MDE Case Study: Using Model Transformations for UML and DSLs

Dennis Wagelaar
System and Software Engineering Lab

Vrije Universiteit Brussel
Use cases for model transformation

➢ Stepwise refinement of design abstractions
  ➢ Multiple alternative/interchangeable refinements for each abstraction

➢ Translation to different languages/formats
  ➢ Translate from one meta-model to another
  ➢ Convert from one repository to another

➢ Code generation
  ➢ Easy model navigation through direct meta-model access
Outline

➔ Case study: Instant Messenger (UML)
  – Goal: use the same ‘code base’ for all Java platforms

➔ Software architecture
  – Explains organisation of the software elements

➔ Build process roadmap
  – All steps involved to go from model to deployed software

➔ Evaluation
  – Experiences, recommendations and outlook
Case study: Instant Messenger
Software architecture

- PIM in UML
  - Using Java as Action Language

- Add-on features in separate UML models
  - Semi platform-dependent models that can be merged with the PIM

- PIM-PSM refinement transformations in ATL
  - Add bindings to platform-specific API

- Configuration Management using DSL
Instant Messenger: PIM (part)
Add-on features

User interfaces:
- AWT
- Swing
- LCDUI

Network protocols:
- PIM
- Local
- SMS
- Jabber
  - DefaultJabber
  - MEJabber

Jabber transports
# Refinement transformations

<table>
<thead>
<tr>
<th>AssociationAttributes</th>
<th>Java2AssociationAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessors</td>
<td>Java2Accessors</td>
</tr>
<tr>
<td>Observer</td>
<td>JavaObserver</td>
</tr>
<tr>
<td>Applet</td>
<td>MIDlet</td>
</tr>
<tr>
<td>Singleton</td>
<td></td>
</tr>
<tr>
<td>AsyncMethods</td>
<td></td>
</tr>
<tr>
<td>DataTypes</td>
<td>Java2DataTypes</td>
</tr>
</tbody>
</table>
Example: AssociationAttributes Transformation

```
module AssociationAttributes;
create ATTRIBUTES : UML refining IN : UML;
...
rule AssociationEndAttribute {
  from s : UML!AssociationEnd (s.isNavigable)
  to t : UML!Attribute {
    name <- s.name,
    owner <- s.navigableFrom(),
    type <- s.type(),
    visibility <- s.visibility,
    ownerScope <- s.targetScope,
    changeability <- s.changeability,
    initialValue <- v),
  v : OUTMODEL!Expression {
    language <- 'java',
    body <- s.instance()}
}
```
AssociationAttributes: Helpers

```python
helper context UML!AssociationEnd def : isSingle() : Boolean =
    self.multiplicity.range->select(r|r.upper<>1)->isEmpty();

helper context UML!AssociationEnd def : type() : UML!Classifier
    if self.isSingle() then
        self.participant
    else
        'java.util.Vector'.class()
    endif
endif
endif;

helper context UML!AssociationEnd def : instance() : String =
    if self.isSingle() then
        'null'
    else
        'new java.util.Vector()'
    endif;
```
Java2AssociationAttributes: Helpers

```java
helper context UML!AssociationEnd def : isSingle() : Boolean =
    self.multiplicity.range->select(r|r.upper<>1)->isEmpty();

helper context UML!AssociationEnd def : type() : UML!Classifier
    if self.isSingle() then
        self.participant
    else
        'java.util.List'.interface()
    endif
endif;

helper context UML!AssociationEnd def : instance() : String =
    if self.isSingle() then
        'null'
    else
        'new java.util.ArrayList()'
    endif;
```
query UMLtoJava = UML!Classifier.allInstances()->collect(e | 
    if e.ignore() then true
    else e.toFileString().writeTo(e.pathName())
    endif);

...

helper context UML!Classifier def : toFileString() : String =
    self.packageDecl() + self.importDecl() + '\n' +
    self.toString();

...
Helper context UML!Class def : toString() : String =
self.visibility() + self.isAbstract() + 'class ' + self.name +
self.extendsClause() + self.implementsClause() +
' {
' +
self.ownedElement->select(e | e.oclIsKindOf(UML!Classifier))->
iterate(e; acc : String = '' | acc + e.toString()) +
self.feature->select(f | f.oclIsKindOf(UML!Attribute))->
iterate(e; acc : String = '' | acc + e.toString()) +
self.feature->select(f | f.oclIsKindOf(UML!Method))->
iterate(e; acc : String = '' | acc + e.toString()) +
'}

...
Code generation: Interface

helper context UML!Interface def : toString() : String =
  self.visibility() + self.isAbstract() + 'interface ' +
  self.name + self.extendsClause() +
  ' {
  self.ownedElement->select(e|e.oclIsKindOf(UML!Classifier))->
          iterate(e; acc : String = '' | acc + e.toString()) +
  self.feature->select(f|f.oclIsKindOf(UML!Method))->
          iterate(e; acc : String = '' | acc + e.toString()) +
  '}

...
Configuration management

- Which features can be combined?
  - Example: LCDUI and AWT don’t compile together

- Which refinement transformations can be combined and in which order?
  - Example: don’t mix “Java2” and “Java1” variants

- Other issues: external resources, packaging, deployment, ...
Configuration DSL

- Domain-Specific Language defined in EMF
  - Each model in this language describes a configuration
  - Meta-model defines which models are valid
  - Ant files for invoking the model transformations can be generated from these configuration models with ATL
  - Meta-model can be split into *general* refinements and *specific* instant messenger features
What about platform dependencies?

- Not all features run on all platforms
- Not all API bindings generated by the refinement transformations run on all platforms
Platform dependencies (2/3)

- External model of platform + constraints
- Use platform constraint annotations in the DSL meta-model:

```mermaid
platform:/resource/uml1cs-transformations/metamodels/Refinements.ecore
refinements
  ContextConstraint
  RefinementConfiguration
    first : AssociationAttributesRefinement
  AssociationAttributesRefinement
  AssociationAttributes -> AssociationAttributesRefinement, Java1Refinement
  RefinementConstraints.owl#AssociationAttributesPlatform
  Java2AssociationAttributes -> AssociationAttributesRefinement, Java2Refinement
```
Platform dependencies (3/3)

➔ Context-Driven Development Toolkit:
  – Uses platform/context models expressed in OWL-DL
  – Uses DL reasoner (eg. Racer) for constraint-checking and context optimisation (best-match)
  – Leverages DSL meta-model annotations to validate and compare concrete configurations
  – Can be used at deploy-time to determine optimal product configuration that is still valid for the client platform
Build process roadmap

1. Validate DSL for target context/platform
2. Configure software product using DSL
3. Generate build script from configuration model
4. Transform models and generate code
5. Package and deploy software for download

Demo
Evaluation: Case study experiences (1/2)

- ATL can be used for real-world models
  - Execution speed is not optimal (esp. model merging)
  - Debugging tools are usable
  - Active support by developers

- Not all platform dependencies can be abstracted out easily
  - Creating design abstractions costs time
  - Alternative: use add-in platform-specific models
Evaluation: Case study experiences (2/2)

→ EMF meta-modelling language lacks power for complex configuration rules in DSL
  – Advanced rule validation can be done with a model transformation
Evaluation: Platform dependencies

Platform dependencies can be managed by an external tool (e.g. CDDToolkit)

- Use meta-model annotations to provide tool input
- Decreases platform testing/debugging effort
- Allows optimised deployment of product configurations
## Evaluation: Tool maturity

<table>
<thead>
<tr>
<th>Tool</th>
<th>Maturity</th>
<th>Audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>Stable</td>
<td>Java developers</td>
</tr>
<tr>
<td>EMF</td>
<td>Stable</td>
<td>Java modelling experts</td>
</tr>
<tr>
<td>ATL</td>
<td>Development</td>
<td>Modelling experts/researchers</td>
</tr>
<tr>
<td>CDDToolkit</td>
<td>Proof-of-concept</td>
<td>Researchers</td>
</tr>
</tbody>
</table>
Evaluation: Recommendations

➔ Use made-to-measure transformations
  – No superfluous functionality (improved performance)
  – No time-consuming fixing of generated models/code

➔ Use transformation bootstrapping
  – Use transformations to generate transformations consisting of repetitive code
  – Use transformations to generate complex build scripts
Evaluation: Outlook

- Adapt generated DSL editor to provide integrated access to:
  - Advanced model validation transformations
  - Platform dependency checking
  - Build file generation

- Translate often-used transformations to Java to improve build time
  - E.g. copying and merging transformations
Questions?

➢ More info on:
Spare slides...
DSL: Instant Messenger features

platform:/resource/uml1cs-instantmessenger-model/metamodels/InstantMessengerFeatures.ecore

- im
  - InstantMessagingClient
    - InstantMessengerConstraints.owl#InstantMessagingClientPlatform
      - network : Network
      - userInterface : UserInterface
      - refinementConfiguration : RefinementConfiguration
      - packaging : Packaging
      - target : EString
    - Jabber -> Network
    - DefaultJabber -> JabberTransport
    - MEJabber -> JabberTransport
    - SMS -> Network
    - Local -> Network
    - Swing -> UserInterface
    - AWT -> UserInterface
    - LCDUI -> UserInterface
    - Network
    - JabberTransport
    - UserInterface
DSL: Refinement transformations
DSL: Example configuration

platform:/resource/uml1cs-instantmessenger-model/configurations/default/default.ecore

**Instant Messaging Client default/applet/**
- Jabber
  - Default Jabber
- Local
- AWT

**Refinement Configuration**
- Association Attributes
  - Accessors
    - Java Observer
    - Singleton
  - Applet
    - Async Methods
    - Data Types
      - UM Lto Java ../../uml1cs-instantmessenger-default/src

Web Applet

platform:/resource/uml1cs-instantmessenger-model/metamodels/InstantMessengerFeaturesPlusRefinements.ecore