Types in MDA

Jim Steel

Irisa, Campus de Beaulieu, 35042 Rennes, France

Abstract. The following people participated in the breakout session held on September 8 as part of the 2nd European Workshop on Model-Driven Architecture, and it is their ideas that are represented herein, and the author's hope that their ideas have been adequately expressed: Marcus Alanen, Rasmus Fogh, Val Jones, Girish Maskeri, Jim Steel, Laurence Tratt, Andrew Watson, Ed Willink.

The MDA paradigm is, at its core, an architecture designed around the definition of languages and transformations between them. However, while specifications such as the MOF discuss some aspects of language definition, particularly structural aspects, there is considerably less discussion of the type systems that may exist in these langauges.

In considering this issue, a number of questions become apparent. Firstly, in this MDA world, what are the concepts that we wish to characterise by a type, what sort of structures do we want to use to describe these types, and under what circumstances may something of one type be used in a situation demanding a different type? The most compelling example for answering these questions is perhaps that of sequencing model transformations; how do we know if the output of one transformation is acceptable as input to another?

At the most general level, a platform's type system can be characterised by answering two questions. What is a type, and what is the substitutability relationship (if any) between two types? Further issues such as type induction can, in most cases, be considered ancillary. Fortunately, there is a large and thorough body of ongoing research into type systems, so answers to these questions are many. However, there is a perceived gap between the more mathematical models of type theory presented in the literature and those witnessed in the "real-world" programming languages.

There are a number of possible definitions to take for type. Taken broadly, it may be characterised as "suitability for some purpose", or more specifically as a "domain of interesting instances". The latter definition raises the question of whether types are defined extensionally or, as is more common, intensionally. The former, as a closed-world assumption, can have both simplifying and limiting consequences for implementation. The latter approach includes techniques such as structural-level typing, such as those used in inheritance-based or structural-conformance typing. Also worth considering are type systems including awareness of semantic domains. For example, under what circumstances is one process definition substitutable for another?

At a practical level, there are many possible choices available for typing MDA models. Using constraints as types is the most general and perhaps the most powerful, but may not be the most usable formalism for the modeller. Using patterns is also powerful, and has gained popularity in recent times. Using classes is a very familiar technique, but is limited in its expressive power in some cases. Alternatively, perhaps some hybrid of these would be useful, such as attaching constraints to classes, or adding limits to the numbers of class instances allowable.

From a wider viewpoint, having modelled type systems, there may also be a need to reason about them in comparison to one another. That is, is a given type system appropriate as a target for some mapping? A concrete example might be seen in numerical analysis, where an implementor might demand some assurance that the type system of a platform will provide sufficient numerical precision. This sort of comparison might be thought of as "type systems for type systems", since it raises similar typing substitutability questions with respect to the type systems themselves.

In conclusion, the consideration of type systems within MDA is one that has, to date, been given little treatment, and remains an avenue for research. We have asked many questions here, and provided few answers. These open questions include consideration of techniques for modelling type systems, including the notions of type and substitutability, using MDA formalisms, and in what situations these might be used.