Model-Driven Development

Episode 16

Another Take on M2M TX

Andrzej Wąsowski

Rolf-Helge Pfeiffer

IT UNIVERSITY OF COPENHAGEN

PROCESS AND SYSTEM MODELS GROUP

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

So far, we have ...

- Argued for usefulness of MDD in software development
- Used class diagrams to define abstract syntax of DSLs.
- Used OCL to write more intricate structural constraints.
- Designed a DSL for Junit's Assert class.
- Defined and implement concrete (textual) syntax.
- Implemented a code generator with Xtend.
- Started project work

Model Transformation

image by nomen.nescio of flickr

Besides code generation, M2M transformations often appear in projects
Today we talk about rule based M2M Transformations.

- Bidirectional M2M TX with QVT Relations
- Unidirectional Rule-based M2M TX with ATL

AGENDA

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

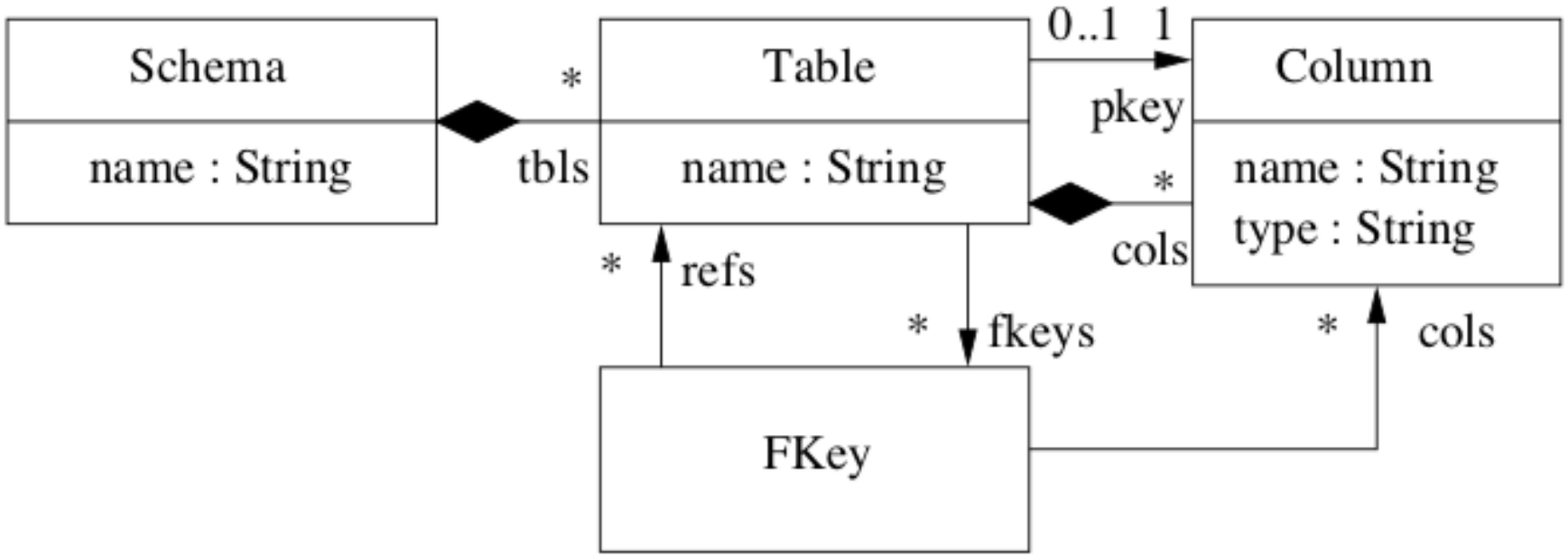
- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

QVT Relations

- An OMG standard for expressing transformations
- Declarative
- Rule-based:
  - Patterns are matched in the source meta-model
  - A rule is triggered when source pattern matches
  - Code to create target model elements is executed.
- Let's see a small example ...

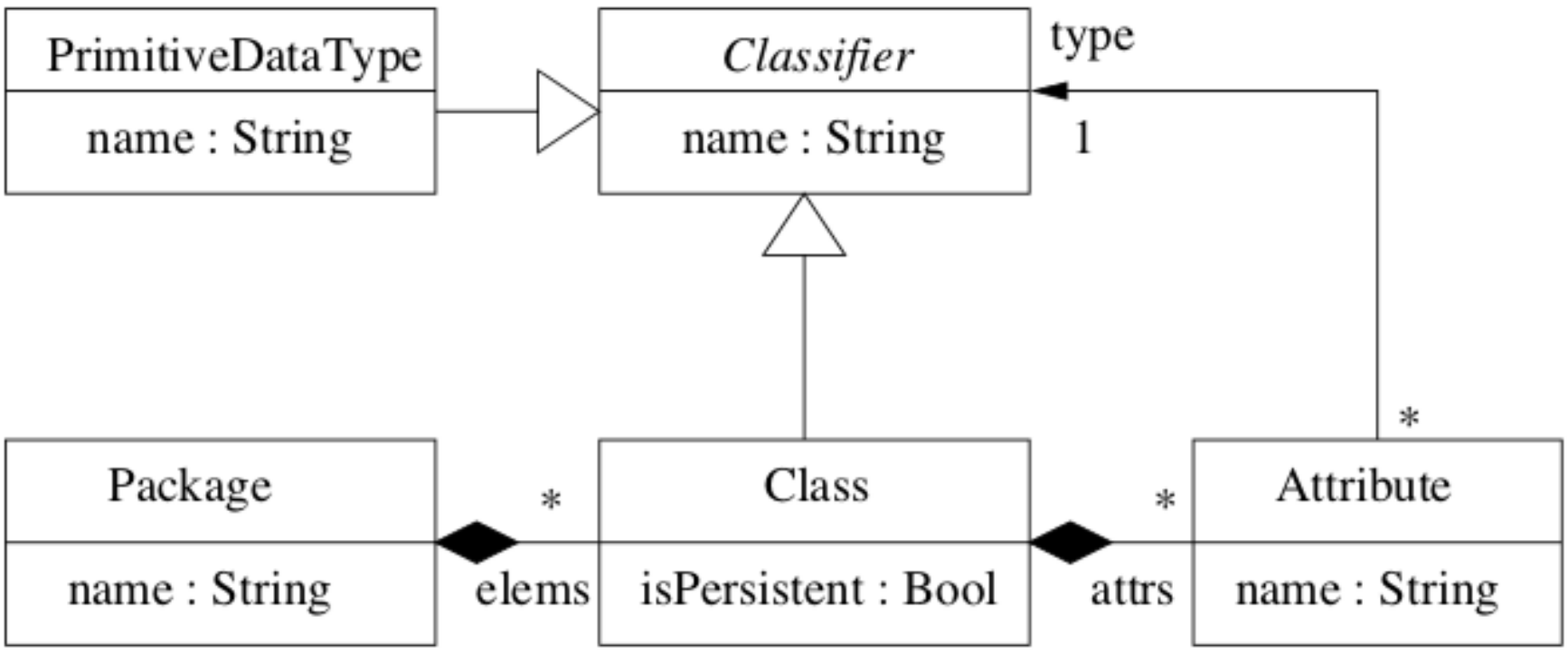
Relational Schema (rdbms)

Target Language



Class Diagrams (uml)

Source Language



QVT Relations Specification

from [Czarnecki and Helsen(2006)]

```
transformation umlRdbms (uml :SimpleUML, rdbms :SimpleRDBMS)
{
  key Table (name, schema)
  key Column (name, table)

  top relation PackageToSchema {
    domain uml p:Package {name = pn}
    domain rdbms s:Schema {name = pn}
  } ...
```

- First identify names as primary keys for Tables and Columns
- The main relation converts between uml Packages and rdbms Schema
- Identifier **pn** relates the two names to be identical
- QVT relations can be executed in either direction!

QVT Relations Specification

from [Czarnecki and Helsen(2006)]

```
transformation umlRdbms (uml :SimpleUML, rdbms :SimpleRDBMS)
{
  key Table (name, schema)
  key Column (name, table)

  top relation PackageToSchema {
    domain uml p:Package {name = pn}
    domain rdbms s:Schema {name = pn}
  } ...
```

- First identify names as primary keys for Tables and Columns
- The main relation converts between uml Packages and rdbms Schema
- Identifier **pn** relates the two names to be identical
- QVT relations can be executed in either direction!

QVT Relations Specification

from [Czarnecki and Helsen(2006)]

```
transformation umlRdbms (uml :SimpleUML, rdbms :SimpleRDBMS)
{
  key Table (name, schema)
  key Column (name, table)

  top relation PackageToSchema {
    domain uml p:Package {name = pn}
    domain rdbms s:Schema {name = pn}
  } ...
```

- First identify names as primary keys for Tables and Columns
- The main relation converts between uml Packages and rdbms Schema
- Identifier **pn** relates the two names to be identical
- QVT relations can be executed in either direction!

QVT Relations Specification

from [Czarnecki and Helsen(2006)]

```
transformation umlRdbms (uml :SimpleUML, rdbms :SimpleRDBMS)
{
  key Table (name, schema)
  key Column (name, table)

  top relation PackageToSchema {
    domain uml p:Package {name = pn}
    domain rdbms s:Schema {name = pn}
  } ...
```

- First identify names as primary keys for Tables and Columns
- The main relation converts between uml Packages and rdbms Schema
- Identifier **pn** relates the two names to be identical
- QVT relations can be executed in either direction!

Example continued ...

```
top relation ClassToTable {
    domain uml c:Class {
      package = p:Package {},
      isPersistent = true,
      name = cn
    }
    domain rdbms t:Table {
      schema = s:Schema {},
      name = cn,
      cols = cl:Column {
          name = cn + '_tid',
          type = 'NUMBER' },
      pkey = cl
    }
}
```

QVT Relations

Odds & Ends

- The direction of execution is specified at runtime
- QVT Relations are restricted: not everything can be implemented
- Some non-reversible transformations cannot
- The QVT standard contains other languages (including imperative)
- An ongoing effort to implement QVT in Eclipse: http://wiki.eclipse.org/M2M/QVT_Declarative_(QVTd)

QVT Relations

Odds & Ends

- The direction of execution is specified at runtime
- QVT Relations are restricted: not everything can be implemented
- Some non-reversible transformations cannot
- The QVT standard contains other languages (including imperative)
- An ongoing effort to implement QVT in Eclipse: http://wiki.eclipse.org/M2M/QVT_Declarative_(QVTd)

QVT Relations

Odds & Ends

- The direction of execution is specified at runtime
- QVT Relations are restricted: not everything can be implemented
- Some non-reversible transformations cannot
- The QVT standard contains other languages (including imperative)
- An ongoing effort to implement QVT in Eclipse: http://wiki.eclipse.org/M2M/QVT_Declarative_(QVTd)

QVT Relations

Odds & Ends

- The direction of execution is specified at runtime
- QVT Relations are restricted: not everything can be implemented
- Some non-reversible transformations cannot
- The QVT standard contains other languages (including imperative)
- An ongoing effort to implement QVT in Eclipse: http://wiki.eclipse.org/M2M/QVT_Declarative_(QVTd)

QVT Relations

Odds & Ends

- The direction of execution is specified at runtime
- QVT Relations are restricted: not everything can be implemented
- Some non-reversible transformations cannot
- The QVT standard contains other languages (including imperative)
- An ongoing effort to implement QVT in Eclipse: http://wiki.eclipse.org/M2M/QVT_Declarative_(QVTd)

- Bidirectional M2M TX with QVT Relations
- **Unidirectional Rule-based M2M TX with ATL**

AGENDA

ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

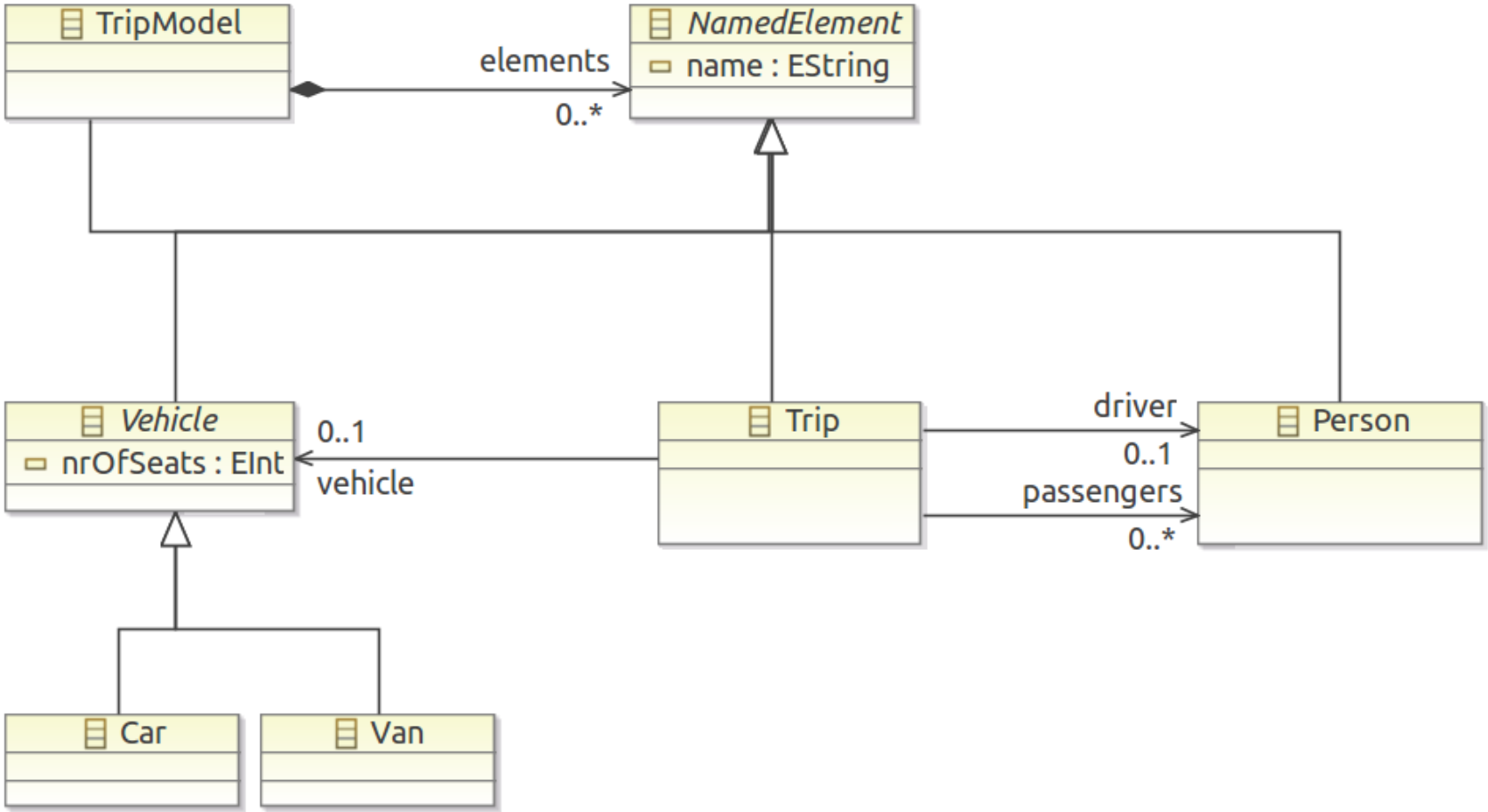
ATL

Also known as Atlas Transformation Language

- Originally developed as a proposal for the QVT Standard
- Lives its own life ever since
- One of the most popular model transformation languages
- Free implementation within the Eclipse Modeling Tools project
- **Declarative part**, imperative part
- We will discuss it on two examples

LooseTrip Meta-Model

Source & Target Language for Next Example



Capitalize Names in ATL

```
1  module DoCapitalizeNames;
2  create OUT: tripLoose from IN: tripLoose;
3
4  abstract rule CapitalizeName {
5      from s :tripLoose!NamedElement
6      to t :tripLoose!NamedElement (name <- s.name.toUpper() )
7  }
8
9  rule CapitalizePerson extends CapitalizeName {
10     from s: tripLoose!Person to t: tripLoose!Person
11 }
12
13
14 rule CapitalizeCar extends CapitalizeName {
15     from s: tripLoose!Car to t: tripLoose!Car
16 }
17
18 rule CapitalizeTrip extends CapitalizeName {
19     from s: tripLoose!Trip
20     to t: tripLoose!Trip (
21             driver <- s.driver,
22             passengers <- s.passengers,
23             vehicle <- s.vehicle
24         )
25 }
26
27 rule CaptializeTripModel extends CapitalizeName {
28     from s: tripLoose!TripModel
29     to t: tripLoose!TripModel ( elements <- s.elements )
30 }
```

Capitalize Names

An Endo-Transformation in Java

```
import java.util.List;

import org.eclipse.emf.mwe2.runtime.workflow.IWorkflowContext;
import org.eclipse.emf.mwe2.runtime.workflow.IWorkflowComponent;

import dk.itu.example.tripLoose.NamedElement;
import dk.itu.example.tripLoose.TripModel;

public class DoCapitalizeNames implements IWorkflowComponent {

    public void invoke(IWorkflowContext ctx) {

        @SuppressWarnings("unchecked")
        List<TripModel> tms = (List<TripModel>) ctx.get("model");
        TripModel tm = tms.get(0);

        for ( NamedElement ne : tm.getElements())
            ne.setName(ne.getName().toUpperCase());
    }

    public void postInvoke() {..

    public void preInvoke() {..

}
```

ATL Example 1

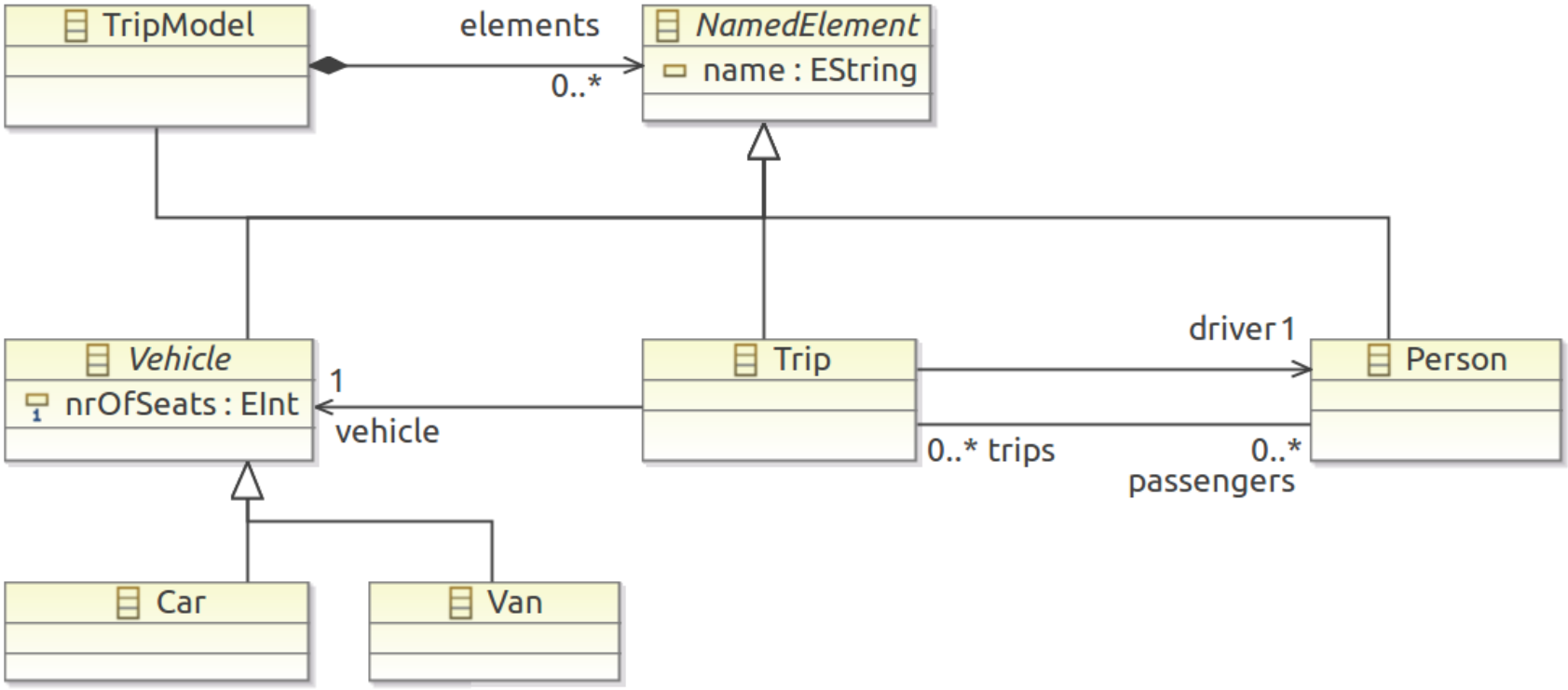
- Java transformation was shorter because it was in-place (accidental)
- Java transformation required writing scheduling (traversal) code
- We wrote a copying ATL transformation to show rules
- We did not have to write any traversal code

- Abstract rules allow to reuse rewrites declaratively
- All four bottom rules inherit the name capitalization

- Discuss the type errors! (line 29)

Trip Meta-Model

Target Language for the Next Example



tripLoose2trip in ATL

```
1  module tripLoose2trip;
2  create OUT: trip from IN: tripLoose;
3
4  helper def : tripsReferringTo(p :tripLoose!Person) : Sequence(tripLoose!Trip) =
5      tripLoose!Trip.allInstances()->select(t | t.passengers->includes(p) );
6
7  rule Person2Person {
8      from
9          s: tripLoose!Person
10     to
11         t: trip!Person (
12             name <- s.name,
13             trips <- thisModule.tripsReferringTo (s)
14         )
15 }
16
17 rule Car2Car {
18     from
19         s: tripLoose!Car
20     to
21         t: trip!Car (
22             name <- s.name,
23             nrOfSeats <- s.nrOfSeats
24         )
25 }
```

ATL Example 2

- Using helper function tripsReferringTo
- Computes trips containing *p* on the passenger list
- Note the OCL syntax of the body of the function
- Select: filters out trips not satisfying the predicate
- Line 13: the helper function is called

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

ATL Odds & Ends

- Unique lazy rules are like create methods in Xtend (automatic caching of created objects)
- Refinements: in-place transformations (possible)
- Imperative sub-language
- Transformation zoo! Online collection of many transformations in ATL
- Some random examples from the zoo:
  - MOF to UML
  - make to ant
  - MySQL schema to class diagrams
  - Extractors from Excel documents
  - ...
- Several past groups had good experience with ATL

- Bidirectional M2M TX with QVT Relations
- Unidirectional Rule-based M2M TX with ATL

AGENDA

Krzysztof Czarnecki and Simon Helsen.
Feature-based survey of model transformation approaches, 2006.