Implementing QVT-R Bidirectional Model Transformations using Alloy

Nuno Macedo Alcino Cunha

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

Universidade do Minho

FASE 2013
March 22, Rome, Italy

Introduction

- In model-driven engineering models are the primary development artifact;
- Several models must coexist in a consistent manner;
- OMG has proposed standards for the specification of models (UML) and constraints over them (OCL);
- The *QVT* (Query/View/Transformation) standard has been proposed to specify model transformations and consistency.

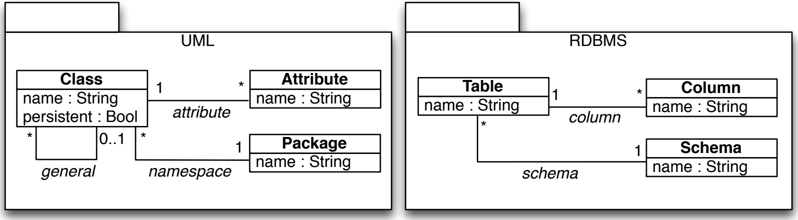
Query/View/Transformation

- The QVT standard proposes three different languages;
- We focus on *QVT Relations* (QVT-R);
- Declarative language where the specification denotes the consistency relation between models;
- Two running modes should be derived:
  - *checkonly* mode (checks consistency);
  - *enforce* mode (updates are propagated in one direction 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.

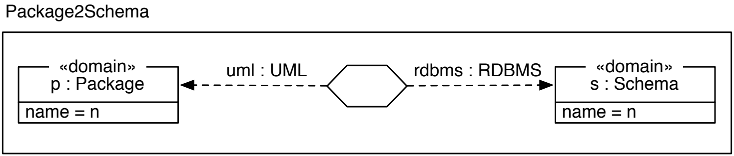
Example: object/relational mapping



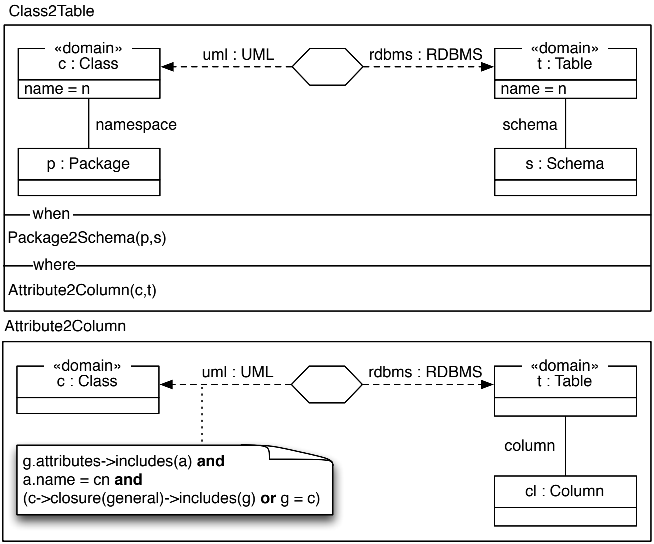

```
context Class inv:
  not self.closure(general)->includes(self)
```

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

- Since updates can be propagated to either model, enforce mode entails a *bidirectional transformation* (BX);
- These need to be inferred from a single QVT-R specification;
- Since models may contain different information, they are *not bijective*;
- The exact edit-sequence of the update is unknown;
- Information from the *original* target model must be retrieved.

QVT-R Semantics

Adoption of QVT as a standard has been slow:

- The standard is ambiguous and incomplete regarding semantics;
- Some work on the formalization of the checking semantics has been done...
- ...but not on the formalization of enforcement semantics (at least until the paper just presented!);
- Tools implement different interpretations or disregard it at all;
- No tool has support for enforce mode over metamodels with OCL constraints.

QVT-R Checking semantics

- *Checking*: for all candidates in the source there must exist a candidate in the target that matches it;
- The standard is omissive about what should happen in circular recursion;
- We chose *not* to allow circular recursion;
- 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-R examples to use the transitive closure.

QVT-R Enforcement semantics

- *Enforcement*:
  - if *keys* are defined, update the matching object;
  - otherwise, create matching elements and delete unbound ones;
- The standard enforces strong syntactic restrictions to guarantee determinism;
- Writing BX with the expected behavior becomes difficult (not even the example from the standard is bidirectional!);
- Deterministic but unpredictable: without keys, new elements are always created, discarding existing ones;
- Disregards the OCL constraints of the metamodel;
- Instead, we follow the clear and predictable *principle of least change*.

Formalization

- For every QVT-R transformation T between M and N we have:
  - a relation $\boldsymbol{T} \subseteq M \times N$ that checks the consistency;
  - transformations $\overrightarrow{\boldsymbol{T}} : M \times N \to N$ and $\overleftarrow{\boldsymbol{T}} : M \times N \to M$ that propagate updates;
- For every metamodel M, we have a function $\Delta_M : M \times M \to \mathbb{N}$ that calculates the distance between instances.

Formalization

- 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):

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

Formalization

- Correctness:

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

- Hippocraticness (check-before-enforce):

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

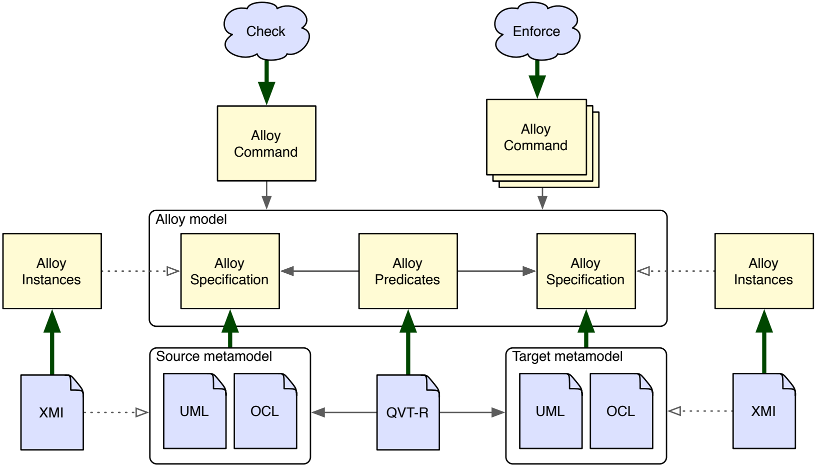
- Principle of least change ($\Rightarrow$ hippocraticness for $\Delta = 0$):

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

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 diagrams to Alloy;
- Building up on that, we propose the translation of QVT-R to Alloy.

QVT-R 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 annotations are translated to Alloy constraints.

Transformations: Checking semantics

- 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: Enforcement semantics

- We follow the principle of least change by incrementally asking for the smallest updates;
- We need to calculate the distance Δ between Alloy models;
- Non-deterministic (for $\Delta \neq 0$);
- Two alternatives:
    - graph edit distance (GED);
    - parametrized edit distance.

Transformations: Enforcement semantics

- Since Alloy atoms are mainly uninterpreted, GED is a natural distance;
- Counts the addition and deletion of vertices and edges;

```
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;
- Hard to control the behavior and non-determinism.

Transformations: Enforcement semantics

- UML class diagrams can be enhanced with edit *operations* specified in OCL;
- We can infer the number of operations required to reach a consistent 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
   ... } }
```

- Finer control over distance but with the overhead of defining operations in OCL.

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
```

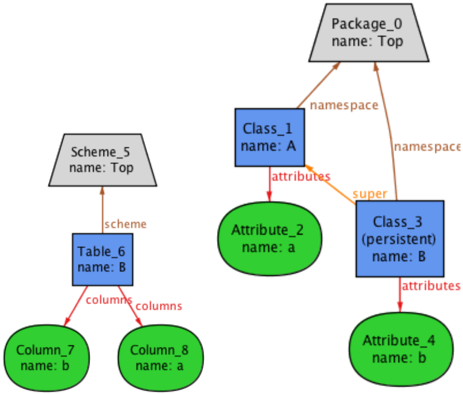
Execution: Enforce mode

- 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'] = Δ }
for 0 but 1 Schema, 1 Table, 3 Column, ⌈log(Δ+1)+1⌉ Int,
          (1+Δ) Package, (2+Δ) Class, (2+Δ) Attribute
```

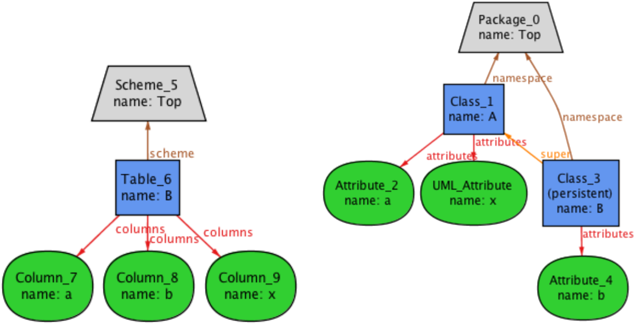
- Guarantees the properties by construction.

Example: Check mode



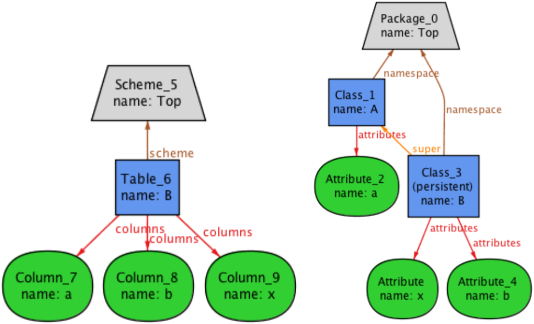

```
check { Uml2Rdbms[Src,Trg] }
for 0 but 1 Schema, 1 Table, 3 Column,
          1 Package, 2 Class, 2 Attribute
```

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: 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
```

Conclusions

- We propose a BX framework for QVT-R with clear semantics where both the metamodels and the transformations can be annotated with unrestricted OCL;
- Implementation over the Eclipse Modeling Framework (EMF) available at `http://github.com/haslab/echo`;

- Working on optimization (by simplifying the Alloy 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).