Kermeta, an Excutable Meta-modelling Language

Jun Li

School of Computer Science, McGill University
jun.li3@mail.mcgill.ca

Abstract. Kermeta [1] is an executable meta-modeling language, which is designed to define both structures specification and operational semantics of meta-models. It is compliant to Essential Meta-Object Facility (EMOF) [7] and built on the top of Eclipse Modeling Framework (EMF) [6]. Kermeta system fully integrated with Eclipse and provides comprehensive programming IDE for rapid development. However, it lacks of a graphical modeling environment to model the operational semantics. Overall, Kermeta creates a way for combining the powers of Object-Oriented programming and meta-modelling.

1 Introduction

Kermeta [1] is an executable meta-modeling language, which is designed to define both structures and behaviors of meta-models. It is a model-oriented language, compliant with the OMG Essential Meta-Object Facility (EMOF) [7] and Ecore of Eclipse Modeling Framework (EMF) [6]. Kermeta also provides tools for Transforming its meta-models from and to EMOF and Ecore meta-models. Kermeta can be used to precisely specify the operational semantics of meta-mode, including static semantic, similar to Object Constraint Language (OCL), and dynamic semantic defined as operations of the meta-model.

As a programming language, Kermeta is an imperative, object-oriented language. It also integrates aspect-oriented features, and supports some design-by-contract features, such as pre and post conditions, and invariants [2].

Kermeta fully integrated with Eclipse, and distributed as Eclipse plug-in. The Kermeta programming IDE provides syntax highlighting, code auto-completion, integrated with outline features, error reporting, debug system. It is a very user-friendly programming environment.

As an action language, Kermeta is powerful for specifying behaviors of meta-model. However, one of the drawbacks of Kermeta is that it lacks of a graphical modeling environment to model the operational semantics. The implementations for specifying the operational semantic need to be coded by hand. But, indeed, Kermeta provides a way to adding the power of object-oriented programming to meta-modelling.

2 Meta-modelling in Kermeta

A Kermeta meta-model consists of two parts, the structure of the meta-model defined by a class diagram and the operational semantic specified by an executable model-program. The class diagram models the meta-model, which defines the elements of modeling formalisms and their relationship constraints. It is similar as Entity-Relationship formalism used for modeling meta-model in AToM^3 [3, 4].

2.1 Class Diagram in Kermeta

The class diagram for Finite State Machine (FSM) meta-model is shown in Fig. 1. There are two kinds of attributes in Kermeta, data attributes and reference. Data attributes are shown in the class boxes. For example, the *State* class has a data attribute *name*. References attributes are shown as relation links between the referee and reference classes. For example, the top-left link shows that the *FSM* class has one reference attribute called *initialState*, which has type *State*.

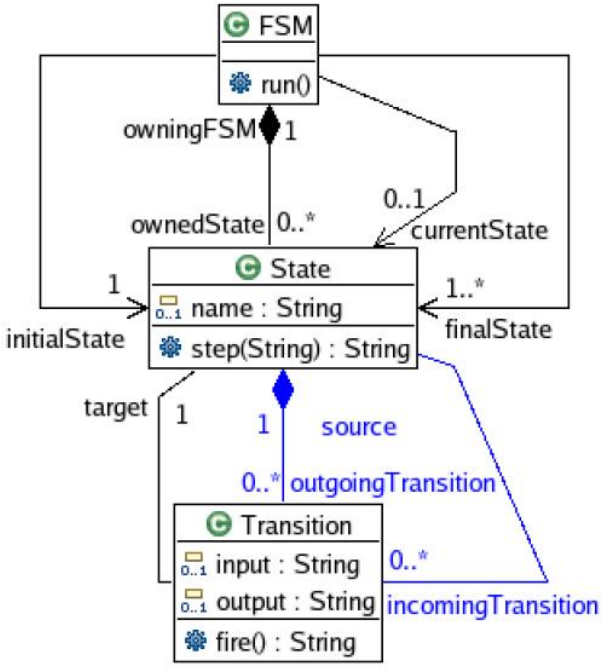

Fig. 1. Kermeta FSA Meta-Model

The operational semantic of the model are defined as operations (similar to methods) in the class diagram. For example, in the middle box, the *State* class has an operation named *step()*, and bottom box shows *Tranistion* class has an operation named *fire()*. By using class diagram, defining behavior of meta-model becomes as defining the behavior of the classes, or the interfaces of classes. Since the class diagram is one of the most widely used UML diagram in object-oriented programming, using it to define the meta-model is very intuitive and understandable for OO programmers.

However, this class diagram can only define the interfaces of the classes. The contains of those operations need to be specified in the program text model. Kermeta system can convert class diagram into Kermeta program. But, users need to add the implementation of those operations by hand. For example, the Kermeta class program of class *State* converted from class diagram, and implementation of operation *step()* are shown in Fig. 2. Based on the FSM meta-model, users

```
class Transition
{
    reference source : State[1..1]#outgoingTransition
    reference target : State[1..1]#incomingTransition
    attribute input : String
    attribute output : String

    // Fire the transition
    operation fire() : String is do
        // update FSM current state
        source.owningFSM.currentState := target
        result := output
    end
}
```

Fig. 2. Kermeta Code for Transition Class

can construct their own FSM model. Fig. 3 shows an example FSM model, which takes a string 'hello!' and produce a string 'world!'

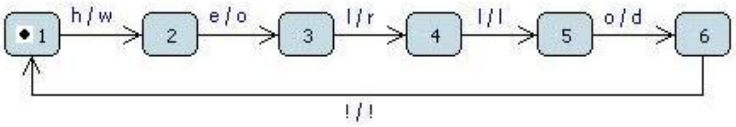

Fig. 3. Kermeta FSA Model, Hello World!

2.2 Data Format of Kermeta Meta-models

Kermeta uses an OMG standard format XMI (XML Metadata Interchange) to store metadata of meta-models. Since XMI is a standard format of Meta-Object Facility, thus Kermeta meta-models can be easily transferred from/to other systems.

The class diagram stored in .km (kermeta program in xmi format), .kmt (Kermeta program in textual syntax), or .kmdi (used for graphic editing) files. The operational semantics of the meta-model is store in a .kmt file, which is an executable program written in Kermeta language [1].

3 Kermeta Meta-modelling System

Fig. 4 shows the structure of the Kermeta meta-modeling system. The meta-model (bottom-left block) is prototyped by class diagram. It can be translate directly into Kermeta class program (middle-middle), which contains the definition of classes, their attributes and operations (similar to methods). The user constructs model (top-left block) based on the meta-model. It can be loaded by using Eclipse Modelling Framework (EMF) functions into system as instance of the model, which composed by the instances of classes defined in the class.

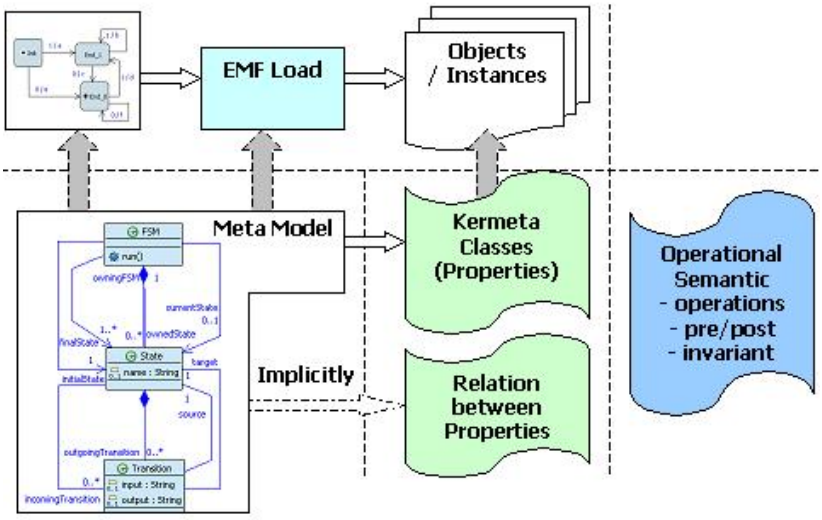

Fig. 4. FSA Model Internal Data

The relations between the classes and their attributes are also defined in the class diagram; however, they are not translated into the class program. They

implicitly exist and are required when using the model instance. Here we call them as relation program, shown in center-bottom of the figure. The operation semantic program (right-bottom block) is a hand-coded program, which consists of implementations of operations, pre and post conditions, and invariants. It has all the functionalities to traverse the graph, update the graph data, and ask user input to do the simulation. A Kermeta program usual consists of those three parts, class program, relation program, and semantic program.

3.1 Simulation in Kermeta

In Kermeta, simulations are done in the text mode. Simulating the user created model is starting from an extra launching program, written in Kermeta. The launching program first loads the user model into memory, then passes the model instance to Kermeta program and activates the simulation function of that program.

The trace of the simulation relies on the output statements written in the Kermeta program. The code will use the logic of the internal data structure of Kermeta meta-model to retrieve the data from the model instance.

3.2 Pro/post Condition of Operations

When an pre or post condition being violated during the simulation, the Kermeta system will raise an exception to notify the program. Fig. 5 shows the exception catch by the program when post condition being violated.

```
before loading
after loading
State : s1
  Transition : s1-(c/NC)->s2
State : s2
  Transition : s2-(x/y)->s2
Current state : s1
give me a letter : c
c
stepping...
[kermeta::exceptions::ConstraintViolatedPost:992]
post notVoidOutput of operation step of class State violated
```

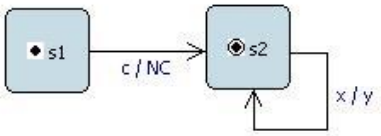

Fig. 5. Simulation Of Pre/post Conditions of Operation Step()

4 Conclusions

Kermeta builds up a bridge for linking Object-Oriented programming with meta-modelling. The class diagram used for defining meta-model facilitates the needs for converting meta-model into Object-Orient program. It also helps OO programmer to apply existed knowledge onto meta-modelling. Kermeta programs use internal data structures defined in the class diagram. Those data structures are easily being manipulated by Object-Orient programs and are very efficient when doing non-graph calculations. Also, the Kermeta programs are more readable, easy to debug in the comprehensive IDE, and meet software engineer aspect, which is a major objective for models translation.

5 Future Work

Interactive tool AToM^3 [3, 4] provides a graphic way to define the operational semantics of meta-models. In AToM^3, the operational semantics can be defined as graph grammar models. Those graph grammar models can be compiled into classes files used by programmed graph transformation systems [5].

In the following project, we plan to explore a process of using AToM^3's graph grammar model to define the operational semantics of Kermeta meta-models, then translate them into Kermeta program, which contains the same functionalities of hand coded program.

Acknowledgments. I would like to thank Professor Hans Vangheluwe, who proposed this topic, lead me to the right direction, and provided useful references.

References

1. F. Fleurey, Z. Drey, D. Vojtisek, C. Faucher.: Kermeta Language, Reference Manual. Internet: http://www.kermeta.org/docs/KerMeta-Manual.pdf, 2006.
2. Muller, PA., Fleurey, F., Jézéquel, JM., Weaving executability into object-oriented meta-languages, in Proc. of MoDELS'05, Montego Bay, Jamaica, October 2005, LNCS 3713, pp. 264–278
3. de Lara, J., Vangheluwe, H.: AToM^3: A tool for multi-formalism and meta-modelling. In European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science 2306, pages 174 - 188. Springer-Verlag (2002).
4. de Lara, J., Vangheluwe, H., Alfonseca, M.: Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. Software and Systems Modeling (SoSyM), 3(3):194-209, August 2004.
5. Eugene Syriani and Hans Vangheluwe. Programmed Graph Rewriting with Time for Simulation-Based Design. In: International Conference on Model Transformation (ICMT 2008). LNCS, Springer-Verlag, 2008.
6. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. and Grose, T. Eclipse Modeling Framework. Addison Wesley Professional, 2003.
7. OMG. MOF 2.0 Core Final Adopted Specification, Object Management Group, http://www.omg.org/cgi-bin/doc?ptc/03-10-04, 2004.