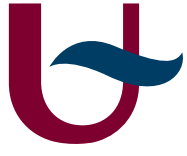
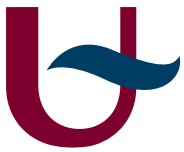
Moodling

An Integrated Approach Towards Example-based Domain-Specific Language Design with Focus on Agility

Lucas Heer

Universiteit Antwerpen

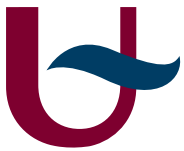




Domain-Specific Modeling Languages

- Modeling languages tailored to a specific domain [1]
- Increasingly used in software and systems development
- Describe structure and behavior of a system
- Abstract syntax $\rightarrow$ Structure (metamodel)
- Concrete syntax $\rightarrow$ Representation
- Semantics $\rightarrow$ Meaning and behavior

[1] Kelly, S. and Tolvanen, J. P. Domain-specific modeling: enabling full code generation. *John Wiley & Sons, 2008*

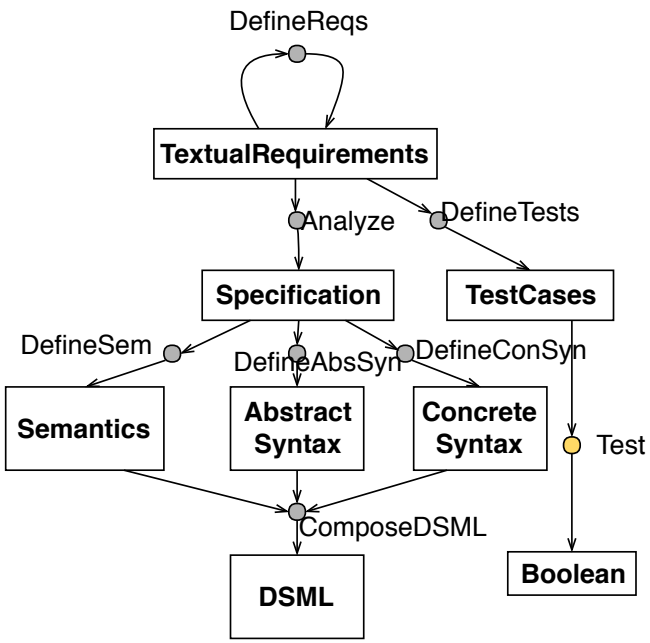


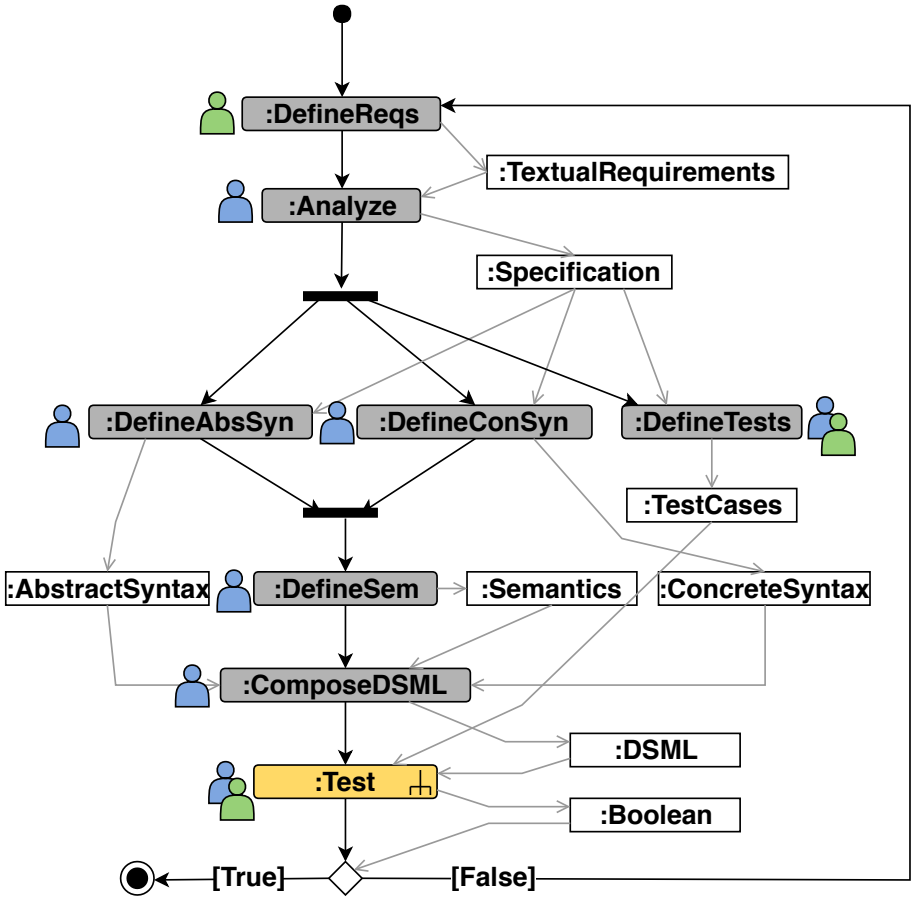
FTG+PM

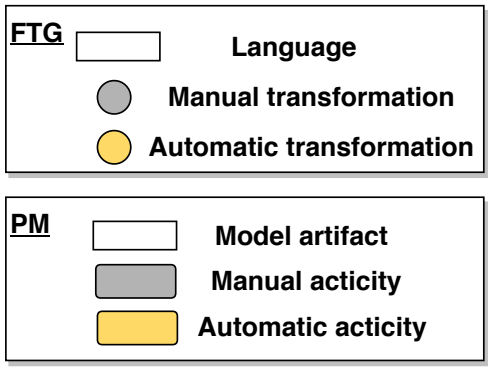
- Formalism used to guide MDE lifecycle [2]
- Describes processes, artifacts, involved languages and transformations
- Used here to describe workflows and processes

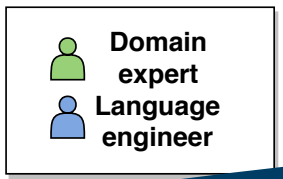
[2] Lúcio, Levi, et al. The formalism transformation graph as a guide to model driven engineering. *School of Computer Science, McGill University, Tech. Rep. SOCS-TR2012*,1, 2012.

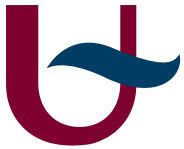
Traditional DSML Design







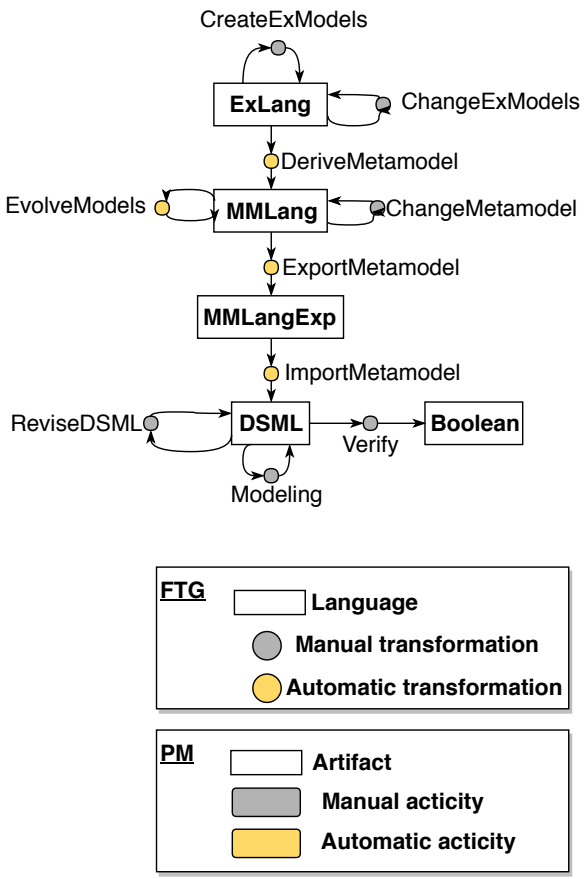


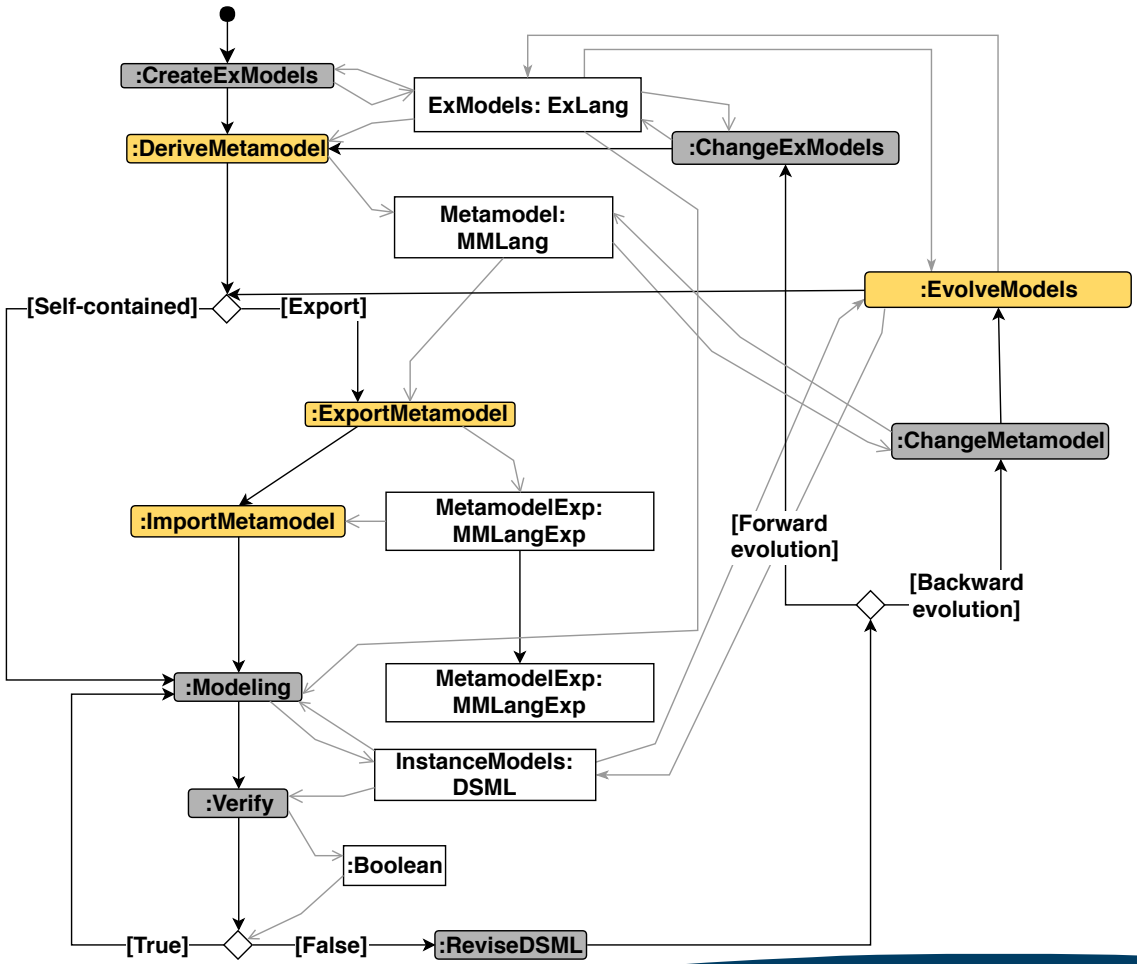


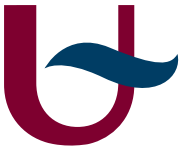
Drawbacks

1. Language engineering knowledge required
2. Slow reaction to new / changing language requirements
3. Late feedback on language adequacy

Example-driven DSML Design







Some Existing Approaches

metaBup[3]

- Sketch example models in general-purpose drawing tool
- Import into EMF to generate metamodel and modeling environment

MLCBD[4]

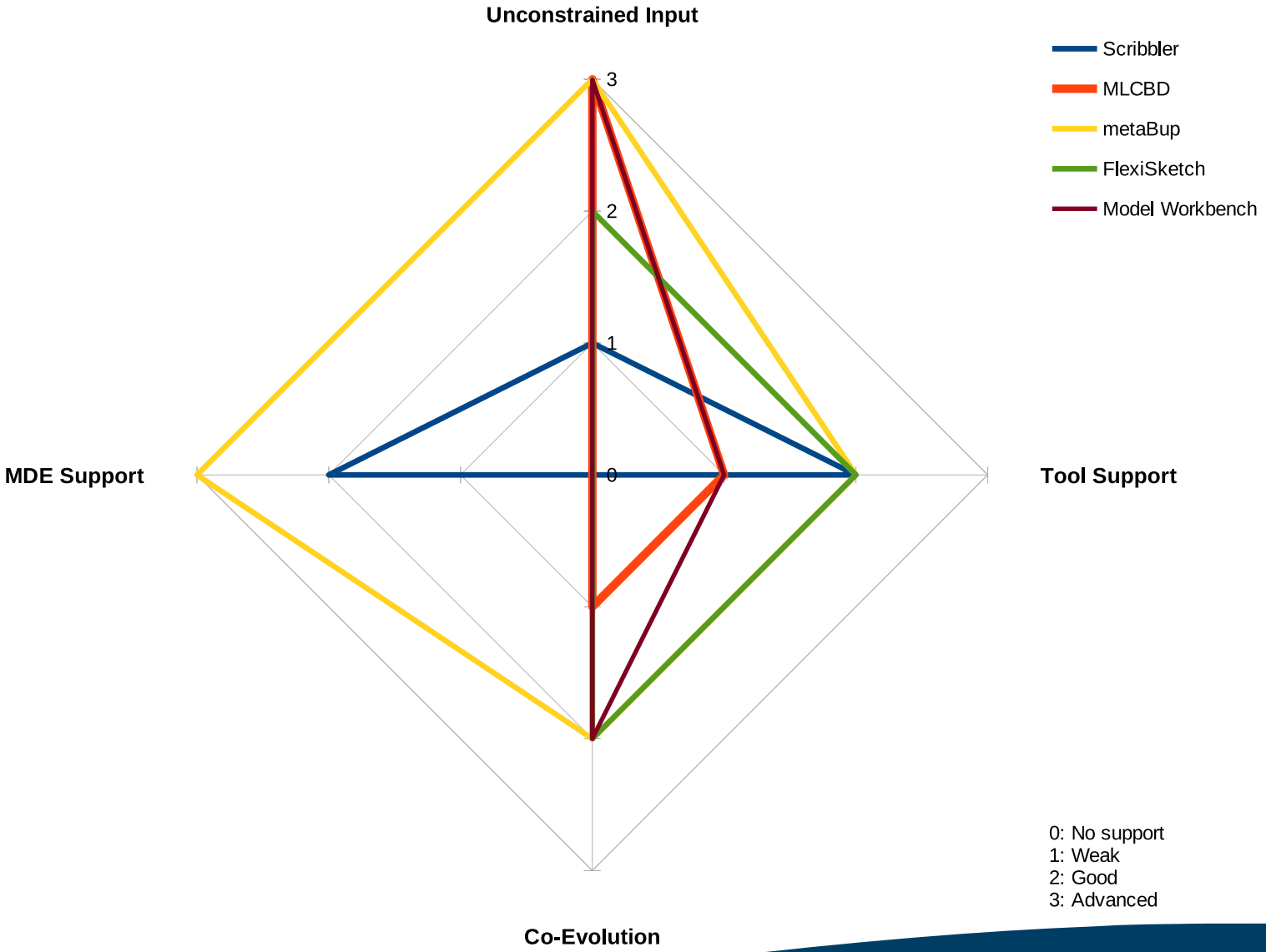
- Self-contained in MS Visio
- Implicit metamodel generation

---

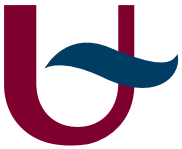
[3] López-Fernández, Jesús J. An agile process for the example-driven development of modelling languages and environments. *PhD thesis*, Autonomous University of Madrid, 2017

[4] Cho, H. A demonstration-based approach for domain-specific modeling language creation, *PhD thesis*, University of Alabama, 2013

Evaluation of Existing Approaches



0: No support
1: Weak
2: Good
3: Advanced

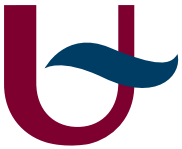


Issues

No existing approach gives compelling answer for:

How to

1. Evolve language without losing models?
2. Use models for MDE activities?
3. Decrease cognitive load?



Solution

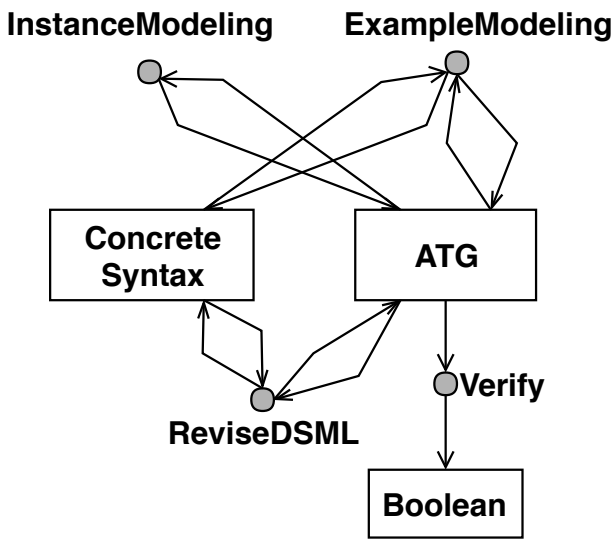
Need an **integrated** approach that increases **agility:**

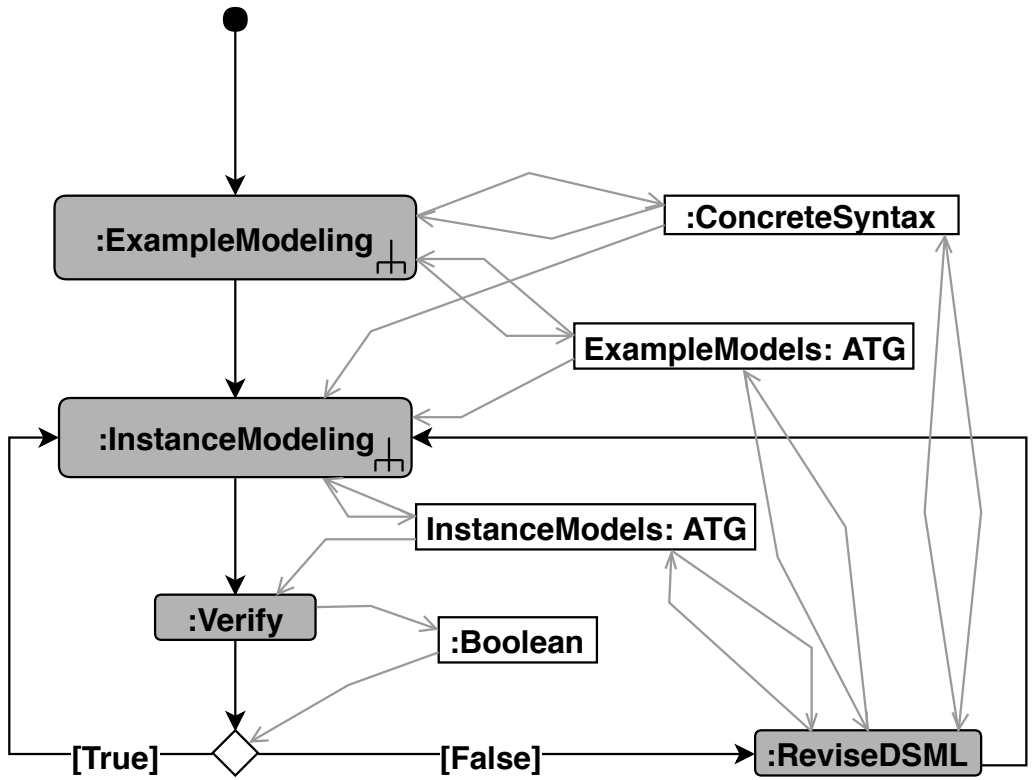
Integrated: Designed around MDE principles and implemented in a metamodeling environment

Agile: Short loop between language design and use without abandoning model artifacts along the way

Moodling Process

FTG+PM for an agile and integrated example-driven DSML design:



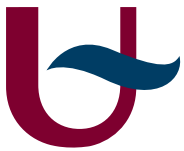


Central Idea

Disadvantages of generating an explicit metamodel from example models:

- Increases number of artifacts (cognitive load, usability)
- Simultaneous co-evolution of example- and instance models required when metamodel is changed
- Knowledge of meta-concepts required

→ **Constrain instance modeling directly by example models**

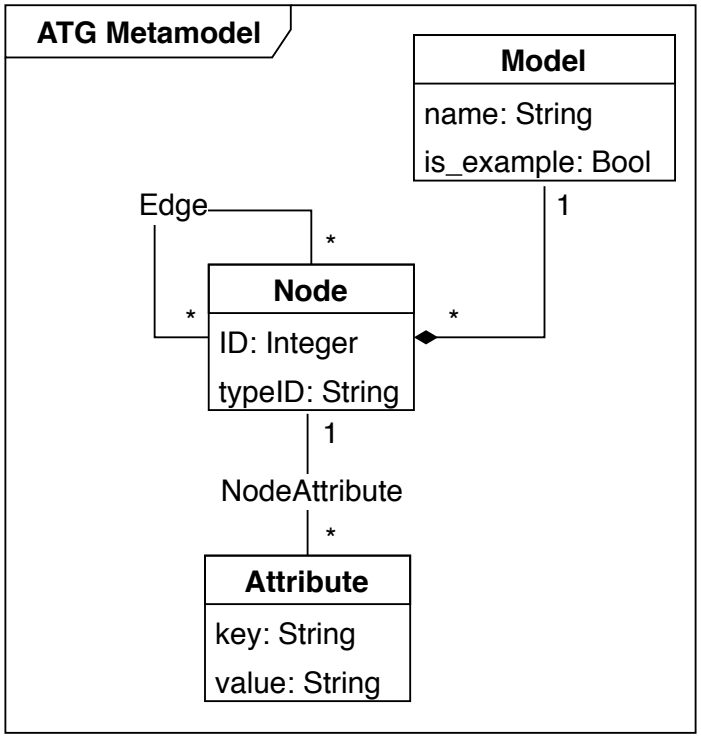


Ingredients

1. A common and generic **metamodel** to which example- and instance models conform to
2. A **conformance relationship** between an instance model and a set of example models
3. A **co-evolution solution** to automatically evolve instance models when example models change

A Common Metamodel

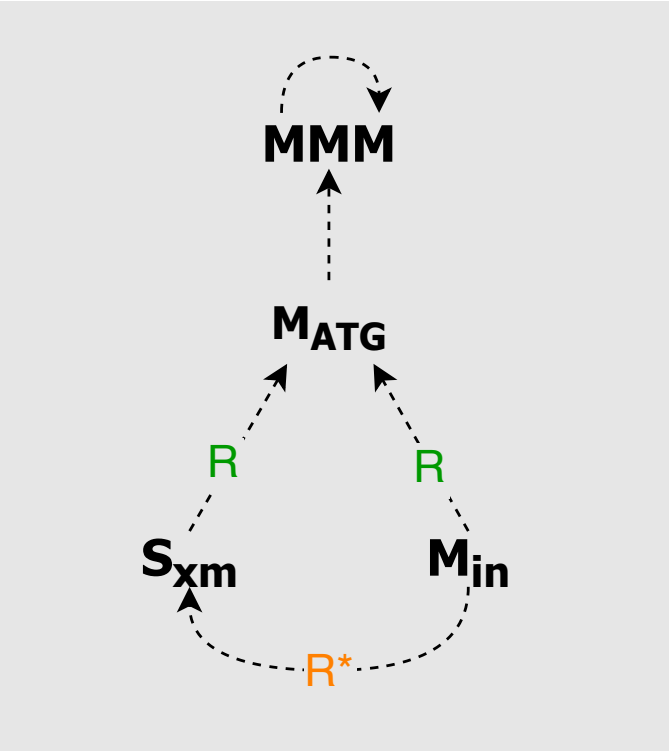
Example- and instance models conform to a **common metamodel**:



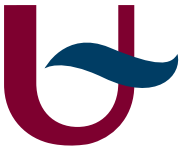
ATG: *Attributed type graph*

Conformance Relationship

Instance models need to conform to the set of example models:



$\rightarrow$ Need to define R^*

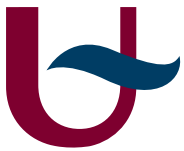


Conformance Relationship

Define **conformance** between an instance model and a set of example models for:

1. Node typing
2. Type cardinality
3. Edge typing
4. Edge cardinality
5. Attribute typing
6. Attribute cardinality

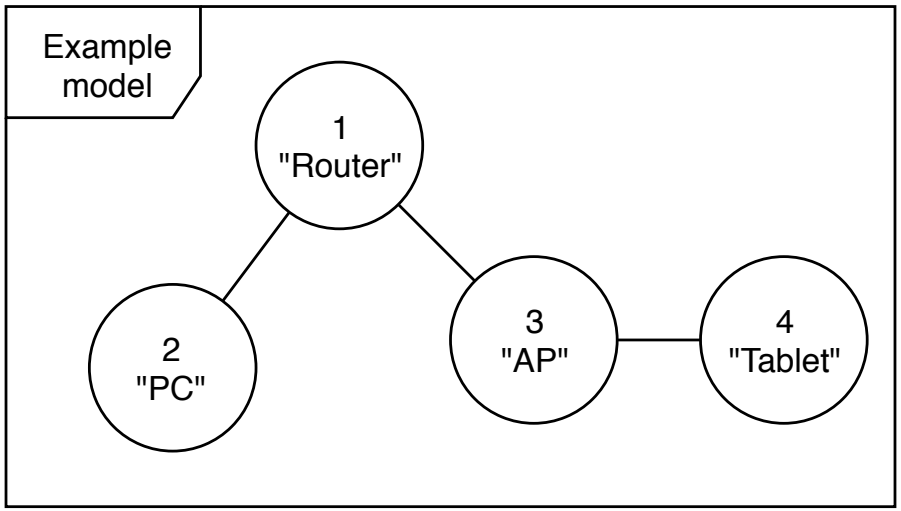
$\rightarrow$ Derived from conformance relationship between model and metamodel

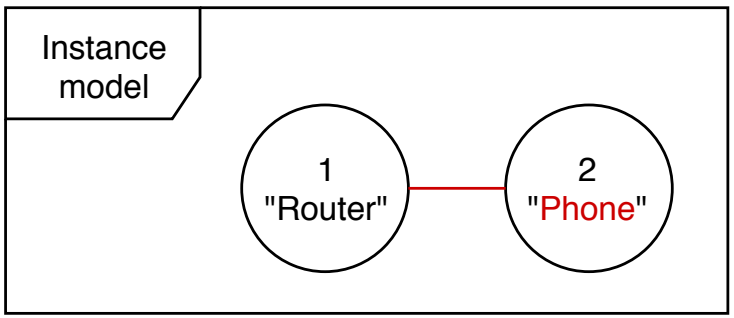


Example: Node Typing

The nodes of an instance model are completely typed if for every node, there is a node with the same type in any example model.

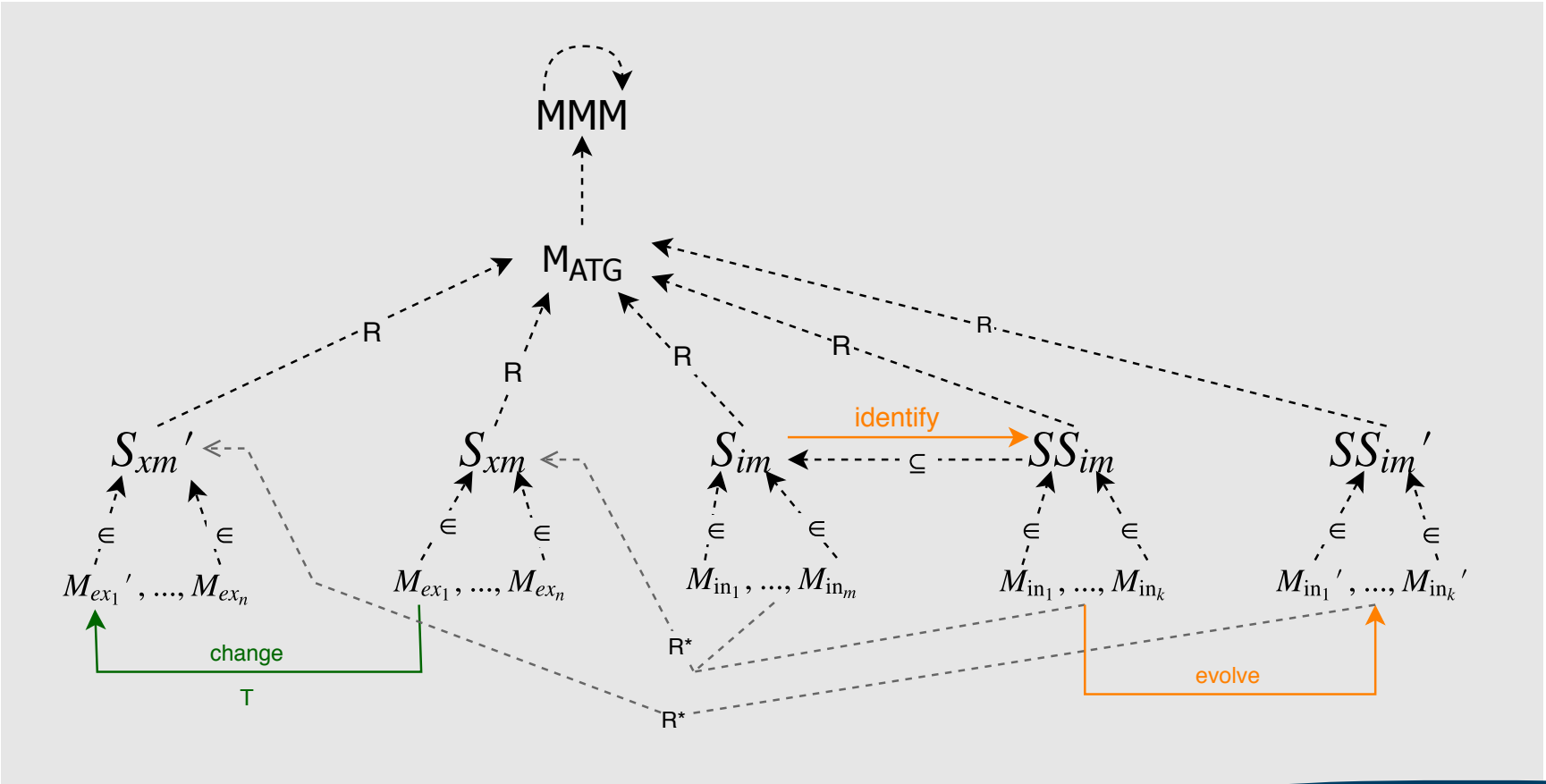
Example: Node Typing

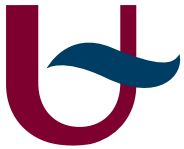




Co-Evolution

Instance models must **co-evolve** when example models change:





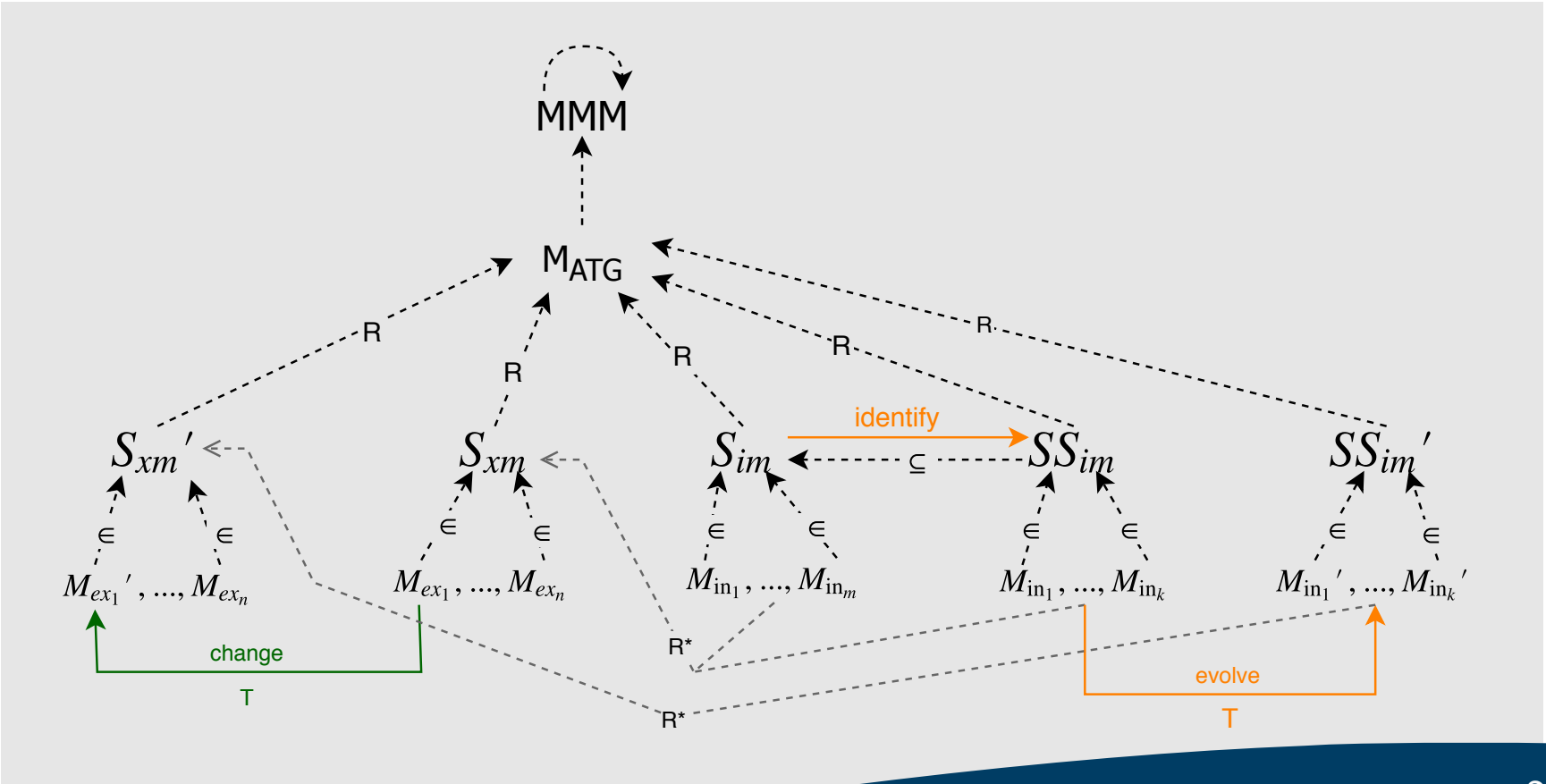
Co-Evolution Classification

Changes to example models have consequences:

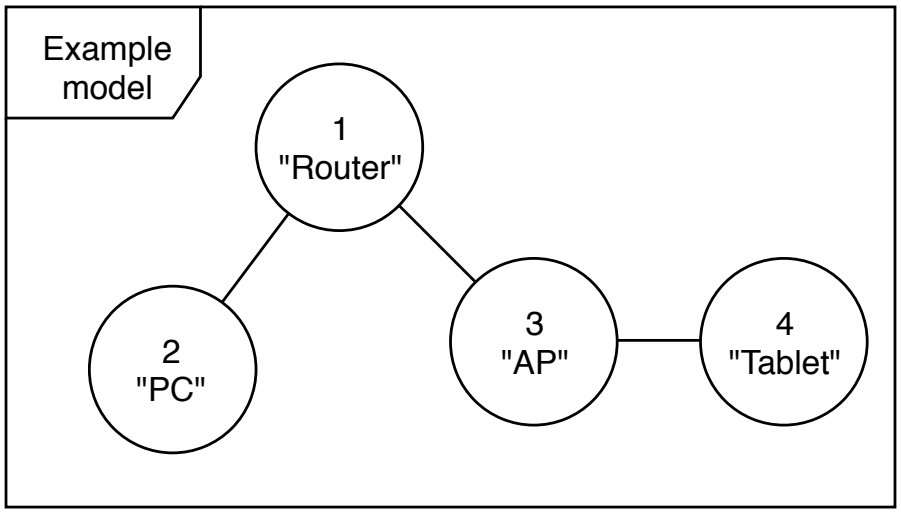
Change operation	**Potential effect**	**Constraint**
Add node	Make type mandatory	Type cardinality
Delete node	Makes type invalid	Node typing
Retype node	Makes type invalid	Node typing
Add edge	Makes edge mandatory	Edge cardinality
Delete edge	Makes edge invalid	Edge typing
Add attribute	Makes attribute mandatory	Attribute card.
Delete attribute	Makes attribute invalid	Attribute typing
Change attribute	Makes attribute invalid	Attribute typing
Delete model	Makes type invalid Makes edge invalid Makes attribute invalid	Node typing Edge Attribute

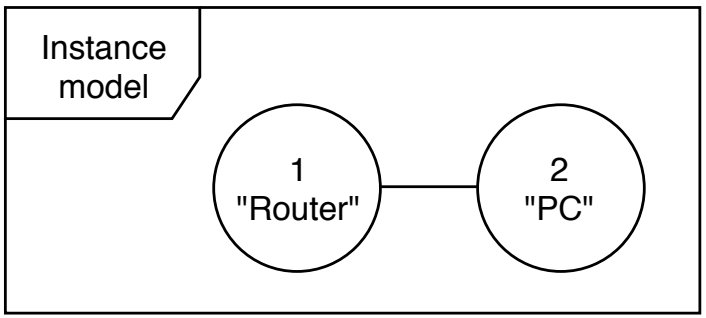
Co-Evolving Instance Models

Detect if change broke conformance and repair by applying the same change transformation to instance models:

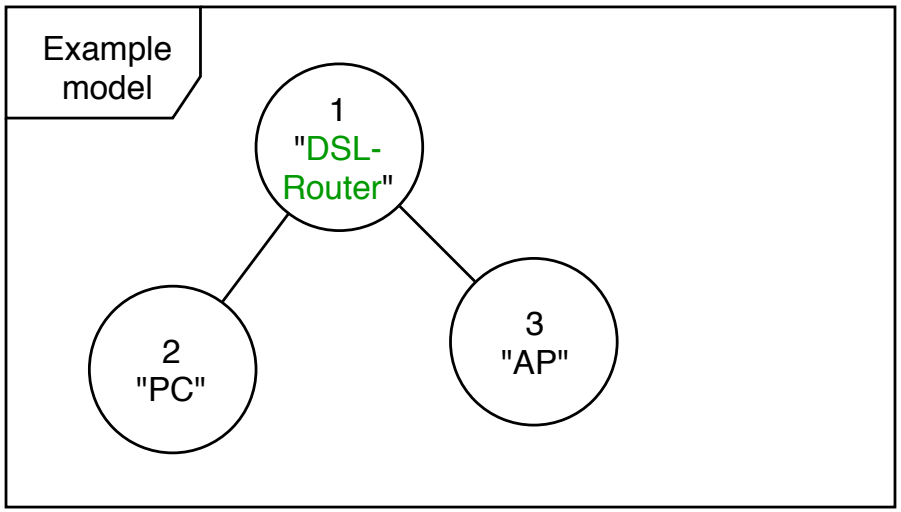


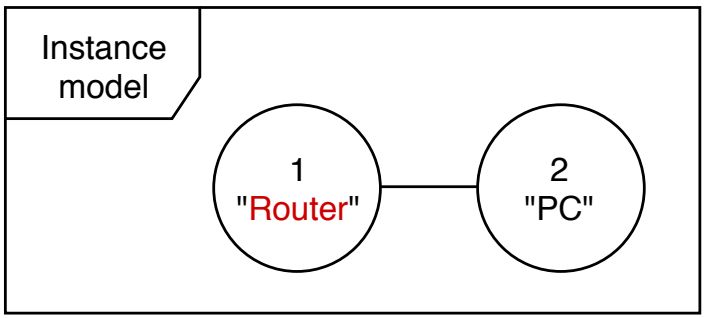
Evolution Example



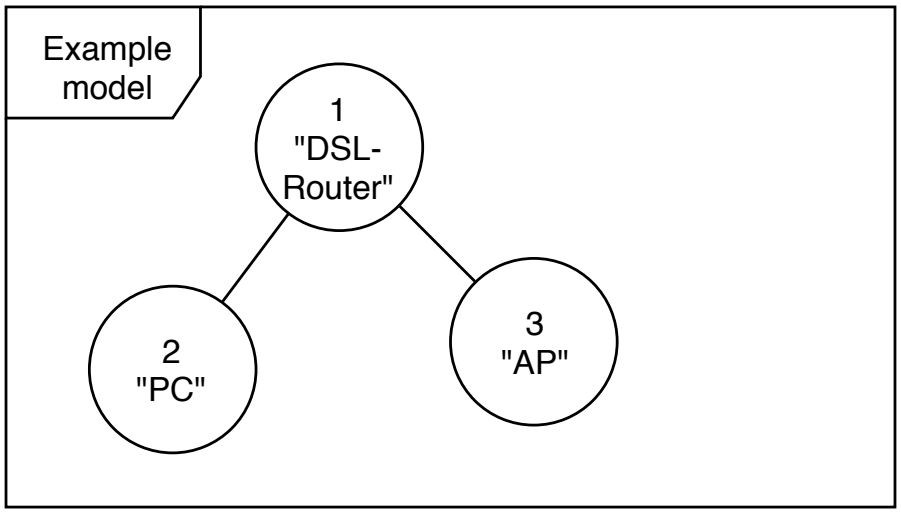


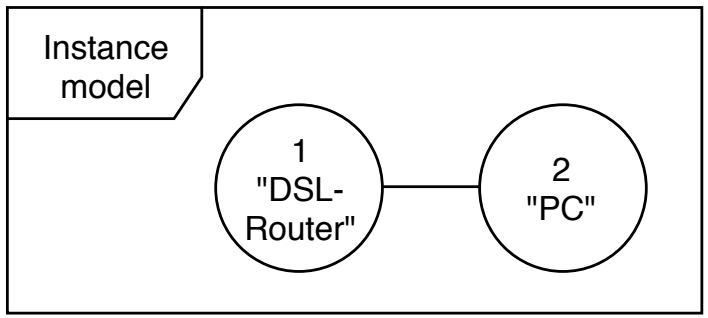
Evolution Example





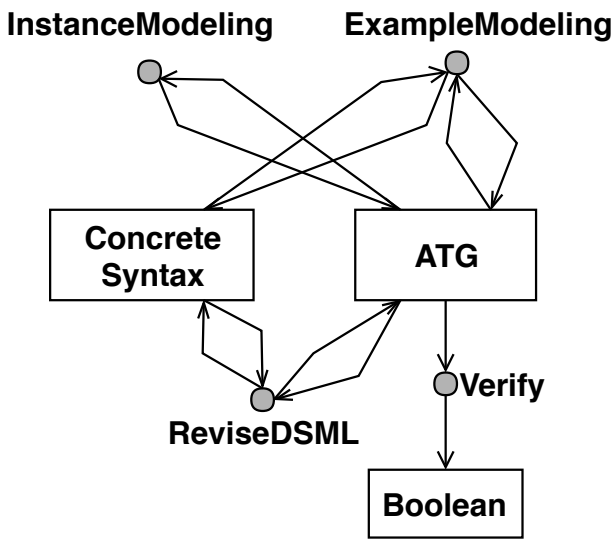
Evolution Example

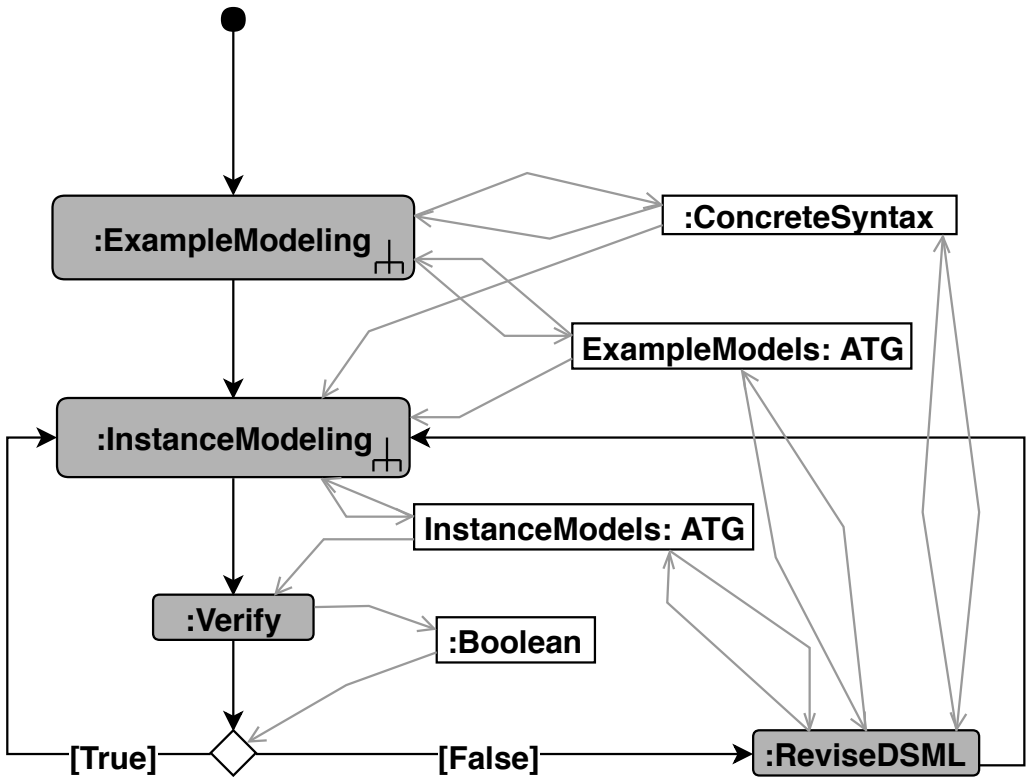




Review: Moodling Process

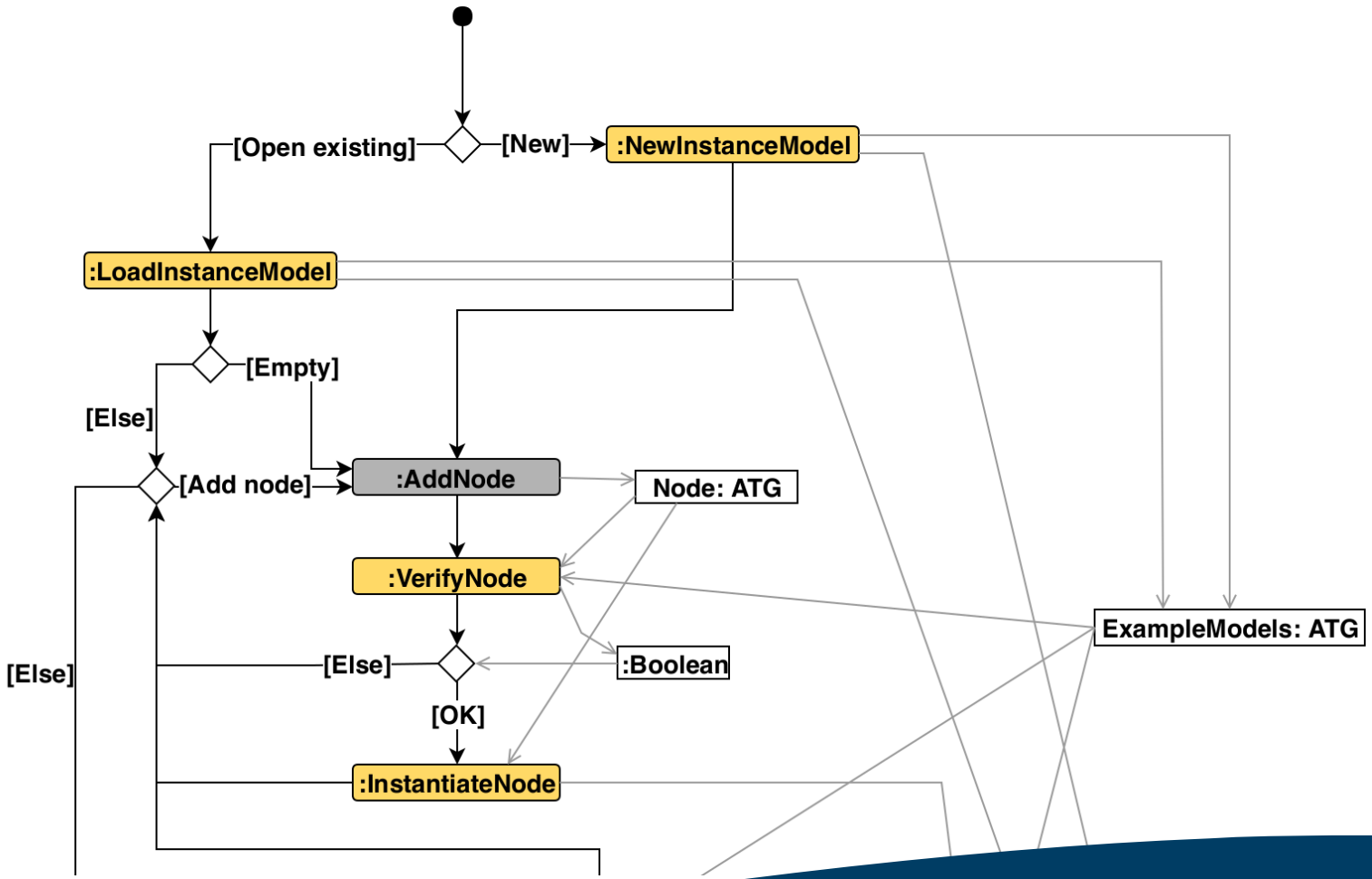
FTG+PM for an agile and integrated example-driven DSML design:





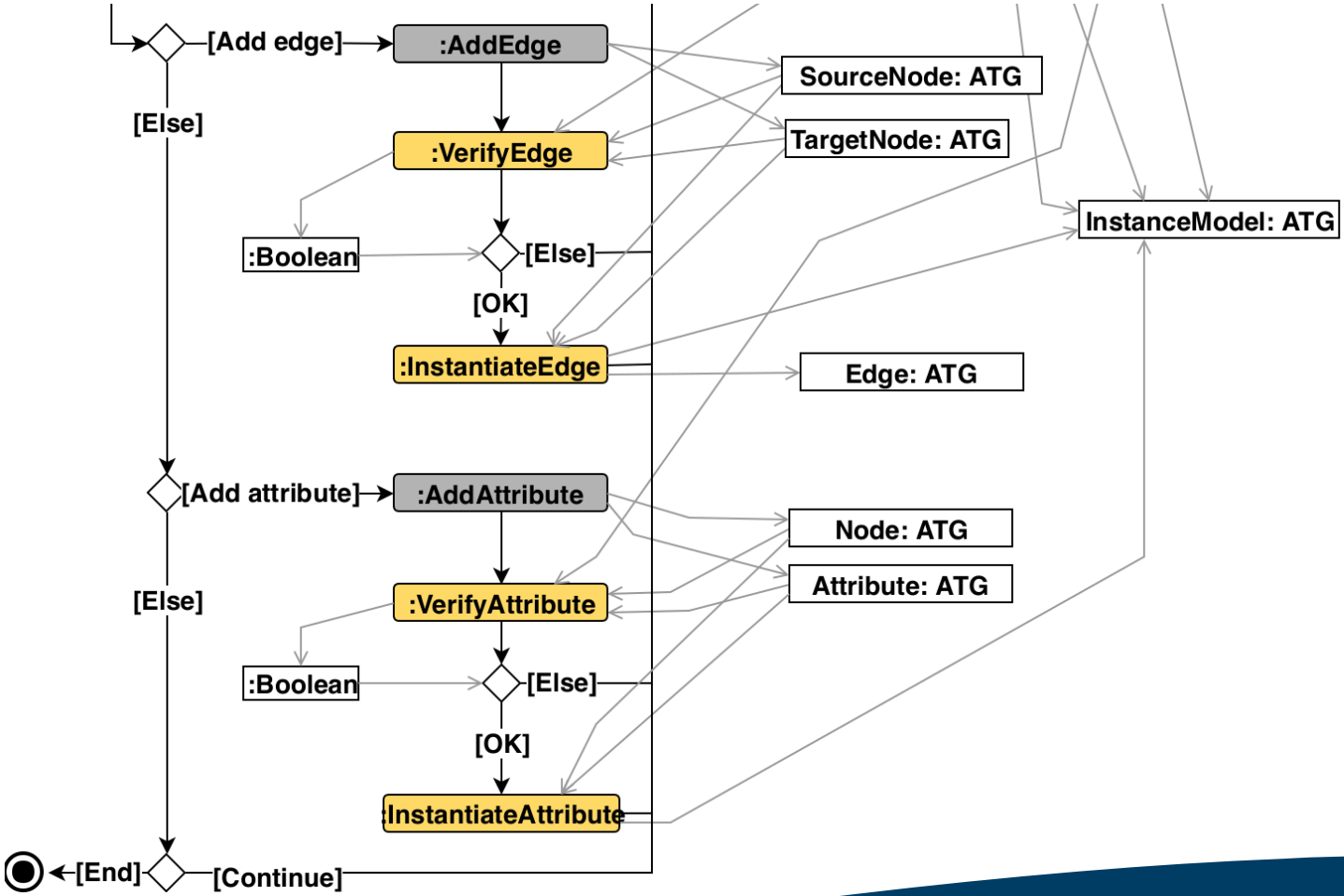
Moodling Process

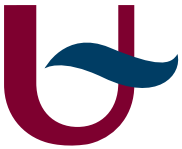
Instance-modeling activity detailed (1):



Moodling Process

Instance-modeling activity detailed (2):

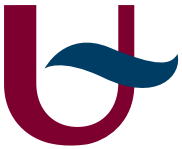




Implementation

Overview:

- *Modelverse* as metamodeling back-end
  - Flexible
  - Very little constraints
  - Python API
- *Qt* for graphical front-end
  - Mature and documented
  - Exhaustive feature set (UI Widgets, 2D, Statecharts, ...)
  - Python bindings



Two UI Modes

Unconstrained example modeling:

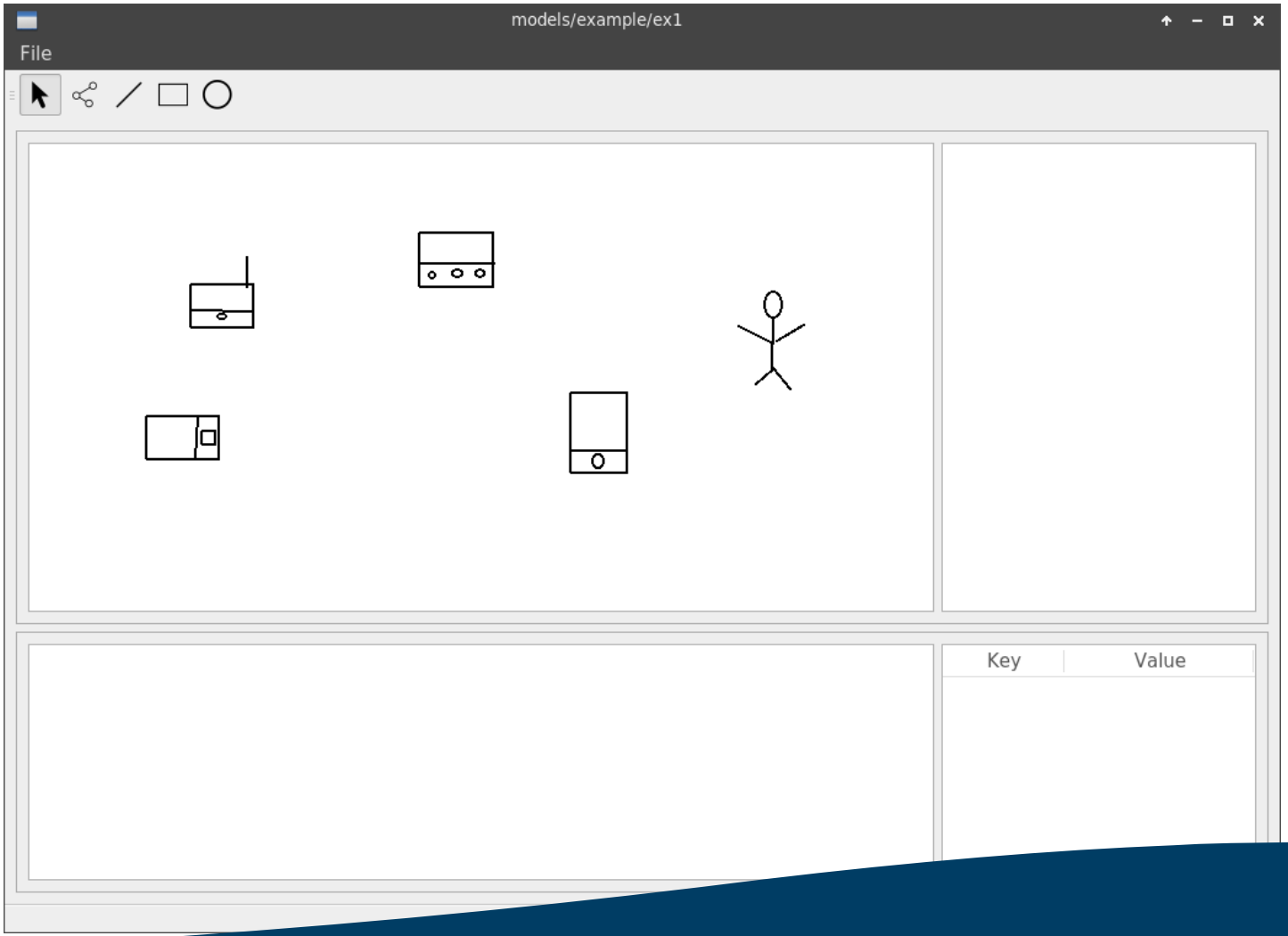
- Sketch example models similar to drawing tool
- Only constraint: conformance to ATG metamodel

Constrained instance modeling:

- Create models that conform to ATG metamodel and example models
- Implements example-model conformance relationship

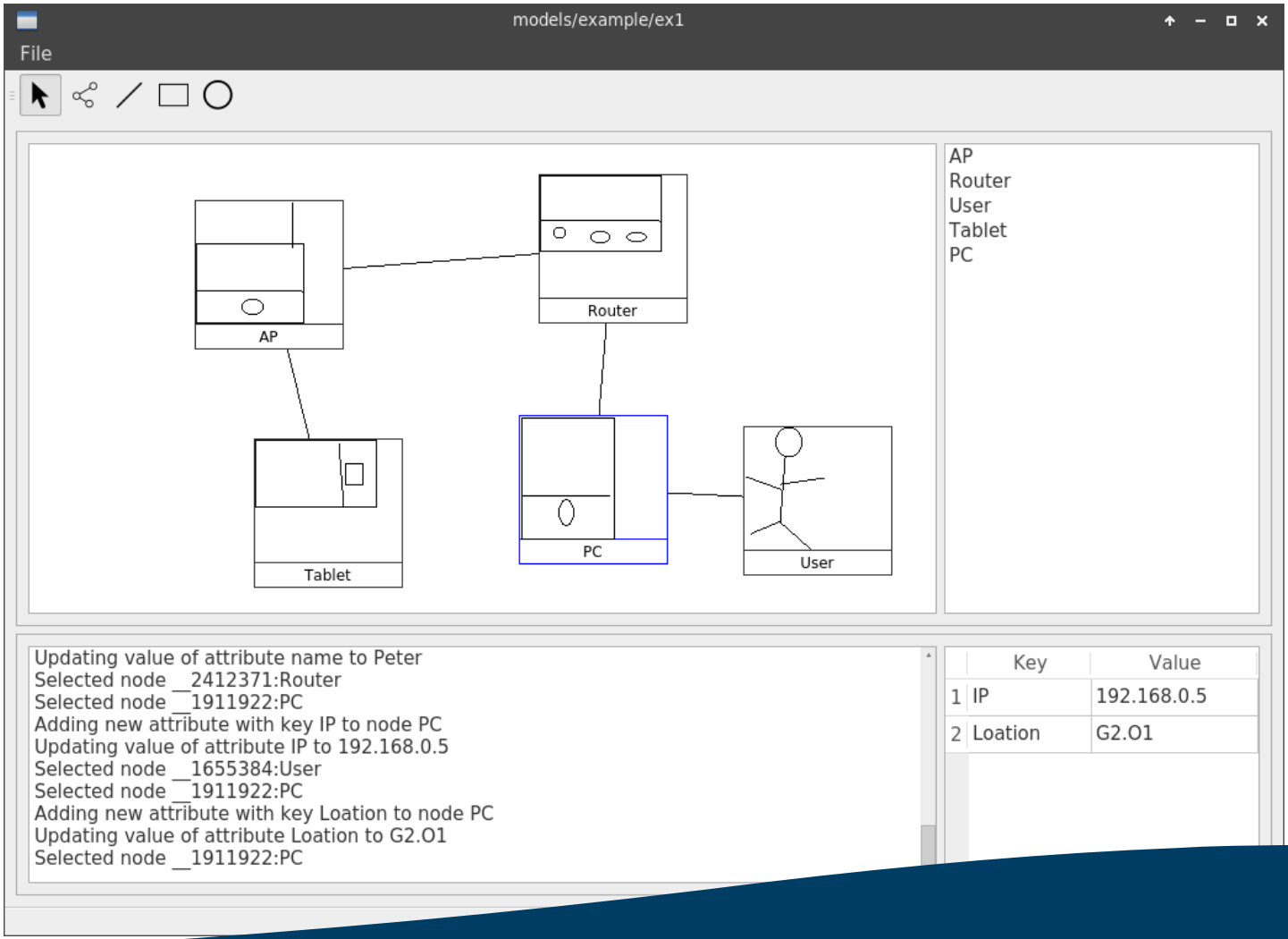
Example Modeling

1. Sketch



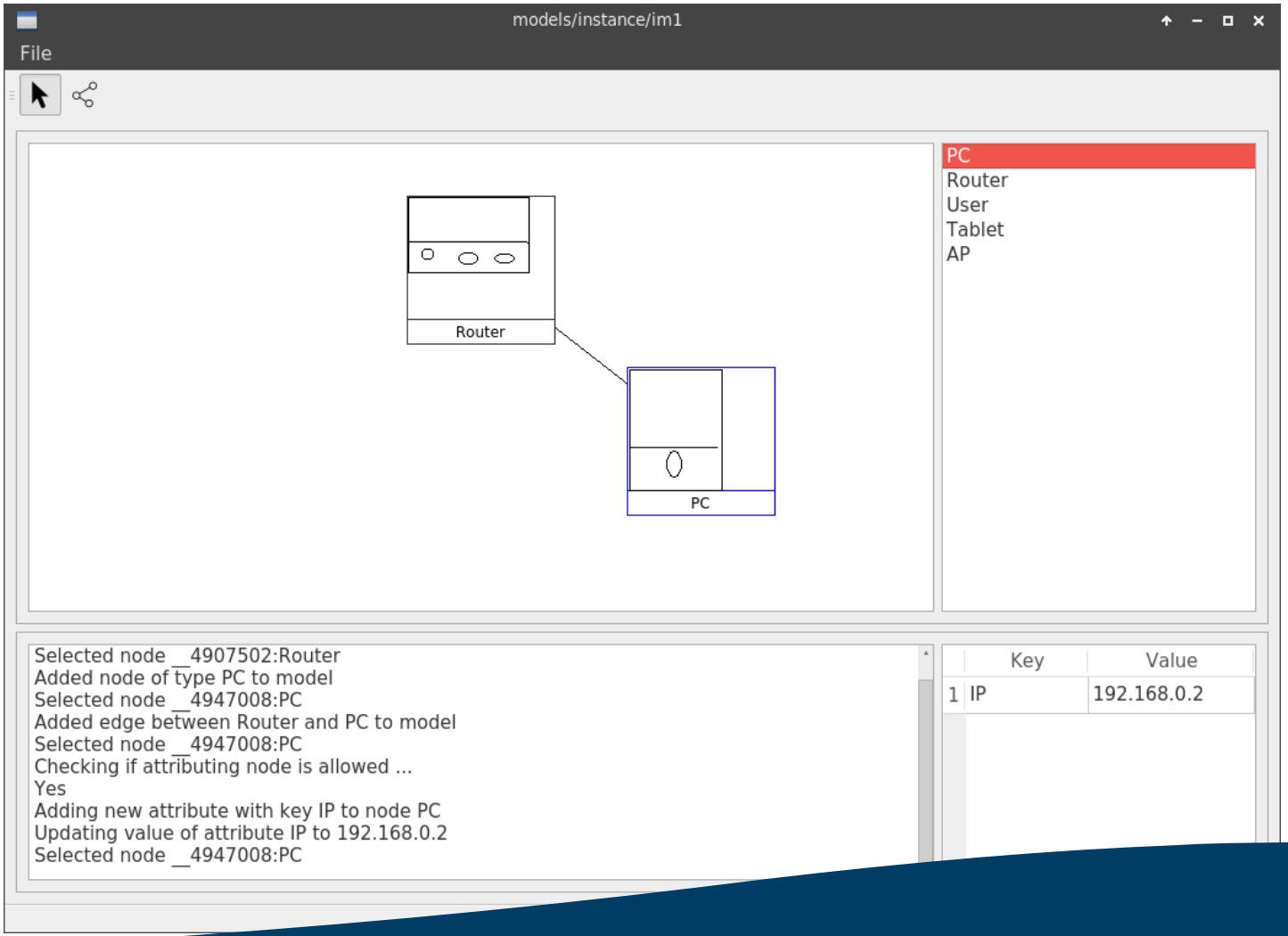
Example Modeling

2. Type, connect and attribute



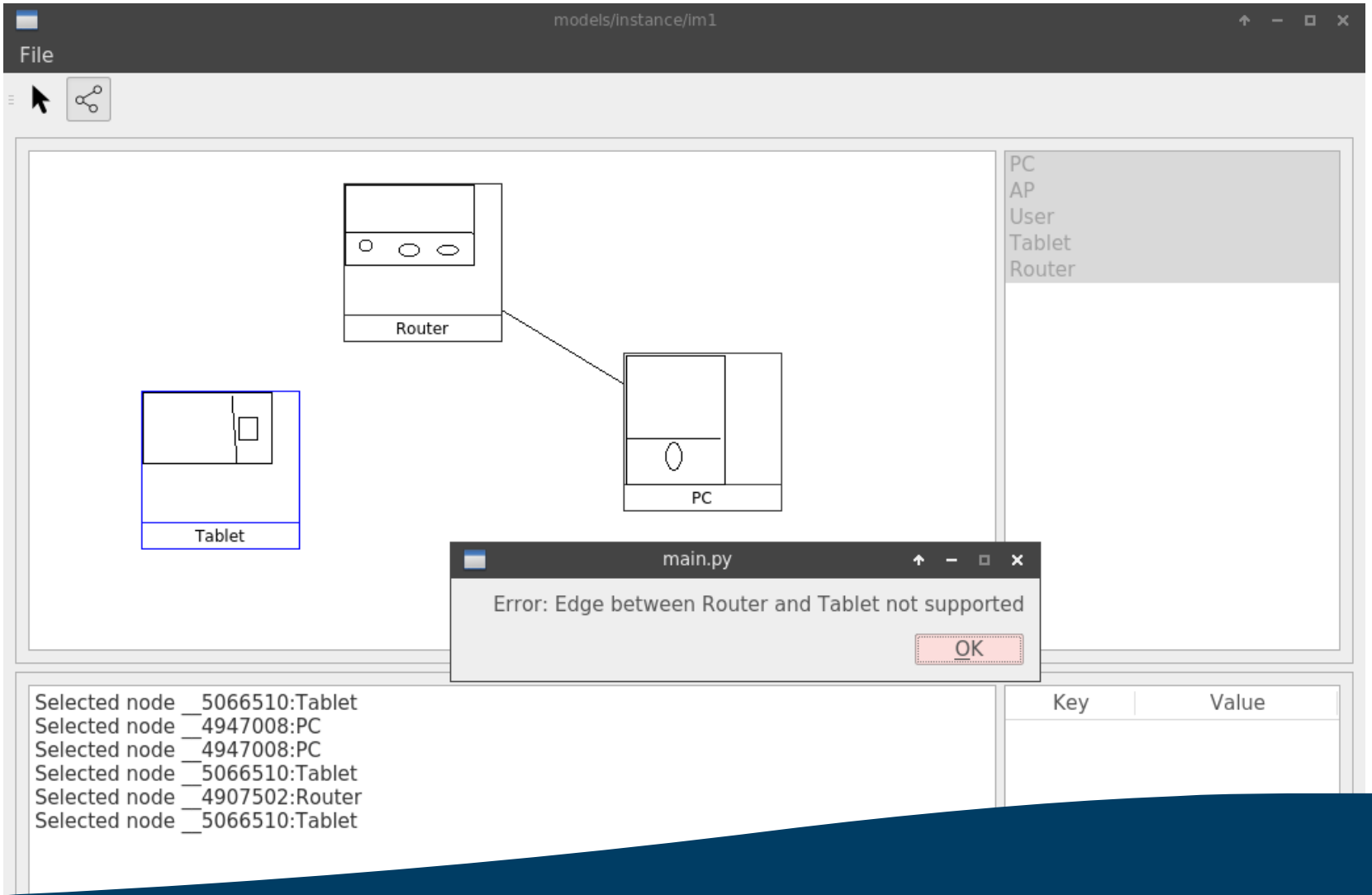
Instance Modeling

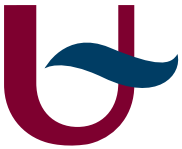
Overview



Instance Modeling

Constrained *add edge* operation





Co-Evolution

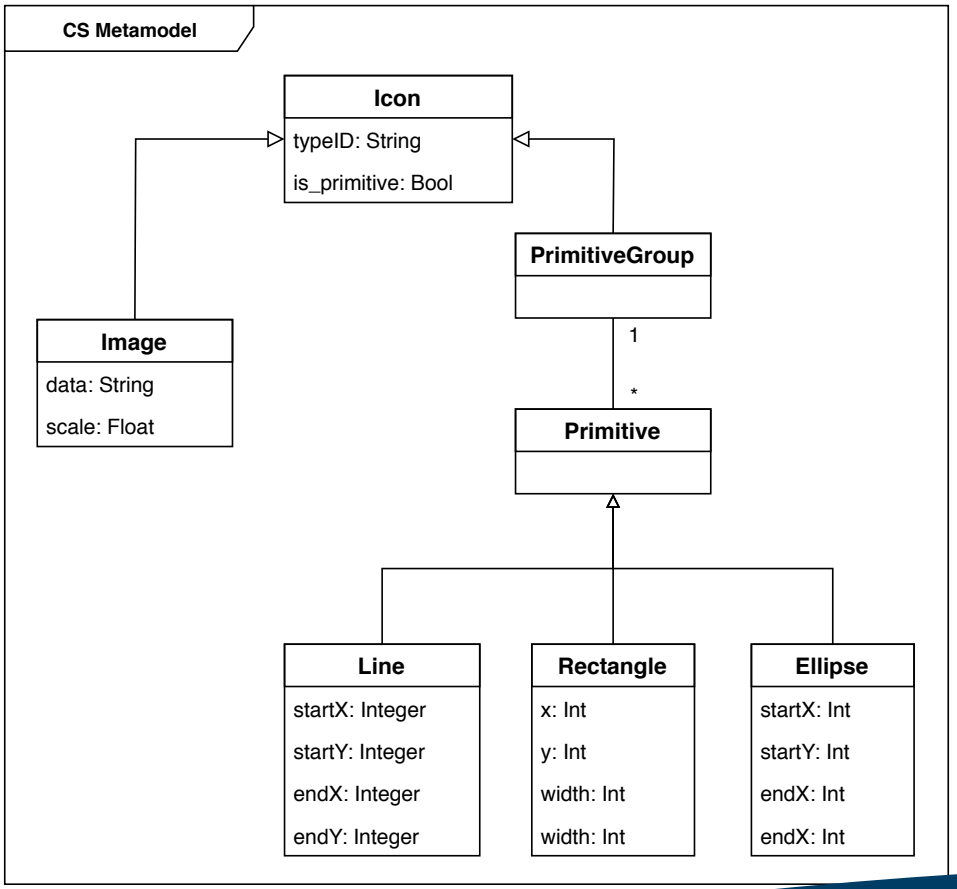
General **scheme:**

Given a change transformation T,

1. Apply it to the example model
2. Check for invalidated instance models
3. Repair conformance by applying T to invalid instance model

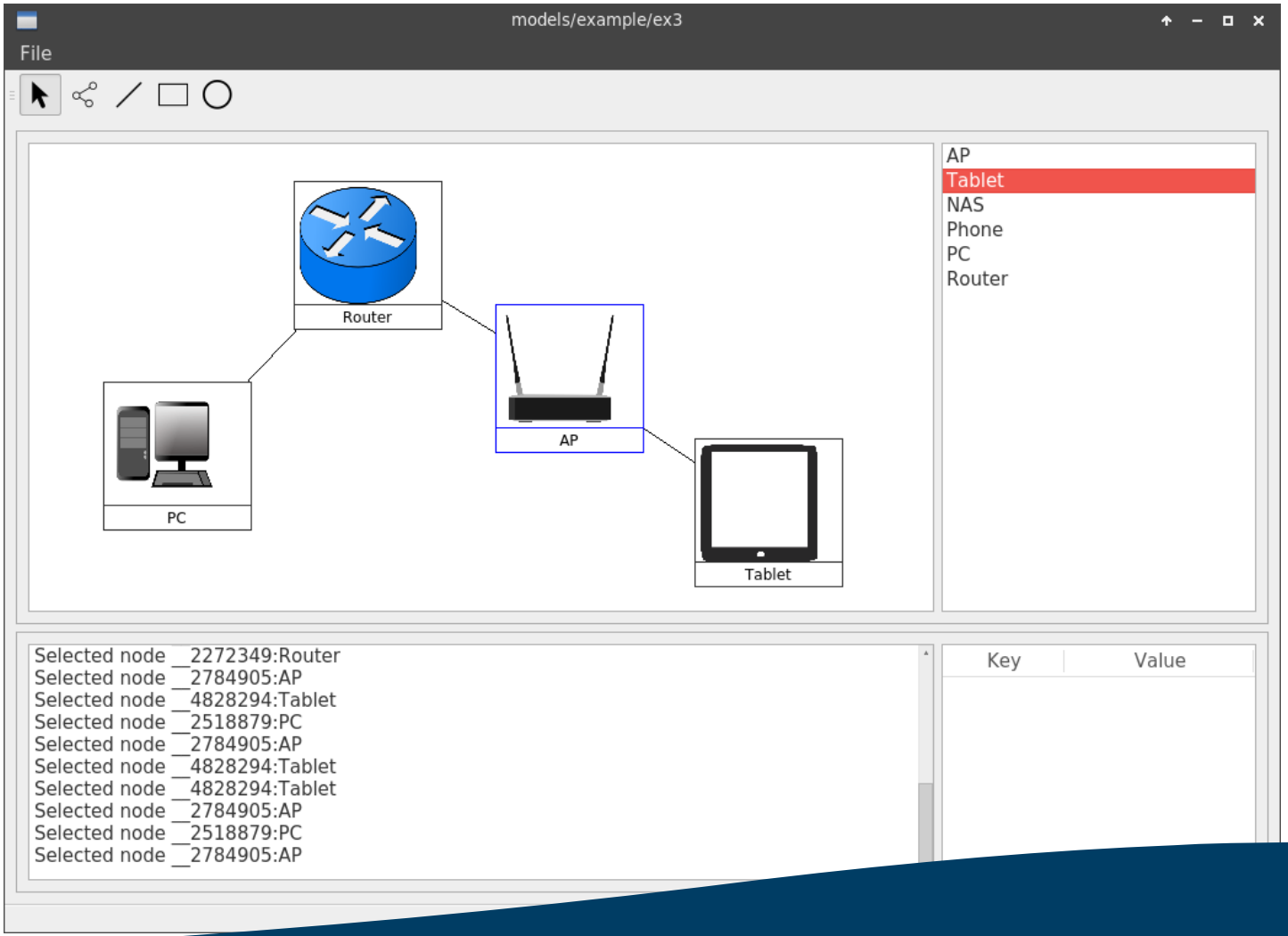
Concrete Syntax

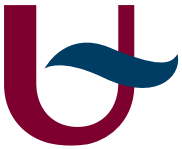
Explicitly modeled → Stored as model of **concrete syntax metamodel:**



Concrete Syntax

Can **evolve** concrete syntax:





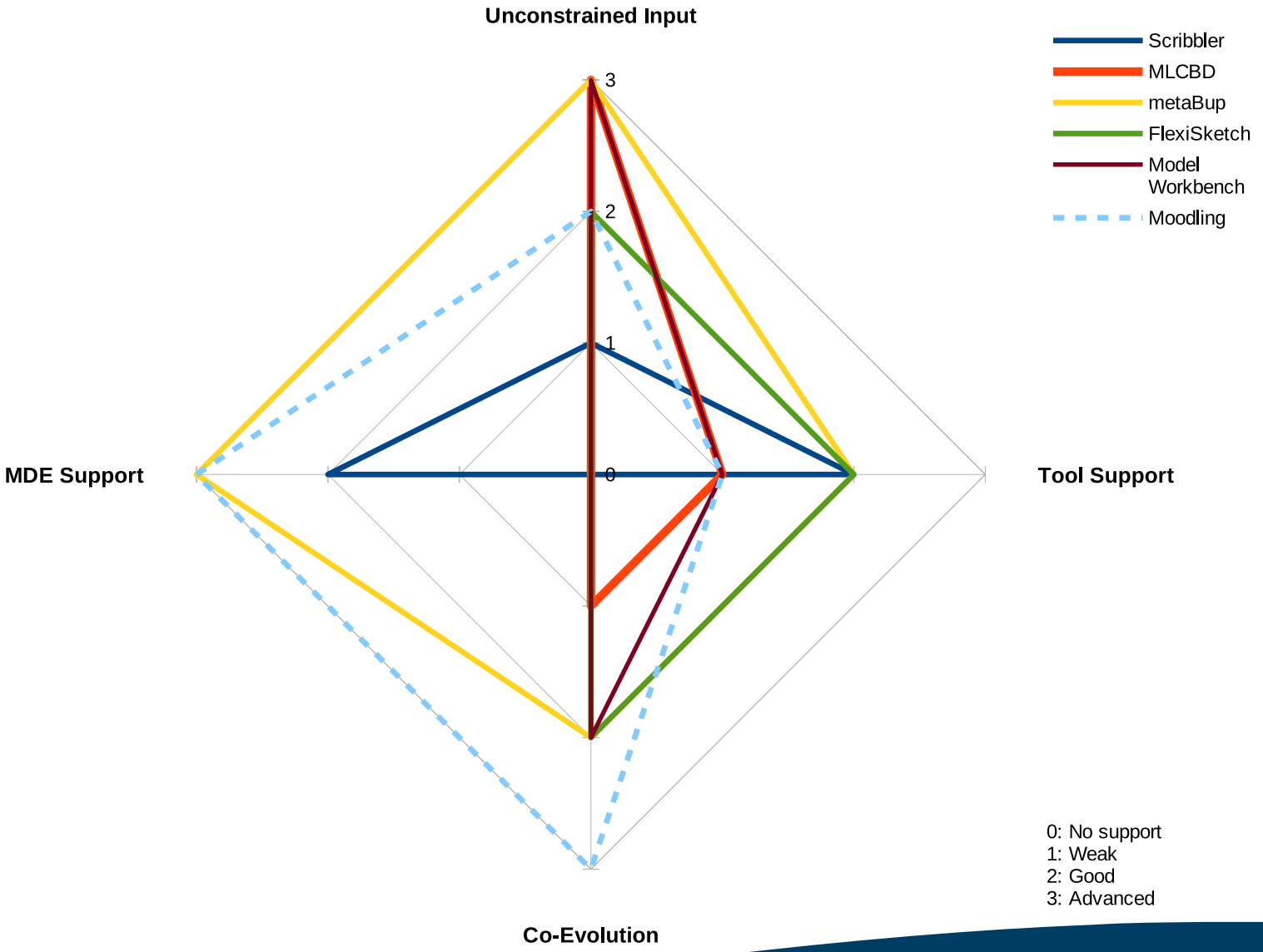
Evaluation

The approach is

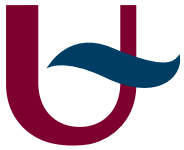
1. **Integrated:** Implemented in a metamodeling environment
   - No difference between example- and instance models (reusability)
   - Can directly apply MDE activities

2. **Agile:** Switch between design and use at any time
   - Automatic co-evolution of instance models when example models change
   - Quickly react to new or changing requirements

Comparison



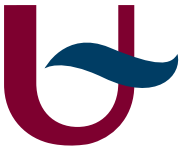
0: No support
1: Weak
2: Good
3: Advanced



Conclusion

Example-driven DSML design process was developed:

- Integrated in metamodeling environment
- Example models are primary language description by defining conformance of instance models
- Metamodeling aspects hidden
- Automated co-evolution of instance models
- Evolution of concrete syntax

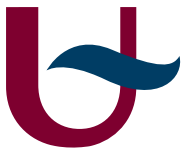


Review

What about ...

1. *RAMification* for transformations? [5]
2. Interoperability with other tools?
3. Constraints, abstraction, inheritance, hierarchy?
4. Requirements?

[5] Kühne, T., et al. Explicit transformation modeling. In *International Conference on Model Driven Engineering Languages and Systems* (240-255), Springer, 2009



Future Work

1. Process integrated, but strict separation of design and use phases still exists
2. Use Moodling for brainstorming and generate metamodel later
3. Preceding requirements engineering phase
4. Performance and usability of front-end