INF 5120 - Lecture #5
Code generation with MOFScript and other technologies

Garan K. Olsen
SINTEF ICT
Cooperative and Trusted Systems

Context of Work

The present courseware has been elaborated in the context of the MODELWARE, MODELPLEX, ATHENA and INTEROP European IST FP6 projects
(http://www.modelware-ist.org/)
(http://www.modelplex-ist.org/)
(http://www.ist-athena.org/)
(http://interop-noe.org/)
**Introduction**

ATHENA Model-Driven Interoperability (MDI) Framework

- MDA & Interoperability
- Metamodelling
- UML Profiles & DSLs
- Model Transformations
- Method Engineering

Reusuable MDI Assets
- Method chunks
- Tools and services
- Models and metamodels
- Model transformations
- DSLs and UML profiles
- Reference examples

---

**Model to Text transformation 1/2**

- MDA places modelling at the heart of the software development process.
- Various models are used to capture various aspects of the system in a platform independent manner.
- Sets of transformations are then applied to these platform independent models (PIM) to derive platform specific models (PSM).
- These PSMs need to be eventually transformed into software artefacts such as code, deployment specifications, reports, documents, etc.
- It is also common to generate code directly from PIM-like models. (DSL approach)
Model to Text transformation 1/2

- QVT standard addresses the needs of model – to – model transformation (e.g., PIM – to – PIM, PIM – to – PSM and PSM – to – PSM).
- The MOF Model to Text (mof2text) standard addresses how to translate a model to various text artefacts such as code, deployment specifications, reports, documents, etc.
- Essentially, the mof2text standard needs to address how to transform a model into a linearized text representation.
- An intuitive way to address this requirement is a template based approach wherein the text to be generated from models is specified as a set of text templates that are parameterized with model elements.

Motivation

- Why do we need model-to-text transformation?
  - Raise the level of abstraction
    - Systems are getting more complex
    - Raise of abstraction has proven useful (for instance: Assembly to COBOL)
  - Automation of the software development process
    - Decrease development time
    - Increase software quality
    - Focus on the creative part
  - Automatic generation of new artefacts from your models
    - Java, EJB, JSP, C#
    - SQL Scripts
    - HTML
    - Test cases
    - Model documentation
Alternatives

- What are the alternatives?
  - Programming languages (e.g. Java),
  - Template/scripting languages (e.g. XSLT, Velocity Template Language, Eclipse Java Emitter Templates - JET),
  - Model Transformation Languages (e.g. ATLAS Transformation Language (ATL)), proprietary UML-based script languages, DSL-based approaches, Other MOF-based text/code generators

- Properties of the alternatives:
  - Neither programming languages nor scripting languages tend to take advantage of source metamodels.
    - However, it can be done programmatically in Java (e.g. using Eclipse Modeling Framework (EMF))
  - Model 2 Model Transformation languages such as ATL is metamodel-based, but is not designed with text generation in mind. However, it can be done also in ATL
  - UML tool script languages are tied to both UML and a vendor, and are not based on standards.
  - DSLs provides the flexibility of metamodel-based tools; they typically hard code code generation for each domain-specific language.
    - The difference between a MOF-based approach and a DSL is not significant, as transformations in MOF-based approaches also will depend on a particular metamodel.
  - Other MOF-based text generators have not been available, but will emerge.

OMG Request for Proposal for a model-to-text transformation language

- OMG RFP Issued in 2004
- Mandatory Requirements:
  - Generation from MOF 2.0 models to text
  - Reuse (if applicable) existing OMG specifications, in particular QVT
  - Transformations should be defined at the metalevel of the source model
  - Support for string conversion of model data
  - String manipulation
  - Combination of model data with hard coded output text
  - Support for complex transformations
  - Multiple MOF models as input (multiple source models)
- Optional Requirements
  - round-trip engineering
  - detection/protection of hand-made changes for re-generation
  - traceability is a (possible) means of supporting the last two.
OMG
MOF Models to Text Transformation Language
*Final Adopted Specification*

**Submitters / supporters**
- Compuware Corporation
- Interactive Objects Software
- Mentor Graphics Corporation
- Object Management Group
- Pathfinder Solutions
- SINTEF
- Softeam
- Tata Consultancy Services
MOF to Text Overview

- A template-based approach where a Template specifies a text template with placeholders for data to be extracted from models.
- These placeholders are essentially expressions specified over metamodel entities with queries being the primary mechanisms for selecting and extracting the values from models.
- These values are then converted into text fragments using an expression language augmented with a string manipulation library.
- Template can be composed to address complex transformation requirements. Large transformations can be structured into modules having public and private parts.

MOF to Text Example

```java
[template public classToJava(c : Class)]
class [c.name] () {
    // Attribute declarations
    [attributeToJava(c.attribute)]
    // Constructor
    [c.name]() {
        
    }
}
[/template]

[template public attributeToJava(a : Attribute)]
[a.type.name] [a.name];
[/template]

class Employee {
    // Attribute declarations
    String name;
    Department dept;
    Double salary;
    // Constructor
    Employee() {
    }
}
```
Language details 1/3

- Instead of defining two templates separately, a template can iterate over a collection by using the `for` block.
- Using the `for` block preserves WYSIWYG-ness and improves readability.
- For example the `classToJava` template can use the `for` block as shown below:

```template
classToJava(c : Class)
```class [c.name] {  
  // Attribute declarations
  [for(a : Attribute | c.attribute)]
  [a.type.name] [a.name];
  [/for]
  // Constructor
  [c.name] () {
  }
}
[/template]
```

The `for` block declares a loop variable `a` of type Attribute and produces for each Attribute in the collection `c.attribute` the text between the `for` and `[/for]`.

Language details 2/3

- A template can have a guard that decides whether the template can be invoked. For example, the following `classToJava` template is invoked only if the class is concrete.

```template
classToJava(c : Class) ? (c.isAbstract = false) 
```

- Complex model navigations can be specified using *queries*. The following example shows use of a very simple query check if the visibility is public.

```query
isPublic(c: Class) : Boolean = c.visibility == 'public'
```

- More advanced:
  - ```query
    allOperations(c: Class) : Set ( Operation ) = c.operation->union(
      c.superClass->select(scjc.jc.isAbstract=true)- >iterate(ac : Class; os:Set(Operation) =
      Set{}| os->union(allOperations(ac))))
    ```

  Shows use of a query `allOperations` to collect operations of all abstract parent classes of a class in a class hierarchy.
As we have seen, a template has WYSIWYG nature with the text to be output being specified in exactly the way it should look in the output.

There may be cases where the quantity of the text producing logic far outweighs the text being produced. In this case, it is more intuitive to specify the text producing logic without use of special delimiters. `@code-explicit`

```
@code-explicit
template public classToJava(c : Class)
'class 'c.name ' {
    // Constructor
    'c.name' () {
    }
}'
[/template]
```

```
@text-explicit
[template public classToJava(c : Class)]
class [c.name/] {
    // Constructor
    [c.name/] () {
    }
}[/template]
```

**OMG Standard for model-to-text**

- MOF2Text: A merge of the different model to text proposals, where MOFScript was one of several
- The only candidate left in the OMG standardization process (model-to-text)
- Many **similarities** with MOFScript:
  - imperative language w/ explicit rule calls
  - reusing selected parts of QVT/OCL
- **Differs** from MOFScript:
  - Mainly syntactical
  - Context type does not have its own slot, inserted in the parameter list
  - More traditional for-statements instead of forEach
  - Escaping direction is flexible: The transformation code can be escaped, or the output text can be escaped (as in MOFScript).
MOFScript Model to Text language and Tool

Introduction

- Model Driven Development (MDD) emphasizes the use of models as first class artifacts
- CIM <-- PIM <-- PSM <-- TEXT / CODE
- MOFScript bridges Model → Text

Public class a extends x

...............B.wsd1
...............Z association type Simple
...............c.Html
MOFScript placed in the 4-layer architecture

What is MOFScript?

- The MOFScript tool is an implementation of the MOFScript model to text transformation language
- Developed at Sintef ICT in the EU-supported MODELWARE project
- An Eclipse plug-in
  - Part of the GMT subproject [http://www.eclipse.org.gmt](http://www.eclipse.org.gmt)
  - Download from [http://www.modelbased.net/mofscript/update](http://www.modelbased.net/mofscript/update)
- Mapping of model artifacts to a multitude of textual languages
- Was part of standardization process within OMG
  - OMG RFP MOF Model to Text Transformation process
MOFScript a transformation language

- Language for writing model to text transformations
- Rules / Operations are called explicit (Procedural language)
- Partly based on the current QVT specification (keeps it within the family)
- Transforms input models to output text files
  - Generate text from any MOF-based model, e.g., UML models or any kind of domain model.

MOF Script - background

- Usability
  - Ease of use: Writing and understanding
  - Few constructs
- End user recognizability
  - Similar to programming and scripting languages
  - Imperatively oriented
- Sequential execution semantics
  - Rules are called explicitly
    - Might also support pattern matching execution
  - Contents of rules is executed sequentially
- Compatibility
  - Alignment with latest QVT (QVTMerge) specification
MOFScript in action

MOF MODELS
- UML
- RDBMS
- BPMN

MOFScript

LEXICAL OUTPUT
- Program code
- XML
- Documentation

MOFScript architecture
MOFScript a model to text tool

- Provides the means of:
  - Editing, compiling and executing
  - Syntax high-lightning
  - Content assist
  - Outline
  - MOFScript Console

The main steps of using the MOFScript tool

- Task: Define a transformation from source model A to text t. (A→t)
  1. Import or define the source metamodel for A.
  2. Write the MOFScript to transform A to t in the MOFScript editor
  3. Compile the transformation. Any errors in the transformation will be presented.
     1. Fix errors, if any
  4. Load a source model corresponding to A’s metamodel.
     1. Using the Eclipse plugin, this is prompted by the tool when trying to execute.
  5. Execute the MOFScript in the MOFScript tool.
     1. The transformation is executed. Output text / files are produced.
Built-in operations

- String operations
  - Various string manipulation operations, such as:
    - size, substring, subStringBeforeAfter, toLower, toUpper, index, trim, normalizeSpace, endsWith, startsWith, replace, equals, equalsIgnoreCase, charAt, isLowerCase, isUpperCase
- Collection library
  - Standard collection operations...
    - HashMap: put, get, clear, size, keys, values, isEmpty, forEach
    - List: add, size, clear, isEmpty, first, last, forEach
    - Model: size, first, isEmpty, forEach
- System and utility operations
  - Various utility functions, such as
    - time, date, getenv, setenv, position, count
- UML2 Operations
  - Operations available when UML2 models are loaded
    - hasStereotype, getAppliedStereotypes, getAppliedStereotype

Textual syntax

- Textmodule
  - textmodule UML2Java {in myMod:uml2}

- Rules
  - myMod.Package::mapPackage () {
    'package ' self.name ';' 
  }

- Files
  - file f2 (c.name + ".java")
    'package ' c.ownerPackage.name';'
  f2.println ("public class " + c.name);

- Escaped output
  - 'public class ' c.name
    ' extends Serializable ;'
Textual syntax

- Entry point rule
  ```
  myMod.Model::main () {
    // code for entry point
  }
  ```

- Iterators
  ```
  self.ownedMember->forEach(c:myMod.Class) {
    '<class name="' + c.name + '/' />
  }
  ```

- Conditional statements
  ```
  if (self.hasStereotype("Feature") { 
    'This is a feature type'
  } else if (self.hasStereotype("Product")) { 
    'This is a product type'
  } else {
    'this is neither'
  }
  ```

Collections

```javascript
var packageNames_List:List
var packageName_Hashtable:Hashtable

self.ownedMember->forEach(p:uml.Package) {
  packageNames_List.add (p.name)
  packageName_Hashtable.put (p.id, p.name)
}

if (packageName_Hashtable.size () > 0) {
  'Listing the package names that does not start with 'S'
  packageName_Hashtable->forEach (s:String | not(s.startsWith("S"))) {
    'Package: ' + s
  }
}
```
Textual syntax

```
uml.Package::interfacePackages () {
    if (self.getStereotype() = "Service"){
        file (self.name.toLower() + ".wsdl")
        self.wsdlReader()
        self.wsdlTypes()
        self.ownedMember->forEach(i:uml.Interface)
        {
            i.wsdlMessages()
            i.wsdlPortType()
            i.wsdlBindings()
            i.wsdlService()
        }
        self.wsdlFooter()
    }
}
```

```
uml.Package::getPackageNameToLower(): String {
    result = self.name.toLower()
}
```

Invoking rules

Return results

See documentation in the User Manual or Help menu in Eclipse

---

Uml2Java Example

```java
//Context class
self.ownedAttribute->forEach(p : uml.Property | p.association = null) {
    p.attributeGetterSetters()
}

// Generate Getter and Setters
uml.Property::attributeGettersSetters () {
    'public ' self.type.name ' get' self.name.firstToUpper() ' () '{
        'return ' self.name ';
    ' }
    'public void set' self.name.firstToUpper() '(' self.type.name ' input ) '{
        self .name ' = input; 
    ' }
}
```

```
public String getBookTitle(){
    return bookTitle;
}

public void setBookTitle(String input){
    bookTitle = input;
}

public Integer getNumberOfPages(){
    return numberOfPages
}

public void setNumberOfPages(Integer input){
    numberOfPages = input;
}
```
FamilyModel example

```java
uml.Class::outputGeneralization()
self.generalization->forEach(g: uml Generalization)
if(!(g.target.isEmpty()))
g.target->forEach(c: uml.Class)
    stdout.println("Generalization target name: "+c.name)
} //g.target forEach
if(!(g.source.isEmpty()))
g.source->forEach(c: uml.Class)
    stdout.println("Generalization source name: "+c.name)
} //g.sourse forEach
} //self.generalization
} //outputGeneralization()
```

Generalization target name: Individual
Generalization source name: Male

MOFScript advanced features
Transformation inheritance

textransformation TestInheritanceSub (in ecmodel.ecore) extends TestInheritanceSuper {
    ecmodele.Package::main() {
        self.printMe()
    }
    ecmodele.Package::printMe() {
        stdout.println("TestInheritanceSub::printMe begin")
        super.printMe();
        stdout.println("TestInheritanceSub::printMe end")
    }
}

Black-Box (Integration with Java)

- A MOFScript transformation can execute externally defined Java class methods.
- Static methods or Java classes with a default constructor

List parameterList;
// Insert parameters to this list
...
java ("org.test.SomeClass", "calculate", parameterList, "c:/Working/TestJava")

uml.Class::classToJava(String classPrefix) when {} {
    java ("org.test.SomeClass", "printClass", self, "c:/Working/TestJava")
}
Traceability

- One of the main challenges in MDD is the management of relations between different artefacts produced in the development process.
- As the systems become more complex, the number of artefacts is increasing. Furthermore, they are often generated.
- Therefore, trace links are needed to fully understand the many dependencies that exist between the different artefacts.

```java
public String getBookTitle() {
    return bookTitle;
}

public void setBookTitle(String input) {
    bookTitle = input;
}

public Integer getNumberOfPages() {
    return numberOfPages;
}

public void setNumberOfPages(Integer input) {
    numberOfPages = input;
}
```

Traceability in MOFScript

- The transformation implementation contains references to model elements that should be substituted in the generated text.
- The references to model elements are the basis of MOFScript traceability.
- All references to model elements that are used to generate a text file will be traced to the text file(s) they occur in.
- The granularity is from model element to line and column in the text file.
Change management

- Proposed metamodel for handling:
  - Traces from a model to text blocks
- Controlled by user
  
  ```
  { unprotect {
    ' // User writes code here ' 
  }
  }
  ```

- The resulting code will be generated with protected blocks
  - Identifying the start and end of the section
  - Code is protected by user-defined tags
  - E.g. comment tags // (Java)

Traceability model
Unprotected Block Example

```java
self.ownedOperation->forEach(o:uml.Operation){
  'in 'o.visibility' void 'o.name'(){
      unprotect{
        '  //User code here for operation'
      }
  }
}
```

```java
public void printAuthor(){
  //User code here for operation
}
```

Traceability in MOF Models to Text

- A Trace block relates text that is produced in a block to a set of model elements that are provided as parameters.
- Mof2Text provides support for tracing model elements to text parts.

```
[template public classToJava(c : Class)]
[trace(c.id())+ "_definition"]
class [c.name][]{
  // Constructor
  [protected("user_code")]
  // user code
  [protected]
}
[/trace]
[/template]
```

- In the example above, the trace block identifies the text to be traced by relating the generated text to model element `c` of type Class.
- The protected block identifies the text part that needs to be preserved between subsequent model-to-text transformations.
Adding your own metamodel to MOFScript

- Create the metamodel
- Place it in the folder:
  - `\eclipse\plugins\org.sintef.mofscript.editor_1.2.0\repository\metamodels`
- This will enable metamodel support in MOFScript

MOFScript Example
Example Model

```
uml.Class::main(){
  file(self.name+ "\ java")
  'package ' packageName';
  import java.util.*;
  self.visibility' class ' self.name'{
    self.ownerAttribute->forEach(p:uml.Property | p.association = null ){
      p.visibility" " p.type.name" "p.name';"n'
    }
    self.ownerAttribute->forEach(p:uml.Property | p.association !=null ){
      // Association: authors:Author{1..-1}
      "" p.visibility' HashMap< "p.type.name', "p.name">_'
      p.name.toLowerCase();"n'
    }
  }
```

Example Transformation
Example Generated Java Code

```java
package org.sintef.no;
import java.util.HashMap;
public class Book {
    private String _title;
    // Association: authors:Author(1..-1)
    protected HashMap<String, Author>_authors;
    // Association: category:Category(1..-1)
    protected HashMap<String, Category>_category;
}
```

Overview

```java
package org.sintef.no;
import java.util.HashMap;
public class Book {
    private String _title;
    // Association: authors:Author(1..-1)
    protected HashMap<String, Author>_authors;
    // Association: category:Category(1..-1)
    protected HashMap<String, Category>_category;
}
```
References

- OMG MOF Model to Text Transformation RFP
- MOFScript submission
- MOFScript tool
  http://www.modelbased.net/mofscript
  http://www.eclipse.org/gmt/mofscript
- OMG MOF to Text http://www.omg.org/docs/ptc/06-11-01.pdf

Demonstration