Model-Driven Language Engineering

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

SINTEF
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
Once upon a time… software development looked simple

- From the object as the only one concept
  - As e.g. in Smalltalk
- To a multitude of concepts

Middleware (middle war)

It's difficult -- in fact, next to impossible -- for a large enterprise to standardize on a single middleware platform. (R. Soley)

Collaborations

Components

Design patterns

Aspects

Middleware

- COM
- DCOM
- HTTP
- HTML
- Sun's Java & EJB
- Microsoft C# & .Net
- Proprietary Middleware (e.g. automotive)

Required Port               Provided Port
<<Component>>

Decoder

Decoded port

DecoderIReceiverI

Data

CORBA

Optimizer

CodeGen

NodeObserver

+ acceptVisitor
+ attachObserver
+ deleteObserver
+ markModel
+ nodeCounter:INTEGER
+ UpateCounter( ... )
Why modeling: master complexity

- Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose.

- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Jeff Rothenberg.
A Model is a *simplified* representation of an aspect of the World for a specific *purpose*.

**Specificity of Engineering:** Model something not yet existing (in order to build it)

$M_0$ (the world)

$M_1$ (modeling space)

Is represented by
Model and Reality in Software

- Sun Tse: *Do not take the map for the reality*
- Magritte

Software Models: from contemplative to productive
Modeling and Weaving

Challenges:
- Product Families
- Reuse of Weaving Process
- Automatic Weaving

- QoS Model
- Security Model
- Business Model
- UI Model
- Use Case Model
- Object Model
- Platform Model
- Design Model
- Test Model
- Code Model

INF5120 – 2011 – Franck Fleurey (franck.fleurey@sintef.no)
Assigning Meaning to Models

- If a model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...

- Then tools must be able to manipulate models
  - Let’s make a model of what a model is!
  - => *meta-modeling*
    » & meta-meta-modeling...
    » Use Meta-Object Facility (MOF) to avoid infinite Meta-recursion
UML2 meta-model (part., © OMG)
Generalizations

Figure 3-33. Examples of generalizations between classes.

NB: Tell you nothing about:
• generalization being acyclic,
• or semantics of dynamic binding
The 4 layers in practice
Comparing Abstract Syntax Systems

Technology #1
(formal grammars
attribute grammars,
etc.)

Technology #2
(MOF + OCL)

Technology #3
(XML Meta-Language)

Technology #4
(Ontology engineering)

M³

EBNF

Pascal Language
Grammar

The UML
meta-Model

A Specific
Pascal Program

A Specific
execution of a Pascal
program

MOF

A XML DTD
Or Schema

A Specific
UML Model

A Specific
phenomenon corresponding to
a UML Model

A XML DTD
or Schema

A XML document

A XML document

KIF

Upper Level
Ontologies

Theories

+Description
Logics
+Conceptual
Graphs
+etc.

+ Xlink, Xpath, XSLT
+ RDF, OIL, DAML
+ etc.

[XMI=MOF+XML+OCL]

(From J. Bézivin)
MDA: the OMG vision

"OMG is in the ideal position to provide the model-based standards that are necessary to extend integration beyond the middleware approach... Now is the time to put this plan into effect. Now is the time for the Model Driven Architecture."

Richard Soley & OMG staff, MDA Whitepaper Draft 3.2
November 27, 2000
Mappings to multiple and evolving platforms

- MOF & UML as the core
- Organization assets expressed as models
- Model transformations to map to technology specific platforms

Platform neutral models based on UML & MOF

- COM+
- DCOM
- CORBA
- Java
- EJB
- .Net
- HTTP
- HTML
- XML
- SOAP
The core idea of MDA: PIMs & PSMs

MDA models

- **PIM**: Platform Independent Model
  - Business Model of a system abstracting away the deployment details of a system
  - Example: the UML model of the GPS system

- **PSM**: Platform Specific Model
  - Operational model including platform specific aspects
  - Example: the UML model of the GPS system on .NET
    - Possibly expressed with a UML profile (.NET profile for UML)
  - Not so clear about platform models
    - Reusable model at various levels of abstraction
      - CCM, C#, EJB, EDOC, ...
Model Driven Engineering: Summary

- **Modeling to master complexity**
  - Multi-dimensional and aspect oriented by definition

- **Models: from contemplative to productive**
  - Meta-modeling tools, meta-models used to define languages

- **Model Driven Engineering**
  - Weaving aspects into a design model
    » E.g. Platform Specificities

- **Model Driven Architecture (PIM / PSM): just a special case of Aspect Oriented Design**

- **Related: Generative Prog, Software Factories**
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
Meta-Models as Shared Knowledge

- Definition of an Abstract Syntax in E-MOF
  - Repository of models with EMF
  - Reflexive Editor in Eclipse
  - JMI for accessing models from Java
  - XML serialization for model exchanges

- Applied in more and more projects
  - SPEEDS, OpenEmbedd, DiVA...
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 {...}
}
DIY with LOGO programs

- Consider LOGO programs of the form:

  repeat 3  [ pendown forward 3 penup forward 4 ]

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

  pendown square 10 *10

http://en.wikipedia.org/wiki/Logo_(programming_language)
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  
    left 90  
    righthilbert :level-1 :size  
    forward :size  
    right 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  
    left 90  
    righthilbert :level-1 :size  
    forward :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
Concrete syntax

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

```
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
```
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
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

Customer

| name: String |
| title: String |
| age: Integer |
| isMale: Boolean |

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
  - collection->forall(elem:T | expr)
  - collection->forall(elem | expr)
  - collection->forall(expr)

- True iff expr holds for each element of the collection

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

- shortcut:
  ```
  context Person inv:
  ownings->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 occurrences 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
OBLIG 1: – “MDE tools in practice”

1a. Training in EMF and EMF Validation Framework

The goal of this exercise is to create a meta-model for the LOGO language and to implement its static semantics using the EMF Validation Framework.

You can choose any version of the logo language you want to implement (see http://en.wikipedia.org/wiki/Logo_(programming_language)) but it should have at least procedure definitions and calls in addition to the basic constructions.

1b. Training in EMFTText

The EMFTText tool allow to specify textual concrete syntax for domain specific modelling languages. The eclipse plugin, documentation and many examples can be found at http://www.emftext.org.

The goal of the exercise is to create a text editor for your LOGO language. Your textual syntax should include proper parsing of expressions with correct priorities of operators and without the need for redundant parenthesis. Hint: make use of the operators annotations available in the latest version of emftext. Your parser should ensure proper scoping of variables and especially procedure parameters.