Adding a Module Concept to the Model Transformation Language Xtend

Bachelor Thesis
Dominik Werle | September 19, 2013

Advisor: Dipl.-Inform. Andreas Rentschler
Second Advisor: Dipl.-Inform. Max Kramer
Motivating example

ActivityModel

Activity

succ

\[\text{actions}\]

\[0..1\]

Process

\[\text{ProcessModel}\]

\[\text{Step}\]

+isStart: bool
+isStop: bool

next

\[0..1\]
Monolithic implementation

transformation Activity2Process(
in a:ActivityModel, out p:ProcessModel);

main() {
  a.rootObjects()[a::Activity]->map Activity2Process();
}

mapping Activity::Activity2Process() : p::Process {
  result.steps := self.actions->map Action2Step();
}

mapping Action::Action2Step() : p::Step {
  result.succ := self.next.late resolveone(p::Step);
  result.isStart := self.oclIsTypeOf(a::StartAction);
  result.isStop := self.oclIsTypeOf(a::StopAction);
}
Evolving meta model

- Adding “name” attribute to Actions/Steps
- What needs to be changed? (= finding location of concerns)
- Structure does not explicitly declare which model elements are accessed in mappings
### Problem
- Transformations get more and more **complex**
- **Maintainability** suffers

### Idea
- Introduce special **module concept** for transformations
- **Explicit interfaces:**
  - meta model visibility, information hiding

### Benefits
- Simplify **location of concerns**
- Facilitate **understandability** (already at design time)

### Actions
- Design **module concept** for model transformations
- **Implement framework** for Xtend
- **Evaluate concept** with maintenance scenarios
Related work

- Classification (Czarnecki, 2006)
- External composition (Vanhooff et al., 2007, Wagelaar, 2008)
- Packaging concepts (Klar et al., 2007)
- TGGs (Ehrig et al., 1998)
- Information hiding (Parnas, 1972)
- Modularity (Szyperski, 1992)

Our approach

- Modularity
- Meta model access restriction
- Model transformations
module interface IMain(
    in a:ActivityModel,
    out p:ProcessModel) {
  mapping a::Activity2Process() : p::Process;
}

module implementation Activity2Process
  export IMain
  import IAction2Step {
    mapping a::Activity::Activity2Process() : p::Process { ... }
  }

module interface IAction2Step(
    in a:ActivityModel, out p:ProcessModel) {
  mapping a::Action::Action2Step() : p::Step;
}

module implementation Action2Step
  export IAction2Step {
    mapping a::Action::Action2Step() : p::Step { ... }
  }

unit of decomposition:
  mapping
Structured implementation

```java
module interface IMain(
in  a:ActivityModel,
    out p:ProcessModel) {
mapping  a::Activity2Process() : p::Process;
}

module implementation Activity2Process
export IMain
import IAction2Step {
mapping  a::Activity::Activity2Process() : p::Process { ... }
}

module interface IAction2Step(
in  a:ActivityModel, out p:ProcessModel) {
mapping  a::Action::Action2Step() : p::Step;
}

module implementation Action2Step
export IAction2Step {
mapping  a::Action::Action2Step() : p::Step { ... }
}
```

Motivation

Related work

Design

Implementation

Evaluation

Conclusion & Future work

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend

September 19, 2013
module interface IMain(
    in a:ActivityModel,
    out p:ProcessModel) {
  mapping a::Activity2Process() : p::Process;
}

module implementation Activity2Process
  export IMain
  import IAction2Step {
    mapping a::Activity::Activity2Process() : p::Process { ... }
  }

module interface IAction2Step(
    in a:ActivityModel, out p:ProcessModel) {
  mapping a::Action::Action2Step() : p::Step;
}

module implementation Action2Step
  export IAction2Step {
    mapping a::Action::Action2Step() : p::Step { ... }
  }
Designing the module concept

Meta model visibility

- Interface concept that is different from other module systems – special to model transformations
- parts of meta model navigable inside the module ⇒ find locations of concern easier
- Model In – What can be read?
- Model Out – What can be created?
Structured implementation

```xtend
module interface IMain(
    in a:ActivityModel[Activity, Action],
    out p:ProcessModel[Process]) {
  mapping a::Activity2Process() : p::Process;
}

module implementation Activity2Process
  export IMain
  import IAction2Step {
    mapping a::Activity::Activity2Process() : p::Process { ... }
}

module interface IAction2Step(
    in a:ActivityModel[StartAction, StopAction],
    out p:ProcessModel[Step]) {
  mapping a::Action::Action2Step() : p::Step;
}

module implementation Action2Step
  export IAction2Step {
    mapping a::Action::Action2Step() : p::Step { ... }
}
```

Motivation

Related work

Design

Implementation

Evaluation

Conclusion & Future work

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend
September 19, 2013
Designing the module concept

Meta model visibility (cont.)
- Support meta model structure: ModelOut = “seff.**”
- Restrict access through associations

```scala
val parentNet = resolveOne(context.encapsulatedComponent, typeof(Graph));
```

Motivation

Related work

Design

Implementation

Evaluation

Conclusion & Future work

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend

September 19, 2013 10/22
Implementation

**Xtend**

- Based on Xtext/Xbase
  - integrated with Eclipse
  - *extendable with new language elements* (active annotations)
- Model transformation environment
  - basic tracing
  - model workflow engine

Two implementation steps

1. Xtend for model transformations (QVT-style M2M Mappings)
2. Module system (Imports and module interfaces)
Separate interface and implementation

- *Java interfaces* and *classes* as module *interfaces* and *implementations*
- Workflow component: wiring module interface and implementation + entry point
- Global *configuration* and *tracing*
- Dependency Injection with *Guice*
Example – Module interface

```java
@TransformationInterface
@ModelIn("cm.repository.*", "cm.seff.*")
@ModelOut("cg.*")
interface CM2CGSEFF {
  def Graph toNet(ProbabilisticBranchTransition self, AssemblyContext context, ProvidedRole role)
  def AbstractAction toDisjunctPlace(...)
  def InputAction toAction(...)
  def OutputAction toAction(...)
  ...
}
```

```java
@Transformation
class CM2CGSEFFImpl implements CM2CGSeff {
  @Import extension CMLibrary
  override Graph toNet(...) {
    ...
  }
  ...
}
```

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend
September 19, 2013

Motivation

<table>
<thead>
<tr>
<th>Related work</th>
<th>Design</th>
<th>Implementation</th>
<th>Evaluation</th>
<th>Conclusion &amp; Future work</th>
</tr>
</thead>
<tbody>
<tr>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
</tr>
</tbody>
</table>

Motivation

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Related work</th>
<th>Design</th>
<th>Implementation</th>
<th>Evaluation</th>
<th>Conclusion &amp; Future work</th>
</tr>
</thead>
<tbody>
<tr>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
<td>oooo</td>
</tr>
</tbody>
</table>
Example – Module interface

```
@TransformationInterface
@ModelIn(#"cm.repository.*", "cm.seff.*")
@ModelOut(#"cg.*")

interface CM2CGSEFF {
  def Graph toNet(ProbabilisticBranchTransition self, AssemblyContext context, ProvidedRole role)
  def AbstractAction toDisjunctPlace(...)
  def InputAction toAction(...)
  def OutputAction toAction(...)
}
```

```
@Transformation
class CM2CGSEFFImpl implements CM2CGSeff {
  @Import
extension CMLibrary
  override Graph toNet(...) { ... }
}
```

Concept

- implementation provides methods
- control dependencies (methods)
- module implements module interface
- only module interfaces imported
- meta model visibility

Implementation

- Xtend type system
- active annotations
- modified Xtend validator

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend

Motivation

- Related work
- Design
- Implementation
- Evaluation

Conclusion & Future work

September 19, 2013
Example – Module interface

```
package trafo.impl

import edu.kit.student.dwerle.xtendfw.annotations.Transformation

@Transformation
class Foo {
  Transformation module Foo does not implement transformation interface
  def normalClassMethod
}
```

```
package trafo.impl

import edu.kit.student.dwerle.xtendfw.annotations.Transformation

@Transformation
class Foo {
  @Import extension NormalClass
  trafo.impl.NormalClass is not a transformation module
}
```
// for each branch, create weighted transitions to new composite action
val parentNet = resolveOne(context.encapsulatedComponent, typeof(Graph));
log('Adding branching action', self2.branchTransitions.
val p = createObject(typeof(ComposedAction)) [ 
  name = bt.entityName
  net = bt.toNet(context, role)
]
interface CM2CGSEFF {
    def Graph toNet(ProbabilisticBranchTransition self, AssemblyContext context, ProvidedRole role)

    def cg.AbstractAction toDisjunctPlace(AbstractAction self, AssemblyContext context, ProvidedRole role)

    def InputAction toAction(StartAction self, AssemblyContext context, ProvidedRole role)

    def OutputAction toAction(StopAction self, AssemblyContext context, ProvidedRole role)

    def Action toAction(InternalAction self, AssemblyContext context, ProvidedRole role)
}

// for each branch, create weighted transitions to new composite action
val parentNet = resolveOne(context.encapsulatedComponent, \n    log('Adding branching action
    self2.branchTransitions.\n    (typeof(ComposedAction)).createObject
    val p = createObject(typeof(ComposedAction)) [ \n        name = bt.entityName
        net = bt.toNet(context, role)
Evaluation

- Problem: toy examples lack complexity
- Therefore: analysis of existing transformation: *SimuCom*
  - Model-to-text transformation from Palladio Context
  - Already “modular” structure (separated by aspects in different files)
SimuCom transformation

- Files translated into modules (according to our concept)
- Specified interfaces

<table>
<thead>
<tr>
<th>Module</th>
<th>reliability</th>
<th>resource</th>
<th>environment</th>
<th>usagemodel</th>
<th>repository</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaCore</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>244</td>
</tr>
<tr>
<td>JavaNamesExt</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>278</td>
</tr>
<tr>
<td>PCMExt</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>480</td>
</tr>
<tr>
<td>ProvidedPorts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>260</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modules: 39

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>2</th>
<th>11</th>
<th>13</th>
<th>9</th>
<th>30</th>
<th>(\sum) = 4987</th>
</tr>
</thead>
<tbody>
<tr>
<td>%LOC</td>
<td>18%</td>
<td>7%</td>
<td>35%</td>
<td>42%</td>
<td>28%</td>
<td>87%</td>
<td></td>
</tr>
<tr>
<td>%Module</td>
<td>10%</td>
<td>5%</td>
<td>28%</td>
<td>33%</td>
<td>23%</td>
<td>77%</td>
<td></td>
</tr>
</tbody>
</table>
Change scenario: Multiple interface inheritance

- **Problem**: Palladio meta model & tooling support interfaces that extend other interfaces
- SimuCom *did not*, changed in a bug fix
- Change is restricted to 4 modules
- Course-grained visibility on package level not enough
  ⇒ inspect modules that access Interface or subclasses

```
Extender <<Extends>> AnInterface

void methodA()
```
## Change scenario: Multiple interface inheritance

<table>
<thead>
<tr>
<th>Module</th>
<th>class</th>
<th>modified</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComposedStructure</td>
<td>X</td>
<td></td>
<td>214</td>
</tr>
<tr>
<td>ContextPattern</td>
<td>X</td>
<td></td>
<td>187</td>
</tr>
<tr>
<td>DelegatorClass</td>
<td>X</td>
<td>X</td>
<td>96</td>
</tr>
<tr>
<td>Dummies</td>
<td>X</td>
<td></td>
<td>89</td>
</tr>
<tr>
<td>JavaCore</td>
<td>X</td>
<td>X</td>
<td>244</td>
</tr>
<tr>
<td>ProvidedPorts</td>
<td>X</td>
<td>X</td>
<td>260</td>
</tr>
<tr>
<td>Repository</td>
<td>X</td>
<td>X</td>
<td>120</td>
</tr>
<tr>
<td>Sensors</td>
<td>X</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Usage</td>
<td>X</td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>SimDummies</td>
<td>X</td>
<td></td>
<td>123</td>
</tr>
<tr>
<td>SimJavaCore</td>
<td>X</td>
<td></td>
<td>262</td>
</tr>
<tr>
<td>SimProvidedPorts</td>
<td>X</td>
<td></td>
<td>118</td>
</tr>
<tr>
<td>SimRepository</td>
<td>X</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

...  

**Modules: 39**

<table>
<thead>
<tr>
<th>LOC</th>
<th>Module</th>
<th>14</th>
<th>4</th>
<th>[ \sum ] = 4987</th>
</tr>
</thead>
<tbody>
<tr>
<td>%LOC</td>
<td></td>
<td></td>
<td></td>
<td>39%</td>
</tr>
<tr>
<td>%Module</td>
<td></td>
<td></td>
<td></td>
<td>36%</td>
</tr>
</tbody>
</table>
Conclusion & Future work

- **Conclusion:**
  - *design of module concept* for model transformations
  - *prototypical implementation* in Xtend
  - examples for improved *maintainability* for model transformations

- **Future work:**
  - integrate into QVTo
  - develop a larger model-to-model transformation from scratch
  - empirical study to test usability
Meta model visibility

- **Meta model visibility**: active annotations @ModelIn and @ModelOut
- **Modified validator** for the Xtend-to-Java-translation
  - class implements an interface with @TransformationInterface?
  - collect @ModelIn and @ModelOut
  - traverse the AST of all methods in the implementation
  - check all method calls and references against meta model visibility
  - attach errors (strict mode) or warnings (lenient mode) appropriately to AST

- Dynamic validation of created objects:
  Factory checks @ModelOut and throws exception if necessary
Change scenario: PCM.ext

- Helper module (due to Xpand+Xtend)
- Refactoring: Pull methods into appropriate modules
  \[ \Rightarrow \text{increase cohesion} \]
- Simplified through interfaces:
  - “Bad smell” localized with interfaces (meta model visibility + import relationship)
  - Import relationship allows for determining target model

<table>
<thead>
<tr>
<th>Used in</th>
<th>( \ldots )</th>
<th>M4</th>
<th>( \ldots )</th>
<th>M8</th>
<th>M9</th>
<th>M10</th>
<th>M11</th>
<th>( \ldots )</th>
<th>M15</th>
<th>M16</th>
<th>( \sum )</th>
<th>Refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasReqDelCon</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1 ( \rightarrow ) M4</td>
<td></td>
</tr>
<tr>
<td>getReqDelCon</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1 ( \rightarrow ) M4</td>
<td></td>
</tr>
<tr>
<td>querySyCalls</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>not obvious</td>
<td></td>
</tr>
</tbody>
</table>

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend
Create methods

- Xtend: only simple tracing (argument hashing)
- Our mappings: create methods
  - Active Annotation: @Creates
  - delegates object creation ⇒ validation of object creation
  - creates trace

- New: “QVT style” tracing:
  - explicit trace resolving
  - modifiers (late, inverse, ...)

- New: Object creation by factory inside framework
  ⇒ simpler syntax

- Patterns for translating QVTo to Xtend
def create_net : CgFactory::eINSTANCE.createGraph toNet(
    Entity self,
    AssemblyContext context,
    ProvidedRole role, Signature operation)
    // ...
}

public Graph toNet(final Entity self,
    final AssemblyContext context,
    final ProvidedRole role,
    final Signature operation) {
    final ArrayList<? > _cacheKey =
        CollectionLiterals.newArrayList(self, context, role,
            operation);
    final Graph _result;
    synchronized (_createCache_toNet) {
        if (_createCache_toNet.containsKey(_cacheKey)) {
            return _createCache_toNet.get(_cacheKey);
        }
        Graph _createGraph = CgFactory.eINSTANCE.createGraph();
        _result = _createGraph;
        _createCache_toNet.put(_cacheKey, _result);
    }
    _init_toNet(_result, self, context, role, operation);
    return _result;
}
Example for Create Methods

```scala
@Creates(typeof(Graph))
override Graph toNet(InterfaceProvidingRequiringEntity self,
  AssemblyContext context,
  ProvidedRole role, Signature operation) {

  val seff = getSimpleBehaviorSpecification(self, operation)
  val steps = seff.steps

  result.name = self.entityName + ": " + role.entityName + ": " + operation.entityName
  result.actions += steps.map[ step |
    toDisjunctPlace(step, context, role) ]
  result.transitions += steps.map[ step |
    toTransition(step) ].filterNull
}
```
Active Annotations

- **Annotation Processor** for an Annotation
- called during *compilation* (to Java)

```java
@Transformation
class Foo {
}

@Transformation
class Foo {
    @Inject @Extension private TracingAPI __tracingAPI;
    @Inject @Extension private ModuleFactory __moduleFactory;
    @Inject @Extension private Config __config;
}

class Foo {
    @Inject extension TracingAPI
    @Inject extension ModuleFactory
    @Inject extension Config
}
```
Active Annotations 2 – @Creates

```java
@Creates(value = typeof(Graph),
         name = "newObject")
def Graph foo(Object self, String a) {
    newObject.setName("bar")
}
```

```java
@Creates(value = Graph.class, name = "newObject")
public Graph foo(final EObject self, final String a) {
    final cg.Graph newObject = this.__moduleFactory.<cg.Graph>createObject(cg.Graph.class);
    this.__tracingAPI.createTrace(self, newObject, "CM2CGRepository", "foo", self, a);
    ___foo(newObject, self, a);
    return newObject;
}
public void ___foo(final Graph newObject, final EObject self, final String a) {
    newObject.setName("bar");
}
```
Basis for configuring the transformation:
model workflow engine 2 (MWE2)

Meta models as Java class hierarchies (EMF)

Already workflow components that allow loading of meta models and models and saving of models

New: ModuleLoader
⇒ loads modules, wiring, initiate transformation
Workflow example

```java
component = ModuleLoader {
    input = "inputSystem"

    output = {
        package = "cg"
        slot = "transformedModel"
    }

    config = {
        key = "providedRoleToCall"
        value = "Provided_IHTTP_DefaultMediaStoreSystem"
    }

    transformationModule = "trafo.impl.CM2CGRepositoryImpl"
    transformationModule = "trafo.impl.CM2CGSEFFImpl"
    transformationModule = "trafo.impl.CMLibraryImpl"
}
```
Extension methods

class CM2CGRepositoryImpl implements CM2CGRepository {
  @Inject extension CM2CGSEFF cm2cgseff
  @Inject extension CMLibrary cmLibrary
  ...

  override Graph toNet(...) {
    ...
    var innerContext = getInnerAssemblyContext(...)
    ...
  }
}

extension
method

Dominik Werle – Adding a Module Concept to the Model Transformation Language Xtend
September 19, 2013 33/22
Bibliography


Motivation
Maintenance tasks

Introducing variants
- introduce subclasses for Action
- different implementation variants separated in interchangeable modules
- information hiding needed