Bridging the Gap Between Formal Methods and Software Engineering Using Model-based Technology

Yann Thierry-Mieg

Bridging the Gap Between Formal Methods and Software Engineering Using Model-based Technology

Yann Thierry-Mieg

Sorbonne Universités UPMC Paris 6, CNRS UMR 7606, France
yann.thierry-mieg@lip6.fr

Abstract. Model-based technology has evolved rapidly in the last decade, bringing immediate benefits to its users. Defining domain specific languages has never been easier, thanks to the infrastructure provided by frameworks such as eclipse EMF and XText. Industrial adoption is easy, you provide specialists with just the language they need.
But this is also an opportunity for formal methods and tools to find a wider user base. A problem hindering adoption of formal methods is the effort one needs to invest in learning a particular formalism and the possible gap that exists between a handcrafted model and the reality. Model translation provides an easy way to obtain formal models from domain models that contain fine grain behavioral information, since a DSL typically also has some runtime or code generation support.
This talk will present our experiences in building tools for model-checking of various languages using models and transformations and thus leveraging the state of the art in model engineering technology.

Unified Modeling Language At the turn of the millenium, the first version of the UML was proposed [10], a mostly graphical modeling language designed to cover all artifacts of a software engineering cycle from requirements through design and development all the way to maintenance and evolution of object-oriented applications.

To achieve this goal, UML is a motley language of compromises, it aggregates several popular notations, and each diagram can be used to build both very abstract and concrete models, depending on the purpose of the diagram, the phase in the process, and the target audience (role in the process) [6].

The development of the language is rapid, and its adoption is widespread, it is pushed by the standardization work of the OMG consortium [8] including the heavyweights of the computer industry. A huge task force is set to actually defining the language in a way that allows interchange of models between tools.

The elegant and general solution adopted is to define the language through it's meta-model, a typed graph storing the syntactic information allowable in concrete model instances. Pushing the reasoning further, and to enable development of a concrete infra-structure supporting the UML [11] and its roughly 900 meta-classes, meta-meta modeling tools [7] are introduced then refined into industrial strength solutions such as the EMF [2].

Domain Specific Models The UML comes with a specialization mechanism called profiles to (further) enrich the language with new concepts. Popular UML tools propose a profile to model databases with "class diagrams" for instance. The initial idea was to leverage existing UML development environments to offer graphical modeling to users, even if they were not strictly using UML. But in practice, UML development environments are much too complex for most modeling purposes, user experience is poor as the learning curve is steep and the tools are cluttered with unused features.

The domain specific modeling language (DSL) approach clears these issues by considering small languages, with a meta-model as simple as possible, and directly linked to the concepts of the domain. DSL are not new, Makefile is a dialect for specifying artifact dependencies and construction rules in a build system, SQL is designed to query relational databases. The novelty comes from the progress of meta-meta model driven tools, now giving us the infrastructure to manipulate a meta-model so that defining such a language is easy.

Meta-Models A meta-meta model is simply a graph, each vertex has a type and carries named and typed properties, which can be primary data, or references to other vertex. The references are further qualified as being either composition or aggregation. A specific designated vertex is the root of a covering tree formed by composition references such that every vertex of the graph is reachable through a single path in this tree.

A language is defined by its meta-model, i.e. an instance of a meta-meta-model, defining the set of allowable types and references a model instance can contain. Meta-models can be graphically modeled using UML class diagram, and emerged from the same task force as the UML, leading to some confusion, but they are in fact wholly independent from the UML. The key is that a tool defined at the M2 level takes meta-models (languages) as input.

Given simply a meta-model as input, EMF [2] can generate support for serializing and reading models, manipulating models through both a language specific and language neutral rich and powerful API, tree-like model editors, support for defining validation rules.

From Graphical to Textual Models Tools such as GMF [4] even promised to offer support for building editors for graphical notations easily. But the complexity of building an ergonomic and seamless user experience for graphical models proved difficult (editing, simulation), the frameworks being so rich that delving into their customization to obtain the desired effect is very difficult. Furthermore, graphical specifications suffer from scale issues and are not adapted to all modeling tasks, for instance code is often more compact and readable than an equivalent control flow graph, even for simple algorithms.

The advent of tools supporting the definition of a textual DSL from a an annotated BNF grammar such as Xtext [12] suddenly opens adoption of model-based technology for a host of information processing applications.

Leveraging Model-based Engineering for Verification For developers of formal verification tools and algorithms this is a golden opportunity to leverage man-decades of software development and build versatile tools that can be seamlessly integrated into existing software development practices. The requirement is to align technology choices with the industry standards and follow mainstream trends in languages and development environments, e.g. Java and Eclipse. The benefits for a small team are huge however as the frameworks generate most of the heavy duty code, and their ease of customization (e.g. through dependency injection [5]) has made much progress.

Case Studies The talk is based and borrows examples from our experience in developing both graphical and textual DSL. We built VeriSensor a semi graphical notation for wireless sensor networks [1], VeriJ a DSL based on a fragment of Java for control and synthesis [13], and more recently the Guarded Action Language (GAL) a DSL to express concurrent semantics [3]. GAL serves as pivot language in our verification approach, we built transformations from existing languages (Promela, Uppaal) to GAL using model-based engineering. The language itself is supported by a powerful symbolic model-checking engine ITS-tools [9]. By building this language-based front-end we claim that we ease adoption of our formal verification tools in the software development process.

References

1. Ben Maïssa, Y., Kordon, F., Mouline, S., Thierry-Mieg, Y.: Modeling and Analyzing Wireless Sensor Networks with VeriSensor: an Integrated Workflow. Transactions on Petri Nets and Other Models of Concurrency VIII, 24–47 (Jul 2013)
2. Eclipse modeling framework (EMF), `http://www.eclipse.org/modeling/emf/`
3. Guarded action language, `http://ddd.lip6.fr/gal.php`
4. Graphical modeling framework (GMF), `http://www.eclipse.org/modeling/gmp/`
5. Google guice, a lightweight dependency injection framework, `https://github.com/google/guice`
6. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)
7. Meta object facility (MOF) 2.5. OMG standard (2015)
8. Object management group (OMG), `http://www.omg.org/`
9. Thierry-Mieg, Y.: Symbolic model-checking using ITS-Tools. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 231–237. Springer (2015)
10. Unified modeling language (UML) 1.1. OMG standard (1997)
11. Unified modeling language infrastructure (UML) 2.4.1. OMG standard (2011)
12. Xtext 'language engineering made easy', `http://eclipse.org/Xtext/`
13. Zhang, Y., Bérard, B., Hillah, L.M., Kordon, F., Thierry-Mieg, Y.: Controllability for Discrete Event Systems Modeled in VeriJ. International Journal of Critical Computer-Based Systems 5(3/4), 218–240 (Sep 2014)