DSM TP 2014
Theory and Practice

5th International Summer School
on Domain Specific Modeling

Antwerp, Belgium
25 - 29 August

Model Transformation

Adapted from the Slides by Prof. Eugene Syriani

Eugene Syriani

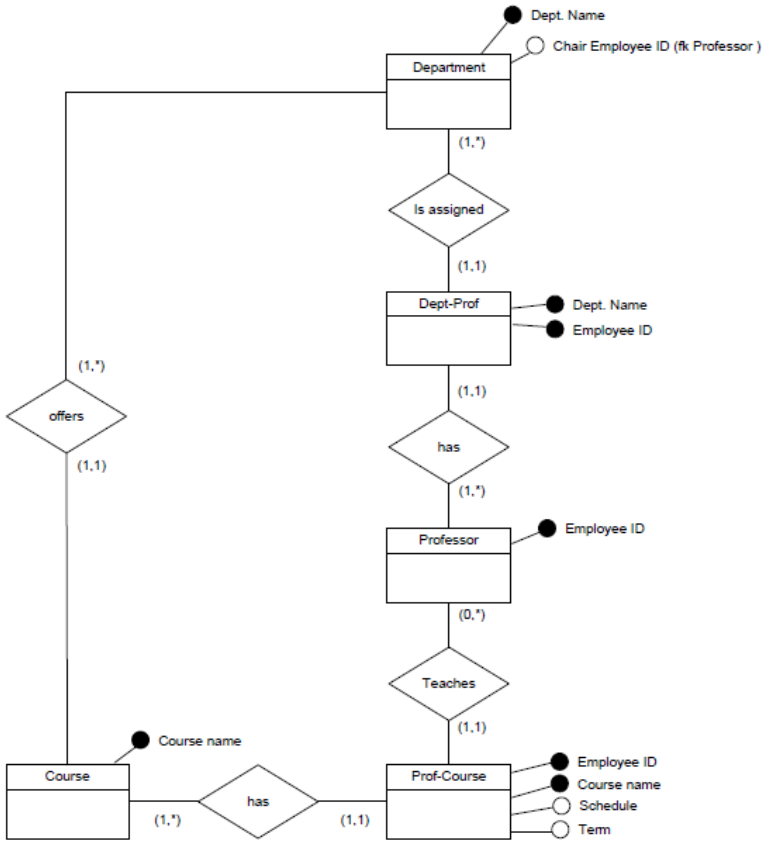
geodes
Département d'**informatique et de recherche opérationnelle**

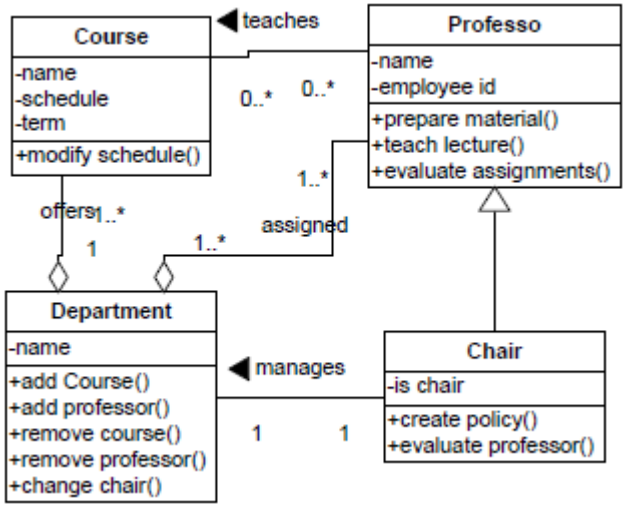
Université de Montréal

Motivation

Motivation

Suppose I ask you to provide a software that converts any E-R diagram into a UML class diagram, how would you achieve that?





Motivation

- Assumptions in E-R:
  - Entities & relations can contain attributes
  - Attributes can be of type: NUM, CHAR, TIMESTAMP, BIT
  - An entity may have one or more primary attributes
  - Relations relate 1-* or *-* entities
  - IS-A relationship between entities can be used
- Assumptions in UML CD:
  - Classes, associations, attributes, and inheritance can be used
  - Attributes may be of any type
  - OCL constraints may be defined

The "programming" solution

- Write a program that takes as input a .ER file and outputs a .UML file
- What are the issues?
  - What if the ER file is a diagram? in XML format? Probably end up limiting input from a specific tool only
  - Similarly in UML, should I output a diagram (in Dia or Visio)? In XMI? In code (Java, C#)?
  - How do I organize my program?
    - Requires knowledge from both domains
    - Need a loader (from input file)
    - Need some kind of visitor to traverse the model, probably graph-like data structure
    - Need to encode a "transformer"
    - Need to develop a UML printer
- Not an easy task after all…

The “modeling” way

1. Describe a meta-model of ER
   - Define concepts and concrete visual syntax
   - Generate an editor
2. Describe a meta-model of UML
3. Define a transformation T: $MM_{ER} \rightarrow MM_{UML}$
   - This is done in the form of rules with pre/post-conditions
     - describes “what to transform” instead of “how to transform”

- Code is automatically generated from the transformation model to a transformation instance that produces the result
- Some model transformation languages give you a bi-directional solution for free!

Pros & Cons

Programming solution

+ Programming techniques are well-proven, it is a reliable solution
+ Defined at the level of the code
+ Evolution, extension and maintenance more tedious
+ More likely to make errors
+ Incoherent abstraction mismatch between
  - The in/output artifacts: they represent designs models
  - The transformation between them: which is pure code

Pros & Cons

Modeling solution

+ In/output & transformation models are all defined at the same level of abstraction, in the same domain:
  - No need to add an extra “programmer” resource to the project
+ Much faster solution thanks to rule-based approach & automatic code synthesis
+ Alteration of the transformation process are automatically reflected in the final software product
+ You get a modeling environment for ER & UML for free!
  - No need to read from external non-standard tool anymore
+ Younger technology, few people understand it & master it, many challenges still need to be solved

Pros & Cons

In practice

- Typically encounter the same problems in modeling as in programing solutions
- The difference is that you can find the problems more easily, fix them very quickly and re-deploy the solution automatically
- Developer not required to be in programmers: who defines the requirements can develop the solution
- The bottom line is that you save time, reduce the cost, fulfill the entire scope, and deliver a high-quality product

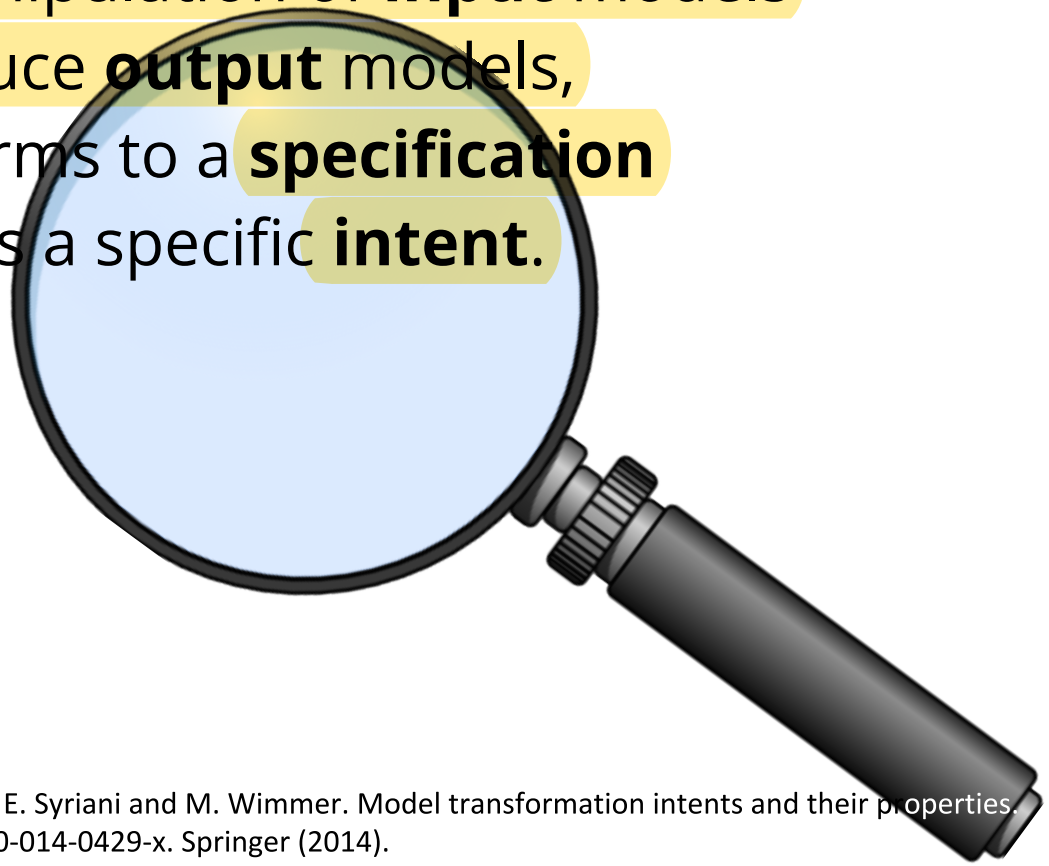
So what are we doing here?

- It seems that Model-based Design is the “Holy Grail” of software engineering
- Well, the devil is in the details…
- We will explore
  - what model transformations are
  - and how to design some

What is Model Transformation?

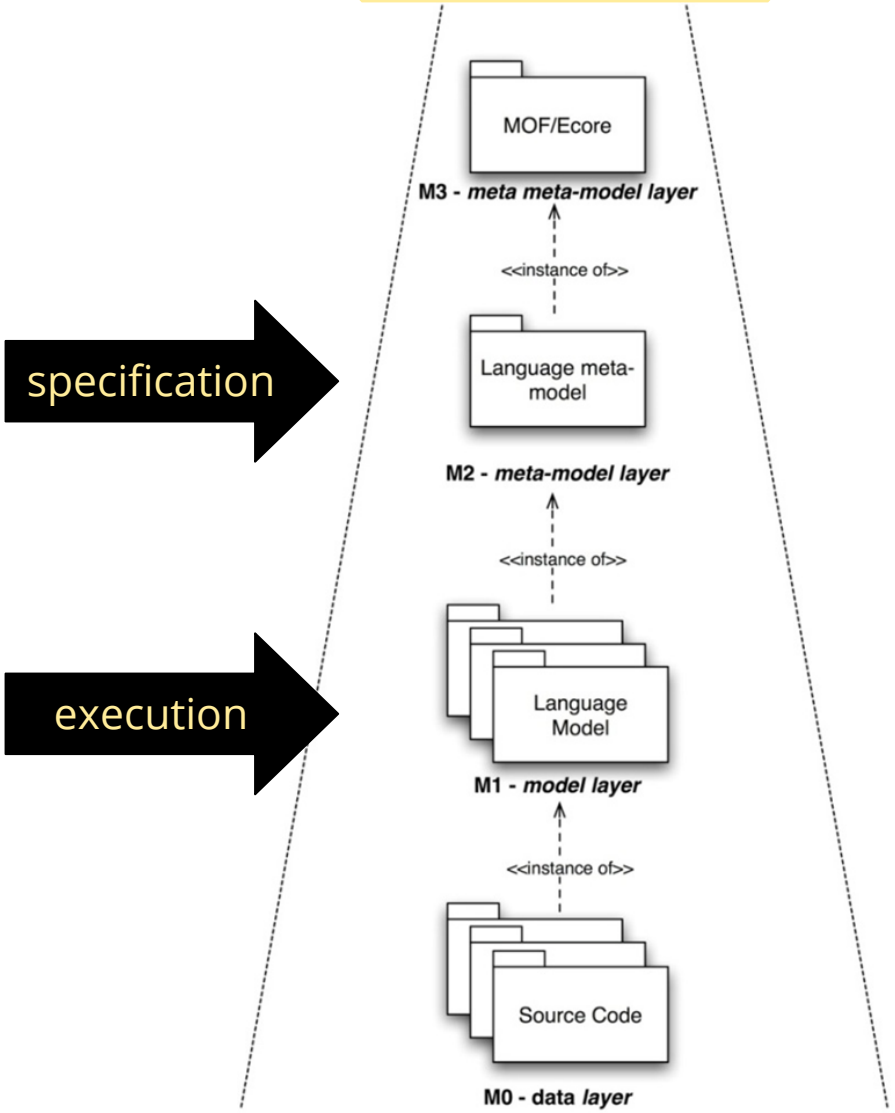
Definition

A model transformation is the
automatic manipulation of **input** models
to produce **output** models,
that conforms to a **specification**
and has a specific **intent**.

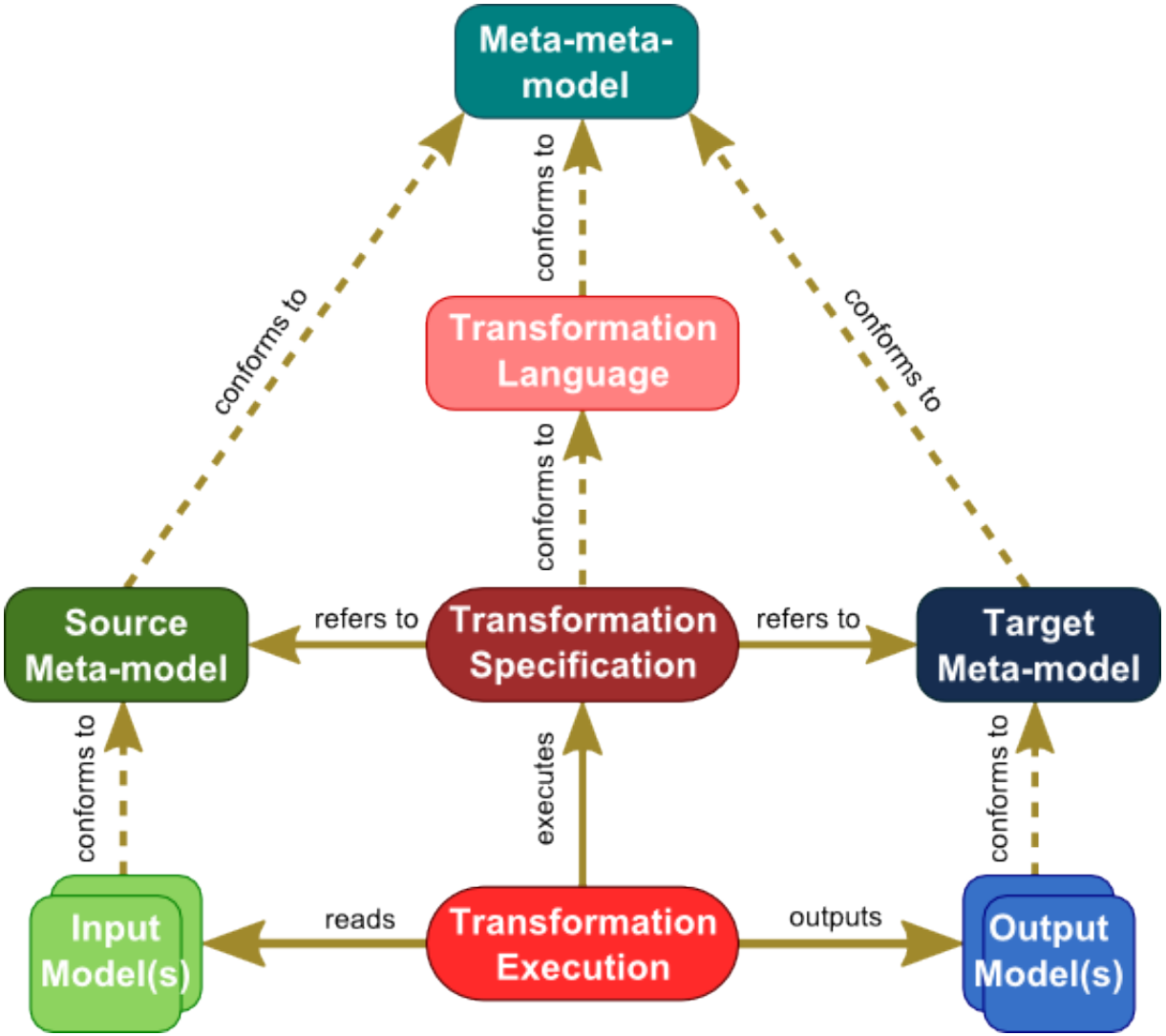


L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani and M. Wimmer. Model transformation intents and their properties. *Journal on Software and Systems Modeling*: 10.1007/s10270-014-0429-x. Springer (2014).

Where should MT be specified and executed?



Terminology



Model transformation approaches

- Model-to-text
  - **Visitor-based**: traverse the model in an object-oriented framework
  - **Template-based**: target syntax with meta-code to access source model
- Model-to-Model
  - **Direct manipulation**: access to the API of M3 and modify the models directly
  - **Operational**: similar to direct manipulation but at the model-level (OCL)
  - **Rule-based**
    - **Graph transformation**: implements directly the theory of graph transformation, where models are represented as typed, attributed, labelled, graphs in category theory. It is a declarative way of describing operations on models.
    - **Relational**: declarative describing mathematical relations. It defines constraints relating source and target elements that need to be solved. They are naturally multi-directional, but in-place transformation is harder to achieve

Typical use cases of Model Transformation

Model transformation intent classification

Refinement
- Refinement
- Synthesis
  - Serialization

Abstraction
- Abstraction
- Reverse Engineering
- Restrictive Query
- Approximation

Semantic Definition
- Translational Semantics
- Simulation

Language Translation
- Translation
- Migration

Constraint Satisfaction
- Model Finding
- Model Generation

Analysis

Editing
- Model Editing
- Optimization
- Model Refactoring
- Normalization
  - Canonicalization

Model Visualization
- Animation
- Rendering
- Parsing

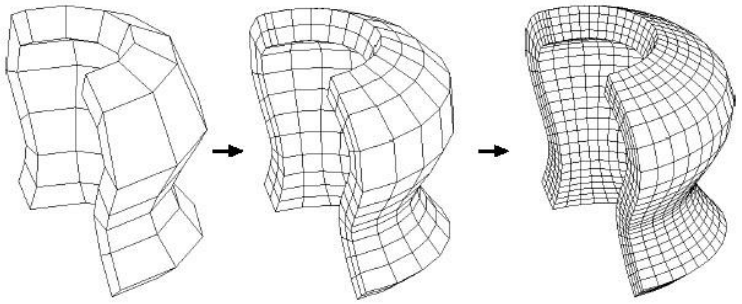
Model Composition
- Model Merging
- Model Matching
- Model Synchronization

L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani and M. Wimmer. Model transformation intents and their properties. *Journal on Software and Systems Modeling*: 10.1007/s10270-014-0429-x. Springer (2014).

Refinement category

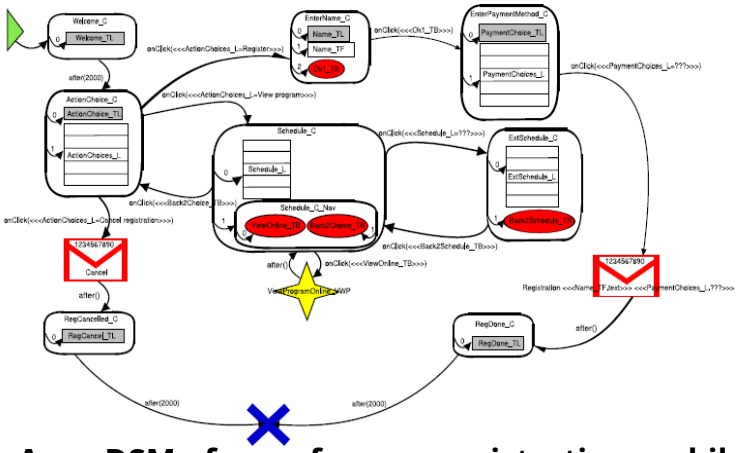
Groups intents that produce a more precise model by reducing design choices and ambiguities with respect to a target platform.

- Refinement
- Synthesis
  - Serialization

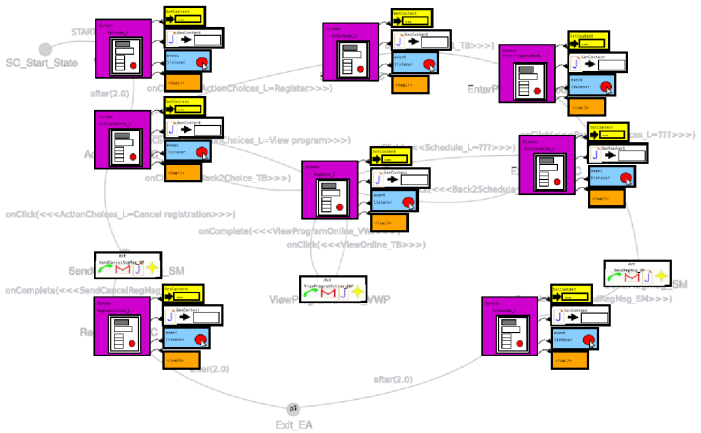


Refinement

- Transform from a higher level specification (e.g., PIM) to a lower level description (e.g., PSM)
- Adds precision to models
- M_1 refines M_2 if M_1 can answer all questions that M_2 can
- Typically M_1 contains at least the same information as M_2



PhoneApps DSM of a conference registration mobile application



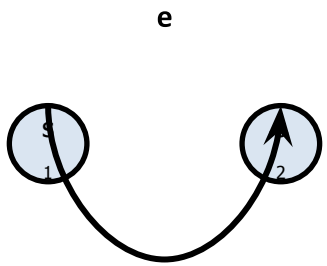
Representation of the model in AndroidAppScreens

PhoneApps DSL To Statecharts

J. Denil et al. Automatic Deployment Space Exploration Using Refinement Transformations. *MPM workshop*, 2011.

Synthesis

- Refinement where the output is an executable artifact expressed in a well-defined language format (typically textual)
- Model-to-code generation: transformation that produces source code in a target programming language
- Refinement often precedes synthesis



Statecharts model

Statecharts to Python Compiler

```
if e == 0:                  # event "e"
        if table[1] and self.isInState(1) and self.testCondition(3):
        if (scheduler == self or scheduler == None) and table[1]:
                self.runActionCode(4)      # output action(s1)
                self.runExitActionsForStates(-1)
                self.clearEnteredStates()
                self.changeState(1, 0)
                self.runEnterActionsForStates$self.StatesEntered, 1)
                self.applyMask(DigitalWatchStatechart.OrthogonalTable[1], table)
                handled = 1
        if table[0] and self.isInState(0) and self.testCondition(4):
                if (scheduler == self or scheduler == None) and table[0]:
                        self.runActionCode(5)      # output action(s2)
                        self.runExitActionsForStates(-1)
                        self.clearEnteredStates()
                        self.changeState(0, 0)
                        self.runEnterActionsForStates$self.StatesEntered, 1)
                        self.applyMask(DigitalWatchStatechart.OrthogonalTable
[0], table)
                        handled = 1
```

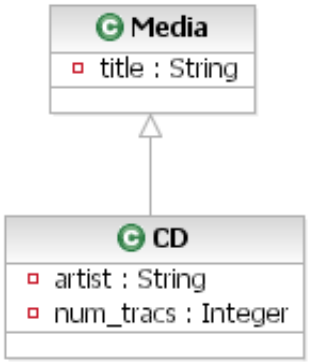
Generated Python code

R. Mannadiar and H. Vangheluwe. Modular synthesis of mobile device applications from domain-specific models. *MPM workshop*, 2011.

Serialization

- Special case of synthesis
- Goal is to store the model on some medium

Ecore model to XMI




```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xmi=http://www.omg.org/XMI
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="ml">
  <xsd:import namespace="http://www.omg.org/XMI"
     schemaLocation="XMI.xsd"/>
  <xsd:complexType name="Media">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="title" type="xsd:string"/>
  </xsd:complexType>
  <xsd:element name="Media" type="ml:Media"/>
  <xsd:complexType name="CD">
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="artist" type="xsd:string"/>
    <xsd:attribute name="num_tracs" type="xsd:int"/>
  </xsd:complexType>
  <xsd:element name="CD" type="ml:CD"/>
</xsd:schema>
```

Model-to-Text transformation

- Generate text automatically from models
  - Executable text
    - source code language
    - serialization
  - Documentation
    - HTML
    - Javadoc
  - Readable artifact
    - Latex
    - Mass mailing letter
  - Any textual artifact…
- We will focus on source code generation, but the techniques are similar for the others as well

Preliminary assumptions

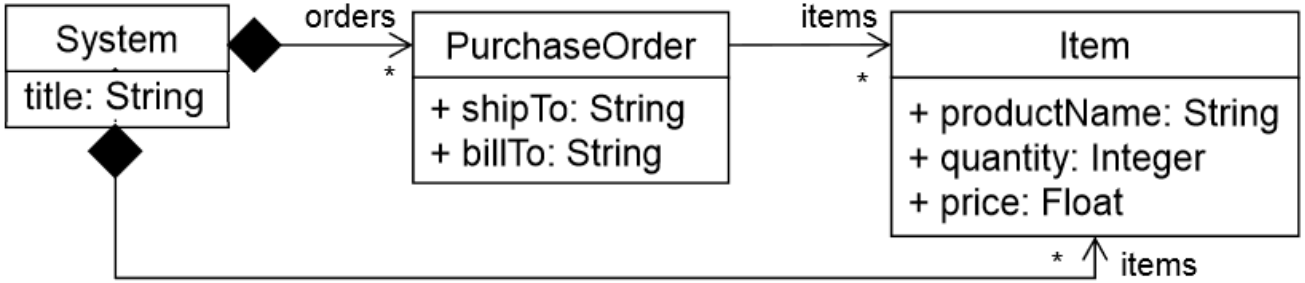
- Meta-model/abstract syntax already exists
- Transformations involving the meta-model always exist
  - No side effects!
- The framework has a context where models are read, and the resulting code can be executed
  - Code generator must generate code that adheres to the context as well

Template-based code generation

Templates

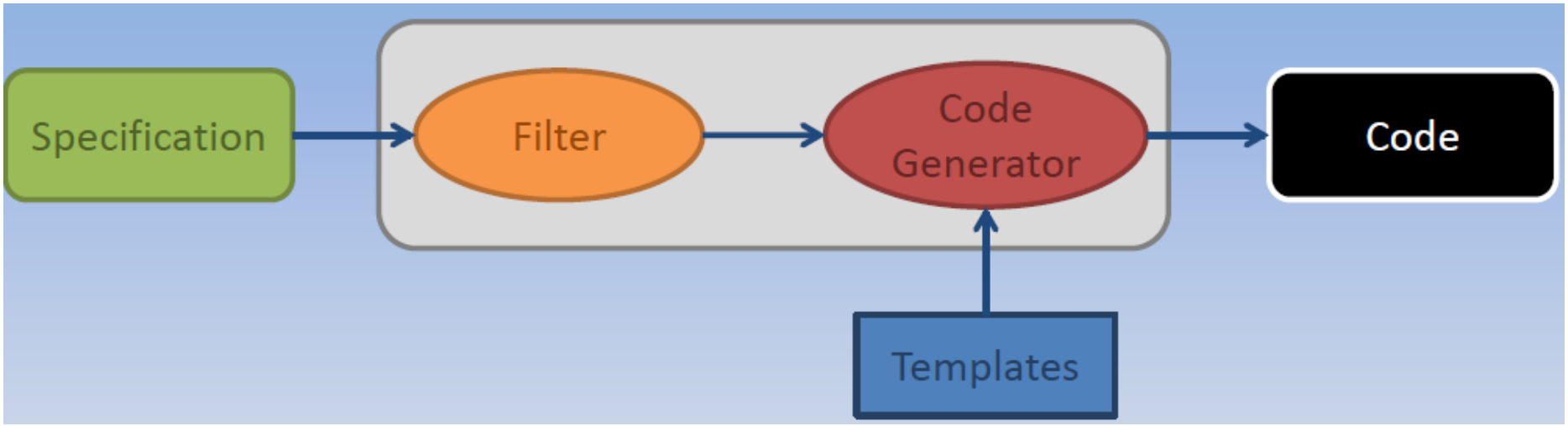
- Static parts
  - Text that will appear “as is” in the output
  - White space and formatting is preserved
- Dynamic parts
  - Executed content
  - Meta-code to access information from source model to select part of the model

Templates




```
<|<html>
<|<body>
      <h3>[%= title %]</h3>
      <ul>
  [%
  for (order in PurchaseOrder.allInstances) {
  %]
        <li>Order billed to [%=order.billTo%] and sent to
                              [%=order.shipTo%]</li>
  [%
  }
  %]
      </ul>
    </body>
</|</html>
```

Template + Filtering



Code generation with XSLT

Model

```
<class name="Person" package="com.mycompany">
    <attribute name="name" type="String"/>
    <attribute name="age" type="int"/>
</class>
```

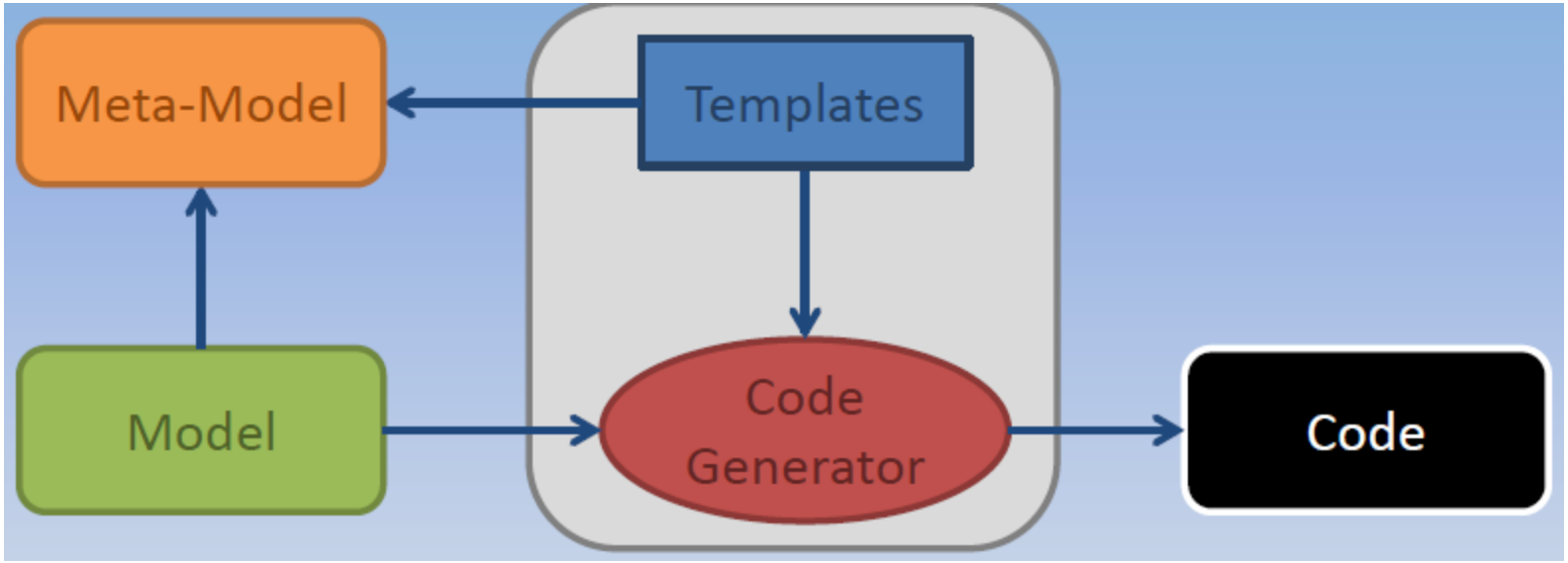
Generated code

```
package com.mycompany
public class Person {
    private String name;
    private int age;
    public String get_name() {return name;}
    public set_name(String name) {this.name = name;}
    public String get_age() {return age;}
    public set_age(String age) {this.age = age;}
}
```

Template

```
<xsl:template match="/class">
    package <xsl:value-of select="@package"/>;
    public class <xsl:value-of select="@name"/> {
        <xsl:apply-templates select="attribute"/>
    }
</xsl:template>

<xsl:template match="attribute">
    private <xsl:value-of select="@type"/>
        <xsl:value-of select="@name"/>;

    public <xsl:value-of select="@type"/>
        get_<xsl:value-of select="@name"/>() {
            return <xsl:value-of select="@name"/>;
    }

    public void set_<xsl:value-of select="@name"/>
        <xsl:value-of select="@type"/>
        <xsl:value-of select="@name"/>) {
            this.<xsl:value-of select="@name"/> =
                <xsl:value-of select="@name"/>;
    }
</xsl:template>
```

Template + Metamodel



Code generation with Xpand

Model

```
<?xml version="1.0" encoding="UTF-8"?>
<data:DataModel
  xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:data="http://www.openarchitectureware.org/oaw4.demo.emf.data
  <entity name="Person">
    <attribute name="name" type="String"/>
    <reference name="autos" toMany="true" target="//@entity.1"/>
  </entity>
  <entity name="Vehicle">
    <attribute name="plate" type="String"/>
  </entity>
</data:DataModel>
```

Generated code

```
public class Person {
  // bad practice
  public String lastName;
  private Vehicle[] autos;
}

public class Vehicle {
    // bad practice
    public String plate;
}
```

Template

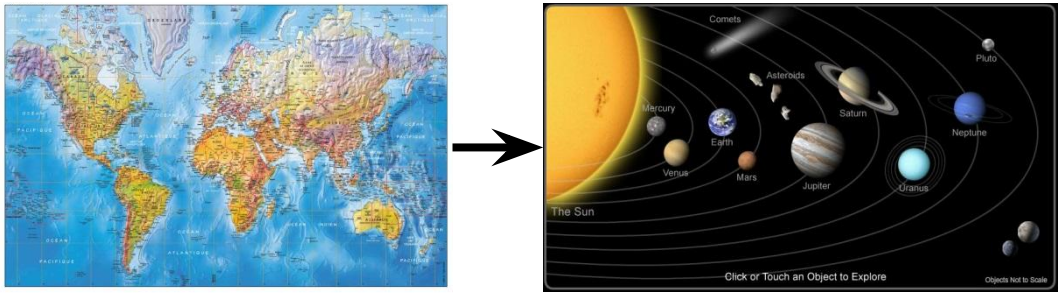
```
«DEFINE Root FOR data::DataModel»
 «EXPAND Entity FOREACH entity»
«ENDDEFINE»

«DEFINE Entity FOR data::Entity»
 «FILE name + ".java"»
  public class «name» {
   «FOREACH attribute AS a»
    // bad practice
    public «a.type» «a.name»;
   «ENDFOREACH»
   «FOREACH reference AS r»
    «IF r.toMany == false»
     private «r.target» «r.name»;
    «ELSE»
     private «r.target»[] «r.name»;
    «ENDIF»
   «ENDFOREACH»
  }
 «ENDFILE»
«ENDDEFINE»
```

13

Abstraction category

Inverse of refinement category.
Groups intents where some information of a model is aggregated or discarded to simplify the model and emphasize specific information.

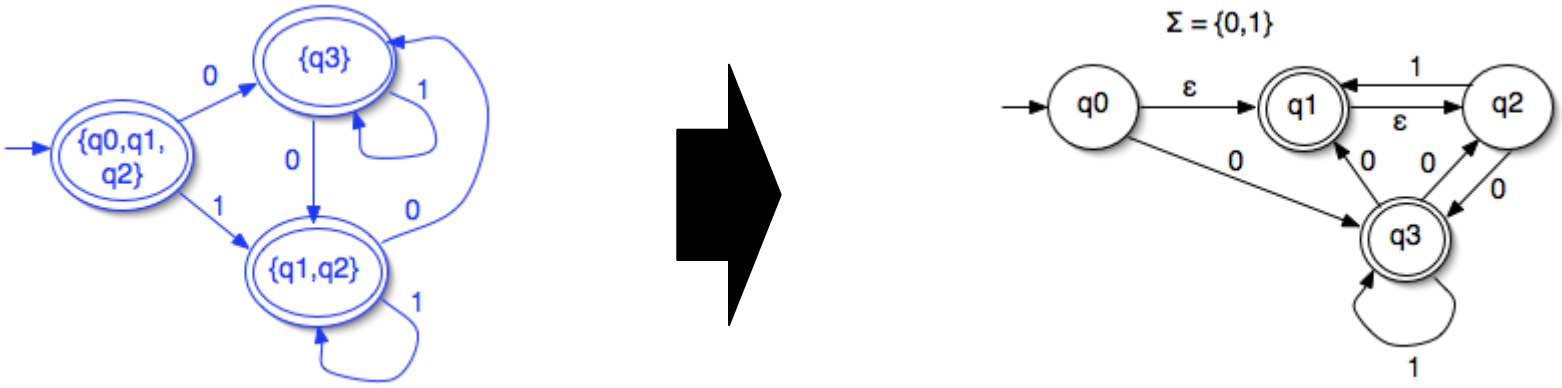
- Abstraction
- Restrictive Query
- *Reverse Engineering*
- *Approximation*



Abstraction

- Inverse of refinement
- Hides some information while revealing other
- If M_1 refines M_2 then M_2 is an abstraction of M_1

DFA to NFA



Deterministic state automata (DFA)

Non-deterministic state automata (NFA)

Abstraction (Continued)

- A view of a model that is not a sub-model, but an aggregation of some of its information is also a abstraction

Example:

"*Find all actors who played together in at least 3 movies and assign the average rating to each clique*" outputs a view of a model representing a subset of IMDB represented as a graph composed of strongly connected components with the ratings aggregating individual ratings.

T. Horn et al. The TTC 2014 Movie Database Case. *Transformation Tool Contest* 2014.

Restrictive Query

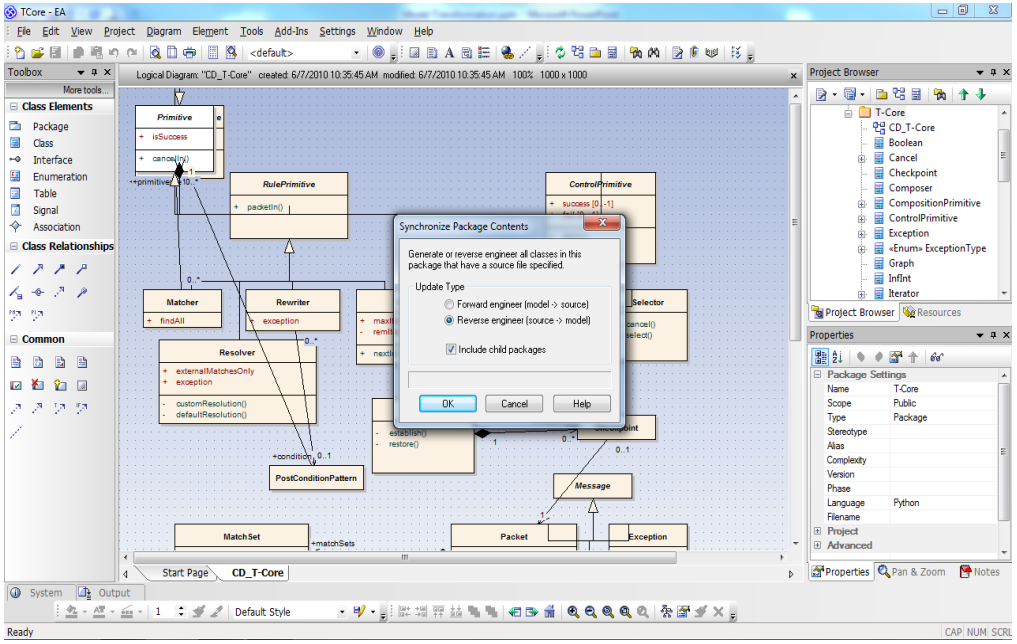
- A query requests some information about a model and returns that information in the form of a proper sub-model or a view
- A view is a projection of (a sub-set of) of the properties of M
- Restrictive query is a special case of abstraction where the result is a sub-model of the input model
  - Any subsequent aggregation or restructuring of the resulting sub-model is an abstraction
- Example: "*Get all the leaves of a tree*"
- Tool support: EMF INC-Query

Reverse Engineering

- Inverse of synthesis

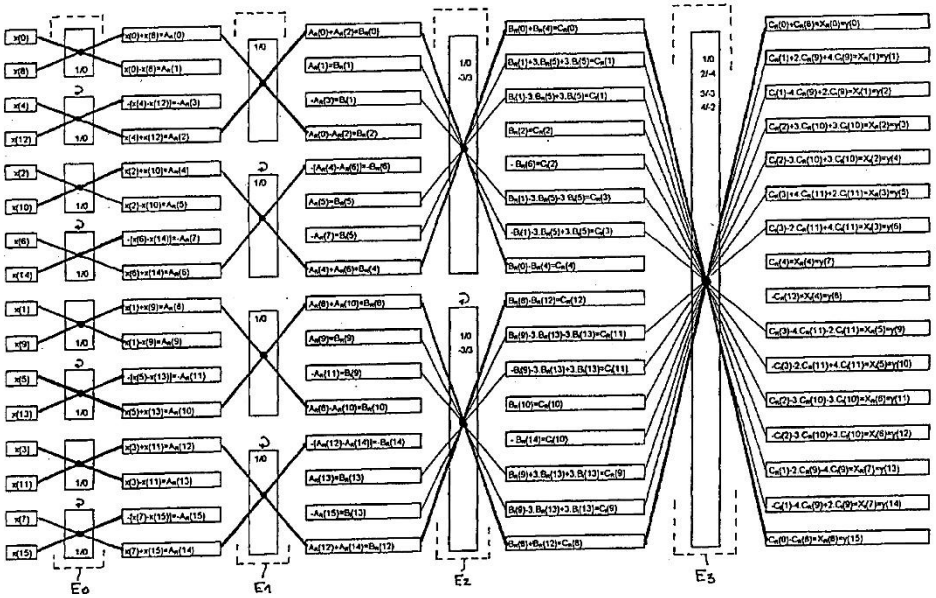
- Example:

  UML class diagrams can be generated from Java code with Fujaba



Approximation

- M_1 approximates M_2 if M_1 is equivalent to M_2 up to a certain margin of error
- M_1 preserves more properties of M_2 as the error decreases
- Margin of error typically based on a distance measure between models



Fast Fourrier Transform approximates a Fourrier Transform

QUESTION

Convert a class diagram to XMI in order to store the model in the cloud.

Input: Class diagram

Output: XML document

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Convert a class diagram to XMI in order to store the model in the cloud.

Input: Class diagram

Output: XML document

- Serialization

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Extract the class hierarchy from a class diagram, with single inheritance, in the form of a directed tree.

Input: Class diagram

Output: Tree

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Extract the class hierarchy from a class diagram, with single inheritance, in the form of a directed tree.

Input: Class diagram

Output: Tree

- Abstraction

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Augment a class diagram by adding navigability, role names, attribute types, method return and parameter types.

Input: Class diagram

Output: Class diagram

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Augment a class diagram by adding navigability, role names, attribute types, method return and parameter types.

Input: Class diagram

Output: Class diagram

- Refinement

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Extract the classes with no super-class from a class diagram.

Input: Class diagram

Output: Class diagram

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

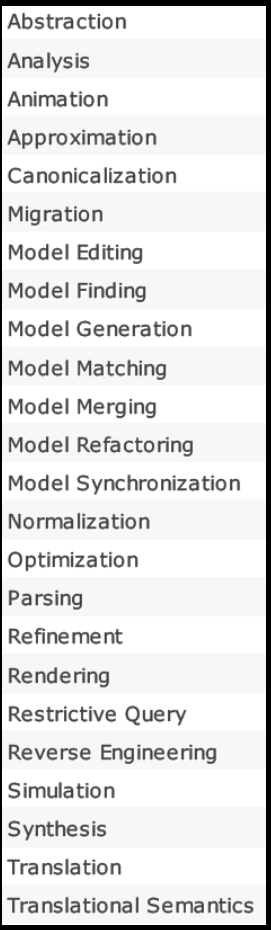
QUESTION

Extract the classes with no super-class from a class diagram.

Input: Class diagram

Output: Class diagram

- Restrictive Query



QUESTION

Generate JavaDocs from a class diagram.

Input: Class diagram

Output: HTML document

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

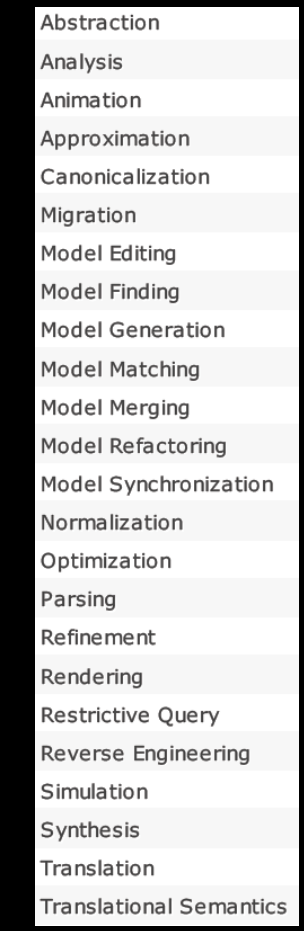
QUESTION

Generate JavaDocs from a class diagram.

Input: Class diagram

Output: HTML document

- Synthesis



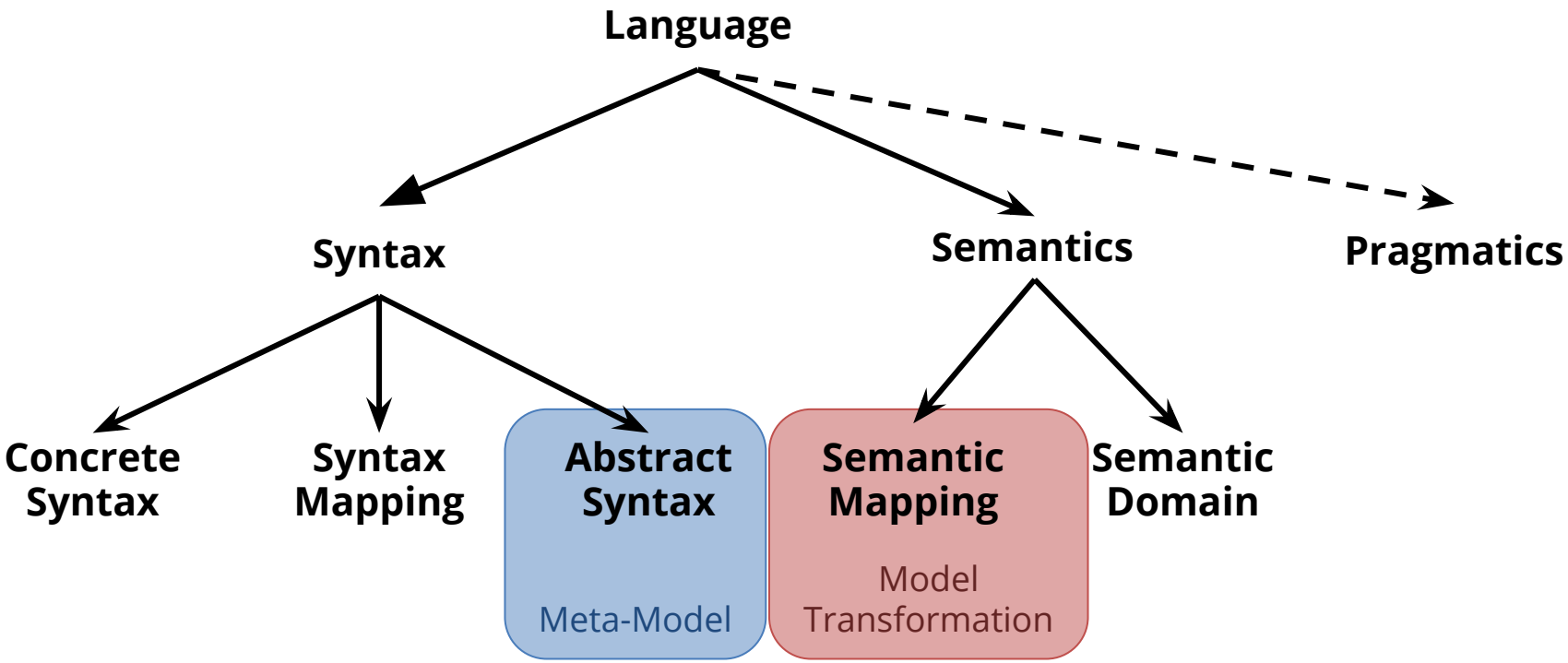
Semantic Definition category

Groups intents whose purpose is to define the semantics of a modeling language.

- Translational Semantics
- Simulation

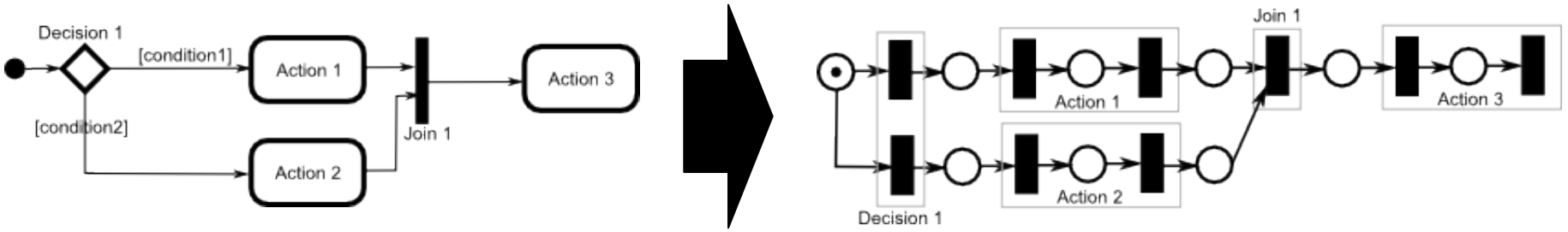
Translational Semantics

- Gives the meaning of a model in a source language in terms of the concepts of another target language
- Typically used to capture the semantics of new DSLs



Translational Semantics

- Example: Causal Block Diagram's semantics expressed as Ordinary Differential Equations

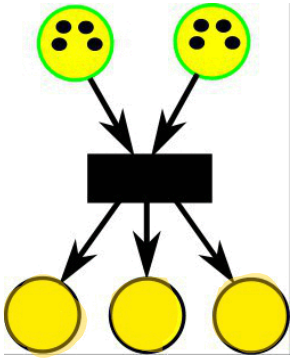


UML activity diagrams to Petri nets

H. Ergin and E. Syriani. Operational semantics of UML activity diagram: An application in project management. *MoDRE workshop*, 2012.

Simulation

- Defines the operational semantics of a modeling language that updates the state of the system modeled
- The source and target meta-models are identical
- The target model is an “updated” version of the source model: no new model is created
- Simulation updates the abstract syntax, which may trigger modifications in the concrete syntax



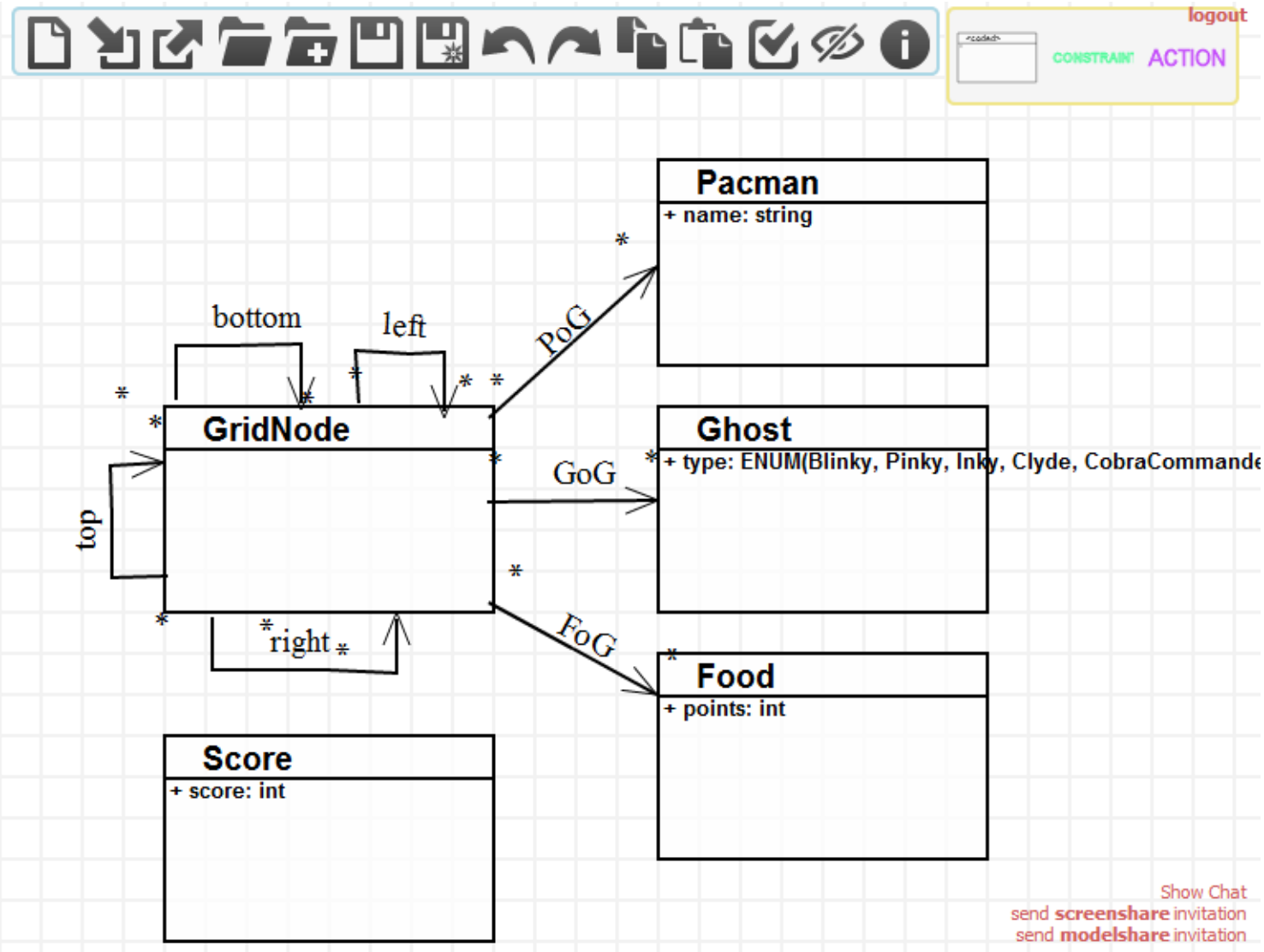
Petri net simulator

T. Kühne et al. Explicit Transformation Modeling. *MODELS workshops*, 2010.

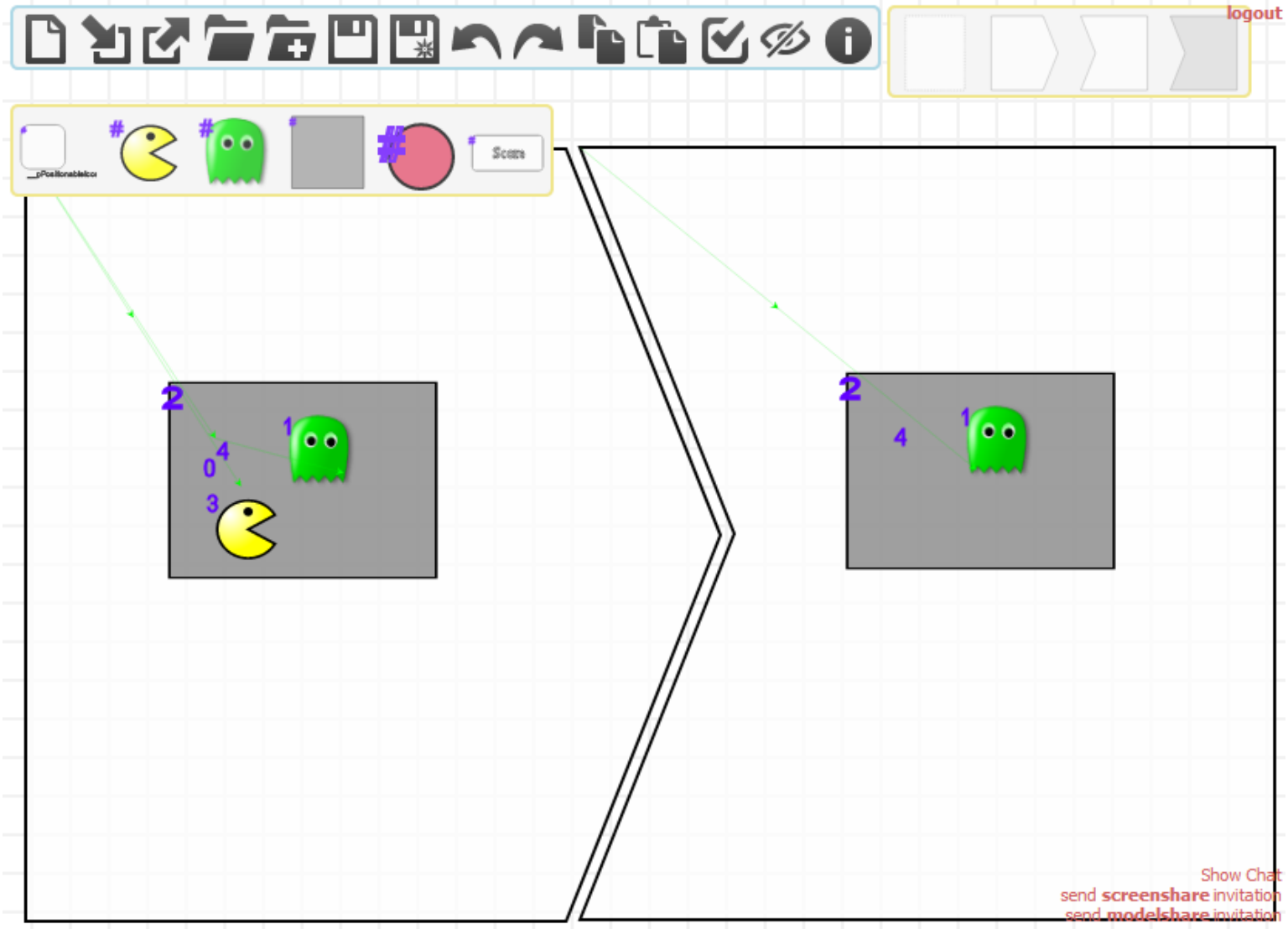
Graph transformation for simulation

- Models are considered as directed, typed, attributed graphs
- Transformations on such graphs are considered as graph rewritings
- Features:
  - Declarative paradigm
  - Rules defined as pre- and post-conditions
- Tools: **MoTif**, Henshin, GReAT

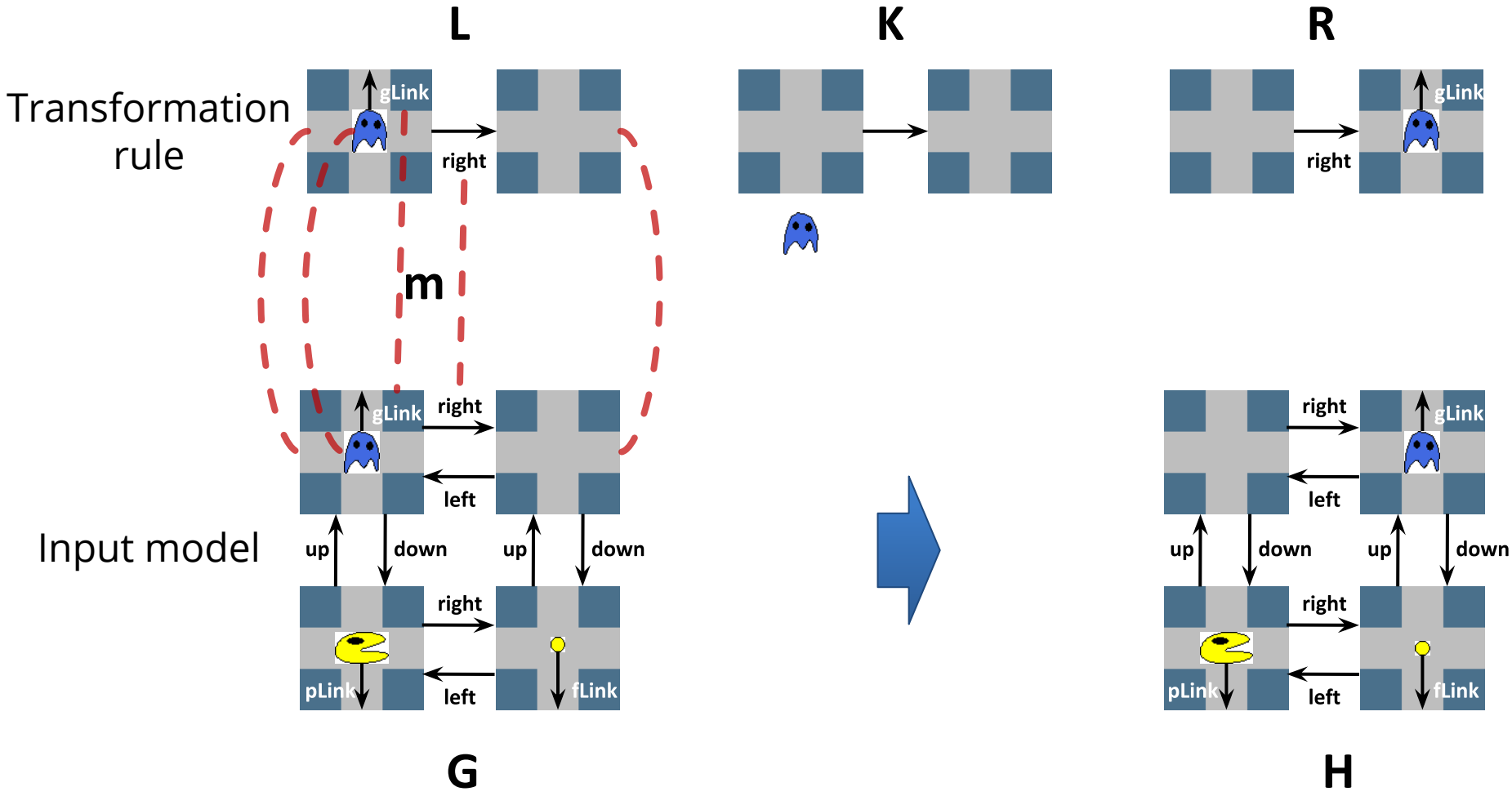
Metamodel of Pacman



Graph transformation rule

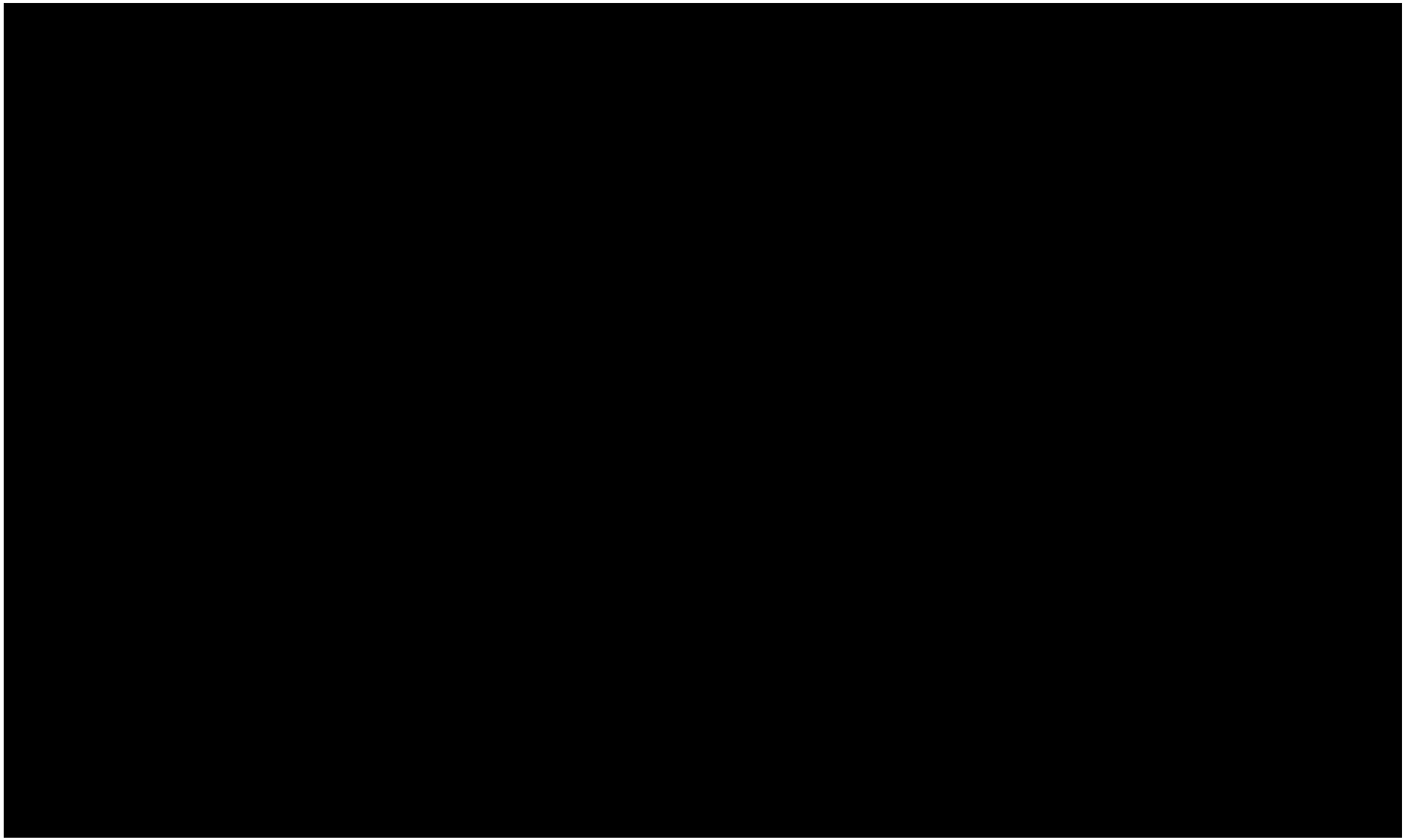


Rule-based graph transformation

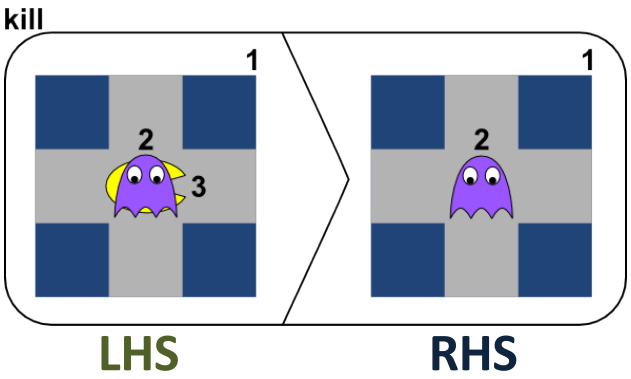


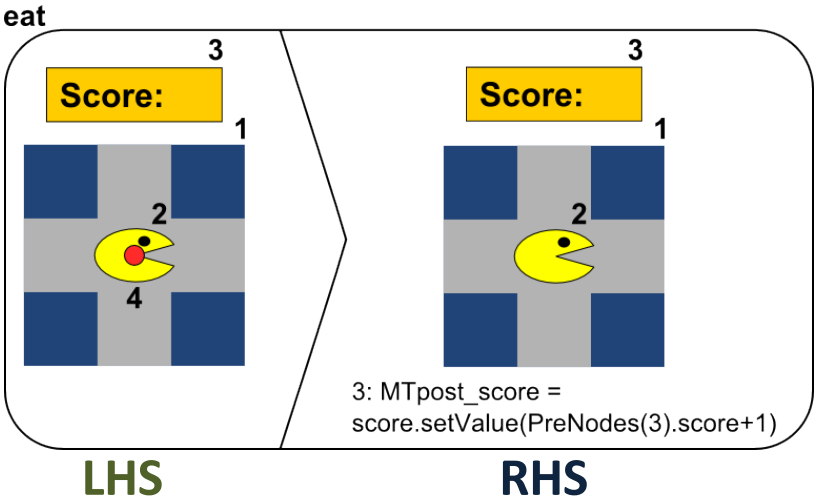
If there exists an occurrence of L in G then replace it with R

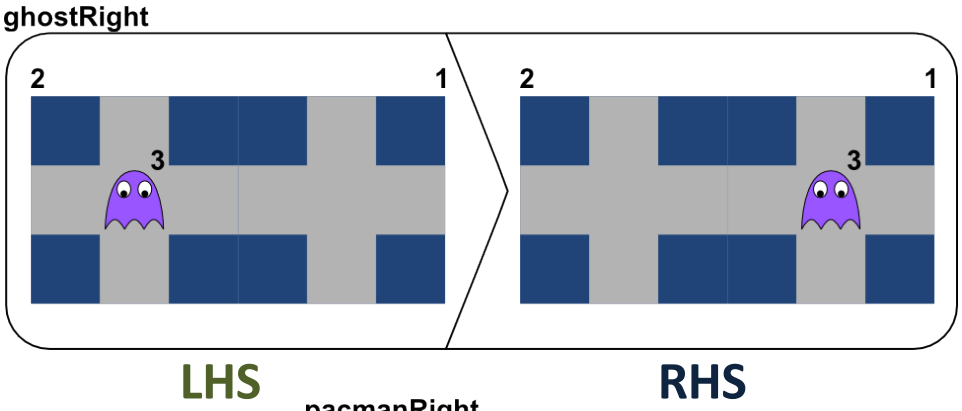
Mechanics of rule application

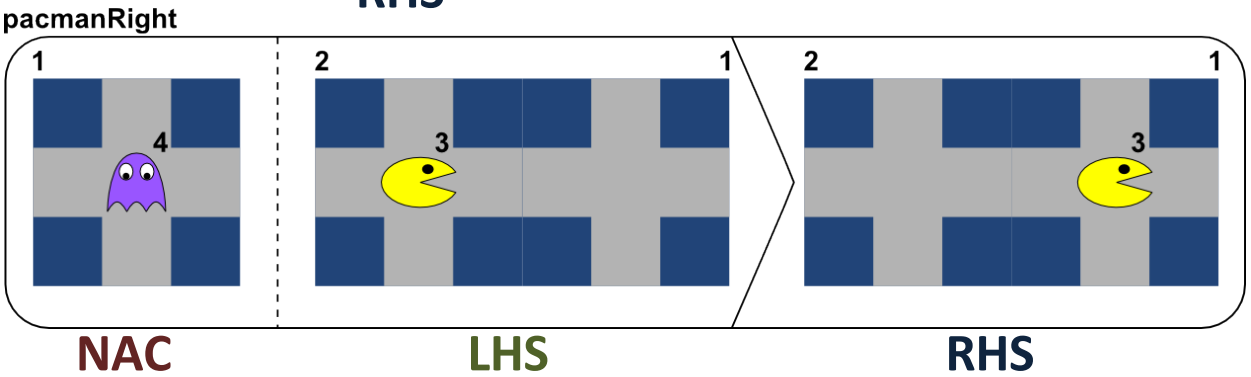


Operational semantics



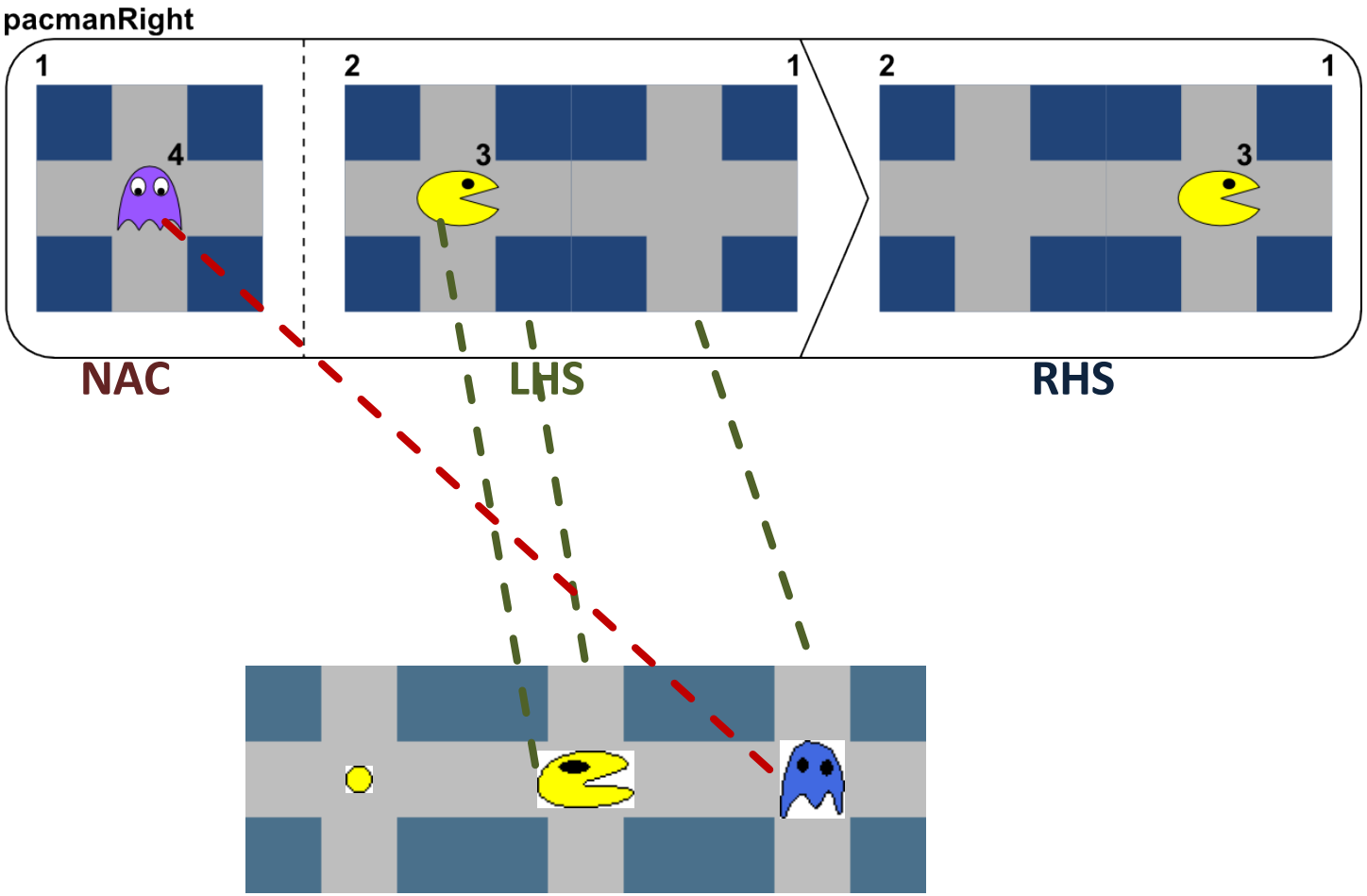






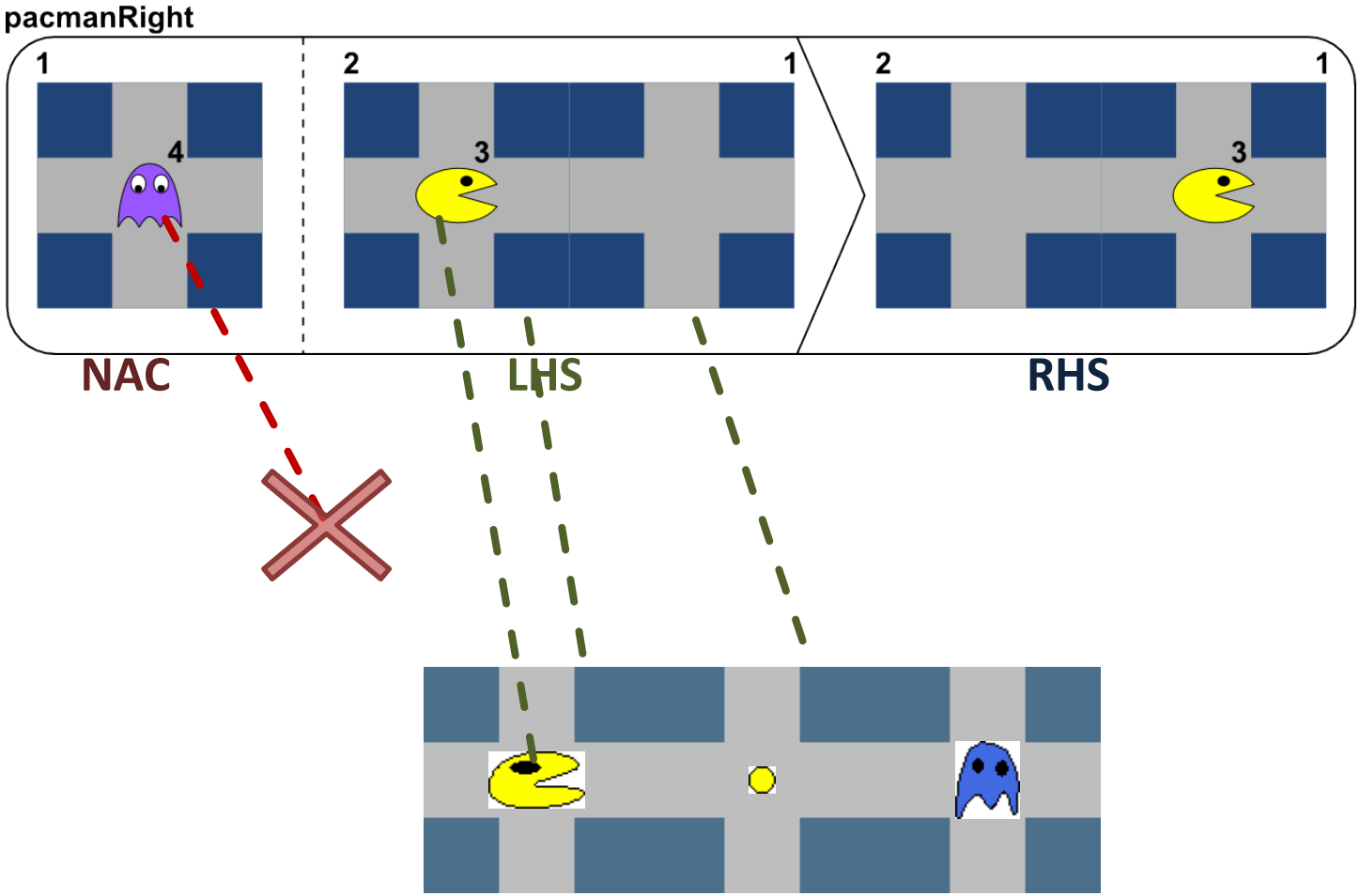
Negative application conditions

Non-applicable rule

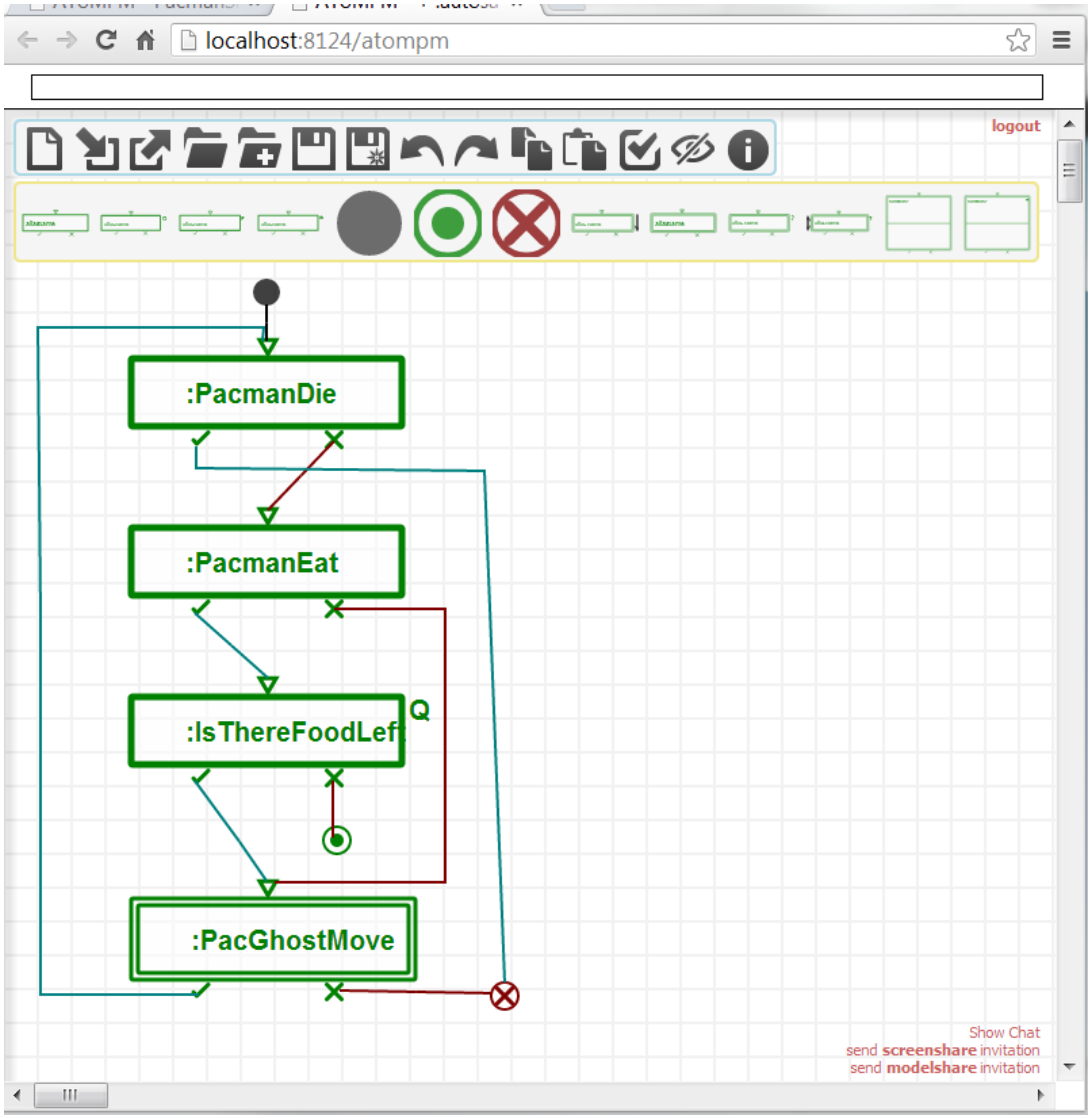


Negative application conditions

Applicable rule

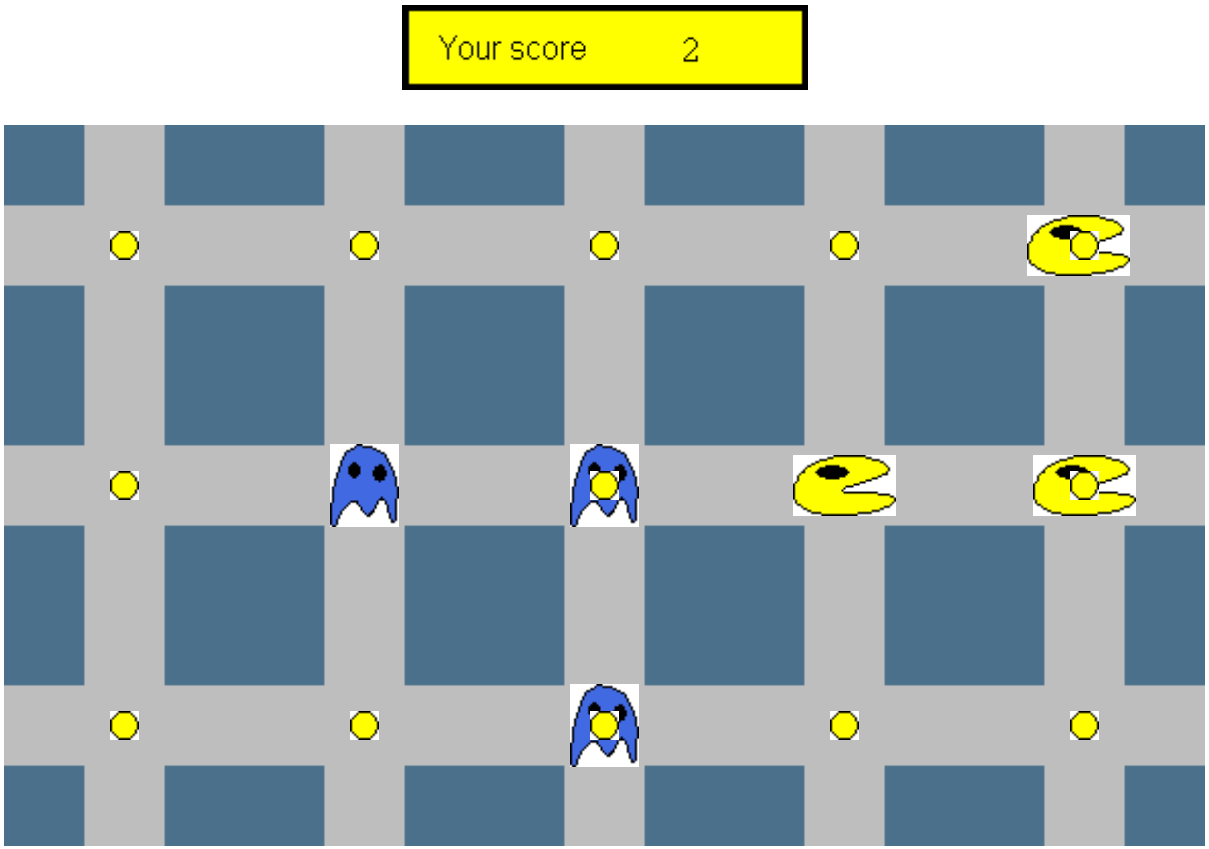


Scheduling of the rules



Simulation of a model

1. pacmanDie
2. pacmanEat
3. isThereFoodLeft
4. ghostMoveLeft
5. ghostMoveRight
6. ghostMoveUp
7. ghostMoveDown
8. pacmanMoveLeft
9. pacmanMoveRight
10. pacmanMoveUp
11. pacmanMoveDown



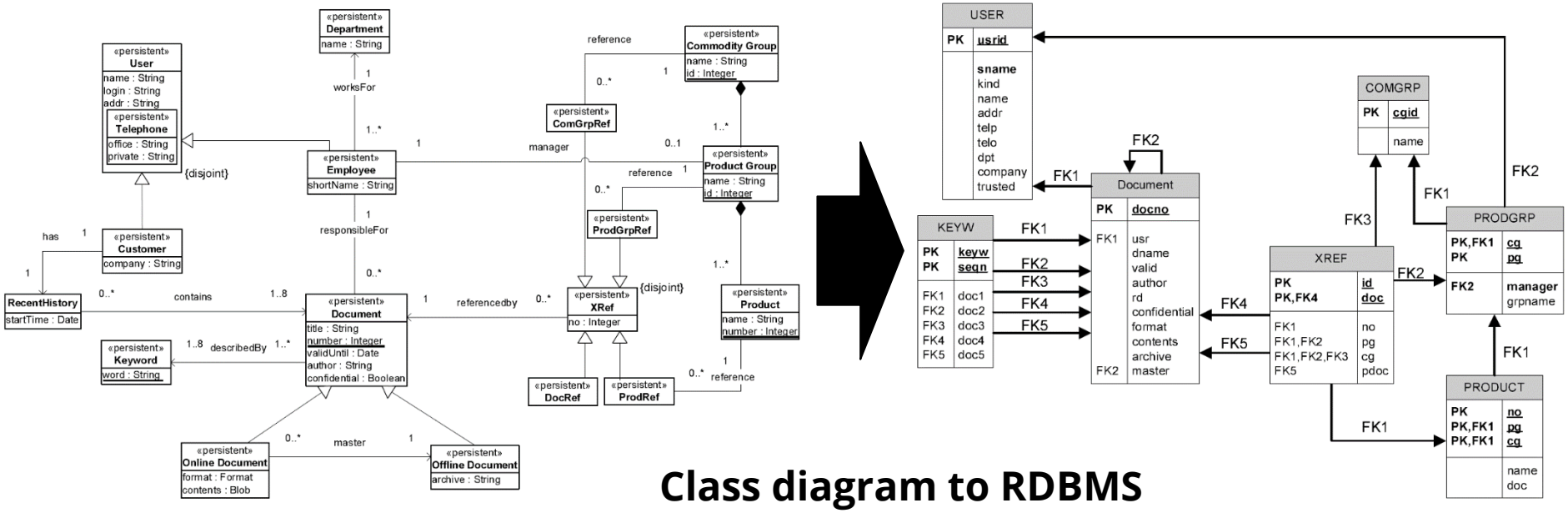
Language Translation Category

Groups intents that define a translation between two modeling languages.

- Translation
- *Migration*

Translation

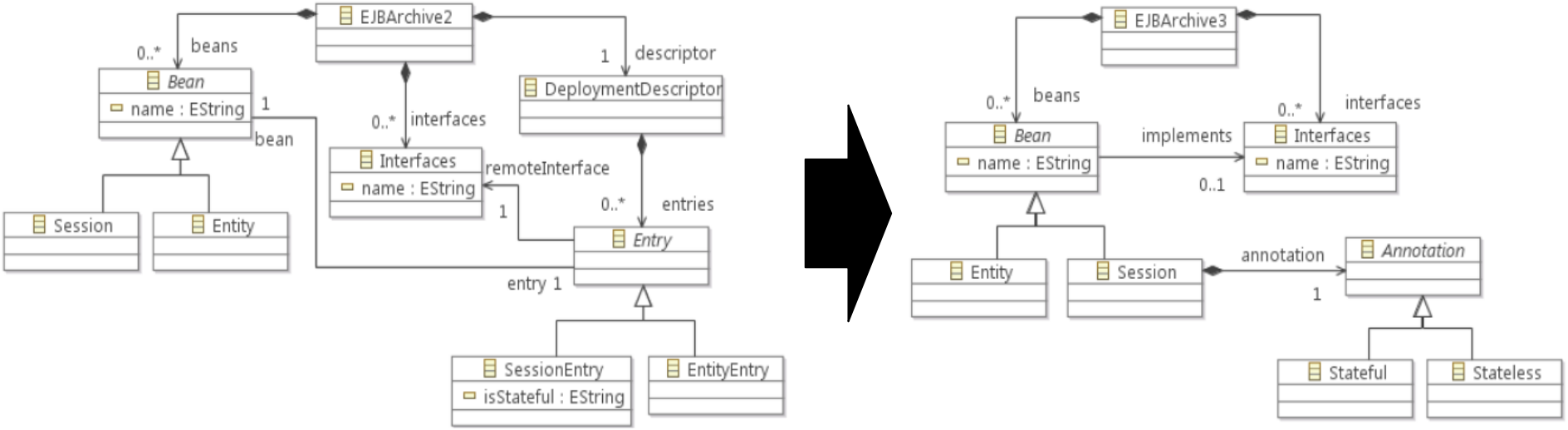
- Maps concepts of a model in a source language to concepts of another target language, while translating the semantics of the former in terms of the latter
- Similar to translational semantics, but the source language already has a semantics



Class diagram to RDBMS

Migration

- Transforms models written in one language into models written in another language or a modified version of it, while keeping the models at the same level of abstraction
- Evolution to new version



Enterprise Java Beans 2.0

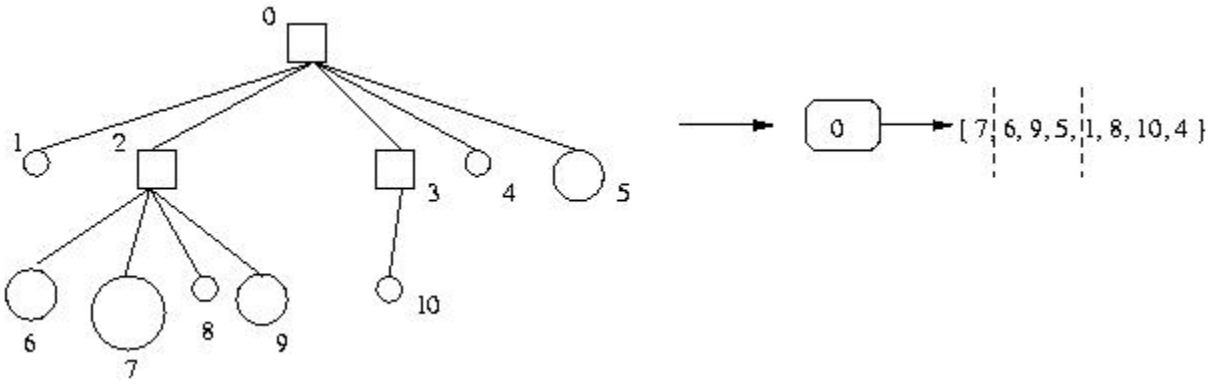
Enterprise Java Beans 3.0

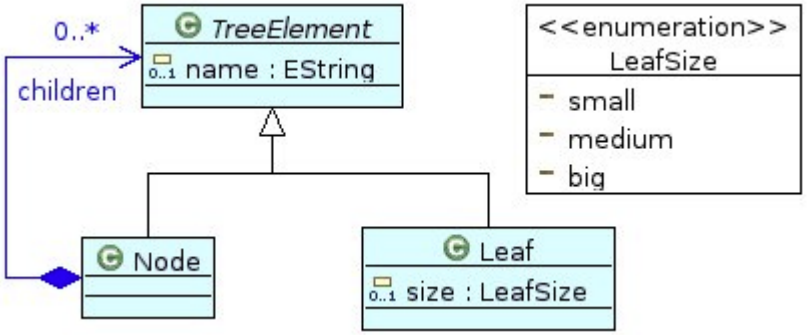
Example: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4634773&tag=1

Model-to-model transformation for translation

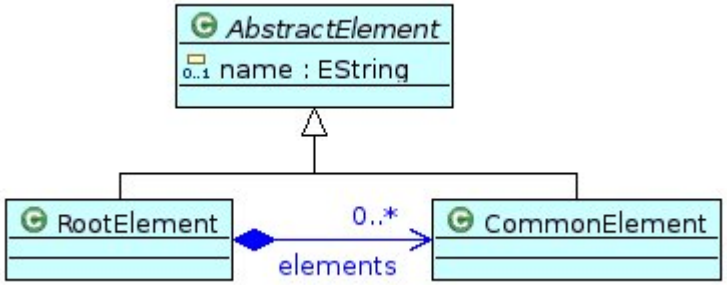
- Declarative paradigm
- Rules defined as non-destructing pre- and post-conditions
  - Source pattern to be matched in the source model
  - Target pattern to be created/updated in the target model for each match during rule application
- Typically models are represented in Ecore
- Input model is read-only
- Output model is write-only
- Tools: **ATL**, ETL, QVT-R

Tree to list example





Tree meta-model



List meta-model

Transformation in ATL: Root and leaves mapping

```
module Tree2List;
create elmList : MMElementList from aTree : MMTree;

rule TreeNodeRoot2RootElement {
    from    -- should be unique
        rt : MMTree!Node (rt.isTreeNodeRoot())
    to
        lstRt : MMElementList!RootElement (
            name <- rt.name,
            elements <- rt.getLeavesInOrder()
        )
}

rule Leaf2CommonElement {
    from
        s : MMTree!Leaf
    to
        t : MMElementList!CommonElement(
            name <- s.name
        )
}
```

Helper functions

```
helper context MMTree!Node def : isTreeNodeRoot() : Boolean =
    self.refImmediateComposite().oclIsUndefined();

helper context MMTree!Node def : getAllChildren () : OrderedSet(MMTree!TreeElement) =
    self.children->iterate( child ; elements : OrderedSet(MMTree!TreeElement) =
        OrderedSet{} |
        if child.oclIsTypeOf(MMTree!Node) then
            elements.union(child.getAllChildren()) -- NODE : recursive call
        else
            elements.append(child) -- LEAF
        endif
        );

helper context MMTree!Node def : getLeavesInOrder() : OrderedSet (MMTree!Leaf) =
    let leavesList : OrderedSet (MMTree!Leaf) =
        self.getAllChildren ()->select(currChild | currChild.oclIsTypeOf(MMTree!Leaf))
    in
        leavesList->select(leaf | leaf.size = #big)
        ->union(leavesList->select(leaf | leaf.size = #medium))
        ->union(leavesList->select(leaf | leaf.size = #small))
    ;
```

Execution semantics

1. Apply entry-point called rules, invoke other rules if specified
2. Evaluate standard rules guards
3. Create the target elements for every match with traceability links
4. ATL resolve algorithm evaluates all bindings: initializations
5. Apply lazy rules, execute action blocks
6. Apply end-point called rules

Execution of a declarative rule in ATL

1. Find all possible matches in the source model
2. Create elements specified in the target pattern on a target model
3. Initialize attributes and links of the newly created elements
4. Create traceability links from the elements in the source model matched by the source pattern to the created elements in the target model

There are 3 types of declarative rules:

5. Standard rules: applied once for each match
6. Lazy rules: applied as many times for each match as referred to by other rules
7. Unique lazy: same as lazy rules but re-use the target elements they created when applied multiple times

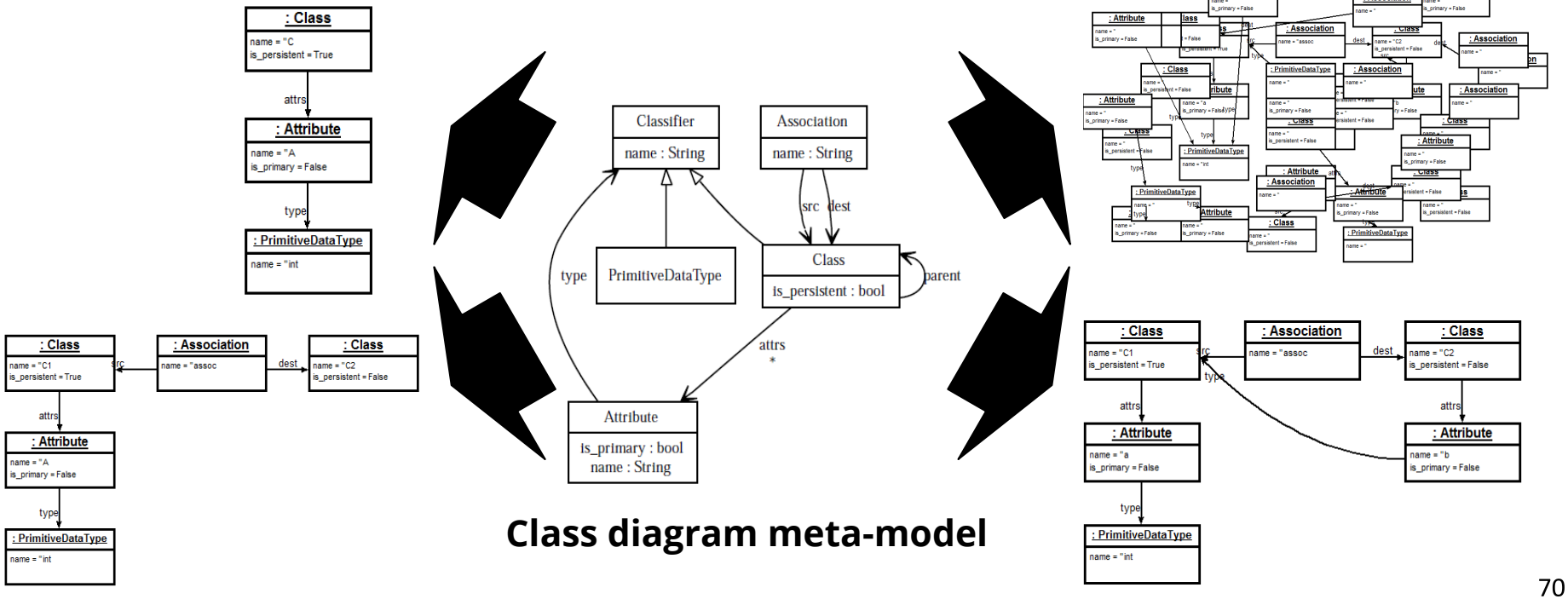
Constraint Satisfaction Category

Groups intents that output models given a set of constraints.

- *Model Generation*
- *Model Finding*

Model Generation

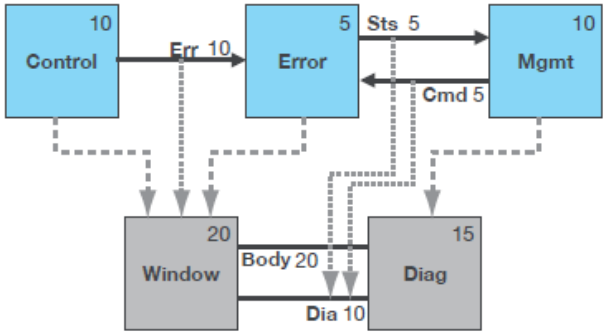
- Automatically produce possible correct instances of a metamodel
- Very useful for testing model transformations by generating input test models to verify the correctness of a transformation

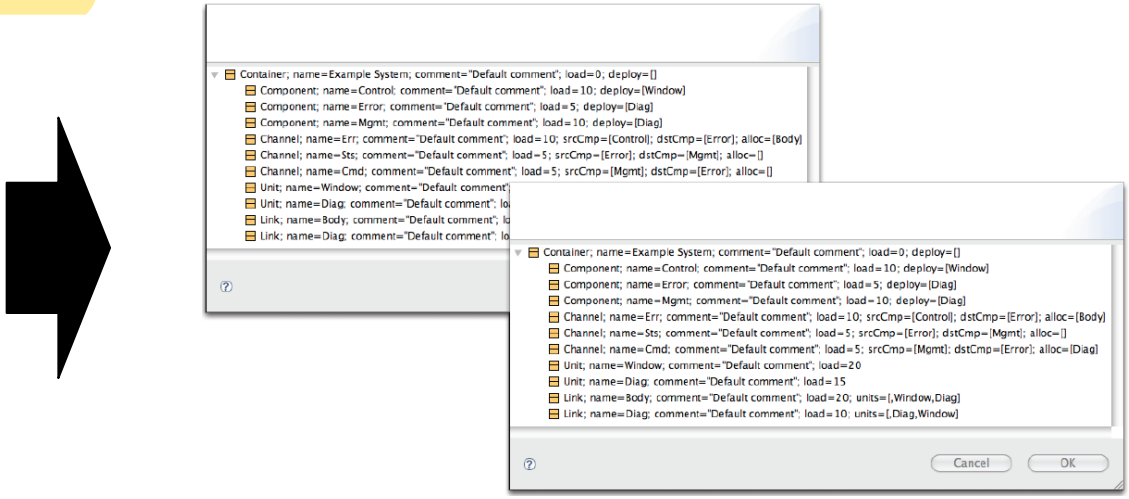


Class diagram meta-model

Model Finding

- Searches for models that satisfy given constraints
  - Several models are generated according to a set of rules and evaluated to check whether they satisfy some constraints
  - If not, a backtracking mechanism reverses some of the applied rules to find another model
- Typically used in design-space exploration to help decide which solution to choose

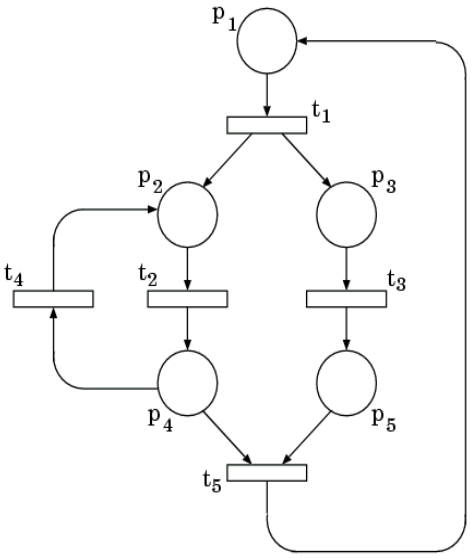


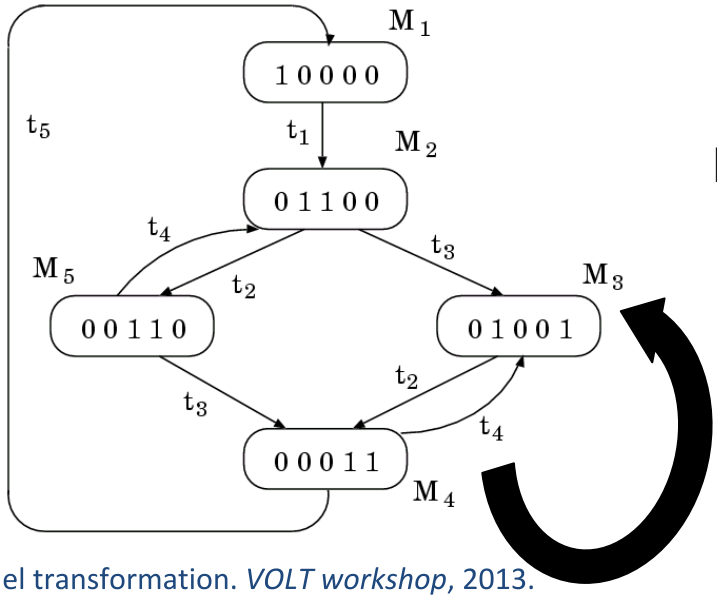


Example: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5457773

Analysis (Category)

- Model transformation that implements analysis algorithm of varying complexities
  - Dead code detection
  - Rule inapplicability detection
  - Model checking of temporal formulae over models





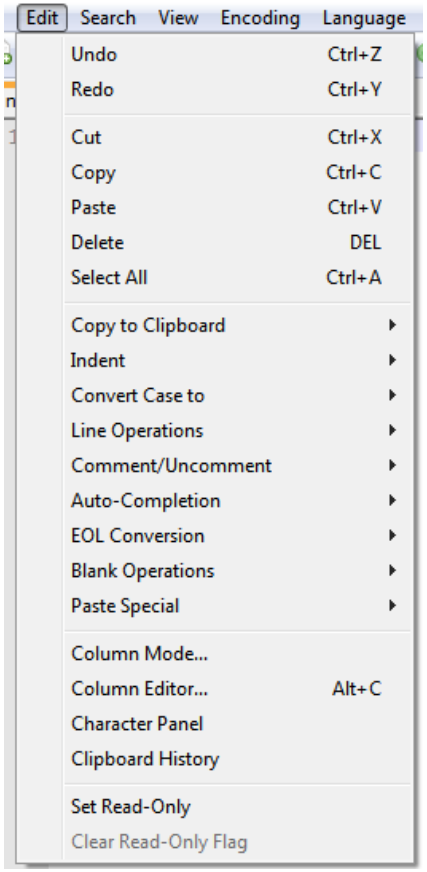
Reachability analysis of Petri nets

L. Lúcio and H. Vangheluwe. Model transformation to verify model transformation. *VOLT workshop*, 2013.

Editing Category

Groups intents that manipulate a model directly.

- Model Editing
- *Optimization*
- *Model Refactoring*
- *Normalization*
  - *Canonicalization*



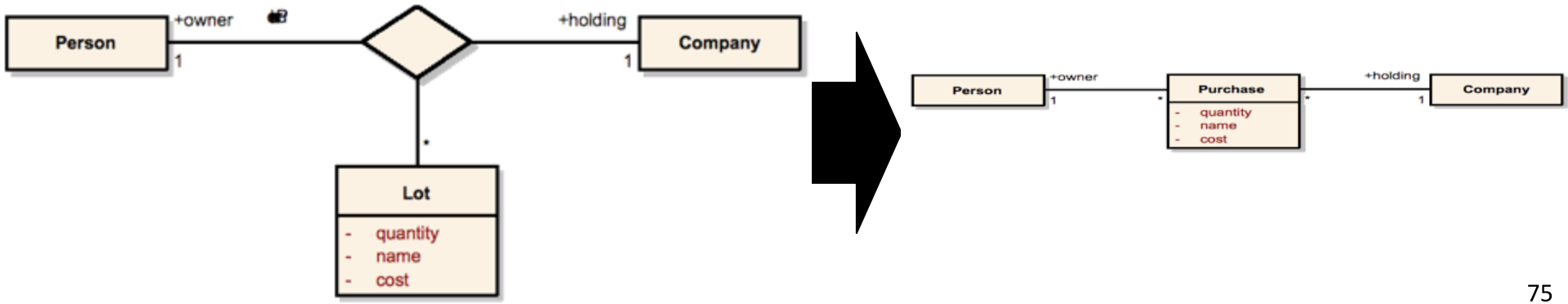
Model Editing

- Simple operations on a model (CRUD operations):
  - **Add** an element to the model;
  - **Remove** an element from the model;
  - **Update** an element's properties;
  - **Access** an element or its properties;
  - **Navigate** through the elements.
- Atomic or bulk operations
- Considered a model transformation when the system is completely and explicitly modeled
- Tool support: any model editor that models CRUD operations (AToMPM)

Optimization

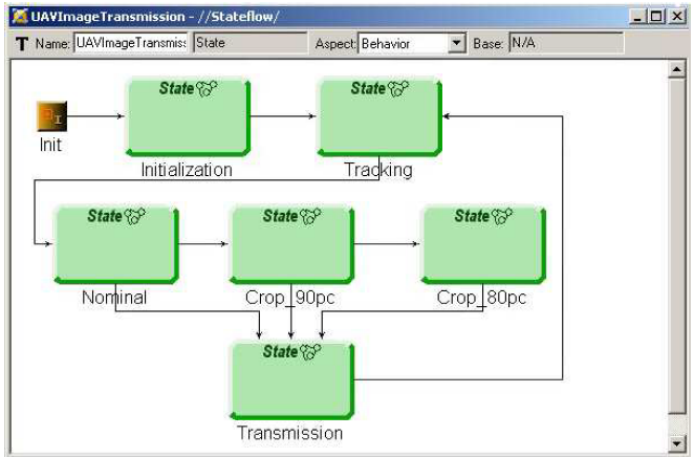
- Special kind of model edition
- Aims at improving certain operational qualities models
  - e.g., Scaleability, efficiency
- Example: Automatic application of design patterns on models

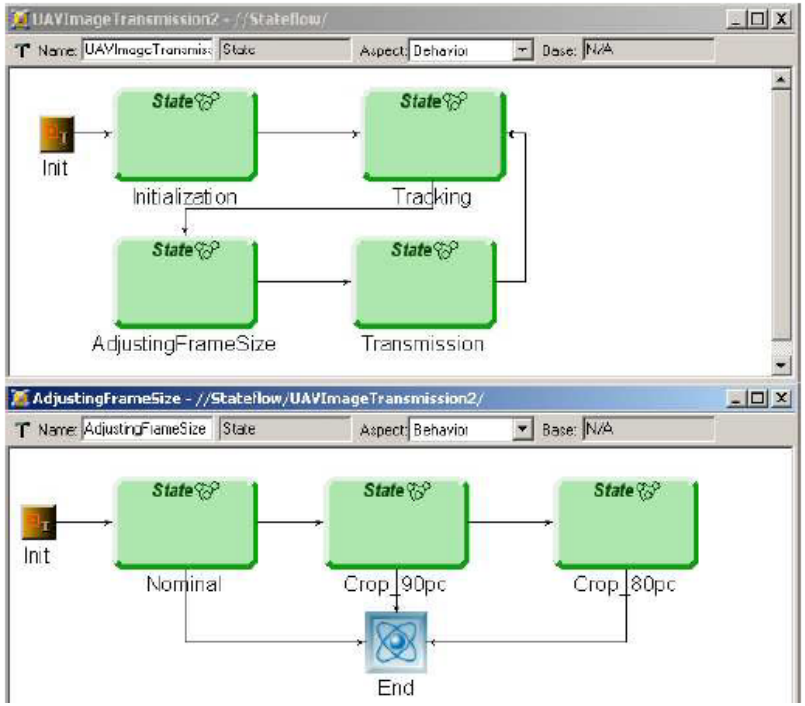
N-ary to binary association



Model Refactoring

- Special kind of model edition
- Restructure the model to improve certain internal quality characteristics without changing its observable behavior
  - Understandability, modifiability, reusability, modularity, adaptability



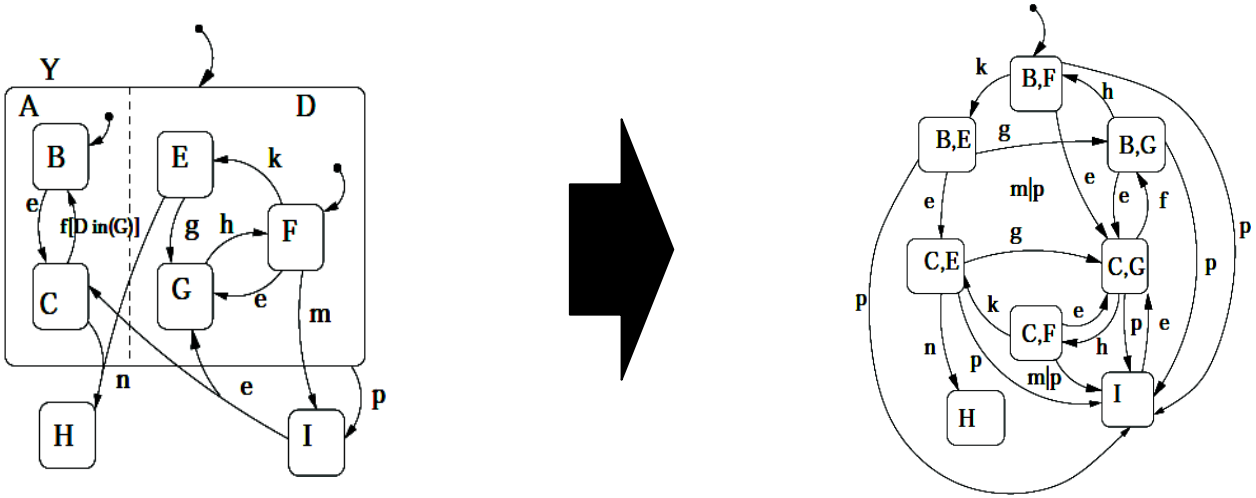


Compose states into composite states

Example: http://link.springer.com/chapter/10.1007%2F3-540-28554-7_9

Normalization

- Special kind of model edition to decrease syntactic complexity of models: simplification
- Translate complex language constructs into more primitive language constructs

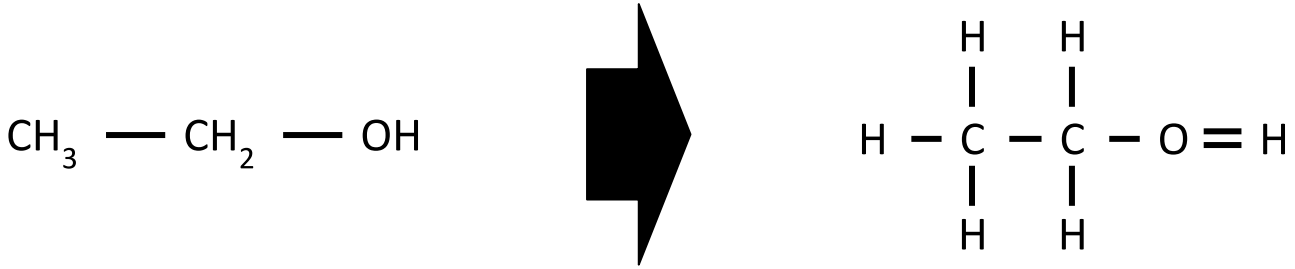


Flatten OR- and AND-states into states and transitions

Example: http://link.springer.com/article/10.1007%2Fs10270-006-0027-7

Canonicalization

- Special kind of normalization where models are normalized in a unique form
- Useful to compare equality of models



Planar semi-developed formula to planar developed formula of ethanol

Example: http://www.jot.fm/issues/issue_2007_10/paper10.pdf

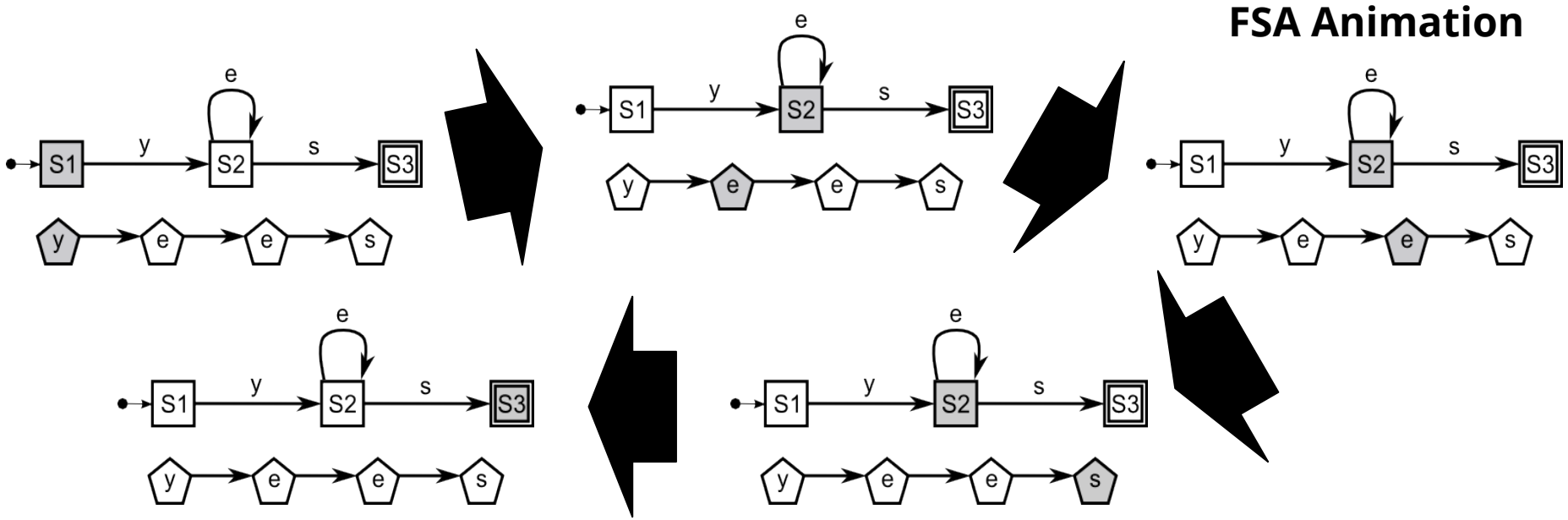
Model Visualization category

Groups intents that deal with the relation between the abstract and concrete syntax of a modeling a language.

- Animation
- *Rendering*
- *Parsing*

Animation

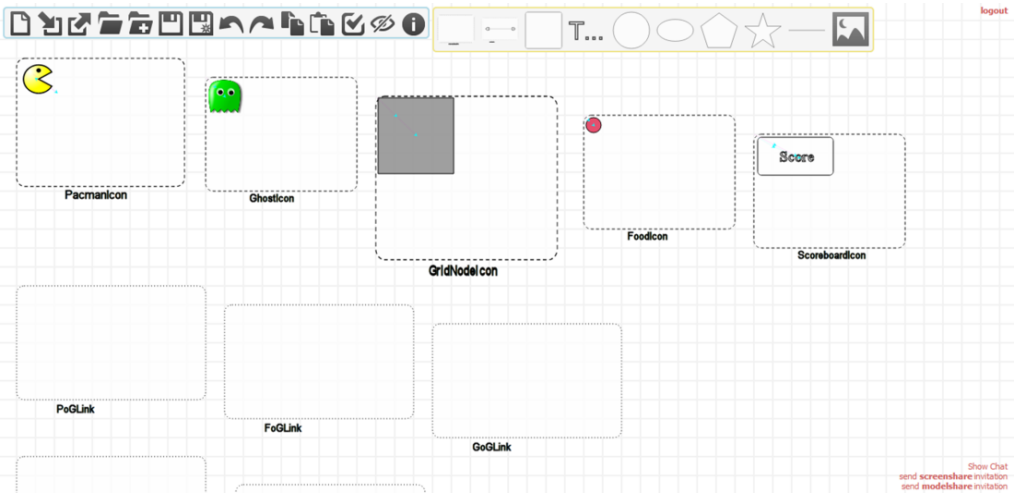
- Visualization of changes in the abstract syntax
  - e.g., from simulation
- Projects the behavior of a model on a specific animation view
- Operates on the concrete syntax of a model



C. Ermel and H. Erigh. Behavior-Preserving Simulation-to-Animation Model and Rule Transformations, *GT-VC workshop*, 2007.

Rendering

- Assigns one or more concrete representations (textual, graphical) to each abstract syntax element or group of elements
- Meta-model of concrete syntax must be explicitly defined
- Tool support: AToMPM

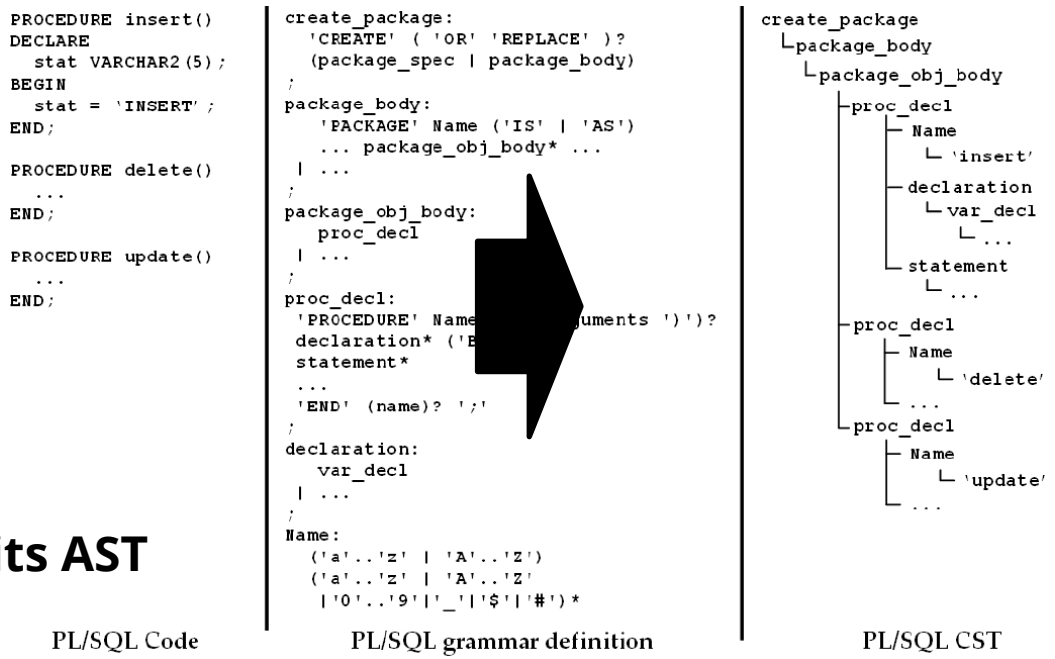


Concrete syntax assignment in AToMPM

E. Guerra and J. de Lara. Event-driven grammars: relating abstract and concrete levels of visual languages, *SoSym*: 6(3), 2007.

Parsing

- Inverse of rendering
- Maps concrete syntax of language back to its abstract syntax
- Requires meta-model of CS and meta-meta-model of language



PL/SQL code in CS to its AST

Example: http://link.springer.com/chapter/10.1007%2F978-3-642-02674-4_7

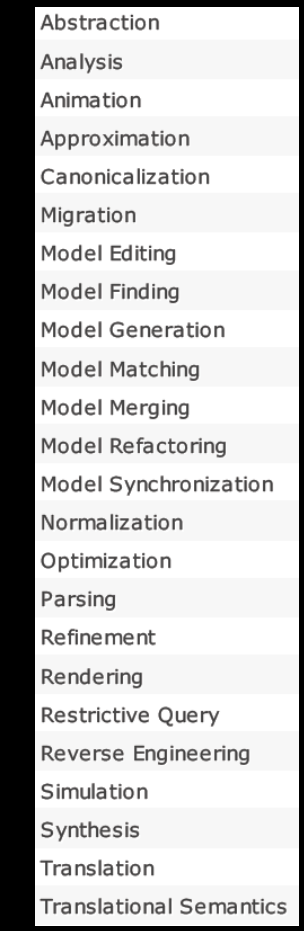
QUESTION

Map a custom DSML for stop watches into a Statecharts model in order to define its behavior.

Input: Watch DSM

Output: Statechart

- Translational Semantics



QUESTION

Define the actions performed by a traffic light to transition from one state to another.

Input: Traffic light model

Output: Traffic light model

- Simulation

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Visualize a Statecharts in SCXML into a graphical state machine model.

Input: XML

Output: State machine DSM

- Rendering

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

QUESTION

Move dots representing vehicles through the map of a city.

Input: City traffic DSM

Output: City traffic DSM

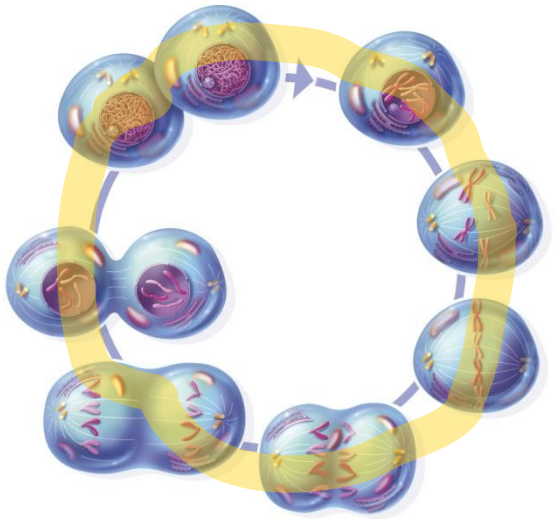
- Animation

- Abstraction
- Analysis
- Animation
- Approximation
- Canonicalization
- Migration
- Model Editing
- Model Finding
- Model Generation
- Model Matching
- Model Merging
- Model Refactoring
- Model Synchronization
- Normalization
- Optimization
- Parsing
- Refinement
- Rendering
- Restrictive Query
- Reverse Engineering
- Simulation
- Synthesis
- Translation
- Translational Semantics

Model Composition Category

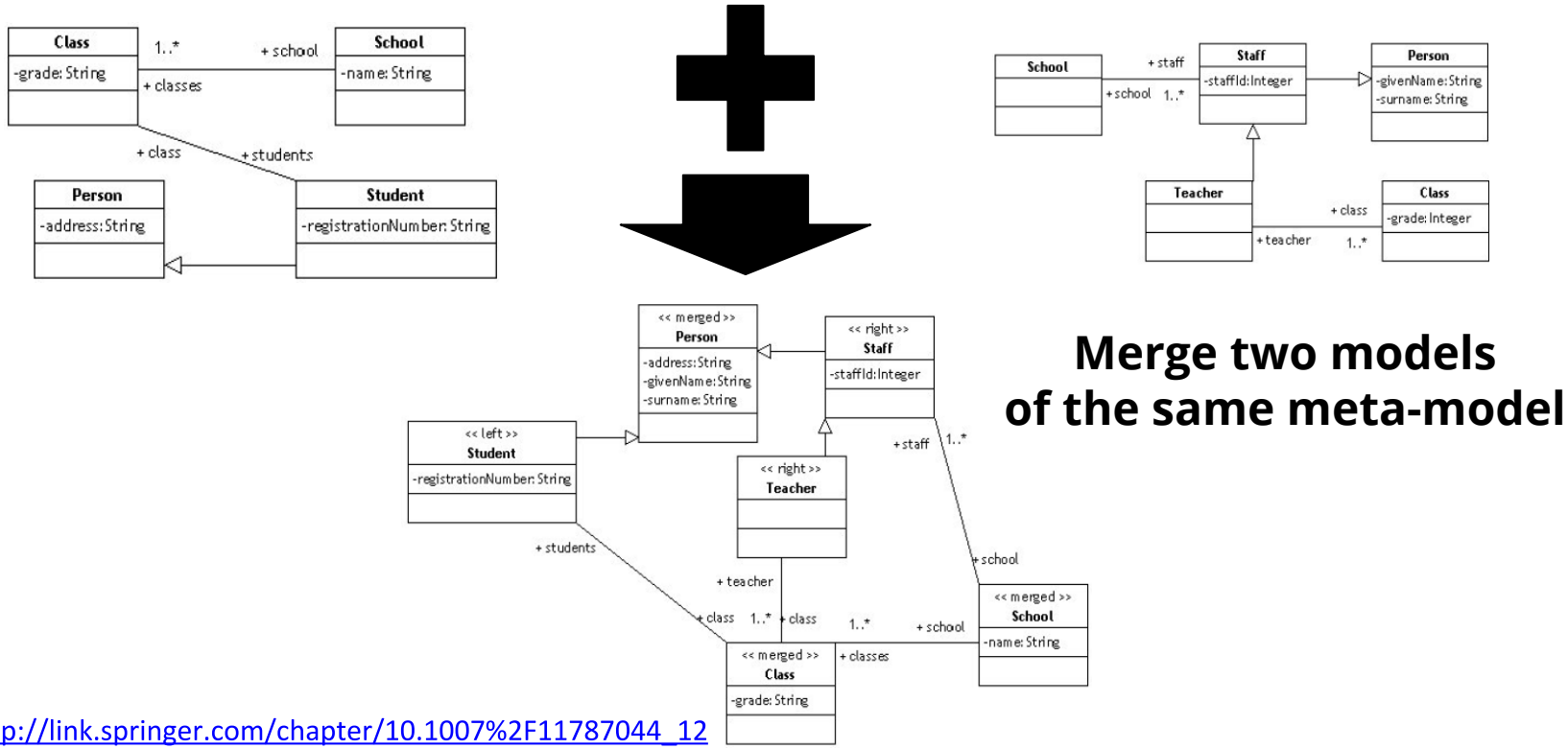
Groups intents that integrate models produced in isolation into a compound model, where each isolated model represents a concern that may overlap with any of the other models.

- *Model Merging*
- *Model Matching*
- Model Synchronization



Model Merging

- Instance of model composition
- Create a new model such that every element from the union of the input models is present exactly once in the merged model

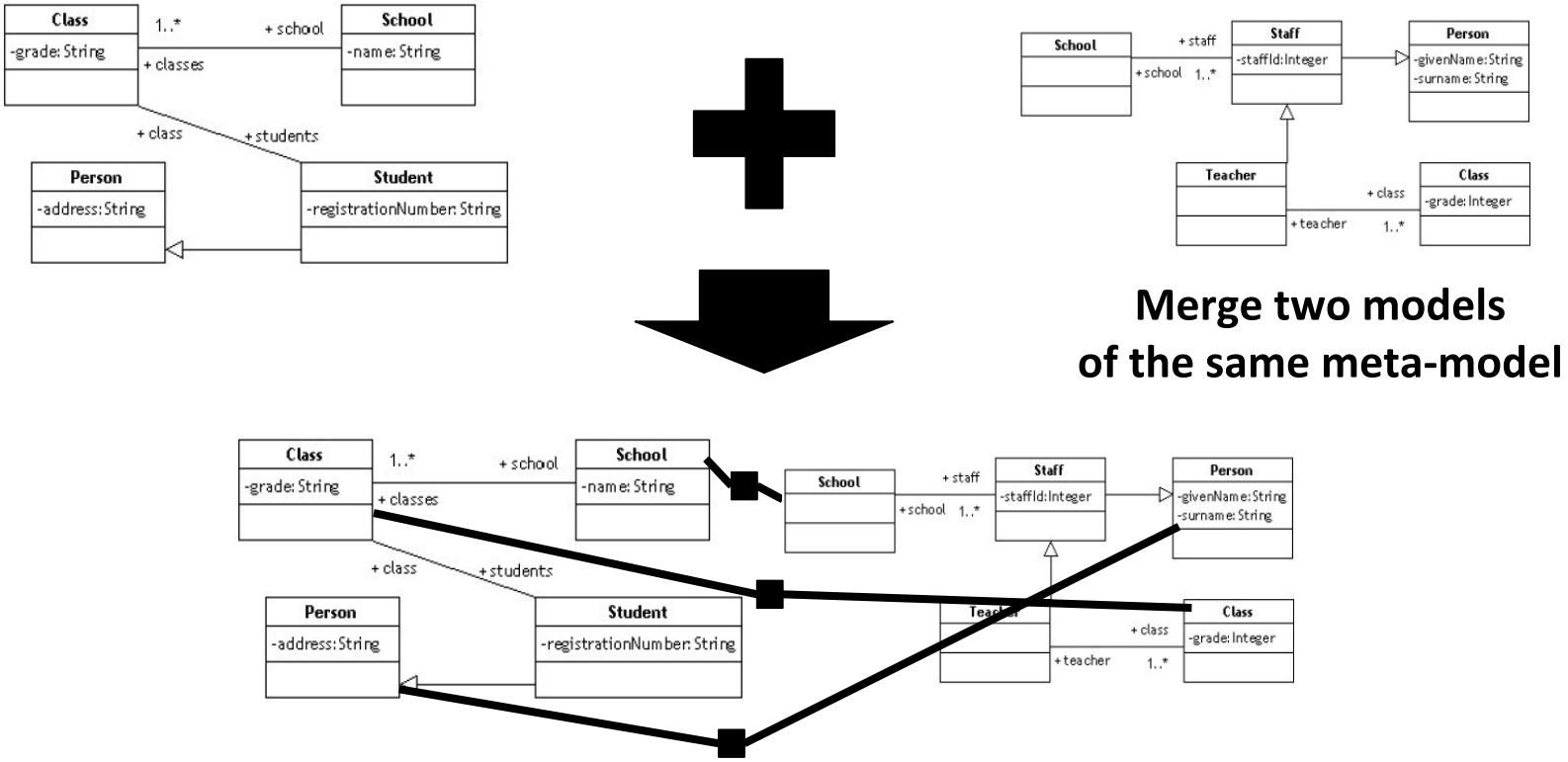


Merge two models of the same meta-model

Example: http://link.springer.com/chapter/10.1007%2F11787044_12

Model Matching

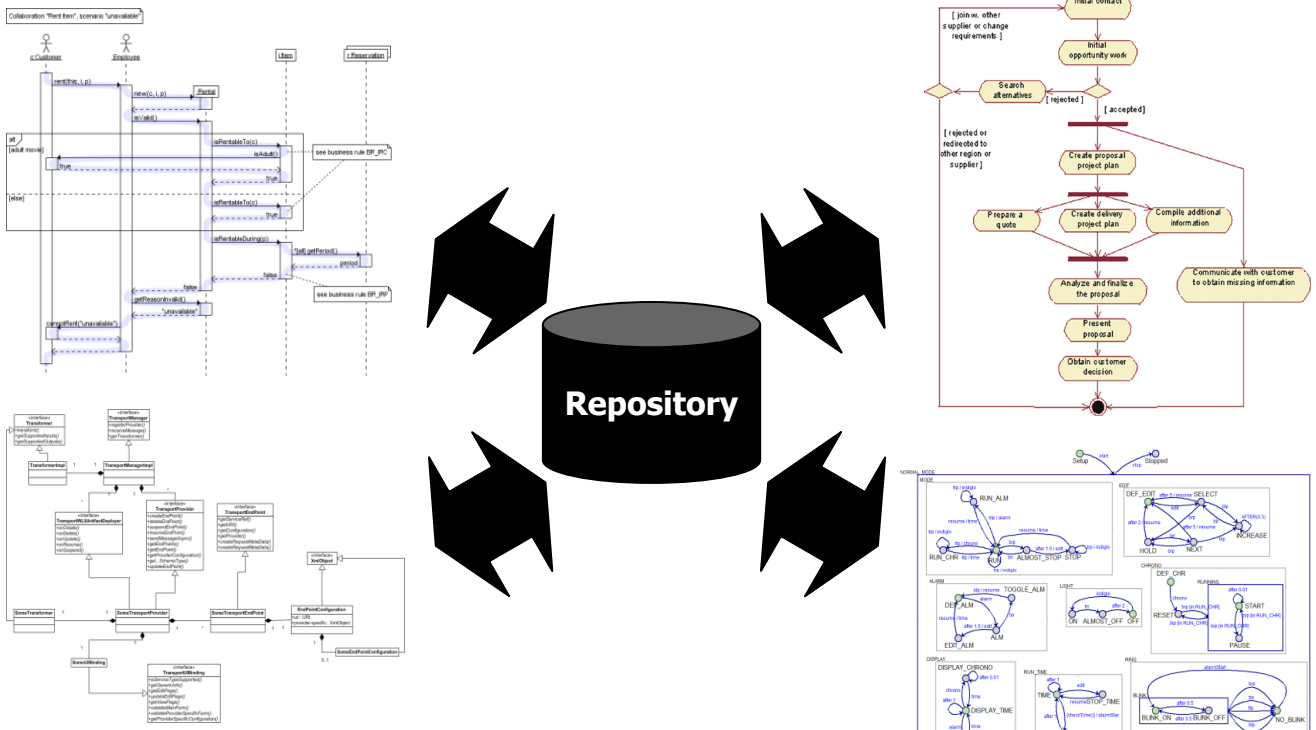
- Creates correspondence links between corresponding entities
  - Model weaving



Example: http://link.springer.com/article/10.1007%2Fs10270-008-0094-z

Model Synchronization

- Integrates models that evolved in isolation and subject to global consistency constraints
- Change propagation to the integrated models
  - Multiple views of a common repository



E. Guerra and J. de Lara. Model View Management with Triple Graph Transformation Systems, *ICGT*, 2006.

Vocabulary

- Relationship between source & target meta-models
  - **Endogenous**: Source meta-model = Target meta-model
  - **Exogenous**: Source meta-model ≠ Target meta-model
- Relationship between source & target models
  - **In-place**: Transformation executed within the same model
  - **Out-place**: Transformation produces a different model

	Endogenous	Exogenous	Either
In-place	Simulation, Editing, Animation, Model Finding, Analysis	X	X
Out-place	Restrictive query, Approximation	Synthesis, Serialization, Reverse eng, Translation, Migration, Rendering, Parsing, Matching, Synchronization, Model generation	Refinement, Abstraction, Merging
Either	Optimization, Refactoring, Normalization	X	X

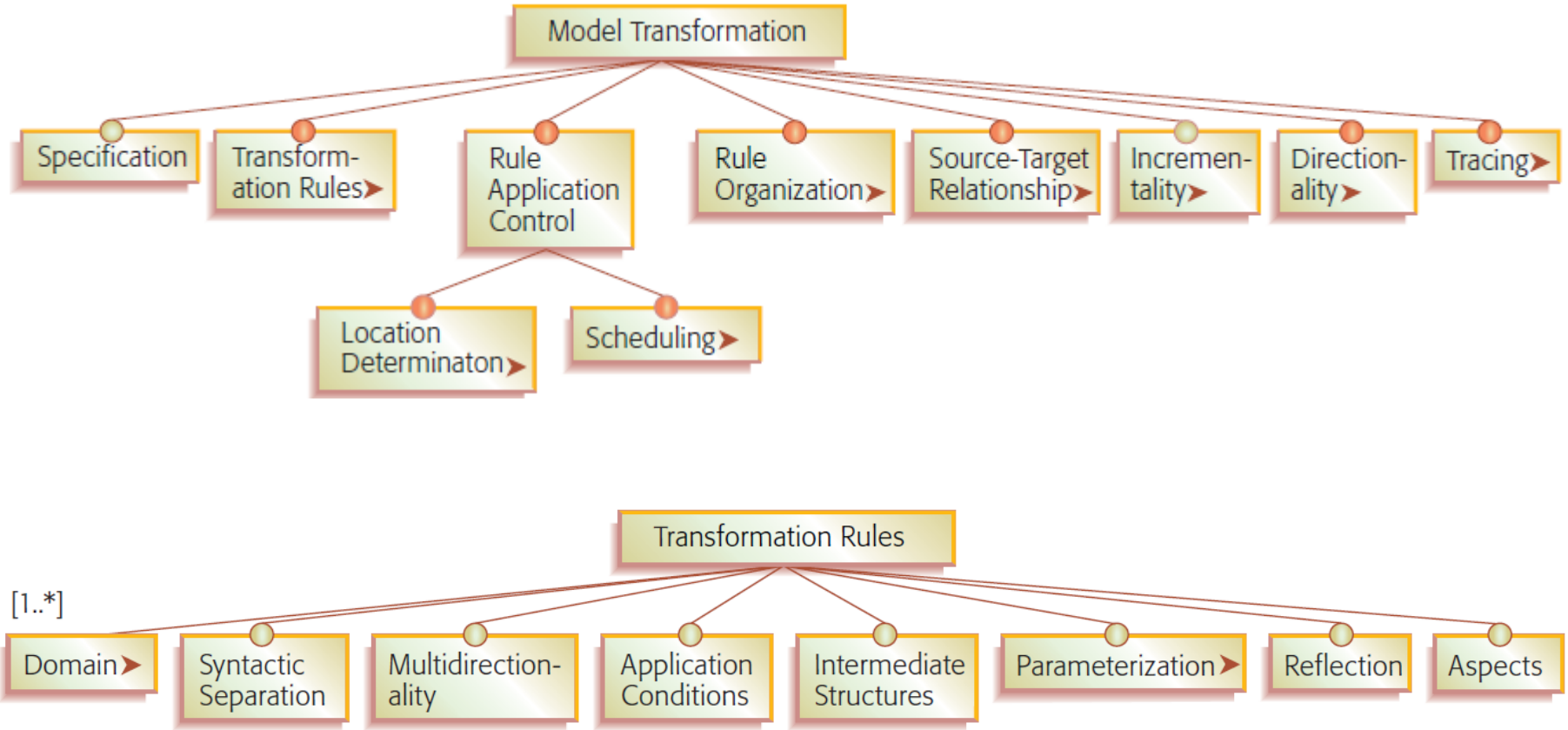
Vocabulary

- **Horizontal**: source and target models reside at the same abstraction level
- **Vertical**: source and target models reside at different abstraction levels

	Endogenous	Exogenous	Either
Horizontal	Simulation, Editing, Animation, Model Finding	Migration, Matching	Merging
Vertical	Restrictive query, Approximation, Analysis, Optimization, Refactoring, Normalization	Synthesis, Serialization, Reverse eng, Translation, Rendering, Parsing, Synchronization, Model generation	Refinement, Abstraction

Model transformation language features

Feature-Based Survey of Model Transformation Approaches

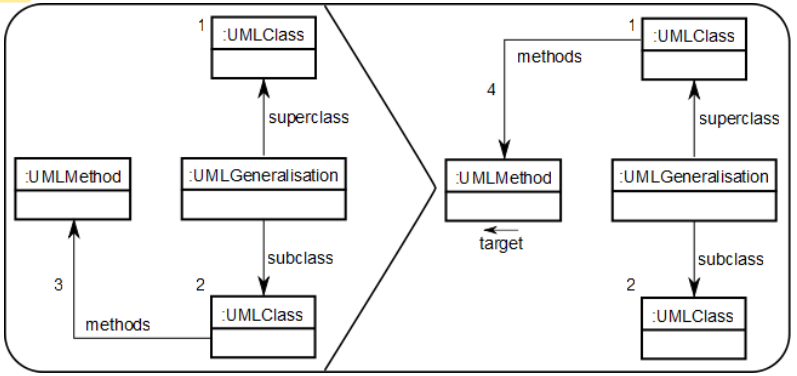


K. Czarnecki and S. Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal, special issue on Model-Driven Software Development*: 45(3), pp. 621-645, 2006.

Rule patterns

- Model fragments
- Using abstract or concrete syntax
- Syntactic separation

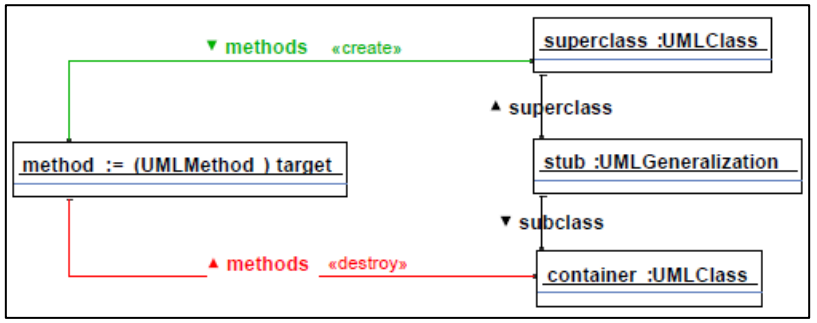
MoTif rule



ATL rule

```
module Person2Contact;
create OUT: MMb from IN: MMa {

rule Start {
     form p: MMa!Person(
          p.function = 'Boss'
     )
     to c: MMb!Contact(
          name <- p.first_name + p.
last_name)
}
```

FUJABA compact notation



Rule constraints

	Imperative	Declarative
Executable	Java API for MOF models	OCL query
Non-executable	X	QVT-R relation

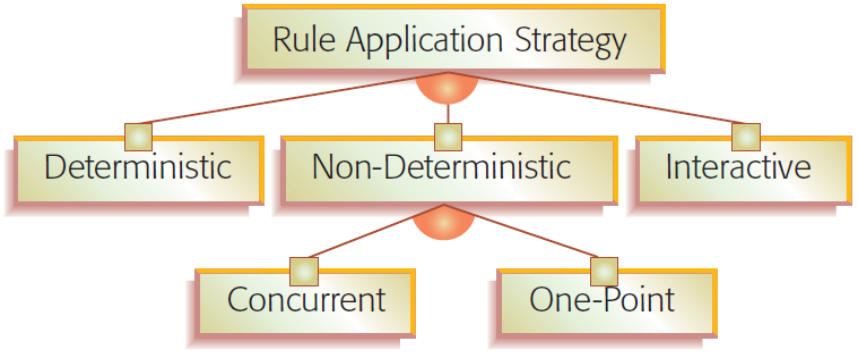
QVT-Relations rule

```
top relation PackageToSchema {
    domain uml p:Package{name=pn}
    domain rdbms s:Schema{name=pn}
}
```

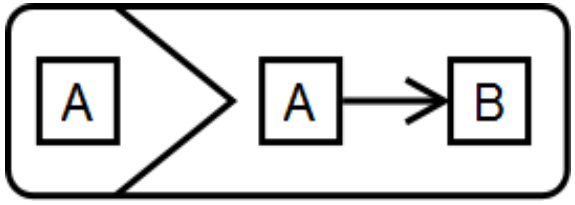
Kermeta operation

```
operation transform(source:PackageHierarchy): DataBase is
do
    result := DataBase.new
    trace.initStep("uml2db")
    source.hierarchy.each{ pkg |
        var scm: Schema init Schema.new
        scm.name := String.clone(pkg.name)
        result.schema.add(scm)
        trace.addlink("uml2db", "package2schema", pkg, scm)
    }
end
```

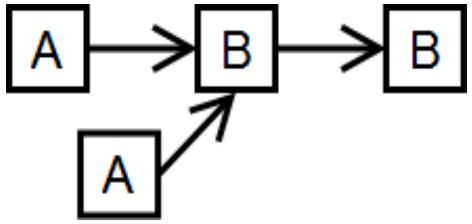
Rule application strategy



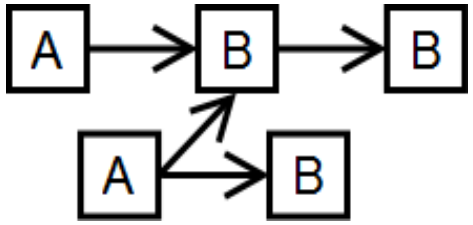
Rule

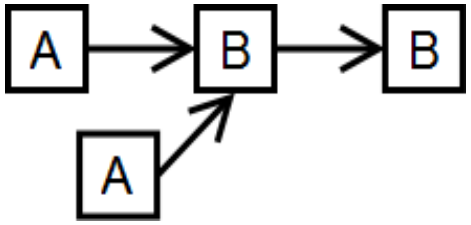


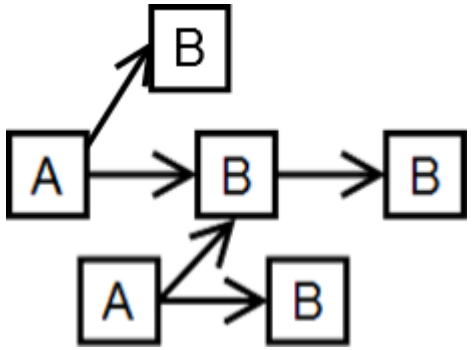
Input model



or on this A

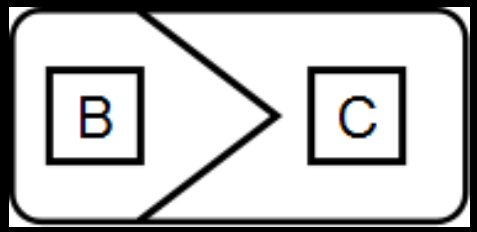




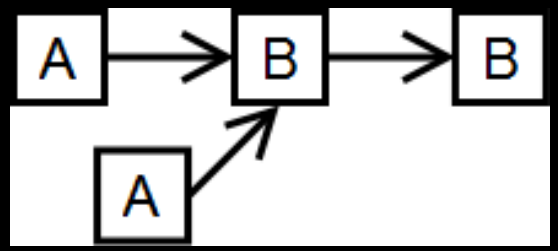


QUESTION

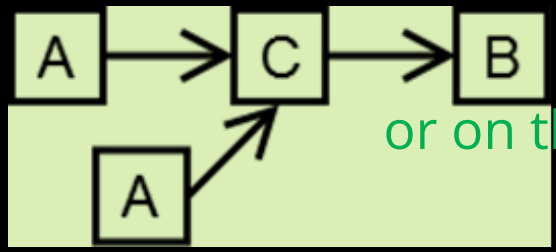
What are the possible outputs of the following rule applied to the following input model?



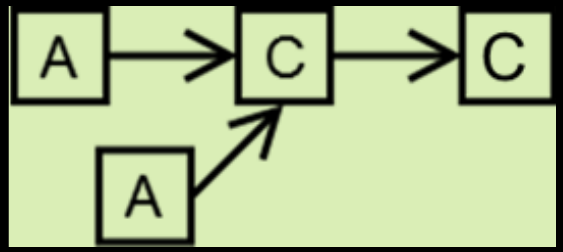
Rule



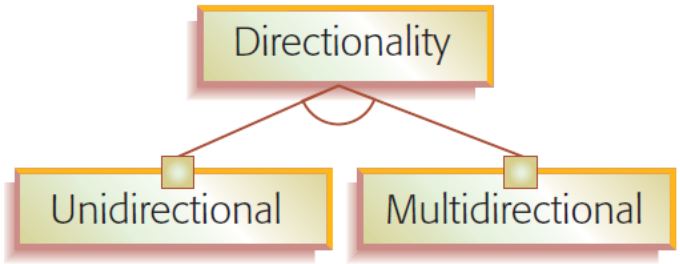
Input model



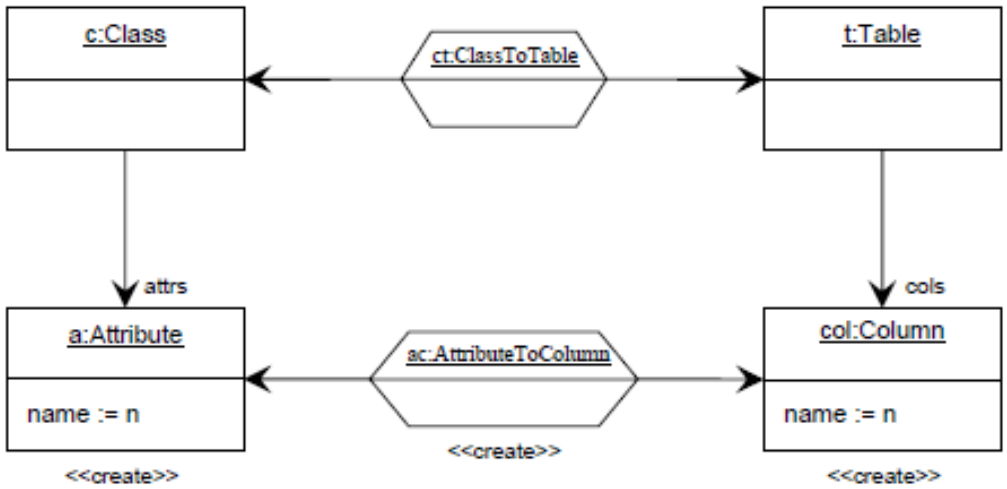
or on this B



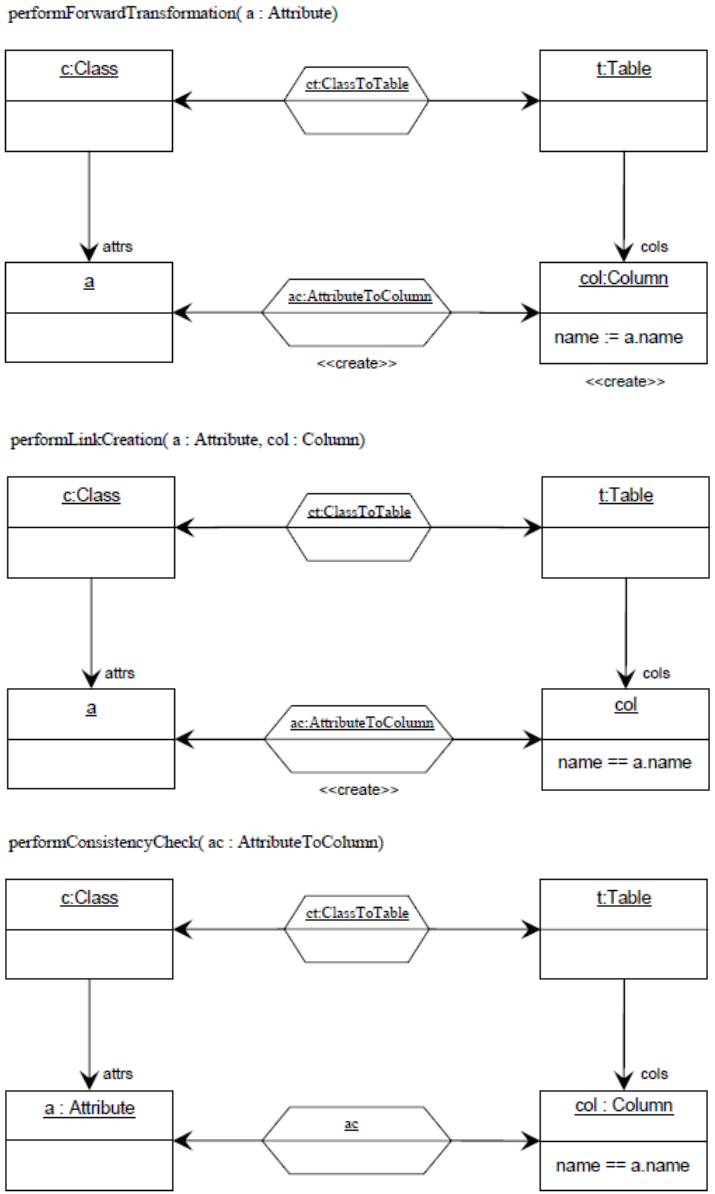
Multi-directional rules



TGG rule

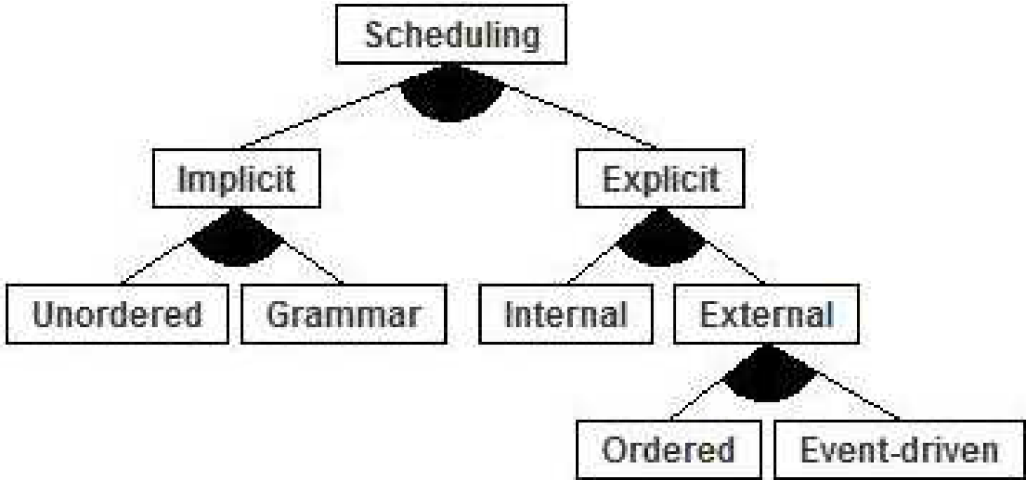


TGG operational rules

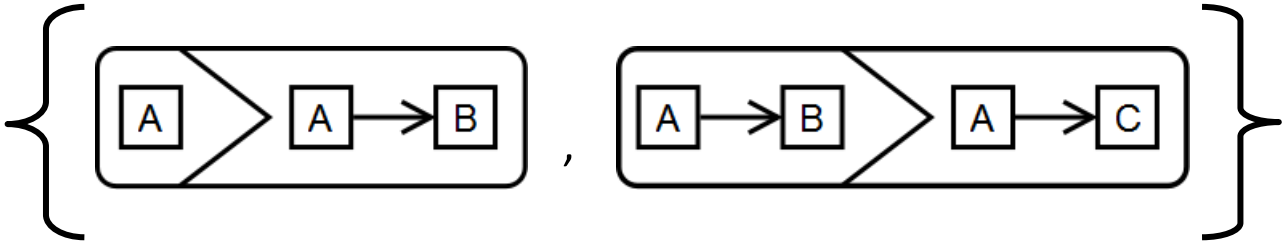


Rule scheduling strategies

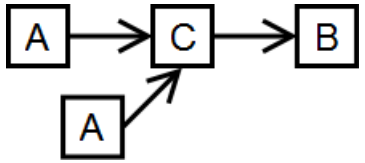
Implicit



Rule

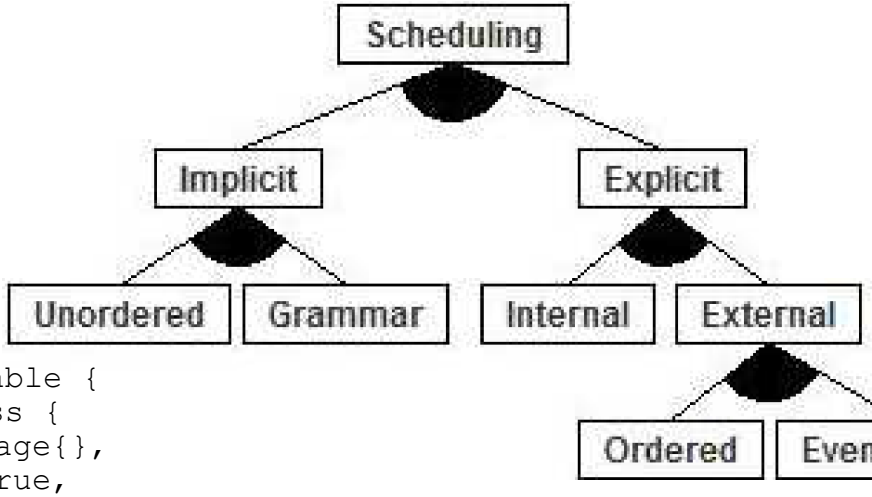


Input model

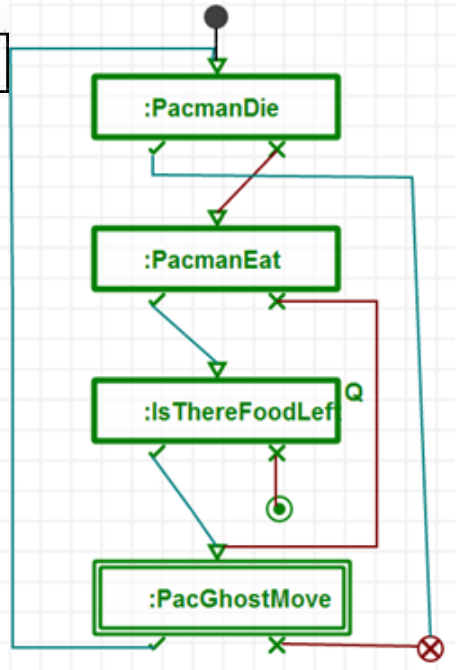


Rule scheduling strategies

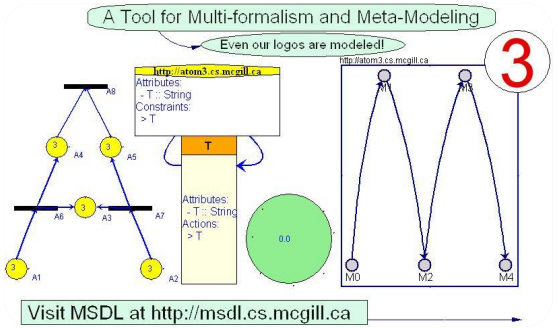
Explicit

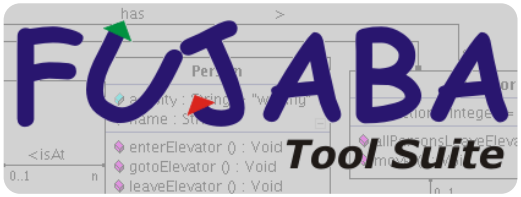


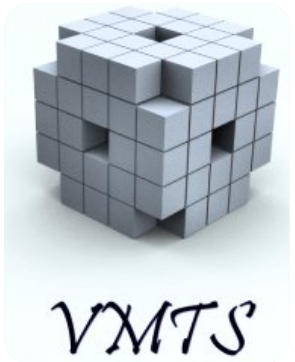

```
top relation ClassToTable {
     domain uml c:Class {
      package = p:Package{},
      isPersistent = true,
      name = cn
      }
     domain rdbms t:Table {
          schema = s:Schema{},
          name = cn,
          cols = cl:Column {
               name = cn +
'_tid',
               type = 'NUMBER'},
          pkey = cl
     }
     when {
          PackageToSchema (p,
s);
     }
     where {
          AttributeToColumn (c,
t);
```



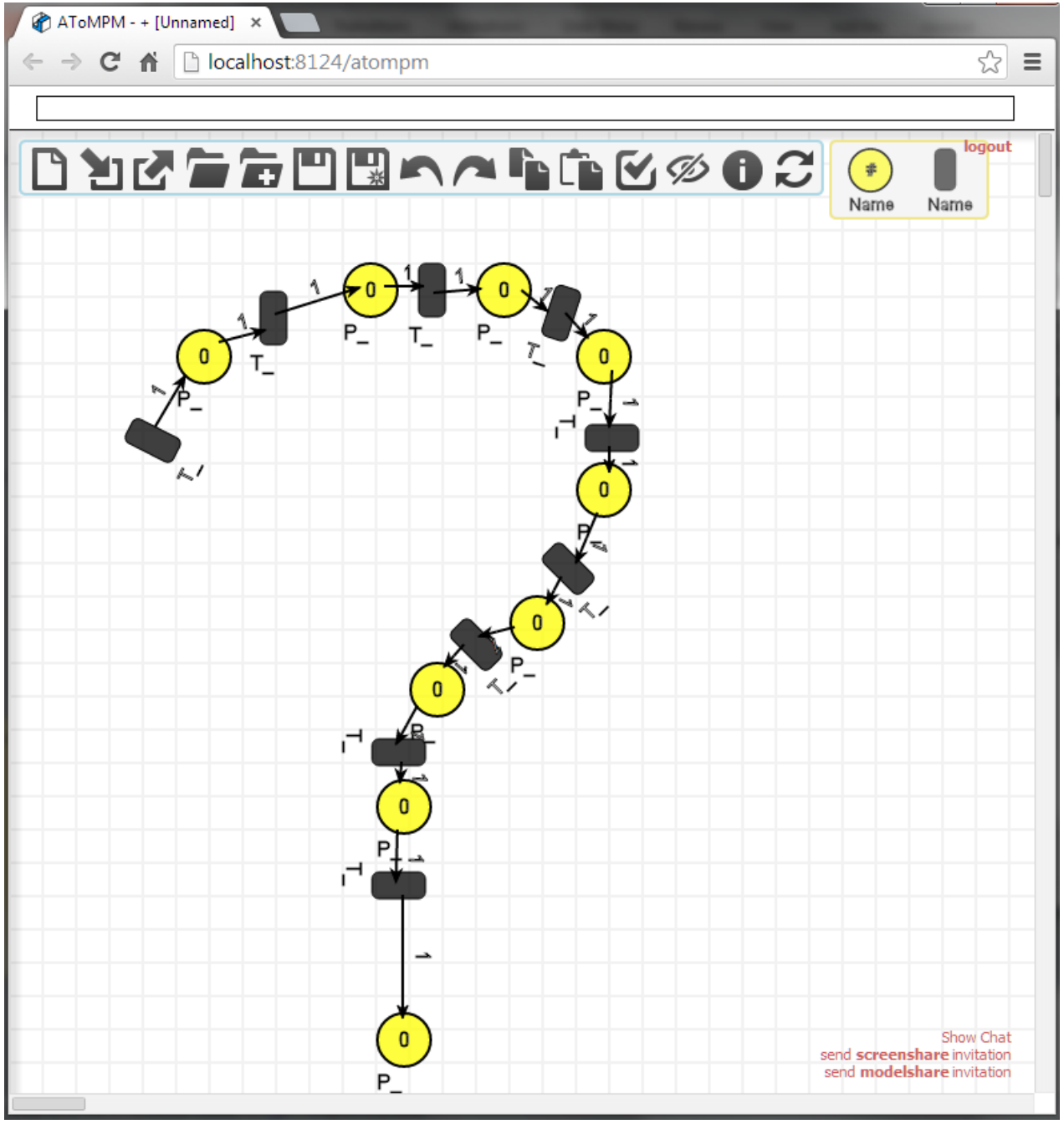
Plethora of model transformation languages







GReA **DSLTran** **ProGReS** **MOLA**


AToMPM - + [Unnamed]
localhost:8124/atompm
logout
Name
Name
Show Chat
send screenshare invitation
send modelshare invitation