PÓPULO

1. The tool.

Pópulo is a tool for executing and debugging UML 2.0 models, whose precise behaviour is specified by means of the UML action language. Using Pópulo, the design phase of a software development process would be as follows:

1. Software designers construct UML models of the system being developed. The precise and complete behaviour of these UML models is specified by meas of the UML action language (which is part of the UML 2.0 standard). Specific UML editors are not enforced by Pópulo, so each development team can continue using their preferred one.
2. These executable UML models are loaded into Pópulo. Using this tool, designers can execute and debug their models, discovering bugs, if there were, at design time. Then, these errors and inaccuracies can be fixed at design time, before moving into an implementation, where it would be more costly.

Pópulo uses the UML action language contained in the UML standard for specifying the precise behaviour of a model. The use of the UML action language helps to ensure compatibility with third-party UML editors (e.g. MagicDraw or Rational) and enables the extension of Pópulo for executing new actions, in case the UML action language were extended by means of UML Profiles.

Next subsections give a general overview of the current status of the implementation and the facilities offered by Pópulo for debugging UML models.

2. Coverage of the UML standard.

Pópulo supports currently the execution of UML activities and actions. Pópulo is able to execute the 80% of the actions contained in the UML standard. Creation of objects, reading and writing of attributes, calls to static and non-static methods, object comparison, reading and writing of variables, creation of links and link objects, reading and writing of as links (instances of associations) as qualified links (instances of qualified associations) as object links (instances of association classes) and raising of exceptions are supported. Only the management of signals as well as some actions that are rarely used, such as StartClassifierBehaviour, are not currently implemented.

Pópulo supports extra actions, not contained in the UML standard, for the carrying out common operations on basic data types, such as integers, reals, strings or booleans. These extensions are provided as UML Profiles and reusable model libraries for being used in the modelling of applications that will be executed in Pópulo.

Regarding activities, Pópulo supports basic object and control flows, fork and join nodes for parallel execution, decision nodes and merge nodes for alternative control flows (whose guards are specified by means of actions), structured activities and variables (whose scope is the structured activity), loops and exception handlers.

As it could be expected of an object-oriented model, message calls are executed using a late-binding schema, being the target method of a message determined at runtime based on the actual type of the called object, in order to deal properly with polymorphism.

3. Pópulo facilities.

Pópulo is currently provided as an Eclipse plugin. Figure 1 shows a screenshot of the tool. This plugin assist UML designers on the debugging of their models. Pópulo provides several views, which are commented below:

Instances and links (Figure 1, label 1 and label 2). It shows all the created objects, grouped by the class they belongs to. These objects include as instances of classes as instances of associations and association classes, i.e. links and link objects.

Call stack (Figure 1, label 3). It visualises the current call stack, i.e. the sequence of activities and structured nodes that have been called and that have not still ended.

Ready and block (Figure 1, label 4 and label 5). The ready tab shows the queue of actions that currently have received all their object and control flows, and therefore they are ready to be executed. The user can modify the ordering of this queue, imposing a specific execution scheduling. For instance, he/she might give priority to the execution paths in which he/she is mainly interested, avoiding the execution of already tested or debugged execution flows. The blocked tab shows all the actions that are waiting for some control or object flow.

Breakpoints (Figure 1, label 6). It allows the definition of breakpoints for the model execution.

Model (Figure 1, label 7). It displays the model being executed using the Eclipse UML2 plugin. However, UML models do not require having been created using this plugin.

Execution Trace (Figure 1, label 8). Information about the execution of the model is shown in this view. For instance, actions executed are shown. For each action, it is also displayed the activity that contains it or the target actions that receive its outputs. The amount of information to be shown can be customized just pressing some buttons, deciding the end-user how many details he/she needs at every time. This window automatically splits when a fork node is reached, showing the parallel execution flows in separated parts of the window, in order to provide a better visualisation. Pópulo, likewise common debuggers, supports breakpoint-based execution and step-by-step execution.

Object properties (Figure 1, label 9). It displays the values of the attributes of an object or the variable.

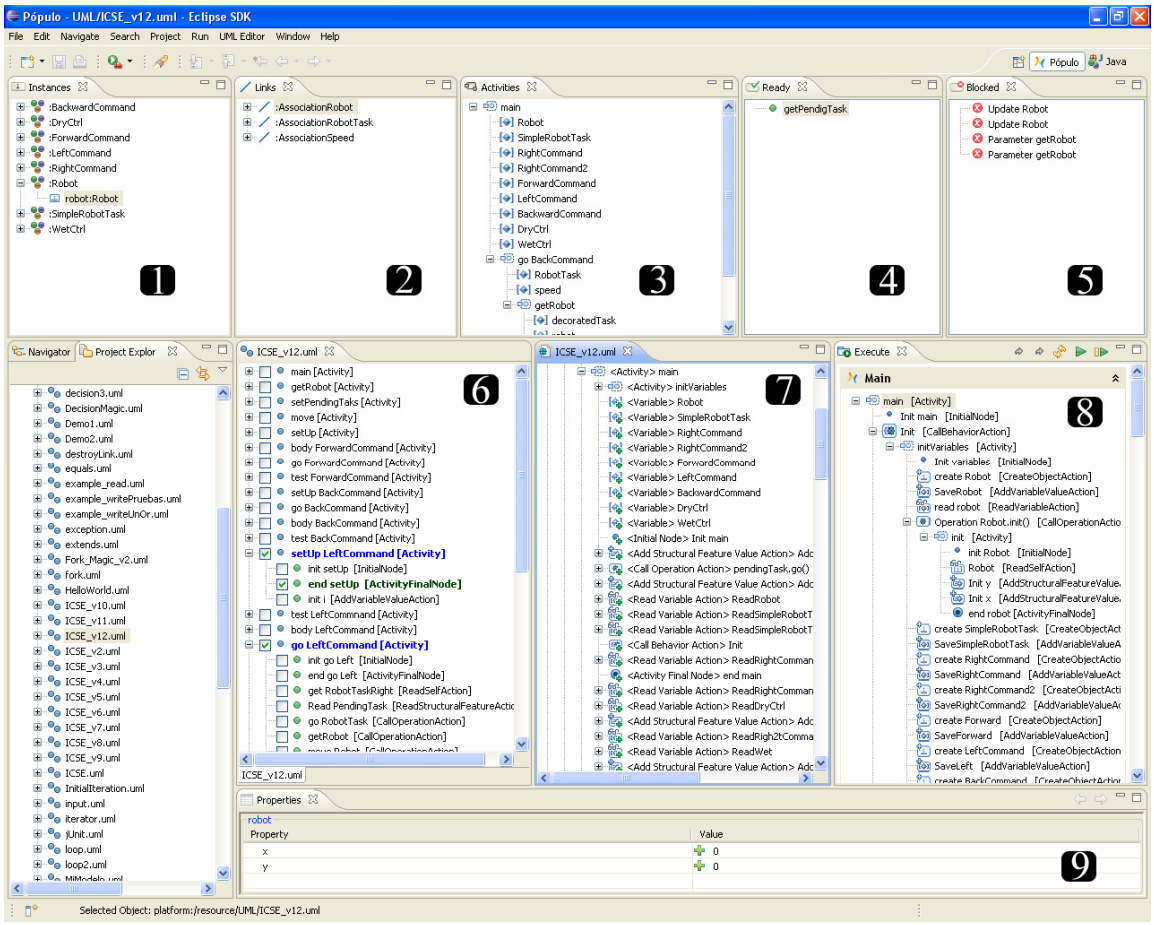
Pópulo - UML/ICSE_v12.uml - Eclipse SDK
File Edit Navigate Search Project Run UML Editor Window Help
Instances
:BackwardCommand
:DryCtrl
:ForwardCommand
:LeftCommand
:RightCommand
:Robot
robot:Robot
:SimpleRobotTask
:WetCtrl
1
Links
:AssociationRobot
:AssociationRobotTask
:AssociationSpeed
2
Activities
main
Robot
SimpleRobotTask
RightCommand
RightCommand2
ForwardCommand
LeftCommand
BackwardCommand
DryCtrl
WetCtrl
go BackCommand
RobotTask
speed
getRobot
decoratedTask
3
Ready
getPendigTask
4
Blocked
Update Robot
Update Robot
Parameter getRobot
Parameter getRobot
5
Navigator
Project Explor
decision3.uml
DecisionMagic.uml
Demo1.uml
Demo2.uml
destroyLink.uml
equals.uml
example_read.uml
example_writePruebas.uml
example_writeUnOr.uml
exception.uml
extends.uml
Fork_Magic_v2.uml
fork.uml
HelloWorld.uml
ICSE_v10.uml
ICSE_v11.uml
ICSE_v12.uml
ICSE_v2.uml
ICSE_v3.uml
ICSE_v4.uml
ICSE_v5.uml
ICSE_v6.uml
ICSE_v7.uml
ICSE_v8.uml
ICSE_v9.uml
ICSE.uml
InitialIteration.uml
input.uml
iterator.uml
jUnit.uml
loop.uml
loop2.uml
ICSE_v12.uml
main [Activity]
getRobot [Activity]
setPendingTaks [Activity]
move [Activity]
setUp [Activity]
body ForwardCommand [Activity]
go ForwardCommand [Activity]
test ForwardCommand [Activity]
setUp BackCommand [Activity]
go BackCommand [Activity]
body BackCommand [Activity]
test BackCommand [Activity]
setUp LeftCommand [Activity]
init setUp [InitialNode]
end setUp [ActivityFinalNode]
init i [AddVariableValueAction]
test LeftCommnand [Activity]
body LeftCommand [Activity]
go LeftCommand [Activity]
init go Left [InitialNode]
end go Left [ActivityFinalNode]
get RobotTaskRight [ReadSelfAction]
Read PendingTask [ReadStructuralFeatureActic
go RobotTask [CallOperationAction]
getRobot [CallOperationAction]
6
ICSE_v12.uml
<Activity> main
<Activity> initVariables
<Variable> Robot
<Variable> SimpleRobotTask
<Variable> RightCommand
<Variable> RightCommand2
<Variable> ForwardCommand
<Variable> LeftCommand
<Variable> BackwardCommand
<Variable> DryCtrl
<Variable> WetCtrl
<Initial Node> Init main
<Call Operation Action> pendingTask.go()
<Call Behavior Action> Init
<Activity Final Node> end main
<Read Variable Action> ReadRobot
<Read Variable Action> ReadDryCtrl
<Read Variable Action> ReadWet
7
Execute
Main
main [Activity]
Init main [InitialNode]
Init [CallBehaviorAction]
initVariables [Activity]
Init variables [InitialNode]
create Robot [CreateObjectAction]
SaveRobot [AddVariableValueAction]
read robot [ReadVariableAction]
Operation Robot.init() [CallOperationActio
init [Activity]
init Robot [InitialNode]
Robot [ReadSelfAction]
end robot [ActivityFinalNode]
create SimpleRobotTask [CreateObjectAct
SaveSimpleRobotTask [AddVariableValueA
create RightCommand [CreateObjectActio
SaveRightCommand [AddVariableValueAct
create RightCommand2 [CreateObjectActi
SaveRightCommand2 [AddVariableValueA
create Forward [CreateObjectAction]
SaveForward [AddVariableValueAction]
create LeftCommand [CreateObjectAction
SaveLeft [AddVariableValueAction]
8
Properties
robot
Property
Value
x
0
y
0
9
Selected Object: platform:/resource/UML/ICSE_v12.uml

Figure 1