UML designer instructions

January 22, 2025

1 Create a New Poka Yoke UML model

Let us first open the Poka Yoke environment within UML Designer. A snapshot of it is shown in Fig. 1, with the main components highlighted. In order to create a new UML model,

1. Right click in the 'Model Explorer' View → New → UML project, and give it a name.
2. Right click on the project → Viewpoint selection → Check 'Poka Yoke' box.

The model will have a UML file by default, called `model.uml`. Within it, there is a model instance denoted `<Model> NewModel`.

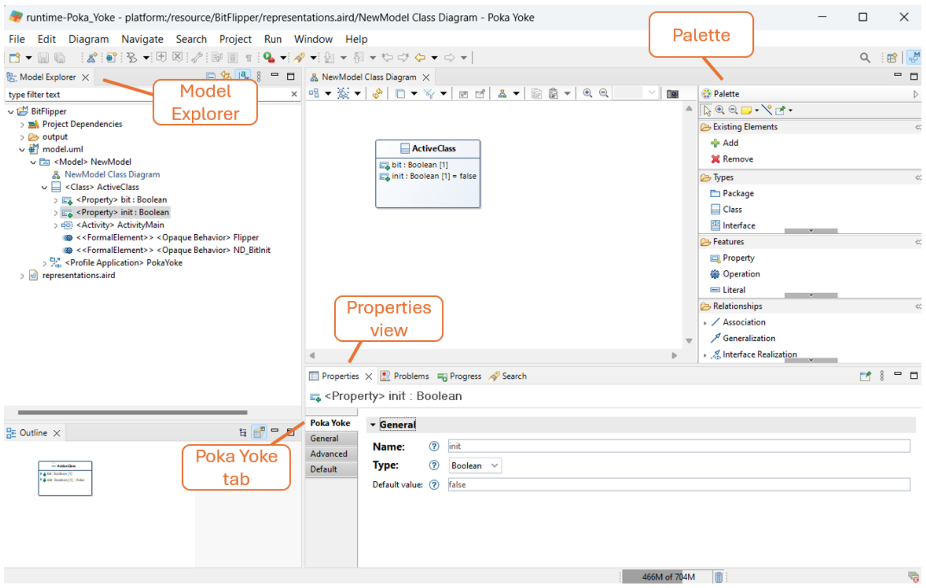
1.1 Create a Class

There are several ways of creating a class in UML designer. From the model explorer view,

1. Right click on the model instance NewModel → New child → Packaged Element → Class. You may want to rename it with a meaningful name.
2. If you need an active class, look in the properties view, 'Advanced' tab, check the 'Active' checkbox.

Graphical Approach. Alternatively, you may create a class from the Class Diagram view. To this end, you may

1. Right click on the model instance `<Model> NewModel` → New Representation → `NewModel` Class Diagram.
2. A new empty class diagram opens, with some small text mentioning 'Your diagram is empty. You can either [...]'. Double click on that text, and the Class Diagram now is blank.
3. On the right hand side, the Palette view contains several folders: Existing elements, Types, Features, Relationships.



runtime-Poka_Yoke - platform:/resource/BitFlipper/representations.aird/NewModel Class Diagram - Poka Yoke

<Property> init : Boolean

Name: init
Type: Boolean
Default value: false

Figure 1: A snapshot of the Poka Yoke environment.

4. To create a class, simply click on the Class element (within Types) and drag it to the blank diagram.

1.2 Create a Property

Properties represent the variables of a model. The supported types are boolean, enumeration and ranged integers (besides complex data types, see 1.3). Only the boolean type is native, and easy to instantiate.

1. Right click on the Class → New Child → Owned Attributes → Property

2. In the properties view, ‘Poka Yoke’ tab, select type you can select the type (e.g. boolean), and set it to a default value.

To create an Enumeration property, first we need to set up the enumeration variables. To do so,

1. Right click on the model → New Child → Owned Type → Enumeration

2. Right click on the enumeration → New Child → Owned Literal → Enumeration Literal

After the completion of these steps, we can create a new property of type Enumeration (with the given enumeration name).

To create an Integer property, first we need to set up the ranged interval variable. To do so,

1. Right click on the model → New Child → Owned Type → Primitive Type
2. In the properties view, within the 'Poka Yoke' tab, select Super type: Integer. Give it a name and write down the minimum and maximum values. Note that two constraints named min and max will appear in the model explorer view.

After the completion of these steps, we can create a new property of type Primitive Type (with the given primitive name).

Graphical Approach. Alternatively, you may create a property from the Class Diagram view. To this end,

1. From the Palette view, simply click on the Property element (within Features) and drag it inside a class. A popup window will open: fill in the Name, then click OK.
2. Double click on the property, in the Properties view, 'Poka Yoke' tab, select the property type from the drop-down menu.

Note that you still need to create an enumeration and integer variable via the standard approach before you can instantiate an integer or enumeration property.

1.3 Create a Data Type

Data types represent more abstract types, e.g. a robot.

1. Right click on the model → New Child → Owned Type → Data Type

Note that data types can have properties (of type boolean, integer, enumeration, or of other data type).

Graphical Approach. Alternatively, you may create a data type from the Class Diagram view. To this end, you may

1. To create a data type, go in the 'Palette' view, and click on the tiny arrow located at the left of Enumeration. A drop down menu will open, revealing DataType.
2. Click on DataType, and drag it to the Class diagram. A pop up window will appear, where you can change its name; finally, click OK.

1.4 Create an Abstract Activity

Activities can either be abstract (which will be synthesised by Poka Yoke) or concrete (a series of behaviors and actions created by the user). Let us consider abstract activities.

1. Right click on the Class → New Child → Classifier Behaviors → Activity
2. By default, the activity is concrete (containing user-defined opaque behaviors, control nodes, constraints).
3. For synthesis purposes, an abstract activity is required: in the properties view, 'General' tab, check the 'Abstract' checkbox.

A graphical representation of the activity is usually an activity diagram,

1. Right click on the activity → New Representation → Activity Diagram.

For abstract activities we need to add a precondition and a postcondition, defining respectively the necessary requirements at the beginning and the end of the the task execution. Note that there is no graphical way of creating an activity nor of adding pre and postconditions.

1.4.1 Create Preconditions

1. Right click on the activity → New Child → Precondition.
2. Right click on the Precondition → New Child → Constraint.
3. Right click on the Constraint → New Child → Specification → Opaque Expression.
4. Open the Opaque Expression. Delete the name, or leave it blank (this is optional, only for visualisation purposes).
5. In the properties view, 'Default' tab, double click on the 'Body' field.
6. In the new window, write on the 'Value' field your precondition constraint. Click Add, then OK.

We follow similar steps for the postcondition.

1.4.2 Create Postconditions

1. Right click on the activity → New Child → Postcondition.
2. Right click on the Postcondition → New Child → Constraint.
3. Right click on the Constraint → New Child → Specification → Opaque Expression.
4. Open the Opaque Expression. Delete the name, or leave it blank (this is optional, only for visualisation purposes).
5. In the properties view, 'Default' tab, double click on the 'Body' field.
6. In the new window, write on the 'Value' field the postcondition constraint. Click Add, then OK.

1.5 Create a Concrete Activity

The creation of a concrete activity starts in a similarly to the abstract activity:

1. Right click on the Class → New Child → Classifier Behaviors → Activity

Notice that you do not check the 'Abstract' checkbox.

A concrete activity is a user-defined sequence of actions, and it can be seen as a finite directed graph consisting of nodes and control flows. A node can be a *control node* (initial, final, fork, join, decision, or merge node), be an *call opaque behavior node* which executes an action by calling an opaque behavior, or be an *opaque action node* containing a guard and zero or more effects, thereby essentially 'inlining' an action. So likewise to UML opaque behaviors, also opaque action nodes are either deterministic or non-deterministic, and either atomic or non-atomic. Concrete UML activities cannot contain preconditions, postconditions, or occurrence constraints. Similarly to abstract activities, also concrete activities can be represented via an activity diagram (see 1.4).

1.5.1 Create a control node

After having created a concrete activity, adding a control node can be achieved by

1. Right click on the concrete activity → New Child → Node → Initial/ Activity Final/ Fork/ Join/ Decision/ Merge node

Note that the final node is denoted as 'Activity Final node'.

Graphical Approach. After having created the concrete activity, go to the activity diagram

1. From the palette view of the activity diagram, you can find the nodes: clicking on the little arrow next to 'Initial node' opens a drop-down menu with all types of nodes.

1.5.2 Create a call behavior node

After having created a concrete activity, adding a call behavior node can be achieved by

1. Right click on the concrete activity → New Child → Node → Call Behavior Action node
2. In the properties view, 'Poka Yoke' tab, you can add guards and effects for the call action.
3. Within the 'Default' tab, the 'Behavior' row allows you to select which behavior to call with a drop down menu.

Graphical Approach. From the palette view of the activity diagram, you can find the Call Behavior node under the 'Opaque Action' drop down menu.

1.5.3 Create an opaque action node

After having created a concrete activity, adding an opaque action node can be achieved by

1. Right click on the concrete activity → New Child → Node → Opaque Action node
2. In the properties view, 'Poka Yoke' tab, you can add guards and effects for the call action.

Graphical Approach. From the palette view of the activity diagram, you can directly find the Opaque Action node.

1.6 Create an Opaque Behavior

Opaque behaviors represents the action that an activity can perform. Opaque behavior can be atomic (the system cannot have any other operation in parallel to an atomic action) or non-atomic, and deterministic (only one effect) or non-deterministic (multiple effects, of which only one is chosen). For synthesis purposes, the user needs to create a behavior to be used in the Activity synthesis.

1. Right click on the Class → New Child → Owned Behaviors → Opaque Behavior.
2. For atomic actions, in the properties view, 'Poka Yoke' tab, check the 'Atomic' checkbox (optional step).
3. In the properties view, 'Poka Yoke' tab, in the 'Guard' field, write the guard (the conditions under which this action can be performed) of choice.
4. In the properties view, 'Poka Yoke' tab, in the 'Effects' field, write the effects (the new assignments that the action grants) of choice.
5. Nondeterministic effects are separated by three tilde ˜ signs. For example, the effect `val:=true ~~~ val:=false` means that the variable `val` will be non-deterministically assigned *either* true *or* false as a consequence of the current opaque behavior. Note that the user has no control over which effect is picked by the execution of the model.

Note that there is no graphical way of adding an opaque behavior.

1.7 Create Constraints

To create general constraints, it is sufficient to follow similar steps to the ones delineated in Sections 1.4.1, 1.4.2.

1.8 Create Occurrence Constraints

Occurrence constraints define the amount of times a specific UML element (e.g. activity, opaque behavior) can be called during the execution. As such, these are modeled as *interval* constraints in UML designer. We might think of a specific action that we need to use (i.e. the minimum of the occurrence constraint is greater or equal than one) and we do not want to use it more than three times (i.e. the maximum of the occurrence constraint is set to three). Let us assume we want to add occurrence constraints to an activity. From the Model explorer view,

1. Right click on the abstract activity → New Child → Owned Rule → Interval Constraint. You may want to change the name to something representative of the constraint. In the properties view, click on the 'Default' tab and add an element in the 'Constrained Element' field. To do so, just click in the empty field of the corresponding 'Value' column, and a small icon with three dots appears. Click on that icon, and select the constrained element from the list; click Add, and then OK.

2. We now need to define the minimum and maximum values for the occurrence constraint. Right click on the interval constraint → New Child → Specification → Interval. Again, you may want to rename it in a meaningful way.

3. We now insert a minimum and maximum value for the occurrence limits. To this end, we add a `Literal Integer` *at the Model level.* Right click on the Model → New Child → Packaged Elements → Literal Integer. For visualisation purposes, erase the name, and fill in the 'Value' field the value you need. Notice that you need two Literals, one for the minimum and one for the maximum occurrence limit.

4. Click on the Interval specification we have just defined. Within the Properties view, click on the 'Advanced' tab, there are the Max and Min fields. Click on the three-dots icon and select the maximum and minimum values represented by the two Literal Integers defined in the previous step.

1.9 Perform Synthesis

1. Right click on the `model.uml` → Perform Synthesis. A new `output` folder is created.

2. Open the output folder, scroll down until the last .uml file, open it, open the model, open the class, open the activity.

3. For the activity diagram representation, right click on the activity → New Representation → Activity Diagram.

1.10 Cameo Simulation

Note that *abstract* activities are not supported for the simulation part. You can however perform the synthesis of an abstract activity, and then translate the synthesised model into a Cameo model.

1. Right click on the `model.uml` $\rightarrow$ Translate UML to Cameo. A new `output` folder is created.
2. Open the output folder, open the folder called `model.uml` (i.e. the name of the original UML file), and there you find a new UML file.
3. Copy the file to Cameo, and simulate it.

2 Synthesis: Bit Flipper Example

Let us now create a bit flipper model. This model initialises a boolean bit non-deterministically, and it flips it if it is false; does not need to do anything otherwise. We can use the following steps:

1. Create a new UML project named BitFlipper.
2. Create an active class named ActiveClass.
3. Create two boolean properties, named `init` and `bit`. The default value of `init` is false, none for `bit`.
4. Create an abstract activity, with precondition `init = false`, and post-condition `bit = true`.
5. Create an initialisation opaque behavior, named `ND_BitInit`, with guard `init = false` and nondeterministic effects:
   `init := true, bit := true`
   `~~~`
   `init := true, bit:= false`.
   This action flags the initialisation as done (`init` is true after the action) and non-deterministically instantiates `bit`.
6. Create a bit flipper opaque behavior, with guard `init = true` and effects `bit := not bit`, named `Flipper`.
7. Perform the synthesis. Figure 2 shows the activity diagram of the synthesised activity.

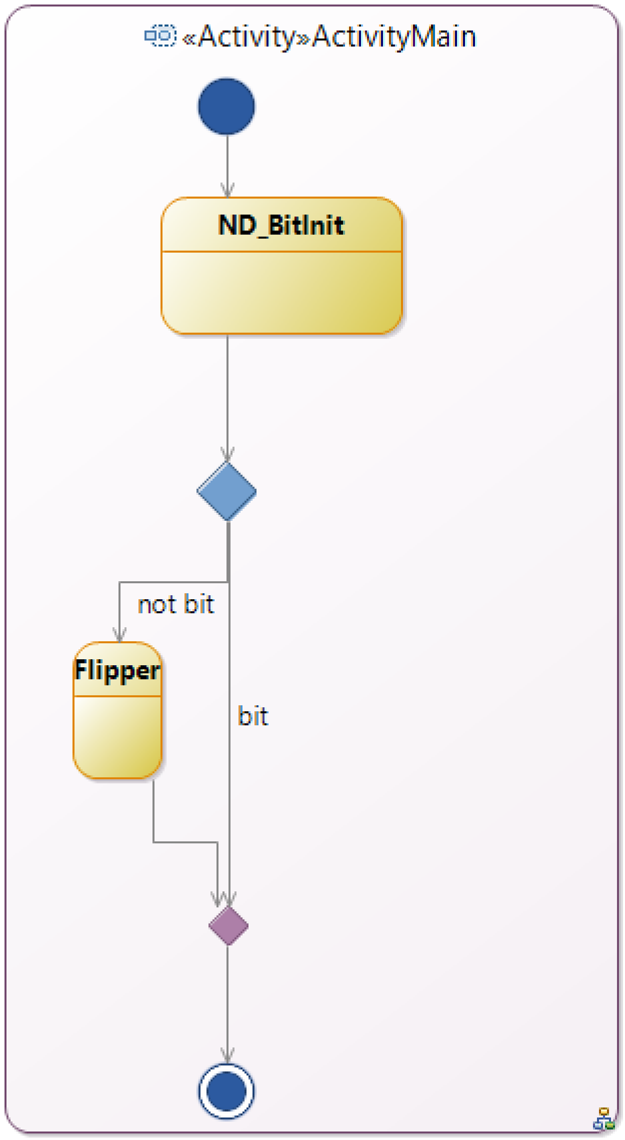


Figure 2: The activity diagram of the synthesised bit flipper.