Model driven Engineering for (software) languages

PROF. PATRIZIA SCANDURRA, UNIBG

Se siete interessati per tirocini/tesi contattatemi:
patrizia.scandurra@unibg.it
http://cs.unibg.it/scandurra/index.html
http://cs.unibg.it/scandurra/tesi.html

15 Gennaio 2010 – Seminario per il corso di Linguaggi e Compilatori

Agenda

- Part 1: MDE overview
  - Core principles
- Part 2: Language engineering by MDE
  - Languages as models
  - The FSM case study
- Part 3: Conclusions
  - MDE applicability
Model Driven Engineering (MDE)

- Emerging **paradigm** for software/system development and analysis where **models** play the role of first-class artifacts
  - Beyond their use as documentation
  - Models can be used to **generate** software artifacts

- All Software Engineering **generative techniques** based on the notions of
  - **models**, **metamodeling**, and **model transformation**
MDE and abstraction in (software) languages

"MDE is a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively."

[Douglas C. Schmidt, IEEE Computer February 2006]

C, C++, C#, Java, …

Modeling languages (domain-specific)

MDE - standards & tools

Three ideas:
- direct representation,
- automation, and
- standards (languages and mappings)
MDE principles: first principle

**Unification principle:** “Everything is a model”

What kind of models?

**Informal**
- Descriptions in natural language or free diagrams
- Poor semantics, *non machine-comprehensible*

**Semi-formal (or lightweight)**
- Expressive and appealing notation, human and machine-comprehensible
- E.g. UML models

**Formal**
- Application of logic and mathematics to computing
- Formal semantics useful for model analysis (simulation, theorem proving, model checking, etc.)
- Machine comprehensible, less human comprehensible
- E.g. Petri nets, Finite State Machines, Abstract State Machine, Process Algebra, etc.
MDE principles: systems, models and metamodels

- **Working definition of “model”**
  
  _Def.:_ “A **model** is a graph-based structure that **represents** some aspects of a given **system** and **conforms** to the definition of another graph called **metamodel**.”

  ![Diagram](image)

  - **Basic concepts:** **system**, **model**, and **metamodel**
  - and two **basic relations**: **representation** and **conformance**

MDE principles: Metamodelling

- **Def.:** “A model conforms to a **metamodel** iff each model element has its **metaelement** within the metamodel.”
  - **Metaelements** are a typing scheme for model elements
  - A **metamodel** typically defines the **language** and processes from which to form a model

  ![Diagram](image)
MDE principles: Metamodelling

- **Def.:** “A metamodel conforms to a meta-metamodel iff each of its elements has its metaelement defined within the meta-metamodel or *metalanguage*”
The three-level model organization stack in various technical spaces

Model transformation and Technical Spaces

- **Technical Spaces** are similarly organized around a set of concepts
- Spaces may be connected via *transformation bridges* (hot research topics)
Common Transformation Pattern

MDE Transformation Pattern

Corollary: "A model transformation is a model"
The OMG’s Metamodeling Framework

- Classes with attributes and operations, possibly inherited from other classes by Generalization
- Associations (simple, composite, shared aggregation) between classes, with cardinality and uniqueness
- Packages to group elements for modularizing
- Data types whose values do not have object identity
  - primitive types: Boolean, Integer, and String
  - data type constructors: Enumeration, Collection, etc.
- Constraints (well-formedness rules) in the Object Constraint Language
- APIs for model manipulation/implementation
  - Java Interfaces
  - CMI (CORBA Metadata Interface), etc.
- Model serialization
  - XMI (XML Metadata Interchange)
  - HUTN (Human Usable Textual Notation), etc.

The MOF meta-language
The MOF meta-language: a metamodel example

The “automaton” metamodel

Abstract Syntax (AS) expressed by a metamodel

“metalevel” boundary

Transformation Languages

**OMG approach:**
- Domain-specific transformation Language
  - Declarative, imperative or hybrid
- MOF 2.0 **Query/Views/Transformation (QVT) standard**
  - Not only transformation, but also for model query and view
- Example: **UML-to-Java transformation**
  - Example is taken from DSTC/IBM/CBOP QVT Submission
  - A declarative proposal
Source Meta-model

Simplified UML meta-model

Target Meta-model

Simplified Java meta-model
Transformation Rules

Transformation declaration and a transformation rule:

**TRANSFORMATION** uml2java(SOURCE UML, TARGET Java)

**TRACKING** TModel;

**RULE** umlClassifierToJavaClass(X, Y)

FORALL UMLClassifier X

WHERE X.name = N

MAKE JavaClass Y,

Y.name = N

**LINKING** X, Y BY JavaClassFromUMLClassifier;

...

---

Part 2

**MDE FOR LANGUAGE ENGINEERING:**

- Languages as models
- The Finite State Machine (FSM) case study
Language engineering

- "traditionally": by grammars and AST
- "new" MDE approach: by models
  - graphs for definition of concepts
  - differentiation of abstraction levels: model and metamodel
  - assignment of notation elements (textual, visual or mixed)

Model-driven language definition

A language $L$ is a five-tuple: $L = < A, C, S, M_C, M_S >$
- $A$: abstract syntax
- $C$: concrete syntax
- $S$: semantic domain (usually, some formal math. framework)
- $M_C$: $C \rightarrow A$ syntactic mapping
- $M_S$: $A \rightarrow S$ semantic mapping

In MDE:
- $A$ is defined by a metamodel (usually a class diagram plus OCL-like structural constraints) with the advantage of deriving from $A$ different alternative concrete notations $C_i$ (textual, graphical, mixed) for various scopes: graphical rendering, model interchange, standard encoding, etc..
- still maintaining the same semantics $M_S$
A model-driven language engineering process

1. Requirement analysis
2. Metamodelling
3. Choice MM framework
4. Concrete syntaxes
5. Validation
6. Semantics
7. Validation
8. Tools

Step 1. Language requirements for FSM

1. Many mathematical definitions of FSM exist. We choose the representation as the tuple

\[(\Sigma, \Gamma, S, S_0, \tau)\]

- \(\Sigma\) is the input alphabet (a finite, non-empty set of symbols)
- \(\Gamma\) is the output alphabet (a finite, non-empty set of symbols)
- \(S\) is a finite, non-empty set of states
- \(S_0 \subseteq S\) initial states
- \(\tau \subseteq S \times \Sigma \times \Gamma \times S\) is the transition relation such that \((s_j, i, o, s_k) \in \tau\)
Step 2. Choice of a metamodeling framework for FSM

EMF framework (with Ecore the metalanguage as MOF)

Step 3. The FSM metamodel (abstract syntax)
Step 3. The FSM metamodel (abstract syntax)

Example of OCL constraint for *well-formedness of models*:

```ocl
class FSM

context FSM

-- The following invariant checks that if there are no states, there are no transitions
inv I0: self.states->isEmpty() implies self.transitions->isEmpty()
```

Step 4. Concrete syntaxes (or notations) for FSM

- They can be: textual, graphical or both
- Moreover they can be:
  - Human-comprehensible for human use to edit models conforming to the metamodel, and as
  - Machine-comprehensible for model handling by software applications
    - Examples: an XMI (XML Metadata Interchange) format and Java APIs
Step 4. Human-comprehensible notations for FSM

```
Step 4. Machine-comprehensible notations for FSM

XMI format
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fsm:FSM xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
 xmlns:fsm="http://www.openarchitectureware.org/xtext/dsl/fsm"
 name="evenFsm" outputAlphabet="eo" inputAlphabet="01">
  <states name="even" isStart="true"/>
  <states name="odd" isStart="true"/>
  <transitions name="t1" input="0" to="/states.1" from="/states.0" output="e"/>
  <transitions name="t2" input="1" to="/states.0" from="/states.0" output="e"/>
  ...
</fsm:FSM>
```
Step 4. Machine-comprehensible notations for FSM

Java API

```java
public interface State extends NamedElement {

    /**
     * @model
     * @generated
     */
    boolean isIsStart();

    /**
     * @param value the new value of the '<em>Is Start</em>' attribute.
     * @see #isIsStart()
     * @generated
     */
    void setIsStart(boolean value);
}
```

Step 4. A human-comprehensible textual notation for FSM

```
fsm evenFsm
inputAlphabet "01" // "e" for even and "0" for odd
outputAlphabet "eo"
states
start even odd
transitions
t1: even - 0 / o -> odd ;
t2: even - 1 / e -> even ;
t3: odd - 0 / e -> even ;
t4: odd - 1 / o -> odd ;
```

FSM EBNF grammar

FSM metamodel

bridge?
Step 4. A human-comprehensible textual notation for FSM

- Many tools are able to derive concrete grammars (by generating ANTLR or JavaCC files, for example) from metamodels
  - EMFText, TCS, TEF (Textual Editing Framework), etc...

- Other tools also feature the derivation of the language metamodel from its concrete grammar
  - like the Xtext by openArchitectureWare

Part 3

CONCLUSIONS
MDE technologies

- **To be used as DSL development toolkit**
  - For developing a family of products or set of tools in a specific domain of interest
  - SW to develop other SW in a generative manner
    - By metamodeling, model editing and model transformations

- **Examples:**
  - etc..

Eclipse-based MDE tools

**On (meta)modeling**

- **Abstract Syntax Development**
  - Eclipse Modeling Framework (EMF): a modeling framework and code generation facility for building tools and other applications based on a structured data model in Ecore

- **Concrete Syntax Development**
  - Graphical Modeling Framework (GMF): a generative component and runtime infrastructure for developing graphical editors based on EMF/GEF
  - Textual Modeling Framework (TMF): tools and frameworks for developing textual syntaxes and corresponding editors based on EMF
  - Xtext: a powerful framework/tool for developing external textual DSLs based on EMF
  - TCS: specification of textual concrete syntaxes by attaching syntactic information to metamodels
Eclipse-based MDE tools
http://www.eclipse.org/modeling/

On model transformations

- **Model-to-Model (M2M)** transformation project

- **Model-to-Text (M2T)** transformation project
  - Java Emitter Templates (JET): code generation framework & facilities that are used by EMF, very similar to Java Server Pages (JSPs).
  - Xpand: a statically-typed template language featuring polymorphic template invocation, aspect oriented programming, etc.

Model-driven toolkit in Open-source for Critical Applications & Systems Development

http://www.topcased.org
A model-driven SoC design toolkit: a project @STMicroelectronics

- A model-driven methodology for SoC design based on
  - A UML profile for SystemC: Structural and behavioral/timing features
  - Rational Unified Process for SoC: UPES/UPSoC with model transformations
- (In-house) HW-SW co-design tool

![Diagram of the model-driven SoC design toolkit]

MDE for Abstract State Machines

**ASMETA tool set**

http://asmeta.sourceforge.net/

- From the ASM metamodel (AsmM) we derived **generated artifacts**, developed new **based design/analysis tools**, and **integrated** existing ones
Model driven Engineering for (software) languages

PROF. PATRIZIA SCANDURRA, UNIBG

Se siete interessati per tirocini/tesi contattatemi:
patrizia.scandurra@unibg.it
http://cs.unibg.it/scandurra/index.html
http://cs.unibg.it/scandurra/tesi.html