1st Training Workshop, Wroclaw, 25-26 September 2006

VIATRA2: A Model Transformation Framework

Introduction & Tool Demonstration

Imre Kocsis

Balázs Polgár

Budapest University of Technology and Economics

Department of Measurement and Information Systems

Fault Tolerant Systems Research Group

DESEREC

Dependable Security by Enhanced Reconfigurability

Information Society
Technologies

Talk Outline

1. Introduction, motivation
2. VIATRA2
3. Qualitative Fault Modeling, Model Transformations & Resilient Systems
4. Tool demonstration

Introduction

Introduction

What are 'model transformations'?

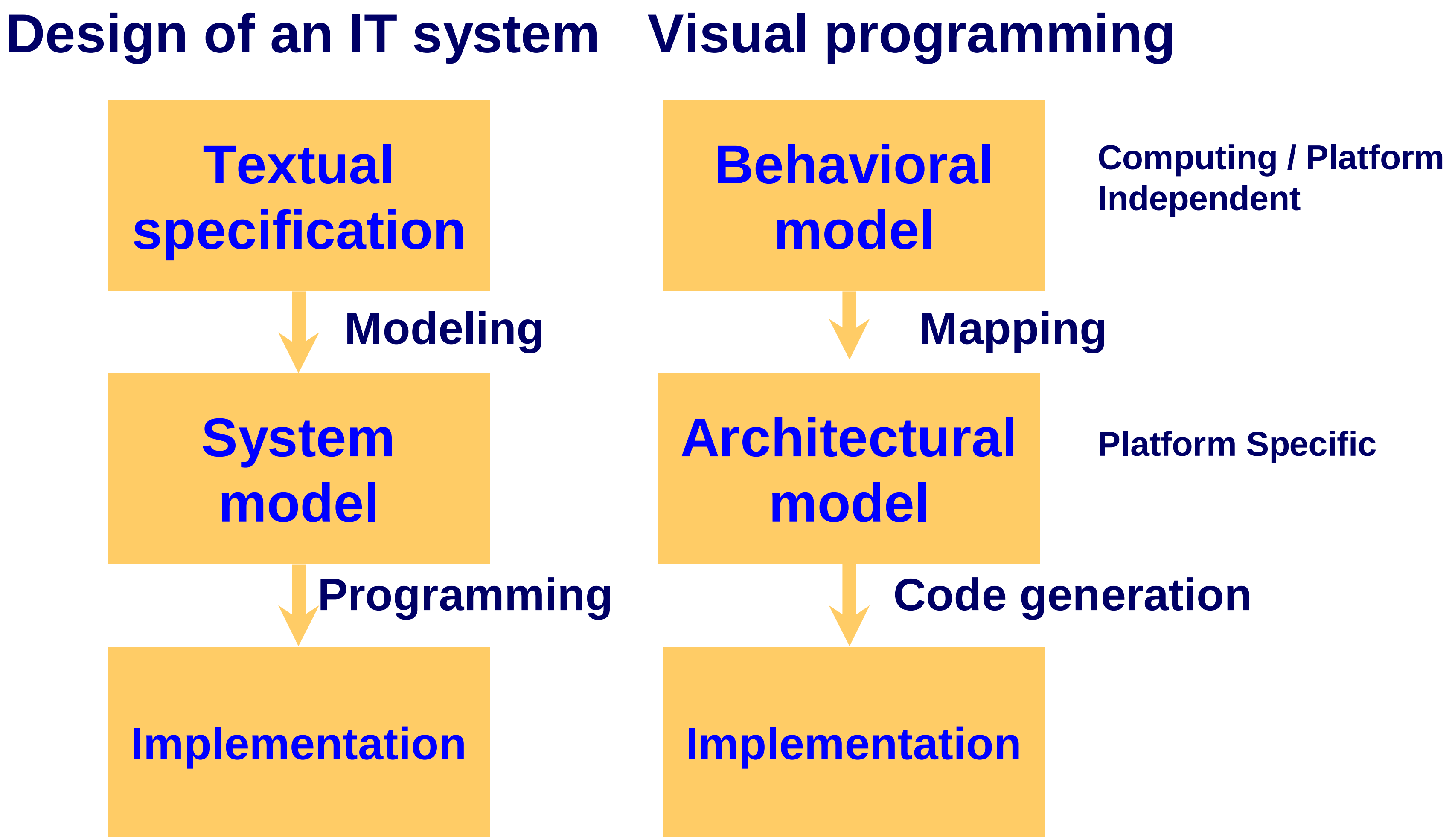
- Model Driven Engineering:
  - 'The systematic use of models as primary engineering artifacts throughout the engineering lifecycle.'
- Best known initiative: OMG Model Driven Architecture
- Metamodel: a collection of notions of a given domain
  - For engineering purposes: precisely defined modeling languages
  - Metametamodels: the languages for model language definition
  - We all know some of them
    - MOF for UML
    - XML Schema for XML languages
    - …

Introduction

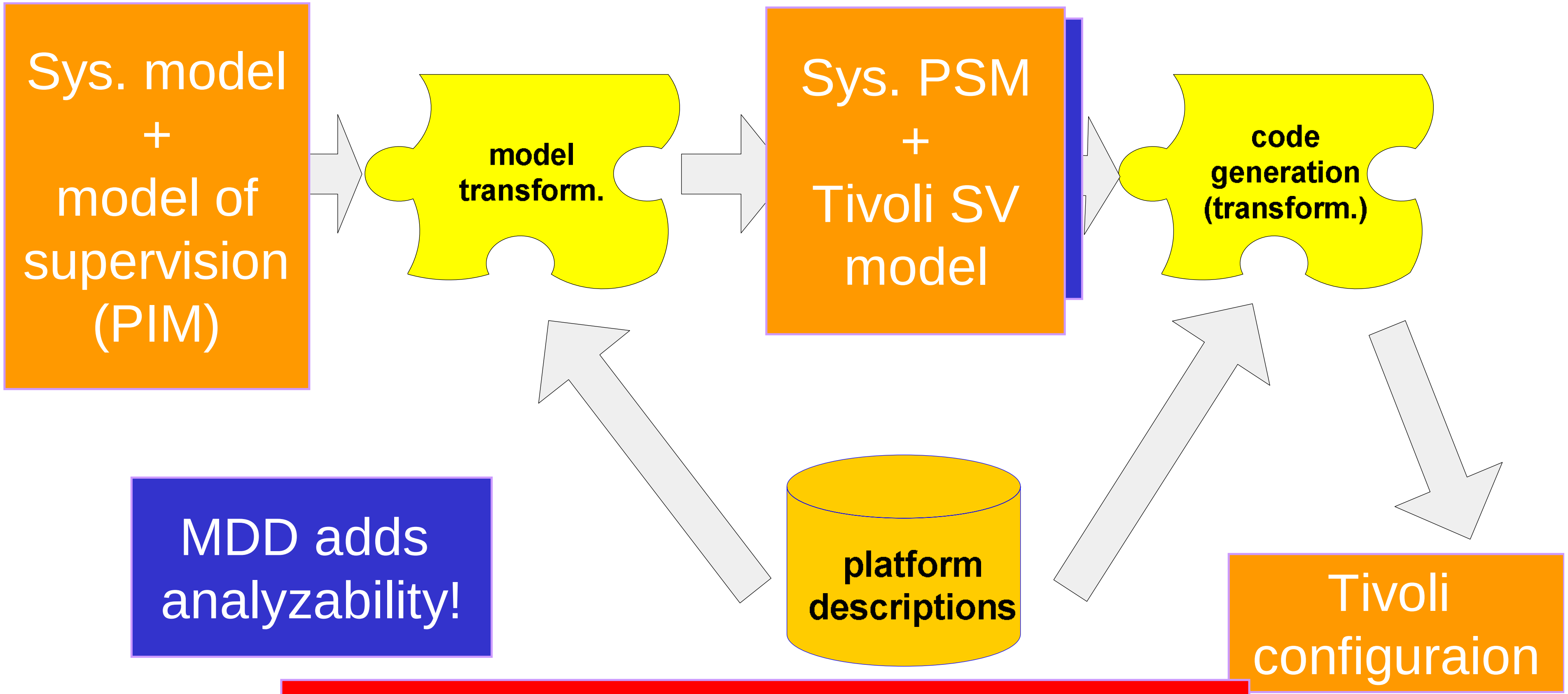
What are ‘model transformations’?

- Model transformations
  - Transformations of models of a given metamodel to models of another metamodel
  - Less cryptically: UML to C++ code, EJB to RDB, MIB to CIM, ...
- Supporting frameworks reaching industrial strength nowadays
  - Mathematically precise language definitions, efficient execution
  - VIATRA2
- For motivation: some applications of model transformations, extensions to resilient system design
  - PIM – PSM – code mapping in MDA
  - ‘Hidden’ formal methods
  - Model transformations in dependability workflows
  - Meta-level fault and dependability mechanisms

Traditional approach vs. MDA



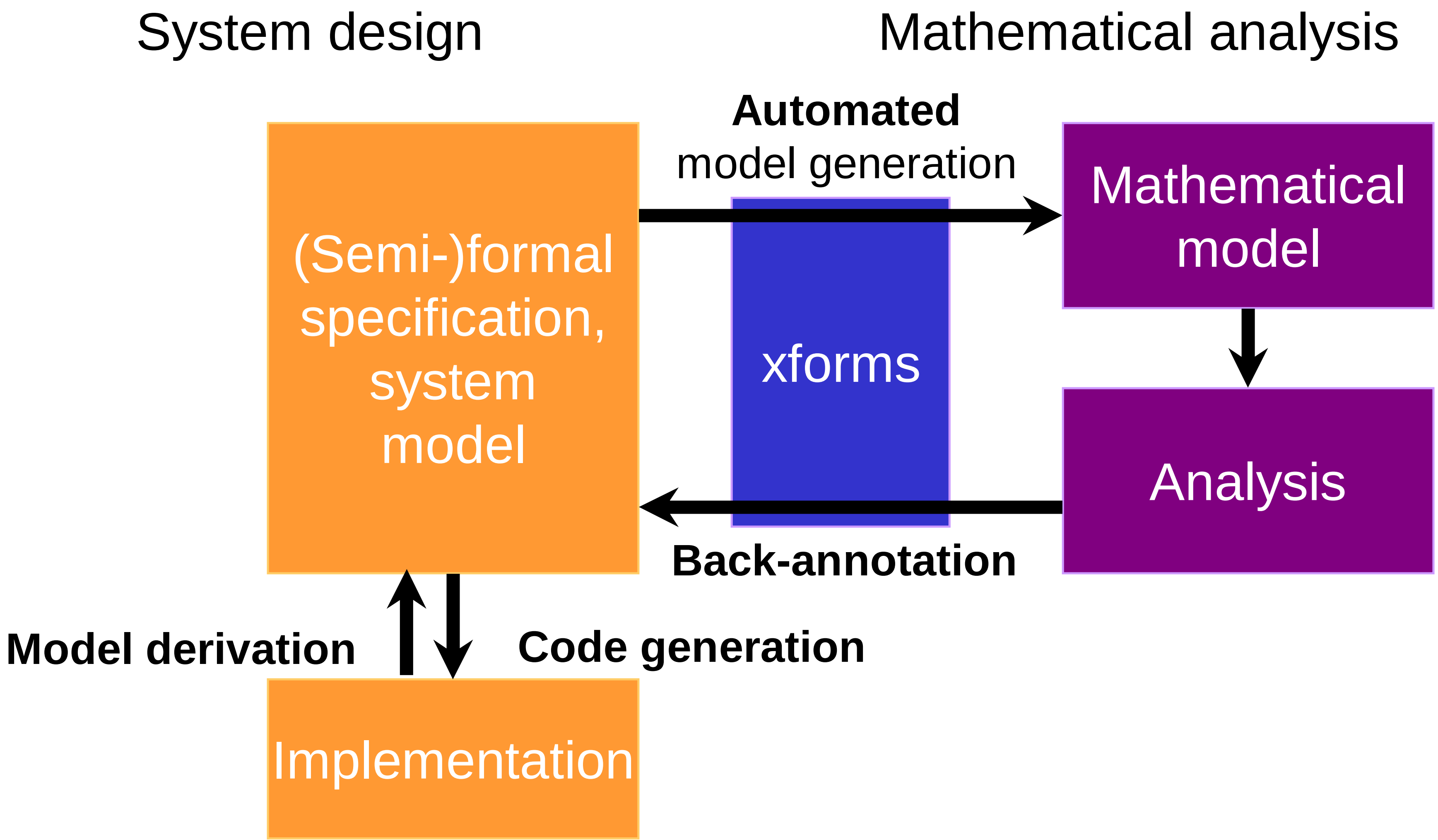
The OMG Model Driven Architecture: Automated Mappings



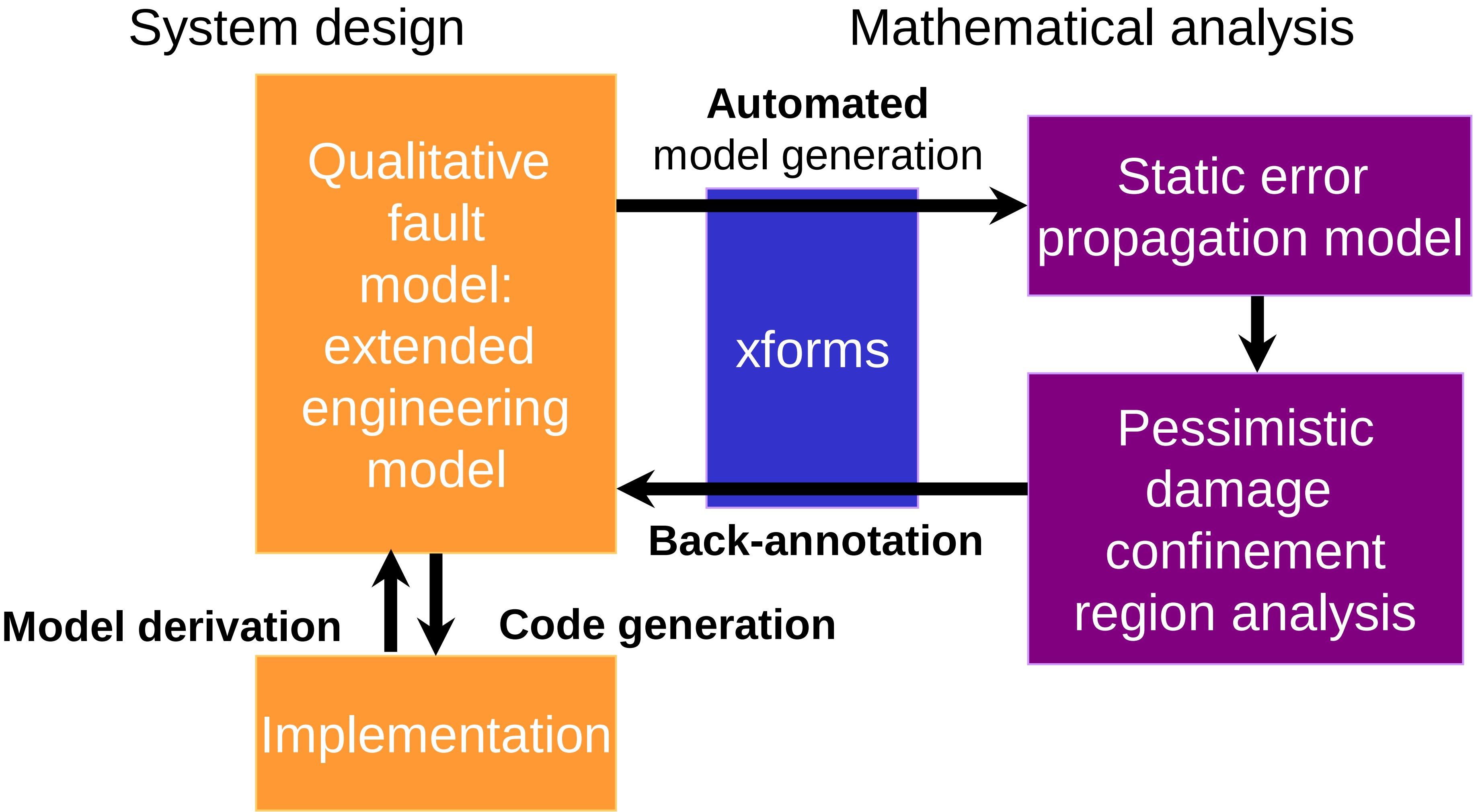
For de … lity
The … ?

MDA for supervised architectures / model based system management: active research at FTSRG (IBM Faculty Award)

Hidden Formal Methods

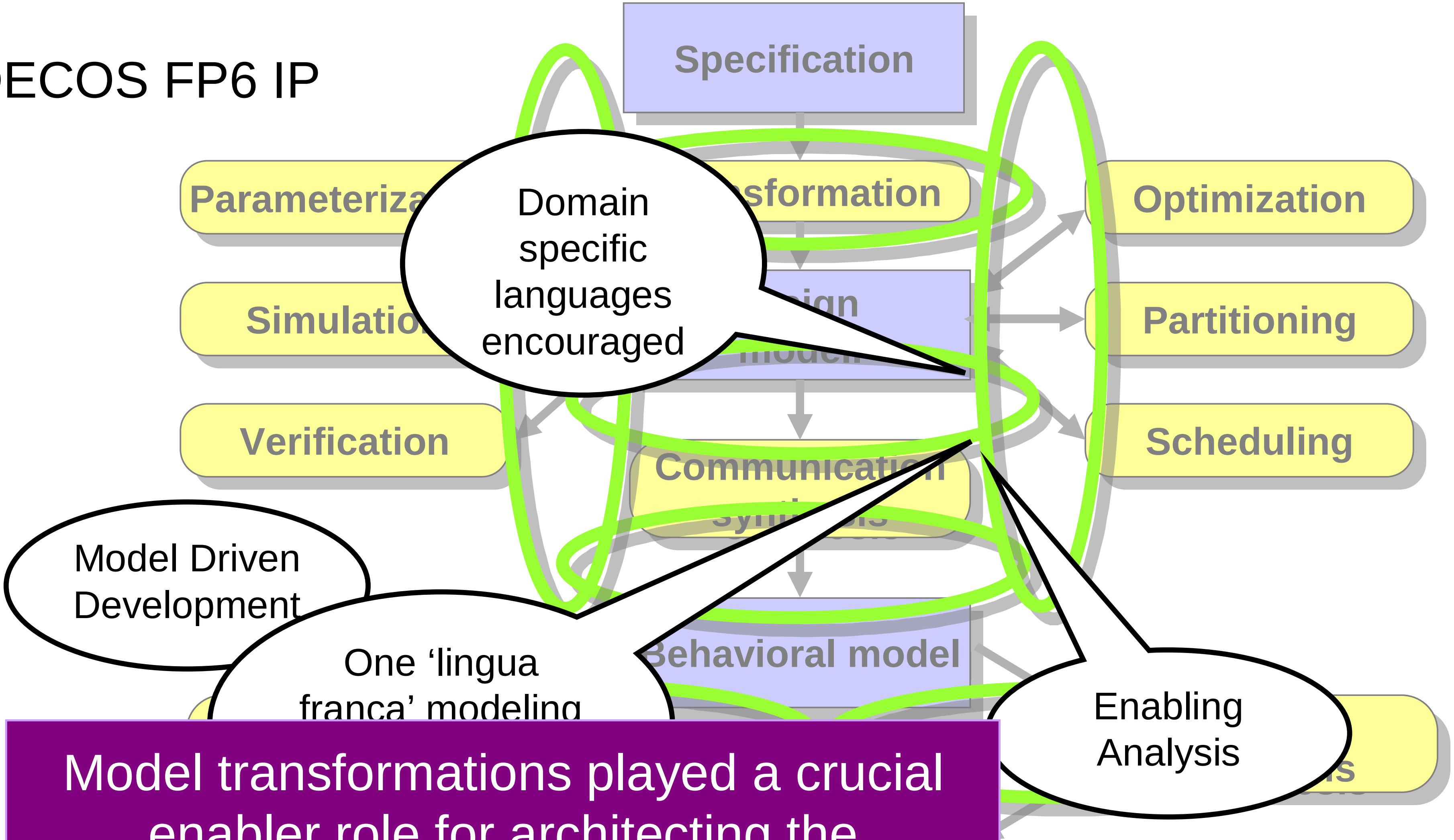


Hidden Formal Methods - Example



Model Transformations in Dependability Workflows

DECOS FP6 IP



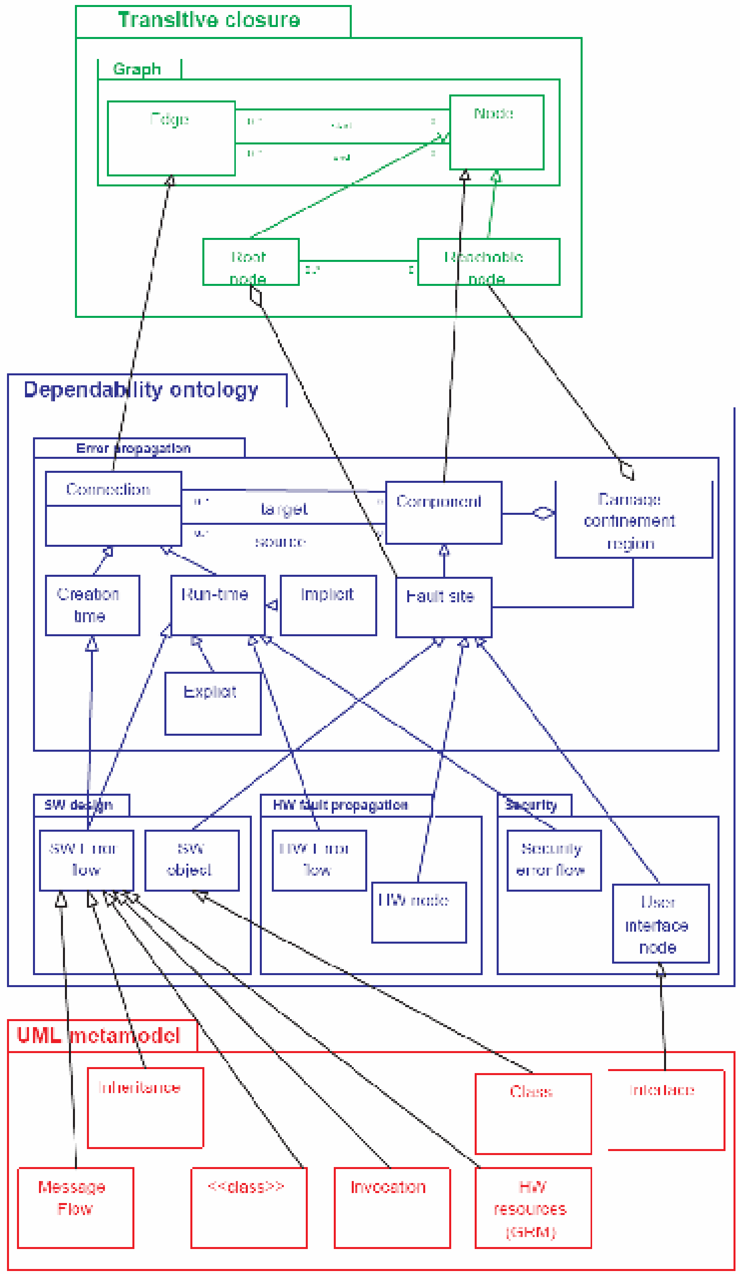
Model transformations played a crucial enabler role for architecting the dependability workflow

Meta-Level Fault Mechanisms

- Faults and the error propagation characteristics are typically largely technology-agnostic and architectural issues

- Thus, can be formulated for classes of (sub)systems in a generalized way
  - Key factors: platform descriptions (PSM), analysis-domain ontologies and (meta)transformations

- New research direction; unexplored area



Tooling: VIATRA2

Analysis of Business Process Models

- Verification by MC
- Fault simulation
- Security analysis (Bell-LaPadula)
- BPEL generation

è *IBM Faculty Award*

Embedded Systems

- PIM & PSM for dependable embedded systems
- PIM & PSM model store
- PIM-to-PSM mapping
- PIM & PSM validation
- Middleware code generation

è *DECOS IP*

SOA

- Performance & Availability analysis
- Configuration generation
- Service Analysis and Deployment

è *SA Forum + SENSORIA IP*

Other

- Design and transformation of domain specific languages
- Model-based generation of graphical user interfaces

VIATRA2

Release 2

Outline

- Introduction
- Features
- The VIATRA2 Framework
- Core concepts
  - Visual and Precise Metamodeling: VPM
  - Transformation definition & execution
  - Code generation
  - Importers
- GUI
- VIATRA2 as an application component

VIATRA =
VIsual Automated model TRAnsformations

- a general-purpose model transformation engineering (*transware*) framework
- that will support the entire life-cycle for transformations
  - *specification*
  - *design*
  - *execution*
  - *validation*
  - *maintenance*
- within and between various modeling languages

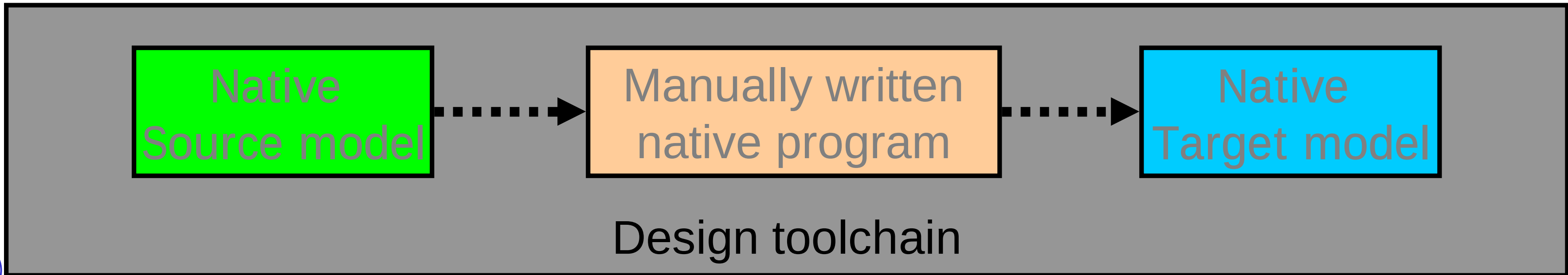
Feautures of the VIATRA2 R2 Framework

- Precise and visual description of source and target modeling languages (metamodeling)
- Precise and visual specification of transformation rules (graph transformation)
- Back-annotation / reverse transformations
- Model transformation engine
  (automatic generation of target models)

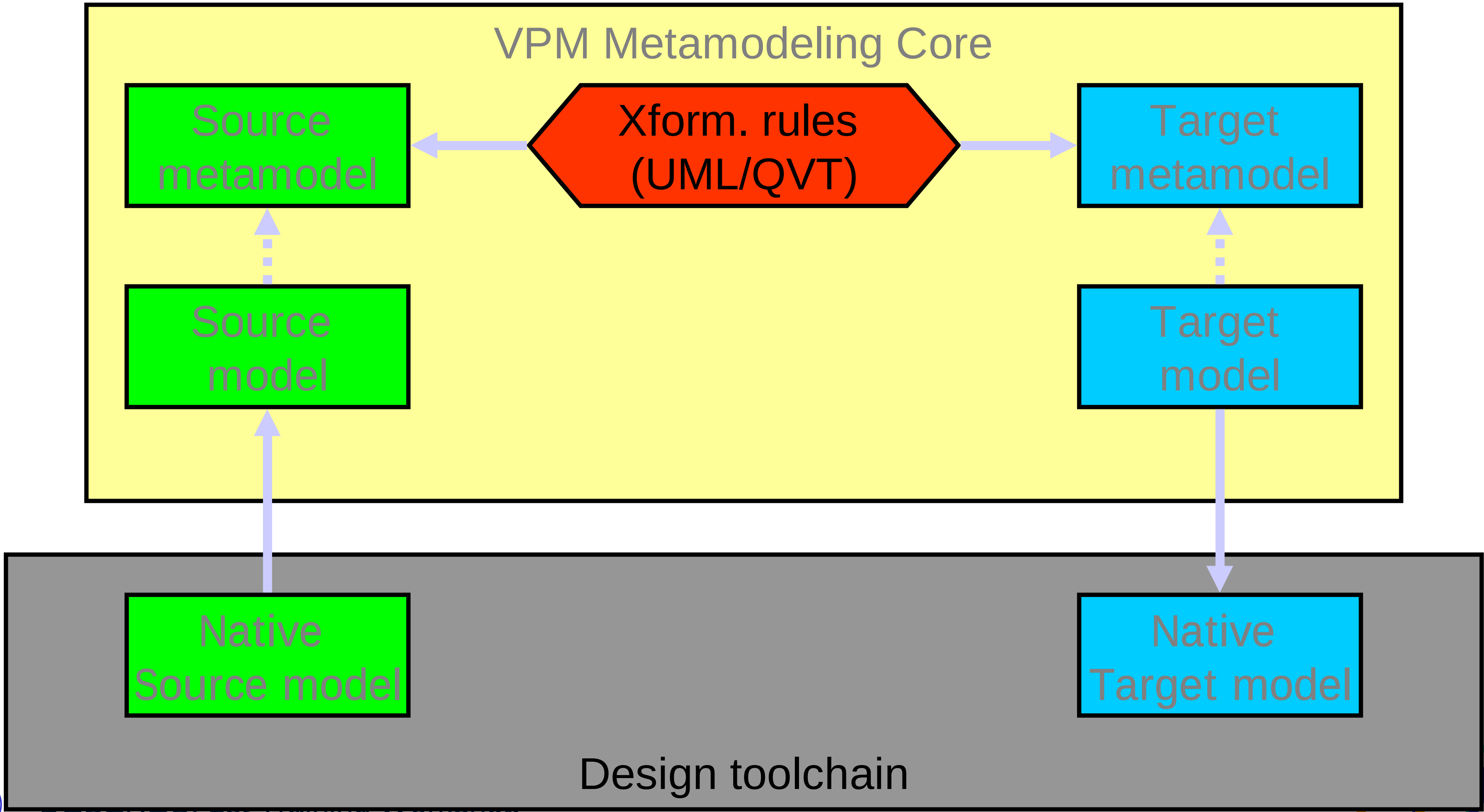
Ongoing research/development:

- Automated generation of platform specific transformers
- Proving correctness and completeness of transformations

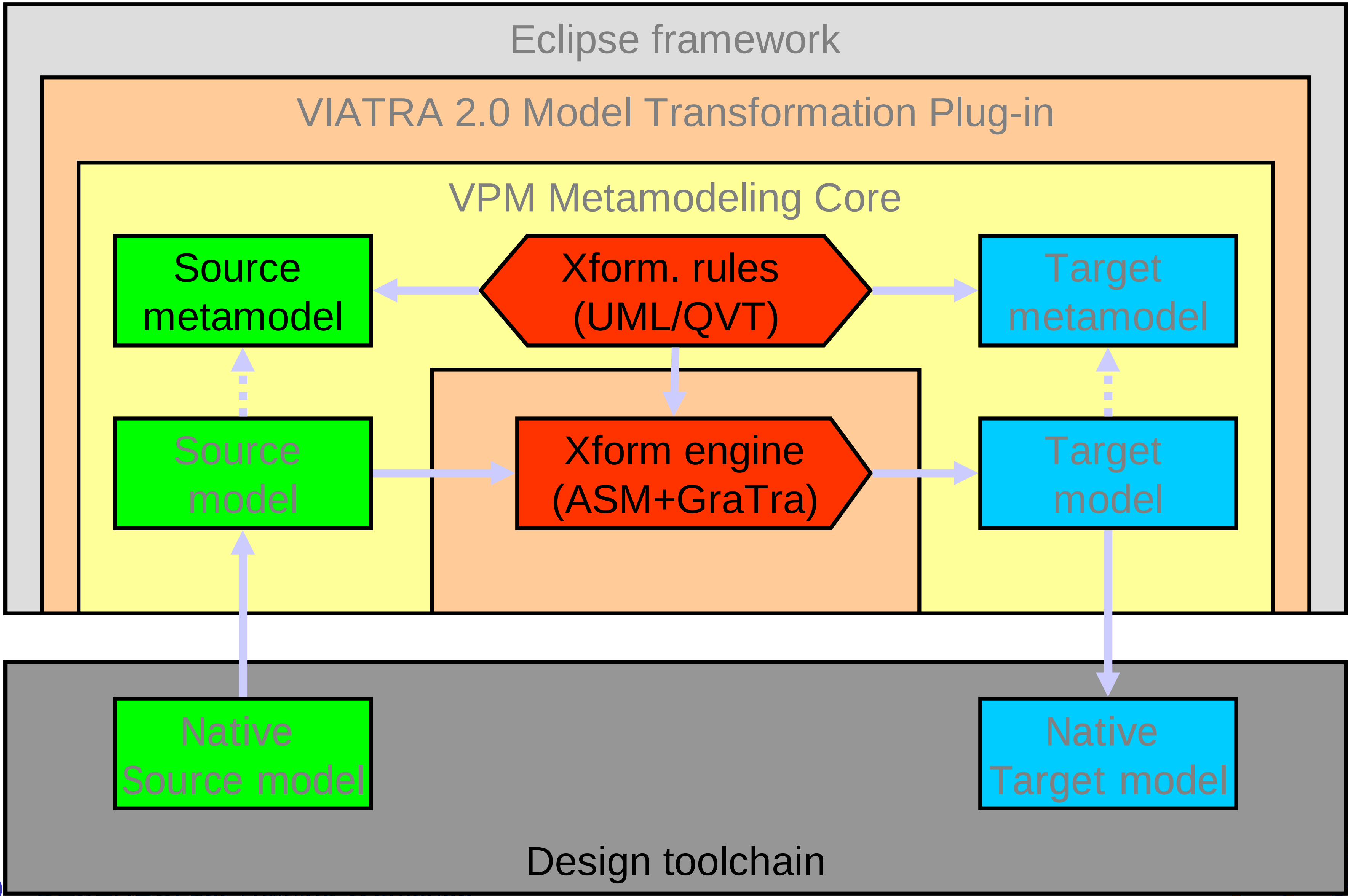
The VIATRA 2.0 framework



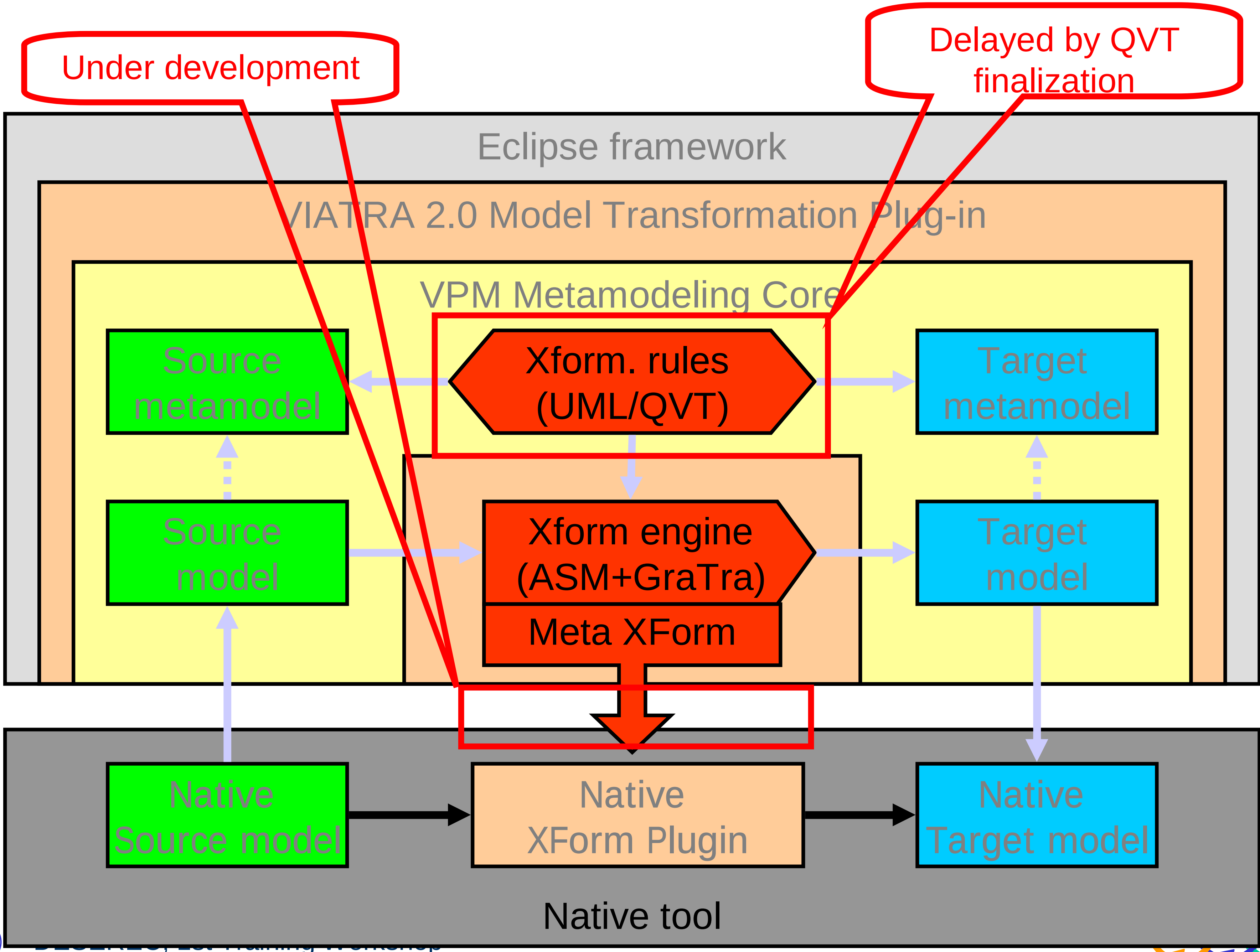
The VIATRA 2.0 framework



The VIATRA 2.0 framework



The VIATRA 2.0 framework



VIATRA2 and Eclipse

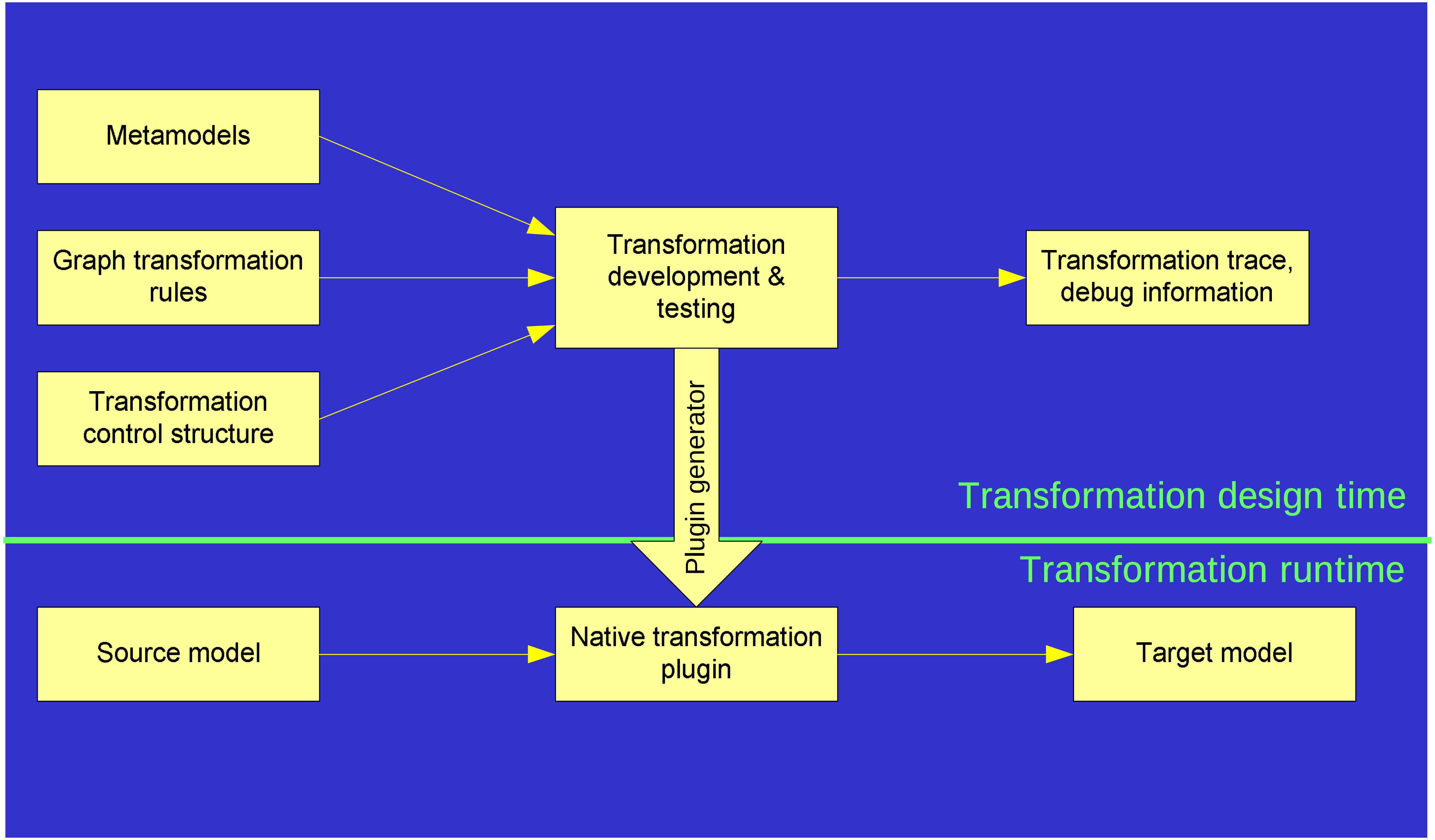
Eclipse quick facts:

- Component-based (‘plugins’)
- Free & Open Source
- Multi-Purpose Development Framework
  - IDE, thin client, application, …
  - A true platform in itself
- THE platform for tool integration today

VIATRA2: an official Eclipse Generative Modeling Tools project

- Realization: a set of Eclipse plugins
- Integration with other Eclipse-based solutions is supported
- Extendability & extension mechanisms
- http://www.eclipse.org/gmt/ (soon to be updated)

Transformation development



Models, Model Manipulation and the 'Last Mile'

Model management:

- **Model space**: Unified, global view of models, metamodels and transformations
  - Hierarchical graph model
  - Complex type hierarchy
  - Multilevel metamodeling

Model manipulation and transformations:
integration of two mathematically precise,
rule and **pattern-based** formalisms

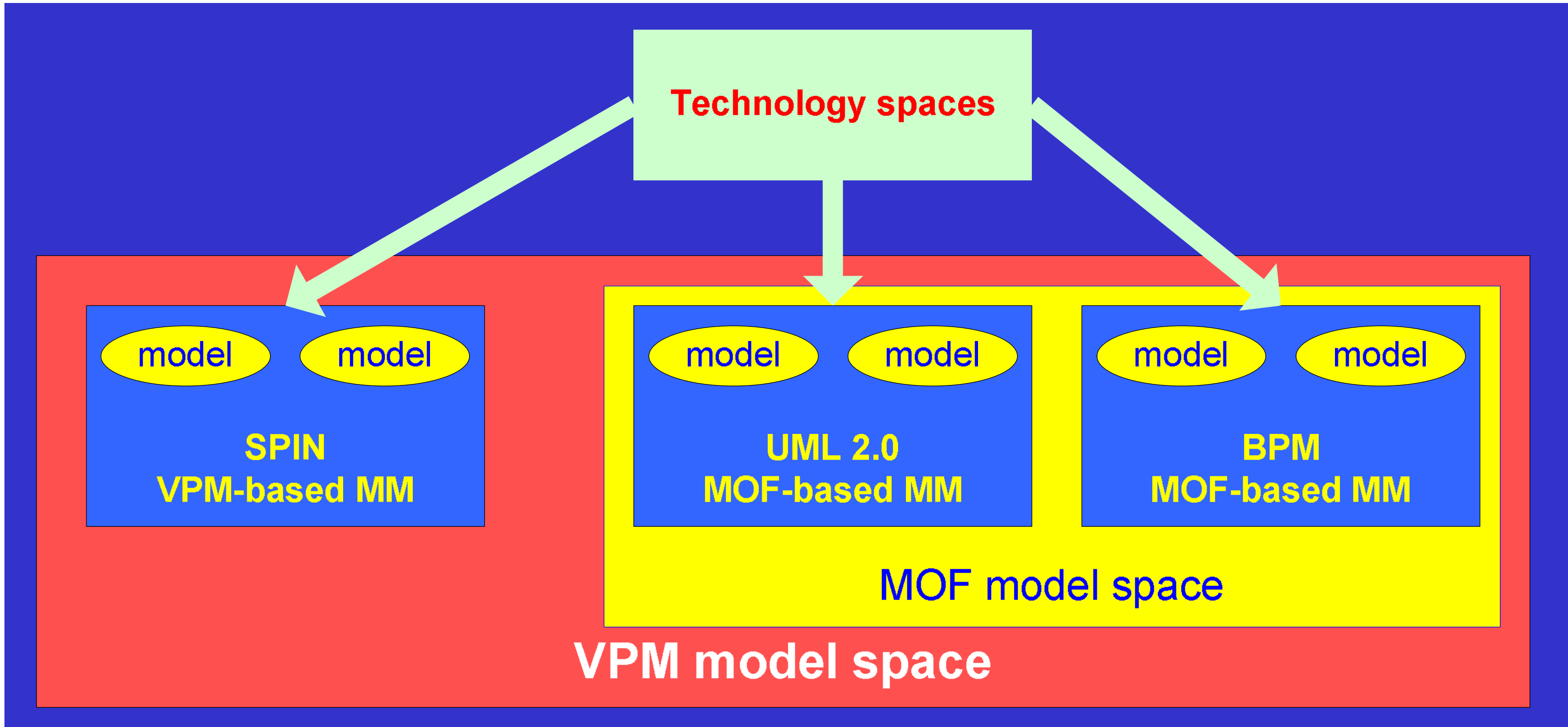
- Graph patterns (GP): structural conditions
- Graph transformation rules (GT): elementary xform steps
- Abstract state machines (ASM): complex xform programs

Code generation:

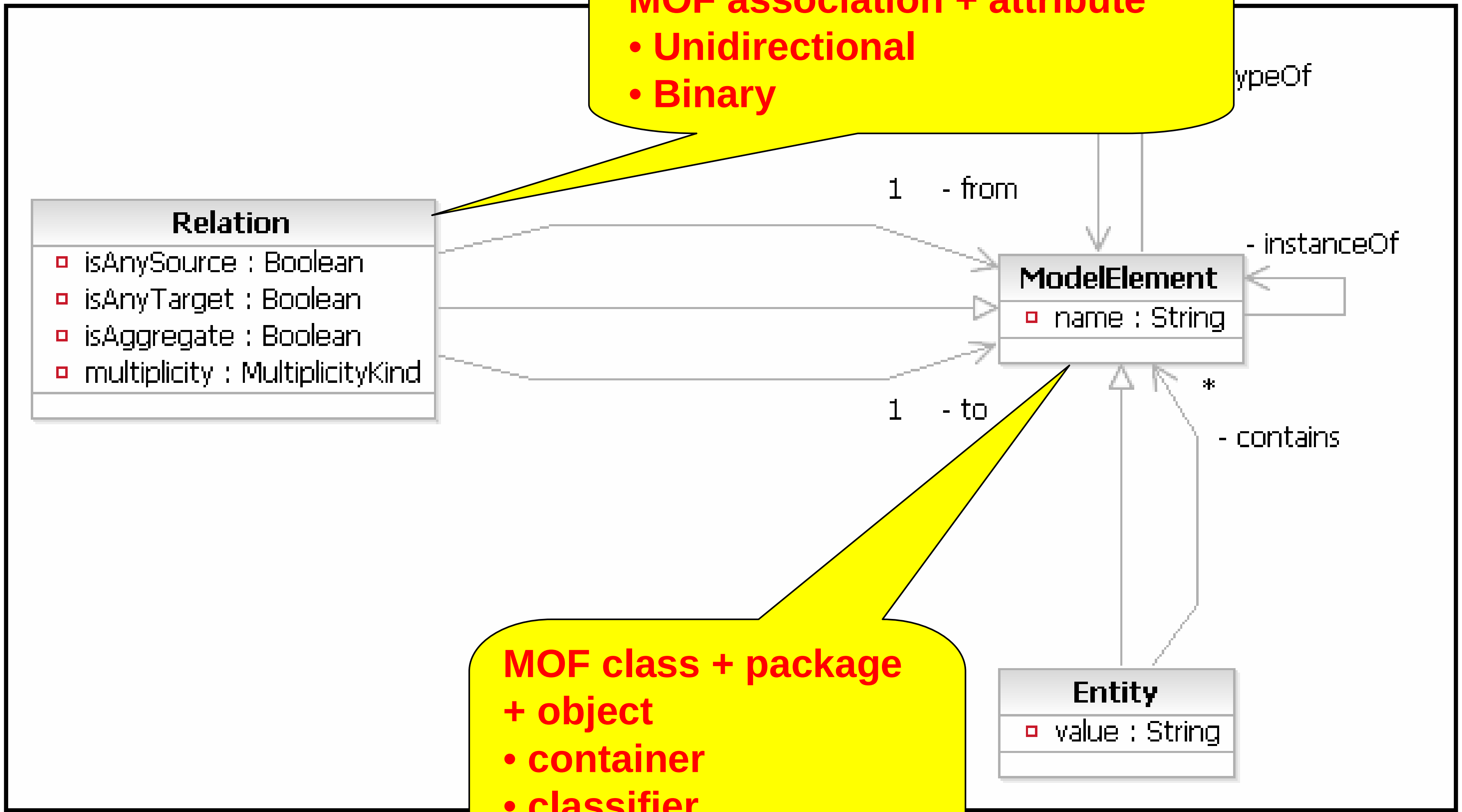
- Special model transformations with
- Code templates and code formatters

VPM: Visual and Precise Metamodeling

- Simple, visual metamodel desing
- Precise semantics
- Multi-level metamodeling: arbitrary meta-level depth
- Simultaneous support of multiple modeling languages



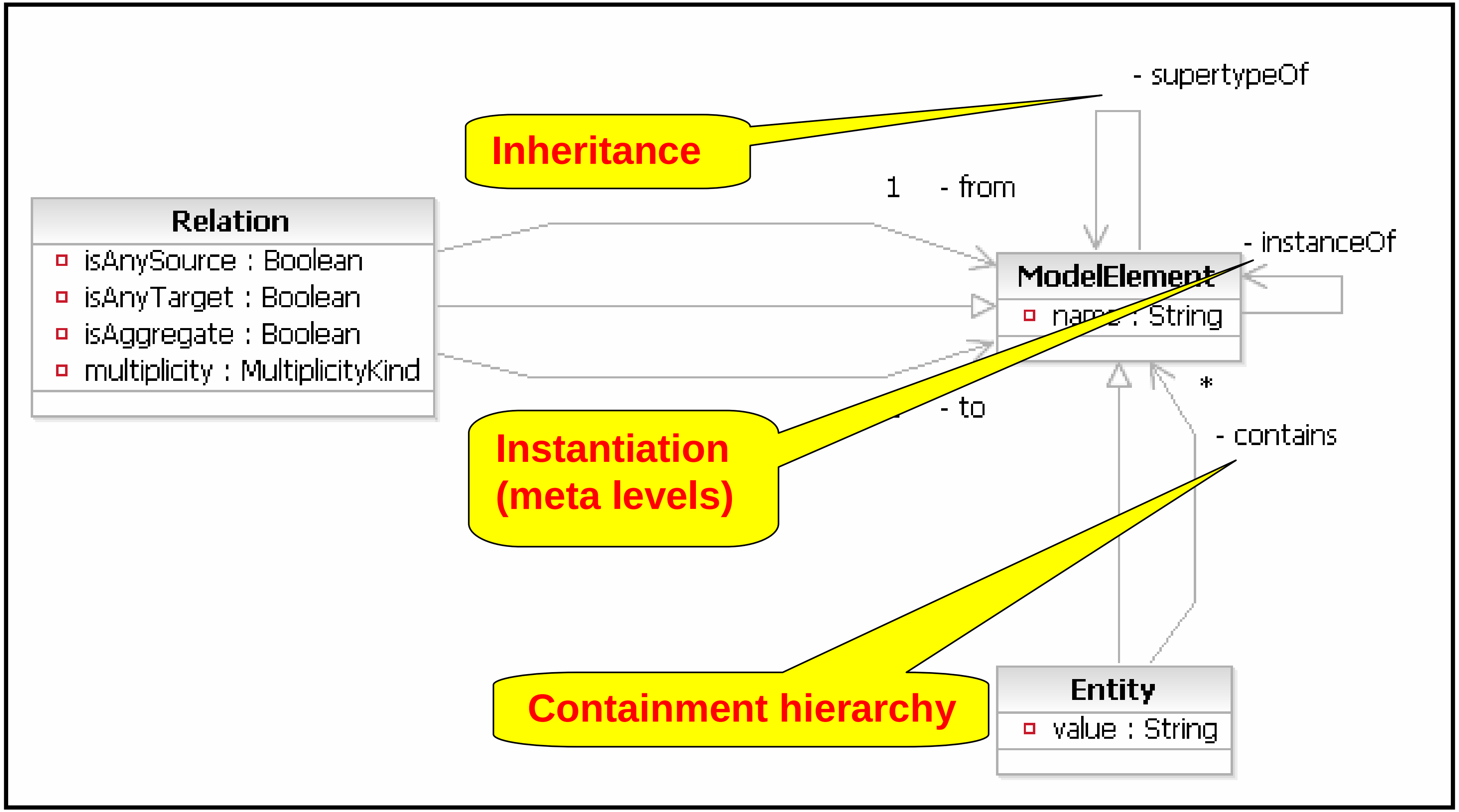
VPM: Visual and Precise Metamodeling



MOF association + attribute
- **Unidirectional**
- **Binary**

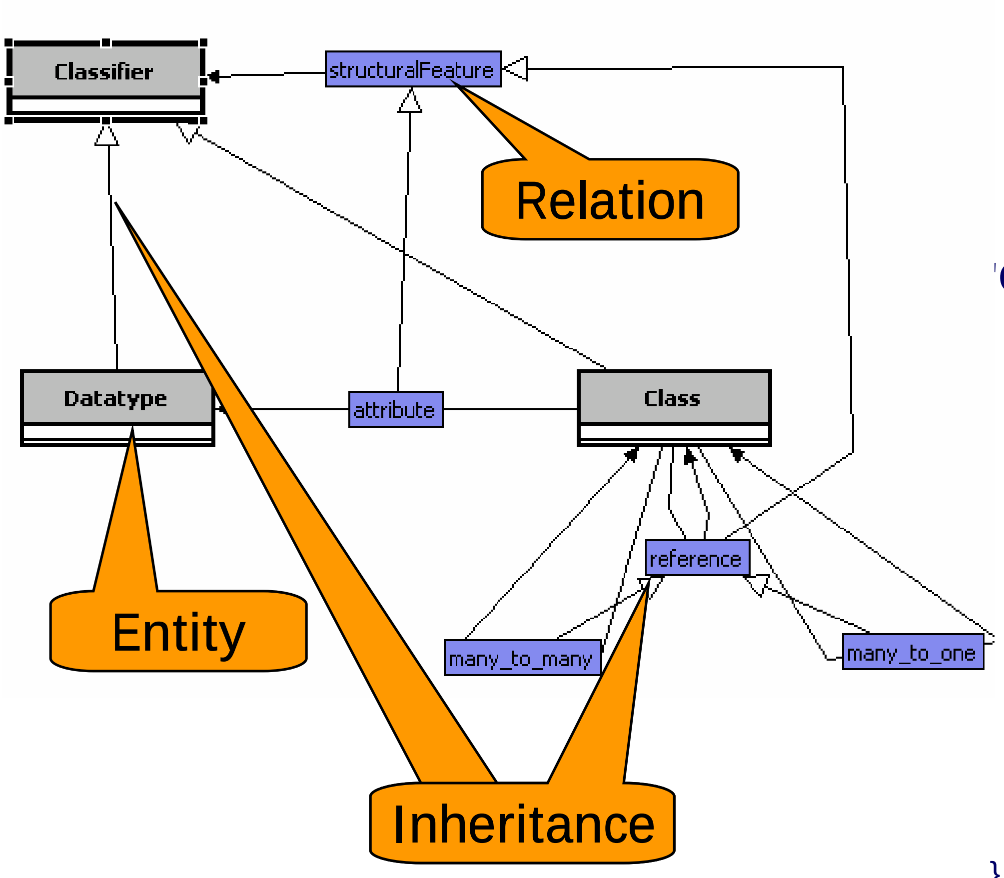
MOF class + package + object
- **container**
- **classifier**
- **constant**
- **object instance**

VPM: Visual and Precise Metamodeling



VTML: VIATRA Textual Metamodeling Language

Example: Ecore Metamodel

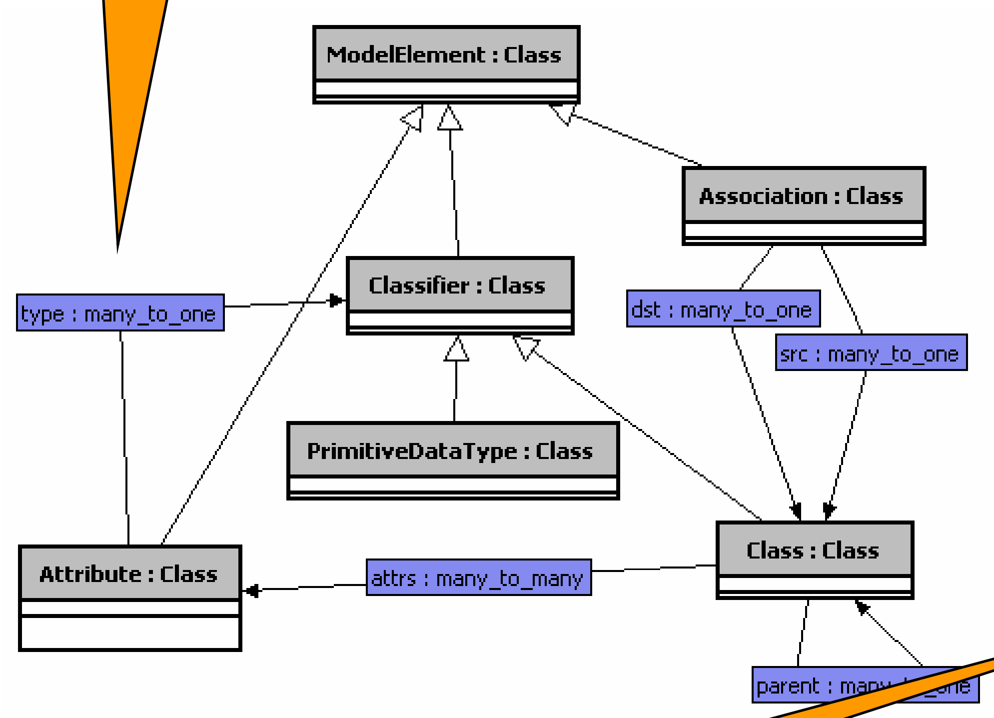


Containment

```
entity(emf) {
  entity(ecore) {
   entity('Classifier');
   entity('Class');
   entity('Datatype');
   relation('structuralFeature', 'Classifier',
'Classifier');
   relation('attribute', 'Class', 'Datatype');
   relation('reference', 'Class', 'Class');
   relation('many_to_one', 'Class', 'Class');
   relation('many_to_many', 'Class', 'Class');
   supertypeOf('Classifier', 'Class');
   supertypeOf('Classifier', 'Datatype');
   supertypeOf('structuralFeature', 'attribute');
   supertypeOf('structuralFeature', 'reference');
   supertypeOf('reference', 'many_to_one');
   supertypeOf('reference', 'many_to_many');
  }
}
```

Example: UML Metamodel as Instance of Ecore

InstanceOf



Metamodel specific types

```
import emf;

entity(uml_class) {
  entity(metamodel)  {
    ecore.'Class'('ModelElement');
    ecore.'Class'('Classifier');
    supertypeOf('ModelElement','Classifier');
    ecore.'Class'('Class') {
      ecore.attribute(name, 'Classifier', 'String');
      ecore.attribute(isPersistent, 'Class', 'Bool');
    }
    supertypeOf('Classifier', 'Class');

    ecore.many_to_one(parent, 'Class', 'Class');
    ecore.many_to_many(attrs, 'Class',
'Attribute');
  }
}
```

Models, Model Manipulation and the 'Last Mile'

Model management:

- **Model space**: Unified, global view of models, metamodels and transformations
  - Hierarchical graph model
  - Complex type hierarchy
  - Multilevel metamodeling

VTML

Model manipulation and transformations: integration of two mathematically precise, **rule** and **pattern-based** formalisms

- Graph patterns (GP): structural conditions
- Graph transformation rules (GT): elementary xform steps
- Abstract state machines (ASM): complex xform programs

VTCL

Architecture ready to integrate alternative transformation languages (via new interpreters)

Code genera ion

- Special model t
- Code templates

Distinguishing feature: Metatransformations (rules that manipulate rules as models)

Ongoing: declarative transformations (description logic, QVT)

Graph patterns

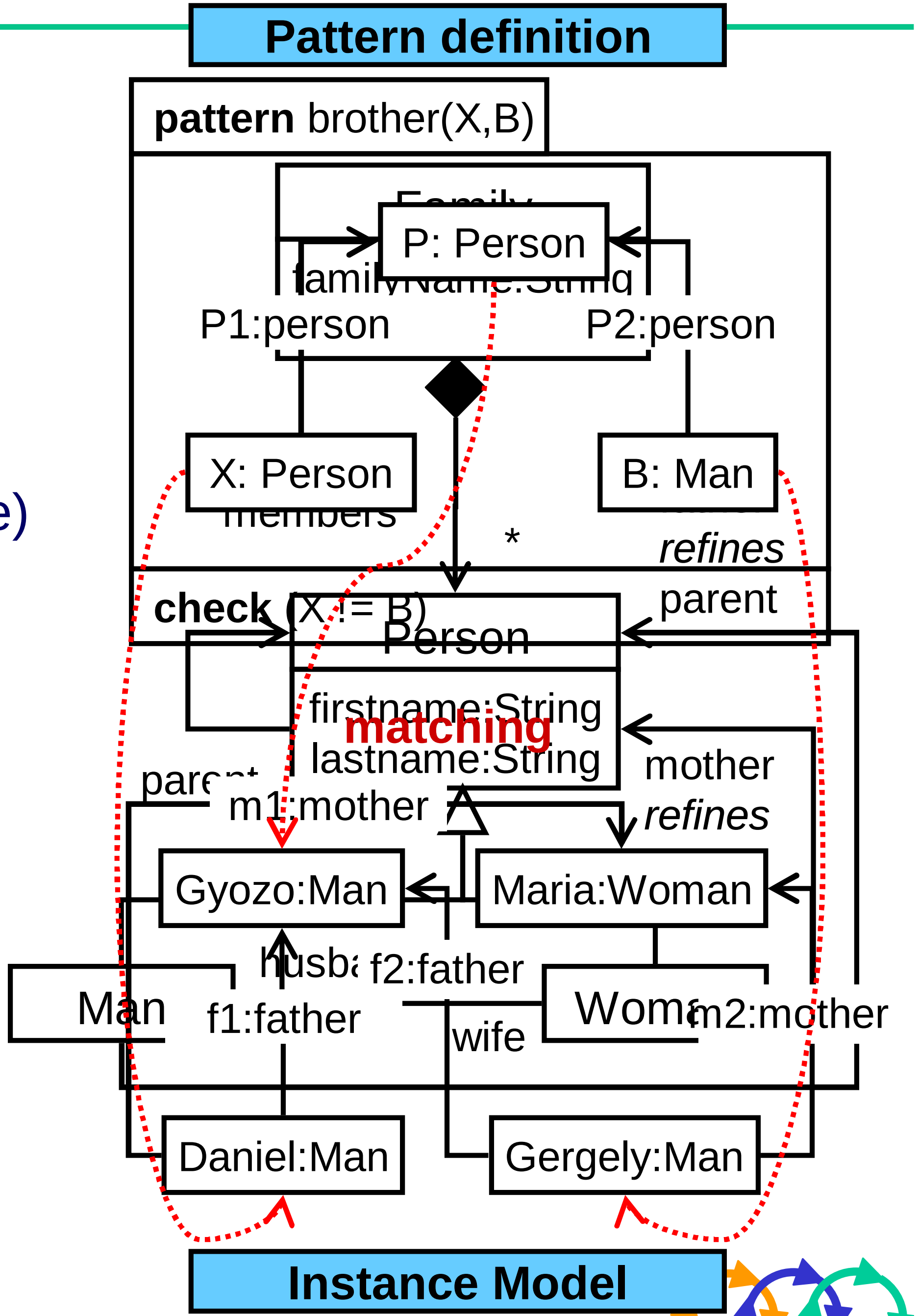
Graph Pattern

- Structural conditions that have to be fulfilled by a part of the model space

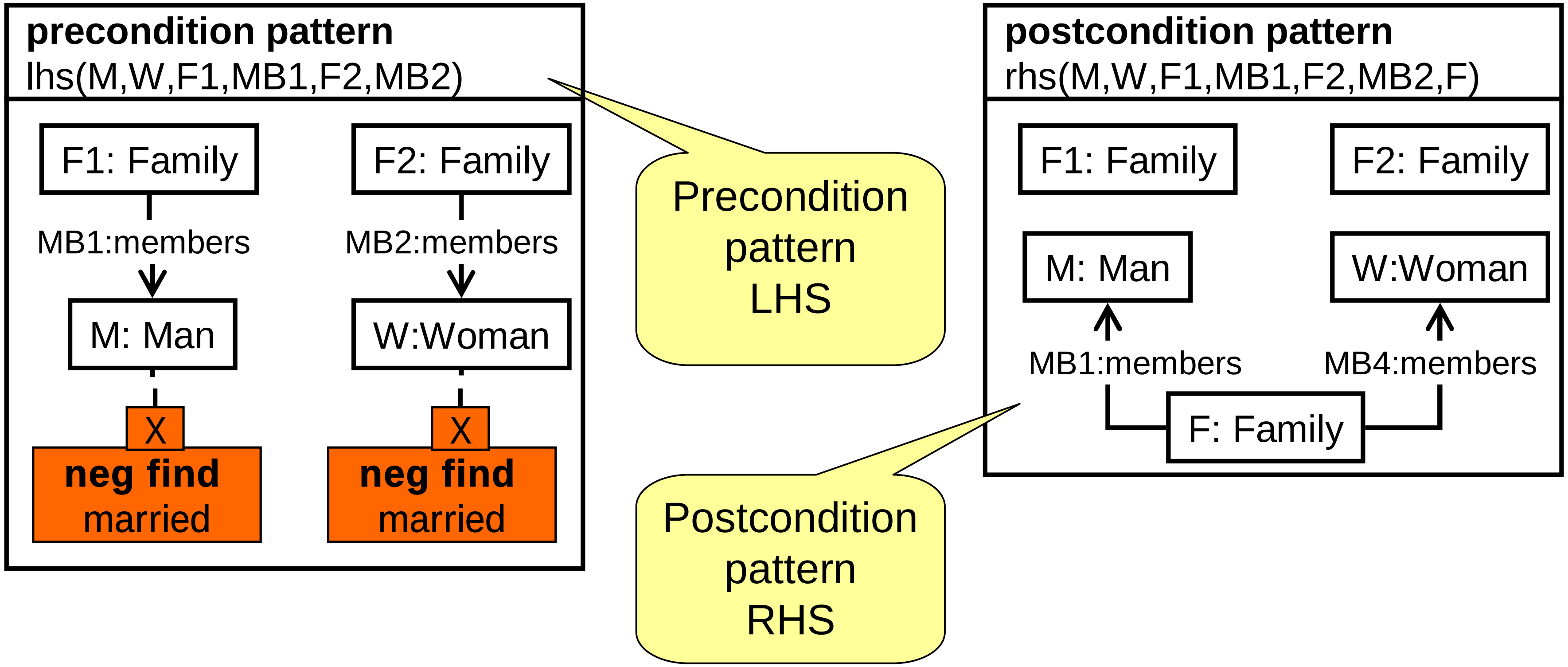
Graph pattern matching

- A model (i.e. part of the model space) can satisfy a graph pattern,
- if the pattern can be matched to a subgraph of the model

- Note that we omit here the 'fine detail' (recursion, OR-patterns, neg-pattern hierarchy,..)



Graph Transformation Rules



Three different kinds

- LHS + RHS
- LHS + actions (ASM, follows)
- Merged LHS-RHS (new, del annotations)

Abstract State Machines

ASM: high-level programming language

- Control structure for xform
- Integrated with GT rules

Examples

- **update** *location* = *term*;
- **parallel** {…} / **seq** {…}
- **let** *var* = *term* **in** *rule*;
- **if** (*formula*) *rule1*; **else** *rule2*;
- **iterate** *rule;*
- **forall/choose** *variables* **with** *formula* **do** *rule*;
- **forall/choose** *variables* **apply** *gtrule* **do** *rule*;

```
forall X below people.models,
        B below people.models
        with find brother(X, B) do seq {
          print(name(X) + "->" + name(B));
        }

let X = people.models.Varro1.Daniel,
Y = people.models.Gyapay1.Szilvia,
F = undef, F2 = undef in
choose Z below people.models
apply marry(X, Y, F) do seq {
          rename(F, "Varro2");
          move(F, people.models);
          iterate
            choose M below people.models,
       W below people.models
        apply marry(M, W, F2) do
                move(F2, people.models);
```

Models, Model Manipulation and the 'Last Mile'

VTML

Model management:

- **Model space**: Unified, global view of models, metamodels and transformations
  - Hierarchical graph model
  - Complex type hierarchy
  - Multilevel metamodeling

Model manipulation and transformations: integration of two mathematically precise, **rule** and **pattern-based** formalisms

VTCL

- Graph patterns (GP): structural conditions
- Graph transformation rules (GT): el
- Abstract state machines (ASM): co

Automatically transformed to VTCL

Code generation:

- Special model transformations with
- Code templates and code formatters

VTTL

Code templates

Code generation

- Code templates
- Code formatters

Code templates

- Text block with references to GTASM patterns, rules
- Compiled into GTASM programs with prints
  - Velocity templates

Code formatters

- Split output code into multiple files
- Pretty printing

```
template printClass(in C) =
{
public class $C {
#(forall At,Typ with attrib(C,At,Typ) do seq{)
private $Typ $At;
#(})
}
}

// Generated
rule printClass(in C) = seq {
        print("public class " + C + "{");
        forall At,Typ with attrib(C,At,Typ)
do
          seq {
            print("private " + Typ + " " + At +
";");
          }
        print("}");
}
```

VIATRA Importers

The step from concrete to abstract syntax!

Model import modules

- Specific to a tool version
- Eclipse plugins
- Can be easily customized, upgrad
- Can be installed/uninstalled separate fro framework

Ease of extendability was of priority (small and easy to use API)

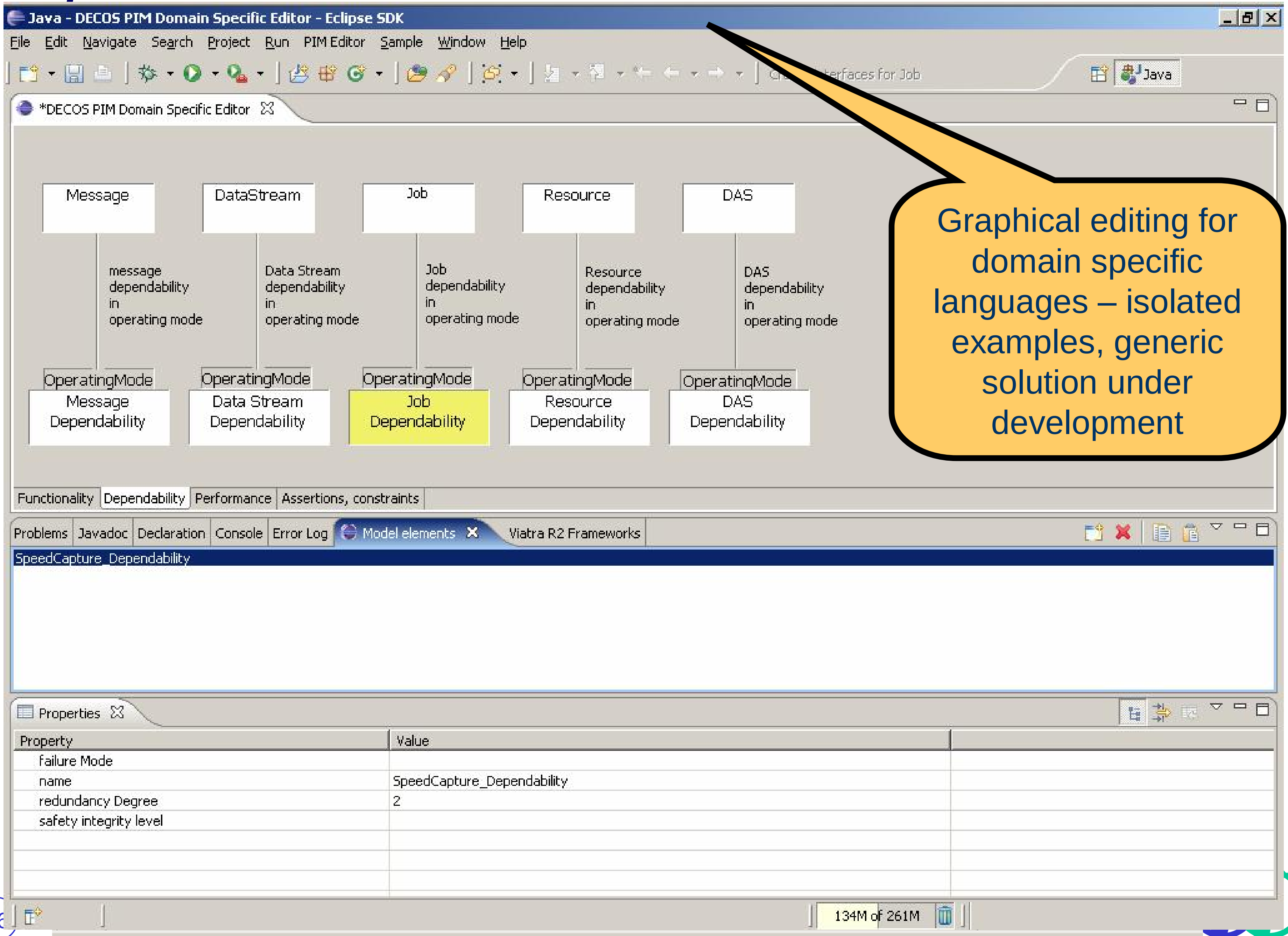
UML 1.x

- Importers for
  - IBM Rational Rose 2002
  - IBM Rational XDE 2003
  - Sparx Systems Enterprise Architect 4.0

UML 2.0

- New metamodel
- Importer for Rational Software Architect 6.0

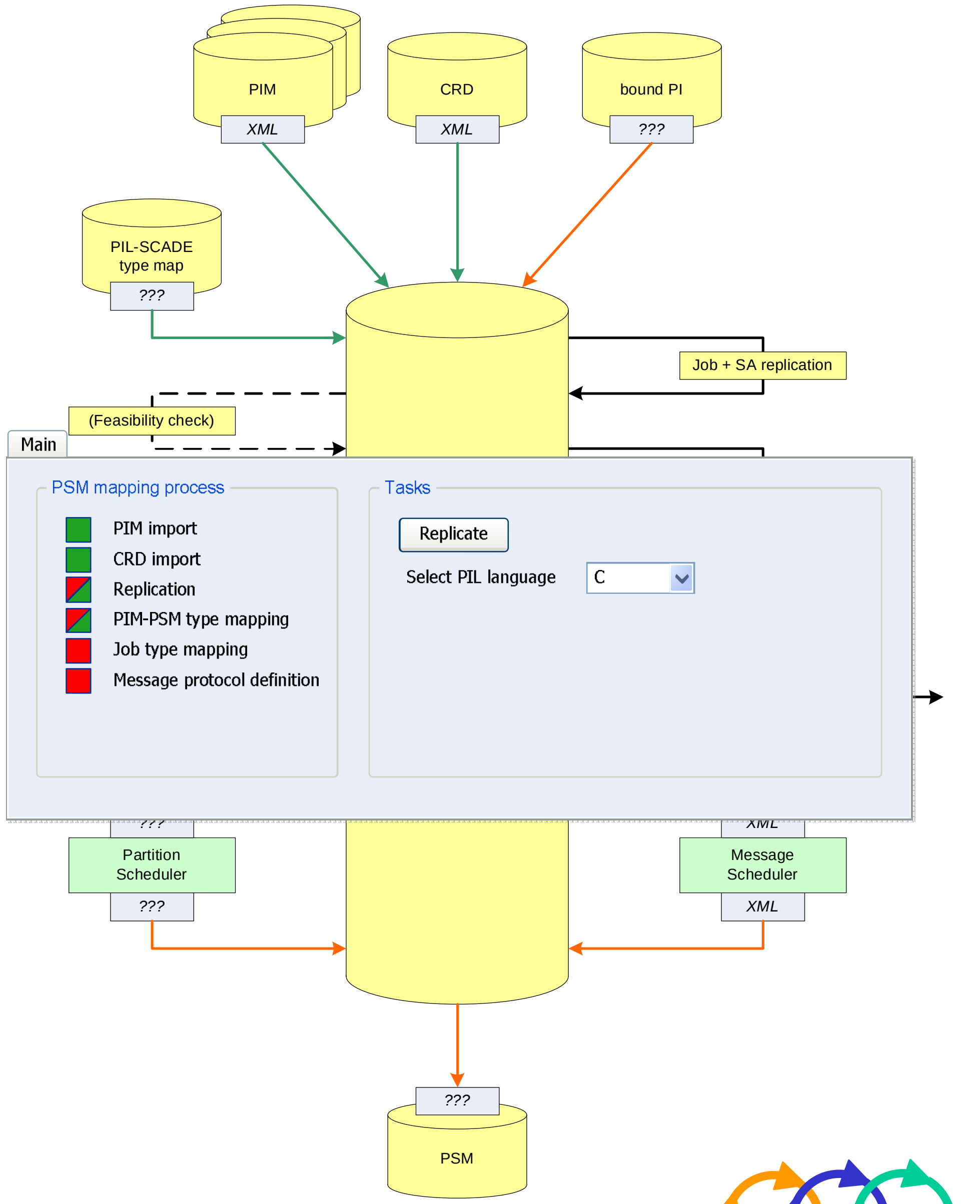
Eclipse-based GUI



–*VIATRA as an Application Component*

"VIATRA application"

- Custom UI above the general framework
- Contains more transformation descriptions and metamodels
- The control flow of transformations can depend on user input
- Example: complex PIM-PSM mapping



Qualitative Fault Modeling, Transformations and Resilient Systems

Qualitative fault modeling

Basis: Architecture design

Metamodel-based fault modeling

- UML General Resource Model (GRM):
  Resource types (active, passive, protected etc.),
  Usage scenarios
- Operational faults are considered
- Faults are introduced here systematically

Common cause failures:

- Introduced by resource sharing

Applications

Origins: mid-nineties (York, TUB)

- A few qualitative values (good, faulty, early, late)

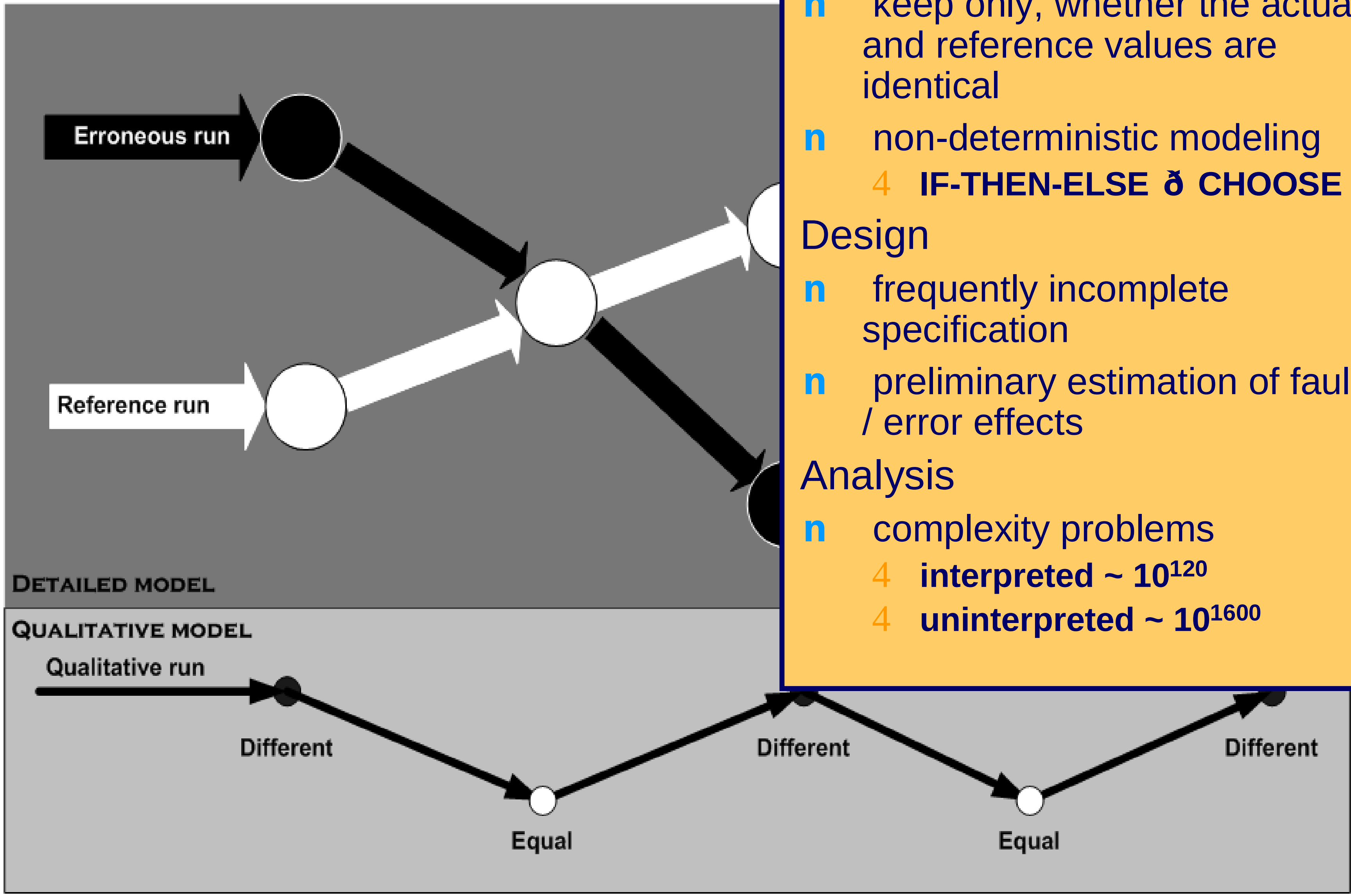
Applications:

- industrial models
- railway interlocking systems
- e-Business processes

Experiences:

- effective both in modelling and analysis (as briefly follows)

Basic idea of qualitative fault modeling



Basic idea:

- keep only, whether the actual and reference values are identical
- non-deterministic modeling
  - **IF-THEN-ELSE ð CHOOSE**

Design

- frequently incomplete specification
- preliminary estimation of fault / error effects

Analysis

- complexity problems
  - **interpreted ~ 10^{120}**
  - **uninterpreted ~ 10^{1600}**

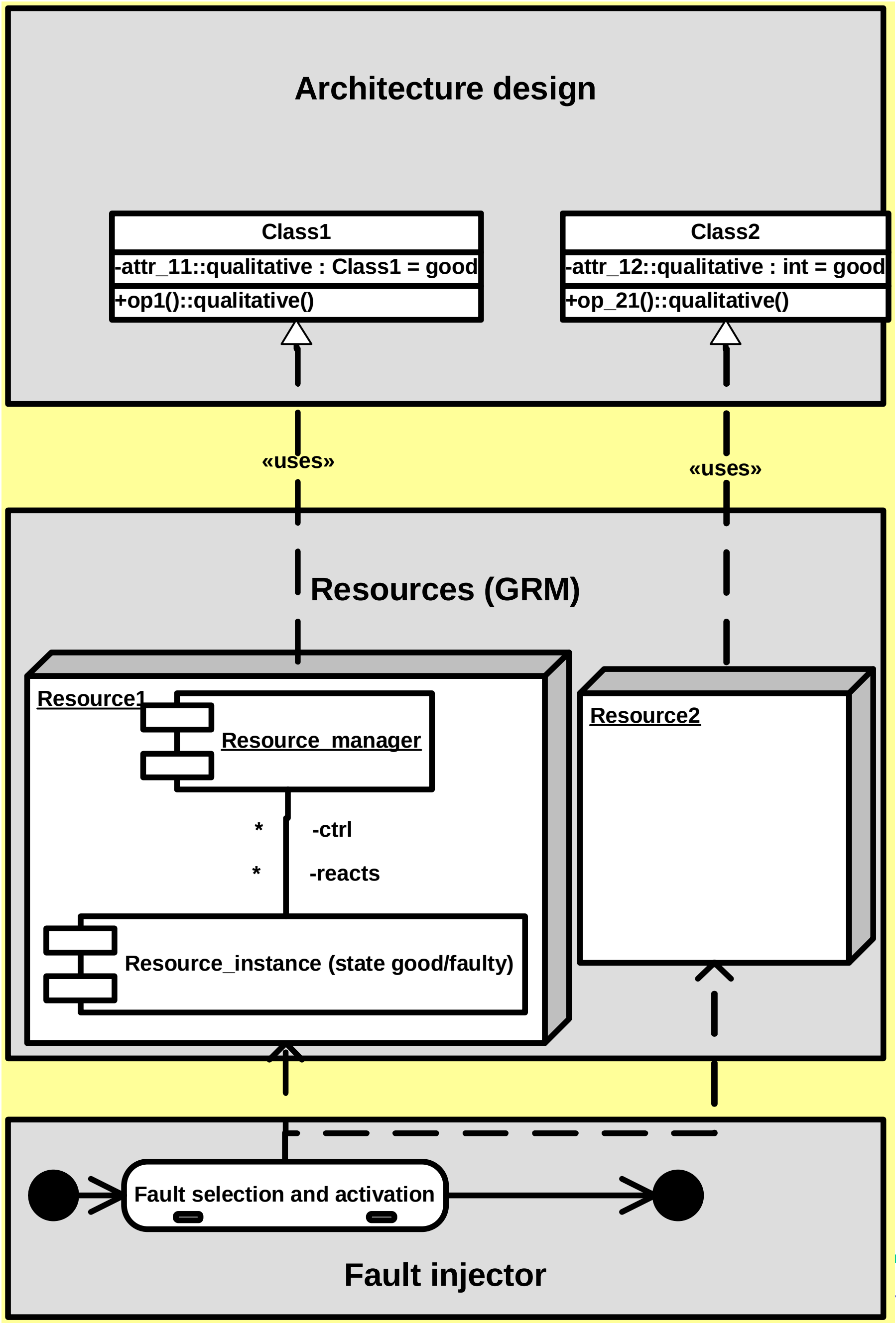
Part of a UML Profile

Model of the inter-actions with resources via GRM

Insertion of (qualitative) faults at the resources

Error propagation through the scenarios

Qualitative fault modeling is tried & tested – the task is to integrate it with MDD



Analysis of error propagation

Extension of the architectural model:
fault effects + error propagation rules

Checking high-level (abstract) operation
in the presence of anticipated faults
(fault simulation)

Estimating system properties:

- Coverage of fault tolerance techniques
- Testability, diagnosability of faults
- Potentially catastrophic fault effects

Dependability analysis

Basis: Architecture design (PSM)

Quantitative reliability/availability analysis:

- Comparison of alternatives
- Elimination of bottlenecks
- Sensitivity analysis

Qualitative dependability analysis:

- Rule-based prediction of faulty behavior

Design patterns for dependability
(redundancy management)

Formal verification of behavior

Basis: Behavioral model (control flow)

- Complex control algorithms
- Event driven, asynchronous operation

→ Exhaustive testing is infeasible

Classical reachability analysis:

- Temporal logic model checking
(general and application-specific requirements)

Additional improvements:

- Semi-decision techniques
(handling large state spaces by

Conclusion: many (classic) V&V activities meaningful in the DESEREC context; to the least tool & process integration can benefit from transformation support

Live Tool Demonstration