View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

The K3 Model-Based Language Workbench

Thomas Degueule, Olivier Barais, Arnaud Blouin, Benoit Combemale

To cite this version:

Thomas Degueule, Olivier Barais, Arnaud Blouin, Benoit Combemale. The K3 Model-Based Language Workbench. 2014. hal-01025283

HAL Id: hal-01025283

https://hal.inria.fr/hal-01025283

Preprint submitted on 17 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The K3 Model-Based Language Workbench

Thomas Degueule*, Olivier Barais†, Arnaud Blouin‡ and Benoit Combemale*
*INRIA, †University of Rennes 1, ‡INSA Rennes, France

***Abstract*—We introduce K3, a model-based language workbench that eases the engineering of domain-specific languages. K3 features state-of-the-art facilities that increase modularity and reusability of software language artifacts to decrease their development costs. Aspect-oriented and executable metamodeling are supported through the K3AL action language. K3SLE provides software language engineering facilities such as model polymorphism and language inheritance, supported by a model-oriented typing system. We present the main components of K3, their integration into Eclipse, and the main research questions they tackle. Finally, we present the plan of the tool demonstration.**

I. Introduction

Models have been mainly used as *descriptive artifacts* allowing engineers to easily master systems complexity and ease their comprehension. Recently, however, the trend behind model-driven engineering (MDE) shifts this point of view towards *active models*, where models become the main artifacts from which software components (*e.g.* code, documentation, compilers) are automatically generated. Model-driven technologies are increasingly used in the industry [9] notably for the definition of domain-specific languages (DSL) [5].

While DSLs are gaining popularity, both their definition and tooling (*e.g.* checkers, compilers) still require significant development efforts that must be balanced with their limited number of users (by definition). K3 is a language workbench that attempts to ease the definition of DSLs and their associated tooling using MDE techniques. It features state-of-the-art facilities to speed up the development of DSLs through increased modularity and reusability in the definition of language artifacts. K3 seamlessly integrates with the different tools defined around the Eclipse Modeling Framework (EMF)[1]. K3 is distributed as an Eclipse bundle providing standard development services such as editors and compilers[2].

In the next sections, we review K3AL and K3SLE, that compose K3, and show their relations within the Eclipse runtime framework, as depicted in Figure I.

II. K3AL: Aspect-Oriented and Executable Metamodeling

K3AL is a DSL built on top of the Xtend programming language [3]. Leveraging the open-class mechanism [1], it allows to "re-open" existing meta-classes of a metamodel to insert new features such as attributes, references, or operations. This mechanism serves as the basis for the definition of *aspects* on metamodels elements, enabling aspect-oriented modeling that helps to model complex software artifacts composed of intertwined and cross-cutting concerns [6].

[1]http://www.eclipse.org/modeling/emf/
[2]https://github.com/diverse-project/k3/

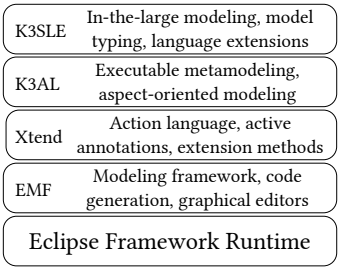

Fig. 1. K3 technology stack

K3AL defines a set of active annotations that can be used in plain Xtend files to express, for instance, aspects, pre/post conditions, or invariants. K3AL uses Xbase [3] as the expression language for defining operations bodies.

```
1  @Aspect(className = FSM)
2  class ExecutableFsm {
3    State currentState
4    def void execute(String input) { ... }
5  }
6
7  @Aspect(className = State)
8  class ExecutableState {
9    def void step(char c) { ... }
10 }
11
12 @Aspect(className = Transition)
13 class ExecutableTransition {
14   def void fire() { ... }
15 }
```

Listing 1. Aspect-oriented modeling with K3AL

Listing 1 shows the definition of a set of aspects used to weave executability into a simple finite-state machine metamodel depicted in the left part of Figure II. An aspect is a plain Xtend class consisting of a set of attributes and methods. The `@Aspect` annotation is used to specify the target meta-class in the target metamodel into which the attributes and methods are woven. In our example, the aspects are used to add a current state to the state machine and to specify the operational semantics of the execution through the definition of a set of methods that effectively runs a state machine model. Once the weaving done, the new features are statically available anywhere in the system.

III. K3SLE: A Meta-Language for Model Typing

The metamodel of a DSL can be seen as the definition of a group of related types, that is, a set of constraints over admissible graphs of objects (models). With that definition,

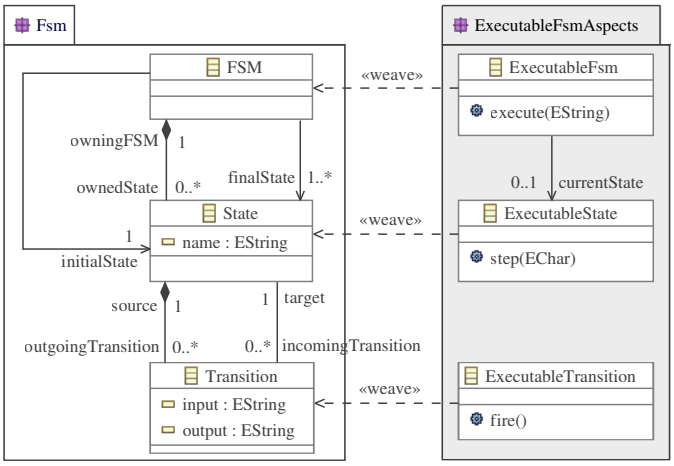

Fig. 2. Weaving executability into a finite-state machine metamodel

illustrated in Figure 3, a model m is a graph of interconnected objects respecting the type constraints MT defined by its metamodel MM. However, this definition differs in the current mainstream modeling workbenches. A metamodel MM is typically implemented with a set of classes in a given programming language (*e.g.* Java in EMF). Then, a model is concretely made of set of instances of these classes. Consequently, the XML serialization provided by EMF explicitly embeds a URI which refers to MM. Thus, in current model-driven technologies a model conforms to one and only one metamodel corresponding to the one used to create the model. As a matter of fact, many of the proposed reuse mechanisms in MDE depend on concrete metamodels [7]. So, while most of the tools would be reusable for a family of close DSLs, it is not possible in practice.

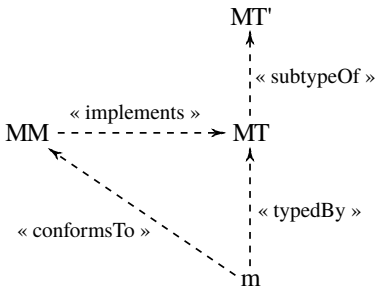

Fig. 3. Typing artifacts in MDE

The K3SLE meta-language aims at solving these problems by abstracting the overly restrictive conformance relation with a typing relation allowing to manipulate a model through different DSLs [2]. The model-oriented type system of K3SLE leverages model typing [8] and family polymorphism [4] to allow any model m conforming to MM to be seen in a polymorphic way as typed by any super type MT' of MT. This model polymorphism mechanism enables the definition of generic model-manipulation tools defined over a model type MT. Thus, any model m can be supplied to these generic tools, providing that its metamodel MM implements a model type MT' that is a subtype of MT.

The main constructs of K3SLE are *model types*, *metamodels*, and *transformations*. Listing 2 shows a simple K3SLE program where the transformation *execute* is reused over two XMI models conforming to two different metamodels. Basic inheritance and implementation relations are also described. Aspects defined with K3AL can also be directly manipulated.

```
1  modeltype FsmMT {
2    ecore "Fsm.ecore"
3  }
4  metamodel ExecFsm implements FsmMT {
5    ecore "Fsm.ecore"
6    exactType ExecFsmMT
7    aspect ExecFsm
8    aspect ExecState
9    aspect ExecTransition
10 }
11 metamodel TimedFsm inherits ExecFsm {
12   ecore "TimedFsm.ecore"
13   exactType TimedFsmMT
14   aspect TimedTransition // Overrides
15                          // ExecTransition
16 }
17 transformation execute(FsmMT m) {
18   m.contents.head.execute("abcdef")
19 }
20 @Main transformation main() {
21   val m1 = ExecFsm.load("m1.xmi", FsmMT)
22   val m2 = TimedFsm.load("m2.xmi", FsmMT)
23   execute.call(m1)
24   execute.call(m2)
25 }
```

Listing 2. Simple K3SLE program

IV. Tool Demonstration

The K3 demonstration we propose motivates the challenges it tackles and illustrates our approach and the different technical K3 components. Starting from the abstract syntax of a language defined using EMF, attendees will be introduced to aspect-oriented modeling through the use of Xtend and K3AL. K3SLE will then be used to develop representative transformations and tools around the language definition. The benefits of model polymorphism will be demonstrated through the definition of language extensions.

References

[1] C. Clifton, G. T. Leavens, C. Chambers, and T. Millstein. MultiJava: Modular open classes and symmetric multiple dispatch for Java. In *ACM Sigplan Notices*, volume 35, pages 130–145. ACM, 2000.

[2] T. Degueule, B. Combemale, O. Barais, A. Blouin, J.-M. Jézéquel, et al. Leveraging Family Polymorphism in MDE. 2014.

[3] S. Efftinge, M. Eysholdt, J. Köhnlein, S. Zarnekow, R. von Massow, W. Hasselbring, and M. Hanus. Xbase: implementing domain-specific languages for java. In *ACM SIGPLAN Notices*, pages 112–121, 2012.

[4] E. Ernst. Family polymorphism. In *ECOOP 2001*, pages 303–326. 2001.

[5] M. Fowler. *Domain-specific languages*. Pearson Education, 2010.

[6] J.-M. Jézéquel. Model driven design and aspect weaving. *Software & Systems Modeling*, 7(2):209–218, 2008.

[7] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger. Reuse in model-to-model transformation languages: are we there yet? *Software & Systems Modeling*, pages 1–36, 2013.

[8] J. Steel and J.-M. Jézéquel. On model typing. *Software & Systems Modeling*, 6(4):401–413, 2007.

[9] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85, 2014.