Model Driven Architecture
Classification of Model Transformation Approaches

Prof. Dr. Peter Thiemann

Universität Freiburg

21.06.2006

1. Classification of Model Transformation Approaches
2. Categories
   - Model To Code
   - Model To Model
3. Status of QVT

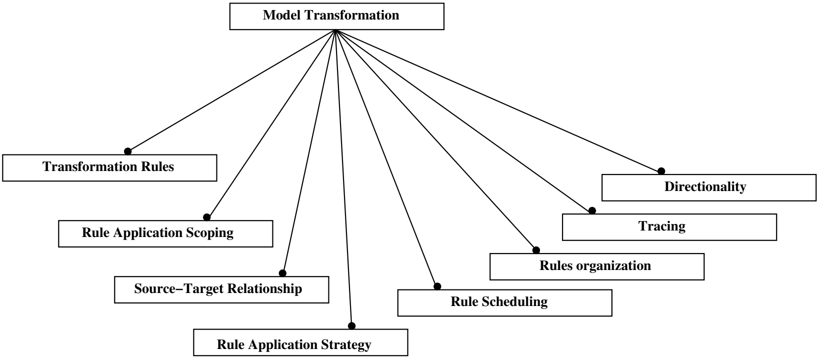
Classification of Model Transformation Approaches

- Survey and categorization
- Feature model to compare different approaches
- Applying domain analysis to the following input data:
  - published in literature: GreAT, UMLX, ATOM, VIATRA, BOTL, ATL, relational, oo logic programming
  - submitted to OMG: QVTP, CDI (CBOP, DSTC, IBM), AST+ (Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Corporation, ...), IOPT (Interactive Objects, Project Technology), CS (Compuware Crop and Sun Microsystems)
  - open-source MDA tools: Jamda, AndroMDA, JET, FUUT-je, GMT
  - commercial MDA tools: OptimaJ, ArcStyler, XDE, Codagen Architect, b+m Generator Framework

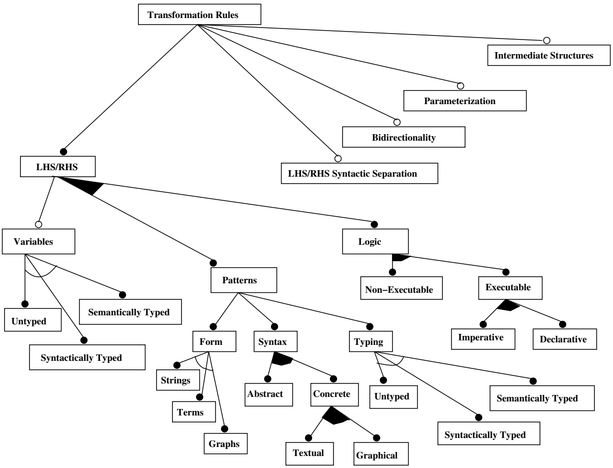
From: Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. OOPSLA 2003

Workshop on Generative Techniques in the Context of Model-Driven Architectures. 2003.

Design Features of Model Transformation Approaches



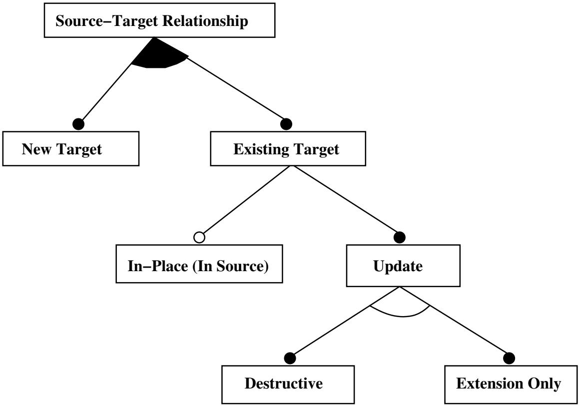
Transformation Rules



Transformation Rules/2

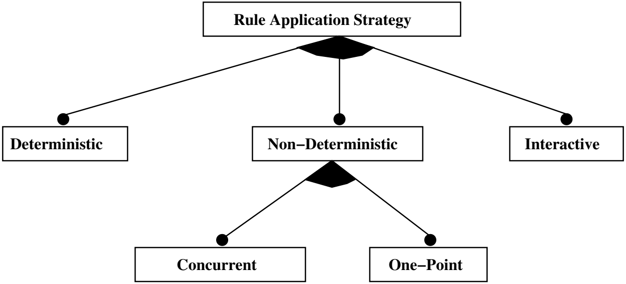
- General form of a rule: LHS $\rightarrow$ RHS
- LHS $\leftrightarrow$ source model; RHS $\leftrightarrow$ target model
- Representation
  - (Meta)*Variables* range over model elements
  - *Patterns* are model fragments with variables
  - *Logic*: computations and constraints on model elements
- Typing
  - syntactic: restricted to instances of a metamodel element
  - semantic: further model constraints

Source-Target Relationship



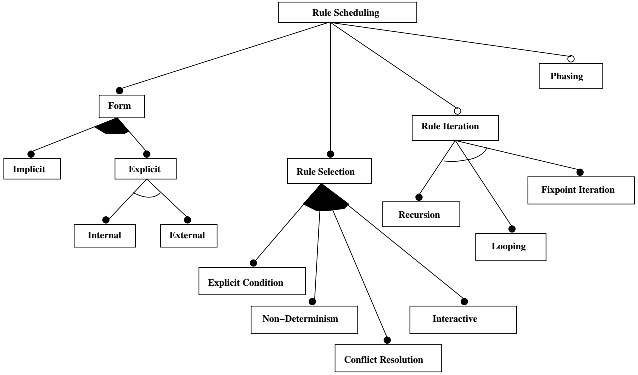
- separate target (CDI)
- only in-place update (VIATRA, GreAT)
- both possible (XDE)

Rule Application Strategy



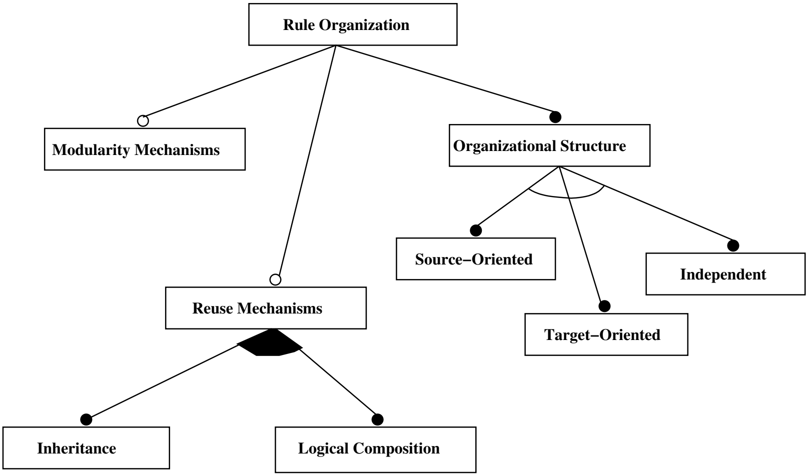
- Rule may match multiple times $\Rightarrow$ strategy required
- Traceability links

Rule Scheduling

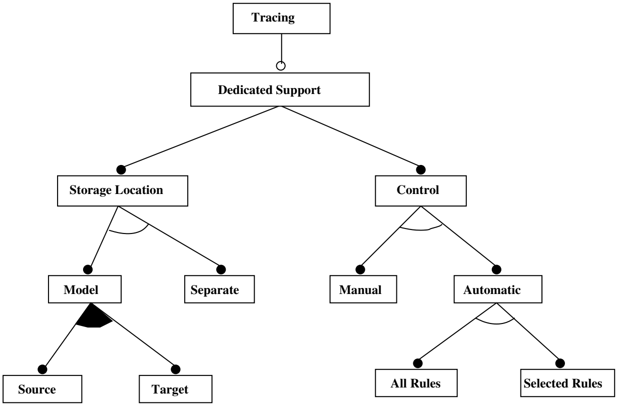


- Order of rule application
- Stratego is a language for expressing strategies and scheduling for term rewriting

Rule Organization

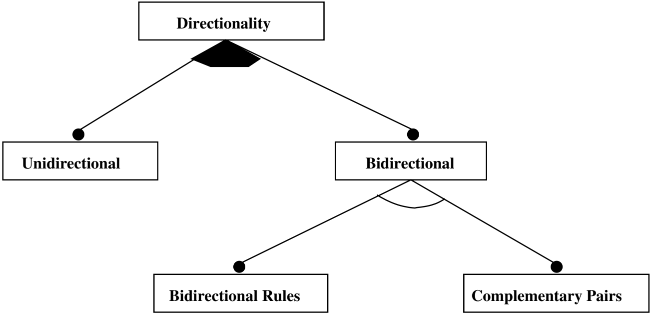


Traceability Links



- some systems expect transformations to encode traceability by themselves
- separate storage preferable

Directionality



- declarative rules more likely to be invertible
- lack of injectivity a problem
- enumerating solutions, establish part of result

Categories of Model Transformations

- model-to-code (model-to-text)
  - special case, but no metamodel; code-as-text
  - also documentation templates, XML
  - visitor vs. template
- model-to-model
  - direct manipulation
  - relational
  - graph transformation
  - structure driven

Model To Code

Visitor Based

- traverse internal representation of model; write to text stream
- Jamda (cf. OAW)

Model To Code

Template Based

- majority of tools
- template consists of
  - target text
  - splices of *metacode*
    - access source information
    - code selection
    - iteration
- often user-defined scheduling through explicit template calls
- “LHS” implicit in access logic
  - Java code
  - declarative queries (OCL, XPath)
- structure of generated code
- no check for syntactic or semantic correctness
- but independent of target language

Model To Model

- less frequently supported in tools
- but
  - intermediate models useful for bridging abstraction gaps
  - $\Rightarrow$ better modularity and maintainability
  - useful for optimization, tuning, debugging
  - generate different views

Model To Model
Direct Manipulation

- Internal representation plus API
- Some infrastructure
- but: transformation rules from scratch
- Jamda, OAW, using JMI

- declarative constraints with executable semantics
- connection to logic programming (matching, search, backtracking)
- QVTP distinguishes
  - relations (bidirectional, non-executable specifications)
  - mappings (unidirectional, executable implementations of relations)
- side-effect free
- strict separation between source and target

- rich theory of transformations on typed, attributed, labeled graphs
- instances: VIATRA, ATOM, GreAT, UMLX, BOTL
- rule:
  - LHS and RHS graph patterns
  - LHS: conditions
  - RHS: computed target elements
- concrete or (MOF) abstract syntax
  - concrete syntax much more concise
  - default abstract sytnax works for any metamodel
- LHS matched and replaced by RHS

Model To Model
Structure Driven

- transformation splits in two phases
  - create target hierarchy
  - set attributes and references in the target
- basic metaphor:
  - copying model elements from source to target
  - modify elements in between

- XMI is an XML language for serializing MOF models
- Q: why not use XML transformation for model transformation?
- A: scalability
  - XMI is unreadable and very verbose
  - XSLT is unwieldy (see C. Cleaveland, Program Generators with XML and Java)
  - generating XSLT from declarative spec is possible but
  - poor efficiency because of XSLT's call-by-value (copying) semantics

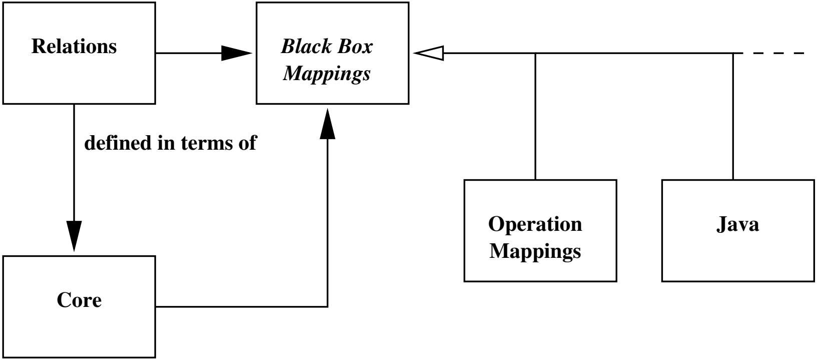
Status of QVT

- RFP April 2002
- Published spec November 2005 (but standardization not yet finished)
- (Code generation from MOF: new RFP April 2004, ongoing)
- Result for QVT:
  - **three different** QVT languages
  - only loosely connected
- Issue reporting closed in March 2006
- Finalization report expected in July 2006
- Tool developers encouraged to provide prototype implementations

Three QVT Languages

- *Relations*
  - declarative, using object patterns
  - creation and deletion of objects implicit
  - automatic trace management
  - graphical syntax
- *Core*
  - declarative, but no patterns
  - based on EMOF and OCL
  - define transformation and trace information as a MOF metamodel
  - too simple for practical use?
- *Operational Mappings*
  - imperative DSL
  - OCL as query language
  - extended with imperative features
  - two modes of use
    - all imperative
    - hybrid with some aspects in Relations or Core

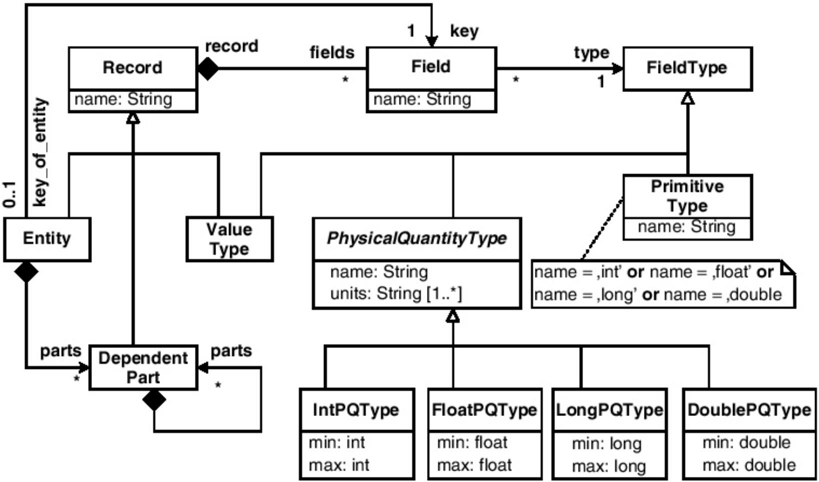
Relationship of the QVT Languages



- M2M Relations2Core (in Relations)
- (not useable for practical implementation)

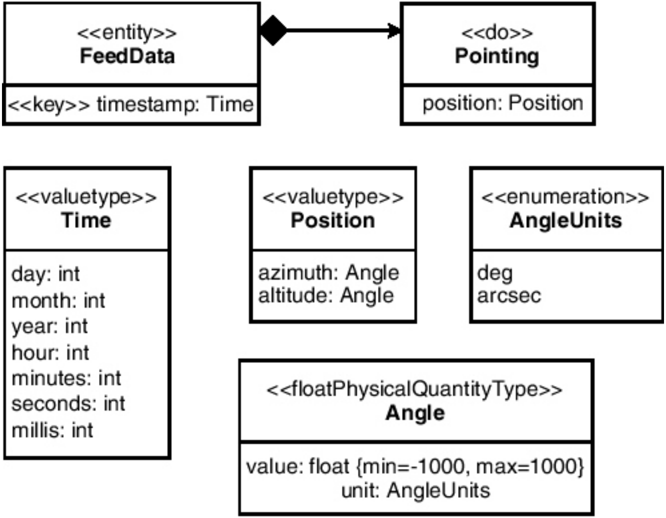
Example Use of QVT/Relations

Source Metamodel

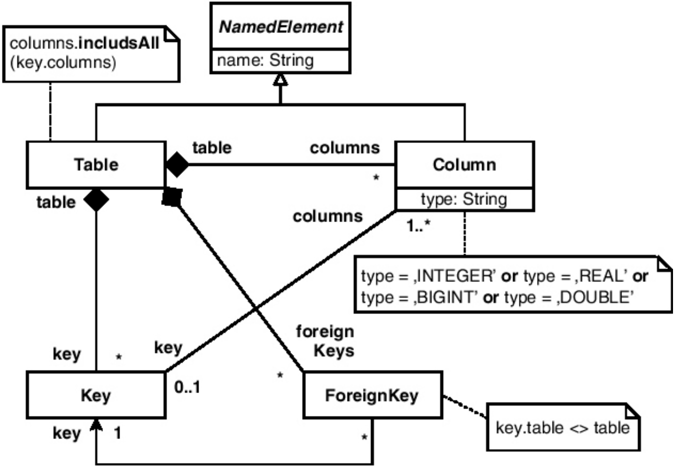


Example Use of QVT/Relations

Instance of Source Metamodel



Example Use of QVT/Relations
Target Metamodel: Database Tables



Example Use of QVT/Relations

Desired Transformation

- All fields of a record are mapped to one ore more columns depending on the field type:
  - Primitive type → one primitive typed column.
  - Value type → columns for each of its fields, where the name of the encompassing field is propagated to disambiguate the names of the nested fields.
  - Physical quantity → one column for each unit, where the name of the column incorporates the unit name and its type is that of the concrete physical quantity.
- Each ALMA entity is mapped to a DB table:
  - All its fields lead to columns, as described before.
  - Its key leads to the table key.
- Each ALMA-dependent part that is owned by an entity is mapped to a DB table, where the name is a concatenation of the entity name and the dependent part name:
  - Fields → columns, as described before.
  - Surrogate key of type INTEGER.
  - The surrounding table for the entity refers to the dependant through a foreign key.

Example Use of QVT/Relations
Generated Tables from Instance

```
1. Table FeedData:
   key timestamp_day : INTEGER
   key timestamp_month : INTEGER
   key timestamp_year : INTEGER
   key timestamp_hours : INTEGER
   key timestamp_minutes : INTEGER
   key timestamp_seconds : INTEGER
   key timestamp_millis : INTEGER
   fk key_FeedData_Pointing : INTEGER
2. Table FeedData_Pointing:
   key key_FeedData_Pointing : INTEGER
   position_azimuth_as_Angle_in_deg : REAL
   position_azimuth_as_Angle_in_arcsec : REAL
   position_altitude_as_Angle_in_deg : REAL
   position_altitude_as_Angle_in_arcsec : REAL
```

```
transformation alma2db(alma : AlmaMM, db : DbMM) {
...
}
```

- Transformation execution
  - verify specified relations
  - modify target model
- direction specified on invocation

Example Use of QVT/Relations

```
top relation EntityToTable {
    prefix, eName : String;
    checkonly domain alma entity:Entity {
        name = eName
    };
    enforce domain db table:Table {
        name = eName
    };
    where {
       prefix = '';
       RecordToColumns(entity, table, prefix);
    }
}
```

Example Use of QVT/Relations

```
relation RecordToColumns {
    checkonly domain alma record:Record {
        fields = field:Field {}
    };
    enforce domain db table:Table {};
    primitive domain prefix:String;
    where {
       FieldToColumns(field, table);
    }
}
```

Example Use of QVT/Relations

Cross-Domain Constraints

```
top relation EntityKeyToTableKey {
  checkonly domain alma entity:Entity {
     key = entityKeyField:Field {}
  };
  enforce domain db table:Table {
     key = tableKey:Key {}
  };
  when {
    EntityToTable(entity, table);
  }
  where {
    KeyRecordToKeyColumns(entityKeyField, table);
  }
}
```

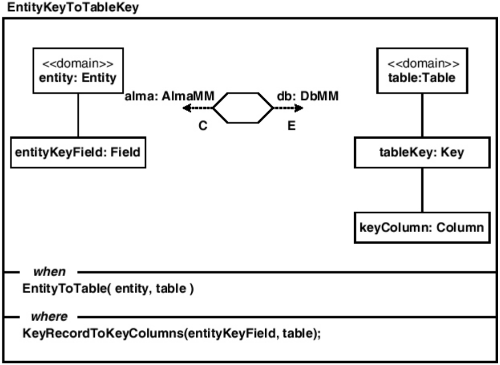
Example Use of QVT/Relations

Auxiliary Functions

```
function
  AlmaTypeToDbType(almaType : String) : String {

  if (almaType = 'int') then 'INTEGER'
  else if (almaType = 'float') then 'REAL'
  else if (almaType = 'long') then 'BIGINT'
  else 'DOUBLE'
}
```

Example Use of QVT/Relations
Graphical Notation



- Graphical notation extending UML Object Diagrams
- Current specification incomplete (primitive domains)
- Does a graphical notation make sense?

Conclusions

- Three languages $\rightarrow$ well-understood domain?
- Requirements are not yet completely known
- Current languages do not address all known requirements
- Bidirectional mappings required?
- Transformation development requires (as yet non-existent) tools
- Complex specification (too many people involved, too much time)
- Standard compliance? (Neither test-suite nor reference implementation)