Implementing QVT-R Bidirectional Model Transformations using Alloy

Nuno Macedo Alcino Cunha

HASLab — High Assurance Software Laboratory
INESC TEC & Universidade do Minho, Braga, Portugal

HASLab Seminar Series
February 27, 2013, Braga

Introduction

- In model-driven engineering models are the primary development artifact;
- Models with possibly overlapping information which must be consistent;
- OMG has proposed standards for the specification of models (UML) and constraints over them (OCL);
- The *QVT* (Query/View/Transformation) language has been proposed to specify *bidirectional transformations* (BX).

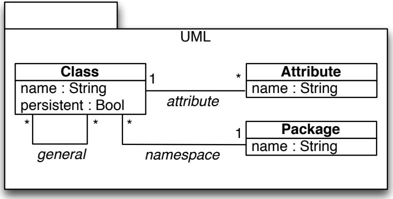
Query/View/Transformation

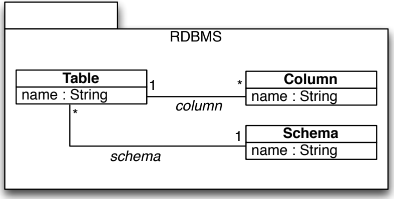
- The QVT standard proposes three different specification languages;
- We focus on *QVT Relations* (QVT-R), a declarative language;
- Specifications denote relations between elements of the model;
- Two running modes:
    - *checkonly* mode (checks consistency);
    - *enforce* mode (propagates updates in order to restore consistency).

QVT Relations

- A QVT-R *transformation* consists of set of QVT-R *relations* between elements of the models;
- In each relation there is a set of *domain patterns* that specify related elements;
- It may also contain *When* and *Where* constraints, that act as pre- and post-conditions.

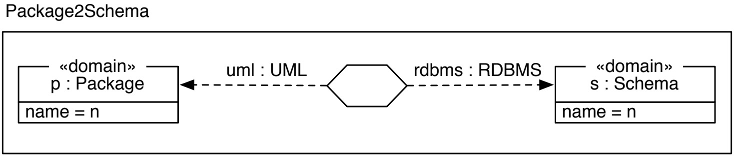
Example: object/relational mapping



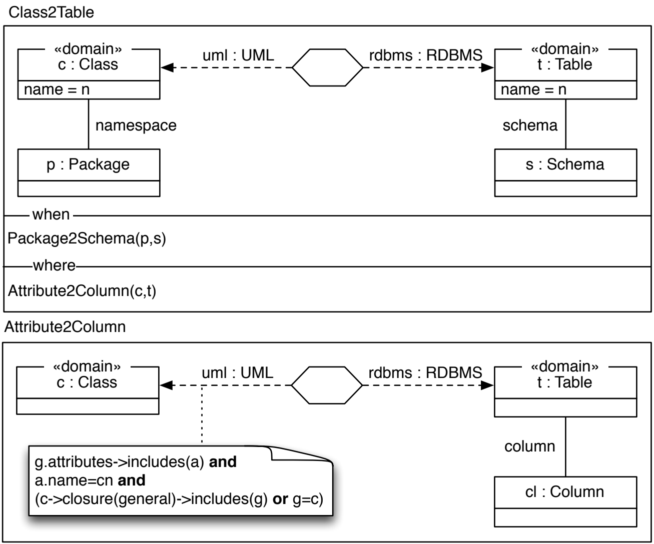


Example: object/relational mapping

```
top relation Package2Schema {
    n:String;
    domain uml p:Package {
        name=n };
    domain rdbms s:Schema {
        name=n };
}
```



Example: object/relational mapping



Bidirectional Transformations

- BXs are artifacts that represent the transformations in both directions;
- These need to be inferred from a single QVT-R specification;
- Since models may contain different information, they are *not bijective*;
- Propagating updates from a source to a new target retrieves information from the *original* target.

Properties

- For every QVT transformation T between M and N we have:
  - a relation $\mathbf{T} : M \to N$ that checks the consistency;
  - transformations $\overrightarrow{\mathbf{T}} : M \times N \to N$ and $\overleftarrow{\mathbf{T}} : M \times N \to M$ that propagate updates;
- These artifacts are also inferred at the QVT relation level, between elements of the models;
- For every metamodel M, we have a function $\Delta_M : M \times M \to \mathbb{N}$ that calculates the distance between instances.

Properties

- Correctness:

$$\forall\, m \in M, n \in N : m \;\mathbf{T}\; (\overrightarrow{\mathbf{T}}\,(m, n))$$
$$\forall\, m \in M, n \in N : (\overleftarrow{\mathbf{T}}\,(m, n)) \;\mathbf{T}\; n$$

- Hippocraticness (check-before-enforce):

$$\forall\, m \in M, n \in N : m \;\mathbf{T}\; n \Rightarrow m = \overrightarrow{\mathbf{T}}\,(m, n) \wedge n = \overleftarrow{\mathbf{T}}\,(m, n)$$

Properties

- Correctness:

$$\forall\, m \in M, n \in N : m \,\mathbf{T}\, (\overrightarrow{\mathbf{T}}\,(m,n))$$
$$\forall\, m \in M, n \in N : (\overleftarrow{\mathbf{T}}\,(m,n)) \,\mathbf{T}\, n$$

- Hippocraticness (check-before-enforce):

$$\forall\, m \in M, n \in N : m \,\mathbf{T}\, n \Rightarrow m = \overrightarrow{\mathbf{T}}\,(m,n) \wedge n = \overleftarrow{\mathbf{T}}\,(m,n)$$

- Principle of least change ($\Rightarrow$ hippocraticness for $\Delta = 0$):

$$\forall\, m \in M, n, n' \in N : m \,\mathbf{T}\, n' \Rightarrow \Delta_N\,(\overrightarrow{\mathbf{T}}\,(m,n), n) \leqslant \Delta_N\,(n', n)$$
$$\forall\, m, m' \in M, n \in N : m' \,\mathbf{T}\, n \Rightarrow \Delta_M\,(\overleftarrow{\mathbf{T}}\,(m,n), m) \leqslant \Delta_M\,(m', m)$$

QVT-R Semantics

Adoption of QVT as a standard has been slow:

- The standard is ambiguous and incomplete regarding semantics;
- Tools implement different interpretations or disregard it at all;
- Some work on the formalization of the check semantics has been done...
- ...but not on the formalization of enforce semantics;
- No tool has support for enforce mode over models with OCL constraints.

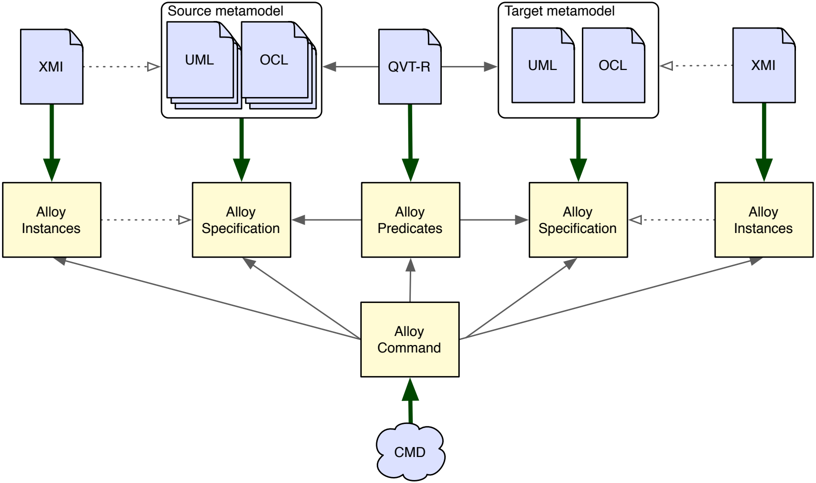
QVT-R Semantics

- The semantics of a QVT transformation consist of running its constituent QVT relations;
- *Check semantics*: for all candidate elements in the domain there must exist a candidate element in the target that matches it;
- *Enforce semantics*:
  - if there are no *keys*, create matching elements and delete unbound ones;
  - the standard enforces strong syntactic restrictions to guarantee determinism;

Alloy

- Alloy is a lightweight model-checking tool based on relational calculus;
- Allows automatic bounded verification of properties and generation of instances via SAT solving;
- We have already developed a tool for the transformation of UML+OCL class models to Alloy;
- Building up on that, we propose the translation of QVT-R to Alloy.

QVT to Alloy Translation



Models

- UML classes and their attributes are directly translated to Alloy signatures and relations;

```
sig Class {
    class : set UML,
    attribute : Attribute -> UML,
    general : Class -> UML,
    ...
}
```

- Alloy is static, so we resort to the local state idiom;
- OCL is also translated to constraints in Alloy.

Transformations: Check semantics

- We follow the check semantics of the standard;
- Each domain pattern produces a predicate in Alloy that represents candidate elements;

```
pred Pattern_P2S_UML [m:UML,p:Package,n:String] {
    n in p.name.m }
```

- These are then used in a forall-there-exists test;

```
pred Top_P2S_RDBMS [m:UML,n:RDBMS] {
    all p:package.m, n:String | Pattern_P2S_UML[m,p,n] =>
        some s:schema.n | Pattern_P2S_RDBMS[n,t,n,s] }
```

- These tests are *directional* (a dual `Top_P2S_UML` is defined).

Transformations: Enforce semantics

- We follow the principle of least change, applying the smallest possible update;
- To do so, we need to calculate the distance Δ between Alloy models;
- Non-deterministic (for $\Delta \neq 0$);
- Two alternatives:
  - graph edit distance (GED);
  - operation sequence.

Transformations: Enforce semantics

- Since Alloy atoms are mainly uninterpreted, GED is a natural distance;
- Counts the addition, deletion, or relabeling of a vertex or edge;

```
fun Delta_UML [m,m':UML] : Int {
    (#((class.m-class.m')+(class.m'-class.m))).plus[
    (#((name.m-name.m')+(name.m'-name.m))).plus[...]] }
```

- General, but “oblivious” metric.

Transformations: Enforce semantics

- UML class diagrams can be enhanced with *operations* over classes;
- We can count the number of operations required two reach a given model;

```
fact { all m:UML, m':m.next | {
   some p:package.m, n:String | setName[p,n,m,m'] or
   some p:package.m, n:String | addClass[p,n,m,m'] or
   ... } }
```

- Edit cost is parametrized but operations must be defined.

Instances

- Object instances are represented in Alloy as singleton sets belonging to the signature representing its type;

```
one sig M extends UML {}
one sig P extends Package {}
one sig A,B extends Class {}
```

- Their attributes can be simply defined as relations between those signatures;

```
fact { class.M = A + B && package.M = P &&
       namespace.M = A->P + B->P &&
       ... }
```

Execution: Checkonly mode

- Runs the checks in all directions;

```
pred Uml2Rdbms [m:UML,n:RDBMS]{
    Top_P2S_RDBMS[m,n] && Top_P2S_UML[m,n] &&
    Top_C2T_RDBMS[m,n] && Top_C2T_UML[m,n] }
```

- The scope is the number of existing elements;

```
check { Uml2Rdbms[Src,Trg] }
for 0 but 1 Schema, 1 Table, 2 Column,
          1 Package, 2 Class, 2 Attribute
```

Execution: Enforce mode

- Asks for consistent models by increasing distance Δ;
- The scope is the number of existing elements plus Δ on the elements of the target model;

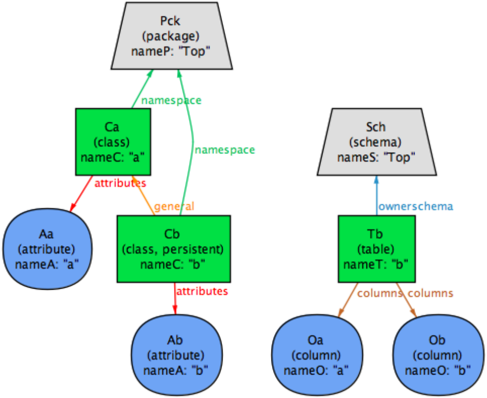
```
run { Uml2Rdbms[Src,Trg'] && Dist_UML[Trg,Trg'] = 1 }
for 0 but 1 Schema, 1 Table, 3 Column,
          2 Package, 3 Class, 3 Attribute, 2 Int
```

- Guarantees the properties by construction.

Recursion

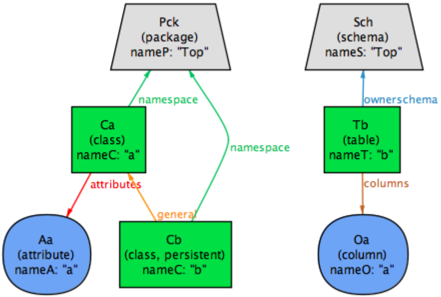
- Recursion calls must not be circular;
- This is usually not desired by programmers;
- However, we can resort to the transitive closure (which has recently been added to the OCL standard);
- We were able to rewrite the classic recursive QVT examples to use the transitive closure.

Example: Check mode



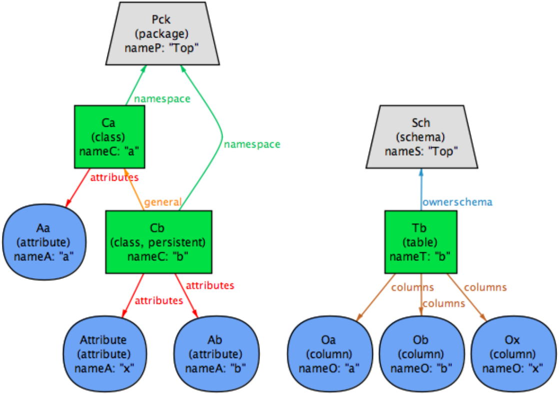

```
check { Uml2Rdbms[Src,Trg'] }
for 0 but 1 Schema, 1 Table, 3 Column,
          1 Package, 2 Class, 2 Attribute
```

Example: Enforce mode



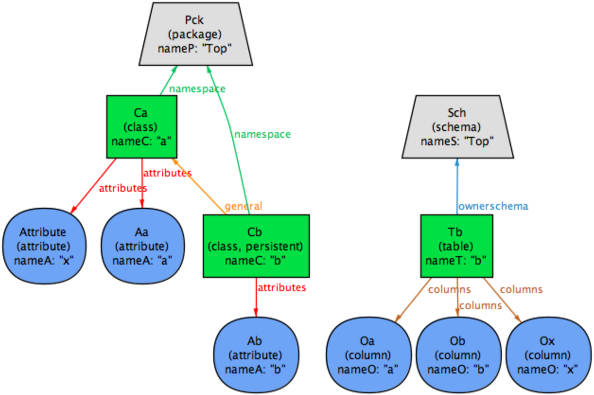

```
run { Uml2Rdbms[Src,Trg'] && Dist_UML[Trg,Trg'] = 3 }
for 0 but 1 Schema, 1 Table, 3 Column,
          4 Package, 5 Class, 5 Attribute, 3 Int
```

Example: Non-deterministic




```
run { Uml2Rdbms[Src,Trg'] && Dist_UML[Trg,Trg'] = 3 }
for 0 but 1 Schema, 1 Table, 3 Column,
          4 Package, 5 Class, 5 Attribute, 3 Int
```

Example: Non-deterministic




```
run { Uml2Rdbms[Src,Trg'] && Dist_UML[Trg,Trg'] = 3 }
for 0 but 1 Schema, 1 Table, 3 Column,
          4 Package, 5 Class, 5 Attribute, 3 Int
```

Conclusions

- We propose a BX framework for QVT where both the models and the transformation can be annotated with generic OCL.
- Functional implementation available at
  http://github.com/alcinocunha/echo

- Working on optimization (by simplifying the predicates and inferring further restrictions from the specifications);
- Studying a generic mechanism to detect and deal with circular recursion (either by resorting to the transitive closure or not).