Towards an execution engine for the UML

François Pennaneac'h and Gerson Sunyé

IRISA/CNRS, Campus de Beaulieu, F-35042 Rennes Cedex, FRANCE
email: pennanea,sunye@irisa.fr

1 Introduction

The current UML specification allows specifying both the structural and dynamic part of a model. The modeling of dynamic behaviour is projected onto many different kinds of diagrams: state charts, sequence diagrams, collaboration diagrams... they all cover an aspect of the same computing problem, providing designers with a useful separation of concerns.
Unfortunately, this scattering of information across different views leads to some inconsistencies when merging the views into a single, coherent abstraction. The UML metamodel syntax and the OCL well-formedness rules can only enforce the static structure of models; the current UML specification does not provide enough information for a model to be executable. This is particularly visible through the many "Expressions" used in the metamodel which basically are strings, with no more semantics.
In order to fill this gap, the OMG issued a Request for Proposal [1] for a *precise, software-independant action specification*. Submissions were due the 15^{th} of August [2]. Its primary goal is to allow designers to give a complete operational description of the behaviour of operations by decomposing them into a partially-ordered set of well semantically defined actions.
Far better than yet another view for specifying behavioural parts of a system (with the new integration problems it could raise), the Action Semantics (AS) could serve as a unified foundation for the whole UML, a position we advocate in this paper.

2 About the AS specification

The AS is helpful to software developers who want to achieve early simulation and testing for their models, and thus need executability (the AS allows for precisely specifying the body of actions, or guards on transitions). A closer look at the AS shows it can have a much deeper impact, and serves the whole UML community, as it provides a formalism for executing models. The UML being itself a UML model, it paves the way for building execution engine specification for the UML, and the derived implementation.
The AS specification is divided into three complementary parts:

- A meta-model: it extends the current UML meta-model. New classes are introduced, most of them being subclasses of the existing abstract meta-class

Action. They allow designers to describe behaviour in an operational way, by specifying the partial-order of actions to be executed. Primitive actions such as the creation,deletion or modification of objects, creation or deletion of links between objects or the sending of messages are provided. Actions supersede the uninterpreted Expressions previously used for the description of bodies in methods, or guards on transitions.
- An execution model: not part of the meta-model, but defines the UML model of a mechanism for describing the evolutions of the modeled system. It introduces the notion of snapshot. Every change to a system results in a new snapshot being created. The history of an object is represented by a succession of snapshots. It paves the way to an UML execution engine, as the role of such an engine is to compute the next snapshot in an object history from the past and current snapshots for each executed action.
- A semantics of actions: actions are described in a denotational way in terms of (pre-,post-)condition pairs expressed in OCL. The post-condition specifies the modifications of the execution model instances after the action has been executed, ie. what the changes between the next snapshot and the current snapshot are. This semantics specifies the *what*, but it does not formally describe the *how* (see our work below).

3 Work in progress

We propose here an introduction to our work, and describe the two main research directions we are currently exploring:

- the first one aims at providing the UML with a complete semantics, in an operational way, by specifying the execution mechanism of an UML specification in terms of an AS program. For instance, the sending of a message to an object is described using AS actions (see below). An application is the realization of simulators for UML specifications,
- the second one aims at unifying the many UML views by providing a common underlying formalism. We choose the AS for this purpose, and propose to map all the other UML aspects into this AS (state chart, sequence diagrams...). Translating all the views into a common formalism allows for their comparison.

Although the pre-/post-condition approach used in the current AS proposal gives a precise semantics of the AS, we think an operational approach is also highly desired:

- It provides an alternative to the complicated OCL assertions used in the denotational approach. Thus, the comparison of both approaches may help track the errors, making the UML specification and documentation more bulletproof,
- It paves the way for UML tool vendors, helping them in providing compliant UML simulators,

– It may also help simplify the current AS proposal, as we think some of the most complicated AS actions may be specified in terms of more basic and primitive actions.

4 Shorts examples

We illustrate our approach with two toy-examples, extracted from our execution engine for the UML.

4.1 Sending a message

We try to formulate what happens when sending a message, using the following definitions extracted from respectively the UML1.3 specification [3] and the AS proposal [2].

– UML1.3 (SendAction definition)
"A send action is an action that results in the (asynchronous) sending of a signal. The signal can be directed to a set of receivers via an objectSetExpression, or sent implicitly to an unspecified set of receivers, defined by some external mechanism".
– AS (SendActionExecution specification):
"Send action generates a send packet and sends copies of it to each target object..."

It is clear this can be formally specified using the AS primitive actions (ie. object creation, link creation..., actions on collections of objects).

– the set of receivers is evaluated (it may be specified with the AS, so it fits in our AS evaluation framework),
– n SendPackets are created (this is an AS CreateObjectAction) and initialized (ie. the value of their *target* attribute is initialized with the n^{th} receiver. This maps to a WriteAttributeAction into the AS),

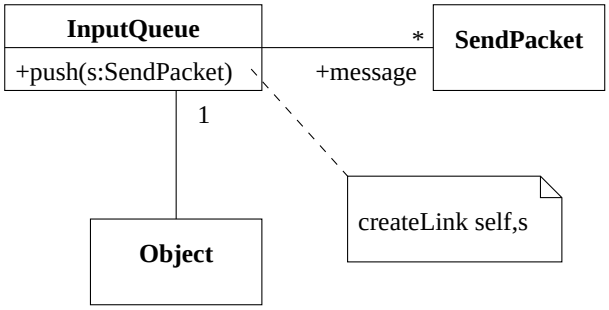

Fig. 1. An active object and its queue

– every SendPacket is "sent", ie. it is added into the input queue of the receiving active object. The queue being itself described with a UML/AS model (it is part of our execution engine, see Fig.1), this maps to an AS CreateLinkAction.

4.2 Executing sequence diagrams

In this section, we describe an application of an AS execution engine for the UML we are currently investigating. We give some tips about how the AS could be used for the definition of a mechanism for sequence diagram execution. For writing AS definitions, we use a straightforward surface-language where expressions use the OCL, and we add some familiar constructs(ie. assignment, loop). Note this is only for illustrative purposes, and any other surface language could be used instead, provided it maps to the AS metamodel (this is the case for our language, and all usual OO languages).
In the UML metamodel, a sequence diagram is represented by an Interaction, and specifies the communication between instances performing a specific task. It owns a number of partially-ordered Messages, see Fig. 2.
A message may be sent only if all its predecessor messages has been sent. We formalize this behaviour below in the `execute()` method (Fig.3).

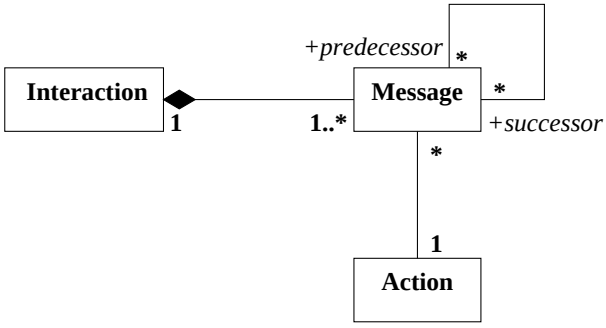

Fig. 2. Interactions in UML (partial)

5 Conclusion

In this paper, we investigate a way to specify and build execution engines for the UML, using the Action Semantics, an extension to the UML for describing the behaviour of actions. We advocate this would give a steady foundation for defining an UML execution engine in an operational way. This could help formalizing the UML, and could also serve as a base for the development of UML

```
Interaction::execute()
-- register already sent messages
      sent_messages := new Set(Message)
-- select a message with no predecessor
      message_queue := self.message->select(m:Message |
                                          m.predecessor.isEmpty)
while
-- is there a message with all predecessors already sent ?
      not message_queue.isEmpty
do
      -- choose one of them, non deterministically
      msg := message_queue.choose
      -- tell the execution engine to execute the action
      msg.action.execute()
      sent_messages := sent_messages->includes(msg)
      -- recompute the set of messages with predecessors sent
      message_queue := self.message->select(m : Message |
                  sent_message->includesAll(m.predecessor))
end -- while
```

Fig. 3. Executing sequence diagrams

tools. This will greatly improve their interoperability and their features, one of the most desired being early testing and simulation, a key to software quality. A prototype of an execution engine for validating the soundness of our approach is currently under development, as an extension to the UMLAUT tool [4].

References

1. Action Semantics for the UML RFP http://cgi.omg.org/cgi-bin/doc?ad/98-11-01
2. Updated Joint Initial Submission against the Action Semantics for UML RFP. http://cgi.omg.org/cgi-bin/doc?ad/00-08-03
3. OMG UML v. 1.3 specification http://cgi.omg.org/cgi-bin/doc?ad/99-06-08
4. UML All pUrposes Transformer http://www.irisa.fr/pampa/UMLAUT