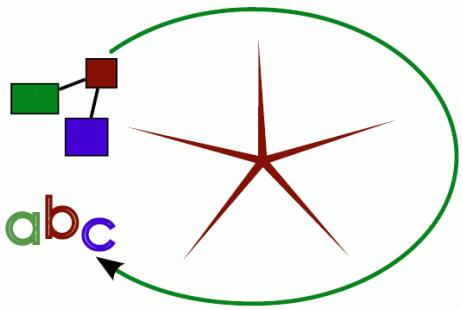
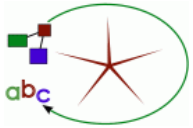


MOF Model to Text Transformation - MOFScript

The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).

June 2006

Outline

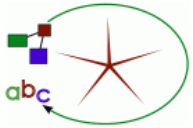


- Background: Model to text transformation
- Introducing the MOFScript language
- MOFScript and the relation to MOF/QVT/OCL
- Details of the MOFScript language: Concrete syntax
- Examples: From UML class diagrams to Java
- MOFScript advanced features
- The MOFScript tool

Context of this work

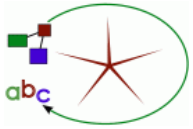
- The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).
- Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).
- To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.

Motivation



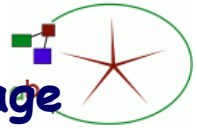
- Why do we need model-to-text transformation?
  - Raise the level of abstraction
    - Systems are getting more complex
    - Raise of abstraction has proven useful (for instance: Assembly to COBOL)
  - Automation of the software development process
    - Decrease development time
    - Increase software 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

Motivation - 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

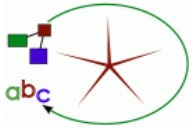
- Properties of the alternatives:
  - Neither programming languages nor scripting languages tend to take advantage of source metamodels.
    - However, it can be done programmatically in Java (e.g. using Eclipse Modelling Framework (EMF))
  - Model 2 Model Transformation languages such as ATL is metamodel-based, 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

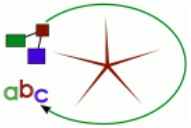
- OMG RFP Issued in 2004
- Mandatory Requirements:
  - Generation from MOF 2.0 models to text
  - Reuse (if applicable) existing OMG specifications, in particular QVT
  - Transformations should be defined at the metalevel of the source model
  - Support for string conversion of model data
  - String 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.

What is MOFScript?



- The MOFScript language is an initial proposal to the OMG model-to-text RFP.
- The MOFScript tool is an implementation of the MOFScript model to text transformation language
  - http://www.modelbased.net/mofscript
  - http://www.eclipse.org/gmt/mofscript
- An Eclipse plug-in
- Developed by SINTEF ICT in the EU-supported MODELWARE project
- Ongoing standardization process within OMG
  - OMG RFP MOF Model to Text Transformation process
  - MOFScript tool and language was part of this process
  - Was in a merging process with other proposal toward the final OMG standard (MOF2Text)

MOFScript in action



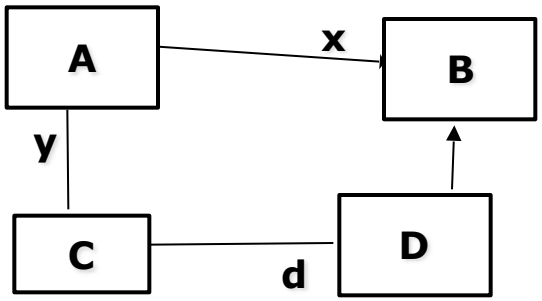
MOF MODELS

- UML
- RDBMS
- BPMN
- WSDL
- …

MOFScript

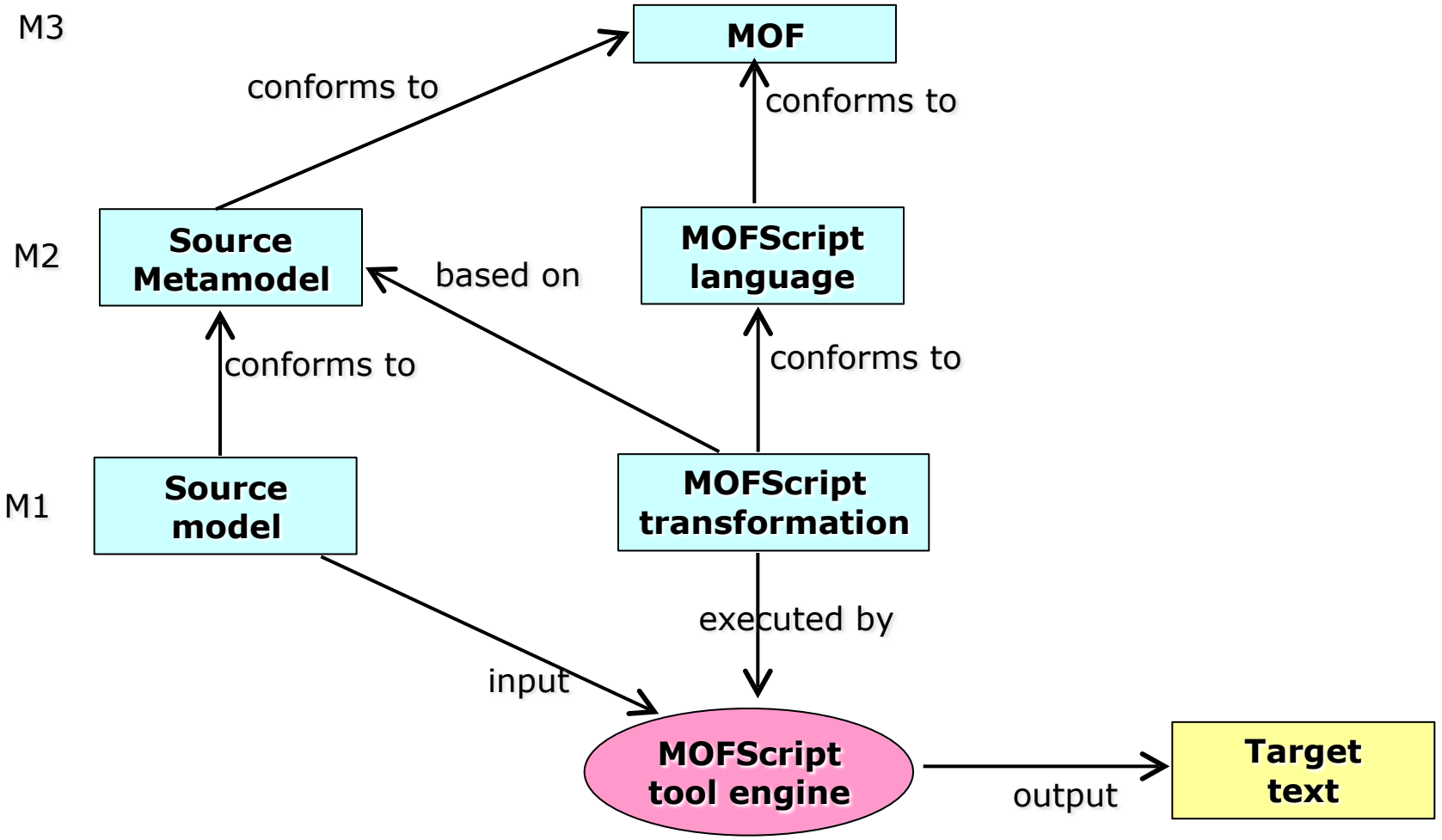
Textual output

- Java, C#, …
- Documentation
- SQL
- XML, HTML…
- …

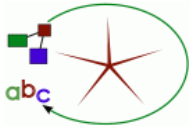


Public class a extends x
……..
B.wsdl
……
Z association type Simple
c.Html

MOFScript placed in the 4-layer architecture

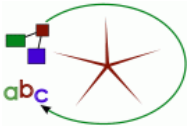


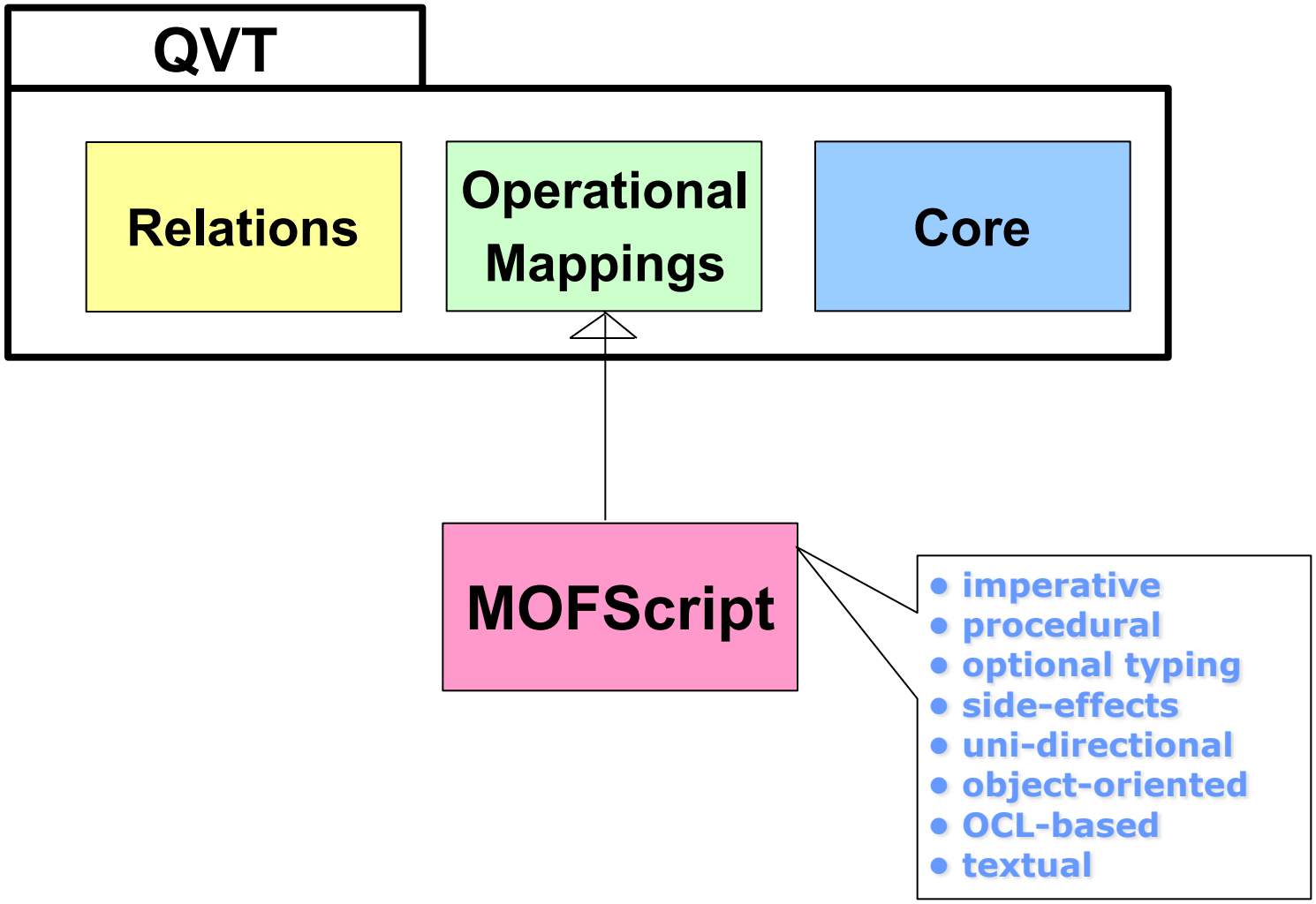
MOFScript - background



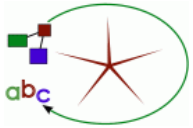
- Usability
  - Ease of use: Writing and understanding
  - Few constructs
- End user recognizability
  - Similar to programming and scripting languages
  - Imperatively oriented
- Sequential execution semantics
  - Rules are called explicitly (except the main() rule)
  - Explicit starting point
  - Contents of rules is executed sequentially
- Compatibility
  - Alignment with QVT standard and OCL

MOFScript extends QVT Mappings

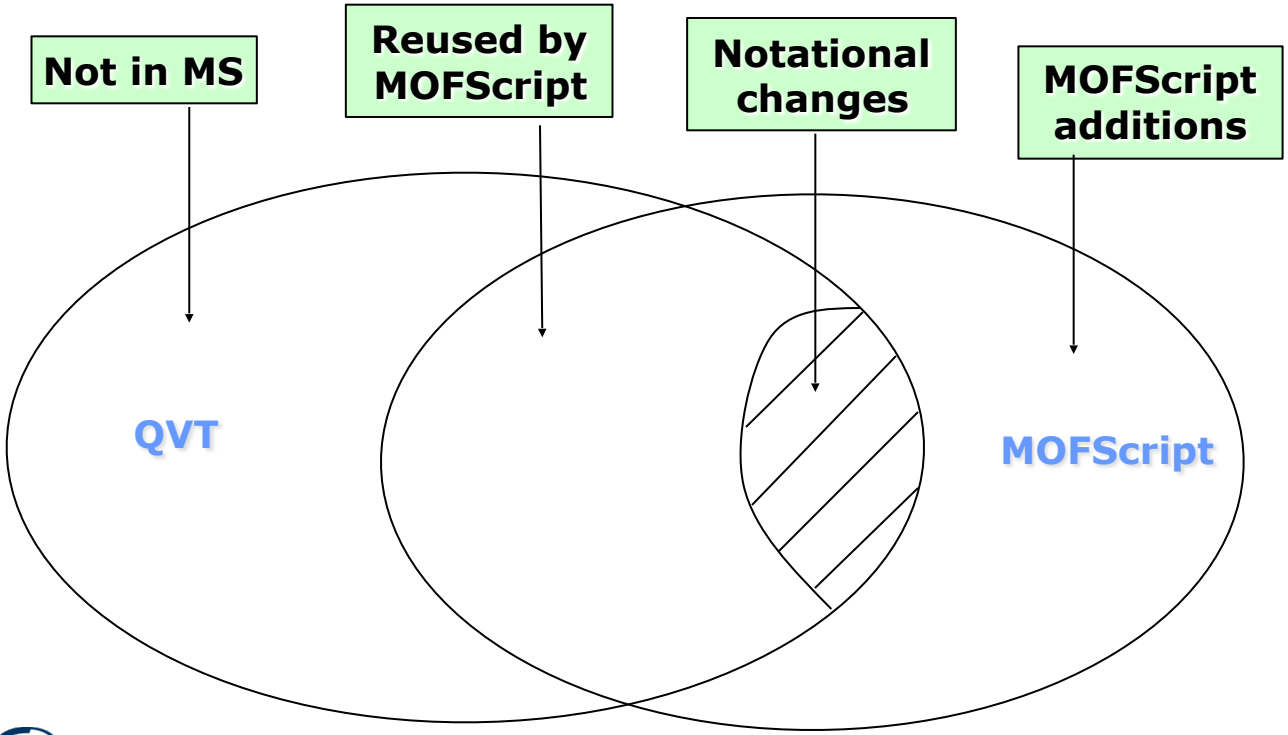


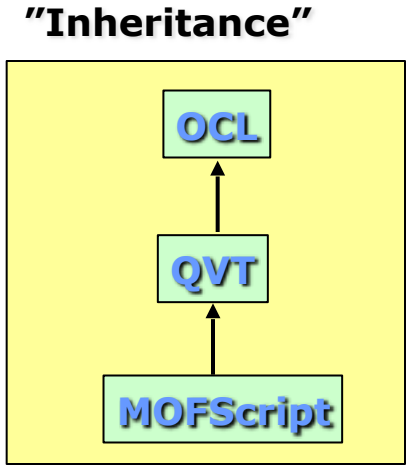


MOFScript relates to QVT

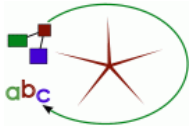


- MOFScript is strongly influenced by OMGs Query/Views/ Transformations (QVT)
  - OMGs standard for model 2 model transformation
  - QVT is reuses and specialises OCL (adds new expressions)
- MOFScript specialises QVT
  - Similar notation, reuse of metamodel concepts.



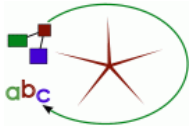


MOFScript - A text transformation language

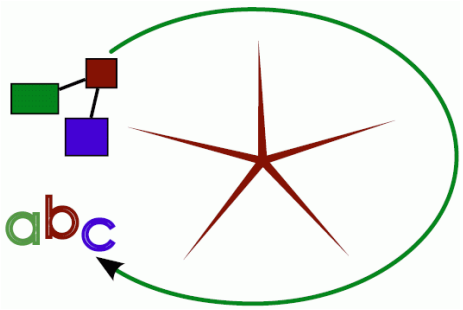


- Built around a set of rules that are called explicitly:
  - rules appear as methods
- The language structure is "flat":
  - A transformation consists of a set of rules/methods
  - Transformations cannot be nested
  - Rules cannot be nested
  - But control structures (if statement and loops) can be nested.
- Transformations may be imported or subtyped for reusing rules and redefining rules
- Data types:
  - String, Integer, Real, Boolean, List, Hashtable, Object
- Variables and constants
  - Global or local, which are optionally typed
    - Untyped variables will assume the type of its assignments
  - variables can be modified

Function libraries

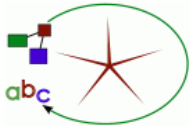


- String library
  - size, substring, subStringBefore|After, toLower, toUpper, indexOf, trim, normalizeSpace, endsWith, startsWith, replace, equals, equalsIgnoreCase, charAt, isLowerCase, isUpperCase
- Collection library
  - Hashmap: put, get, clear, size, keys, values, isEmpty, forEach,
  - List: add, size, clear, isEmpty, first, last, forEach
- Utility library
  - generateID, time, date, getenv, setenv, indent, tab, space, position, count
- UML2 Operations
  - m.hasStereoType ("stereotype"): Boolean
  - m.getAppliedStereotypes () : Collection
  - m.getAppliedStereotype ("named stereotype") : Stereotype
  - m.getValue (stereoType, "property name"), hasValue (stereoType, "prop name")



MOFScript: Concrete syntax

Textual syntax



- Texttransformation

```
texttransformation UML2Java (in myMod:uml2)
```

- Rules

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

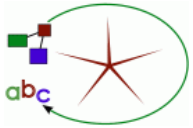
- Files

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

- Escaped output

```
  'public class ' c.name
      ' extends Serializable {'
```

Syntax of a text mapping operation (text transformation rule)



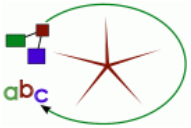
Template

```
abstract? <metamodelContextType>::methodName(Type1 p1, Type2 p2,...) :
    returnType
     when { … }
{   // Main body   }
```

Example

```
uml.Class::classToJava(String classPrefix)
     when {self.getStereotype() = 'Entity' }
{ //body }
```

Simplification of the QVT rule template



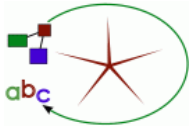
Template

```
abstract mapping <dirkind0> <metamodelContextType>::methodName
        (<dirkind1> p1:P1, <dirkind2> p2:P2) : r1:R1, r2:R2
            when { … }
            where { … }
        {
            init { … }
            population { … }
            end { … }
        }
```

Changed notation:
p1:P1 ➔ P1 p1

Only one (unnamed) returnType allowed

Textual syntax



- Entry point rule

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

- Iterator (forEach)

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

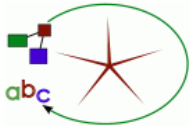
- While loops

```
while (myCounter > 0) {
  'counter value : ' counter
}
```

- 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 '
}
```

Textual syntax



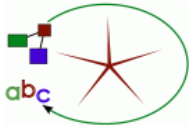
- Collections

```
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 which do not start with 'S' '
  packageName_Hashtable->forEach (name:String |
                                  not(name.startsWith("S")) {
     ' Package: ' + name
  }
}
```

Textual syntax



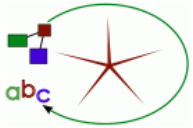
- Invoking rules

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

- Return results

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

Printing to files - example



Scope of file f1

```
rule1() {
    file f1 ("f1.java");
    println("first output from rule 1");
    rule2();
    ...
    f1.println("second output from rule 1");
}

rule2() {
    println("first output from rule 2");
    file f2 ("f2.java");
    println("second output from rule 2");
}
```

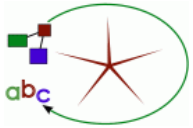
f1.java

f2.java

Scope of file f2

The current output stream in the runtime stack will be applied to this statement

Textual syntax

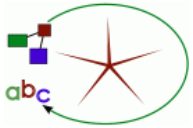


- Properties and variables

```
property aProperty:String = "myProp"
var packageNames:List
var packageIdList:Hashtable
self.ownedMember->forEach(p:uml.Package) {
   packageNames.add (p.name)
   packageIdList.put (p.id, p.name)
}

if (packageIdList.size () > 0) {
  ' Listing the package names which do not start with S '
  packageIdList->forEach (s:String | not(s.startsWith("S")) {
     ' Package: ' s
  }
}
```

Helper rules / functions



- How to define helper rules that do not have a context type?
  - Use the keyword module instead of context type in a transformation rule (or ignore the context type altogether)

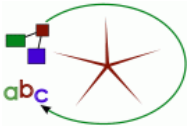
Template

```
(module::)? ruleName(Type1 p1, Type2 p2,...) : returnType
{   // Main body   }
```

Example

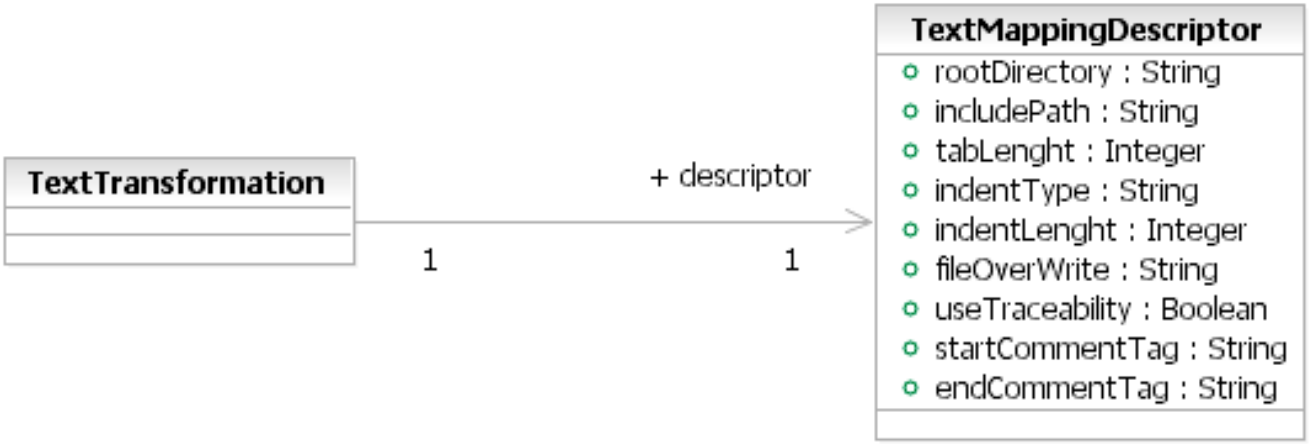
```
module::factorial(integer n) : integer
{ if (n = 1) result = 1; else result = n * factorial (n – 1); }
```

Output Files (Output devices)

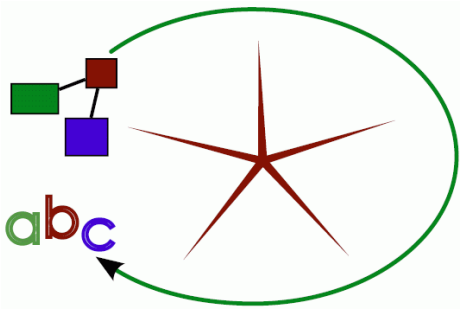


- The built-in **stdout** keyword represents console output. This is used if no other output device is currently active.
- The specification shows only how to use files or standard console as output, but describes that other may be defined (SQL db, CVS/SVN etc.). How to use these are not further described.
- print/println may have an explicit file which overrides and replaces the current one: f1.print()
- Current output file: Applies to anonymous print/println or escaped output
- The default file has dynamic scope and will be in scope until the defining file rule is finished executing.

Configuration properties: TextMappingDescriptor

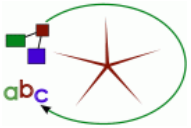


- White space handling, file paths, comment delimiters
- fileOverWrite (always | never | merge). Defines the strategy when a file already exists.
  - Not implemented in the current MOFScript tool.



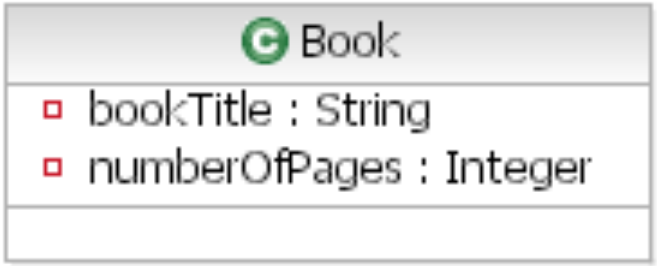
Examples: From UML class diagrams to Java

Uml2Java Example: Class and attributes



```
//Context class
self.ownedAttribute->forEach(p : uml.Property | p.association = null) {
   p.attributeGetterSetters()
}
// Generate Getter and Setters
 uml.Property::attributeGetterSetters () {
    'public ' self.type.name ' get' self.name.firstToUpper() ' () {\n'
    '\treturn ' self.name ';\n }\n'
    'public void set' self.name.firstToUpper() '(' self.type.name ' input ) {\n '
     '\t'self .name ' = input; \n }\n '
 }
```

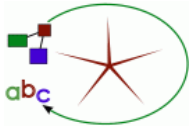
Source



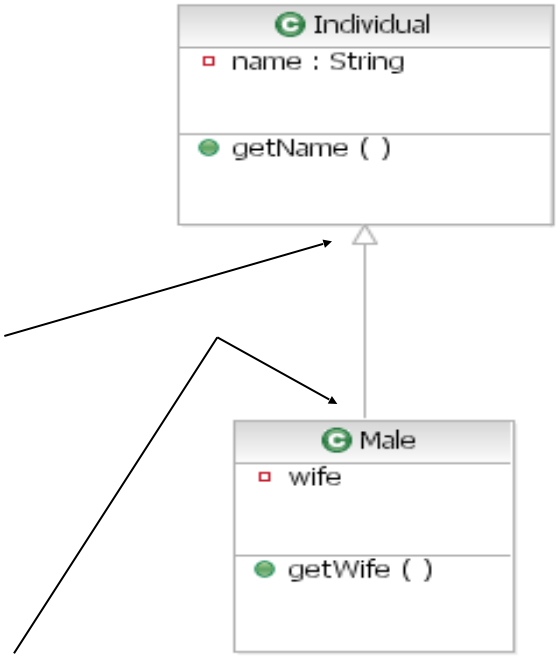
Target

```
public String getBookTitle(){
   return bookTitle;
}
public void setBookTitle(String input){
   bookTitle = input;
}
public Integer getNumberOfPages(){
   return numberOfPages;
}
public void setNumberOfPages(Integer input){
   numberOfPages = input;
}
```

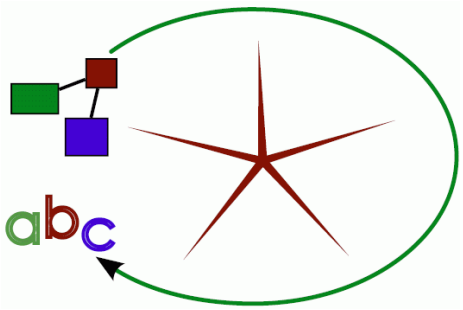
UML2Java example: Generalization property



```
uml.Class::outputGeneralization(){
  self.generalization->forEach(g: uml.Generalization){
     if(not g.target.isEmpty()){
        g.target->forEach(c: uml.Class){
           stdout.println("Generalization target name: "+ c.name )
        } //g.target forEach
    }//if target
    if(not g.source.isEmpty()){
        g.source->forEach(c:uml.Class){
            stdout.println("Generalization source name: "+c.name)
        }//g.source forEach
    }//if source
}//self.generalization
}//outputGeneralization()
```

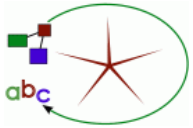



```
Generalization target name: Individual
Generalization source name: Male
```



MOFScript advanced features

Transformation inheritance

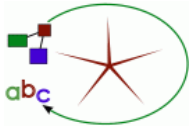


```
texttransformation TestInheritanceSub (in ecmodel:ecore) extends
    TestInheritanceSuper {

    ecmodel.EPackage::main() {
        self.printMe()
    }

    ecmodel.EPackage::printMe() {
        stdout.println ("TestInheritanceSub::printMe begin")
        super.printMe();
        stdout.println ("TestInheritanceSub::printMe end")
    }
}
```

Black-box transformations – Integration with Java



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

Method signature

```
java (String className, String methodName, List/Something parameters, String
    classpath)
```

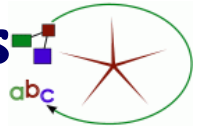
Example: Compute the factorial of 6

```
java ("org.test.MathClass",
    "factorial", 6, "c:/
    Working/TestJava/")
```

MathClass.Java

```
class MathClass {
public static integer factorial (Integer n) {..}
```

Using external Java methods w/ advanced parameters



***Example*: Multiple parameters**

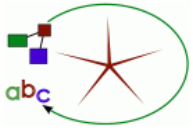
```
List parameterList;
// Insert parameters to this list
...
java ("org.test.SomeClass", "calculate",
    parameterList, "c:/Working/TestJava/")
```

***Example*: source model element as parameter**

```
class SomeClass {
    public static printClass (uml.Class class) {...}
}
```

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

Change management

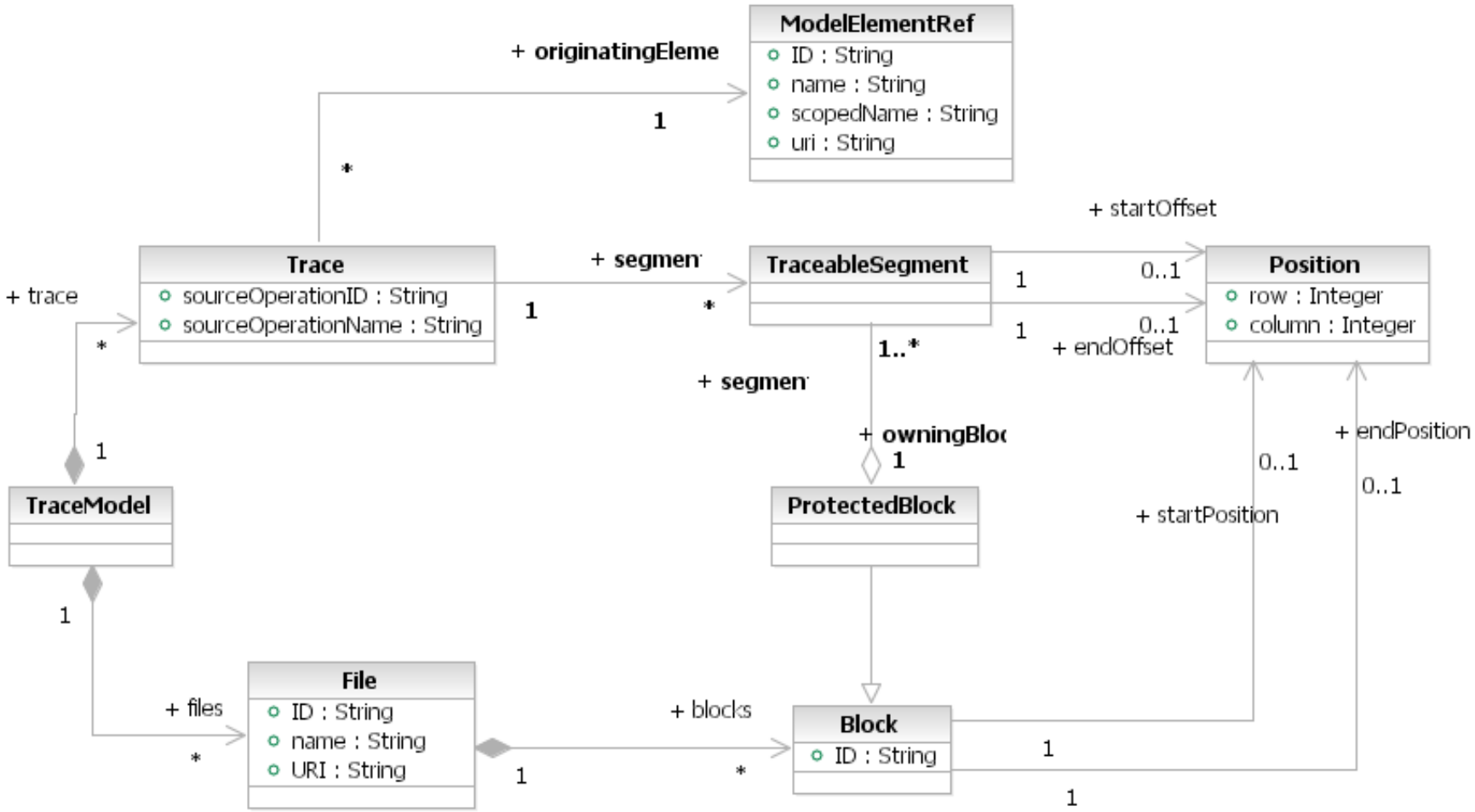


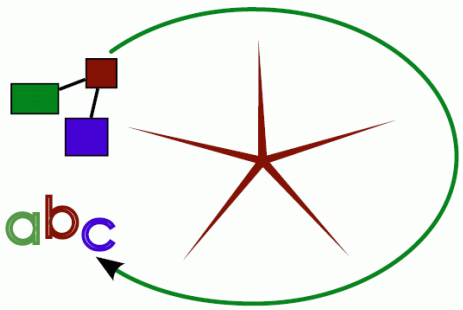
- Proposed metamodel for handling:
  - Links from a model toward code blocks
  - Traces from a model to text blocks
- Controlled by user

```
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

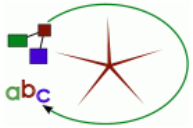
Traceability model





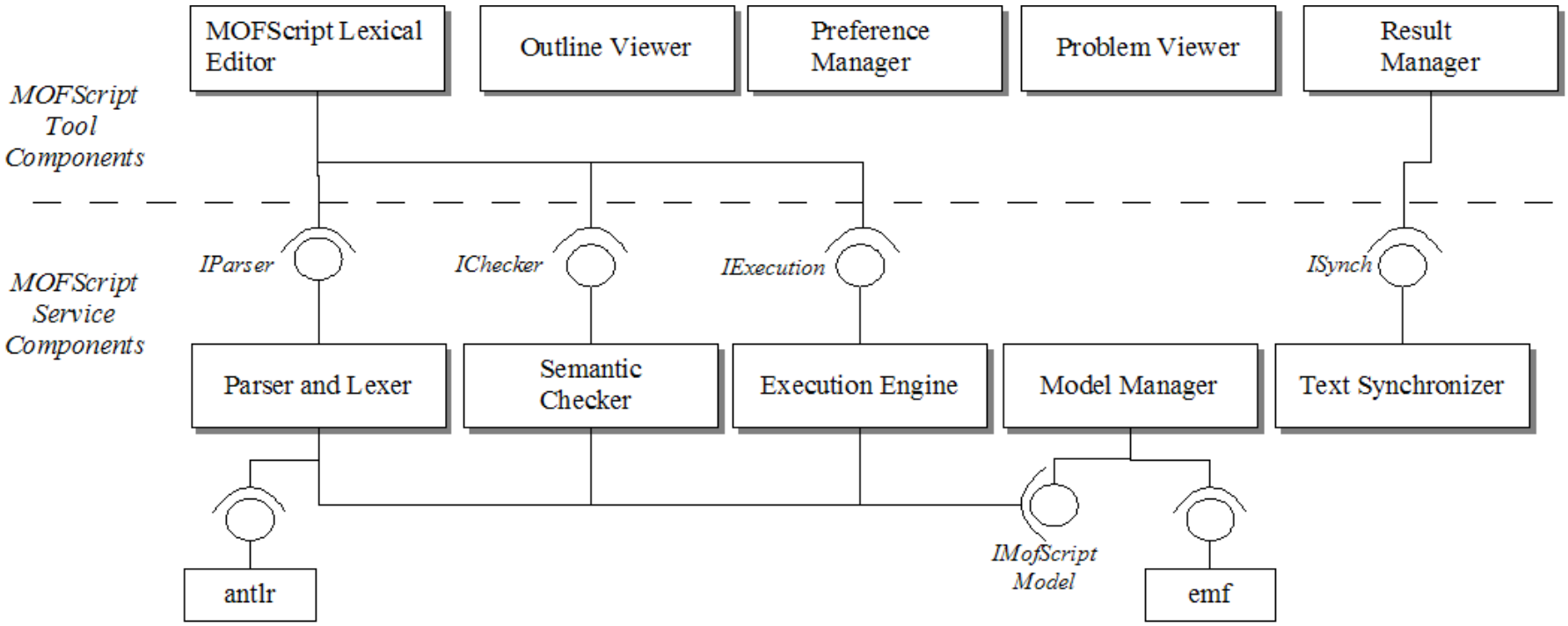
The MOFScript tool

The MOFScript tool



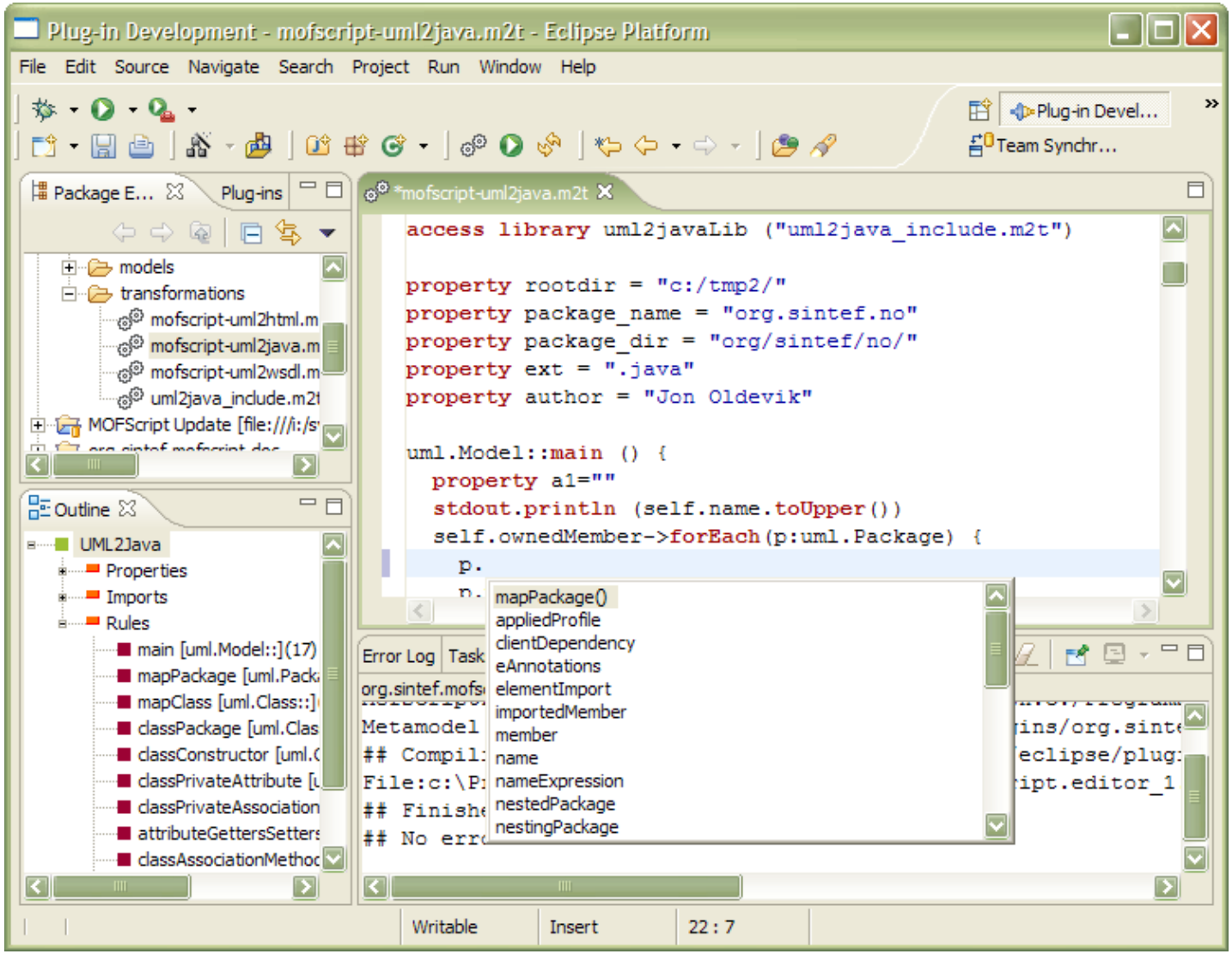
- An Eclipse plug-in
- Developed at SINTEF ICT
- Does not implement the whole specification, particularly with respect to the inheritance from QVT/OCL
- Supports EMF meta-models and models as input
  - UML2, Ecore, or your own based on EMF

MOFScript architecture

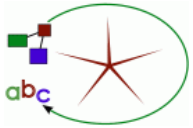


MOFScript a model to text tool

- Editing, compiling and executing
- Code-completion drop-down box
- Source code tree outline and colored layout
- MOFScript console
- File properties

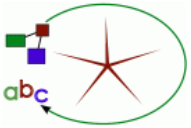


The main steps of using the MOFScript tool



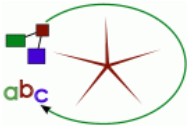
- Task: Define a transformation from source model A to text t. (A→t)
  1. Import or define the source metamodel for A.
  2. Write the MOFScript to transform A to t in the MOFScript editor
  3. Compile the transformation. Any errors in the transformation will be presented.
     1. Fix errors, if any
  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.

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).

OMG Standard. Example: MOF2Text Templates



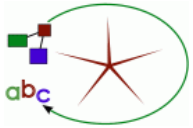
Template:

```
[template public ClassToTable(c: Class)]
	CREATE TABLE [c.name/] (
		[c.name+'_id'/] NUMBER;
	);
	ALTER TABLE [c.name/] ADD (
		CONSTRAINT [c.name+'_pky'/] PRIMARY KEY ([c.name+'_id'/]);
	);
[/template]
```

For the class 'Person' the following text will be generated:

```
	CREATE TABLE Person (
		Person_id NUMBER;
	);
	ALTER TABLE Person ADD (
		CONSTRAINT Person_pky PRIMARY KEY (Person_id);
	);
```

References



- OMG MOF Model to Text Transformation RFP
  - http://www.omg.org/cgi-bin/apps/doc?ad/04-04-07.pdf
- MOFScript submission
  - http://www.omg.org/cgi-bin/apps/doc?ad/05-05-04.pdf
- MOFScript tool
  - http://www.modelbased.net/mofscript
  - http://www.eclipse.org/gmt/mofscript