Model-Driven Language Engineering

Franck Fleurey
e-mail : Franck.Fleurey@sintef.no
http://www.fleurey.com

SINTEF
Example with StateMachines

Model

Meta-Model
Breathing life into Meta-Models

// MyKermetaProgram.kmt
// An E-MOF metamodel is an OO program that does nothing
require "StateMachine.ecore" // to import it in Kermeta
// Kermeta lets you weave in aspects
// Contracts (OCL WFR)
require “StaticSemantics.ocl”
// Method bodies (Dynamic semantics)
require “DynamicSemantics.kmt”
// Transformations

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

aspect class FSM {
operation reset() : Void {
currentState := initialState
}

class Minimizer {
operation minimize (source: FSM): FSM {
 ...
}
}
Outline

- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up
The LOGO Language

- Consider LOGO programs of the form:

  ```logo
  repeat 3 [ pendown forward 3 penup forward 4 ]
  ```

  ```logo
  to square :width
    repeat 4 [ forward :width right 90]
  end
  ```

  ```logo
  pendown square 10 *10
  ```
Fractals in LOGO

```
; lefthilbert
to lefthilbert :level :size
  if :level != 0 [  
    left 90  
    righthilbert :level-1 :size
    forward :size
    right 90  
    lefthilbert :level-1 :size
    forward :size
    lefthilbert :level-1 :size
    right 90
    forward :size
    righthilbert :level-1 :size
    left 90
  ]
end

; righthilbert
to righthilbert :level :size
  if :level != 0 [  
    right 90  
    lefthilbert :level-1 :size
    forward :size
    left 90  
    righthilbert :level-1 :size
    forward :size
    righthilbert :level-1 :size
    left 90
    forward :size
    lefthilbert :level-1 :size
    right 90
  ]
end
```
Case Study: Building a Programming Environment for Logo

- Featuring
  - Edition in Eclipse
  - On screen simulation
  - Compilation for a Lego Mindstorms robot
Model Driven Language Engineering : the Process

- Specify abstract syntax
- Specify concrete syntax
- Build specific editors
- Specify static semantics
- Specify dynamic semantics
- Build simulator
- Compile to a specific platform
Meta-Modeling LOGO programs

- Let’s build a meta-model for LOGO
  - Concentrate on the abstract syntax
  - Look for concepts: instructions, expressions…
  - Find relationships between these concepts
    » It’s like UML modeling!

- Defined as an ECore model
  - Using EMF tools and editors
LOGO metamodel

ASMLogo.ecore
Concrete syntax

- Any regular EMF based tools
- Textual using Sintaks
- Graphical using GMF or TopCased
Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up
Static Semantics with OCL

- Complementing a meta-model with Well-Formedness Rules, aka Contracts e.g.;
  - A procedure is called with the same number of arguments as specified in its declaration
- Expressed with the OCL (Object Constraint Language)
  - The OCL is a language of typed expressions.
  - A constraint is a valid OCL expression of type Boolean.
  - A constraint is a restriction on one or more values of (part of) an object-oriented model or system.
Contracts in OO languages

- Inspired by the notion of Abstract Data Type
- Specification = Signature +
  - Preconditions
  - Postconditions
  - Class Invariants
- Behavioral contracts are inherited in subclasses
OCL

- Can be used at both
  - M1 level (constraints on Models)
    » aka Design-by-Contract (Meyer)
  - M2 level (constraints on Meta-Models)
    » aka Static semantics

- Let’s overview it with M1 level exemples
Simple constraints

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
</tr>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>age: Integer</td>
</tr>
<tr>
<td>isMale: Boolean</td>
</tr>
</tbody>
</table>

title = if isMale then 'Mr.' else 'Ms.' endif
age >= 18 and age < 66
name.size < 100
Non-local contracts: navigating associations

- Each association is a navigation path
  - The context of an OCL expression is the starting point
  - Role names are used to select which association is to be traversed (or target class name if only one)

```
Person 1 owner ownership ownings * Car

Context Car inv:
  self.owner.age >= 18
```
Navigation of 0..* associations

- Through navigation, we no longer get a scalar but a collection of objects
- OCL defines 3 sub-types of collection
  - **Set**: when navigation of a 0..* association
    - Context Person inv: ownings return a Set[Car]
    - Each element is in the Set at most once
  - **Bag**: if more than one navigation step
    - An element can be present more than once in the Bag
  - **Sequence**: navigation of an association {ordered}
    - It is an ordered Bag
- Many predefined operations on type collection

Syntax::

Collection->operation
Collection hierarchy

Collection

Set
- minus
- symmetricDifference
- asSequence
- asBag

Bag
- asSequence
- asSet

Sequence
- first
- last
- at(int)
- append
- prepend
- asBag
- asSet
Basic operations on collections

- **isEmpty**
  - *true* if collection has no element

- **notEmpty**
  - *true* if collection has at least one element

- **size**
  - Number of elements in the collection

- **count (elem)**
  - Number of occurrences of element *elem* in the collection

Context Person inv:
age<18 implies ownings->isEmpty
select Operation

- possible syntax
  - `collection->select(elem:T | expr)`
  - `collection->select(elem | expr)`
  - `collection->select(expr)`

- Selects the subset of `collection` for which property `expr` holds

- e.g.
  ```
  context Person inv:
  ownings->select(v: Car | v.mileage<100000)->notEmpty
  ```

- shortcut:
  ```
  context Person inv:
  ownings->select(mileage<100000)->notEmpty
  ```
ForAll Operation

- possible syntax
  - \texttt{collection->forall(elem:T | expr)}
  - \texttt{collection->forall(elem | expr)}
  - \texttt{collection->forall(expr)}

- True iff \textit{expr} holds for each element of the \textit{collection}

- e.g.

  context Person inv:  
  owning->forall(v: Car | v.mileage<100000)

- shortcut:

  context Person inv:  
  owning->forall(mileage<100000)
# Operations on Collections

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>The number of elements in the collection</td>
</tr>
<tr>
<td>count(object)</td>
<td>The number of occurences of object in the collection.</td>
</tr>
<tr>
<td>includes(object)</td>
<td>True if the object is an element of the collection.</td>
</tr>
<tr>
<td>includesAll(collection)</td>
<td>True if all elements of the parameter collection are present in the current collection.</td>
</tr>
<tr>
<td>isEmpty</td>
<td>True if the collection contains no elements.</td>
</tr>
<tr>
<td>notEmpty</td>
<td>True if the collection contains one or more elements.</td>
</tr>
<tr>
<td>iterate(expression)</td>
<td>Expression is evaluated for every element in the collection.</td>
</tr>
<tr>
<td>sum(collection)</td>
<td>The addition of all elements in the collection.</td>
</tr>
<tr>
<td>exists(expression)</td>
<td>True if expression is true for at least one element in the collection.</td>
</tr>
<tr>
<td>forAll(expression)</td>
<td>True if expression is true for all elements.</td>
</tr>
</tbody>
</table>
Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:

- A procedure is called with the same number of arguments as specified in its declaration:
Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:
  
  ```
  context ProcDeclaration
  inv unique_names_for_formal_arguments :
  args -> forAll ( a1 , a2 | a1. name = a2.name
  implies a1 = a2 )
  ```

- A procedure is called with the same number of arguments as specified in its declaration:
  
  ```
  context ProcCall
  inv same_number_of_formals_and_actuals :
  actualArgs -> size = declaration .args -> size
  ```
EMF Validation Framework

- [http://www.eclipse.org/modeling/emf/?project=validation](http://www.eclipse.org/modeling/emf/?project=validation)
- Tutorial available in the EMF documentation
EMFText

- http://www.emftext.org
- Tutorial, documentation and lots of example on the web site.
- Annotations for expressions
Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up
Operational Semantics of State Machines

- A model

- Its metamodel

- Adding Operational Semantics to OCL Metamodels
Kermeta workbench snapshot
Kermeta: a *Kernel* metamodelling language

- Strict EMOF extension
- Statically Typed
  - Generics, Function types (for OCL-like iterators)
- Object-Oriented
  - Multiple inheritance / dynamic binding / reflection
- Model-Oriented
  - Associations / Compositions
  - Model are first class citizens, notion of model type
- Aspect-Oriented
  - Simple syntax for static introduction
  - Arbitrary complex aspect weaving as a framework
- Still “kernel” language
  - Seamless import of Java classes in Kermeta for GUI/IO etc.
**EMOF ↔ Kermit**

---

**Class FSM**

```java
class FSM {
    attribute ownedState : State[0..*]#owningFSM
    reference initial FSM : State[1..1]
    reference currentState : State
    operation run() : kermeta::standard::~Void is do
    end
    operation reset() : kermeta::standard::~Void is do
    end
}
```

**Class State**

```java
class State {
    reference owning FSM : FSM[1..1]#ownedState
    attribute name : String
    attribute outgoingTransition : Transition[0..*]#source
    reference incomingTransition : Transition#target
    operation step(c : String) : kermeta::standard::~Void is do
    end
}
```

**Class Transition**

```java
class Transition {
    reference source : State[1..1]#outgoingTransition
    reference target : State[1..1]#incomingTransition
    attribute input : String
    attribute output : String
    operation fire() : String is do
    end
}
```
Assignment semantics

**Composition**

Before

After

**Association**

Before

d2.c := c1

After
Example

operation fire() : String

source.owningFSM.currentState := target
result := output
operation step(c : String) : String

// Get the valid transitions
var validTransitions : Collection<Transition>
validTransitions := outgoingTransition.select { t |
    t.input.equals(c)
}

// Check if there is one and only one valid transition
if validTransitions.empty then raise NoTransition.new end
if validTransitions.size > 1 then
    raise NonDeterminism.new
end

// fire the transition
result := validTransitions.one.fire
operation run() : Void

from var str : String
until str == "exit"
loop
    stdio.writeln("current state is " + currentState.name)
    str := stdio.read("Enter an input string or 'exit'
    to exit simulation : ")
    stdio.writeln(str)
    if str != "exit" then
        do
            stdio.writeln("Output string : " + currentState.step(str))
            rescue (ex : FSMException)
                stdio.writeln("ERROR : " + ex.toString)
        end
    end
end
stdio.writeln("* END OF SIMULATION *")
/**
 * Load a sample FSM from a xmi2 file
 */

operation loadFSM() : FSM is do
    var repository : EMFRepository init EMFRepository.new
    var resource : EMFResource
    resource ?= repository.createResource("../models/fsm_sample1.xmi", "/models/fsm_sample1.xmi", "/metamodels/fsm.ecore")
    resource.load

    // Load the fsm (we get the main instance)
    result ?= resource.instances.one
end
Operational Semantics for LOGO

- Expressed as a mapping from a meta-model to a virtual machine (VM)
- LOGO VM ?
  - Concept of Turtle, Lines, points...
  - Let’s Model it !
  - (Defined as an Ecore meta-model)
Virtual Machine - Model

- Defined as an Ecore meta-model

VMLogo.ecore
require "VMLLogo.ecore"
require "TurtleGUI.kmt"

aspect class Point {
  method toString() : String is do
    result := "[" + x.toString + "," + y.toString + "]"
  end
}

aspect class Turtle {
  operation setPenUp(b : Boolean) is do
    penUp := b
  end
  operation rotate(angle : Integer) is do
    heading := (heading + angle).mod(360)
  end
}
Map Instructions to VM Actions

- Weave an interpretation aspect into the meta-model
  - add an `eval()` method into each class of the LOGO MM

```java
aspect class PenUp {
    eval (ctx: Context) {
        ctx.getTurtle().setPenUp(true)
    }

    ... aspect class Clear {
        eval (ctx: Context) {
            ctx.getTurtle().reset()
        }
    }
```
Meta-level Anchoring

- Simple approach using the Kermit VM to « ground » the semantics of basic operations
- Or reify it into the LOGO VM
  - Using eg a stack-based machine
  - Ultimately grounding it in kermita though

```java
aspect class Add {
  eval (ctx: Context) : Integer {
    result = lhs.eval(ctx)
    + rhs.eval(ctx)
  }
}
```

```java
aspect class Add {
  eval (ctx: Context) {
    lhs.eval(ctx) // put result
    // on top of ctx stack
    rhs.eval(ctx) // idem
    ctx.getMachine().add()
  }
}
```
Handling control structures

- Block
- Conditional
- Repeat
- While
Operational semantics

```
require "ASMLogo.ecore"
require "LogoVMSemantics.kmt"

aspect class If {
    operation eval(context : Context) : Integer is do
        if condition.eval(context) != 0 then
            result := thenPart.eval(context)
        else result := elsePart.eval(context)
        end
    end
}

aspect class Right {
    operation eval(context : Context) : Integer is do
        context.turtle.rotate(angle.eval(context))
    end
}
```
Handling function calls

- Use a stack frame
  - Owned in the Context
Getting an Interpreter

- Glue that is needed to load models
  - ie LOGO programs

- Vizualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure
Simulator

- Execute the operational semantics

TO k :scale
PENDOWN
FORWARD *(30, :scale)
PENUP
BACK *(10, :scale)
RIGHT 45
FORWARD *(14, :scale)
PENDOWN
BACK *(14, :scale)
PENUP
RIGHT 90
FORWARD *(14, :scale)
PENDOWN
BACK *(14, :scale)
PENUP
RIGHT 45
FORWARD *(20, :scale)
LEFT 180

CLEAR
$k(4)

Launching logo interpreter on file: /home/
Tortue trace vers [0,120]
Tortue se deplace en [0,80]
Tortue se deplace en [39,119]
Tortue trace vers [0,80]
Tortue se deplace en [39,41]
Tortue trace vers [0,80]
Tortue se deplace en [0,0]
Execution terminated successfully.
Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up
Implementing a model-driven compiler

- Map a LOGO program to Lego Mindstroms
  - The LOGO program is like a PIM
  - The target program is a PSM
  - => model transformation

- Kermeta to weave a « compilation » aspect into the logo meta-model

```java
aspect class PenUp {
    compile (ctx: Context) {
        ...
    }
}

aspect class Clear {
}
```
Specific platform

- Lego Mindstorms Turtle Robot
  - Two motors for wheels
  - One motor to control the pen
Model-to-Text vs. Model-to-Model

- Model-to-Text Transformations
  - For generating: code, xml, html, doc.
  - Should be limited to syntactic level transcoding

- Model-to-Model Transformations
  - To handle more complex, semantic driven transformations
Model-to-Text Approaches

- For generating: code, xml, html, doc.
  
  - Visitor-Based Approaches:
    - Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - Iterators, Write()
  
  - Template-Based Approaches
    - A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
    - The structure of a template resembles closely the text to be generated
    - Textual templates are independent of the target language and simplify the generation of any textual artefacts
Classification of Model-to-Model Transformation Techniques

1. General purpose programming languages
   - Java/C#

2. Generic transformation tools
   - Graph transformations, XSLT

3. CASE tools scripting languages
   - Objecteering, Rose

4. Dedicated model transformation tools
   - OMG QVT style

5. Meta-modeling tools
   - Metacase, Xactium, Kermeta
Logo to NXC Compiler

- Step 1 – Model-to-Model transformation

- Step 2 – Code generation with template
Step 1: Model-to-Model

- Goal: prepare a LOGO model so that code generation is a simple traversal
  - => Model-to-Model transformation

- Example: local2global
  - In the LOGO meta-model, functions can be declared anywhere, including (deeply) nested, without any impact on the operational semantics
  - for NXC code generation, all functions must be declared in a “flat” way at the beginning of the outermost block.
  - => implement this model transformation as a local-to-global aspect woven into the LOGO MM
Step 1: Model-to-Model example

// aspect local-to-global
aspect class Statement {
    method local2global(rootBlock: Block) is do
    end
}
aspect class ProcDeclaration
    method local2global(rootBlock: Block) is do
        ...
    end
}
aspect class Block
    method local2global(rootBlock: Block) is do
        ...
    end
}
...
Step 2: Kermeta Emitter Template

- NXC Code generation using a template
Execution

```
TO k :scale
  PENDOWN
  FORWARD *(30, :scale)
  PENUP
  BACK *(10, :scale)
  RIGHT 45
  FORWARD *(14, :scale)
  PENDOWN
  BACK *(14, :scale)
  PENUP
  RIGHT 90
  FORWARD *(14, :scale)
  PENDOWN
  BACK *(14, :scale)
  PENUP
  RIGHT 45
  FORWARD *(20, :scale)
  LEFT 180
END

CLEAR
$k(4)
```
Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up
Logo Summary (2)

- Integrate all aspects coherently
  - syntax / semantics / tools

- Use appropriate languages
  - MOF for abstract syntax
  - OCL for static semantics
  - Kermeta for dynamic semantics
  - Java for simulation GUI
  - ...

- Keep separation between concerns
  - For maintainability and evolutions
From LOGO to Mindstorms

- Embedded source code inside the robot
- Transformation written in Kermeta
- Interaction between the current simulation (Kermeta) and the GUI (Java)
- Simulator in Kermeta
- Result of a real execution
- Result of a simulation interpreted with Kermeta
- Logo Semantic in Kermeta
- Static constraints in OCL
Kermeta

- Kermeta is an open-source initiative
  - Started January 2005
  - More than 10 active developers

- Feel free to use
  - Start with a meta-model in EMF
    » Get XML an reflective editor for free
  - Weave in static semantics in OCL
  - Weave in an interpreter,
    » connect to a simulation platform
  - Weave in one or more compilers
  - Finally care for concrete syntax issues

- Feel free to contribute!
  - www.kermeta.org
Thank you!

- Questions?