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](image1.png)

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](image2.png)
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.

A UMLActivityDiagram is modelized 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.

Figure 3. The UMLActivityDiagram metamodel
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].

![Diagram of MSProject metamodel](image)

**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 getPredecessors() helper.

```atl
module UmlActivityDiagram2MSProject;
create OUT : MSProject from IN : UML;

-- HELPERS
-- This helper stores an integer value used to assign an ID to each generated Task.
-- CONTEXT: thisModule
-- RETURN: Integer
helper def: curId : Integer = 0;

-- This helper computes the value to be assigned to the ID field of a generated Task. It increments the value stored by the "curId" helper and returns its new value as a String.
-- CONTEXT: thisModule
-- RETURN: String
helper def: getId() :
  String =
  thisModule.refSetValue('curId', thisModule.curId + 1).curId.toString();

-- This helper computes the set of StateVertex that will be associated with the predecessors of a Task.
-- The computed set contains all the ActionState and Pseudostate of "initial" type (ie. the initial state) that point to the current StateVertex (the context one) either directly or through "fork" and "join" Pseudostate.
-- WARNING: this helper is not able to deal with loops. However, there shall be no loops in a diagram dedicated to the representation of a project.
-- CONTEXT: UML!StateVertex
-- RETURN: Set(UML!StateVertex)
helper context UML!StateVertex def: getPredecessors() :
  Set(UML!StateVertex) =
  let trans : Set(UML!StateVertex) = self.incoming in
  if trans.isEmpty() then
    Set()
  else
    trans->collect(t | t.source)
      ->iterate(e; ret : Set(UML!StateVertex) = Sequence{} |
        if e.isKindOf(UML!ActionState) then
          ret->including(e)
        else
          if e.isKindOf(UML!Pseudostate) then
            if e.kind = #pk_initial then
              ret->including(e)
            else
              ret->including(e.getPredecessors())
          endif
        else
          ret
      )
  endif;
```
-- RULES

-- Rule 'Main'
-- This rule generates the Project element. Contained tasks are those
-- associated with:
--  * UML Final State
--  * UML Action State
--  * UML Pseudostate of "initial" kind.
rule Main {
  from
  s : UML!StateMachine
  to
  pro : MSProject!MSProject (tasks <- UML!StateVertex.allInstances())
}

-- Rule 'Pseudostate'
-- This rule generates a Task for the Pseudostate of "initial" type (that is,
-- the diagram initial state).
-- The generated initial Task has no predecessors (since it corresponds to the
-- initial state of the UML activity diagram).
rule Pseudostate {
  from
  s : UML!Pseudostate (s.kind = #pk_initial)
  to
  t : MSProject!Task (UID <- thisModule.getId(),
    name <- s.name,
    predecessors <- Set{})
}

-- Rule 'StateVertex'
-- This rule generates Tasks for both ActionStates and FinalStates.
-- The set of predecessors of a Task is computed by the getPredecessors helper.
-- It corresponds to the set of ActionState/"initial" Pseudostate pointing to
-- the current state directly, or through one or several "fork" and "join"
-- Pseudostates.
rule StateVertex {
  from
  s : UML!StateVertex (s.oclIsKindOf(UML!FinalState)
    or s.oclIsKindOf(UML!ActionState))
  to
  t : MSProject!Task (UID <- thisModule.getId(),
    name <- s.name,
    predecessors <- s.getPredecessors())
}
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;
}
References

