Model Migration Case for TTC 2010

Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack

Department of Computer Science, University of York, UK.
{louis,dkolovos,paige,fiona}@cs.york.ac.uk

Abstract. Using an example from the Unified Modelling Language, we invite submissions to explore the ways in which model transformation languages can be used to update models in response to metamodel changes.

1 Introduction

MDE introduces additional challenges for controlling and managing software evolution [8]. For example, when a metamodel evolves, instance models might no longer conform to the structures and rules defined by the metamodel. When an instance model does not conform to its metamodel, it cannot be manipulated with metamodel-specific editors, cannot be managed with model management operations and, in some cases, cannot be loaded with modelling tools. Model migration is a development activity in which instance models are updated to re-establish conformance in response to metamodel evolution.

Several approaches to automating model migration have been proposed. Sprinkle et al. [12] were the first to describe co-evolution as distinct from the more general activity of model-to-model transformation. Presently, various languages are used for specifying model migration, such as the Atlas Transformation Language (ATL) [6] in work by Cicchetti et al. [1], and the general-purpose programming language Groovy [7] in COPE [5]. There is little work, however, that compares the languages used for specifying model migration.

To explore and compare ways in which model migration can be specified, we propose a case from the evolution of the UML. The way in which activity diagrams are modelled in the UML has changed significantly between versions 1.4 and 2.1 of the specification. In the former, activities were defined as a special case of state machines, while in the latter they are defined atop a more general semantic base1 [11].

2 Activity Diagrams in UML

Activity diagrams are used for modelling lower-level behaviours, emphasising sequencing and co-ordination conditions. They are used to model business processes and logic [10]. Figure 1 shows an activity diagram for filling orders. The

1 A variant of generalised coloured Petri nets.
diagrams is partitioned into three *swimlanes*, representing different organisational units. *Activities* are represented with rounded rectangles and *transitions* with directed arrows. *Fork* and *join* nodes are specified using a solid black rectangle. *Decision* nodes are represented with a diamond. Guards on transitions are specified using square brackets. For example, in Figure 1 the transition to the restock activity is guarded by the condition [not in stock]. Text on transitions that is not enclosed in square brackets represents a trigger event. In Figure 1, the transition from the restock activity occurs on receipt of the asynchronous signal called *receive stock*. Finally, the transitions between activities might involve interaction with objects. In Figure 1, the Fill Order activity leads to an interaction with an object called *Filled Object*.

Between versions 1.4 and 2.2 of the UML specification, the metamodel for activity diagrams has changed significantly. The sequel summarises most of the changes. For full details, refer to [9] and [10].

### 3 Evolution of Activity Diagrams

Figures 2 and 3 are simplifications of the activity diagram metamodels from versions 1.4 and 2.2 of the UML specification, respectively. In the interest of clarity, some features and abstract classes have been removed from Figures 2 and 3.

Some differences between Figures 2 and 3 are: activities have been changed such that they comprise nodes and edges, actions replace states in UML 2.2, and the subtypes of control node replace pseudostates. For full details of the changes made between UML 1.4 and 2.2, refer to [9] and [10].

The model to be migrated is shown in Figure 1, and is based on [9, pg3-165]. A migrating transformation should migrate the activity diagram shown in Figure 1 from UML 1.4 to UML 2.2. The UML 1.4 model, the migrated, UML 2.2 model, and the UML 1.4 and 2.2 metamodels are available from [2].

Submissions will be evaluated based on the following three criteria:

- **Correctness**: Does the transformation produce a model equivalent to the migrated UML 2.2. model included in the case resources?
- **Conciseness**: How much code is required to specify the transformation? (In [12] et al. propose that the amount of effort required to codify migration should be directly proportional to the number of changes between original and evolved metamodel).
- **Clarity**: How easy is it to read and understand the transformation? (For example, is a well-known or standardised language?)

Submissions might also consider the three extensions discussed below, in Sections 3.1, 3.2 and 3.3.

---

2 http://www.cs.york.ac.uk/~louis/ttc/
Fig. 1. Activity model to be migrated.
Fig. 2. UML 1.4 Activity Graphs (based on [9]).

Fig. 3. UML 2.2 Activity Diagrams (based on [10]).
3.1 Alternative Object Flow State Migration Semantics

Following the submission of this case, much discussion on the TTC forums\(^3\) revealed an ambiguity in the UML 2.2 specification. Consequently, the migration semantics for the ObjectFlowState UML 1.4 concept are not clear from the UML 2.2 specification.

In the core task described above, instances of ObjectFlowState should be migrated to instances of ObjectNode. Any instances of Transition that have an ObjectFlowState as their source or target should be migrated to instances of ObjectFlow. Listing 1.1 shows an example application of this migration semantics. The top line of Listing 1.1 shows instances of UML 1.4 metaclasses, include an instance of ObjectFlowState. The bottom line of Listing 1.1 shows the equivalent UML 2.2 instances according to this migration semantics. Note that the Transitions, t1 and t2, is migrated to an instance of ObjectFlow. Likewise, the instance of ObjectFlowState, s2, is migrated to an instance of ObjectFlow.

```
s1:State <- t1:Transition -> s2:ObjectFlowState <- t2:Transition -> s3:State
```

Listing 1.1. Migrating Actions

This extension consider an alternative migration semantics for ObjectFlowState. For this extension, instances of ObjectFlowState (and any connected Transitions) should be migrated to instances ObjectFlow, as shown by the example in Listing 1.2. Note that the UML 2.2 ObjectFlow, f1, replaces t1, t2 and s2.

```
s1:State <- t1:Transition -> s2:ObjectFlowState <- t2:Transition -> s3:State
s1:ActivityNode <- f1:ObjectFlow -> s3:ActivityNode
```

Listing 1.2. Migrating Actions

3.2 Concrete Syntax

The UML specifications provide no formally defined metamodel for the concrete syntax of UML diagrams. However, some UML tools store diagrammatic information in a structured manner using XML or a modelling tool. For example, the Eclipse UML 2 tools [4] store diagrams as GMF [3] diagram models. As such, submissions might explore the feasibility of migrating the concrete syntax of the activity diagram shown in Figure 1 to the concrete syntax in their chosen UML 2 tool. To facilitate this, the case resources include an ArgoUML [2] project containing the activity diagram shown in Figure 1.

3.3 XMI

Because XMI has evolved at the same time as UML, UML 1.4 tools most likely produce XMI of a different version to UML 2.2 tools. For instance, ArgoUML produces XMI 1.2 for UML 1.4 models, while the Eclipse UML2 tools produce XMI 2.1 for UML 2.2. As an extension to the case outline above, submissions might consider how to migrate a UML 1.4 model represented in XMI 1.x to a UML 2.1 model represented in XMI 2.x. To facilitate this, the UML 1.4 model shown in Figure 1 is available in XMI 1.2 as part of the case resources.

Acknowledgement. The work in this paper was supported by the European Commission via the MADES project, co-funded by the European Commission under the “Information Society Technologies” Seventh Framework Programme (2009-2012).

References