Model-Driven Engineering
(or: Why I'd like write programs that write programs rather than write programs)

Jean-Marc Jézéquel

e-mail: jezequel@irisa.fr
Twitter @jmjezequel
Airbus

• Junior Model Based Systems Development Team Member
  • As part of the Model Based Systems Engineering Development Team "MBSD", you will support fulfilling the ADS global engineering model based development vision by participating to the extended organization, and contribute motivating project teams to achieve a high level of performance and quality in delivering model based development projects that provide exceptional business value to users. You will contribute to several concurrent high visibility development projects using advanced modeling methods in a fast-paced environment that may cross multiple business lines.

• Required skills
  • Undergraduate or graduate degree in a technical field, and first experience in MBSE field.
  • First experience with meta-modeling and model transformation between domains and/or other state of the art techniques.
  • First experience with systems engineering tools and representations (e.g., NoMagic, SysML, UML, or similar).
  • ...

Unity: Senior Modeling Language Engineer

Unity is the world’s leading platform for creating and operating real-time 3D (RT3D) content

• Role description
  • You are a language or tools developer with a passion for creating great user experiences...In this role you will use your expertise in building tools or domain specific modeling languages to shape the future of game design and development...

• Responsibilities
  • Design and build modeling languages and editors that empower game designers in ways never seen before
  • Work as a part of a cross-discipline team to build rapid prototypes that you can transform quickly into production-ready features

• Requirements
  • Strong understanding of data structures and algorithms
  • Fluent in C# or another statically typed language
  • Knows how to translate user needs into product features

• Bonus points
  • Expertise in developing domain-specific languages and editors before
  • ...

© J.-M. Jézéquel, 2012-2020
Software: Low code & No code approaches

• Shortage of programmers even for simple applications
  • Mobile phone apps, web, etc.
• Build programs that write programs
  • Put problem-solving capabilities into the hands of non-IT professionals
    • Users can more quickly and easily create business apps that help them do their jobs
• Industrial platforms
  • Google AppSheet, Mendix, Microsoft PowerApps, OutSystems, Robocoder Rintagi, Salesforce Lightning, Wix Editor X, etc.
  • Analysts at Gartner estimate that the low-code market grew 23% in 2020 to reach $11.3 billion, and will grow to $13.8 billion in 2021 and almost $30 billion by 2025

Example: Microsoft PowerApps 1/2

• Quickly create apps that work on any device using a Microsoft Office-like experience, templates to get started quickly and a visual designer to automate workflows.
• Use built-in connections, or ones built by your company, to connect PowerApps to cloud services
• Build additional data connections and APIs to any existing business systems, thus empowering any users in your organization to create the apps they need.
• Data security and privacy controls are respected by PowerApps, so you can manage data access and maintain corporate policies
  • Hum hum!
Example: Microsoft PowerApps 2/2

- Model-driven apps
  - start with your data model – building up from the shape of your core business data and processes in the Dataverse to model forms, views, and other components.
  - automatically generate great UI that’s responsive across devices.
- When you create a model-driven app, you can use all the power of the Dataverse to rapidly configure your forms, business rules, and process flows
- Dataverse is a data platform that allows you to store and model business data
  - securely store and manage data within a set of standard and custom tables, and you can add columns to those tables when you need them.

Goal of this module

- Learn the principles to build a use lowcode/nocode approach
  - Not to use lowcode/nocode!
- Use it when you identify a niche market where
  - the technical aspect of applications is always the same (or can be configured among a small set of options)
  - The functional aspect is simple enough that it can be described through a formalism close to natural language
    - Eg workflow
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

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.*
Modeling in Science & Engineering

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

$M_1$ (modeling space)

$M_0$ (the world)

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

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

Complex Software Intensive Systems

- Multiple concerns
- Multiple viewpoints & stakeholders
- Multiple domains of expertise
- => Need languages to express them!
  - In a meaningful way for experts
  - With tool support (analysis, code gen., V&V..)
    - Which is still costly to build
  - At some point, all these concerns must be integrated
Modeling Languages

• General Purpose Modeling Languages
  • UML and its profiles (MARTE for RT...)
• Domain Specific Modeling Languages
  • Airbus, automotive industry...
  • Matlab/Simulink
  • Lowcode/nocode
• General Purpose Programming Languages
  • With restrictions (not everything allowed)
    • GWT (Google Web Toolkit)
• Annotations, aspects...
• In any case, Need for Language Processors

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
Generalizations

Figure 3.33: Examples of generalizations between classes.

NB: Tell you nothing about:
• generalization being acyclic,
• or semantics of dynamic binding
Example with StateMachines

Model

- S1
- S2
- S3

a/b  x/y

S1 -> S2: b/a
S2 -> S3: y/x
S3 -> S1: x/y

Meta-Model

- FSM
  - run(): currentState
  - reset(): initialState
- State
  - name: EString
  - step(): currentState
- Transition
  - input: EString
  - output: EString
  - fire()

The 4 layers in practice

- M3 (UML)
- M2 (Meta)
- M1 (Models)
- M0 (Run-time instances)

Figure 1.8: An example of the four-layer metamodel hierarchy.
Comparing Abstract Syntax Systems

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

Technology #2  (MOF + OCL)

Technology #3  (XML Meta-Language)

Technology #4  (Ontology engineering)

\[ M^3 \]
- EBNF

\[ M^2 \]
- Pascal Language Grammar
- The UML meta-Model
- A specific execution of a Pascal program
- A specific phenomenon corresponding to a UML Model

\[ M^1 \]
- A specific Pascal Program
- A Specific UML Model
- A XML DTD or Schema
- A XML document

Upper Level Ontologies

KIF Theories

+Description Logics
+Conceptual Graphs
+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

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 Kermet
- 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

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.xtend"
// Transformations

class FSM { public def void reset() { currentState = initialState }

class Minimizer { public def FSM minimize (source: 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

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: find the program concepts

• Comment (?)
• ProcDeclaration, Parameter (formal)
• If
• ParameterCall, NotEqual, Constant (BinaryExp, Expression)
• Block
• Left (Primitive Instruction), Constant
• ProcedureCall, Expression, Expression
• ...

; 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
  ]
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
    • i.e. 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 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)

![Diagram of Person to Car association](image)

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:
  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
Outline

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

Operational Semantics of State Machines

• A model

• Its metamodel

• Adding Operational Semantics to OO Metamodels
Kermeta Action Language: XTEND

- **Xtend**
  - flexible and expressive dialect of Java
  - compiles into readable Java 5 compatible source code
  - can use any existing Java library seamlessly

- **Among features on top of Java:**
  - Extension methods
    - enhance closed types with new functionality
  - Lambda Expressions
    - concise syntax for anonymous function literals (like in OCL)
  - ActiveAnnotations
    - annotation processing on steroids
  - Properties
    - shorthands for accessing & defining getters and setter (like EMF)

---

Example with Xtend

```java
public def String fire()

_self.source.owningFSM.currentState = _self.target
return _self.output
```

---

© J.-M. Jézéquel, 2012-2020
def String step(String c) {
    // Get the valid transitions
    var validTransitions = _self.outgoingTransition.filter[t | t.input.equals(c)]

    // Check if there is one and only one valid transition
    if(validTransitions.empty) throw new NoTransition
    if(validTransitions.size > 1) throw new NonDeterminism

    // Fire the transition
    return validTransitions.get(0).fire()
}

def void run() {
    // reset if there is no current state
    if (_self.currentState == null) _self.currentState = _self.initialState
    var str = ""
    while (str != "quit") {
        println("Current state : " + _self.currentState.name)
        str = Console.instance.readLine("give me a letter : ")
        try {
            var textRes = _self.currentState.step(str)
            if (textRes == void || textRes == "") textRes = "NC"
            println("string produced : " + textRes)
        } catch (NonDeterminism err) {
            println(err.toString)
            str = "quit"
        } catch (NoTransition err) {
            println(err.toString)
            str = "quit"
        }
    }
}
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
Virtual Machine - Semantics

```kmt
require "VMLogo.ecore"
require "TurtleGUI.kmt"

aspect class Point {
  def String toString() {
    return "[" + x.toString() + "," + y.toString() + "]"
  }
}

aspect class Turtle {
  def void setPenUp(b : Boolean) {
    penUp = b
  }
  def void rotate(angle : Integer) {
    heading = (heading + angle).mod(360)
  }
}
```

Map Instructions to VM Actions

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

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

aspect class Clear {
  def int eval (ctx: Context) {
    ctx.getTurtle().reset()
  }
}
```
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

```java
aspect class Add {
  def int eval (ctx: Context) {
    return lhs.eval(ctx) + rhs.eval(ctx)
  }
}
```

```java
aspect class Add {
  def void 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

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

aspect class If {
  def int eval(context : Context) {
    if (condition.eval(context) != 0)
      return thenPart.eval(context)
    else return elsePart.eval(context)
  }
}

aspect class Right {
  def int eval(context : Context) {
    return context.turtle.rotate(angle.eval(context))
  }
}
```

Handling function calls

- Use a stack frame
  - Owned in the Context
- Bind formal parameters to actual
- Push stack frame
- Execute method body
- Pop stack frame
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
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 {
  def void 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
    - PIM to PSM a la OMG MDA
    - Refining models
    - Reverse engineering (code to models)
    - Generating new views
    - Applying design patterns
    - Refactoring models
    - Deriving products in a product line
    - ... any model engineering activity that can be automated...
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

Model to Text in practice

• For simple cases, use the template mecanism of Xtend
  • Output = ```template expression```
• Many template generators for MDE do exist
  • E.g. Acceleo (from Obeo) is quite popular in industry
    • a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard
    • http://www.eclipse.org/acceleo/
Example with Acceleo

- A template that prints the class name, its comments and attributes

```
<template name="MyClass">
  <class name="MyClass" extends="Object">
    <!-- Class comments -->
    <attribute name="myAttribute" type="int">
      <!-- Attribute comments -->
    </attribute>
    <super />
    <sub />
    <references />
  </class>
</template>
```

Model-to-Model: Typical Example

From UML to RDBMS

```
Class
  name : String
  type : Classifier

DataType

Attribute
  name : String
  multiValued : Boolean

Table
  col : int
  Column
```

Transformation

```
Name
  type : String
```

```
Type
  key : int
```
M2M: Reuse Engineering Know-How (Design/Test/…)

Design pattern application (parametric collaboration)

Element stereotype

…and also Tagged values & Contracts

The result we want: design patterns application

© J.-M. Jézéquel, 2012-2020
Classification of Model Transformation Tools

- Several approaches
  - Graph-transformation-based approaches
  - Relational approaches (aka Logic Programming)
  - Structure-Driven (OO) approaches
  - Hybrid approaches

- Rich ecosystem of tools, e.g.
  - ATL: a transformation language developed by Inria
  - GReAT: a transformation language available in the GATE
  - Henshin: a model transformation language for EMF, based on graph transformation concepts, providing state space exploration capabilities
  - Kermeta: a general purpose modeling and programming language, also able to perform transformations
  - Mia-TO: a transformation language developed by Mia-Software
  - QVT: the OMG has defined a standard for expressing M2M transformations, called MOF/QVT or in short QVT.
  - SiTrA: a pragmatic transformation approach based on using a standard programming language, e.g. Java, C#
  - Stratego/XT: a transformation language based on rewriting with programmable strategies
  - Tefkat: a transformation language and a model transformation engine
  - UML-RSDS [9]: a model transformation and MDD approach using UML and OCL
  - VIATRA: a framework for transformation-based verification and validation environment

Model to Models in Practice

- M2M Transformations as OO Programs
  - Kermeta/Xtend used to be a good choice
  - But now with modern Java, can be in plain Java using JMI

```java
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Attribute extends StructuralFeature {
    public boolean isDerived();
    public void setDerived(boolean newValue);
}
```

```java
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Operation extends BehavioralFeature {
    public boolean isQuery();
    public void setQuery(boolean newValue);
    public java.util.List getExceptions();
}
```
“Programming style” Issues

- The transformation is simply an object-oriented program that manipulates model elements
  - Uses the OO structure of the meta-model to cleanly modularize the transformation
- OO techniques
  - Customizability through inheritance/dyn. binding
  - Pervasive use of GoF like Design Patterns
Defining the metamodels

UML2RDBMS template method

• Create tables
  • Tables are created from classes marked as persistent in the input model

• Create columns
  • For each persistent class process all attributes and outgoing associations to create corresponding columns. The foreign keys are created but the cols property cannot be filled and the corresponding columns cannot be created because primary keys of references table cannot be known before it has been processed.

• Update foreign-keys
  • The foreign-key columns are created in the table that contains the foreign-key and the property cols of foreign-keys is updated.

=> Handle details/variability into subclasses
Writing the transformation

```java
package Class2RDBMS;

require kermeta // The kermerta standard library
require "trace.kmt" // The trace framework
require "../metamodels/ClassMM.ecore" // Input metamodel in ecore
require "../metamodels/RDBMSMM.kmt" // Output metamodel in kermeta

class Class2RDBMS {
    /** The trace of the transformation */
    reference class2table : Trace<Class, Table>

    /** Set of keys of the output model */
    reference fkeys : Collection<FKey>

    def RDBMSModel transform(inputModel : ClassModel) {
        // Initialize the trace
        class2table = new Trace<Class, Table>()
        fkeys = new Set<FKey>()
        result = new RDBMSModel()

        // Create tables
        getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
            var Table table = new Table()
            table.name = c.name
            class2table.storeTrace(c, table)
            result.table.add(table)
        }

        // Create columns
        getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
            createColumns(class2table.getTargetElem(c), c,"")
        }

        // Create foreign keys
        fkeys.each{ k | k.createFKeyColumns }
    }
}
```

© J.-M. Jézéquel, 2012-2020
Object-orientation

• Classes and relations, multiple inheritance, late binding, static typing, class genericity, exception, typed function objects

• OO techniques such as patterns, may be applied to model transformations
  • Template method as above
  • Command, undo-redo
    • Refactorings example

    \[
    \text{abstract class RefactoringCommand} \\
    \{ \\
    \text{operation check() : Boolean is abstract} \\
    \text{operation transform() : Void is abstract} \\
    \text{operation revert() : Void is abstract} \\
    \}
    \]

Software Engineering Concerns

• Modularity in the small and the large
  • classes & packages

• Reliability
  • static typing, typed function objects and exception handling

• Extensibility and reuse
  • inheritance, late binding and genericity

• V & V
  • test cases
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

```java
// aspect local-to-global
aspect class Statement {
    def void local2global(rootBlock: Block) {
    }
}
aspect class ProcDeclaration {
    def void local2global(rootBlock: Block) {
        ...
    }
}
aspect class Block {
    def void local2global(rootBlock: Block) {
        ...
    }
}
...
```

Step 2: Model to text

- NXC Code generation using a template
  - Left as an exercise
Execution

```
TO k :scale
  PENDOWN
  FORWARD *(30, :scale)
  PENUP
  BACK *(10, :scale)
  RIGHT 45
  FORWARD *(14, :scale)
  PENDOWN
  PENUP
  BACK *(14, :scale)
  RIGHT 90
  FORWARD *(14, :scale)
  PENDOWN
  PENUP
  BACK *(14, :scale)
  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 (1)

- ASMLogo.ecore
- TurtleGUI.java
- VMLogo.ecore
- TurtleGUI.kmt
- TurtleController.kmt
- LogoVMSemantics.kmt
- LogoDynSemantics.kmt
- LogoSimulator.kmt
- logo.sts
- NXC.ket
- Logo4NXC.kmt
- LogoNXCompiler.kmt
- Editor.java
- LogoStaticSem.ocl

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

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

To learn more...

Jean-Marc Jézéquel
Benoit Combermate
Didier Vojtsek

INGÉNIEURIE DIRIGÉE par les MODÈLES
Des concepts à la pratique

© J.-M. Jézéquel, 2012-2020