

1. ATL Transformation Example

1.1. Example: UMLActivityDiagram → MSProject

The UMLActivityDiagram to MSProject example describes a transformation from a loop free UML activity diagram (describing some tasks series) to a MS Project. The transformation is based on a simplified subset of the UML State Machine metamodel. This transformation produces a project defined in conformance to a limited subset of the MSProject metamodel.

1.1.1. Transformation overview

The aim of this transformation is to generate an MSProject project from a UML Activity Diagram. The composition of the input activity diagram is restricted to “initial”, “fork” and “join” pseudostates and actionstates. Moreover, the activity diagram must be loop free in order to be transformable into a project.

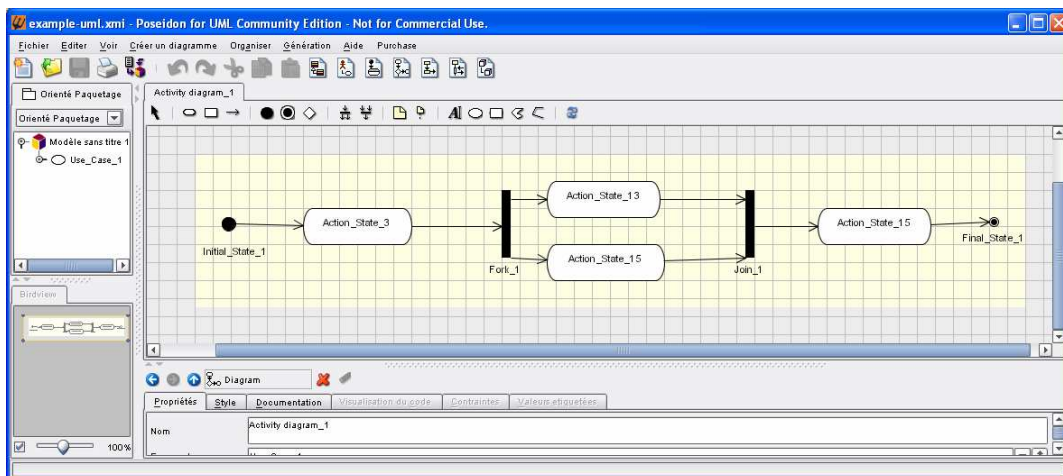


Figure 1. A UML activity diagram example

As an example of the transformation, Figure 2 provides a screen capture of an MSProject project generated from the initial activity diagram presented in Figure 1.

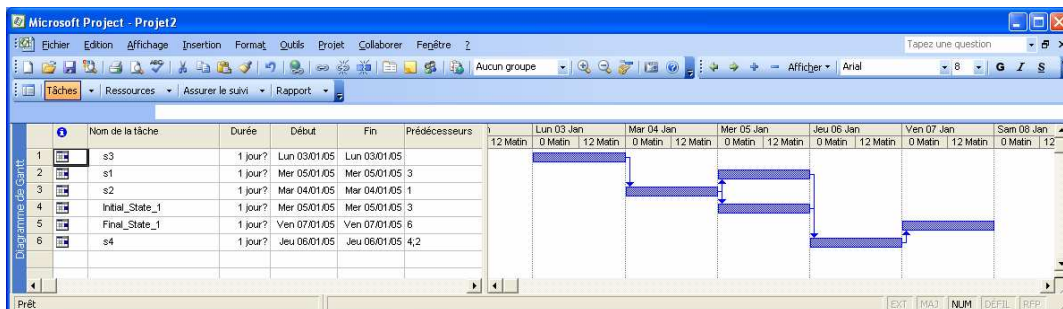


Figure 2. The corresponding MSProject project

1.2. Metamodels

This transformation is based on a simplified subset of the UML State Machine metamodel [1] which only deals with information that is relevant in the scope of this transformation. The considered metamodel is presented in Figure 3.

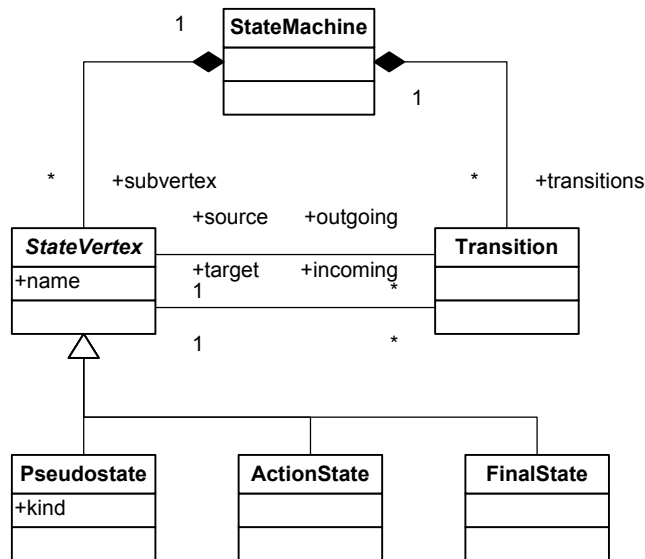



Figure 3. The UMLActivityDiagram metamodel

A UMLActivityDiagram is modeled by a StateMachine element. This element contains both StateVertex (i.e. state) and Transition elements. An abstract StateVertex can be either a FinalState, an ActionState (i.e. a state within the diagram), or a Pseudostate of different kinds:

- "initial" Pseudostate: an initial state (with no incoming Transitions);
- "fork" Pseudostate: a state with a single incoming transition and several outgoing ones;
- "join" Pseudostate: a state with several incoming transitions and a single outgoing one.

Each Transition is associated with an incoming and an outgoing StateVertex. According to its type, a StateVertex can have none to several incoming and outgoing Transitions.

	ATL TRANSFORMATION EXAMPLE	
	UmlActivityDiagram to MSProject	Date 04/04/2005

The transformation also relies on a simple Project definition [2]. The metamodel considered here is described in Figure 4, and provided in Appendix I in km3 format [3].

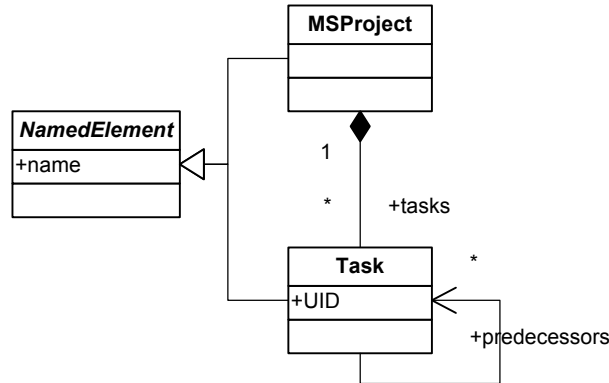


Figure 4. The MSProject metamodel

Within this metamodel, a project is associated with a MSProject element. Such an element contains Task, identified by a unique UID attribute, which can have predecessors among other defined Tasks.

1.3. Rules Specification

These are the rules to transform a UMLActivityDiagram model to a MSProject model:

- For the root StateMachine element, a MSProject element is created.
- For each “initial” Pseudostate, a Task element without predecessors is created. “join” and “fork” Pseudostates are not associated with output Tasks.
- For each ActionState or FinalState, a Task element is created. The set of its predecessors contains the Tasks associated with ActionStates and “initial” Pseudostates that point to the current state directly or through series of “join” or “fork” Pseudostates.

1.4. ATL Code

This ATL code for the UMLActivityDiagram to MSProject transformation consists of 3 helpers and 3 rules. The curId helper is an integer variable used to provide an incremented UID to each generated Task of the output Project model. The is performed by the getId() helper, which makes use of the curId variable. Each time it is called, this helper increments the curId value and returns its new value into a String.

The getPredecessors() helper computes the set of predecessors of a given task. For this purpose, it navigates the Transition pointing to the current state. If the other end of the Transition is an “initial” Pseudostate, or an ActionState, it returns the associated Task as the unique predecessor. Otherwise (if the previous state is either a “fork” or a “join”), it recursively performs the predecessors computing on the previous state.


The rule Main allocates a MSProject element. This element is linked to all the Tasks elements to be generated during the transformation.

The rule Pseudostate allocates a Task element for each Pseudostate of “initial” type. This new Task is given a unique UID, the name of the source Pseudostate, and an empty set of predecessors.

The rule StateVertex allocates a new Task element for each ActionState and FinalState in the input model. This new Task is given a unique UID and the name of the source state. Its predecessors set is computed by the getPredecessos() helper.

```
1  module UmlActivityDiagram2MSProject;
2  create OUT : MSProject from IN : UML;
3
4
5  -----
6  -- HELPERS -----
7  -----
8
9  -- This helper stores an integer value used to assign an ID to each generated
10 -- Task.
11 -- CONTEXT:  thisModule
12 -- RETURN: Integer
13 helper def: curId : Integer = 0;
14
15 -- This helper computes the value to be assigned to the ID field of a generated
16 -- Task. It increments the value stored by the "curId" helper and returns its
17 -- new value as a String.
18 -- CONTEXT:  thisModule
19 -- RETURN: String
20 helper def: getId() : String =
21     thisModule.refSetValue('curId', thisModule.curId + 1).curId.toString();
22
23
24 -- This helper computes the set of StateVertex that will be associated with
25 -- the predecessors of a Task.
26 -- The computed set contains all the ActionState and Pseudostate of "initial"
27 -- type (ie. the initial state) that point to the current StateVertex
28 -- (the context one) either directly or through "fork" and "join" Pseudostate.
29 --
30 -- WARNING: this helper is not able to deal with loops. However, there shall
31 -- be no loops in a diagram dedicated to the representation of a project.
32 --
33 -- CONTEXT:  UML!StateVertex
34 -- RETURN: Set(UML!StateVertex)
35 helper context UML!StateVertex def: getPredecessors() : Set(UML!StateVertex) =
36     let trans : Set(UML!StateVertex) = self.incoming in
37
38     if trans.isEmpty() then
39         Set{}
40     else trans->collect(t | t.source)
41         ->iterate(e; ret : Set(UML!StateVertex) = Sequence{} |
42             if e.ocIsKindOf(UML!ActionState) then
43                 ret->including(e)
44             else
45                 if e.ocIsKindOf(UML!Pseudostate) then
46                     if e.kind = #pk_initial then
47                         ret->including(e)
48                     else
49                         ret->including(e.getPredecessors())
50                     endif
51                 else
52                     ret
53                 endif
54             endif
55         )
56     endif;
```

```
57
58
59
60 -----
61 -- RULES -----
62 -----
63
64 -- Rule 'Main'
65 -- This rule generates the Project element. Contained tasks are those
66 -- associated with:
67 -- * UML Final State
68 -- * UML Action State
69 -- * UML Pseudostate of "initial" kind.
70 rule Main {
71     from
72         s : UML!StateMachine
73     to
74         pro : MSProject!MSProject (
75             tasks <- UML!StateVertex.allInstances()
76         )
77 }
78
79 -- Rule 'Pseudostate'
80 -- This rule generates a Task for the Pseudostate of "initial" type (that is,
81 -- the diagram initial state).
82 -- The generated initial Task has no predecessors (sine it corresponds to the
83 -- initial state of the UML activity diagram).
84 rule Pseudostate {
85     from
86         s : UML!Pseudostate (
87             s.kind = #pk_initial
88         )
89     to
90         t : MSProject!Task (
91             UID <- thisModule.getId(),
92             name <- s.name,
93             predecessors <- Set{}
94         )
95 }
96
97 -- Rule 'StateVertex'
98 -- This rule generates Tasks for both ActionStates and FinalStates.
99 -- The set of predecessors of a Task is computed by the getPredecessors helper.
100 -- It corresponds to the set of ActionState/"initial" Pseudostate pointing to
101 -- the current state directly, or through one or several "fork" and "join"
102 -- Pseudostates.
103 rule StateVertex {
104     from
105         s : UML!StateVertex (
106             s.oclIsKindOf(UML!FinalState)
107             or s.oclIsKindOf(UML!ActionState)
108         )
109     to
110         t : MSProject!Task (
111             UID <- thisModule.getId(),
112             name <- s.name,
113             predecessors <- s.getPredecessors()
114         )
115 }
```

	ATL TRANSFORMATION EXAMPLE	
	UmlActivityDiagram to MSProject	Date 04/04/2005


I. MSProject metamodel in km3 format

```
package MSProject {
  class MSProject {
    reference tasks[1-*] container : Task;
  }

  abstract class NamedElement {
    attribute name : String;
  }

  class Task extends NamedElement {
    attribute UID : String;
    reference predecessors[*] : Task;
  }
}

package PrimitiveTypes {
  datatype String;
}
```

	ATL TRANSFORMATION EXAMPLE	
	UmlActivityDiagram to MSProject	Date 04/04/2005

References

- [1] Unified Modeling Language (UML), version 1.5.
<http://www.omg.org/technology/documents/formal/uml.htm>.
- [2] Overview of XML for Project. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/pjsdk/html/pjxml_overview_HV01051036.asp.
- [3] KM3: Kernel MetaMetaModel. Available at <http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/doc/atl/index.html>.