A taxonomy for Bidirectional Model Transformation and its Application

Romeo Marinelli

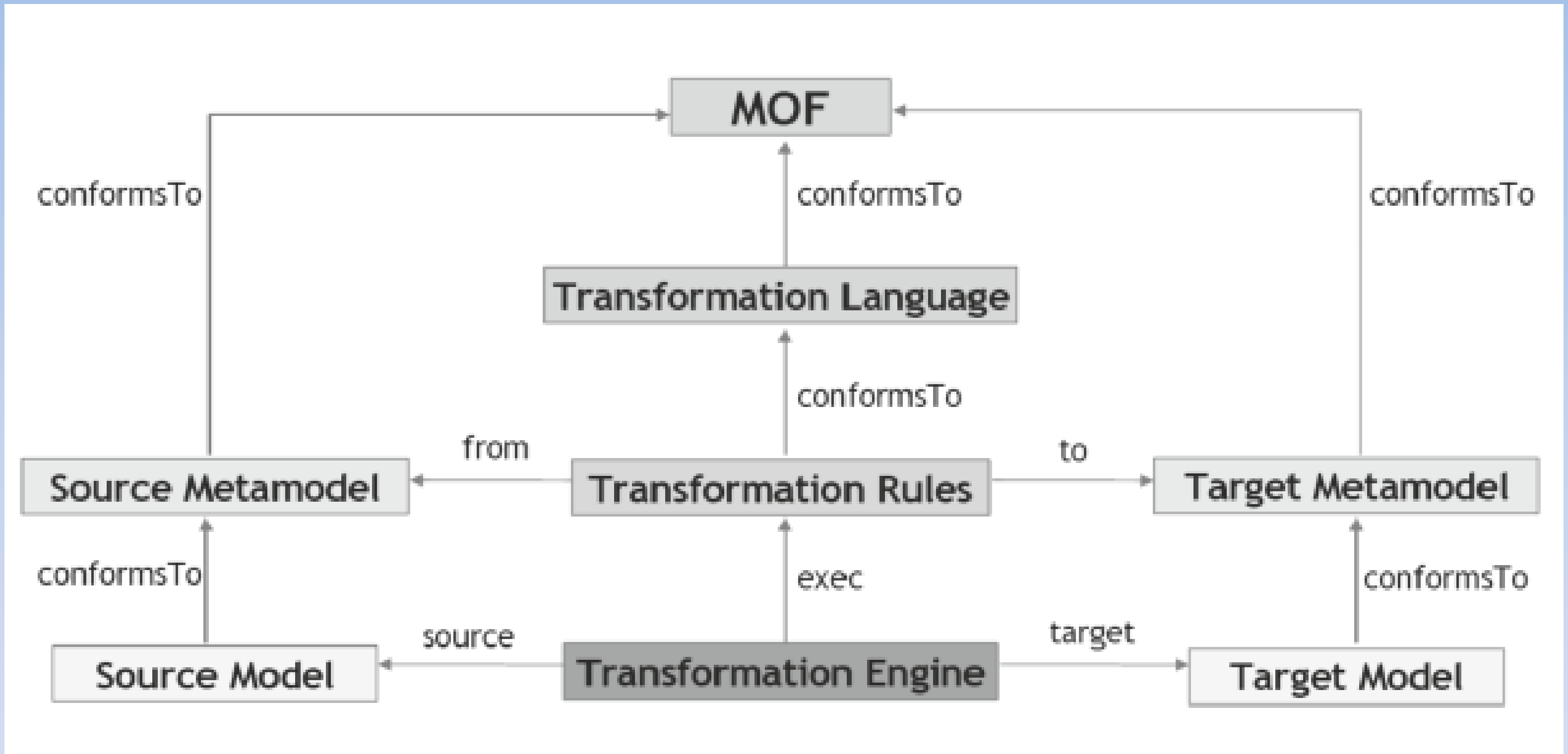
PhD Student

Università degli Studi di L’Aquila
Dipartimento di Ingegneria e Scienze dell’Informazione e Matematica

Agenda

- Motivation
- Introduction
- Bidirectionality
- Language/Tools for BX
- Taxonomy for BX
- Application
- Conclusion
- References

Model transformation



Motivation

- **Bidirectional model transformation (BX)**
- Is a model transformation among models in which, **the same model can sometimes be input** and **other times be output**.
- Bidirectional transformations are necessary in situations where people are working on more than one model and **the models must be kept consistent**.
- Then a change to either model might necessitate a change to the other, in order to maintain **consistency between the models**.

Motivation

- **Bidirectionality**

  is a relevant aspects in **model transformations**: often it is assumed that during development only the source model of a transformation undergoes modifications, however in practice it is necessary for developers to **modify both the source and the target models** of a transformation and **propagate changes in both directions**.

Motivation

- **Bidirectional model transformation** has many potential applications in software development, including
- **model synchronization**,
- **round-trip engineering**,
- **software evolution** by keeping different models coherent to each other,
- **multiple-view software development**.

Motivation

- Given that there are **many different existing tools** for BX,
- It is of **crucial** relevance **to investigate** common characteristics of tools, in order to have a better understanding on
  - **how the user can chose the tool that best suit for his applications**.

Introduction

- This work **investigates and suggests** a number of **objective criteria** to be taken into consideration to provide a concrete answers to the question.

- Based on the answers, **the developer** can then **select** the **BX approach** that is most suited for his needs.

Introduction

- It is based on the **analysis** of some **existing papers** and the main features of **existing tools for BX**.

- In particular **Dagstuhl seminar** (2008, 2011) and papers of **Tom Mens**, **Krzysztof Czarnecki**.

Languages for BX

1 - TGG (Triple Graph Grammar)

2 - Lenses (delta-lenses, lenses inside Scala)

3 - JTL (Januas Transformation Language)

4 - GRoundTram (Graph Roundtrip Transformation for Models)

5 - QVT-R (Queries/Views/Transformations)

6 - BiFlux (ICSE 2014 - India)

Tools for BX

1 - TGG: eMoflon, EMORF, MoTE, TGG Interpreter, FUJABA

2 - Lenses: Boomerang

3 - JTL

4 - GRoundTram

5 - QVT-R: Medini-QVT

6 - BiFlux

Existent tools can be analyzed based on following features:

1. Functional and declarative
2. Compositional and not compositional
3. Change propagation (totally/partial) - incrementality
4. Data model (tree/graph)
5. Interoperability
6. Platform used (standardization)
7. Validity: models and model transformation

Taxonomy for BX

- general requirements GR
- functional requirements FR
- not functional requirements NFR

Taxonomy General Requirements

level of automation

complexity of the transformation

visualization

level of industry application

maturity level

Taxonomy Functional Requirements

correctness of the transformations,

inconsistency management, modularity, traceability, change propagation, incrementality, uniqueness, termination, symmetric/asymmetric behavior, type of artifact, data model, endogenous/exogenous, mechanism of transformation, in-place/out-of-place transformations

Taxonomy Not Functional Requirement

extensibility/modifiability,

usability and utility, scalability, robustness,

verbosity and conciseness, interoperability,

reference platform (standardizzation),

verificability and validity of a transformation

General Requirements

- **Level of automation.** A BX transformation between models that can be performed in a completely automatic way is said fully-automated whereas a transformation that need to be performed manually (or at least needs a certain amount of manual intervention) is said human-in-the-loop (partially automated). Manual intervention is needed to address and resolve the ambiguity, incompleteness and inconsistency in the requirements that are (partially) expressed in natural language.

- **Complexity of the transformation** (It is not considered but it could/should be treated by using metrics.)

- **Visualization.** (Visualization means, the way in which a model, a metamodel and a model transformation is presented to user. It may be visual or textual)

- **Level of industry application** (It indicates whether the language / tool is only used in academic world or It is also used at industrial level.)

- **Maturity level** (theoretical/practical approach)

Functional Requirements

- **Correctness** (The correctness of a model transformation is analyzed in two ways: syntactic and semantic correctness. If the target model conforms to the target metamodel specification, then the model transformation is syntactically correct. If the model transformation preserves the behavior of the source model, then it is semantically correct)
- **Inconsistency management** (The ability to deal with incomplete or inconsistent models)
- **Modularity** (The ability to compose existing transformations into new composite ones. The language/tool can be compositional/not compositional)
- **Traceability, change propagation** (Traceability is the property of having a record of links between the source and target elements as well as the various stages of the transformation process. The language/tool has to correctly propagate the changes made to a model, in the other direction.)
- **Incrementality** (only the changes on a model are propagated to the other side)
- **Uniqueness** (BX generates unique target models for each source model)

Functional Requirements

- **Termination** (If the model transformation always terminates and leads to a result)
- **Symmetric/Asymmetric** (behavior of transformation related to models)
- **Type of artifact** (programs (program transformations - source code) or models (model transformations))
- **Data model** (graph/tree)
- **Endogenous/Exogenous** (Endogenous are transformations between models expressed in the same language, Exogenous are transformations between models expressed using different languages)
- **Transformation mechanism** (declarative / imperative / mixed or hybrid / functional)
- **In-place/Out-of-place** (A BX may be considered in-place when its source and target models are both bound to the same model at runtime, out-of-place otherwise)

Not Functional Requirements

- **Extensibility/modifiability** (Regarding to the tool, means the ease in which, it can be extended with new features (extensibility of the tool). Regarding to the artifact, means the ability/ease of a BX transformation to be modified and adapted to provide different or additional features.)

- **Usability and utility** (The language or tool should be useful, which means that it has to serve a practical purpose. On the other hand, it has to be usable too, which means that it should be intuitive and efficient to use)

- **Scalability** (The language or tool should be able to cope with large and complex transformations or transformations of large and complex software models)

- **Robustness** (If most of the unexpected errors can be handled and the model transformation can manage with the all invalid source models, then it provides robustness.)

Not Functional Requirements

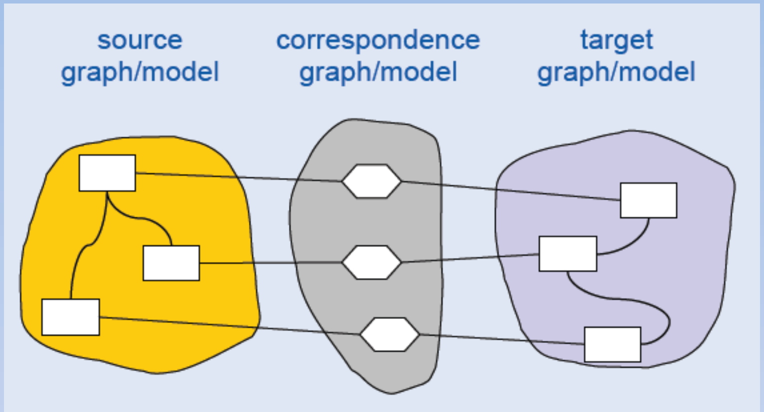
- **verbosity and conciseness** (Conciseness means that the transformation language should have as few syntactic constructs as possible. From a practical point of view, however, this often requires more work to specify complex transformations. Hence, the language should be more verbose by introducing extra syntactic sugar for frequently used syntactic constructs.)

- **interoperability** (The ease in which the tool can be integrated with other tools to be used in the process of software engineering (in model-driven way))

- **reference platform (standardizzation)** (whether the transform. tool is compliant to all relevant standards (e.g., XML, UML, MOF))

- **verificability and validity** (Ability to test, verify and validate models and transformations.)

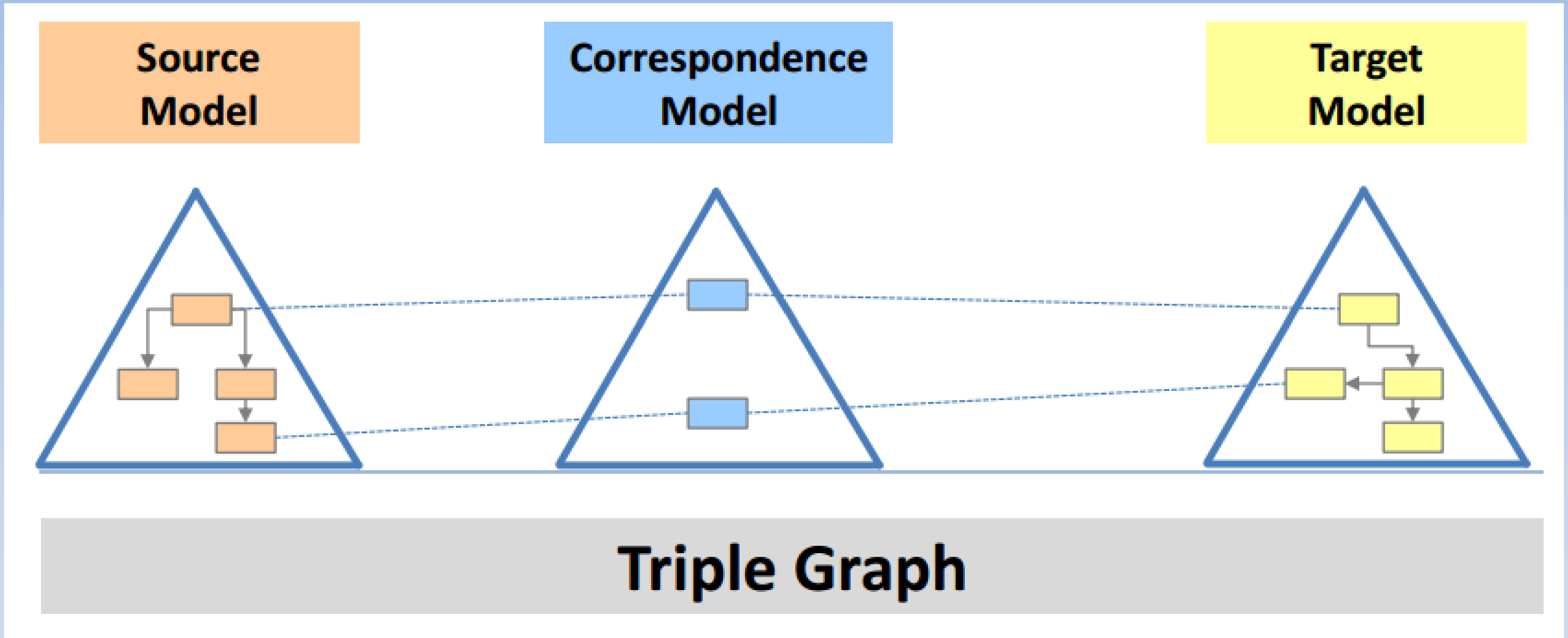
TGG



Triple Graph Grammars (TGGs) are a formalism for the **rule-based specification** of mappings between different kinds of graphs resp. different kinds of models.

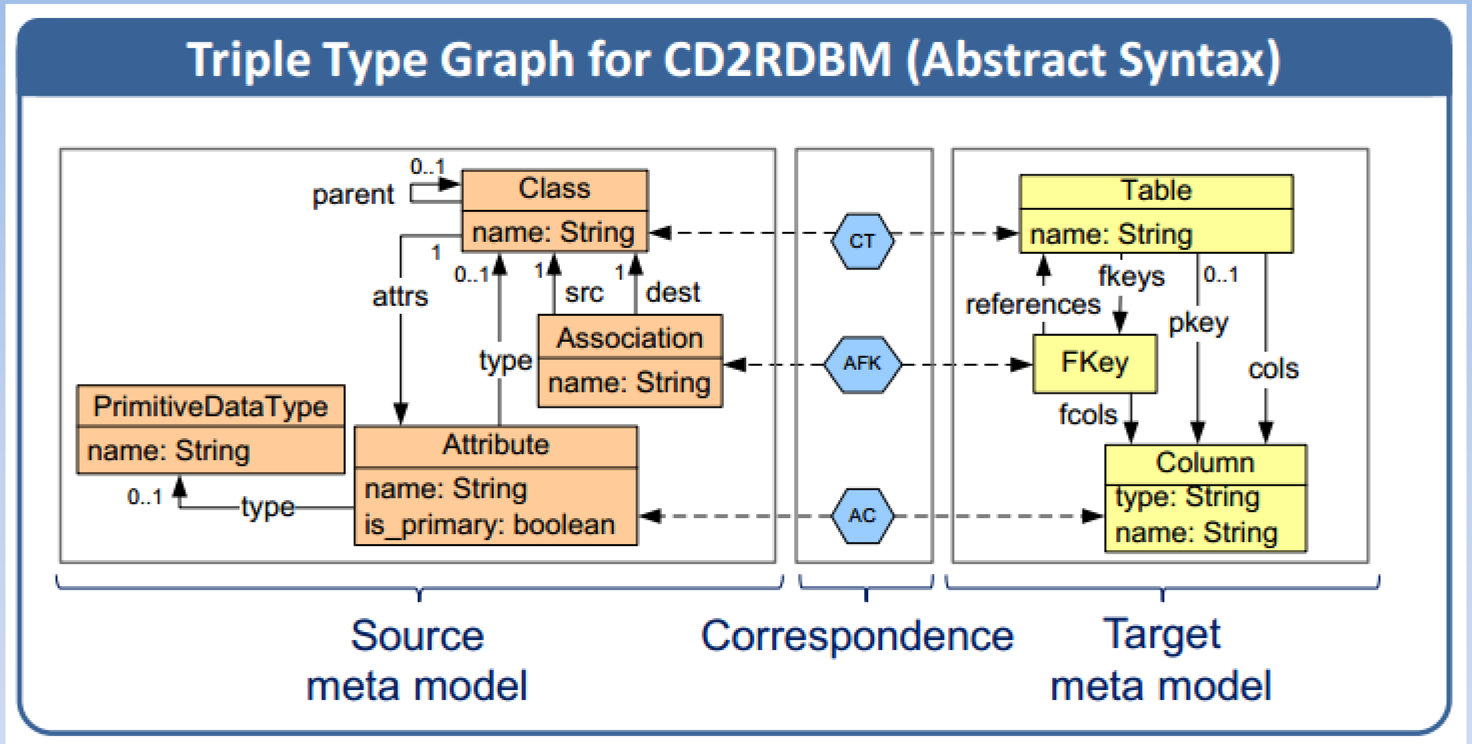
TGGs can be employed for model-to-model (M2M) transformations. In contrast to many other model transformation languages, **the developer** does not have to “program” a sequence of model transformation steps, but **specifies graphical rules** that describe the mapping between model patterns.

TGG



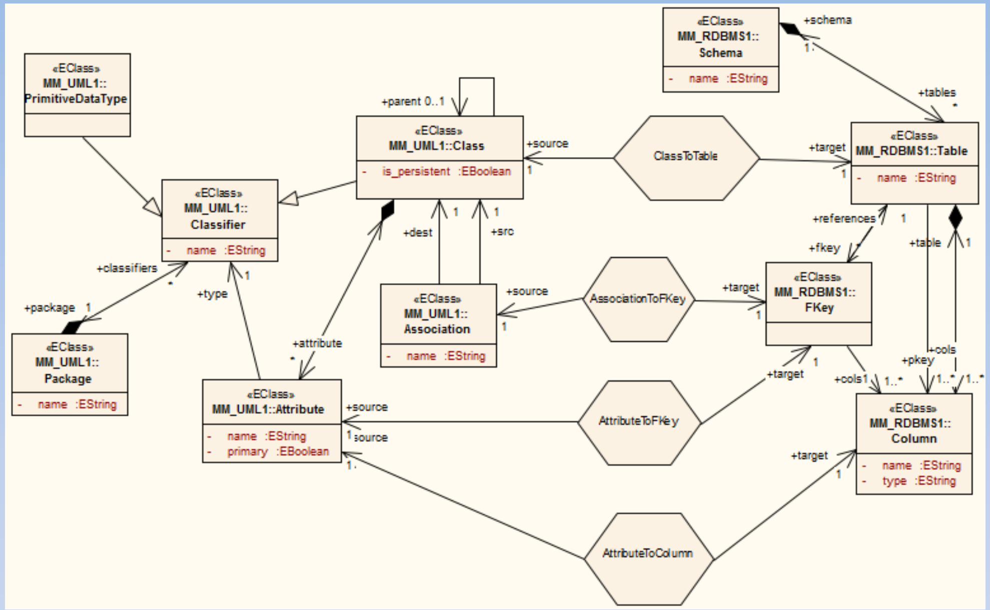
Source
Model
Correspondence
Model
Target
Model
Triple Graph

TGG



Triple Graph Grammars:
UML2RDBMS bidirectional model transformation

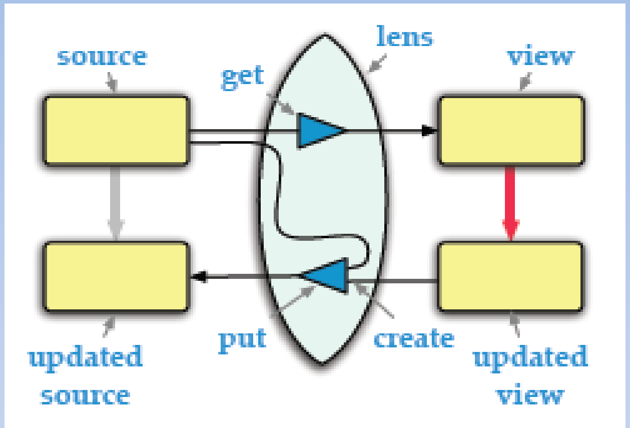
TGG

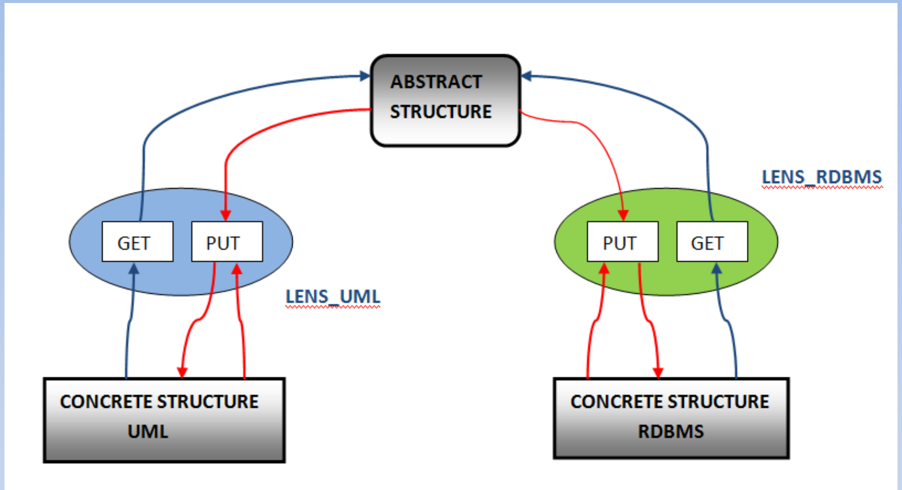


Triple Graph Grammars (TGGs) features:

DECLARATIVE - incrementality - graphical - graph - fully-automated - industrial and academic - compositional in the rule - model in XMI format - Ecore/EMF - **good Interoperability** - **EMoflon, EMORF**

Lenses - Boomerang





Lenses (Foster), are **asymmetric bidirectional transformations**, i.e., **one of the two structures that are synchronized has to be an abstraction of the other**. (view-update-problem)

The forward transformation **get** derives an abstract structure from a given concrete structure. The backward transformation **put** takes an updated abstract structure and the original concrete structure to yield an updated concrete structure.

Lenses

```
(* prima lente: struttura astratta verso il modello uml *)

 let person : lens =
   Xml.attr1_elt_swap NL1 "class" "name" (copy Id)
     begin
       Xml.simple_elt NL2 "name" (copy Name) .
                qins SP " " .
       Xml.simple_elt NL2 "citta" (copy Citta) .
       qins SP " "
     end

 let persons : lens =
     copy ""
   | person . ( qins Nl "\n" . person )*

 let file : lens =
   Xml.elt NL0 "uml" persons
```

```
let file_uml : string =
<<
  <uml>
     <class name="person">
        <name>Pino Pinotti</name>
        <citta>Roma</citta>
     </class>
     <class name="customer">
        <name>Gino Ginotti</name>
        <citta>Milano</citta>
     </class>
     <class name="customer">
        <name>Pippo Pippotti</name>
        <citta>Napoli</citta>
     </class>
  </uml>
>>
```

```
let file_ascii : string =
<<
   Pino Pinotti Roma person
   Gino Ginotti Milano customer
   Pippo Pippotti Napoli customer
>>
```

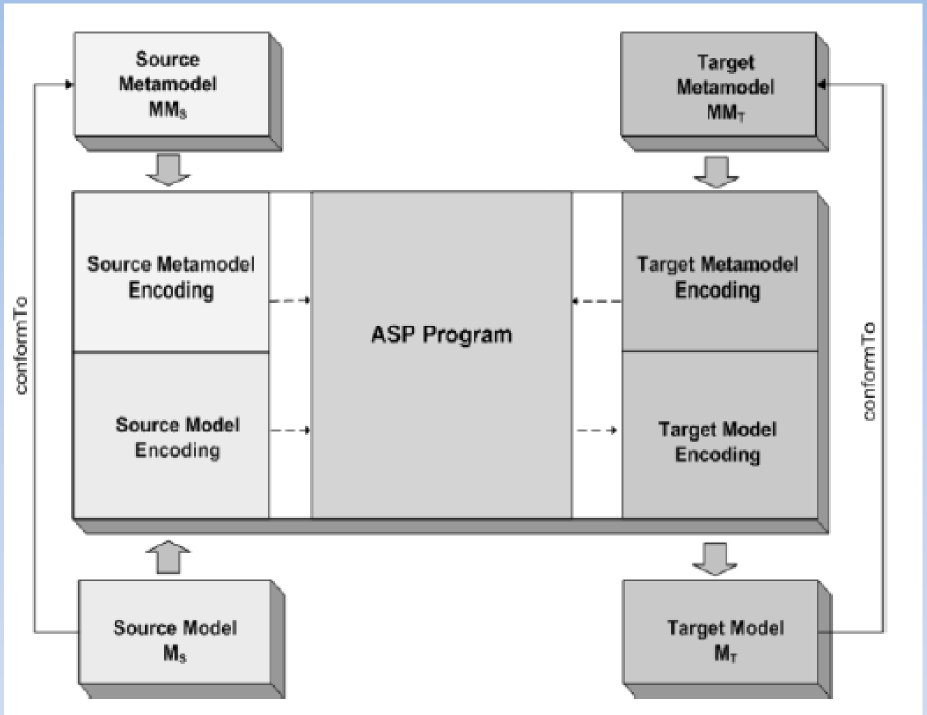
Lenses features:

COMPOSITIONAL - functional – ASYMMETRIC - tree - textual

model expressed by XML-file - **not integrable with others tools**

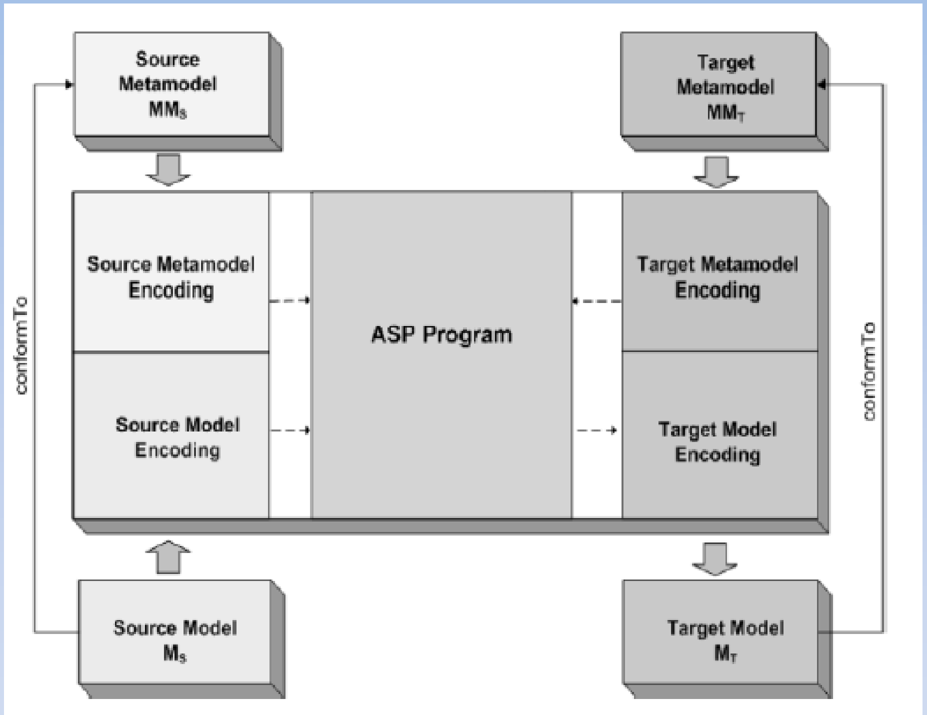
No EMF – no validity - Boomerang

JTL



JTL is a constraint-based model transformation language specifically tailored to support bidirectionality. The implementation relies on the **Answer Set Programming** (ASP), which is a form of **declarative** programming oriented towards difficult search problems.

JTL

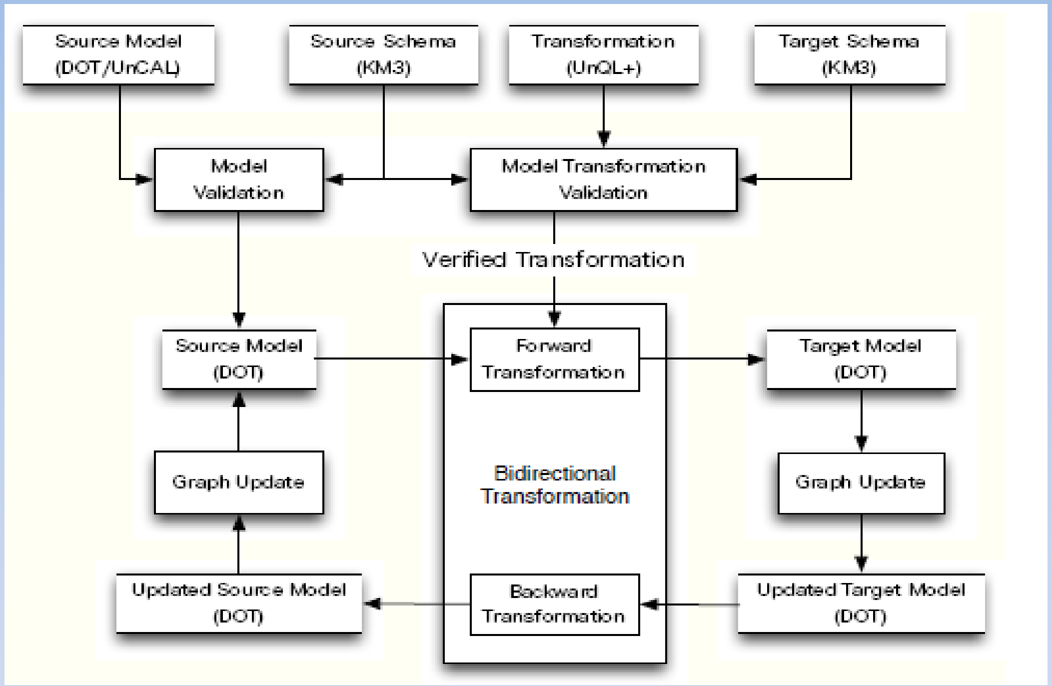


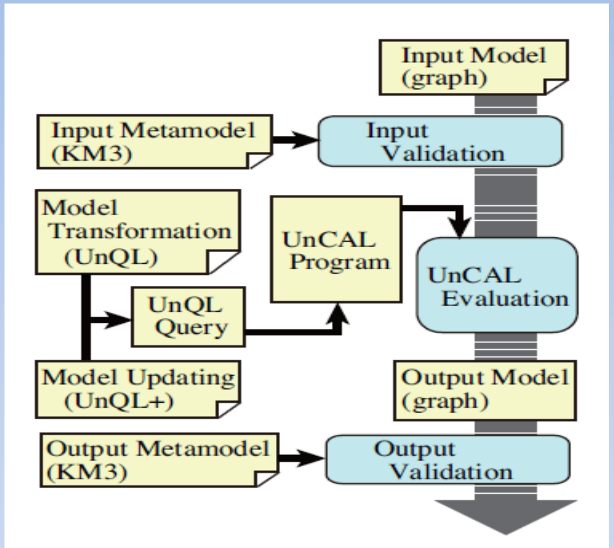
JTL features:

DECLARATIVE - symmetric - **graph** - textual – academic

not uniqueness (<u>the backword transformation can generate all possible models</u>)

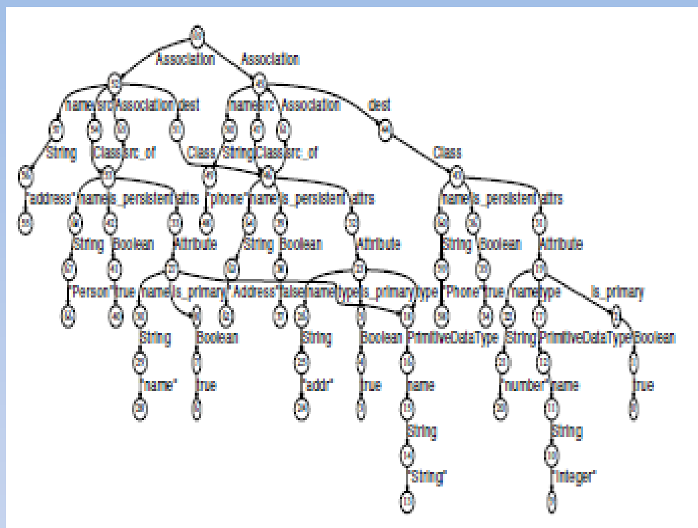
GroundTram





GroundTram is based on **UnQL (compositional graph querying language - MT) and UnCAL (Graph Algebra - Model).**
While **UnQL is an interface language for users to write queries**, **UnCAL is its core language for internal implementation**. UnCAL has a set of constructors and operators, by which arbitrary **graphs can be represented**.

GroundTram



```
select {tables : $table} where
  $persistentClass in
    (* select classes *)
    (select $class where
      {Association.(src|dest).Class : $class} in $db,
      {is_persistent : {Boolean : true}} in $class),
  $table in
    (* replace Attribute *)
    (replace attrs → $g
      by (select {Column : $a} where
          {attrs.Attribute:$a} in $persistentClass)
       in $persistentClass)
```

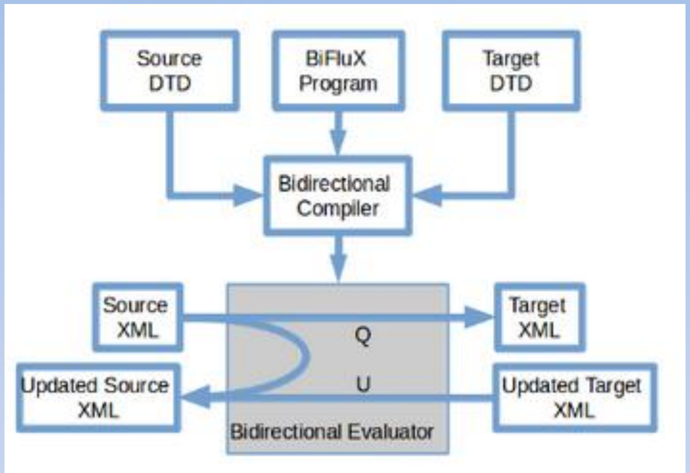
GroundTram features:

VALIDATE (model/transformation) – COMPOSITIONAL

functional - It is not integrable with others tools

own standard - textual and graphical - academic

BiFlux



BiFlux is a Bidirectional XML update language (BIdirectional FunctionaL Updates for XML), inspired by the FLUX-XML update language.
A program precisely describes **how to update a source document with a target document**, in an intuitive way, such that **there is a unique "inverse" source query for each update program**.

BiFlux

Source DTD (people.dtd)

```
<!DOCTYPE people [
        <!ELEMENT people (person*)>
        <!ELEMENT person (name,city)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT city (#PCDATA)>
]>
```

Source Model (s.xml)

```
<people>
        <person>
                <name>Hugo</name>
                <city>Tokyo</city>
        </person>
        <person>
                <name>Sebastian</name>
                <city>Kiel</city>
        </person>
        <person>
                <name>Zhenjiang</name>
                <city>Tokyo</city>
        </person>
</people>
```

```
START = updatePeople($source/people, $view/fromtokyo)

PROCEDURE updatePeople(source $source AS s:people, view $view AS v:fromtokyo) =
UPDATE $sperson IN $source/person BY
        MATCH    -> {}
|       UNMATCHV ->  CREATE VALUE <person> <name/> <city>Tokyo</city> </person>
FOR VIEW $vname IN $view/name
MATCHING SOURCE BY name VIEW BY $vname
WHERE city/text() = 'Tokyo'
```

BiFlux features:

COMPOSITIONAL - Functional - Symmetric - **tree**

fully-automated - academic - **validity**

it can not be integrated with other tools

Conclusion

- In recent times, after Foster's lenses we are moving more and more from declarative approaches toward **functional approaches**.
- Currently, the **compositionality** and **modularity** is one of the most important aspects to be consider.

Future works

- to analyze other tools and
- extend the grid

References

- **A Taxonomy of Model Transformations** - Tom Mens, Krzysztof Czarnecki
- A Taxonomy of Model Transformation and its Application to Graph Transformation - Tom Mens and Pieter Van Gorp
- **Towards a Catalog of Non-Functional Requirements for Model Transformations** - Soroosh Nalchigar, Rick Salay, and Marsha Chechik
- A Comparison of Taxonomies for Model Transformation Languages - Gabriel Tamura and Anthony Cleve
- **A Survey of Triple Graph Grammar Tools** - Giese, Schurr
- Feature-Based Survey of Model Transformation Approaches - Czarnecki, Helsen
- Metamodeling and model transformations in modeling and simulation - Deniz Cetinkaya, Alexander Verbraeck
- Model Transformation – the Heart and Soul of Model-Driven Software Development. Shane Sendall, Wojtek Kozaczynski

References

- Bidirectional Transformations: A Cross-Discipline Perspective GRACE 2008
- Dagstuhl Seminar on Bidirectional Transformations (BX) - Zhenjiang Hu, Andy Schurr- 2011 and 2005
- Toward bidirectionalization of ATL with GRoundTram - Isao Sasano, Zhenjiang Hu, Soichiro Hidaka (Composizionality)
- **Semantical Correctness and Completeness of Model Transformations** Using Graph and Rule Transformation - Hartmut Ehrig and Claudia Ermel 2009 (Correctness)
- Some Model Transformation Approaches: a Qualitative Critical Review - May Dehayni, Kablan Barbar (Traceability)
- An Algebraic Approach to Bidirectional Model Transformation, Hidaka, Hu, Kato, Nakano
- **Towards Compositional Approach to Model Transformation for Software Development**, Hidaka, Hu
- GRoundTram: An Integrated Framework for Developing Well-Behaved Bidirectional Model Transformations, Hidaka, Hu, Kato, Nakano

References

- UnQL, Buneman
- Boomerang: **Resourceful Lenses for String Data** - Foster, Pierce
- JTL: a bidirectional and change propagating transformation language, Eramo, Pierantonio

The end

Thanks for your attention