Model-Driven Language Engineering

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

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.

Modeling in Science & Engineering

- A Model is a simplified representation of an aspect of the World for a specific purpose

Model and Reality in Software

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

© J.-M. Jézéquel et al., 2010
Modeling and Weaving

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

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

Assigning Meaning to Models
If a model is no longer just fancy pictures to decorate your room or 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

UML2 meta-model (part., © OMG)

Generalizations

NB: Tell you nothing about:
- generalization being acyclic,
- or semantics of dynamic binding

The 4 layers in practice

Comparing Abstract Syntax Systems

(M2) Formal grammars (MOF + OCL)

(M3) XML Meta-Language

(M4) Ontology engineering

(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

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
      - COM, CIW, 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 Specifics
- 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

Meta-Model

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

Fractals in LOGO

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

end

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

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

LOGO metamodel

Concrete syntax

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

Do It Yourself

- Within Eclipse
  - Load/Edit/Save Models
    - Conforming to the LOGO meta-model
    - ie LOGO programs

- Install & Run the MDLE4LOGO Bundle
  - On your own PC
  - Or follow the beamed demo

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 examples

**Simple constraints**

```plaintext
Customer

<table>
<thead>
<tr>
<th>name: String</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = if isMale then 'Mr.' else 'Ms.' endif</td>
</tr>
<tr>
<td>age: Integer</td>
</tr>
<tr>
<td>isMale : Boolean</td>
</tr>
</tbody>
</table>

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
```

**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**
static semantics for logo

- **count(object)**: The number of occurrences of object in the collection.
- **contains(object)**: True if the object is an element of the collection.
- **includes(object)**: True if the object is in the collection.
- **isEmpty** (context Person inv): True if the collection contains no elements.
- **includesAll(collection)**: True if all elements of the parameter collection are present in the current collection.
- **isEmpty** (collection): True if the collection contains no elements.
- **forAll(expression)**: True if expression is true for at least one element in the collection.
- **size** (collection): Number of elements in the collection.
- **size** (elem): Number of occurrences of element elem in the collection.
- **includes(object)**: True if the object is an element of the collection.
- **isEmpty** (collection): True if the collection contains no elements.
- **forAll(expression)**: True if expression is true for at least one element in the collection.
- **size** (collection): Number of elements in the collection.
- **size** (elem): Number of occurrences of element elem in the collection.

**Possible syntax**
- `collection->select(elem | exp)`
- `collection->select(elem | exp)`
- `collection->select(exp)`

**Shortcuts**
- `context Person inv:`
  - `ownings->select(v: Car | v.mileage<100000)->notEmpty`
- `context Person inv:`
  - `ownings->select(mileage<100000)->notEmpty`
- `context Person inv:`
  - `ownings->forall(mileage<100000)

**Description**
- **isEmpty** (context Person inv): True if collection has no element.
- **notEmpty** (collection): True if collection has at least one element.
- **size** (collection): Number of elements in the collection.
- **count (elem)** (collection): Number of occurrences of element elem in the collection.

**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>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>includes</td>
<td>True if the object is in the collection.</td>
</tr>
<tr>
<td>contains</td>
<td>True if the object is an element of the collection.</td>
</tr>
<tr>
<td>forAll(expression)</td>
<td>True if expression is true for at least one element in the collection.</td>
</tr>
<tr>
<td>sum(collection)</td>
<td>The addition of all elements in the collection.</td>
</tr>
<tr>
<td>iterate(expression)</td>
<td>Expression is evaluated for every element in the collection.</td>
</tr>
<tr>
<td>minus</td>
<td>Possible syntax: A procedure is called with the same number of arguments as specified in its declaration.</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:
  ```plaintext
  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:
  ```plaintext
  context ProcCall
  inv same_number_of_formals_and_actuals :
  actualArgs -> size = declaration.args -> size
  ```

Operational Semantics of State Machines

- A model
- Its metamodel
- Adding Operational Semantics to OO Metamodels

Kermeta Rationale

- Model, meta-model, meta-metamodel, DSLs...
  - Meta-bla-bla too complex for the normal engineer
- On the other hand, engineers are familiar with
  - OO programming languages (Java, C#, C++, ...)
  - UML (at least class diagram)
  - May have heard of Design-by-Contract
- Kermeta leverages this familiarity to make Meta-modeling easy for the masses
Kermeta: a Kernel metadata modeling 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/O etc.

EMOF ↔ Kermeta

Example

operation fire() : String

source.owningFSM.currentState := target
result := output

operation run() : Void

from var s : String
until s = "END OF SIMULATION"
loop
    if s = "end" then
        break
    else
        stdio.writeln("ERROR: ", ex.toString)
        among FSMException
        raise ex
    end
end

Example

operation step(c : String) : String

-- Get the valid transitions
var validTransitions : Collection<Transition>
validTransitions := outgoingTransition.select [ it.input.equals(c) ]

-- Check if there is one and only one valid transition
if validTransitions.isEmpty then raise NoTransition, new End
if validTransitions.size > 1 then raise NonDetermination, new End
result = validTransitions.one.fire

Assignment semantics

Composition

Association

Before

After

Before

After

Composition

Association
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

Map Instructions to VM Actions

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

Meta-level Anchoring

- Simple approach using the Kermeta 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 kermeta though
Handling control structures

- Block
- Conditional
- Repeat
- While

Operational semantics

```java
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
  - i.e. LOGO programs
- Visualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure

Simulator

- Execute the operational semantics

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 Mindstorms
  - The LOGO program is like a PIM
  - The target program is a PSM
  - $\Rightarrow$ model transformation
- Kermeta to weave a « compilation » aspect into the logo meta-model

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
- **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 2: Kermeta Emitter Template
- **NXC Code generation using a template**

### Execution
```
To a turtle:
    FOREWARD +10.0, reverseDir
    PFORWARD +10.0, reverseDir
    REVERSE +10.0, reverseDir
    REVERSE +10.0, reverseDir
```

### 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
  - Ker-meta for dynamic semantics
  - Java for simulation GUI
  - ...
- Keep separation between concerns
  - For maintainability and evolutions

Ker-meta in real projects

- Artist2, the European network of excellence on real-time embedded systems
- UsineLogicielle, a System@tic project where Ker-meta based operational semantic is associated to functional requirement for test synthesis purposes.
- Speeds, a European FP6 project for aspect-oriented metamodeling of avionics and automotive systems, including operational semantics aspects
- OpenEmbedd, a French project building a MDE platform for real-time systems.
- Mopcom, a French project applying MDE to hardware for generating SOC and introduce dynamic reconfigurability to them.
- Topcased, a consortium that aims to build modelers for avionics and system engineering
- DIVA, a European FP7 STREP project on handling Dynamic variability in complex, adaptive systems

From LOGO to Mindstorms

- Transformation written in Ker-meta
- Embedded source code inside the robot
- Result of a simulation interpreted with Ker-meta.

Conclusion and Wrap-up

- Ker-meta 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 as an effective editor for free
  - Weave in static semantics in OCL
  - Weave in an interpreter,
  - connect to simulation platform
  - Weave in one or more compilers
  - Finally care for concrete syntax issues
- Feel free to contribute!
  - www.ker-meta.org

Thank you!

- Questions?