the drop-down arrow and select 'MDGBPSimExecutionEngineExtension'.

3. In the list of (Java) templates, click on 'MDGBPSimExecutionEngineExtension Compute Value'. The template contents display in the 'Template' panel.

4. Find this line:

    double %bpsimPropertyParameterName% = (double) distribution.next();

    Change it to:

    %**if** bpsimPropertyParameterName == "noOfIssues" or bpsimPropertyParameterName == "noOfVisitors"%
    double %bpsimPropertyParameterName% = (int) distribution.next();
    //double %bpsimPropertyParameterName% = Math.ceil(distribution.next());
    //double %bpsimPropertyParameterName% = Math.floor(distribution.next());
    //double %bpsimPropertyParameterName% = Math.round(distribution.next());
    %**else**%
    double %bpsimPropertyParameterName% = (double) distribution.next();
    %**endIf**%

5. Replace the property parameter names with your own property parameters.

6. Click on the Save button, close the Code Template Editor and reload the project.

As presented, for each specified parameter the code template will simply replace any 'real' number initialized by the distribution with an integer. If you prefer, you can use one of the commented lines instead:

- Math.ceil() will take the 'real' number and convert it to the next highest integer

- Math.floor() will take the 'real' number and convert it to the next lowest integer

- Math.round() will take the 'real' number and round it up or down depending on whether it is greater than or less than n.5

Time Parameter Behavior

In the BPSim configuration, you can set a number of Time parameters for an Activity, such as Queue Time and Wait Time. You can also set a Result Request on each of these, for a Custom simulation. However, the BPSim simulation engine combines these parameters into a single 'Processing Time' quantity.

Consider the simple model TimeParameter in the Example Model (Example Model > Model Simulation > BPSim Models > Time Parameter), represented by this diagram:
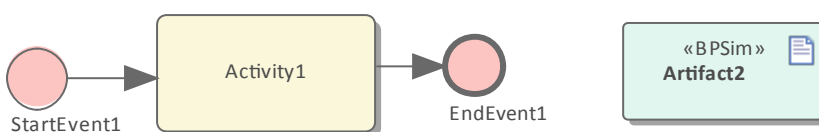
If you double-click on the Artifact2 element, the Configure BPSim window displays. Click on the Activity1 element in the diagram to expand the Activity group, to select Activity1 in the hierarchy at the left of the dialog, and to display the first tab, 'Time', for the element in the configuration, as shown.
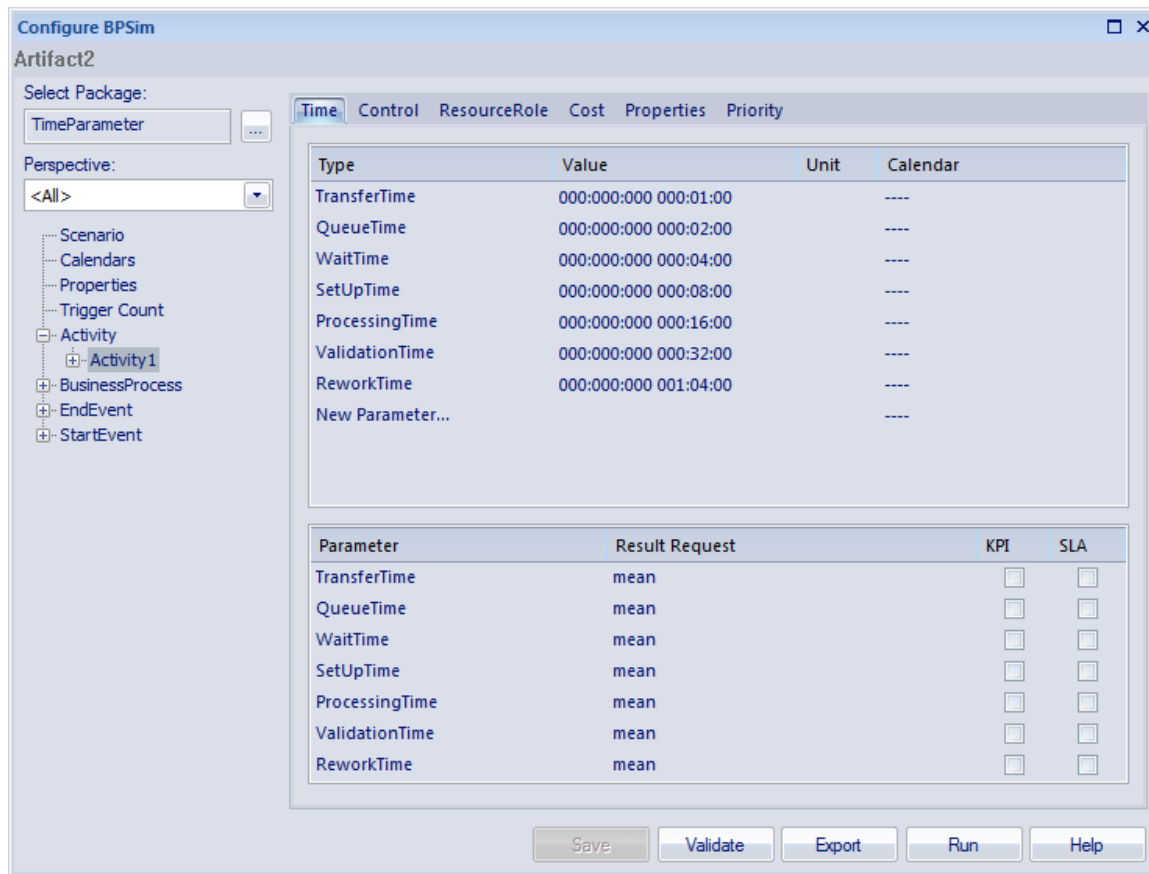
Note that in the upper panel there are seven system-supplied 'Time' parameters, which have been given initial values of - in order - 1, 2, 4, 8, 16, 32 and 64 minutes (the 64 minutes is 1 hour and 4 minutes). Note also that in the lower panel, each of these has a Result Request for the mean runtime value of the parameter.
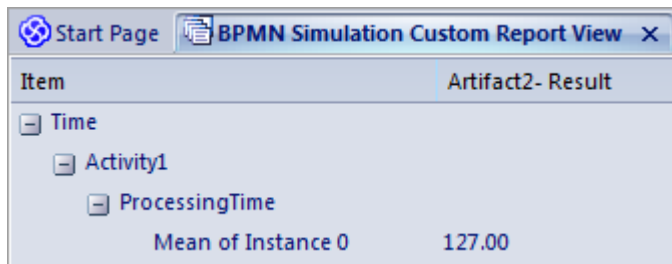
Click on the Run button, and on the 'BPSim Simulation Controller' dialog click on the Run button and select 'Standard Simulation'. The simulation is configured to cycle through the process once. When the simulation is complete, click on the Open Result button, and on the 'BPMN Simulation Report View' right-click and select the 'Show Only Non-Empty Items' option. This gives you, for the Activity1 element on which the parameters were set, these results:

All of these derived results are 127 minutes, the sum of the initial values of the original seven 'Time' parameters. The individual parameters are not processed separately.

If you return to the 'BPSim Simulation Controller' dialog and click on the Run button, selecting 'Custom Simulation' this time, the Open Result button displays the 'BPMN Simulation Custom Report View'. In the configuration, the Result Requests were for the mean values of the seven parameters. In the Report View for the simulation, you only see the mean of the single aggregated parameter, ProcessingTime, as 127 minutes.

Compare BPSim Configurations

When you develop a BPSim configuration, you can define a wide range of parameters to set prior to running simulations and observing the effects of those settings, and of changes in selected settings. To facilitate using and managing multiple 'what-if' scenarios, it is recommended that you create copies of the original configuration (as Artifact elements) and make the setting changes in the copies.

A useful facility in creating variations of a configuration is to apply inheritance, whereby the data and parameters you do NOT intend to vary are held in one configuration, and only those parameters that you change are held in another. The 'variable' configuration uses (inherits) the common data held in the base configuration, so you do not have to re-create that common data in the 'variable' configuration.

You can then run simulations on the changed configurations and on the original 'baseline' and compare the simulation reports to see what differences occurred in the run-time variables, and then run and display the comparisons of the configurations to see what changes in parameter settings gave rise to those run-time differences.

By running simulations under the original and copy configurations, comparing the results and the changes that caused the results, and modifying the model accordingly, you can achieve a very high degree of control in streamlining the business process you are developing.

Access

Context Menu	On diagram or Project Browser	Right-click Business Process Simulation Artifact	Show BPSim Configuration

The BPMN Simulation Configuration Report View

This view initially displays a hierarchy of the parameters in the selected configuration that have values and, in the column under the configuration name, the values set for those parameters. You can expand or collapse individual tab and parameter items using the +/- expansion/contraction boxes, or the hierarchy as a whole by right-clicking on the view and selecting the 'Collapse All' or 'Expand All' options. To review the results more easily, you can also drag the tab off the workspace to become a floating display, and expand the display to full screen size.

To compare two (or more) configurations, click on another configuration Artifact element in the Project Browser or diagram and either:

- Drag it onto the report view or

- Select the 'Show BPSim Configuration Results' option

The parameter hierarchy now contains any additional parameters in that configuration, and its parameter values display in a column to the left of the original configuration values.

You can review and manipulate the information on the report using options available from a right-click context menu.

BPMN Simulation Configuration Report Options

Option	Description
Collapse All	Select this option to collapse the parameter hierarchy down to just the parent tab names.
Expand All	Select this option to expand the parameter hierarchy to the lowest value type.
Show Only Different items	(When you have two or more configurations shown.) Select this option to restrict the display to those parameters where the values differ between the configurations. Click on the option again to deselect it.
Highlight Different Items	(When you have two or more configurations shown and where their parameter values are different.) Displays the differing parameter values in red. This option is disabled if you select the 'Show Only Different Items' option.
Show Only Non Empty Items	Select this option to filter the display to show only parameters that have a specific value other than 0.
Remove Scenario	(When you have selected an item within a configuration.) Select this option to remove the whole configuration from the report.
Edit Parameter	(When you have selected an item within a configuration.) Select this option to display the appropriate tab in the Configure BPSim window, with the parameter and value highlighted, and edit the value. When you have edited the value and clicked

	off it, the Save button is enabled; click on this button to return to the Report View. The hierarchy is collapsed, but when you expand it again the selected parameter value shows the change you made.

BPSim Charts

The Artifacts page of the Diagram Toolbox provides two icons specifically to generate charts that reflect selected results from BPSim simulations. These are:

- BPSim Result Chart - to generate a Chart that reflects selected results from a series of standard BPSim simulations
- BPSim Custom Result Chart - to generate a Chart that reflects results from a series of customized BPSim simulations

As for other Chart Artifacts, both BPSim Chart types can be quickly configured to display the simulation results in variations of a line chart, two-dimensional bar chart or 3-dimensional bar chart.

Prerequisites

To populate the Charts created from the Business Process Simulation Artifacts, you select the Result Artifacts created during the simulation of each configuration that you want to show. Therefore, the initial simulations must be performed first, and the Report Artifacts generated.

Access

Display the 'Artifacts' page of the Diagram Toolbox using any of the methods outlined in the table.

Then, drag the BPSim <type> Chart Artifact icon onto the diagram - a new chart element is created.

Double-click on the new chart element to open the 'Properties' dialog, showing the 'BPSim Chart' page.

Ribbon	Design > Diagram > Toolbox > Artifacts
Keyboard Shortcuts	Alt+5 | Aritfacts
Other	You can display or hide the Diagram Toolbox by clicking on the ⟫ or ⟪ icons at the left-hand end of the Caption Bar at the top of the Diagram View.

Select Results to Display In Chart

Complete the fields on the 'BPSim Chart' page of the Chart 'Properties' dialog.

| Option	Action
Base Report	Click on the [...] button to display the 'Select <<BPSimReport>>Artifact' dialog, and select the report Artifact for the simulation results against which other simulation results will be compared. This Results Artifact is also added to the 'Experiments' panel as the first in the list of report results to be compared.
Type	Click on the drop-down arrow and select the format type of the Chart in which to display the results - 2D Bar, 3D Bar or Line. After you have specified the report parameters to compare, you can select the 'Appearances' page of the dialog and define the appearance of the bar chart or Time line graph.
Report Schema	Expand the hierarchy as necessary and select the checkbox against each property to display on the chart. Each property will be represented by a separate line or group of bars on the chart. Usually you would select similar objects (such as different resources) and the same single property for each object (such as the degree of utilization of the resource). You have a wealth of properties to examine and compare, but any more than a couple on the same chart makes the chart hard to read.
Experiments (BPSim Reports)	This panel lists the simulation report results (as BPSim Result Artifacts) that you have selected to compare using the chart, in the sequence in which their selected parameter will be shown on the Chart. Usually, the Base Report remains as first in the list and the result for its parameter is shown at the left of the Chart. If you want to change the sequence, click on the Result Artifact name and click on the appropriate Up/Down green arrow button.
	To add further Result Artifact names to the list, click on the Add button and browse for and select the Artifact from the 'Select <<BPSimReport>> Artifact' dialog.

A BPSim Chart Example

In the Help Desk Phone Support example, we created three BPSim artifacts to do what-if analysis of: 'How many supports do we need to answer the customer's questions over the phones in an economical way?'

We started with two supports, then tried three and five supports. After simulation, we have a BPSim Report based on different configurations: TwoSupport-Result, ThreeSupport-Result, FiveSupport-Result.

These are the steps to create a chart to compare the customer's average time waiting for support:

1.   Create a BPSim Result Chart on the diagram, and name it *Average Waiting Time.*

2.   Double-click on the chart to open the Properties window, then open the 'BPSim Chart' tab.

3.   Click on the [...] button to select a Base Report, from which we define the schema (legends) to use in the chart. Select 'TwoSupport-Result'.

4.   Choose this schema:
         - Time | Service Customer | Average Time Waiting For Resource

5.   Click on the Add button to add another two BPSim Reports: 'ThreeSupport-Result' and 'FiveSupport-Result'.

6.   Click on the OK button, and adjust the size of the chart element. This chart gives us direct information.
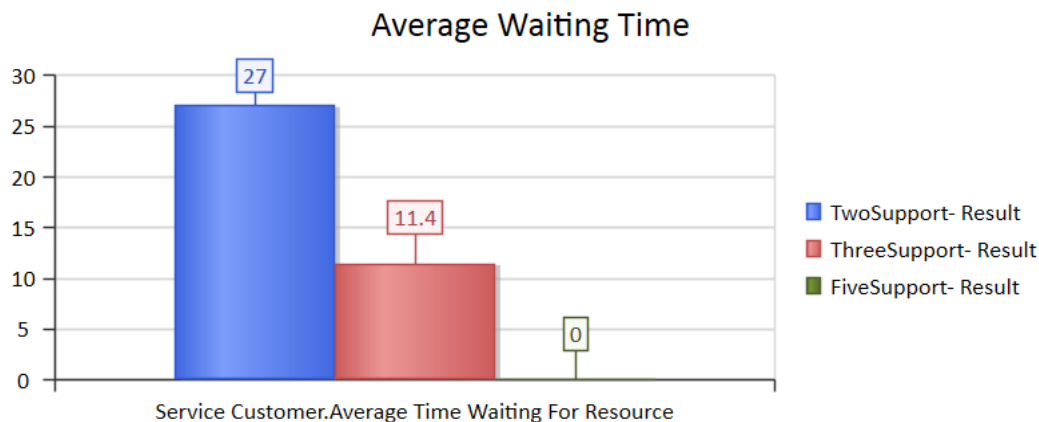
These are the steps to create a chart to compare the degree of use of Support:

1.   Create a BPSim Result Chart on the diagram, and name it *Utilisation Rate.*

2.   Double-click on the chart to open the Properties window, then open the 'BPSim Chart' tab.

3.   Click on the [...] button and select a Base Report, from which to define the schema (legends) to use in the chart. Select 'TwoSupport-Result'.

4.   Choose this schema:
         - Resource | Support | Degree Of Utilisation

5.   Click on the Add button to add another two BPSim Reports: 'ThreeSupport-Result' and 'FiveSupport-Result'.

6.   Click on the OK button and adjust the size of the chart element. This chart provides specific information.

BPSimResultChart - UtilisationRate
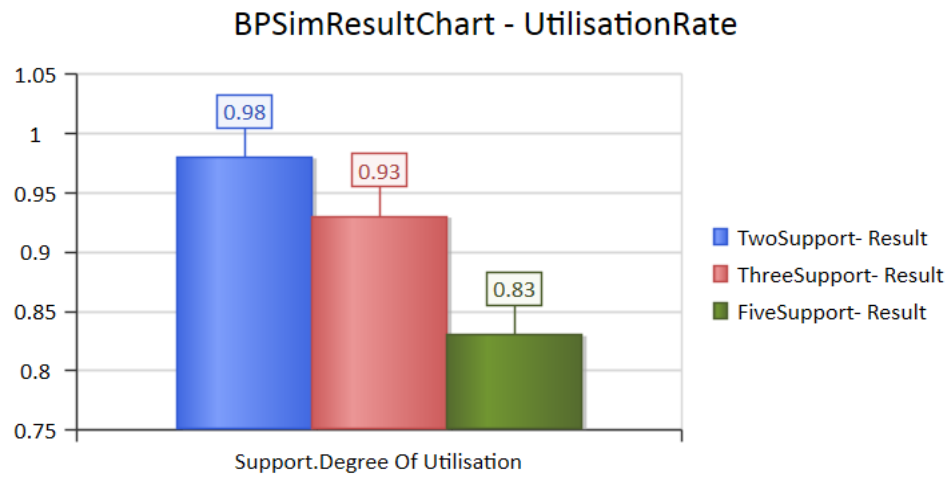
BPSim Examples

This document contains many examples of BPMN modeling, BPSim Configuration and Analysis of the simulation results.

These examples can be accessed from the EAExample model.

The EABPSim Execution Engine is required to run the simulations.

# Help Desk Phone Support Simulation - Resource Perspective

In this example, we create a very simple model to simulate a Help Desk phone support process.

We set up a scenario in which resources are limited and the requests have to be put in a waiting queue for a resource. Then we try to seek a balance point between customer's waiting time and the number of resources, using a what-if analysis.

Firstly, we model this process step-by-step, starting with a simple parameter setting than can be calculated with pen and paper, then verifying it with EABPSim. After that, we perform a what-if analysis that might help the manager to make a decision.

Create BPMN Model

The model itself is very simple, consisting of a Start Event, a Task and an End Event.
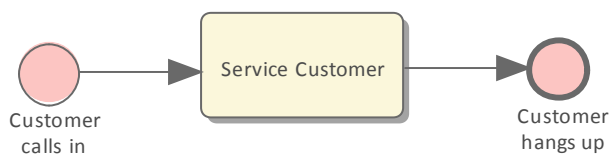
- Create a Start Event called *Customer calls in*
- Add a Sequence Flow to a target Abstract Task called *Service Customer*
- Add a Sequence Flow to a target End Event called *Customer hangs up*

Create a BPMN2.0 Resource named *Support;* this element will be used in the BPSim Configuration.

Pen and Paper Analysis

We will use pen and paper to analyze this case:

- 7 customers call in at 2 minute intervals
- 2 support resources are available
- Each service will take 10 minutes

Pen & Paper Analysis on a BPSim Example

	Wait Time		Total Time
Customer7	18		28
Customer6	12		22
Customer5	12		22
Customer4	6		16
Customer3	6		16
Customer2			10
Customer1			10

0 2   4   6   8   10   12   20 22   30 32   40

🟥 Customer waiting
🟦 Support serving the Customer
🟨 Support is wait / free (drinking coffee)
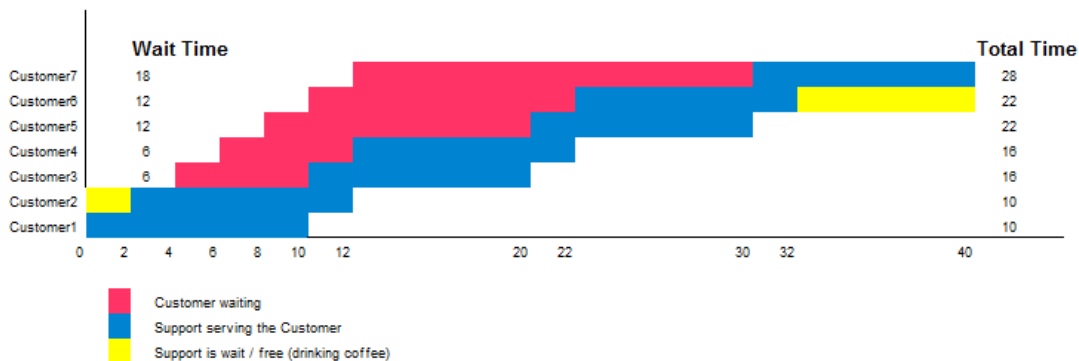
Now we can perform some calculations using a pen and paper based approach:

Total Time Waiting For Resource:	(18+12+12+6+6+0+0) = 54	number of 🟥
Total Time In Task:	(28+22+22+16+16+10+10) = 124	
Average Time in Task:	124/7 = 17.71	
Average Time Waiting For Resource:	54/7 = 7.71	
Maximum Time In Task:	28	
Maximum Time Waiting For Resource:	18	
Sum of Support's wait time:	2+8 = 10	number of 🟨
Sum of Support's time for Task:	7*10 = 70	number of 🟦
Degree of Utilisation:	70/(10+70) * 100% = 87.5%	

Seen from this result, it is already a very complicated computation for such a simple model when resource constraints are applied. When the process expands and more constraints are applied, analyzing the process with pen and paper will quickly become impossible. We will demonstrate how EABPSim can help.

BPSim Configuration

- Open the Configure BPSim window ('Simulate > BPMN/BPSim > Manage > Open BPSim Manager')
- Create a Business Process Simulation Artifact named *Pen & Paper Analysis 7 Customers*
- Select the Package containing the corresponding BPMN 2.0 model

All the BPMN elements will be loaded in to the Configure BPSim window.

- In the left-hand tree on the window, expand the 'Resource' group and click on *Support;* click on the 'Resource' tab and on the drop-down arrow of the 'New Parameter' field, then select 'Quantity' and, in the 'Value' field, type '2'
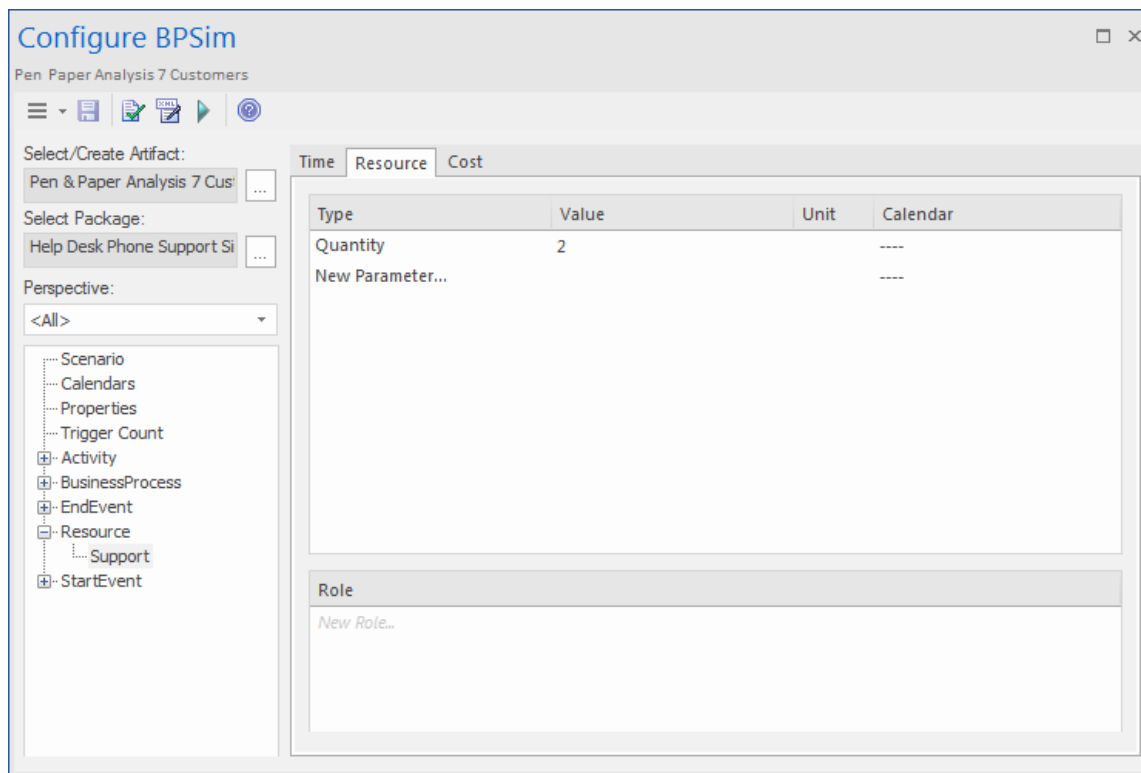
- In the left-hand tree, expand the 'StartEvent' group, then click on *Customer calls in* and on the 'Control' tab

- Click on the drop-down arrow of the 'New Parameter' field and select, in turn:

    - 'InterTriggerTimer' and, in the 'Value' field, click on the [...] button, select the 'Constant' tab
      and 'Numeric', type '2' in the 'Constant Numeric' field and select 'minutes' in the 'TimeUnit' field
    - 'TriggerCount', and in the 'Value' field type '7'
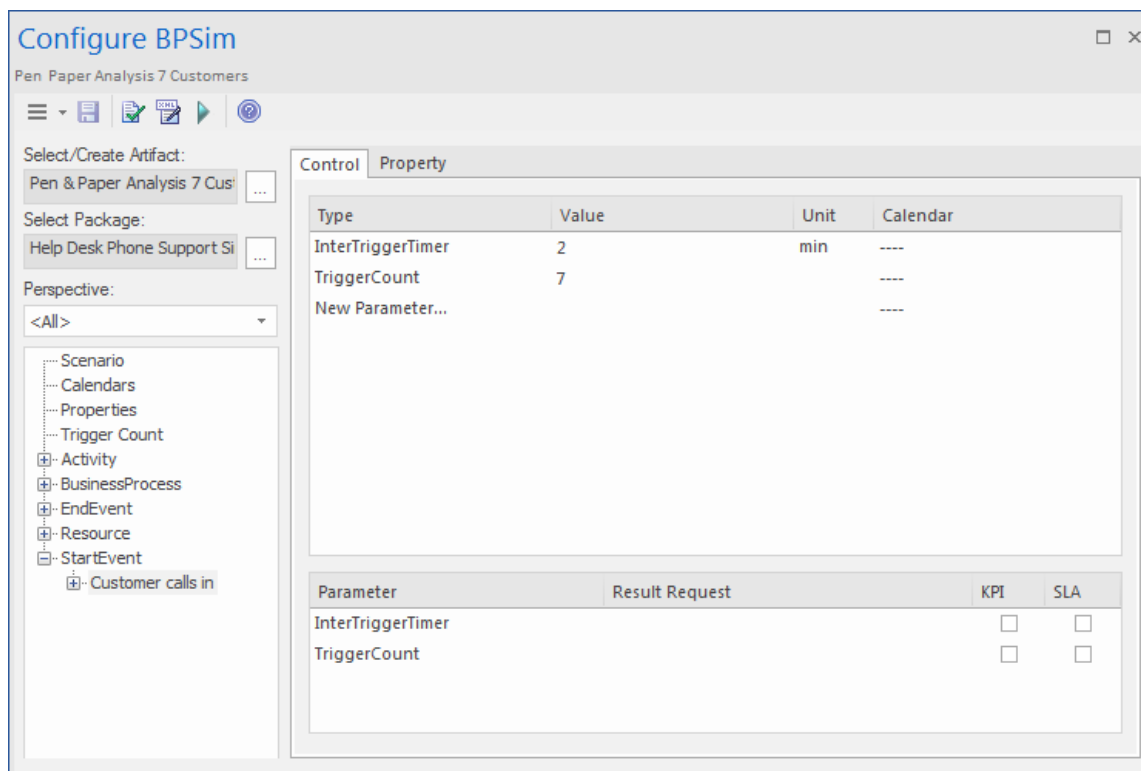
- In the left-hand tree, expand the 'Activity' group, then click on *Service Customer* and on the 'Time' tab

- Click on the drop-down arrow of the 'New Parameter' field and select 'ProcessingTime' and, in the 'Value' field, click on the [...] button and select the 'Constant' tab and 'Numeric', type '10' in the 'Constant Numeric' field and select 'minutes' in the 'TimeUnit' field

- Click on the 'ResourceRole' tab, click on the Add Resource Role button; in the new row in the 'Resource Roles defined in BPMN' panel, click on the [...] button in the 'BPMN Resource' field and select the *Support* element you created earlier

- In the 'Quantity' field, type '1'

In the 'Resource Selection Expression for Activity' panel the automatically generated expression *bpsim:getResource('Support',1)* displays.
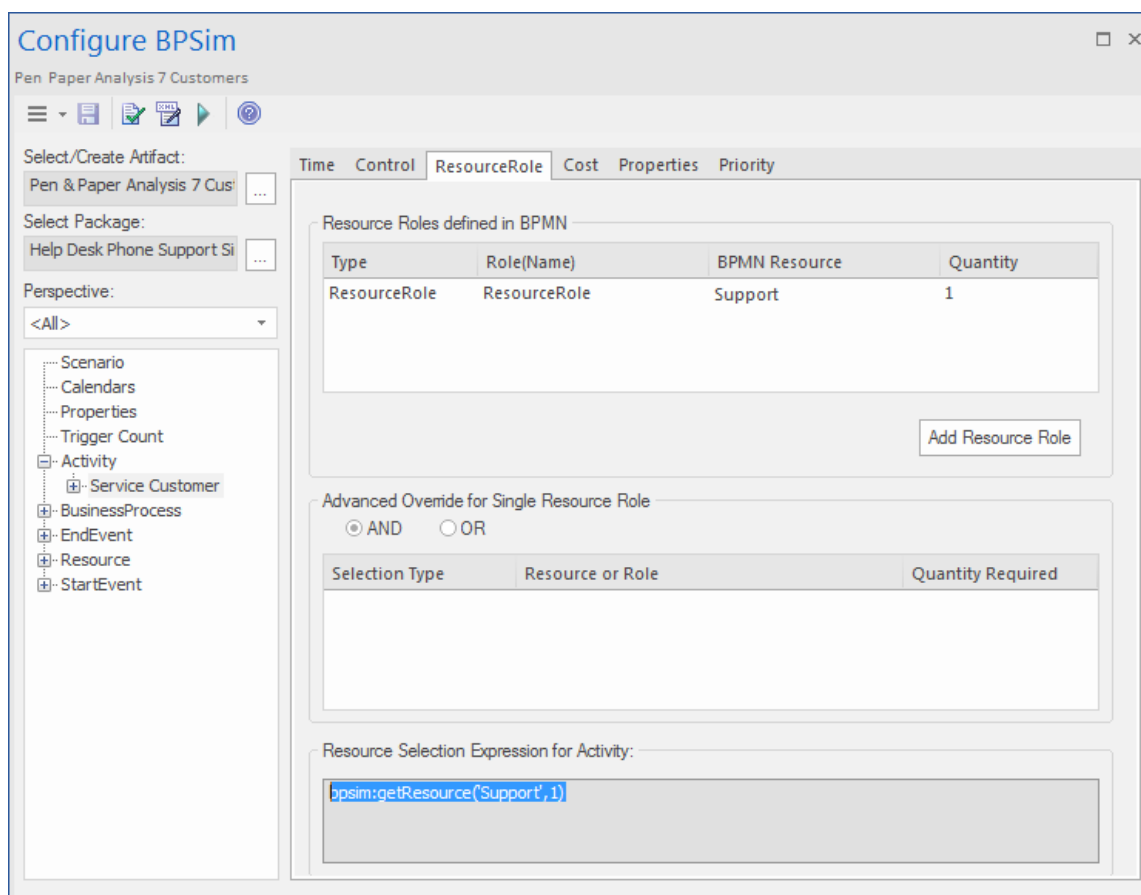
Run the Simulation

- On the Configure BPSim window Toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays

- Click on the Run button drop-down arrow and select 'Standard Simulation'

- When the simulation is complete, click on the [ ] button on the 'BPSim Simulation Controller' dialog toolbar to open the 'BPMN Simulation Report View'

Tip: You can select the 'Show Only Non-Empty Items' option to filter the results.

The results match the pen and paper analysis.

Simulation - 2 Support Resources for 20 customers

You can create a new Business Process Simulation Artifact by copying an existing BPSim Configuration. Copy the *Pen & Paper Analysis 7 customers* element and press Ctrl+Shift+V to paste, giving the new element the name *TwoSupport*.

- Double-click on *TwoSupport* to open the 'Configure BPSim' dialog; you can see that all the configurations are kept from the copied source

- Expand the 'StartEvent' group and click on *Customer calls in*; click on the 'Control' tab and, in the 'Value' field for the 'TriggerCount' parameter, change the figure to '20'

Run simulation and analyze results

From the report, you can see that:

- The 'Average Time Waiting For Resource' is 27 minutes and the 'Maximum Time Waiting For Resource' is 54 minutes

- The two Support resources - are they busy? If they were not, we might have to change the process to use all of their time and reduce the customer's waiting time; however, the 'Degree Of Utilisation' is at 98%, which indicates that the resources had almost no idle time

'What-If' I have more staff? Compare 2 Support Resources with 3 and 5 Support Resources

- Copy *TwoSupport* and press Ctrl+Shift+V to paste, giving the new element the name *ThreeSupport*

- Double-click on *ThreeSupport* to open the 'Configure BPSim' dialog, expand the 'Resource' group and click on *Support*, then click on the 'Resource' tab and, for the 'Quantity' parameter, change the 'Value' field to '3'

- Copy *TwoSupport* and press Ctrl+Shift+V to paste, giving the new element the name *FiveSupport*

- Double-click on *FiveSupport* to open the 'Configure BPSim' dialog, expand the 'Resource' group and click on *Support*, then click on the 'Resource' tab and, for the 'Quantity' parameter, change the 'Value' field to '5'

- Run the simulations and do a comparison; in the Project Browser:
  - Ctrl+click on *TwoSupport*, *ThreeSupport* and *FiveSupport*, then right-click and select the 'Show BPSim Configuration' option
  - Ctrl+click on *TwoSupport-Result*, *ThreeSupport-Result* and *FiveSupport-Result*, then right-click and select the 'Show BPSim Report' option

Tips:

- *Use the [button icon] button and click on the 'Show only different items' option for both views*

- *You can dock the views together, so they provide direct comparisons: THESE are the differences in results caused by THOSE differences in configuration*

- *Toggle the filter bar to filter the items that interest you*

Analysis

The docked comparison views show the configuration differences and the corresponding result differences.

- The customer's waiting time dropped from 27 minutes (2 Support resources) to 11.4 minutes (3 Support resources) and further down to 0 minutes (5 Support resources)

- The 'Degree Of Utilisation' dropped from 98% (2 Support resources) to 93% (3 Support resources) and further down to 83% (5 Support resources)

The customers will most likely be satisfied with 5 Support resources; however, the cost might be out of budget. So 3 or possibly 4 Support resources might be a balance point for this case. You are recommended to copy one of the Business Process Simulation Artifacts and configure and run a simulation for 4 Support resources.

Show Result with Chart

- Drag a 'BPSim Result Chart' icon from the Toolbox onto the diagram and create a BPSim Result Chart Artifact; call it *Average Waiting Time*

- Double-click on the Artifact to display the element 'Properties' dialog and click on the 'BPSim Chart' tab

- Click on the [...] button and select a Base Report from which to define the schema (legends) to use in the chart; select *TwoSupport-Result*

- Choose the schema 'Time' | 'Service Customer' | 'Average Time Waiting For Resource'

- Click on the Add button to add another two BPSim Reports: *ThreeSupport-Result* and *FiveSupport-Result*

- Click on the OK button and adjust the size of the chart element; this chart gave us very straightforward information
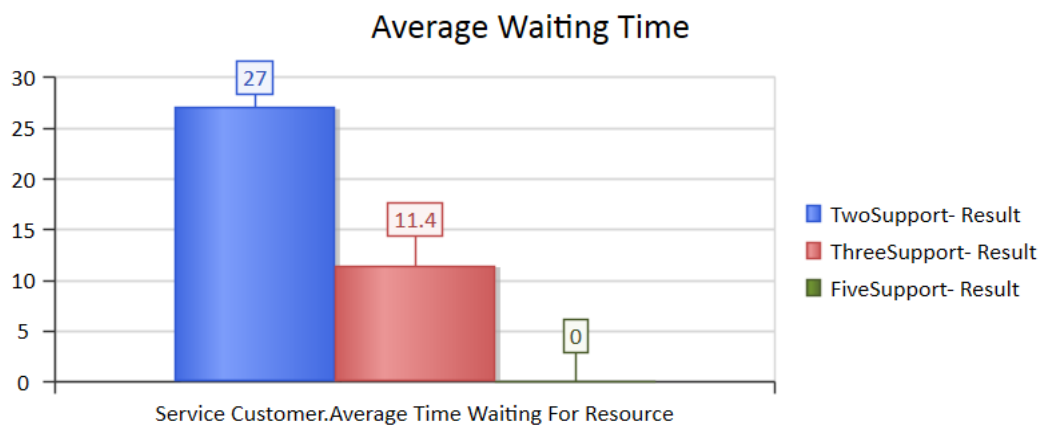
- Create another BPSim Result Chart Artifact on the diagram, called *Utilisation Rate*

- Double-click on the Artifact to display the element 'Properties' dialog and click on the 'BPSim Chart' tab

- Click on the [...] button and select a Base Report from which to define the schema (legends) to use in the chart; select *TwoSupport-Result*

- Choose the schema 'Resource' | 'Support' | 'Degree Of Utilisation'

- Click on the Add button to add another two BPSim Reports: *ThreeSupport-Result* and *FiveSupport-Result*

- Click on the OK button and adjust the size of the chart element
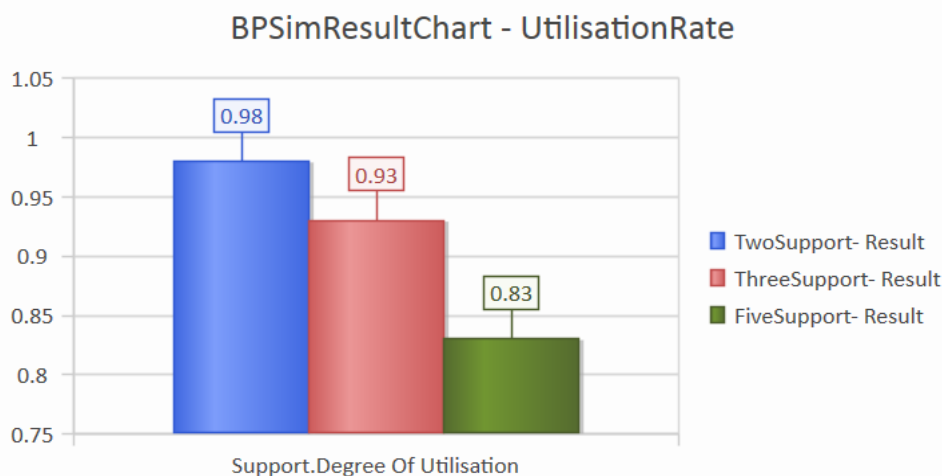
# Calendar-Based Help Desk Phone Support Simulation

In this example, we create a very simple model to simulate the Help Desk telephone support process, based on calendar settings. We assume that:

- Customers call in at different intervals on weekdays and weekends

- Processing times differ between weekdays and weekends

- There are different numbers of support resources on weekdays and weekends

We model this process step by step, then create calendars and configure the Business Process simulation, which is simple enough to be calculated with pen and paper. After that, we run the simulation to compare that result with the pen and paper analysis.

Create BPMN Model

The model itself is very simple, consisting of a Start Event, a Task and an End Event.
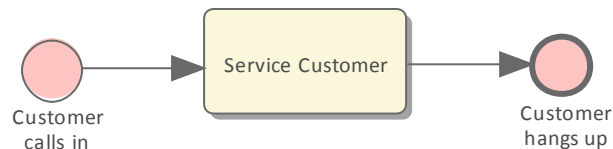
- Create a Start Event *Customer calls in*

- Add a Sequence Flow to the target abstract task Activity *Service Customer*

- Add a Sequence Flow to the target End Event *Customer hangs up*

Create a BPMN2.0 Resource named *Support*. Create a BPMN2.0::ResourceRole inside *Service Customer*, give it the name *support* and set the tag *resourceRef* to the name of the Resource element *Support*.

Pen and Paper Analysis

We can use pen and paper to analyze this case:

- The simulation duration is 2 hours and 10 minutes, from 8:00AM to 10:10AM

- Customers call in every 20 minutes on weekdays

- Customers call in every 60 minutes on weekends

- To service a customer takes 50 minutes on weekdays

- To service a customer takes 40 minutes on weekends

- There are 2 support resources on weekdays

- There is 1 support resource on weekends
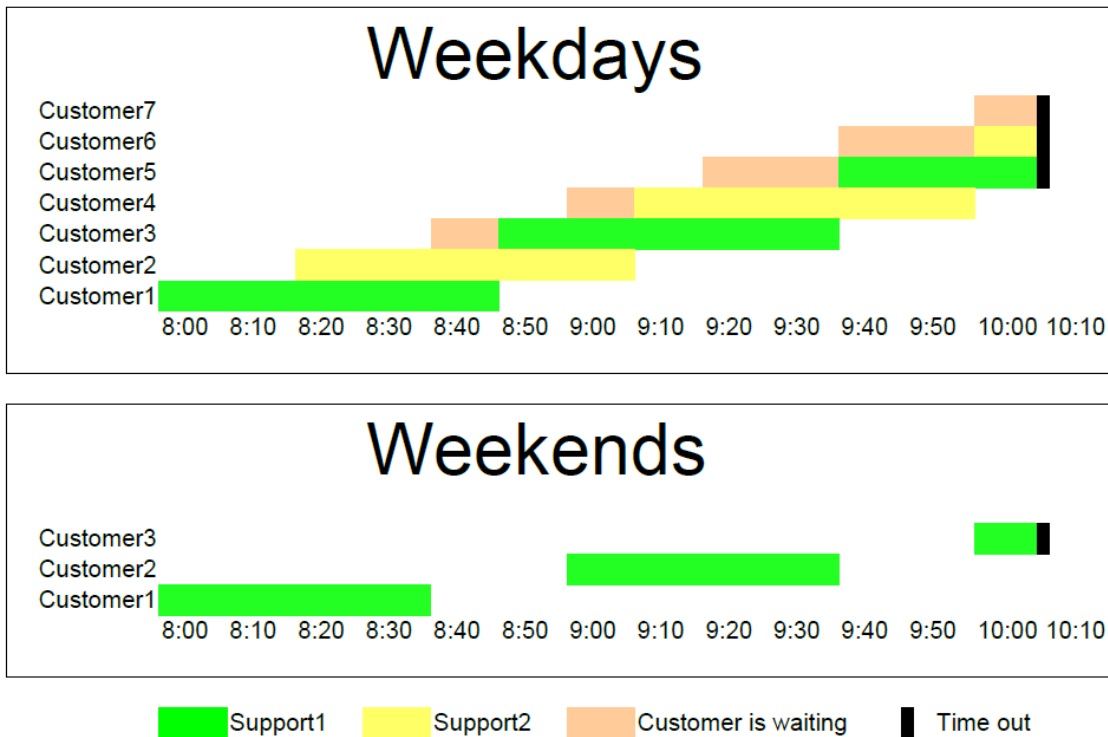
Looking at this result, when resource constraints are applied the computation is very complicated for such a simple model.

On weekdays

- 7 customers called at intervals of 20 minutes over a duration of 2 hours and 10 minutes

- 4 customer calls were terminated normally

- 2 customer calls were interrupted because of timeout

- 1 customer call was not answered

- Support1 worked continuously for 130 minutes, Support2 worked continuously for 110 minutes

On weekends

- 3 customers called at intervals of 60 minutes over a duration of 2 hours and 10 minutes

- 2 customer calls were terminated normally

- 1 customer call was interrupted because of timeout

- Support1 worked 90 minutes, in 40-minute blocks with an interval of 20 minutes between calls

Now we will see how EABPSim can help.

BPSim Configuration

In this section, we first create the Calendars, then we set up the Duration and Start parameters.

For element parameters, you can specify one or more calendars for a given parameter. However, **if any calendar is set for a parameter value, a default value (without any calendar specified) must exist**, otherwise the simulation will not work.

Clicking on the [icon] button on the Configure BPSim window toolbar will automatically check this constraint for you.

Task	Action
Create BPSim Artifact and Set Package	• Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager') • Create a Business Process Simulation Artifact named *Calendar Based Support Process Simulation* • Select the Package containing the corresponding BPMN 2.0 model All the BPMN elements will be loaded in to the Configure BPSim window.
Calendars	Click on the 'Calendars' tab, click on the New button to display the 'Event Recurrence' dialog and complete the fields as shown to create a calendar. Click on the OK button and overtype the *Calendar_n* text with a name for the calendar. Create two calendars. • In 'Event Time' set 'Start' to 08:00AM and 'End' to 05:00PM; in 'Recurrence Pattern' select 'Weekly' and select the checkboxes against 'Monday' through to 'Friday'; in 'Range of recurrence' set 'Start' to '04/1/2019' and select 'No end date'; overtype *Calendar_1* with 'Weekdays' • In 'Event Time' set 'Start' to 08:00AM and 'End' to 05:00PM; in 'Recurrence Pattern' select 'Weekly' and select the checkboxes against 'Saturday' and 'Sunday'; in 'Range of recurrence' set 'Start' to '02/1/2019' and select 'No end date'; overtype *Calendar_2* with 'Weekend'
Start & Duration	Click on the 'Scenario' tab. • Duration - input 0000 002:10:00, which means 0 days, 2 hours and 10 minutes • Start - for Weekday simulation input 1/07/2016 8:00AM, which means 1st of July, 2016, which is a Friday • Start - for Weekend simulation input 2/07/2016 8:00AM, which means 2nd of July, 2016, which is a Saturday
Customer arrivals	In the left-hand tree on the dialog, expand the 'StartEvent' group and click on *Customer calls in;* click on the 'Control' tab and, using the drop-down arrow of the 'New Parameter' fields, create these parameters: • InterTriggerTimer - Value: 100:000:000 000:00:00; Calendar: ---- (this default value is necessary) • InterTriggerTimer - Value: 000:000:000 000:20:00; Calendar: Weekdays • InterTriggerTimer - Value: 000:000:000 001:00:00; Calendar: Weekends
Processing Times	Expand the 'Activity' group and click on *Service Customer*; click on the 'Time' tab and, using the drop-down arrow of the 'New Parameter' fields, create these parameters: • ProcessingTime - Value: 100:000:000 000:00:00; Calendar: ---- (this default value is necessary) • ProcessingTime - Value: 000:000:000 000:50:00; Calendar: Weekdays • ProcessingTime - Value: 000:000:000 000:40:00; Calendar: Weekends
Resources	Expand the 'Resource' group and click on *Support*; click on the 'Resource' tab and, using the drop-down arrow of the 'New Parameter' fields, create these parameters: • Quantity - Value: 0; Calendar: ---- (this default value is necessary) • Quantity - Value: 2; Calendar: Weekdays

	• Quantity - Value: 1; Calendar: Weekends
Resource Selection (allocation)	In the 'Activity' group, click on *Service Customer* and click on the 'ResourceRole' tab. Check that the 'Resource SelectionExpression for Activity' panel shows **bpsim::getResource('Support',1)** as an expression. This expression is loaded from your BPMN model by default. You can do some advanced configurations for resource selection for a task.

Run Simulation

Weekdays

- Click on the 'Scenario' tab and set the 'Start' value to '1/7/2016'
- On the dialog Toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button drop-down arrow and select 'Standard Simulation'

A file with the name *Calendar Based Support Process Simulation - Result* is generated. This report file contains the result for a weekday simulation.

Weekends

- Click on the 'Scenario' tab and set the 'Start' value to '2/7/2016'
- On the dialog Toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button drop-down arrow and select 'Standard Simulation'

The *Calendar Based Support Process Simulation - Result* file is updated to show the result for a weekend simulation.

In each case, check the match between the result file and our analysis with pen and paper.

Car Repair Process

This example simulates the process flow of a Car Repair shop. The BPSim configuration:

- Uses a property parameter initialized by distribution to generate a random number of issues for each customer
- Applies probability to simulate:
    - Acceptance of the initial estimate or not
    - Whether new issues will be found during repair
- Increments or decrements the property parameter's value in each task
- Uses the property parameter's value on conditions of Sequences outgoing from Gateways
- Simulates the customer arrivals for a given start and duration

Create BPMN Model

Create the main process
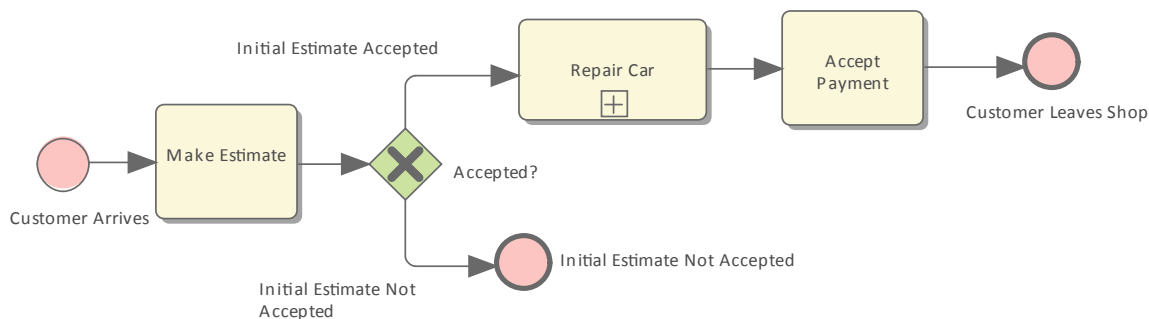
- Create a Start Event *Customer Arrives*
- Add a Sequence Flow to a target abstract task Activity *Make Estimate*
- Add a Sequence Flow to a target Exclusive Gateway *Accepted?*
- Add Sequence Flows to:
    - A target End Event *Initial Estimate Not Accepted*
    - A target subProcess *Repair Car*
- From *Repair Car*, add a Sequence Flow to a target abstract task Activity *Accept Payment*
- Add a Sequence Flow to a target End Event *Customer Leaves Shop*

Create the sub process Repair Car
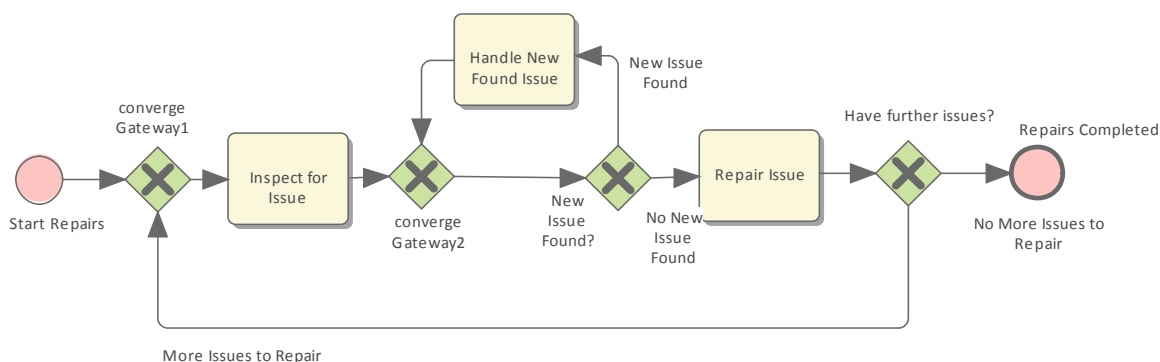
- Create a Start Event *Start Repairs*

- Add a Sequence Flow to a target Exclusive Gateway *converge Gateway1*

- Add a Sequence Flow to an abstract task Activity *Inspect for Issue*

- Add a Sequence Flow to an Exclusive Gateway *converge Gateway2*

- Add a Sequence Flow to an Exclusive Gateway *New Issue Found?*

- Add Sequence Flows to:
  - A target abstract task Activity *Handle New Found Issue*, then add a Sequence Flow back to *converge Gateway2*
  - A target abstract task *Repair Issue*, then add a Sequence Flow to a target Exclusive Gateway *Have further issues?*

- From the Gateway *Have further issues?* add Sequence Flows to:
  - A target End Event *Repairs Completed*
  - *converge Gateway1*

Configure BPSim

Task	Description
Artifact and Package	- Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager') - Create a Business Process Simulation Artifact named *IntermediateEvent - Boundary - Error* - Select the Package containing the corresponding BPMN 2.0 model All of the BPMN elements will be loaded in to the Configure BPSim window.
Start and Duration	We will simulate the processes in a car repair shop whose opening hours are from 9:00 am to 5:00 pm, which is a period of 8 hours. We also suppose that a customer walking in after 4:50 pm will not be served on that day. Therefore the simulation Start time is 9:00 am and the duration is 7 hours and 50 minutes. On the left hand panel of the 'Configure BPSim' dialog, click on the 'Scenario' item in the tree; the 'Scenario' tab is activated. - In the 'Start' field, select any date and change the time section to '9:00 AM' - Set the 'Duration' field to '0000 007:50:00' (Tips: the format is: dddd hh:mm:ss)
Customer Arrives	We will simulate a customer arriving every 24 minutes. The first customer arrives at 9:00 AM and the last arrives at 4:36 PM (the customer arriving at 5:00 PM will not be served today because that is constrained by the 'Duration' setting). With a pen and paper, we can calculate that there are 20 customers served (9:00 AM to 4:36PM = 456 minutes, number of customers: 456/24 + 1 = 19 + 1 = 20). We will verify this with the simulation result later. Expand the 'StartEvent' group and select *Customer Arrives*. Click on the 'Control' tab and the 'New Parameter' drop-down arrow, and select 'InterTriggerTimer'. In the 'Value' and 'Unit' fields type *24* and select 'min' (for '24 minutes').
Property Parameters	We suppose each customer's car might initially have a different number of issues. This could be reflected using a random number generator. BPSim provides a number of distributions to suit your needs. In this example, we use a TruncatedNormal distribution to initialize the property *noOfIssues*. Tasks *Repair Issue* and *Handle New Found Issue* will decrement and increment the value of *noOfIssues* respectively.

	• Click on the *Customer Arrives* Start Event and on the 'Properties' tab • Overtype *New Property* with *noOfIssues*, and in the 'Value' field click on the [...] button; the 'Parameter Value' dialog displays • Click on the 'Distribution' tab and select the 'TruncatedNormal' distribution type; in the fields 'Mean', 'StandardDeviation', 'Min' and 'Max' type the values 2, 1, 1 and 1000 respectively **Important Note:** Some distributions, such as 'TruncatedNormal', return a floating point value but the property is used as an integer. Setting the property's type is important, especially in condition expressions when testing with equality. For example, the condition expression *getProperty('noOfIssues') = 0* will almost never be satisfied because *noOfIssues* was initialized by a floating point distribution. ***Tip: How to customize the type for a property*** *After you create the property and set a value, click on the Save button on the toolbar so that the created property name is shown in the bottom panel of the 'Property' tab; click on the 'Value Type' drop-down arrow and select the value 'int' instead of the default 'double'.* • Click on the Activity *Repair Issue* and on the 'Properties' tab, and overtype the *New Property* text with *noOfIssues* • In the 'Value' field click on the [...] button; the 'Parameter Value' dialog displays • Click on the 'Expression' tab and, in the 'Expression' field, type *{noOfIssues} -1*; click on the OK button • Click on the Activity *Handle New Found Issue* and on the 'Properties' tab, and overtype the *New Property* text with *noOfIssues* • In the 'Value' field click on the [...] button; the 'Parameter Value' dialog displays • Click on the 'Expression' tab and, in the 'Expression' field, type *{noOfIssues} +1*; click on the OK button
Probability on Sequence Flows	We assume that 1/3 of the customers will not accept the initial estimate for repairs and 2/3 will accept it. We also assume that for 1/4 of the repairs new issues will be found, and for 3/4 of the repairs no new issues will be found. Expand the 'Gateway' group in the left-hand tree, and expand the *Accepted?* element. For the: • *Initial Estimate Accepted* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Probability'; in the 'Value' field type 0.67 • *Initial Estimate Not Accepted* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Probability'; in the 'Value' field type 0.33 In the 'Gateway' group, expand the *New Issue Found?* element. For the: • *No More Issues to Repair* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Probability'; in the 'Value' field type 0.75 • *More Issues to Repair* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Probability'; in the 'Value' field type 0.25
Condition on Sequence flows	We use an expression to return a boolean value as a Sequence Flow's condition, which plays a key role in the flow's logic.

	In the 'Gateway' group, expand the *Have Further Issues?* element. For the: • *More Issues to Repair* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Condition'; in the 'Value' field click on the [...] button to open the parameter dialog, click on the 'Expression' tab and type **{noOfIssues} != 0** in the 'Expression' field • *No More Issues to Repair* Sequence Flow, click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Condition'; in the 'Value' field click on the [...] button to open the parameter dialog, click on the 'Expression' tab and type **{noOfIssues} = 0** in the 'Expression' field Note: All the outgoing transitions from a Gateway should include 100% of the logic; for example, you would not enter **{noOfIssues} > 10** and **{noOfIssues} < 5** as condition expressions, because values in the range **[5, 10]** will not be handled by any outgoing Sequence Flows.

Run Simulation

- From the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button and select 'Standard Simulation'
- When the simulation is complete, it provides this result:

BPSim Simulation Controller : Scenario 1: Main flow	✕

Type	Message
Info	Export BPMN Model - Succeeded.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

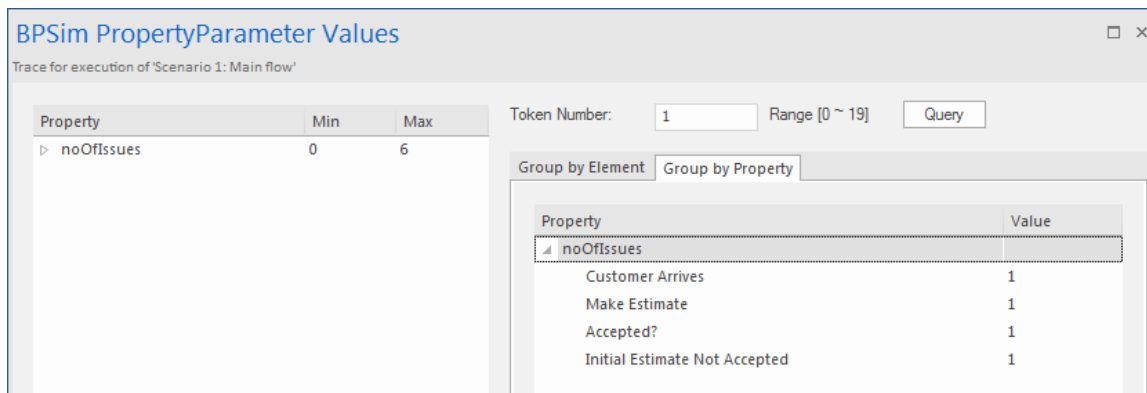| Element	Runtime Value
◢ Token Status	
<<StartEvent>>Customer Arrives	20
<<Activity>>Make Estimate	20
<<Gateway>>Accepted?	20
<<StartEvent>>Start Repairs	12
<<Gateway>>converge Gateway1	45
<<Activity>>Inspect for Issue	45
<<Gateway>>converge Gateway2	64
<<Gateway>>New Issue Found?	64
<<Activity>>Handle New Found Issue	19
<<Activity>>Repair Issue	45
<<Gateway>>Have further issues?	45
<<EndEvent>>Repairs Completed	12
<<Activity>>Accept Payment	12
<<EndEvent>>Customer Leaves Shop	12
<<EndEvent>>Initial Estimate Not Accepted	8

Token Analysis

- 20 customers arrived, matching the number we calculated manually (see *Customer Arrives* in the *Configure BPSim* table)

- 8 customers out of the 20 did not accept the initial estimate, whilst 12 of the 20 accepted and had their car repaired; these figures roughly match the 1/3 and 2/3 probabilities

- 64 tokens passed the Gateway *New Issue Found?*, of which19 had new issues and 45 did not; these figures roughly match the 1/4 and 3/4 probabilities

Analysis on individual customers

Click on the [icon] button on the toolbar to open the 'BPSim PropertyParameter Values' dialog. As there are 20 customers (tokens), you can input a value between 0 and 19 in the 'Token Number' field and click on the Query button to do some analysis:
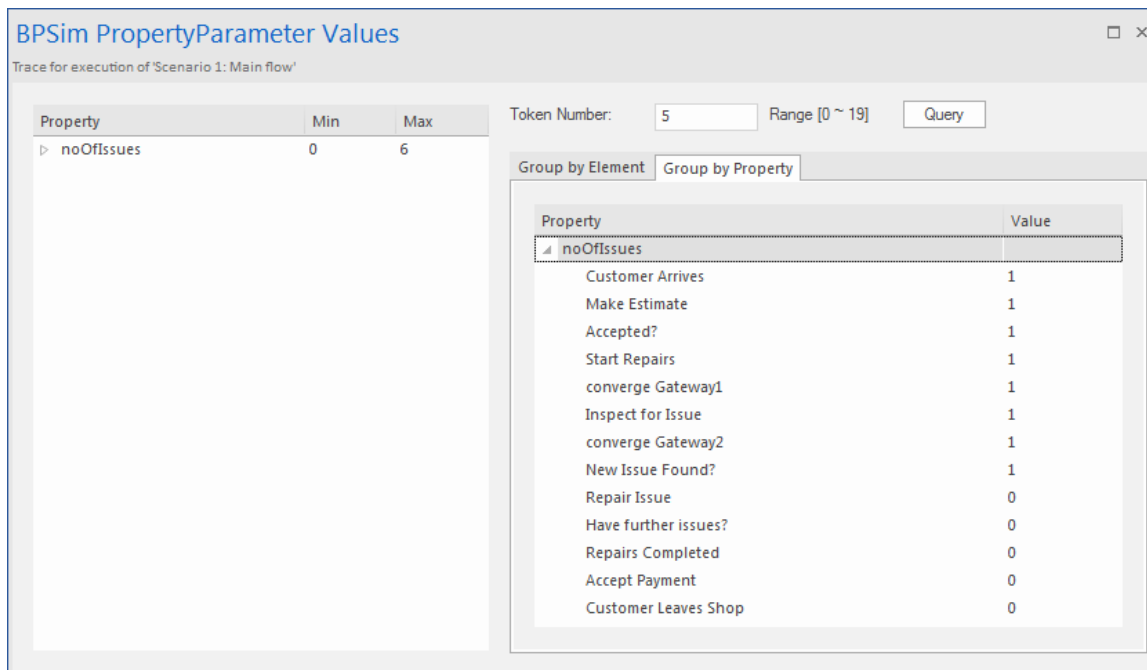
- This customer did not accept the initial estimate, as shown on the 'Group by Property' tab:

- This customer's car had only one issue, which was fixed:

- This customer's car had three known issues and three other issues were found during repair, so in total six issues got fixed (perhaps this is a really old car); switching to the 'Group by Element' tab:

BPSim PropertyParameter Values

Trace for execution of 'Scenario 1: Main flow'

Property	Min	Max
▷ noOfIssues	0	6

Token Number: 13   Range [0 ~ 19]   Query

Group by Element | Group by Property

Element	Value
▲ Customer Arrives	
noOfIssues	3
▷ Make Estimate	
▷ Accepted?	
▷ Start Repairs	
▷ converge Gateway1	
▷ Inspect for Issue	
▷ converge Gateway2	
▲ New Issue Found?	
noOfIssues	3
noOfIssues	4
noOfIssues	5
noOfIssues	6
noOfIssues	5
noOfIssues	4
noOfIssues	3
noOfIssues	2
noOfIssues	3
noOfIssues	4
noOfIssues	5
noOfIssues	4
noOfIssues	3
noOfIssues	2
noOfIssues	1

BPMN2.0 Event Examples

An Event is something that happens during the course of a Process. These Events affect the flow of the Process, usually having a cause or an impact, and in general requiring or allowing for a reaction. For example, the start of an Activity, the end of an Activity, the change of state of a document, or the arrival of a Message could all be considered to be Events.

Events allow for the description of 'event-driven' Processes. In these Processes, there are three main types of Events:

- Start Events, which indicate where a Process will start

- End Events, which indicate where the path of a Process will end

- Intermediate Events, which indicate where something happens between the start and end of a Process

Within these three types, Events can be one of two subtypes:

- Events that catch a trigger - all Start Events and some Intermediate Events are catching Events

- Events that throw a Result - all End Events and some Intermediate Events are throwing Events that could eventually be caught by another Event

In this section, we provide examples illustrating many of the commonly-used BPMN 2.0 events. In each example, we provide step-by-step BPMN modeling and BPSim configuration instructions, and a thorough analysis of the simulation result. All the examples are available in the EAExample model.

Error Event

When an Intermediate Error Event connects to the boundary of an Activity, it becomes part of an exception flow. The event is triggered when a token causes a fault name to be raised in the normal flow, going through to an Error End Event.

Create BPMN Model
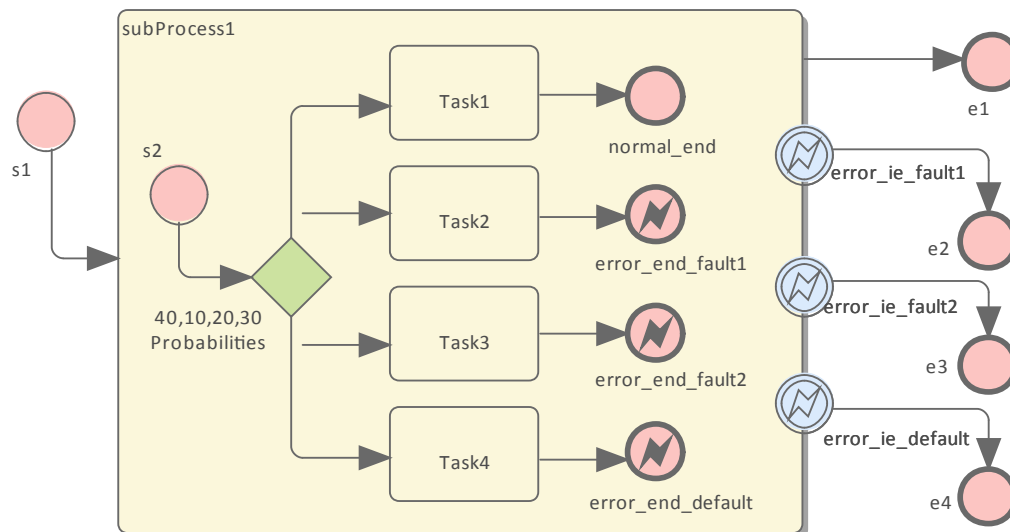
Create the main process

- Create a Start Event *s1*

- Add a Sequence Flow to a target Activity element *subProcess1;* enlarge the Activity and right-click, selecting the 'Is Expanded' option, then open the 'Properties' dialog and set 'Type' to 'subprocess'

- Add a Sequence Flow to a target End Event element *e1* ('Type' set to 'None')

- Create three Boundary Intermediate Events, dragging the elements from the Toolbox and dropping them on *subProcess1*; from the instant menus select 'Edge-Mounted' and 'Error':
    - *error_ie_fault1*; add a Sequence Flow to a target EndEvent element *e2* ('Type' set to 'None')
    - *error_ie_fault2*; add a Sequence Flow to a target EndEvent element *e3* ('Type' set to 'None')
    - *error_ie_default*; add a Sequence Flow to a target EndEvent elemenr *e4* ('Type' set to 'None')

Create the subprocess

Within the *subProcess1* Activity:

- Create a Start Event *s2*, 'Standalone' and set 'Type' to 'None'

- Create a Sequence Flow to a target Gateway element set to 'Exclusive' and with the name '*40,10,20,30 Probabilities'*

- Create Sequence Flows to four target Activiy elements of Type 'abstractTask' called:
    - *Task1*, and add a Sequence Flow to a target EndEvent called *normal_end*, 'Type' set to 'None'
    - *Task2*, and add a Sequence Flow to a target EndEvent called *error_end_fault1,* 'Type' set to 'Error'
    - *Task3*, and add a Sequence Flow to a target EndEvent called *error_end_fault2,* 'Type' set to 'Error'
    - *Task4*, and add a Sequence Flow to a target EndEvent called *error_end_default,* 'Type' set to 'Error'

Create BPMN2.0::Error elements

Create the Error elements *Fault1* and *Fault2*, which will be used as error code by Events.

- Double-click on the *error_end_fault1* element and, in the 'Properties' dialog 'BPMN2.0' tab, locate the 'errorRef' tag

- In the 'Value' field, click on the [...] button and browse to the Package containing this model

- Click on the Add New button and, in the 'Name' field, type the name *Fault1*, then click on the Save button

- Again click on the Add New button and, in the 'Name' field, type the name *Fault2*, then click on the Save button

- Click on the OK button, and again on the next OK button

Set up Events for Error Codes

- Double-click on the *error_end_fault1* element and, in the 'Properties' dialog 'BPMN2.0' tab, locate the 'errorRef' tag

- In the 'Value' field, click on the [...] button and browse to the Package containing this model

- Click on *Fault1*, then on the OK button, and again on the OK button.

Do the same for these elements:

- *error_end_fault2*, clicking on *Fault2*

- *error_ie_fault1*, clicking on *Fault1*

- *error_ie_fault2*, clicking on *Fault2*

Configure BPSim

Object	Action
Artifact & Package	- Open the Configure BPSim window ('Simulate > BPMN/BPSim > Manage > Open BPSim Manager') - Create an Artifact named 'IntermediateEvent - Boundary - Error' (in the 'Select/Create Artifact' field, click on the [...] button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
s1	- From the tree on the left of the Configure BPSim window, expand 'StartEvent' and click on 's1' - On the 'Control' tab, in the 'New Parameter...' field, click on the drop-down arrow and select 'TriggerCount' - In the 'Value' field, type '100'
Probability	From the tree on the left of the Configure BPSim window, expand 'Gateway	40,10,20,30 Probabilities'. *Tips: You can also float the Configure BPSim window, then click on the element or connectors on the BPMN diagram; the element in the Configure BPSim window will be automatically selected.* For each of the *Taskn* elements, in the 'Control' tab click on the 'New Parameter' drop-down arrow and select 'Probability', then type the corresponding value in the 'Value' field: - For Task1 type '0.4' - For Task2 type '0.1' - For Task3 type '0.2' - For Task4 type '0.3'

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Controller' dialog
- Click on the Run button and select 'Standard Simulation'
- The results of the simulation resemble this:

Analysis:

The Probability set on the Sequence Flows outgoing from *40,10,20,30 Probabilities* are 0.4, 0.1, 0.2 and 0.3 respectively.

- 36 out of 100 passes finished at *normal_end*, which flowed to *e1*
- 11 out of 100 passes finished at *error_end_fault1*, which triggered *error_ie_fault1* by the ErrorRef *Fault1,* and the exception flowed to *e2*
- 23 out of 100 passes finished at *error_end_fault2*, which triggered *error_ie_fault2* by ErrorRef *Fault2,* and the exception flowed to *e3*

- 30 out of 100 passes finished at *error_end_default*, which triggered *error_ie_default* because they did not set ErrorRef and the exception flowed to *e4*

The numbers 36, 11, 23 and 30 add up to 100, which was set as the TriggerCount in *s1,* so they match the 100% probability

Escalation Event

In BPMN, Escalation is the non-interrupting counterpart of Error, with similar throw-catch behavior. Unlike Error, however, the normal flow and exception flow exits from the Activity are parallel paths, not alternative.

Create BPMN Model
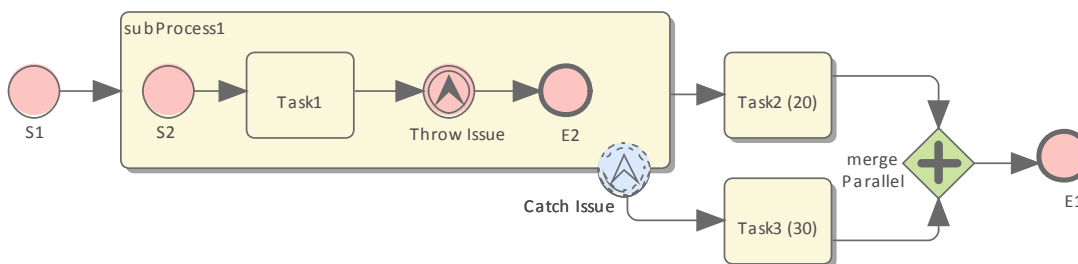
Create the main process

- Create a Start Event *S1*

- Add a Sequence Flow to a target Activity *subProcess1;* enlarge the Activity and right-click, selecting the 'Is Expanded' option, then open the 'Properties' dialog and set 'Type' to 'subprocess'

- Add a Sequence Flow to a target abstractTask Activity element *Task2 (20)* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')

- Add a Sequence Flow to a target parallel Gateway element *merge Parallel* (open the 'Properties' dialog and set the 'Type' field to 'parallel')

- Add a Sequence Flow to a target End Event *E1*

- On *subProcess1*, add a boundary non-interrupting Escalation Event *Catch Issue* (drag the 'Intermediate Event' icon onto *subProcess1*, and from the instant menus select 'Edge Mounted' and 'Escalation'; double-click on the element to display the 'Properties' dialog and add the name, then in the 'Type' field select 'Boundary Non-Interrupting > Escallation')

- Add a Sequence Flow to a target abstractTask Activity element *Task3 (30)* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')

- Add a Sequence Flow to the target element *merge Parallel*

Create the sub process

- Within (or under) *subProcess1*, create a Start Event *S2*

- Add a Sequence Flow to a target abstractTask Activity element *Task1* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')

- Add a Sequence Flow to a target Throwing Escalation Intermediate Event *Throw Issue* (open the 'Properties' dialog and in the 'Type' field select 'Throwing > Escalation')

- Add a Sequence Flow to a target End Event *E2*

Create BPMN2.0::Escalation elements

From the Diagram Toolbox, expand the 'BPMN 2.0 Types' page, drag the 'Escalation' icon onto the diagram, and give the element the name *Escalation1*; this will be used as the escalation code by the Events.

Set up Events for Escalation Codes:

- Double-click on *Throw Issue* and in the 'Value' field for the escalationRef tag click on the ⬚ icon and locate and select *Escalation1*

- Double-click on *Catch Issue* and, again, in the 'Value' field for the escalationRef tag click on the ⬚ icon and locate and select *Escalation1*

(The exception flow exits from the Activity are parallel.)

Configure BPSim

Task	Action
Artifact & Package	• Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager') • Create an Artifact named 'Escallation Event Simulation' (in the 'Select/Create Artifact' field, click on the ⬚ button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
Trigger Count of Start Event	• From the tree on the left of the Configure BPSim window, expand 'StartEvent' and click on *S1* • On the 'Control' tab, in the 'New Parameter...' field, click on the drop-down arrow and select 'TriggerCount' • In the 'Value' field, type '1'
ProcessingTime	• In the left-hand tree expand 'Activity' and click on *Task2 (20)*; in the 'Value' field for 'Processing Time' type '20' and in the 'Unit' field type 's' (for 20 seconds) • Click on *Task3 (30)*; in the same way, set 'ProcessingTime' to 30 seconds
dummyVariable for Trace	In order to show the exact trace of a given token, you must set a dummy variable on *S1*. • In the left-hand hierarchy click on *S1*, and on the 'Properties' tab overtype the *New Property* text with the name of a variable (such as 'dummyVariable') • In the 'Value' field, click on the ⬚ button and, on the '<<StartEvent>>S1 : <variable name>' dialog click on 'Numeric' and type a 'Constant Numeric' value of '0'; click on the OK button

Run Simulation

- On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog

- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

- After simulation, click on the 🖼 button on the tool bar to display the 'BPSim PropertyParameter Values' dialog

- Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable' (or the name you assigned to the variable)

BPSim PropertyParameter Values

Trace for execution of 'Escalation Event Simulation'

Property	Min	Max
▷ dummyVariable	0	0

Token Number: 0   Range [0 ~ 0]   Query

Group by Element | **Group by Property**

Property	Value
⊿ dummyVariable	
S1	0
S2	0
Task1	0
Throw Issue	0
Task3 (30)	0
E2	0
Task2 (20)	0
merge Parallel	0
merge Parallel	0
E1	0

Analysis:

Unlike *Error*, the normal flow and exception flow exits from *subProcess1* are not alternative paths but parallel. This feature can be easily discovered from the trace:

- *E2* and *Task2 (20)* still get traversed after*Task3 (30)* started
- *E1* was reached after *mergeParallel* was traversed twice

Event Sub-Process

Event Sub-Processes enable your system to handle an Event within the context of a given Sub-Process or Process. An Event Sub-Process always begins with a Start Event; it is not instantiated by normal control flow, but only when the associated Start Event is triggered. Event Sub-Processes are self-contained and MUST not be connected to the rest of the Sequence Flows in the Sub-Processes.

- If the isInterrupting attribute of its Start Event is set, an Event Sub-Process cancels execution of the enclosing Sub-Process

- If the isInterrupting attribute is not set, execution of the enclosing Sub-Process continues in parallel to the Event Sub-Process

In this example, we demonstrate how Interrupting and Non-Interrupting Event Sub-Processes affect the life line of the enclosing Sub-Process and Process.

Create BPMN Model
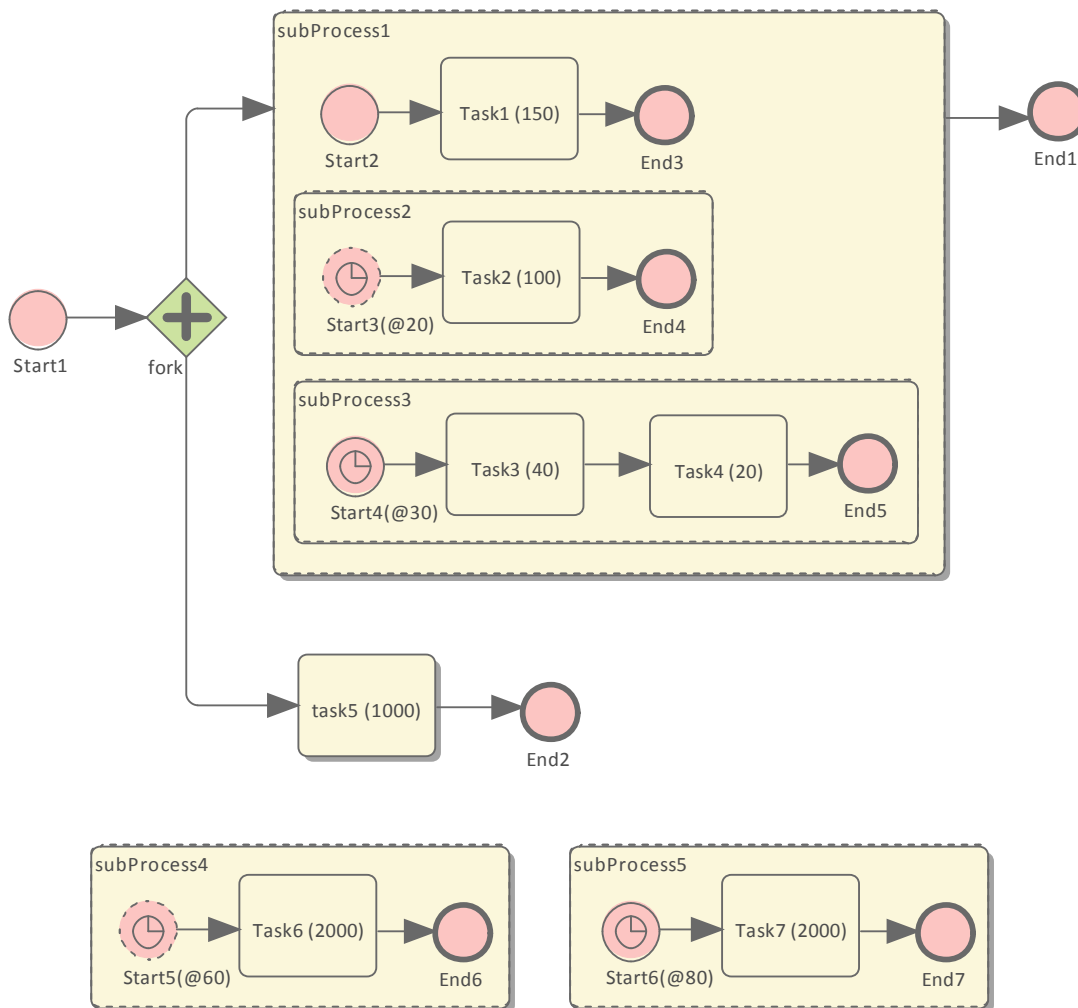
Model the main process

- Create a StartEvent *Start1*

- Add a Sequence Flow to a target Parallel Gateway element *fork*

- Add a Sequence Flow to

- a Sub-Process *subProcess1*, and from that add a *Sequence Flow* to a target End Event element *End1*
- an abstract Task *Task5*, and from that add a Sequence Flow to a target End Event element *End2*

Tips on how to model an Event Sub-Process

- Drag an Activity from the 'BPMN2.0 - Business Process' toolbox onto the diagram

- Double-click on the Activity to display the 'Properties' dialog and, in the 'Type' field, select 'subProcess'; set 'triggeredByEvent' to 'true' and click on the OK button

- Right-click on the element and select the 'Is Expanded' option; this will display the element name on the top left corner

Model the Event Sub-Processes for the main process

- Create an Event Sub-Process *subProcess4*
  - Create a Timer Start Event *Start5(@60)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Non-Interrupting > Timer'; click on the OK button
  - Add a Sequence Flow to a target abstract task Activity *Task6(2000)*
  - Add a Sequence Flow to a target End Event element *End6*

- Create an Event Sub-Process *subProcess5*
  - Create a Timer Start Event *Start6(@80)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Interrupting > Timer'; click on the OK button
  - Add a Sequence Flow to a target abstract task Activity *Task7(2000)*
  - Add a Sequence Flow to a target End Event element *End7*

Model the Sub-Process subProcess1 and the enclosed Event Sub-Processes

- Create a StartEvent *Start2*
  - Add a *Sequence* Flow to a target abstract task Activity *Task1(150)*
  - Add a Sequence Flow to a target End Event *End3*

- Create an Event Sub-Process *subProcess2*
  - Create a Timer Start Event *Start3(@20)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Non-Interrupting > Timer'
  - Add a Sequence Flow to a target abstract task Activity *Task2(100)*
  - Add a Sequence Flow to a target End Event element *End4*

- Create an Event Sub-Process *subProcess3*
  - Create a Timer Start Event *Start4(@30)*, then double-click on it to display the 'Properties' dialog and, in the 'Type' field, select 'Event Sub-Process Interrupting > Timer'
  - Add a Sequence Flow to a target abstract task Activity *Task3(40)*
  - Add a Sequence Flow to a target abstract task Activity *Task4(20)*
  - Add a Sequence Flow to a target End Event element *End5*

Configure BPSim

Using this table, we create the Artifact in the configuration Package and configure the parameter values of each element.

Task	Action
Create Artifact	- Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager') - Create an Artifact named 'Event Sub Process Interrupting and Non-Interrupting' (in the 'Select/Create Artifact' field, click on the ⬚ button and select its parent Package and click on the Add New button, then type in the

	element name and click on the Save button and the OK button)
	Then all the BPMN elements will be loaded in to the Configure BPSim window.
InterTriggerTimer for Start Events in Event Sub-Process	From the tree on the left of the 'Configure BPSim' dialog, expand 'StartEvent'.
	For each of the elements listed here, on the 'Control' tab click on the drop-down arrow in the 'New Parameter...' field and select the parameter 'InterTriggerTimer'.
	Click on the `...` button in the 'Value' field to open the 'Parameter' dialog and select 'Constant > Numeric', then type in the value and select 'seconds'.
	• Start3(@20): 20 seconds
	• Start4(@30): 30 seconds
	• Start5(@60): 60 seconds
	• Start6(@80): 80 seconds
ProcessingTime for Tasks	From the tree on the left of the Configure BPSim window, expand 'Activity'.
	For each of the elements listed here, on the 'Time' tab click on the drop-down arrow in the 'New Parameter...' field and select the parameter 'ProcessingTime'. Click on the `...` button in the 'Value' field to open the 'Parameter' dialog and select 'Constant > Numeric', then type in the value and select 'seconds'.
	• Task1 (150): 150 seconds
	• Task2 (100): 100 seconds
	• Task3 (40): 40 seconds
	• Task4 (20): 20 seconds
	• Task5 (1000): 1000 seconds
	• Task6 (2000): 2000 seconds
	• Task7 (2000): 2000 seconds

Run Simulation

• On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog

• Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

BPSim Simulation Controller : Event Sub-Process Interrupting & Non-Interrupting   ✕

Type	Message
Info	Export BPMN Model - Succeeded.
Info	Loading BPMN Model - Succeeded.
Info	Simulation Engine Initialized
Info	Simulation Started.
Info	Simulation Finished.
Info	Simulation Result Generated.

Element	Runtime Value
⊿ Token Status	
<<StartEvent>>Start1	1
<<Gateway>>fork	1
<<Activity>>task5 (1000)	1
<<StartEvent>>Start2	1
<<Activity>>Task1 (150)	1
<<StartEvent>>Start3(@20)	1
<<Activity>>Task2 (100)	1
<<StartEvent>>Start4(@30)	1
<<Activity>>Task3 (40)	1
<<StartEvent>>Start5(@60)	1
<<Activity>>Task6 (2000)	1
<<Activity>>Task4 (20)	1
<<StartEvent>>Start6(@80)	1
<<Activity>>Task7 (2000)	1
<<EndEvent>>End7	1

Analysis

Reading the results, it might not be completely straightforward to see what has happened; however, if we draw the lifeline for each task in a Timing diagram, it becomes clearer.
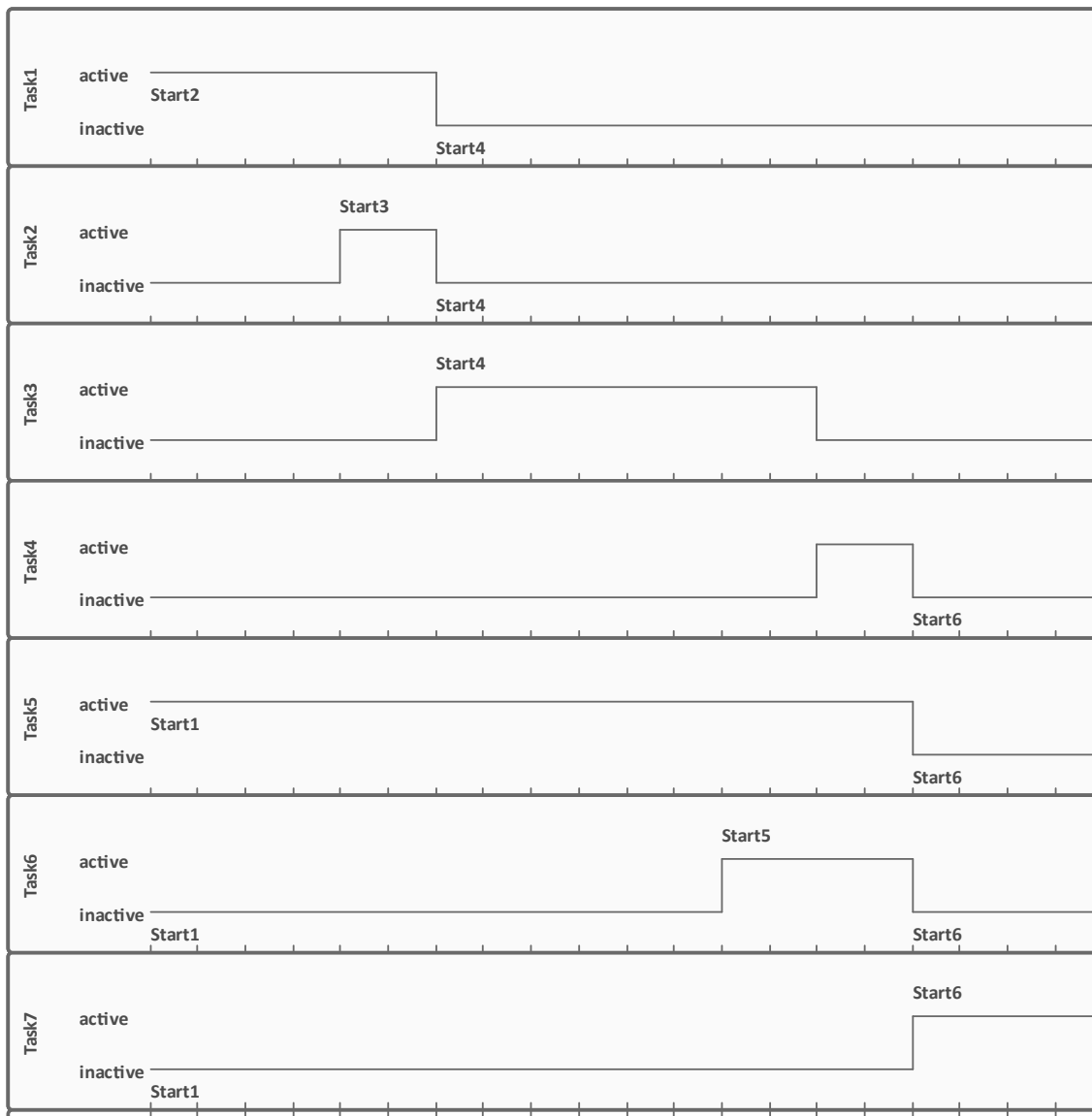
- Event *Start3(@20)* is Non-Interrupting, it did not stop *Task1* at 20 seconds
- Event *Start4(@30)* is Interrupting, it stopped *Task1* and *Task2* at 30 seconds; it did not affected *Task5* because *Task5*'s enclosing process (main process) level is higher than *Start4*'s enclosing Sub-Process (*subProcess1*)
- Event *Start5(@60)* is Non-Interrupting, it started *Task6* at 60 seconds without affecting *Task3* or *Task5*
- Event *Start6(@80)* is Interrupting, it started *Task7* at 80 seconds and interrupted the running Tasks (*Task4*, *Task5*, *Task6*) that were in the same or lower level of its enclosing process
- Only *End7* is reached as expected

Fibonacci Number Generator with Link Event

A Link Event is a mechanism for connecting two sections of a Process. Link Events can be used:

- To create looping scenarios, as generic 'Go To' objects within the Process level

- To avoid long Sequence Flow lines; paired Link Events can be used as 'off-page' connectors for printing a Process across multiple pages

The use of Link Events is limited to a single Process level (that is, they cannot link a parent Process with a sub-Process).

There can be multiple source Link Events, but there can only be one target Link Event.

- The target Link Event marker is unfilled, to 'catch' from the source link

- The source Link Event marker is filled to 'throw' to the target link

When the EABPSim Execution Engine is running the simulation, the source-target Link Events are paired by element NAME, so they cannot be empty.

Create BPMN Model
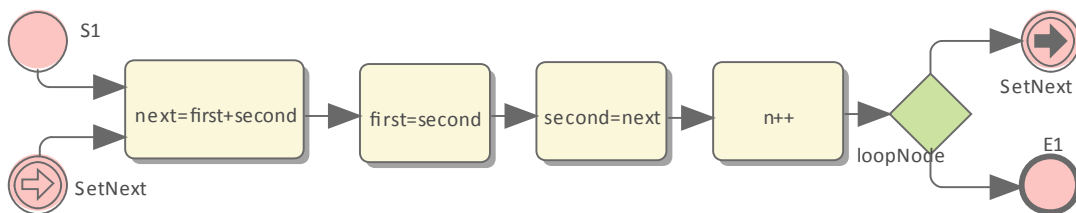
- Create a StartEvent *S1*

- Add a Sequence Flow to a target abstractTask Activity element *next=first+second* (open the 'Properties' dialog and set the 'Type' field to 'abstractTask')

- Add a Sequence Flow to a target abstractTask Activity element *first=second*

- Add a Sequence Flow to a target abstractTask Activity element *second=next*

- Add a Sequence Flow to a target abstractTask Activity element *n++*

- Add a Sequence Flow to a target exclusive Gateway element *loopNode* (on the instant menu, select 'Exclusive')

- Add a Sequence Flow to each of these target elements:
  - A Throwing Link Intermediate Event element *SetNext* (open the 'Properties' dialog and set the 'Type' field to 'Throwing > Link') and
  - An End Event element *E1*

- Create a Catching Link Intermediate Event element *SetNext* (open the 'Properties' dialog and set the 'Type' field to 'Catching > Link')

- Add a Sequence Flow to the target element *next=first+second*

Configure BPSim

We will use Property Parameters to define how the sequence flow forms a loop during which a Fibonacci Number will be generated. The loop mechanism is implemented via the pair of Link Events.

Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager')

Task	Action
Element: S1	In the element type list on the left, expand the Start Event group and click on *S1*.
	Click on the 'Control Tab' and on the 'New Parameter' drop-down arrow; select

	'TriggerCount'. In the 'Value' field type '1'. **Click on the 'Properties' tab** Overtype the *New Property* text to create these properties: • N - and type '10' in the 'Value' field as the total number of Fibonacci Numbers to be generated • first - and type '1' in the 'Value' field • second - and type '1' in the 'Value' field • n - and type '0' in the 'Value' field as the *n*th new Fibonacci Number
Element: next=first+second	In the element type list, expand the Activity group and click on *next=first+second*. Click on the 'Properties' tab and overtype the *New Property* text with 'next '. In the 'Value' field, click on the ⟦...⟧ button, click on the 'Expression' tab and type the expression '{first}+{second}'. Click on the OK button.
Element: first=second	In the element type list, in the Activity group click on *first=second*. Click on the 'Properties' tab and overtype the *New Property* text with 'first '. In the 'Value' field, click on the ⟦...⟧ button, click on the 'Expression' tab and type the expression '{second}'. Click on the OK button.
Element: second=next	In the element type list, in the Activity group click on *second=next*. Click on the 'Properties' tab and overtype the *New Property* text with 'second'. In the 'Value' field, click on the ⟦...⟧ button, click on the 'Expression' tab and type the expression '{next}'. Click on the OK button.
Element: n++	In the element type list, in the Activity group click on *n++*. Click on the 'Properties' tab and overtype the *New Property* text with 'n'. In the 'Value' field, click on the ⟦...⟧ button, click on the 'Expression' tab and type the expression '{n}+1'. Click on the OK button.
Conditions of Gateway	In the element type list, expand the Gateway group and the LoopNode element and click on *SetNext*. Click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Condition'. In the 'Value' field, click on the ⟦...⟧ button, click on the 'Expression' tab and type the expression '{n} <={N}'. Click on the OK button. Now click on *E1*. Click on the 'Control' tab and on the 'New Parameter' drop-down arrow, and select 'Condition'.

	In the 'Value' field, click on the [...] button, click on the 'Expression' tab and type the expression '{n} > {N}'. Click on the OK button.

Run Simulation

- On the 'Configure BPSim' dialog, in the toolbar, click on the 'Run' icon; the 'BPSim Simulation Controller' dialog displays

- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

- When the simulation is complete, click on the [icon] icon in the toolbar; the 'BPSim PropertyParameter Values' dialog displays

- Click on the Query button and on the 'Group by Element' tab, and expand 'next=first+second'; all the attributes's snapshot values are listed

- Apply a filter 'next' (right-click on the list header, select 'Toggle Filter Bar' and type 'next' under the 'Element' heading); the results will resemble this image:
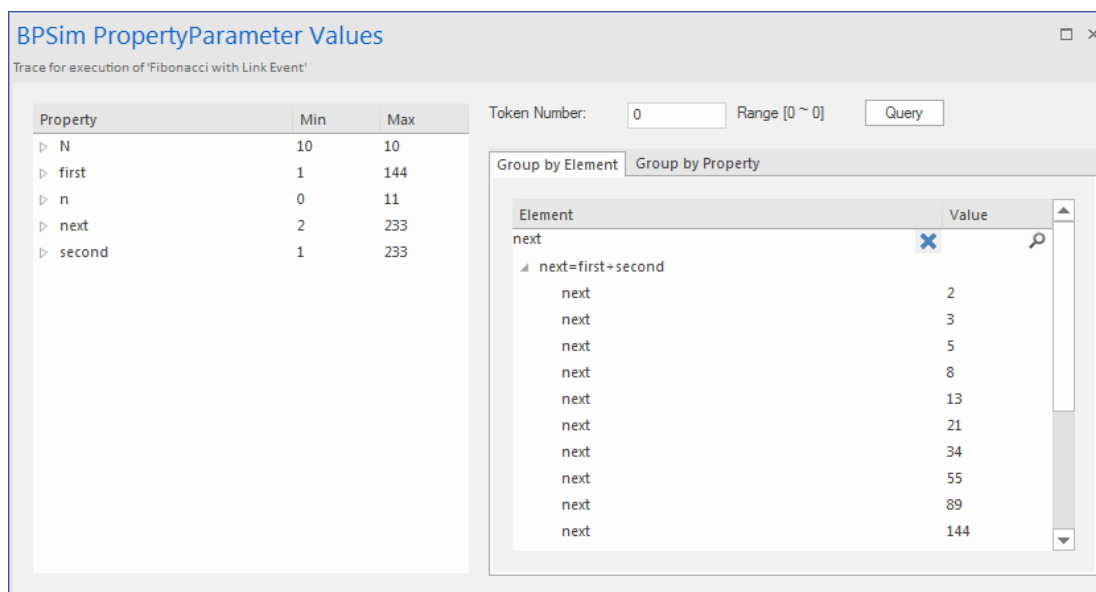
Ten more Fibonacci numbers are generated:

   2,3,5,8,13,21,34,55,89,144

Message Event

When used in normal sequence flow, the Message Event can be used to either send or receive a message.

- When sending a message to a participant, the values of all property parameters are copied; once the message is sent the token continues along the sequence flow
- When receiving a message, the event is triggered when a message is received.

This example demonstrates the Message Event features. We will first create the BPMN model, then configure BPSim and run the simulation.

Create BPMN Model
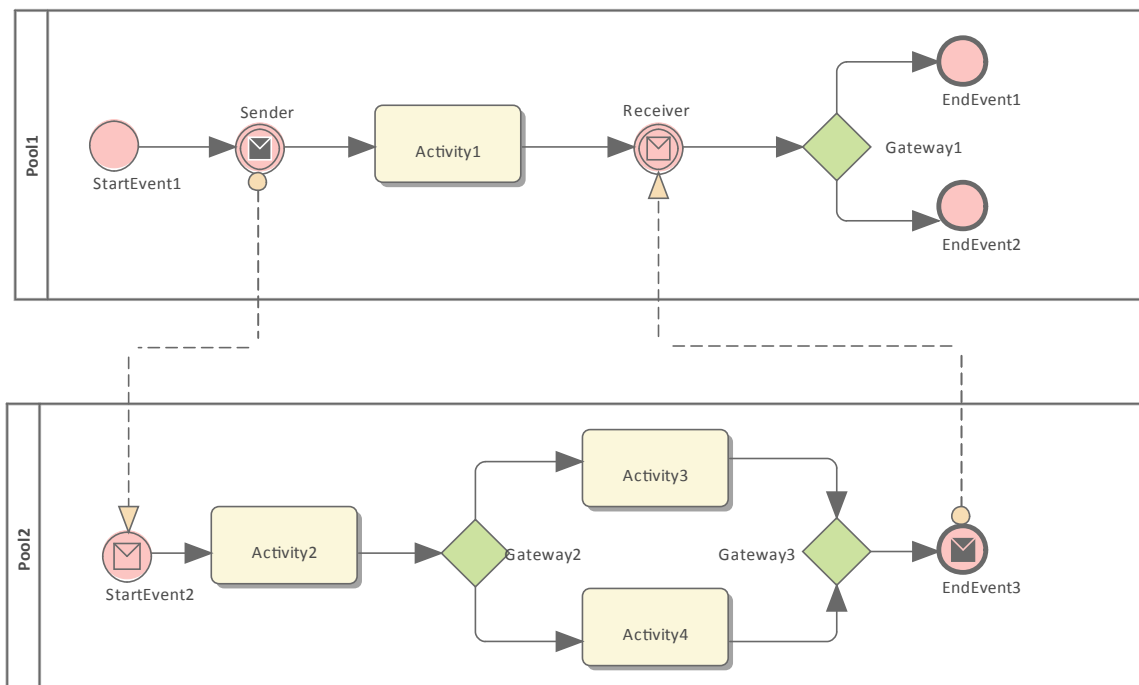
Sequence

Pool1

- The token starts from StartEvent1
- On receiving the token, Sender (a Throwing Intermediate Message Event) creates a message and copies the current property values to the message
- Sender sends the *message* to the 'To' participant (Pool2, StartEvent2)
- Sender forwards the *token* along its sequence flow, as far as Receiver
- The token waits at Receiver until a message arrives

Pool2

- StartEvent2 receives a message and starts a token
- StartEvent2 copies the values from the message and sets these in the token
- StartEvent2 forwards the token along its sequence flow until EndEvent3
- EndEvent3 creates a message and copies the current property values to the message

- EndEvent3 sends the message to the 'To' participant (Pool1, Receiver)

Pool1 continued
- The waiting Receiver is awakened and the property values are updated from the arriving message

Create Diagram
- Create a BPMN 2.0 Collaboration diagram
- Select the option 'Create this diagram within a new CollaborationModel'
- Create *Pool1* and *Pool2* by dragging the 'Pool' icon from the Toolbox onto the diagram

Within Pool1
- Create a Start Event of type 'None', named *StartEvent1*
- Add a Sequence Flow to the target Intermediate Event of type 'Throwing Message', called *Sender*
- Add a Sequence Flow to the target Activity of type 'abstract', called *Activity1*
- Add Sequence Flows to the target:
    - End Event of type 'None', called *EndEvent1*
    - End Event of type 'None', called *EndEvent2*

Within Pool2
- Create a Start Event of type 'Message', called *StartEvent2*
- Add a Sequence Flow to the target Activity of type 'abstract', called *Activity2*
- Add a Sequence Flow to the target Gateway of type 'Exclusive', called *Gateway2*
- Add Sequence Flows to the target:
    - Activity of type 'abstract', called *Activity3*
    - Activity of type 'abstract', called *Activity4*
- Add Sequence Flows from *Activity3* and *Activity4* to the target Gateway of type 'Exclusive', called *Gateway3*
- Add a Sequence Flow to the End Event of type 'Message', called *EndEvent3*

Message Flows
- Add a Message Flow from *Sender* to *StartEvent2*
- Add a Message Flow from *EndEvent3* to *Receiver*

Configure BPSim

In order to show the Message Flow's ability to carry values, we create a Property Parameter 'M1' and change its value in each Activity. We then use the value of 'M1' as part of the expression for the condition of the Sequence Flow.

Task	Description
Create Artifact and Package	- Open the Configure BPSim window ('Simulate > BPMN/BPSim > Manage > Open BPSim Manager') - Click on the [...] button in the 'Select/Create Artifact' field and create an Artifact called 'MessageEvent-StartEvent-IntermediateEvent-EndEvent' - In the 'Select Package' field select the Package containing the model All the BPMN elements from the model are loaded into the Configure BPSim

	window.
Property Values	We will give the Property Parameter 'M1' an initial value of 10 at *StartEvent1*. Then we change the value as the token flows through the processes and the value is copied across participants. In the element list on the left of the dialog: • Expand the 'StartEvent' group, click on *StartEvent1* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the [...] button and select 'Constant' and 'Numeric, and type '10' in the 'Constant Numeric' field • Expand the 'Activity' group, click on *Activity1* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} + 100' in the 'Expression' field • Click on *Activity2* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} + 10' in the 'Expression' field • Click on *Activity3* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} + 10' in the 'Expression' field • Click on *Activity4* and on the 'Properties' tab, and overtype *New Property* with 'M1'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} + 1' in the 'Expression' field *Tip: The format of* {PropertyName} *is a convenient short form of* getProperty("PropertyName").
Control Parameters	We only need one token in this simulation for evaluating the behavior of the model. • In the expanded 'StartEvent' group, click on *StartEvent1* and the 'Control' tab; click on the 'New parameter' drop-down arrow and select 'Trigger Count', and type in a 'Value' of '1' Now set up the conditions for the Gateways' outgoing Sequence Flows. In the element list on the left of the dialog, expand the 'Gateway' group: • Expand *Gateway1*, click on *EndEvent1* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} >= 50' in the 'Expression' field • Click on *EndEvent2* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} < 50' in the 'Expression' field • Expand *Gateway2*, click on *Activity3* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} >= 15' in the 'Expression' field • Click on *Activity4* and on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'Condition'; in the 'Value' field click on the [...] button and select 'Expression', and type '{M1} < 15' in the 'Expression' field

Run Simulation

- On the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays

- Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'; the simulation starts

- When the simulation is finished, click on the ![button] button; the 'Property Parameter Values' dialog displays, tracing the values for properties during simulation

- In the 'Token Number' field type '0', then click on the Query button and on the 'Group by Property' tab
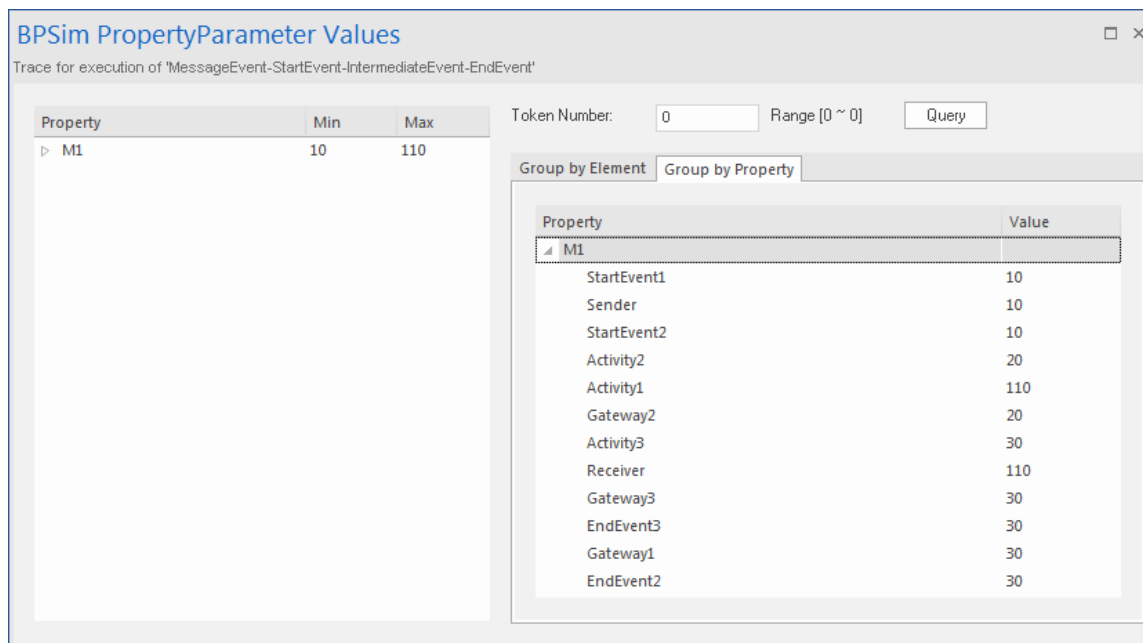
Analysis

As the 'ProcessingTime' of *Activity1* was set as a distribution value, it turns out that:

- [Process1]'s 'M1' value after *Pool1.StartEvent1* is '10', as expected

- *[Process2] *Pool2.StartEvent2*'s 'M1' value is '10'; this value is carried from a message sent by *Pool1.Sender*

Now there are actually two 'M1's - *Process1.M1* and *Process2.M1*

- [Process2] *Pool2.Activity2* increased *Process2.M1* by 10; *[Process2.M1 == 20]*

- [Process1] *Pool1.Activity1* increased *Process1.M1* by 100; *[Process1.M1 == 110]*

- [Process2] The condition expressions are evaluated; as '20 > 15', the token will flow to *Activity3 [Process2.M1 == 20]*

- [Process2] *Pool2.Activity3* increased *Process2.M1* by 10; *[Process2.M1 == 30]*

- [Process1] *Pool1.Receiver* is reached and waiting *[Process1.M1 == 110]*

- [Process2] *Pool2.Gateway3* serves as a Merge node and continues to *EndEvent3 [Process2.M1 == 30]*

- *[Process1] *Pool1.Receiver is* woken up by a message(carrying *M1 == 30*) and *Process1.M1*'s value changes from 110 to 30

- [Process1] The condition expressions are evaluated; as '30 < 50', the token will flow to *EndEvent2 [Process1.M1 == 30]*

Notes

- The lines marked with an asterisk (*) are the effects of Message Flows
- The order within a process is defined; however, the order between two processes is not always predictable
- The Throwing Message Event forks another process; the Catching Message serves as thread synchronization

Signal Events

A Signal Event provides the facility of loosely coupling 'throwers' and 'catchers' by publish-subscribe integration. A 'thrower' will broadcast a signal rather than addressing it to a particular process; any process listening for that particular event could trigger a new instance using a Signal Start Event.

A Signal can be thrown from either a throwing Intermediate Event or a throwing End Event, and can be caught in a Start Event or a catching Intermediate Event (including a boundary Signal Event).

In this example, we demonstrate these Signal Events and their impact on the lifelines of tasks, via BPSim parameter settings.

- Start Signal Event:
    - *Start By Signal1* in top level process (Pool1)
    - *Start By Signal2* Interrupting in event sub-process *eventSubProcess2*
    - *Start By Signal1* Non Interrupting in event sub-process *eventSubprocess1*

- Throwing Intermediate Signal Event:
    - *Broadcast Signal1*

- Catching Intermediate Signal Event:
    - *Receive Signal1 (normal)*
    - *Receive Signal2 (normal)*
    - *Receive Signal2 (boundary Interrupting)*
    - *Receive Signal1 (boundary non-interrupting)*
    - *Receive Signal2 (in Event Gateway)*

- End Signal Event:
    - *End By Throwing Signal2*

Create BPMN Model

In order to demonstrate the ability to communicate across processes via a Signal Event, we create a Collaboration model with a main Pool and a process in another Pool (*Pool1*).
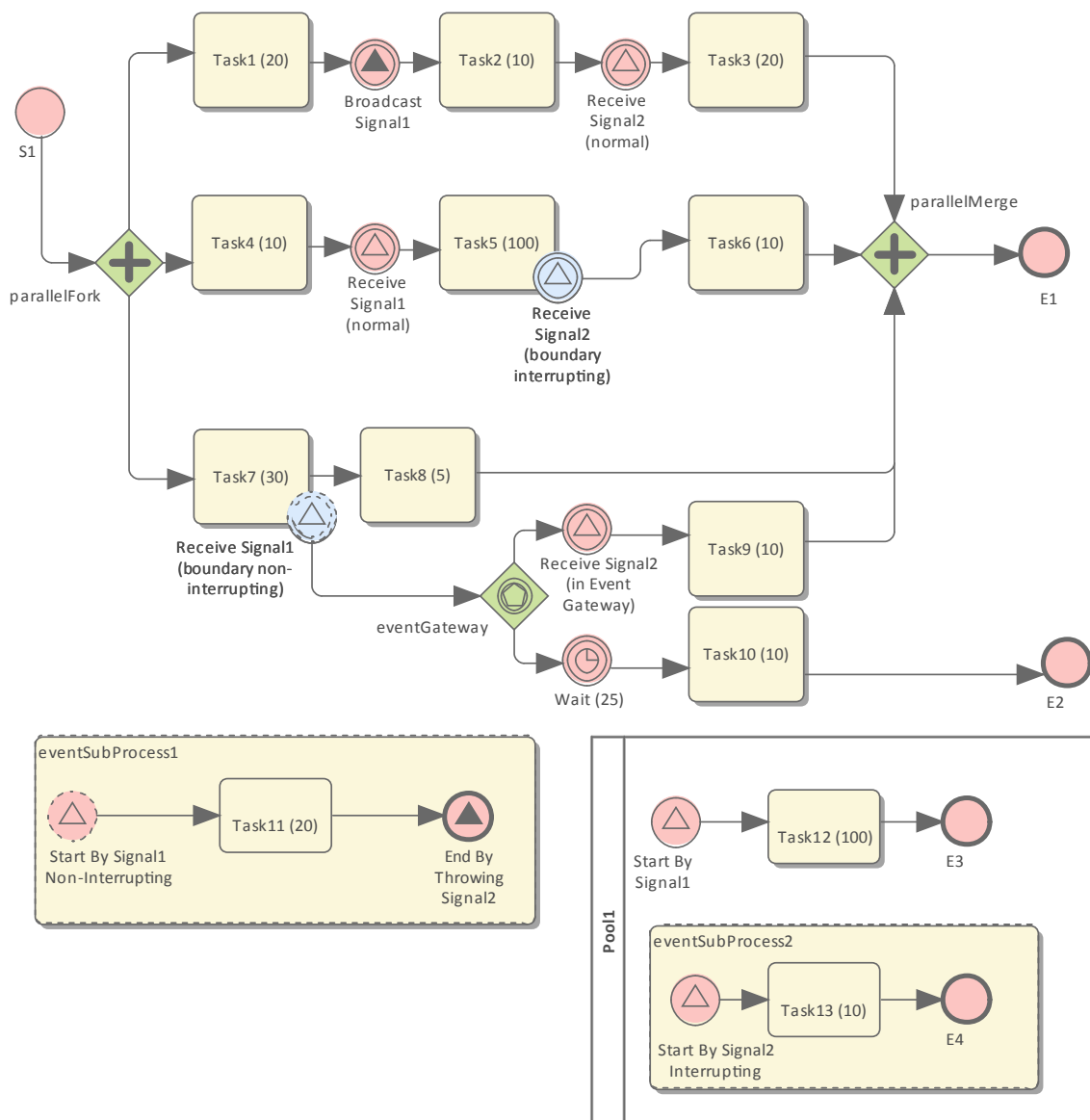
Create the Collaboration and main process

Create a new BPMN2.0 Collaboration diagram called *CollaborationForTestingSignalEvents*, (choose the option 'Create this diagram within a new Collaboration Model'). Right-click on the diagram name in the Project Browser and select the 'Encapsulate Process' option.

A Pool *PoolMain* and a process *BusinessProcess_PoolMain* are created, and these tags are set with the automatic values:

- *CollaborationForTestingSignalEvents.mainPool* is set to *PoolMain*
- *PoolMain.processRef* is set to *BusinessProcess_PoolMain*

Create the elements for the main process

Create a Start Event *S1* and add a Sequence Flow to a Fork Parallel Gateway *parallelFork*

Add Sequence Flows to:

- An Abstract Task *Task1 (20)* and then add this chain of Sequence Flows:
  - To a Throwing Intermediate Signal Event *Broadcast Signal1*
  - Then to an Abstract Task *Task2 (10)*
  - Then to a Catching Intermediate Signal Event *Receive Signal2 (normal)*
  - Then to an Abstract Task *Task3 (20)*

- Then to a Merge Parallel Gateway *parallelMerge*
- Then to an End Event *E1*

- An Abstract Task *Task4 (10)* and then add this chain of Sequence Flows:
  - To a Catching Intermediate Signal Event *Receive Signal1 (normal)*
  - Then to an Abstract Task *Task5 (100),* on which you create a Boundary Interrupting Catching
    Intermediate Signal Event *Receive Signal2 (boundary interrupting)*
  - Then to an Abstract Task *Task6 (10)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

- An Abstract Task *Task7 (30),* and then add this chain of Sequence Flows:
  - To an Abstract Task *Task8 (5)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

On *Task7 (30),* create a Boundary Non-interrupting Catching Intermediate Signal Event *Receive Signal1 (boundary non-interrupting)*. Add a Sequence Flow to an Event Gateway *eventGateway*, and to that add Sequence Flows to:

- A Catching Intermediate Signal Event *Receive Signal2 (in Event Gateway),* and then this chain of Sequence Flows:
  - To an Abstract Task *Task9 (10)*
  - Then to the earlier Merge Parallel Gateway *parallelMerge*

- A Catching Intermediate Timer Event *Wait (25),* and then this chain of Sequence Flows:
  - To an Abstract Task *Task10 (10)*
  - Then to an End Event *E2*

Create an Event sub-process (triggered by a non-interrupting Start Signal Event) in the main process

- Create an Activity *eventSubProcess1* and, in its 'Properties' dialog, set the 'Type' field to *subProcess* and change the attribute 'triggeredByEvent' to *true*

- Within *eventSubProcess1* create a Start Event *Start By Signal1 Non Interrupting* and, in its 'Properties' dialog, set the 'Type' field to *Event Sub-Process Non-Interrupting > Signal*

- Add a Sequence Flow to a target Abstract Task *Task11 (20)*

- Add a Sequence Flow to a target End Event *End By Throwing Signal2* and, in the element 'Properties' dialog, set the 'Type' field to *Signal*

Create another process

- From the Toolbox, drag and drop the 'Pool' icon onto the diagram and name the element *Pool1*

- Right-click on *Pool1* in the Project Browser and select the 'Encapsulate Process' option; a process *BusinessProcess_Pool1* is created and the tag 'Pool1.processRef' is set to *BusinessProcess_Pool1*

Create the main process for *Pool1*

- Create a Signal Start Event *Start By Signal1*

- Add a Sequence Flow to a target Abstract Task *Task12 (100)*

- Add a Sequence Flow to a target End Event *E3*

Create an Event sub-process to interrupt *Pool1*

- Create an Activity *eventSubProcess1* and, in the 'Properties' dialog, set the 'Type' field to *subProcess*; change the attribute 'triggeredByEvent' to *true*

- Within *eventSubProcess2* create a Start Event *Start By Signal2 Interrupting* and, in the 'Properties' dialog, set the 'Type' field to *Event Sub-Process Interrupting > Signal*

- Add a Sequence Flow to a target Abstract Task *Task13 (10)*

- Add a Sequence Flow to a target End Event *E4*

Create the BPMN2.0 Signal Elements and configure for Signal Events

In the BPMN 2.0 Toolbox, expand the 'BPMN 2.0 - Types' page and drag the 'Signal' icon onto the diagram; name the element *Signal1*. Drag the icon onto the diagram again to create *Signal2*. These are root elements (which can be used by all processes) so they will be created directly under the model Package.

Double-click on each of the Signal Event elements and, in the 'Value' field for the 'signalRef' tag, click on the [...] button and browse to the appropriate Signal element.

Tips: Alternatively, you can drag the Signal element from the Project Browser and drop it on the Event elements in the diagram; a context menu displays, from which you select the 'set signalRef' option.

- Set signalRef to 'Signal1' on:
  - *Broadcast Signal1*
  - *Start By Signal1* in top level process (*Pool1*)
  - *Start By Signal1 Non Interrupting* in Event sub-process *eventSubprocess1*
  - *Receive Signal1 (normal)*
  - *Receive Signal1 (boundary non-interrupting)*

- Set signalRef to 'Signal2' on:
  - *Start By Signal2 Interrupting* in Event sub-process *eventSubProcess2*
  - *Receive Signal2 (normal)*
  - *Receive Signal2 (boundary Interrupting)*
  - *Receive Signal2 (in Event Gateway)*

Configure BPSim

In this section, we create the Configuration Artifact, specify the model Package and configure the parameter values of each element.

The configuration is quite simple because none of the Signal Events require any BPSim configurations. All we have to do is set the processing time for tasks so we can observe how processes, threads and tasks are started and interrupted.

Task	Description
Set up configuration	- Open the Configure BPSim window ('Simulate > BPSim > Manage > Open BPSim Manager') - Create an Artifact named 'SignalEvent Complete Example' (in the 'Select/Create Artifact' field, click on the [...] button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the Configure BPSim window.
Non-Signal Events	- In the element list on the left of the dialog, expand the 'StartEvent' group, then click on *S1* and on the 'Control' tab; click on the 'New Parameter' drop-down arrow and select 'Trigger Count', then type '1' in the 'Value' field - Expand the 'IntermediateEvent' group, then click on *Wait (25)* and on the 'Control' tab; click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer', then click on the [...] button in the 'Value' field; select 'Constant' and 'Numeric', and type '25' in the 'Constant Numeric' field and 'seconds' in the 'TimeUnit' field
Dummy variable for Process	The simulation controller displays a list showing the runtime token count for each element. For example, 4 tokens have passed the Gateway element *parallelMerge* in a simulation. This is quite useful for certain statistics and analysis. However, it does not show WHEN *parallelMerge* was traversed during the simulation. In order to get the exact trace for a single token we use the property trace utility, which relies on property parameters. So we create a dummy parameter.

	In the 'Configuration BPSim' dialog, expand the 'BusinessProcess' group. • Click on *BusinessProcess_Main* and on the 'Properties' tab, and overtype *New Property* with *dummyVariable;* in the 'Value' field, click on the ... button and on 'Constant' and 'Numeric', and in the 'Constant Numeric' field type '0' • Click on *BusinessProcess_Pool1* and perform exactly the same actions as for *BusinessProcess_Main*
Processing Time for Tasks	Expand the 'Activity' group and for each Task element listed here: select the 'Time' tab, click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then click on the ... button on the 'Value' column, select 'Constant' and 'Numeric', type the value as indicated into the 'Constant Numeric' field and select 'seconds' in the 'TimeUnit' field. • Task1 (20): 20 seconds • Task2 (10): 10 seconds • Task3 (20): 20 seconds • Task4 (10): 10 seconds • Task5 (100): 100 seconds • Task6 (10): 10 seconds • Task7 (30): 30 seconds • Task8 (5): 5 seconds • Task9 (10): 10 seconds • Task10 (10): 10 seconds • Task11 (20): 20 seconds • Task12 (100): 100 seconds • Task13 (10): 10 seconds

Run Simulation

• On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog

• Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

• After simulation, click on the ⬚ button on the tool bar to display the 'BPSim PropertyParameter Values' dialog

• Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable'

Analysis

From the direct results of the simulation it might not be obvious what has taken place; however, if we draw the lifeline for each task, it becomes quite clear.
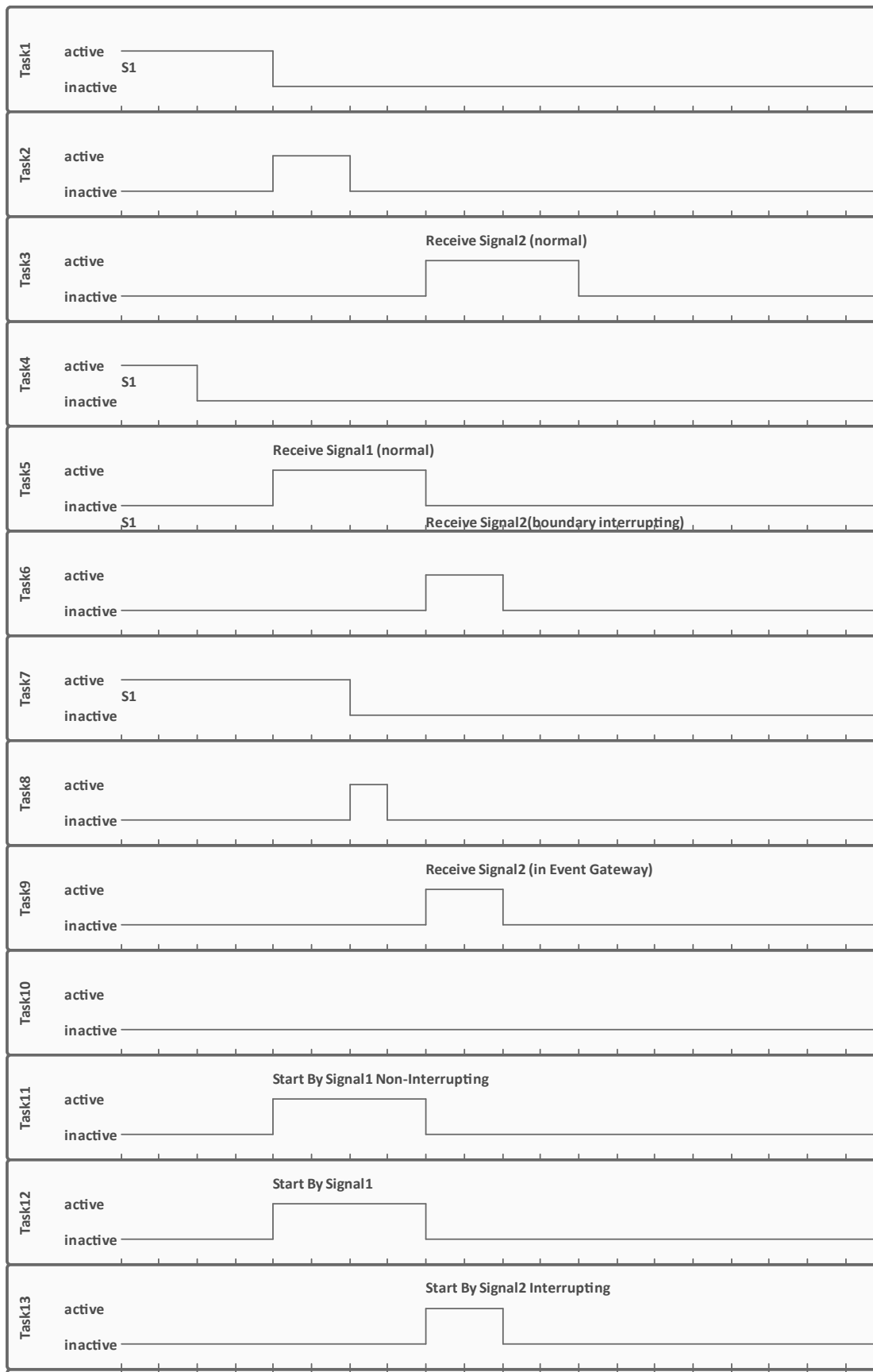
- *Task1, Task4* and *Task7* started in parallel

- *Task2* started immediately after *Task1* finished (without stopping at the throwing event)

- At 20 seconds, *Signal1* was broadcast by the Throwing Intermediate Event *Broadcast Signal1* and:
  - *Receive Signal1 (normal)* was activated and *Task5* started
  - *Start By Signal1 Non-Interrupting* was activated and *Task11* in *eventSubProcess1* started
  - *Start By Signal1* was activated and *Task12* in *Pool1 s*tarted

- At 40 seconds, *Signal2* was broadcast by the End Event *End By Throwing Signal2* and:
  - *Receive Signal2 (normal)* was activated and *Task3* started
  - *Task5* was interrupted and *Task6* started
  - *Receive Signal2 (in Event Gateway)* was activated and *Task9* started
  - *Start By Signal2 Interrupting* was activated, and:
    - > The main process in *Pool1* was interrupted and *Task12* stopped
    - > *Task13* in *eventSubProcess2* started

- The *eventSubProcess2* inside *BusinessProcess_Pool1* finished when *E4* was reached at 50 seconds

- The *BusinessProcess_MainPool* finished when *E1* was reached at 60 seconds

- The Intermediate Timer Event *Wait (25)* did not get activated because the signal event in the Gateway was activated first; as a result, *Task10* was never started

Note: The actual running time for each task can be observed from the generated BPSimReport element, by:

1. Double-clicking on the <<BPSimReport>> element.

2. Expanding the 'Time' group.

3. Expanding the task element.

4. Checking 'Total Time In Task'.

For example, for element *Task5 (100)*, although we set its processingTime to be 100 seconds, the **Total Time In Task** was 20 seconds, which was interrupted by *Receive Signal2 (boundary interrupting)* at 20 seconds.

Timer Event - Boundary

Create BPMN Model
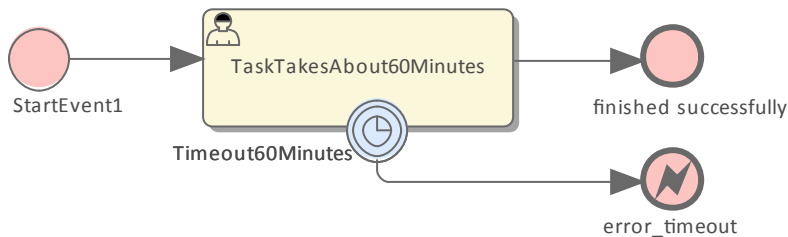
- Create a Start Event *StartEvent1*

- Add a Sequence Flow to a target userTask *TaskTakesAbout60Minutes*

- Add a Sequence Flow to a target End Event *finished successfully*

- Create an Intermediate Event, dragging the icon from the Toolbox and dropping it onto *TaskTakesAbout60Minutes;* select 'Edge-Mounted' and 'Timer' from the automatic menus, and call the element *Timeout60Minutes*

- Add a Sequence Flow to a target End Event (Error) *error_timeout*

Configure BPSim

In this section, we create the Configuration Artifact, identify the parent Package and set the parameter values of each element.

Objects	Action
Create Artifact and Package	- Open the 'Configure BPSim' dialog ('Simulate > BPSim > Manage > Open BPSim Manager') - Create an Artifact named 'IntermediateEvent - Boundary - Timer - TruncatedNormal' (in the 'Select/Create Artifact' field, click on the [...] button and select its parent Package, click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the 'Configure BPSim' dialog.
StartEvent1	In the element list on the left of the dialog, expand the 'StartEvent' group and click on *StartEvent1*. - Click on the 'Control' tab - Click on the 'New Parameter' drop-down arrow and select 'TriggerCount' - In the 'Value' field type '100'
TaskTakesAbout60Minutes	In the element list on the left of the dialog, expand the 'Activity' group and click on *TaskTakesAbout60Minutes*. - Click on the 'Time' tab - Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime' - In the 'Value' field click on the [...] button and select 'Distribution' and 'TruncatedNormal'

	• In the 'Mean' field type '50' • In the 'StandardDeviation' field type '10' • In the 'Min' field type '0' • In the 'Max' field type '1000' • Click on the OK button
Timeout60Minutes	In the element list on the left of the dialog, expand the 'IntermediateEvent' group and click on *Timeout60Minutes*. • Click on the 'Control' tab • Click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer' • Set the value to '000:000:000 001:00:00' (that is, 1 hour)

Run Simulation

• On the Configure BPSim window toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog

• Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

• After simulation, click on the [icon] button on the tool bar to display the 'BPSim PropertyParameter Values' dialog

• Click on the Query button and on the 'Group by Property' tab, and expand 'dummyVariable'

On simulation, we get this result:

Analysis

Since the ProcessingTime of *TaskTakesAbout60Minutes* was set as a distribution value, it turns out that:

- 93 out of 100 finished in 1 hour, so the normal flow to *finished successfully* takes effect
- 7 out of 100 finished in more than 1 hour, so the exception flow to *error_timeout* takes effect

Other Configurations

In the example folder, there are two other Business Process Simulation Artifacts that set the ProcessingTime as a constant value of 50 minutes and 80 minutes, other settings remain the same.

Run simulation on these two Artifacts:

- ProcessingTime configured to be 50 minutes always finishes in normal flow
- ProcessingTime configured to be 80 minutes always finishes in exception flow

Timer Event - Standalone Intermediate Event

When a Timer Intermediate Event is used within the normal sequence flow as a standalone element, it acts as a delay mechanism.

Create BPMN Model
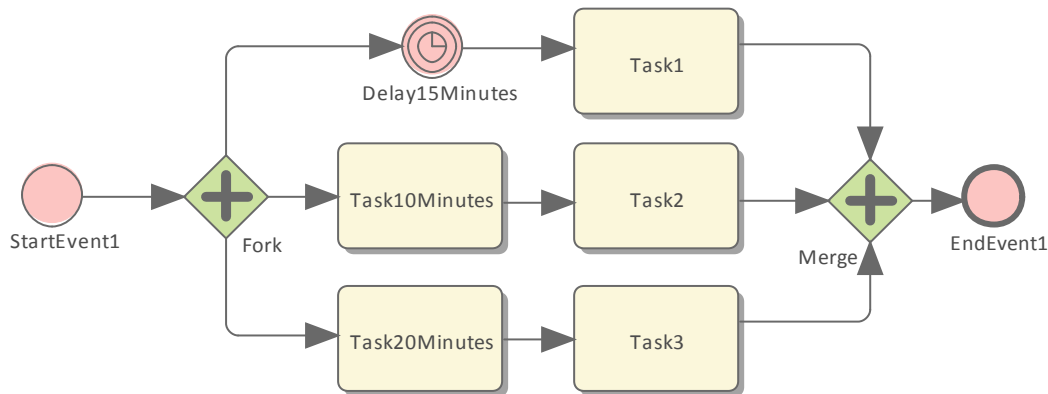
- Create a Start Event called *StartEvent1*
- Add a Sequence Flow to a target Parallel Gateway called *Fork*
- Add Sequence Flows to:
  - A Standalone Timer Intermediate Event called *Delay15Minutes*, and from that a Sequence Flow to an Activity called *Task1*
  - An Activity called *Task10Minutes*, and from that a Sequence Flow to an Activity called *Task2*
  - An Activity called *Task20Minutes*, and from that a Sequence Flow to an Activity called *Task3*
- From *Task1*, *Task2* and *Task3* create Sequence Flows to a Merge Parallel Gateway called *Merge*
- Add a Sequence Flow to a target EndEvent called *EndEvent1*

Configure BPSim

In this section, we create the Configuration Artifact, specify the model Package and configure the parameter values of each element.

Object	Action
Create Artifact and Package	• Open the 'Configure BPSim' dialog ('Simulate > BPSim > Manage > Open BPSim Manager') • Create an Artifact named 'IntermediateEvent - Standalone - Timer' (in the 'Select/Create Artifact' field, click on the [...] button and select its parent Package and click on the Add New button, then type in the element name and click on the Save button and the OK button) Then all the BPMN elements will be loaded in to the 'Configure BPSim' dialog.
StartEvent1	• In the element list on the left of the dialog, expand the 'StartEvent' group, then click on *StartEvent1* and on the 'Control' tab • Click on the 'New Parameter' drop-down arrow and select 'Trigger Count', then

	type '1' in the 'Value' field
	• Click on the 'Properties' tab
	• Overtype *New Property* with *dummyProperty;* in the 'Value' field, click on the [...] button and on 'Constant' and 'Numeric', and in the 'Constant Numeric' field type '0'
	With this property, the 'Property Trace' dialog will be able to show the sequence of element flows during simulation.
Delay15Minutes	• In the element list on the left of the dialog, expand the 'IntermediateEvent' group, then click on *Delay15Minutes* and on the 'Control' tab
	• Click on the 'New Parameter' drop-down arrow and select 'InterTriggerTimer', then set the 'Value' field to 15 minutes ('000:000:000 000:15:00')
Task10Minutes	• In the element list on the left of the dialog, expand the 'Activity' group, then click on *Task10Minutes* and on the 'Time' tab
	• Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then set the 'Value' field to 10 minutes ('000:000:000 000:10:00')
Task20Minutes	• In the element list on the left of the dialog, expand the 'Activity' group, then click on *Task20Minutes* and on the 'Time' tab
	• Click on the 'New Parameter' drop-down arrow and select 'ProcessingTime', then set the 'Value' field to 20 minutes ('000:000:000 000:20:00')

Run Simulation

• On the 'Configure BPSim' dialog Toolbar, click on the 'Run' icon to open the 'BPSim Simulation Controller' dialog

• Click on the 'Run' icon drop-down arrow and select 'Standard Simulation'

• After simulation, click on the [icon] button on the tool bar to display the 'BPSim PropertyParameter Values' dialog

• Click on the Query button and on the 'Group by Property' tab

Analysis

The *Fork* Parallel Gateway will activate the outgoing Sequence Flows simultaneously (the order is undefined and not important). However, we would expect the order of the Tasks to be exactly:

- *Task2*
- *Task1*
- *Task3*

This order is determined by the BPSim parameters set on two of the Activities (ProcessingTime) and the Timer Intermediate Event (InterTriggerTimer). The sequence shown in the 'BPSim PropertyParameter Values' dialog confirms that *Task2* comes ahead of *Task1*, which comes ahead of *Task3*.

Paint Wall Process Simulation (Call Activity)

This is a simple example to simulate the process of painting a wall. We define the main process as preparing the surface and then painting it three times. Preparing the surface is further divided into tasks such as sanding and cleaning.

We assume that applying each of the three coats of paint is the same process, except that the time randomly spent on each coat might be different.

Create BPMN Model
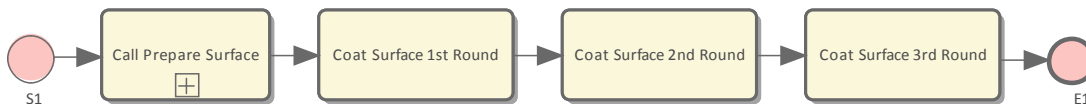
Create the main process - Paint Wall Process

- Create a Start Event called *S1*
- Add a Sequence Flow to a target callProcessActivity called *Call Prepare Surface*
- Add a Sequence Flow to a target callGlobalTaskActivity called*Coat Surface 1st Round*
- Add a Sequence Flow to a target callGlobalTaskActivity called *Coat Surface 2nd Round*
- Add a Sequence Flow to a target callGlobalTaskActivity called *Coat Surface 3rd Round*
- Add a Sequence Flow to a target End Event called *E1*
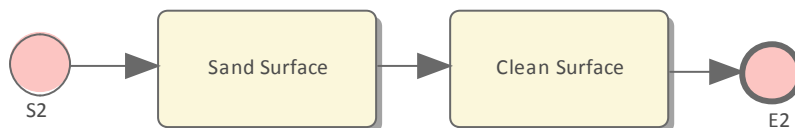
Create the re-used process - Prepare Surface Process

- Create a Start Event called *S2*
- Add a Sequence Flow to a target Abstract Task called *Sand Surface*
- Add a Sequence Flow to a target Abstract Task called *Clean Surface*
- Add a Sequence Flow to a target End Event called *E2*

Set Global Task and re-used process to call Activities

- Create a Global Task Activity called *Coat Surface*
- Double-click on each of *Coat Surface 1st Round, Coat Surface 2nd Round* and *Coat Surface 3rd Round*, and set the tag 'calledActivityRef' to *Coat Surface*

Tip: You can also drag the Global Task 'Coat Surface' from the Project Browser and drop it on the Call Activity element, clicking the 'set calledActivityRef' option on the context menu.

- Double-click on *Call Prepare Surface* and set the tag 'calledActivityRef' to *Prepare Surface Process*

Tip: You can also drag the process 'Prepare Surface Process' from the Project Browser and drop it on the Call Activity element, clicking the 'set calledActivityRef' option on the context menu.

Configure BPSim

- Open the 'Configure BPSim' dialog ('Simulate > BPSim > Manage > Open BPSim Manager')
- Create a Business Process Simulation Artifact named *Paint Wall Simulation*
- Select the Package containing the corresponding BPMN 2.0 model

All of the BPMN elements will be loaded in to the 'Configure BPSim' dialog.

Object	Activity
Fixed Scaling Time	On the left hand panel of the Configure BPSim window, expand the 'Activity' group, then: • Click on *Sand Surface* and on the 'Time' tab, then click on the 'New Parameter' drop-down arrow and select 'Processing Time'; in the 'Value' field, change the setting to 000:000:000 000:30:00 (that is, 30 minutes) • Click on *Clean Surface* and on the 'Time' tab, then click on the 'New Parameter' drop-down arrow and select 'Processing Time'; in the 'Value' field, change the setting to 000:000:000 000:10:00, (that is,10 minutes)
Random Coating Time	On the left hand panel of the Configure BPSim window, expand the 'GlobalTask' group, then: • Click on *Coat Surface* • Click on the 'Time' tab, then click on the 'New Parameter' drop-down arrow and select 'Processing Time' • In the 'Value' field click on the [...] button, and on the parameter dialog click on the 'Distribution' tab and on 'Poisson' • In the 'Mean' field type '10', then click on the OK button With this setting, the mean value of the random numbers generated by the Poisson distribution is 10. If you prefer, you can choose other types of distribution.
TriggerCount on S1	On the left hand panel of the Configure BPSim window, expand the 'Start Event' group, then: • Click on *S1* • Click on the 'Control' tab, then click on the 'New Parameter' drop-down arrow and select 'TriggerCount' • In the 'Value' field type '1'

Run Simulation

- From the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button and select 'Standard Simulation'
- When the simulation is complete, it provides this result:

Flow Analysis

For the only token started on *S1*, we can see from the 'BPSim Simulation Controller' dialog how the flow develops:

- When reaching the callProcessActivity, the called process is activated; so we have *S2 ~ E2*

- When reaching a callGlobalTaskActivity, the called Global Task is activated - the notation reads: *GlobalTask name (called activity name)*; the global *Coat Surface* was called three times:
    - *Coat Surface(Coat Surface 1st Round)*
    - *Coat Surface(Coat Surface 2nd Round)*
    - *Coat Surface(Coat Surface 3rd Round)*

Time Analysis

Click on the button on the 'BPMN Simulation Controller' dialog toolbar. The 'BPMN Simulation Report View' displays:

Tips: Type a text string in the Filter bar to show only the records containing that text string in that column field.

After expanding the report (click on the 📁 button on the 'Report View' toolbar), the report displays as shown in the example, from which we can make this analysis:

- *Coat Surface 1st Round* took 8 minutes; *Coat Surface 2nd Round* took 10 minutes; *Coat Surface 3rd Round* took 12 minutes - the figures 8, 10, 12 are randomly generated by the Poisson(10) distribution; what is important here is that each call instance of the Global Task has its own values

- *Coat Surface* has a total time collected from all instances: $8 + 10 + 12 = 30$

- The Call Activity *Call Prepare Surface* took 40 minutes, composed of *Sand Surface* (30 minutes) and *Clean Surface* (10 minutes)

- The Sum Of Processing Time for *Paint Wall Process* is 70 minutes, composed of the four Call Activities: $40 + 8 + 10 + 12 = 70$

Customized Simulation

We can configure a 'Result Request' on BPMN elements to customize the simulation report so that we only report on the parameters we are interested in.

Configure Result Request

- On the left hand side of the 'Configure BPSim' dialog, expand the 'Activity' group, then click on *Coat Surface 1st Round* and on the 'Time' tab

- In the lower panel on the right of the dialog, click on 'ProcessingTime', click on the drop-down arrow in the 'Result Request' field and select the checkbox against 'sum'; click on the OK button
- Repeat these steps for the Activities *Call Prepare Surface*, *Coat Surface 2nd Round*, *Coat Surface 3rd Round*
- Expand the 'Business Process' group and repeat these steps for *Paint Wall Process*

Run Simulation

- On the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays
- Click on the Run button drop-down arrow and select, in this case, 'Customized Simulation'

Flow Analysis

The Flow Analysis is exactly the same as for a Standard Simulation.

Time Analysis

On the 'BPSim Simulation Controller' dialog toolbar, click on the ▣ button; the 'BPMN Simulation Report View' displays.

The Time Analysis is the same as for a Standard Simulation; however, the report only contains the 'sum' results we requested.

Note: Currently, in the Time Analysis, we can not request ProcessingTime either on the called process itself or on the Activities contained by the called process. If you have this requirement, use the Standard Simulation.

BPSim Cost Parameters

BPSim 1.0 provides the means to set cost parameters and receive cost statistics from process simulation experiments. BPSim provides a framework to determine **variable** costs according to two parameters, both related to the level of activity performed in the simulated process. These parameters are:

- Completion cost ('Fixed Cost' in the BPSim specification) - The cost incurred whenever an operation is completed; this cost can be related to Task, Process, Sub-Process, Call Activity or Resource elements

- Time cost ('Unit Cost' in the BPSim specification) - The cost incurred whenever a task, process, sub-process, call activity or resource is busy for a period of time

The cost parameters are supported on Activities, resources and processes.

- Activity - Completion cost is incurred whenever an Activity finishes; Time cost (Unit Cost * Time) is incurred whenever an activity finishes

- Resource - Completion cost is incurred whenever each involved resource finishes an activity; Time cost (Unit Cost * Time) is incurred whenever each involved resource finishes an activity

- Process - Completion cost is incurred whenever a process finishes; Time cost (Unit Cost * Time) is incurred whenever a process finishes

Those costs that are known without the need for simulation - for example, overall labor employment costs - are not supported by BPSim.

In this topic, we discuss two examples that demonstrate the configuration and simulation of cost parameters.

- Set Cost on Activity
- Set Cost on Resource

Set Cost Parameters on Activity
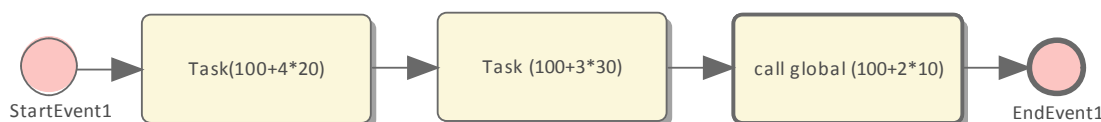
Create the BPMN Model (Activities)

- In the Project Browser, create a *StartEvent1*, a *GlobalTask1*, two AbstractTasks, and an *EndEvent1*

- Ctrl+drag the elements from the Project Browser onto a diagram, pasting *GlobalTask1* as an Invocation (Call Activity) called *call global (100+2*10)*

- Give the elements names and connect them with Sequence Flows; the two AbstractTasks should be called:
  - *Task (100+3*30)* and
  - *Task (100+4*20)*

BPSim Configuration

Create a Business Process Simulation configuration Artifact in the diagram, right-click on it and select the 'Configure BPSim' option. Set the configuration to link to the Package containing the BPMN model elements and configure these BPSim parameters as indicated.

Parameter	Settings
	- Click on the 'Scenario' tab and, for the 'Time Unit' parameter, click on the

| Scenario Parameters	'Value' drop-down arrow and select 'hours'
	• In the 'Value' field for the 'Duration' parameter, set the value to '0001 00:00:00' (1 day)
	This Time Unit is used to calculate the Time Cost (Time cost = Unit Cost * Time), so make sure the Unit Cost is based on the correct Time Unit.
Control Parameters	• On the left side of the dialog, expand the 'StartEvent' group and click on *StartEvent1*
	• Click on the 'New Parameter' drop-down arrow and select 'TriggerCount'
	• In the 'Value' field type '1'
Time Parameters	On the left side of the dialog, expand the 'Activities' group.
	• Click on *Task(100+4*20)* and on the 'Time' tab, then click on the 'New Parameter' drop-down arrow and select 'ProcessingTime'; set the 'Value' field to '000:000:000 004:00:00' (4 hours)
	• Click on *Task(100+3*30)* and do the same, setting the 'Value' field to '000:000:000 003:00:00' (3 hours)
	Expand the 'GlobalTask' group.
	• Click on *GlobalTask1* and do the same, setting the 'Value' field to '000:000:000 002:00:00' (2 hours)
Cost Parameters	• Click on *Task(100+4*20)* and on the 'Cost' tab, then click on the 'New Parameter' drop-down arrow and select:
	- 'FixedCost' - in the 'Value' field click on the [...] button, select the 'Constant' tab and 'Floating', and in the 'Constant Floating' field type '100'; click on the OK button - 'UnitCost' - do the same, setting the 'Constant Floating' field to '20'

	• Click on *Task(100+3*30)* and do the same, setting:
	  - 'FixedCost' to '100
	  - 'UnitCost' to '30'

	• Click on *GlobalTask1* and do the same, setting:
	  - 'FixedCost' to '100
	  - 'UnitCost' to '10'

	• Expand the 'BusinessProcess' group, click on *BPSim Cost* and do the same, setting:
	  - 'FixedCost' to '50'
	  - 'UnitCost' to '5

Simulation

• From the 'Configure BPSim' dialog toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays

• Click on the Run button and select 'Standard Simulation'

• When the simulation is complete, click on the 🗐 button on the 'BPMN Simulation Controller' dialog toolbar; the 'BPMN Simulation Report View' displays

• Filter the report by clicking on the ⁞ᵛ ▾ button and selecting 'Show only Non-Empty Items'

BPMN Simulation Report View

Item	CostOnActivitySimulation- Result
▷ Time	
▷ Control	
▷ Resource	
◢ Cost	
◢ <<Activity>>Task (100+3*30)	
Total Completion Cost	100.00
Total Time Cost	90.00
◢ <<Activity>>call global (100+2*10)	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<Activity>>Task(100+4*20)	
Total Completion Cost	100.00
Total Time Cost	80.00
◢ <<GlobalTask>>GlobalTask1	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<GlobalTask>>GlobalTask1(<<Activity>>call global (100+2*10))	
Total Completion Cost	100.00
Total Time Cost	20.00
◢ <<BusinessProcess>>BPSim Cost	
Total Completion Cost	50.00
Total Time Cost	45.00

Analysis

Activity	Analysis
Task(100+4*20)	• Total Completion Cost is 100, matching the FixedCost (100) setting in BPSim • Total Time Cost is 80, calculated as ProcessingTime (4 hours) * UnitCost (20/hour)
Task(100+3*30)	• Total Completion Cost is 100, matching the FixedCost (100) setting in BPSim • Total Time Cost is 90, calculated as ProcessingTime (3 hours) * UnitCost (30/hour)
call global (100+2*10)	• Total Completion Cost is 100, matching the FixedCost (100) on *GlobalTask1* setting in BPSim • Total Time Cost is 20, calculated as ProcessingTime (2 hours) * UnitCost (10/hour) on *GlobalTask1*
BPSim Cost process	• Total Completion Cost is 50, matching the FixedCost (50) setting in BPSim • Total Time Cost is 45, calculated as Total ProcessingTime of all tasks (4 + 3 + 2 = 9 hours) * UnitCost (5/hour)

Set Cost Parameters on Resource
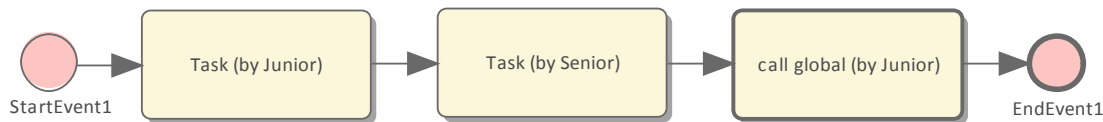
Create the BPMN Model (Resources)

- In the Project Browser create a *StartEvent1*, a *GlobalTask1*, two abstractTasks called *Task (by Junior)* and *Task (by Senior)*, and an *EndEvent1*

- Ctrl+drag the elements from the Project Browser onto a diagram, pasting *GlobalTask1* as an Invocation (Call Activity) named *call global (by Junior)*

- Connect the elements with Sequence Flows

- Create two BPMN2.0 Resource elements: *Junior Developer* and *Senior Developer*

BPSim Configuration

Create a Business Process Simulation configuration Artifact in the diagram, right-click on it and select the 'Configure BPSim' option, then set the configuration to link to the Package containing the BPMN model elements and configure these BPSim parameters as indicated.

Parameter	Setting
Scenario Parameters	- Click on the 'Scenario' tab and, for the 'Time Unit' parameter, click on the 'Value' drop-down arrow and select 'hours' - In the 'Value' field for the 'Duration' parameter, set the value to '0001 00:00:00' (1 day) This Time Unit is used to calculate the Time Cost (Time cost = Unit Cost * Time), so make sure that the Unit Cost is based on the correct Time Unit.
Control Parameters	- On the left side of the dialog, expand the 'StartEvent' group and click on *StartEvent1* - Click on the 'New Parameter' drop-down arrow and select 'TriggerCount' - In the 'Value' field type '1'
Time Parameters	On the left side of the dialog, expand the 'Activities' group. - Click on *Task (by Junior)* and on the 'Time' tab, then click on the 'New Parameter' drop-down arrow and select 'ProcessingTime'; set the 'Value' field to '000:000:000 004:00:00' (4 hours) - Click on *Task (by Senior)* and do the same, setting the 'Value' field to '000:000:000 003:00:00' (3 hours) Expand the 'GlobalTask' group. - Click on *GlobalTask1* and do the same, setting the 'Value' field to '000:000:000 002:00:00' (2 hours)
Resource Parameters	On the left of the dialog, expand the 'Resource' group.

	• Click on *Junior Developer* and on the 'Resource' tab, then click on the 'New Parameter' drop-down arrow and select 'Quantity'; in the 'Value' field type '10'
	
	• Click on *Senior Developer* and do the same, typing '5' in the 'Value' field
	
	Assign Resource to Activities
	On the left of the dialog, expand the 'Activity' group and click on:
	
	• *Task (by Junior)* and on the 'Resource Role' tab, then - if the 'Resource Roles defined in BPMN' panel is empty - click on the Add Resource Role button, choose 'Junior Developer' and type '1' in the 'Quantity' field; check that the 'Resource Selection Expression for Activity' panel now contains:
	    bpsim::getResource('Junior Developer', 1)
	
	• *call global (by Junior)* and do exactly the same thing
	
	• *Task (by Senior)* and do the same thing, selecting 'Senior Developer' and setting 'Quantity' to '1'; the 'Resource Selection Expression for Activity' panel should contain:
	    bpsim::getResource('Senior Developer', 1)

Cost Parameters	• Click on *Junior Developer* and on the 'Cost' tab, then click on the 'New Parameter' drop-down arrow and select:
	
	  - 'FixedCost' - in the 'Value' field click on the ⟦...⟧ button, select the 'Constant' tab and 'Floating', then in the 'Constant Floating' field type '100' and in the 'CurrencyUnit' field type 'AUD'; click on the OK button
	  - 'UnitCost' - do the same, setting the 'Constant Floating' field to '20'
	
	• Click on *Senior Developer* and do the same, setting:
	  - 'FixedCost' to '100'
	  - 'UnitCost' to '30'

Simulation

• From the Configure BPSim window toolbar, click on the Run button; the 'BPSim Simulation Controller' dialog displays

• Click on the Run button and select 'Standard Simulation'

• When the simulation is complete, click on the ⟦🗐⟧ button on the 'BPMN Simulation Controller' dialog toolbar; the 'BPMN Simulation Report View' displays

• Filter the report by clicking on the ⟦☰⟧ button and selecting 'Show only Non-Empty Items'

Analysis

Resource	Results
Junior Developer	• Total Completion Cost is '200', calculated as FixedCost (100) * number of Activities involved (2) • Total Time Cost is '120', calculated as ProcessingTime (4 + 2 = 6 hours) * UnitCost (20/hour)
Senior Developer	• Total Completion Cost is '100', calculated as FixedCost (100) * number of Activities involved (1) • Total Time Cost is '90', calculated as ProcessingTime (3 hours) * UnitCost (30/hour)

Export a BPSim Configuration

When you have defined a BPSim Configuration in a model, you can export it to an XMI file to be imported into other projects. The BPMN 2.0 model on which the configuration is based is also exported with the configuration. This binds the model to the appropriate BPSim Configuration when you import the XMI file into another project.

Access

Ribbon	Publish > Model Exchange > Export XMI > Export XMI for Current Package : Publish
Context Menu	On Diagram or in Project Browser	Right-click Business Process Simulation Artifact	Export BPSim Configuration
Keyboard Shortcuts	Ctrl+Alt+E : Publish
Other	Toolbar of Configure BPSim window	[XML icon] Export icon

Publish Model Package

The process of exporting a BPSim configuration and its model uses the 'Publish Model Package' dialog for publishing a model to an XMI file.

Option	Description
Package	Defaults to the name of the Package containing the Business Process Simulation Artifact.
Filename	Type in or browse for (click on the [...] icon) the file path and XML filename into which to export the model.
XML Type	Select 'BPMN 2.0 XML'.
Export	Click on this button to export the configuration and BPMN 2.0 model. The export is complete when a confirmation message displays in the 'Progress' field.
Format XML Output	Defaults to selected; leave selected.
View XML	If you want to examine the exported XML, click on this button.

Notes

- To import the model from XMI into a new project, select the target Package in the new project and select the 'Publish > Model Exchange > Import XMI > Import Model XMI' ribbon option

DMN Modeling and Simulation

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG). It provides a standard approach for describing, modeling and implementing repeatable decisions within an organization or an initiative. It is also intended to facilitate the sharing and interchange of decision models between organizations.

The modeling notation comprises a visual grammar that allows decisions and business rules to be documented in a way that makes them readable by both business and technical audiences thus ensuring that decisions and rules are not misinterpreted. The resulting Decision Model also provides a definition of how to evaluate the logic of decisions defined in Decision Tables using the Friendly Enough Expression Language (FEEL).

The purpose of DMN is to provide the constructs that are needed to model decisions, so that organizational decision-making can be readily depicted in diagrams, accurately defined by business analysts.

In this topic, we will introduce DMN Expression, DMN Simulation Artifact and how you can use Enterprise Architect to automate the decision-making process.

DMN Expressions

- DMN Expression: Decision Table
- DMN Expression: Boxed Context
- DMN Expression: Literal Expression
- DMN Expression: Function
- DMN Expression: Invocation

DMN Data

- DMN ItemDefinition
- DMN InputData
- DataSet for InputData

DMN Simulation

- Configure a DMN Simulation Artifact
- Run, Step through or Debug the DMN Simulation

Code Generation & Connect to BPMN

- Generate the DMN Server in Java, JavaScript, C++, or C#
- Run/Debug testing of the Java version of the DMN Server
- Connect the DMN Server with the Enterprise Architect BPSim Execution Engine

Common Errors & Solutions

- 'Run validation' will help you locate most of the modeling issues; run this before simulation and code generation

- Variable Types: as DMN models use the FEEL language (Simulate with JavaScript), typing variables is not compulsory; however, when generating code to languages that are compiled, you do have to type a variable - there are context menu options and tag values for setting a type to the variables

- Since a DMN expression allows for spaces, in order to clarify the composite Input Data there must be a space before and after the '.' in the expression; for example, 'Applicant data . Age' is valid, whereas 'Applicant data.Age' is not valid
  Note that when using the Auto Completion feature this issue will not arise

Introduction

Now more than ever, in a world turned on its head by new and innovative business and technical ideas and disruptive ways of working, does an organization need to have a clear understanding of its choices and the decisions it makes. Unmanaged complexity is the enemy, and agility the friend that heralds business success and enables an organization to respond quickly to changes in its business circumstances. Without a clear and communicable model it is almost impossible for an organization to embrace the changes that confront them daily in the digital world.

The description and implementation of decisions, which has been inexplicably and somewhat invisibly part of almost all disciplines, has now been synthesized into a rigorous and formal discipline of its own with a new way of modeling and describing decisions, Inputs, Outcomes, Rules, Business Knowledge, Authorities and more. Once you have seen the Decision Model and Notation in action and been introduced to the countless benefits it brings, you will not be able to go back to old and arcane ways of working.
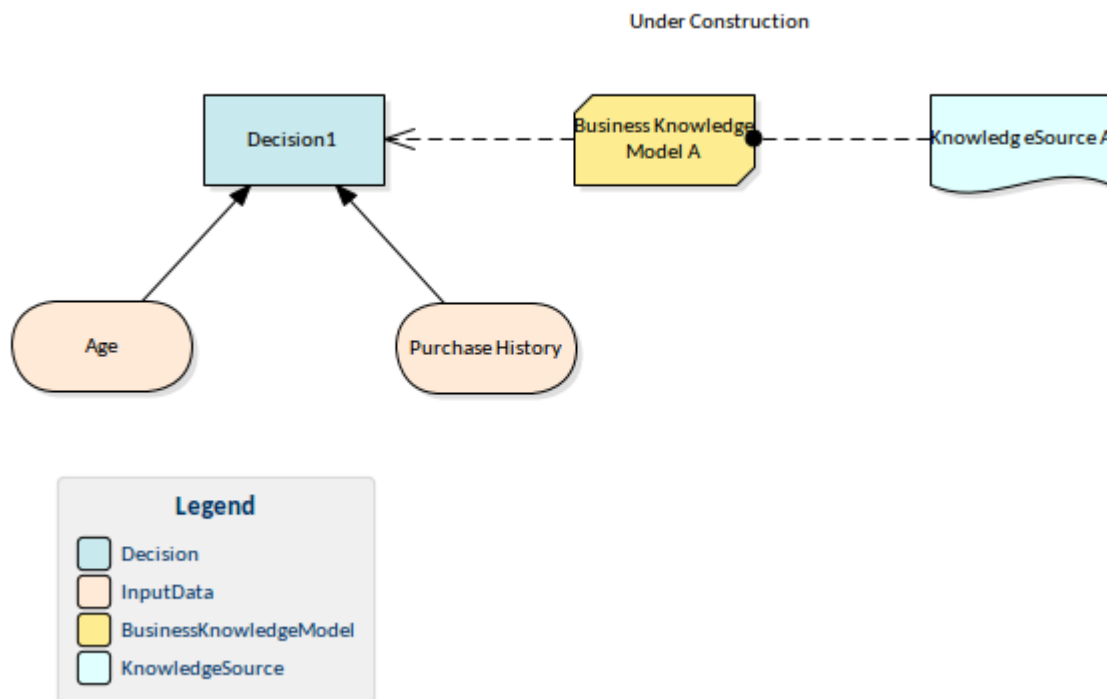
Enterprise Architect has become the tool of choice for many Business and Technical leaders because of its flexible, extensible, standards-based and pragmatic approach to modeling complex systems. As a collaboration platform it is a tool for all disciplines, allowing Decision Models to be created, integrated, managed, documented, simulated and generated to programming code. The models can be visualized and integrated with a range of other models, including Business Process diagrams, Use Case models, User Stories, Test Cases, Database models, Implementation Artifacts and programming code to list just the main models.

What is Decision Model and Notation

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG). It provides a standard approach for describing, modeling and implementing repeatable decisions within an organization or an initiative. It is also intended to facilitate the sharing and interchange of decision models between organizations.

The modeling notation consists of a visual grammar that allows decisions and business rules to be documented in a way that makes them readable by both business and technical audiences, thus ensuring that decisions and rules are not misinterpreted. The resulting Decision Model also provides a definition of how to evaluate the logic of decisions defined in Decision Tables, using the Friendly Enough Expression Language (FEEL).
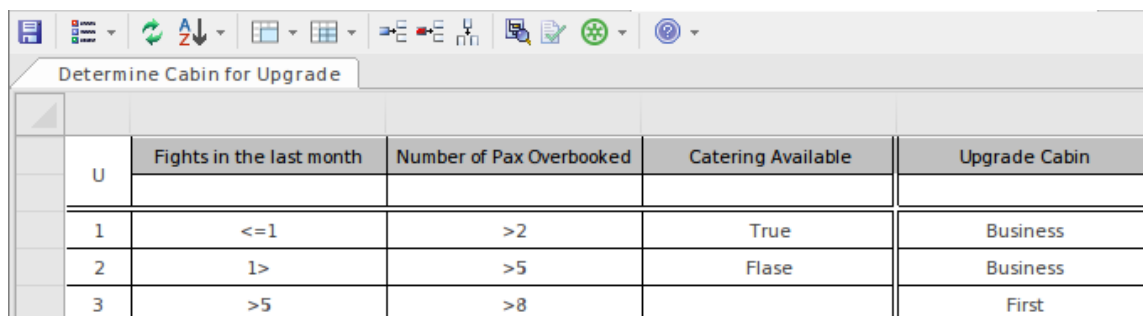
DMN defines two levels of the language, which neatly align with organizational roles:

Requirements Level

Aligns with the business people who devise, analyze and define the decision rules. This provides a mechanism for the people who own the business to model them without the need for technical knowledge of how they are implemented.

Logic Level

Aligns with implementation teams who augment the definitions to include decision logic in the form of Decision Tables that can be maintained by the business staff, or expression logic managed by technical staff.

U	Fights in the last month	Number of Pax Overbooked	Catering Available	Upgrade Cabin
1	<=1	>2	True	Business
2	1>	>5	Flase	Business
3	>5	>8		First

Determine Cabin for Upgrade

Why Use Decision Model and Notation

There are many reasons for using DMN at an Enterprise or Initiative level that will result in both business and technical value. There are many benefits described in the next topic but the most compelling of these are:

Reduced Complexity

As discussed earlier, currently business rules and decisions are commonly located in a myriad of places and in a wide variety of formats. A common place to find them is in Business Process Diagrams that are overly complex with cascading sets of Gateways that describe the outcomes. The diagrams are commonly unwieldy and do little to reveal the inputs, business knowledge, authorities or the logic of how the decisions are made.

Automation and Execution

As the Business Analysts define the decisions, inputs, business knowledge and the technologists elaborate the logic in the form of Expressions and Decision Tables, potentially creating Decision Services, the models can be used to generate programming language code that can be executed to automate the decisions and make them available to runtime systems.

Agility and Responsiveness to Change

As a result of separating the decision models from the Business Processes models and creating an implementation and automation pipeline, the business and technology teams can respond at record speed to business change. These models then become a platform that facilitates agility and the seizing of opportunities.

A First Example

Imagine you are an Airline reservation officer working at the check-in counter for a busy domestic airline. Getting the aircraft off on-time is critical as delays can result in fees applied by the airport controllers, needing to fly at a lower altitude increasing the cost of fuel, and other penalties.

A message from the supervisor appears on your screen saying that the economy cabin is overbooked; you will need to upgrade some passengers to Business or First Class — but which passengers should be chosen and which cabin should they be upgraded to? A decision needs to be made but what factors should be considered? This can be recorded in a Decision Model using a Decision Requirements diagram.
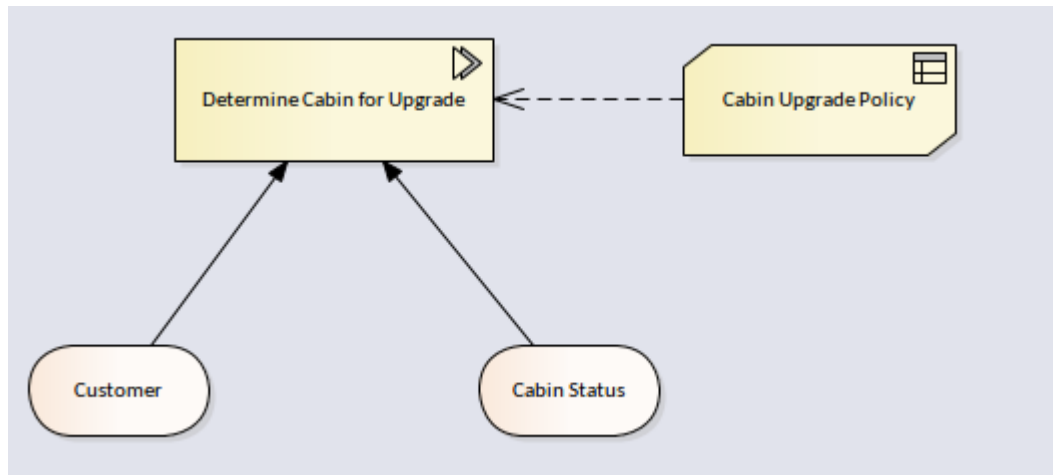
This is helpful but the busy check-in officer would still need to weigh up all the factors and make an unbiased decision. Should a disgruntled passenger be given priority over a Gold level frequent flyer, or should the fact that a particular passenger is connecting to an international flight take precedence. These 'rules' can all be recorded in a Decision table, making it clear which passengers should get an upgrade and to which cabin: Business or First Class. This will make it much easier to make the decision and the rules can be formulated, agreed upon and checked for consistency back at head office. In this example we have kept it simple and used two factors: firstly the number of flights the passenger has made in the last month and secondly how overbooked the cabin is.

Cabin Upgrade Policy		Input Parameter Values for Simulation	
	(Flights in the last month, Number of Pax Overbooked)		
U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin
			Business Class, First Class
1	<=1	<=2	Business Class
2	<=1	(2..8]	Business Class
3	<=1	>8	First Class
4	(1..5]	<=2	Business Class
5	(1..5]	(2..8]	Business Class
6	(1..5]	>8	First Class
7	>5	<=2	Business Class
8	>5	(2..8]	Business Class
9	>5	>8	First Class

The table is divided into columns and rows. There are two types of columns: inputs that are required to make the decision and outputs that are the result of applying the rules.

This is again very helpful but still requires the busy check-in officer to be able to source all the required information required to find the right row in the Decision table. Even if all this information were available, a wrong decision could still result from human error in selecting the wrong row in the table.

Fortunately the Decision Models can be automated and generated to programming code that can be executed by an application. So our busy check-in officer would not need to do anything or make any decisions; as she was checking in the passengers, if a particular passenger was entitled to an upgrade it would be visible on the computer screen. In the next diagram the model has been simulated so that the business and technical staff can agree that the model has been defined correctly. Any number of user defined data sets can be used to test the model before generating out the programming code that will run in the check-in system and display the result to the end user.
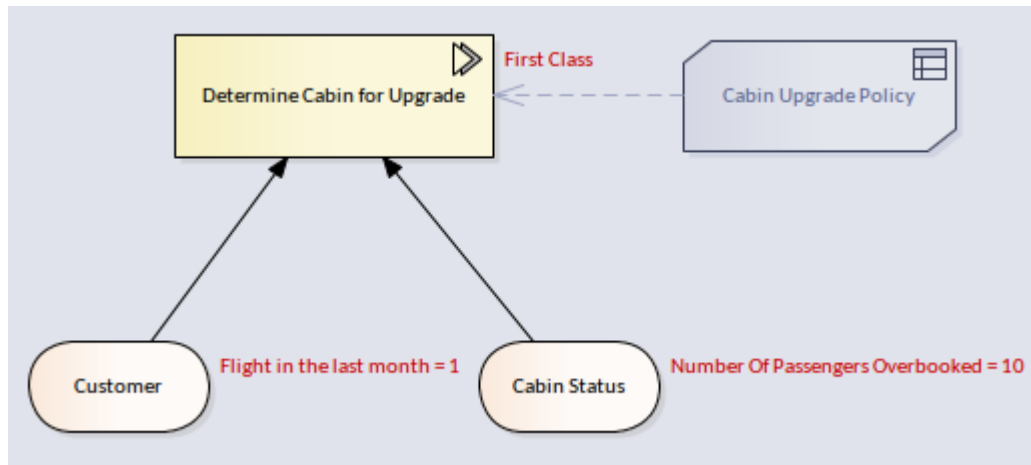
When developing the models a business or technical user can step through the simulation and the system will show that user which row in the Decision table was fired to determine the output. This is very useful in models that are made up of multiple decisions.

Cabin Upgrade Policy	Input Parameter Values for Simulation		
U	(Flights in the last month = 1, Number of Pax Overbooked = 10)		
	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin
	1	10	First Class
1	<=1	<=2	Business Class
2	<=1	(2..8]	Business Class
3	<=1	>8	First Class
4	(1..5]	<=2	Business Class
5	(1..5]	(2..8]	Business Class
6	(1..5]	>8	First Class
7	>5	<=2	Business Class
8	>5	(2..8]	Business Class
9	>5	>8	First Class

It is common for the rules that govern the upgrade decision to change. For example, the Marketing Department might decide they want to reward passengers that travel on long-haul flights. The Decision Requirements diagram can be altered to include the new input, the Decision table modified, and the programming code regenerated. Once the changes have been pushed through to the airport systems, the right passengers will be automatically upgraded. The check-in officer could still view the Decision tables during a training and briefing session to understand the rules.

DecisionTable

A Decision table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

Toolbar Options

Option	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to switch between Rule-as-row and Rule-as-column for the Decision table.
	Click on 'Sort By Input' to sort the rules by input columns; click on 'Sort By Output' to sort the rules by output columns. The columns can be dragged and dropped to organize the sorting order.
	Click on this button to merge adjacent input entry cells from different rules with the same content.
	Click on this button to split entry cells that have been merged.
	Click on this button to edit parameters for the Business Knowledge Model.
	Click on this button to append an input column for the Decision table.
	Click on this button to append an output column for the Decision table.
	Click on this button to append a rule for the Decision table.
	Click on this button to show or hide the allowed values fields for the 'Input' and 'Output' columns. The allowed value defined for an input or output will be used for validation and auto completion editing.
	Click on this button to validate the Decision table. Enterprise Architect will perform a series of validations to help the you locate errors in the Decision table.

▷	This button is enabled when a Decision table is defined for a BusinessKnowledgeModel.
	Select the 'Input Parameter Values for Simulation' tab, complete the fields and click on this button. The test result will be presented on the Decision table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted.
	You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel.
	Menu options are available for this tool bar button. For more information, see the *BusinessKnowledgeModel and Test Harness* Help topic.

Decision Table Hit Policy

The Hit Policy specifies the result of the Decision table in cases of overlapping rules. The single character in a particular Decision table cell indicates the table type and unambiguously reflects the decision logic.

Single Hit Policies:

- **U**nique: no overlap is possible and all rules are disjoint; only a single rule can be matched (this is the default)
- **A**ny: there might be overlap, but all the matching rules show equal output entries for each output, so any match can be used
- **P**riority: multiple rules can match, with different output entries; this policy returns the matching rule with the highest output priority
- **F**irst: multiple (overlapping) rules can match, with different output entries; the first hit by rule order is returned

Multiple Hit Policies:

- **O**utput order: returns all hits in decreasing output priority order
- **R**ule order: returns all hits in rule order
- **C**ollect: returns all hits in arbitrary order; an operator ('+', '<', '>', '#') can be added to apply a simple function to the outputs

Collect operators are:

- **+** (sum): the result of the Decision table is the sum of all the distinct outputs
- **<** (min): the result of the Decision table is the smallest value of all the outputs
- **>** (max): the result of the Decision table is the largest value of all the outputs
- **#** (count): the result of the Decision table is the number of distinct outputs

Example of Unique hit policy

The 'Unique' hit policy is the most popular type of Decision table and all rules are disjoint.

| Post-bureau risk category table | Input Parameter Values for Simulation |

(Existing Customer = true, Application Risk Score = 90, Credit Score = 590)			
U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
	true	90	590	MEDIUM
1	false	<120	<590	HIGH
2	false	<120	[590..610]	MEDIUM
3	false	<120	>610	LOW
4	false	[120..130]	<600	HIGH
5	false	[120..130]	[600..625]	MEDIUM
6	false	[120..130]	>625	LOW
7	false	>130	-	VERY LOW
8	true	<=100	<580	HIGH
9	true	<=100	[580..600]	MEDIUM
10	true	<=100	>600	LOW
11	true	>100	<590	HIGH
12	true	>100	[590..615]	MEDIUM
13	true	>100	>615	LOW

Example of Priority hit policy

In a table with the 'Priority' hit policy, multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority.

| Eligibility rules | Input Parameter Values for Simulation |

(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)			
P	Pre-Bureau Risk Categ...	Pre-Bureau Affordability	Age	Eligibility
				INELIGIBLE, ELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	<18	INELIGIBLE
4	-	-	-	ELIGIBLE

Note: The output order is defined as INELIGIBLE, ELIGIBLE; which means INELIGIBLE has higher priority than ELIGIBLE.

One possible simulation result might resemble this:

| Eligibility rules | Input Parameter Values for Simulation |

(Pre-Bureau Affordability = false, Pre-Bureau Risk Category = HIGH, Age = 25)			
P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	HIGH	false	25	INELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	<18	INELIGIBLE
4	-	-	-	ELIGIBLE

The matching rules are highlighted, but output with INELIGIBLE is the pick based on output order.

Example of Collection-Sum hit policy

For a Decision table with the 'Collect-Sum' (C+) hit policy, the result of the Decision table is the sum of all the distinct outputs.

	Application risk score model	Input Parameter Values for Simulation		
	(Age = 40, Marital Status = M, Employment Status = EMPLOYED)			
C+	Age	Marital Status	Employment Status	Partial score
	40	M	EMPLOYED	133
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	S	-	25
7	-	M	-	45
8	-	-	UNEMPLOYED	15
9	-	-	STUDENT	18
10	-	-	EMPLOYED	45
11	-	-	SELF-EMPLOYED	36

In this example, the output Partial Score is calculated as 43 + 45 + 45 = 133

Set type for Input/Output Clause

You have to set the type for Input Clause or Output Clause for these reasons:

- Display: The string will be shown in Italic font (otherwise, each string literal must be inside quotes)
- Validation: The range, gap and overlap validations are supported only on numbers
- Code Generation for typed languages such as C++, C# and Java

You might ask: since the BKM parameter already has types defined, why do we need to define types for the Input Clauses?

The answer is: the Input Clause could be an expression. For example, a parameter 'Credit Score' is a number type, but the Input Clause could be 'Credit Score > 500?', which is a boolean type.

Right-click on the Input Clause or Output Clause and select the appropriate type for it.

Credit contingency factor table	Input Parameter Values for Simulation		
	(Risk Category)		
U	Risk Category	✓ type: string	dit Contingency Factor
	DECLINE, HIGH, MEDIUM, LOW	type: boolean	
1	HIGH, DECLINE	type: number	0.6
2	MEDIUM	type: date	0.7
3	LOW, VERY LOW	type: time	0.8
		type: duration	
		Delete Input Column	

Literal Expression

A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block. When the expression is getting complicated, you might prefer a Boxed Context, or in order to improve the readability you can encapsulate some logic as a function in the DMN Library. See the example at the end of this page.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

Toolbar Options

Options	Description
🖫	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
🗐	Click on this button to edit parameters for the Business Knowledge Model.
⊨	This option is disabled for Literal Expressions.
✖	This option is disabled for Literal Expressions.
⬆	This option is disabled for Literal Expressions.
⬇	This option is disabled for Literal Expressions.
🗒	Click on this button to validate the Literal Expression. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
▷	This button is enabled when the literal expression is defined for a BusinessKnowledgeModel.

Example — Payment of 2 Decimals

This Business Knowledge Model (BKM) *Payment_2_decimal* is implemented as a Literal Expression.
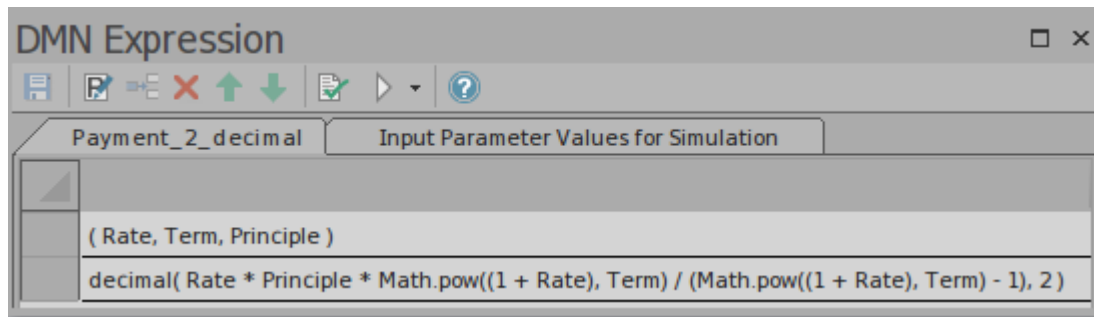
- The BKM defines three parameters: Rate, Term and Principle

Give values for the Test Harness and evaluate the model:
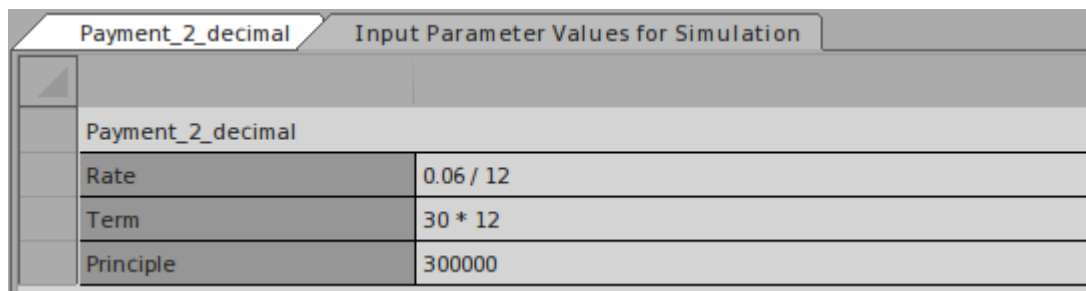
- The runtime parameter value will be displayed; for example, Rate = 00.005
- The BKM's result will be evaluated by the literal expression and the value is displayed on the declaration line; for example, return = 1798.65
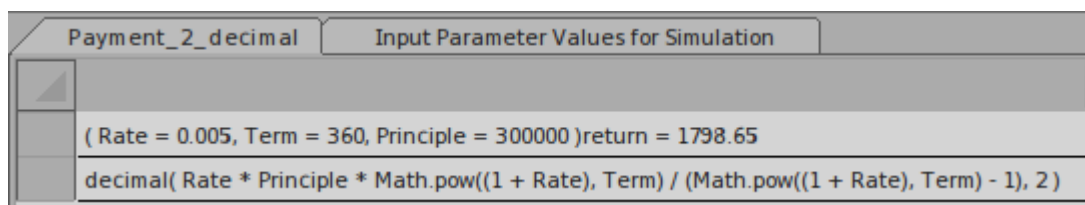
Although the implementation is one line, it is quite complicated. We can re-factor this model with Built-In function and Boxed Context to improve readability:
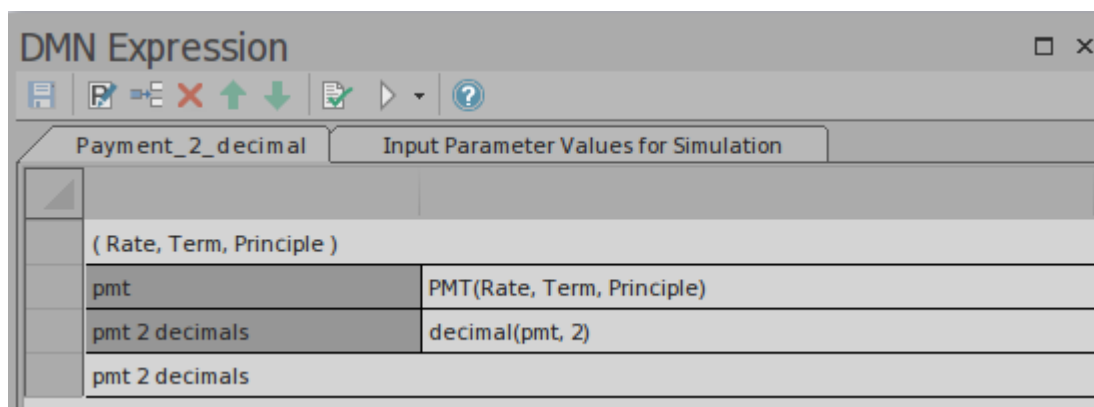
- The Boxed Context defines two variable-expression paired entries; these variables serve as 'local variables', which can be used in later expressions
- Return value: the expression can use the value of 'local variables'

- Any expressions in a Boxed Context can use built-in functions that are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

The simulation result is exactly the same as a Literal Expression:

Payment_2_decimal	Input Parameter Values for Simulation
(Rate = 0.005, Term = 360, Principle = 300000)return = 1798.65	
pmt = 1798.6515754582708	PMT(Rate, Term, Principle)
pmt 2 decimals = 1798.65	decimal(pmt, 2)
pmt 2 decimals	

Expression Editor and Intelli-sense Support

In accordance with the FEEL language specification, the parameter names can contain spaces. This feature makes the expression easier to read. Enterprise Architect also provides Intelli-sense support for editing the expressions with minimal typing and fewer mistakes.

Given a decision hierarchy such as this, the expression in 'Decision3' should be able to use the required decisions (variables).
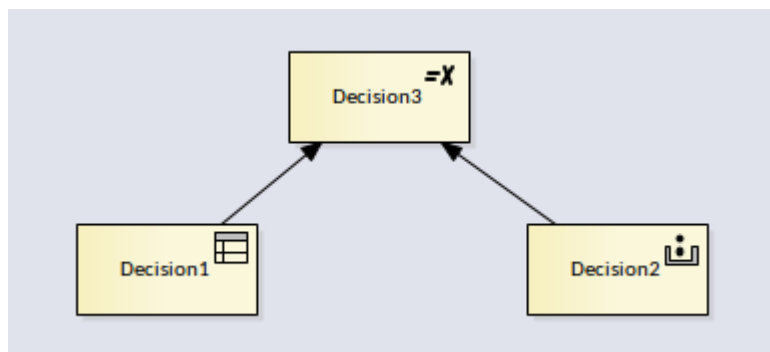
Right-click on the Expression and select the menu option 'Edit Expressions...' to display the 'Expression' dialog.
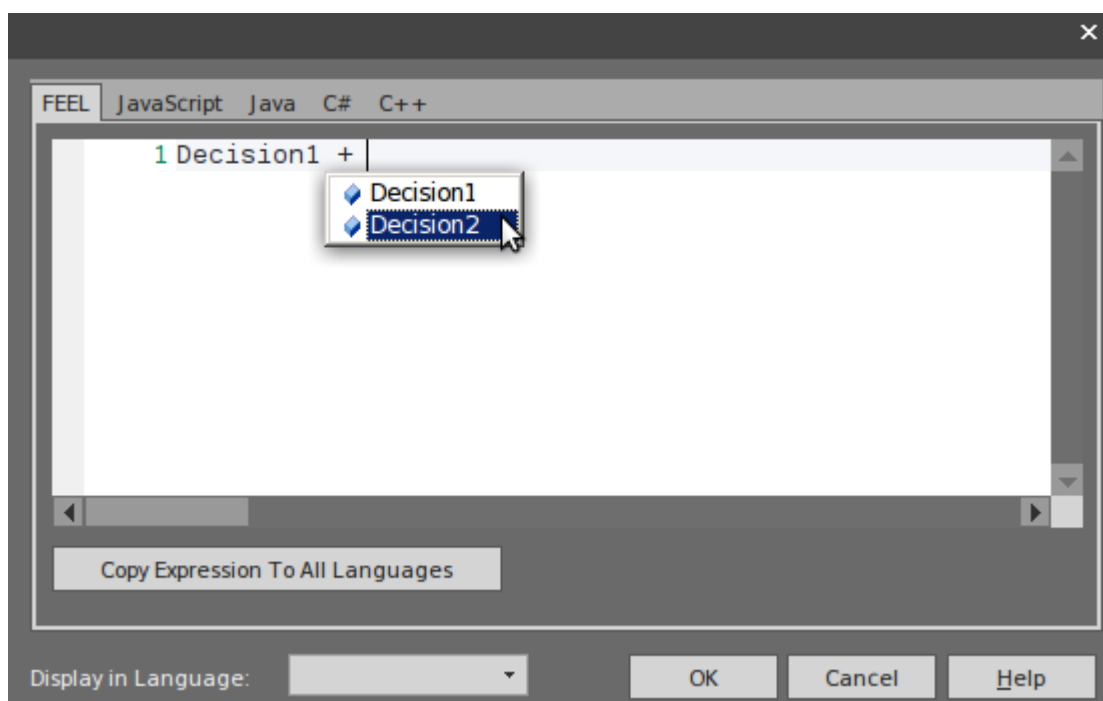
Press Ctrl+Space to show the Intelli-sense menu:

- For 'BKM', all the parameters will be included

- For 'Decision', all the required Decisions will be included

The DMN Model can be generated as source code in JavaScript, Java, C# or C++; since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Boxed Context

A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

Toolbar Options

| Options	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to edit parameters for the Business Knowledge Model.
	Click on this button to add a 'Context Entry' to the boxed context. Each 'Context Entry' consists of a variable and an expression.
	This button is enabled when a ContextEntry's variable is selected. Click on this button to delete the selected Context Entry.
	This button is enabled when a 'Context Entry' variable is selected. Click on this button to move the selected 'Context Entry' up one position.
	This button is enabled when a 'Context Entry' variable is selected. Click on this button to move the selected 'Context Entry' down one position.
	Click on this button to validate the Boxed Context. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
	This button is enabled when the Boxed Context is defined for a Business Knowledge Model.

Example — Installment calculation
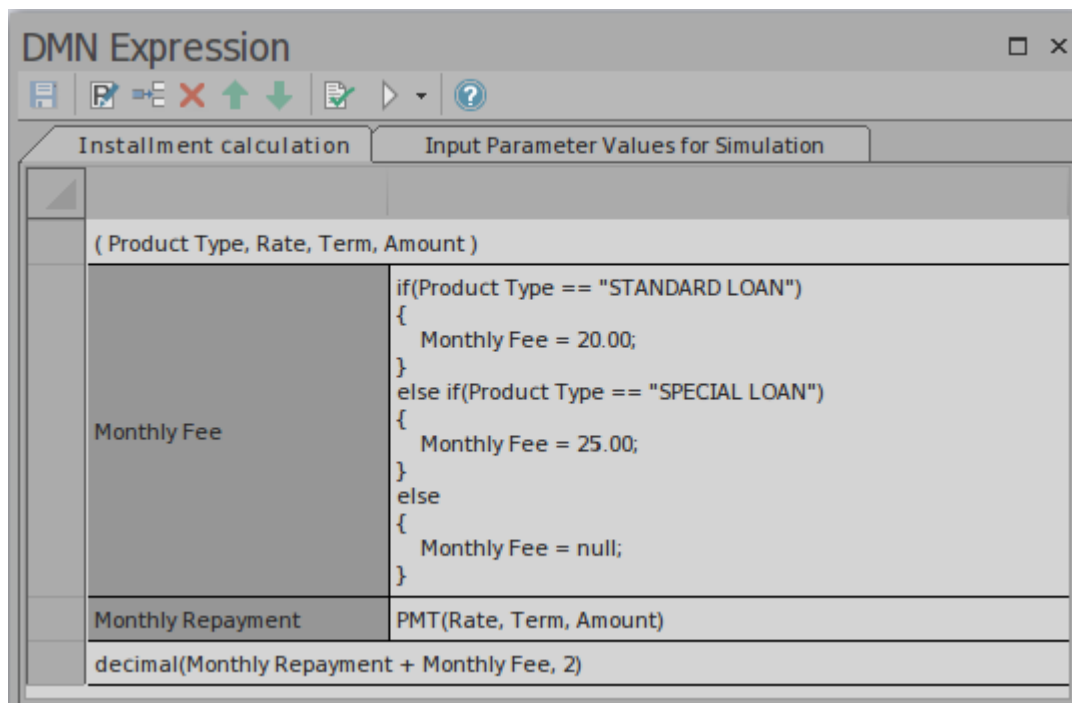
The Business Knowledge Model (BKM) *Installment calculation* is implemented as Boxed Context.

- The BKM defines four parameters: Product Type, Rate, Term and Amount

- The Boxed Context defines two variable-expression pair entries, these variables serve as 'local variables' that can be used in later expressions

- Return value: The expression can use the value of 'local variables'

- Any expressions in a Boxed Context can use built-in functions, which are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

Expression Editor and Intelli-sense Support

The parameter and Context Entry's variable name can contain spaces, according to the FEEL language specification. This feature makes the expression easy to read. In order to help you edit the expressions with less typing and making fewer mistakes, Enterprise Architect provides Intelli-sense support for editing expressions:

Right-click on the Expression | Edit Expressions... the 'Expression' dialog displays
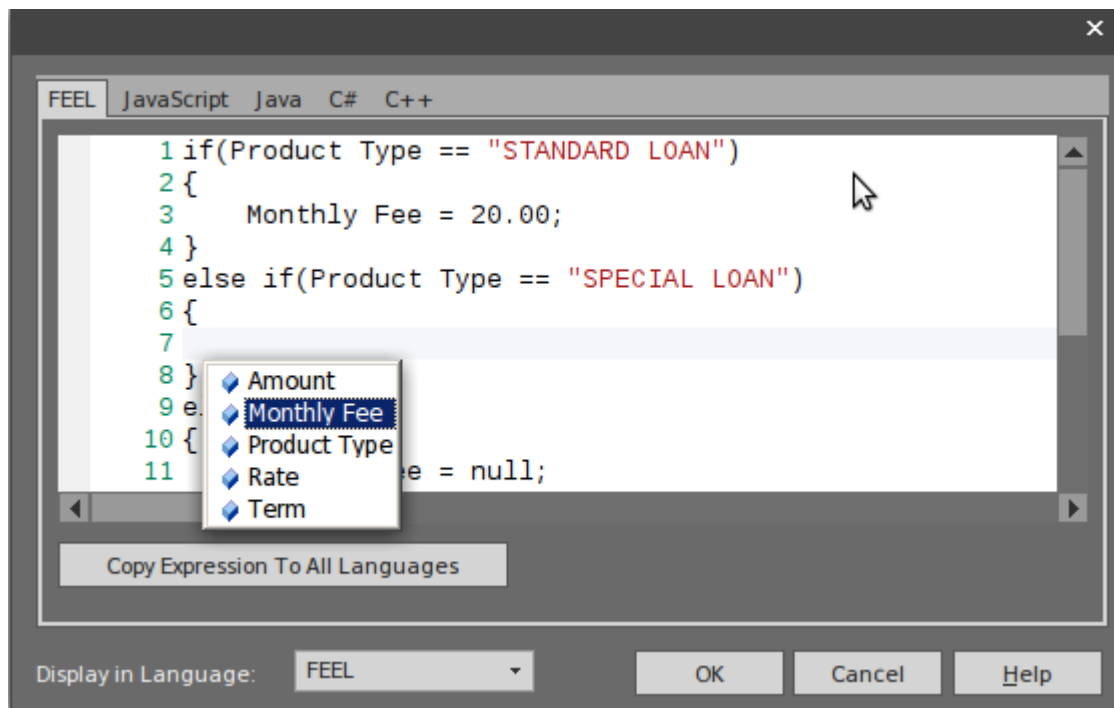
Press Ctrl+Space to show the Intelli-sense menu:

- All the Context Entry Variables earlier than the current one will be included (the context entries later than the current one are excluded)

- For BKM, all the parameters will be included

- For Decision, all the required Decisions will be included

The DMN model can be generated as source code for JavaScript, Java, C# and C++. Since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Test Harness for Business Knowledge Model

Select the 'Input Parameter Values for Simulation' tab and complete the fields.

Click on the Test Harness button on the toolbar; the test result will be presented in the Boxed Context.

- The runtime parameter value will be displayed; for example, 'Rate = 0.00375'

- The 'Context Entry' variable's runtime value will be displayed; for example, 'Monthly Repayment = 1520.05'

- The BKM's result will be evaluated by the last entry and the values displayed on the declaration line; for example, 'return = 1540.06'

You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel.

Menu options are available for this tool bar button. For more information, see the *Business Knowledge Model and Test Harness* Help topic.

Invocation

An invocation is a container for the parameter bindings that provide the context for the evaluation of the body of a business knowledge model. There are two common use cases for an Invocation:

- Bind Input Data to Business Knowledge Model
- Bind parameters or context entry variables to Business Knowledge Model

An example of each is provided at the end of this page.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel with Invocation type
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

Toolbar Options

| Options	Description
🖫	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
🔃	Click on this button to edit parameters for the Business Knowledge Model.
🔁	Click on this button to synchronize with the invoked Business Knowledge Model. For example, if the Business Knowledge Model changes name, parameters, outputs or types, click on this button to synchronize these changes.
🗐	Click on this button to set or change a Business Knowledge Model as an invocation.
🔀	Click on this button to open the invoked Business Knowledge Model in the DMN Expression window.
🖼	When a Business Knowledge Model is implemented as a Decision table, it could define multiple output clauses; the invocation on this Business Knowledge Model might have to specify which output is requested. Click on this button to list all the available outputs in a context menu; the currently configured output is checked.
☑	Click on this button to validate the Invocation. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
▷	This button is enabled when the Invocation is defined for a BusinessKnowledgeModel. Select the 'Input Parameter Values for Simulation' tab, complete the fields and click

	on this button. The test result will be presented on the Decision table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted.
	You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel.
	Menu options are available for this toolbar button. For more information, see the *BusinessKnowledgeModel and Test Harness* Help topic.

Example 1 — Bind Input Data to Business Knowledge Model

A full example can be created with a Model Pattern (Ribbon: Simulate > Decision Analysis > DMN > Apply Perspective > DMN Decision > Decision With BKM : Create Pattern(s))
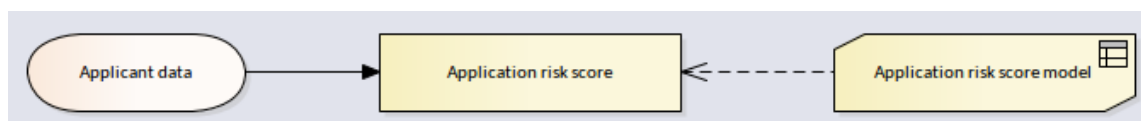

In this example, Input Data *Applicant Data* is typed to *Applicant data Definition,* which has three components.

Applicant data : Applicant data Definition		
	Marital Status : string	"M"
Applicant data Definition	Employment Status : string	"EMPLOYED"
	Age : number	40

The Business Knowledge Model *Application risk score model* is implemented as a Decision table with three inputs and one output.

| Application risk score model		Input Parameter Values for Simulation	
(Age, Marital Status, Employment Status)			
C+	**Age**	**Marital Status**	**Employment Status**	**Partial score**
	[18..120]	S,M	UNEMPLOYED,EMPLOYE...	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6		S	-	25
7		M	-	45
8			UNEMPLOYED	15
9	-		STUDENT	18
10		-	EMPLOYED	45
11			SELF-EMPLOYED	36

The Decision *Application risk score* is implemented as an Invocation to bind the Input Data's 'leaf' components to the BKM's parameters.
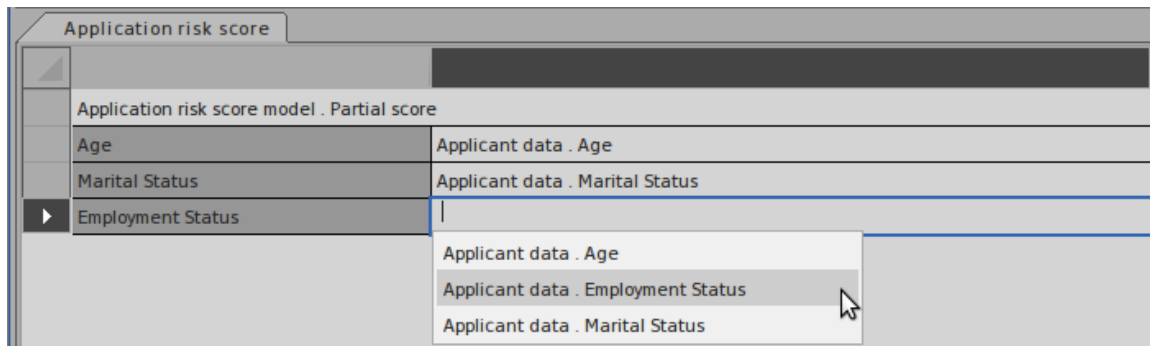
In order to make the binding easier, Auto-Completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's documentation.

Example 2 — Bind Context Entry variables to Business Knowledge Model

The full example can be created using a Model Pattern (Access - Ribbon: Simulate > Decision Analysis > DMN > Apply Perspective > DMN Business Knowledge Model > Business Knowledge Model Invocation : Create Pattern).
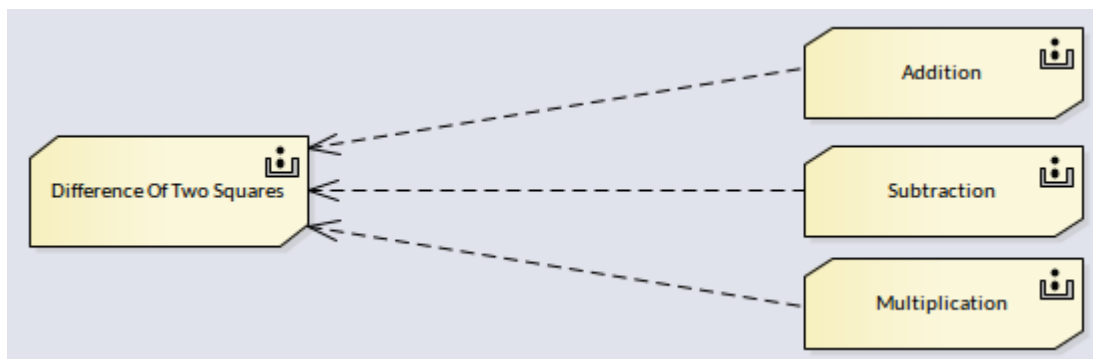
In this example, the BKM *Difference Of Two Squares* is implemented as Boxed Context:

- The variable *sum of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Addition*
- The variable *difference of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Subtraction*
- The variable *difference of squares* is implemented as an invocation by binding local variables *sum of ab* and *difference of ab* to BKM *Multiplication*

In order to make the binding easier, auto-completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's document.

Item Definition

An important characteristic of data items in Decision Models is their structure.

The ItemDefinition element is used to model the structure and the range of values of the input and the outcome of decisions.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create an ItemDefinition
Other	Double-click on a DMN ItemDefinition

Toolbar Options

Option	Description
	Click on this button to save the configuration to the current ItemDefinition.
	Click on this button to add a child component to the selected ItemDefinition.
	Click on this button to add a sibling component to the selected ItemDefinition.
	Click on this button to delete the selected component.
	Click on this button to validate the ItemDefinition; Enterprise Architect will perform a series of validations to help you identify errors in the ItemDefinition.

Allowed Value Enumerations

Each 'Leaf' component of the ItemDefinition can define a string of Allowed Value Enumerations.

For example:

The Allowed Value Enumerations play an important role in Auto Completion. The idea is that you type these values once, and choose from a list later on.

Taking the 'Strategy' ItemDefinition as an example, we can fast fill the 'Allowed Values' field for a Decision table by selection:

Then fast fill the Decision table rules by selection:
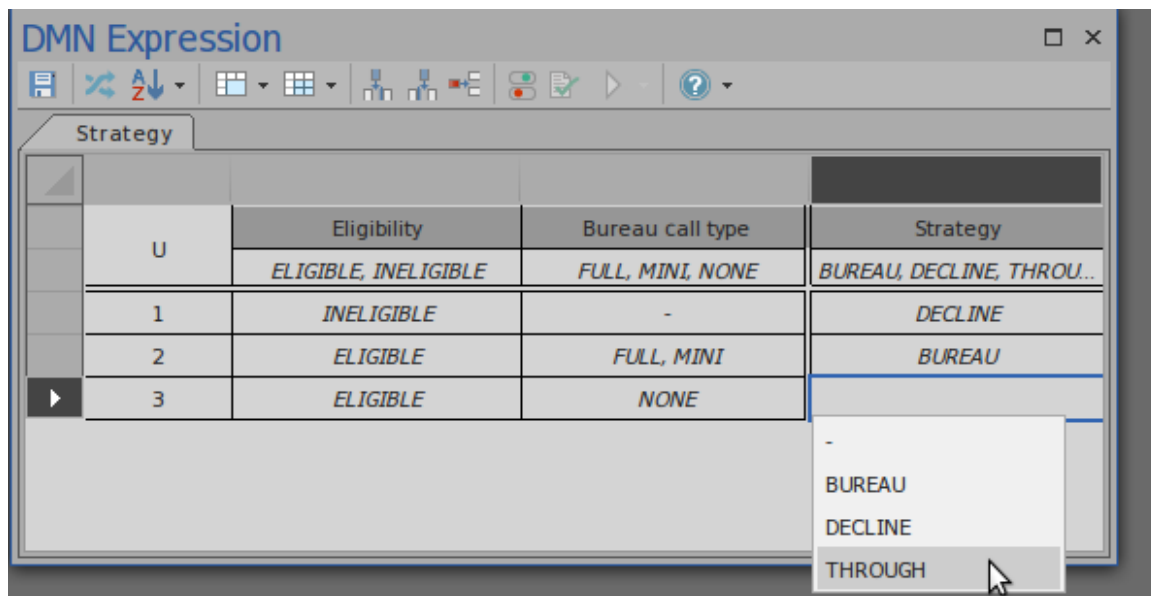
Types of Component

An ItemDefinition element SHALL be defined using only one of these alternative methods:

- Set to a built-in type
- Composition of ItemDefinition elements

In other words, if an ItemDefinition is 'leaf' (no child components), it must be set to a built-in type; if an ItemDefinition has child components, it can not be set a built-in type.

The FEEL language has these built-in types:

- **number**
- **string**
- **boolean**
- days and time duration
- years and months duration
- time
- date and time

Note: 'number', 'string' and 'boolean' are supported by Enterprise Architect for simulation.

In order to set a type for a 'leaf' ItemDefinition, you can use one of these three methods:

- Use the context menu (Recommended)
- Type in the cell after the name by appending ': string', ': boolean' or ': number'
- Input 'string', 'boolean' or 'number' in the tag 'type' for the ItemDefinition
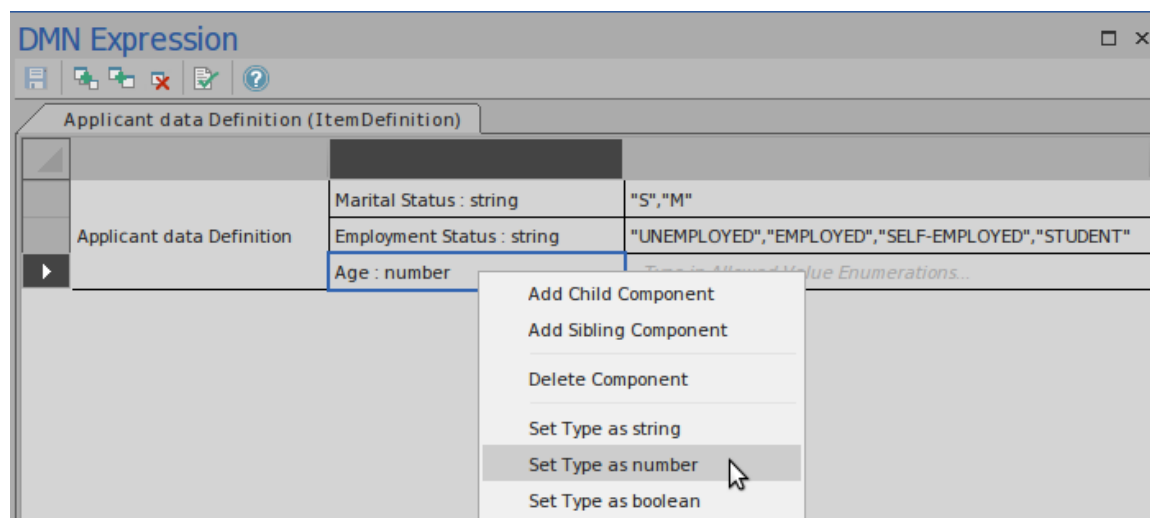
Input Data

An InputData types to an ItemDefinition and carries values used for Decisions.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select / create an InputData
Other	Double-click on a DMN InputData

Toolbar Options

Option	Description
🖫	Click on this button to save the configuration to the current InputData.
🔁	Click on this button to select an ItemDefinition as the type of this InputData.
🗒	Click on this button to open the ItemDefinition element that types this InputData.
📝	Click on this button to open the dialog for editing data sets for this input data. Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the results of a decision by choosing different data sets.
📝	Click on this button to validate the InputData. Enterprise Architect will perform a series of validations to help you identify errors in the InputData.

Auto Completion

If the typing ItemDefinition has a defined 'Allowed Value', then the value for the InputData can be simply chosen from the enumerations.

Data Sets

Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the result of a decision by choosing different data sets.

- The values in the 'default' data set will show in the DMN Expression window when the InputData is selected
- You can add, duplicate or delete a dataset and set an existing dataset as default
- You can export the datasets to a CSV file and import them from a CSV file
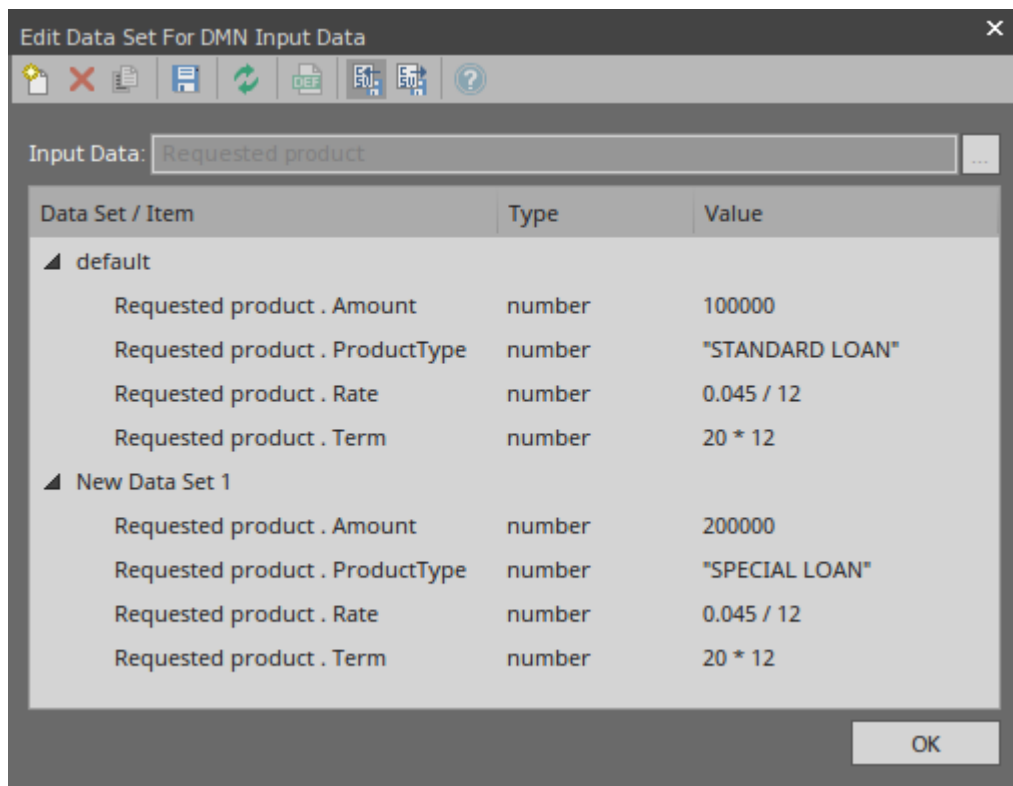
Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression > click on InputData item : ✎ icon
Other	In a diagram, double-click on the DMN InputData element : ✎ icon.

Toolbar Options

Option	Description
✎	Click on this button to create a new dataset.

✖	Click on this button to delete the selected dataset.
	Click this button to duplicate the selected dataset.
	Click on this button to save the datasets to the InputData.
	Click on this button to reload the datasets for the InputData.
DEF	Click on this button to set the selected dataset as default.
	Click on this button to import datasets from a CSV file.
	Click on this button to export the datasets to a CSV file.

BusinessKnowledgeModel & Test Harness

A Business Knowledge Model encapsulates logic without reflecting the usage context. It is implemented as a function definition, with parameters and a DMN expression as body (such as Decision table, Boxed Context or Literal Expressions).

Because the definition of a Business Knowledge Model is self-contained, it is possible to perform 'Unit Testing' by providing arguments for the parameters. For this reason, a Business Knowledge Model will have an 'Input Parameter Values for Simulation' tab in the DMN Expression window.

Parameters of a Business Knowledge Model

In the DMN Expression window, click on the Edit Parameters button on the toolbar to open the 'Edit Parameters' dialog:
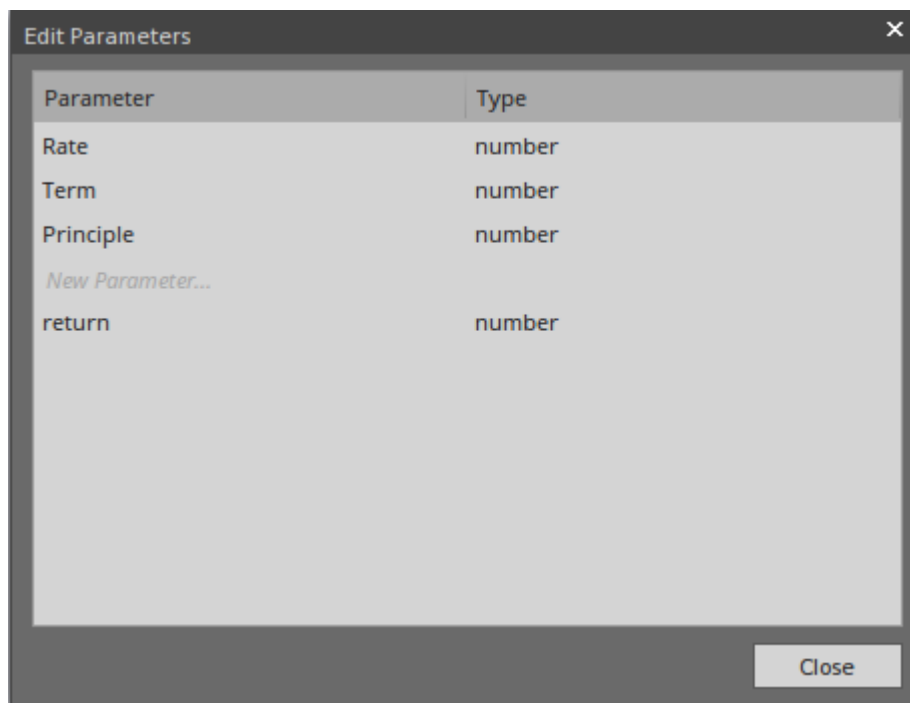
You can perform these actions on the parameters:

- Add a new parameter by typing in the 'New Parameter...' row
- Delete an existing parameter using the context menu
- Modify the name of the existing parameter by in-place editing in the cell
- Change the type of the existing parameter

Test Harness

The Business Knowledge Models (BKMs) described in this section are available from the Model Wizard (Ctrl+Shift+M); select the Requirements | Decision Modeling page.

Decision Table

Example Access:

- Create pattern for 'DMN Decision | A Complete Example'
- Navigate in the Project Browser to 'A Complete Example | Business Knowledge Models | Eligibility rules'
- Double-click on the 'Eligibility rules' element to open the BKM in the DMN Expression window

When a Decision table is created for a Business Knowledge Model, we can test this BKM by binding some values:

Eligibility rules	Input Parameter Values for Simulation			
	(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)			
P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	VERY LOW, LOW, MEDIU...			*INELIGIBLE, ELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

We can provide test values such as these:

Eligibility rules	Input Parameter Values for Simulation
Eligibility rules . Eligibility	
Pre-Bureau Affordability	true
Pre-Bureau Risk Category	"VERY LOW"
Age	16

Click on the Run button on the tool bar; this result is obtained:

Eligibility rules	Input Parameter Values for Simulation			
	(Pre-Bureau Affordability = true, Pre-Bureau Risk Category = VERY LOW, Age = 16)			
P	Pre-Bureau Risk C...	Pre-Bureau Afford...	Age	Eligibility
	VERY LOW	*true*	*16*	*INELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

- The runtime parameter value will take the place of 'Allowed Values' in simulation mode
- Valid rule(s) are highlighted
- Since this Decision table's hit policy is P (Priority) the final result is determined by the order of 'output values'; since 'INELIGIBLE' and 'ELIGIBLE' are the output values and 'INELIGIBLE' comes ahead of 'ELIGIBLE', rule #3 will give the final result and this applicant is 'INELIGIBLE'.

Boxed Context / Literal Expressions

Example Access:

- Create the pattern for 'DMN Business Knowledge Model | Business Knowledge Model Literal Expression'
- Navigate in the Project Browser to 'Business Knowledge Model Literal Expression | Payment'

Double-click on the 'Payment' element to open the BKM in the DMN Expression window

Similar to Decision table, the Business Knowledge Model implemented as a Boxed Expression can be tested as well.

Take this 'Payment_2_decimal' BKM as an example:
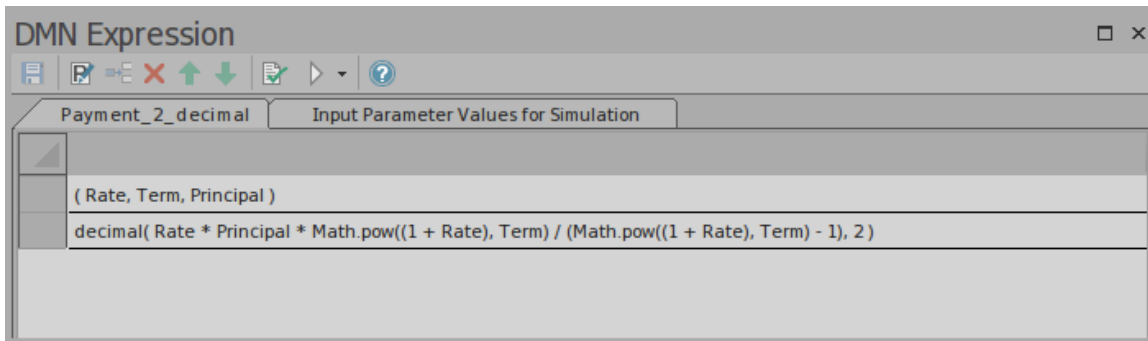
This BKM will calculate the monthly repayment based on interest rate, number of terms and principal amount.

We could provide test values such as these:
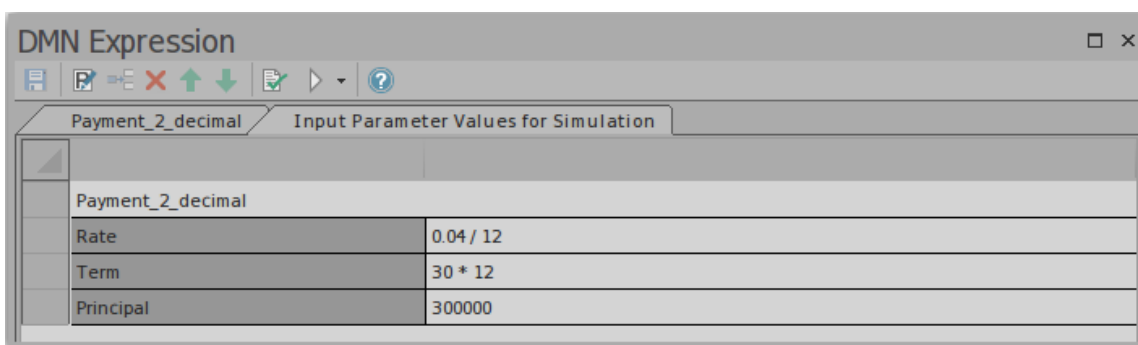
Click on the Run button on the tool bar; this result is obtained:
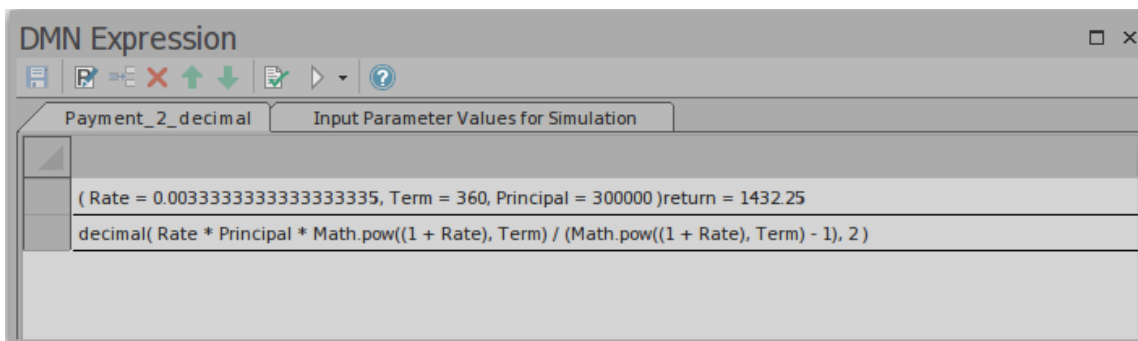
The runtime parameter and return value will be displayed with an equals sign '=', followed by the runtime value.

In this example, given an annual Rate of 4% for 30 years and a principal of $300,000, the monthly repayment is $1,432.25

Note: The DMN Library already has a PMT function defined; this example mainly demonstrates how Literal Expression works and how to test it with a set of arguments.

DMN Expression Auto Completion

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of expressions is implemented largely by 'text'.

To make modeling easy and reliable, Enterprise Architect provides an Auto Completion facility, helping provide the:

· Allowed Values of ItemDefinition

· Input/Output Entries of a Decision Table

· InformationRequirement

Allowed Values of ItemDefinition

The idea is to define allowed value enumerations in ItemDefinition, then compose a list for selection whenever these values are requested.

In this example, ItemDefinition 'Applicant data . Employment Status' defines an enumeration of allowed values.

When editing values for the InputData typed to this ItemDefinition, press the Spacebar on the keyboard to display a list of values to select from.

We could also define multiple data sets for the InputData, as the Auto Completion feature is available on this dialog.
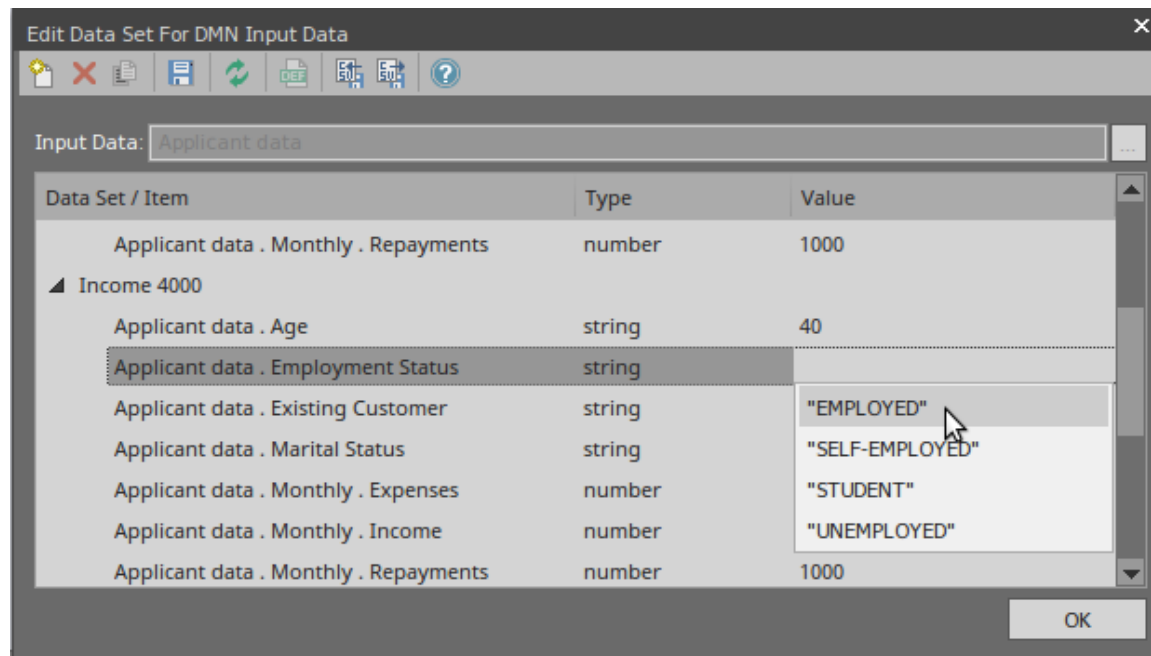
On the Simulation window, you could change the test value to simulate the model; the Auto Completion feature is available on this list as well.
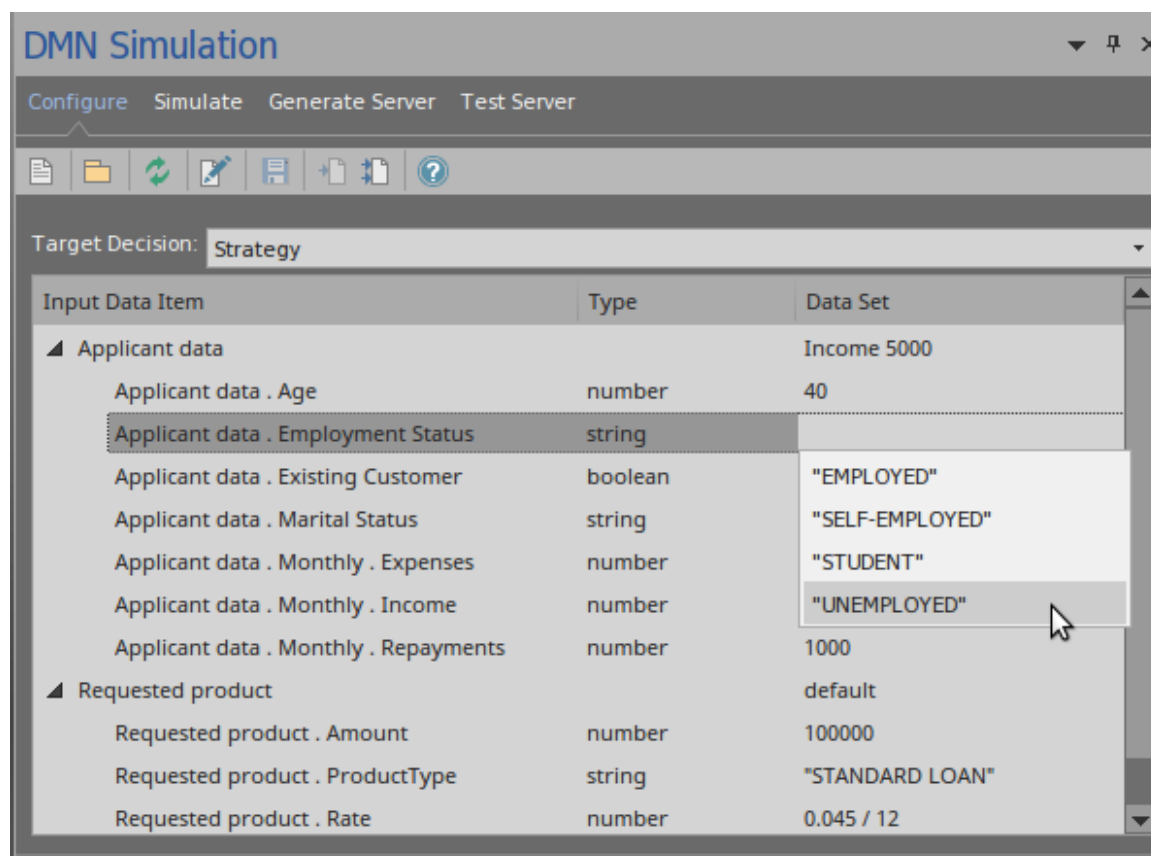
Input/Output Entries of a Decision Table

Take the 'Strategy' ItemDefinition as an example:
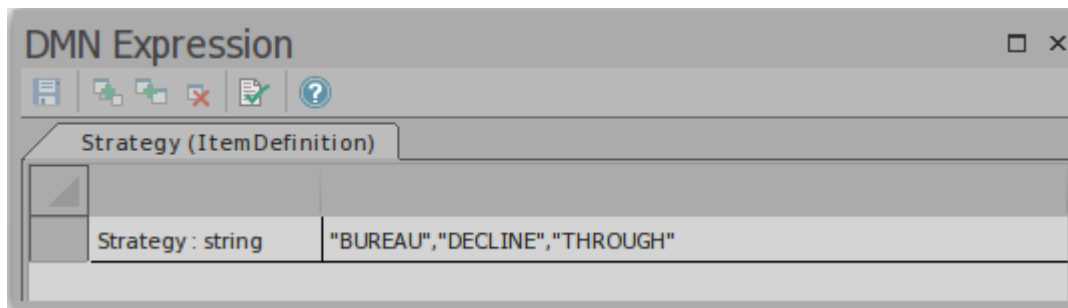
We can quickly fill the 'Allowed Values' field for a Decision table by selection:
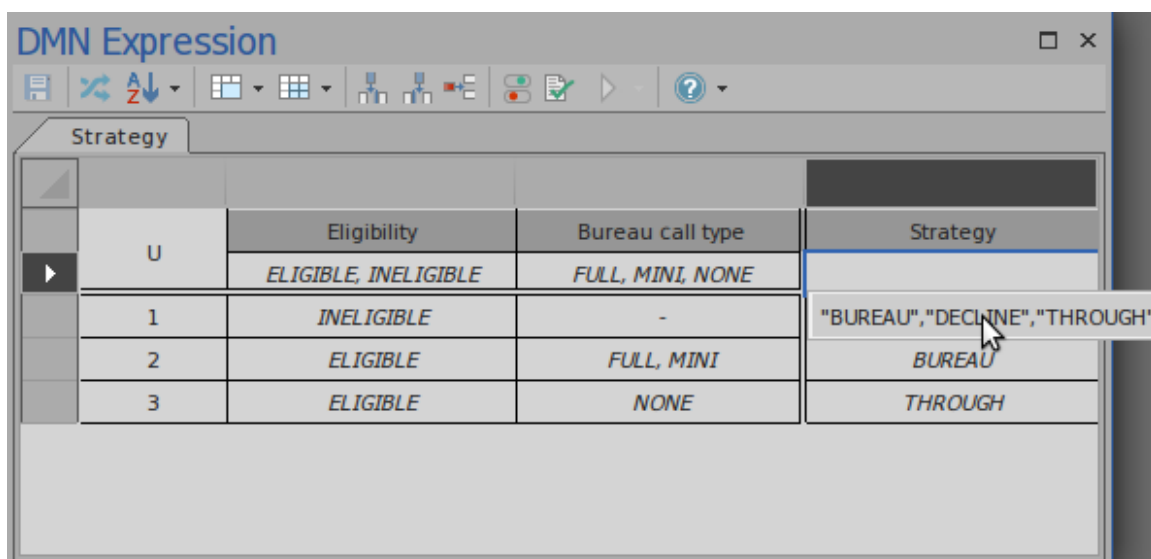
Then we can quickly fill the Decision table rules by selection:
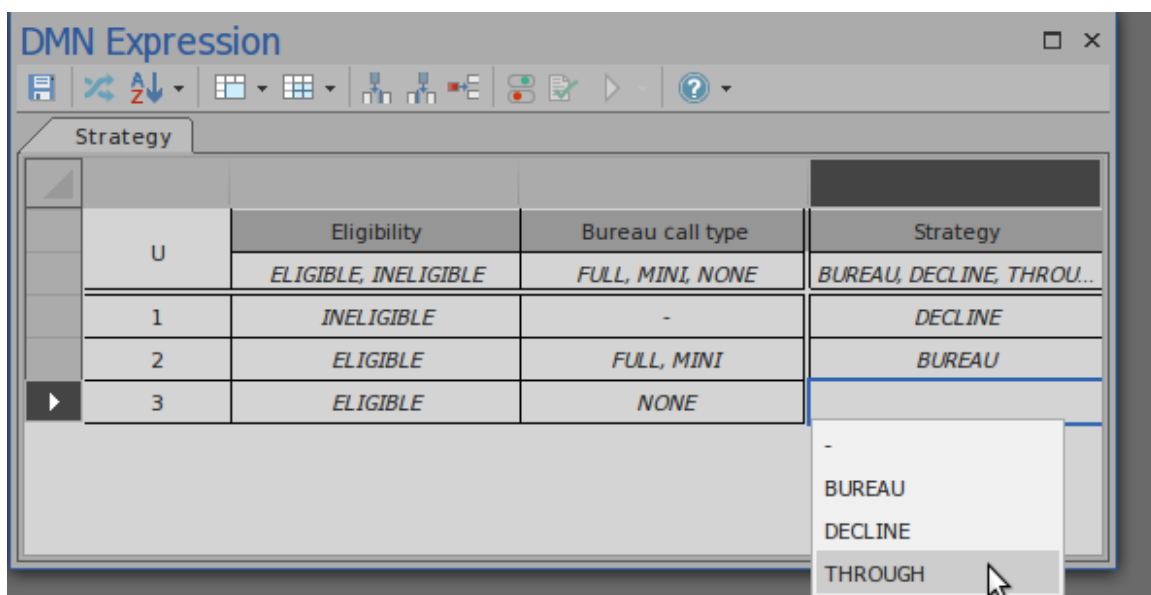
Information Requirement

On a decision hierarchy, a decision might access required decisions and input data; these required elements form a list of variables that can be used by the decision.
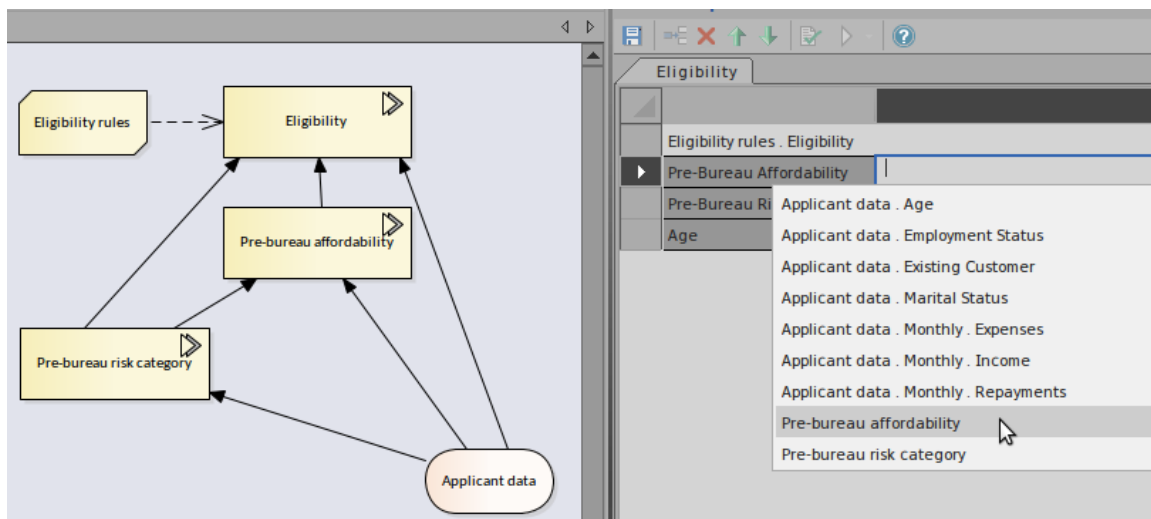
In this example, Decision 'Eligibility' requires two decisions - 'Pre-bureau risk category' and 'Pre-bureau affordability' - and one Input Data item 'Applicant data'.

When setting the binding values for the invoked BusinessKnowledgeModel 'Eligibility rules', an Auto Completion list will prompt for selection.

In this list, there are sub-decision names - leaf components of the input data.

With this feature, you can easily set up an invocation.

DMN Expression Validation

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of expressions are implemented largely by 'text'.

To make modeling easy and reliable, Enterprise Architect provides two features: Auto Completion and Validation.

- Auto Completion: You can select a text string from a list of enumerations rather than type the text in

- Validation: This identifies modeling errors caused by typos, logic incompleteness, inconsistency, and so on

In this topic, we will show you how to validate a DMN Expression.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Simulate : Validate icon

Common validations

Variable Name Validation
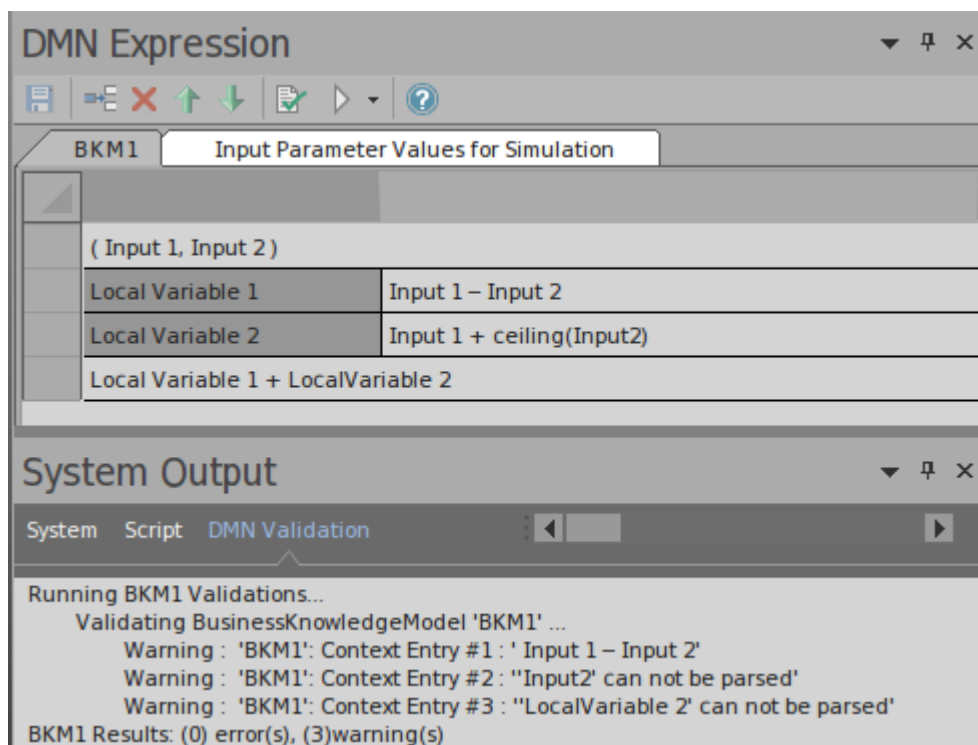
In this example, BusinessKnowledgeModel BKM1 defines two parameters, 'Input 1' and 'Input 2', and two local variables, 'Local Variable 1' and 'Local Variable 2'.

- Context Entry #1 failed because there is a typo: it should be operator '-', but the user typed in '–' instead

- Context Entry #2 failed because there is no space between 'Input' and the number 2; note that the function ceiling is defined in DMN Library so it can be successfully parsed

- Context Entry #3 failed because there is no space between 'Local' and 'Variable'

It is hard to identify these kinds of errors by eyesight, running validation can help identify errors and then you can perform an easy fix.

Dependency Validation

A decision might require other decisions, input data and business knowledge models; these relationships are identified by informationRequirement and KnowldegeRequirement connectors.

When the graph is getting complex, it is quite possible that some connectors are missing or the wrong connector type is being used.
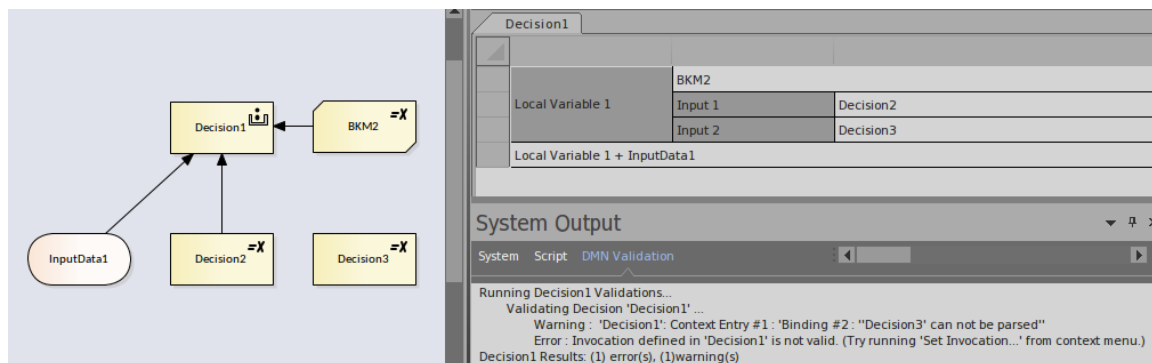
In this example, click on the Validate button, Enterprise Architect will show that:

- 'Decision3' is used by 'Decision1' by binding to a parameter of the called BKM2; however, it is not defined - an InformationRequirement connector is missing

- The Invocation defined in 'Decision1' is not valid; the connector type from 'BKM2' to 'Decision1' should be a KnowledgeRequirement

After fixing these problems, run the validation again:
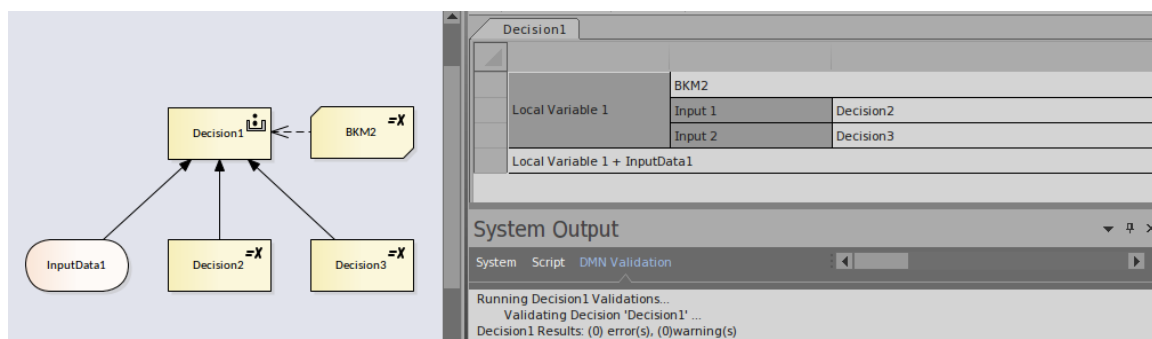
DMN Decision Table Validation

A Decision table is one of the most common and powerful DMN Expressions to express the decision logic. However, modeling a Decision table can also be complicated, especially if multiple input clauses are used in combination for many Decision table rules. Luckily, Enterprise Architect provides a feature to validate a Decision table; this topic explains how to apply this feature.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Manage > Open DMN Simulation > Configure : Validate button

Entries out of range detection

It is good practice to define 'allowed values' for the input clause and output clause of a Decision Table.

Firstly, the enumerations (comma separated string) will enable the Auto Completion feature, where you can choose the value from a list rather than typing it in the entry field.

Secondly, the 'allowed values' enable Enterprise Architect to perform a range check for the entry values.

In this example:

- The 'Age' input clause defines a range of [20..120], causing the first rule [18..21] to fail
- The Marital Status clause defines an enumeration of 'S, M', causing rule #7 with value 'D' to fail

You can fix this issue by changing the 'allowed value' range or the rule entry, based on the actual business rules.

Completeness detection — report gaps in the rules

The gaps in rules for a Decision table mean that, given a combination of input values, no rule is matched. This indicates that some logic or rule might be missing (unless a default output is defined).

When the Decision table defines rules with lots of number ranges, it is hard to detect gaps by eyesight and too hard to compose test cases.

For example,

DMN Expression

Post-bureau risk category table | Input Parameter Values for Simulation

(Existing Customer, Application Risk Score, Credit Score)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
1	false	<120	<590	HIGH
2	false	<120	[590..610]	MEDIUM
3	false	<120	>610	LOW
4	false	[120..130]	<600	HIGH
5	false	[120..130]	[600..625]	MEDIUM
6	false	[120..130]	>625	LOW
7	false	>130	-	VERY LOW
8	true	<=100	<580	HIGH
9	true	<=100	(580..600]	MEDIUM
10	true	<=100	>600	LOW
11	true	>100	<590	HIGH
12	true	>100	[590..615]	MEDIUM
13	true	>100	>615	LOW

System Output

System   Script   DMN Validation

Running Post-bureau risk category table Validations...
    Validating BusinessKnowledgeModel 'Post-bureau risk category table' ...
        Warning : DecisionTable "Post-bureau risk category table" Completeness Violation: No rule exists for Existing Customer:"true", Application Risk Score:"<=100", Credit Score:"580"
Post-bureau risk category table Results: (0) error(s), (1)warning(s)

The validation reports a gap in the rules. You might first perform a merge on the Decision table to focus on the ninth input 'Credit Score' and easily detect the error in input entry **(**580..600], which should be **[**580..600].

Rule Overlaps detection for Unique Hit Policy

When rules overlap, given a combination of input values, multiple rules are matched. This is a violation if the Decision table specifies 'Unique' as its Hit Policy.

When the Decision table defines rules with lots of number ranges, it is hard to detect gaps by eyesight and too hard to compose test cases.

For example:

DMN Expression

Post-bureau risk category table | Input Parameter Values for Simulation

(Existing Customer, Application Risk Score, Credit Score)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
1	false	<120	<590	HIGH
2	false	<120	[590..610]	MEDIUM
3	false	<120	>610	LOW
4	false	[120..130]	<610	HIGH
5	false	[120..130]	[600..625]	MEDIUM
6	false	[120..130]	>625	LOW
7	false	>130	-	VERY LOW
8	true	<=100	<580	HIGH
9	true	<=100	(580..600]	MEDIUM
10	true	<=100	>600	LOW
11	true	>100	<590	HIGH
12	true	>100	[590..615]	MEDIUM
13	true	>100	>615	LOW

System Output

System   Script   DMN Validation

Running Post-bureau risk category table Validations...
    Validating BusinessKnowledgeModel 'Post-bureau risk category table' ...
        Warning : DecisionTable "Post-bureau risk category table" Consistency Violation: Rules "4, 5" overlap with input "false, [120..130], [600..610)"
Post-bureau risk category table Results: (0) error(s), (1)warning(s)

The validation reports an overlap in the rules. You might first perform a merge on the Decision table to focus on the 3rd input 'Credit Score' and easily detect the overlap between '<610' and '[600..625]'. You could fix this issue either by changing rule #4 to '<600' or by changing rule #5 to '[610..625]', based on the actual business rules.

DMN Simulation

After a Decision Model is created, you can configure a DMN Simulation Artifact and Validate, Run, Step Through or Debug the model.

By switching data sets, you can do what-if analysis to ensure the model meets the requirements of the business.

You can also generate code for the DMN Server. Currently, four languages are supported: Java, JavaScript, C++ and C#.

In order to simulate BPMN and DMN together, Enterprise Architect's BPSim Execution Engine accepts a DMN Server in Java language. Before the BPSim Execution Engine executes, ensure that the DMN Server is tested. Luckily, we can Run or Debug the testing for the Java DMN Server easily in Enterprise Architect.

Configure DMN Simulation

- Create a DMNSimConfiguration element in a Package and double-click on it to open it; all DMN elements in this Package (Decision, BusinessKnowledgeModel, InputData ItemDefinition) will be loaded to the Simulation window

- The 'Target Decision' combo box will be filled with all the decisions; choose a target decision - the dependent InputDatas will be filled in the list

- Choose a defined dataset by clicking on the Dataset cell in the list (for example, choose Dataset 'Income 5000' for InputData 'Applicant data'; choose 'default' for InputData 'Requested product')

Simulate DMN Model

- When a Target Decision is specified, the 'Simulation' page will be filled with the decisions, in dependency order

- Each decision row can be expanded to show the invoked BusinessKnowledgeModel

- Click on the Run button to evaluate all the decision values based on the values for Input Data

- Click on the Step button to evaluate a single decision and watch the DMN Expression window, which clearly shows the input value for the decision and output based on the input; the diagram containing the decision hierarchy will

highlight the executed decisions and show the runtime results on a label
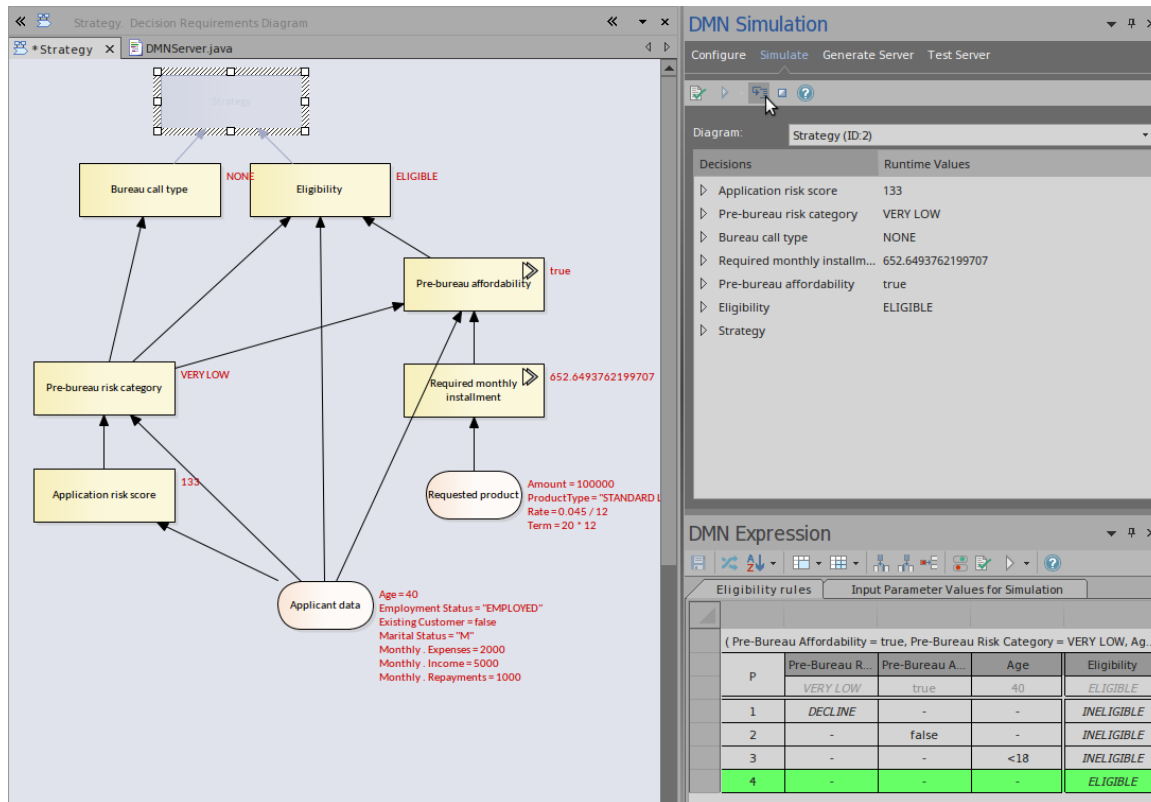
In this example, the decision 'Eligibility' returns a string 'ELIGIBLE' and invokes BusinessKnowledgeModel 'Eligibility rules' by binding the parameters as shown:

- Bind 'Pre-Bureau Affordability' to the dependent decision 'Pre-bureau affordability' (runtime value: True)
- Bind 'Pre-Bureau Risk Category' to the dependent decision 'Pre-bureau risk category' (runtime value: VERY LOW)
- Bind 'Age' to the field 'Age' in dependent input data 'Applicant data' (runtime value: 40)

The BusinessKnowledgeModel 'Eligibility rules' has a Hit Policy P (Priority), meaning that multiple rules can match, but only one hit should be returned; the ordering of the list of output values is used to specify the (decreasing) priority.

In this run time case ('Pre-Bureau Affordability' = true, 'Pre-Bureau Risk Category' = VERY LOW, 'Age' = 40), only one rule with output 'ELIGIBLE' matches.

Configure DMN Simulation

A DMNSimConfiguration Artifact contains information to simulate a DMN model depicted by Decision Requirements diagrams.

Access

Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation > Configure Page
Other	Double-click on a DMNSimConfiguration element

Toolbar Options

Option	Description
	Click on this button to select or create a DMNSimConfiguration element.
	Click on this button to set a Package for the DMNSimConfiguration Artifact. All DMN elements under this Package or its sub-Packages will be loaded.
	Click on this button to reload DMN elements from the configured Packages. For example, when some DMN elements are modified, run this command to reload the Package so that the changes will be taken into account for DMN Simulation.
	Click on this button to open the dialog for editing data sets for the selected input data.
	Click this button to save the DMN Simulation window information to the DMNSimConfiguration element, including the: • Target Decision • Selected Dataset for each dependent InputData
	Click on this button to export the 'name = value' records for all the InputDatas and their specified DataSets to a BPMN 2.0 DataObject (write to the Notes).
	Click on this button to export the 'name = value' records for the selected InputData and its specified Dataset to a BPMN 2.0 DataObject (write to the Notes). The values defined in the BPMN 2.0 DataObject will be read by the Enterprise Architect BPSim Execution Engine.

DMNSimConfiguration Artifact

You can create a DMNSimConfiguration element in one of these ways:

- In the DMN Simulation Window, click on the Artifact button on the toolbar
- On a Decision Requirements Diagram, drag the DMNSimConfiguration element from the toolbox

By default, all DMN elements in the current Package (Decision, BusinessKnowledgeModel, InputData ItemDefinition) will be loaded to the Simulation window.

Decision Hierarchy

When a Package is loaded, a Decision Requirements Graph (DRG) and decision dependency hierarchy are created. The DMN InformationRequirement connectors determine the hierarchy.

- All the decisions will be listed in the 'Target Decision' combo box
- When a 'Target Decision' is specified, all the dependent Decisions, invoked BusinessKnowledges, InputDatas and the typed ItemDefinitions will be determined

DataSet & Input Data

When the Target Decision is selected, all the dependent InputDatas are added to the list.

You can choose a dataset for simulation from the list of defined datasets of an InputData.
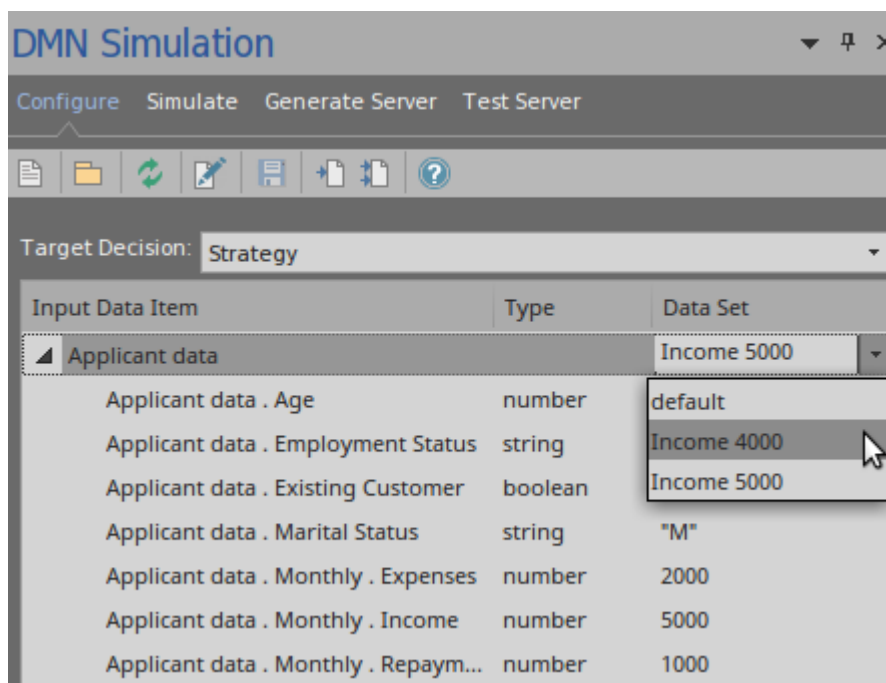
You can also change the dataset value in the list, if the ItemDefinition specifies an Allowed Value Enumeration. The auto-completion feature is supported and you can simply choose from a list of enumerations.
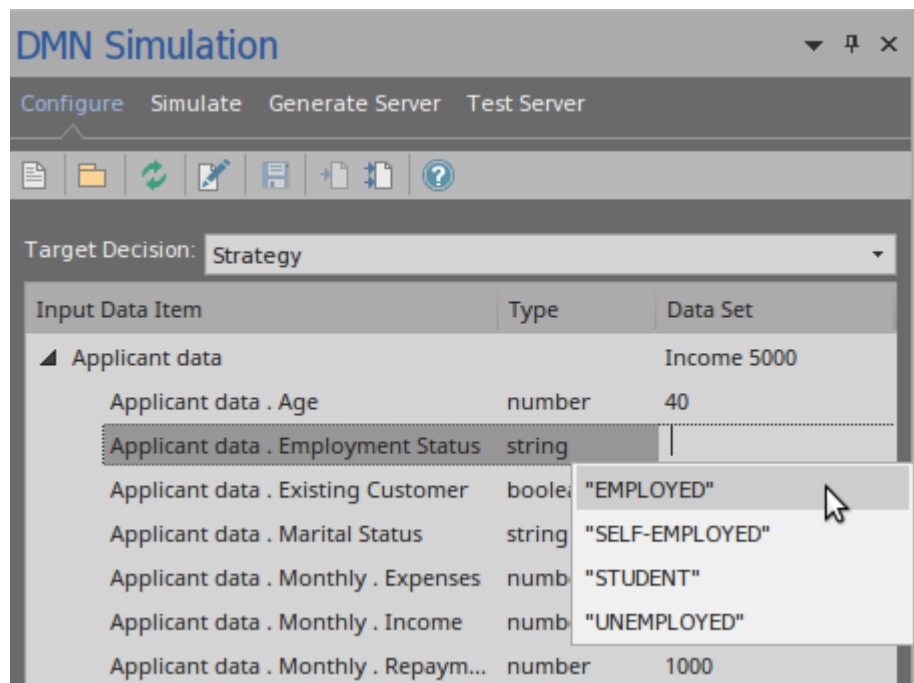
You can also edit the datasets for an InputData by clicking on the Dataset button on the toolbar to open the 'Dataset Edit' dialog.
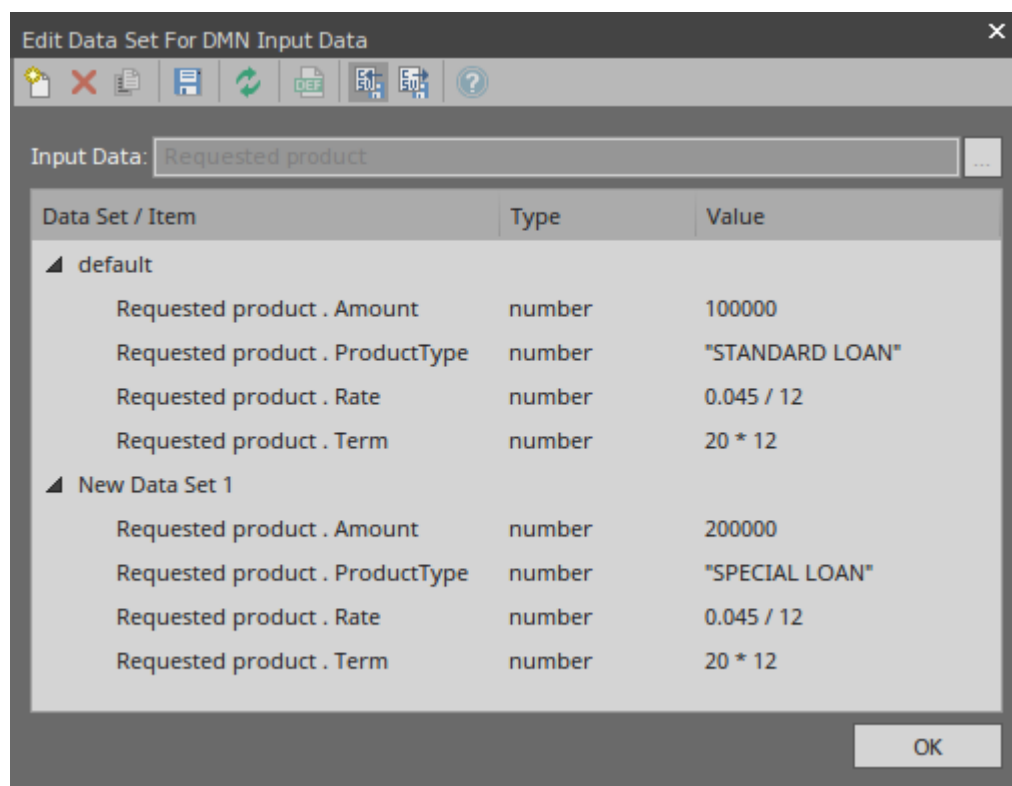
Simulate DMN Model

A DMNSimConfiguration Artifact contains information to simulate a DMN model depicted by Decision Requirements diagrams.

Access

Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation	Simulate Page
Other	Double-click on a DMNSimConfiguration element |Simulate Page

Toolbar Options

Option	Description
	Click on this button to validate all the dependent DMN elements based on the Target Decision. Note: A Decision/BusinessKnowledgeModel/InputData/ItemDefinition that is on the Target Decision hierarchy will not be considered. For example, you have some unfinished Decision elements in the Package; as they have no relationship to the Target Decision, they will not impact the simulation.
	Click on this button to execute the decision hierarchy in order. The results will be represented on the diagram and shown in the 'Runtime values' column.
	Click on this button to step through the decision hierarchy in order. One click will evaluate one decision element. With this feature, you will be able to see the decision-making process; the decision logic and runtime values will be displayed clearly in the DMN Expression window.
	Click on this button to exit the simulation mode.

Simulation Overview

When a Target Decision is specified, the 'Simulation' page will be filled with the decisions, in dependency order.

When Executing or Stepping Through the Decision Hierarchy, the decisions will be evaluated in order:

- The runtime result will be showing in the Decision row
- The runtime result will be displayed on the diagram
- The decision logic and input/output data will be presented in the DMN Expression window
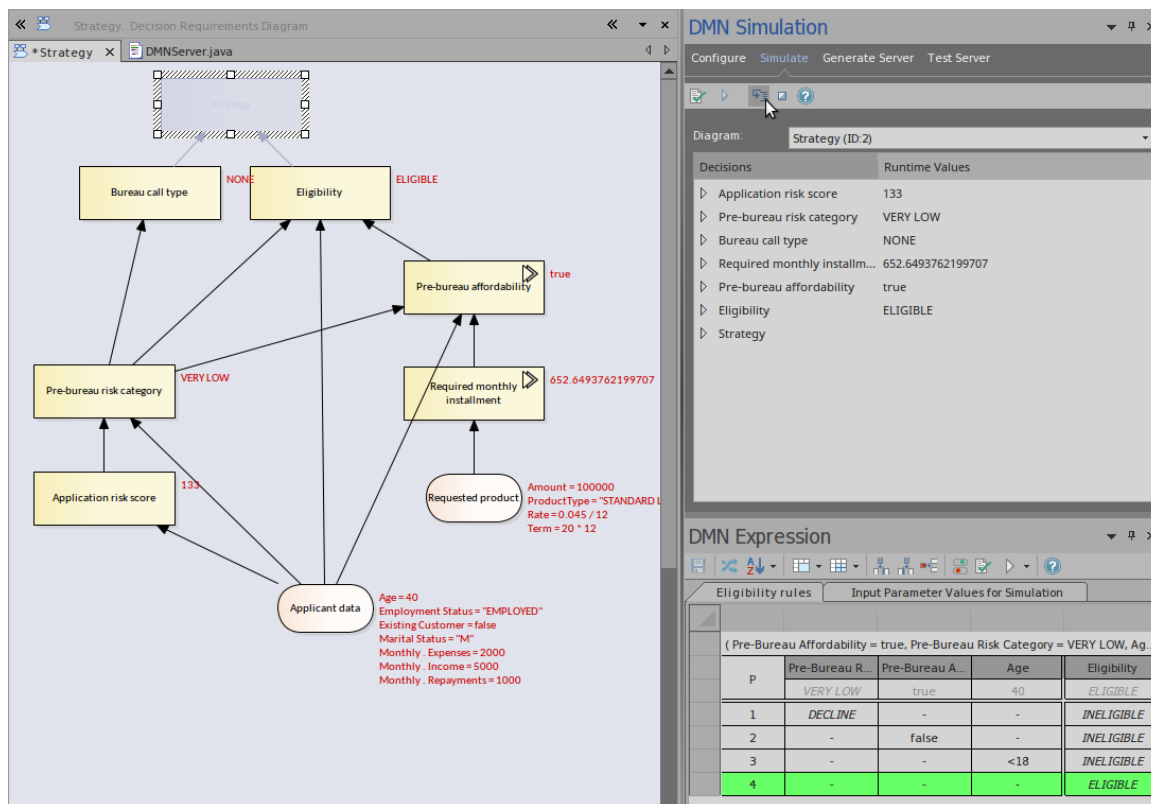
Invocation Hierarchy

You can expand the Decision in the list to see the invocation hierarchy.
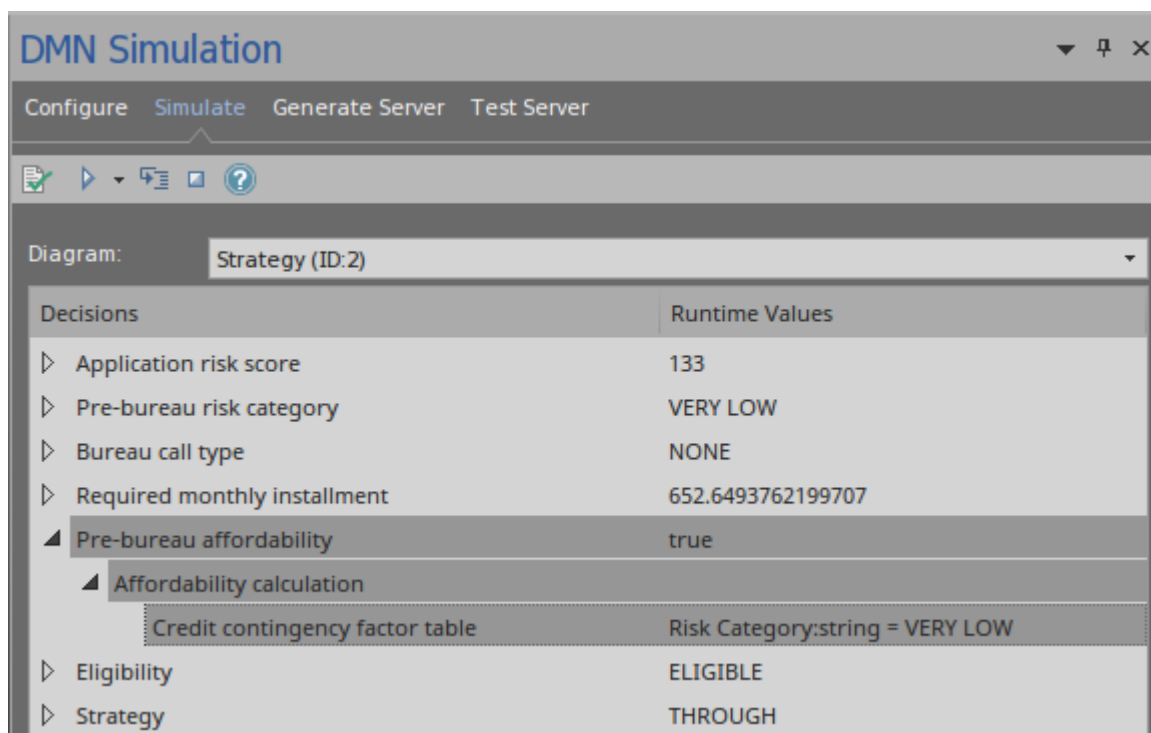
For example:

- Decision 'Pre-bureau affordability' invokes BusinessKnowledgeModel 'Affordability calculation'

- BusinessKnowledgeModel 'Affordability calculation' further invokes another BusinessKnowledgeModel 'Credit contingency factor table'

Advanced Debugging

Although Enterprise Architect provides a validation feature to help you locate many modeling issues and DMN expression issues, the simulation might still fail (rarely but possible) due to uncaught issues.

However, Enterprise Architect provides the ability to debug the code that is running behind the simulation. You can also modify the code and run it in cycles until the issue is found and fixed.

The Execute button on the toolbar displays a menu with these options:
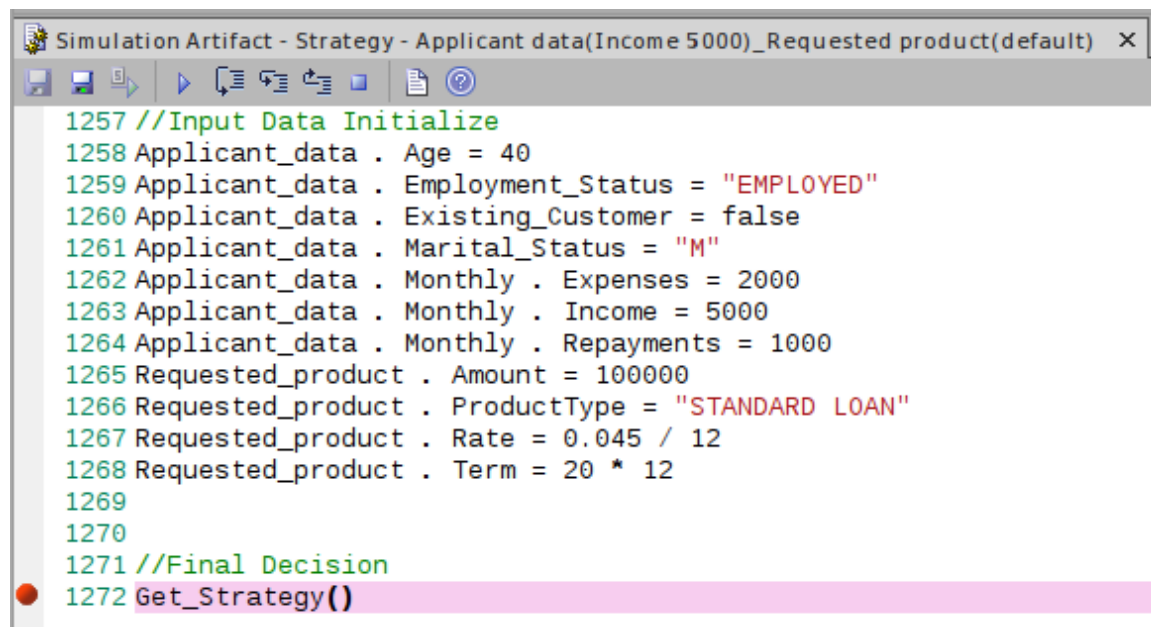
- Generate New Script (Scripting Window)

- Update Selected Script (Scripting Window)

- Run Selected Script (Scripting Window)

- Edit DMN Template

If you select 'Generate New Script (Scripting Window), the Scripting window displays showing a script created in a Package named 'DMN'.

- The default script name is composed of these parameters: 'ArtifactName - TargetDecision - InputData1(DataSet)_ InputData2(DataSet)_...'

Double-click on this file to open it in the Enterprise Architect Script editor, set a breakpoint, and debug the file.

By selecting the script in the Scripting Window, and if the script matches the model (by the 'Simulation Script Identifier' in the script), you enable the menu option 'Run Selected Script'.

You can customize the DMN Template to generate the correct script for simulation.

DMN Module Code Generation & Test

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C#. The DMN Module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or your own project.

Enterprise Architect also provides a 'Test Module' page, which is a preprocess for integrating DMN with BPMN. The concept is to provide one or more BPMN2.0::DataObject elements, then test if a specified target decision can be evaluated correctly or not.

If any error or exception occurs, you can create an Analyzer Script to debug the code of the DMN Module and Test Client.

After this 'Test Module' process, Enterprise Architect guarantees that the BPMN2.0::DataObject elements work well with the DMN Module.

You then configure BPSim by loading DataObjects and assigning DMN Module decisions to BPSim Properties, which will be further used as conditions on the Sequence Flows outgoing from a Gateway.

Modeling & Simulation

We took the example available from Model Patterns - Ribbon: Simulate | DMN | Apply Perspective | DMN Decision | Decision with BKM
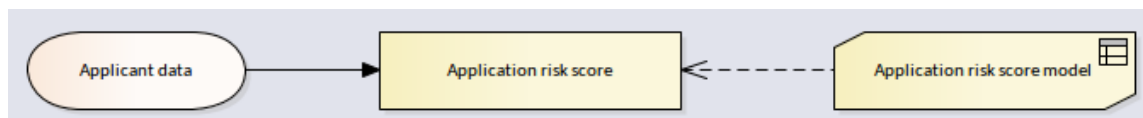
Here is a brief overview of the model:

- Item Definition

- Input Data

- Business Knowledge Model (Decision Table)

- Decision (Invocation)

- Simulation

- Export InputData to a BPMN2.0 DataObject

Activate the 'Configure' page, and click on the Export DataObject button on the toolbar.

DMN Module: Code Generation

Activate the 'Generate Module' tab of the DMN Simulation window, select the DMN elements you want to generate to the server, specify the file path and language, then click on the Generate button.

(Note: For Java, the path has to match the Package structure.)

- Add the elements to the module, then click on the Add button on the toolbar to open the 'DMN Element Selection' dialog
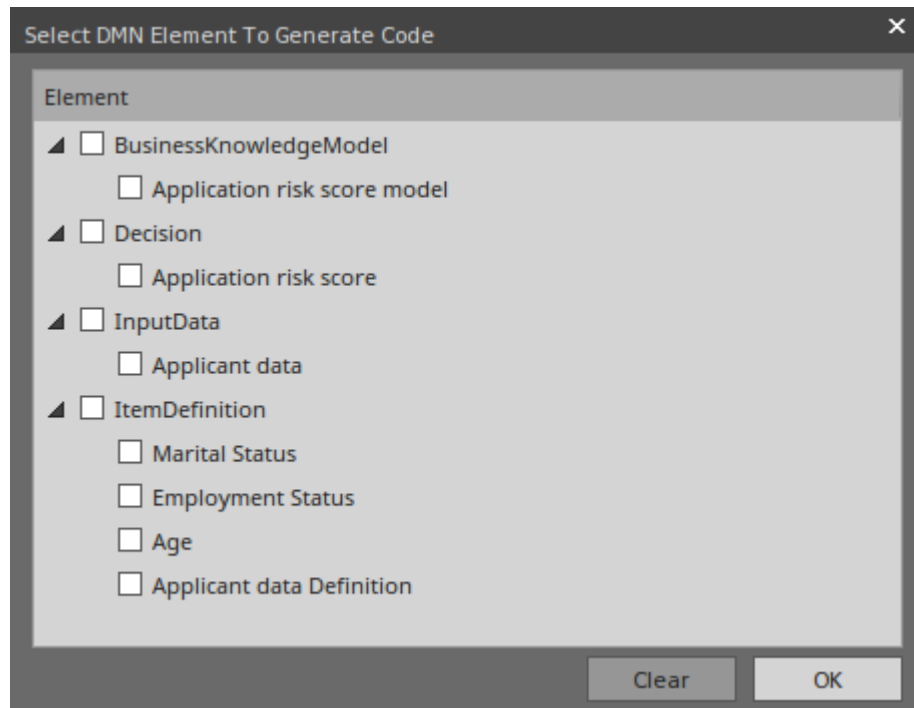
- Click on the decision you want to generate to the server; in this example we select the decision 'Application risk score'

Note: All the dependencies will be selected automatically.
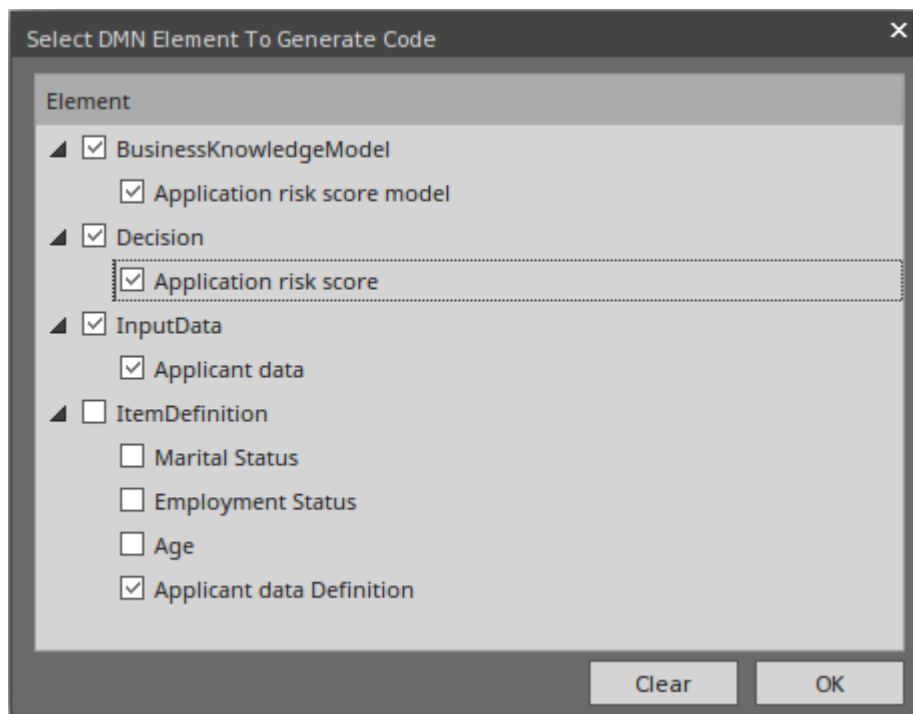
- Generate the DMN Module; give the Java file a Package name, then click on the Generate button
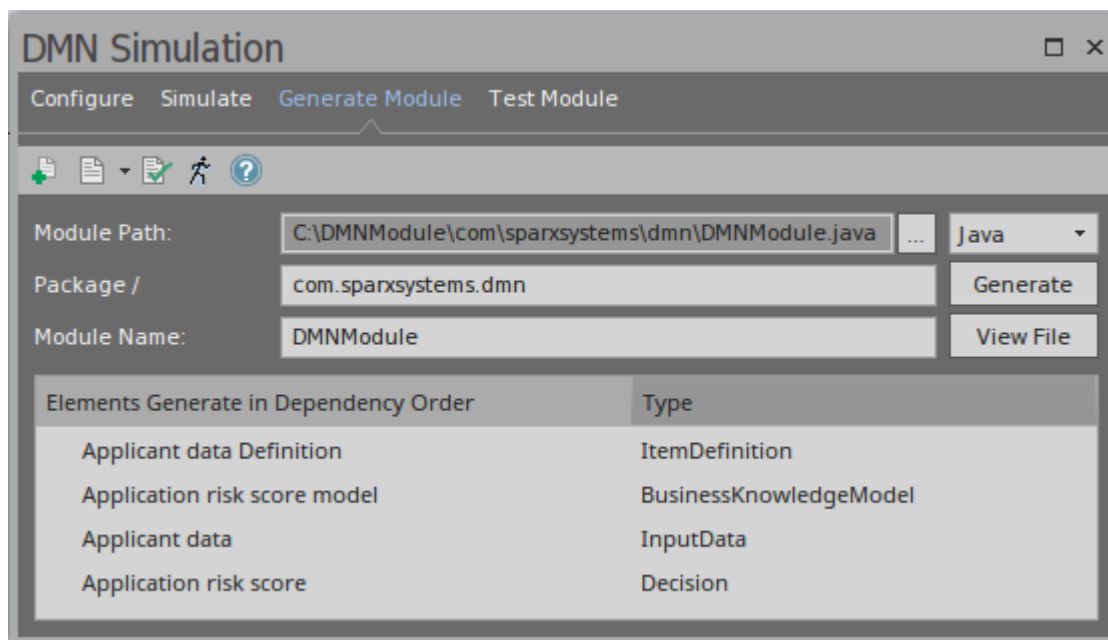
Note: In this example, the Module Path ends with '\com\sparxsystems\dmn', which matches the Package 'com.sparxsystems.dmn'.

- Click on the Test button to access the 'Test Server' page

DMN Server: Test client

If this page was activated from the 'Generate Server' page, the 'DMN Module' field will be filled automatically with the generated DMN Server's path. Otherwise, click the [...] button to browse for a DMN Server file.

Click the Add DataObject button to add one or more BPMN2.0 DataObject(s) to the list, choose a decision from the combo box, then click the Run button on the toolbar.
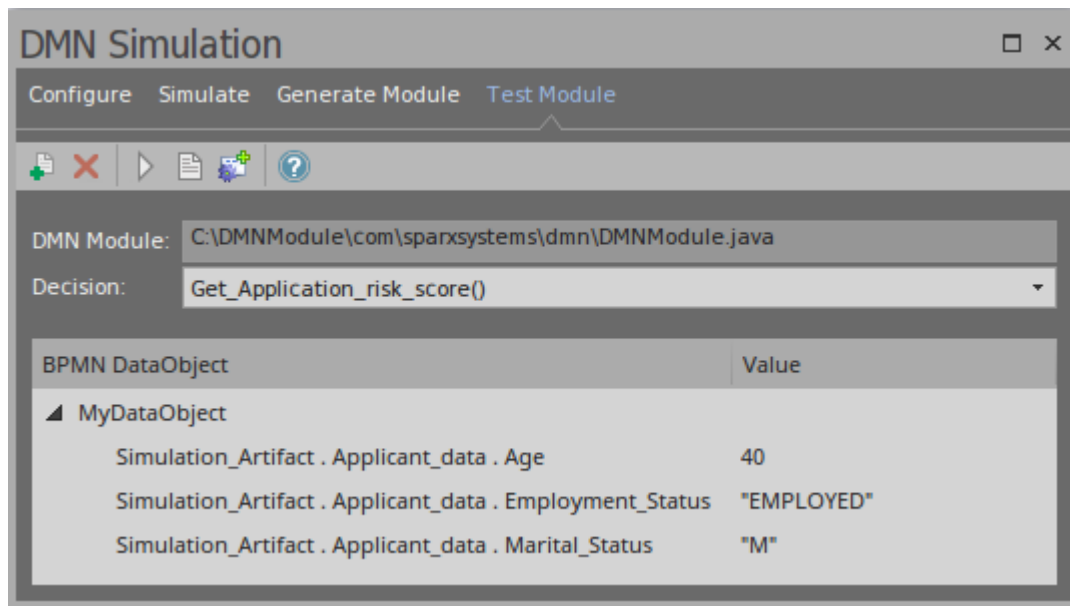
In the System Output window, this message indicates the DMN Server and BPMN2.0 DataObject can work well with each other to evaluate the selected decision.

Running Test Client for DMN Server...

  dmnServer.Application_risk_score: 133.0

Result : 133.0

The Running completed successfully.

If there are errors, create an Analyzer script by clicking the toolbar button and fix the issue.

Important: This 'Test Module' step is recommended before integrating DMNServer.java to the Enterprise Architect BPSim Execution Engine.

Integrate DMN Module Into UML Class Element

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C# and test it.

The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured. Since a Class element can define a StateMachine, after integration with the DMN module the Executable StateMachine simulation will generically be able to use the power of the DMN Module.

In this topic, we will explain the process of integrating a DMN Model with a UML Class element:

- Class element
- DMN Model(s)
- DMN Binding to Class & Intelli-sense
- Code Generation on Class Element

Class Element's Requirement

Suppose we have a Class *Applicant* with an operation *AffordabilityForProduct* that evaluates whether the applicant can afford a loan product.

A simplified model resembles this:
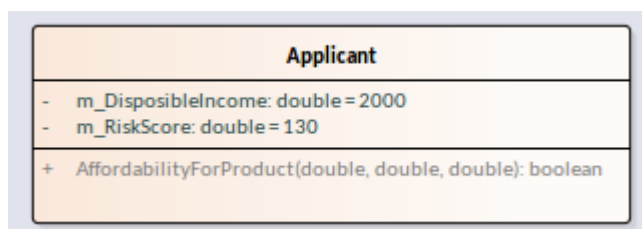
The Class *Applicant* contains two attributes, which are actually calculated from more basic data such as the applicant's monthly income, expenses, existing repayments, age and employment status.

In this example, however, we simplify the model by skipping these steps and providing disposable income and risk score directly. In the 'DMN Complete Example' (Model Patterns), you can see all the more detailed steps.

DMN Model(s)

In this example, we have two disjoint DMN Models to show that a UML Class can integrate multiple DMN Models.

- Installment Calculator

This DMN Model computes the monthly repayment based on amount, rate and terms. It is composed of an InputData, a Decision and a Business Knowledge Model.
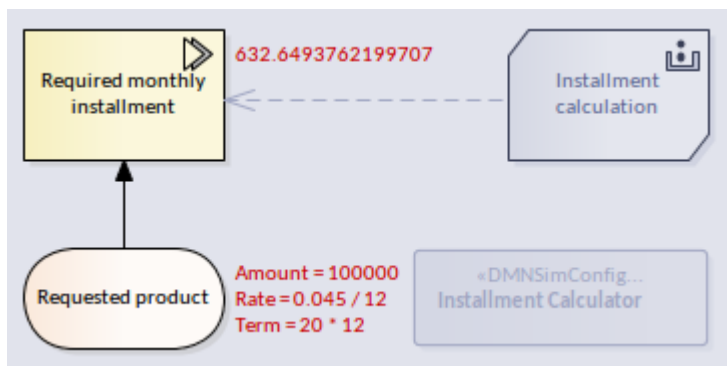
- Credit Contingency Factor Calculator

This DMN Model computes the credit contingency factor based on the applicant's risk score. It is composed of an InputData, two Decisions and two Business Knowledge Models.
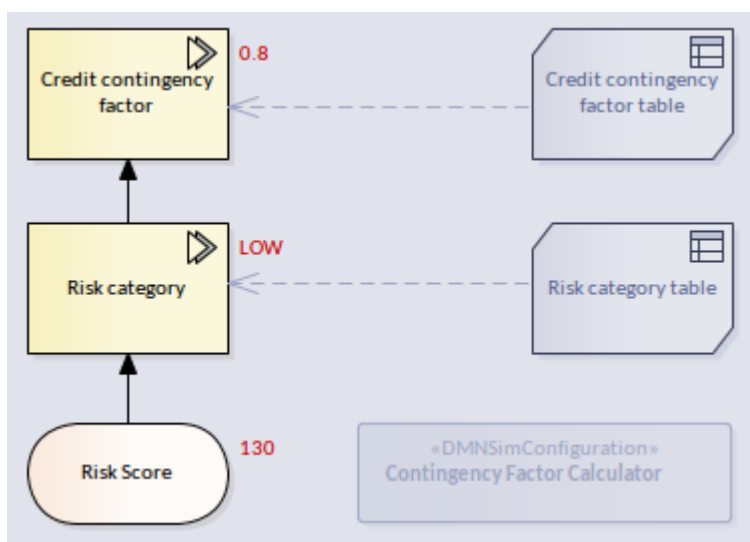
Note: In this example, we focus on how to integrate DMN Modules into a Class Element; the DMN element's detail is not described here. The full example is available in Model Patterns and the EAExample model.

- Generate code for both DMN Models
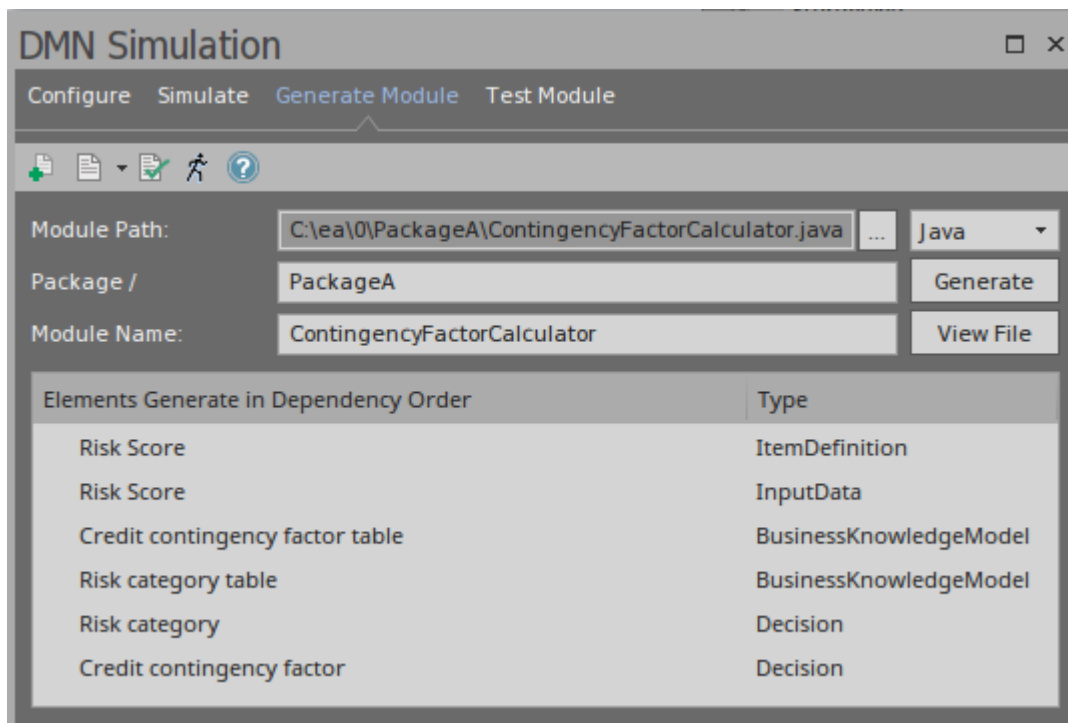
Click on the Generate button, and check that you can see this string in the System Output window, 'DMN' page:

DMN Module is successfully compiled.
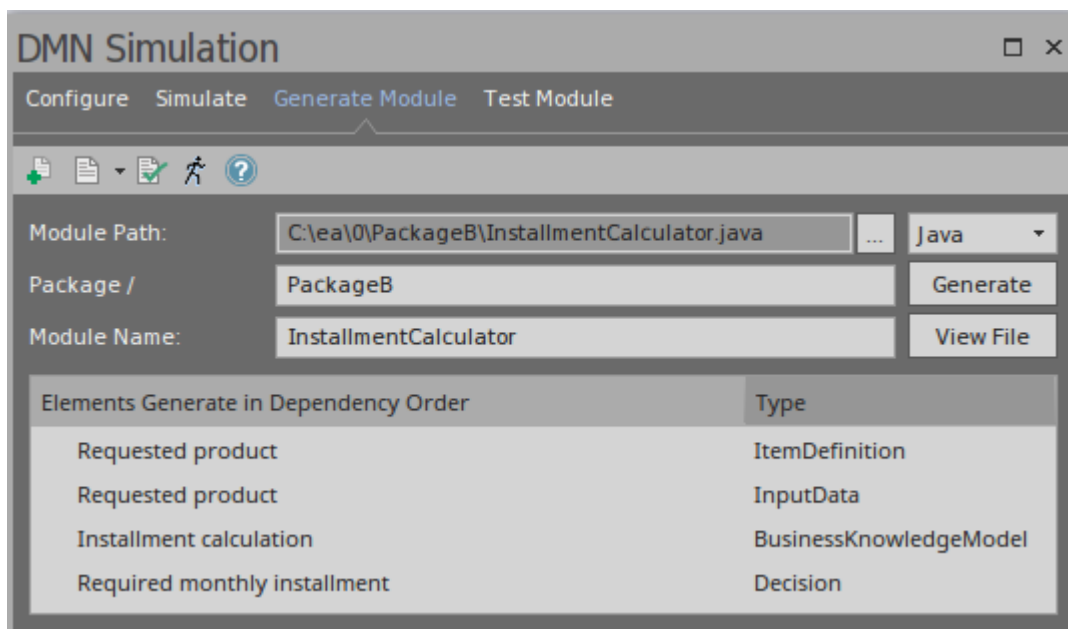
Note: Since this model uses a built-in function PMT, the DMN Library has to be included:
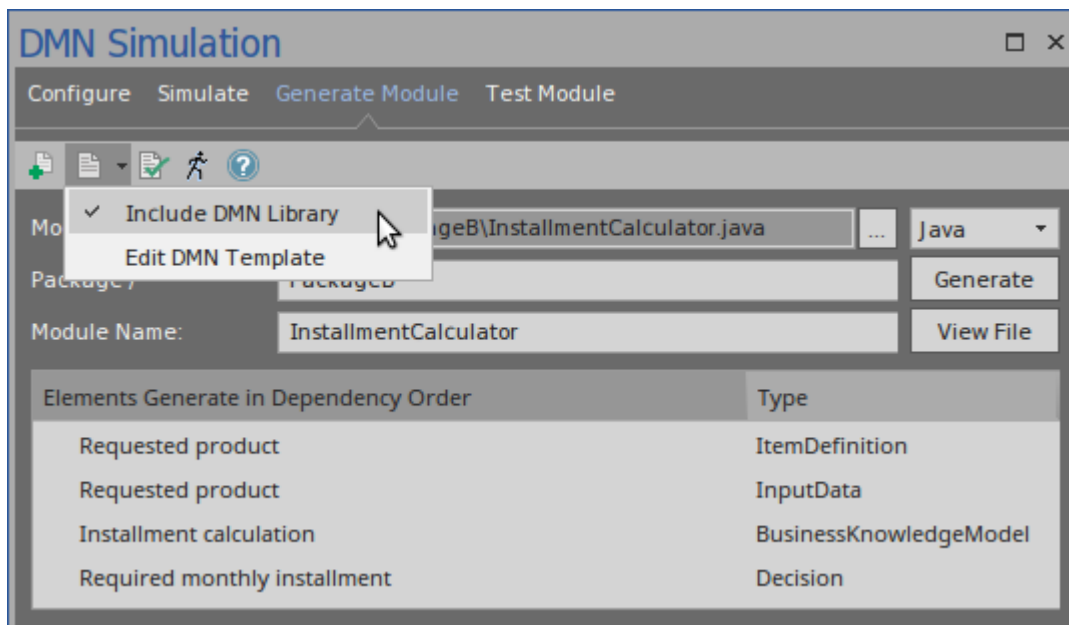
Click on the Generate button, and check that you can see this string in the System Output window, 'DMN' page:

DMN Module is successfully compiled.

DMN Binding to Class & Intelli-sense

Put the two DMNSimConfiguration Artifacts on the Class diagram.
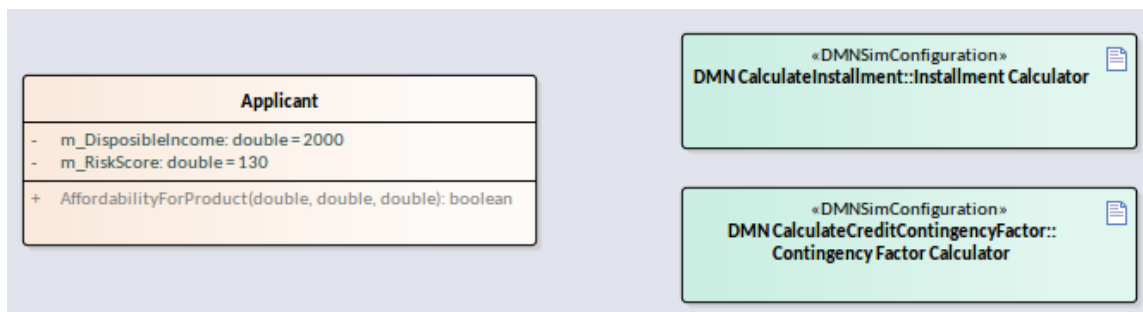
Use the Quick Linker to create a Dependency connector from the Class *Applicant* to the DMN Artifact.

On creation of the connector, the dialog will prompt you to choose the operation to be bound to the DMN module.
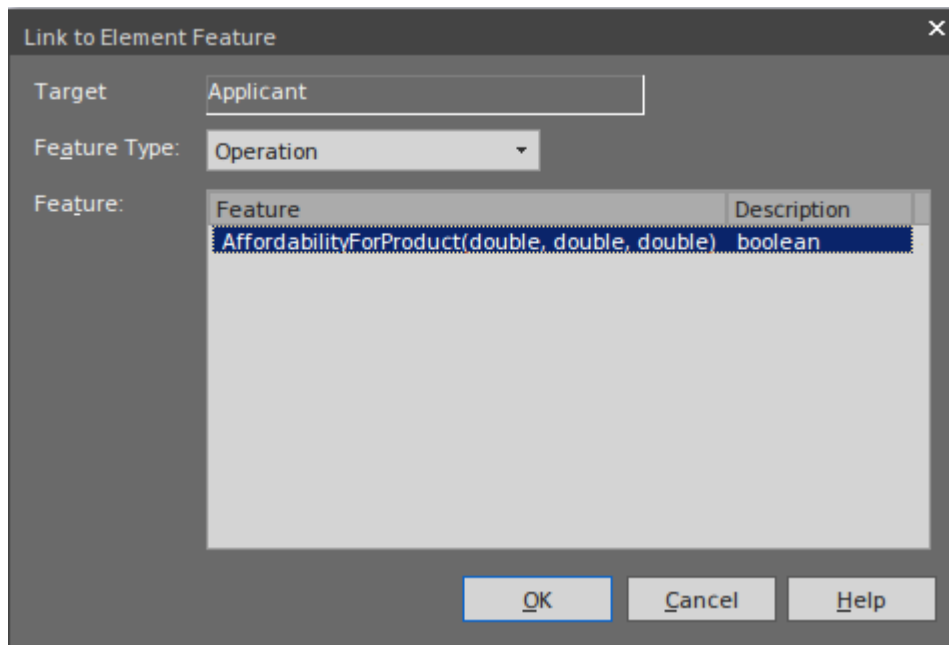
Here is what has happened after the DMN Binding:

- The operation takes a stereotype <<dmnBinding>>
- The Dependency connector is linked to the operation
- Multiple DMN Artifacts can be bound to the same operation
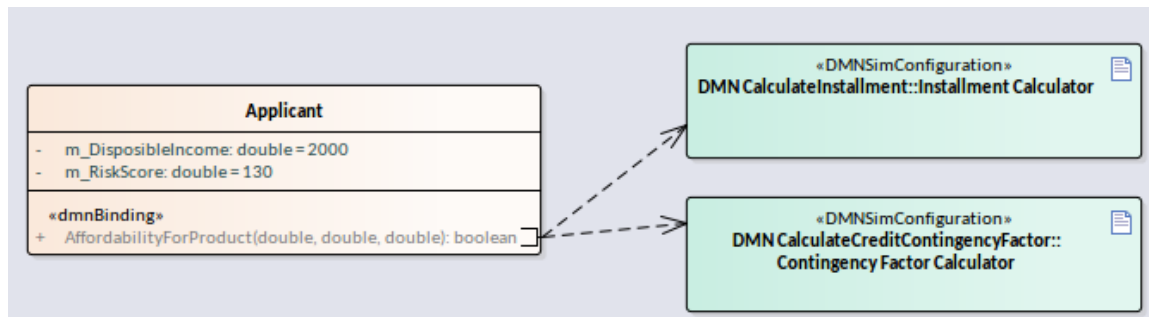
After DMN Bindings, Intelli-sense for the operation's code editor will support DMN Modules. To trigger the Intelli-sense, use these key combinations:

- Ctrl+Space - in most of the cases
- Ctrl+Shift+Space - when Ctrl+Space does not work after a parenthesis '('; for example, a function's arguments, or inside an 'If' condition's parentheses
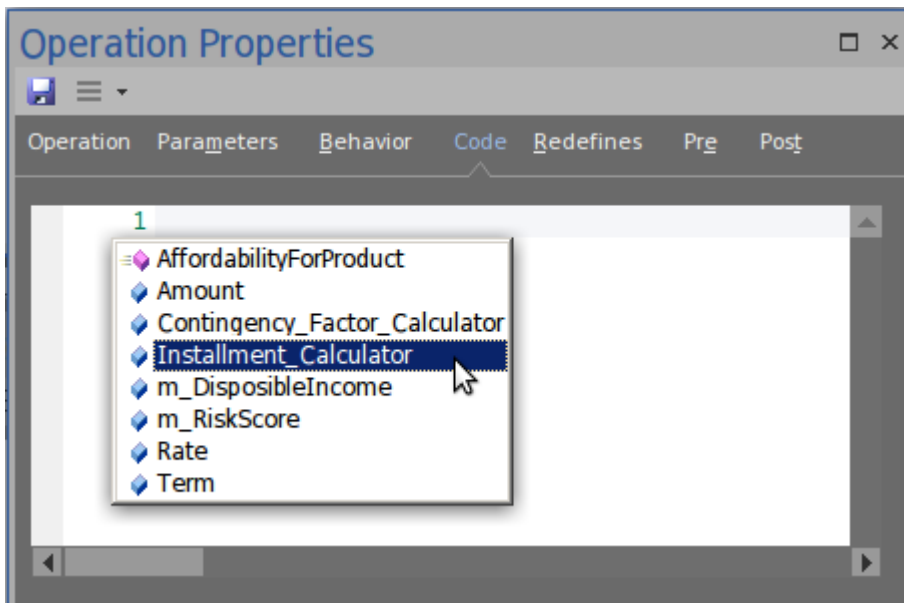
- Class attributes will be listed - m_RiskScore, m_DisposibleIncome
- Operation parameters will be listed - Amount, Rate, Term
- Operations will be listed - AffordabilityForProduct
- All bound DMN Modules will be listed - Contingency_Factor_Calculator, Installment_Calculator

It is quite easy to compose the code with Intelli-sense support. On accessing the DMN Module, all the Input Datas, Decisions and Business Knowledge Models will be listed for selection.
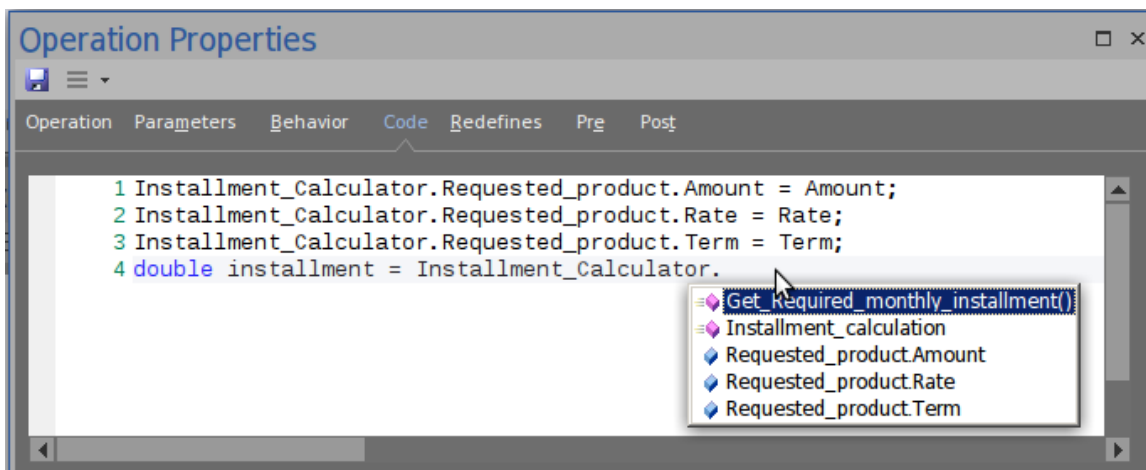
This illustration shows that we are selecting Get_Required_monthly_installment() from the Installment_Calculator.

This is the final implementation for the operation.
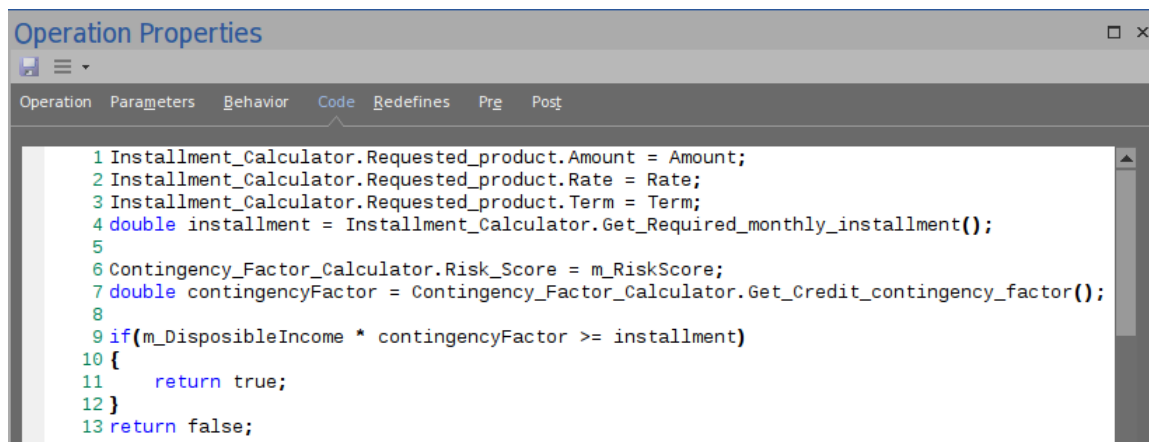
Code Generation for Class (With DMN Integration)

'Generate Code on Class Applicant' produces this code:

- The DMN Module(s) are generated as attributes of the Class
- The dmnBinding operation's code is updated

Note: Regardless of whether the generation option is 'Overwrite' or' Synchronize', the operation's code will be updated if it has the stereotype 'dmnBinding'.

Executable StateMachines
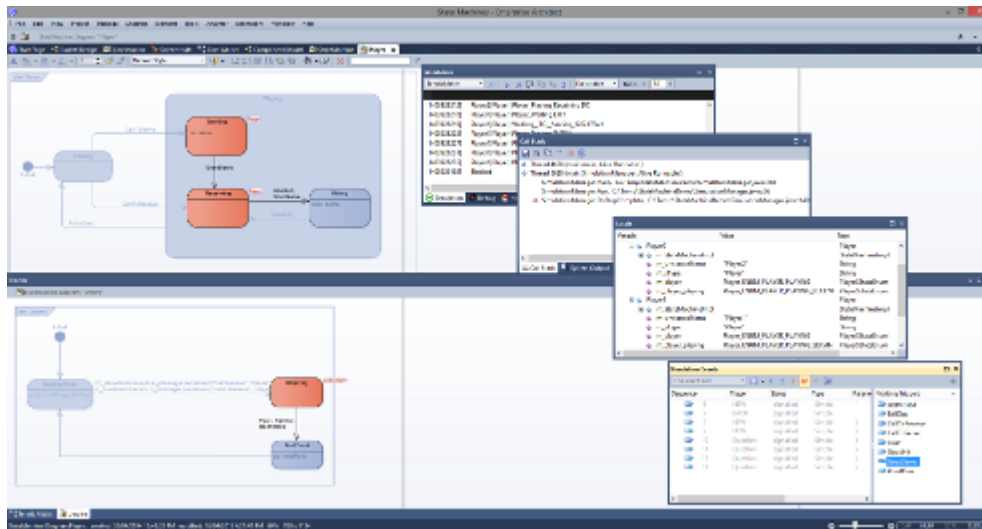
Executable StateMachines provide a powerful means of rapidly generating, executing and simulating complex state models. In contrast to dynamic simulation of State Charts using Enterprise Architect's Simulation engine, Executable StateMachines provide a complete language-specific implementation that can form the behavioral 'engine' for multiple software products on multiple platforms. Visualization of the execution uses and integrates seamlessly with the Simulation capability. Evolution of the model now presents fewer coding challenges. The code generation, compilation and execution is taken care of by Enterprise Architect. For those having particular requirements, each language is provided with a set of code templates. Templates can be customized by you to tailor the generated code in any ways you see fit.

These topics introduce you to the basics of modeling Executable StateMachines and tell you how to generate and simulate them.

The creation and use of Executable StateMachines, and generating code from them, are supported by the Unified and Ultimate editions of Enterprise Architect.

Overview of Building and Executing StateMachines

Building and using Executable StateMachines is quite straightforward, but does require a little planning and some knowledge of how to link the different components up to build an effective executing model. Luckily you do not have to spend hours getting the model right and fixing compilation errors before you can begin visualizing your design.

Having sketched out the broad mechanics of your model, you can generate the code to drive it, compile, execute and visualize it in a matter minutes. These points summarize what is required to start executing and simulating StateMachines.

Facility	Description
Build Class and State models	The first task is to build the standard UML Class and State models that describe the entities and behavior to construct. Each Class of interest in your model should have its own StateMachine that describes the various states and transitions that govern its overall behavior.
Create an Executable StateMachine Artifact	Once you have modeled your Classes and State models, its time to design the Executable StateMachine Artifact. This will describe the Classes and objects involved, and their initial properties and relationships. It is the binding script that links multiple objects together and it determines how these will communicate at runtime. Note that it is possible to have two or more objects in an Executable StateMachine Artifact as instances of a single Class. These will have their own state

	and behavior at run-time and can interact if necessary.
Generate Code and Compile	Whether it is JavaScript, C++, Java or C# that you use, Enterprise Architect's engineering capabilities provide you with a powerful tool, allowing you to regenerate the executable at any time, and without the loss of any customized code you might have made. This is a major advantage over a project's lifetime. It is probably also worth noting that the entire code base generated is independent and portable. In no way is the code coupled with any infrastructure used by the simulation engine.
Execute StateMachines	So how do we see how these StateMachines behave? One method is to build the code base for each platform, integrate it in one or more systems, examining the behaviors, 'in-situ', in perhaps several deployment scenarios. Or we can execute it with Enterprise Architect. Whether it is Java, JavaScript, C, C++ or C#, Enterprise Architect will take care of creating the runtime, the hosting of your model, the execution of its behaviors and the rendition of all StateMachines.
Visualize StateMachines	Executable StateMachine visualization integrates with Enterprise Architect's Simulation tools. Watch state transitions as they occur on your diagram and for which object(s). Easily identify objects sharing the same state. Importantly, these behaviors remain consistent across multiple platforms. You can also control the speed at which the machines operate to better understand the timeline of events.
Debug StateMachines	When states should change but do not, when a transition should not be enabled but is, when the behavior is - in short - undesirable and not immediately apparent from the model, we can turn to debugging. Enterprise Architect's Visual Execution Analyzer comes with debuggers for all the languages supported by ExecutableStateMachine code generation. Debugging provides many benefits, one of which might be to verify / corroborate the code attached to behaviors in a StateMachine to ensure it is actually reflected in the executing process.

Executable StateMachine Artifact

An Executable StateMachine Artifact is key to generating StateMachines that can interact with each other. It specifies the objects that will be involved in a simulation, their state and how they connect. A big advantage in using Executable StateMachine Artifacts is that each of several parts in an Artifact can represent an instance of a StateMachine, so you can set up simulations using multiple instances of each StateMachine and observe how they interact. An example is provided in the *Example Executable StateMachine* Help topic.

Creating the Properties of an Executable StateMachine

Each Executable StateMachine scenario involves one or more StateMachines. The StateMachines included are specified by UML Property elements; each Property will have a UML Classifier (Class) that determines the StateMachine(s) included for that type. Multiple types included as multiple Properties can end up including many StateMachines, which are all created in code and initialized on execution.

Action	Description
Drop a Class from the Project Browser on to the <<Executable StateMachine>> Artifact	The easiest way to define properties on an Executable StateMachine is to drop the Class onto the Executable StateMachine from the Project Browser. On the dialog that is shown, select the option to create a Property. You can then specify a name describing how the Executable StateMachine will refer to this property. Note: Depending on your options, you might have to hold down the Ctrl key to choose to create a property. This behavior can be changed at any time using the 'Hold Ctrl to Show this dialog' checkbox.
Use and Connect Multiple UML Properties	An Executable StateMachine describes the interaction of multiple StateMachines. These can be different instances of the same StateMachine, different StateMachines for the same instance, or completely different StateMachines from different base types. To create multiple properties that will use the same StateMachine, drop the same Class onto the Artifact multiple times. To use different types, drop different Classes from the Project Browser as required.
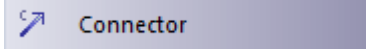
Defining the initial state for properties

The StateMachines run by an Executable StateMachine will all run in the context of their own Class instance. An Executable StateMachine allows you to define the initial state of each instance by assigning property values to various Class attributes. For example you might specify a Player's age, height, weight or similar if these properties have relevance to the scenario being run. By doing this it is possible to set up detailed initial conditions that will influence how the scenario plays out.

Action	Description
Set Property Values dialog	The dialog for assigning property values can be opened by right-clicking on a Property and selecting 'Features	Set Property Values', or by using the keyboard shortcut Ctrl+Shift+R.
Assign a value	The 'Set Property Values' dialog allows you to define values for any attribute defined in the original Class. To do this, select the variable, set the operator to '=' and enter the required value.

Defining relationships between properties

In addition to describing the values to assign to variables owned by each property, an Executable StateMachine allows you to define how each property can reference others based on the Class model that they are instances of.

Action	Description
Create a connector	Connect multiple properties using the Connector relationship from the Composite toolbox. ⤴ Connector Alternatively, use the Quick Linker to create a relationship between two Properties and select 'Connector' as the relationship type.
Map to Class model	Once a connector exists between two properties, you can map it back to the Association it represents in the Class model. To do this, select the connector and use the keyboard shortcut Ctrl+L. The 'Choose an Association' dialog displays, which allows the generated StateMachine to send signals to the instance filling the role specified in the relationship during execution.

Modeling Executable StateMachines

Most of the work required to model an Executable StateMachine is standard UML based modeling of Classes and State models. There are a couple of conventions that must be observed to ensure a well formed code base. The only novel construct is the use of a stereotyped Artifact element to form the configuration of an Executable StateMachine instance or scenario. The Artifact is used to specify details such as:

- The code language (JavaScript, C#, Java, C++ including C)

- The Classes and StateMachines involved in the scenario

- The instance specifications including run-state; note that this could include multiple instances of the same StateMachine, for example where a 'Player' Class is used twice in a Tennis Match simulation

Basic Modeling Tools and Objects for Executable StateMachines

These are the primary modeling elements used when building Executable StateMachines.

Object	Details
Classes and Class Diagrams	Classes define the object types that are relevant to the StateMachine(s) being modeled. For example, in a simple Tennis Match scenario you might define a Class for each of a Player, a Match, a Hit and an Umpire. Each will have its own StateMachine(s) and at runtime will be represented by object instances for each involved entity. See the *UML modeling guide* for more information on Classes and Class diagrams.
StateMachines	For each Class you define that will have dynamic behavior within a scenario, you will typically define one or more UML StateMachines. Each StateMachine will determine the legal state-based behavior appropriate for one aspect of the owning Class. For example, it is possible to have a StateMachine that represents a Player's emotional state, one that tracks his current fitness and energy levels, and one that represents his winning or losing state. All these StateMachines will be initialized and started when the StateMachine scenario begins execution.
Executable StateMachine Artifact	This stereotyped Artifact is the core element used to specify the participants, configuration and starting conditions for an Executable StateMachine. From the scenario point of view it is used to determine which Instances (of Classes) are involved, what events they might Trigger and send to each other, and what starting conditions they operate under. From the configuration aspect, the Artifact is used to set up the link to an analyzer script that will determine output directory, code language, compilation script and similar. Right-clicking on the Artifact will allow you to generate, build, compile and visualize the real time execution of your StateMachines.

StateMachine Constructs Supported

This table details the StateMachine constructs supported and any limitations or general constraints relevant to each type.

Construct	Description
StateMachines	• Simple StateMachine: StateMachine has one region • Orthogonal StateMachine: StateMachine contains multiple regions

	Top level region (owned by StateMachine) activation semantics: **Default Activation:** When the StateMachine starts executing. **Entry Point Entry:** Transitions from Entry Point to vertices in the contained regions. • *Note 1: In each Region of the StateMachine owning the Entry Point, there is at most a single Transition from the entry point to a Vertex within that Region* • *Note 2: This StateMachine can be referenced by a Submachine State - connection point reference should be defined in the Submachine State as sources/targets of transitions; the Connection point reference represents a usage of an Entry/Exit Point defined in the StateMachine and referenced by the Submachine State* **Not Supported** • Protocol StateMachine • StateMachine Redefinition
States	• Simple State: has no internal Vertices or Transitions • Composite State: contains exactly one Region • Orthogonal State: contains multiple Regions • Submachine State: refers to an entire StateMachine
Composite State Entry	• Default Entry • Explicit Entry • Shallow History Entry • Deep History Entry • Entry Point Entry
Substates	• Substates and Nested Substates Entry and Exit semantics, where the transition includes multiple nested levels of states, will obey correct execution of nested behaviors (such as OnEntry and OnExit).
Transitions support	• External Transition • Local Transition • Internal Transition (draw a self Transition and change Transition kind to Internal) • Completion Transition and Completion Events • Transition Guards • Compound Transitions • Firing priorities and selection algorithm For further details, refer to the *UML Specification*.
Trigger and Events	An Executable StateMachine supports event handling for Signals only. To use Call, Timing or Change Event types you must define an outside mechanism to generate signals based on these events.

Signal	Attributes can be defined in Signals; the value of the attributes can be used as event arguments in Transition Guards and Effects.
	
	For example, this is the code set in the effect of a transition in C++:
	
	   if(signal->signalEnum == ENUM_SIGNAL2)
	   {
	      int xVal = ((Signal2*)signal)->myVal;
	   }
	
	Signal2 is generated as this code:
	
	   class Signal2 : public Signal{
	   public:
	      Signal2(){};
	      Signal2(std::vector<String>& lstArguments);
	      int myVal;
	   };
	
	Note: Further details can be found by generating an Executable StateMachine and referring to the generated 'EventProxy' file.

Initial	An Initial Pseudostate represents a starting point for a Region. It is the source for at most one Transition; there can be at most one Initial Vertex in a Region.
Regions	**Default Activation & Explicit Activation:** Transitions terminate on the containing State: • If an initial Pseudostate is defined in the Region: **Default activation** • If no initial Pseudostate is defined, the Region will remain inactive and the containing State is treated as a Simple State • Transition terminates on one of the Region's contained vertices: **Explicit activation**, resulting in the default activation of all of its orthogonal Regions, unless those Regions are also entered explicitly (multiple orthogonal Regions can be entered explicitly in parallel through Transitions originating from the same Fork Pseudostate) For example, if there are three Regions defined for a Orthogonal State, and *RegionA* and *RegionB* have an Initial Pseudostate, then *RegionC* is explicitly activated. Default Activation applies to *RegionA* and *RegionB*; the containing State will have three active Regions.
Choice	Guard Constraints on all outgoing Transitions are evaluated dynamically, when the compound transition traversal reaches this Pseudostate.
Junction	Static conditional branch: guard constraints are evaluated before any compound transition is executed.
Fork / Join	Non-threaded, each active Region moves one step alternately, based on a completion event pool mechanism.
EntryPoint / ExitPoint Nodes	Non-threaded for orthogonal State or orthogonal StateMachine; each active Region moves one step alternately, based on a completion event pool mechanism.
History Nodes	• DeepHistory: represents the most recent active State configuration of its owning State • ShallowHistory: represents the most recent active Substate of its containing State, but not the Substates of that Substate

Deferred Events	Draw a self Transition, and change the Transition *kind* to Internal. Type 'defer();' in the 'Effect' field for the transition.
Connection Point References	A Connection Point Reference represents a usage (as part of a Submachine State) of an Entry/Exit Point defined in the StateMachine referenced by the Submachine State. Connection Point References of a Submachine State can be used as sources and targets of Transitions. They represent entries into or exits out of the StateMachine referenced by the Submachine State.
State behaviors	State 'entry', 'doActivity' and 'exit' behaviors are defined as operations on a State. By default, you type the code that will be used for each behavior into the 'Code' panel of the Properties window for the Behavior operation. Note that you can change this to type the code into the 'Behavior' panel, by customizing the generation template.
	The 'doActivity' behavior generated will be run to completion before proceeding. The code is not concurrent with other entry behavior; the 'doActivity' behavior is implemented as 'execute in sequence after entry' behavior.

References to Behaviors within other Contexts/Classes

If the Submachine State references behavior elements outside the current context or Class, you must add an <<import>> connector from the current context Class to the container context Class. For example:

Submachine State S1 in Class1 refers to StateMachine ST2 in Class2

Therefore, we add an <<import>> connector from Class1 to Class2 in order for Executable StateMachine code generation to generate code correctly for Submachine State S1. (On Class 1, click on the Quick Linker arrow and drag to Class 2, then select 'Import' from the menu of connector types.)

Reusing Executable StateMachine Artifacts

You can create multiple models or versions of a component using a single executable Artifact. An Artifact representing a resistor, for example, could be re-used to create both a foil resistor and a wire wound resistor. This is likely to be the case for similar objects that, although represented by the same classifier, typically exhibit different run states. A property named 'resistorType' taking the value 'wire' rather than 'foil' might be all that is required from a modeling point of view. The same StateMachines can then be re-used to test behavioral changes that might result due to variance in run-state. This is the procedure:

Step	Action
Create or open Component diagram	Open a Component diagram to work on. This might be the diagram that contains your original Artifact.
Select the Executable StateMachine to copy	Now find the original Executable StateMachine Artifact in the Project Browser.
Create the New Component	Whilst holding the Ctrl key, drag the original Artifact on to your diagram. You will be prompted with two questions.
	The answer to the first is **Object** and to the second **All.** Rename the Artifact to differentiate it from the original and then proceed to alter its property values.

Code Generation for Executable StateMachines

The code generated for an Executable StateMachine is based on its language property. This might be Java, C, C++, C# or JavaScript. Whichever language it is, Enterprise Architect generates the appropriate code, which is immediately ready to build and run. There are no manual interventions necessary before you run it. In fact after the initial generation, any Executable StateMachine can be generated, built and executed at the click of a button.

Language Supported

An Executable StateMachine supports code generation for these platform languages:

- Microsoft Native C/C++
- Microsoft .NET (C#)
- Scripting (JavaScript)
- Oracle Java (Java)

From Enterprise Architect Release 14.1, code generation is supported without dependency on the simulation environment (compilers). For example, if you don't have Visual Studio installed, you can still generate code from the model and use it in your own project. The compilers are still needed if you want to simulate models in Enterprise Architect.

Simulation Environment (Compiler Settings)

If you want to simulate the Executable StateMachine model in Enterprise Architect, these platforms or compilers are required for the languages:

Language Platform	Example of Framework Path
Microsoft Native (C/C++)	C:\Program Files (x86)\Microsoft Visual Studio 12.0 C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional (or other editions)
Microsoft .NET (C#)	C:\Windows\Microsoft.NET\Framework\v3.5 (or higher)
Scripting (JavaScript)	N/A
Oracle Java (Java)	C:\Program Files (x86)\Java\jdk1.7.0_17 (or higher)

Access

Ribbon	Simulate > Executable > Statemachines > Generate, Build and Run or
	Simulate > Executable > Statemachines > Generate

Generating Code

The 'Simulate > Executable > Statemachines' ribbon options provide commands for generating code for the StateMachine. Select the Executable Statemachine Artifact first, then use the ribbon option to generate the code. The 'Executable Statemachine Code Generation' dialog displayed depends on the code language.

Generating Code (Java on WIndows)

Project output directory	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory. The names of the generated classes and their source file paths are displayed after this.
Executable Statemachine Target Machine	Select the 'Local' option.

| Java JDK | Enter the installation directory of the Java JDK to be used. |

Generating Code (Java on Linux)

Project output directory:	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory. The names of the generated classes and their source file paths are displayed when the path is changed
Executable Statemachine Target Machine	Select the 'Remote' option.
Operating System	Select Linux.

Port	This is the debugger port to be used. You will find references to this port number in the 'Debug' and 'DebugRun' sections of the Analyzer Script generated.

Generating Code (Other Languages)

At the same time the System Output window opens at the 'Executable StateMachine Output' page, on which progress messages, warnings or errors are displayed during code generation.

On the 'Executable StateMachine Code Generation' dialog, the 'Artifact' field and 'Language' field display the element name and coding language as defined in the element's 'Properties' dialog.

Field/Option	Description
Project output directory	Displays the directory in which the generated code files will be stored. If necessary, click on the button at the right of the field to browse for and select a different directory.
Project build environment	The fields and information in this panel vary depending on the language defined in the Artifact element and in the script. However, each supported language provides an option to define the path to the target frameworks that are required to build and run the generated code; examples are shown in the *Languages Supported* section of this topic. This path, and its Local Paths ID, are defined in the 'Local Paths' dialog and shown here on the 'Executable StateMachine Code Generation' dialog.

Generate

Click on this button to generate the StateMachine code. The code generation will overwrite any existing files in the project output directory. The set of files will include all required files including those for each Class referenced by the StateMachine.

⊗ ConsoleManager.cs	12/06/2015 10:56 …	C# Language File	2 KB
⊗ ContextManager.cs	12/06/2015 10:56 …	C# Language File	24 KB
⊗ EventProxy.cs	12/06/2015 10:56 …	C# Language File	2 KB
⊗ SimulationManager.cs	12/06/2015 10:56 …	C# Language File	4 KB

Each Executable StateMachine that is generated will also generate an Execution Analyzer script, which is the configuration script for building, running and debugging the Executable StateMachine.
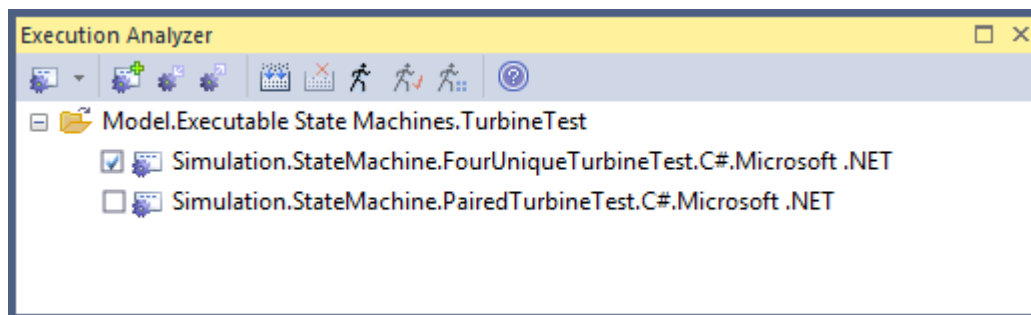
Building Code

The code generated by an Executable StateMachine can be built by Enterprise Architect in one of three ways.

Method	Description
Execution Analyzer Script	The generated Execution Analyzer script includes a command to build the source code. This means that when it is active, you can build directly using the Execution Analyzer script, including using the built-in shortcut Ctrl+Shift+F12.
Ribbon Build Command	The 'Simulate > Executable > Statemachines' drop-down menu provides a 'Build' command, which you can use to compile the code that already exists in the target directory. This can be used directly after generating the code or if you have modified the generated code in any way.
Ribbon Generate, Build and Run Command	The 'Simulate > Executable > Statemachines' drop-down menu provides a shortcut to allow changes made to a StateMachine to be quickly tested. This command updates the code before building it, and then starts the simulation.
Build Output	When building, all output is shown on the 'Build' page of the System Output window. You can double-click on any compiler errors to open a source editor at the appropriate line.

Leveraging existing code

Executable StateMachines generated and executed by Enterprise Architect can leverage existing code for which no Class model exists. To do this you would create an abstract Class element naming only the operations to call in the external

codebase. You would then create a generalization between this interface and the StateMachine Class, adding the required linkages manually in the Analyzer Script. For Java you might add .jar files to the Class path. For native code you might add a .dll to the linkage.

Debugging Execution of Executable StateMachines

Creation of Executable StateMachines provides benefits even after the generation of code. Using the Execution Analyzer, Enterprise Architect is able to connect to the generated code. As a result you are able to visually debug and verify the correct behavior of the code; the exact same code generated from your StateMachines, demonstrated by the simulation and ultimately incorporated in a real world system.

Debugging a StateMachine

Being able to debug an Executable StateMachine gives additional benefits, such as being able to:

- Interrupt the execution of the simulation and all executing StateMachines

- View the raw state of each StateMachine instance involved in the simulation

- View the source code and Call Stack at any point in time

- Trace additional information about the execution state through the placement of tracepoints on lines of source code

- Control the execution through use of actionpoints and breakpoints (break on error, for example)

- Diagnose changes in behavior, due to either code or modeling changes

If you have generated, built and run an Executable StateMachine successfully, you can debug it! The Analyzer Script created during the generation process is already configured to provide debugging. To start debugging, simply start running the Executable StateMachine using the Simulation Control. Depending on the nature of the behavior being debugged, however, we would probably set some breakpoints first.

Breaking execution at a state transition

Like any debugger we can use breakpoints to examine the executing StateMachine at a point in code. Locate a Class of interest in either the diagram or Project Browser and press F12 to view the source code. It is easy to locate the code for State transitions from the naming conventions used during generation. If you want to break at a particular transition, locate the transition function in the editor and place a breakpoint marker by clicking in the left margin at a line within the function. When you run the Executable StateMachine, the debugger will halt at this transition and you will be able to view the raw state of variables for any StateMachines involved.

Breaking execution conditionally

Each breakpoint can take a condition and a trace statement. When the breakpoint is encountered and the condition evaluates to True the execution will halt. Otherwise the execution will continue as normal. You compose the condition using the names of the raw variables and comparing them using the standard equality operands: < > = >= <=. For example:

   (this.m_nCount > 100) and (this.m_ntype == 1)

To add a condition to a breakpoint you have set, right-click on the breakpoint and select 'Properties'. By clicking on the breakpoint while pressing the Ctrl key, the properties can be quickly edited.

Tracing auxillary information

It is possible to trace information from within the StateMachine itself using the TRACE clause in, for example, an *effect*. Debugging also provides trace features known as Tracepoints. These are simply breakpoints that, instead of breaking, print trace statements when they are encountered. The output is displayed in the Simulation Control window. They can be used as a diagnostic aid to show and prove the sequence of events and the order in which instances change state.

Viewing the Call Stack

Whenever a breakpoint is encountered, the Call Stack is available from the Analyzer menu. Use this to determine the sequence in which the execution is taking place.

Execution and Simulation of Executable StateMachines

One of the many features of Enterprise Architect is its ability to perform simulations. An Executable StateMachine generated and built in Enterprise Architect can hook into the Simulation facilities to visually demonstrate the live execution of the StateMachine Artifact.

Starting a simulation

The Simulation Control toolbar provides a Search button that you use to select the Executable StateMachine Artifact to run. The control maintains a drop-down list of the most recent Executable StateMachines for you to choose from. You can also use the context menu on an Executable StateMachine Artifact itself to initiate the simulation.

Controlling speed

The Simulation Control provides a speed setting. You can use this to adjust the rate at which the simulation executes. The speed is represented as a value between 0 and 100 (a higher value is faster). A value of zero will cause the simulation to halt after every step; this requires using the toolbar controls to manually step through the simulation.

Notation for active states

As the Executable StateMachine executes, the relevant StateMachine diagrams are displayed. The display is updated at the end of every step-to-completion cycle. You will notice that only the active State for the instance completing a step is highlighted. The other States remain dimmed.

It is easy to identify which instance is in which State, as the States are labeled with the name of any instance currently in that particular state. If two or more artifact properties of the same type share the same state, the State will have a separate label for each property name.

Generate Timing Diagram

After completing the simulation of an Executable Statemachine, you can generate a Timing diagram from the output. To do this:

In the Simulation window toolbar, click on 'Tools | Generate Timing Diagram'.

Example: Simulation Commands
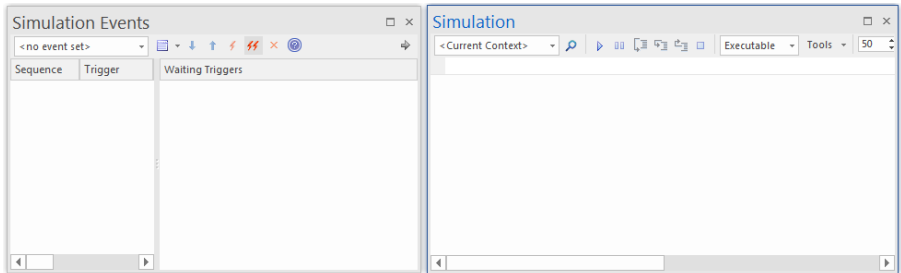
This example demonstrates how we can use the Simulation window to observe Trace messages or send commands to control a StateMachine. Through the example, you can examine:

- An attribute of a context - the member variable defined in the Class, which is the context of the StateMachine; these attributes carry values in the scope of the context for all State behaviors and transition effects, to access and modify

- Each attribute of a Signal - the member variable defined in the Signal, which is referenced by an Event and which can serve as an Event Parameter; each Signal Event occurrence might have different instances of a Signal

- The use of the 'Eval' command to query the runtime value of a context's attribute

- The use of the 'Dump' command to dump the current state's active count; it can also dump the current event deferred in the pool

This example is taken from the EAExample model:

Example Model.Model Simulation.Executable StateMachine.Simulation Commands

Access

Ribbon	• Simulate > Core > Events (for the Simulation Events window)
	• Simulate > Dynamic Simulation > Open Simulation Window)
	These two windows are frequently used together in the simulation of Executable StateMachines.

Create Context and StateMachine

In this section we will create a Class called TransactionServer, which defines a StateMachine as its behavior. We then create an Executable StateMachine Artifact as the simulation environment.

Create the Context of the StateMachine
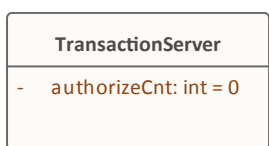
1. Create a Class element called *TransactionServer*.

2. In this Class, create an attribute called *authorizeCnt* with initial value 0.

3.   In the Project Browser, right-click on *TransactionServer* and select the 'Add | State Machine' option.

Create the StateMachine
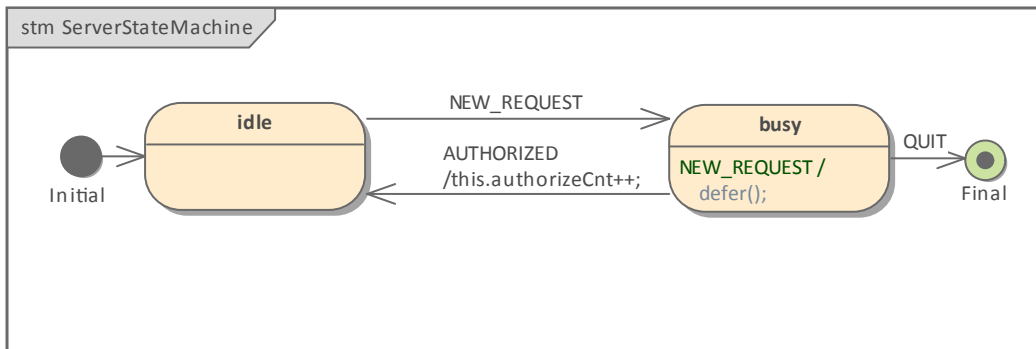
1.   Create an Initial pseudostate called *Initial*.

2.   Transition to a State called *idle*.

3.   Transition to a State called *busy*, with the trigger NEW_REQUEST.

4.   Transition:
      - To a Final pseudostate called *Final*, with the trigger QUIT
      - Back to *idle*, with the trigger AUTHORIZED, with the effect "this.authorizeCnt++;"

Create a Deferred Event for the State *busy*

1.   Draw a self-transition for *busy*.

2.   Change the 'kind' of the transition to 'internal'.

3.   Specify the Trigger to be the event you want to defer.

4.   In the 'Effect' field, type 'defer();'.

Create a Signal and attributes

1.   Create a Signal element called *RequestSignal.*

2.   Create an attribute called *requestType* with type 'int'.

3.   Configure the Event NEW_REQUEST to reference *RequestSignal.*

Create the Executable StateMachine Artifact
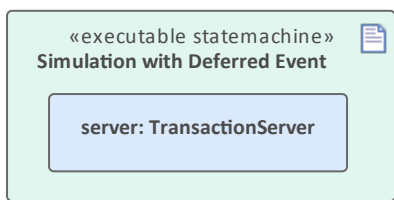
1.   From the 'Artifacts' page of the Diagram Toolbox, drag an Executable StateMachine icon onto the diagram and call the element *Simulation with Deferred Event.*

2.   Ctrl+Drag the*TransactionServer* element from the Project Browser and drop it onto the Artifact as a property, with the name *server.*

3.   Set the language of the Artifact to JavaScript, which does not require a compiler (for the example; in production you could also use C, C++, C#, or Java, which also support Executable StateMachines).

4. Click on the Artifact and select the 'Simulate > Executable > Statemachines > Generate, Build and Run' ribbon option.

Simulation window and Commands

When the simulation starts, *idle* is the current state.

The Simulation window shows that the Transition Effect, Entry and Do behavior is finished for state *idle*, and the StateMachine is waiting for a trigger.
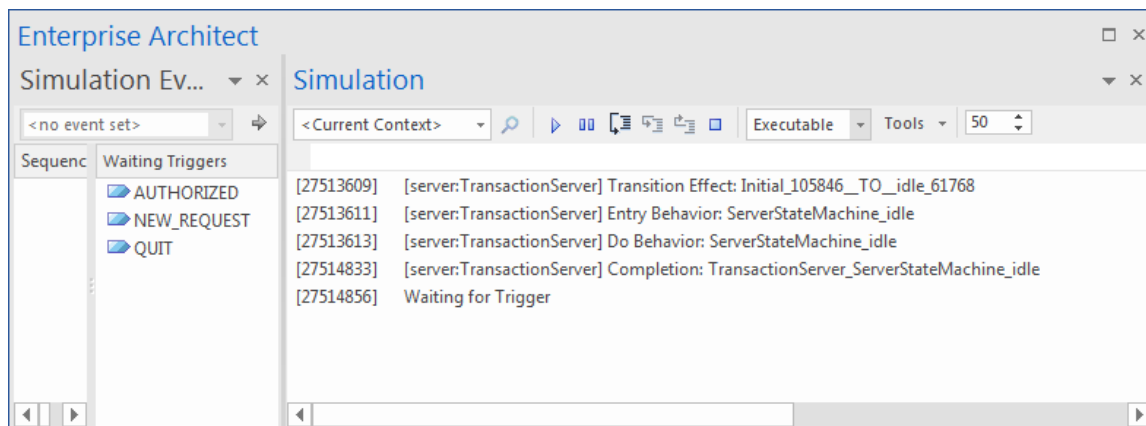
Event Data via Values for Signal attributes

For the Trigger Signal Event NEW_REQUEST, the 'Trigger Parameter Entry' dialog displays to prompt for values for the listed attributes defined in the Signal *RequestSignal*, referenced by NEW_REQUEST.
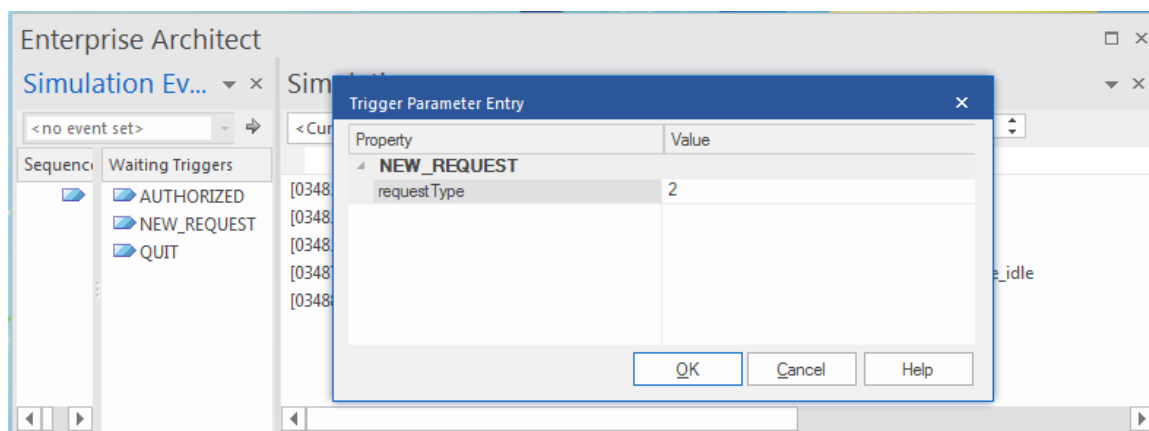
Type the value '2' and click on the OK button. The Signal attribute values are then passed to invoked methods such as the

State's behaviors and the Transition's effects.

These messages are output to the Simulation window:

[03612562]   Waiting for Trigger

[03611358]   Command: **broadcast NEW_REQUEST.RequestSignal(2)**

[03611362]   [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:2)

[03611367]   [server:TransactionServer] Event Dispatched: NEW_REQUEST.RequestSignal(requestType:2)

[03611371]   [server:TransactionServer] Exit Behavior: ServerStateMachine_idle

[03611381]   [server:TransactionServer] Transition Effect: idle__TO__busy_61772

[03611390]   [server:TransactionServer] Entry Behavior: ServerStateMachine_busy

[03611398]   [server:TransactionServer] Do Behavior: ServerStateMachine_busy

[03612544]   [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_busy

[03612562]   Waiting for Trigger

We can broadcast events by double-clicking on the item listed in the Simulation Events window. Alternatively, we can type a command string in the text field of the Simulation window (underneath the toolbar).
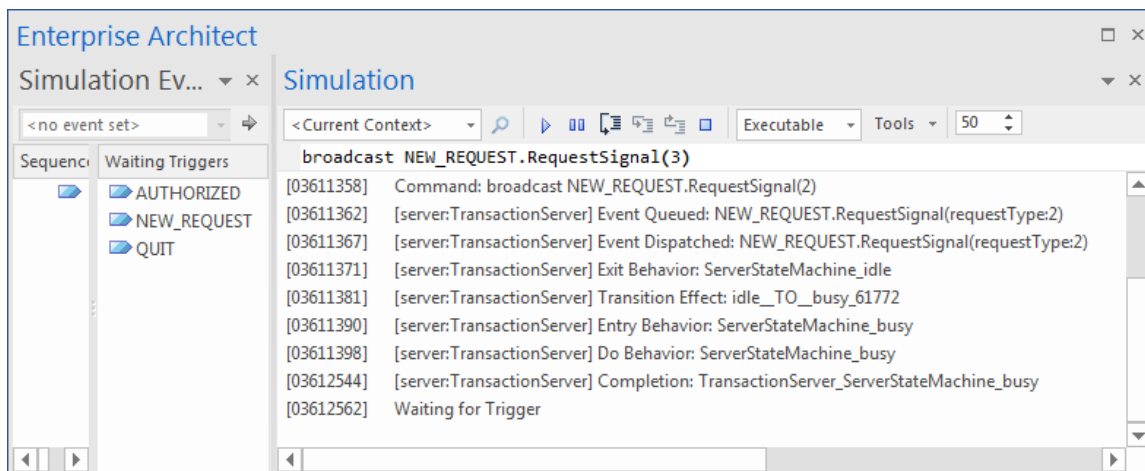
[03612562]   Waiting for Trigger

[04460226]   Command: **broadcast NEW_REQUEST.RequestSignal(3)**

[04460233]   [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:3)

[04461081]   Waiting for Trigger

The Simulation message indicates that the event occurrence is deferred (Event Queued, but not dispatched). We can run further commands using the text field:

[04655441]   Waiting for Trigger

[04664057]   Command: **broadcast NEW_REQUEST.RequestSignal(6)**

[04664066]   [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:6)

[04664803]   Waiting for Trigger

[04669659]   Command: **broadcast NEW_REQUEST.RequestSignal(5)**

[04669667]   [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:5)

[04670312]   Waiting for Trigger

[04674196]   Command: **broadcast NEW_REQUEST.RequestSignal(8)**

[04674204]   [server:TransactionServer] Event Queued: NEW_REQUEST.RequestSignal(requestType:8)

[04674838]   Waiting for Trigger

dump: Query 'active count' for a State and Event pool

Type *dump* in the text field; these results display:
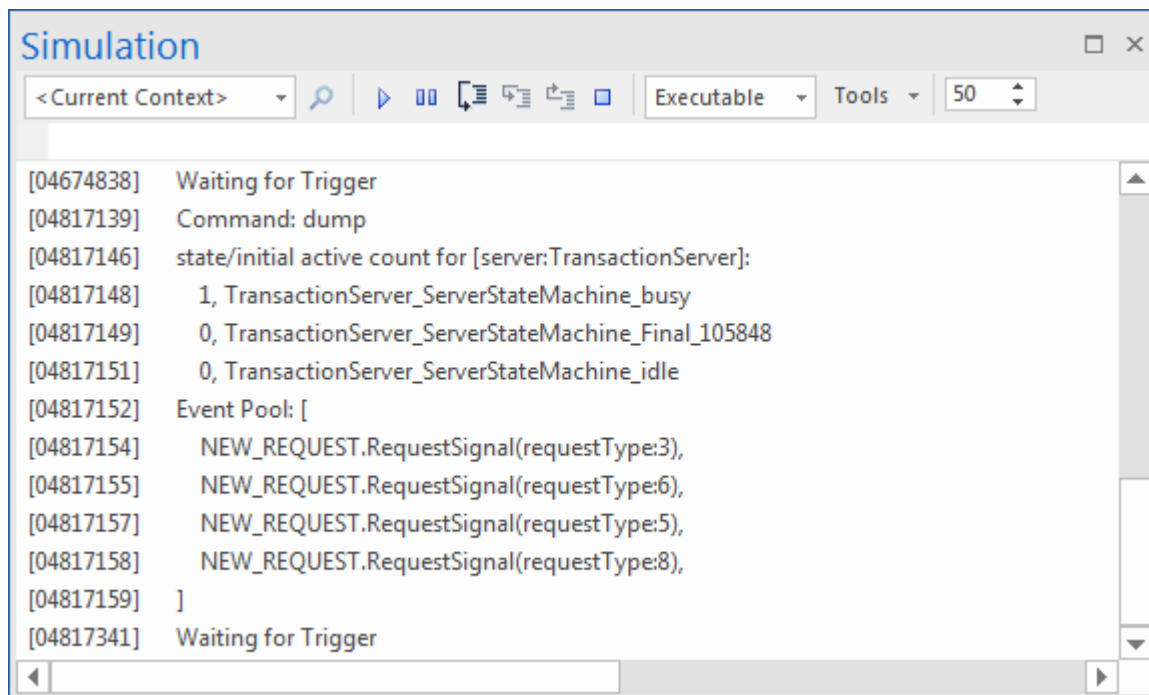
From the 'active count' section, we can see that *busy* is the active state (active count is 1).

Tips: For a Composite State, the active count is 1 (for itself) *plus* the number of active regions.

From the 'Event Pool' section, we can see that there are four event occurrences in the Event Queue. Each instance of the signal carries different data.

The order of the events in the pool is the order in which they are broadcast.

eval: Query Run Time Value of the Context

Trigger AUTHORIZED,

[04817341]   Waiting for Trigger

[05494672]   Command: broadcast AUTHORIZED

[05494678]   [server:TransactionServer] Event Queued: AUTHORIZED

[05494680]   [server:TransactionServer] Event Dispatched: AUTHORIZED

[05494686]   [server:TransactionServer] Exit Behavior: ServerStateMachine_busy

[05494686]   [server:TransactionServer] **Transition Effect: busy__TO__idle_61769**

[05494687]   [server:TransactionServer] Entry Behavior: ServerStateMachine_idle

[05494688]   [server:TransactionServer] Do Behavior: ServerStateMachine_idle

[05495835]   [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_idle

[05495842]   [server:TransactionServer] **Event Dispatched: NEW_REQUEST.RequestSignal(requestType:3)**

[05495844]   [server:TransactionServer] Exit Behavior: ServerStateMachine_idle

[05495846]   [server:TransactionServer] Transition Effect: idle__TO__busy_61772

[05495847]   [server:TransactionServer] Entry Behavior: ServerStateMachine_busy

[05495850]   [server:TransactionServer] Do Behavior: ServerStateMachine_busy

[05496349]   [server:TransactionServer] Completion: TransactionServer_ServerStateMachine_busy

[05496367]   Waiting for Trigger

- The transition from *busy* to *idle* is made, so we expect the effect to be executed

- One event is recalled from the pool and dispatched when *idle* is completed, causing *busy* to become the active state

- Type *dump* and notice that there are three events left in the pool; the first one is recalled and dispatched

[05693348]   Event Pool: [

[05693349]   NEW_REQUEST.RequestSignal(requestType:6),

[05693351]   NEW_REQUEST.RequestSignal(requestType:5),

[05693352]   NEW_REQUEST.RequestSignal(requestType:8),

[05693354]   ]

Type *eval server.authorizeCnt* in the text field. This figure indicates that the run time value of 'server.authorizeCnt' is 1.
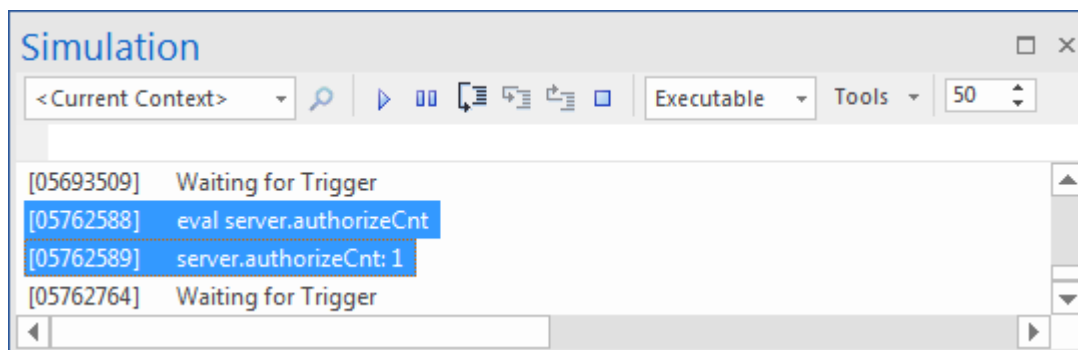
Trigger AUTHORIZED again. When the StateMachine is stable at *busy*, there will be two events left in the pool. Run *eval server.suthorizeCnt* again; the value will be 2.

Access Context's member variable from State behavior and Transition Effect

Enterprise Architect's Executable StateMachine supports simulation for C, C++, C#, Java and JavaScript.

For C and C++, the syntax differs from C#, Java and JavaScript in accessing the context's member variables. C and C++ use the pointer '->' while the others simply use '.'; however, you can always use *this.variableName* to access the variables. Enterprise Architect will translate it to this->variableName for C and C++.

So for all languages, simply use this format for the simulation:

   this.variableName

Examples:

In the transition's effect:

   this.authorizeCnt++;

In some state's entry, do or exit behavior:

   this.foo += this.bar;

Note: by default Enterprise Architect is only replacing 'this->' with 'this' for C and C++; For example:

   this.foo = this.bar + myObject.iCount + myPointer->iCount;

will be translated to:

   this->foo = this->bar + myObject.iCount + myPointer->iCount;

A complete list of commands supported

Since the Executable StateMachine Artifact can simulate multiple contexts together, some of the commands can specify an instance name.

run statemachine:

As each context can have multiple StateMachines, the 'run' command can specify a StateMachine to start with.

- run instance.statemachine
- run all.all
- run instance
- run all
- run

For example:

   run

   run all

   run server

   run server.myMainStatemachine

broadcast & send event:

- broadcast EventString
- send EventString to instance
- send EventString (equivalent to broadcast EventString)

For example:

   broadcast Event1

   send Event1 to client

dump command:

- dump
- dump instance

For example:

   dump

   dump server

   dump client

eval command:

- eval instance.variableName

For example:

    eval client.requestCnt

    eval server.responseCnt

exit command:

- exit

The EventString's format:

- EventName.SignalName(argument list)

Note: the argument list should match the attributes defined in the signal **by order**.

For example, if the Signal defines two attributes:

- foo
- bar

Then these EventStrings are valid:

- Event1.Signal1(10, 5)   --------- foo = 10; bar = 5
- Event1.Signal1(10,)   --------- foo = 10; bar is undefined
- Event1.Signal1(,5)   --------- bar = 10; foo is undefined
- Event1.Signal1(,)   --------- both foo and bar are not defined

If the Signal does not contain any attributes, we can simplify the EventString to:

- EventName

Example: Simulation in HTML with JavaScript

We already know that users can model an Executable StateMachine and simulate it in Enterprise Architect with the generated code. In the *CD Player* and the *Regular Expression Parser* examples, we will further demonstrate how you can integrate the generated code with your real projects.

Enterprise Architect provides two different mechanisms for client code to use a StateMachine:

- Active State Based - the client can query the current active state, then 'switch' the logic based on the query result
- Runtime Variable Based - the client does not act on the current active state, but does act on the runtime value of the variables defined in the Class containing the StateMachine

In the *CD Player* example, there are very few states and many buttons on the GUI, so it is quite easy to implement the example based on the Active State Mechanism; we will also query the runtime value for the current track.

In the *Regular Expression Parser* example the StateMachine handles everything, and a member variable *bMatch* changes its runtime value when states change. The client does not register how many states are there or which state is currently active.

In these topics, we demonstrate how to model, simulate and integrate a CD Player and a Parser for a specified Regular Expression, step by step:

- CD Player
- Regular Expression Parser

CD Player

The behavior of a CD Player application might appear intuitive; however, there are many rules related to when the buttons are enabled and disabled, what is displayed in the text fields of the window and what happens when the user supplies events to the application.

Suppose our example CD Player has these features:

- Buttons - Load Random CD, Play, Pause, Stop, Previous Track, Next Track and Eject
- Displays - Number Of Tracks, Current Track, Track Length and Time Elapsed

StateMachine for CD Player

A Class *CDPlayer* is defined with two attributes: *currentTrack* and *numberOfTracks*.
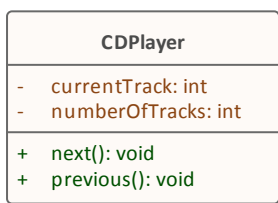
A StateMachine is used to describe the states of the CD Player:
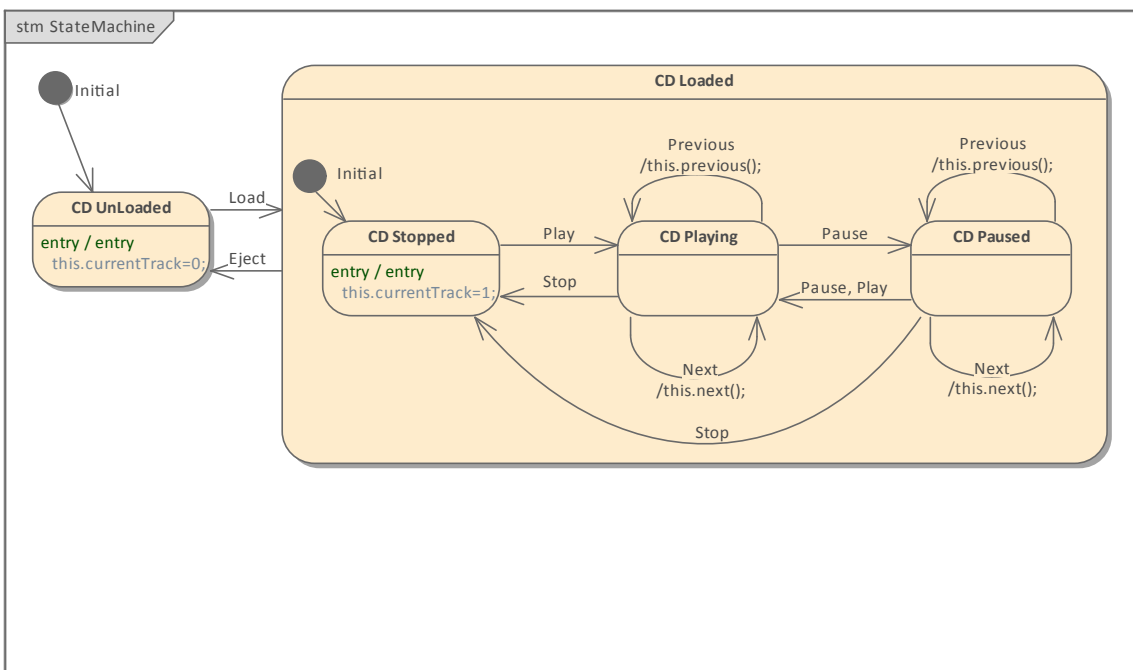
- On the higher level, the StateMachine has two States: *CD UnLoaded* and *CD Loaded*
- *CD Loaded* can be composed of three simple States: *CD Stopped*, *CD Playing*, *CD Paused*
- Transitions are defined with triggers for the events Load, Eject, Play, Pause, Stop, Previous and Next
- State behaviors and transition effects are defined to change the value of attributes defined in *CDPlayer*; for example, the 'Previous' event will trigger the self transition (if the current state is *CD Playing* or *CD Paused*) and the effect will be executed, which will decrement the value of *currentTrack* or wrap to the last track

We can create an Executable StateMachine Artifact and create a property typing to *CDPlayer*, then simulate the

StateMachine in Enterprise Architect to make sure the model is correct.
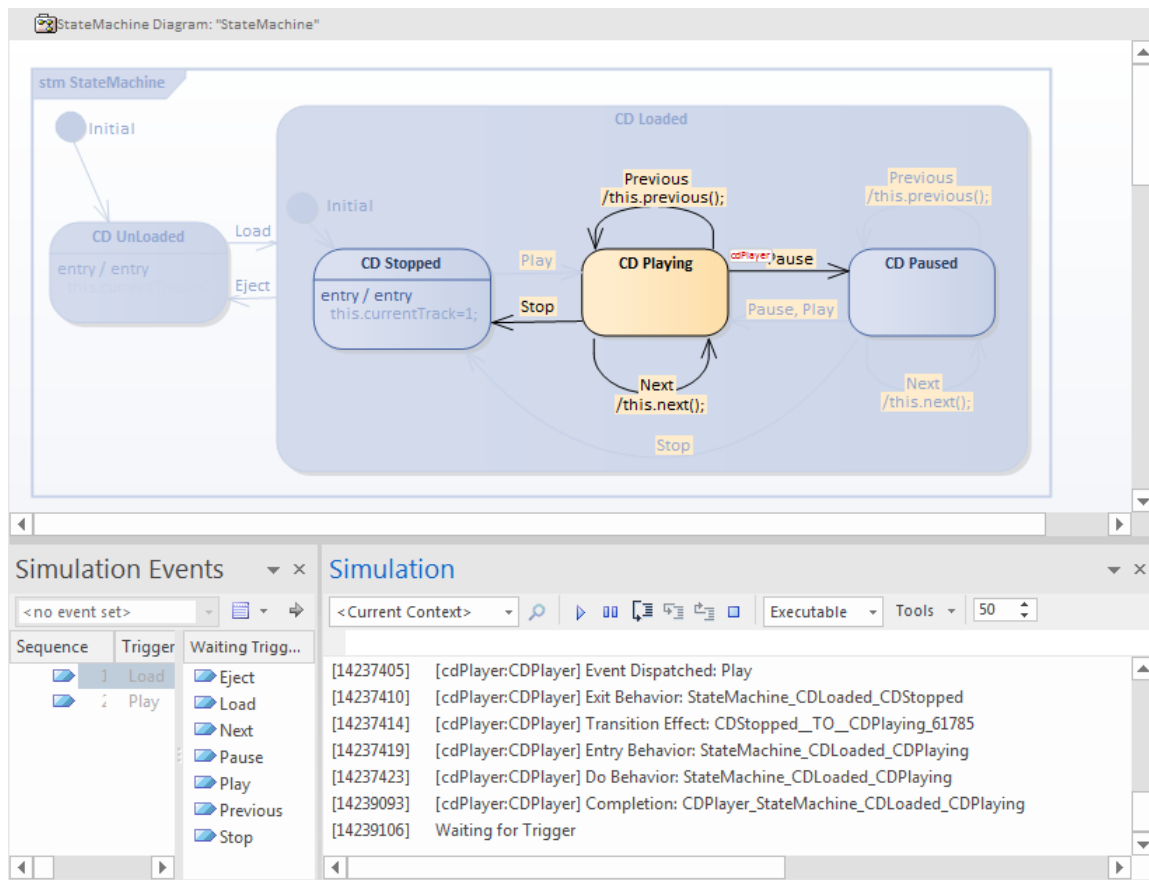
Inspect the code generated

Enterprise Architect will generate these files in a folder that the user has specified:

- Back-end code: CDPlayer.js, ContextManager.js, EventProxy.js
- Client code: ManagerWorker
- Front-end code: statemachineGUI.js, index.html
- Other code: SimulationManager.js

File	Description
/CDPlayer.js	This file defines the Class CDPlayer and its attributes and operations. It also defines the Class's StateMachines with the State behaviors and the transition effects.
/ContextManager.js	This file is the abstract manager of contexts. The file defines the contents that are independent of the actual contexts, which are defined in the generalization of the *ContextManager*, such as *SimulationManager* and *ManagerWorker*. The simulation (Executable StateMachine Artifact) can involve multiple contexts; for example, in a tennis game simulation there will be one *umpire* typed to Class *Umpire,* and two players - *playerA* and *playerB* - typed to Class *Player*. Both Class *Umpire* and Class *Player* will define their own StateMachine(s).
/EventProxy.js	This file defines Events and Signals used in the simulation. If we are raising an Event with arguments, we model the Event as a Signal Event, which specifies a Signal Class; we then define attributes for the Signal Class. Each

	Event occurrence has an instance of the Signal, carrying the runtime values specified for the attributes.
/SimulationManager.js	This file is for simulation in Enterprise Architect.
/html/ManagerWorker.js	This file serves as a middle layer between the front-end and back-end. • The front-end posts a message to request information from the ManagerWorker • Since the ManagerWorker generalizes from ContextManager, it has full access to all the contexts such as querying the current active state and querying the runtime value of a variable • The ManagerWorker will post a message to the front-end with the data it retrieved from the back-end
/html/statemachineGUI.js	This file establishes the communication between the front-end and the ManagerWorker, by defining *stateMachineWorker*. It: • Defines the functions *startStateMachineWebWorker* and *stopStateMachineWebWorker* • Defines the functions *onActiveStateResonse* and *onRuntimeValueResponse* with place-holder code: //to do: write user's logic You could simply replace this comment with your logic, as will be demonstrated later in this topic
/html/index.html	This defines the HTML User Interface, such as the buttons and the input to raise Events or display information. You can define CSS and JavaScript in this file.

Customize index.html and statemachineGUI.js

Make these changes to the generated files:

• Create buttons and displays

• Create a CSS style to format the display and enable/disable the button images

• Create an ElapseTimeWorker.js to refresh the display every second

• Create a TimeElapsed function, set to Next Track when the time elapsed reaches the length of the track

• Create JavaScript as the button 'onclick' event handler

• Once an event is broadcast, request the active State and runtime value for *cdPlayer.currentTrack*

• On initialization, request the active State

In statemachineGUI.js find the function *onActiveStateResonse_cdPlayer*

• In CDPlayer_StateMachine_CDUnLoaded, disable all buttons and enable btnLoad

• In CDPlayer_StateMachine_CDLoaded_CDStopped, disable all buttons and enable btnEject and btnPlay

• In CDPlayer_StateMachine_CDLoaded_CDPlaying, enable all buttons and disable btnLoad and btnPlay

• In CDPlayer_StateMachine_CDLoaded_CDPaused, enable all buttons and disable btnLoad

In statemachineGUI.js find the function *onRuntimeValueResponse*

• In *cdPlayer.currentTrack*, we update the display for current track and track length

The Complete Example

The example can be accessed from the 'Resources' page of the Sparx Systems website, by clicking on this link:

CD Player Simulation

Click on the Load Random CD button, and then on the Start Simulation button.

Regular Expression Parser

StateMachine for Regular Expression Parser

The Class *RegularExpressionParser* is defined with one attribute: bMatch.

RegularExpressionParser
+   bMatch: boolean = false
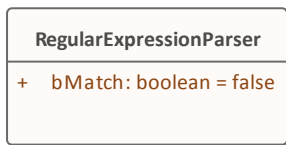
A StateMachine is used to describe the regular expression (a|b)*abb

Regular Expression **(a|b)*abb** implemented in Statemachine.
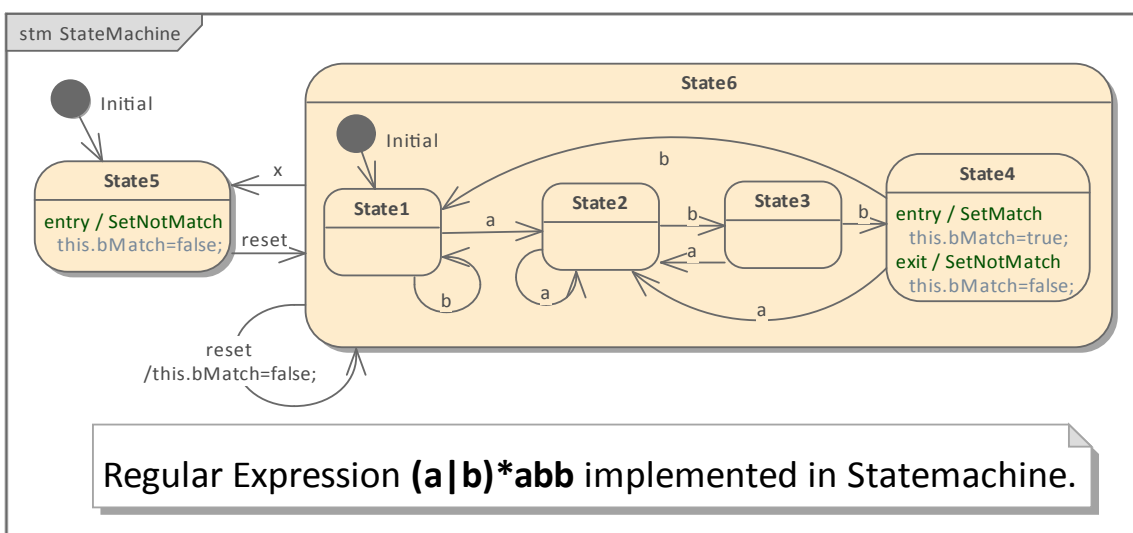
- The transition triggers are specified as events *a*, *b*, *x* and *reset*
- On entry to State4, bMatch is set to True; on exit from State4, bMatch is set to False
- On entry to State5, bMatch is set to False
- On self transition of State6, bMatch is set to False

Customize index.html and statemachineGUI.js

Make these changes to the generated files:
- Create an HTML input field and an image to indicate the result
- Create JavaScript as the field's *oninput* event handler
- Create the function 'SetResult' to toggle the pass/fail image
- Create the function 'getEventStr', which will return 'a' on '*a*' and 'b' on '*b*', but will return 'x' on any other character
- On initialize, broadcast '*reset*'
- On the broadcast event, request the runtime variable 'regxParser.bMatch'

In *statemachineGUI.js*, find the function 'onRuntimeValueResponse'.
- In 'regxParser.bMatch', we will receive 'True' or 'False' and pass it into 'SetResult' to update the image

The Complete Example

The example can be accessed from the 'Resources' page of the Sparx Systems website, by clicking on this link:

Regular Expression Parser Simulation

Entering a State

The semantics of entering a State depend on the type of State and the manner in which it is entered.

In all cases, the entry Behavior of the State is executed (if defined) upon entry, but only after any effect Behavior associated with the incoming Transition is completed. Also, if a doActivity Behavior is defined for the State, this Behavior commences execution immediately after the entry Behavior is executed.

For a Composite State with one or more Regions defined, a number of alternatives exist for each Region:

- *Default entry*: This situation occurs when the owning Composite State is the direct target of a Transition; after executing the entry Behavior and forking a possible doActivity Behavior execution, State entry continues from an initial Pseudostate via its outgoing Transition (known as the default Transition of the State) if it is defined in the Region
  If no initial Pseudostate is defined, this Region will not be active

- *Explicit entry*: If the incoming Transition or its continuations terminate on a directly contained Substate of the owning composite State, then that Substate becomes active and its entry Behavior is executed after the execution of the entry Behavior of the containing composite State
  This rule applies recursively if the Transition terminates on an indirect (deeply nested) Substate

- *Shallow history entry*: If the incoming Transition terminates on a shallowHistory Pseudostate of this Region, the active Substate becomes the Substate that was most recently active (except FinalState) prior to this entry, unless this is the first entry into this State; if it is the first entry into this State or the previous entry had reached a Final, a default shallow history Transition will be taken if it is defined, otherwise the default State entry is applied

- *Deep history entry*: The rule for this case is the same as for shallow history except that the target Pseudostate is of type deepHistory and the rule is applied recursively to all levels in the active State configuration below this one

- *Entry point entry*: If a Transition enters the owning composite State through an entryPoint Pseudostate, then the outgoing Transition originating from the entry point and penetrating into the State in this region is taken; if there are more outgoing Transitions from the entry points, each Transition must target a different Region and all Regions are activated concurrently

For orthogonal States with multiple Regions, if the Transition explicitly enters one or more Regions (in the case of a Fork or entry point), these Regions are entered explicitly and the others by default.

In this example, we demonstrate a model with all these entry behaviors for an orthogonal State.

Modeling a StateMachine
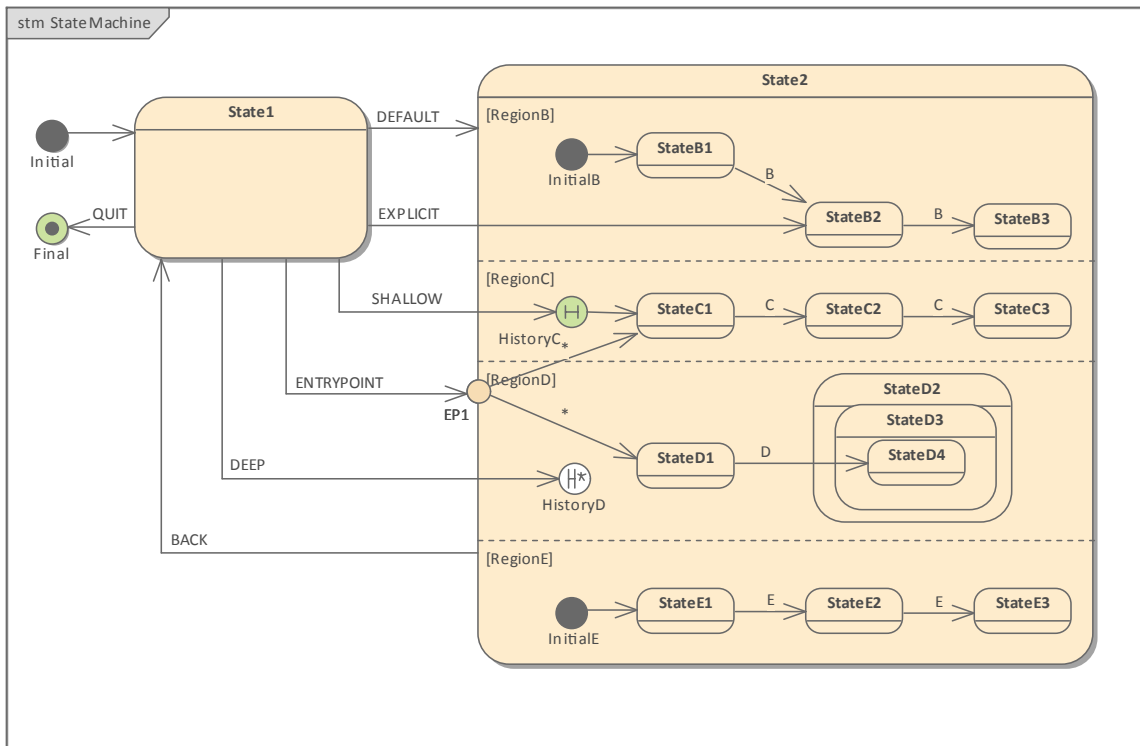
Context of StateMachine

1. Create a Class element named *MyClass*, which serves as the context of the StateMachine.

2. Right-click on *MyClass* in the Project Browser and select the 'Add | StateMachine' option.

StateMachine

1. Add to the diagram an *Initial* Node, a State named *State1*, a State named *State2*, and a Final element named *final.*

2. Enlarge *State2* on the diagram, right-click on it and select the 'Advanced | Define Concurrent Substates' option, and define *RegionB, RegionC, RegionD* and *RegionE.*

3. Right-click on *State2* and select the 'New Child Element | Entry Point' option to create the Entry Point *EP1*.

4. In *RegionB*, create the elements *InitialB*, transition to *StateB1*, transition to *StateB2*, transition to *StateB3*; all transitions triggered by Event *B.*

5. In *RegionC*, create the elements shallow *HistoryC* (right-click on History node | Advanced | Deep History | uncheck), transition to *StateC1*, transition to *StateC2*, transition to *StateC3*; all transitions triggered by Event *C.*

6. In *RegionD*, create the elements deep *HistoryD* (right-click on History node | Advanced | Deep History | check), transition to *StateD1*, create *StateD2* as parent of *StateD3*, which is parent of *StateD4*; transition from *StateD1* to *StateD4*; triggered by Event *D.*

7. In *RegionE*, create the elements *InitialE*, transition to *StateE1*, transition to *StateE2*, transition to *StateE3*; all transitions triggered by Event *E.*

8. Draw transitions from the Entry Point *EP1* to *StateC1* and *StateD1.*

Draw transitions for different entry types:

1. Default Entry: *State1* to *State2*; triggered by Event DEFAULT.

2. Explicit Entry: *State1* to *StateB2*; triggered by Event EXPLICIT.

3. Shallow History Entry: *State1* to *HistoryC*; triggered by Event SHALLOW.

4. Deep History Entry: *State1* to *HistoryD*; triggered by Event DEEP.

5. Entry Point Entry: *State1* to *EP1*; triggered by Event ENTRYPOINT.

Other Transitions:

1. Composite State Exit: from *State2* to *State1*; triggered by Event BACK.

2. *State1* to *Final*, triggered by Event QUIT.

Simulation

Artifact

Enterprise Architect supports C, C++, C#, Java and JavaScript. We use JavaScript in this example because we don't need to install a compiler. (For other languages, either Visual Studio or JDK are required.)

1. On the 'Artifacts' page of the Diagram Toolbox, drag the Executable StateMachine icon onto a diagram and create an Artifact named *EnteringAStateSimulation*. Set the language to JavaScript.

2. Ctrl+drag the *MyClass* element from the Project Browser onto the *EnteringAStateSimulation* Artifact, select the 'Paste as Property' option and give the Property the name *myClass*.

«executable statemachine»
EnteringAStateSimulation

myClass: MyClass
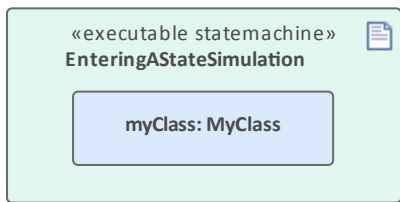
Code Generation

1. Click on *EnteringAStateSimulation* and select the 'Simulate > Executable > StateMachines > Generate, build and run' ribbon option.

2. Specify a directory for the generated source code.

Note: The contents of this directory will be cleared before generation; make sure you specify a directory that is used only for StateMachine simulation purposes.
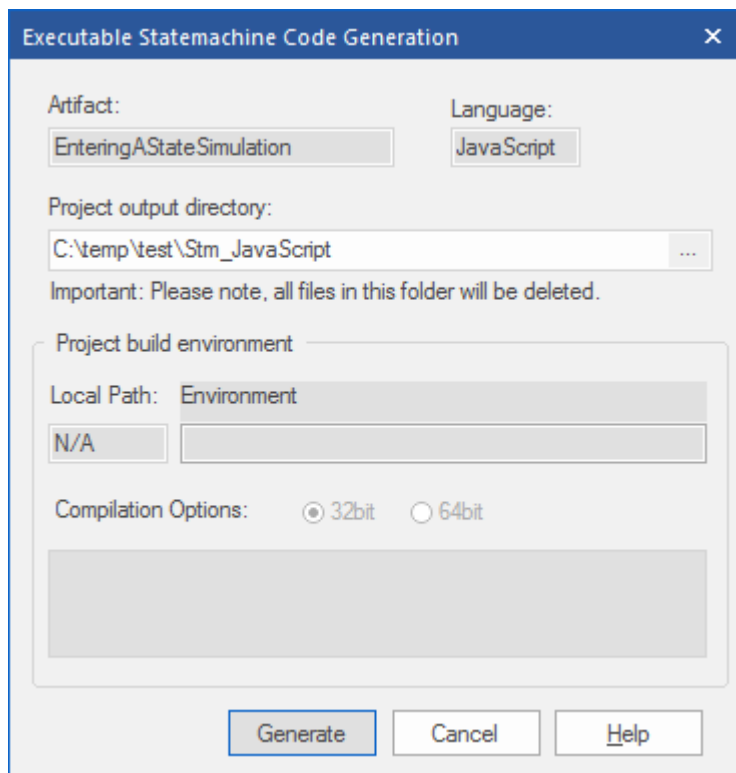
Run Simulation

Tips: You can view the execution trace sequence from the Simulation window, which you open by selecting the 'Simulate > Core > Dynamic Simulation > Open Simulation Window' ribbon option

When the simulation begins, *State1* is active and the StateMachine is waiting for events.
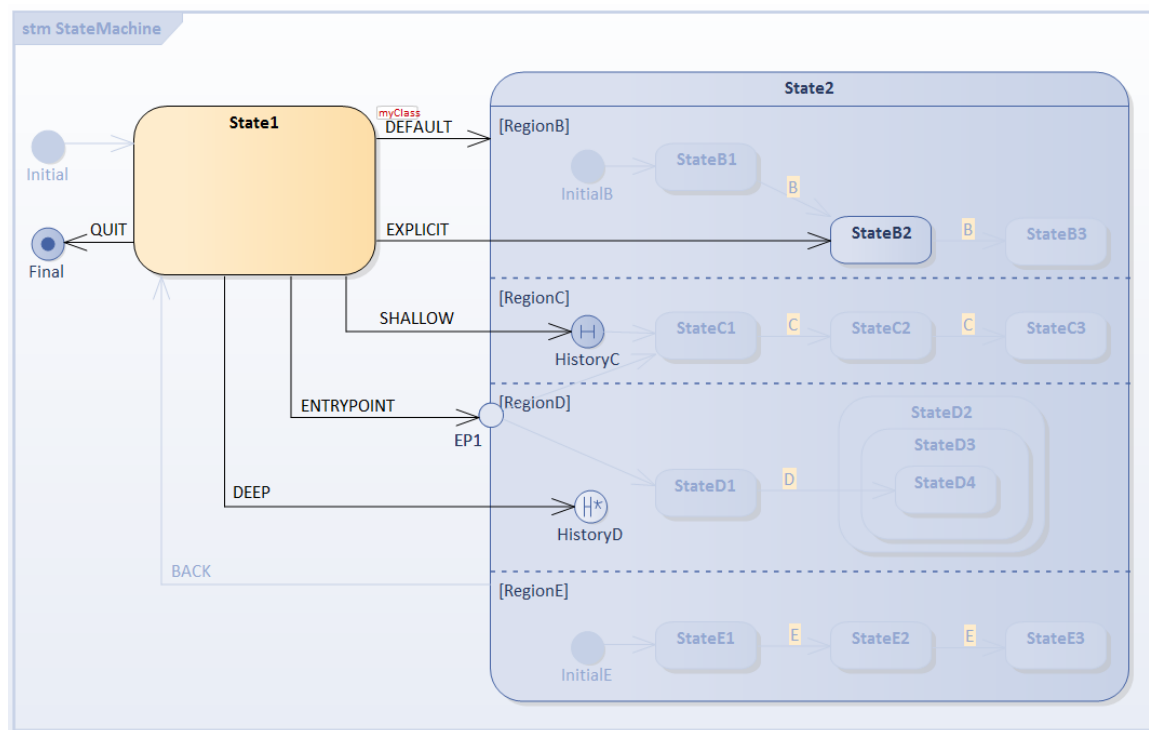
Open the Simulation Events (Triggers) window using the 'Simulate > Core > Events' ribbon option.

1) Select the Default Entry: Trigger Sequence [DEFAULT].
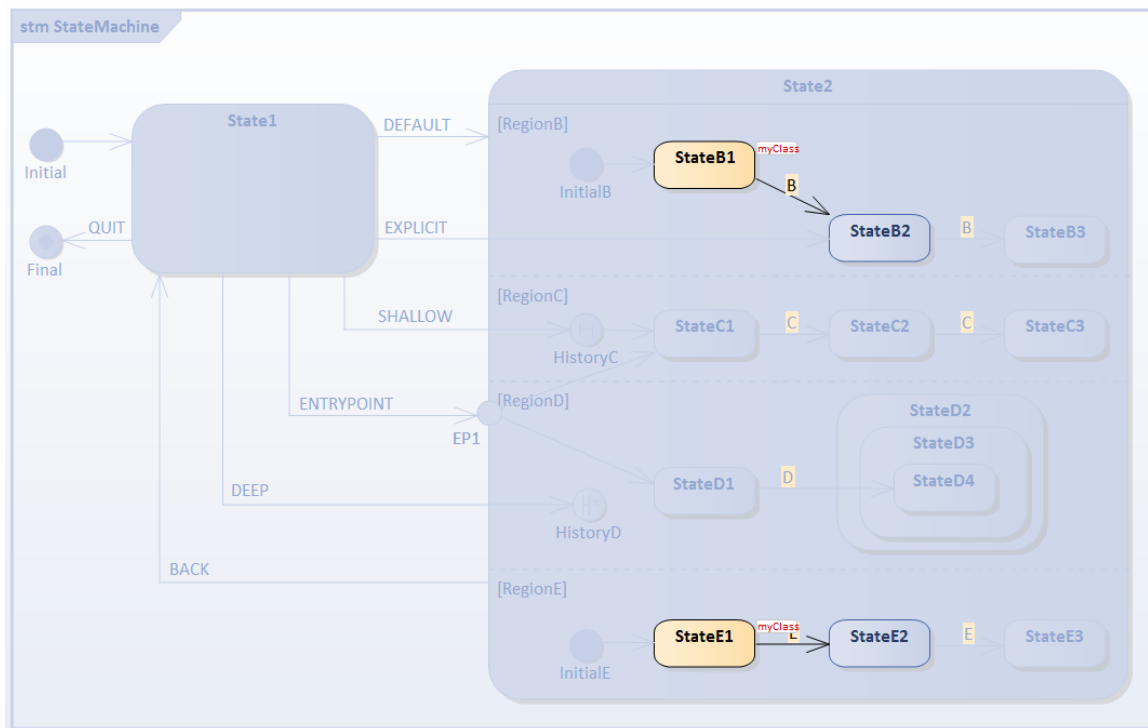
- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state
- *RegionC* and *RegionD* are inactive because no Initial Pseudostates were defined

Select the Trigger [BACK] to reset.

2) Select the Explicit Entry: Trigger Sequence [EXPLICIT].
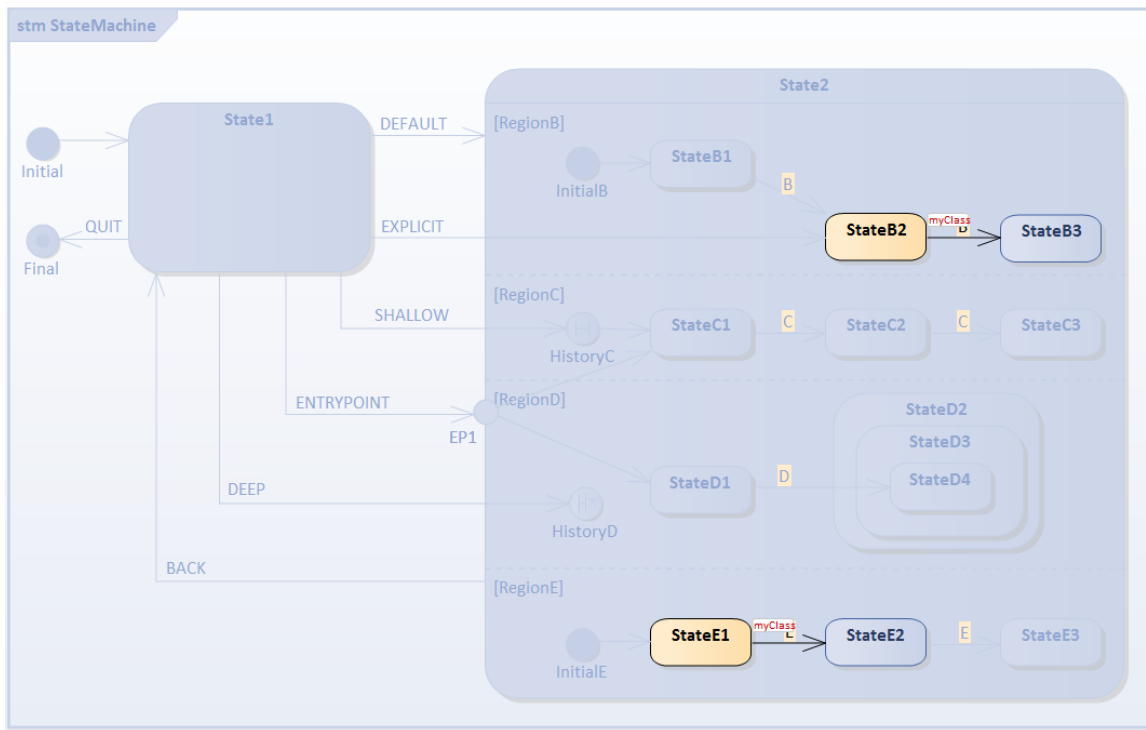
- *RegionB* is activated because the transition targets the contained vertex *StateB2*
- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state
- *RegionC* and *RegionD* are inactive because no Initial Pseudostates were defined

Select the Trigger [BACK] to reset.

3) Select the Default History Transition: Trigger Sequence [SHALLOW].
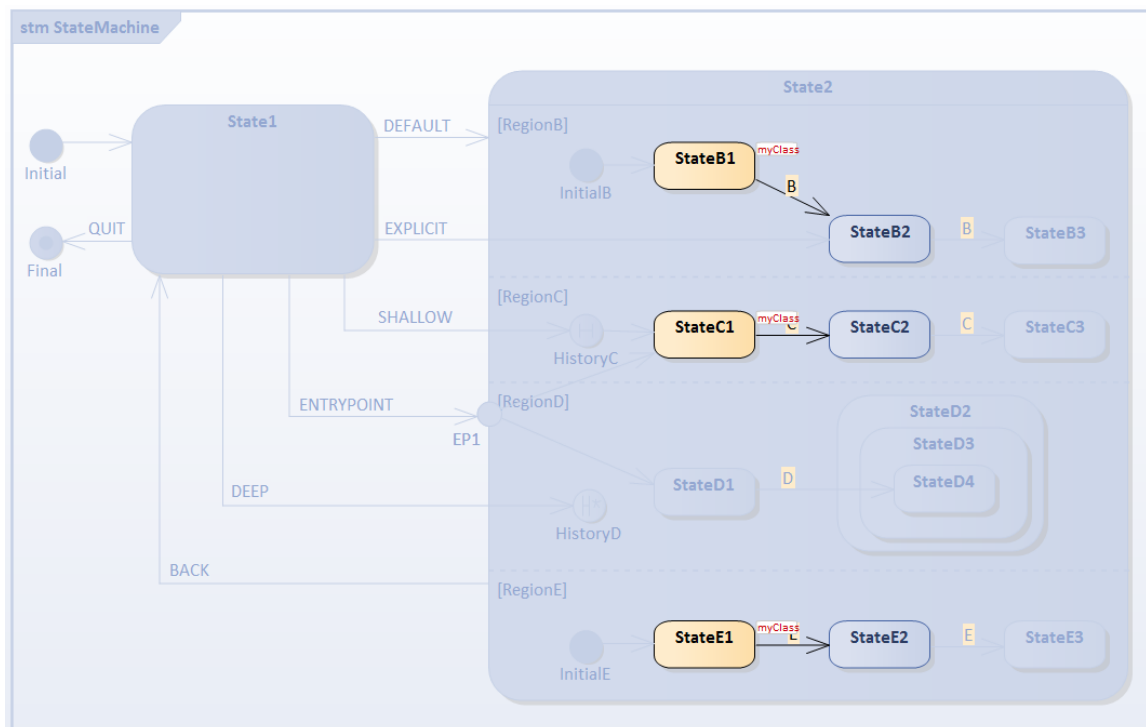
- *RegionC* is activated because the transition targets the contained vertex *HistoryC*; since this region is entered for the first time (and the History pseudostate has nothing to 'remember'), the transition outgoing from *HistoryC* to *StateC1* is executed

- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state

- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state

- *RegionD* is inactive because no Initial Pseudostate was defined

4) Prepare for testing Shallow History Entry: Trigger Sequence [C, C].
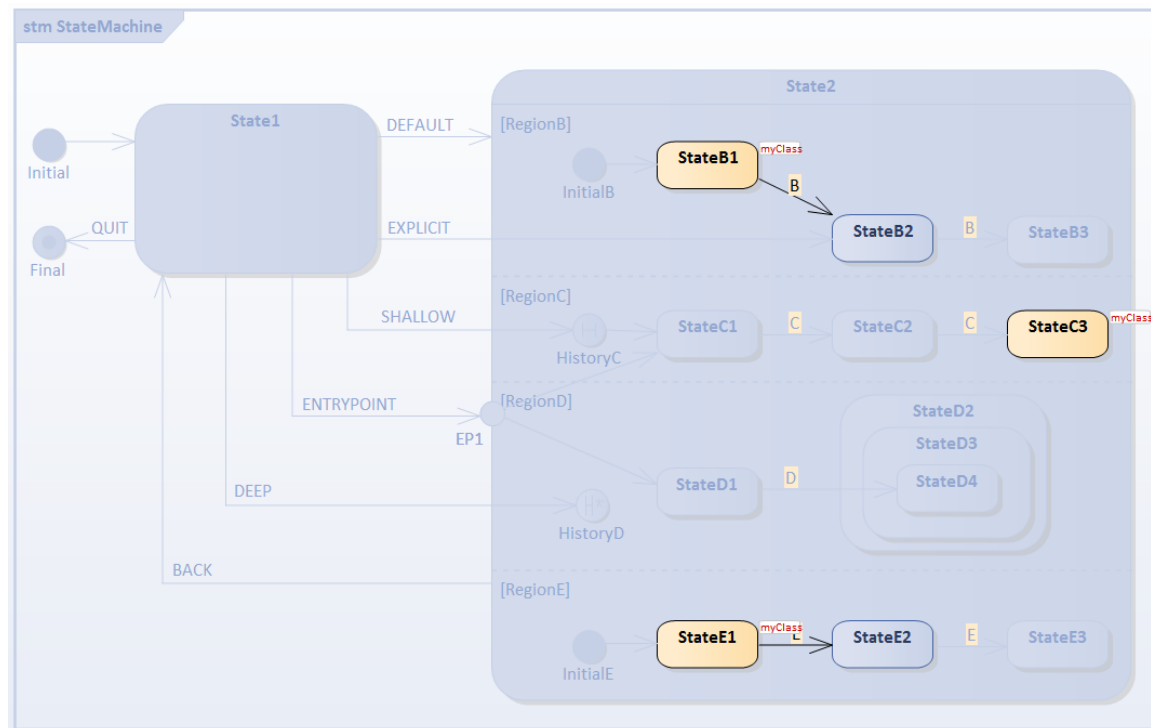
- We assume shallow history pseudostate *HistoryC* can remember *StateC3*

Select the Trigger [BACK] to reset.

5) Select the Shallow History Entry: Trigger Sequence [SHALLOW].
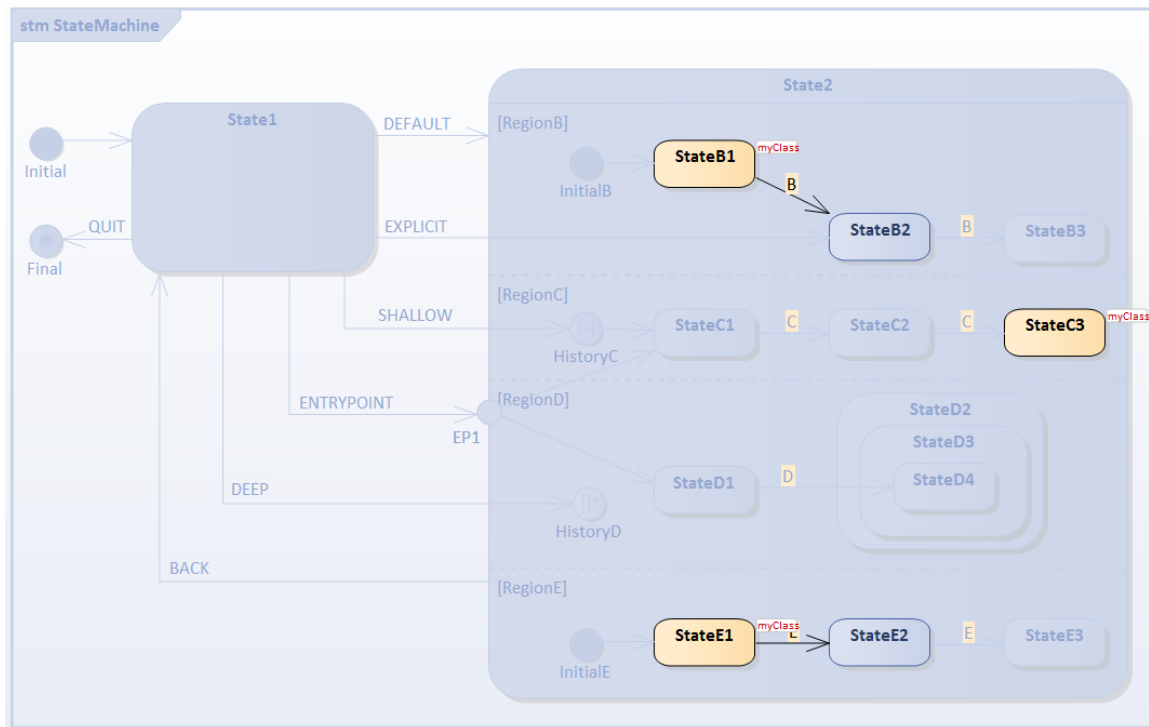
- For *RegionC*, *StateC3* is activated directly

Select the Trigger [BACK] to reset.

6) Select the Entry Point Entry: Trigger Sequence [ENTRYPOINT].
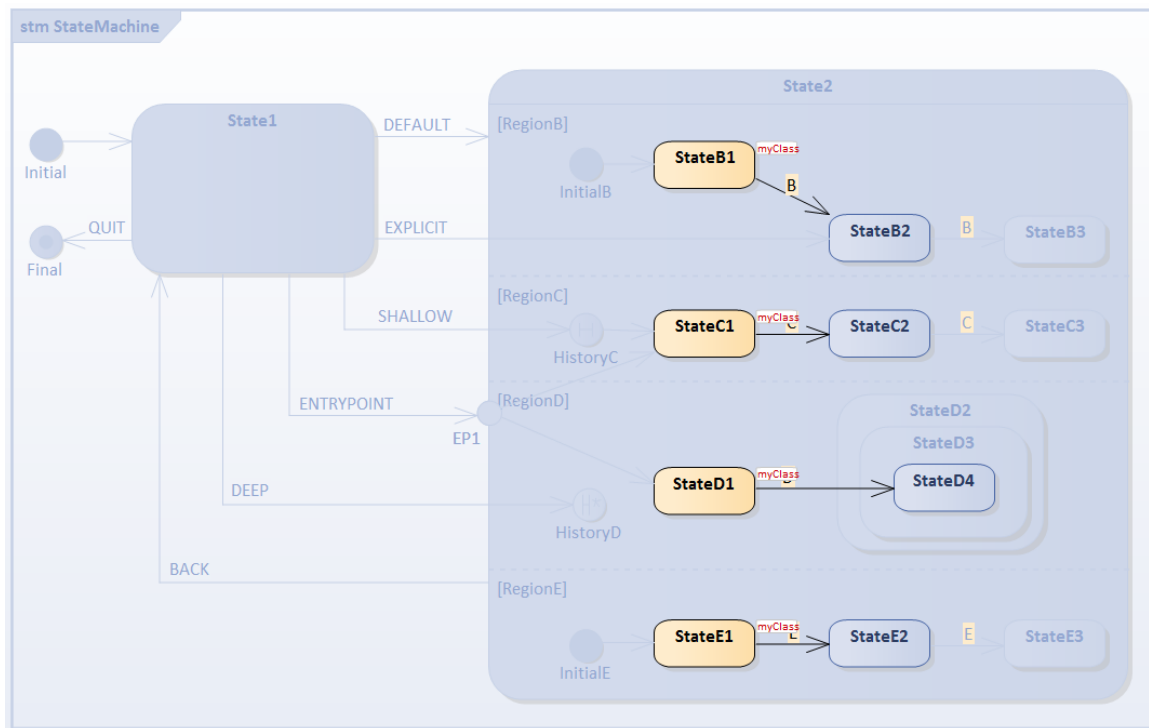
- *RegionC* is activated because the transition from *EP1* targets the contained *StateC1*
- *RegionD* is activated because the transition from *EP1* targets the contained *StateD1*

- *RegionB* is activated because it defines *InitialB*; the transition outgoing from it will be executed, *StateB1* is the active state

- *RegionE* is activated because it defines *InitialE*; the transition outgoing from it will be executed, *StateE1* is the active state

7) Prepare for testing Deep History: Trigger Sequence [D].
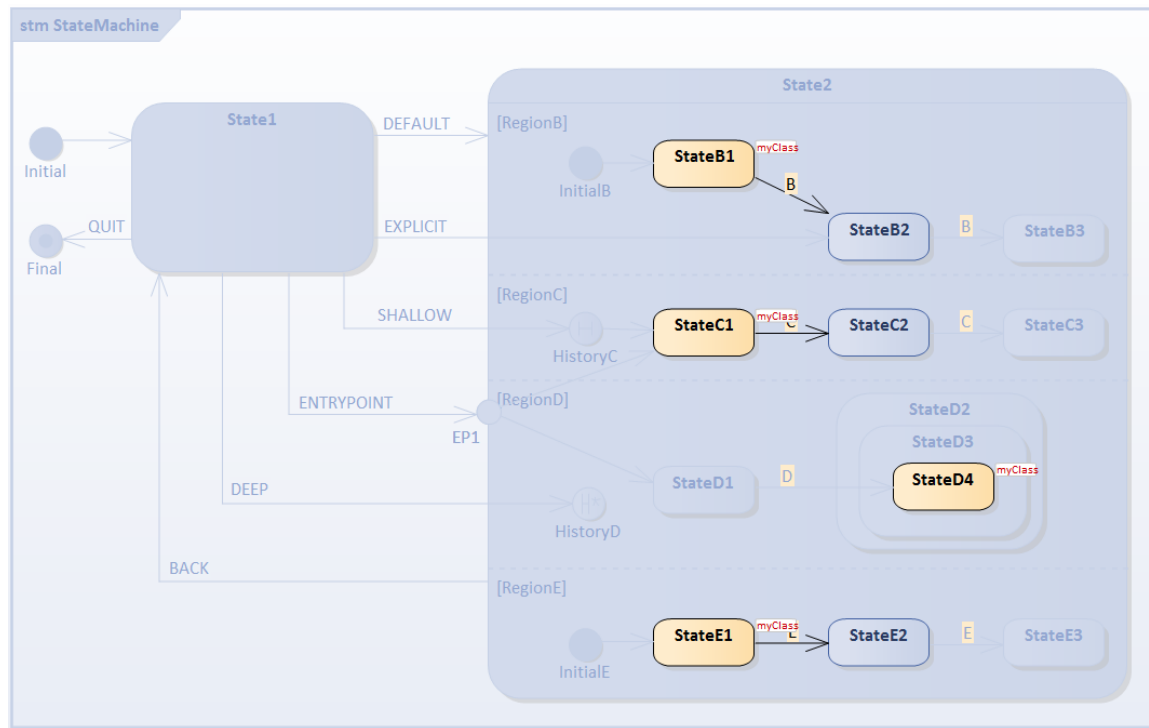
- We assume deep history pseudostate *HistoryD* can remember *StateD2*, *StateD3* and *StateD4*

Select the Trigger [BACK] to reset.

8) Select the Deep History Entry: Trigger Sequence [DEEP].
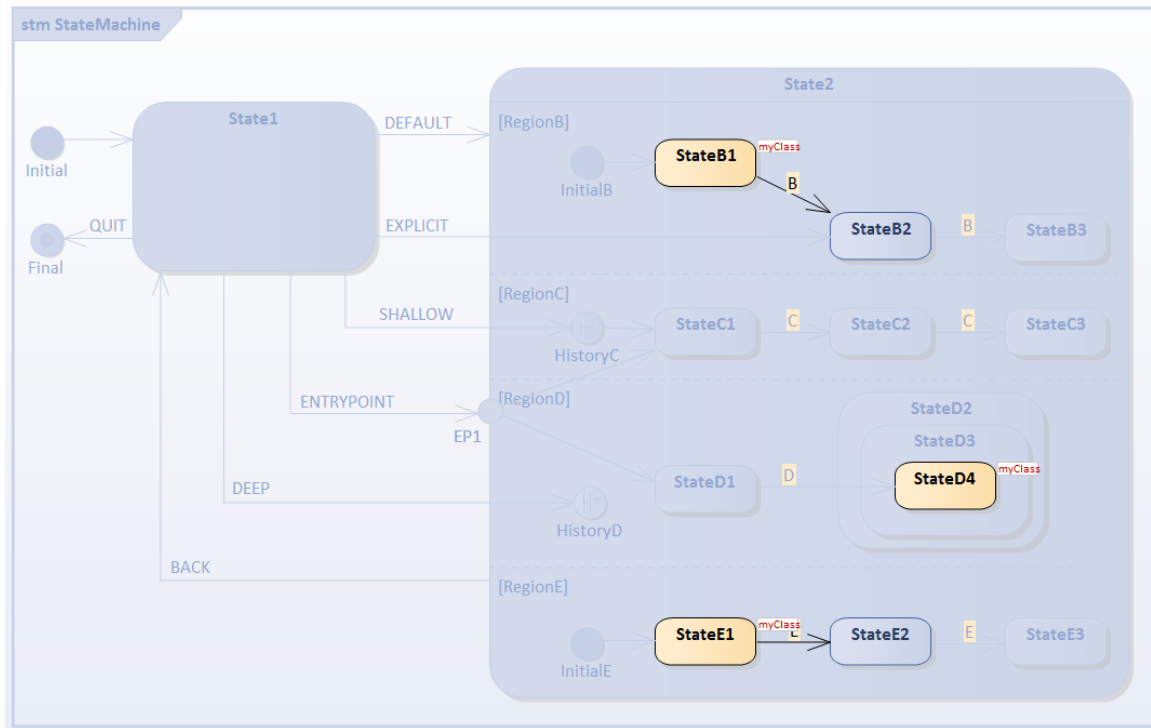
- For *RegionD*, *StateD2*, *StateD3* and *StateD4* are entered; the traces are:
  - myClass[MyClass].StateMachine_State1 EXIT

    - myClass[MyClass].State1__TO__HistoryD_105793_61752 Effect

    - myClass[MyClass].StateMachine_State2 ENTRY

    - myClass[MyClass].StateMachine_State2 DO

    - myClass[MyClass].InitialE_105787__TO__StateE1_61746 Effect

    - myClass[MyClass].StateMachine_State2_StateE1 ENTRY

    - myClass[MyClass].StateMachine_State2_StateE1 DO

    - myClass[MyClass].InitialB_105785__TO__StateB1_61753 Effect

    - myClass[MyClass].StateMachine_State2_StateB1 ENTRY

    - myClass[MyClass].StateMachine_State2_StateB1 DO

    - myClass[MyClass].StateMachine_State2_StateD2 ENTRY

    - myClass[MyClass].StateMachine_State2_StateD2_StateD3 ENTRY

    - myClass[MyClass].StateMachine_State2_StateD2_StateD3_StateD4 ENTRY

Example: Fork and Join

Fork pseudostates split an incoming Transition into two or more Transitions, terminating in Vertices in orthogonal Regions of a Composite State. The Transitions outgoing from a Fork pseudostate cannot have a guard or a trigger, and the effect behaviors of the individual outgoing Transitions are, at least conceptually, executed concurrently with each other.

Join pseudostates are a common target Vertex for two or more Transitions originating from Vertices in different orthogonal Regions. Join pseudostates perform a synchronization function, whereby all incoming Transitions have to complete before execution can continue through an outgoing Transition.

In this example, we demonstrate the behavior of a StateMachine with Fork and Join pseudostates.

Modeling StateMachine
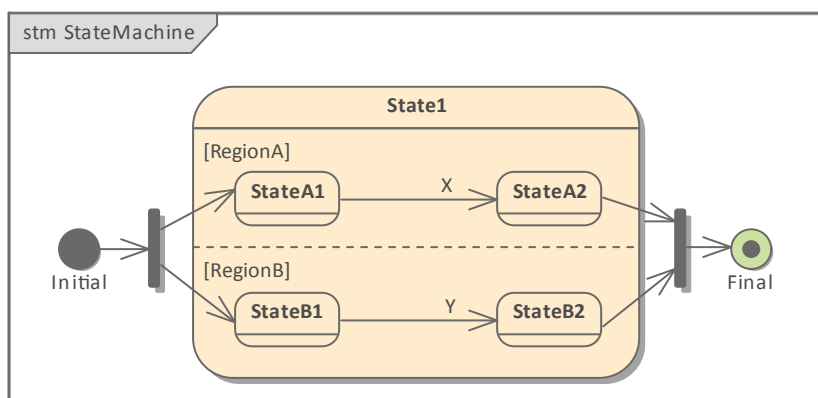
Context of StateMachine

- Create a Class element named *MyClass*, which serves as the context of a StateMachine

- Right-click on *MyClass* in the Project Browser and select the 'Add | StateMachine' option

StateMachine

- Add an *Initial* Node, a *Fork*, a State named *State1*, a *Join*, and a *Final* to the diagram

- Enlarge *State1*, right-click on it on the diagram and select the 'Advanced | Define Concurrent Substates | Define' option and define *RegionA* and *RegionB*

- In *RegionA*, define *StateA1*, transition to *StateA2*, triggered by event *X*

- In *RegionB*, define *StateB1*, transition to *StateB2*, triggered by event *Y*

- Draw other transitions: *Initial* to *Fork*; *Fork* to *StateA1* and *StateB1*; *StateA2 and StateB2* to *Join*; *Join* to *Final*

Simulation

Artifact

Enterprise Architect supports C, C++, C#, Java and JavaScript; we will use JavaScript in this example because we don't need to install a compiler (for the other languages, either Visual Studio or JDK are required).

- From the Diagram Toolbox select the 'Artifacts' page and drag the Executable StateMachine icon onto the diagram to create an Artifact; name it *ForkNJoinSimulation* and set its 'Language' field to 'JavaScript'

- Ctrl+Drag *MyClass* from the Project Browser and drop it on the *ForkNJoinSimulation* Artifact as a Property; give it

the name *myClass*
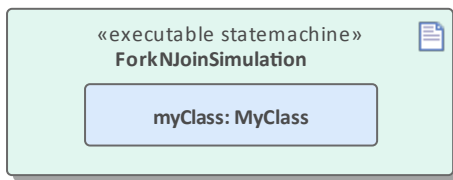
Code Generation

- Click on *ForkNJoinSimulation* and select the 'Code > Executable > StateMachines > Generate, Build and Run' ribbon option
- Specify a directory for the generated source code

Note: The contents of this directory will be cleared before generation; make sure you point to a directory that exists only for StateMachine simulation purposes.
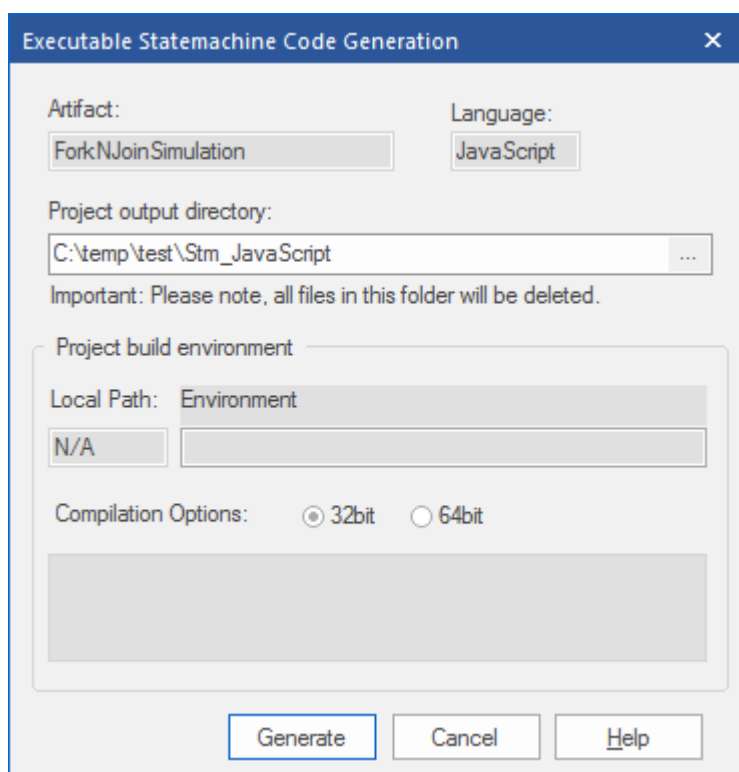

Run Simulation

When the simulation is started, *State1*, *StateA1* and *StateB1* are active and the StateMachine is waiting for events.
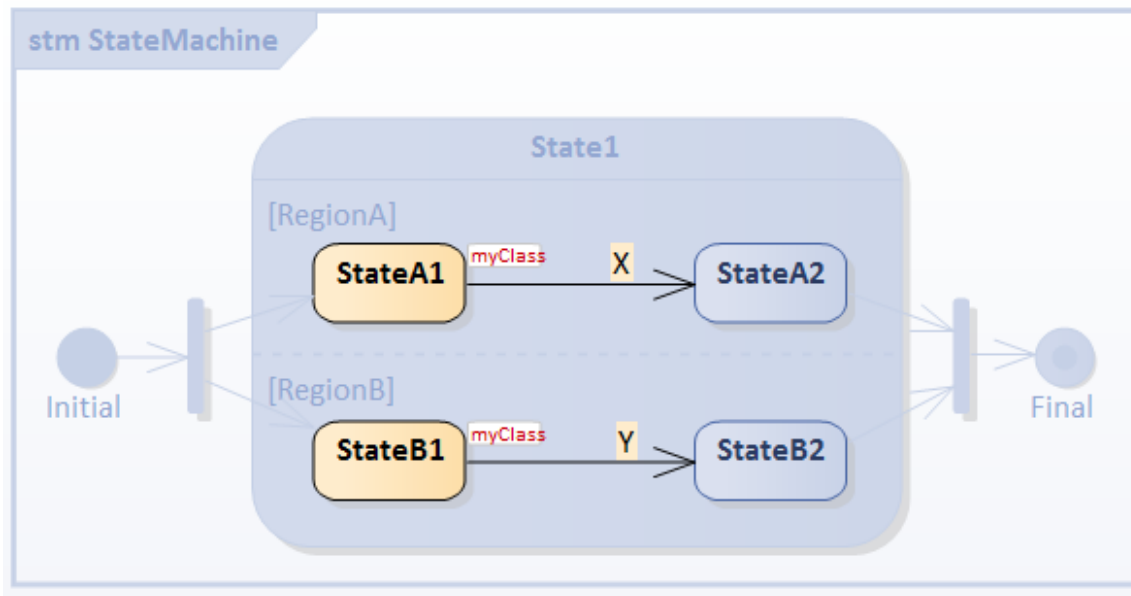
Select the 'Simulate > Core > Events' ribbon option to display the Simulation Events window.

On Trigger event *X*, *StateA1* will exit and enter *StateA2*; after the entry and doActivity behavior has run, the completion events of *StateA2* are dispatched and recalled. Then the transition from *StateA2* to the *Join* pseudostate is enabled and traversed.

Note: *Join* must wait for all incoming Transitions to complete before execution can continue through an outgoing Transition. Since the branch from *RegionB* is not complete (because *StateB1* is still active and waiting for triggers) the transition from *Join* to *Final* will not be executed at this moment.
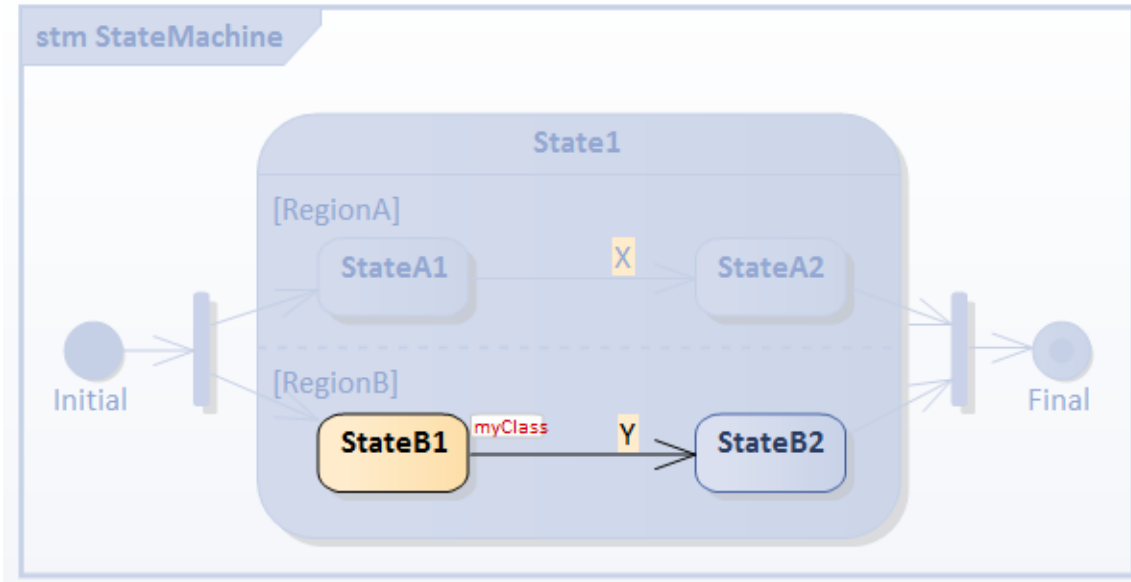
On Trigger event *Y*, *StateB1* will exit and enter *StateB2*; after the entry and doActivity behavior has run, completion events of *StateB2* are dispatched and recalled. Then the transition from *StateB2* to the *Join* pseudostate is enabled and traversed. This satisfies the criteria of all the incoming transitions of *Join* having completed, so the transition from *Join* to *Final* is executed. Simulation has ended.

*Tips: Y*ou can view the execution trace sequence from the Simulation window ('Simulate > Core > Dynamic Simulation > Open Simulation Window' ribbon option).

myClass[MyClass].Initial_82285__TO__fork_82286_82286_61745 Effect

myClass[MyClass].StateMachine_State1 ENTRY

myClass[MyClass].StateMachine_State1 DO

myClass[MyClass].fork_82286_82286__TO__StateA1_57125 Effect

myClass[MyClass].StateMachine_State1_StateA1 ENTRY

myClass[MyClass].StateMachine_State1_StateA1 DO

myClass[MyClass].fork_82286_82286__TO__StateB1_57126 Effect

myClass[MyClass].StateMachine_State1_StateB1 ENTRY

myClass[MyClass].StateMachine_State1_StateB1 DO

Trigger X

myClass[MyClass].StateMachine_State1_StateA1 EXIT

myClass[MyClass].StateA1__TO__StateA2_57135 Effect

myClass[MyClass].StateMachine_State1_StateA2 ENTRY

myClass[MyClass].StateMachine_State1_StateA2 DO

myClass[MyClass].StateMachine_State1_StateA2 EXIT

myClass[MyClass].StateA2__TO__join_82287_82287_57134 Effect

Trigger Y

myClass[MyClass].StateMachine_State1_StateB1 EXIT

myClass[MyClass].StateB1__TO__StateB2_57133 Effect

myClass[MyClass].StateMachine_State1_StateB2 ENTRY

myClass[MyClass].StateMachine_State1_StateB2 DO

myClass[MyClass].StateMachine_State1_StateB2 EXIT

myClass[MyClass].StateB2__TO__join_82287_82287_57132 Effect

myClass[MyClass].StateMachine_State1 EXIT

myClass[MyClass].join_82287_82287__TO__Final_105754_57130 Effect

Example: Deferred Event Pattern

Enterprise Architect supports the Deferred Event Pattern.

To create a Deferred Event in a State:

1. Create a self transition for the State.

2. Change the 'kind' of the transition to 'internal'.

3. Specify the Trigger to be the event you want to defer.

4. In the 'Effect' field, type 'defer();'.

To Simulate:

1. Select 'Simulate > Dynamic Simulation > Simulator > Open Simulation Window'. Also select 'Simulate > Dynamic Simulation > Triggers' to open the Simulation Events window.

2. The Simulator Events window helps you to trigger events; double-click on a trigger in the 'Waiting Triggers' column.

3. The Simulation window shows the execution in text. You can type 'dump' in the Simulator command line to show how many events are deferred in the queue; the output might resemble this:
    24850060]   Event Pool: [NEW,NEW,NEW,NEW,NEW,]

Deferred Event Example

This example shows a model using Deferred Events, and the Simulation Events window showing all available Events.

We firstly set up the contexts (the Class elements containing the StateMachines), simulate them in a simple context and raise the event from outside it; then simulate in a client-server context with the Send event mechanism.

Create Context and StateMachine

Create the server context
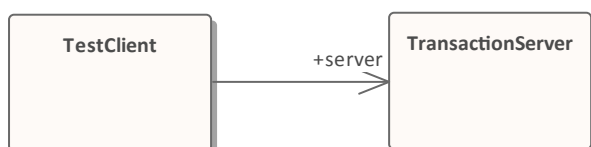
Create a Class diagram and:

1. A Class element *TransactionServer*, to which you add a StateMachine *ServerStateMachine*.

2. A Class element *TestClient*, to which you add a StateMachine *ClientStateMachine*.

3. An Association from *TestClient* to *TransactionServer*, with the target role named *server*.

Modeling for *ServerStateMachine*
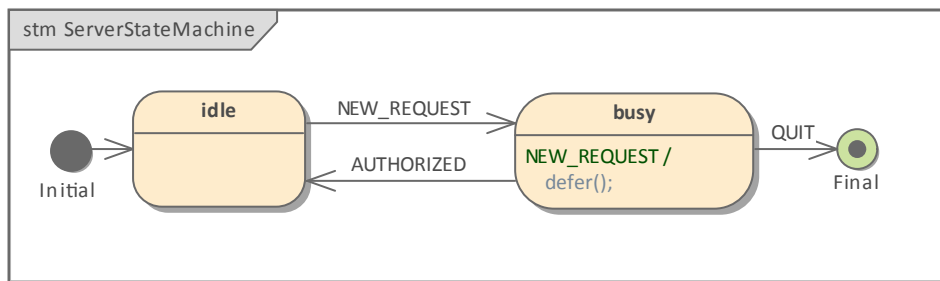
1.   Add an Initial Node *Initial* to the StateMachine diagram, and transition to a State *idle*.
2.   Transition (with event NEW_REQUEST as Trigger) to a State *busy*.
3.   Transition (with event QUIT as Trigger) to a Final State *Final.*
4.   Transition (with event AUTHORIZED as Trigger) to *idle*.
5.   Transition (with event NEW_REQUEST as Trigger and *defer();* as effect) to *busy*

Modeling for *ClientStateMachine*
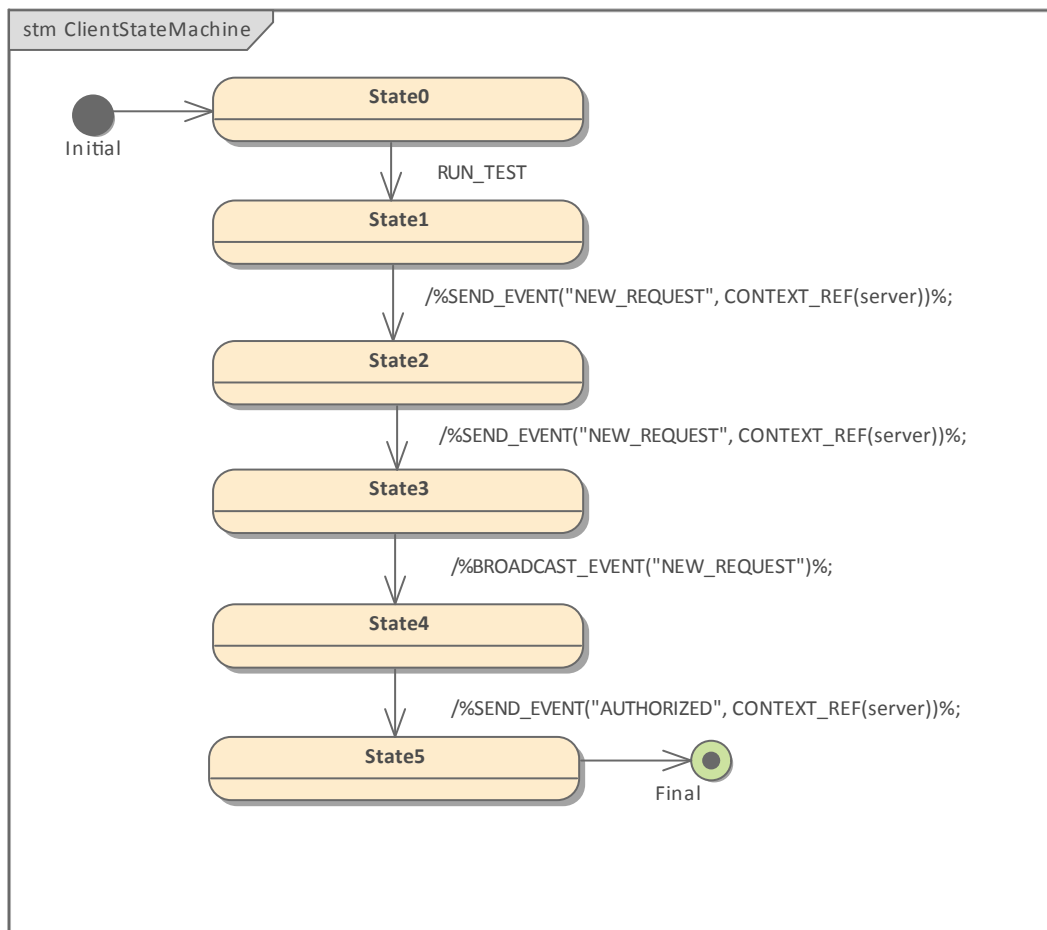
1.   Add an Initial Node *Initial* to the StateMachine diagram, and transition to a State *State0*.
2.   Transition (with event RUN_TEST as trigger) to a State *State1*.
3.   Transition (with effect: %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) to a State *State2*.
4.   Transition (with effect: %SEND_EVENT("NEW_REQUEST", CONTEXT_REF(server))%;) to a State *State3*.

5.   Transition (with effect: %BROADCAST_EVENT("NEW_REQUEST")%;) to a State *State4*.

6.   Transition (with effect: %SEND_EVENT("AUTHORIZED", CONTEXT_REF(server))%;) to a State *State5*.

7.   Transition to a Final State *Final.*

Simulation in a simple context

Create the Simulation Artifact
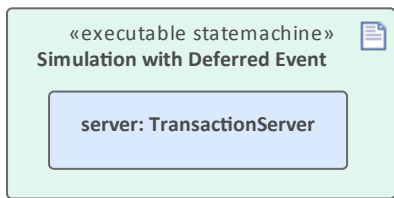
1.   Create an Executable StateMachine Artifact with the name *Simulation with Deferred Event* and the 'Language' field set to *JavaScript.*

2.   Enlarge it, then Ctrl+drag the *TransactionServer* element onto the Artifact and paste it as a property with the name *server*.

Run the Simulation

1.   Select the Artifact, then select the 'Simulate > Compiled Simulation > Build & Run' option, and specify a directory for your code (Note: all the files in the directory will be deleted before simulation starts).

2.   Click on the Generate button.

3.   Select the 'Simulate > Dynamic Simulation > Triggers' option to open the Simulation Event window.
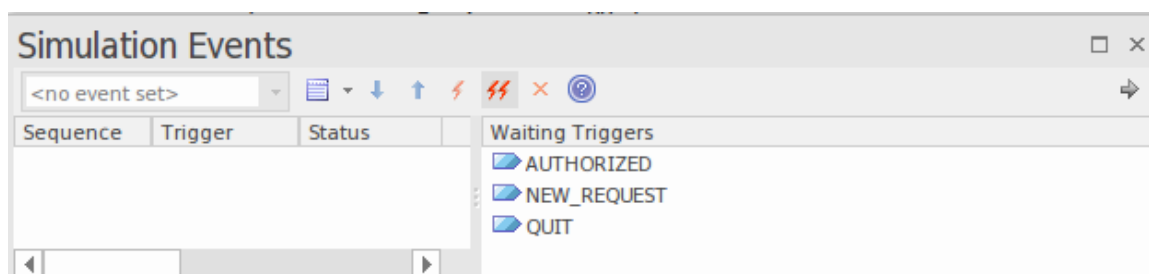
When simulation starts, *idle* will be the active state.
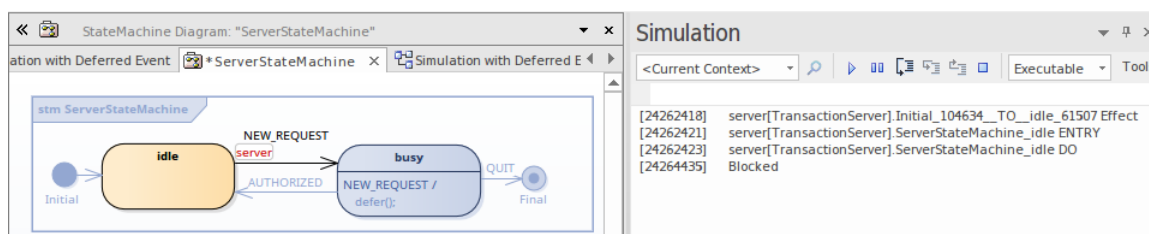
1.   Double-click on NEW_REQUEST in the Simulation Event window to execute it as the Trigger; *idle* is exited and *busy* is activated.
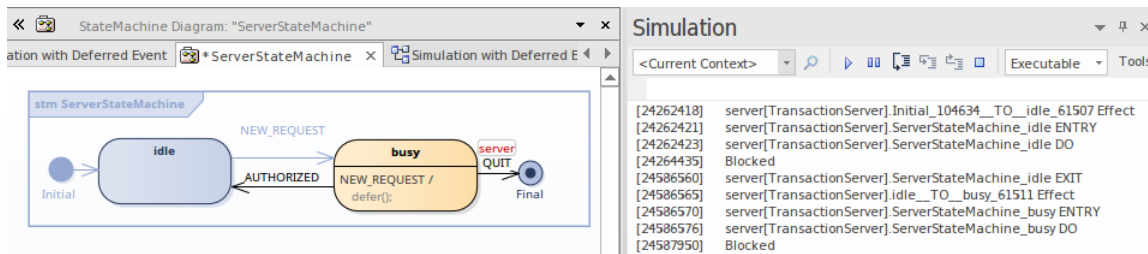
1. Double-click on NEW_REQUEST in the Simulation Event window to execute it again as the Trigger; *busy* remains activated, and an instance of NEW_REQUEST is appended in the Event Pool.

2. Double-click on NEW_REQUEST in the Simulation Event window to execute it a third time as the Trigger; *busy* remains activated, and an instance of NEW_REQUEST is appended in the Event Pool.

3. Type *dump* in the Simulation window command line; notice that the event pool has two instances of NEW_REQUEST.
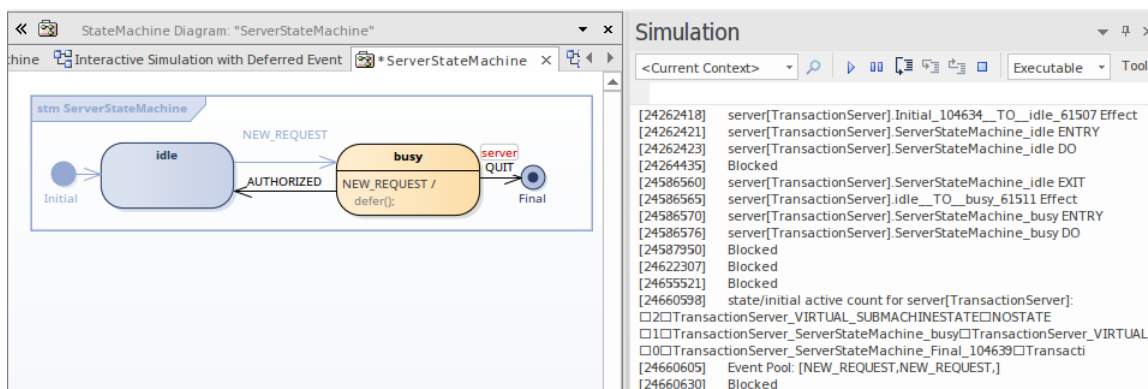
1. Double-click on AUTHORIZED in the Simulation Event window to execute it as the Trigger; these actions take place:
   - *busy* is exited and *idle* becomes active
   - a NEW_REQUEST event is retrieved from the pool, *idle* is exited and *busy* becomes active

2. Type *dump* in the Simulation window command line; there is now only one instance of NEW_REQUEST in the Event Pool.
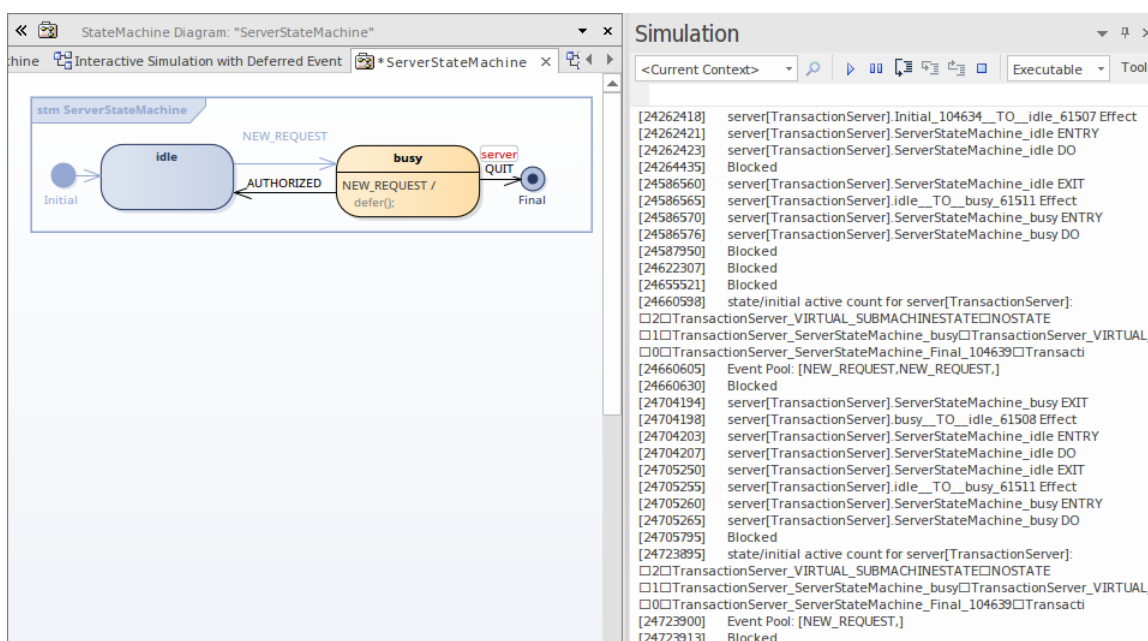
Interactive simulation via Send/Broadcast Event

Create the Simulation Artifact
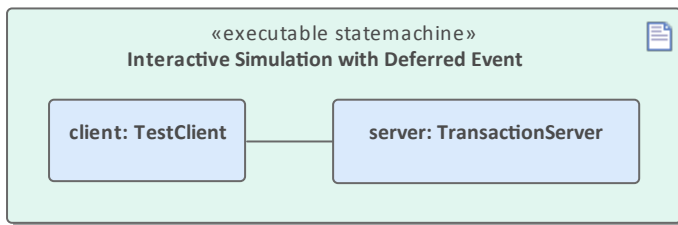
1. Create an Executable StateMachine Artifact with the name *Interactive Simulation with Deferred Event* and the 'Language' field set to *JavaScript;* enlarge the element.

2. Ctrl+Drag the *TransactionServer* element onto the Artifact, and paste it as a property with the name *server.*

3. Ctrl+Drag the *TestClient* element onto the Artifact, and paste it as a property with the name *client.*

4. Create a connector from *client* to *server.*

5. Click on the connector and press Ctrl+L to select the association from the *TestClient* element to the *TransactionServer* element.

Run Interactive Simulation

1. Launch the simulation in the same way as for the simple context.

   Once the simulation has started, the *client* remains at *State0* and the *server* remains at *idle.*
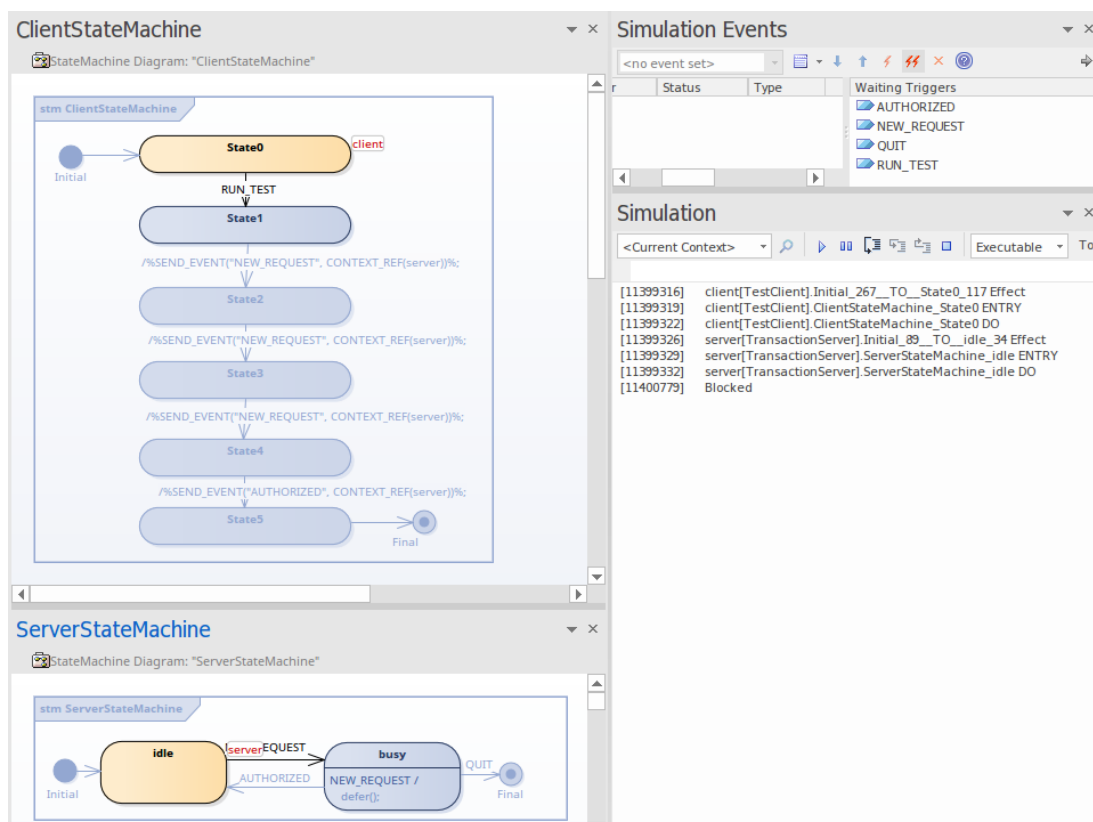
2. Double-click on RUN_TEST in the Simulation Event window to trigger it. The event NEW_REQUEST will be triggered three times (by SEND_EVENT and BROADCAST_EVENT) and AUTHORIZED will be triggered once by SEND_EVENT.
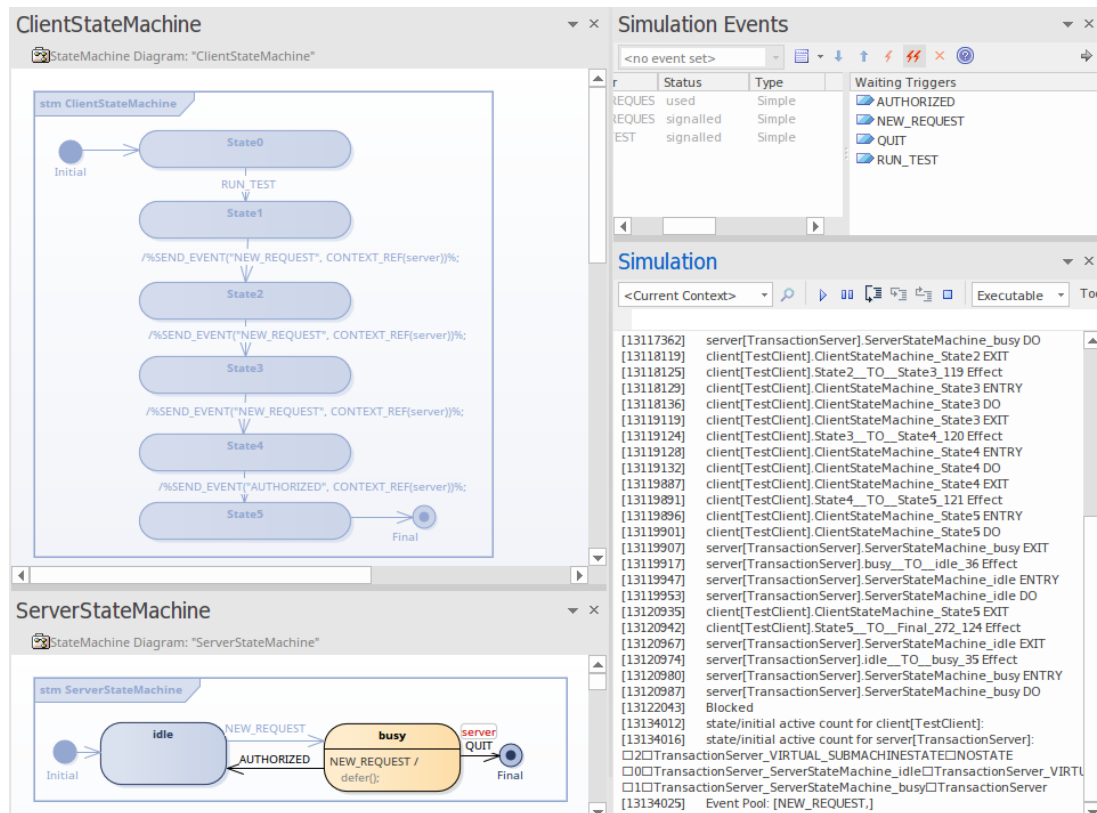
Type *dump* in the Simulation window command line, There is one instance of NEW_REQUEST left in the Event Pool. The result matches our manual triggering test.

Example: Entry and Exit Points (Connection Point References)

Enterprise Architect provides support for Entry and Exit points, and for Connection Point References. In this example, we define two StateMachines for *MyClass* - *StateMachine* and *SubMachine*.
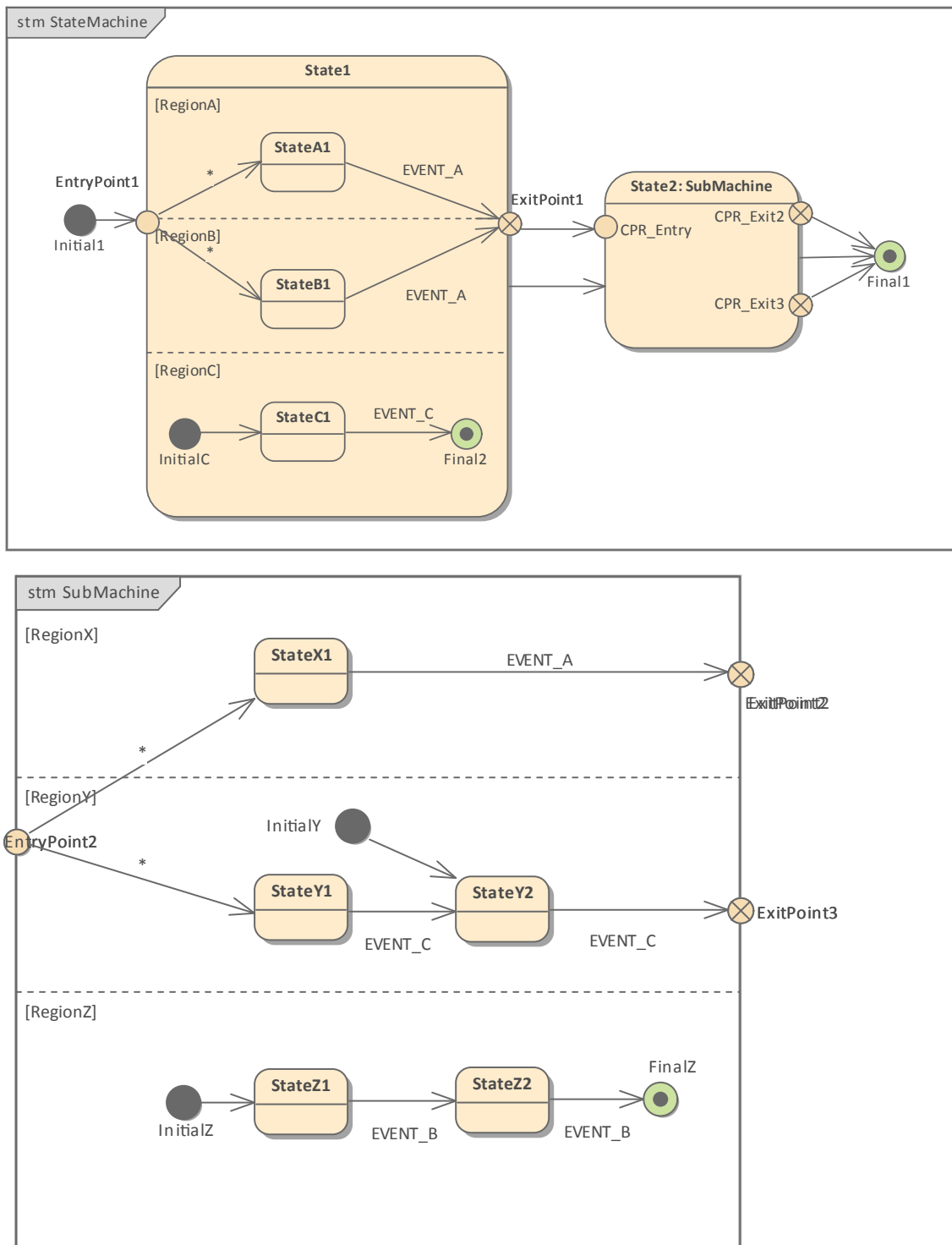
- *State1* is a Composite State (also called an Orthogonal State because it has multiple Regions) with three Regions:

RegionA, *RegionB* and *RegionC*

- *State2* is a SubMachine State calling *SubMachine*, which has three Regions: *RegionX*, *RegionY*, and *RegionZ*
- *EntryPoint1* is defined on *State1* to activate two of the three Regions; *EntryPoint2* is defined on *SubMachine* to activate two of the three Regions
- *ExitPoint1* is defined on *State1*; two exit points *ExitPoint2* and *ExitPoint3* are defined on *SubMachine*
- Connection Point References are defined on *State2* and bind to the Entry/Exit Points of the typing SubMachine
- Initial nodes are defined to demonstrate default activation of the Regions

Entering a State: Entry Point Entry

EntryPoint1 on *State1*

When a Transition targeted on *EntryPoint1* is enabled, *State1* is activated followed by the contained Regions.
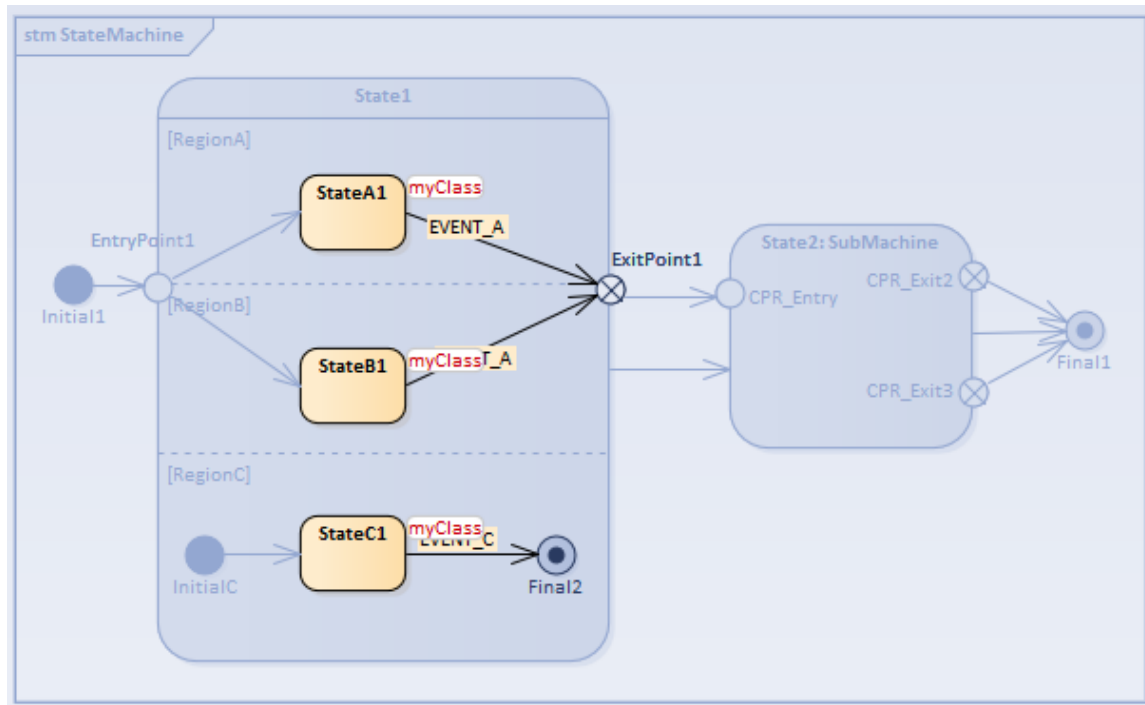
- Explicit activation occurs for *RegionA* and *RegionB*, because each of them is entered by a Transition terminating on one of the Region's contained Vertices
- Default activation occurs for *RegionC*, because it defines an Initial pseudostate *InitialC* and the Transition originating from the *InitialC* to *StateC1* starts execution

EntryPoint2 on *SubMachine*

The Trigger Sequence to be simulated is: [EVENT_C, EVENT_A].

When a Transition targeted on Connection Point Reference *CPR_Entry* on *State2* is enabled, *State2* is activated, followed by the SubMachine's activation through the binding entry points.
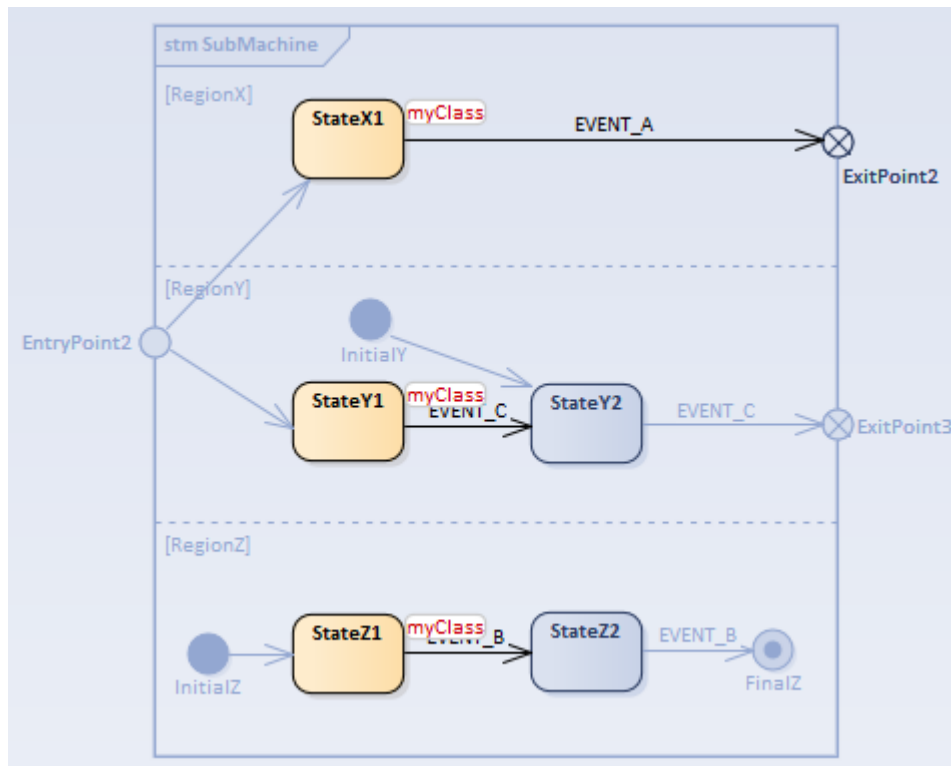
- Explicit activation occurs for *RegionX* and *RegionY*, because each of them is entered by a Transition terminating on one of the Region's contained Vertices - *StateX1* in *RegionX*, *StateY1* in *RegionY*

- Default activation occurs for *RegionZ*, because it defines an Initial pseudostate *InitialZ* and the Transition originating from *InitialZ* to *StateZ1* starts execution

Entering a State: Default Entry

This situation arises when the Composite State is the direct target of a Transition.

Default Entry of *State2*

The Trigger Sequence to be simulated is: [EVENT_A, EVENTC].

When a Transition targeted directly on *State2* is enabled, *State2* is activated, followed by default activation for all the Regions of the SubMachine.
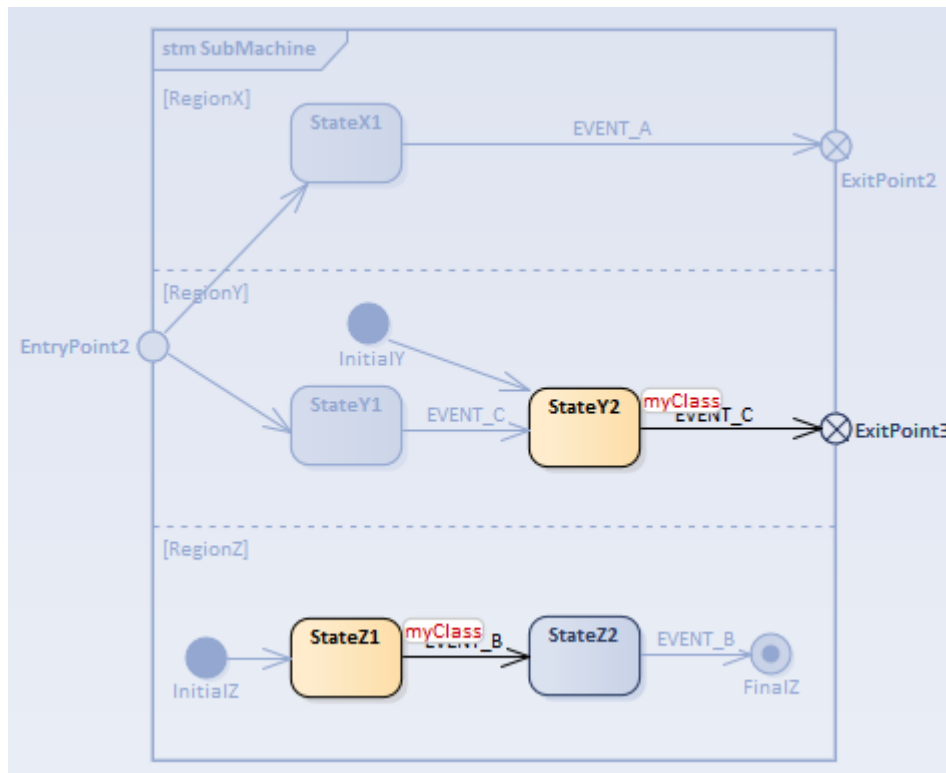
- *RegionX's* State is inactive because it does not define an Initial node
- *RegionY* is activated through *InitialY* and the Transition to *StateY2* is executed
- *RegionZ* is activated through *InitialZ* and the Transition to *StateZ1* is executed

State Exit

State1 Exit

- Trigger Sequence [EVENT_C, EVENT_A]: *RegionC* is inactivated first, then *RegionA* and *RegionB*; after the exit behavior of *State1* is executed, the Transition outgoing from *ExitPoint1* is enabled
- Trigger Sequence [EVENT_A, EVENT_C]: *RegionA* and *RegionB* are inactivated first, then *RegionC*; after the exit behavior of *State1* is executed, the Transition outgoing directly from *State1* is enabled

State2 Exit

Trigger Sequence [EVENT_C, EVENT_A], so the current state resembles this:
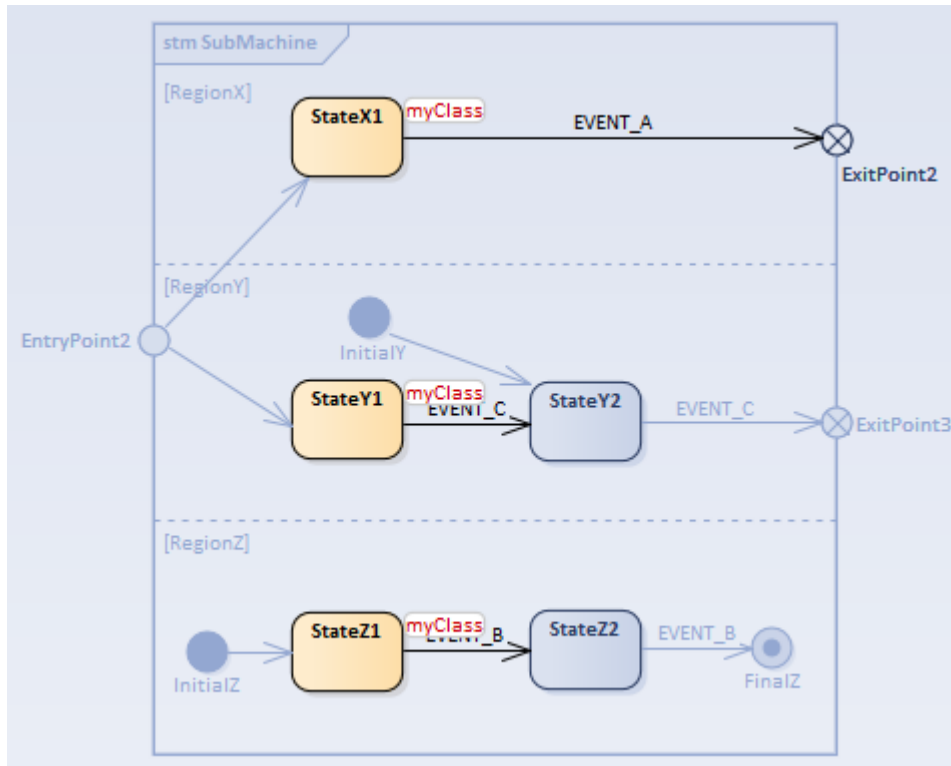
- Trigger Sequence [EVENT_A, EVENT_C, EVENT_C, EVENT_B, EVENT_B]: *RegionX* is inactivated first, then *RegionY*, and *RegionZ* is the last; after the exit behavior of *State2* is executed, the Transition outgoing directly from *State2* is enabled

- Trigger Sequence [EVENT_A, EVENT_B, EVENT_B, EVENT_C, EVENT_C]: *RegionX* is inactivated first, then *RegionZ*, and *RegionY* is the last; after the exit behavior of *State2* is executed, the Transition outgoing from *CPR_Exit3* is enabled (*ExitPoint3* on *SubMachine* is bound to *CPR_Exit3* of *State2*)

- Trigger Sequence [EVENT_C, EVENT_C, EVENT_B, EVENT_B, EVENT_A]: *RegionY* is inactivated first, then *RegionZ*, and *RegionX* is the last; after the exit behavior of *State2* is executed, the Transition outgoing from *CPR_Exit2* is enabled (*ExitPoint2* on *SubMachine* is bound to *CPR_Exit2* of *State2*)

Example: History Pseudostate

State History is a convenient concept associated with Regions of Composite States, whereby a Region keeps track of the configuration a State was in when it was last exited. This allows easy return to that State configuration, if necessary, when the Region next becomes active (for example, after returning from handling an interrupt), or if there is a local Transition that returns to its history.

Enterprise Architect supports two types of History Pseudostate:

- Deep History - representing the full State configuration of the most recent visit to the containing Region; the effect is the same as if the Transition terminating on the deepHistory Pseudostate had, instead, terminated on the innermost State of the preserved State configuration, including execution of all entry Behaviors encountered along the way

- Shallow History - representing a return to only the top-most substate of the most recent State configuration, which is entered using the default entry rule

In this example, the Classes *DeepTurbineManager* and *ShallowTurbineManager* are exactly the same except that the contained StateMachine for the first has a deepHistory Pseudostate and for the second has a shallowHistory Pseudostate.

Both StateMachines have three Composite States: *Turbine_01*, *Turbine_02* and *Turbine_03*, each of which has *Off* and *On* States and a History Pseudostate in its Region.

In order to better observe the difference between Deep History and Shallow History, we execute the two StateMachines in one simulation.
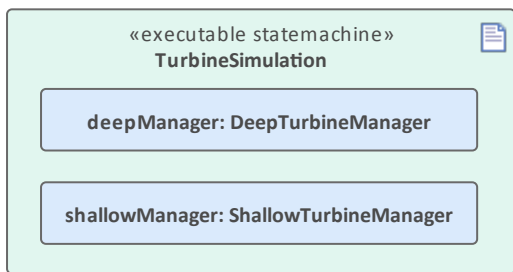
The StateMachine in *DeepTurbineManager* is illustrated in this diagram:
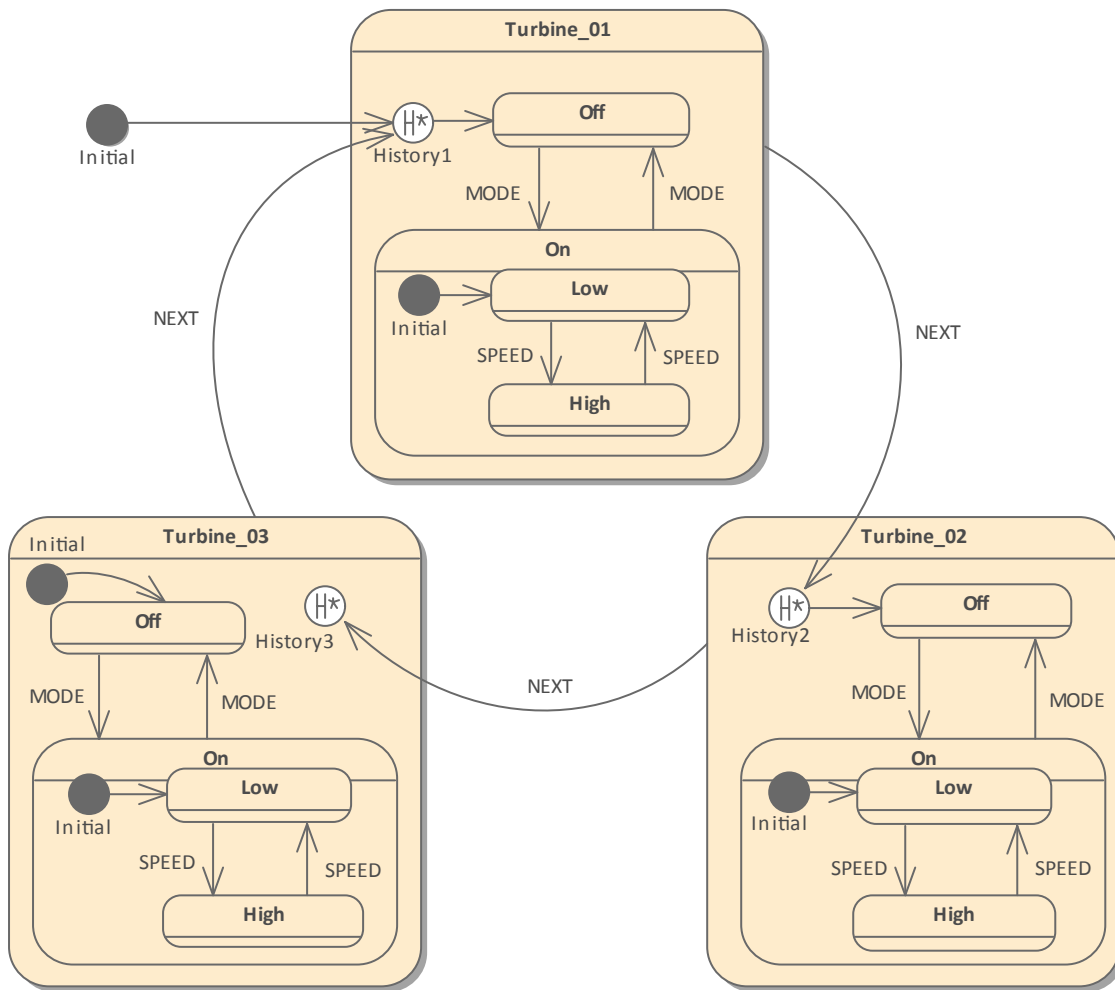
The StateMachine in *ShallowTurbineManager* is illustrated in this diagram:
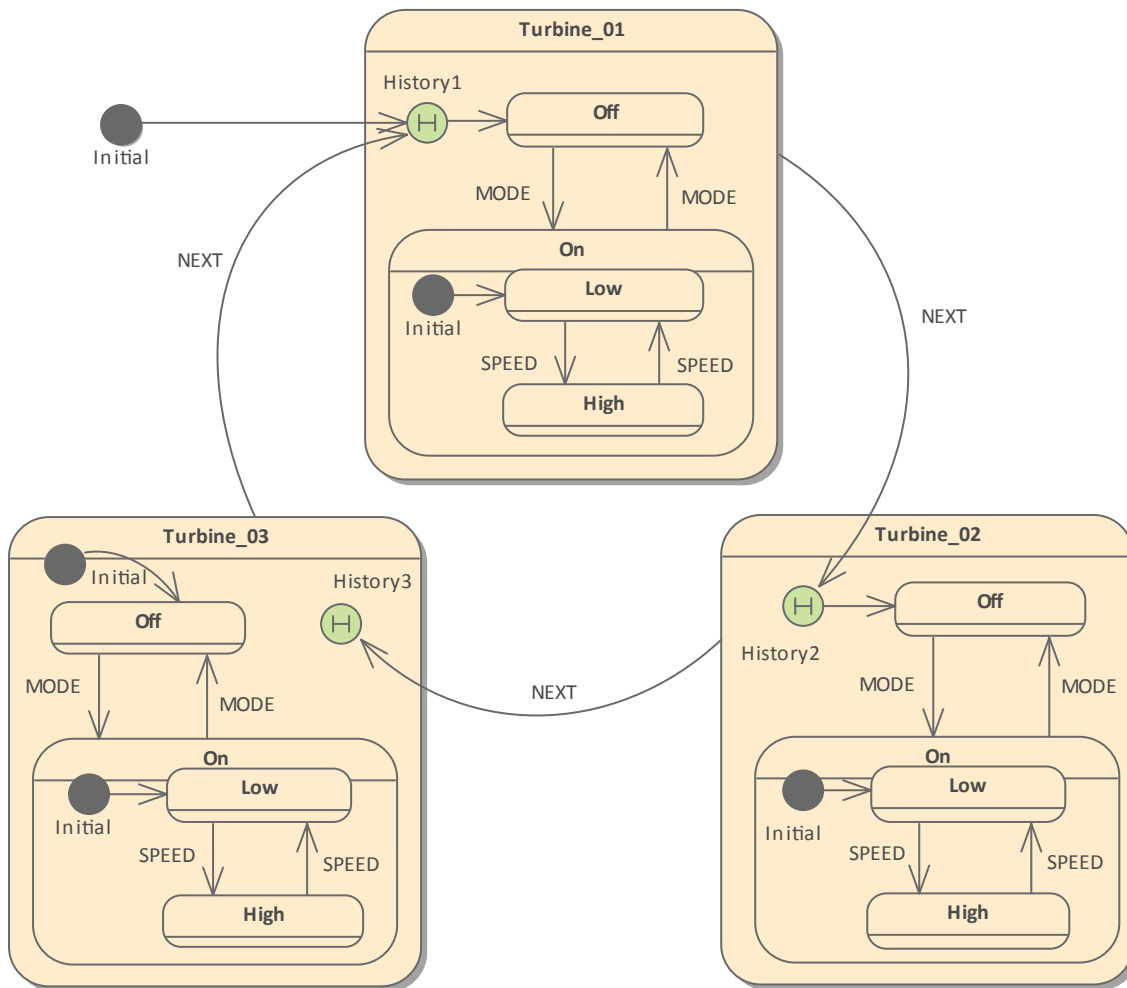
Tip: If you right-click on the History node on the diagram and select the 'Advanced | Deep History' option, you can toggle the type of History Pseudostate between shallow and deep.

First Time Activation of States

After simulation starts, *Turbine_01* and its substate *Off* are activated.

Trigger Sequence: [MODE, SPEED]
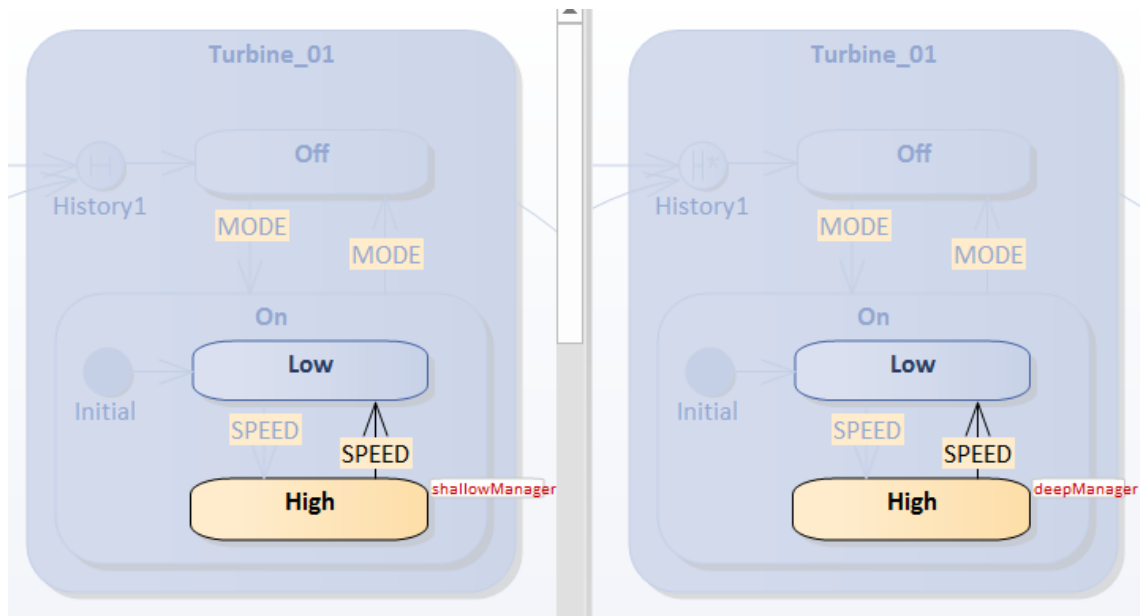
Then the active State configuration includes:

- Turbine_01
- Turbine_01.On
- Turbine_01.On.High

This applies to both *deepManager* and *shallowManager.*
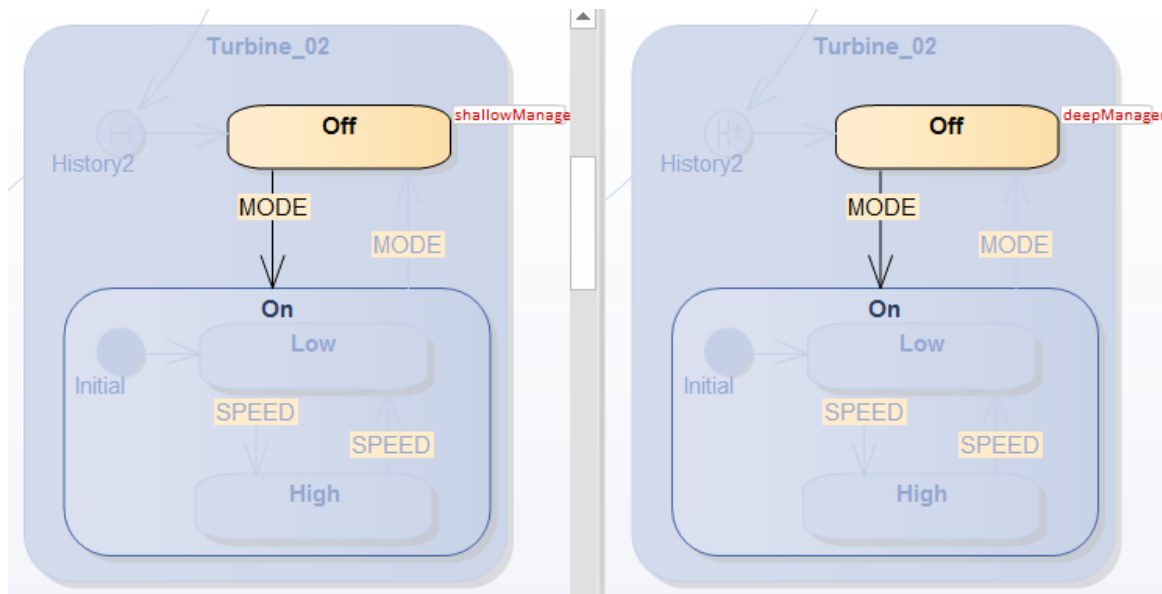
Trigger Sequence: [NEXT]

This trace sequence can be observed from the Simulation window (Simulate > Core > Dynamic Simulation > Open Simulation Window):

01   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_High EXIT

02   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On EXIT

03   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 EXIT

04   shallowManager[ShallowTurbineManager].Turbine_01__TO__History2_105720_61730 Effect

05   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 ENTRY

06   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 DO

07   shallowManager[ShallowTurbineManager].History2_105720__TO__Off_61731 Effect

08   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off ENTRY

09   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off DO

Note: Since *deepManager* has exactly the same trace as *shallowManager*, the trace for *deepManager* is filtered out from this sequence.

We can learn that:

- Exiting a Composite State commences with the innermost State in the active State configuration (see lines 01 - 03 in the trace sequence)
- The Default History Transition is only taken if execution leads to the History node (see line 04) and the State has never been active before (see line 07)

Then the active State configuration includes:

- Turbine_02
- Turbine_02.Off

This applies to both *deepManager* and *shallowManager.*

Trigger Sequence: [NEXT, MODE]
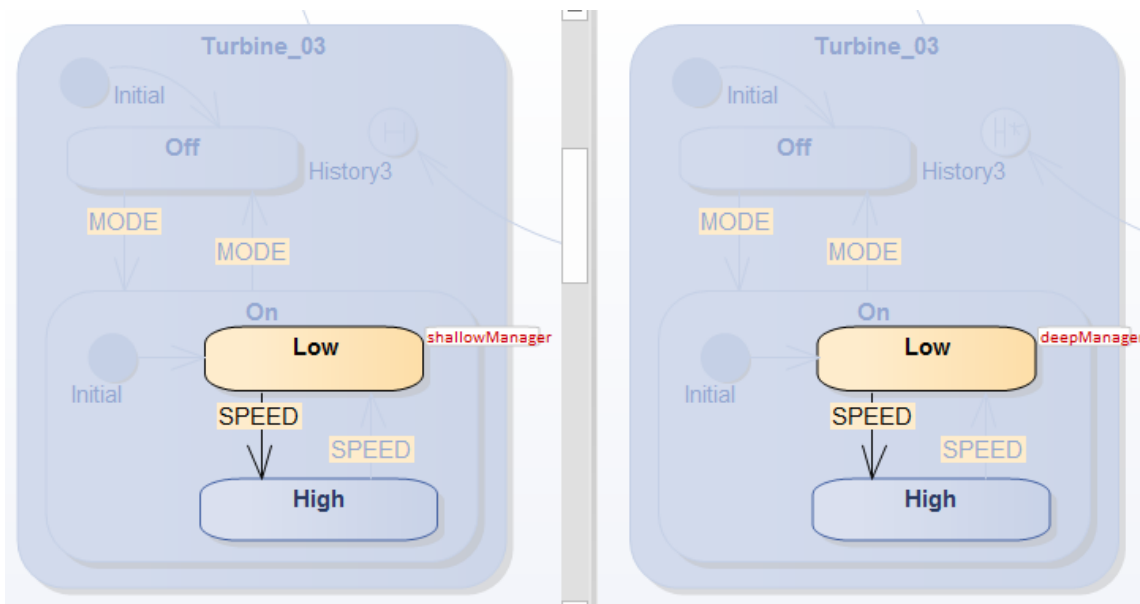
This trace sequence can be observed from the Simulation window:

Trigger [NEXT]

01   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off EXIT

02   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 EXIT

03   shallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effect

04   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRY

05   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO

06   shallowManager[ShallowTurbineManager].Initial_105706__TO__Off_61718 Effect

07   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off ENTRY

08   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_Off DO

Trigger [MODE]

Message omitted...

Note: Since *deepManager* has exactly the same trace as *shallowManager,* the trace for *deepManager* is filtered out from this sequence.

We can learn that:

- Since there is no default History Transition defined for *History3*, the standard default entry of the State is performed; an Initial node is found in the Region contained by *Turbine_03,* so the Transition originating from *Initial* is enabled (see line 06)

Then the active state configuration includes:

- Turbine_03

- Turbine_03.On

- Turbine_03.On.Low

This applies to both *deepManager* and *shallowManager.*

History Entry of States

As a reference, we show the Deep History snapshot of each Turbine after its first activation:

Turbine_01

- Turbine_01.On

- Turbine_01.On.High

Turbine_02

- Turbine_02.Off

Turbine_03

- Turbine_03.On

- Turbine_03.On.Low

When we further Trigger NEXT, Turbine_01 will be activated again.
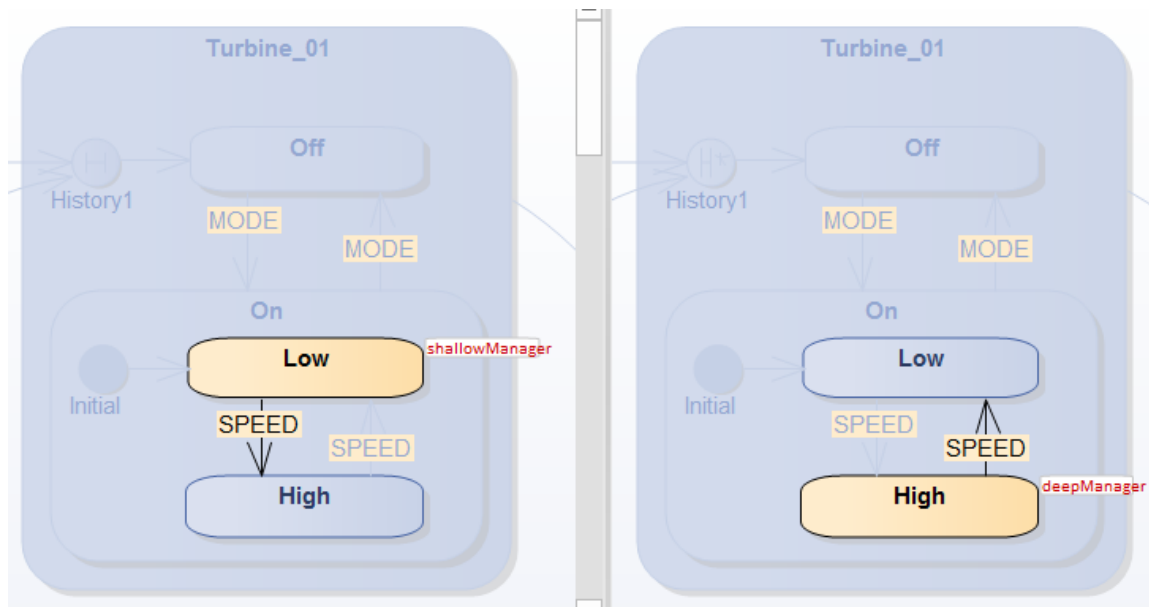
This trace sequence can be observed from the Simulation window:

For shallowManager:

    Trigger [NEXT]

01   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low EXIT

02   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On EXIT

03   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 EXIT

04   shallowManager[ShallowTurbineManager].Turbine_03__TO__History1_105711_61732 Effect

05   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 ENTRY

06   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01 DO

07   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On ENTRY

08   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On DO

09   shallowManager[ShallowTurbineManager].Initial_105721__TO__Low_61729 Effect

10   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low ENTRY

11   shallowManager[ShallowTurbineManager].StateMachine_Turbine_01_On_Low DO

We can learn that:

- The shallowHistory node restores Turbine_01 as far as Turbine_01.On

- Then the Region contained by Composite State Turbine_01.On will be activated by the *Initial* node, which activated at *Low*

For deepManager:

    Trigger [NEXT]

01   deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low EXIT

02   deepManager[DeepTurbineManager].StateMachine_Turbine_03_On EXIT

03   deepManager[DeepTurbineManager].StateMachine_Turbine_03 EXIT

04   deepManager[DeepTurbineManager].Turbine_03__TO__History1_105679_61708 Effect

05   deepManager[DeepTurbineManager].StateMachine_Turbine_01 ENTRY

06   deepManager[DeepTurbineManager].StateMachine_Turbine_01 DO

07   deepManager[DeepTurbineManager].StateMachine_Turbine_01_On ENTRY

08   deepManager[DeepTurbineManager].StateMachine_Turbine_01_On_High ENTRY

We can learn that:

- The *deepHistory* node restores Turbine_01 as far as Turbine_01.On.High

Trigger [NEXT] to exit Turbine_01 and activate Turbine_02

Both *shallowManager* and *deepManager* activate Turbine_02.Off, which is the History snapshot when they exited.

Trigger [NEXT] to exit Turbine_02 and activate Turbine_03

Both *shallowManager* and *deepManager* activate Turbine_03.On.Low. However, the sequences of *shallowManager* and *deepManager* are different.

For *shallowManager*, the *shallowHistory* can only restore as far as Turbine_03.On. Since an *Initial* node is defined in Turbine_03.On, the Transition originating from *Initial* will be enabled and Turbine_03.On.Low is reached.

01   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02_Off EXIT

02   shallowManager[ShallowTurbineManager].StateMachine_Turbine_02 EXIT

03   shallowManager[ShallowTurbineManager].Turbine_02__TO__History3_105713_61725 Effect

04   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 ENTRY

05   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03 DO

06   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On ENTRY

07   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On DO

08   shallowManager[ShallowTurbineManager].Initial_105727__TO__Low_61728 Effect

09   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low ENTRY

10   shallowManager[ShallowTurbineManager].StateMachine_Turbine_03_On_Low DO

For *deepManager*, the *deephistory* can restore as far as Turbine_03.On.Low directly.

01   deepManager[DeepTurbineManager].StateMachine_Turbine_02_Off EXIT

02   deepManager[DeepTurbineManager].StateMachine_Turbine_02 EXIT

03   deepManager[DeepTurbineManager].Turbine_02__TO__History3_105680_61701 Effect

04   deepManager[DeepTurbineManager].StateMachine_Turbine_03 ENTRY

05   deepManager[DeepTurbineManager].StateMachine_Turbine_03 DO

06   deepManager[DeepTurbineManager].StateMachine_Turbine_03_On ENTRY

07   deepManager[DeepTurbineManager].StateMachine_Turbine_03_On_Low ENTRY

Example Executable StateMachine

Example Class Model

This image shows a sample Class model that is used by the StateMachines described in this topic.

TurbineController
+   simulationLength: int
+   tickLength: double

Owns +master

Turbine
+   currentHeat: double
+   heatDisipation: double
+   heatProduction: double
+   heatTolerance: double

+   cool(int): void
+   warm(int): void

+next

Next
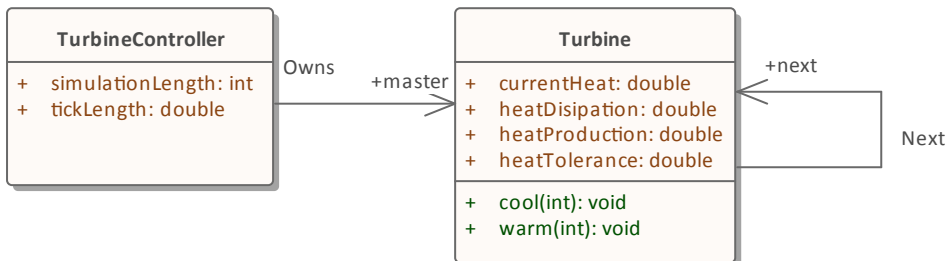
Example StateMachines

These two diagrams show the definitions of two StateMachines. The first references another StateMachine of the same type, while the second drives any instances of the first that exist.

stm RunningState

Standby
TICK /
    cool(1)
SPEEDUP /
    defer();
SPEEDDOWN /
    defer();

Off

Initial

START

SHUTDOWN

ACTIVATE

TICK [currentHeat >= heatTolerance]
/%SEND_EVENT("ACTIVATE",
    CONTEXT_REF(next))%;

On

[Speed] H* History

Low

SPEEDUP

SPEEDDOWN

Medium

SPEEDUP

SPEEDDOWN

High

[Heat]

Initial

Heating
TICK [currentHeat < heatTolerance] /
    warm(1)
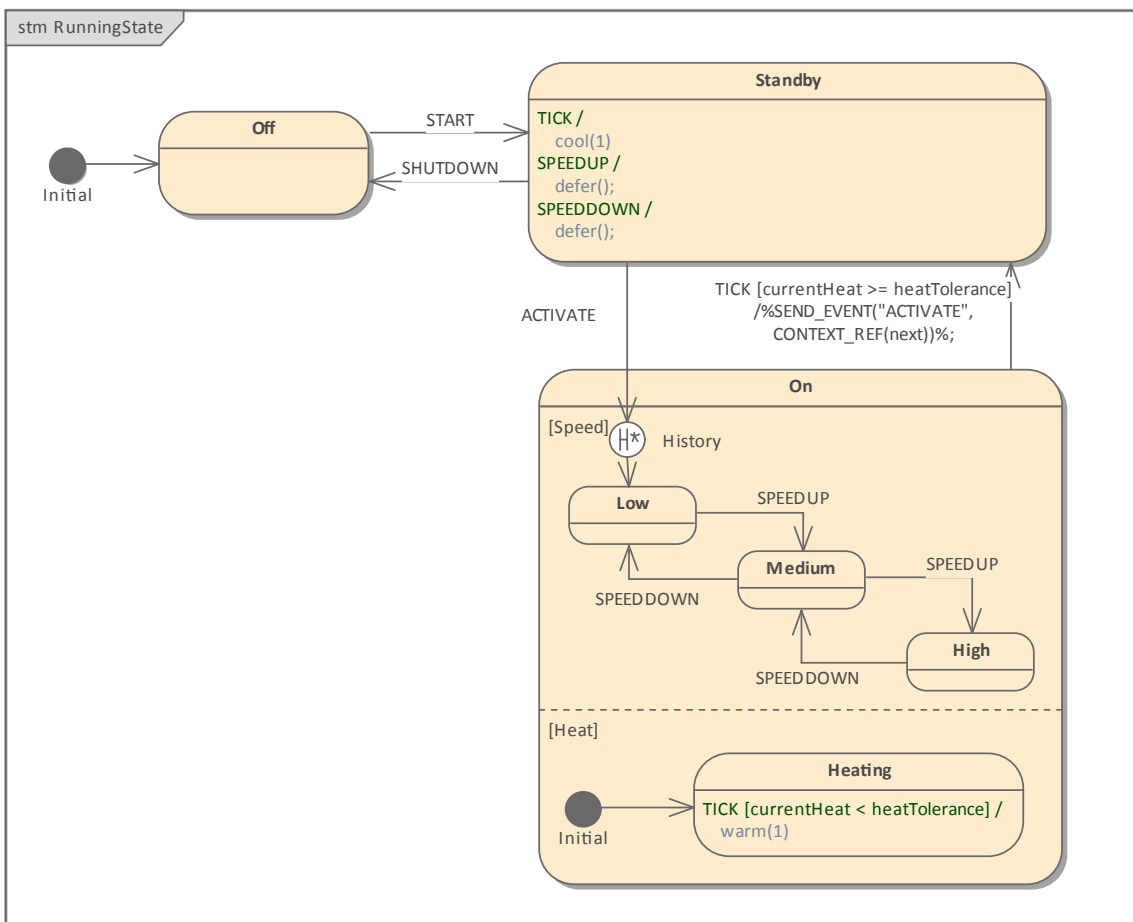
And the top level controller.
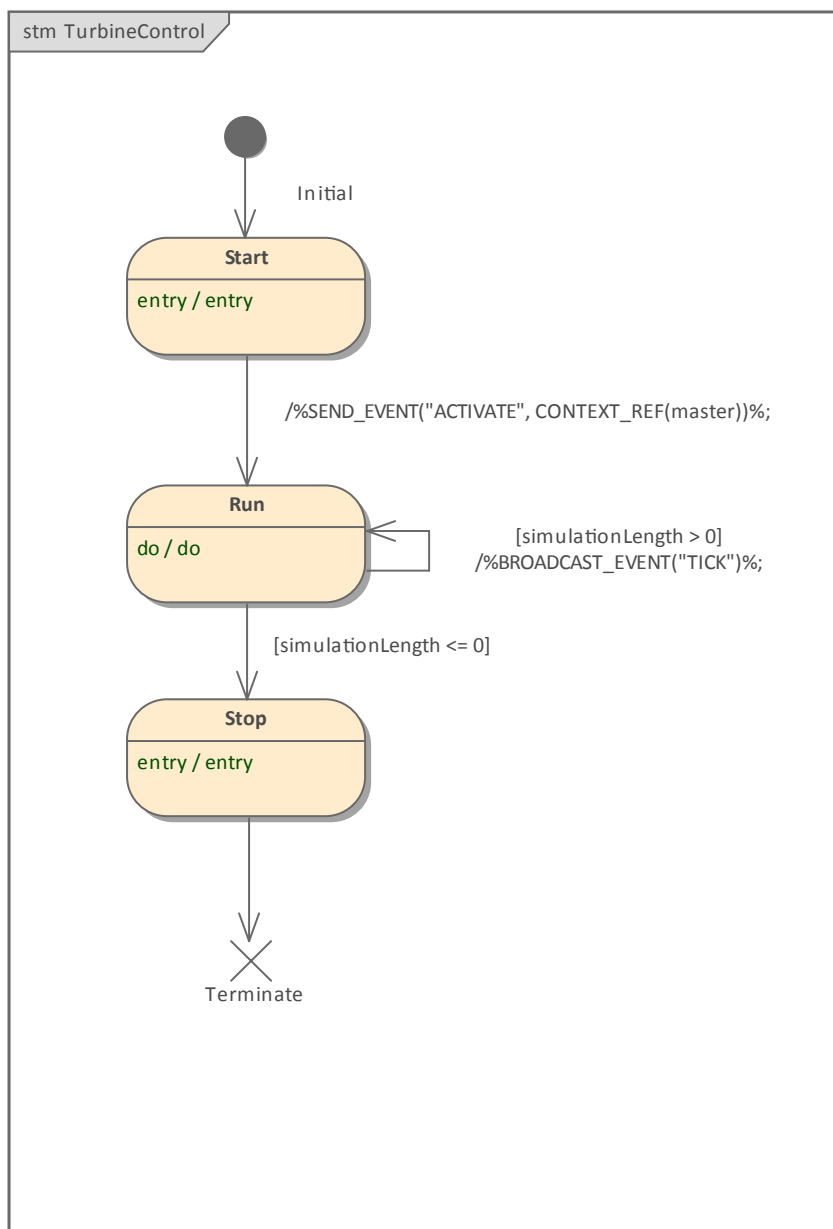
Example Artifacts

From the example Class and StateMachine diagrams, we can create Executable StateMachines as shown here.
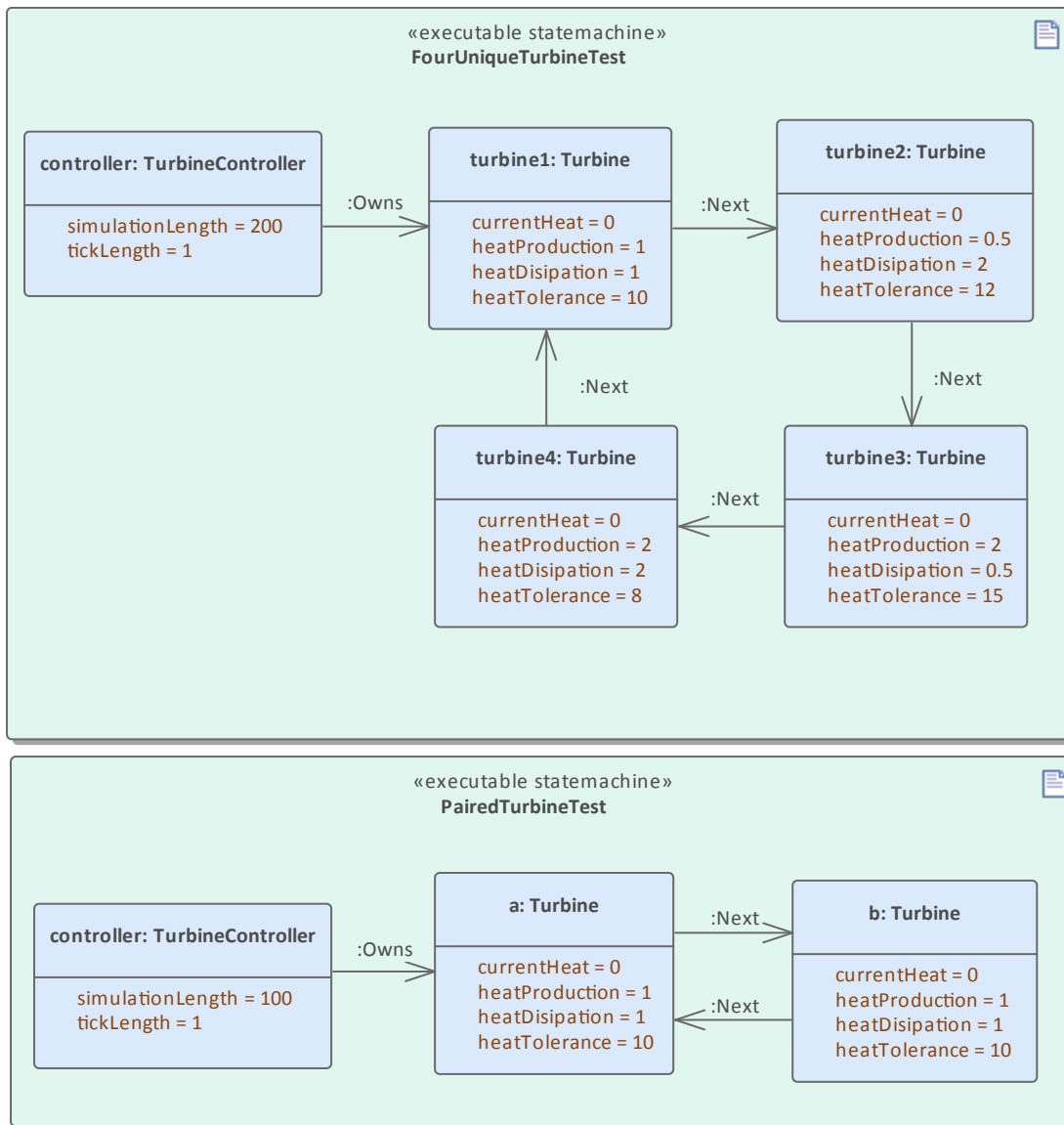
Note how property values have been set for each property, and the links between elements identify the relationships that exist in the Class model.

Simulation Results

When running a simulation, Enterprise Architect will highlight the currently active States in any StateMachines. Where multiple instances of a StateMachine exist, it will also show the names of each instance in that State.
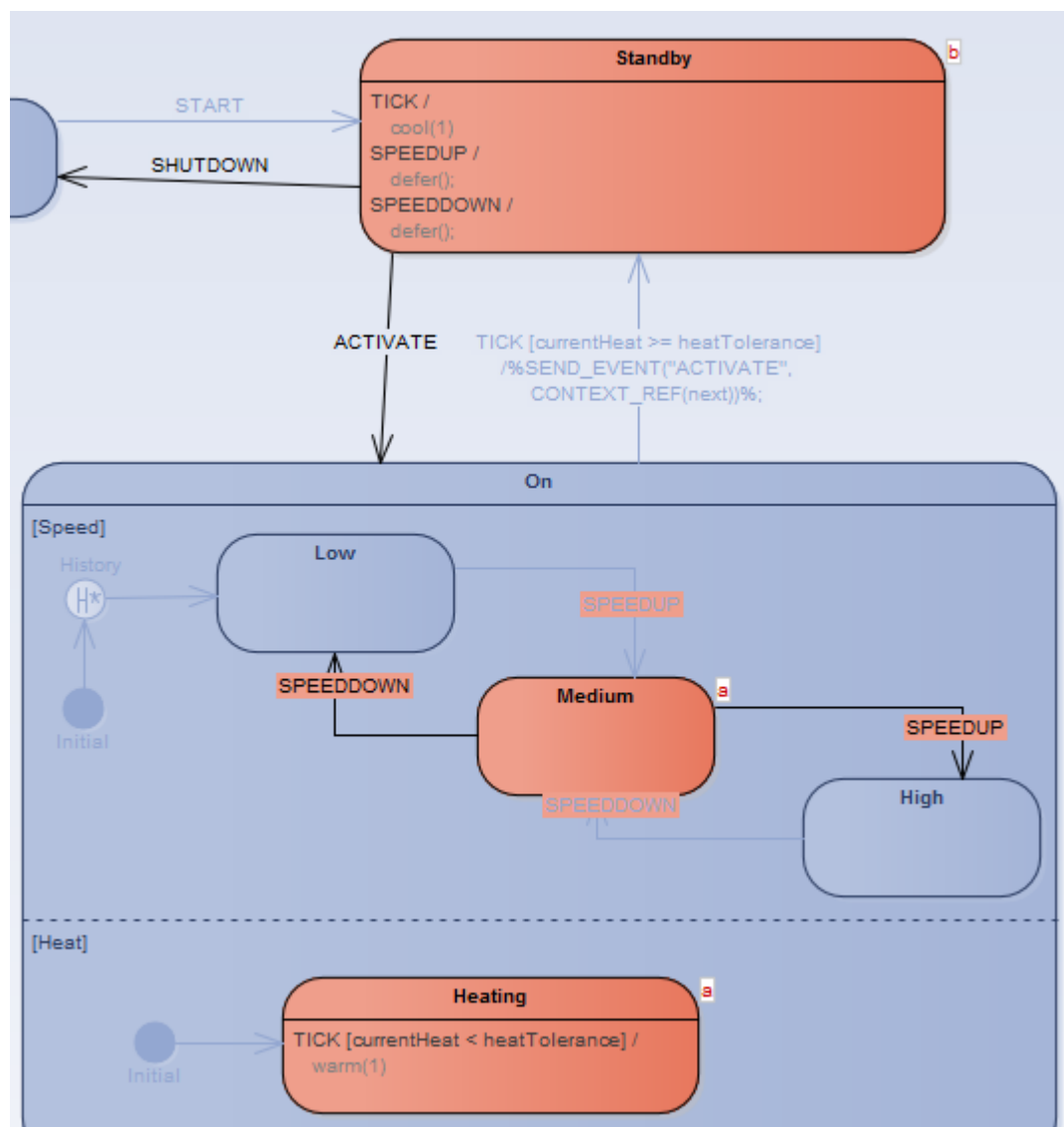
Parametric Simulation using OpenModelica

Enterprise Architect provides integration with OpenModelica to support rapid and robust evaluation of how a SysML model will behave in different circumstances.

This section describes the process of defining a Parametric model, annotating the model with additional information to drive a simulation, and running a simulation to generate a graph.

Introduction to SysML Parametric Models

SysML Parametric models support the engineering analysis of critical system parameters, including the evaluation of key metrics such as performance, reliability and other physical characteristics. These models combine requirements models with system design models, by capturing executable constraints based on complex mathematical relationships. Parametric diagrams are specialized Internal Block diagrams that help you, the modeler, to combine behavior and structure models with engineering analysis models such as performance, reliability, and mass property models.

For further information on the concepts of SysML Parametric models, refer to the official OMG SysML website and its linked sources.

SysMLSimConfiguration Artifact

Enterprise Architect helps you to extend the usefulness of your SysML parametric models by annotating them with extra information that allows the model to be simulated. The resulting model is then generated as a Modelica model that can be solved (simulated) using OpenModelica.

The simulation properties for your model are stored against a Simulation Artifact. This preserves your original model and supports multiple simulations being configured against a single SysML model. The Simulation Artifact can be found on the 'Artifacts' Toolbox page.

User Interface

The user interface for the SysML simulation is described in the *Configure SysML Simulation Window* topic.

OpenModelica Examples

To aid your understanding of how to create and simulate a SysML parametric model, three examples have been provided to illustrate three different domains. These examples and what you are able to learn from them are described in the *SysML Simulation Example*s topic.

Interfacing with OpenModelica

For details on installing OpenModelica and connecting Enterprise Architect to it, see the Help topic covering the platform where Enterprise Architect is installed.

Installation

Platform	Detail
Windows	If Enterprise Architect is installed on a Windows platform, see the *OpenModelica on Windows* Help Topic.
Linux	If Enterprise Architect is installed on a Linux platform, see the *OpenModelica on Linux* Help Topic.

OpenModelica on Windows

When installing OpenModelica for Enterprise Architect operating on a Windows platform, you firstly install the OpenModelica application, then configure the settings in Enterprise Architect to access OpenModelica.

Install OpenModelica

Step	Action
1	Download the OpenModelica Installer from: https://openmodelica.org/download/download-windows
2	Double-click on the OpenModelica installer and follow the 'Wizard' instructions. We recommend that you accept the default path for installation.
3	Check that you can locate the executable omc.exe. For example: C:\OpenModelica1.9.2\bin\omc.exe

Access

Use either of these access paths to display the 'Modelica Solver Path' dialog.

Method	Select
Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager > > Configure Modelica Solver
Other	Double-click on an Artifact with the SysMLSimConfiguration stereotype > > Configure Modelica Solver

Configure the Solver

For Windows, the 'Modelica Solver Path' dialog resembles this:
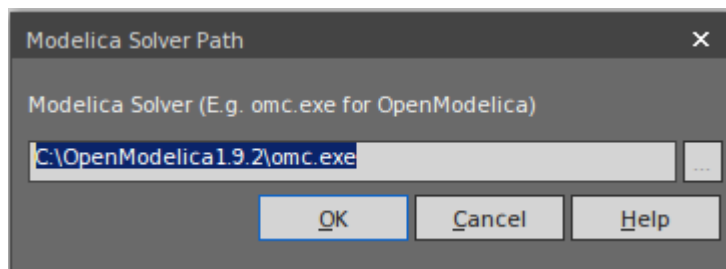
Type or browse for the path to the Modelica solver to use OpenModelica in Enterprise Architect.

OpenModelica on Linux

If Enterprise Architect is installed on Linux it is necessary to operate with OpenModelica installed on the same platform. The OpenModelica Linux installation is publicly documented for Debian and Ubuntu; however, it can also be installed under Linux Mint.

This Help topic provides guidance on:

1. Installation of OpenModelica on:

- Linux Debian / Ubuntu

- Linux Mint

2. Configuring Enterprise Architect to access OpenModelica.

Linux Debian / Ubuntu

To install OpenModelica on a Linux Debian / Ubuntu system refer to the URL:

   https://openmodelica.org/download/download-linux

This provides the instructions for Debian / Ubuntu Packages.

Run these scripts in a terminal:

Step	Action
1	To add OpenModelica to your additional repository list: for deb in deb deb-src; do echo "$deb http://build.openmodelica.org/apt `lsb_release -cs` nightly"; done | sudo tee /etc/apt/sources.list.d/openmodelica.list
2	Import the GPG key used to sign the releases: wget -q http://build.openmodelica.org/apt/openmodelica.asc -O- | sudo apt-key add -
3	Update and install OpenModelica: sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-.* # Installs optional Modelica libraries (most have not been tested with OpenModelica)
4	To check this installation, ensure that you can find the file /usr/bin/omc by, for example, executing this command on the terminal: • ~ $ /usr/bin/omc --version Your installation is successful if the command returns a string resembling this: • OpenModelica 1.13.0~dev-1322-g53a43cf

Linux Mint

To install OpenModelica on Linux Mint, you initially perform an install for Ubuntu and then modify the Linux Mint code name to match the Ubuntu code name.

This is a list of mappings of the Linux Mint code name to the Ubuntu code name (to be used in later steps):

- Linux Mint 17.3 (Rosa) = Ubuntu 14.04 (Trusty): **rosa** = **trusty**

- Linux Mint 18 (Sarah) = Ubuntu 16.04 (Xenial): **sarah** = **xenial**

- Linux Mint 18.1 (Serena) = Ubuntu 16.04 (Xenial): **serena** = **xenial**

- Linux Mint 18.2 (Sonya) = Ubuntu 16.04 (Xenial): **sonya** = **xenial**

- Linux Mint 18.3 (Sylvia) = Ubuntu 16.04 (Xenial): **sylvia** = **xenial**

- Linux Mint 19 (Tara) = Ubuntu 18.04 (Bionic): **tara** = **bionic**

Click Here for a full list of Linux Mint History and the mappings with Ubuntu.

Step	Action
1	Run this script in a terminal: for deb in deb deb-src; do echo "$deb http://build.openmodelica.org/apt \`lsb_release -cs\` nightly"; done | sudo tee /etc/apt/sources.list.d/openmodelica.list
2	To change the repository URL in Linux Mint: • On the Linux Mint main screen select: 'Menu | Search Bar | Software Sources (type in password) | Additional repositories | Select 'Openmodelica' | Edit URL' • Change the Linux Mint name (for example, *rosa)* to the corresponding Ubuntu name (for example, *trusty*) as in the list at the top of this table; that is: deb http://build.openmodelica.org/apt **rosa** nightly deb http://build.openmodelica.org/apt **trusty** nightly • Click on the OK button
3	• Select 'Openmodelica(Sources)'| Edit URL • Change the Linux Mint name according to the list at the top of the table For example, change the Linux Mint name *rosa* to the corresponding Ubuntu name *trusty* • Click on the OK button
4	To update and install OpenModelica, run these scripts in a terminal: sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-.* # Installs optional Modelica libraries (most have not been tested with OpenModelica)

Access

Use either of these access paths to display the 'Modelica Solver Path' dialog, to configure the solver.

Method	Select
Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager > > Configure Modelica Solver
Other	Double-click on an Artifact with the SysMLSimConfiguration stereotype > > Configure Modelica Solver

Configure the Solver

The 'Modelica Solver Path' dialog resembles this:
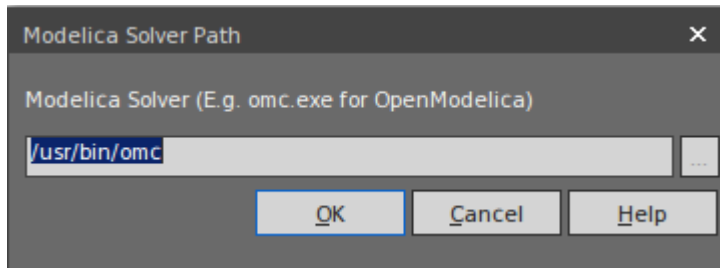
Type in or browse for the path to the Modelica solver to use.

Creating a Parametric Model

In this topic we discuss how you might develop SysML model elements for simulation (assuming existing knowledge of SysML modeling), configure these elements in the Configure SysML Simulation window, and observe the results of a simulation under some of the different definitions and modeling approaches. The points are illustrated by snapshots of diagrams and screens from the SysML Simulation examples provided in this chapter.

When creating a Parametric Model, you can apply one of three approaches to defining Constraint Equations:

- Defining inline Constraint Equations on a Block element
- Creating re-usable Constraint Blocks, and
- Using connected constraint properties

You would also take into consideration:

- Flows in physical interactions
- Default Values and Initial Values
- Simulation Functions
- Value Allocation, and
- Packages and Imports

Access

Ribbon	Simulate > SysMLSim > Manage > SysMLSim Configuration Manager

Defining inline Constraint Equations on a Block

Defining constraints directly in a Block is straightforward and is the easiest way to define constraint equations.

In this figure, constraint 'f = m * a' is defined in a Block element.
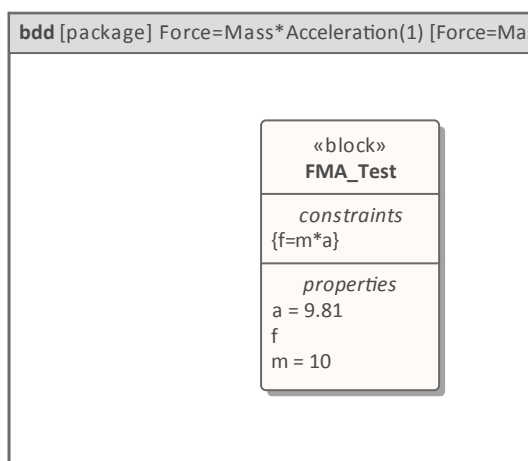
Tip: You can define multiple constraints in one Block.

1. Create a SysMLSim Configuration Artifact 'Force=Mass*Acceleration(1)' and point it to the Package 'FMA_Test'.

2. For 'FMA_Test', in the 'Value' column set 'SysMLSimModel'.

3. For Parts 'a', 'm' and 'f', in the 'Value' column set 'a' and 'm' to 'SimConstant' and (optionally) set 'f' to 'SimVariable'.

4. On the 'Simulation' tab, in the 'Properties to Plot' panel, select the checkbox against 'f'.

5. Click on the Solve button to run the simulation.
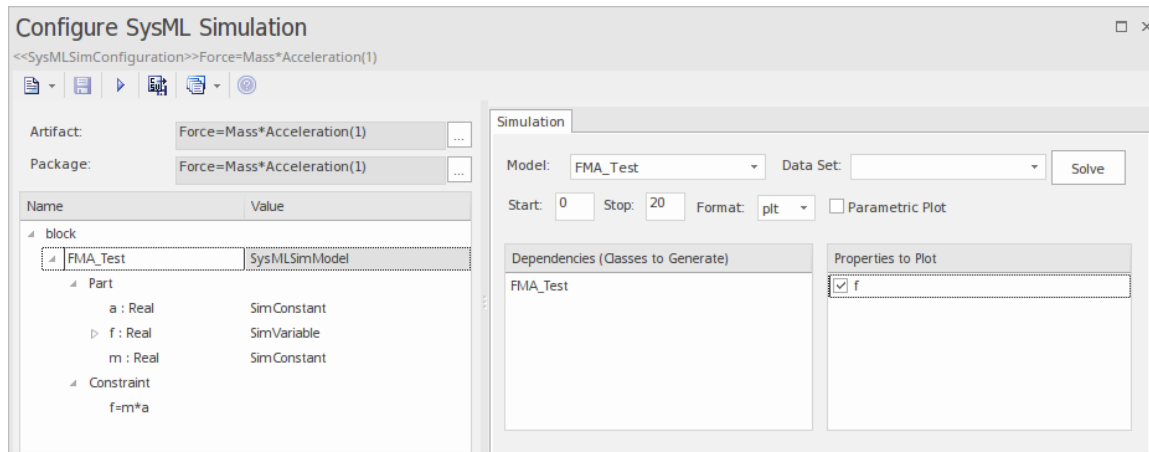
A chart should be plotted with f = 98.1 (which comes from 10 * 9.81).

Connected Constraint Properties

In SysML, constraint properties existing in Constraint Blocks can be used to provide greater flexibility in defining constraints.

In this figure, Constraint Block 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.
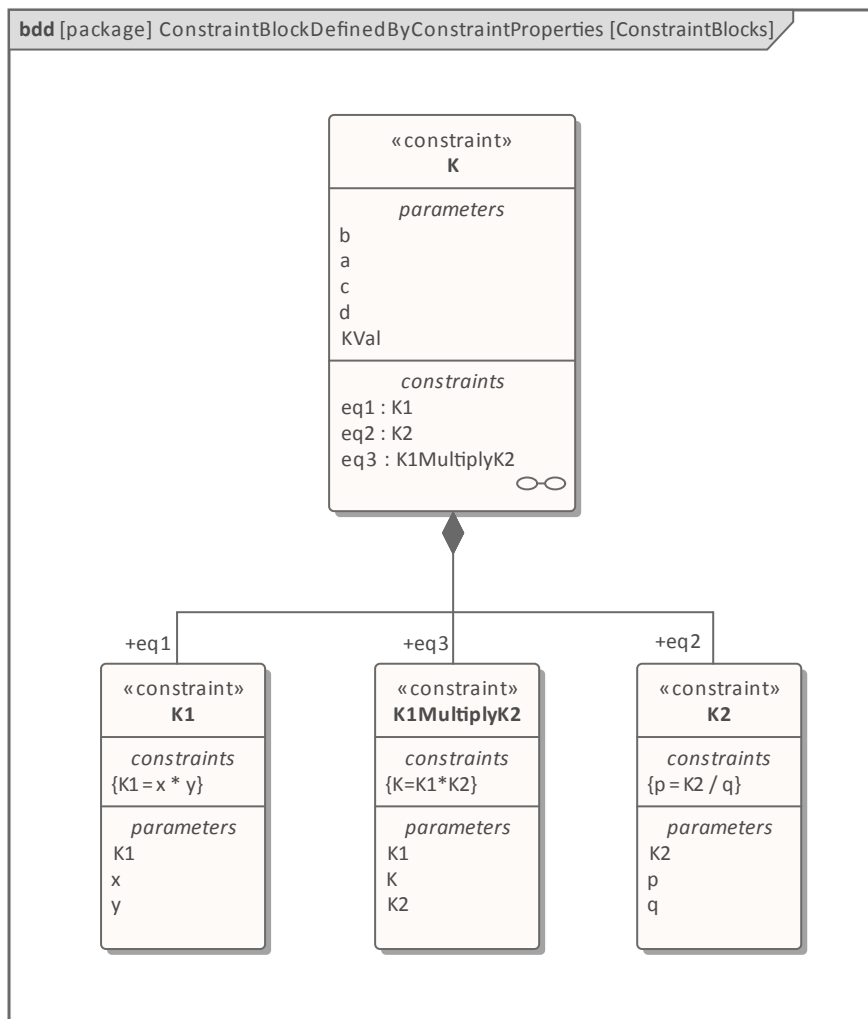
Create a Parametric diagram in Constraint Block 'K' and connect the parameters to the constraint properties with Binding connectors, as shown:
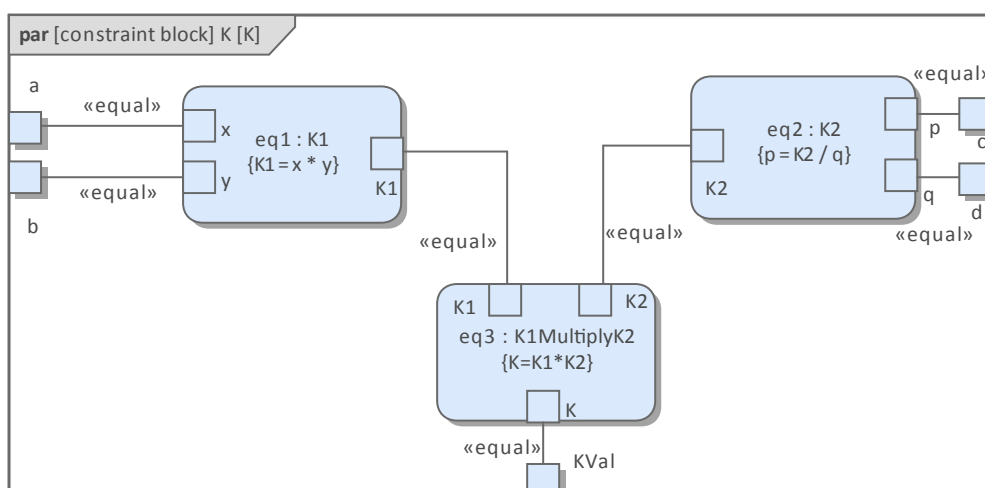
- Create a model MyBlock with five Properties (Parts)
- Create a constraint property 'eq' for MyBlock and show the parameters
- Bind the properties to the parameters
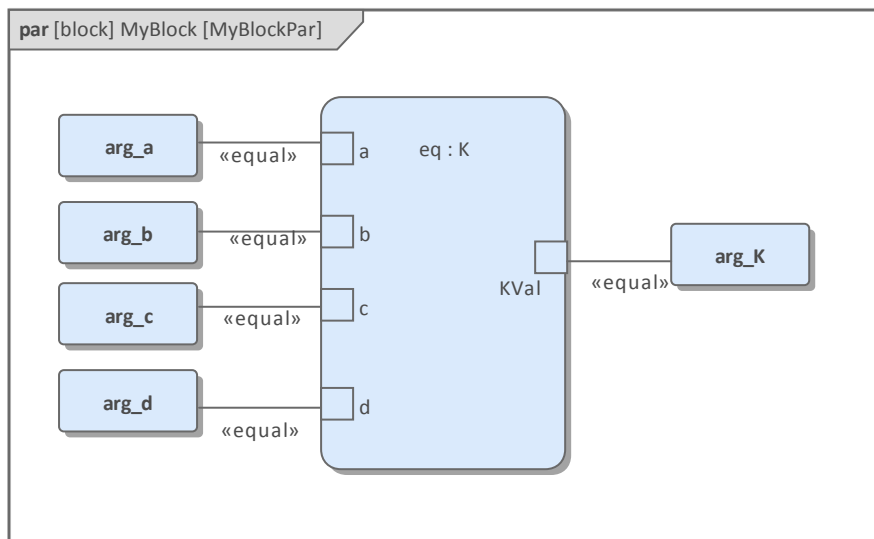
- Provide values (arg_a = 2, arg_b = 3, arg_c = 4, arg_d = 5) in a data set
- In the 'Configure SysML Simulation' dialog, set 'Model' to 'MyBlock' and 'Data Set' to 'DataSet_1'
- In the 'Properties to Plot' panel, select the checkbox against 'arg_K'
- Click on the Solve button to run the simulation
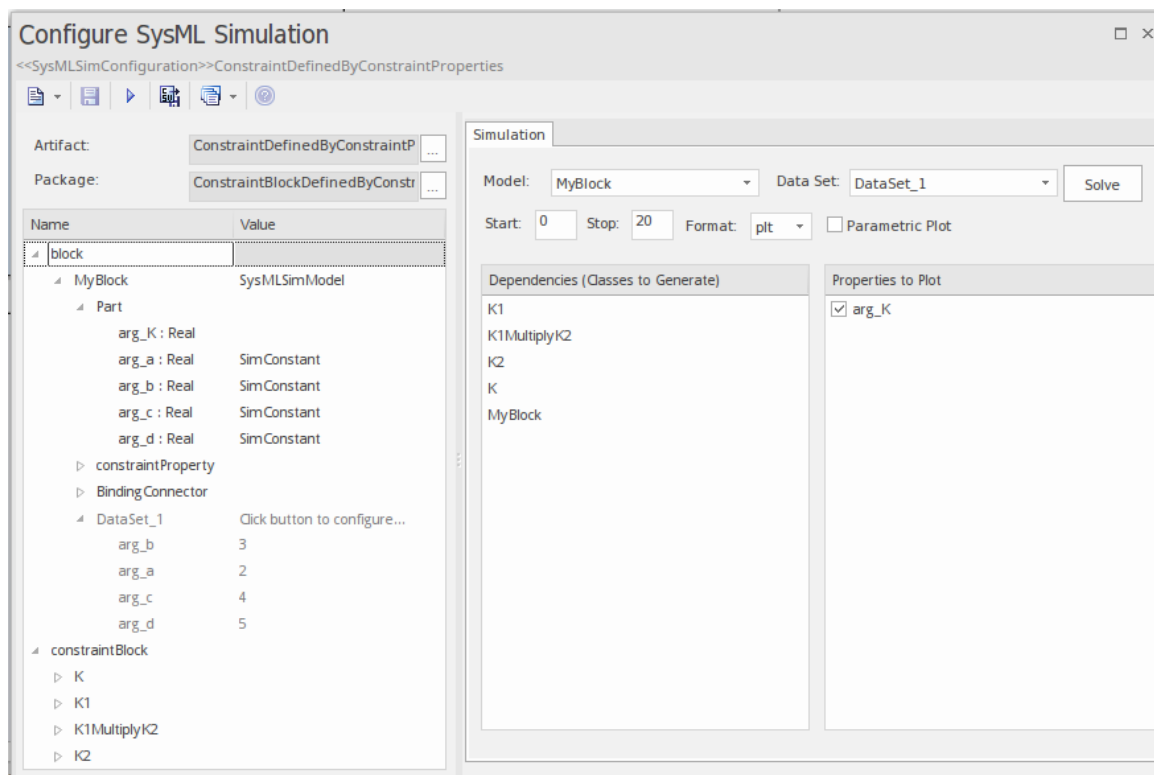
The result 120 (calculated as 2 * 3 * 4 * 5) will be computed and plotted. This is the same as when we do an expansion with pen and paper: K = K1 * K2 = (x*y) * (p*q), then bind with the values (2 * 3) * (4 * 5); we get 120.

What is interesting here is that we intentionally define K2's equation to be 'p = K2 / q' and this example still works.

We can easily solve K2 to be p * q in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the Enterprise Architect SysMLSim can still get it right.

In summary, the example shows you how to define a Constraint Block with greater flexibility by constructing the

constraint properties. Although we demonstrated only one layer down into the Constraint Block, this mechanism could work on complex models for an arbitrary level of use.

Creating Reuseable Constraint Blocks

If one equation is commonly used in many Blocks, a Constraint Block can be created for use as a constraint property in each Block. These are the changes we make, based on the previous example:

- Create a Constraint Block element 'F_Formula' with three parameters 'a', 'm' and 'f', and a constraint 'f = m * a'
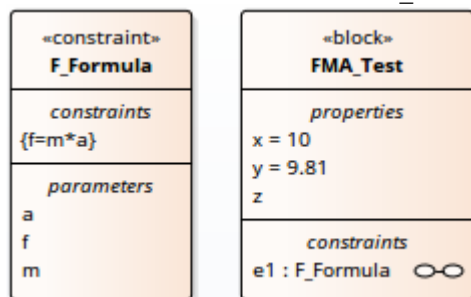
Tip: Primitive type 'Real' will be applied if property types are empty

- Create a Block 'FMA_Test' with three properties 'x', 'y' and 'z', and give 'x' and 'y' the default values '10' and '9.81' respectively

- Create a Parametric diagram in 'FMA_Test', showing the properties 'x', 'y' and 'z'

- Create a Constraint Property 'e1' typed to 'F_Formula' and show the parameters

- Draw Binding connectors between 'x—m', 'y—a', and 'f—z' as shown:
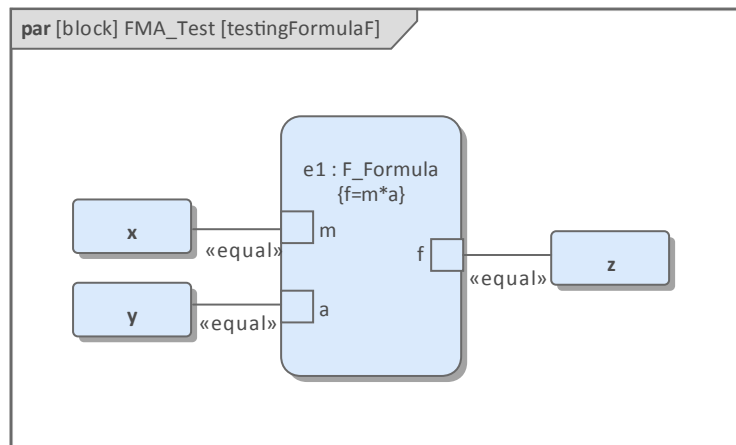
- Create a SysMLSimConfiguration Artifact element and configure it as shown in the dialog illustration:
  - In the 'Value' column, set 'FMA_Test' to 'SysMLSimModel'
  - In the 'Value' column, set 'x' and 'y' to 'SimConstant'
  - In the 'Properties to Plot' panel select the checkbox against 'Z'
  - Click on the Solve button to run the simulation
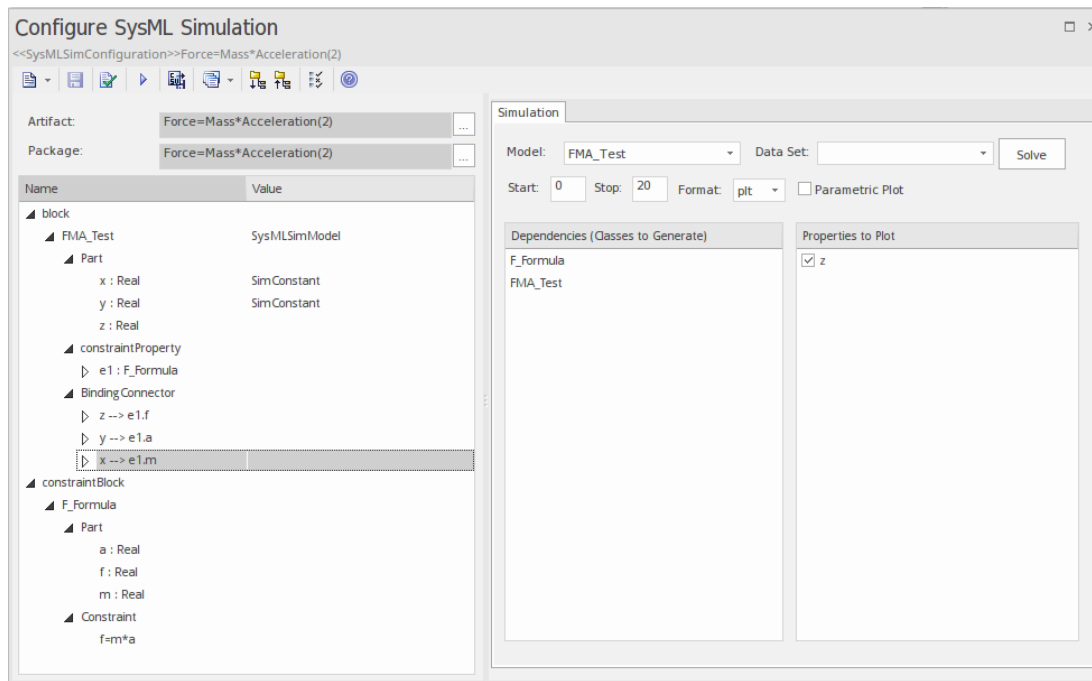
A chart should be plotted with f = 98.1 (which comes from 10 * 9.81).

Flows in Physical Interactions

When modeling for physical interaction, exchanges of conserved physical substances such as electrical current, force, torque and flow rate should be modeled as flows, and the flow variables should be set to the attribute 'isConserved'.

Two different types of coupling are established by connections, depending on whether the flow properties are potential (default) or flow (conserved):

- Equality coupling, for potential (also called effort) properties
- Sum-to-zero coupling, for flow (conserved) properties; for example, according to Kirchoff's Current Law in the electrical domain, conservation of charge makes all charge flows into a point sum to zero

In the generated Modelica code of the 'ElectricalCircuit' example:

```
connector ChargePort
    flow Current i;        //flow keyword will be generated if 'isConserved' = true
    Voltage v;
end ChargePort;

model Circuit
    Source source;
    Resistor resistor;
    Ground ground;
equation
    connect(source.p, resistor.n);
    connect(ground.p, source.n);
```

```
        connect(resistor.p, source.n);
    end Circuit;
```

Each connect equation is actually expanded to two equations (there are two properties defined in ChargePort), one for equality coupling, the other for sum-to-zero coupling:

```
    source.p.v = resistor.n.v;
```

```
    source.p.i + resistor.n.i = 0;
```

Default Value and Initial Values

If initial values are defined in SysML property elements ('Properties' dialog > 'Property' page > 'Initial' field), they can be loaded as the default value for a SimConstant or the initial value for a SimVariable.

In this Pendulum example, we have provided initial values for properties 'g', 'L', 'm', 'PI', 'x' and 'y', as seen on the left hand side of the figure. Since 'PI' (the mathematical constant), 'm' (mass of the Pendulum), 'g' (Gravity factor) and 'L' (Length of Pendulum) do not change during simulation, set them as 'SimConstant'.
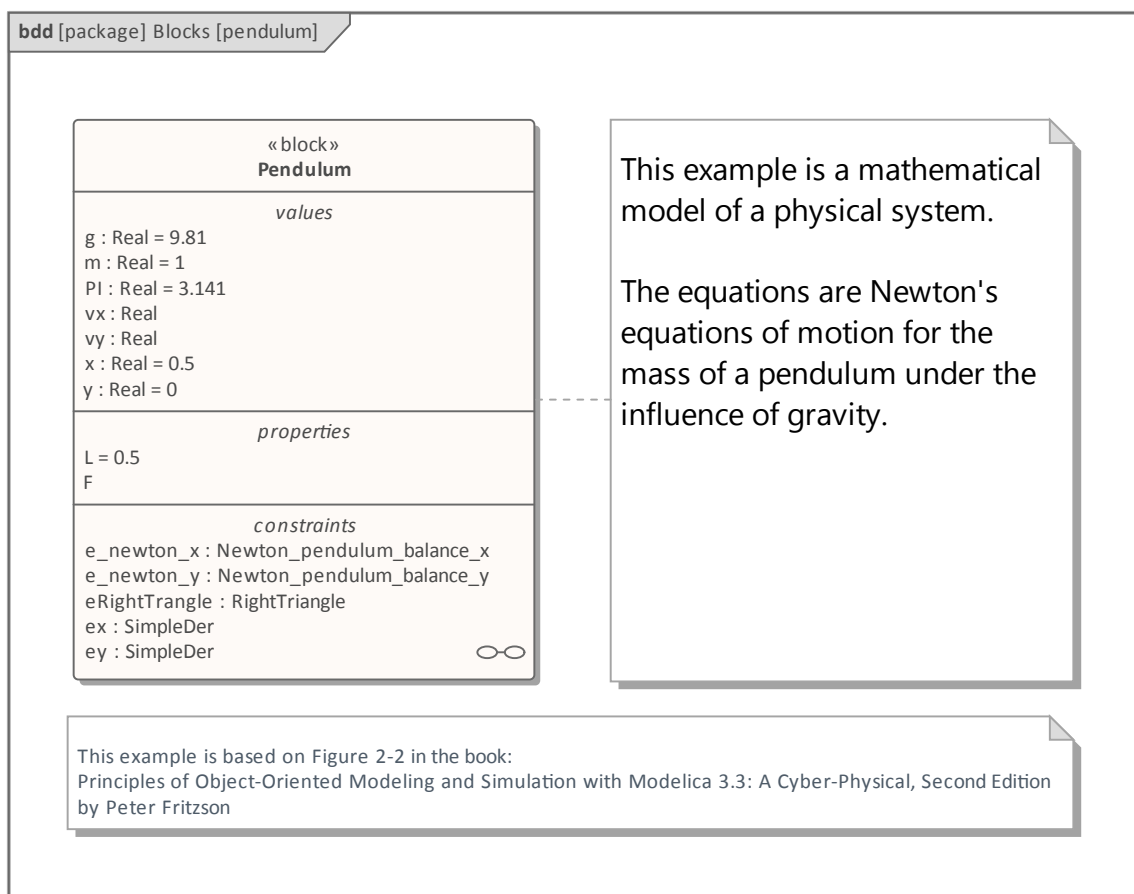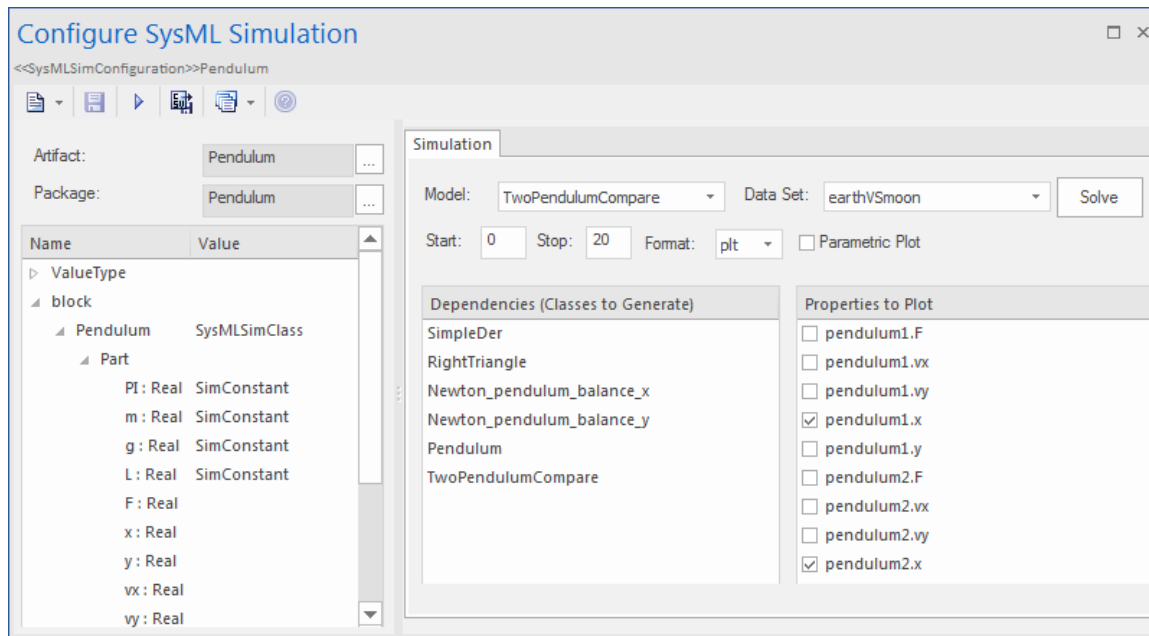
The generated modelica code resembles this:

```
class Pendulum
        parameter Real PI = 3.141;
        parameter Real m = 1;
        parameter Real g = 9.81;
        parameter Real L = 0.5;
        Real F;
        Real x (start=0.5);
        Real y (start=0);
        Real vx;
        Real vy;
        ......
    equation
    ......
    end Pendulum;
```

- Properties 'PI', 'm', 'g' and 'L' are constant, and are generated as a declaration equation
- Properties 'x' and 'y' are variable; their starting values are 0.5 and 0 respectively, and the initial values are generated as modifications

Simulation Functions

A Simulation function is a powerful tool for writing complex logic, and is easy to use for constraints. This section describes a function from the TankPI example.

In the Constraint Block 'Q_OutFlow', a function 'LimitValue' is defined and used in the constraint.

```
bdd [package] constraints [SimFunctions]

        ┌─────────────────────────────────────────┐
        │             «constraint»                │
        │              Q_OutFlow                  │
        ├─────────────────────────────────────────┤
        │  «SimFunction»                          │
        │  +  LimitValue(double, double, double, *double): int │
        ├─────────────────────────────────────────┤
        │              constraints                │
        │  {a=LimitValue(min, max, -b*c)}         │
        ├─────────────────────────────────────────┤
        │              parameters                 │
        │  a                                      │
        │  b                                      │
        │  c                                      │
        │  max                                    │
        │  min                                    │
        └─────────────────────────────────────────┘
```
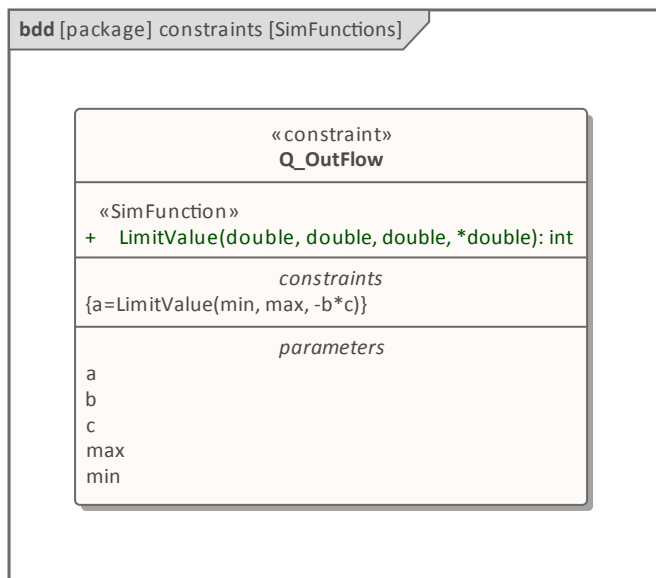
- On a Block or Constraint Block, create an operation ('LimitValue' in this example) and open the 'Operations' tab of the Features window

- Give the operation the stereotype 'SimFunction'

- Define the parameters and set the direction to 'in/out'

   Tips: Multiple parameters could be defined as 'out', and the caller retrieves the value in format of:

   (out1, out2, out3) = function_name(in1, in2, in3, in4, ...);   //Equation form

   (out1, out2, out3) := function_name(in1, in2, in3, in4, ...);   //Statement form

- Define the function body in the text field of the 'Code' tab of the operation Properties window, as shown:

   pLim :=
           if p > pMax then
               pMax
           else if p < pMin then
               pMin
           else
               p;

When generating code, Enterprise Architect will collect all the operations stereotyped as 'SimFunction' defined in Constraint Blocks and Blocks, then generate code resembling this:

   function LimitValue

      input Real pMin;

      input Real pMax;

      input Real p;

      output Real pLim;

         algorithm

            pLim :=

               if p > pMax then

                   pMax

        else if p < pMin then

            pMin

        else

          p;

  end LimitValue;

Value Allocation

This figure shows a simple model called 'Force=Mass*Acceleration'.
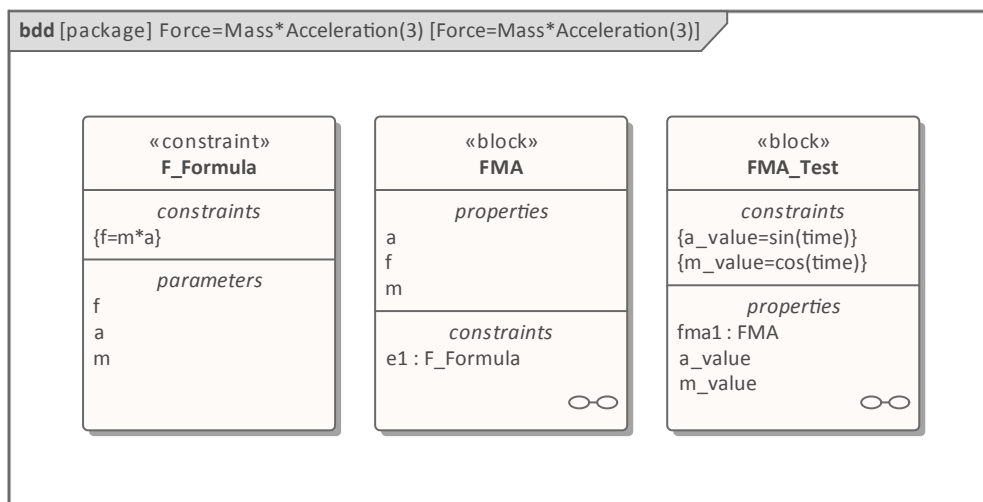
- A Block 'FMA' is modeled with properties 'a', 'f', and 'm' and a constraintProperty 'e1', typed to Constraint Block 'F_Formula'

- The Block 'FMA' does not have any initial value set on its properties, and the properties 'a', 'f' and 'm' are all variable, so their value change depends on the environment in which they are simulated

- Create a Block 'FMA_Test' as a SysMLSimModel and add the property 'fma1' to test the behavior of Block 'FMA'

- Constraint 'a_value' to be 'sin(time)'

- Constraint 'm_value' to be 'cos(time)'

- Draw Allocation connectors to allocate values from environment to the model 'FMA'
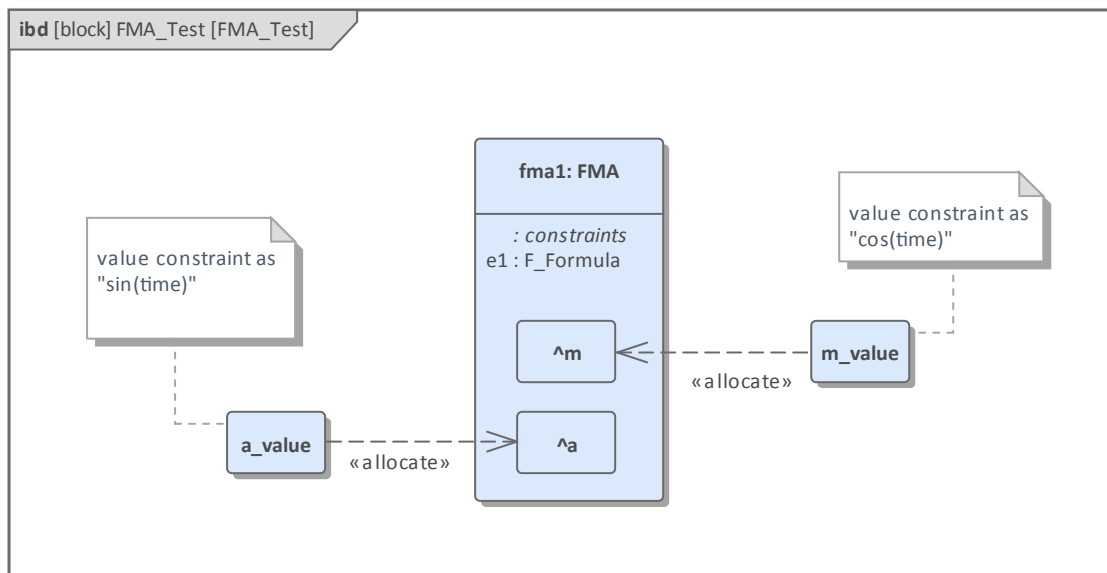
- Select the 'Properties to Plot' checkboxes against 'fma1.a', 'fma1.m' and 'fma1.f'
- Click on the Solve button to simulate the model
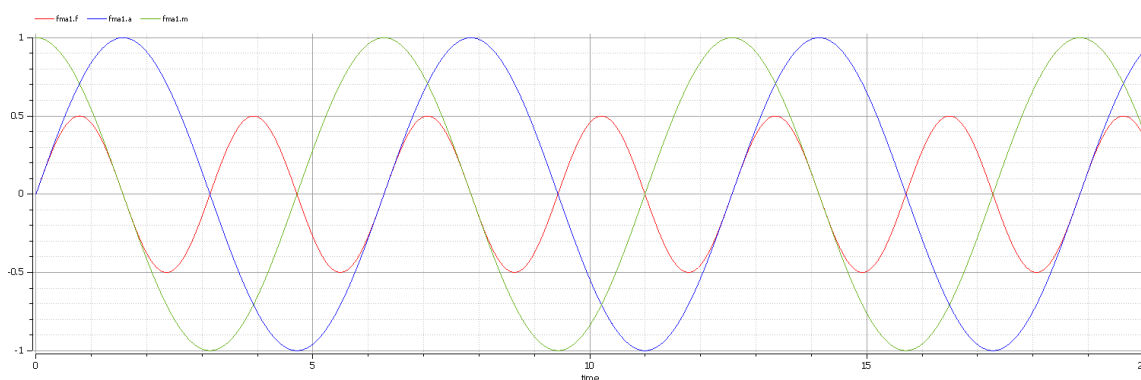
Packages and Imports

The SysMLSimConfiguration Artifact collects the elements (such as Blocks, Constraint Blocks and Value Types) of a Package. If the simulation depends on elements not owned by this Package, such as Reusable libraries, Enterprise Architect provides an Import connector between Package elements to meet this requirement.

In the Electrical Circuit example, the Artifact is configured to the Package 'ElectricalCircuit', which contains almost all of the elements needed for simulation. However, some properties are typed to value types such as 'Voltage', 'Current' and 'Resistance', which are commonly used in multiple SysML models and are therefore placed in a Package called 'CommonlyUsedTypes' outside the individual SysML models. If you import this Package using an Import connector, all the elements in the imported Package will appear in the SysMLSim Configuration Manager.

ElectricalCircuit

+ ElectricalCircuit
+ ChargePort
+ Circuit
+Ground
+ GroundConstraint
+ Resistor
+ ResistorConstraint
+ Source
+ SourceConstraint
+ TwoPinComponent
+ TwoPinComponentConstraint

(from Modelica Examples)

«import»

CommonlyUsedTypes

+ Current
+ Resistance
+ Voltage

(from Modelica Examples)
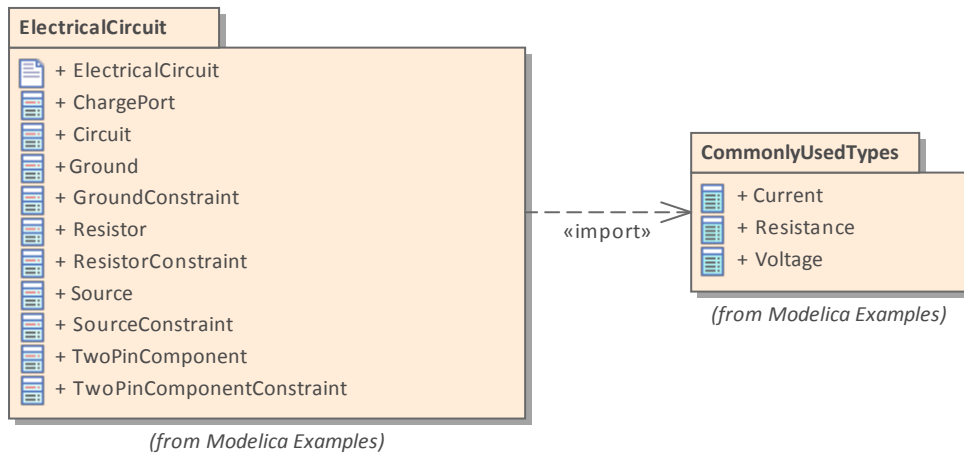
Configure SysML Simulation Window

The Configure SysML Simulation window is the interface through which you can provide run-time parameters for executing the simulation of a SysML model. The simulation is based on a simulation configuration defined in a SysMLSimConfiguration Artifact element.
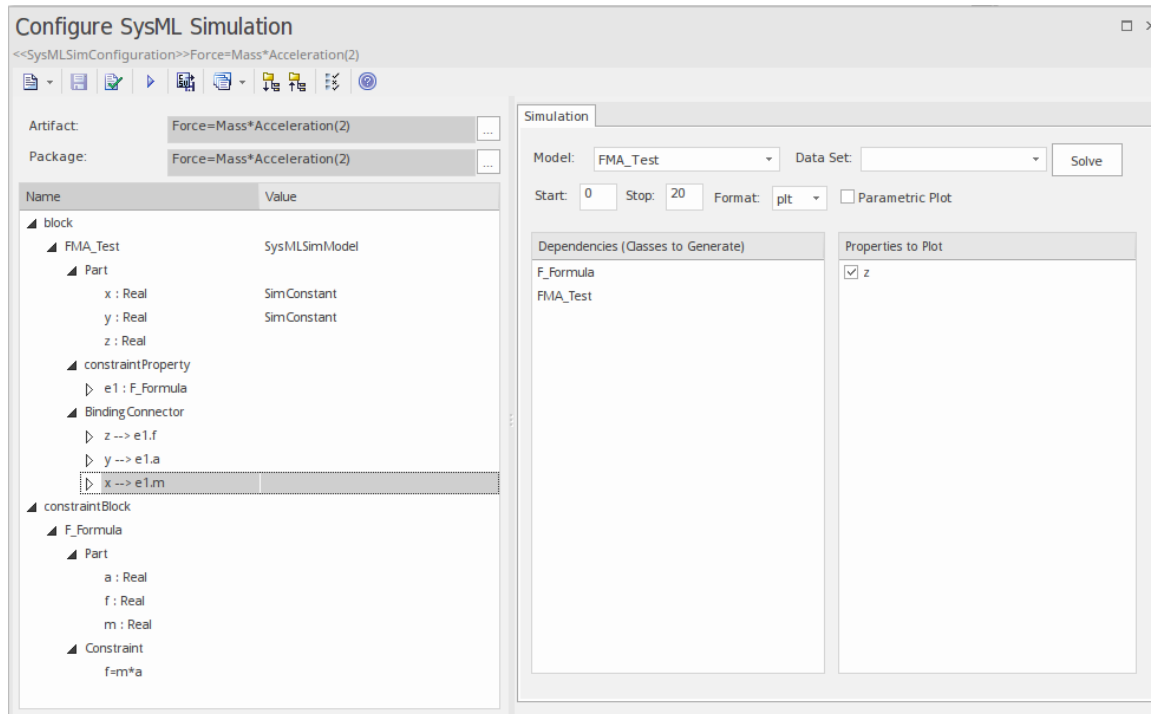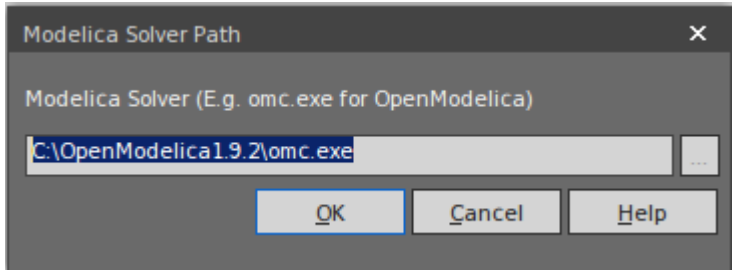
Access

Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager
Other	Double-click on an Artifact with the SysMLSimConfiguration stereotype.

Toolbar Options

Option	Description
	Click on the drop-down arrow and select from these options: • Select Artifact — Select and load an existing configuration from an Artifact with the SysMLSimConfiguration stereotype (if one has not already been selected) • Create Artifact — Create a new SysMLSimConfiguration or select and load an existing configuration artifact • Select Package — Select a Package to scan for SysML elements to configure for simulation • Reload — Reload the Configuration Manager with changes to the current

	Package
	• Configure Modelica Solver — Display the 'Modelica Solver Path' dialog, in which you type or browse for the path to the Modelica solver to use **Modelica Solver Path** ✕ Modelica Solver (E.g. omc.exe for OpenModelica) C:\OpenModelica1.9.2\omc.exe [...] [OK] [Cancel] [Help]
🖫	Click on this button to save the configuration to the current Artifact.
▷	Click on this button to generate, compile and run the current configuration, and display the results.
▦	After simulation, the result file is generated in either plt, mat or csv format. That is, with the filename: • ModelName_res.plt • ModelName_res.mat or • ModelName_res.csv Click on this button to specify a directory into which Enterprise Architect will copy the result file.
▤ ▾	Click on this button to select from these options: • Generate Modelica Code — Generate the code without compiling or running it • Open Modelica Simulation Directory — Open the directory into which Modelica code will be generated • Edit Modelica Templates — Customize the code generated for Modelica, using the Code Template Editor

Simulation Artifact and Model Selection

Field	Action
Artifact	Click on the [...] icon and either browse for and select an existing SysMLSimConfiguration Artifact, or create a new Artifact.
Package	If you have specified an existing SysMLSimConfiguration Artifact, this field defaults to the Package containing the SysML model associated with that Artifact. Otherwise, click on the [...] icon and browse for and select the Package containing the SysML model to configure for simulation. You must specify (or create) the Artifact before selecting the Package.

Package Objects

This table discusses the types of object from the SysML model that will be listed under the 'Name' column in the Configure SysML Simulation window, to be processed in the simulation. Each object type expands to list the named objects of that type, and the properties of each object that require configuration in the 'Value' column.

Many levels of the object types, names and properties do not require configuration, so the corresponding 'Value' field does not accept input. Where input is appropriate and accepted, a drop-down arrow displays at the right end of the field; when you click on this a short list of possible values displays for selection. Certain values (such as 'SimVariable' for a

Part) add further layers of parameters and properties, where you click on the ⟨ ... ⟩ button to, again, select and set values for the parameters. For datasets, the input dialog allows you to type in or import values, such as initial or default values; see the *Model Analysis using Datasets* topic.

Element Type	Behavior
ValueType	ValueType elements either generalize from a primitive type or are substituted by SysMLSimReal for simulation.
Block	Block elements mapped to SysMLSimClass or SysMLSimModel elements support the creation of data sets. If you have defined multiple data sets in a SysMLSimClass (which can be generalized), you must identify one of them as the default (using the context menu option 'Set as Default Dataset'). As a SysMLSimModel is a possible top-level element for a simulation, and will not be generalized, if you have defined multiple datasets the dataset to use is chosen during the simulation.

Properties	Properties within a Block can be configured to be either SimConstants or SimVariables. For a SimVariable, you configure these attributes:
	
	• isContinuous — determines whether the property value varies continuously ('true', the default) or discretely ('false')
	
	• isConserved — determines whether values of the property are conserved ('true') or not ('false', the default); when modeling for physical interaction, the interactions include exchanges of conserved physical substances such as electrical current, force or fluid flow
	
	• changeCycle — specifies the time interval at which a discrete property value changes; the default value is '0'
	  - changeCycle can be set to a value other than 0 only when isContinuous = 'false'
	  - The value of changeCycle must be positive or equal to 0

Port	No configuration required.
SimFunction	Functions are created as operations in Blocks or Constraint Blocks, stereotyped as 'SimFunction'. No configuration is required in the Configure SysML Simulation window.
Generalization	No configuration required.
Binding Connector	Binds a property to a parameter of a constraint property. No configuration required.
Connector	Connects two Ports. No configuration required in the Configure SysML Simulation view. However, you might have to configure the properties of the Port's type by determining whether the

	attribute isConserved should be set as 'False' (for potential properties, so that equality coupling is established) or 'True' (for flow/conserved properties, so that sum-to-zero coupling is established).
Constraint Block	No configuration required.

Simulation Tab

This table describes the fields of the 'Simulation' tab on the Configure SysML Simulation view.

Field	Action
Model	Click on the drop-down arrow and select the top-level node (a SysMLSimModel element) for the simulation. The list is populated with the names of the Blocks defined as top-level, model nodes.
Data Set	Click on the drop-down arrow and select the dataset for the selected model.
Start	Type in the initial wait time before which the simulation is started, in seconds (default value is 0).
Stop	Type in the number of seconds for which the simulation will execute.
Format	Click on the drop-down arrow and select either 'plt', 'csv' or 'mat' as the format of the result file, which could potentially be used by other tools.
Parametric Plot	• Select this checkbox to plot Legend A on the y-axis against Legend B on the x-axis. • Deselect the checkbox to plot Legend(s) on the y-axis against time on the x-axis Note: With the checkbox selected, you must select two properties to plot.
Dependencies	Lists the types that must be generated to simulate this model.
Properties to Plot	Provides a list of variable properties that are involved with the simulation. Select the checkbox against each property to plot.

Model Analysis using Datasets

Every SysML Block used in a Parametric model can, within the Simulation configuration, have multiple datasets defined against it. This allows for repeatable simulation variations using the same SysML model.

A Block can be typed as a SysMLSimModel (a top-level node that cannot be generalized or form part of a composition) or as a SysMLSimClass (a lower-level element that can be generalized or form part of a composition). When running a simulation on a SysMLSimModel element, if you have defined multiple datasets, you can specify which dataset to use. However, if a SysMLSimClass within the simulation has multiple datasets, you cannot select which one to use during the simulation and must therefore identify one dataset as the default for that Class.

Access

Ribbon	Simulate > SysMLSim > Manage > SysMLSim Configuration Manager > in "block" group > Name column > Context menu on block element > Create Simulation DataSet

Dataset Management

Task	Action
Create	To create a new dataset, right-click on a Block name and select the 'Create Simulation Dataset' option. The dataset is added to the end of the list of components underneath the Block name. Click on the [...] button to set up the dataset on the 'Configure Simulation Data' dialog (see the *Configure Simulation Data* table).
Duplicate	To duplicate an existing dataset as a base for creating a new dataset, right-click on the dataset name and select the 'Duplicate' option. The duplicate dataset is added to the end of the list of components underneath the Block name. Click on the [...] button to edit the dataset on the 'Configure Simulation Data' dialog (see the *Configure Simulation Data* table).
Delete	To remove a dataset that is no longer required, right-click on the dataset and select the 'Delete Dataset' option.
Set Default	To set the default dataset used by a SysMLSimClass when used as a property type or inherited (and when there is more than one dataset), right-click on the dataset and select the 'Set as Default' option. The name of the default dataset is highlighted in bold. The properties used by a model will use this default configuration unless the model overrides them explicitly.

Configure Simulation Data

This dialog is principally for information. The only column in which you can directly add or change data is the 'Value' column.

Column	Description
Attribute	The 'Attribute' column provides a tree view of all the properties in the Block being edited.
Stereotype	The 'Stereotype' column identifies, for each property, if it has been configured to be a constant for the duration of the simulation or variable, so that the value is expected to change over time.
Type	The 'Type' column describes the type used for simulation of this property. It can be either a primitive type (such as 'Real') or a reference to a Block contained in the model. Properties referencing Blocks will show the child properties specified by the referenced Block below them.
Default Value	The 'Default Value' column shows the value that will be used in the simulation if no override is provided. This can come from the 'Initial Value' field in the SysML model or from the default dataset of the parent type.
Value	The 'Value' column allows you to override the default value for each primitive value.
Export / Import	Click on these buttons to modify the values in the current dataset using an external application such as a spreadsheet, and then re-import them to the list.

Modeling and Simulation with Modelica Library

This feature is available from Enterprise Architect Release 14.1.

The Modelica Libraries are comprehensive resources that provide many useful types, functions and models. When creating SysML models in Enterprise Architect, you can reference resources available in the Modelica Libraries.

Referencing a Type defined in Modelica Library

To configure a simulation to reference a Modelica Library, you first create a ValueType element pointing to the Modelica library, and register this in the Simulation configuration.

First, create an element for a Referenced Modelica Type

- Create a ValueType element with the full name of the Modelica Library path:
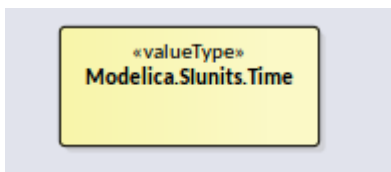
Configure the ValueType element as 'ReferencedType':

- Double-click on the SysMLSimConfiguration element to open the 'Configure SysML Configuration' tab

- Navigate to the ValueType element

- In the drop-down field set the value to 'ReferencedType'
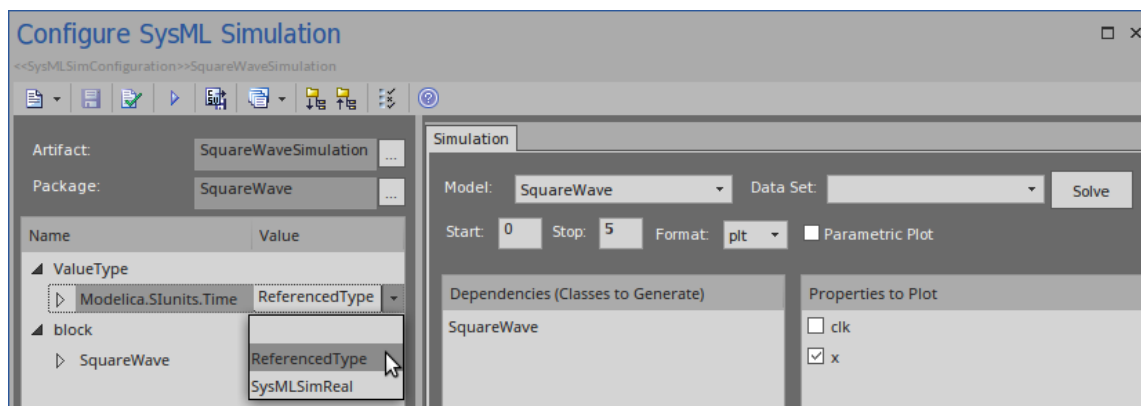
As the ValueType element is configured as 'ReferencedType', the element will not display in the 'Dependencies' list and will not be generated as a new Class definition to the Modelica file.

Next, set the type for a Property to the ValueType element

In Enterprise Architect, a SysML Property can be set to be a primitive type or an element such as a Block or a ValueType.

Option 1:

- Select the Property (Part or Port)

- Press Alt+1 to open the element Properties window

- Switch to the 'Property' tab and choose 'Select Type...'

- Browse to the ValueType element you created

Option 2:

- Select the Property (Part or Port)

- Press Ctrl+L on the Property
- Browse to the ValueType element you created

Including a Modelica Library in a Simulation

When using referenced types from a Modelica library in a model, you must load the Modelica model in the environment for the simulation to work.

- Expand the menu option and select 'Include Modelica Library'
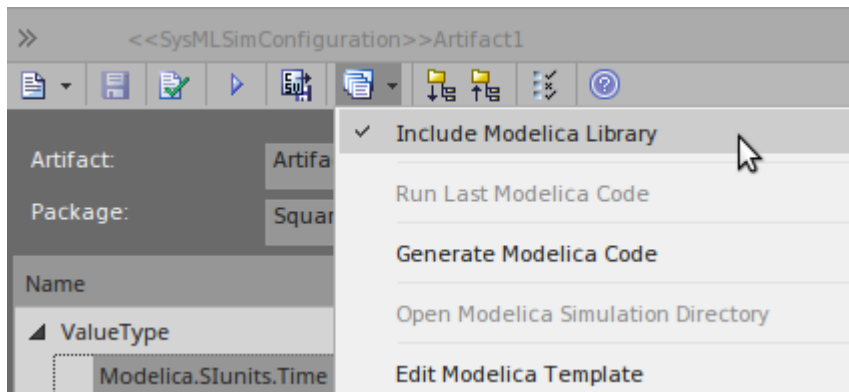
- If this option is ticked, this function will be generated to 'Solve.mos' by default:

  loadModel(Modelica);

Click Here for a detailed description of the *loadModel()* scripting function.

Customize the Modelica Script Template

You can modify the Modelica script template to add extra libraries required by the model and simulation.

Access

Ribbon: Code | Configure | Options | Edit Code Templates In the 'Language' field select 'Modelica' and in the Scripts list select 'SysMLSim Script'

```
10 %if sysmlSim_IncludeLibrary == "T"%
11 msg := "--------------- Loading Modelica Model... ---------------";
12 loadModel(Modelica);
13 %endIf%
```

As you are appending extra libraries after 'loadModel(Modelica)', the libraries' resources can be referenced by your model.

SquareWave Example

This example is based on Figure 13-11 in: *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical*, Second Edition, by Peter Fritzson.
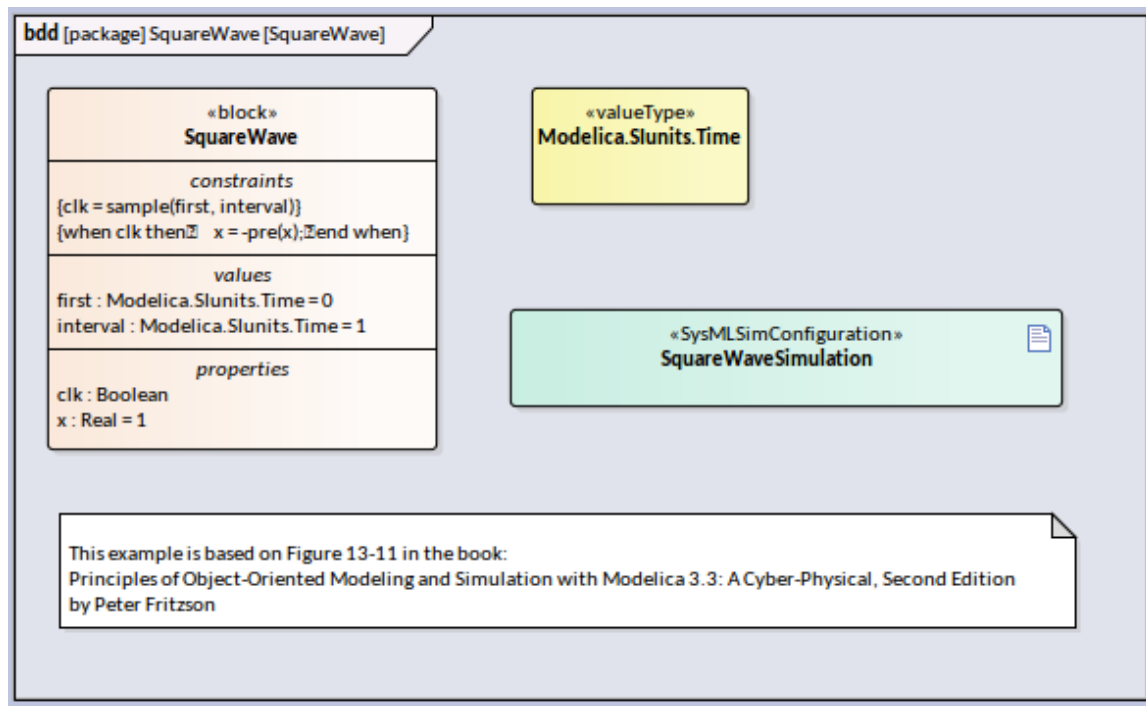
In this example:

- We create a ValueType *Modelica.SIunits.Time*, which is used for the property *first* and *interval* of the Block *SquareWave*

- ValueType *Modelica.SIunits.Time* is configured as 'ReferencedType' in the SysML Simulation window

- Select the menu item 'Include Modelica Library'
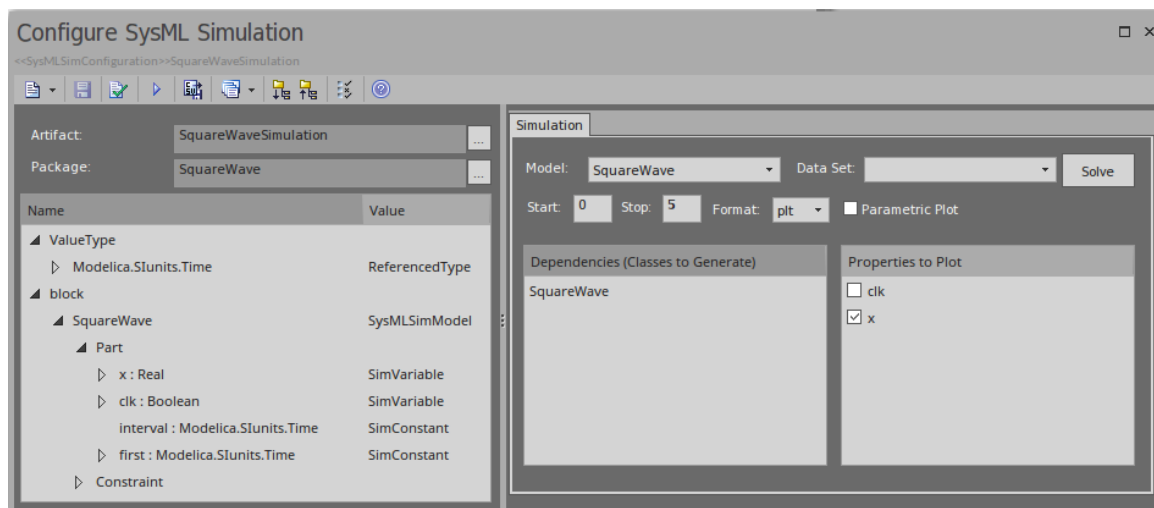
Run the simulation; the variable *x* is plotted like this:
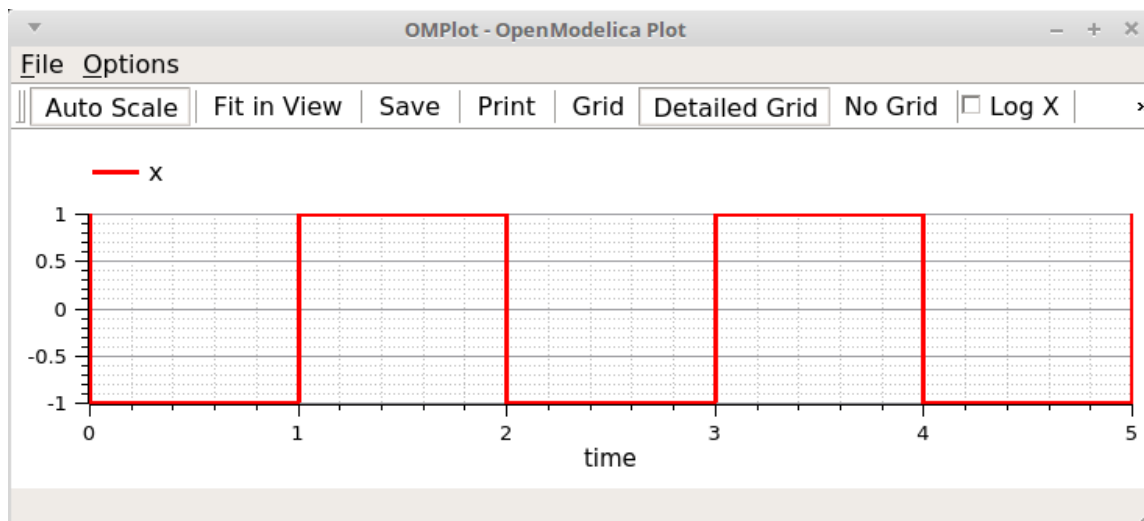
SampleTime Example

This example is based on Figure 13-13 in: *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical*, Second Edition, by Peter Fritzson.
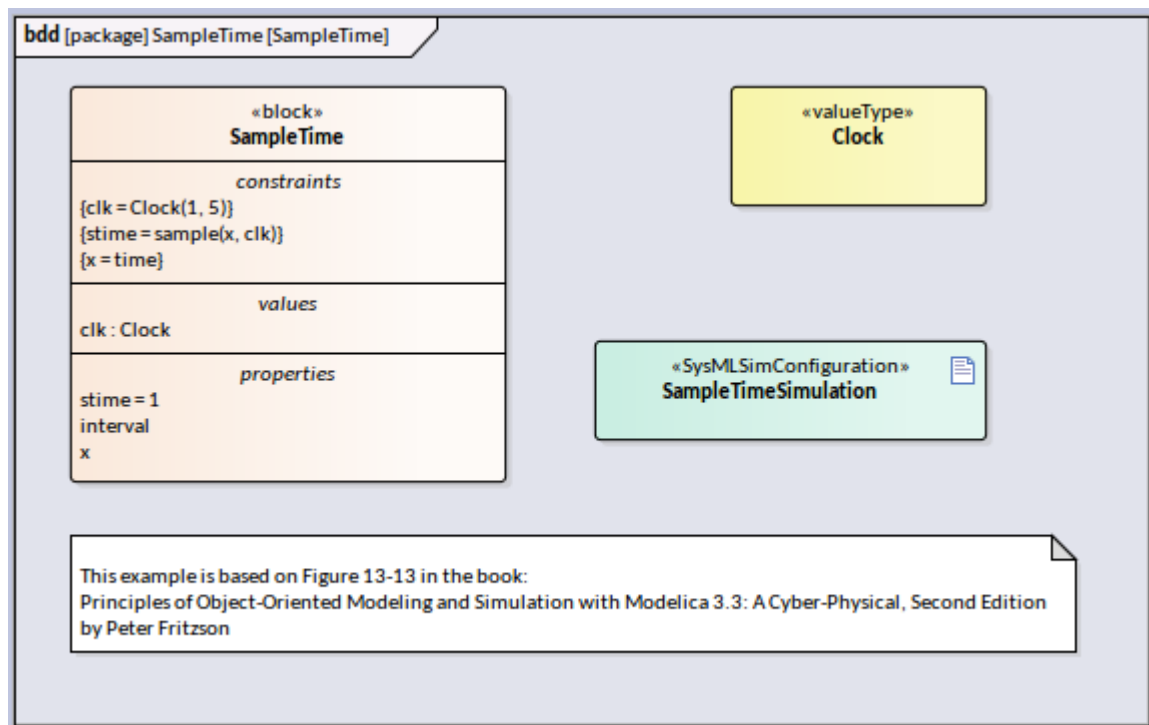
In this example:

- We created a ValueType *Clock*, which is used for the property *clk* of Block *SampleTime*
- ValueType *Clock* is configured as 'ReferencedType' in the SysML Simulation window
- The menu item 'Include Modelica Library' is unselected
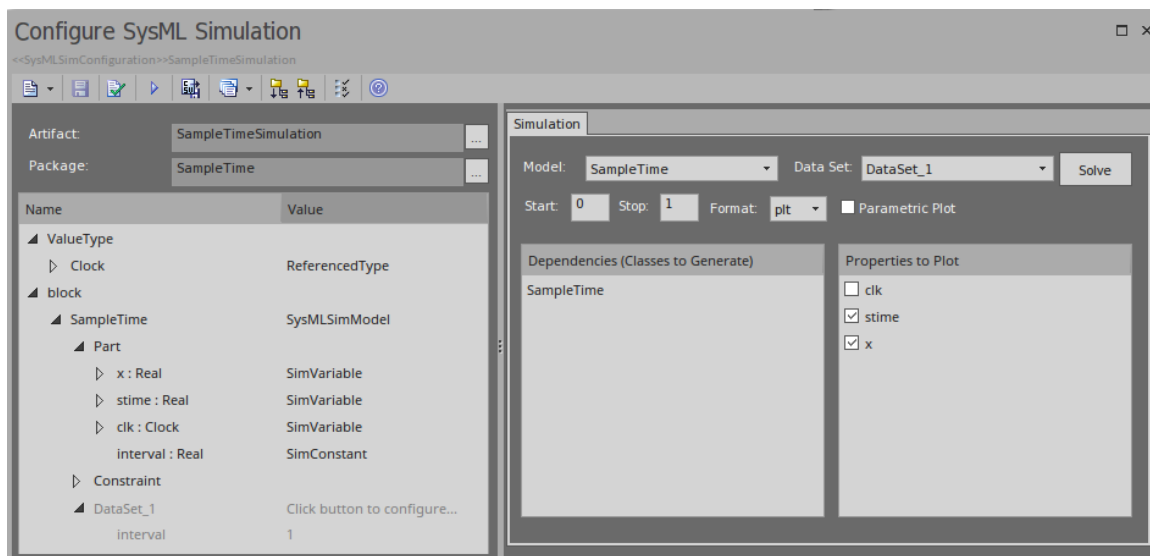
Run the simulation; the variable *x* and *stime* plot resembles this:
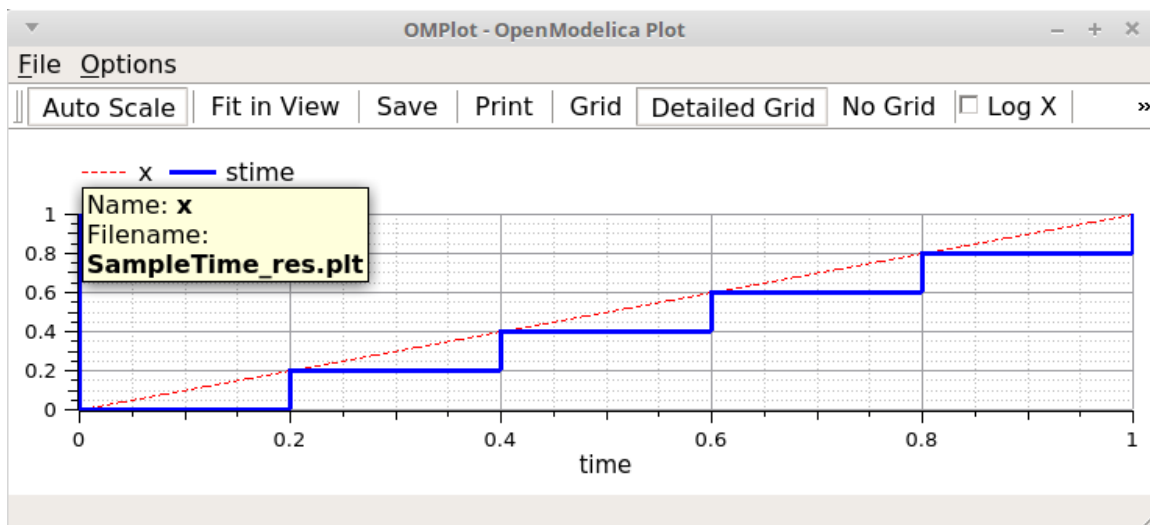
SysML Simulation Examples

This section provides a worked example for each of: creating a SysML model for a domain, simulating it, and evaluating the results of the simulation. These examples apply the information discussed in the earlier topics.

Electrical Circuit Simulation Example

The first example is of the simulation of an electrical circuit. The example starts with an electrical circuit diagram and converts it to a parametric model. The model is then simulated and the voltage at the source and target wires of a resistor are evaluated and compared to the expected values.

Electrical Circuit Simulation Example

Mass-Spring-Damper Oscillator Simulation Example

The second example uses a simple physical model to demonstrate the oscillation behavior of a string under tension.

Mass-Spring-Damper Oscillator Simulation Example

Water Tank Pressure Regulator

The final example shows the water levels of two water tanks where the water is being distributed between them. We first simulate a well-balanced system, then we simulate a system where the water will overflow from the second tank.

Water Tank Pressure Regulator

Electrical Circuit Simulation Example

In this section, we will walk through the creation of a SysML parametric model for a simple electrical circuit, and then use a parametric simulation to predict and chart the behavior of that circuit.

Circuit Diagram

The electrical circuit we are going to model, shown here, uses a standard electrical circuit notation.
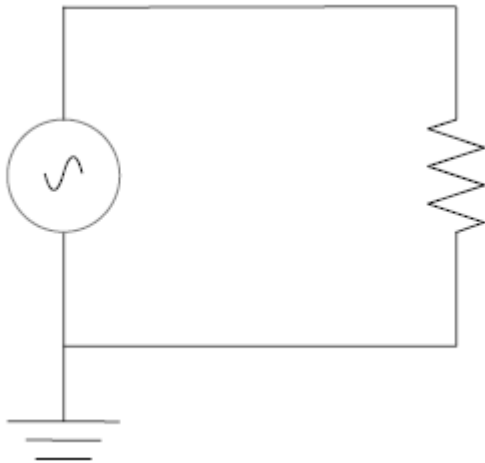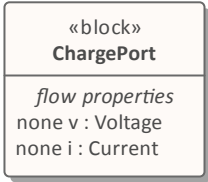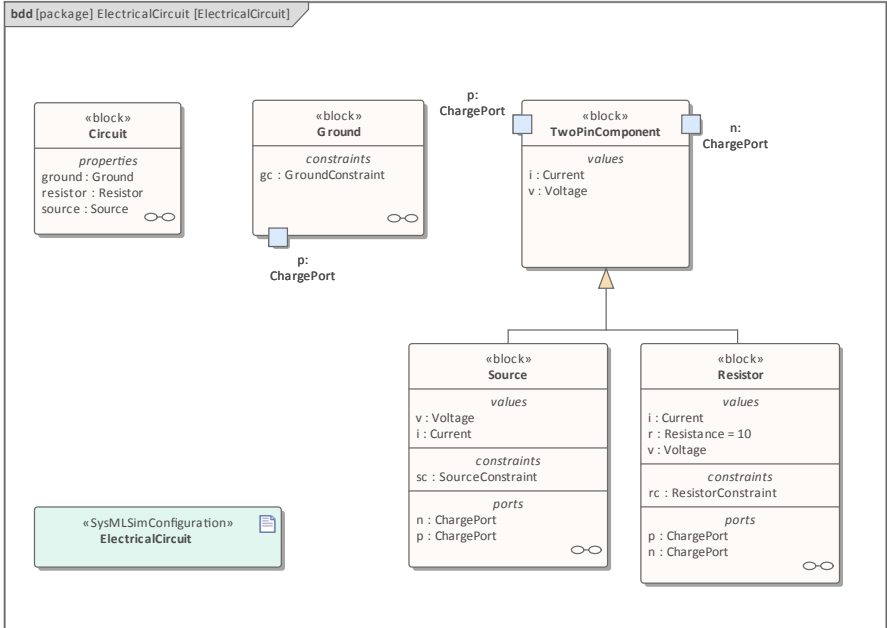
The circuit includes an AC power source, a ground and a resistor, connected to each other by wires.

Create SysML Model

This table shows how we can build up a complete SysML model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

Component	Action
Types	Define Value Types for the Voltage, Current and Resistance. Unit and quantity kind are not important for the purposes of simulation, but would be set if defining a complete SysML model. These types will be generalized from the primitive type 'Real'. In other models, you can choose to map a Value Type to a corresponding simulation type separate from the model. Additionally, define a composite type called ChargePort, which includes properties

for both Current and Voltage. This type allows us to represent the electrical energy at the connectors between components.

bdd [package] ElectricalCircuit [Composite Typ

«block»
ChargePort

flow properties
none v : Voltage
none i : Current

Blocks	In SysML, the circuit and each of the components will be represented as Blocks. Create a Circuit Block in a Block Definition diagram (BDD). The circuit has three parts: a source, a ground, and a resistor. These parts are of different types, with different behaviors. Create a Block for each of these part types. The three parts of the Circuit Block are connected through Ports, which represent electrical pins. The source and resistor have a positive and a negative pin. The ground has only one pin, which is positive. Electricity (electric charge) is transmitted through the pins. Create an abstract block 'TwoPinComponent' with two Ports (pins). The two Ports are named 'p' (positive) and 'n' (negative), and they are of type ChargePort.

This figure shows what the BDD should look like, with the Blocks Circuit, Ground, TwoPinComponent, Source and Resistor.

Internal Structure	Create an Internal Block diagram (IBD) for Circuit. Add properties for the Source, Resistor and Ground, typed by the corresponding Blocks. Connect the Ports with connectors. The positive pin of the Source is connected to the negative pin of the Resistor. The positive pin of the Resistor is connected to the negative pin of the Source. The Ground is also connected to the negative pin of the Source.

Notice that this follows the same structure as the original circuit diagram, but the symbols for each component have been replaced with properties typed by the Blocks we have defined.

Constraints	Equations define mathematical relationships between numeric properties. In SysML, equations are represented as constraints in Constraint Blocks. Parameters of Constraint Blocks correspond to SimVariables and SimConstants of Blocks ('i', 'v', 'r' in this example), as well as to SimVariables present in the type of the Ports ('pv', 'pi', 'nv', 'ni' in this example).
	Create a Constraint Block 'TwoPinComponentConstraint' to define parameters and equations common to sources and resistors. The equations should state that the voltage of the component is equal to the difference between the voltages at the positive and negative pins. The current of the component is equal to the current going through the positive pin. The sum of the currents going through the two pins must add up to zero (one is the negative of the other). The Ground constraint states that the voltage at the Ground pin is zero. The Source constraint defines the voltage as a sine wave with the current simulation time as a parameter. This figure shows how these constraints should look in a BDD.

bdd [package] ElectricalCircuit [Constraints]

| Bindings | The values of Constraint parameters are equated to variable and constant values with binding connectors. Create Constraint properties on each Block (properties typed by Constraint Blocks) and bind the Block variables and constants to the Constraint parameters to apply the Constraint to the Block. These figures show the bindings for the Ground, the Source and the Resistor respectively.

For the Ground constraint, bind gc.pv to p.v. |
|---|---|

par [block] Ground [Ground]

For the Source constraint, bind:

- sc.pi to p.i
- sc.pv to p.v
- sc.v to v
- sc.i to i
- sc.ni to n.i and
- sc.nv to n.v

For the Resistor constraint, bind:

- rc.pi to p.i
- rc.pv to p.v
- rc.v to v
- rc.i to i
- rc.ni to n.i
- rc.nv to n.v and
- rc.r to r

Configure Simulation Behavior

This table shows the detailed steps of the configuration of SysMLSim.

Step	Action
SysMLSimConfiguration Artifact	• Select 'Simulate > SysMLSim/Modelica > Manage > SysMLSim Configuration Manager' • From the first toolbar icon drop-down, select 'Create Artifact' and create the Artifact element • Select the Package that owns this SysML Model

Create Root elements in Configuration Manager	• ValueType • block • constraintBlock
ValueType Substitution	Expand ValueType and for each of Current, Resistance and Voltage select 'SysMLSimReal' from the 'Value' combo box.
Set property as flow	• Expand 'block' to ChargePort | FlowProperty | i : Current and select 'SimVariable' from the 'Value' combo box • For 'SysMLSimConfiguration' click on the [...] button to open the 'Element Configurations' dialog • Set 'isConserved' to 'True'
SysMLSimModel	This is the model we want to simulate: set the Block 'Circuit' to be 'SysMLSimModel'.

Run Simulation

In the 'Simulation' page, select the checkboxes against 'resistor.n.v' and 'resistor.p.v' for plotting and click on the Solve button.
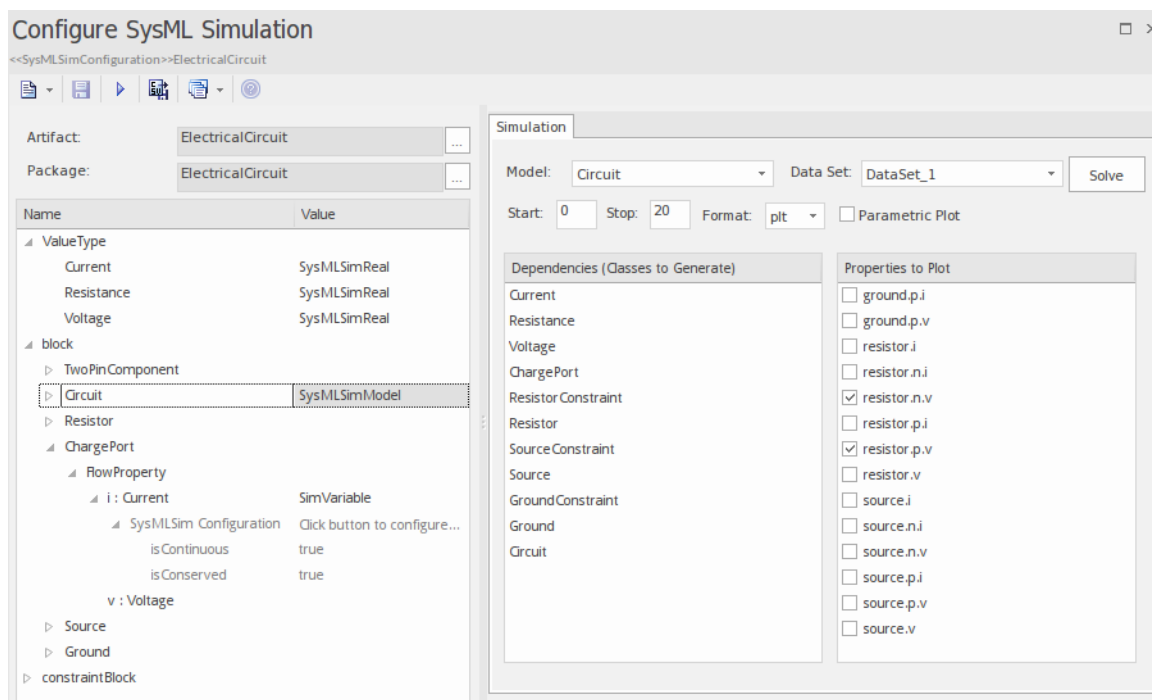
The two legends 'resistor.n.v' and 'resistor.p.v' are plotted, as shown.
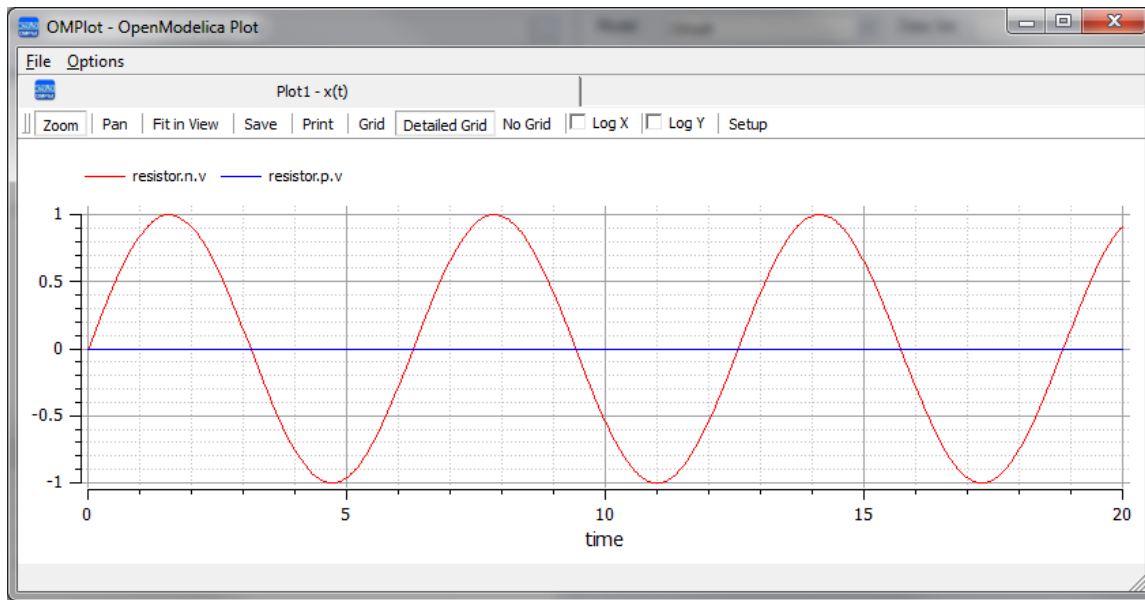
Mass-Spring-Damper Oscillator Simulation Example

In this section, we will walk through the creation of a SysML parametric model for a simple Oscillator composed of a mass, a spring and a damper, and then use a parametric simulation to predict and chart the behavior of this mechanical system. Finally, we perform what-if analysis by comparing two oscillators provided with different parameter values through data sets.

System being modeled
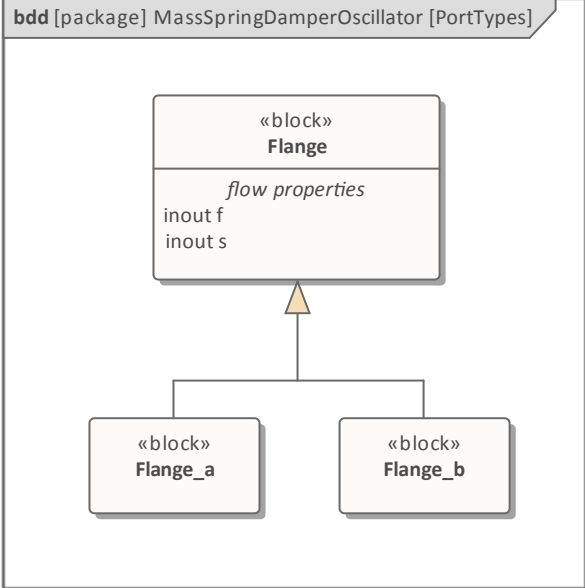
A mass is hanging on a spring and damper. The first state shown here represents the initial point at time=0, just when the mass is released. The second state represents the final point when the body is at rest and the spring forces are in equilibrium with gravity.
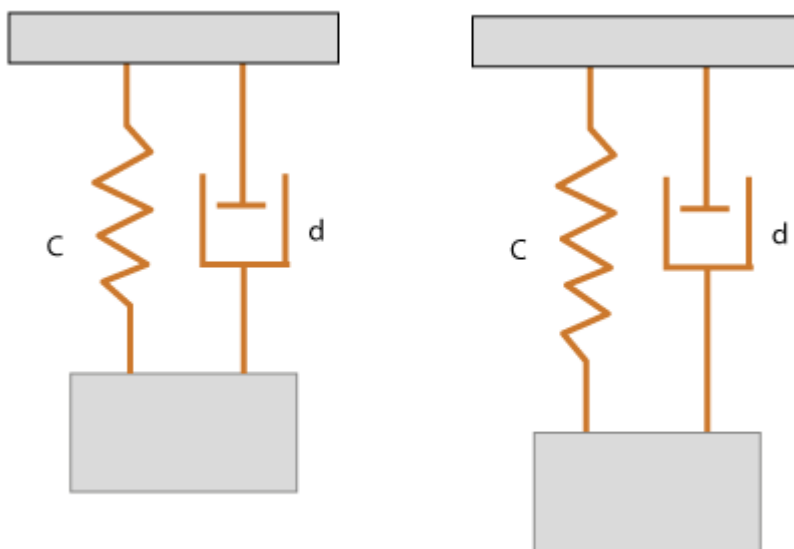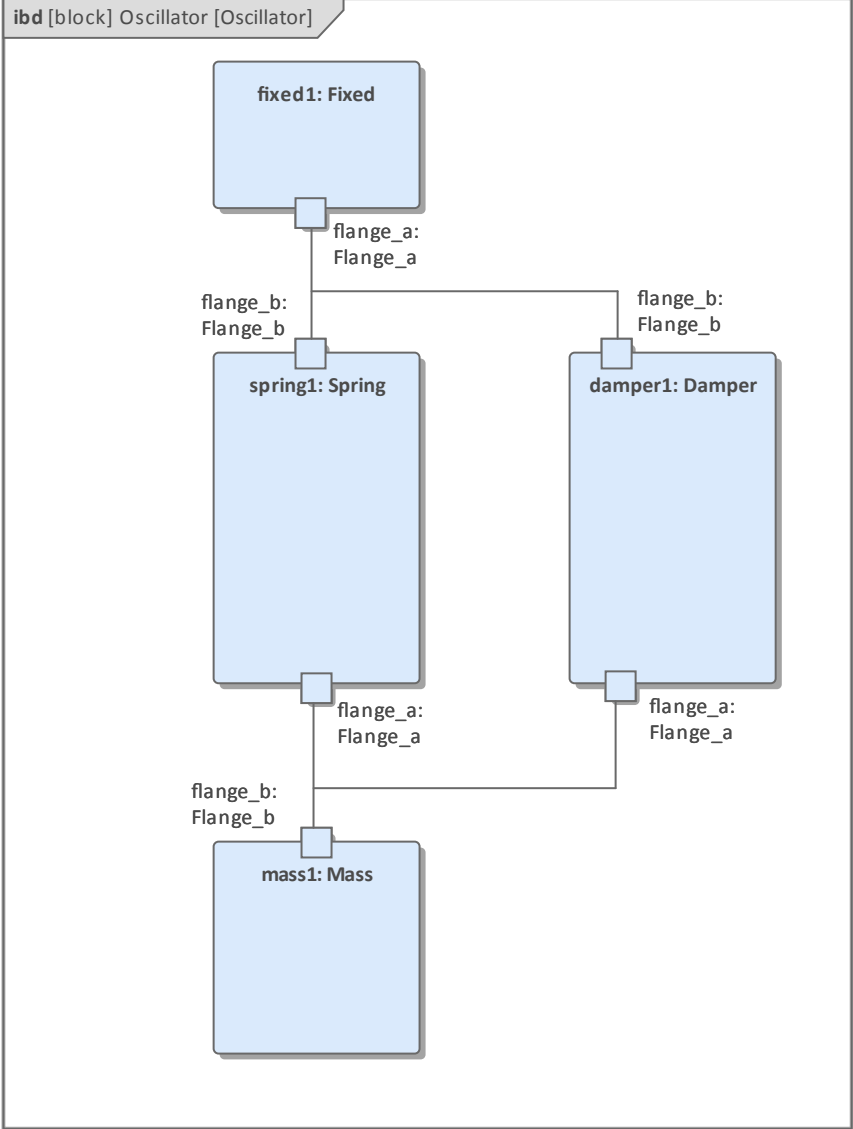
Create SysML Model

The MassSpringDamperOscillator model in SysML has a main Block, the *Oscillator*. The Oscillator has four parts: a fixed *ceiling*, a *spring*, a *damper* and a *mass body*. Create a Block for each of these parts. The four parts of the Oscillator Block are connected through Ports, which represent mechanical flanges.

Components	Description
Port Types	The Blocks 'Flange_a' and 'Flange_b' used for flanges in the 1D transitional mechanical domain are identical but have slightly different roles, somewhat analogous to the roles of PositivePin and NegativePin in the electrical domain. Momentum is transmitted through the flanges. So the attribute *isConserved* of flow property *Flange.f* should be set to True.

	bdd [package] MassSpringDamperOscillator [PortTypes] «block» **Flange** *flow properties* inout f inout s «block» «block» **Flange_a** **Flange_b**
Blocks and Ports	• Create Blocks 'Spring', 'Damper', 'Mass' and 'Fixed' to represent the spring, damper, mass body and ceiling respectively • Create a Block 'PartialCompliant' with two Ports (flanges), named 'flange_a' and 'flange_b' — these are of type Flange_a and Flange_b respectively; the 'Spring' and 'Damper' Blocks generalize from 'PartialCompilant' • Create a Block 'PartialRigid' with two Ports (flanges), named 'flange_a' and 'flange_b' — these are of type Flange_a and Flange_b respectively; the 'Mass' Block generalizes from 'PartialRigid' • Create a Block 'Fixed' with only one flange for the ceiling, which only has the Port 'flange_a' typed to Flange_a

bdd [package] MassSpringDamperOscillator [blocksAndPorts]

«SysMLSimConfiguration»
MassSpringDamperOscillator

«block»
Oscillator

parts
damper1 : Damper
fixed1 : Fixed
mass1 : Mass
spring1 : Spring

+fixed1

«block»
Fixed

constraints
{flange_a.s = s0}

properties
s0 = 1.0

flange_a:
Flange_a

+damper1

«block»
Damper

constraints
{v_rel=der(s_rel)}
{f = d * v_rel}
{lossPower = f * v_rel}

properties
d = 25
lossPower
v_rel

+spring1

«block»
Spring

constraints
{f = c*(s_rel - s_rel0)}

properties
s_rel0 = 2
c = 10000

+mass1

«block»
Mass

constraints
{v = der(s)}
{a = der(v)}
{m*a = flange_a.f + flange_b.f}
{flange_a.f = - m * g}

properties
a
g = 9.81
m = 1
v

«block»
PartialCompliant

constraints
{s_rel=flange_b.s - flange_a.s}
{flange_b.f = f}
{flange_a.f = -f}

properties
s_rel = 0
f

flange_a:
Flange_a

flange_b:
Flange_b

flange_a:
Flange_a

«block»
PartialRigid

constraints
{flange_a.s = s-L/2}
{flange_b.s = s + L/2}

properties
L = 1
s = -0.5

flange_b:
Flange_b

This example is based on "Section 5.4 Connectors, Components, and Coordinate Systems" in the book:
Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical, Second Edition
by Peter Fritzson

Internal structure	Create an Internal Block diagram (IBD) for 'Oscillator'. Add properties for the fixed ceiling, spring, damper and mass body, typed by the corresponding Blocks. Connect the Ports with connectors. • Connect 'flange_a' of 'fixed1' to 'flange_b' of 'spring1' • Connect 'flange_b' of 'damper1' to 'flange_b' of 'spring1' • Connect 'flange_a' of 'damper1' to 'flange_a' of 'spring1' • Connect 'flange_a' of 'spring1' to 'flange_b' of 'mass1'

	ibd [block] Oscillator [Oscillator] **fixed1: Fixed** flange_a: Flange_a flange_b: Flange_b flange_b: Flange_b **spring1: Spring** **damper1: Damper** flange_a: Flange_a flange_a: Flange_a flange_b: Flange_b **mass1: Mass**
Constraints	For simplicity, we define the constraints directly in the Block elements; optionally you can define Constraint Blocks, use constraint properties in the Blocks, and bind their parameters to the Block's properties.

Two Oscillator Compare Plan

After we model the Oscillator, we want to do some what-if analysis. For example:

- What is the difference between two oscillators with different dampers?
- What if there is no damper?
- What is the difference between two oscillators with different springs?
- What is the difference between two oscillators with different masses?

Here are the steps for creating a comparison model:

- Create a Block named 'OscillatorCompareModel'
- Create two Properties for 'OscillatorCompareModel', called *oscillator1* and *oscillator2,* and type them with the Block *Oscillator*
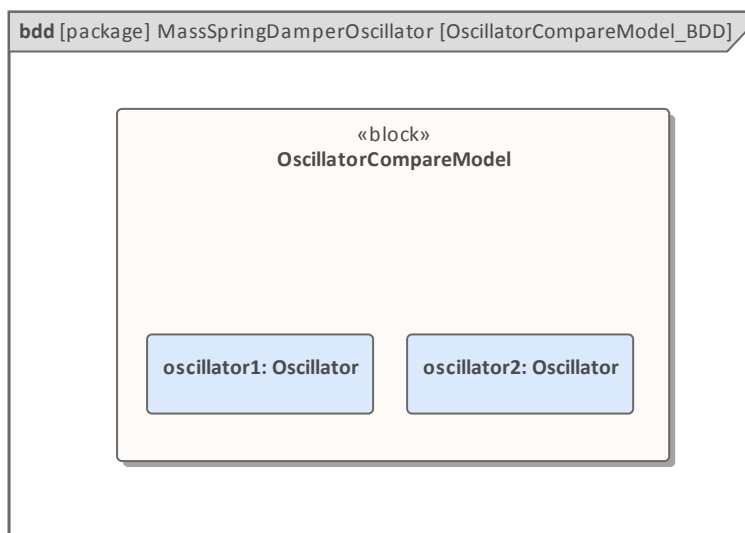
Setup DataSet and Run Simulation

Create a SysMLSim Configuration Artifact and assign it to this Package. Then create these data sets:

- Damper: small VS big
  provide 'oscillator1.damper1.d' with the value 10 and 'oscillator2.damper1.d' with the larger value 20
- Damper: no vs yes
  provide 'oscillator1.damper1.d with the value 0; ('oscillator2.damper1.d' will use the default value 25)
- Spring: small vs big
  provide 'oscillator1.spring1.c' with the value 6000 and 'oscillator2.spring1.c' with the larger value 12000
- Mass: light vs heavy
  provide 'oscillator1.mass1.m' with the value 0.5 and 'oscillator2.mass1.m' with the larger value 2

The configured page resembles this:

On the 'Simulation' page, select 'OscillatorCompareModel', plot for 'oscillator1.mass1.s' and 'oscillator2.mass1.s', then choose one of the created datasets and run the simulation.

Tip: If there are too many properties in the plot list, you can toggle the Filter bar using the context menu on the list header, then type in 'mass1.s' in this example.
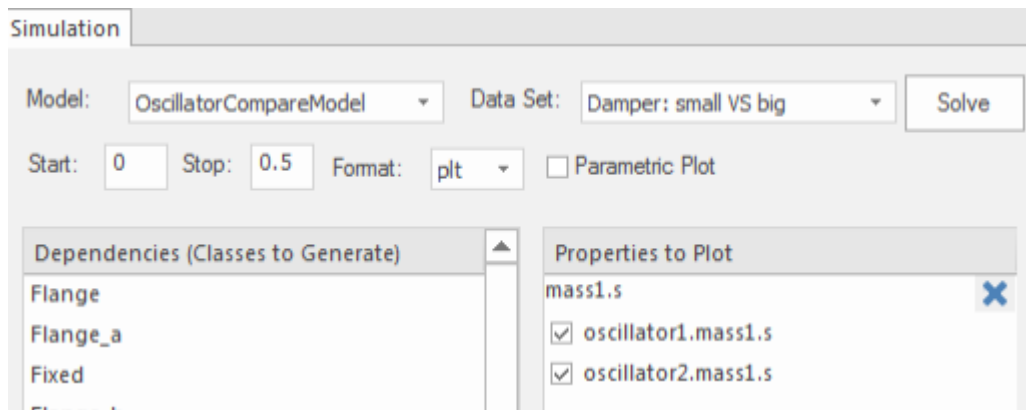
These are the simulation results:

- Damper, small vs big: the smaller damper makes the body oscillate more
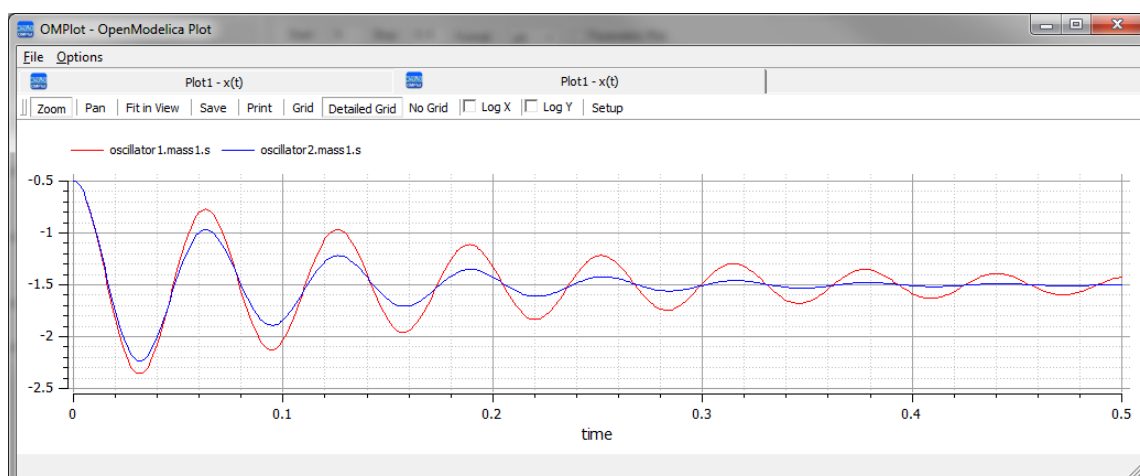
- Damper, no vs yes: the oscillator never stops without a damper
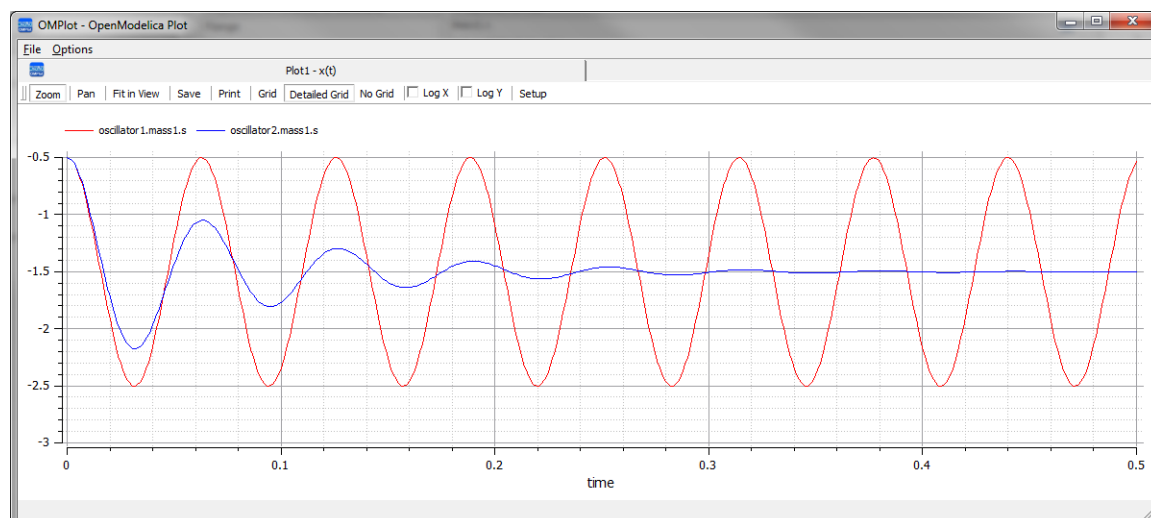
- Spring, small vs big: the spring with smaller 'c' will oscillate more slowly
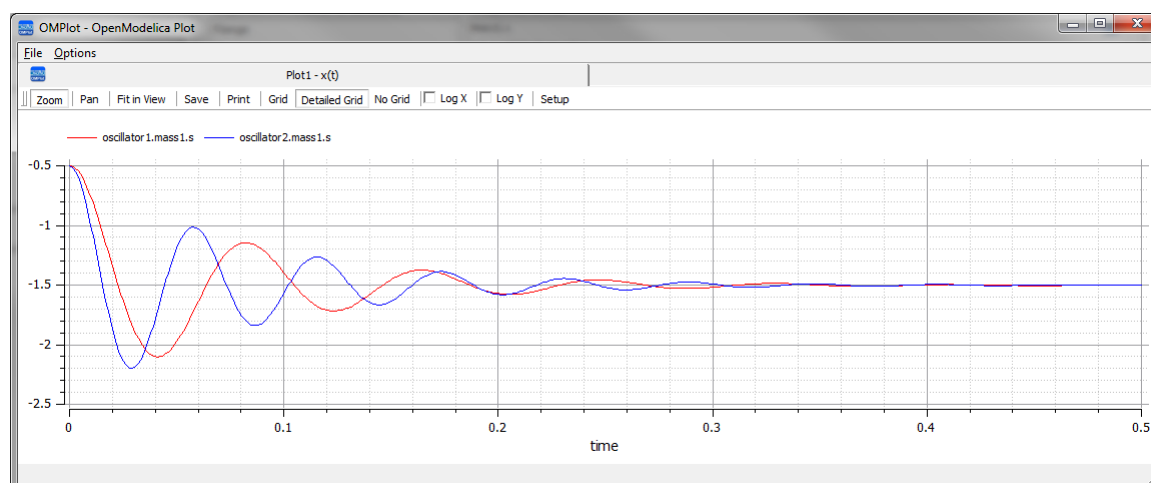
- Mass, light vs heavy: the object with smaller mass will oscillate faster and regulate quicker
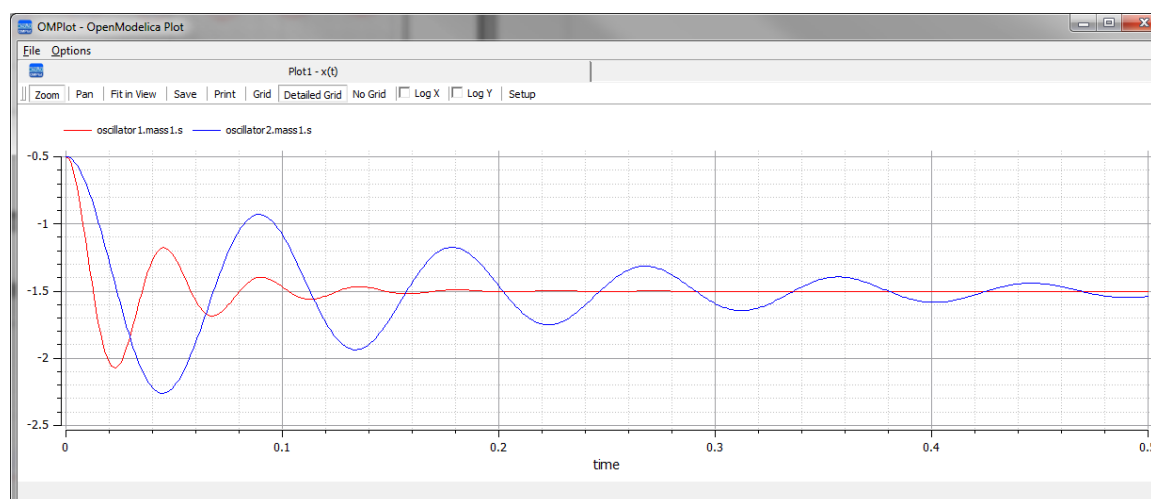
Water Tank Pressure Regulator

In this section we will walk through the creation of a SysML parametric model for a Water Tank Pressure Regulator, composed of two connected tanks, a source of water and two controllers, each of which monitors the water level and controls the valve to regulate the system.

We will explain the SysML model, create it and set up the SysMLSim Configurations. We will then run the simulation with OpenModelica.
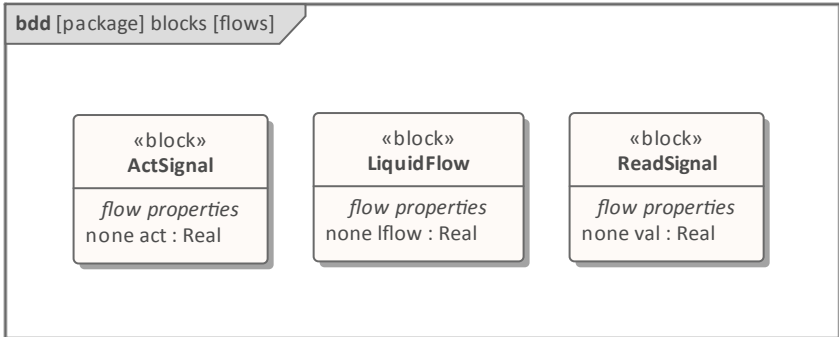
System being modeled

This diagram depicts two tanks connected together, and a water source that fills the first tank. Each tank has a proportional–integral (PI) continuous controller connected to it, which regulates the level of water contained in the tanks at a reference level. While the source fills the first tank with water, the PI continuous controller regulates the outflow from the tank depending on its actual level. Water from the first tank flows into the second tank, which the PI continuous controller also tries to regulate. This is a natural and non domain-specific physical problem.
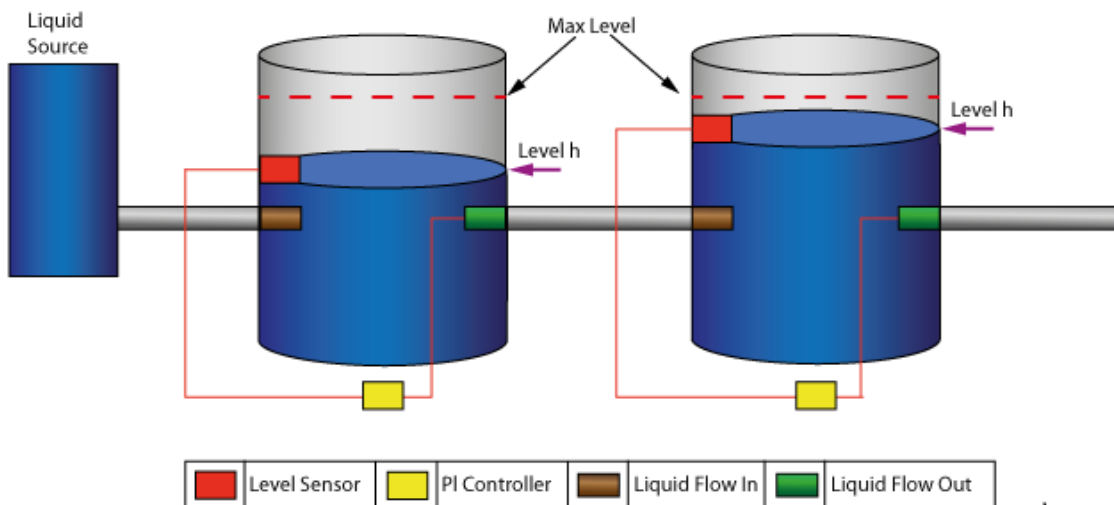
Create SysML Model

Component	Discussion
Port Types	The tank has four ports; they are typed to these three blocks: • ReadSignal: Reading the fluid level; this has a property 'val' with unit 'm' • ActSignal: The signal to the actuator for setting valve position • LiquidFlow: The liquid flow at inlets or outlets; this has a property 'lflow' with unit 'm³/s"

	bdd [package] blocks [flows] «block» **ActSignal** / *flow properties* / none act : Real «block» **LiquidFlow** / *flow properties* / none lflow : Real «block» **ReadSignal** / *flow properties* / none val : Real

Block Definition Diagram	LiquidSource: The water entering the tank must come from somewhere, therefore we have a liquid source component in the tank system, with the property *flowLevel* having a unit of 'm³/s'. A Port 'qOut' is typed to 'LiquidFlow'.
	
	Tank: The tanks are connected to controllers and liquid sources through Ports.
	
	• Each Tank has four Ports:
	  - qIn: for input flow
	  - qOut: for output flow
	  - tSensor: for providing fluid level measurements
	  - tActuator: for setting the position of the valve at the outlet of the tank
	
	• Properties:
	  - area (unit='m²'): area of the tank, involved in the *mass balance* equation
	  - h (unit = 'm'): water level, involved in the *mass balance* equation; its value is read by the sensor
	  - flowGain (unit = 'm²/s'): the output flow is related to the valve position by *flowGain*
	  - minV, maxV: Limits for output valve flow
	
	BaseController: This Block could be super of a PI Continuous Controller and PI Discrete Controller.
	
	• Ports:
	  - cIn: Input sensor level
	  - cOut: Control to actuator
	
	• Properties:
	  - Ts (unit = 's'): Time period between discrete samples (not used in this example)
	  - K: Gain factor
	  - T (unit = 's'): Time constant of controller
	  - ref: reference level
	  - error: difference between the reference level and the actual level of water, obtained from the sensor
	  - outCtr: control signal to the actuator for controlling the valve position
	
	PIcontinuousController: generalize from BaseController
	
	• Properties:
	  - x: the controller state variable

| Constraint Blocks | The flow increases sharply at time=150 to a factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to handle. |

The central equation regulating the behavior of the tank is the *mass balance* equation.

The output flow is related to the valve position by a 'flowGain' parameter.

The sensor simply reads the level of the tank.

The Constraints defined for 'BaseController' and 'PIcontinuousController' are illustrated in these figures.

Internal Block Diagram	This is the Internal Block diagram for a system with a single tank.

This is the Internal Block diagram for a system with two connected tanks.
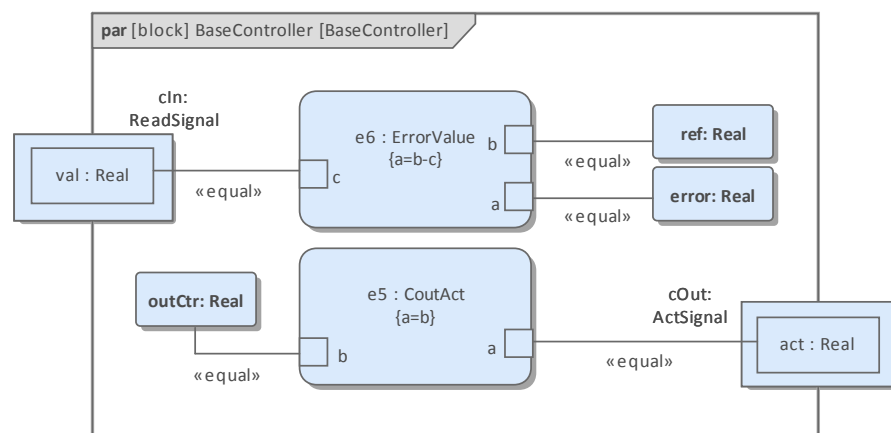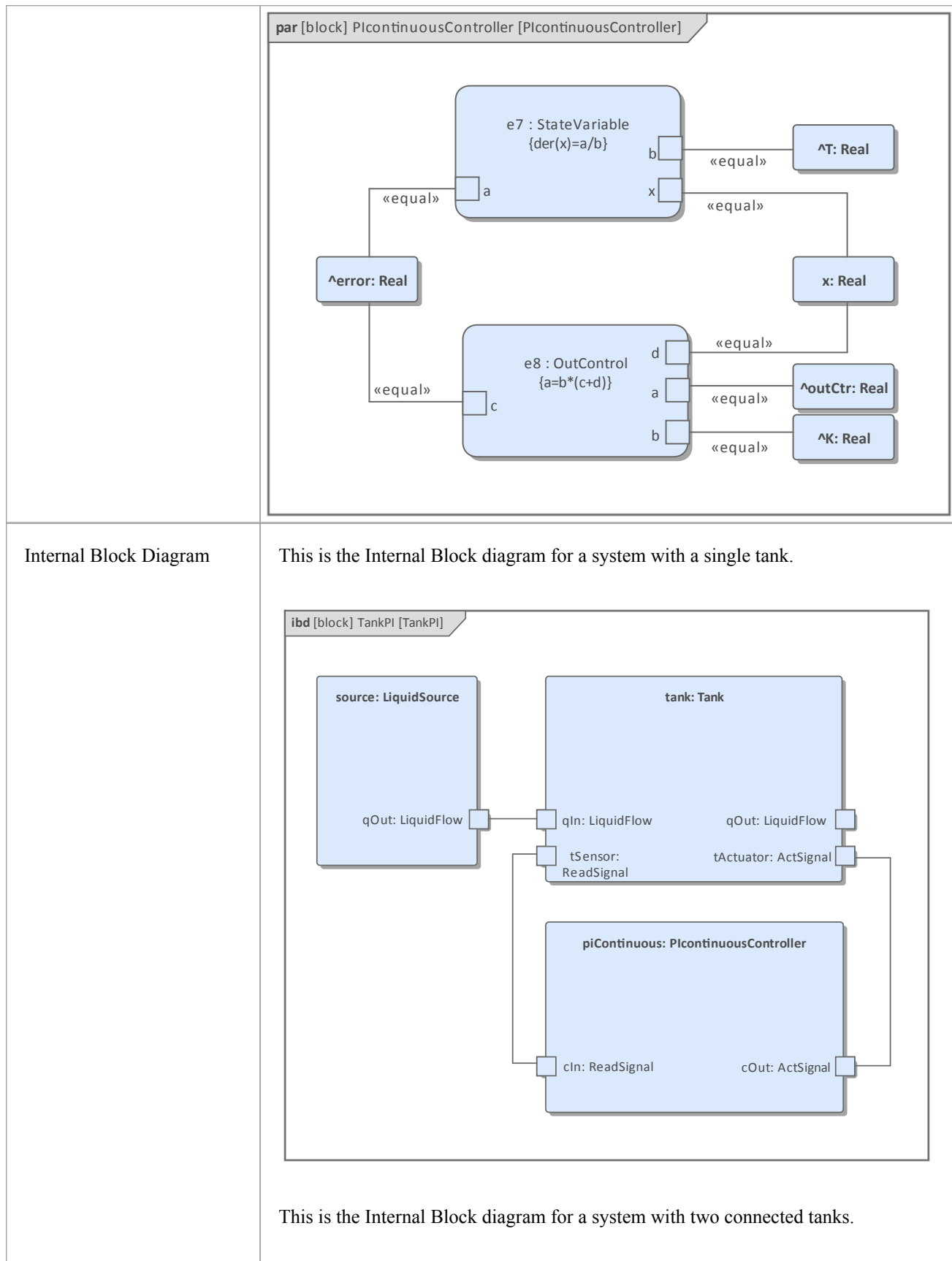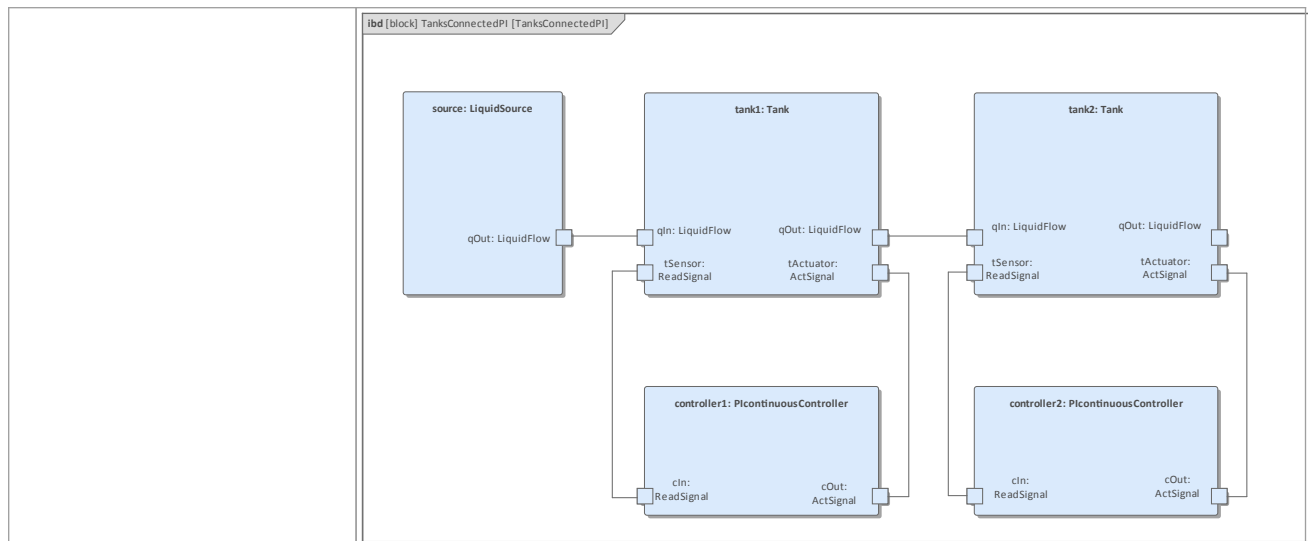
Run Simulation

Since *TankPI* and *TanksConnectedPI* are defined as 'SysMLSimModel', they will be filled in the combo box of 'Model' on the 'Simulation' page.

Select *TanksConnectedPI*, and observe these GUI changes happening:

- 'Data Set' combobox: will be filled with all the data sets defined in *TanksConnectedPI*
- 'Dependencies' list: will automatically collect all the Blocks, Constraints, SimFunctions and ValueTypes directly or indirectly referenced by *TanksConnectedPI* (these elements will be generated as Modelica code)
- 'Properties to Plot': a long list of 'leaf' variable properties (that is, they don't have properties) will be collected; you can choose one or multiple to simulate, and they will become legends of the plot

Create Artifact and Configure

Select 'Simulate > SysMLSim > Manage > SysMLSim Configuration Manager'

The elements in the Package will be loaded into the Configuration Manager.

Configure these Blocks and their properties as shown in this table.

Note: Properties not configured as 'SimConstant' are 'SimVariable' by default.
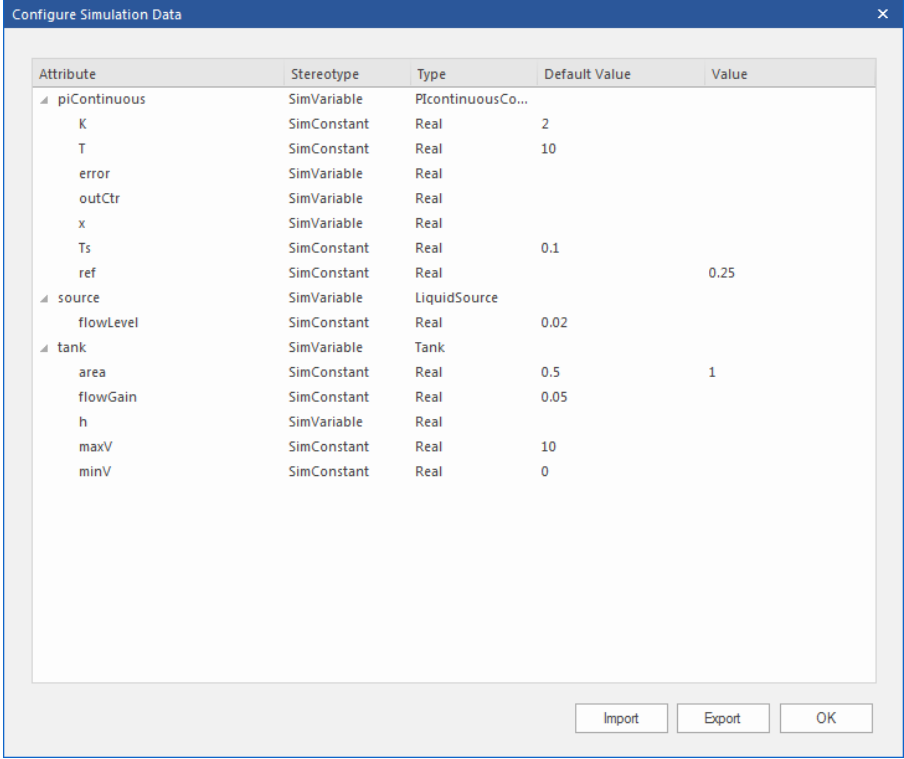
Block	Properties
LiquidSource	Configure as 'SysMLSimClass'. Properties configuration: • flowLevel: set as 'SimConstant'
Tank	Configure as 'SysMLSimClass'. Properties configuration: • area: set as 'SimConstant'

	• flowGain: set as 'SimConstant' • maxV: set as 'SimConstant' • minV: set as 'SimConstant'
BaseController	Configure as 'SysMLSimClass'. Properties configuration: • K: set as 'SimConstant' • T: set as 'SimConstant' • Ts: set as 'SimConstant' • ref: set as 'SimConstant'
PIcontinuousController	Configure as 'SysMLSimClass'.
TankPI	Configure as 'SysMLSimModel'.
TanksConnectedPI	Configure as 'SysMLSimModel'.

Setup DataSet

Right-click on each element, select the 'Create Simulation Dataset' option, and configure the datasets as shown in this table.

Element	Dataset
LiquidSource	flowLevel: 0.02
Tank	h.start: 0 flowGain: 0.05 area: 0.5 maxV: 10 minV: 0
BaseController	T: 10 K: 2 Ts: 0.1
PIcontinuousController	No configuration needed. By default, the specific Block will use the configured values from super Block's default dataSet.
TankPI	What is interesting here is that the default value could be loaded in the 'Configure Simulation Data' dialog. For example, the values we configured as the default dataSet on each Block element were loaded as default values for the properties of TankPI. Click the icon on each row to expand the property's internal structures to arbitrary depth.

Attribute	Stereotype	Type	Default Value	Value
⊿ piContinuous	SimVariable	PIcontinuousCo...		
K	SimConstant	Real	2	
T	SimConstant	Real	10	
error	SimVariable	Real		
outCtr	SimVariable	Real		
x	SimVariable	Real		
Ts	SimConstant	Real	0.1	
ref	SimConstant	Real		0.25
⊿ source	SimVariable	LiquidSource		
flowLevel	SimConstant	Real	0.02	
⊿ tank	SimVariable	Tank		
area	SimConstant	Real	0.5	1
flowGain	SimConstant	Real	0.05	
h	SimVariable	Real		
maxV	SimConstant	Real	10	
minV	SimConstant	Real	0	

Import   Export   OK

Click on the OK button and return to the Configuration Manager. Then these values are configured:

- tank.area: 1 this overrides the default value 0.5 defined in the Tank Block's data set
- piContinuous.ref: 0.25

TanksConnectedPI	• controller1.ref: 0.25 • controller2.ref: 0.4

Simulation and Analysis 1

Select these variables and click on the Solve button. This plot should prompt:

- source.qOut.lflow
- tank1.qOut.lflow
- tank1.h
- tank2.h
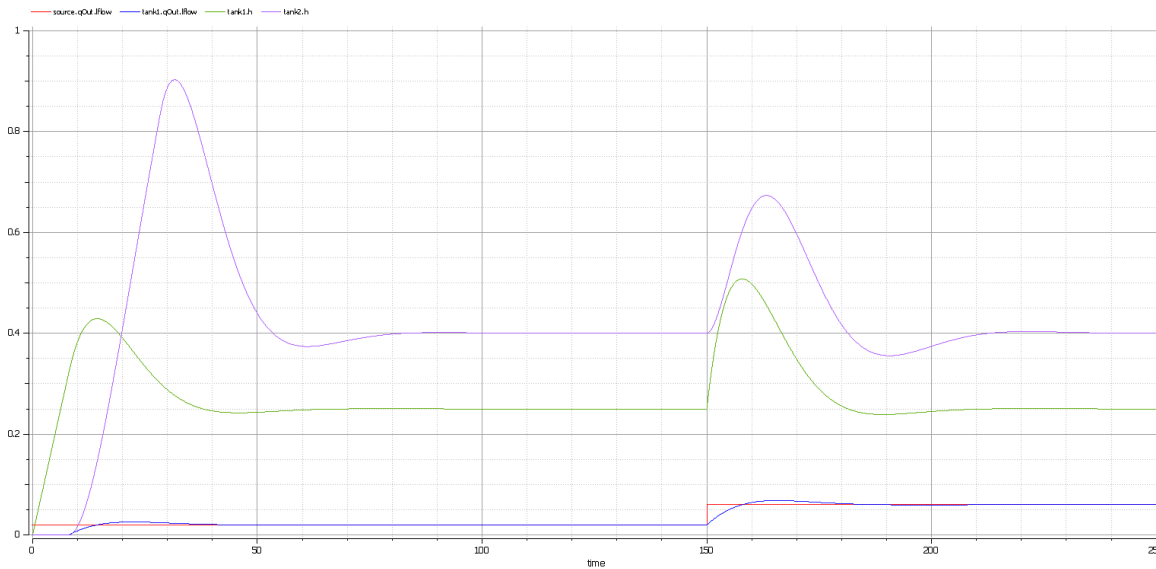
Here are the analyses of the result:

- The liquid flow increases sharply at time=150, to 0.06 m³/s, a factor of three of the previous flow level (0.02 m³/s)

- Tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through the data set)

- Both tank1 and tank2 regulated twice during the simulation; the first time regulated with the flow level 0.02 m³/s; the second time regulated with the flow level 0.06 m³/s

- Tank2 was empty before flow came out from tank1

Simulation and Analysis 2

We have set the tank's properties 'minV' and 'maxV' to values 0 and 10, respectively, in the example.

In the real world, a flow speed of 10 m³/s would require a very big valve to be installed on the tank.

What would happen if we changed the value of 'maxV' to 0.05 m³/s ?

Based on the previous model, we might make these changes:

- On the existing 'DataSet_1' of TanksConnectedPI, right-click and select 'Duplicate DataSet', and re-name to 'Tank2WithLimitValveSize'

- Click on the button to configure, expand 'tank2' and type '0.05' in the 'Value' column for the property 'maxV'

- Select 'Tank2WithLimitValveSize' on the 'Simulation' page and plot for the properties

- Click on the Solve button to execute the simulation
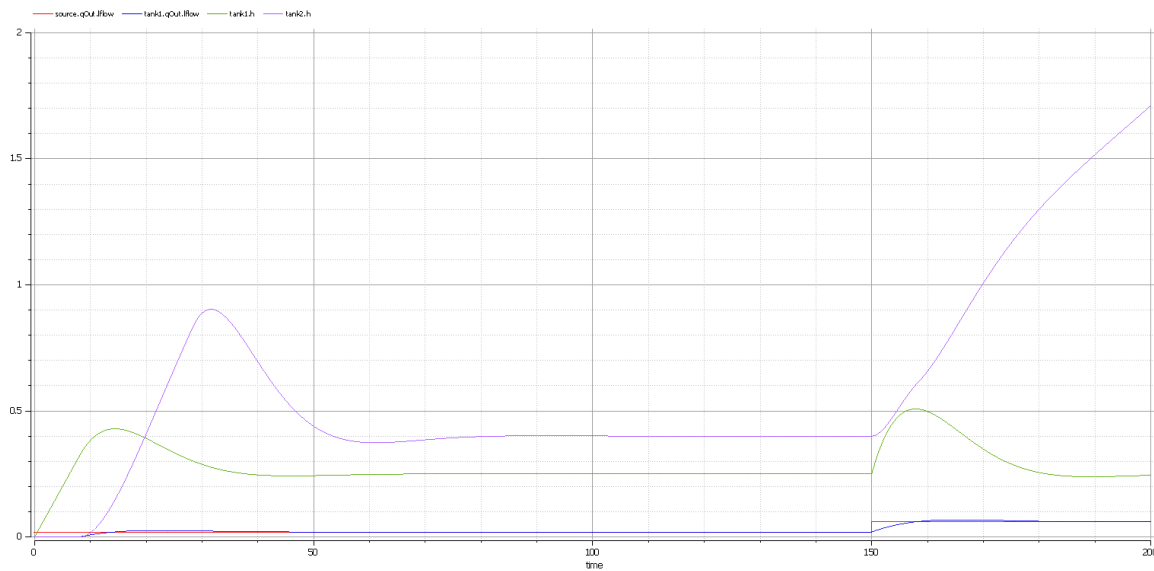
Here are the analyses of the result:

- Our change only applies to tank2; tank1 can regulate as before on 0.02 m³/s and 0.06 m³/s

- When the source flow is 0.02 m³/s, tank2 can regulate as before

- However, when the source flow increases to 0.06 m³/s, the valve is too small to let the out flow match the in flow; the only result is that the water level of tank2 increases

- It is then up to the user to fix this problem; for example, change to a larger valve, reduce the source flow or make an extra valve

In summary, this example shows how to tune the parameter values by duplicating an existing DataSet.

Troubleshooting OpenModelica Simulation

Common Simulation Issues

This table describes some common issues that can prevent a model being simulated. Check the output in the 'Build' tab of the System Output window. The messages are dumped from the OpenModelica compiler (omc.exe), which normally points you to the lines of the Modelica source code. This will help you locate most of the errors.

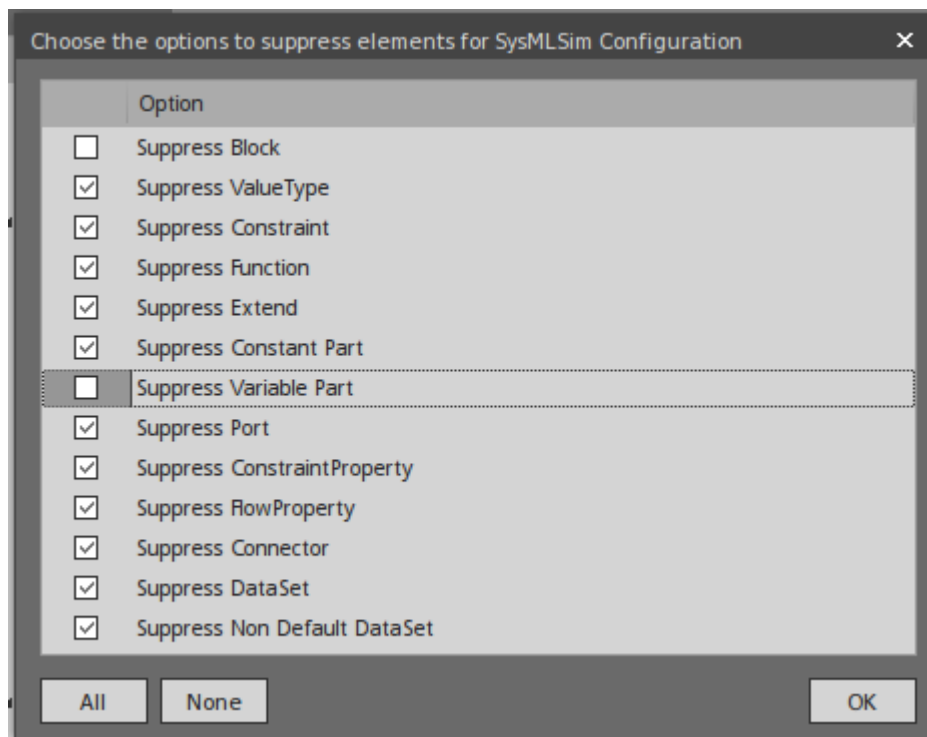| Issue |
| --- |
| The number of equations is less than the number of variables. You might have forgotten to set some properties to 'SimConstant', which means the value doesn't change during simulation. You might have to provide the 'SimConstant' property values before the simulation is started. (Set the values through a Simulation Data Set.) |
| The Blocks that are typing to Ports might not contain conserved properties. For example, a Block 'ChargePort' contains two parts — 'v : Voltage' and 'i: Current'. The property 'i : Current' should be defined as SimVariable with the attribute 'isConserved' set to 'True'. |
| SimConstants might not have default values — they should be provided with them. |
| A SimVariable might not have an initial value to start with — one should be provided. |
| The properties might be typed by elements (Blocks or Value Type) external to the configured Package; use a Package Import connector to fix this. |

SysML Simulation Configuration Filters

The 'SysML Simulation Configuration' dialog shows all the elements in the Package by default, including Value Types, Blocks, Constraint Blocks, Parts and Ports, Constraint Properties, Connectors, Constraints and Data Sets. For a medium-sized model, the full list can be quite long and it can be difficult for the user to find a potential modeling error.

In the TwoTanks example, if we clear the Tank.area property 'SimConstant' and then do a validation, we will find this error:

Error: Too few equations, under-determined system. The model has 11 equation(s) and 13 variable(s).

This error indicates that we might have forgotten to set some properties to 'SimConstant'.

What we can do now is click on the second button from the right on the toolbar (Filter for the configuration) and open the dialog shown here. Click on the All button, then deselect the 'Suppress Block' and 'Suppress Variable Part' checkboxes and click on the OK button.

Now we will have a much shorter list of variables, from which we can find that 'area' does not change during simulation. Then we define this as a 'SimConstant' and provide an initial value to fix the issue.

Model Validation Examples

Message	Discussion
Variable not defined in Constraint	In the TwoTanks example, when we browse to 'constraintBlock.Outcontrol.Constraint', suppose we find a typing error: we typed 'v' instead of 'b' in the constraint.
	So, instead of:
	$a=b*(c+d)$
	We typed:
	$a=v*(c+d)$
	Click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab:
	Validating model...
	Error: Variable v not found in scope OutControl. (Expression: " a=v(c+d);")*
	Error: Error occurred while flattening model TanksConnectedPI
	Number of Errors and Warnings found: 2
	Double-click on the error line; the configuration list displays with the constraint highlighted.
	Change 'v' back to 'b' and click on the Validate button again. No errors should be found and the issue is fixed.
	Tips: Using the SysML Simulation Configuration view is a shortcut way of

	changing the constraints for a Block or Constraint Block. You can:
	• Change a constraint in place
	• Delete using the context menu of a constraint
	• Add a new constraint using the context menu of a Block or Constraint Block
Duplicate Variable Names	In the TwoTanks example, browse to *block.tank.constraintProperty.e1*. Suppose we gave two properties the same name: • Right-click on *e1*, select 'Find in Project Browser', and change the name to *e2;* reload the 'SysML Simulation Configuration' dialog Click on the Validate button on the toolbar; these error messages appear in the 'Modelica' tab: *Validating model...* *Error: Duplicate elements (due to inherited elements) not identical: (Expression: "SensorValue e2;")* *Error: Error occurred while flattening model TanksConnectedPI* *Number of Errors and Warnings found: 2* Double-click on the error line; the configuration list displays with the constraint properties highlighted. Change the name of one of them from *e2* back to *e1* and click on the Validate button again; no errors should be found and the issue is fixed.
Properties defined in Constraint Blocks not used	In the TwoTanks example, in the Project Browser, we browse to the element 'Example Model.Systems Engineering.ModelicaExamples.TwoTanks.constraints.OutFlow'. Suppose we add a property '*c*' and potentially a new constraint, but we forget to synchronize for the instances — the constraint properties. This will cause a *Too few equations, under-determined system* error if we don't run validation. Reload the Package in the 'SysML Simulation Configuration' dialog and click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab: *Validating model...* *Error: ConstraintProperty 'e4' is missing parameters defined in the typing ConstraintBlock 'OutFlow'. (Missing: c)* *Error: Too few equations, under-determined system. The model has 11 equation(s) and 12 variable(s).* *Number of Errors and Warnings found: 2* Double-click on the error line; the configuration list displays with the constraint property highlighted. The constraint property is typed to *outFlow* and the new parameter '*c*' is missing. Right-click on the constraint property in the configuration list, select 'Find in all Diagrams', then right-click on the Constraint property on the diagram and select 'Features | Parts / Properties' and select the 'Show Owned / Inherited' checkbox, then click on 'c'. Reload the model in the 'SysML Simulation Configuration' dialog and click on the Validate button. These error messages will appear in the 'Modelica' tab: *Validating model...* *Error: ConstraintProperty 'e4' does not have any incoming or outgoing binding connectors for parameter 'c'.* *Error: Too few equations, under-determined system. The model has 11*

	equation(s) and 12 variable(s).
	Number of Errors and Warnings found: 2
	In order to fix this issue, we can do either of two things based on the real logic:
	1. If the property 'c' is necessary in the Constraint Block and a constraint is defined by using 'c', then we need to add a property in the context of the constraint property and bind to the parameter 'c'.
	2. If the property 'c' is not required, then we can click on this property in the Constraint Block and press Ctrl+D. (The corresponding constraint properties will have 'c' deleted automatically.)