Softw Syst Model (2009) 8:447–449
DOI 10.1007/s10270-009-0124-5

EDITORIAL

Editorial to the theme issue on metamodelling

Thomas Kühne

Published online: 4 August 2009

Linguistics, Mathematics, and Computer Science are inherently reflective disciplines. Linguists use language to talk about language. Mathematicians use mathematical methods to study mathematics itself, and Computer Scientists describe descriptions in order to facilitate automation. For instance, one of the most important concepts in computer science, the universal Turing machine, can be regarded as a program that executes other programs.

All the above examples of self-application are variations of a common "meta" theme. All of them require care in avoiding paradoxes which may be caused by uncontrolled self-reference. Linguists are faced with associating semantics to sentences like "This statement is false." This self-referencing sentence is commonly referred to as "The Liar" and its semantics refuses assignment into "true" and "false" categories. Linguists, therefore, strictly distinguish between "meta-language" and "object-language" to avoid the paradoxical nature of such antinomies. Mathematicians had to abandon naïve set theory because of Russell's set paradox, which is nothing else but a mathematical version of the aforementioned "Liar". Computer scientists have to deal with both the blessing and curse of self-referencing programs and descriptions. On the one hand, we can use this power to build interpreters which may even be parametrised with a language definition, on the other hand we have to deal with non-terminating programs and potentially inconsistent circular definitions.

T. Kühne (✉)
Victoria University of Wellington, Wellington, New Zealand
e-mail: Thomas.Kuehne@ecs.vuw.ac.nz

1 History of metamodelling

Early "meta" success stories in computer science date back as far as 1960. The programming language LISP was described using itself, giving rise to a so-called metacircular interpreter. At the same time, the first compiler (or meta-compiler) for the Atlas computer was written, i.e., a compiler that produced a compiler. This principle, to use grammar and semantics definitions to configure programs so that they behave in customised ways, lived on in environments like the Cornell program synthesizer, and led directly to today's domain-specific programming/modelling metacase tools. Metamodelling, as applied in such tools, has a number of associated advantages:

- the tool can be reused many times for a vast range of applications.
- the definition of a language is not hidden in the code of a tool, making it easier to understand and correct.
- the definition can be altered by users of the tool instead of requiring a new tool release by the manufacturer.
- one can reason about the definition and the artefacts it describes.

Despite these early and continued successes, "meta" technologies were not always viewed favourably. Early metamodellers were occasionally suspected of withdrawing into esoteric pursuits, escaping the hard realities of real, first-order modelling. The prestige and importance of metamodelling, however, has changed considerably and irrevocably in the late 1990's. In particular, the Object Management Group's use of metamodelling for defining the UML and its model-driven architecture initiative paved the way for metamodelling to become fashionable.

2 Metamodelling today

The latest incarnation in a series of attempts for raising the levels of abstraction at which we instruct hardware, i.e., model-driven development, is unthinkable without a metamodelling foundation. Languages, their definitions, and their definitions' definition are described in metamodelling description hierarchies. The same applies to the equally vital notion of transformations. Here, the self-application theme gives rise to so-called "higher-order transformations", i.e., transformations which transform transformations, which could also aptly be referred to as "meta transformations".

Today, metamodelling is an established and successful technology in a large variety of application areas such as data integration, repositories, process definition, model transformation, and (domain-specific) language definition. In addition to these linguistic applications, metamodelling is also used with an ontological flavour to remove accidental complexity in multi-level modelling scenarios. All the above applications gain value from explicitly modelling the definitions of artefacts in order to make these definitions flexible or the subject of further analysis. In summary, metamodelling has firmly established itself as an indispensable technology for state of the art MDX software development (where $X \in \{\mathrm{A},\mathrm{D},\mathrm{E},\ldots\}$).

The success of metamodelling notwithstanding, many open problems remain to be addressed. The journal *Software and Systems Modeling* hence invited original, high-quality submissions for this theme issue on Metamodelling. An extensive reviewing process with four reviews per submission and a substantial revision procedure led to the acceptance of four articles addressing a wide range of topics from theoretical to practical considerations.

3 In this issue

Despite the wide-spread use of metamodels for defining domain-specific languages, the former's formal foundation has not received the same level of attention. Pursuing a formal definition of the structural semantics of domain-specific languages is easily motivated by the desire to describe modelling artefacts independently from their implementation and the attractiveness of being able to apply analysis techniques to their metamodels and corresponding transformations. In *Formalizing the Structural Semantics of Domain-Specific Modeling Languages*, Ethan Jackson and Janos Sztipanovits use a non-monotonic extension of Horn logic to represent a language's syntax, well-formedness constraints, and corresponding transformations. This allows them to smoothly integrate syntax definitions with constraints and provide a notion of equivalence for languages. Furthermore, they can use structure preserving maps to guarantee that legal input will always be transformed to legal output. To illustrate the applicability of their formal foundation, they apply it to a scaled-down version of the Model-Integrated Computing (MIC) tool suite.

Once a model transformation has been designed it should be validated before it is relied upon in practice. One particular validation approach requires the automatic generation of a large number of sentences in the input language. In *Generating Instance Models from Meta Models*, Karsten Ehrig, Jochen Küster, and Gabriele Taentzer propose a method for using metamodels for such large scale testing. To this end, the authors introduce instance-generating graph grammars, thus allowing well-known techniques for grammar-based languages to be transferred to metamodel-based languages. The article contains a formal proof which guarantees that the derived grammars exactly produce the models induced by their respective metamodels.

Domain-specific languages can be developed more efficiently if it is possible to reuse commonly recurring building blocks. In *Supporting Domain-Specific Model Patterns with Metamodeling*, Tihamér Levendovszky, László Lengyel, and Tamás Mészáros suggest that language developers should be allowed to draw on domain-specific modelling design patterns. In order to support the representation and instantiation of such incomplete templates without requiring metamodels to be explicitly adapted, the authors introduce the theoretical foundation for a weakened form of instantiation. They demonstrate that their so-called "partial instantiation" corresponds to a useful class of implied metamodel relaxations by presenting a number of supported domain-specific design patterns from various domains.

Recently a number of extensions to standard metamodelling have been proposed, such as strict metamodelling, multi-level modelling with ontological classification, and deep instantiation. To date no single language with a formal semantics has combined all these extensions. In *Nivel: A Metamodelling Language with a Formal Semantics*, Timo Asikainen and Tomi Männistö propose such a language and define its semantics by mapping it to the Weight Constraint Rule Language. One interesting aspect of the latter is that it features cardinality constraints which allow a direct representation of attribute cardinality bounds and cardinality constraints. The authors go beyond the above mentioned metamodelling extensions, explicitly discussing how deep instantiation applies to associations and how to interpret generalisation between associations. They demonstrate the applicability of their language design by applying it to a case study on feature modelling.

Acknowledgments I thank Colin Atkinson with whom I had the pleasure to start venturing into metamodelling land. I am grateful to the Editors-in-Chief for suggesting a theme issue on Metamodelling. I furthermore thank Martin Schindler who did an outstanding job of supporting me in putting this theme issue together. The authors deserve

the most credit for making this theme issue possible by submitting their work and revising it according to the reviewers' comments. Finally, I am indebted to the reviewers for their timely efforts which in many cases directly led to considerable improvements.

Author biography

Thomas Kühne is an Associate Professor at Victoria University of Wellington, New Zealand. Prior to that he was an Assistant Professor at the Technische Universität Darmstadt, Germany, an Acting Professor at the University of Mannheim, Germany, and a Lecturer at Staffordshire University, UK. His research interests include object technology, programming language design, metamodelling, and model-driven development. He received a Ph.D. and M.Sc. from the Technische Universität Darmstadt, Germany in 1998 and 1992, respectively.