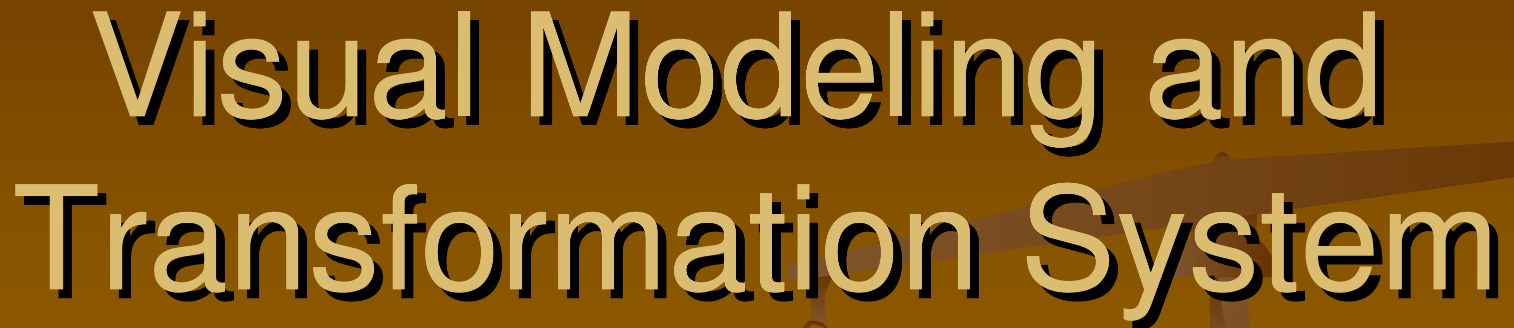
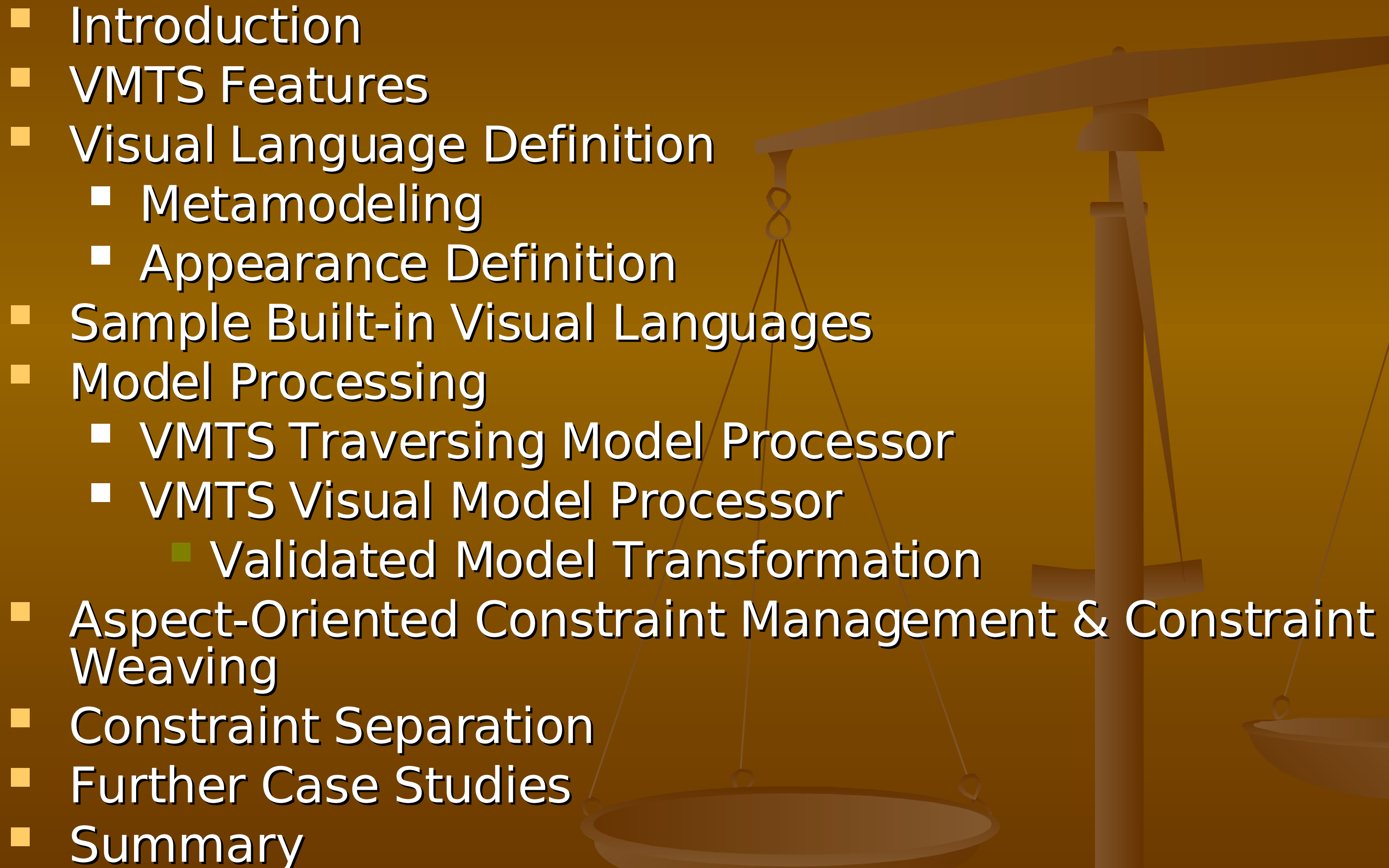


http://www.vmts.aut.bme.hu

Outline

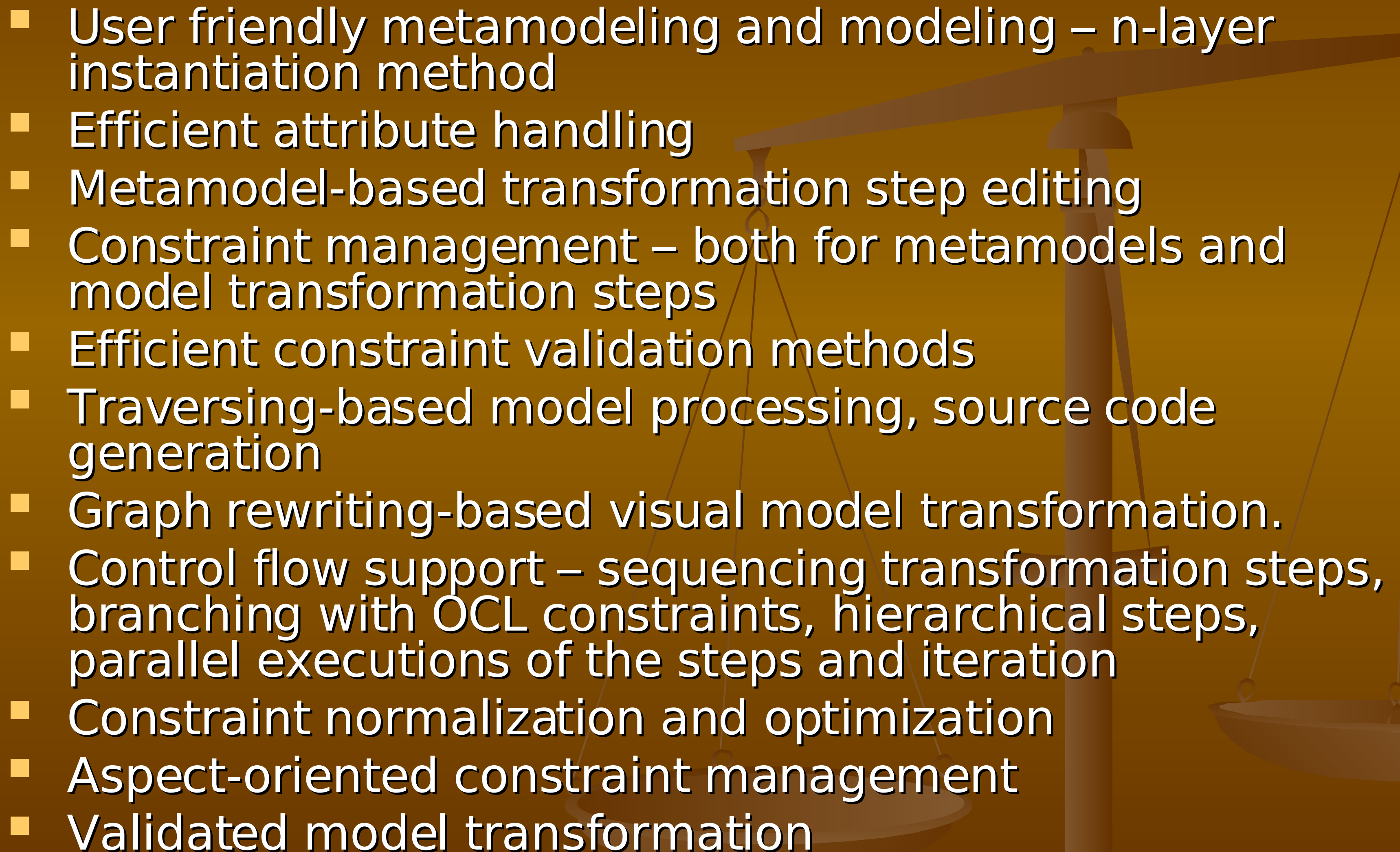
- Introduction
- VMTS Features
- Visual Language Definition
  - Metamodeling
  - Appearance Definition
- Sample Built-in Visual Languages
- Model Processing
  - VMTS Traversing Model Processor
  - VMTS Visual Model Processor
    - Validated Model Transformation
- Aspect-Oriented Constraint Management & Constraint Weaving
- Constraint Separation
- Further Case Studies
- Summary



Introduction

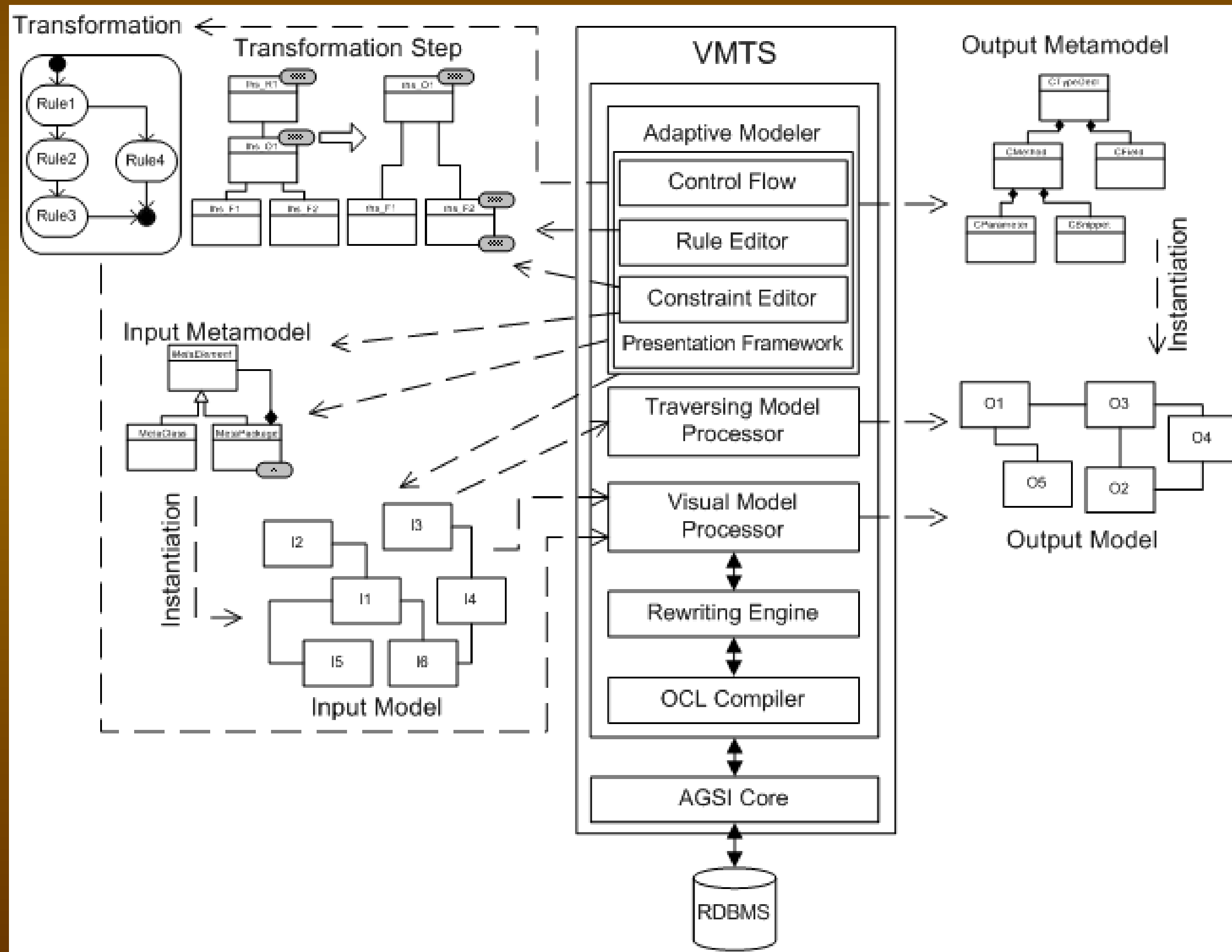
- VMTS is an n-layer metamodeling environment which supports editing models according to their metamodels, and allows specifying OCL constraints. Models are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel (“visual vocabulary”).
- Also, VMTS is a model transformation system, which transforms models using graph rewriting techniques. Moreover, the tool facilitates the verification of the constraints specified in the transformation step during the model transformation process.

VMTS Features

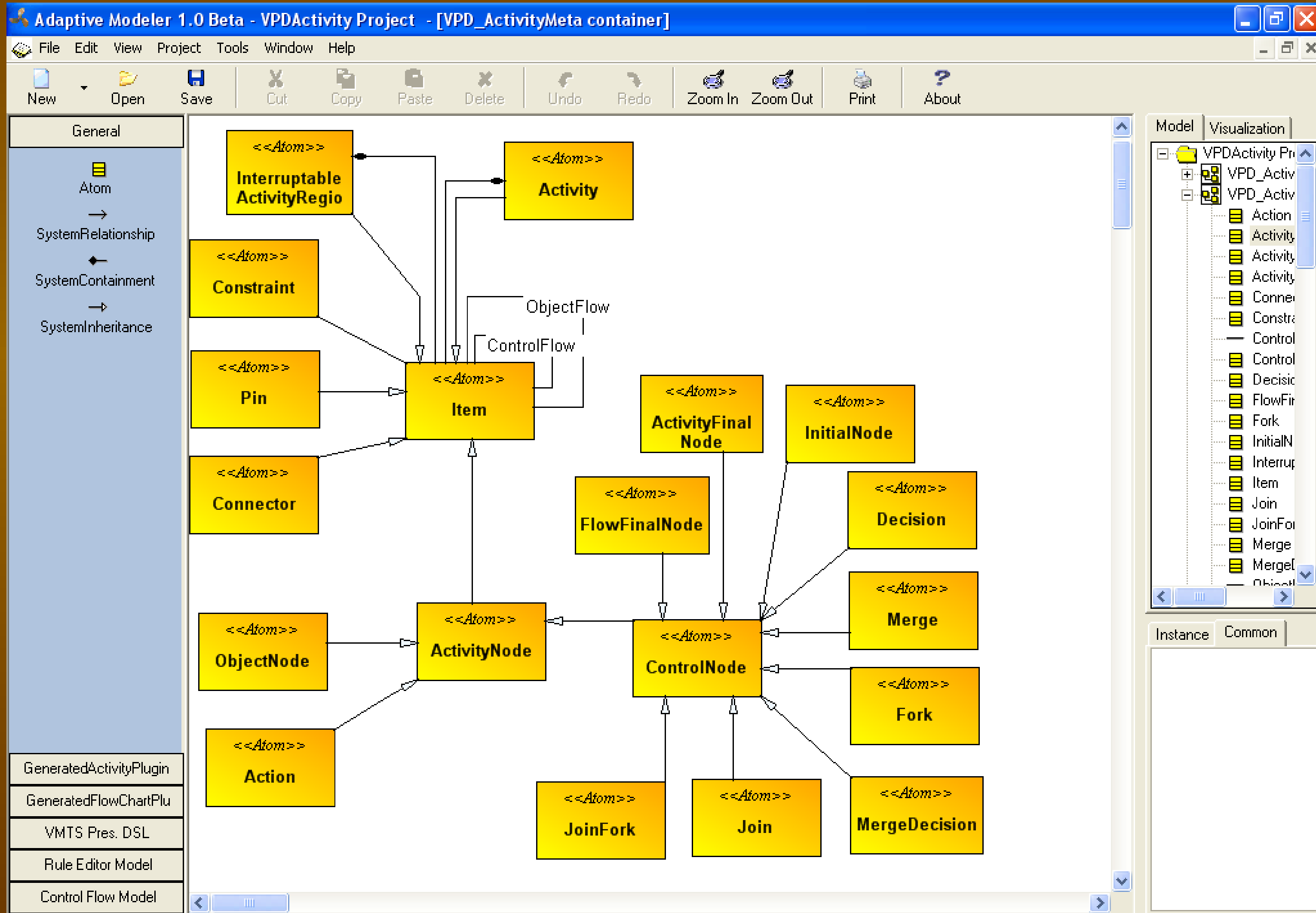


- User friendly metamodeling and modeling – n-layer instantiation method
- Efficient attribute handling
- Metamodel-based transformation step editing
- Constraint management – both for metamodels and model transformation steps
- Efficient constraint validation methods
- Traversing-based model processing, source code generation
- Graph rewriting-based visual model transformation.
- Control flow support – sequencing transformation steps, branching with OCL constraints, hierarchical steps, parallel executions of the steps and iteration
- Constraint normalization and optimization
- Aspect-oriented constraint management
- Validated model transformation

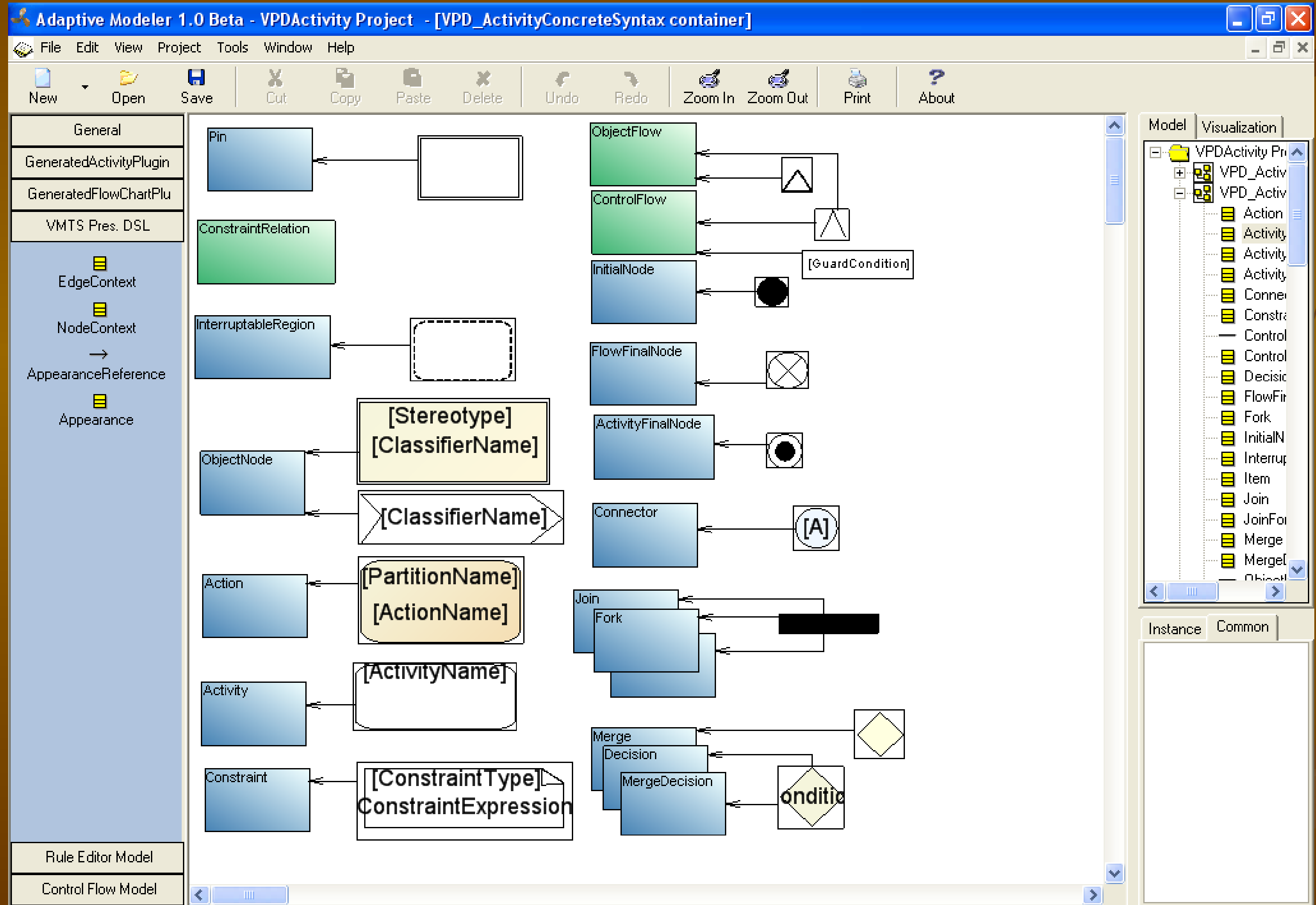
Visual Modeling and Transformation System (VMTS)



Metamodeling – Metamodel Definition with VMTS – UML Activity Metamodel

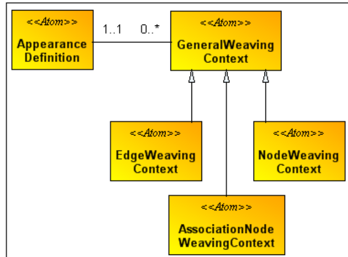


Appearance Definition with VMTS Presentation DSL – UML Activity Concrete Syntax

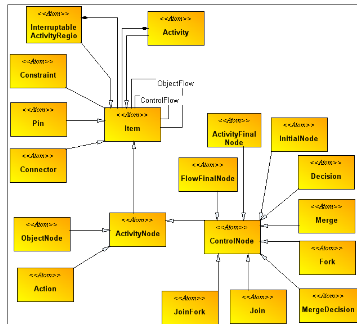


Plug-in Generation with Model Transformation

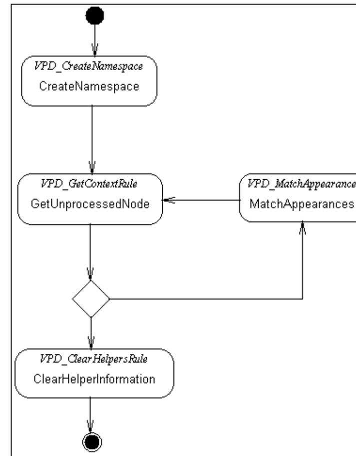
VPD Metamodel



Activity Metamodel

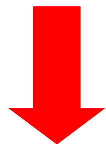


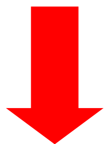
Transformation

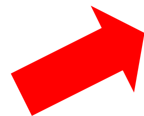


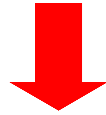
Generated Source Code

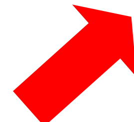
```
QSTATE QWatch::Stopped(QEvent const *e)
{
   switch (e->sig)
   {
      case Q_ENTRY_SIG: printf("\nStopped-ENTRY;");
                        displayStopwatch();
                        return 0;

      case Q_EXIT_SIG:  printf("\nStopped-EXIT;");
                        hideStopwatch();
                        return 0;

      case CLEAR:       printf("\nStopped-CLEAR;");
                        clearStopwatch();
                        Q_TRAN(QWatch::Stopped);
                        return 0;

      case START:       printf("\nStopped-START;");
                        Q_TRAN(QWatch::Started);
                        return 0;
   }
   return (QSTATE)&QWatch::Stopwatch;
}
```

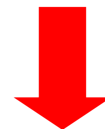


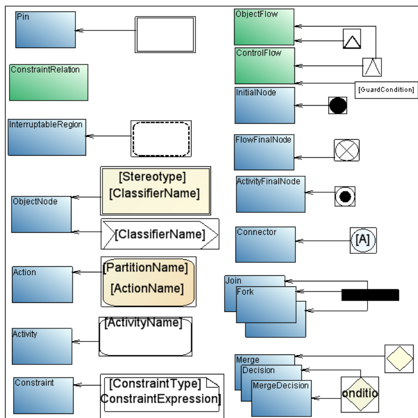




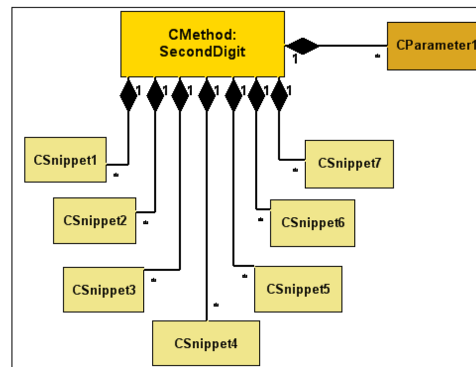




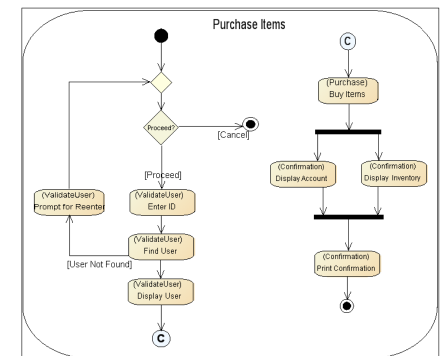




VPD Model - Appearance Definition



Generated CodeDOM Model



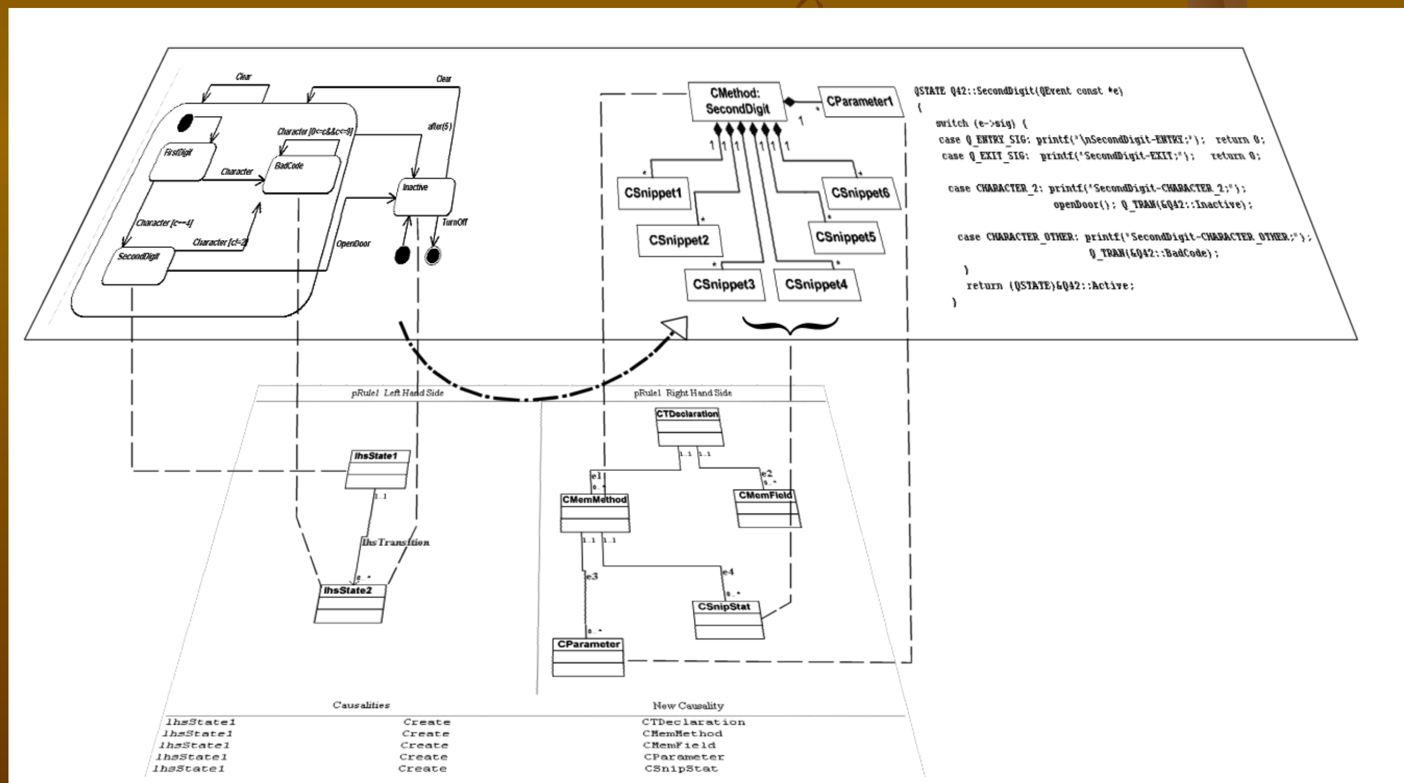
Sample Model with the Generated Plugin

VMTS Traversing Model Processor

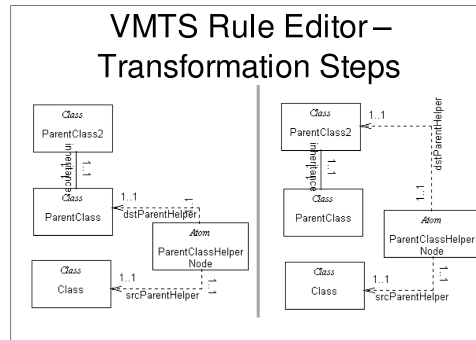
- Traversing Processor Wizard
  - Walks through the model, and
    - Generates domain-specific implementation
    - Validates the model and model constraints
    - Completes the already existing models
    - Optimizes the model structures
- Traversing Processor Executer
  - Runs the implemented traversing routines

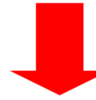
Metamodel-Based Rewriting

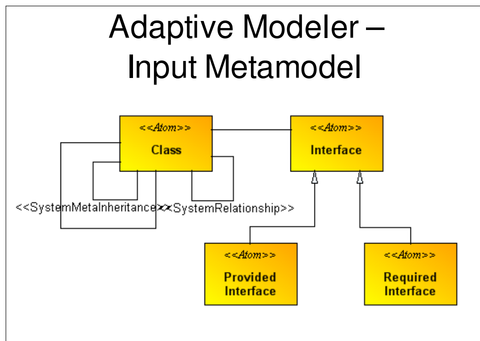
- An instantiation of LHS must be found in the graph the rule being applied to instead of the isomorphic subgraph of the LHS.

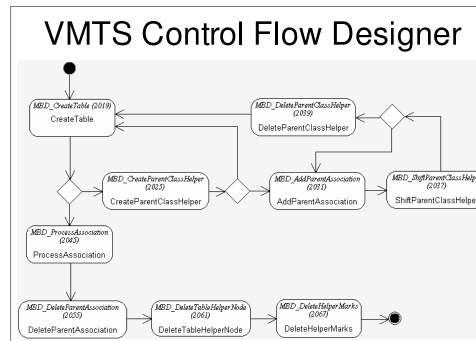


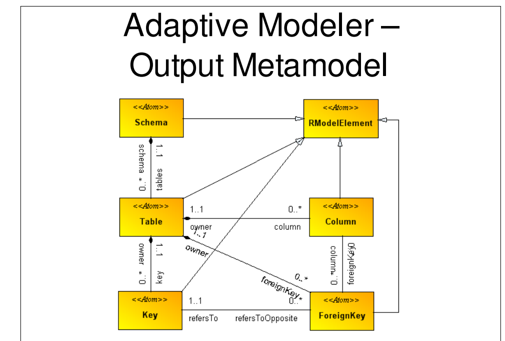
VMTS Metamodel-Based Model Transformation



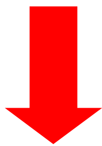


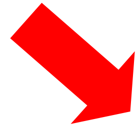


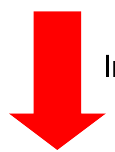




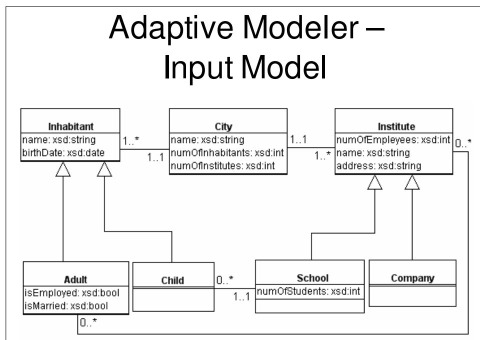
Instantiation

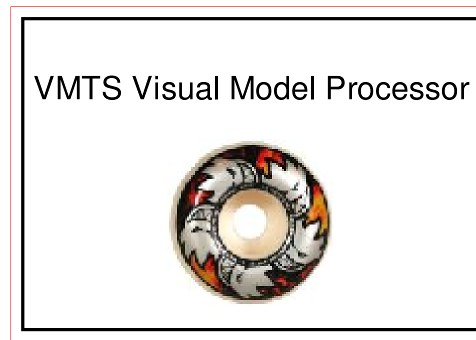


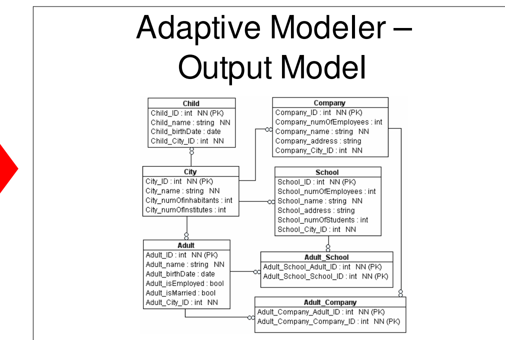




Instantiation

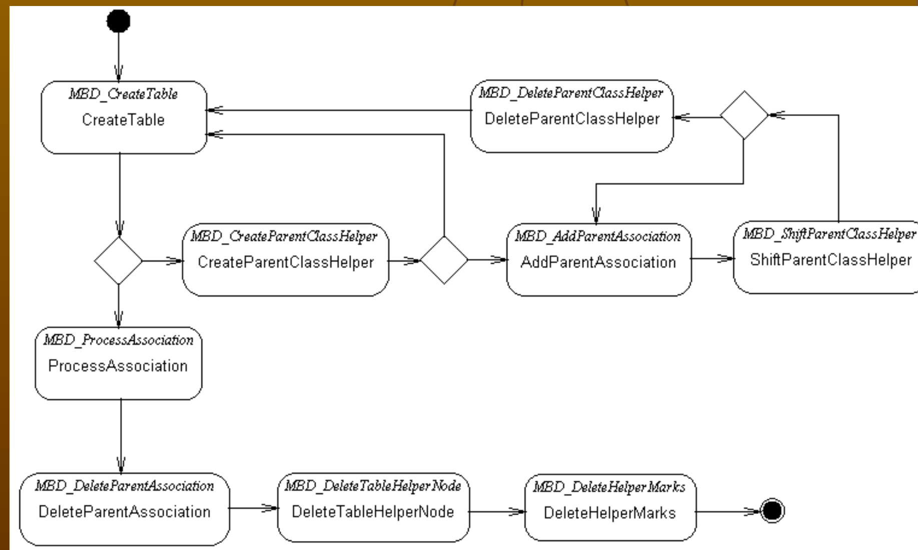






VMTS Visual Control Flow Language (VCFL) and Its Termination Properties

- VMTS is an n-layer metamodeling environment – directed, labeled graphs
- VCFL – sequencing transformation rules, branching with OCL constraints, hierarchical rules, parallel execution of the rules, and iteration.
- Metamodel-based rules – multiplicity, OCL constraints
- Internal causalities – create, delete, modify – XSLT scripts
- External causalities – parameter passing
- VCFL transformation rule composing and termination algorithm



Validated Model Transformation

- Motivation
  - At the implementation level, system validation can be achieved by testing. There is no real possibility that the testing covers all the possible cases.
  - In case of model transformation environments, it is not enough to validate that the transformation engine itself works as it is expected. The transformation specification should also be validated.
  - There are several transformation environment, but only few of them supports some (offline) validation method.
  - There is a need for a solution that can validate model transformation specifications: online validated model transformation that guarantees if the transformation finishes successfully, the generated output (database schema) is valid, and it is in accordance with the requirements above.

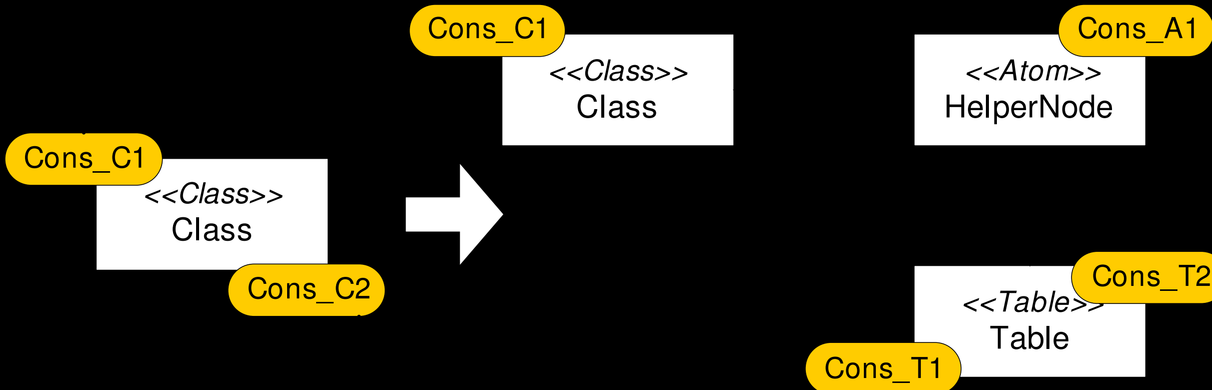
Validated Model Transformation /2

- A precondition assigned to a transformation step is a boolean expression that must be true at the moment when the transformation step is fired.
- A postcondition assigned to a transformation step is a boolean expression that must be true after the completion of a transformation step.
- If a precondition of a transformation step is not true then the transformation step fails without being fired.
- If a postcondition of a transformation step is not true after the execution of the transformation step then the transformation step fails.
- A direct corollary of this is that an OCL expression in LHS is a precondition to the transformation step, and an OCL expression in RHS is a postcondition to the transformation step.
- A transformation step can be fired if and only if all conditions enlisted in LHS are true. Also, if a transformation step finished successfully then all conditions enlisted in RHS must be true.
- A model transformation is validated if satisfies a set of high-level constructions (e.g. validation, preservation, guarantee type constructs).
- Successful execution of the step guarantees that the output model fulfills the conditions required by high-level constructs.

Defining Transformation Steps with Constraints

context Class inv NonAbstract:
not self.abstract

context Atom inv ClassAttrsAndTableCols:
self.class.attribute->forAll(self.table.column->
exists(c | (c.columnName = class.attribute.name))

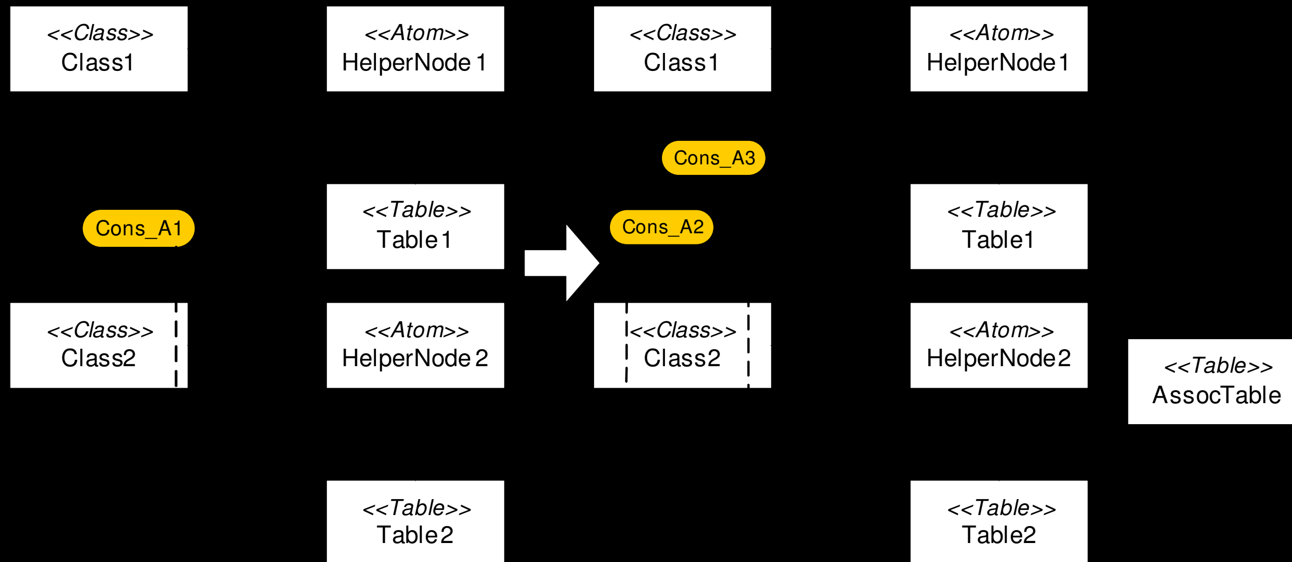


context Class inv Processed:
not self.isProcessed

context Table inv PrimaryAndForeignKey:
not self.columns->exists(c | (c.is_primary_key or
c.is_foreign_key) and c.allows_null)

context Table inv PrimaryKey:
self.columns->exists(c | c.datatype = 'int' and c.is_primary_key)

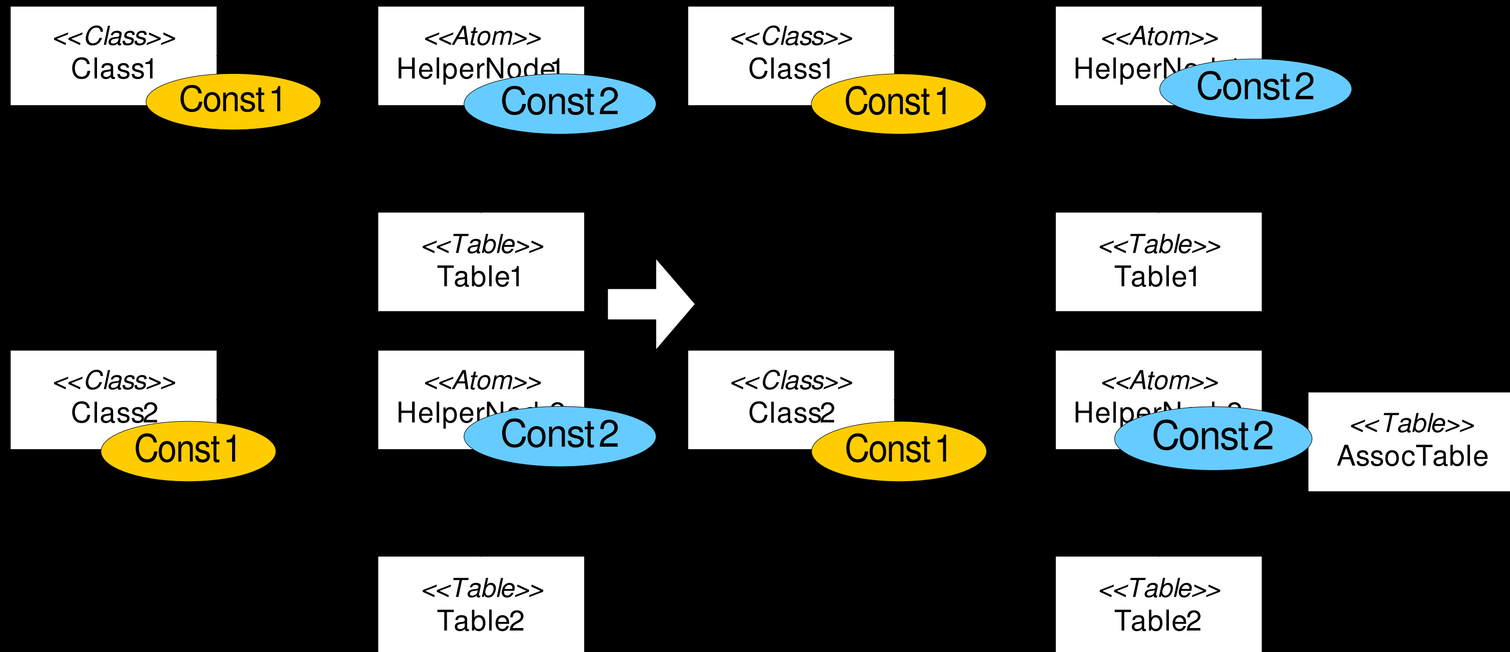
Defining Transformation Steps with Constraints /2



Aspect-Oriented Constraint Management

- Motivation
  - Transformation consists of several steps, many times not only a transformation rule but a whole transformation is required to validate, preserve or guarantee a certain property
  - The same constraint appears numerous times in the transformation → crosscuts the transformation
- Aspect-oriented constraint management
  - Aspect-oriented constraints
  - Constraint aspects
  - Weaver algorithms
- Results:
  - Consistent constraint management
  - Reusable constraints and transformation rules
  - Weaving algorithms facilitates to require from not only individual rules, but from whole transformations to validate, preserve or guaranty certain properties.

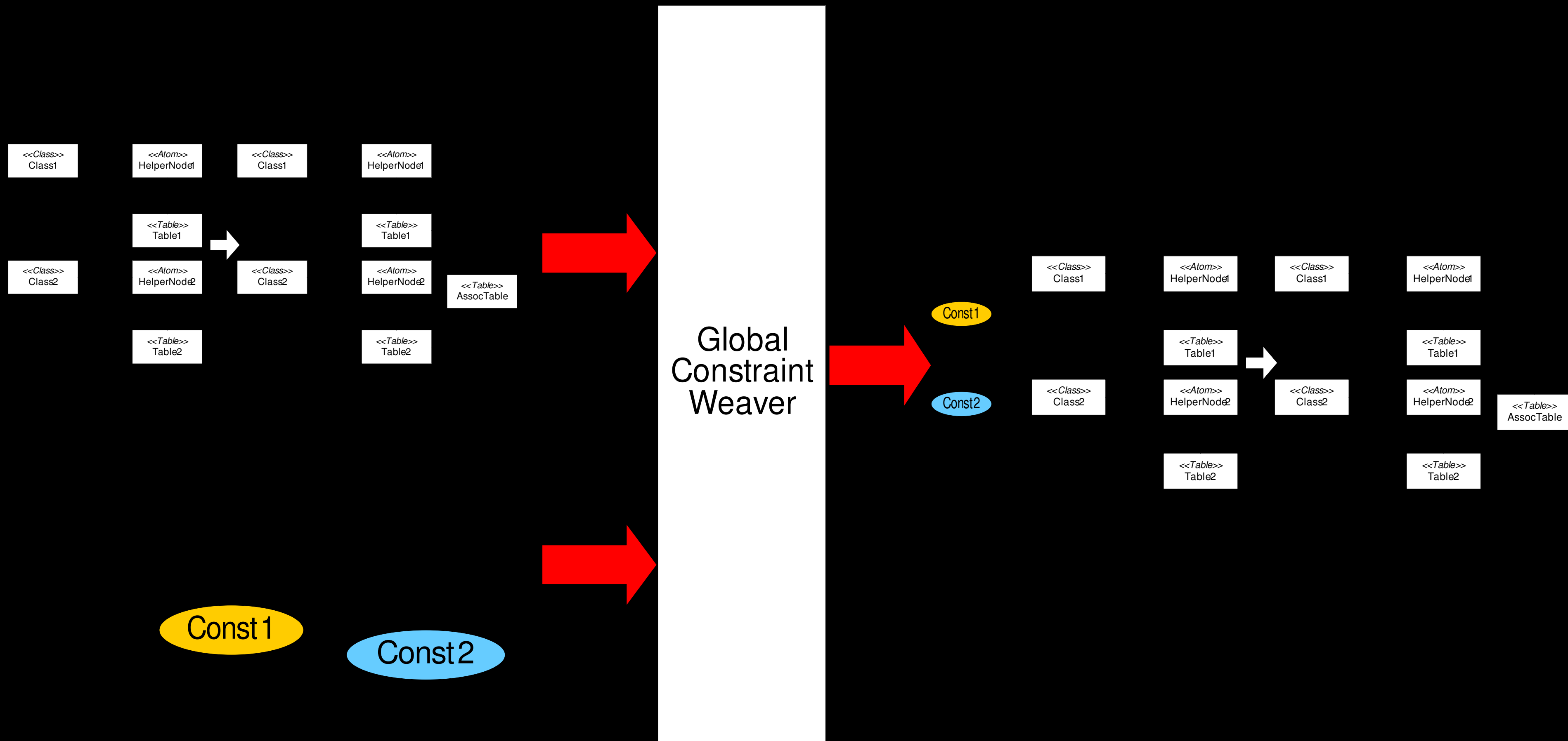
Removing Crosscutting Constraints



context Atom **inv** Const2:
self.class.attribute->forAll(self.table.column->
exists(c | (c.columnName =
class.attribute.name))

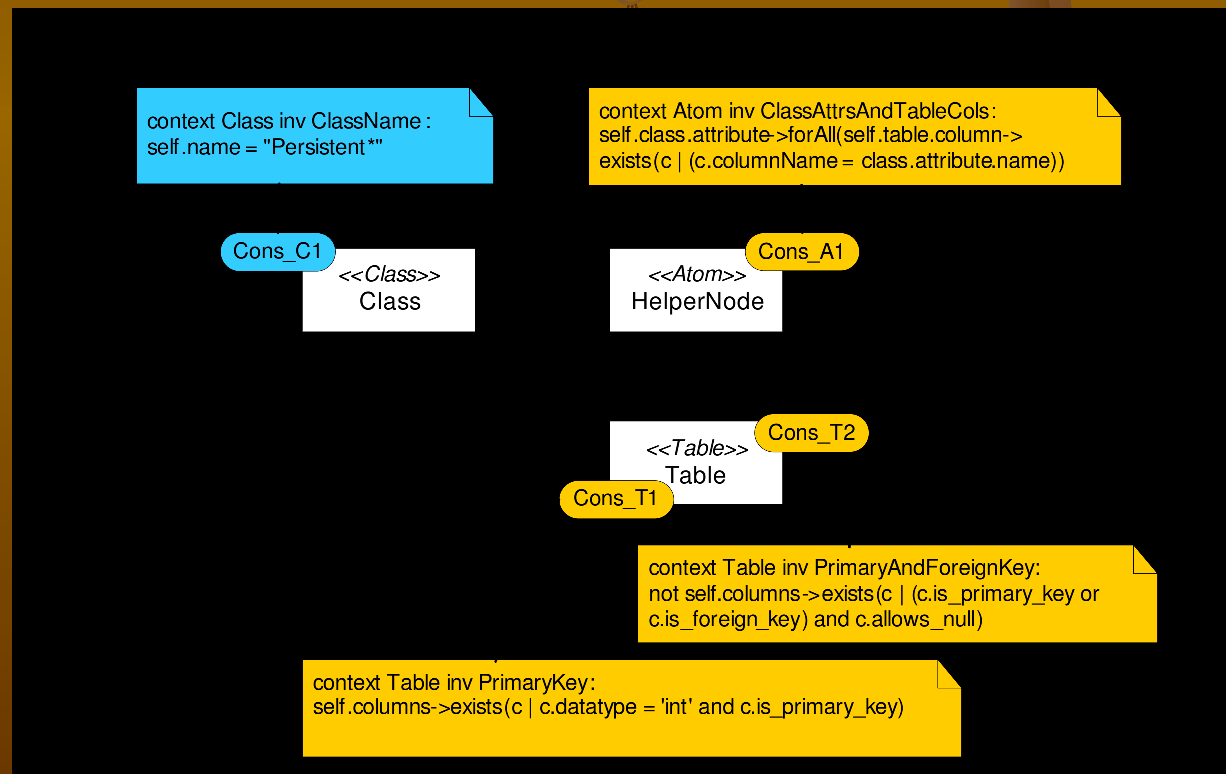
context Class **inv** Const1:
not Abstract

Global Constraint Weaver

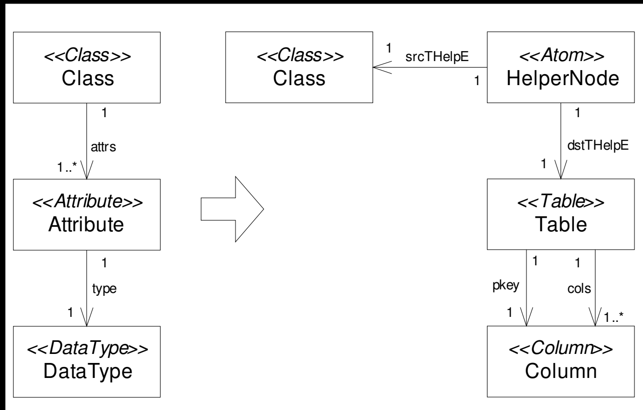


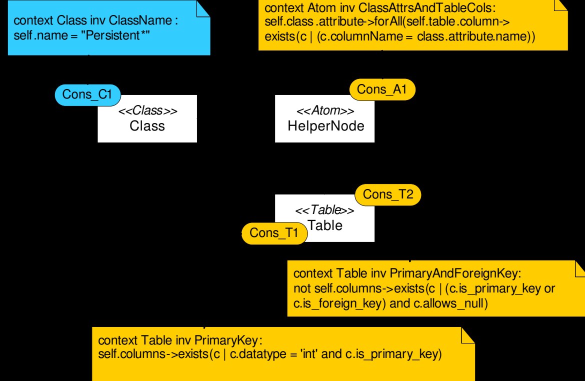
Constraint Aspect

- A constraint aspect is a pattern (structure) built from metamodel elements to which OCL constraints are assigned. A constraint aspect contains not only textual conditions described by the OCL constraints but it has:
  - Structure, type and multiplicity conditions,
  - OCL constraints, and
  - Weaving constraints.

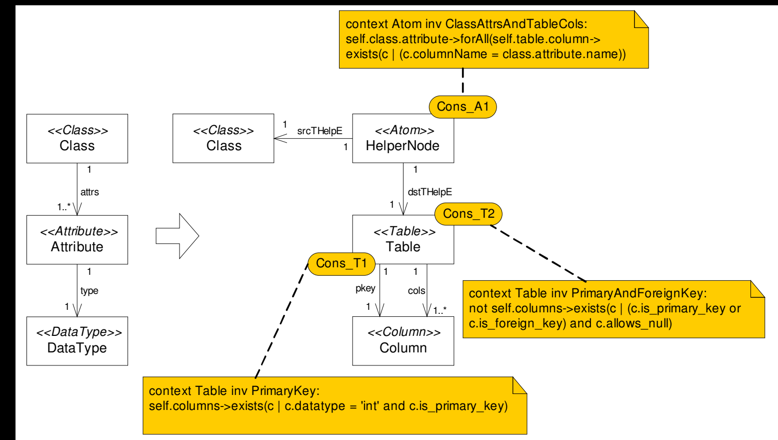


Constraint Aspect Weaver



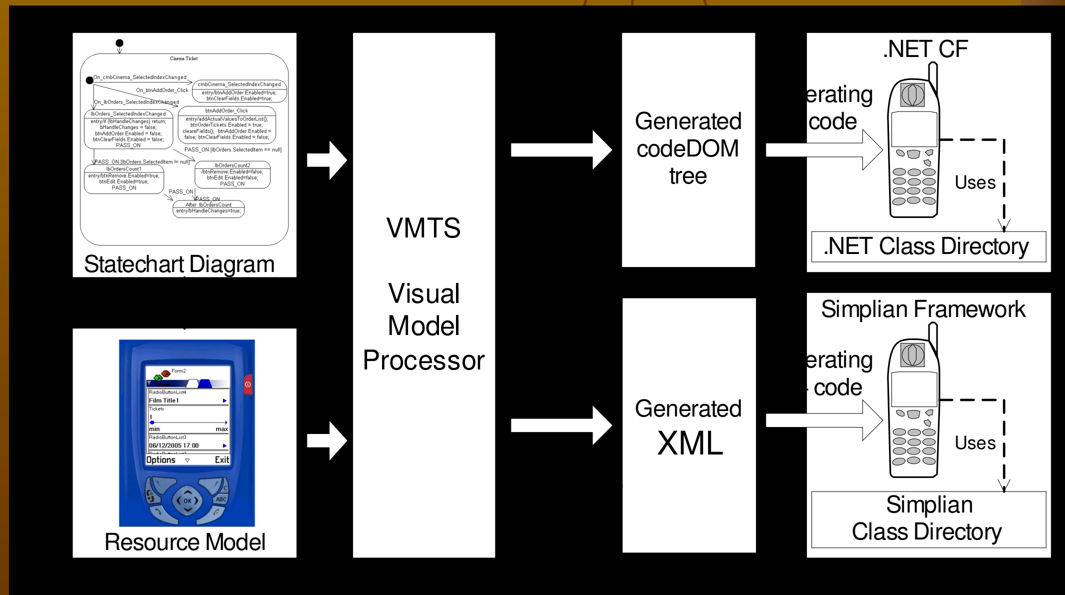


Constraint Aspect Weaver



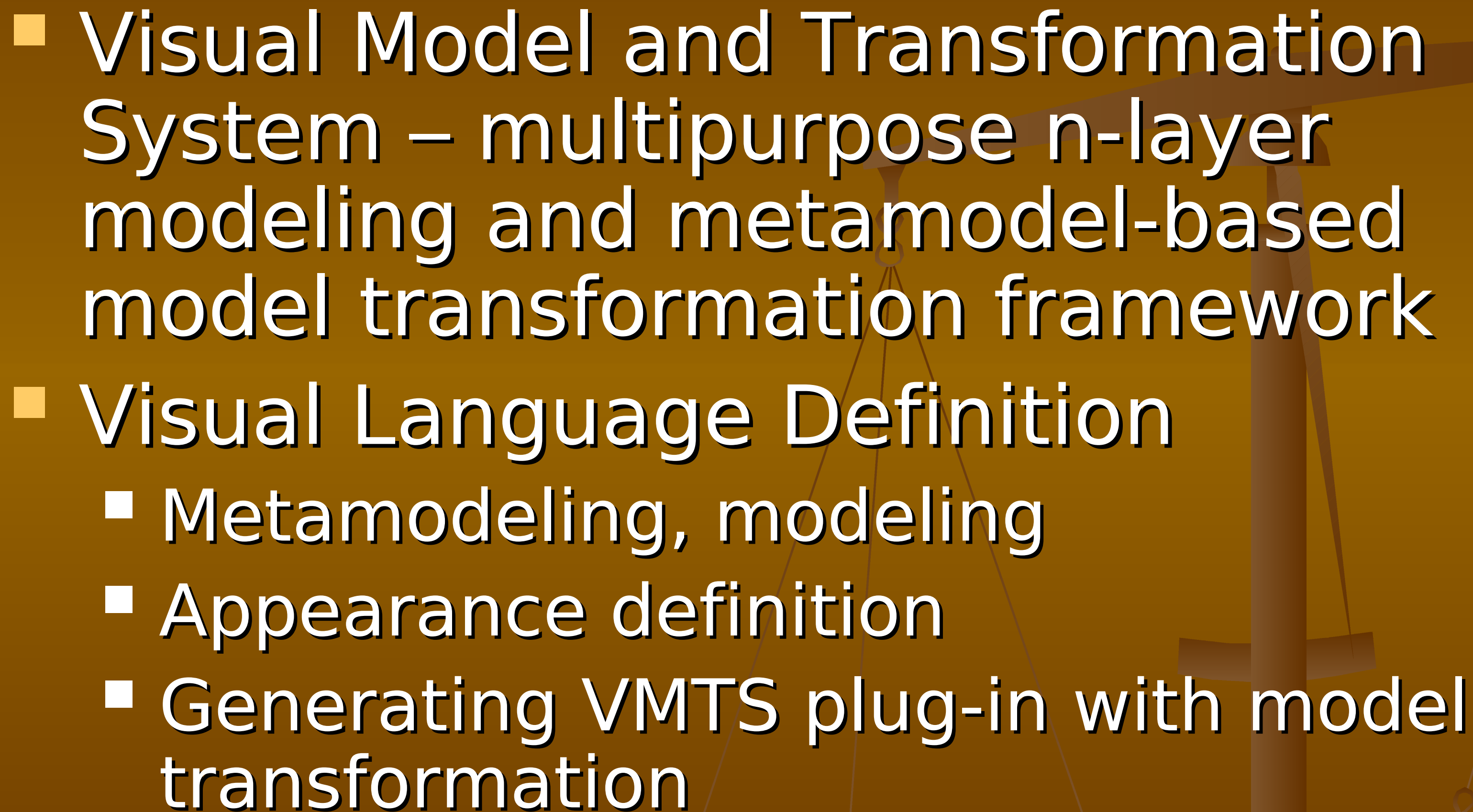
Sample Case Studies

- Generating C++ and C# source code from class diagram
- Generating C++ source code from statechart diagram for Quantum Framework
- Generating relational database model from class diagram (*Class2RDBMS*)
- Flattening hierarchical statechart diagram
- Model-based software evolution
- Model-based unification of mobile platforms



Summary /1

- Visual Model and Transformation System – multipurpose n-layer modeling and metamodel-based model transformation framework
- Visual Language Definition
  - Metamodeling, modeling
  - Appearance definition
  - Generating VMTS plug-in with model transformation



Summary /2

- Optimized constraint handling and validation
  - Aspect-oriented constraint management (consistency, reuse)
  - More efficient constraint propagation and validation methods (Constraint Aspects)
- Validated model transformation
  - Preconditions, postconditions – OCL constraints enlisted in model transformation steps.
  - Using the weaving algorithms we can require from not only individual steps, but from whole transformations to validate, preserve or guaranty certain properties.
- Constraint separation facilitates to reduce the complexity of validation type constraints that makes them understandable and working with them becomes easier.

Open Issues – CAMPaM 2007

- Domain Specific Design Patterns
  - Partial Instantiation
- General Simulation Modeling
  - DSLs
  - UI Programmability
  - Transformation Debugging
- Round-Trip Engineering Support
  - Traceability
  - Sophisticated diff mechanisms
  - Testing models