

Slicing the Three-layer Architecture: A Semantic Foundation for Behavioural Specification

Michelle L. Crane
Queen's University, Kingston, Ontario, Canada
crane@cs.queensu.ca

Abstract

We outline a research proposal whose overall goal is to contribute to the definition of a formal semantics for UML, and indeed visual behavioural modelling languages in general. Specifically, we aim to validate the three-layer semantic architecture, used as a way of explaining the behavioural semantics of UML. The validation includes a definition of the semantics of UML actions and activities, as well as a prototype interpreter.

1 Introduction

Languages are used for the communication of ideas. Communication becomes compromised when the various participants do not fully understand, or agree upon, the semantics of the language being used. Visual modelling languages, especially those using common elements, such as text boxes and connecting arrows, can be very susceptible to this kind of miscommunication. Consider, for instance, that a state machine diagram could have three different interpretations, depending on the formalism used to interpret it [5].

It is important to formally, i.e., precisely and unambiguously, define the semantics of visual modelling languages. Developers must understand the semantics in order to communicate with users and other developers. Tool vendors must understand the semantics, not only so that developers can use their tools, but so that tools by different vendors can be linked together into tool chains. This concept of tool interoperability is becoming more important in Model Driven Development (MDD), in addition to the automatic manipulation, transformation, and analysis of models, all of which require that the semantics of the language under examination is well-understood.

A language is defined by its notation (syntax), together with the meaning of syntactic elements (semantics). This meaning is expressed by mapping the syntax to a semantic domain [6], usually one that is well-understood. Although visual modelling languages are becoming increasingly widespread, formally defining their semantics has proven difficult. The Unified Modeling Language (UML) is a “general-purpose visual modeling language” [11] that can be used in the analysis and design of software systems. Although well-documented (over 1000 pages of specifications), UML does not yet have a formal semantics. The abstract syntax is carefully laid out in the specification, but the meaning of the syntactic elements is discussed in prose, with a smattering of Object Constraint Language (OCL) constraints. The run-time semantics of UML is defined as a mapping of modelling concepts onto a corresponding execution. This semantics is not formally defined in the UML specification. Instead, the specification outlines a three-layer semantics architecture, which “identifies key semantic areas...and how they relate to each other” [10].

This three-layer architecture is shown in Figure 1. Each layer depends on those below it, but not vice versa. The bottom layer represents the structural foundations of UML, including concepts such as values, objects, links, messages, etc. The middle layer is the behavioural base, which contains mechanisms for individual object behaviour, as well as behaviour between objects. More importantly, this layer also contains the description of a set of UML actions. The top layer represents different behavioural specifications in UML, all of which rely on the behavioural base. For instance, activities, state machines, and interactions all make use of the actions in order to express behaviour. These actions are explained in terms of constructs in the structural foundations.

The key to this architecture lies in the middle layer,

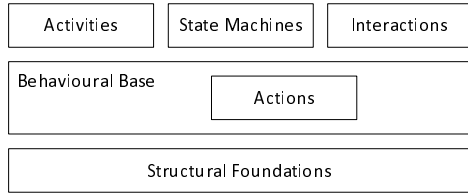


Figure 1: The UML semantics layers: the Semantics Foundation consists of the bottom two layers

i.e., actions. A fundamental premise of UML behavioural semantics is the assumption that all behaviour is ultimately caused by actions. Actions are “fundamental units of behaviour” [10]. As an analogy, actions are comparable to executable statements in a traditional programming language. The advantage of this approach to defining the run-time semantics is the fact that once UML actions are clearly mapped to the structural foundation, it should be relatively easy to define the semantics of different behavioural formalisms.

The three-layer architecture, recently added to UML, has been neither elaborated nor validated. It appears to provide an appealing skeletal framework; however, essential pieces are still missing before it can be leveraged for the definition of a formal semantics for behavioral specification in UML. There is no formal specification of the structural foundations. There is no clear mapping of UML actions to that foundation. Finally, there is no clear mapping from behavioural formalisms to individual actions.

2 Proposed Research

Hypothesis: The layered semantic architecture is a valid approach for formally defining and implementing behavioural semantics in UML.

We propose to demonstrate the validity of this hypothesis by accomplishing the following four high-level tasks.

2.1 Semantics of Actions

The System Model [2, 3, 4] was created as the “basis for a semantic model for UML 2.0”; its goal is to represent the structural foundations of UML (lowest layer of the three-layer semantic architecture from Figure 1). The System Model is described in a precise fashion, using mathematics, specifically, sets, relations, and functions. By choosing pure mathemat-

ics to describe UML, the authors have aimed to avoid limitations or biases inherent in other formal specification formalisms.

Using the System Model as our semantic domain, we will formally define the semantics for the behaviour of individual UML actions. The semantics will be axiomatic; pre- and post-conditions will be defined on the structural foundations described by the System Model.

There are a total of 45 actions in UML, 36 of which are concrete. Some actions will have absolutely no effect on the System Model’s data store, e.g., the ReadStructuralFeature action (used to read the value of an object’s attribute). Other actions will either add to the System Model’s data store, e.g., CreateObjectAction (used to create a new object), or modify it in some way, e.g., the AddStructuralFeatureValueAction (used to assign a value to an object’s attribute).

2.2 Semantics of Activities

Although the mapping of actions to the System Model will be performed on an individual basis, it is difficult to examine actions in isolation, especially when the execution of one action requires prior execution of another. Activities are used to compose actions. For instance, the activity diagram in Figure 2 contains the actions necessary to create a new *Car* object, and assign a value to its *year* attribute. The AddStructuralFeatureValueAction cannot be executed until an object has been created.

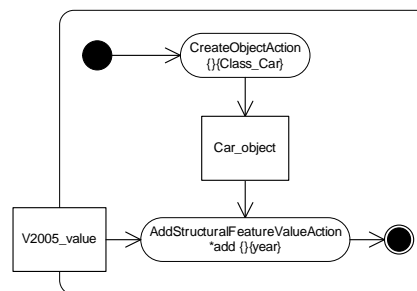


Figure 2: Activity diagram with the actions necessary to create a new *Car* object, and assign a value to its *year* attribute.

We will define a formal semantics for the behaviour of a subset of UML activities. This subset will include the sequential composition of actions, as well as the fork/join and decision/merge control nodes. Both data and control flow will be accommodated. The semantics will be operational; the execution of

activities will be described in terms of how the System Model changes as a result of the execution of individual actions and control nodes.

2.3 Implementation

A large part of our research involves the implementation of the System Model, individual UML actions, and the subset of UML activities. This implementation is part of a larger prototype activity/action “virtual machine/interpreter”, based on the formal semantics mappings.

The combination of a both a formal mapping, and a practical implementation, provides a certain synergy. By contemplating a formal mapping, we learn how a particular action or construction *should* behave, in isolation. By actually constructing an implementation, we learn how the behaviours of individual elements affect each other. This kind of experimentation helps indicate which parts of the System Model might need refining. In addition, it is an excellent way of pinpointing the semantic variation points inherent in UML itself.

2.4 Evaluation

The theoretical portions of this work, e.g., the definition of the semantics of actions and activities, will be evaluated partially by our implementation. By implementing the System Model, actions, and activities, we will be able to find errors, ambiguities, and incompleteness in our theoretical mappings. In addition, we will perform close comparison with the UML specification and other existing literature. Questions arising from this comparison between our interpretation and the specification will be forwarded to subject matter experts.

The implementation itself will be evaluated in the following manner. A small set of test suites will be used that in some sense “covers” the activity diagrams that our interpreter targets. For example, there will be at least one test case for each UML activity and each control node considered by our implementation. The test cases, i.e., individual executions of actions, will be evaluated by using strategically placed assertions. For instance, the post-conditions of an action will be implemented as assertions and checked immediately after the termination of the action’s execution. In addition to the rather binary nature of test cases, we will make use of graphical displays of System Model states created during each

execution. These visualizations will aid in our understanding of the effect of an actions execution and facilitate debugging. They are also very useful in explaining the behaviour of UML actions to interested parties. The visualizations, as well as detailed textual descriptions of state changes, will be thoroughly inspected in order to ensure that the implementation of each action and activity node behaves as expected. Finally, a “reference appendix” will be created, which will show the central theoretical concepts of the System Model matched with their implementation counterparts. This appendix can be used to find mismatches or omissions as well is to explain exactly how the System Model has been implemented.

3 Related Work

One of the basic premises of UML semantics is that all behaviour is “ultimately caused by actions” [10]. Actions were originally introduced into UML 1.5, and in UML 2.0 activities were brought more in line with these actions to produce a more procedural model [11]. UML 2 includes 45 primitive actions to model the manipulation of objects and links, model communication, and model computation. Because UML actions are relatively new, there is little current research on formally defining their semantics. One of the best sources of information about UML actions is a series of articles by Conrad Bock (see [1]), which explain (in natural language) how actions should execute. Research is currently being performed on the concept of executable UML (xUML) [9], focusing on determining a subset of UML, including actions, that could be used to create executable models.

In UML, an activity “represents the execution of a computation” [11]. Defining the semantics of UML activities is a vibrant research area; although the fact that activities substantially changed with the introduction of UML 2 narrows the field somewhat. However, there has been research on defining the semantics of UML 2 activities based on several formalisms, such as Abstract State Machines [12], Petri nets [13], dynamic meta modelling [7], etc.

Our work can also be compared to other research that executes activity diagrams. ActiveChartSIDE [12] is a particularly well-developed project (using Microsoft Visio, Visual Basic, and C#), with an interpreter that allows the simulation and debugging of activity diagrams. Another project is a plug-in for IBM’s Rational Software Architect, which permits the animation and debugging of activity diagrams [8].

Neither of these two approaches make use of the UML actions as fundamental units of behaviour. As of yet, there are no published accounts on the formal semantics of individual UML actions.

4 Contributions

The proposed research advances the field of formal semantics of UML in the following ways:

1. Modelling and implementing the System Model will allow us to find inconsistencies, ambiguities and gaps in both the System Model and UML specification, as well as determine if the System Model is under- or over-specified.
2. We will define a formal semantics for UML actions.
3. We will define a formal semantics for basic UML activities, including those containing sequential composition, fork/join, and decision/merge nodes.
4. An interpreter for actions and activities will be created. This interpreter will implement the formal semantics of actions and activities. It will take as input text files representing the actions and activities.
5. We will examine the thesis that the three-layer architecture is a suitable framework for defining the behavioural semantics of UML. By creating our prototype implementation, we will demonstrate that the architecture is valid.

5 Conclusion

The overall goal of this research is to contribute to the definition of a formal semantics for UML, specifically UML actions and activities. When complete, our results will be forwarded to the authors of the System Model, as well as the authors of the actions and activities chapters of the UML specification, in order to improve those documents.

References

- [1] C. Bock. UML 2 activity and action models, part 6: Structured activities. *Journal of Object Technology*, 4(4):43–66, 2005.

- [2] M. Broy, M.V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The Structural Data Model. Technical Report TUM-I0612, TUM, Jun 2006.
- [3] M. Broy, M.V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The Control Model. Technical Report TUM-I0710, TUM, Feb 2007.
- [4] M. Broy, M.V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The State Machine Model. Technical Report TUM-I0711, TUM, Feb 2007.
- [5] M.L. Crane and J. Dingel. UML vs. classical vs. Rhapsody statecharts: not all models are created equal. *Software and Systems Modelling*, 2007. Online first DOI: 10.1007/s10270-006-0042-8.
- [6] D. Harel and B. Rumpe. Meaningful modeling: What’s the semantics of “semantics”? *IEEE Computer Magazine*, 37(10):64–72, 2004.
- [7] J.H. Hausmann. *Dynamic Meta Modeling*. PhD thesis, University of Paderborn, Oct 2005.
- [8] A. Kirshin and D. Dotan. UML model simulator. Poster handout at MoDELS 2006, Oct 2006.
- [9] OMG. Semantics of a foundational subset for executable UML models. Initial Submission ad/06-05-02, 2006.
- [10] OMG. Unified Modeling Language: Superstructure version 2.1. Technical Report ptc/06-01-02, Object Management Group, Jan 2006.
- [11] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, second edition, 2005.
- [12] S. Sarstedt, J. Kohlmeyer, A. Raschke, M. Schneiderhan, and D. Gessenharter. ActiveChartsIDE. Poster at the European Conference on Model Driven Architecture (ECMDA 2005), Nov 2005.
- [13] H. Störrle and J. Hausmann. Towards a formal semantics of UML 2.0 activities. In *Proceedings German Software Engineering Conference*, volume P-64 of *LNI*, pages 117–128, 2005.