
Formalizing QVT-Relations to certify transformations at authoring-time

Miguel Garcia

<http://www.sts.tu-harburg.de/~mi.garcia>

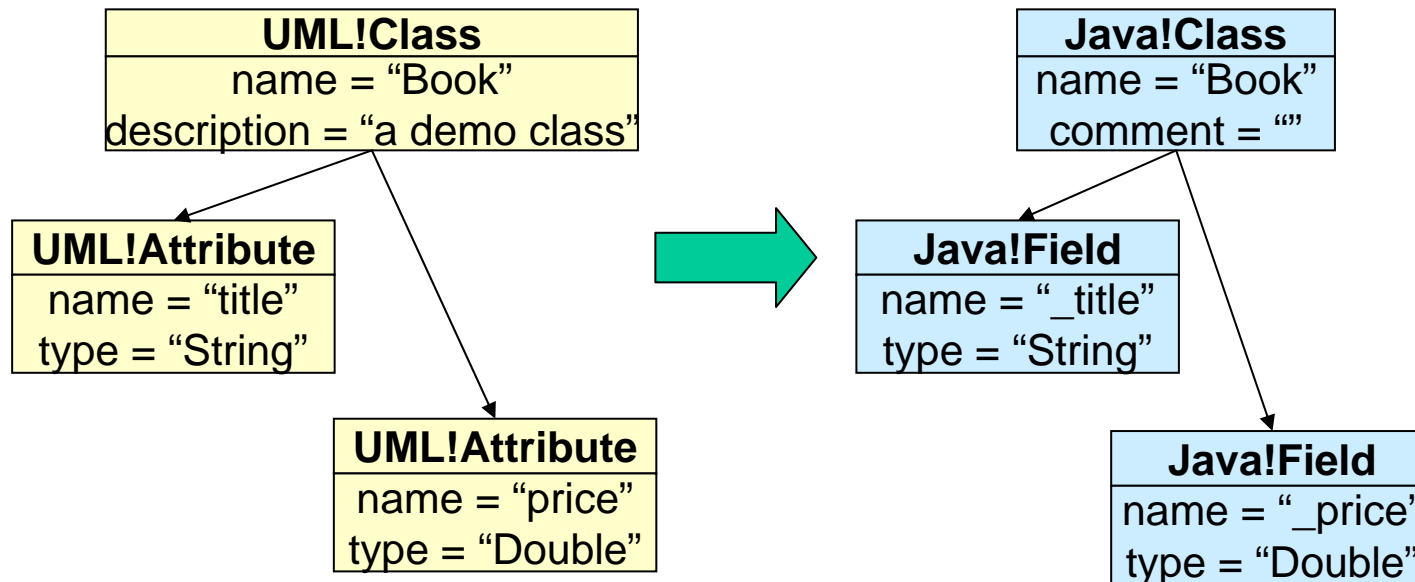
2008-09-16



Agenda

- **Industrial-strength model transformations (or, why testing alone won't cut it)**
- **QVT-Relations in a nutshell**
- **Finding bugs in the UML2RDBMS sample transformation, tooling used**
- **What others are doing, Future work**

Some model transformations are “straightforward to get right”

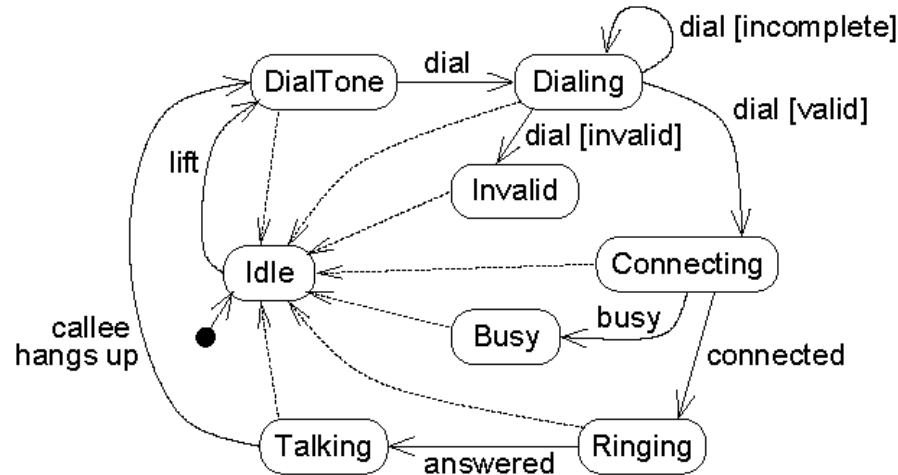


(because they bridge a short gap between input and output languages)

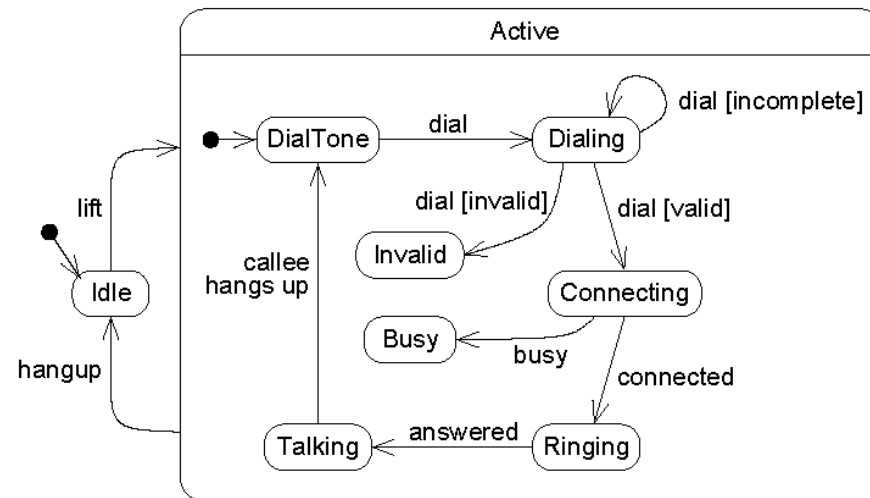
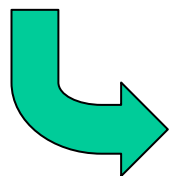
(but that's not representative of real-world transformations)

Reproduced from: Yingfei Xiong, *Towards Automatic Model Synchronization from Model Transformations*,
<http://www.ipl.t.u-tokyo.ac.jp/~xiong/papers/ASE07.ppt>

Other transformations are more like refactorings, with non-trivial preconditions and effects



Group shared states



Reproduced from: Sunyé, G., Pollet, D., Traon, Y. L., and Jézéquel, J. Refactoring UML Models. In Proc 4th Intl Conf on UML, 2001. LNCS 2185. Springer-Verlag, London, 134-148.

Moreover, a toolchain may comprise transformations developed independently

- For example, a statechart may be fed as input to the transformation “*Express state machine in pre/post constraints*” [1]
- That’s for example what AutoJML [2] does:

AutoJML is a JML specification generator. It generates specifications based on other, higher level, specification formalisms such as UML state diagrams, or security protocol specifications. The output is a combination of Java skeleton code and JML class and method specifications. JML stands for the Java Modeling Language.

- **It would be great to make sure that the transformations in a pipeline collaborate as expected, much like each compilation phase (in a 3GL compiler) fulfills its contract.**

[1] Lano, K. Catalogue of Model Transformations. <http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf>

[2] <http://autojml.sourceforge.net/>

Finally, a transformation should take into account dependent software artefacts

- Refactoring OCL annotated UML class diagrams [3]
- Rewriting queries to account for schema evolution (impact analysis upon logical schema changes) [4]

UML refactorings and their impact on the textual syntax of existing OCL constraints [3]

<i>RenameClass</i>	Rename context
<i>RenameAttribute</i>	Rename usages
RenameOperation	Rename usages
RenameAssociationEnd	Rename usages
PullUpAttribute	No impact
PullUpOperation	No impact
PullUpAssociationEnd	No impact
PushDownAttribute (single target)	Relocate OCL constraint
PushDownOperation (single target)	Relocate OCL constraint
PushDownAssociationEnd (single target)	Relocate OCL constraint
ExtractClass	No impact
ExtractSuperclass	No impact
MoveAttribute	Forward navigation
MoveOperation	Forward navigation
MoveAssociationEnd	Forward navigation

[3] Slaviša, M., Baar, T.
Refactoring OCL Annotated UML Class Diagrams.
In: Software and Systems Modeling (SoSyM),
vol. 7, num. 1, 2008, p. 25-47

[4] Andy Maule, Wolfgang Emmerich, and David S. Rosenblum. Impact analysis of database schema changes.
In ICSE '08: Proc. of the 30th Intl Conf on Software Engineering, pages 451–460, New York, NY, USA, 2008. ACM.
<http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/ICSE2008/schemaChangeICSE08/icse08.pdf>

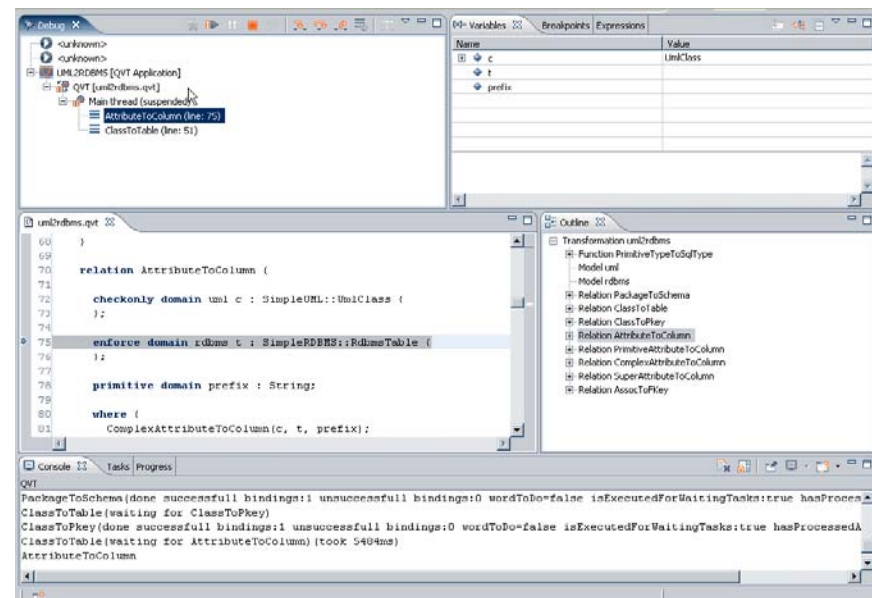
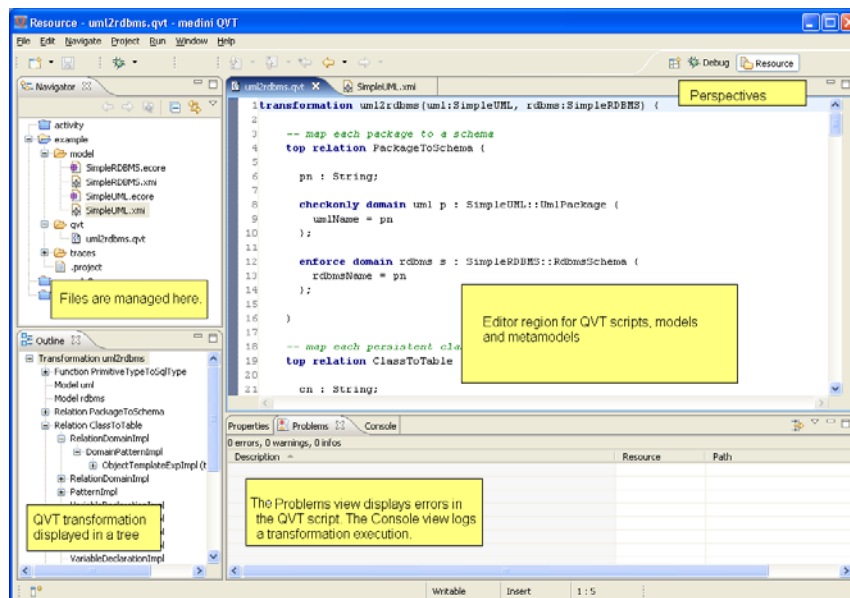
Many opportunities to overlook special cases, resulting in non-well-formed models

- Specially when the output is the Abstract Syntax Tree of programming language code (a particular case of model transformation)
 - Undeclared local variables
 - Unintended hiding (parameter name vs. class field, for example)
 - Type errors
 - API usage violations
- You may have heard of “tracing”, “visual debugging of transformations”, and so on (manual effort)
- What about having instead an automatic procedure to

**Obtain the assurance at authoring-time
that the transformation being analyzed
will produce valid output,
for all possible runtime executions on valid input**

QVT-Relations pros ...

- Dedicated syntax for object pattern matching
- Language-aware IDE support
 - tracing (depicts which objects match which patterns)
 - step debugging (the ups and downs of backtracking)
 - other goodies (Content Assist, AST view, navigation)



Screen captures reproduced from medini QVT technical documentation

... and cons

- **Other languages realize pattern matching without new constructs**
 - Scala provides it out of the box [5]
 - Java libraries exist to encapsulate that functionality (e.g. MatchO [6])
 - Manually implementing pattern matching is cumbersome because of backtracking (involving undoing tentative matches upon running out of choices)
- **If QVT-Relations is so good at pattern matching, then why are visitors equally concise?**
 - The case distinctions performed by the tree-walker code in a visitor require nested `if-then-else-endif` in QVT-Relations, one for each branching point in the AST inheritance hierarchy of the DSL being matched

[5] Burak Emir, Martin Odersky, and John Williams. Matching Objects With Patterns. Tech report, January 2007. <http://lamp.epfl.ch/~emir/written/MatchingObjectsWithPatterns-TR.pdf>

[6] Joost Visser: "Matching Objects Without Language Extension", in Journal of Object Technology, vol. 5, no. 8, November-December 2006, pages 81-100, http://www.jot.fm/issues/issue_2006_11/article2

Every DSL should have its WFRs formulated in a machine-processable language (e.g., OCL)

A bird's eye view of the static semantics of QVTR

```
context QVTTemplate::PropertyTemplateItem
inv property_part_of_class:
self.objContainer.referredClass.eAllStructuralFeatures
->includes(self.referredProperty)
```

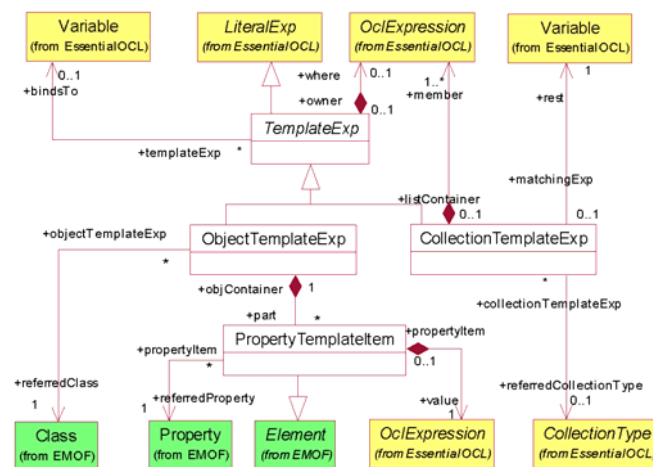
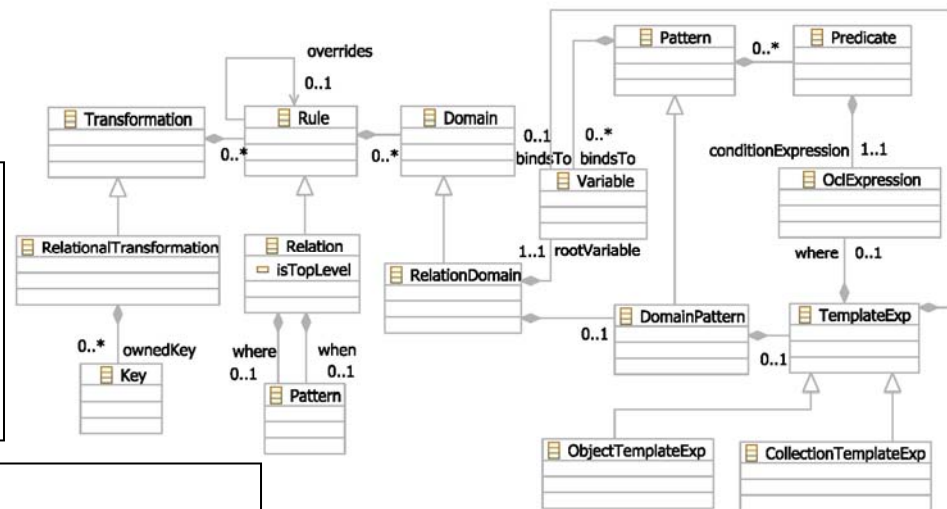
```
context QVTTemplate::ObjectTemplateExp
inv bindingVariableTypeConformance:
not bindsTo->isEmpty()
implies assignmentCompatible(bindsTo.eType, referredClass)

context QVTRelation::DomainPattern
inv varOfRootTemplateEqRootVarOfRelDomain:
not templateExpression->isEmpty()
implies templateExpression.bindsTo =
relationDomain.rootVariable
```

```
context QVTRelation::RelationCallExp
inv actualFormalsConformance :
argument->size () = referredRelation.domains!size ()
and
argument->forall ( arg |
let i : Integer = self.argument!indexOf ( arg ) in
assignmentCompatible(referredRelation . domain->at(i).eType, arg.eType ))
```

```
context QVTRelation::Relation
inv noInvocationsToTopLevels:
not self.where->isEmpty ()
implies
self.where.allRelationInvocations()->forall ( ri |
ri.referredRelation.isTopLevel = false )

context QVTRelation::DomainPattern
def allRelationInvocations () : Set ( RelationCallExp ) =
self.oclAsType(Pattern).allRelationInvocations()->union (
collectRelationInvocations(templateExpression.where))
```



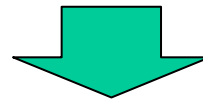
Our tool of choice to certify QVTR transformations: Alloy

- Alloy (<http://alloy.mit.edu/>) allows declaring “possible worlds” consisting of mathematical relations connecting atomic symbols
- Three kinds of automatic analyses are possible:
 - *Model Finding*, whose output are (visual) depictions of concrete worlds that satisfy the specified constraints
 - Assertions can be given, which are claimed to follow from the rest of the spec. *Counterexample Finding* reveals concrete worlds that are conformant save for the broken assertion
 - For unsatisfiable predicates, *Unsat Core* can be used to highlight the relevant portions of the Alloy spec that contributed to unsatisfiability

QVTR snippet, Alloy snippet

```
relation AttributeToColumn
{
  checkonly domain uml c:Class {};
  enforce domain rdbms t:Table {};
  primitive domain prefix:String;
  where {
    PrimitiveAttributeToColumn(c, t, prefix);
    ComplexAttributeToColumn(c, t, prefix);
    SuperAttributeToColumn(c, t, prefix);
  }
}
```

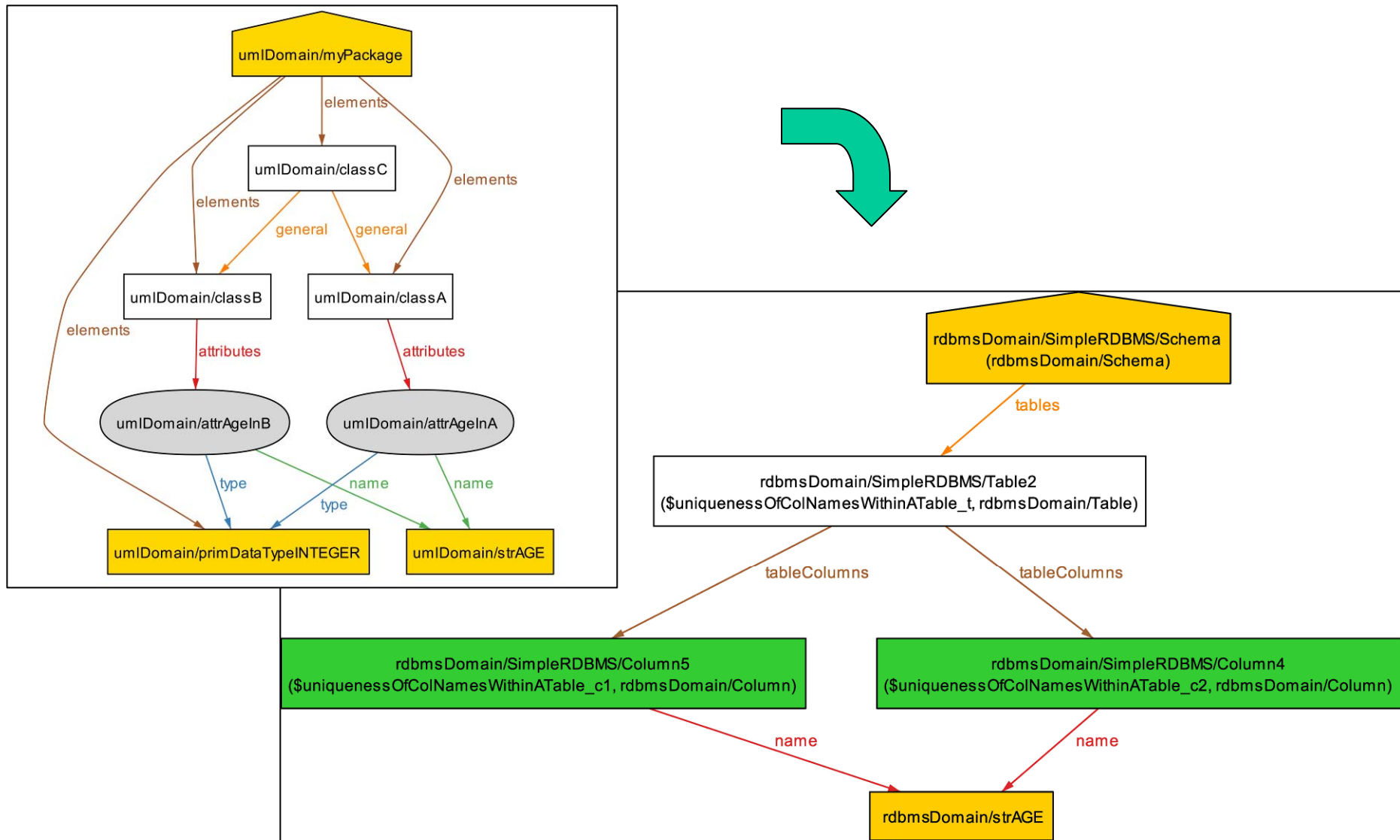
```
relation SuperAttributeToColumn
{
  checkonly domain uml c:Class
  {general=sc:Class {}};
  enforce domain rdbms t:Table {};
  primitive domain prefix:String;
  where {
    AttributeToColumn(sc, t, prefix);
  }
}
```



```
pred AttributeToColumn[c:umlDomain/Class, t:rdbmsDomain/Table] {
/* given that the string prefix being received as argument is constantly empty,
it has been optimized away by applying the constant propagation optimization */
PrimitiveAttributeToColumn[c,t]
-- assume for now no ComplexAttributeToColumn[c,t]
SuperAttributeToColumn[c,t]
-- no other columns other than supported by the above conditions
t.tableColumns.originatingClass = { cWithAttr:c.(*general) | some
cWithAttr.attributes }
-- all col:t.tableColumns | col.name in col.originatingClass.attributes.name
}

pred SuperAttributeToColumn[c:umlDomain/Class, t:rdbmsDomain/Table] {
  all superC:c.(^general) | PrimitiveAttributeToColumn[superC, t]
}
```

Counterexample for output well-formedness of the UML2RDBMS transformation



Ideally, the validation of model transformations would be supported by seamless tooling

As of now it's not seamless, but several building blocks are in place

OCL Editors

1. <http://squam.info/ocleditor>
2. <http://www.sts.tu-harburg.de/~mi.garcia/pubs/2007/ocltools/OCLTools.pdf>

OCL to Alloy

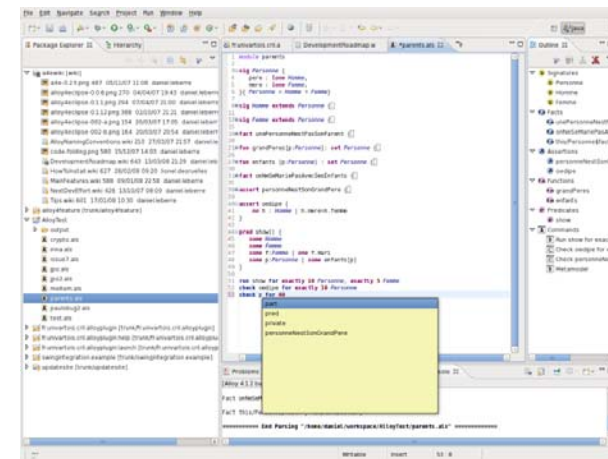
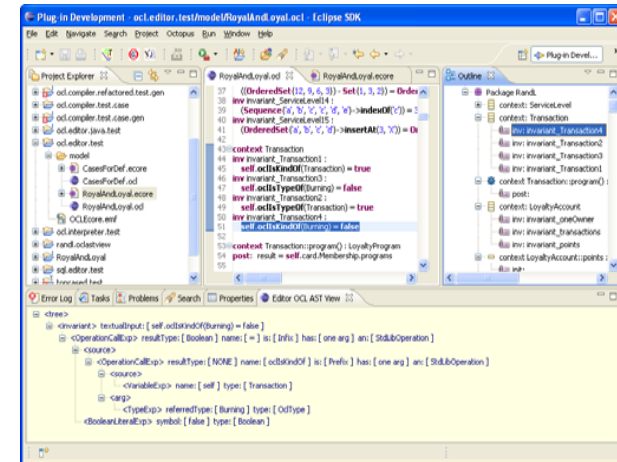
<http://www.cs.bham.ac.uk/~bxb/UML2Alloy>

Eclipse plugin for Alloy

<http://code.google.com/p/alloy4eclipse/>

QVT-Relations to Alloy

Would you like to contribute to this project? ☺



What others are doing (general theme: moving from research into practice)

- **ICMT2008 - International Conference on Model Transformation**
<http://www.model-transformation.org/ICMT2008>
- **twomde'08 - Transformation and Weaving Ontologies in Model Driven Engineering**
<http://isweb.uni-koblenz.de/events/TWOMDE2008>
- **MoDeVVA 2008 - Model Driven Engineering, Verification, and Validation**
<http://www.cs.colostate.edu/~ghosh/modevva2008/>
(previous editions at <http://www.modeva.org/2007/>)

Future Work for the MDSD community

- Establishing a repository of reusable, certified transformations
- Agreement on having DSL authors also define visual syntax in tandem with abstract and textual syntax (thus allowing automatic generation of IDEs)
- Improving the performance of model repositories
Have you heard about LINQ4EMF? 😊
<http://www.sts.tu-harburg.de/~mi.garcia/pubs/2008/ese/linq4emf.pdf>

Yes, you're part of the MDSD community!