Mashup of Meta-Languages: Building Modular and Efficient DSML

The KERMETA Experience

O. Barais, B. Baudry, J. Bourcier, A. Blouin, B. Combemale, J.-M. Jézéquel, N. Plouzeau, D. Vojtisek

Didier Vojtisek – EPI Triskell – SED Rennes

April 11, 2012
Inria / Triskell

- Inria: French research institute
  - Information and Communication Science and Technologies

- Triskell team led by Prof J.-M. Jézéquel
  - One of the leader team in the domain (e.g. in terms of published papers to the MODELS conference, first paper in UML'98)
  - Implication in QVT RFP, CVL RFP, ...

- This talk is about best practices to be included in Eclipse/OMG
  - Kermeta experience: development started in 2005
Context

- DSML based tools are faster to build and easier to maintain, and improve software quality
  - Thanks to generative approaches
  - Thanks to early validation & verification
- Building DSML tools involves several DSMLs at the meta level
  - *Mashup of metalanguages!*

What our users want to build:

- Editors (textual, graphical, …)
- Documentation generators
- Test generators
- Simulators
- Analyzers
- Refactoring
- Checkers (static & dynamics)
- Compilers
- Translators
- Code generators
- Etc.
Kermeta workbench goals

- Offer a generic solution covering most model manipulation needs
- Design of DSML
  - modular and safe
- Implementation of DSML
  - scalable, interoperable and efficient @ runtime
Designing a DSML

Editors
(textuals, graphicals, …)

Documentation generators

Test generators

Simulators

Analyzers

Refactoring

Checkers
(static & dynamics)

Compilers

Translators

Code generators

Etc.
Designing a DSML

Abstract Syntax (+ static semantics)

Concrete syntax
- xText, EMFText, GMF, Topcased, Antlr…

Dynamic Semantics
- QVT*, Kermeta AL, Java EMF…

MOF/Ecore + OCL

Kermeta workbench makes it possible to mashup these meta-languages
Designing a DSML
Separation of Concerns

- A DSML involves several concerns (syntax, semantics...), for many purposes (edition, simulation, generation, interoperability...)
  - Use appropriate language for each use
  - Compose them as necessary

Solution in the Kermeta Workbench: Static Introduction
- Ecore: abstract syntax (metamodel) is the core language
- OCL: constraints (inv., pre-, post-)
- Kermeta Action Language: side effects = model manipulation
- Additional extension mechanisms: add classes, operations, attributes, property opposite, inheritance ...

Composition mechanism:
- Model element composition keyword: `aspect`
- Model composition keyword: `require`

Not to be confused with AspectJ aspects!
Principle of the mashup

context FSM
inv: ownedState->forAll(s1,s2|s1.name=s2.name implies s1=s2)

aspect class FSM{
  reference currentState : State
  operation run() : Void is do
    ...
  end
  ...
}

aspect class Transition{
  operation fire() : EString is do
    source.owningFSM.currentState := target
    result := output
  end
}

Model of the resulting composition

Building modular and efficient DSL tools with Kermeta
Action language feature highlight

- **Statically Typed**
  - Generics, Function types (for OCL-like iterators)

- **Object-Oriented**
  - Multiple inheritance / dynamic binding / reflection

- **Model-Oriented**
  - Model are first class citizens, notion of model type

- **Design by contract**

- **Aspect-Oriented**
  - Simple syntax for static introduction

- **Still “kernel” language**
  - Use JVM bytecode compatibility for GUI/IO etc.
Separation of concerns using OO Visitor or AO Visitor

MyCompiler

main()

<<Abstract>>

Visitor

visit(e : VisitableElement)

Concrete Visitor

visit(e : VisitableElement)

Concrete Element

metamodel .ecore

metamode_visitable .ecore

v.visit(self)

accept(v : Visitor)

MyCompiler

main()

root

Concrete Element

Simu1 .kmt

Simu1 .java

Simu1.kmt

Simu1.java

compiler1 .java

compiler2 .java

compiler1 .kmt

compiler2 .kmt

Ecore-to-EcoreVisitable

<<require>>

<<require>>

<<require>>

<<require>>

<<require>>

Building modular and efficient DSL tools with Kermeta
Implementing a DSML: Mashup of the concerns into bytecode

Standard EMF tooling

Ecore

Kermeta (static semantics)

Kermeta (behavior semantics)

KMT Aspect Node

KMT Aspect Transition

KMT Class Main

Ecore

interface Node {...}
interface Transition {...}
class NodeImpl {...}
class TransitionImpl {...}
class Factory {...}

KMT to Scala

trait NodeAspect {...}
trait TransitionAspect {...}
trait Main {...}
trait MainAspect {...}
class Factory {...}

Scala

Java to ByteCode

JVM-based DSL Runtime

<use>

Java to ByteCode

 Scala to ByteCode
Implementing a DSML

Consequence

- Runtime is more efficient
- Ex Hashtable: no search, ask directly the object
- Ex visitor: No more "ping pong" overhead between accept and visit methods

• Ex: fUML visitor reference java implementation vs. Kermeta implementation

Published fUML samples
Future work: compile model type too

Model Type – intuitive presentation

One model type can be inferred (or manually mapped) from the effective metamodel*

* Effective metamodel = subgraph of the metamodel with the minimal set of classes typing objects belonging to the typed/conforming model
Conclusion: Kermeta Workbench

• Modular design of DSMLs
  • One metalanguage per language concern
    • Ecore, OCL, Kermeta Action Language
    • But also: QVTo, fUML, Alf, Ket, Xsd…
    • Specific architecture supporting this
  • Static introduction mechanism

• Efficient implementation of DSMLs
  • Mashup of the metalanguages to efficient bytecode
    • is extensible
  • Integrates with third-party tools (EMF compliant)
Conclusion: Kermeta Action Language

- Provides a model oriented action language
  - to implement (E)Operation’s bodies
  - allows common model manipulation tasks

- Imperative, statically typed, object-oriented, aspect-oriented, model-oriented, DbC, Unit testing

- Java and Scala compliant
- EMF based
- Run as Eclipse plugin or as standard Java application
Conclusion Kermeta

Smoothly interoperates with Eclipse/EMF

Open Source, available on the Inria Forge
- (V1.4.x or V2-beta4)
- official V2 -> 06/12

- Home page
  - http://www.kermeta.org

- Development page
  - didier.vojtisek@inria.fr

Breathe life into your metamodels

Ideas from kermeta could be pushed to other tools