New Advances In MDE

Jordi Cabot
INRIA & École des Mines de Nantes,

Nantes, France

http://www.emn.fr/x-info/atlanmod/

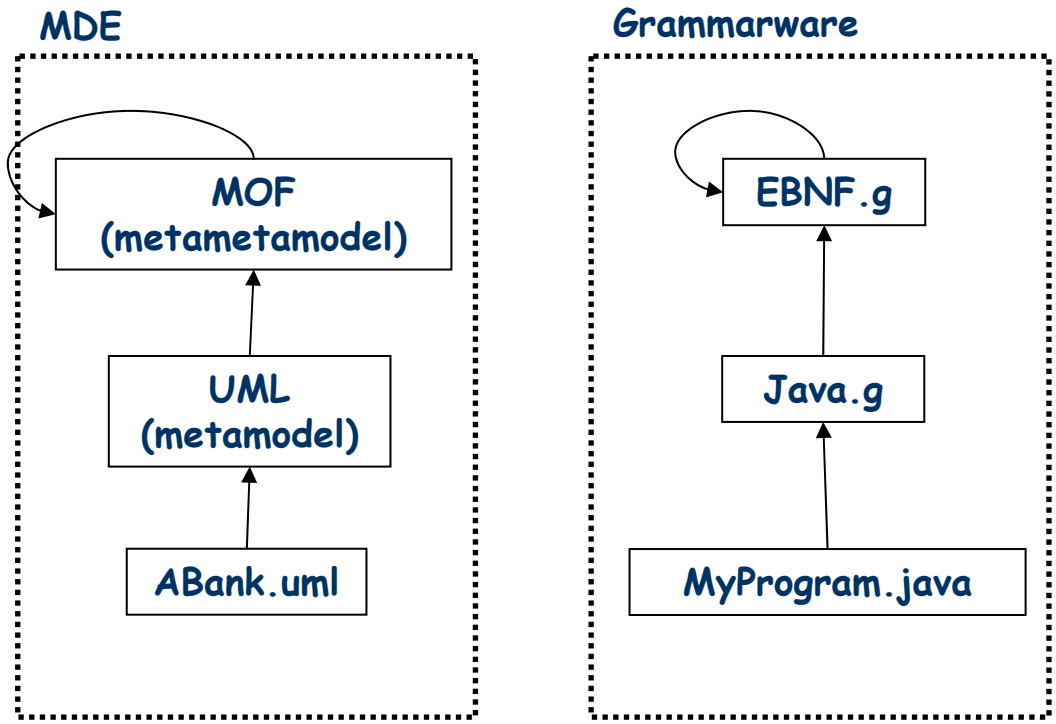
The Team

- AtlanMod is a joint INRIA – École de Mines de Nantes research team
- Participates in several national and european projects
  - OPEES, CESAR, GALAXY, IDM++,...
- Strong links with industrial partners: Obeo, Mia-Software, BNP Paribas, Prodevelop...
- Permanents (4) + Non-permanents (8)
  - Jordi Cabot (faculty)
  - Frédéric Jouault (faculty)
  - Massimo Tisi (faculty)
  - Hugo Brunelière (research engineer)

The Focus of the Team: MDE

- We emphasize the use of models as key artifacts in all software engineering activities. Because this more abstract view:
  - Improves the productivity
  - Reduces the number of defects
  - Facilitates the reusability, evolution and maintenance
  - Increases the decomposition and modularization
  - …
- This “new” paradigm is called MDE (model-driven engineering or “model-driven everything”)
- MDE goes beyond MDD (model-driven development) that mainly focused on code-generation as the main activity
- MDE involves MDD but also: MDReengineering, collaborative development, system adaptation and evolution,…

Technical Spaces



At the core: Model Transformations – ATL

We can transform models: ATL

- ATL : ATLAS Transformation Language . ATL is a language and a virtual machine dedicated to model to model transformations

- A model transformation is the automatic creation of target models from source models.

- E.g., we can use ATL to transform analysis models to design models (e.g. UML class diagrams -> relational models)

- ***Everything is a model: ATL transformations are also models and can be manipulated as such <-Same for all other proposals***

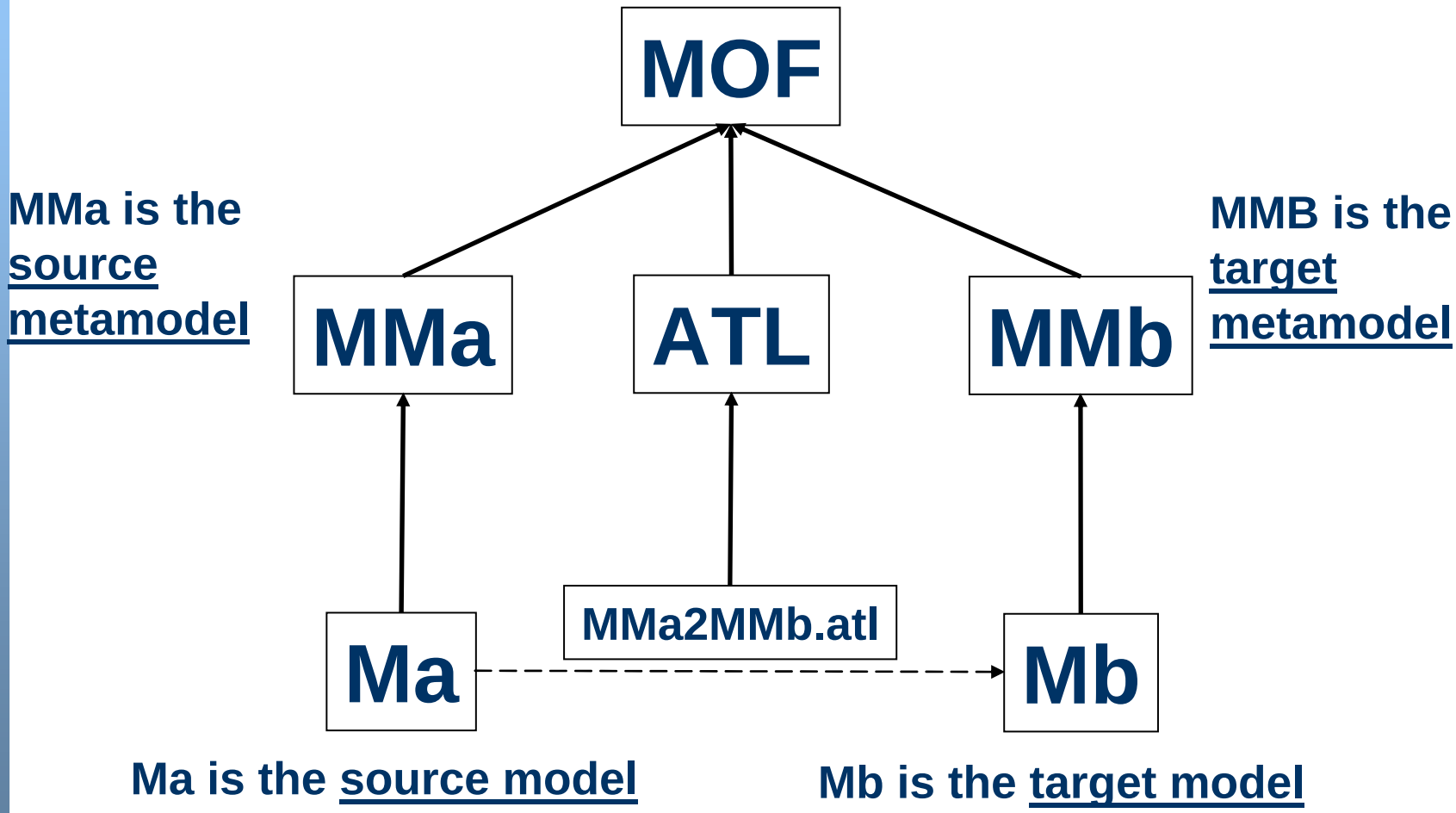
ATL

- The language is a declarative-imperative hybrid:
  - Declarative part:
    - Matched rules with automatic traceability support,
    - Side-effect free navigation (and query) language: OCL 2.0
  - Imperative part:
    - Called rules,
    - Action blocks.

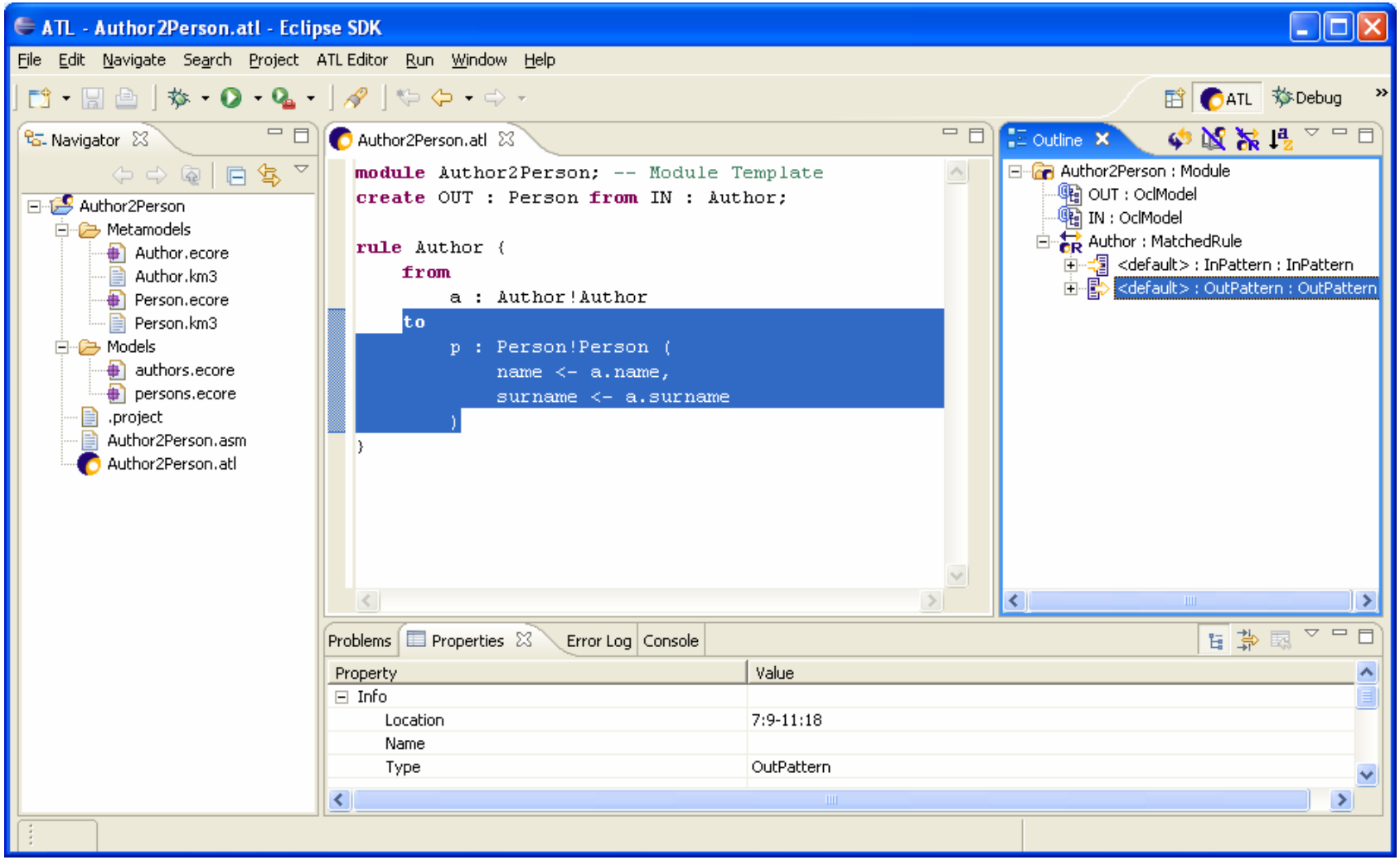
- Core principle: Read only source models – write only target models
  - Difference wrt Graph Transformations

ATL

Operational context of ATL

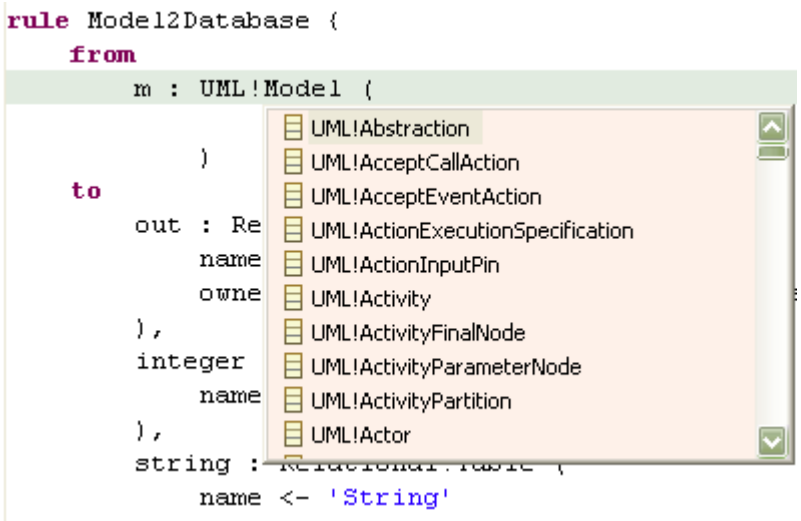


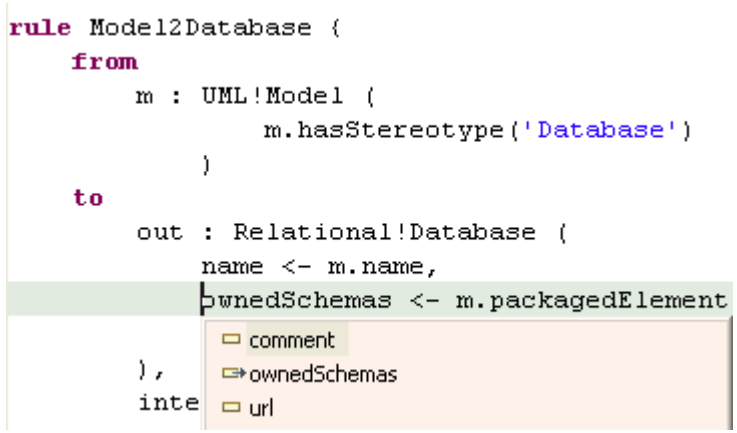
ATL Editor


ATL - Author2Person.atl - Eclipse SDK
File Edit Navigate Search Project ATL Editor Run Window Help
ATL
Debug
Navigator
Author2Person
Metamodels
Author.ecore
Author.km3
Person.ecore
Person.km3
Models
authors.ecore
persons.ecore
.project
Author2Person.asm
Author2Person.atl
Author2Person.atl
module Author2Person; -- Module Template
create OUT : Person from IN : Author;
rule Author {
from
a : Author!Author
to
p : Person!Person (
name <- a.name,
surname <- a.surname
)
}
Outline
Author2Person : Module
OUT : OclModel
IN : OclModel
Author : MatchedRule
<default> : InPattern : InPattern
<default> : OutPattern : OutPattern
Problems
Properties
Error Log
Console
Property
Value
Info
Location
7:9-11:18
Name
Type
OutPattern

ATL Editor

- ATL Editor features:
  - Type completion
  - Left-side bindings completion
  - Basic code templates





Model transformations: Open Challenges

- Better support for HOT (High-Order Transformations): transformations that manipulate transformations
- Bidirectionality (derive B->A if I define A->B)
- Incrementality/synchronization (incrementally change the target model after updates on the source model)
- Transformations on infinite models (streaming, lazy evaluation):
- Optimization (parallel execution)
- Composition and transformation chains
- ...

Relating Models – AMW and AML

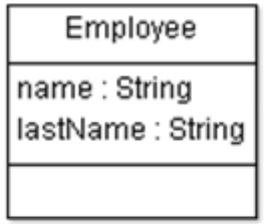
Beyond Model transformations

- Transformations (even if declaratively specified) are still a low-level manipulation technique
- We may be interested in expressing relationships between models and model elements
  - Ex. to express that an element *X* is a refinement of an element *Y* in a different model
- Some of these relationships can be automatically derived (matching elements in both models using heuristics)
- We can reuse this relationships to generate transformations

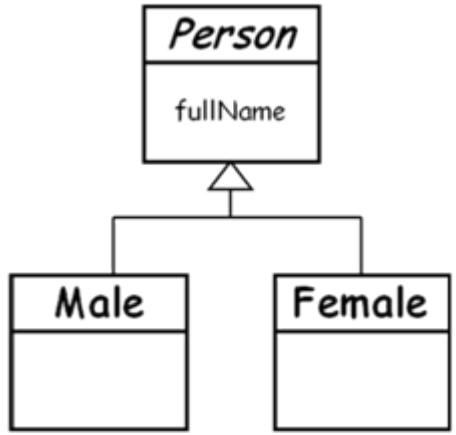
Motivating example:

- Enterprise 1 and 2 have been joined. Enterprise 1 has to Migrate its RH database schema to the Enterprise 2's.

Schema 1

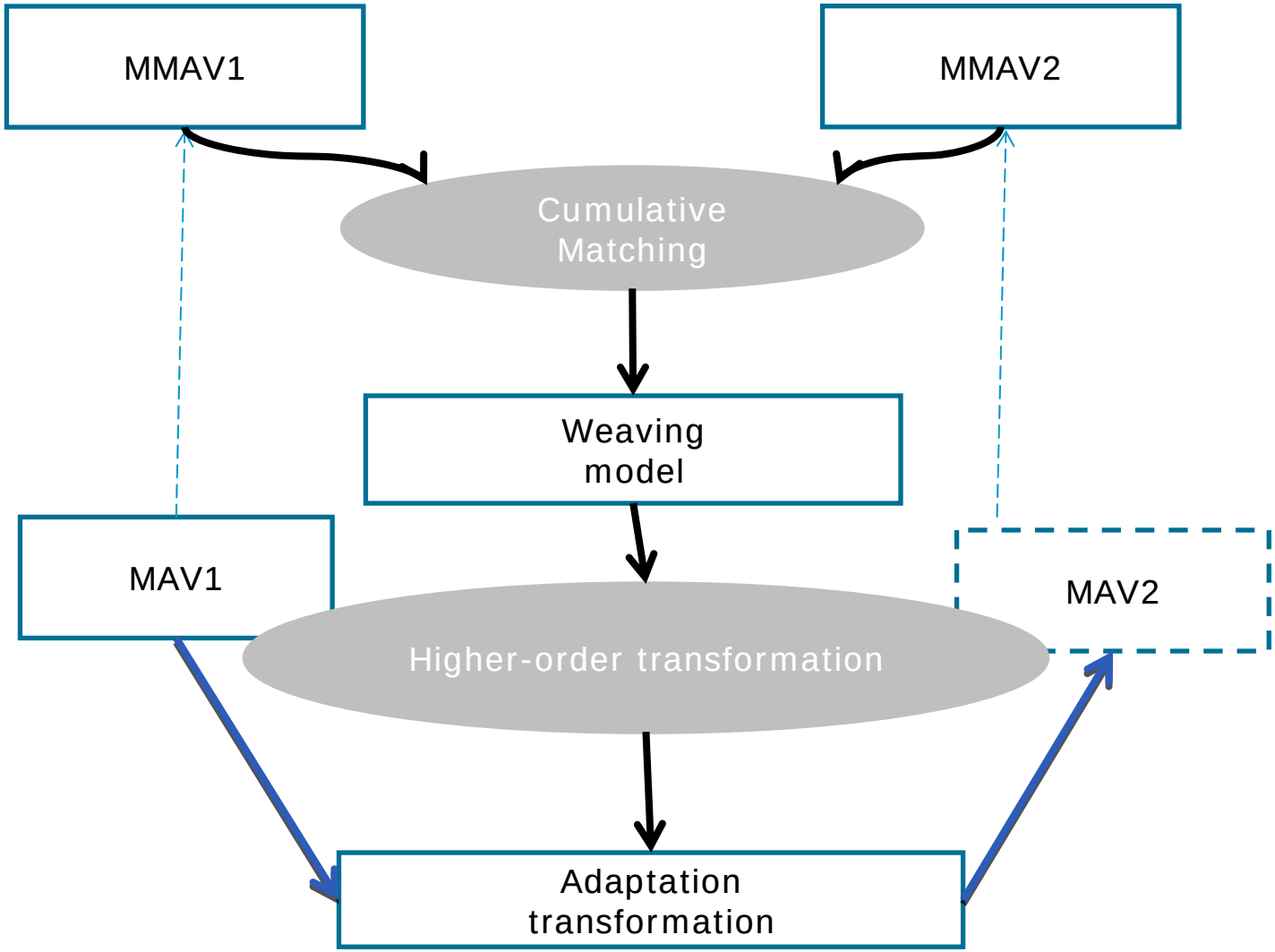


Schema 2



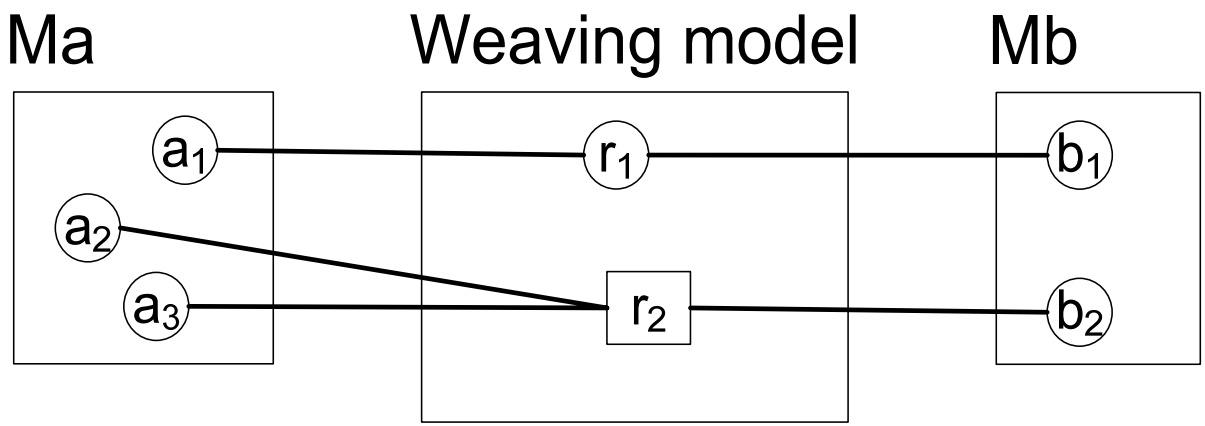
- ***Do we have a better migration strategy than simply writing an ATL Transformation?***

Concepts in mind



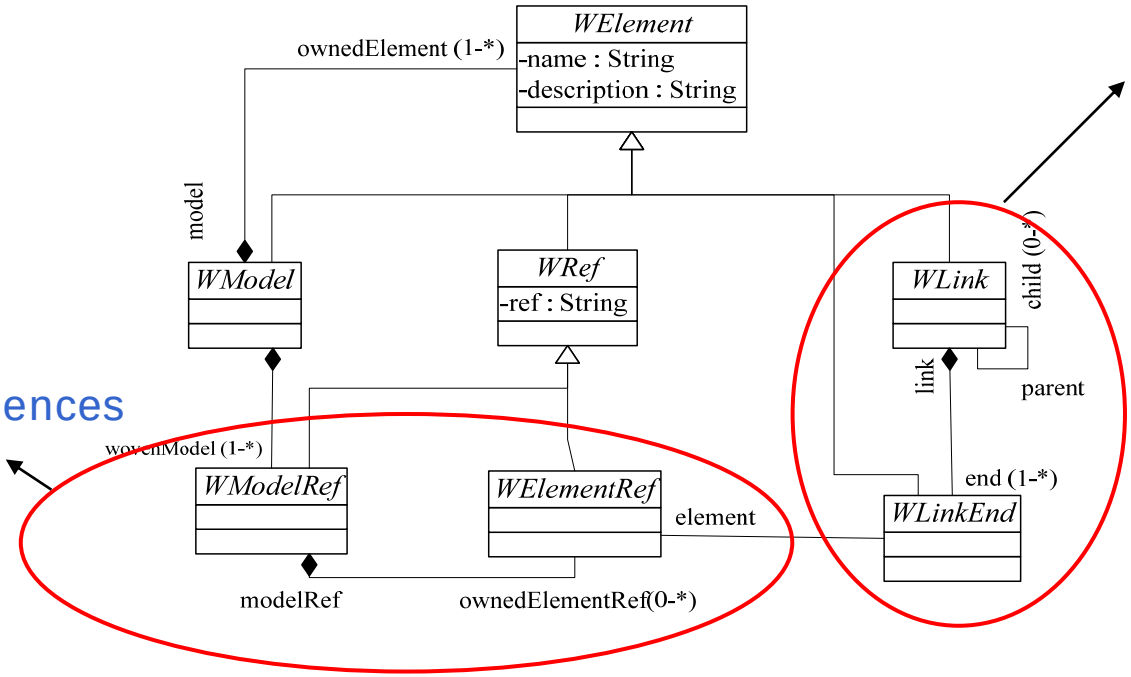
Weaving model

- Solution to capture relationships between (meta)model elements
- Relationships are represented by a **weaving model**
  - The model elements in the weaving model represent the relationships between the related elements
  - As any kind of model, the weaving model can be saved, stored, transformed, modified
  - Different kinds of links
    - Equality, concatenation, equivalence, etc.



Core weaving metamodel

- Supports basic link management

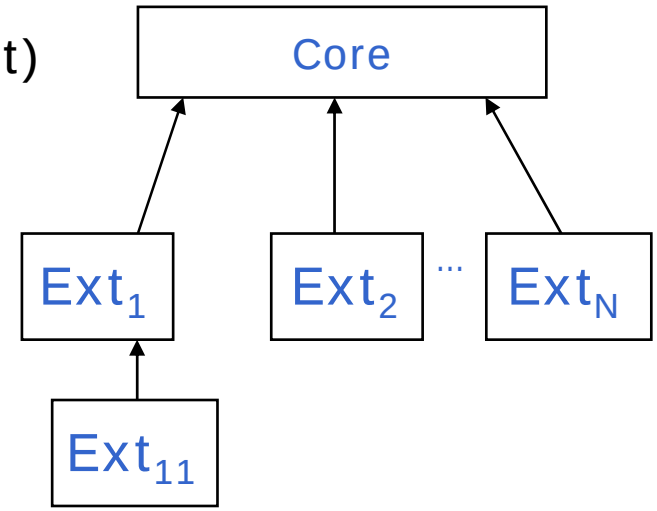


How to link model elements (one link has N endpoints)

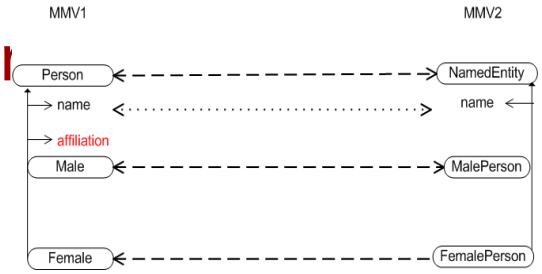
External references

Weaving metamodel extensions

- Core is extended with different kinds of relationships
  - Comparison
    (Equivalence, Addition, Delete)
  - Traceability
    (source generates target)



AMW: Model Weaver



time.txt | Class2Relational-... | Persons_Comp_cp_n... | Persons_Comp_cp_n... | Persons_Comp_cp_n...

leftM

- EPackage Persons
  - EClass Person
    - EAttribute fullName
  - EClass Male
  - EClass Female
- EPackage PrimitiveTypes
  - EData Type String

Weaving model

- platform:/resource/Persons2PersonsNew/match/Persons_Comp_
- Match Model
  - <ownedElement> Element Equal sim: 1
    - <child> Element Equal sim: 0,341
      - <<left>> Left Element Male
      - <<right>> Right Element MalePerson
    - <child> Element Equal sim: 0,41
      - <<left>> Left Element Female
      - <<right>> Right Element FemalePerson
    - <child> Attribute Equal sim: 0,694
      - <<left>> Left Element Person
      - <<right>> Right Element NamedEntity
  - Method Cartesian product
  - Method string comparison - Levenshtein distance
  - Method Propagation weaving model
  - Method Similarity flooding
  - Method Threshold selection
  - Method Link rewriting

rightM

- EPackage Persons
  - EClass NamedEntity
    - EAttribute fullName
  - EClass MalePerson
  - EClass FemalePerson
- EPackage PrimitiveTypes
  - EData Type String

Plugged panels

Adaptive interface

Problems | Properties | Error Log | Console | Package Explorer

Property	Value
_LeftElement	
Description	
Element	<ownedElementRef> Element Ref Person
Link	
Name	Person
ElementRef	
Description	
Model	
Model Ref	<leftM> Model Ref leftM
Name	Person
Ref	/0/Person

Identification mechanism

Matching (Ontology, Schemas)

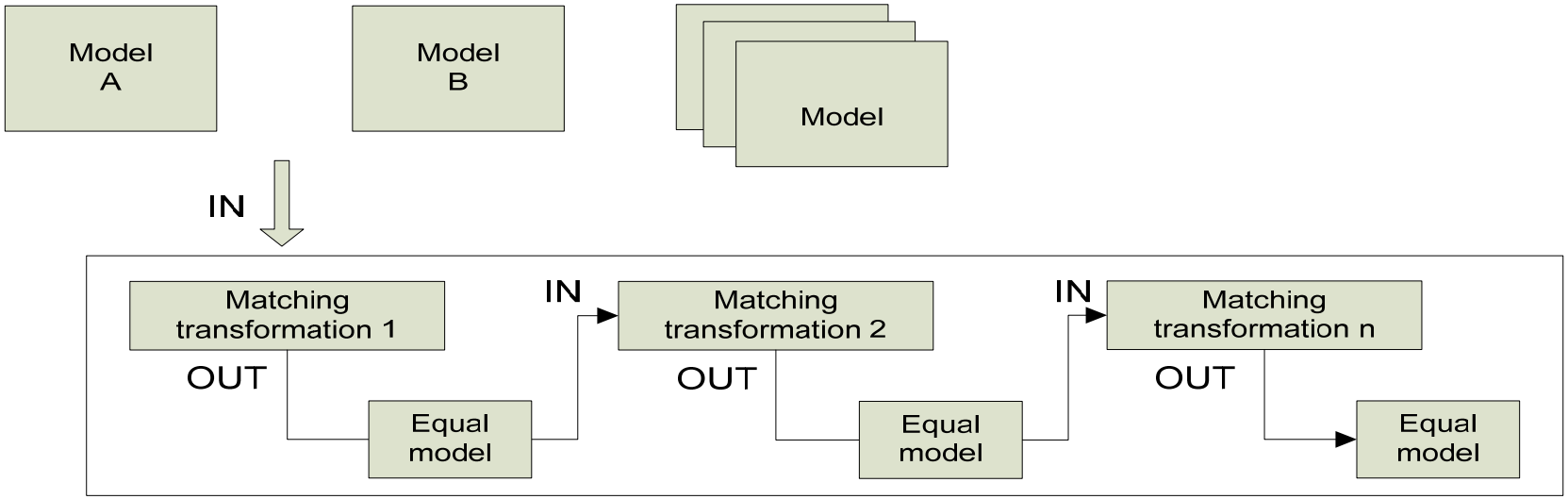
- Identifies ***mappings***, i.e., equivalence relations between two elements, e.g., *a* and *b, a* in a model *A*, and *b* in a model *B*
- A set of heuristics (i.e., a strategy) is needed to compare/align models
- Goal:
  - improve heuristics for (different kinds of) models
  - Minimize the effort of writing matching strategies

AML

- The AtlanMod Matching Language (AML)
  - A DSL for expressing strategies that match two models
  - Captures and mechanizes a significant portion of matching code
  - Aids to reason about matching strategy improvements

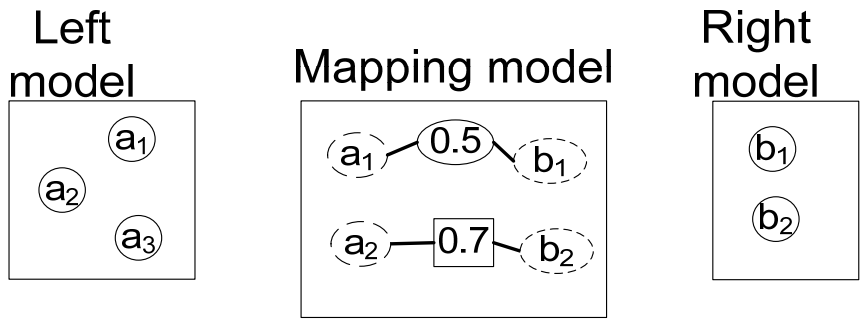
AML

- A matching strategy is a chain of matching model transformations
  - Each matching transformation
    - Instruments a heuristic
  - Takes as input a set of models
    - An equal model
    - A set of additional models
  - Yields an equal model



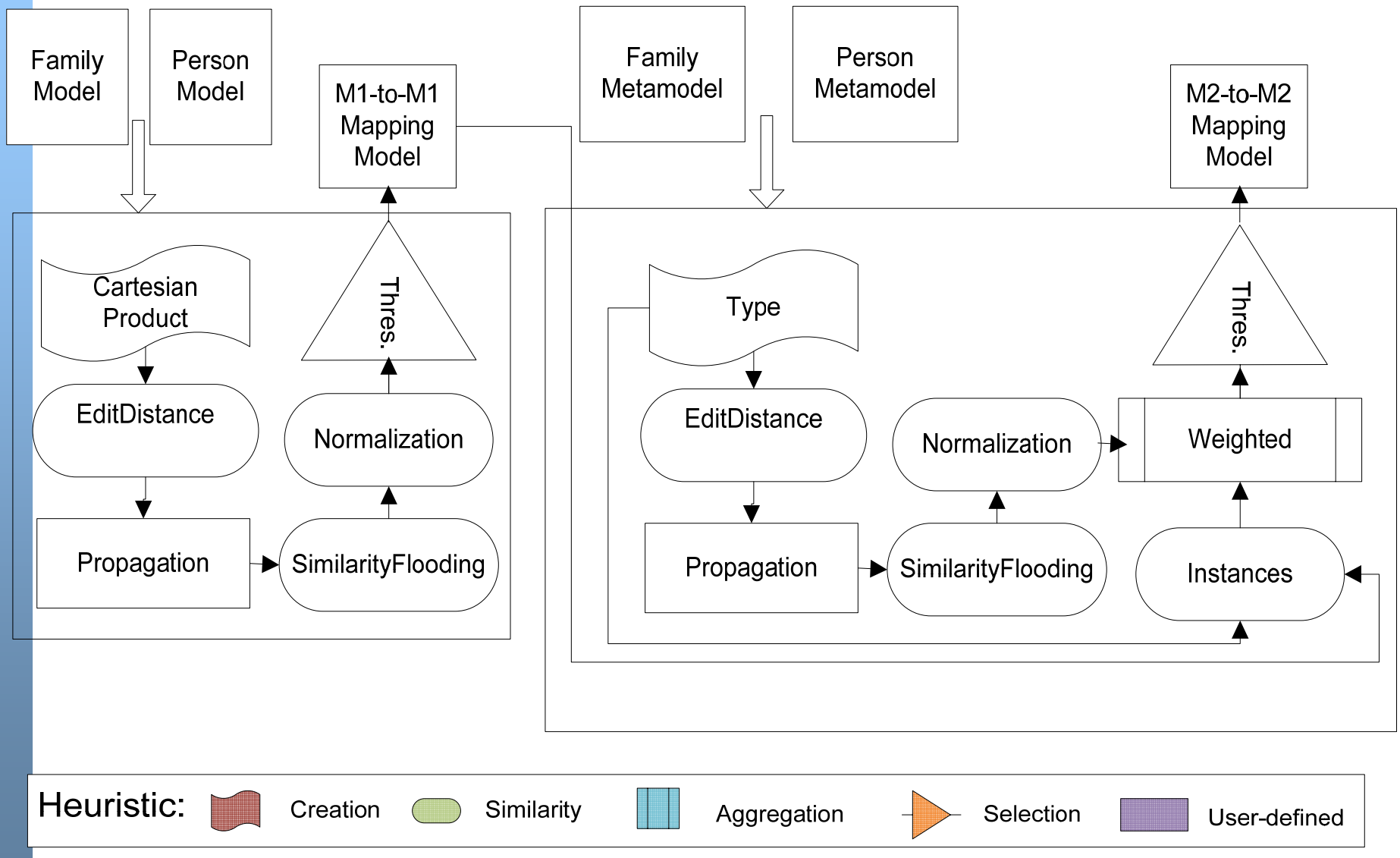
AML: Concepts (2/2)

- An equal model contains a set of mapping elements
  - An equal element references to
    - An element *a* (in a leftModel)
    - An element *b* (in a rightModel)
  - Has attached a similarity value [0-1]



- We have implemented several matching heuristics from research literature

Example: A strategy



KM3 – Creating your own domain specific language

UML vs DSLs

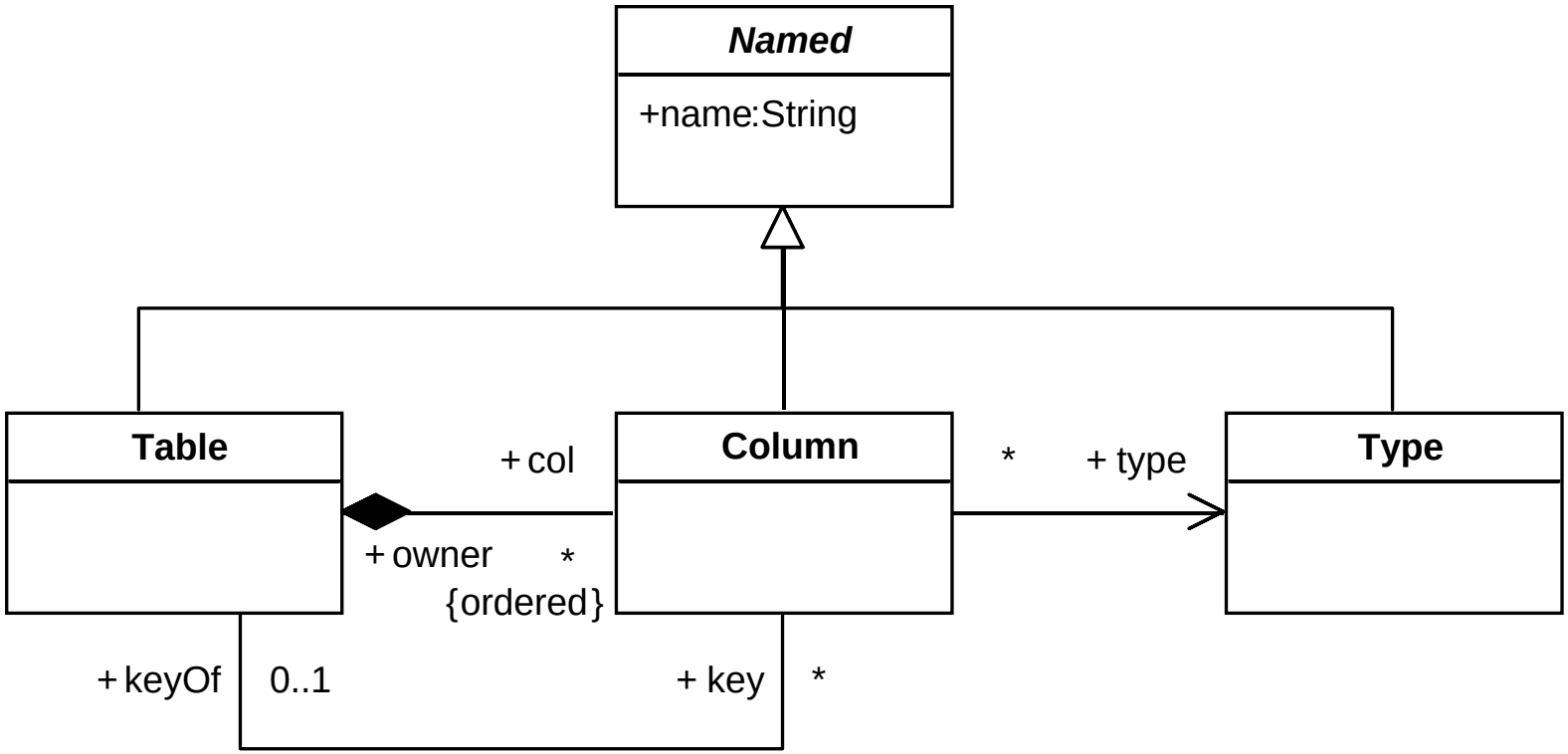
- DSLs are a very hot topic right now
- DSLs allow designers to create modeling languages that perfectly fit a given domain (e.g. smaller and with elements that better map the domain semantics)
- UML can be regarded as a collection of DSLs -> possible evolution of the UML
- UML or DSLs? My pragmatic approach -> Create a DSL **only** when UML + simple profiles do not fit

Overview of KM3

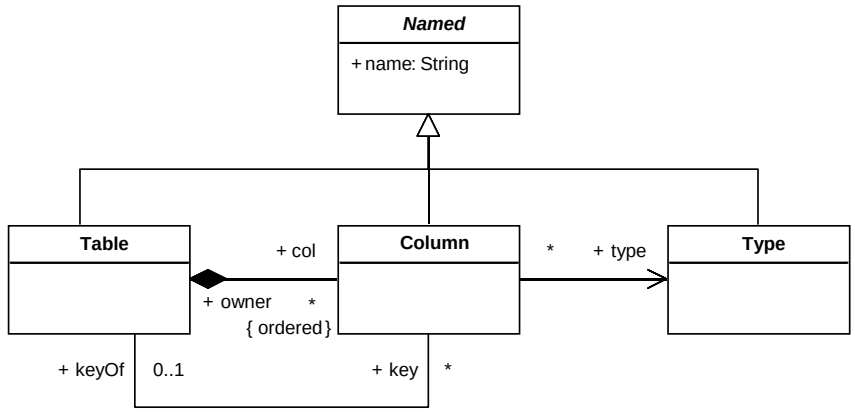
- KM3 is a metametamodel:
  - That is similar to MOF (OMG) and Ecore (Eclipse EMF),
  - That is simple (i.e. less concepts than Ecore or MOF 1.4),
  - With a simple textual concrete syntax,
  - With a precise definition based on first order logic.

- Core concepts:
  - Class used to type nodes of the model
    - Supporting class inheritance,
    - Owning references,
  - Reference used to type edges of the model:
    - Having an opposite reference

Example: a relational metamodel



Example: relational




```
package Relational {

        abstract class Named {
                attribute name : String;
        }

        class Table extends Named {
                reference col[*] ordered container : Column oppositeOf owner;
                reference key[*] : Column oppositeOf keyOf;
        }

        class Column extends Named {
                reference owner : Table oppositeOf col;
                reference keyOf[0-1] : Table oppositeOf key;
                reference type : Type;
        }

        class Type extends Named {}
}

[...]
```

KM3 in Use

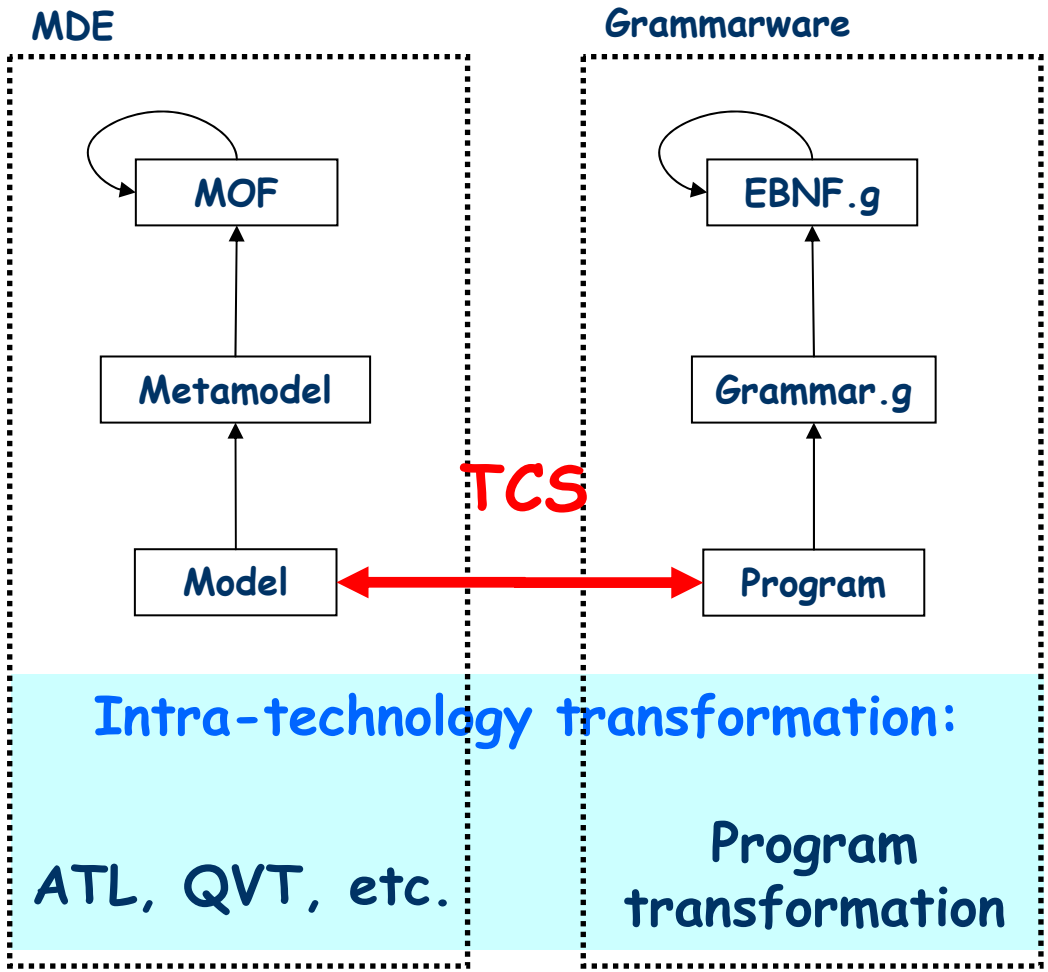
- KM3 used as a pivot metamodel
- KM3 can be used to represent various metamodels:
  - There is a library with already 234 metamodels in various domains (e.g., BibTeX, COBOL, DTD, HTML, Java)
- KM3 metamodels can be created from and translated to other MDE variants:
  - MOF 1.4, Ecore, MetaGME, Microsoft DSL Tools, etc.

TCS - Textual Syntax Specification

Textual Concrete Syntaxes

- EMF provides abstract syntax for models.
- Two main ways of displaying and editing models:
  - Graphical notations -> GMF used to develop editors for graphical notations
  - Textual notations -> TMF Project (TCS) to create editors for custom textual notations.
- Graphical notations are often useful to describe structural concepts (e.g. class diagrams)
- Textual notations are
  - Better fit for describing behavior or algorithms (e.g. expressions)
  - Gaster and easier to modify,
  - Rich set of existing tooling for dealing with text files (diff, merge, copy & paste, search & replace ...).

Textual Concrete Syntaxes



TCS Overview

- Declaratively specify textual concrete syntaxes for metamodels:
  - Customizable/user-friendly,
  - Without repeating what is in the metamodels,
- Automatically parse programs into models:
  - Keep tracing information (e.g. line number),
- Automatically serialize models into programs:
  - Pretty printing (automatic indentation),
- Provide a featured editor.

- ***XText has the same goal. Differences***

  - In Xtext the metamodel is derived from the grammar.

  - TCS binds a concrete syntax to an existing metamodel.

- ***MPS follows an innovative approach:***

  - No concrete syntax (even if you have this impression)

Example: excerpt from the definition of KM3 (in KM3 and TCS)

Metamodel excerpt:

```
abstract class ModelElement {
        attribute name : String;
        reference "package" : Package oppositeOf contents;
}

class Package extends ModelElement {
        reference contents[*] ordered container : ModelElement oppositeOf "package";
}
```

Excerpt of corresponding TCS model:

```
template ModelElement abstract;

template Package main context
        :       "package" name "{"
                        contents
                "}"
        ;
```

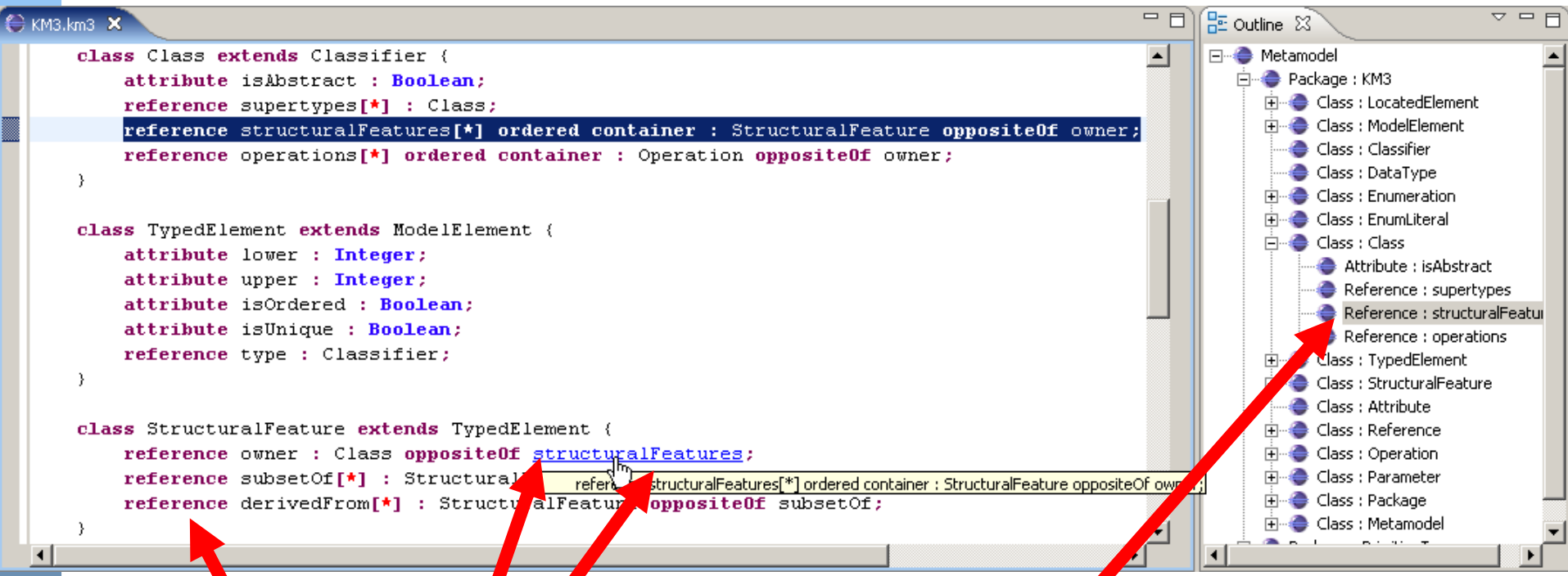
Excerpt of generated grammar:

```
modelElement : package_ ;

package_ : "package" identifier LCURLY
        (modelElement ( modelElement )* |)
        RCURLY ;
```

Textual Generic Editor for Eclipse

- Eclipse editor plugin:



- Hyperlinks
- Text hovers
- Syntax highlighting
- Outline with bidirectional synchronization

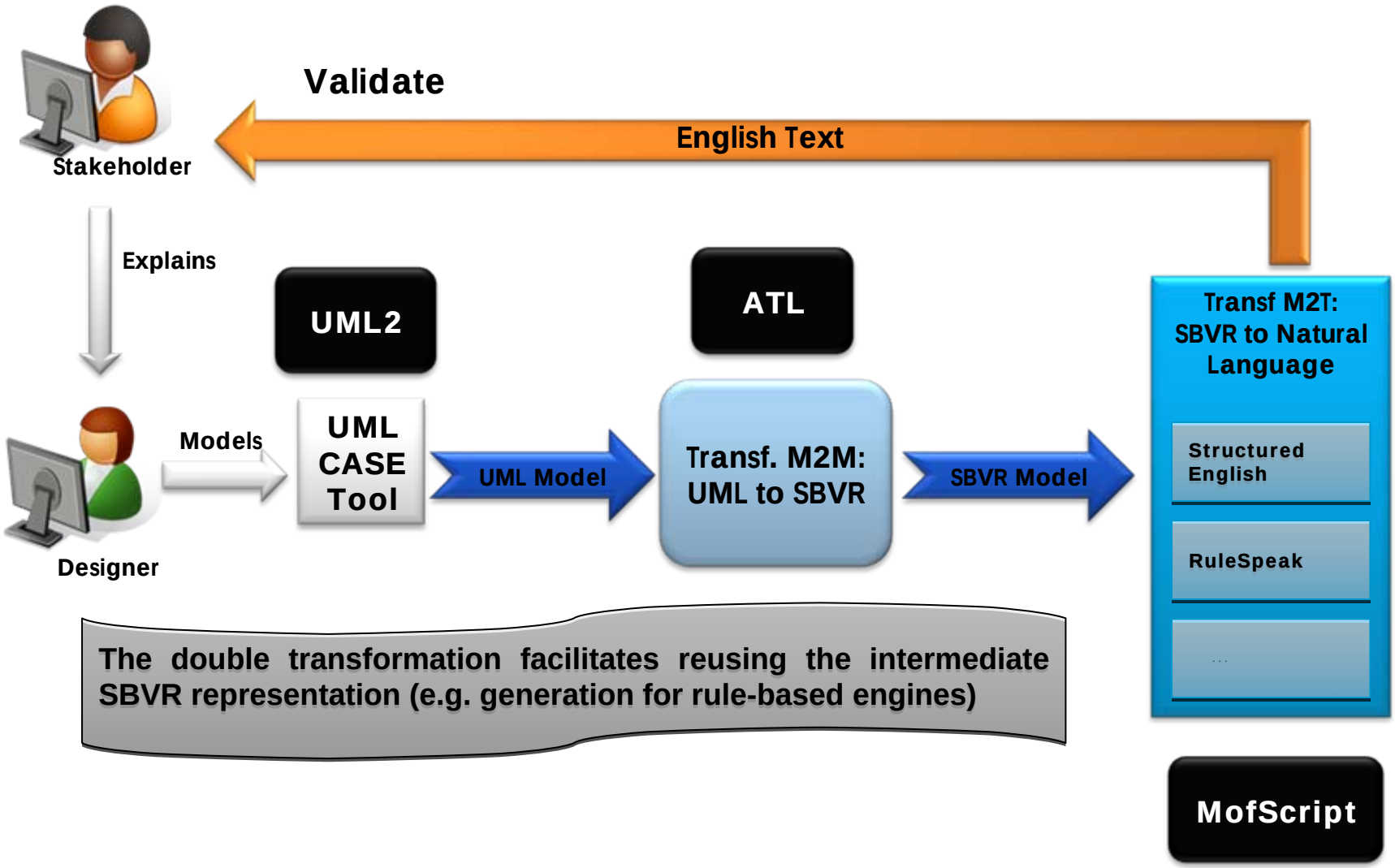
Validation and VerificatioN

(because it’s time to worry about the qualityf of our models!)

Verification vs Validation

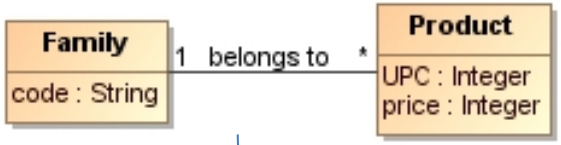
- V&V is a grand challenge for SE research
- Verification: Do the models right
- Validation: Do the right models
- Only stakeholders can validate (but not direclty the models)
- No good solution for verification (problem is EXP)

Validation of UML Models



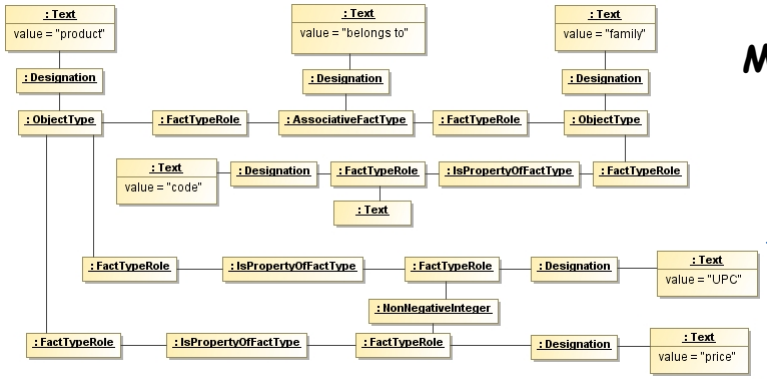
Validation of UML Models

Input UML Model



context Product inv: price>0

M2M ATL Transf.



MOFScript Transf.

SBVR intermediate representation

Structured English

product
Concept type: object type
Necessity: each price *of* product *is greater than* zero

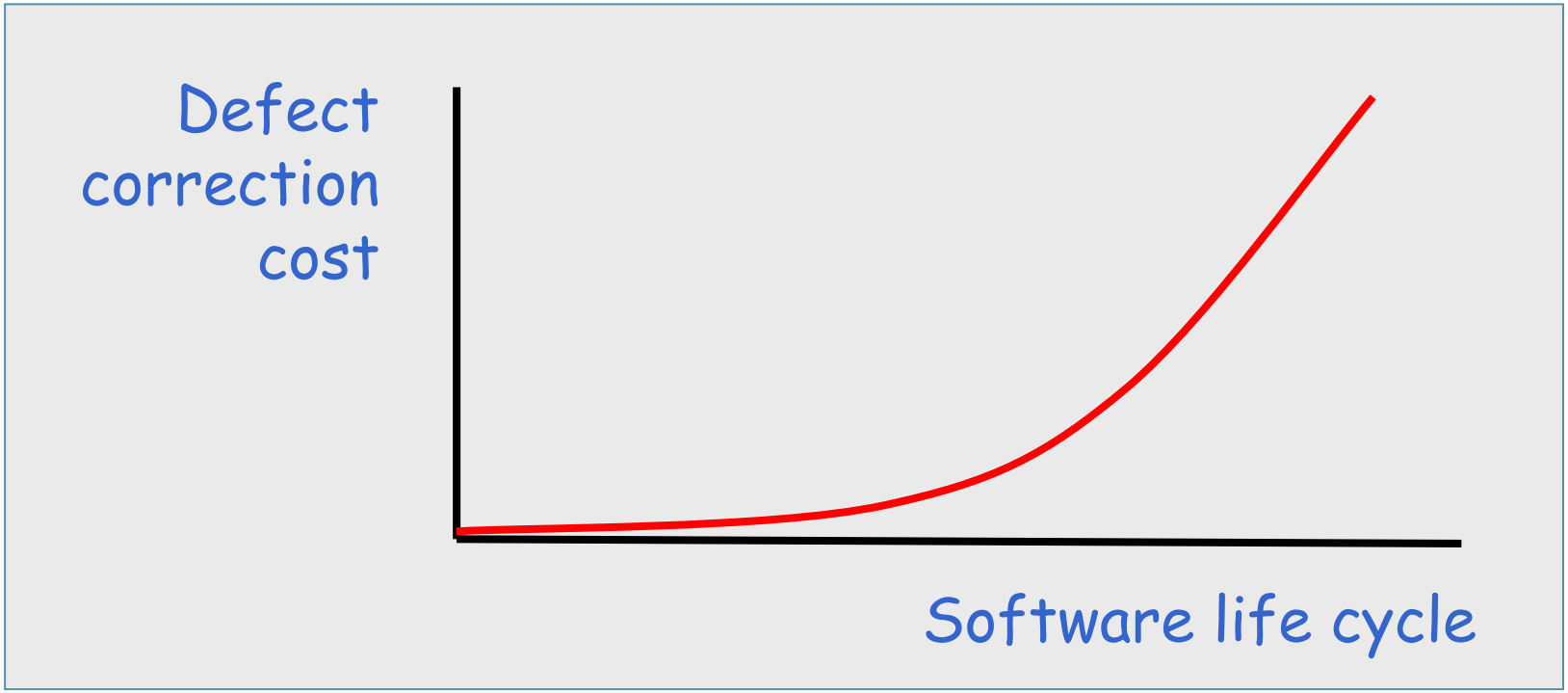
price
Concept type: role
Definition: integer that *represents the value of a* product

product *has* price
Concept type: is-property-of fact type
Necessity: each product *has* exactly one price

family *has* code
Concept type: is-property-of fact type
Necessity: each family *has* exactly one code

product *belongs to* family
Concept type: associative fact type
Necessity: each product *belongs to* exactly one family

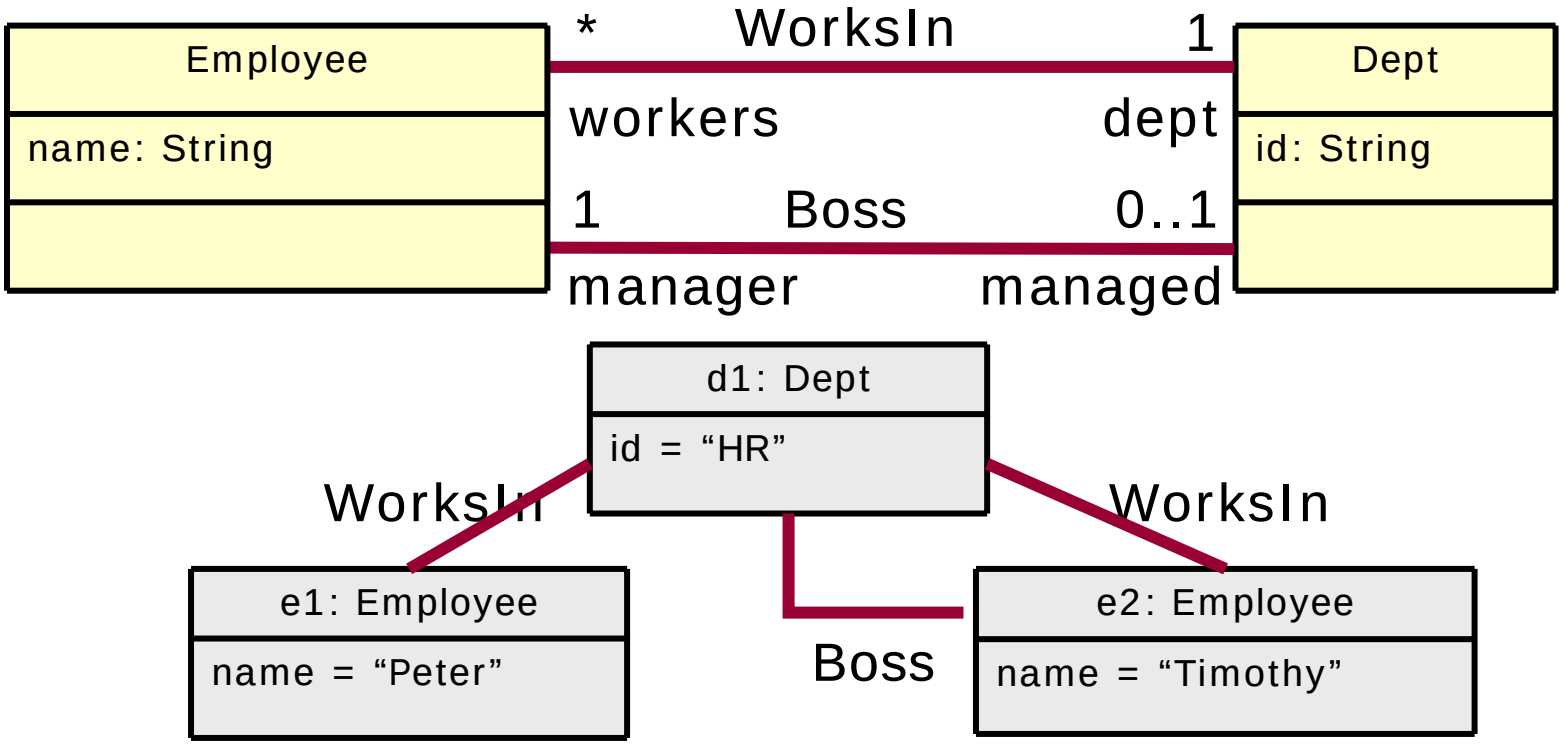
Motivation



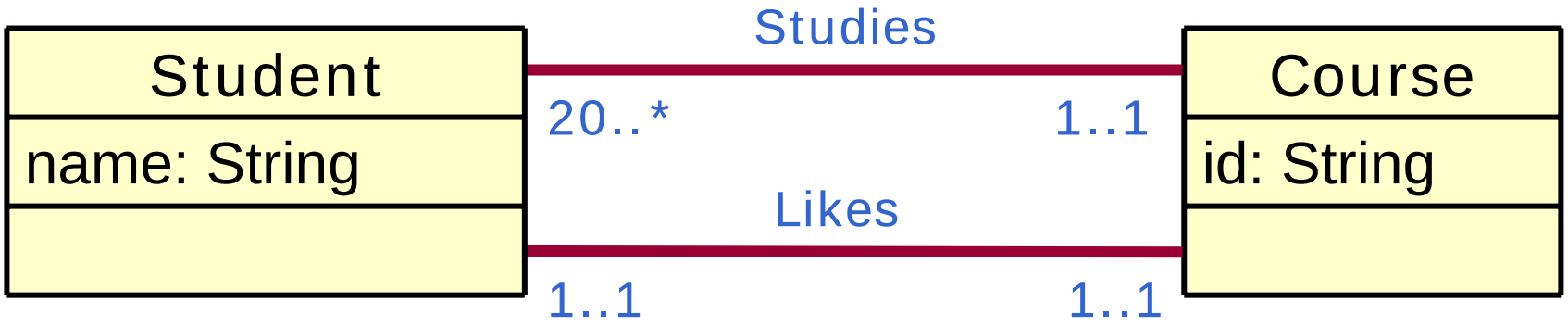
- Find defects ASAP
- Models are the first "formal" spec
- Goal: Find defects in the model

Model Quality : Verification

- Verification checks whether the model satisfies a set of correctness properties, being satisfiability the most basic one. Liveliness, redundancy,... can be expressed in terms of this one
- A model is satisfiable if it is possible to create a valid instantiation of that model. Otherwise it is useless, users won't be able to work with the model
- A instantiation is valid if it satisfies all model constraints



Examples of inconsistency (1)

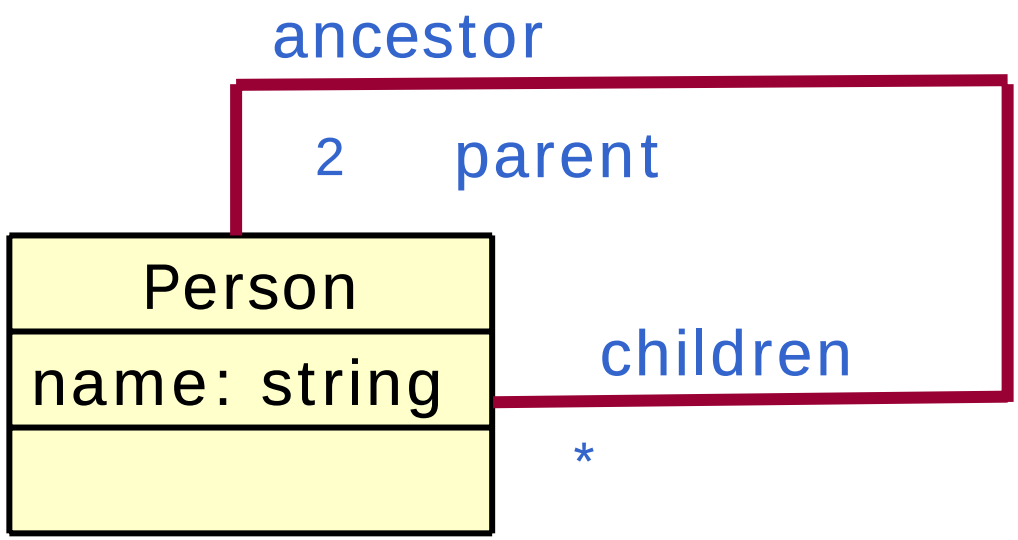


Likes: $|Course| = |Student|$

Studies: $|Course| \geq 20 * |Student|$

Only empty or infinite instances!

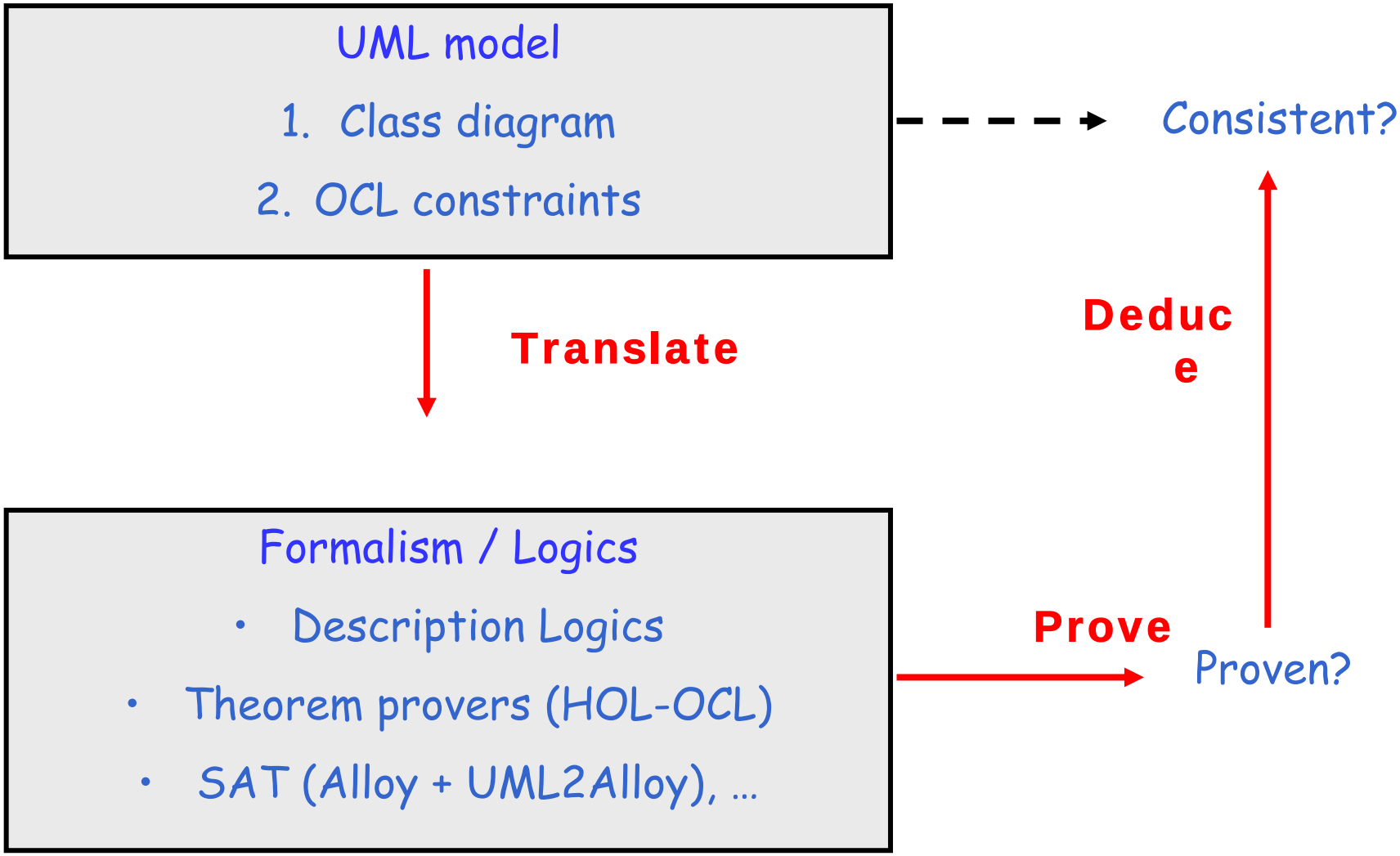
Examples of inconsistency (2)



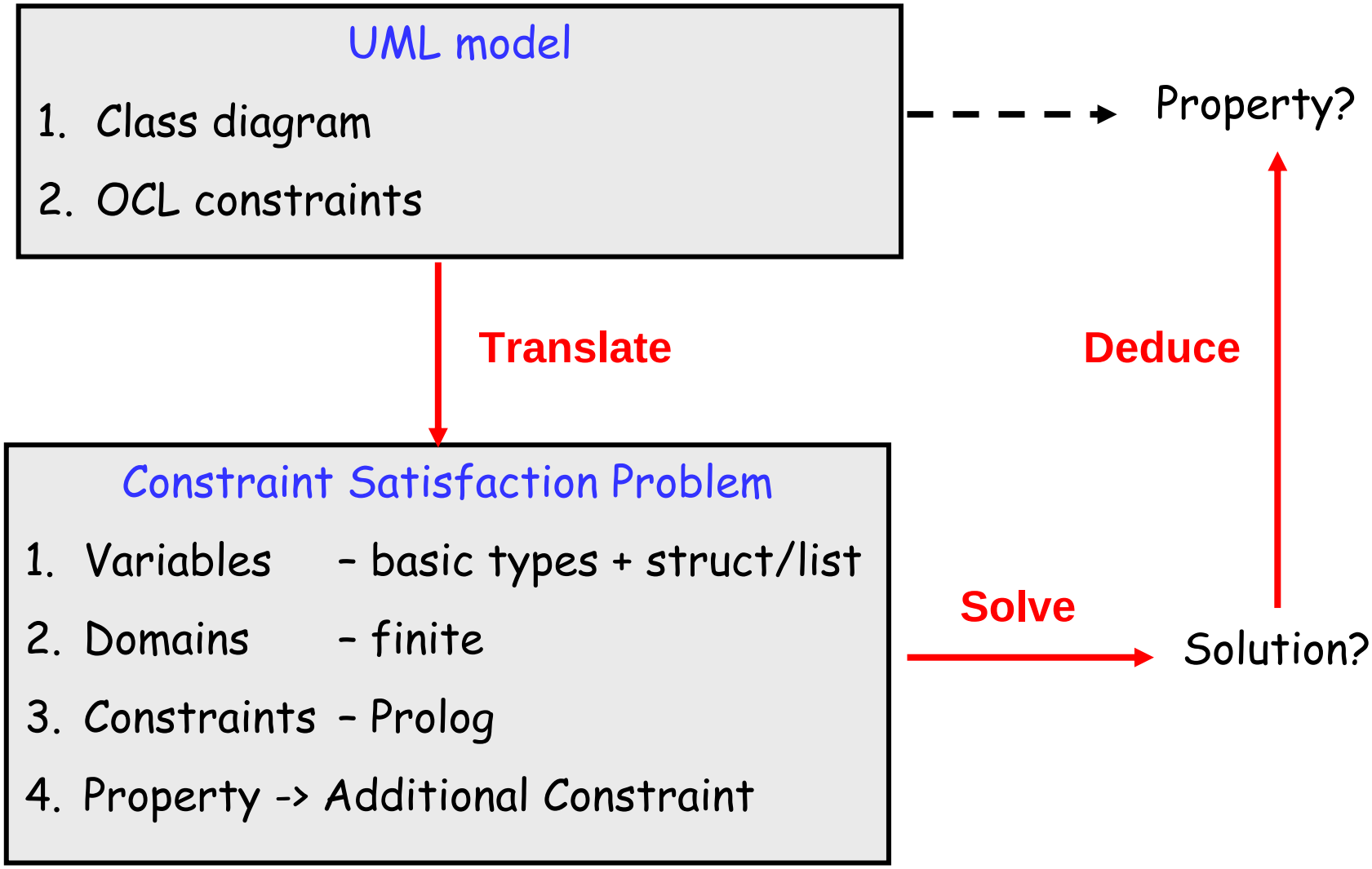

```
// Nobody can be his own ancestor
context Person inv: self.ancestor->excludes(self)
```

Only empty or infinite instances!

Previous work



Model Quality: Verification



Resolution of the CSP

Define cardinality variables

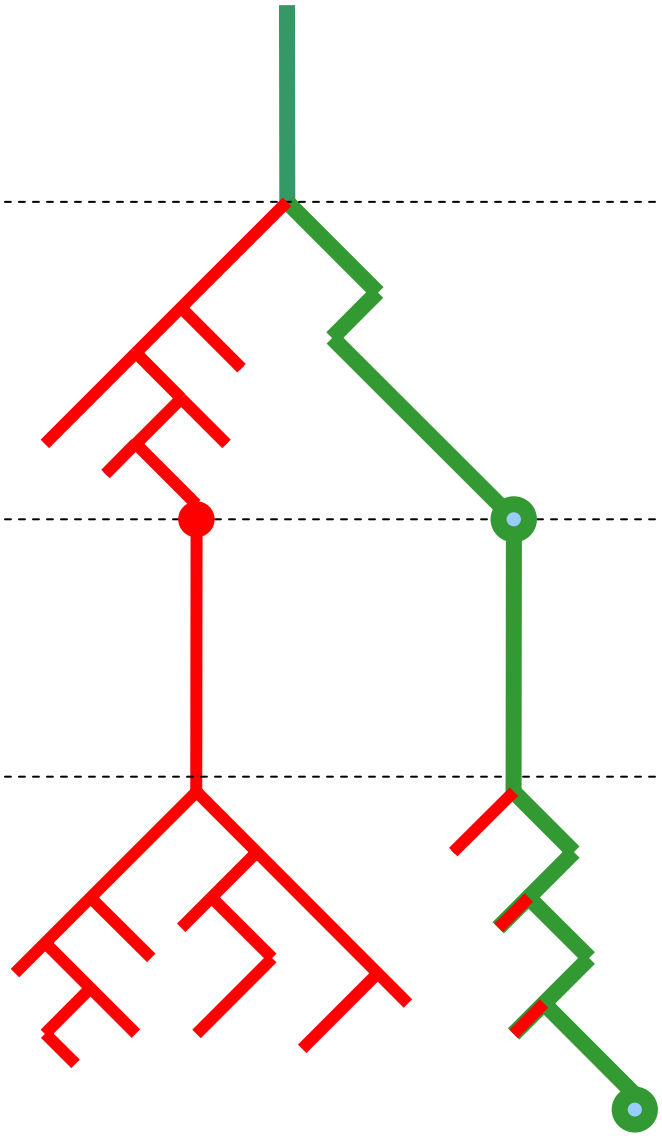
Constraints on cardinalities

Assign cardinalities

Define attribute variables

Constraints on attributes

Assign attributes



Conclusions: features

- Constructive approach
  - Answer is example/counterexample
  - Result presented graphically
- Transparency of the underlying formalism
  - User intervention not required
  - Translation and proof are automatic
- Full OCL support
  - Includes arithmetic, iterator expressions, …

Conclusions: drawbacks

- Bounded verification
  - Maximum number of objects & links
  - Domain of each attribute
- Incomplete
  - No information outside bounded search space
- Prototype implementation
- Integration with CASE tools
- Limitations of OCL parser

Correctness of other modeling elements

- The same approach can be applied to check correctness of DSLs (i.e. is the DSL satisfiable?)
- We have adapted the approach to the verification of operations and (graph) model transformations:
  - Applicability
  - Weak/Strong Executability
  - Determinism
  - ….
- Planned: adapting them to ATL transformations

Open Challenges

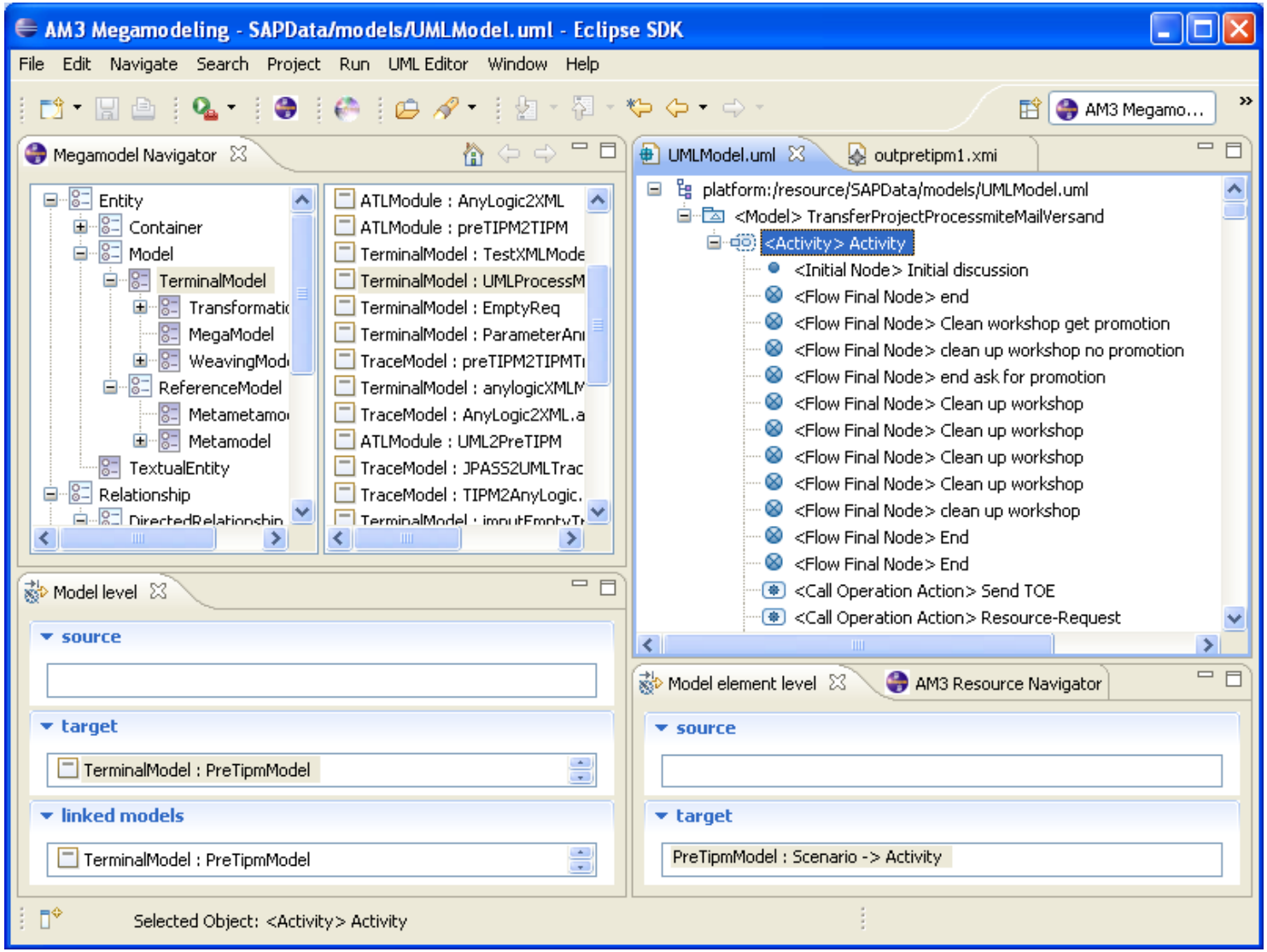
- Performance
- Incremental verification (if we know the model is correct and change a small subset of the model I don't want to reverify everything again(
- Feedback. If the model is not correct, can you explain me why?
- Choosing the best formalism for a <model, property> pair
- Automatic quality assistant that provides hints to the user while modeling
- Adaptation to ATL transformations

Megamodeling

AM3 Megamodeling Solution

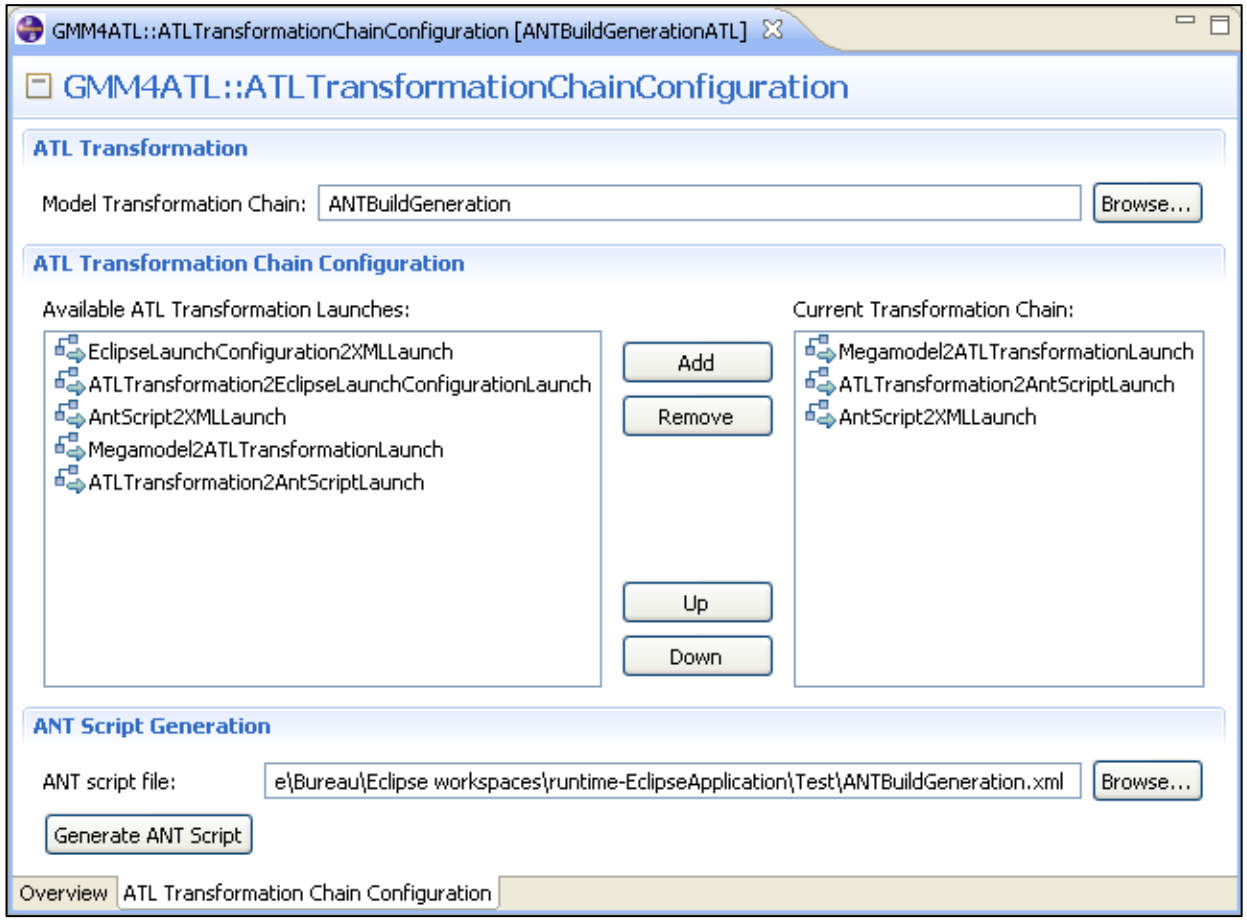
- AM3 provides support for modeling in the large - Global Model Management
- A MDE project usually requires to manage a set of MDE resources (models, metamodels, transformations, …) and the relationships between them → Megamodel
- Similar to a metadata repository on involved modeling artifacts
- AM3 provides facilities to create, handle, manage and use the megamodel

AM3 Environment



AM3 Megamodeling Solution

- Megamodel Action (e.g. Transformation chain - ATL ANT script generation)



MaaS: Modeling as a Service

MaaS: Modeling as a Service

- Service-orientation becoming the standard way of designing and deploying software applications over the internet (Software as a Service or SaaS)

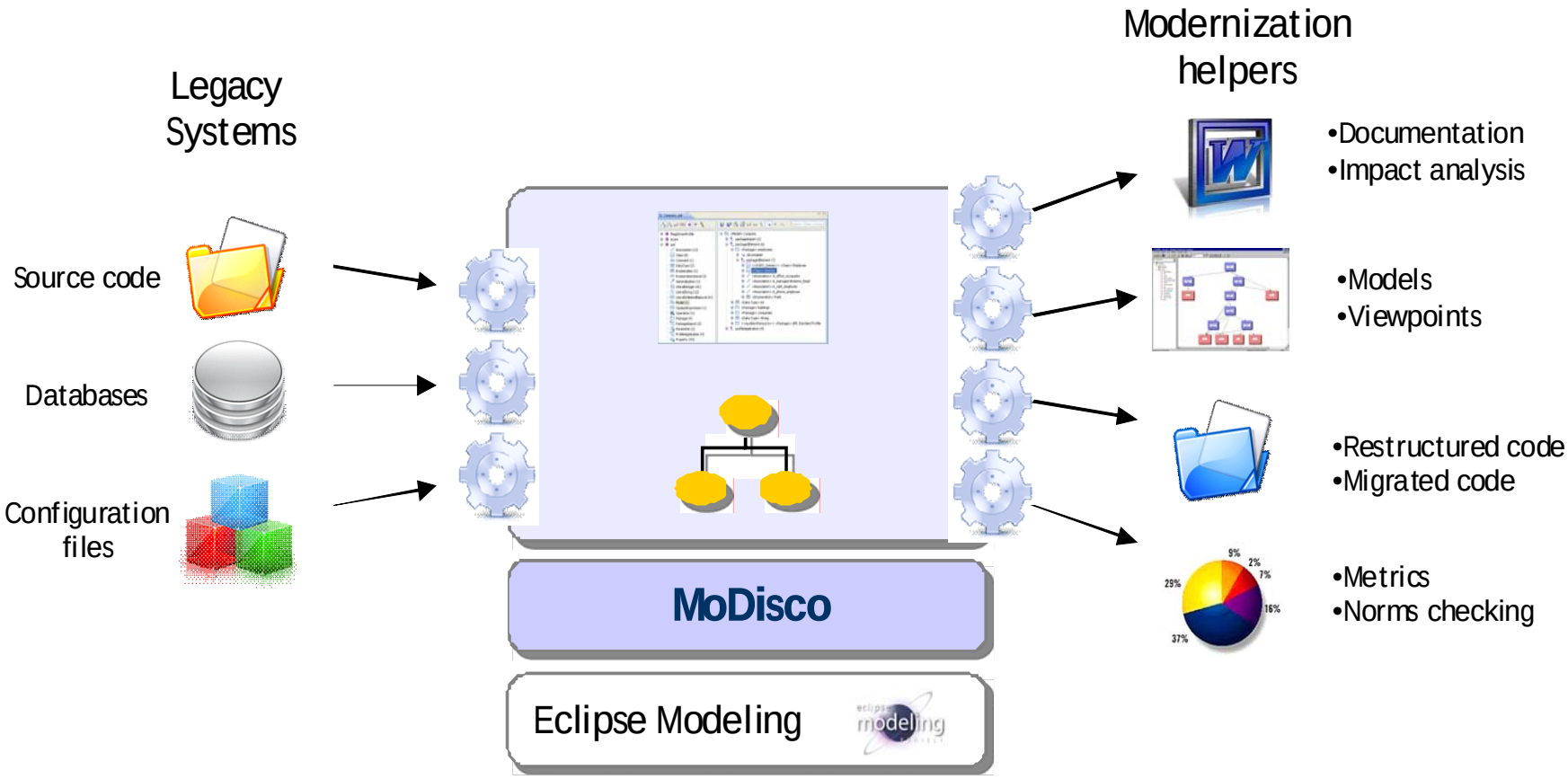
- MDE techniques themselves could be moved to the cloud:
  - Deployment and on-demand execution of modeling and model-driven services over the Internet

- Cloud as the primary infrastructure for MDE tools**?**

- Benefits:
  - Scalability
  - modeling mash-ups as a combination of model-driven engineering services from different vendors,
  - Easier deployment and evolution of software applications
  - Collaborative modeling
  - …

Model-driven Reverse Engineering / ADM

MoDISCO



- ***Instead of adhoc Rev. Eng. Solutions, we use an intermediate model-based representation of the legacy system***

MoDISCO

- Modernizing an existing software system implies:
  - Describing the information extracted out of the artifacts of this system
  - Understanding the extracted information
  - Transforming this information to new artifacts facilitating the modernization (metrics, document, transformed code, ...)
- MoDisco aims at supporting these three phases by providing :
  - Metamodels to describe existing systems
  - Discoverers to automaticaly create models of these systems
  - Generic tools to understand and transform complex models created out of existing systems

MoDISCO

org.eclipse.gmt.am3.platform.core.javaxmi

Types (0)

- ArrayType (1)
- Assignment (19)
- Block (67)
- BlockComment (2)
- BooleanLiteral (1)
- CastExpression (7)
- CatchClause (2)
- ClassDeclaration (46)
- ClassInstanceCreation (9)
- CompilationUnit (11)
- ConstructorDeclaration (15)
- EnhancedForStatement (4)
- ExpressionStatement (54)
- FieldDeclaration (48)
- IfStatement (12)
- ImportDeclaration (41)
- InfixExpression (25)
- Initializer (1)
- InstanceofExpression (1)
- InterfaceDeclaration (45)
- Javadoc (28)
- LineComment (75)
- MethodDeclaration (87)

Instances

type filter text

- [ClassDeclaration] MegaModelConstants
- [ClassDeclaration] AM3Platform
  - */eContainer*
  - name = AM3Platform
  - proxy = false
  - /qualifiedName = org.eclipse.gmt.am3.platform.core.AM3Platform
  - comments (29)
  - originalCompilationUnit (1)
  - usagesInImports (1)
  - abstractTypeDeclaration (0)
  - annotations (0)
  - anonymousClassDeclarationOwner (0)
  - modifier (1)
  - usagesInTypeAccess (3)
  - bodyDeclarations (30)
  - commentsBeforeBody (0)
  - commentsAfterBody (0)
  - package (1)
  - superInterfaces (0)
  - typeParameters (0)
  - superClass (0)
- [ClassDeclaration] Activator

MoDISCO

- Remember since the Java code is represented as an instance of the Java Metamodel, we could
  - Query the model (e.g. using OCL)
  - Transform it to other representations (e.g. generate a C# version)
  - Analyze it
  - ...

  using existing Eclipse/EMF tools

Tool interoperability

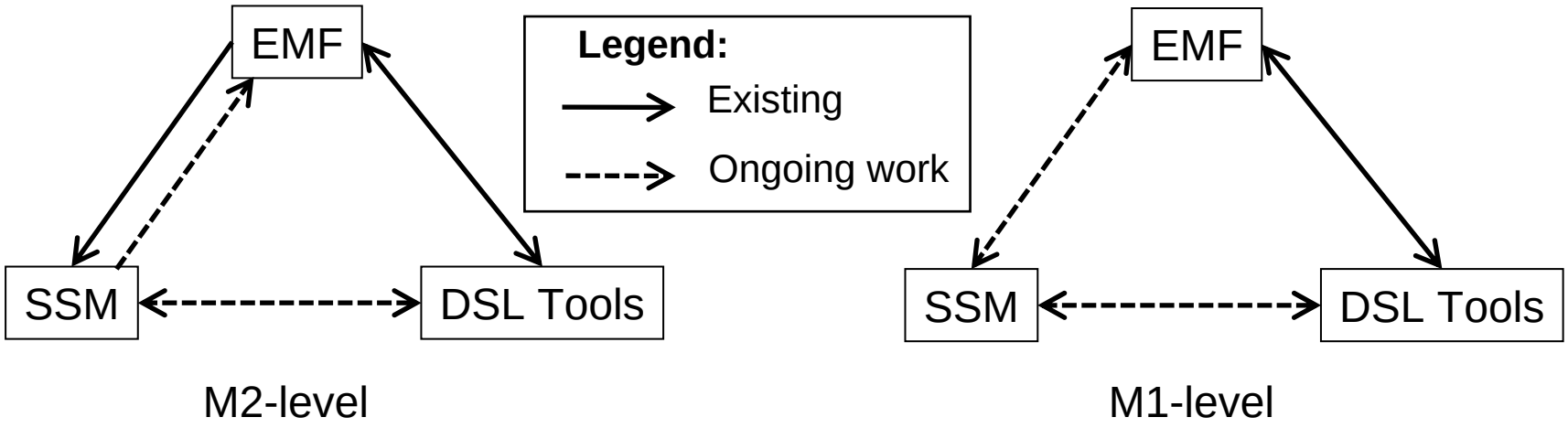
Tool interoperability

- MDEcan be useful to bridge tools/platforms/systems

- Three steps
  - Extraction of the metamodel of each tool (if no explicit metamodel, derive it from the tool API or storage format(
  - Define mappings between metamodels
  - Generate transformations to exchange data

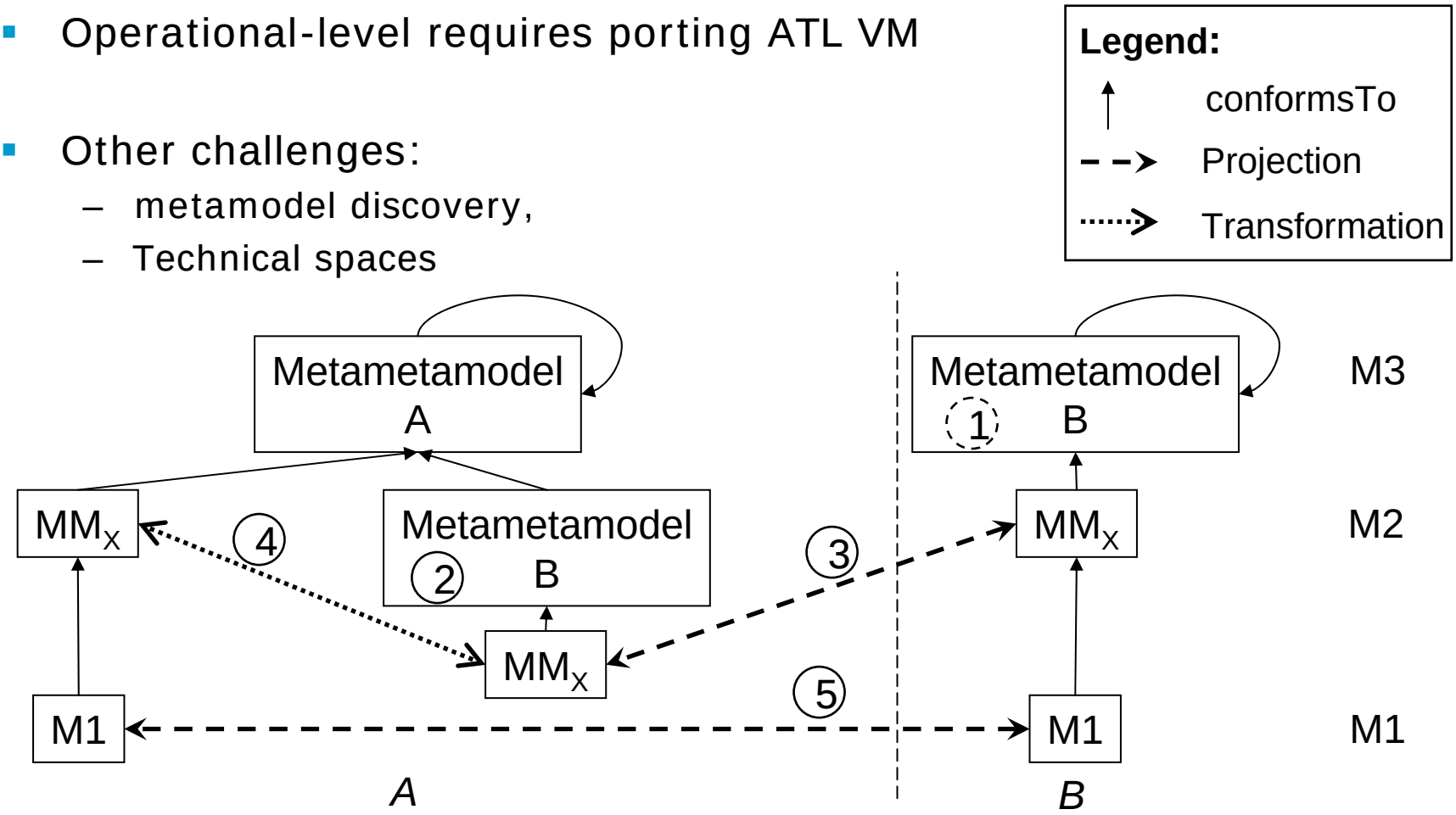
Example: Bridging Eclipse and Microsoft Modeling

- Eclipse -> EMF. Microsoft SQL Server Modeling (former Oslo) and DSL Tools.
- Practical bridges between the three tools/platforms would be useful in industrial environments
- Most general scenario: tool with *variable* metamodels → need to interchange both metamodels and models



General Architecture

- Interchange at the data-level
- Operational-level requires porting ATL VM
- Other challenges:
  - metamodel discovery,
  - Technical spaces

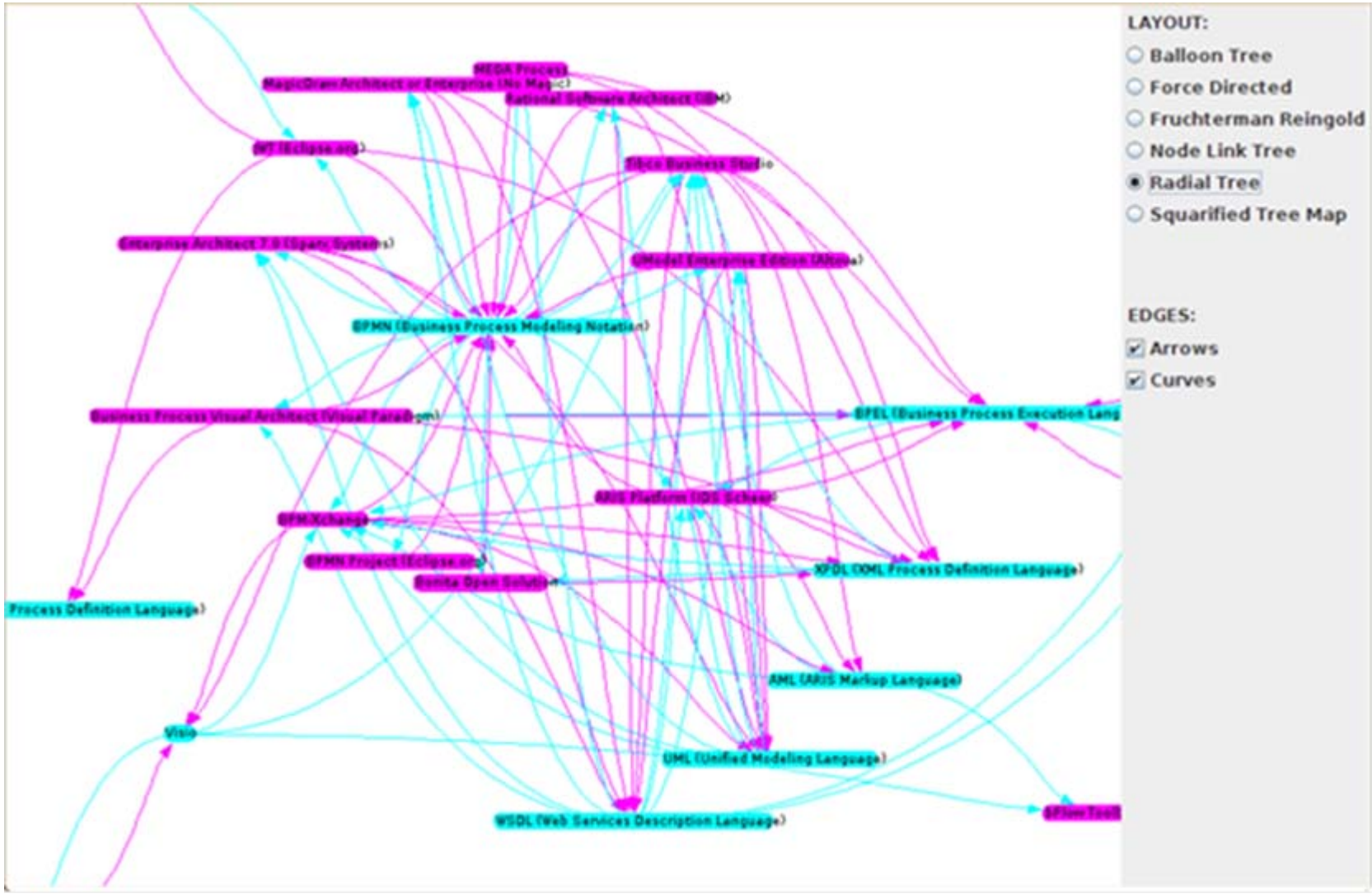


Model-driven cartography

- For complex business scenarios, the first step is to represent the current reality of the company
- We use model-driven techniques to represent and visualize all the tools/platforms/components used by an organization, and more importantly all the dependences between them

- This allows a company to validate its tool ecosystem and to reason on “what if” situations (e.g. what if I replaced tool A with the new tool B? What other tools would be affected? Can I still exchange information between the tools? Do I need to create a new bridge?...)

Model-driven cartography



Ontologies and MDE

Ontologies and MDE

- There is no fundamental difference between a model and a ontology
- Can MDE help ontology engineering? YES
- Can ontologies help MDE? YES
  - Using general knowledge of a domain to suggest corrections or additions to a user model

Modeling temporal and geographical information

Temporal and geographical information

- It's definitely interesting but you don't see many papers on this in general modeling/MDE conferences
- Most approaches based on some kind of UML profile for temporal and/or geographical information
- There are also temporal versions of OCL
  - Context Employee inv SalaryCannotDecrease:
    self.salary.at(t)>=self.salary.at(t+1)
- Can MDE help ontology engineering? YES
- Would DSLs help?

Example: MDE for Datawarehouses

- Conceptual modeling has proved to be very useful in the development of data warehouse systems.

- Main benefits -> benefits of conceptual modeling:
  - Implementation-independent view of the system
  - Possibility of (semi)automatic code-generation
  - Better maintainability and evolution
  - ...

Conceptual Modeling of DWH (1/2)

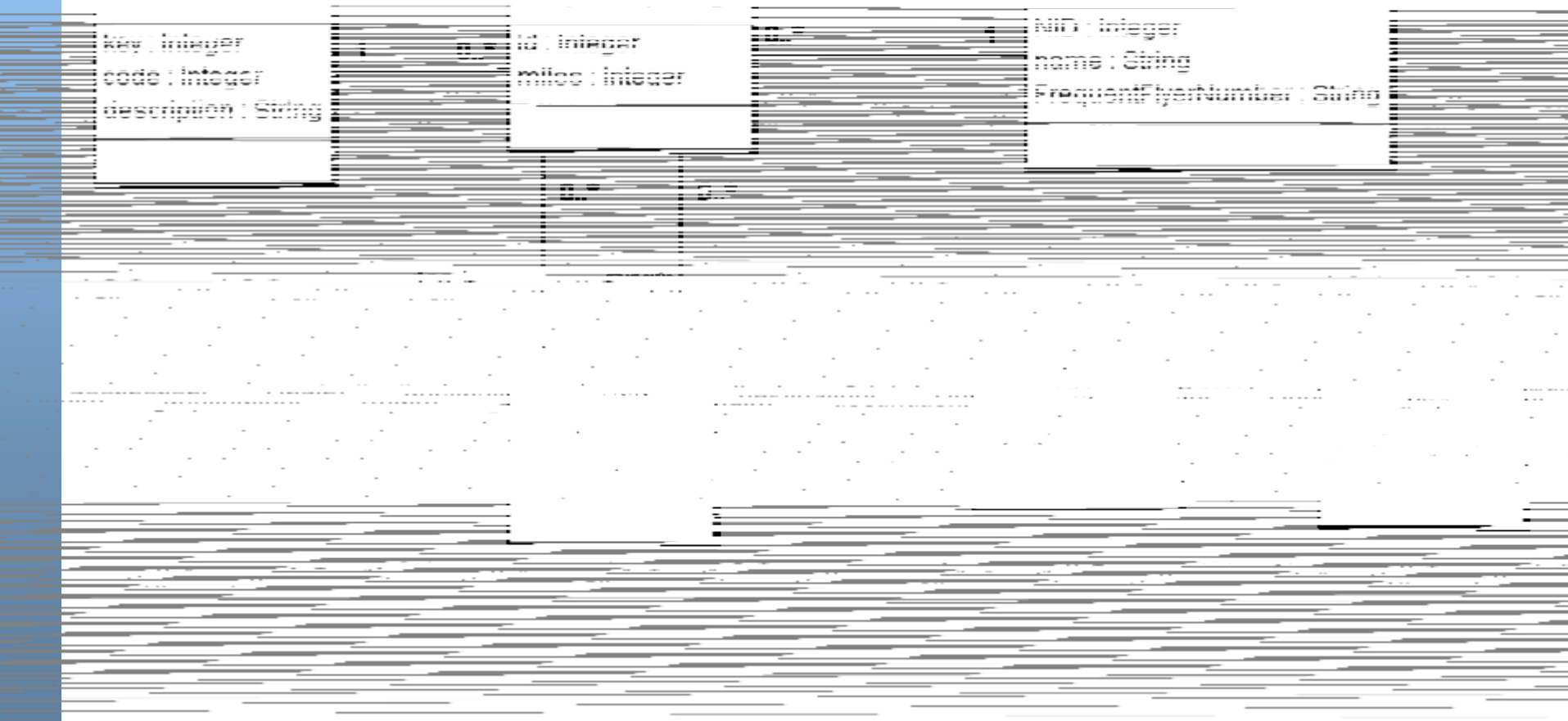
- Modeling multidimensional concept at conceptual level
  - Data structured in a multidimensional space
  - Dimensions specify different ways the data can be viewed, aggregated, and sorted
    - E.g., according to time, store, customer, product, etc.
  - Events of interest for an analyst are represented as facts which are associated with cells or points in the multidimensional space and which are described in terms of a set of measures

- abstracted logical details:
  - technology: relational, multidimensional, ...
  - logical variations: star, snowflake schema, ...

- automatically obtain a logical representation
  - model-driven approach

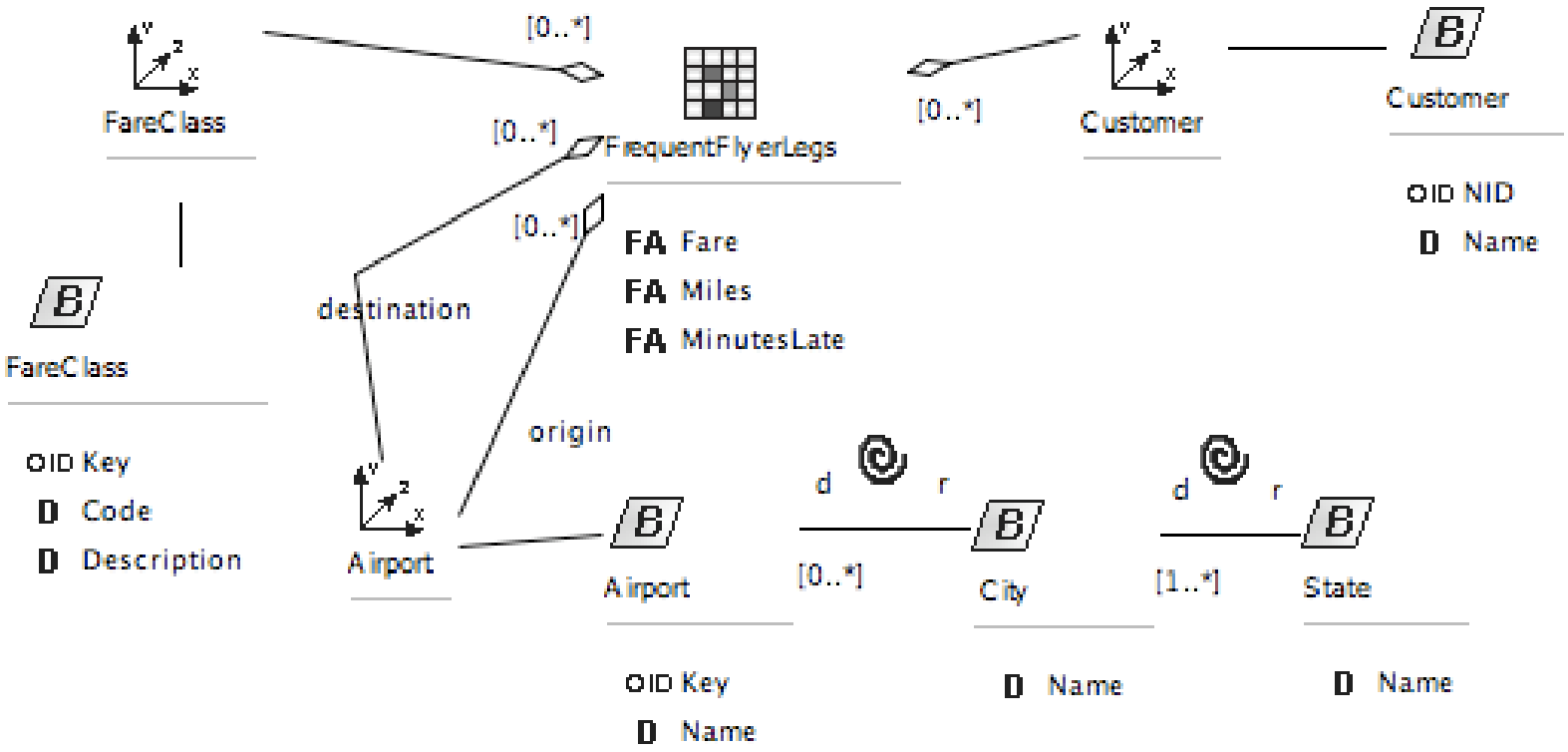
Conceptual Modeling of DWH

An airline's marketing department wants to analyze the flight activity of each member of its frequent flyer program

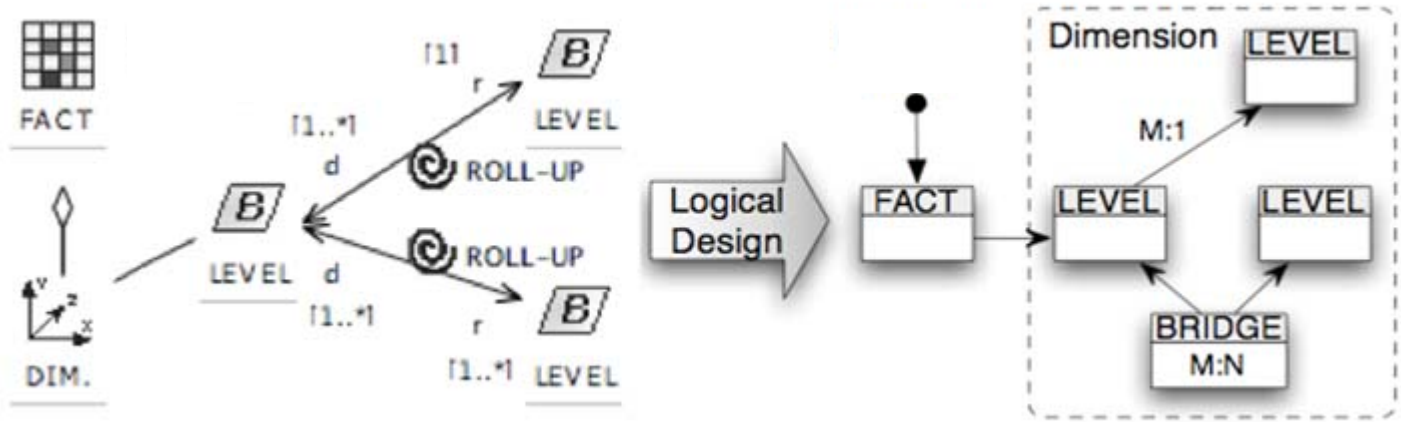


Conceptual Modeling of DWH (1/2)

… once annotated with the Profile becomes …



Conceptual Modeling of DWH



… BUT (there's always a 'but')

- Right now, only the structural aspects of the DWH are modeled but decision makers require a set of multidimensional queries
- These multidimensional queries are not specified as part of the Conceptual Schema (CS) of the DWH
- They are only added once the DWH is implemented
- As a result:
  - Breaks the MDE approach
  - The completeness of the DWH cannot be validated until it is implemented (i.e. DWH contains enough information?)
  - Definition of queries requires expertise in the target platform
  - No reusability
  - …

- ***This limitation affect not only multidimensional models but, in general, all kinds of CSs (informative function ignored)***

Limitations of CM languages

- The main restriction for defining queries at the CS level -> poor support in current CM languages
- In particular, CM languages exhibit a lack of rich constructs for the specification of aggregation functions (key in DWH systems)
- Usually only basic ones (sum, avg,…) are covered but DWH systems require richer analysis functions (e.g. rank, percentile, min, max,…)
- For instance, OCL (most popular query language for CSs) only includes the *sum*, *size* and *count* functions

- ***If a designer wants to know the ranking of frequent flyers he has to build the ranking function himself***
- ***Very time consuming and error-prone***

- *Don't you prefer to have the "*" operator even if "+" is enough?*

Extending OCL

- Extension classified in three different groups of functions:
  - Distributive functions: can be defined by structural recursion
    - Max, min, sum, count, count distinct,…
  - Algebraic functions: finite algebraic expressions over distributive functions
    - Avg, variance, stddev, covariance, …
  - Holistic functions: the rest
    - Mode, descending rank, ascending rank, percentile, median

- ***These operations can be combined to provide more advanced ones (e.g. top(x) that is implemented using rank)***