Engineering BX

Richard Paige

Dept of Computer Science, University of York

Background in Model-Driven Engineering of BX

Richard Paige

Dept of Computer Science, University of York

Overview

- My lectures will focus on approaches for *engineering* bidirectional transformations.
  - With some consideration of engineering transformations in general.
  - Emphasis on approaches with tools.
- Touch on parts of the engineering lifecycle:
  - Requirements, architecture, design, implementation, verification.
- With a focus on Model-Driven Engineering concepts, languages, tools and technologies.

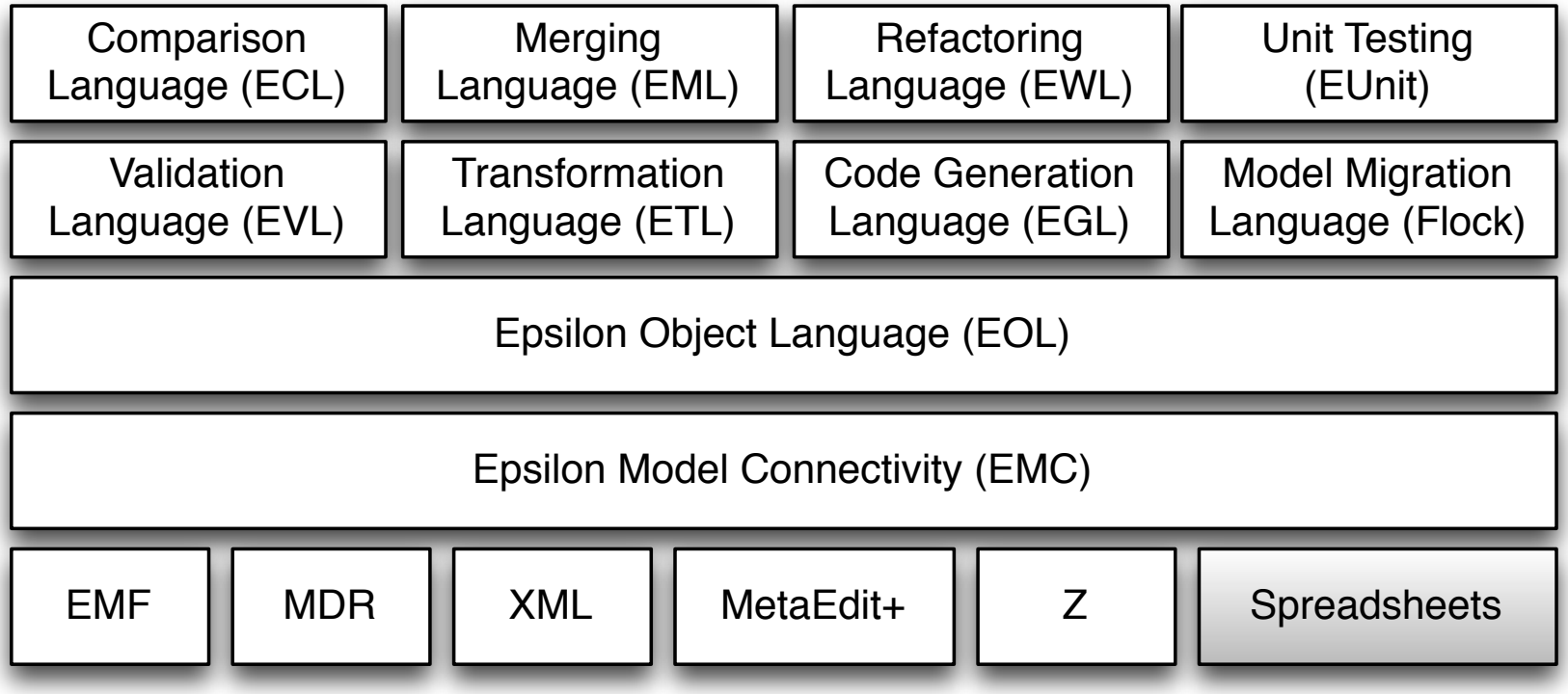
Overview

- Three lectures:
  1. Introduction to background on MDE for BX: foundations and concepts; languages and tools (state of the art); open challenges.
  2. Requirements engineering for BX: concepts, processes, MDE languages for RE for BX.
  3. Architecture, design and a little bit of verification for BX: languages, patterns, tools.

Context

- I build tools for model management.
  - E.g., model transformation, validation, merging, migration.
- And do applied research in MDE.
  - Four current projects on applied model management: scalability; “BX” for assurance; technical obsolescence; incremental querying for data analytics.

Context


Comparison Language (ECL)
Merging Language (EML)
Refactoring Language (EWL)
Unit Testing (EUnit)
Validation Language (EVL)
Transformation Language (ETL)
Code Generation Language (EGL)
Model Migration Language (Flock)
Epsilon Object Language (EOL)
Epsilon Model Connectivity (EMC)
EMF
MDR
XML
MetaEdit+
Z
Spreadsheets

Today's Lecture

- Foundations of MDE for BX:
  - Terminology: model, metamodel, types of transformations, traceability.
  - Typical applications of BX in MDE.
- State of the art: MDE languages & tools for BX (not just BX!)
- Challenges and open questions.

Foundations of MDE for BX

Model

- A structured description of phenomena of interest.
  - Captures static or dynamic characteristics.
  - Processed by automated tools.
- Structure can be defined in a number of ways:
  - Schema (explicit or implicit), typing rules, constraints, metamodel…
  - Many approaches to defining structure in MDE are based on graphs (distinction from grammarware).
  - There are accepted de facto standards (Ecore)
  - Don’t restrict ourselves to ‘just’ Ecore.

Metamodel

- A *metamodel* is a specification of the abstract syntax and (parts of the) static semantics of a language.
- The relationship between model and metamodel is called *conformance*.
  - Namely, a model conforms to a metamodel.
- Technologies for metamodelling:
  - Ecore (EMF), MOF, XMI, typed graphs, MetaDepth, … (DSLs for metamodelling)
  - NB: metamodel != grammar

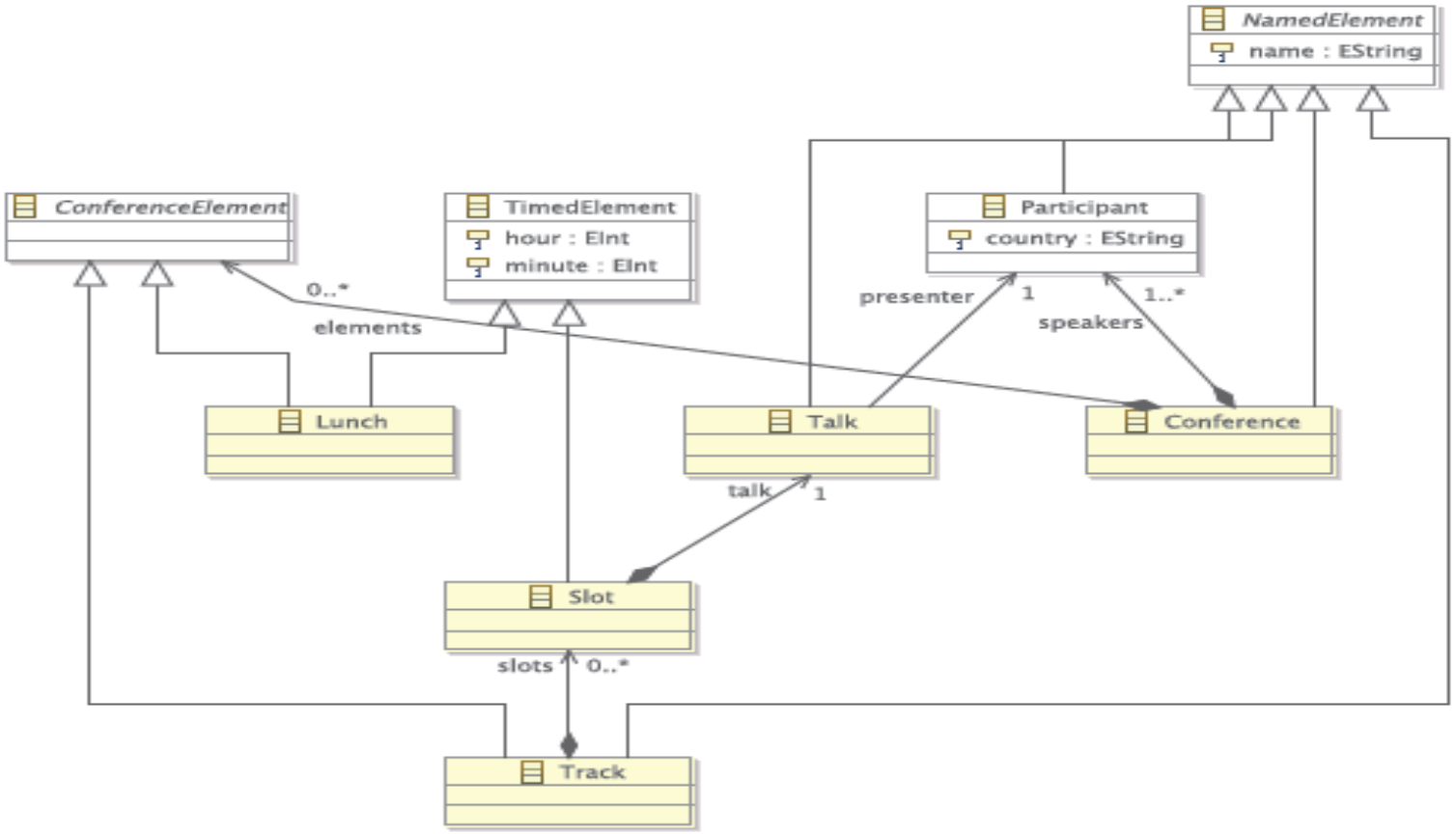
Example: TED Conference Management

- Develop a customised editor for domain experts (conference managers).
- Lets domain experts build conference models that take into account important conference timetabling concepts.
- Use EMF/Ecore

Example: Metamodel

- Key domain concepts:
  - Tracks, consisting of a number of slots in which talks can be scheduled.
  - Talks have participants (who may have to give several talks, so we must avoid clashes)
  - Lunch
- In defining a metamodel we identify recurring concepts, including naming and timing (abstract these).

Example: Metamodel



Constraints

- Metamodels capture some static semantic rules (like multiplicity).
- Richer constraints may be needed to prevent undesirable/illegitimate models from being created.
- If multiple models/metamodels are being used, inter-model constraints may be used to establish *consistency.*
  - e.g., EVL, xlinkit, OCL (with a union metamodel)
  - e.g., QVT-R checkonly mode

Operations

- Given metamodels and conforming models, we may want to apply operations to them.
  - Match/compare
  - Merge
  - Check (constraint, critique)
  - Generate (text, concrete syntax)
  - Migrate
  - Transform (update, source-to-target, bx, …)
  - ….

Comment

- Question on Monday: “How do we actually define the relations between models?”
- One way: operations on models!
  - E.g., comparison
  - E.g., transformation
  - E.g., constraint evaluation across models
- Can also populate relations by example.

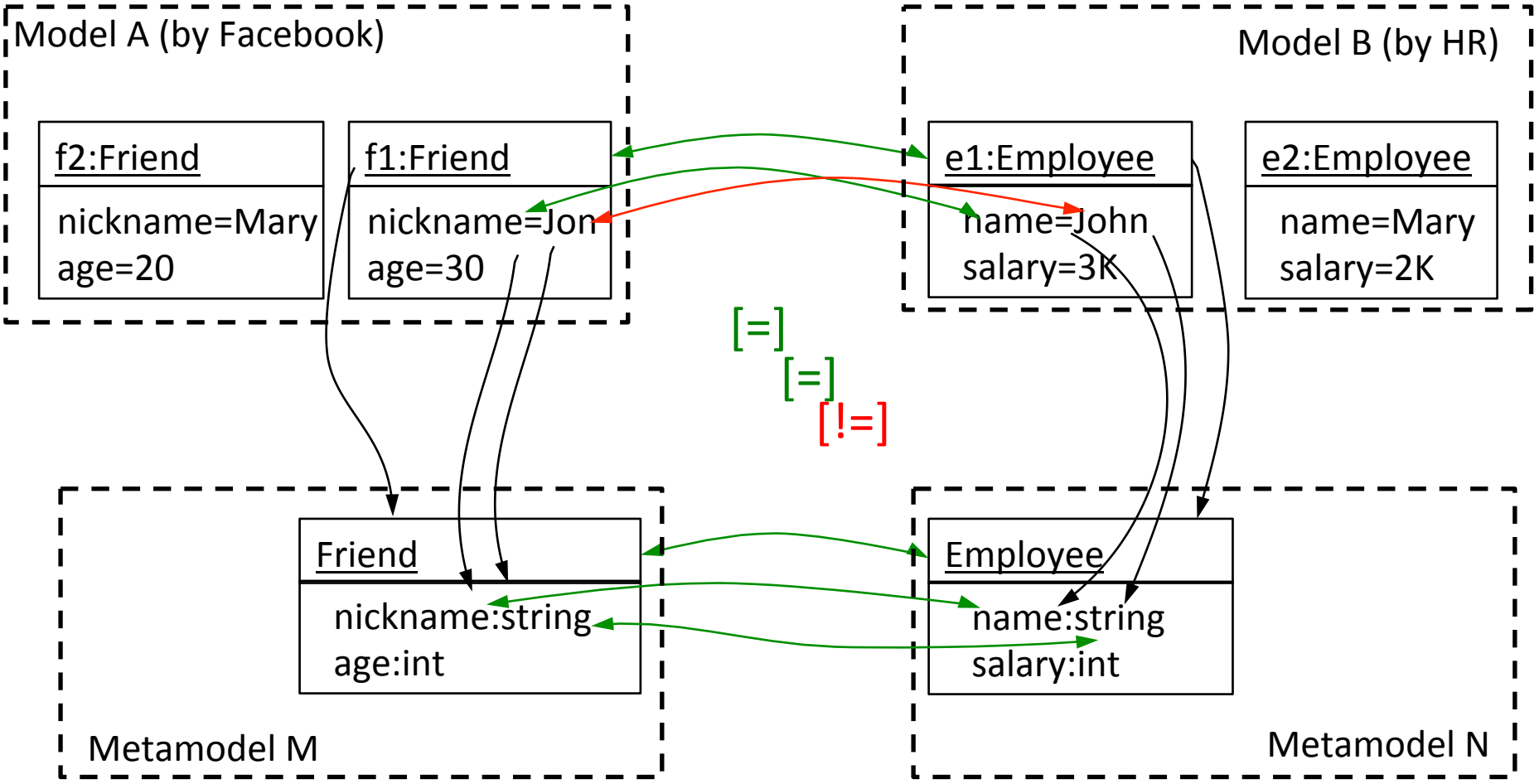
Transformations

- *Unidirectional*: from a source model to a target model.
  - Defined in terms of metamodels, usually languages are "linguistically similar".
- *Update-in-place*: modifications made to one source/target model; normally unidirectional.
- *Bidirectional*: source and target models are established to be consistent at well-defined points in time.
  - e.g., after repository check-in; after check-then-enforce has run in QVT-R
  - Could be defined as part of language semantics, or in external processes.
- *Model-to-text/grammar:* output is no longer a model but either free-form text or text conforming to grammar

Traceability

- All operations on models have side-effects: they generate trace-links.
- Trace-links relate model elements (not just models).
  - Different types of trace-links (e.g., *contains*, *regenerates*, - see Aizenbud-Reshef's work)
- Many MDE tools (Epsilon, ATL) generate such trace-links and allow them to be persisted.
- Basis for validation and verification of operations on models.
  - Loss of trace-links is one reason why tool builders may be worried about "history-ignorant" BX.

Trace Models



Example from Zinovy Diskin

Delta lenses

- A theory for trace models
- Algebraic structures comprising
  - Basic operations over models and model deltas: delta composition, delta reversal, delta propagation, tile composition
  - Basic laws these operations and their composition should satisfy
- Product line: sync scenario ---> delta lens
- …are active research topics

Today's Lecture

- Foundations of MDE for BX:
  - Terminology: model, metamodel, types of transformations, traceability.
  - Typical applications of BX in MDE.
- State of the art:
  - BX scenarios in MDE
  - MDE languages & tools for BX (not just BX!)
- Challenges and open questions.

MDE BX Scenarios

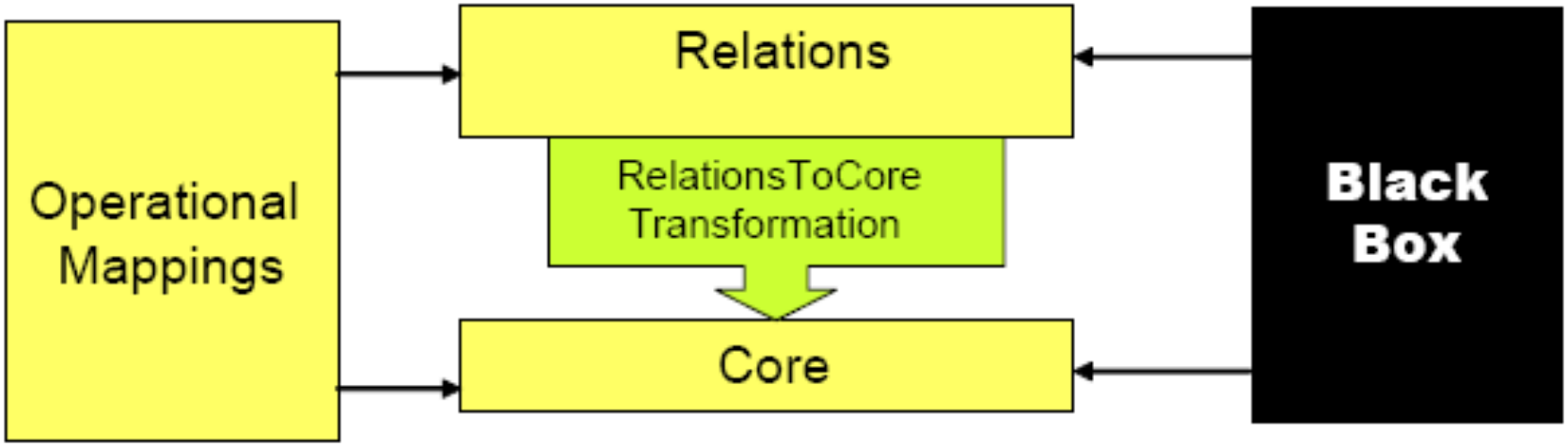
- Round-trip engineering (models to code to models)
  - E.g., obsolescence work in MONDO project
- Supporting multiple stakeholders editing the same models
  - Collaborative modelling (cf MONDO project)
- Synchronising documentation and code
  - E.g., assurance cases and design artefacts (what if not all changes can be back-propagated?)
- Reflecting analysis results in models
  - E.g., MARTE model->UPPAAL/TRIO->MARTE
  - This scenario implemented using generated transformations (more later) and also using traceability/merge.

Languages and Tools

- Numerous languages and tools to support BX, building on MDE technologies and concepts.
- The “big beast”: the OMG’s Query, Views and Transformations (QVT).
  - Standardised early.
  - Standardised too early?
  - Standardised at the wrong time?

Languages-QVT

- OMG standard for model transformation
- RFP issued by OMG on MOF Query/Views/ Transformations
- Source and target models conform to MOF metamodels.



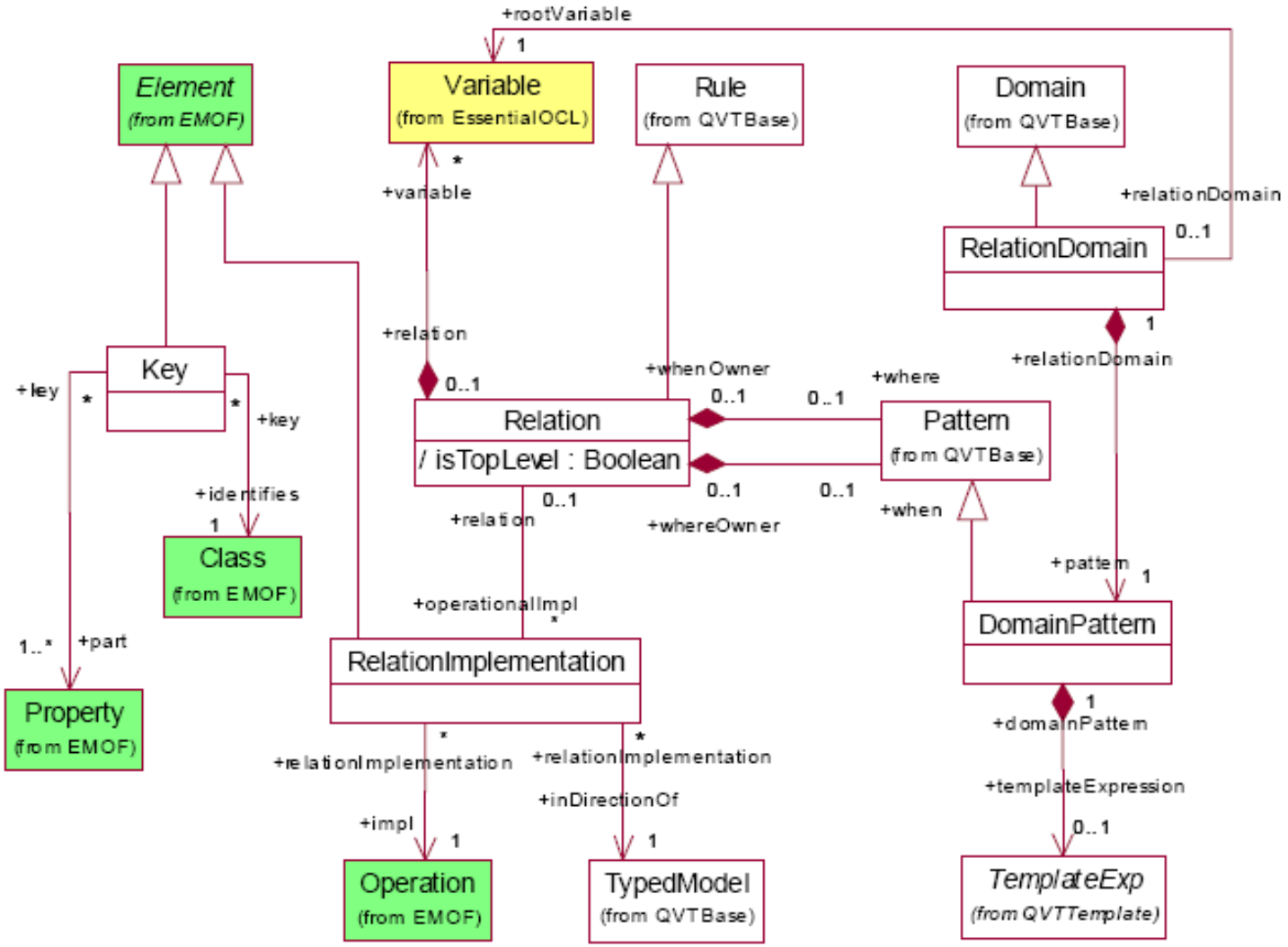
QVT Overview

- Core
  - Pattern matching over a flat set of variables
  - Defined using 'minimal' extensions to EMOF and OCL
  - Fine grained (one mapped identity per rule)
  - "Simple" transformation language
- Relations
  - Object pattern matching and object template creation
  - Coarse grained (many mapped identities per rule)
  - Specification of relations over model elements
- Operational
  - Procedural definition of transformations
  - Extends Relations language with imperative constructs

Core vs Relations

- Core is a small extension of EMOF classes and OCL expressions
  - manipulates trace models explicitly
- Relations adds
  - extended pattern syntax
  - implicit trace models
- Both languages are meant to have a similar evaluation semantics
  - multi-direction execution
  - incremental update / change propagation semantics
  - implicit object creation *and deletion*
- Relations is mapped (reduced) to Core to provide its full semantics

Relations- Abstract Syntax



Concrete Syntax

- Relation
  - Transformation between candidate models is specified as a set of relations that must hold

```
transformation umlRdbms (uml :
 SimpleUML, rdbms : SimpleRDBMS) {..}
```

  - A relation declares constraints that must be satisfied by the elements of candidate models
    - **Domain**
    - **When Pattern**
    - **Where Pattern**

Concrete Syntax - Domain

- Distinguished typed variable that can be matched in a model
- A domain has patterns
  - Set of variables and a set of constraints that model elements bound to those variables must satisfy to qualify as a valid binding of the pattern

```
relation PackageToSchema /* map each package to a schema *
{
   domain uml p:Package {name=pn}
   domain rdbms s:Schema {name=pn}
}
```

When & Where Clauses

- *When*
  - specifies the conditions under which the relationship needs to hold (preconditions according to Eclipse QVT)
- *Where*
  - specifies the condition that must be satisfied by all model elements participating in the relation (postconditions according to Eclipse QVT)

- The when and where clauses may contain any arbitrary OCL expressions in addition to the relation expressions.

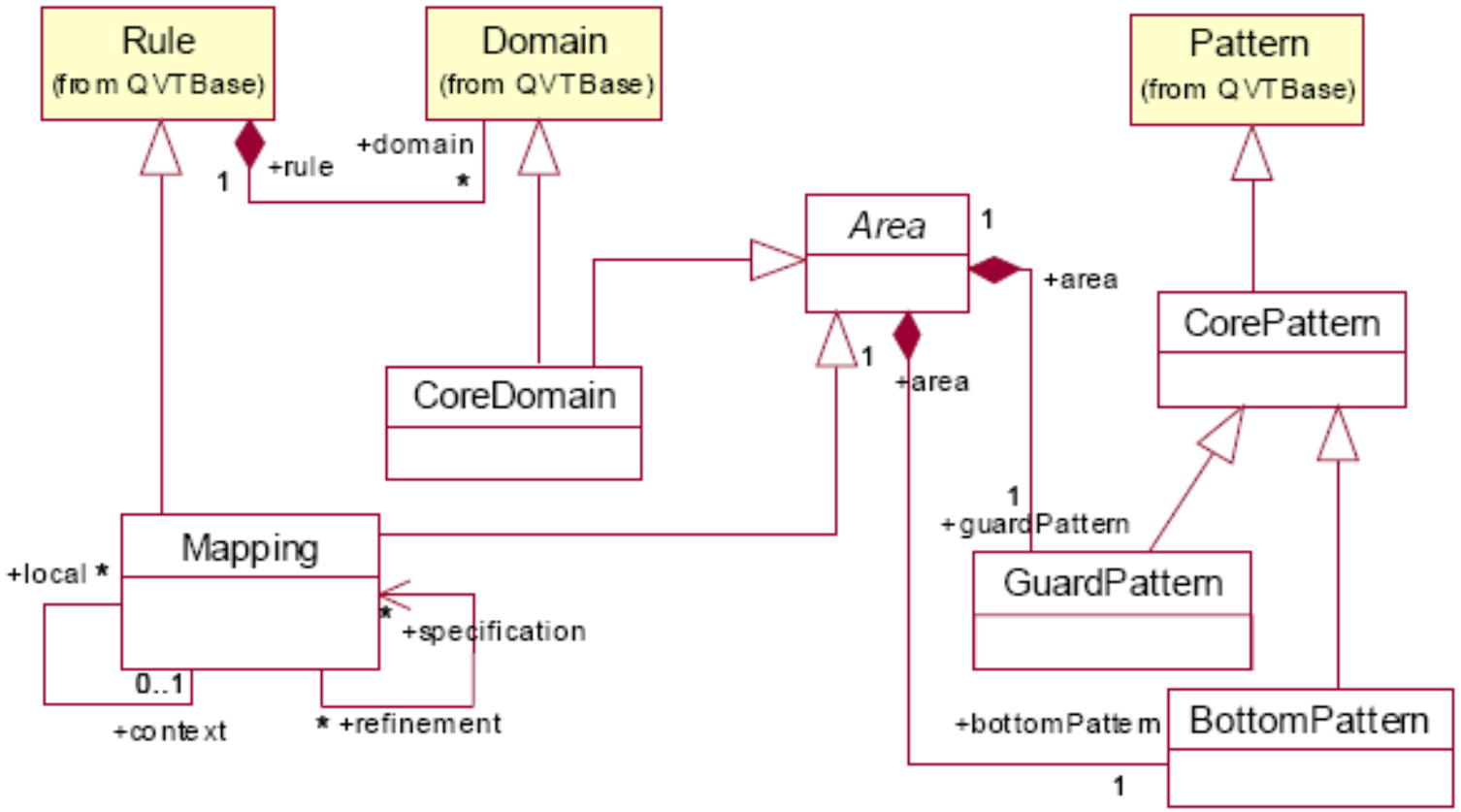
Example


```
relation ClassToTable   /* map each persistent class to a table */
  {
    domain uml c:Class {
         namespace = p:Package {},
         kind='Persistent',
         name=cn
     }
    domain rdbms t:Table {
          schema = s:Schema {},
          name=cn,
          column = cl:Column {
               name=cn+'_tid',
               type='NUMBER'},
                    primaryKey = k:PrimaryKey {
                name=cn+'_pk',
                column=cl}
     }
    when {
      PackageToSchema(p, s);
    }
    where {
      AttributeToColumn(c, t);
    }
  }
```

QVT Core

- Meant to be a “normal form” for QVT transformations
- A QVT Core transformation consists of a number of Mapping Rules.
- Each Mapping Rule consists of a collection of Patterns
- Patterns consist of variables and OCL expressions.
- A binding of a pattern is a unique set of values for all of its variables, for which all the OCL expressions hold

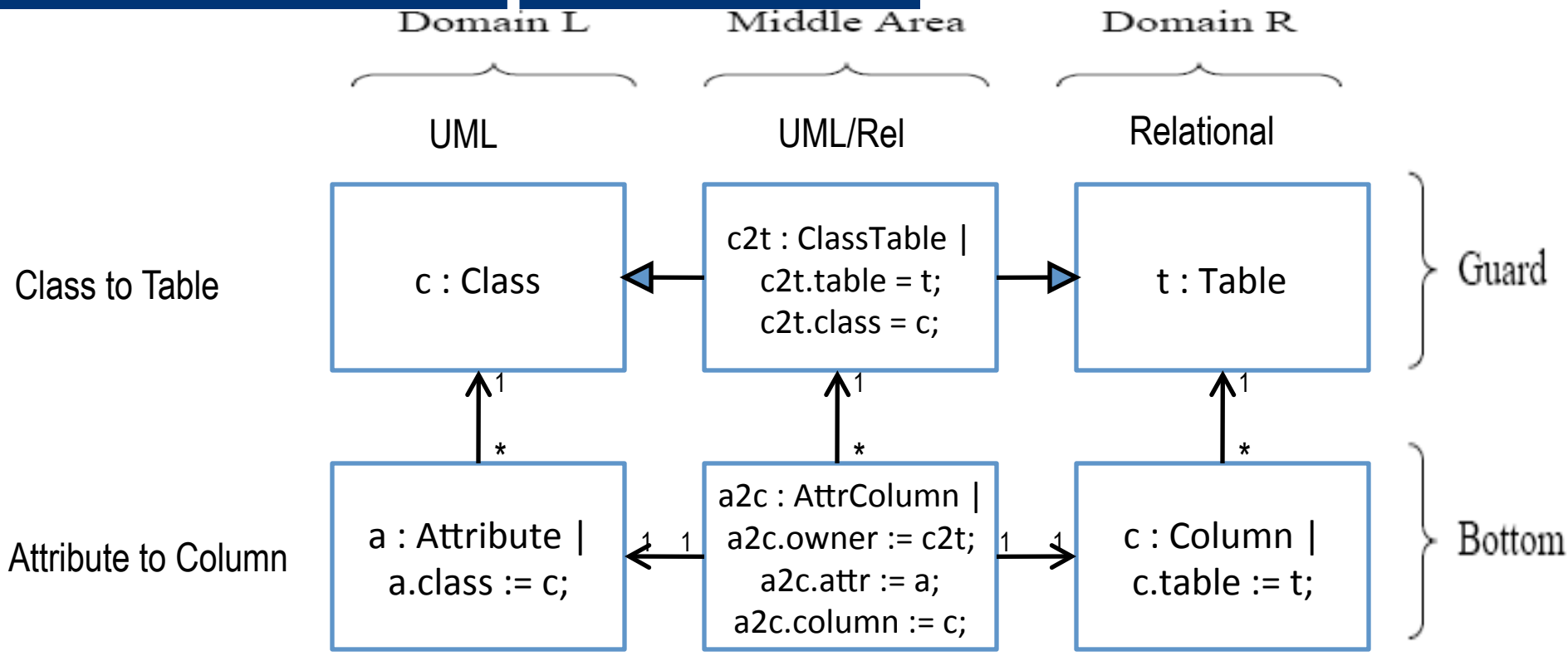
QVT Core – Abstract Syntax



QVT Core

- Domain
  - Has an associated model type of the transformation (model candidates)
- Area
  - Consists of two patterns
- Pattern
  - Set of variables, predicates and assignments
  - Can be matched or enforced
  - Can depend on each other
- Guard Pattern
  - Narrow the selection of model elements to be considered for the mapping
  - Only used for defined a context
- Bottom Pattern
  - Defines the derivations
  - Can have realized variables, assignments and black-box operation
- Mapping
  - One are for the trace and one for each model type
  - Defines a relation between bottom patterns

Mapping Rule Example



Bottom pattern is evaluated using the variable values of the valid binding of the guard pattern.

Tools

- Medini ~~QVT~~
  - Claims to be a reasonably complete, but currently unsupported, implementation of QVT-R.
  - Claims to implement the QVT standard but diverges from semantics in a number of ways
    - (e.g., deletion of elements, no checkonly mode)
  - Open source (EPL)

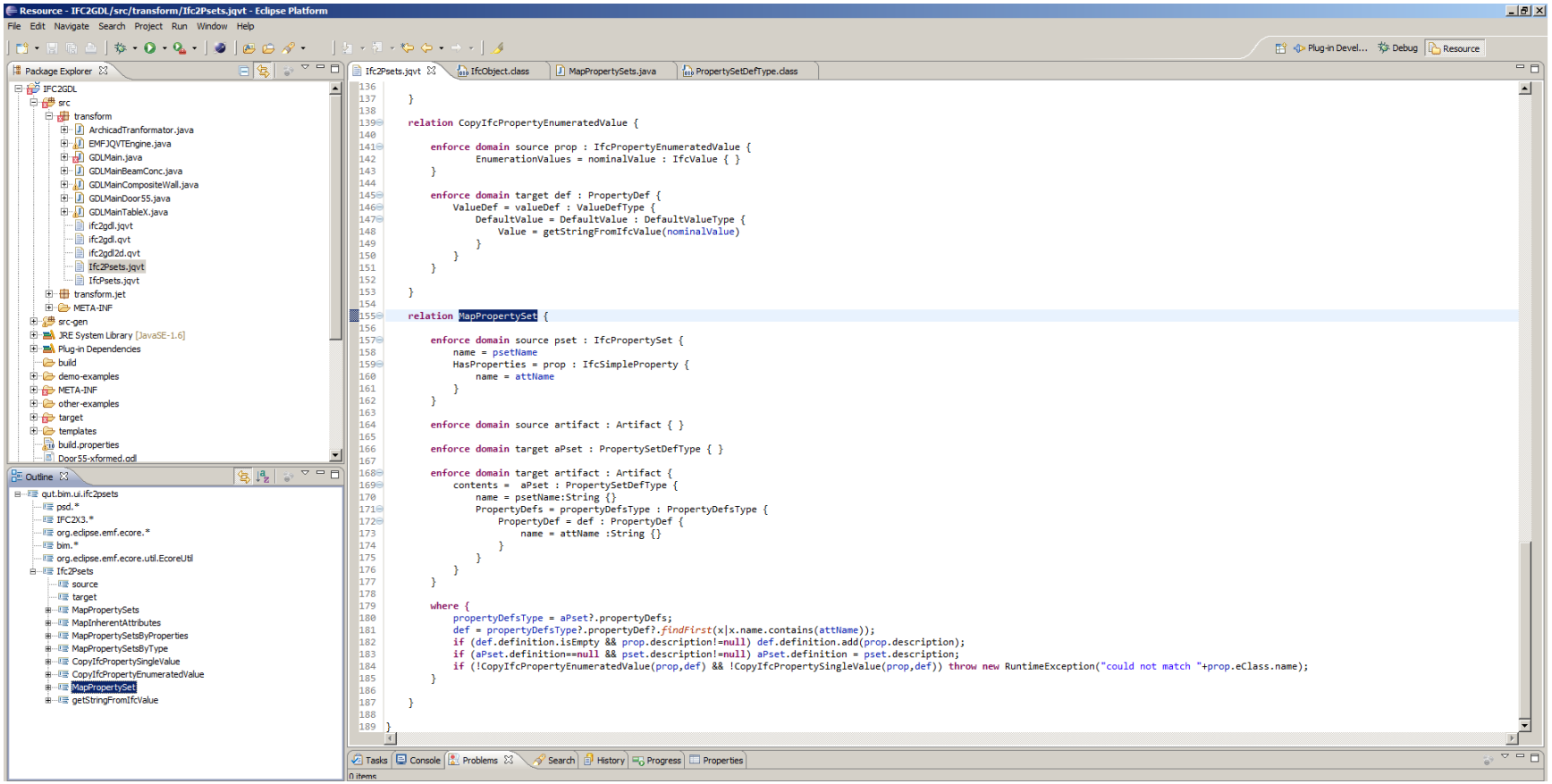
Tools

- ModelMorf:
  - From TCS, claims to be an implementation of QVT-R circa 2007.
  - Pre-Beta is available to researchers; a proprietary version is available to commercial users (internal to TCS only).
  - Perdita's research shows it more faithfully implements the QVT standard than Medini, but it's still not QVT.

Tools

- jQVT (QVT-like)
  - A QVT-like engine that is defined on the Java type system instead of EMF (uses Xbase language instead of OCL)
  - Generates native Java code from jQVT scripts.
  - Bidirectional
- Echo:
  - Uses QVT-R for inter-model consistency and model repair.
  - Alloy model finder used to generate models.
  - Bidirectional.

jQVT



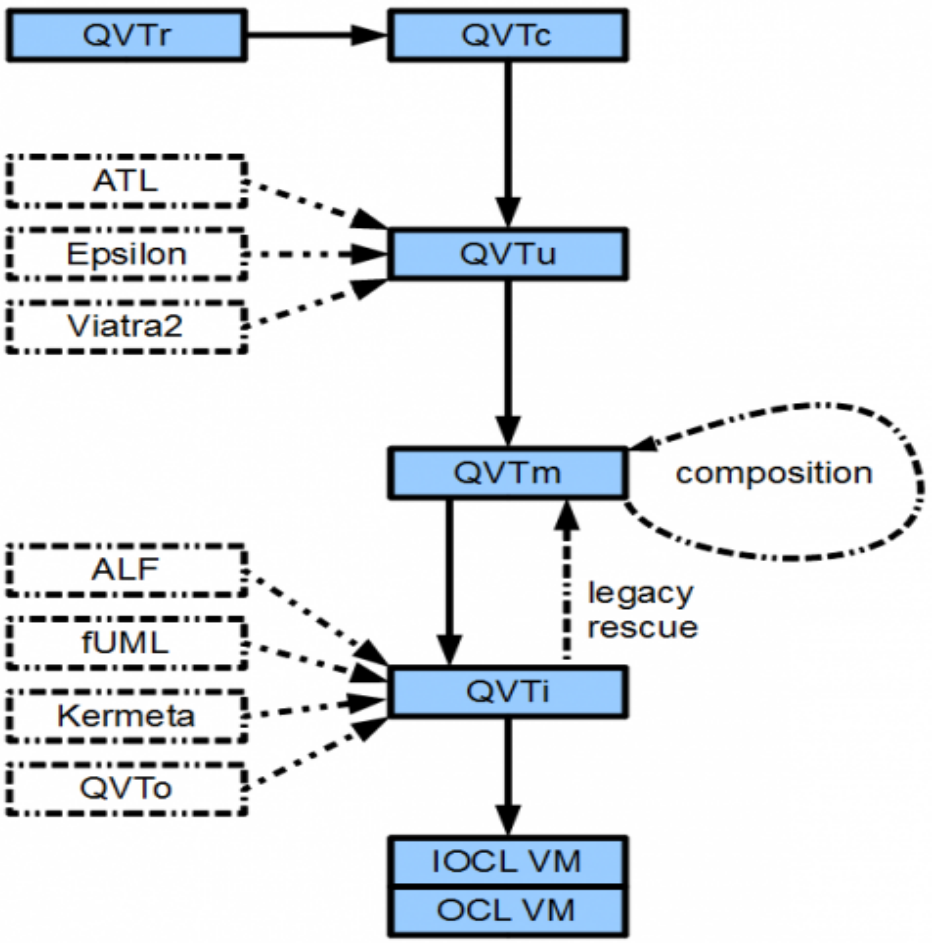

```
    }

    relation CopyIfcPropertyEnumeratedValue {

        enforce domain source prop : IfcPropertyEnumeratedValue {
                EnumerationValues = nominalValue : IfcValue { }
        }

        enforce domain target def : PropertyDef {
            ValueDef = valueDef : ValueDefType {
                DefaultValue = DefaultValue : DefaultValueType {
                    Value = getStringFromIfcValue(nominalValue)
                }
            }
        }

    }

    relation MapPropertySet {

        enforce domain source pset : IfcPropertySet {
            name = psetName
            HasProperties = prop : IfcSimpleProperty {
                name = attName
            }
        }

        enforce domain source artifact : Artifact { }

        enforce domain target aPset : PropertySetDefType { }

        enforce domain target artifact : Artifact {
            contents =  aPset : PropertySetDefType {
                name = psetName:String {}
                PropertyDefs = propertyDefsType : PropertyDefsType {
                    PropertyDef = def : PropertyDef {
                        name = attName :String {}
                    }
                }
            }
        }

        where {
            propertyDefsType = aPset?.propertyDefs;
            def = propertyDefsType?.propertyDef?.findFirst(x|x.name.contains(attName));
            if (def.definition.isEmpty && prop.description!=null) def.definition.add(prop.description);
            if (aPset.definition==null && pset.description!=null) aPset.definition = pset.description;
            if (!CopyIfcPropertyEnumeratedValue(prop,def) && !CopyIfcPropertySingleValue(prop,def)) throw new RuntimeException("could not match "+prop.eClass.name);
        }

    }

}
```

Tools

- JTL:
  - The Janus Transformation Language.
  - Bidirectional transformations: changes in one model propagated to the other model.
  - If a change makes the other model inconsistent, an approximation (“closest match”) is computed.
  - Based on answer set programming: as such, there can be several solutions to a transformation (results may need to be constrained).

Tools

- Eclipse QVT
  - QVT-O is well defined and active (Eclipse M2M)
  - QVT-R and QVT-Core – work in progress (Ed Willink, York)



Tools

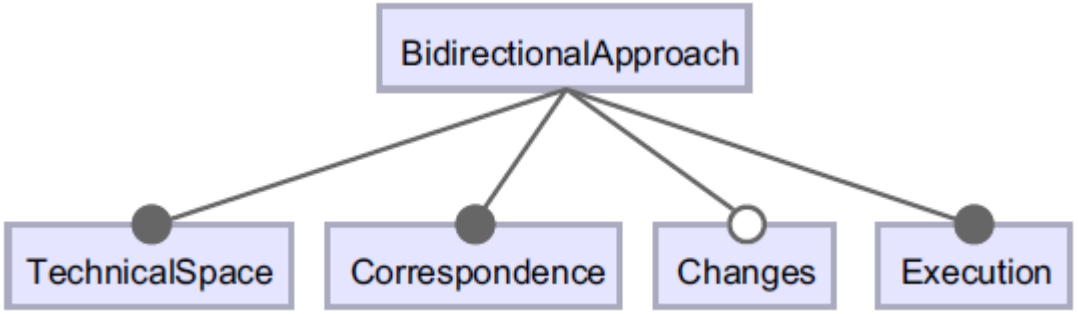
- XRound
  - Template-based bidirectional (asymmetric) language for XML-based models.
- XSugar
  - BX between two syntaxes for the same language; bijective.
- Epsilon (EWL+EVL) event-driven approach
  - Two update-in-place transformations (on source, target) executed when inter-model consistency rules are violated – later this week.

Tools

- Bidirectionalisation approaches:
  - ATL-Inversion: a HOT that generates an ATL backward transformation given a forward one (prototype).
  - Semantic bidirectionalisation, e.g., Voigtlander (based on collecting extensional behaviour) – see POPL'09 paper.
  - GRoundTram, based on a bidirectional interpretation of a graph query language (see ICMT'11 paper)

Hot Off the Press

- Excellent feature model-based classification of BX approaches in July 2016 issue of SOSYM.
- Main points of variation: technical space, correspondence, changes, execution.



Challenges

- MDE for BX does possess some clear theory and pragmatic tools
  - rather siloed (TGG+delta lens work is best example)
  - tools that evidently implement the theory?
  - all manage trace-links in a systematic way [delta lenses]
- The importance of metamodelling for understanding BX (typing is important).
  - Do we yet have a good “type theory” for MDE and metamodelling? (see, e.g., work by Steel and Jezequel, plus others)

Challenges

- How can theory from other communities be applied in understanding, improving and simplifying standard languages for BX in SE
  - QVT is still too complex (now we're in a world where we must distinguish Eclipse QVT and OMG QVT)
- More scenarios for bidirectionality:
  - When do we really need it?
  - In my industry projects, I have only had one precise requirement for a "BX" (reflect analysis results).
  - BX requirements sometimes emerge (discussion with Anthony on Monday)

Challenges

- MDE practitioners tend to use operational transformations a lot
  - and model-to-concrete-syntax transformations
  - and model-to-text transformations
- How do these fit into the BX space?
- We use metamodels but PL community seems to do fine with grammars.
  - Lessons to be learned? Value added?
- We have a diverse set of modeling languages that need to be supported.

Challenges

- How do we combine/unify:
  - {Delta lens, …} theory
  - MDE languages that support BX (QVT-R, OCL, Ecore) that both exist and are used.
  - Support for traceability (trace-links as a side effect of running model management operations)
    - both as a pragmatic tool and as a theoretical basis.

Next time!

- Requirements engineering for BX:
  - How do we gather requirements for a BX
  - Are standard requirements engineering processes applicable? (preview: yes!)
  - How do we specify requirements for BX?
    - *trans*ML.
  - Challenges for RE for BX.