Defining Semantics for Rigorous Development in UML

K. Lano, J. Bicarregui*, A. Evans†

August 30, 1998

Abstract

The Unified Modelling Language (UML) is becoming the de facto industry standard notation for object-oriented analysis and design. In this paper we outline a semantic framework for UML which would support formal analysis and verification as part of a rigorous development process for critical systems.

1 Introduction

The UML [8] combines and extends elements of previous OO notations such as OMT, Booch and Objectory. In contrast to these methods, its notations are precisely defined using the Object Constraint Language (OCL) and a meta-model to express the allowed forms of diagrams and their properties. In previous papers we have shown how the semantic meaning of some UML diagrams can also be precisely defined [4, 3]. This semantics supports the use of *transformational development*: the refinement of abstract models towards concrete models, using design steps which are known to be correct with respect to the semantics (all properties of the abstract model are preserved in the refined model).

For highly critical applications (systems where the consequence of incorrect functioning may include loss of life or severe financial loss), it is important that the development process used can help detect and eliminate errors. The process should in particular support the *verification* of refined models against abstract models by comparing their semantics.

Our proposed process can be summarised as follows:

1. Requirements – modelled using Yourdon context diagrams and UML use case diagrams (without dependencies between use cases [9]).
2. Essential Specification – described using UML class diagrams, operation schemas (from Fusion and Octopus) and statecharts.
3. Design – modelled using UML class diagrams, statecharts, sequence diagrams and collaboration diagrams.

The verifiable relationships between these models are:

1. Each input event/message on the system context diagram should have a system response described by an operation schema.
2. The effect described by the operation schema for an event **e** must be established by the completed response sequence to **e** described in design level statecharts, that is, by the transitions specified for **e** and (recursively) their generated events and transitions.

   A refinement relationship should be defined between the abstract state used in the operation schemas, and the state used in the statecharts.

*Dept. of Computing, Imperial College, 180 Queens Gate, London SW7 2BZ
†Dept. of Computing, University of York

3. Design level class diagrams should enforce all the properties asserted in the specification level class diagrams.
4. Sequence diagrams should be consistent with collaboration diagrams: the structure of object intercalling should be the same.
5. Collaboration diagrams should be consistent with statecharts: messages sent by an object in response to a message **m** should correspond to events generated from transitions for **m** in the statechart of the object.

2 Semantic Model

The semantic model of UML used here is based on the set-theoretic Z-based model of Syntropy [2]. A mathematical semantic representation of UML models can be given in terms of *theories* in a suitable logic, as in the semantics presented for Syntropy in [1] and VDM^{++} in [7]. In order to reason about real-time specifications the more general version, Real-time Action Logic (RAL) [7] can be used.

A RAL theory has the form:

theory *Name*

types *local type symbols*

attributes *time-varying data, representing instance or class variables*

actions *actions which may affect the data, such as operations, statechart transitions and methods*

axioms *logical properties and constraints between the theory elements.*

Theories can be used to represent classes, instances, associations and general submodels of a UML model. These models are therefore taken as *specifications*: they describe the features and properties which should be supported by any implementation that satisfies the model. In terms of the semantics, theory **S** satisfies theory **T** if there is an interpretation σ of the symbols of **T** into those of **S** under which every property of **T** holds:

$$\mathbf{S} \vdash \sigma(\varphi)$$

for every theorem φ of **T**.

In addition to standard mathematical notation such as $\mathbb{F}$ for "set of finite sets of", etc, RAL theories can use the following notations:

1. For each classifier or state **X** there is an attribute $\overline{\mathbf{X}} : \mathbb{F}(\mathbf{X})$ denoting the set of existing instances of **X**[1].
2. If α is an action symbol, and **P** a predicate, then $[\alpha]\mathbf{P}$ is a predicate which means "every execution of α establishes **P** on termination", that is, **P** is a *postcondition* of α.
3. For every action α there are functions $\uparrow(\alpha, \mathbf{i})$, $\downarrow(\alpha, \mathbf{i})$, $\leftarrow(\alpha, \mathbf{i})$ and $\rightarrow(\alpha, \mathbf{i})$ of $\mathbf{i} : \mathbb{N}_1$ which denote the activation, termination, request send and request arrival times, respectively, of the **i**-th invocation of α. These times are ordered as:

   $$\leftarrow(\alpha, \mathbf{i}) \ \leq \ \rightarrow(\alpha, \mathbf{i}) \ \leq \ \uparrow(\alpha, \mathbf{i}) \ \leq \ \downarrow(\alpha, \mathbf{i})$$

   Also

   $$\mathbf{i} \leq \mathbf{j} \ \Rightarrow \ \leftarrow(\alpha, \mathbf{i}) \ \leq \ \leftarrow(\alpha, \mathbf{j})$$

Either Z or OCL notation could be used for axioms in theories, representing the semantics or constraints of UML models. In [6] we define a translation from OCL into Z.

Temporal logic makes representation and reasoning about dynamic models (state machines, interaction diagrams, etc) more concise than using a formalism such as pure Z. However it would be possible to work just in Z, by using sequences of states to represent the allowed behaviours of objects over time.

[1] Alternative notation for $\overline{\mathbf{X}}$ is $\mathbf{ext}(\mathbf{X})$, the *extension* of **X** [10].

3 Example

We will illustrate the use of the semantics by showing some of the development and verification steps for a simple traffic light system.

The system to be constructed is a controller for two pairs of traffic lights at a crossroads. Traffic lights 1 and 3 must always show the same indication, as must lights 2 and 4. Traffic lights cycle from Green to Amber to Red on the 'go red' cycle, and Red, Red and Amber, Green, on the 'go green' cycle. There is a delay of 3 seconds in the Red and Amber state, and 5 seconds in the Amber state. The safety requirement is that at least one pair of traffic lights must be red at any given time.

The system responds to a signal 'change_direction'. The response should be to set the currently red signals to green, and the currently green signals to red.

We model the system simply as a collection of two traffic light pairs containing distinct traffic lights **tl**1 and **tl**3, and **tl**2 and **tl**4. The abstract object model is given in Figure 1. **State** is the enumerated type {**green**, **amber**, **red**, **red_amber**} for the illumination state of an individual traffic light.

The invariant that the light pairs always illuminate the same lamps is expressed as:

tl1.**tlstate** = **tl**3.**tlstate** **and**
tl2.**tlstate** = **tl**4.**tlstate**

or more generally, as an invariant of **TrafficLightPair**:

traffic_light[1].**tlstate** = **traffic_light**[2].**tlstate**

where we use an extended OCL notation **composite**[**i**] to refer to the **i**-th element in a composite list of objects.

The safety constraint is formalised as:

traffic_light_pair[1].**traffic_light**[1].**tlstate** = **red** **or**
traffic_light_pair[2].**traffic_light**[1].**tlstate** = **red**

We need to show that these invariants are maintained by operation schemas and their implementations.

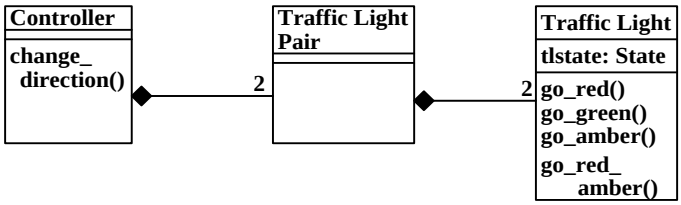

Figure 1: Abstract Object Model of Traffic Light Controller

The operation schemas express the required effects of the operations listed in the use cases of the system, without any decomposition into methods of individual objects. As we discuss in [5], this style of essential model description is often clearer than the artificial localisation of such specifications used in Syntropy essential models [2].

operation *change_direction*

reads *traffic_light_pair*, *traffic_light*

writes *tlstate*

precondition *true*

postcondition

```
if tl1.tlstate@pre = green
then
  tl1.tlstate = red and tl2.tlstate = green and
  tl3.tlstate = red and tl4.tlstate = green
else
  if tl1.tlstate@pre = red
  then
    tl1.tlstate = green and tl2.tlstate = red and
    tl3.tlstate = green and tl4.tlstate = red
```

where

$$
\begin{aligned}
\mathbf{tl1} &= \mathbf{traffic\_light\_pair}[1].\mathbf{traffic\_light}[1] \\
\mathbf{tl2} &= \mathbf{traffic\_light\_pair}[2].\mathbf{traffic\_light}[1] \\
\mathbf{tl3} &= \mathbf{traffic\_light\_pair}[1].\mathbf{traffic\_light}[2] \\
\mathbf{tl4} &= \mathbf{traffic\_light\_pair}[2].\mathbf{traffic\_light}[2]
\end{aligned}
$$

The notation **e@pre** denotes the value of **e** at activation of the operation. If an attribute **e** does not occur in the **writes** list, then **e** can be used instead of **e@pre** since these two values are the same.

In the design step, details of how this reaction is computed are given, via state machines (Figures 2 and 3).

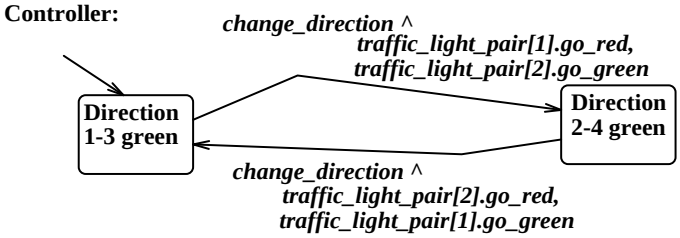

Figure 2: Statechart of **Controller**

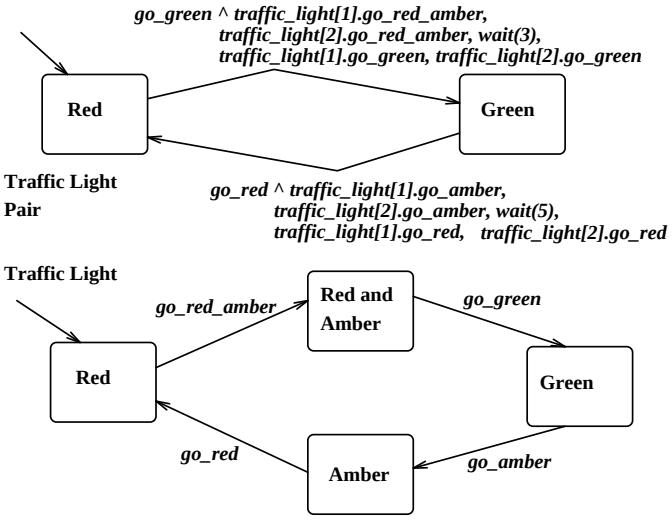

Figure 3: Statechart of **TrafficLightPair** and **TrafficLight**

Verification that the reaction achieves the effect specified in the operation schema is direct. For example, in the case that direction 1-3 is initially green, the transition for **change_direction** in the **Controller**

state machine terminates once the transition for **go_red** on the state machine for **traffic_light_pair**[1] and then the transition for **go_green** on the state machine for **traffic_light_pair**[2] terminate. The first of these transitions results in a state where

$$\begin{array}{l} \textbf{traffic\_light\_pair}[1].\textbf{traffic\_light}[1].\textbf{tlstate} = \textbf{red} \\ \textbf{traffic\_light\_pair}[1].\textbf{traffic\_light}[2].\textbf{tlstate} = \textbf{red} \\ \textbf{traffic\_light\_pair}[2].\textbf{traffic\_light}[1].\textbf{tlstate} = \textbf{red} \\ \textbf{traffic\_light\_pair}[2].\textbf{traffic\_light}[2].\textbf{tlstate} = \textbf{red} \end{array}$$

and the second in the specified state

$$\begin{array}{l} \textbf{traffic\_light\_pair}[1].\textbf{traffic\_light}[1].\textbf{tlstate} = \textbf{red} \\ \textbf{traffic\_light\_pair}[1].\textbf{traffic\_light}[2].\textbf{tlstate} = \textbf{red} \\ \textbf{traffic\_light\_pair}[2].\textbf{traffic\_light}[1].\textbf{tlstate} = \textbf{green} \\ \textbf{traffic\_light\_pair}[2].\textbf{traffic\_light}[2].\textbf{tlstate} = \textbf{green} \end{array}$$

as required. It can also be checked that each transition maintains the safety invariant.

Conclusion

Representation of the semantics of static structure diagrams and much of the semantics of state machines (excluding history entry to states) can be directly expressed in RAL theories and used as the basis for verification during development. Sequence diagrams and collaboration diagrams can also be semantically expressed using the $\uparrow(\alpha, \mathbf{i})$ terms, etc., but do not have such elegant or usable semantics.

References

[1] J C Bicarregui, K C Lano, T S E Maibaum, *Objects, Associations and Subsystems: a hierarchical approach to encapsulation*, ECOOP 97, LNCS, 1997.

[2] Cook S., Daniels J., *Designing Object Systems: Object-oriented Modelling with Syntropy*, Prentice Hall, 1994.

[3] K. Lano, J. Bicarregui, *Semantics and Transformations for UML Models*, UML 98 Conference, Mulhouse, France, 1998.

[4] K. Lano, J. Bicarregui, *Formalising the UML in Structured Temporal Theories*, ECOOP 98 Workshop on Behavioural Semantics, Technical Report TUM-I9813, Technische Universitat Muchen, 1998.

[5] K. Lano, R. France, J-M. Bruel, *A Semantic Comparison of Fusion and Syntropy*, to appear in *Object-oriented Systems*, 1998.

[6] Lano K., Bicarregui J., *UML Refinement and Abstraction Transformations*, ROOM 2 Workshop, Bradford University, 1998.

[7] K Lano, *Logical Specification of Reactive and Real-Time Systems*, to appear in *Journal of Logic and Computation*, 1998.

[8] Rational Software et al, *UML Notation Guide*, Version 1.1, http://www.rational.com/uml, 1997.

[9] A. Simons, I. Graham, *37 Things That Don't Work in Object-Oriented Modelling with UML*, ECOOP 98 Workshop on Behavioural Semantics, Technical Report TUM-I9813, Technische Universitat Muchen, 1998.

[10] R Wieringa, W. de Jonge, P. Spruit, *Roles and Dynamic Subclasses: A Model Logic Approach*, IS-CORE report, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1993.