INTEROP-ESA – First International Conference on Interoperability of Enterprise Software and Applications, February 23-25 2005, Geneva Switzerland

An Empirical Study of the UML Model Transformation Tool (UMT)

Roy Grønmo, Jon Oldevik

SINTEF, Norway

The work reported in this paper is carried out in the context of MODELWARE, an EU IP-project in FP62003/IST/2.3.2.3

Research Challenge

- OMG's MDA focus has drawn the attention towards model-driven transformation tools
- The UML Model Transformation Tool is just one of many proposed approaches, languages and tools
- How to evaluate these languages/tools?
- *Is it possible to identify a set of fairly **objective** and **easily checkable** requirements that measures the quality of a model-driven transformation tool?*
- Tool-dependant vs. language-dependant testing
- Ease-of-use? Robustness? Pricing? etc.
- Narrowed model scope in this study: ~~MOF~~ ✍ UML

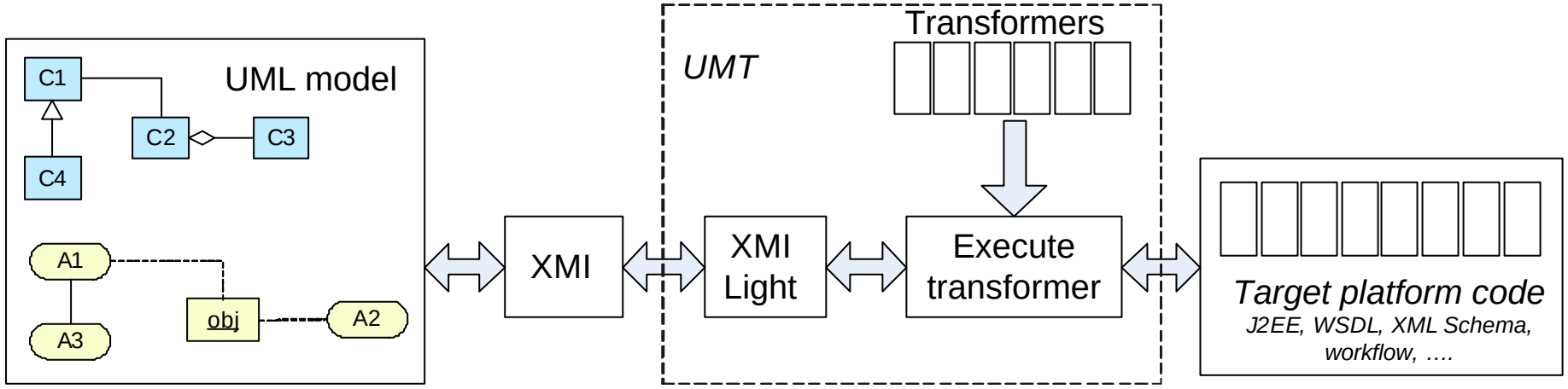
Quality of Transformation Language

- *Commonly-used language*. New, proprietary languages have a higher learning curve and require more effort to adopt.
- *Inheritance*. Easier to update and maintain transformations.
- *Graphical notation*. Most important for UML-to-UML. The graphical models provide a higher-level view on the transformation that is easier to communicate than the lexical counterpart.
- *Lexical notation*. Scales better than graphical. Handle complex or detailed transformations.
- *Declarative*. This makes the language side-effect free hopefully reduces code errors.
- *Bidirectional*. A language is bidirectional if it supports the definition of a transformation that is applicable two-ways. Easier specification and maintenance of bidirectional transformations.
- *XML support.* Built-in support to consume or produce XML, such as special support for handling general XML elements, attributes and namespaces. Important for text-to-UML and UML-to-text. Text formats nowadays often use XML.

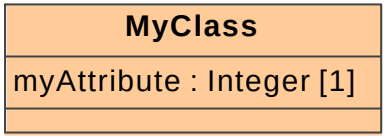
Tool Requirements

- Text-to-UML (reverse engineering). Legacy code, web services etc. exist without a relation to higher-level, graphical UML models
- UML-to-UML. Refinement of models, PIM-to-PSM, Transforming between different model views
- UML-to-text. Needed to implement running system as well as for documentation purposes.
- UML tool independence. Desirable so that the solution is not tied to a single UML tool.
- No proprietary intermediate structures. Such an additional structure increases the complexity for the transformation architect.
- Traceability. Explicit traces between source and target elements. Helps the user understand the transformation and its effects
- Metamodel-based. Source and target metamodels are explicitly defined and exploited in order to drive the transformation specification. Makes it easier to specify the transformation and to check that a transformation is valid.

UMT Architecture



Why have we invented XMI Light?



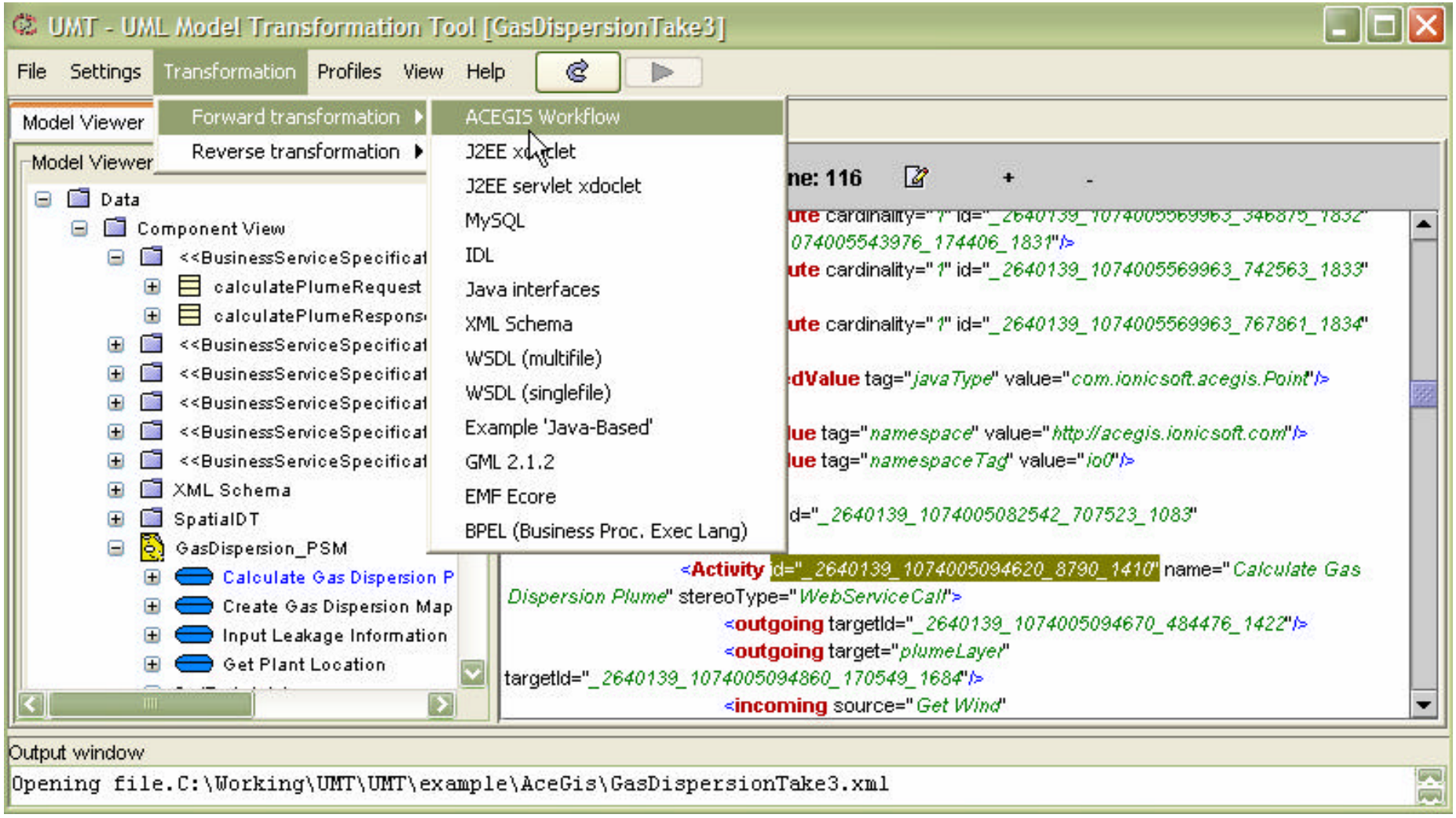
XMI Light

```
<datatype id="dt_Integer" name="Integer"/>
 <class abstract="false" id="class_MyClass" name="MyClass">
        <attribute cardinality="1..1" name="myAttribute" type="dt_Integer"/>
</class>
```

XMI 1.2

```
<UML:Class xmi.id = 'sm$1264eab:fe524af40b:-7fd4' name = 'MyClass' visibility = 'public'
        isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
        <UML:Classifier.feature>
          <UML:Attribute xmi.id = 'sm$1264eab:fe524af40b:-7fd5' name = 'myAttribute'
            visibility = 'public' isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable'>
            <UML:StructuralFeature.multiplicity>
              <UML:Multiplicity xmi.id = 'sm$1264eab:fe524af40b:-7fd6'>
                <UML:Multiplicity.range>
                  <UML:MultiplicityRange xmi.id = 'sm$1264eab:fe524af40b:-7fd7' lower = '1' upper = '1'/>
                </UML:Multiplicity.range>
              </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
            <UML:StructuralFeature.type>
              <UML:DataType xmi.idref = 'sm$1264eab:fe524af40b:-7fc7'/>
            </UML:StructuralFeature.type>
          </UML:Attribute>
        </UML:Classifier.feature>
      </UML:Class>
<UML:DataType xmi.id = 'sm$1264eab:fe524af40b:-7fc7' name = 'integer' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
```

UMT – Graphical User Interface

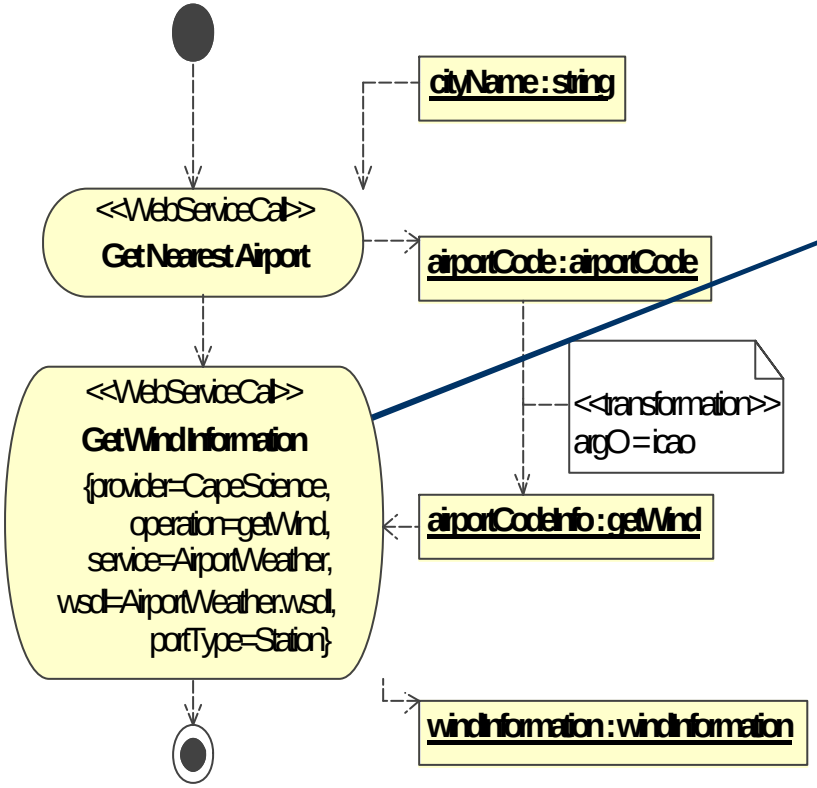


UMT – Transformation Kinds

- *UML-to-text transformation*.
  - UML models are exported to XMI from a UML tool.
  - XMI is converted to XMI Light by UMT.
  - **Write transformation** from XMI Light to the desired text or code format. Use XSLT or Java with XMI Light as input source
- *UML-to-UML transformation*.
  - UML models are exported to XMI from a UML tool.
  - XMI is converted to XMI Light by UMT.
  - **Write transformation** from XMI Light to XMI Light using XSLT/Java.
  - Convert XMI Light to XMI by UMT.
  - Import XMI into a UML tool
- *Text-to-UML transformation*.
  - **Write transformation** from text to XMI Light using Java (or XSLT).
  - Convert XMI Light to XMI by UMT.
  - Import XMI into a UML tool

Example: UML to BPEL —

Step 1: Producing XMI Light from UML (xmi)




```
<ActivityGraph name="UML2004">
        <PseudoState kind="initial"…/>
        <PseudoState kind="final"…/>
        <Activity name="Get Nearest Airport"…/>
        <ObjectFlow name="cityName"…/>
        <ObjectFlow name="windInformation"…/>
        <ObjectFlow name="airportCode"…/>
        <ObjectFlow name="airportCodeInfo…/>
<Activity name="Get Wind Information"
        stereoType="WebServiceCall">
    <outgoing targetId="final-ID"/>
    <outgoing target="windInformation"…/>
    <incoming sourceId="Get Nearest Airport-ID"/>
    <incoming source="airportCodeInfo"…/>
    <taggedValue tag="provider" value="CapeScience"/>
    <taggedValue tag="operation" value="getWind"/>
    <taggedValue tag="service" value="AirportWeather"/>
    <taggedValue tag="wsdl" value="AirportWeather.wsdl"/>
    <taggedValue tag="portType" value="Station"/>
</Activity>
</ActivityGraph>
```

Example: UML to BPEL —

Step 2: Producing BPEL from XMI Light

```
<stylesheet>
  <template name="ActivityGraph" .../>
  <template match="Activity">
    <choose>
      <when test="@stereoType='WebServiceCall'">
        <variable name="operation"
                  select="taggedValue[@tag='operation']/@value"/>
        <variable name="portType"
                  select="taggedValue[@tag='portType']/@value"/>
        <out:invoke partner="{$provider}-{$operation}"
                    portType="{$portType}"
                    operation="{$operation}">
          <call-template name="inputOutputDataObjects"/>
        </out:invoke>
        ...
    </choose>
        <call-template name="outgoingFlows">
          <with-param name="current" select="."/>
        </call-template>
  </template>
  <template name="genDataTransformations".../>
</stylesheet>
```

```
<process
  <sequence>
    <receive name="cityName"…/>
    <invoke operation="GetNearestAirport"…/>
    <invoke portType="Station"
                outputVariable="windStr"
                operation="getWind"
                inputVariable="airportCodeInfo"
                partner="CapeScience-getWind"/>
    <reply name="windInformation"…/>
  </sequence>
</process>
```

Transformation Languages/Tools

Requirements	UMT w/XSLT	ATL	YATL	BOTL	MOLA	UMLX	VMT
Commonly-used language	**yes**	no	no	no	no	no	partly
Inheritance	**no**	yes	no	no	no	no	yes
Graphical notation	**no**	no	no	yes	yes	yes	yes
Lexical notation	**yes**	yes	yes	no	no	no	yes
Declarative	**yes**	no	no	yes	no	yes	no
Bidirectional	**no**	no	no	yes	no	no	no
XML support	**yes**	no	no	no	no	no	no
Text-to-UML	**yes**	no	no	no	yes	no	no
UML-to-UML	**yes**	yes	yes	yes	yes	yes	yes
UML-to-text	**yes**	yes	no	no	no	no	no
UML tool independence	**yes**	yes	yes	yes	no	yes	no
No proprietary intermediate structures	**no**	yes	yes	yes	yes	yes	yes
Traceability	**no**	no	yes	no	no	yes	yes
Metamodel/MOF-based	**no**	yes	yes	yes	yes	yes	no

OMG Proposals: QVTMerge, QVT-Compuware/Sun, and MOFScript

Requirements	UMT w/XSLT	QVTMerge	QVT-Compuware/ Sun	MOFScript
Commonly-used language	yes	no	no	no
Inheritance	no	yes	yes	yes
Graphical notation	no	yes	yes	no
Lexical notation	yes	yes	yes	yes
Declarative	yes	yes/no	no	no
Bidirectional	no	yes	yes	no
XML support	yes	no	no	yes
Text-to-UML	yes	no	no	no
UML-to-UML	yes	yes	yes	no
UML-to-text	yes	no	no	yes
UML tool independence	yes	yes	yes	yes
No proprietary intermediate structures	no	yes	yes	yes
Traceability	no	yes	no	no
Metamodel/MOF-based	no	yes	yes	yes

Conclusions about UMT

- The three kinds of model transformations are unified in one tool/approach
- Proven success with reverse-engineering and code generation
- Drawbacks:
  - Not metamodel-based
  - No specialized model transformation language
  - Proprietary format: XMI Light
  - No graphical notation

Discussion

- Declarative vs. imperative
- Graphical vs. textual
- MDA community: ”XML, XMI and XSLT is too low level”
- OCL-based solutions
- Metamodel-based. Hard work to define the source and target metamodel ✍ metamodel repository?
- Most focus so far has been on model-to-model transformations

Related Work

- Watch closely: OMG's QVT and MOF Model to text
- Evaluations and recommendations for QVT submissions:
  - Gardner et al. *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard*. 2003.
  - Langlois et al. *THALES recommendations for the final OMG standard on Query / Views / Transformations*. 2003.
- Graphical composition of transformations – Sendall et al. *Supporting Model-to-Model Transformations: The VMT Approach*. 2003.
- "OCL quickly leads to complex query statements even for simple queries" – Stein et al. *A graphical notation to specify model queries for MDA transformations on UML models*. 2004

Future Work

- Evaluate QVT and Model-to-text submissions
- Detailing of the requirements
- How to measure – more details
- Different levels of support
- Assigning weights
- Required vs. optional
- Computation algorithm – overall score
- What about *Ease-of-use*? – Reference examples and case studies are needed

UMT is available at SourceForge

- Open Source License: Lesser GNU Public License
- Dowload: **umt-qvt.sourceforge.net**

- **Questions**