UML Deficiencies from the perspective of Automatic Performance Model Generation

Submission to

OOPSLA '99 Workshop on Rigorous Modelling and Analysis with the UML: Challenges and Limitations

D.H.Akehurst, A.G.Waters, University of Kent at Canterbury, UK.

1. Introduction

Our experience of UML semantics and the use of UML for system design arises principally from research on a project aimed at the automatic generation of performance models from distributed system designs. The intention of this project (funded by British Telecom) [1] was to use, as a basis, the designs produced as part of the system design lifecycle. Hence, performance prediction could be provided with a minimum of effort and without the need for specific performance modelling expertise.

A major element of the work involved in this project was the analysis of design models, their interpretation and subsequent translation into a set of performance modelling constructs. This required the identification of a generic model for design models – a design meta-model – and subsequently a precise specification of a mapping between each of the design meta-model concepts and the appropriate set of performance modelling concepts.

We chose UML as the means to capture design models for a number of reasons, primarily its use by BT and its acceptance as the main OO design notation. Another potential advantage was the presence of a pre-defined meta-model. However, as is now well known, the UML meta-model lacks an unambiguous and precise definition, and does not address all of the necessary areas of system design required for the specification of distributed systems or the detailed precision required for generation of a performance model.

This submission describes our requirements of the UML for the purpose of performance model generation, the key deficiencies we identified with the use of the UML and the interpretation of its meta-model, and a brief account of some of our solutions.

2. UML Deficiencies with respect to Performance Model Generation Requirements

Need for precise understanding of the design concepts. In order to interpret the design model, it is necessary to have a precise understanding of each individual design concept at the meta-model level, to know exactly what it means with respect to the system functionality, and how it affects the other concepts to which it is related. This understanding enables the specification of a mapping from meta-model concepts to performance model components, appropriately parameterised, that will correctly model the behaviour of the designed system.

The UML is imprecise and ambiguous in the meaning of its concepts in many areas, making it hard to use its meta-model to meet this requirement.

Need for software, hardware and behavioural specifications. The most mature area of the UML is that used for the specification of the software structure. However, in order to generate a performance model, it is equally important to specify the behaviour of the system and the hardware on which it will run. The hardware provides the system resources and the depletion of these is what ultimately limits the performance. The behaviour of the system determines the order and quantity of the resources used.

The hardware and behaviour specification areas of the UML still require a significant amount of work, to enable them to satisfactorily meet this requirement

Need for an implementation model. A system performance model predicts the performance of a particular system. A number of different systems could be implemented from a certain system design model. An implementation model defines a specific system, defining details about the number of instances of particular design components and their subsequent configuration and connectivity.

The details that distinguish one system implementation from another are information such as the number of clients connected to a particular server, the characteristics of those connections and the characteristics of the particular computing platform supporting the server. These detailed definitions are needed to characterise specific systems in order to predict their performance.

The UML facility for implementation model specification is significantly immature and does not satisfactorily meet this requirement.

3. Approaches to resolving UML Deficiencies

The following items address specific deficiencies within the UML that are mainly due to poor definition within the meta-model.

Modelling and notation concepts: The UML meta-model defines a set of modelling concepts, which have a separate description to that of the UML notation used to visualise the concepts. There seem to be a number of places where a concept, that could be considered notational, has been included in the meta-model definition. There is not a clear definition of what these concepts mean or how they are intended to integrate with other meta-model concepts. Examples of such concepts are Associations and States.

It is our opinion that the notational models and meta-models should be clearly distinguished and that there should be a precisely defined mapping from one to the other – thus allowing alternative representation of the same meta-model concepts, as is currently the case with sequence and collaboration diagrams.

We have developed a technique for separately defining design meta-models and language models, with the facility to specify a mapping (in OCL) between the two. This enables a number of different languages to be used to *view* the design model. With the introduction of viewpoints to structure the variety of views that can be generated for a single design model, we have a model-viewpoint-language architecture suitable for system design and conformant with the Reference Model for Open Distributed Processing standard [2].

Ambiguous Behaviour Specification concepts: There are a number of areas and components in the meta-model that address the specification of behaviour: the Common Behaviour Package containing Actions and Signals; the State Machine and Activity Packages containing state based behavioural concepts; the Collaboration Package containing interaction concepts; and the Use Case Package. Each of these groups of behavioural elements effectively addresses the specification of behaviour using a different set of concepts, all loosely linked to the structural specification elements of the meta-model (classes, relationships, etc.) but not consistent or integrated with each other, nor fully and unambiguously integrated with the structural elements.

In our opinion, a single set of behavioural concepts should be identified, which can be mapped to different visualisations of that behaviour – either state based or interaction based. The visualisation should subsequently be exactly that: alternative visualisations of the same meta-model concepts, viewed differently to emphasise different aspects of the behaviour.

To achieve this it is necessary to identify the common behavioural concepts, between the different forms of behaviour visualisation. This in turn requires a clear understanding of two aspects of object behaviour. Firstly the generic behaviour (or semantics) of an object, in a possibly multi-threaded environment, and secondly how the concepts for defining specific object behaviour interact with the (structural) network of objects and with each other.

Specification of Hardware components: For a specific distributed system, the specification of the hardware used within the system is a significant part of the system design. The provision, within the UML, of Deployment diagrams and the meta-model concept of a Node does not adequately support the specification of the hardware technology forming part of a distributed system.

The definition of a Node as a subclass of Classifier should enable the specification of characteristics and a possible type hierarchy for hardware components. The notation descriptions in the literature do not show much use of these aspects – and it is thus not clear how such specifications should be constructed. The apparent lack of discussion of these aspects within the UML community raises the question as to whether the hardware specification concepts are adequate, unnecessary or simply so good that no aspect of their use raises significant interest.

Our solution was to use class diagrams for hardware specification, however we believe that this is not a good long-term solution and improved hardware specification techniques are a requirement of the UML.

Implementation models are immature: The deployment and component diagrams are defined by the UML as implementation diagrams. However, they lack a clear description of how they should be used for the specification of (possibly large) distributed system implementation models.

One primary example is a clear definition of how the subsystem concept is applicable to implementation models. The concept of a subsystem exists within the meta-model as a subtype of both a classifier and a package and is defined to contain both “specification elements and realization elements”. But how instances of such a defined subsystem are to be connected to other implementation components (e.g. Nodes) is not addressed at all.

Our solution to this problem is to introduce the concept of a subsystem containing a number of “access points”. An access point has a set of internal and a set of external connections. The internal connections are defined within the specification of the subsystem contents (i.e. the subsystem ‘class’). When instantiated the subsystem is connected to its environment via the external connections of its access points.

As with the specification of hardware components, it is expected that the understanding of this area of the UML will improve and mature only if it gains increased use within the community. However, unless there is some attempt at a clear definition within the meta-model, this is unlikely to happen.

4. Summary

The primary deficiencies within the UML, both for the purpose of design in general, and to enable automatic generation of performance models in particular, result from a lack of precise definition of how many of the meta-model concepts are to be used.

5. References

[1] Peter Utton, Gino Martin; Further Experiences with Software Performance Modelling; *Proceedings of the First International Workshop on Software and Performance, WOSP 98*; October 1998; pp. 14-15.

[2] ISO/IEC; Open Distributed Processing - Reference Model - Part 1: Overview; *International Standard 10746-1, ITU Recommendation X.901*; July 1995