2IS55 Software Evolution

Implementing evolution: Model-Driven Engineering

Marcel van Amstel

Goal:
- Raising the level of abstraction
  ... from the computing domain to the problem domain

MDE combines:
- Domain-specific modeling languages
- Model transformations

Domain-Specific Languages

Examples
- HTML
- SQL
- MediaWiki
- (La)TeX
- PROMELA (SPIN)
- YACC
- ATL

Domain-Specific Languages

“*a language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain*” [2]

**Domain-Specific Languages**

**Pros**
- Expression of the solution in terms of domain concepts
- Enhanced productivity, reliability, maintainability, and portability
- Domain knowledge contained in language
- Mostly concise and largely self-documenting

**Cons**
- Cost of DSL implementation and education
- Difficulty of finding the right scope
- Difficulty of balancing between domain-specificity and general-purpose constructs
- Potentially less efficient code

---

**Domain-Specific Languages**

"A problem domain is defined by consensus, and its essence is the shared understanding of some community" [3]

**Domain Analysis:**
1. Identify domain concepts
2. Define semantic notions
3. Define syntactic carrier


---

**Domain-Specific Languages**

**Language implementation**

**Implementation Strategies**
- Stand-alone
- Embedded
- Translation

---

**Domain-Specific Languages**

**Model Transformation**

Problem domain → Domain-specific models → Model transformation → Implementation platform → Solution domain
Model Transformation

Approaches
- Direct model manipulation
- Intermediate representation
- Transformation engine

Types of model transformations
- Text to Model
- Model to Model
- Model to Text
- Text to Text

Model Transformation formalisms
- ATL
- Xtend
- Xtext
- Xpand
- QVT Relations
- QVT Operations
- QVT Core
- ASF+SDF
- Stratego/XT
- VIATRA
- Tefkat
- ETL (Epsilon)
- GrGen
- ...

Taxonomy
- Horizontal
- Vertical

Further distinction: syntactic vs. semantic transformation

Meta-metamodel

Approach

ATL

Ecore
Assigning attributes

``` ATL
rule example1{
  from in: MM1!MetaClassA
to out: MM2!MetaClass1{
    attr <- in.attr
  }
}
```

Assigning references

Three cases:

- Model target element generated by current rule
- Default target model element generated by another rule
- Non-default target model element generated by another rule

``` ATL
rule example2{
  from in: MM1!MetaClassB
to out: MM2!MetaClass1{
    attr <- in.attr
  }
}
```

Model target element generated by current rule

``` ATL
rule example{
  from in: MM1!MetaClass
  to out1: MM2!MetaClass1{
    RefToMetaClass2 <- out2
  },
  out2: MM2!MetaClass2{}
}
```

Default target model element generated by another rule

``` ATL
rule example1{
  from in: MM1!MetaClassA
to out: MM2!MetaClass1{
    RefToMetaClass2 <- out2
  }
}
```

Non-default target model element generated by another rule

``` ATL
rule example1{
  from in: MM1!MetaClassA
to out: MM2!MetaClass1{
    RefToMetaClassB <- in.RefToMetaClassB
  }
}
```

Assigning enumerations

- Types
  - EnumArray: [0, 1]

``` ATL
rule example{
  from in: MM1!MetaClassB
to out: MM2!MetaClass1{
  enum <- #Integer
}
```

- Quotes are only needed in case of reserved words
ATL

Rules

- Matched rules
- Lazy matched rules
- Unique lazy matched rules
- Called rules

ATL

Lazy matched rules

lazy rule rule_name{
from is: MM1!MetaClass
using {
<variable definitions>
}

to out1: MM2!MetaClass1(
<bindings1>
),
out2: MM2!MetaClass2(
<bindings2>
)
do {
<imperative block>
}
}

- Generates new target elements for every call to the rule
- Invoked from other rules as follows:
  thisModule.rule_name(<model element of type MM1!MetaClass>)

ATL

Unique lazy matched rules

unique lazy rule rule_name{
from is: MM1!MetaClass
using {
<variable definitions>
}

to out1: MM2!MetaClass1(
<bindings1>
),
out2: MM2!MetaClass2(
<bindings2>
)
do {
<imperative block>
}
}

- Always returns the same target elements for a given source element, i.e., target elements are generated only once per source element

ATL

Called rules

rule rule_name(<parameters>){

using {
<variable definitions>
}

to out1: MM2!MetaClass1(
<bindings1>
),
out2: MM2!MetaClass2(
<bindings2>
)
do {
<imperative block>
}

- For generating target elements from imperative code

ATL

Helpers

- Helper with context
  helper context MM!MetaClass def: helper_name(<parameters>) return_type =
  let <variable definitions>
  in <expression>
  - Invocation:
    model element of type MM!MetaClass.helper_name(<parameters>)
  - The context should never be of a collection type

- Helper without context
  helper def: helper_name(<parameters>) return_type =
  let <variable definitions>
  in <expression>
  - Invocation:
    thisModule.helper_name(<parameters>)

- For OCL functions refer to the ATL user guide
Model Transformation Quality

- Model driven engineering is becoming increasingly important
- Model transformations are similar to traditional software artifacts
  - Used by multiple developers
  - Changed according to changing requirements
  - Reused if possible

Model Transformations become the next maintenance nightmare

Make the quality of model transformations measurable

M → M'

M.F. van Amstel: The Right Tool for the Right Job: Assessing Model Transformation Quality
To appear in Proceedings of the Fourth IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'10)

Model Transformation Quality

Quality:
- Understandability
- Modifiability
- Reusability
- Modularity
- Completeness
- Consistency

North-Holland, 1978

MDE Case Study

Goal:
- Simulation, verification and implementation of the same model

Concurrent objects
- Controllers
- Hardware
  - Conveyors
  - Motors
  - Sensors
- Communication
  - Wireless
  - Wired

Simulation models

DSL models

Model checking models

Code

MT

MT
### MDE Case Study

#### Platforms:
- Simulation
- POOSL
- Execution
  - NQC
- Verification
  - PROMELA/SPIN

<table>
<thead>
<tr>
<th>Platforms</th>
<th>Communication Primitives</th>
<th>#Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSL</td>
<td>Synchronous</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Asynchronous</td>
<td>Both</td>
</tr>
<tr>
<td></td>
<td>Lossless</td>
<td>Unlimited</td>
</tr>
<tr>
<td></td>
<td>Lossy</td>
<td>Limited</td>
</tr>
</tbody>
</table>

### MDE Case Study

#### Model Transformations

- DSL (Asynchronous)
- DSL (Lossless Communication)
- DSL (Limited number of objects)