Model transformation with a dedicated imperative language

IRISA Rennes (France) - Triskell team

Jean-Marc Jézéquel

Didier Vojtisek

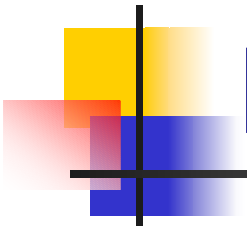
Jean-Philippe Thibault

Frédéric Fondement

IRISA

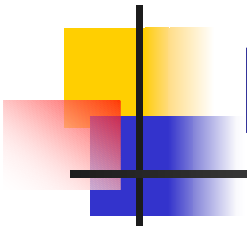
institut de recherche en informatique et systèmes aléatoires

TRISKELL



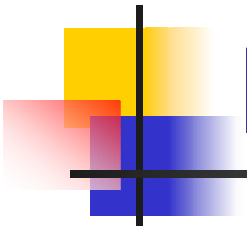
Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
- And soon…



Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
- And soon…



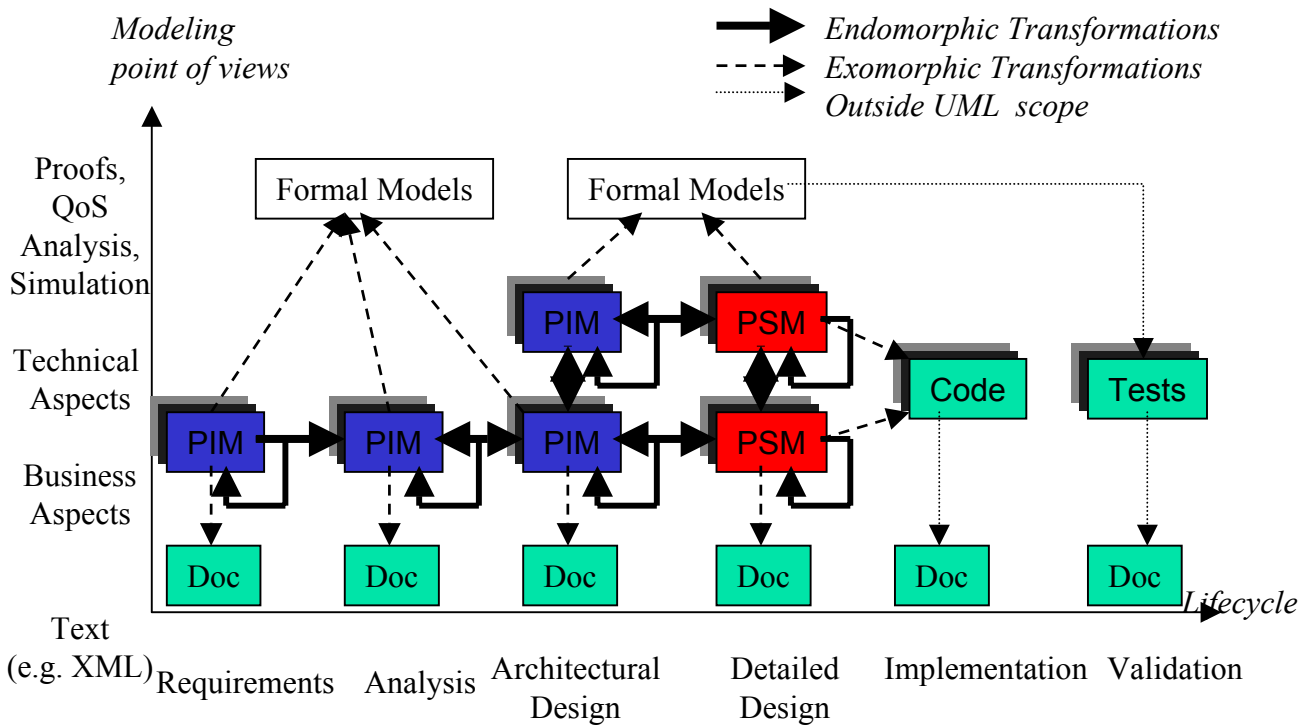
Model driven approaches

- « From contemplative to productive models» *Jean Bézivin*
- Based on different models most of the time of different meaning and level of abstraction.
- These models have to match / communicate / be composed

- Model transformation is a key point !

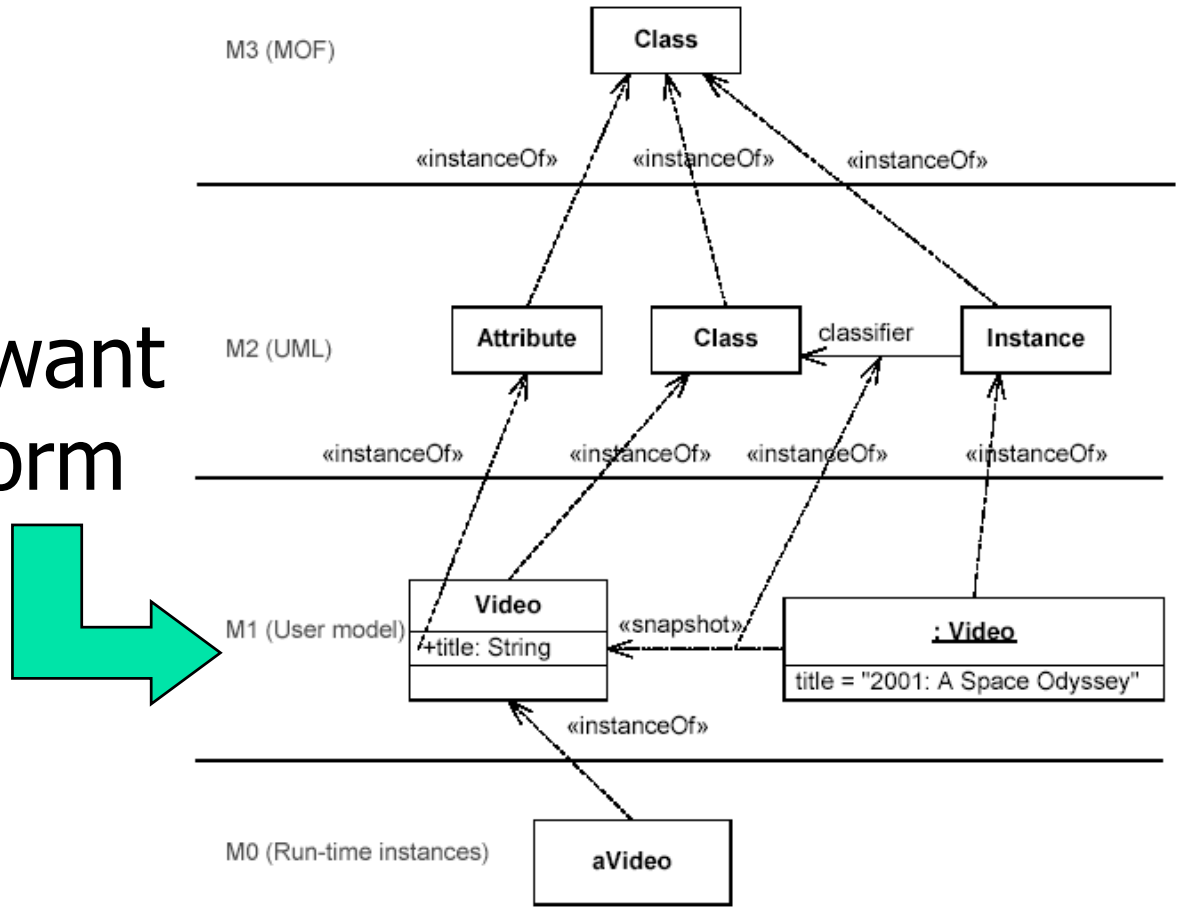
EX: MDA from the OMG

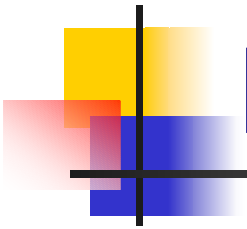
- Successive refinements



The OMG 4 layers architecture

What we want to transform

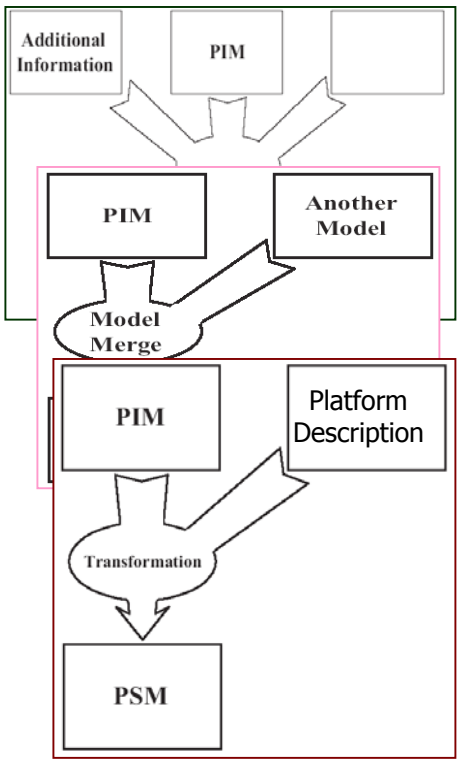


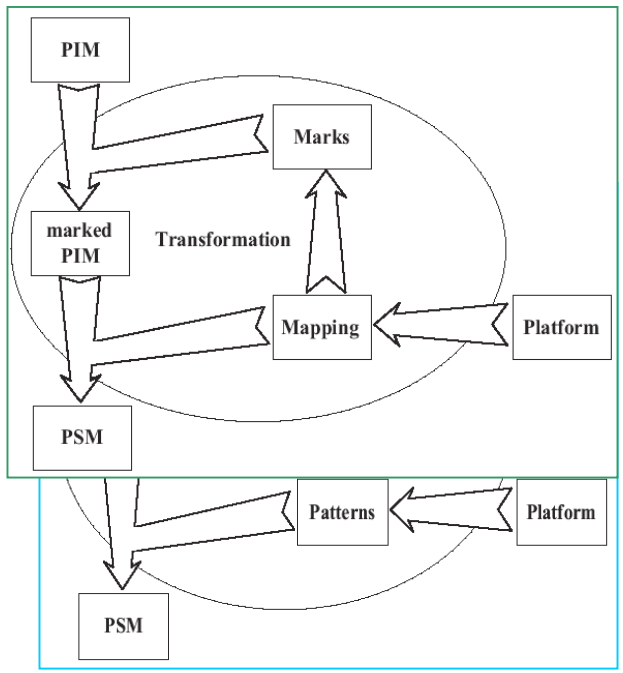


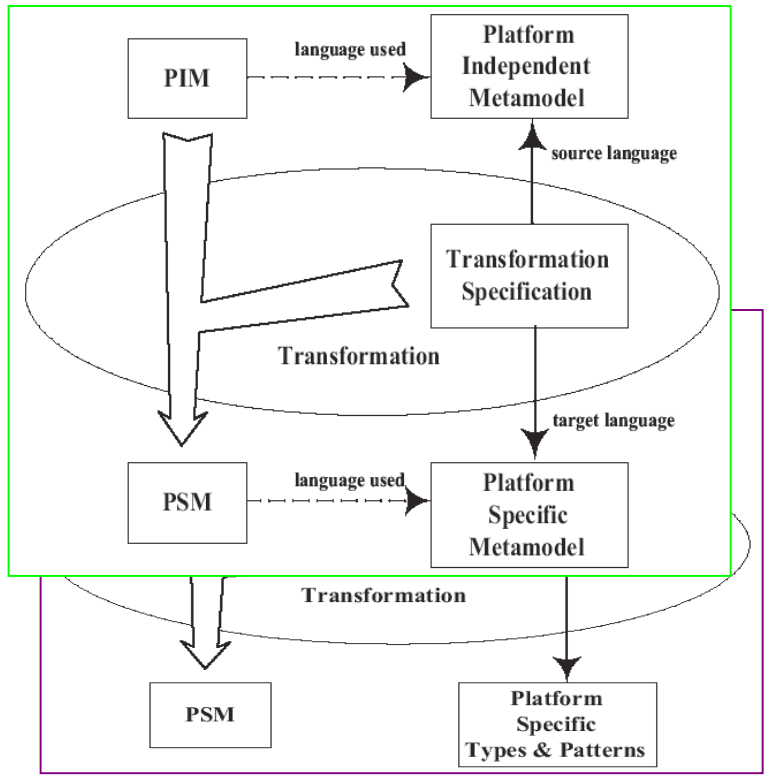
Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
- And soon…

Patterns of transformation

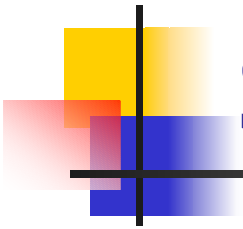






Etc. …

MDA Guide, OMG



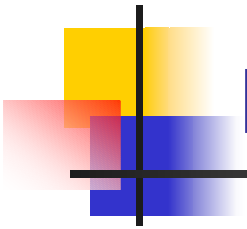
Something interesting…

Model transformation is a program: just apply the best programming practices !

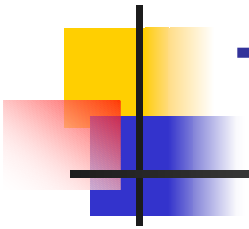
- Design and analysis
  - Models of transformations at different abstraction level
- Tracability, versionning, testing…

- **MDE**: transformation of transformation !
  - Such as XML with XSLT, a transformation may transform the model of a transformation
  - For instance to adapt a generic transformation (PIT) to a specific tool (PST)…

Transformation tools: requirements *(Bézivin Farcet Jézéquel Langlois Pollet)*

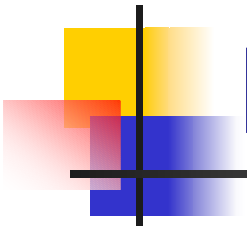


- Depends only on metamodels
  (not on models)
- Must be cascadable
- Can represent generic tasks, not depending on the level of abstraction
- Must be adaptable to slightly different problems
- Must be maintainable



Transformation tools now…

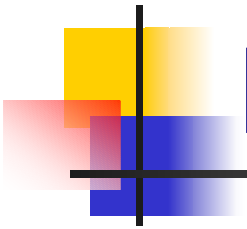
- An upcoming standard: OMG MOF QVT
  - Obviously, not yet implemented
- Many dedicated transformations
  - code generators, object to relational mappings, …
- Much less dedicated tools
  - Univers@lis, J, JMI implementations,…
  - No generic solution (UML, real-time,…)
  - Proprietary solutions



Plan

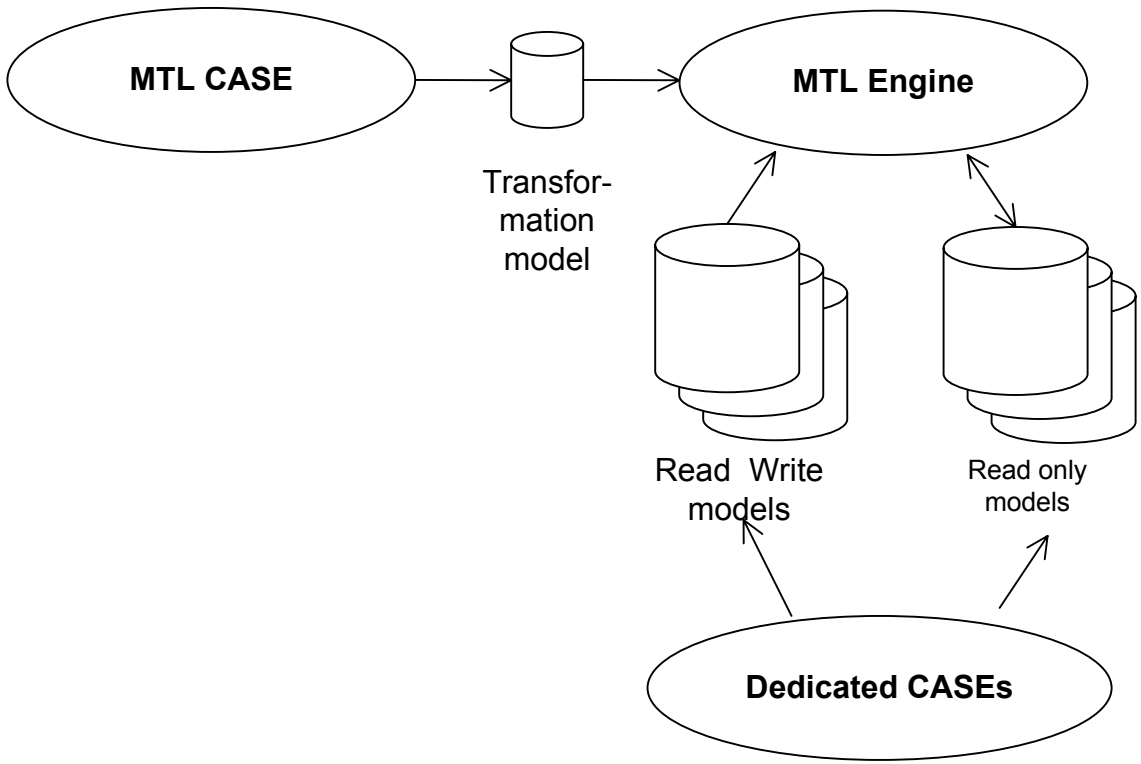
- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…

Model Transformation Language (MTL)

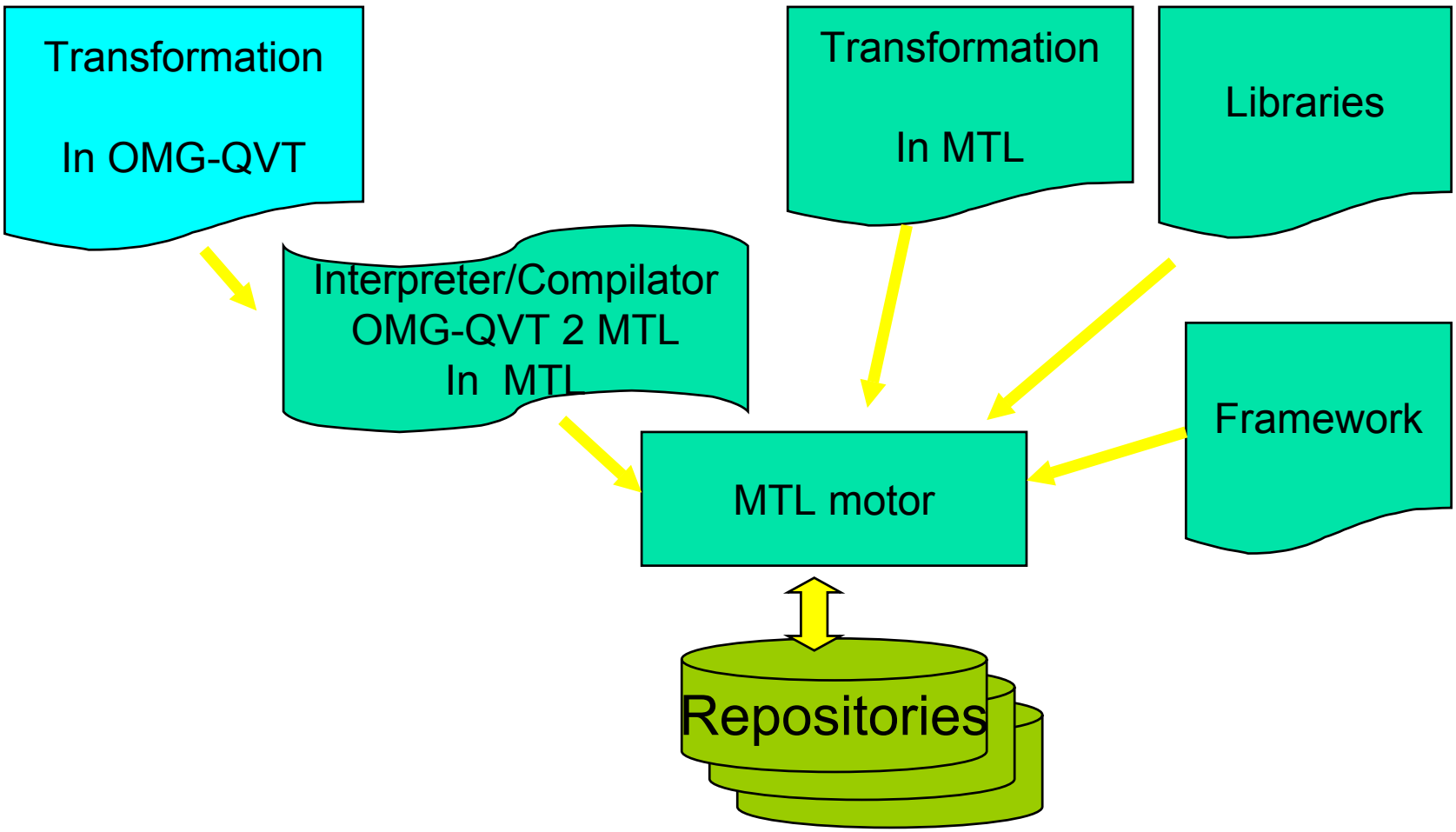


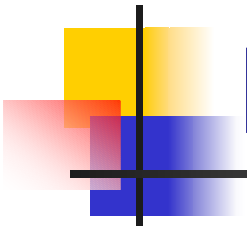
- The IRISA solution for model manipulations
- A dedicated language for model transformation (DSL ?)
- To be used as a motor when the OMG MOF QVT will be realised

MTL architecture



MTL Position





Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…

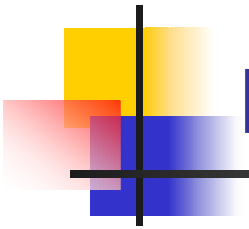
Transformation tools: requirements *(Bézivin Farcet Jézéquel Langlois Pollet)*

- Depends only on metamodels
  (not on models)

=>

- Manipulates models
  - Of any kind of metamodel
  - In any kind of repository

Transformation tools: requirements *(Bézivin Farcet Jézéquel Langlois Pollet)*



- Must be cascadable

=>

- Re-usable libraries of transformations
- Interoperability
  - Can call other (transformation ?) tools
    - Native libraries
  - Can be called by other (transformation ?) tools

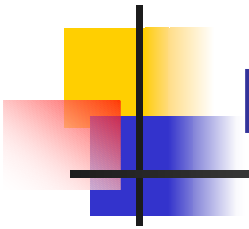
Transformation tools: requirements *(Bézivin Farcet Jézéquel Langlois Pollet)*

- Can represent generic tasks, not depending on the level of abstraction
- Must be adaptable to slightly different problems

=>

- OO genericity (multiple inheritance)
  - For classes
  - For libraries
- Concept of view manipulation
  - Views are virtual models whose metamodel is described by a MTL Library

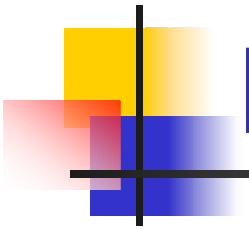
Transformation tools: requirements *(Bézivin Farcet Jézéquel Langlois Pollet)*



- Must be maintainable

=>

- Programming language with well-known concepts
  - Easy to learn
  - Existing maintenance solutions
- Independency from the model repositories



Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…

From the programmer point of view (1/2)

- Typed language
  - Static typing for MTL
  - Implicit typing for model elements
- Object-oriented language
  - Based on the OMG UML class diagrams
    - Packages
    - Classes
    - Associations (N-ary, class-associations, qualifiers…)
    - Visibility
    - Exception mechanism
    - …
- Methods (behaviours) in imperative style

From the programmer point of view (2/2)

- Integrated model manipulation
  - MTL object and model elements are manipulated the same way
  - No constraint on the number of manipulated models
- An abstract language
  - Based on MOF + OCL MM (+ QVT ?)
  - Many compatible concrete syntax may be defined
    - Full textual
    - Structure in UML class diagrams + methods in text
    - Structure in UML class diagrams + methods in an adapted activity graphs
  - Allows transformations of transformations
    - Adapt a transformation to a specific platform

Adding known techniques and specific innovating solution

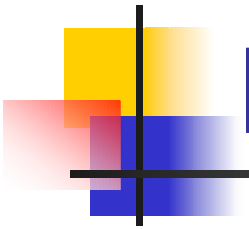
MTL =

« Old » well-known techniques

- OCL
  - One of the best solution for model manipulation
  - Standard library
- + Side effects
  - Model modification
  - MTL objects modification
- + Structuration
  - UML class diagrams

The MTL specificity

- + MTL Libraries are "templated"
  - Models to be manipulated – found at runtime
  - Views as MTL "abstract" libraries – for generic manipulations

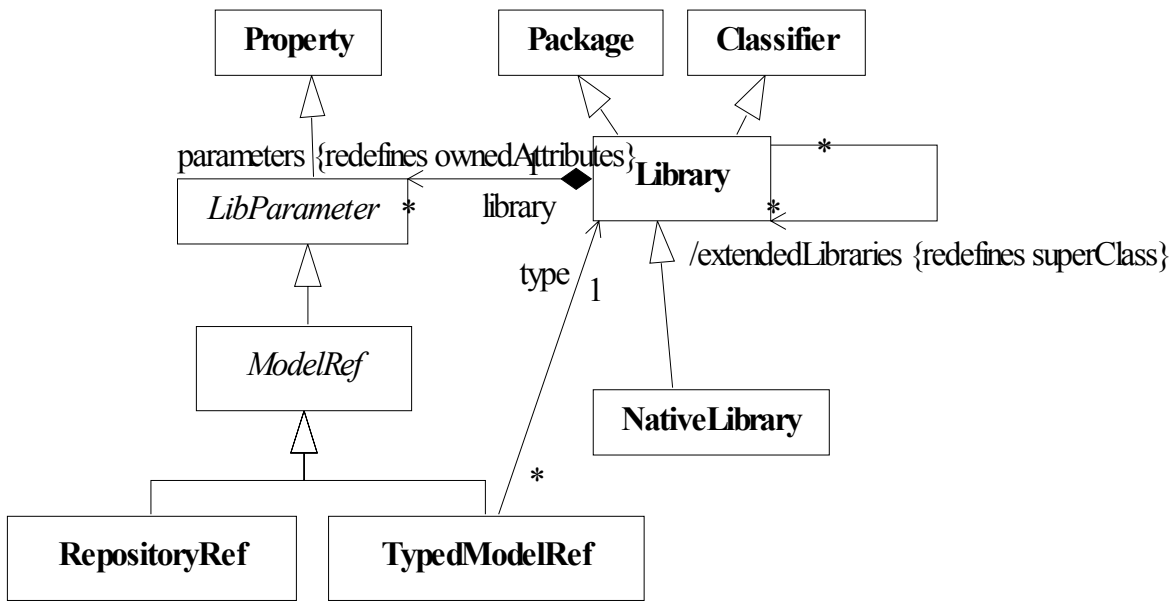


Plan

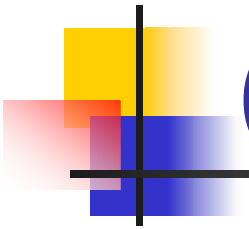
- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…

Model integration

- Everything is declared in a library which may be "templated" by a number of models or views
  - Libraries are "instanciated"
  - Declared elements can access real models and real adaptors (library subclass of the given view)

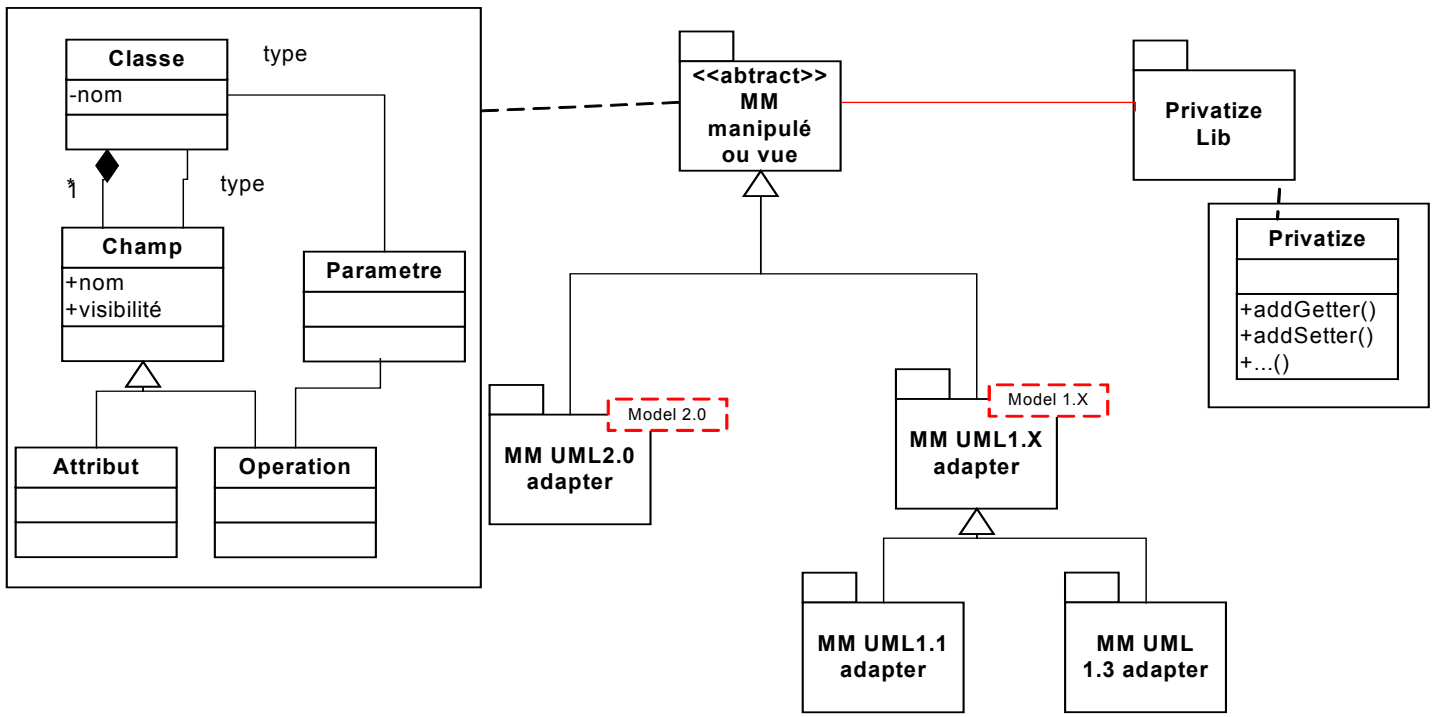


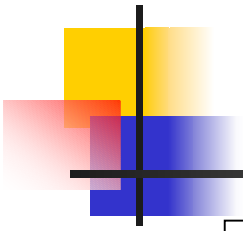
How to use views ? (motivation)



- Write transformations independent from metamodels of the manipulated models
  1. Describe manipulated concepts (PI MM!) in a library (as an example Class, Field…)
  2. Write in an inheriting library (PS MM!) how your concepts are mapped into the real metamodels (UML 1.4, CWM RDB,…)
  - This is the MDA pattern !

An example of view

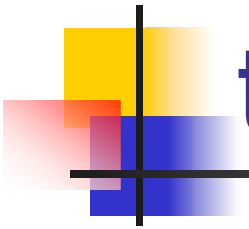




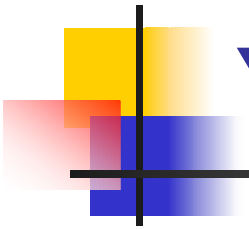
Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…

Independency from repository tools

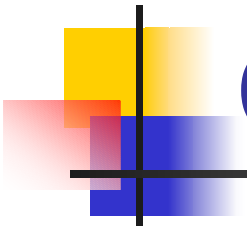


- Model manipulation implies model repositories !
  - Many of them are already available, with different techniques and standards
    - OMG MDA / JMI (Novosoft, CIM, MDR, EMF, Univers@lis,…)
    - UML CASE (Rose, Objecteering, UMLAUT, Poseidon,…)
    - Object-Oriented Databases / OQL (Poet, Jasmine,…)
    - Relational databases (PostgreSQL, Oracle,…)
    - Distributed systems (CORBA, EJB, .net,…)
    - …
  - Many others in the future
- MTL must not depend on repository technology !



Yet another API…

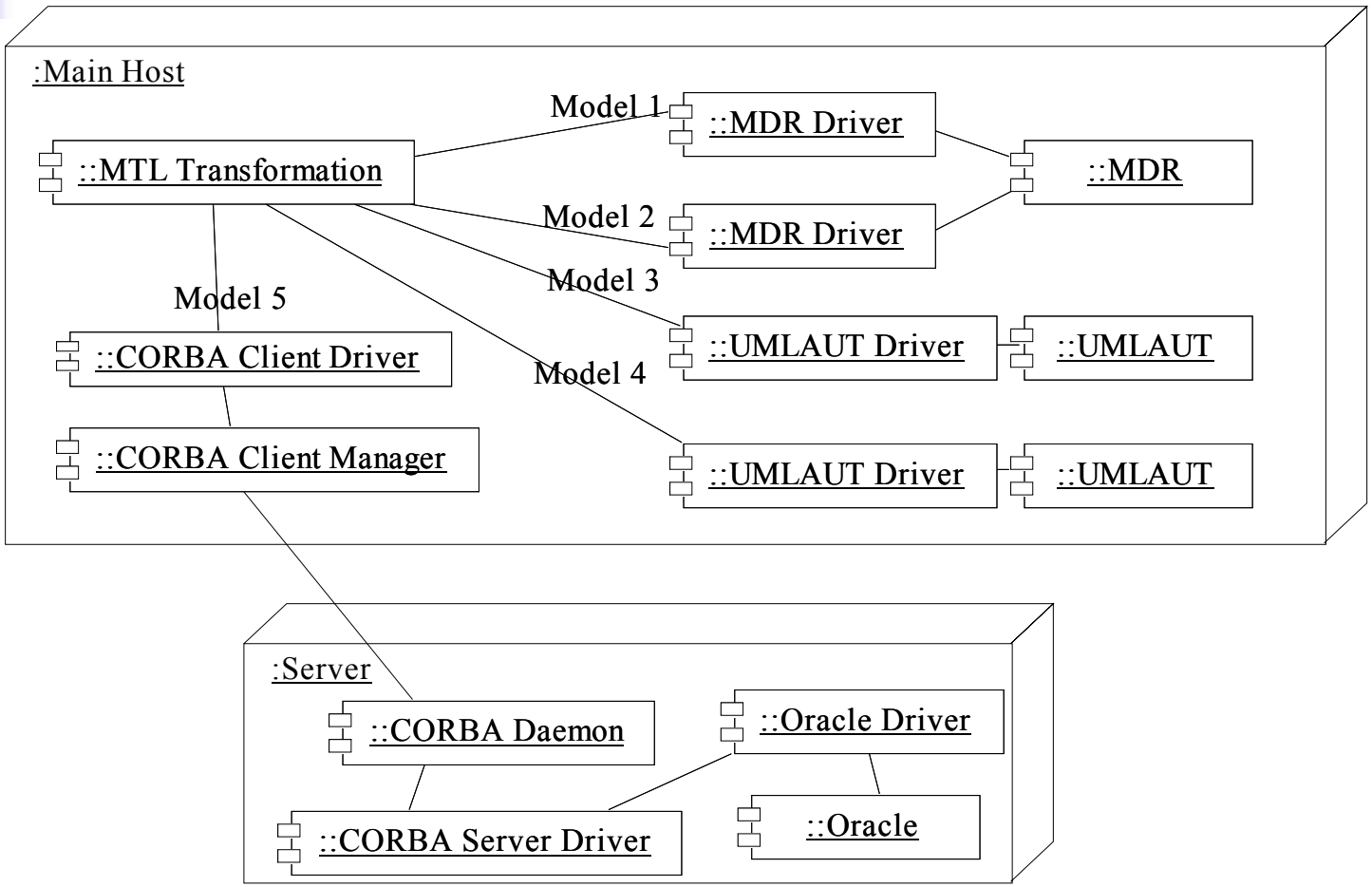
- We have introduced a new API for model manipulation
  - IDL compatible
  - The most basic concepts of the MOF
    - No reflection
  - "Drivers" must adapt the tool to the API
    - Already written: MDR
- DON'T MIND !
  - MTL (motor / compiled programs ?) use this API
  - No knowledge of this API required: everything is in the language



Capabilities

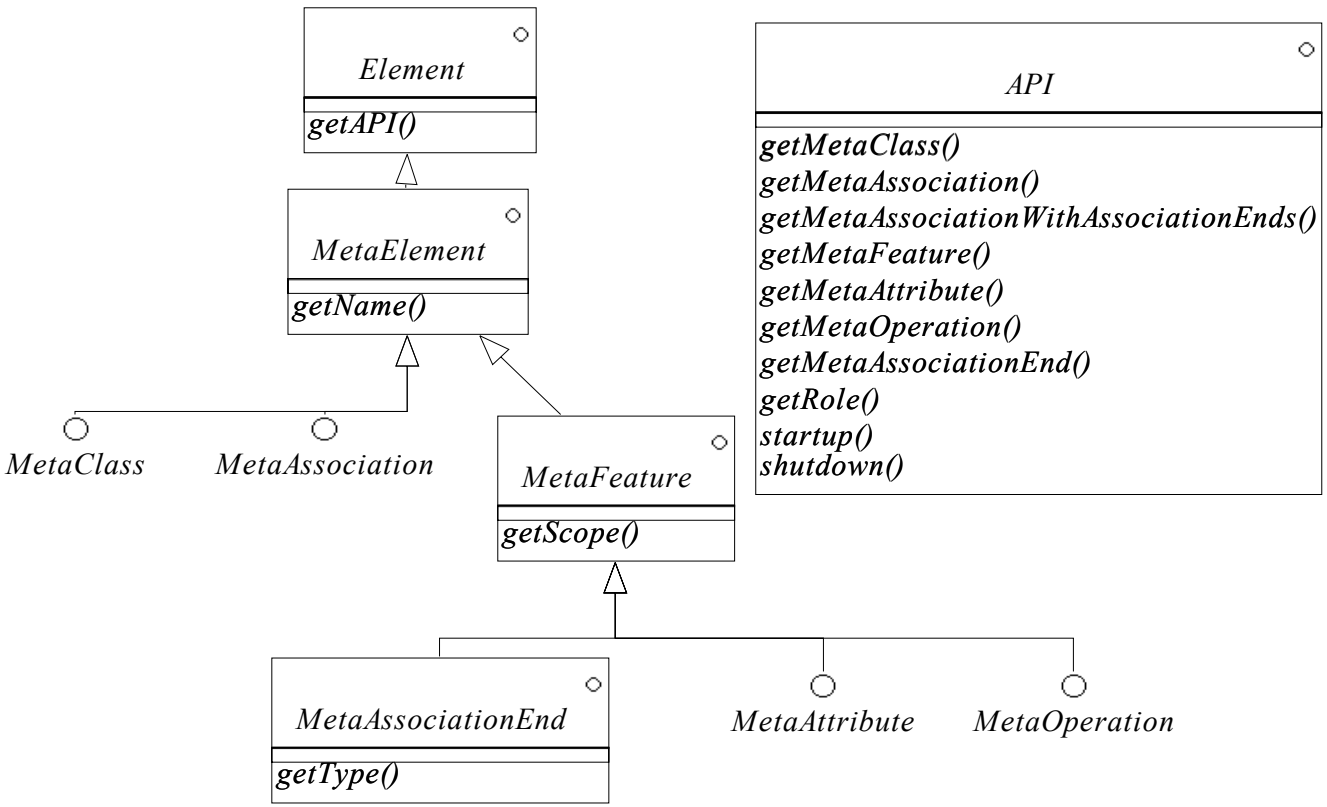
- Create, find or delete an instance of
  - a class (found with its qualified name)
  - an association (found either with its qualified name or its association ends)
- Field access (found with its name)
  - attributes, references, operation – if supported !
  - attribute modification
- Optional parts (may be not supported – PST = Platform Specific Transformation!):
  qualified links, reflection, dedicated methods…

An example


:Main Host
::MTL Transformation
Model 1
::MDR Driver
::MDR
Model 2
::MDR Driver
Model 3
::UMLAUT Driver
::UMLAUT
Model 4
::UMLAUT Driver
::UMLAUT
Model 5
::CORBA Client Driver
::CORBA Client Manager
:Server
::CORBA Daemon
::Oracle Driver
::CORBA Server Driver
::Oracle

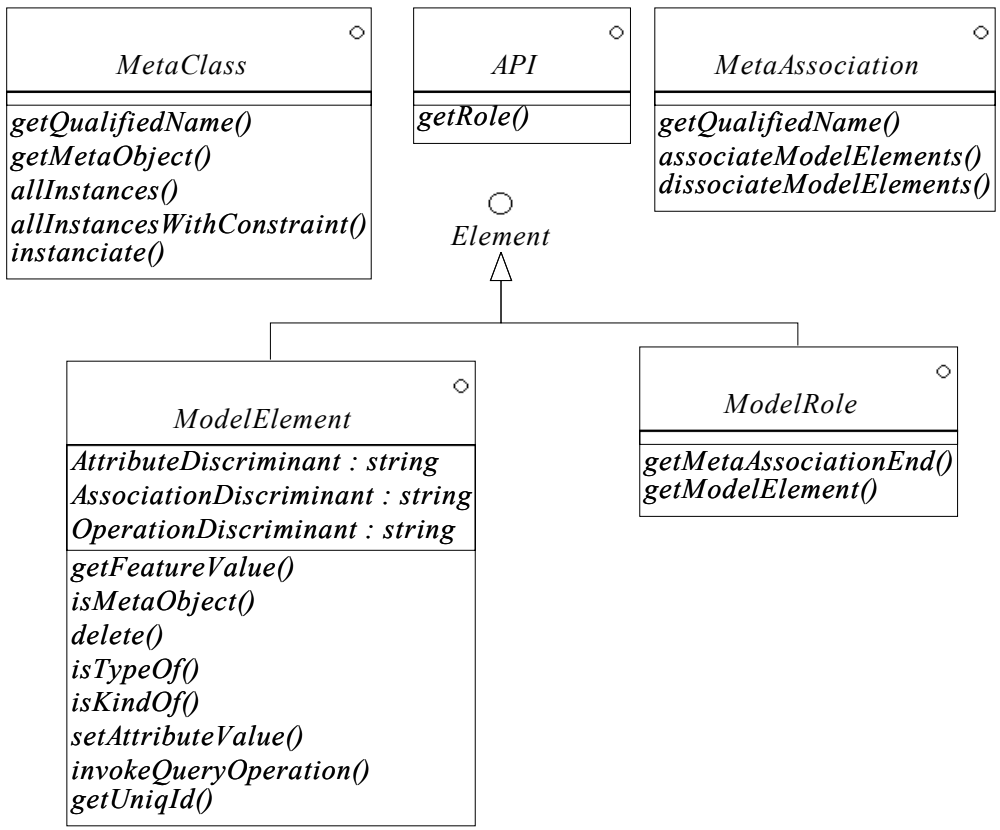
The meta-level

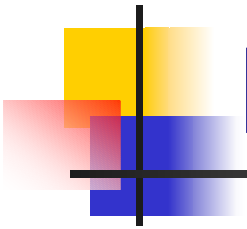
- Information *from MTL*



The model-level

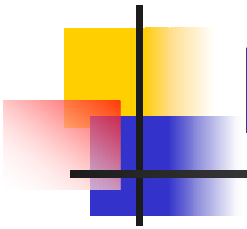
- Information *from the repository driver*





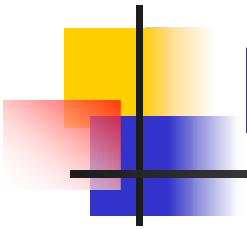
Plan

- Model Driven Engineering
- Model transformation
- MTL concepts
  - Respected requirements
  - Overview
  - Models and views
  - Repository access
- And soon…



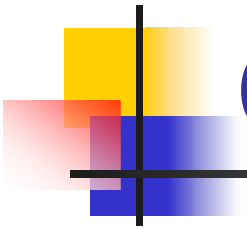
BasicMTL

- Offers main characteristics of MTL
  - Strongly typed for himself, layzy typed for models
  - Object oriented (libraries, classes, attributes and operations, multi inheritance for classes and libraries)
  - Model manipulation (repository access)
  - Action language independent from the platform
  - Predefined types and operations inspired from OCL
  - Views – Adapter mechanism
  - Exceptions
- *Platform independent* (from standards and real platforms)
  - Independence is adaptability (to the future…)



BasicMTL and MTL

- BasicMTL will be available soon
- It offers less possibilities than MTL
- By transformation (in BasicMTL), it can become MTL
  - BasicMTL is used as a “bootstrap” for MTL
- It will permit testing main MTL concepts !



Conclusion

- We propose to see a transformation language as a classical language
  - Ease of learning
  - Apply well known methodologies
- Still have to implement it !
  - BasicMTL quite soon (validation of concepts)
  - Adaptation to the QVT standard later